PHYS 210: Introduction to Computational Physics Fall 2009 Homework 2
Version 4: Monday October 12
Due: Wednesday, October 14, 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. The following assignment requires
   (a) Working with the `xmaple` graphical user interface (GUI) 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).
   (c) Writing a `bash` script that interfaces with `maple`. (Problem 5).
2. In order to complete your homework, you can use
   (a) `hyper`, either in the Computer Lab, or via remote connection from other computer(s) using `ssh`, with X forwarding enabled for the cases where worksheets are required. Recall that if you don't have Linux installed on your computer(s) that run Windows, that you can download and install the free `PuTTY` and `Xming` packages. (see the *Course Software Availability for Personal Machines* web page, available via the main course page). This software will allow you to `ssh` into `hyper` from your Windows system(s) (using `PuTTY`) and use graphical applications such as `xmaple` (using `Xming`).
   (b) Your own laptop or home PC, provided that you have `Maple 12` installed. Recall that you can borrow CDs from us, or from the Physics and Astronomy system administrators for this purpose (and for Linux, Windows and Mac OS).
3. **IMPORTANT!!** However you choose to complete the homework, all of the files specified below must exist and be in their proper locations within your `/phys210/$LOGNAME/hw2` directory on `hyper`. Furthermore, it is *your* responsibility to ensure that if any of the worksheets and/or `Maple` source files *have* been uploaded from another computer, that they work properly on `hyper`. I.e. Ben and I must be able to start up `xmaple` or `maple` (as appropriate) and `read` your source files, or open and execute your worksheets, without encountering errors.
4. Whenever working with *any* worksheet in `xmaple`, be sure to save your work frequently, using, for example, `Ctrl-S`. This will minimize the amount of work that you lose should the interface crash (as it has been known to do from time to time).
5. **IMPORTANT / WARNING!!** It may take you several *hours* to properly complete **Problem 1**—it is not advised that you leave its completion until the last minute.
6. Please follow all instructions for each problem carefully, again ensuring that all requested files are in their correct locations—i.e. within subdirectories of `/phys210/$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`.
7. Do not do any of your work, or save any files, in your home directory on `hyper`, or anywhere else that is accessible by your fellow students.
8. 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: In your `hw2` directory create a subdirectory `a1`. Using Chapter 2 of the *Maple 9 Learning Guide*, make a facsimile of the `Maple` worksheet I went through in class on Sep. 24 and 29 and save it as `a1.mws` in that subdirectory (i.e. as `hw2/a1/a1.mws`). Note that both my worksheet and Chapter 2 of the *Learning Guide* are available online in PDF form via `Course Home Page -> Course Notes -> Maple -> Worksheet [PDF 16 pages] ... Chapter 2 [PDF 38 pages] of Maple 9 Learning Guide ...` You should refer to both of these documents while doing this exercise.

You are to work though Chapter 2 *in its entirety*, essentially entering everything that follows a `Maple` prompt (`>`) into your worksheet. Note, however, that there is one example—namely the use of *side relations* in conjunction with `simplify`—that returns an entirely different result in `Maple 12` than it did in `Maple 9`. You should omit this example, as I did. There are also places where the output from `Maple 12` will be slightly different than that from `Maple 9`. Again, I have tried to point out most of these instances via comments in my worksheet, but if you encounter others that I have missed, don't worry about it.

Your worksheet should include annotations corresponding to the various sections and sub-sections of the Chapter, as mine does. Note that I did the annotations in a very simple-minded fashion, by inserting text blocks into the worksheet. This can be done by first selecting the execution group (any part of the worksheet delimited by a left square bracket, `[`) where you want to insert text: select the execution group by single clicking with the left mouse button on the bracket. Then, from the menu bar at the top, choose `Insert -> Paragraph -> Before` or `Insert -> Paragraph -> After` depending on where you want the text to go. The cursor will then be positioned within this new paragraph (and outside any execution groups), and you can type in the annotation. You can also control the appearance of the annotation by highlighting the text (hold down the left mouse button and sweep over the text), then selecting the font, font size and style (bold, italic etc.) from the pull down menus and buttons that appear whenever you are entering or manipulating text. Type

```
> ?worksheet[reference[contextbar]]
```

for additional details.

A more sophisticated way to do at least some of the annotation would be to use `xmaple`'s sectioning capabilities. Type

```
> ?sections[overview]
```

for details about this facility, should you want to use it.

Finally, please observe the cautions made above 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.

Problem 2: Make the subdirectory `hw2/a2`, and within that subdirectory, and using `xmaple`, create a worksheet called `a2.mws` 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 (including the $O(x^{10})$ term), for $0 \le x \le 0.01$ (2.5)

Since there may be confusion about this point, note that your answer to item (2.4), *must* include the explicit form of the $O(x^{10})$ term, i.e. the leading-order *error* term should be $O(x^{11})$. For item (2.5) 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, you *can* convert a series into a polynomial using the `convert` command. Type

```
> ?convert[polynom]
```

for full details.

Problem 3: Write `Maple` procedures with headers and functionalities as follows

1. `lreverse := proc(l::list)`
   `reverse` returns a list in which the elements of its list argument `l` appear in reverse order.

   Examples:

```
> lreverse( [1, 2, 3, 4] );
      [4, 3, 2, 1]

> lreverse( [ [a, b], c, {d,e,f}, 4, [g, h, i] ] );
      [[g, h, i], 4, {d, e, f}, c, [a, b]]

> lreverse( [] );
      []

> lreverse( a );
Error, invalid input: lreverse expects its 1st argument, l,
to be of type list, but received a
```

2. `ljoin := proc(l1::list, l2::list)`
   `ljoin` returns a list which contains the elements of list `l1` followed by the elements of list `l2` (the original orderings of elements of both lists is maintained).

   Examples:

```
> ljoin( [1, 2, 3, 4] , [10, 11, 12, 13]);
      [1, 2, 3, 4, 10, 11, 12, 13]

> ljoin( [1, 2, 3, 4], [] );
      [1, 2, 3, 4]

> ljoin( [], [1, 2, 3, 4] );
      [1, 2, 3, 4]

> ljoin( [1, 2, 3], b );
Error, invalid input: ljoin expects its 2nd argument, l2, to be of
type list, but received b
```

3. `lprod := proc(l::list(algebraic))` $\cdots$
   `lprod` returns the product of all elements in the input list, `l`.

   Examples:

```
> lprod( [2, 3, 6] );
      36

> lprod( [x, y, z] );
       x y z

> lprod( [a, [1,2], b] );
Error, invalid input: lprod expects its 1st argument, l, to be of
type list(algebraic), but received [a, [1, 2], b]
```

   If `l` is the null list, `lprod` should call `error("argument is null list")`.

Test your procedures thoroughly with various input—invalid as well as valid—including null lists (`[]`). Note that the *only* place that a null list should be treated as an invalid argument is in `lprod`. Also note that, for the most part, `Maple`'s built in type-checking will ensure that invalid input is automatically detected and reported. All three procedure definitions should be commented (the commenting doesn't have to be extensive), and must be prepared in a *single* `Maple` source file (plain text file) called `hw2/a3/procs`. The TA and I must be able to read `hw2/a3/procs` into a `maple` or `xmaple` session using the `read` command without encountering errors. We will test your procedures with our own input.

Problem 4: 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

$$v(t) = v_0 + \int_{t_0}^{t} a(t')\,dt' \tag{1}$$

$$s(t) = s_0 + \int_{t_0}^{t} v(t')\,dt' \tag{2}$$

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/a4`, create a `Maple` procedure with the following header

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

where

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

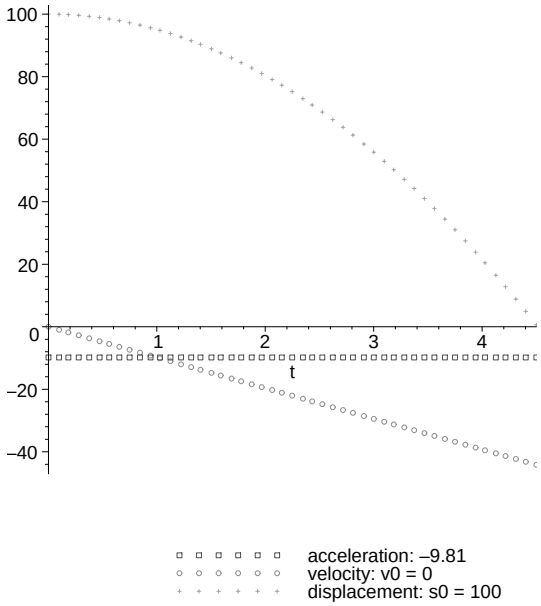
Save your definition of `kinematics` , along with any supporting procedures you write, in a *text file* called `kinematics` (i.e. in `hw2/a4/kinematics`). As with the previous question, the TA and I must be able to read your `kinematics` file into `xmaple` running on `hyper` without encountering errors.

Using (1) and (2), 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 will also probably find it useful to try `?plot[details]`, as well as to refer to other specific help pages covering the intricacies of `plot`.

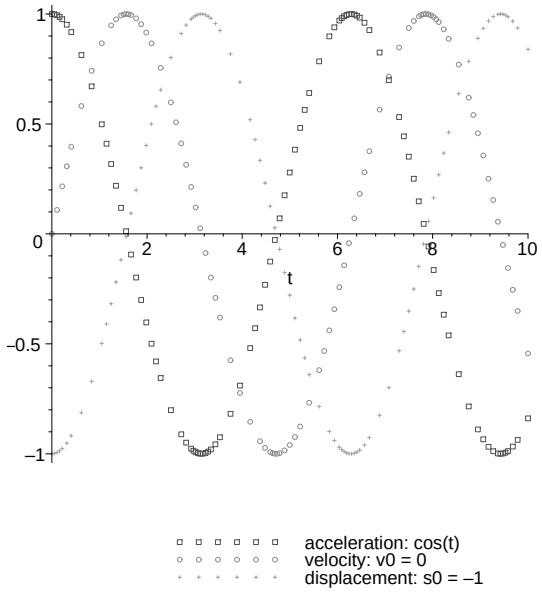
The plots that `kinematics` generates should include legends that duplicate, as closely as possible, those produced by my implementation. per the examples shown on the next page. To that end you may wish to consider getting help on the topics `plot[legend]` and `sprintf` using `?plot[legend]` and `?sprintf`, respectively.

Here are some sample invocations along with the corresponding output (which, again, in all cases, is a single plot which will appear within the `xmaple` worksheet in which it is executed—no other output is required). Observe the use of `Time`, 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 symbols that my implementation employs (see `?plot[options]`).

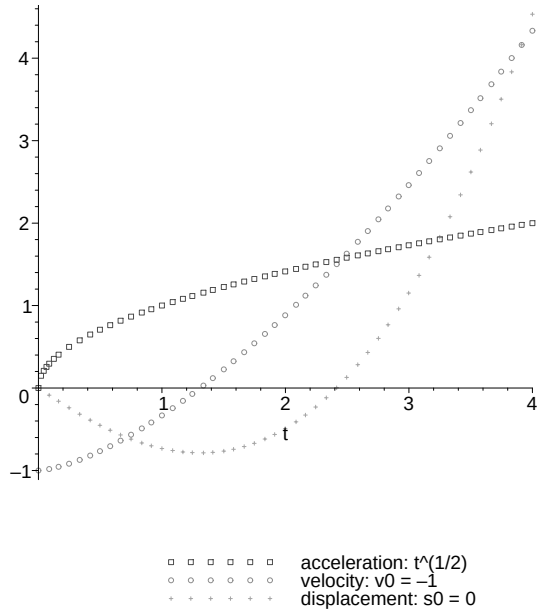
```
> kinematics(t->-9.81, t, 100, 0, 0, 4.5);
```



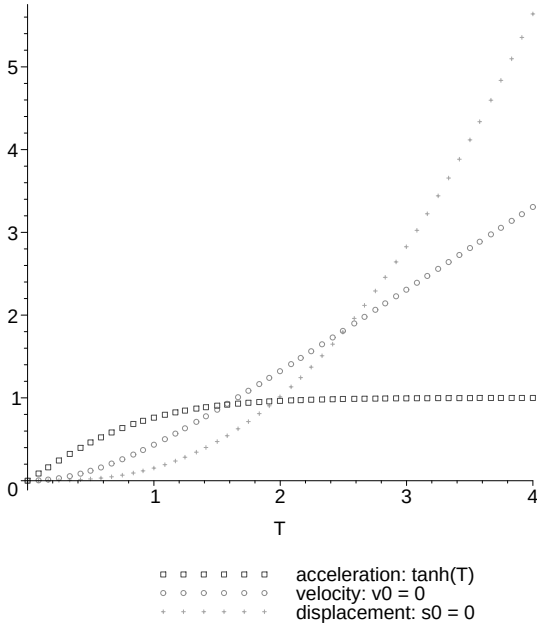

```
> kinematics(t->cos(t), t, -1, 0, 0, 10);
```




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




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



Problem 5: In subdirectory `hw2/a5` create a `bash` script called `calc` that uses `maple` (i.e. command-line `Maple`) to perform a floating point evaluation of essentially any `Maple` expression that *will* evaluate to a floating point value, and which returns that value on standard output.

Specifically, `calc` is to have the following usage

```
usage: calc expr [digits]

       Uses maple to evaluate expr in the floating point domain,
       and to the precision specified by digits: digits defaults to 12,
       and must satisfy 1 <= digits <= 50.
```

Note that the first argument, `expr`, which should evaluate to a floating point result in `Maple`, is *required*, but that the second argument, `digits`, is *optional* and should default to 12. Observe that the default for `calc` is thus *different* than the "standard" `Maple` default, which is 10 digits.

As usual, a few example invocations should aid in understanding how the script is to function:

```
# Invocation with no arguments, usage message is triggered.
% calc
usage: calc expr [digits]

       Uses maple to evaluate expr in the floating point domain,
       and to the precision specified by digits: digits defaults to 12,
       and must satisfy 1 <= digits <= 50.

# Compute Pi to calc's default number of digits (12)
% calc Pi
3.14159265359

% calc Pi 40
3.1415926535897932384626433832795028841 97

% calc Pi 60
calc: digits must be <= 50.

% calc Pi 0
calc: digits must be >= 1.

# In the next invocation, expr (i.e. sin(x)) does NOT evaluate to a
# floating point value, and calc simply produces an empty line of
# output, denoted [empty line].  Also note the use of single quotes
# around sin(x) to prevent shell evaluation of the special characters
# ( and )

% calc 'sin(x)'
                        <- empty line

% calc 'sin(0.2)^3' 30
.784137974753707229432310892408e-2

% calc 'exp(3.2*cos(0.6))/5!' 50
.11690217850242085009432241803609746795583660538774

% calc 'exp(2000)' 20
.3881180194284368576 5e869
```

Again, just so there's no confusion, your script should *not* produce the output `<- empty line` when `calc 'sin(x)'` is executed!! It should simply produce an empty line, or, equally acceptable, *no* line of output.

Notes and hints:

1. Your script should include some comments, but, again, the commenting does not need to be extensive.
2. The *only* error checking that your implementation of `calc` need do is to:
   (a) Ensure that the script is invoked with one or two arguments.
   (b) Ensure that the second argument is an integer that is not less than 1, nor greater than 50. If the argument *is* less than 1 or greater that 50, the appropriate error message appearing in the sample invocations above should be generated, and the script should exit.

   You can *assume* that the second argument, if supplied, *is* an integer. Thus, for example, if you invoke `calc` as follows,

   ```
   % calc Pi forty
   calc: line 30: test: forty: integer expression expected
   ```

   then it is perfectly acceptable for `calc` to output an error message such as the above, and nothing else.

   Most importantly, you do *not* need to do any checking to see whether `expr` is a valid `Maple` expression, and, if it is, whether it will evaluate to a floating point value. In other words, this is a case where your key task is to make sure that your version of `calc` does the correct thing when given valid input. For almost all cases of *invalid* input, you simply don't have to worry about what the script does.
3. Note that command-line `maple` reads its input from standard input and writes its output to standard output, so that you can use input redirection (including a here document) to supply input to `maple`, and, if necessary, output redirection to capture the output in a file. Note that if you want to combine a here document that uses `cat` with output redirection, or a pipeline, the appropriate syntax is of the form

   ```
   cat<<END > outputfile
       .
       .
   END
   ```

   and

   ```
   cat<<END | command1 | command2 ...
       .
       .
   END
   ```

   respectively; i.e. the output redirection (including `>>` and/or the pipeline) should appear on the *same line* as the `cat` command.
4. Your script needs to isolate the actual floating point number generated by `Maple`'s evaluation of `expr` in the floating point domain. To this end, consider doing the following:
   (a) Assigning the `Maple` expression that does the floating point evaluation to some fixed variable name, such as `ans`. Thus, part of the input to `maple` that `calc` generates would always be of the form

   ```
   ans := ...
   ```

   where `...` denotes the appropriate `Maple` expression.
   (b) If you follow this suggestion, you should be able to isolate the *single line of output* from `maple` that contains the desired floating point number using a pipeline of `grep` commands.
   (c) To help you complete the isolation of the floating point number, I have written a simple script, called `strip-non-numeric` which lives in ~`phys210/bin`, and thus should be in your path. `strip-non-numeric` is a *filter*, which means that it reads from standard input and writes to standard output. Its purpose is to strip (remove) any contiguous sequence of non-numeric characters from the beginning of any line of input, then dump the stripped output to standard output. Here's a usage example, along with the resulting output, where `echo` is used to provide the standard input to the script

   ```
   % echo 'ans := 2.3456' | strip-non-numeric
   2.3456
   ```

You are free, and in fact encouraged, to use `strip-non-numeric` in your implementation of `calc`

(If you're dying to understand how `strip-non-numeric` works, you can do a little research on the "stream editor", `sed`, which is a relative of `vi`. `man sed` will get you going, but you can also just ask me should you be interested.)

(d) To disable `maple`'s default "two-dimensional pretty-printed" output of expressions such as

```
> exp(2000.0);
                                                    869
                                      0.3881180194 10
```

begin your input to `maple` with the statement

```
interface(prettyprint=0);
```

If you don't do this, then your script very likely will not work properly for invocations such as

```
% calc 'exp(2000)' 20
```

i.e. for invocations that generate floating point values whose magnitudes are very large or very small.

5. Remember to add the `-x` option to the `bash` interpreter while you are developing your script, i.e. use

```
#! /bin/bash -x
```

as the first line of the script, so that execution tracing is enabled.

6. You may find it useful to experiment interactively with `maple` (i.e. command-line `Maple`) to get a feel for the typical output that is produced when assigning the evaluation of an expression in the floating point domain to `ans`, or whatever variable you choose to use per hint 4 (a) above. Don't forget to execute

```
interface(prettyprint=0);
```

before you begin any such experimentation.