PHYS 410/555: Computational Physics   Fall 2004   Homework 2
DUE: Thursday, October 14, 10:00 AM   Report bugs to choptuik@physics.ubc.ca

Important: *The following assignment requires*

1. *Working with the* `xmaple` *graphical user interface (GUI) to produce a* `Maple` *worksheet (Problem 1).*
2. *Preparing source code for* `Maple` *procedures in plain-text files that can be input into* `maple` *or* `xmaple` *via the* `read` *command (Problems 2, 3, 4 and 5).*

To complete problem 1 (i.e. the one requiring the GUI), I suggest you use `xmaple` *via the console of one of the* `lnx` *machines, via one of the PC "X-terms" in Henn 205 or via your own machine, If you are using one of the X-terms, you will need to* `ssh` *into one of the* `lnx` *machines, and then start up* `xmaple`*, as* `maple/xmaple` *is not installed on* `lts1`*. Note that if you create the worksheet on some machine other than* `lnx[123]`*, you should still ensure that* `xmaple` *on the* `lnx` *machines can read the worksheet, and that the execution of the worksheet on the* `lnx` *machines is as you expect.*

Whenever working with **ANY** worksheet in `xmaple`, be sure to save your work frequently, using, for example, `Ctrl-S`.

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.*

Problems 2, 3, 4 and 5 do not *require the use of the worksheet interface (GUI), and you should thus be able to complete them using "command-line"* `maple`*, from any shell running on the* `lnx` *machines.*

Please follow all instructions below carefully, and ensure that all requested files are in their correct locations **within your** `lnx` **account** *(with their correct names!) when you have completed the assignment.*

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: Using Chapter 2 of the *Maple V Learning Guide*, make and save a facsimile of the `Maple` worksheet I went through in class. Note that a Postscript version of my worksheet is available on-line via the *Class Notes* web page—please refer to that document as well as the *Learning Guide* itself 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 are several examples that do not work as documented in the *Learning Guide.* You should omit these examples, as I did. Your worksheet should include annotations corresponding to the various sections and sub-sections of the Chapter, as mine does. Observe that complete instructions for adding comments, headings, titles, etc. are available via `Maple`'s on-line help facility. (For example, in `Maple 8`, bring up the main help window—by selecting `Introduction` from the `Help` menu—click on the `Worksheet Interface` hyperlink, then `Overview of Document Processing`, then `Insert Elements into a Worksheet`, etc.) When you have finished, ensure that your worksheet, or a copy thereof, is saved as `~/hw2/a1/a1.mws` on your `lnx` account. (Note that `.mws` is the standard extension for `xmaple` worksheet files.) Also observe the cautions above concerning (a) the time it may take to complete this problem, and (b) the frequent use of `Ctrl-S`, or some other save mechanism.

Problem 2: Write `Maple` procedures as follows:

1. `luniq := proc(l::list)` ⋯

   `luniq` returns `true` if and only if all elements of `l` are distinct (i.e. not equal to another list element). If `l` is the empty list, `luniq` returns true.

   Examples:

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

   > luniq([1,2,3,1]);
           false

   > luniq([diff(exp(sin(x)), x$6), cos(x), diff(diff(exp(sin(x)), x$5), x)]);
           false

   > luniq([]);
           true
   ```

2. `lpair := proc(l1::list, l2::list)` ⋯

   If `l1` and `l2` are both lists of length `N`, `lpair` returns a new list, also of length `N`, whose `i`-th element is the 2-element list `[ l1[i] , l2[i] ]`.

   Examples:

   ```
   > lpair([w, x, y, z],[1, 2, 3, 4]);
           [[w, 1], [x, 2], [y, 3], [z, 4]]

   > lpair([],[]);
           [];

   > lpair([1, 2, 3],[a, b]);
   Error, (in lpair) input lists are not of equal length
   ```

3. `lreduce := proc(l1::list, l2::list(binarynumeric))` ⋯

   The input parameters `l1` and `l2` must be non-null lists of equal length. Further, each element of `l2` must be of type `binarynumeric`, that is, either `0` or `1`. You must make the datatype `binarynumeric` known to `type` by defining the procedure `` `type/binarynumeric` ``:

```
`type/binarynumeric` := proc <your-definition-here> end;
```

In the above, "<your-definition-here>" is to be replaced with appropriate Maple code.

Once `` `type/binarynumeric` `` has been appropriately defined, type should work as follows:

```
> type(0, binarynumeric);
       true

> type(1, binarynumeric);
       true

> type(2, binarynumeric);
       false

> type(a, binarynumeric);
       false
```

Given that the above constraints are satisfied, the output of lreduce is a list consisting of those elements of l1 which correspond to elements of l2 which are equal to 1 (with the order of elements of l1 preserved in the output list).

Examples:

```
> lreduce([1,4,2,3],[0,1,0,1]);
         [4, 3]

> lreduce([1,4,2,3],[0,0,0,1]);
         [3]

> lreduce([1,4,2,3],[0,0,0,0]);
         []

> lreduce([1,4,2,3],[0,0,1]);
Error, (in lreduce) input lists are not of equal length

> lreduce([],[]);
Error, (in lreduce) null list input is invalid

> lreduce([1,4,2,3],[0,2,0,1]);
Error, lreduce expects its 2nd argument, l2, to be of type
   list(binarynumeric), but received [0, 2, 0, 1]
```

Ensure that your procedures are suitably "bullet-proof"; test them thoroughly with various input—invalid as well as valid—including null lists ([]). Have your routines output error messages via ERROR when appropriate, as shown in the above usage examples.

All three procedure definitions should be adequately commented, and must be prepared in a *single* Maple source file (plain text file) called ~/hw2/a2/procs. I must be able to read ~/hw2/a2/procs into a maple or xmaple session using the read command. Your procedures will be tested with my own input.

Problem 3a: Later in the course, we shall consider finite difference approximations (FDAs) to differential operators such as d/dx, d^2/dx^2, etc. We will generally formulate these approximations on a *uniform mesh* (or *uniform grid*, or *uniform lattice*) of points; that is on a *discrete* set of points, x_j, so that a continuum interval $[a, b]$ [1] is replaced with the discrete set

$$x_j \equiv a + jh, \;\; j = 0, 1, \cdots N - 1\,, \tag{3.1}$$

where N is the total number of lattice points (including the "boundary points" $x_0 = a$ and $x_{N-1} = b$), and the *mesh spacing* (or *grid spacing*, or *lattice spacing*), h, is given by

$$h = \frac{b-a}{N-1}\,.$$

Note that the definition (3.1) implies that the spacing between any and every consecutive pair of lattice points x_j and x_{j+1} is h—this is what is meant by a *uniform* mesh. We note that a significant motivating factor for the use of uniform meshes is that accurate finite difference expressions are especially easily formulated on them.

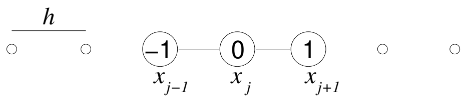
Having defined the discrete domain, or mesh, we consider discrete functions defined on the mesh, and adopt a standard finite-difference notation so that, e.g., $u_j \equiv u(x_j)$, denotes a grid-approximation to some continuum function $u(x)$. On a uniform lattice (and modulo possible problems—ignored here—near the end-points of the lattice), a particular FDA for a particular differential operator is *independent* of the specific lattice site at which the approximation is applied (i.e. the FDA is *translationally invariant*). [2] For example, as we will shortly establish, the following is a so-called second-order accurate approximation of $du(x)/dx|_{x=x_j}$:

$$\frac{u_{j+1} - u_{j-1}}{2h} \tag{3.2}$$

That is, we have

$$\frac{u_{j+1} - u_{j-1}}{2h} = \left.\frac{du}{dx}\right|_{x=x_j} + O(h^2)\,. \tag{3.3}$$

As a side-comment, we note at this point that it is often convenient to represent a formula such as (3.2) *pictorially*, using what is known variously as a finite difference *stencil*, *molecule* or *star*. For example, the above approximation could be represented as



where the circles indicate which of the particular grid function values (i.e. which of the $u_{j'}$) in the neighbourhood of x_j are involved in the approximation (the nearest neighbours in this case), and the coefficients (numbers) in the circles are the relative "weights" of the u_j in the FDA. (Note that (3.2) includes an overall factor or $1/2h$ which is *not* accounted for in the figure.)

Let us now turn to the task of verifying that (3.2) *is* an $O(h^2)$ approximation to $du(x)/dx|_{x=x_j}$. To do so, we consider Taylor series expansions about $x = x_j$, *where the expansion parameter is the appropriate multiple of the grid spacing*, h.[3] Specifically, we have

$$\begin{aligned}
u_{j+1} \equiv u(x_{j+1}) \equiv u(x_j + h) &= u_j + h\left.\frac{du}{dx}\right|_{x=x_j} + \frac{1}{2!}h^2\left.\frac{d^2u}{dx^2}\right|_{x=x_j} + \frac{1}{3!}h^3\left.\frac{d^3u}{dx^3}\right|_{x=x_j} + \frac{1}{4!}h^4\left.\frac{d^4u}{dx^4}\right|_{x=x_j} + O(h^5)\\
u_{j-1} \equiv u(x_{j-1}) \equiv u(x_j - h) &= u_j - h\left.\frac{du}{dx}\right|_{x=x_j} + \frac{1}{2!}h^2\left.\frac{d^2u}{dx^2}\right|_{x=x_j} - \frac{1}{3!}h^3\left.\frac{d^3u}{dx^3}\right|_{x=x_j} + \frac{1}{4!}h^4\left.\frac{d^4u}{dx^4}\right|_{x=x_j} + O(h^5)
\end{aligned}$$

[1] i.e. all x such that $a \le x \le b$, for given interval limits a and b

[2] Here I am assuming that the differential operator has *constant* coefficients.

[3] Ensure that you understand this point; contact the instructor, or ask a classmate if you don't.

Substituting the above expansions into (3.2) we find, after a little algebra, that

$$\frac{u_{j+1}-u_{j-1}}{2h} = \left.\frac{du}{dx}\right|_{x=x_j} + \frac{1}{6}h^2 \left.\frac{d^3u}{dx^3}\right|_{x=x_j} + O(h^4) = \left.\frac{du}{dx}\right|_{x=x_j} + O(h^2)$$

as advertised.

We now consider what is essentially a general form for an FDA of an abitrary constant-coefficient differential operator D. We posit that any FDA approximation of $D[u(x)]|_{x=x_j}$ can be written as

$$D[u(x)]|_{x=x_j} = \sigma \sum_{i=i_{\min}}^{i=i_{\max}} c_i u_{j+i} + O(h^p)\,.$$

Here $i_{\min}$, $i_{\max}$, $c_{i_{\min}} \cdots c_{i_{\max}}$, σ and p are all characteristics of the FDA, and are defined as follows:

1. $i_{\min}$ and $i_{\max}$ determine the "left" and "right" limits, respectively, of the difference stencil, so that the total width/diameter of the stencil is $i_{\max} - i_{\min} + 1$ (for (3.2) we thus have $i_{\min} = -1$,$i_{\max} = 1$ and a stencil diameter of 3 in accord with the above figure).

   An FDA is termed *centred* if

   (a) $i_{\min} = -i_{\max}$

   (b) The c_i are symmetric (antisymmetric) about $i = 0$ for even- (odd-) order derivatives. Note that all of the difference schemes used in this problem are centred.

2. The $c_{i_{\min}} \cdots c_{i_{\max}}$ are *integer* coefficients that determine the relative "weights" of the unknowns, u_{j+i}, that appear in the FDA.

3. σ is an overall scale factor, which invariably is of the form $1/(\kappa h^d)$ (why $1/h^d$?), where d is the differential *order* of the operator, D (e.g. $d/dx \Rightarrow d = 1$, $d^2/dx^2 \Rightarrow d = 2$, etc.), and the integer κ is chosen so that the c_i are all integers (so for (3.2) we have $d = 1$, $\kappa = 2$).

4. p is an integer that characterizes the *order* of accuracy of the approximation (in the continuum limit, $h \to 0$) and can be determined via Taylor series expansion as illustrated in the example above. We thus speak of an $O(h^2)$ scheme such as (3.2) as *second order*, an $O(h)$ FDA as *first order*, etc.

Now, observe that for each distinct FDA of the form (3.2), the coefficients $c_{i_{\min}} \cdots c_{i_{\max}}$ can conveniently be represented as a rank-1 `Maple` array. For example, to represent formula (3.2) we could define

```
> c := array(-1..1, [-1,0,1]);
```

with an associated scale factor $\sigma = 1/(2h)$.

Given this preamble, we finally arrive at the problem specification *per se*!

Write a `Maple` procedure with header

```
fdaeval := proc(c::array(integer), sigma::algebraic) ... end:
```

that uses Taylor series expansion about $x = x_j$ as in the above example to evaluate

$$D[u(x)]|_{x=x_j} = \sigma \sum_{i=i_{\min}}^{i=i_{\max}} c_i u_{j+i}\,,$$

thus returning explicit verification of (correctly!) constructed difference schemes. The following examples illustrate typical invocations and corresponding return values for the procedure:

```
> # O(h^2) centred approximation to first derivative
> a1 := array(-1..1, [-1,0,1]);
> fdaeval(a1,1/(2*h));
                                          (3)         2
                         D(u)(x) + 1/6 (D   )(u)(x) h

> # O(h^2) centred approximation to second derivative
> a2 := array(-1..1, [1,-2,1]);
> fdaeval(a2,1/(h^2));
                        (2)                   (4)         2
                      (D   )(u)(x) + 1/12 (D   )(u)(x) h

# O(h^2) centred approximation to third derivative
> a3 := array(-2..2, [-1,2,0,-2,1]);
> fdaeval(a3,1/(2*h^3));
                        (3)                  (5)         2
                      (D   )(u)(x) + 1/4 (D   )(u)(x) h
```

Notes and Hints

1. As in the above examples, your procedure should always return an expression which involves a differential operator applied to `u(x)`, and which is also a function of `h`; i.e. use *global* expressions (variables) for `u(x)` and `h` respectively.

2. Consider the use of `Maple` expressions such as

   ```
   > taylor(u(x+h), h, ord);
   ```

   where you must determine an appropriate value of `ord`, (note that "appropriate value" may depend on the length of the coefficient array, i.e. on the width, or diameter, of the FDA).

3. The output from `?array` is likely to be of help in completing this question.

4. The dollar (`$`) operator will "convert" a `Maple` range into the equivalent `Maple` series of integers, *vis*

   ```
   > rng := -2..2;
                 rng := -2 .. 2
   > $ rng;
                 -2, -1, 0, 1, 2
   ```

5. **IMPORTANT:** *No* error checking beyond that provided automatically by `Maple`'s argument-type checking facility is required in your implementation of `fdaeval`. Students sending e-mail regarding error checking in this problem are likely to be told to `RT(F)M :-)`.

Your implementation of a thoroughly tested `fdaeval`, including any auxiliary procedures you define (if any) must be prepared in the `Maple` source file (plain-text file) `~/hw2/a3/fd0`. Commenting of your procedure(s) can be minimal. You should also feel free to leave any `Maple` source files that you code for testing purposes in `~/hw2/a3/`. Such files, however, will not be graded.

Problem 3b: Code another version of `fdaeval` that has the following header

```
fdaeval := proc(rng::range, c::list(integer), sigma::algebraic) ... end:
```

with sample invocations/output as follows.

```
> # O(h^2) centred approximation to first derivative
> fdaeval(-1..1, [-1,0,1], 1/(2*h));
                                           (3)          2
                         D(u)(x) + 1/6 (D     )(u)(x) h

> # O(h^2) centred approximation to second derivative
> fdaeval(-1..1, [1,-2,1], 1/(h^2));
                        (2)                  (4)         2
                      (D    )(u)(x) + 1/12 (D    )(u)(x) h

# O(h^2) centred approximation to third derivative
> fdaeval(-2..2, [-1,2,0,-2,1], 1/(2*h^3));
                        (3)                 (5)          2
                      (D    )(u)(x) + 1/4 (D    )(u)(x) h
```

Follow the same instructions re testing/commenting etc. as for the first part of this problem, except use `fd1` (rather than `fd0`) as the filename of the `Maple` source file containing the definition of the new `fdaeval` and support procedures (if any).

In `~/hw2/a3/README`, comment on the relative ease of use ("naturalness") of the two implementations (i.e. which version do you think that a "generic" user is likely to prefer, and why?).

Hint: With judicious use of one or more local variables of the right type, your implementation of this version of `fdaeval` can be accomplished as a relatively minor modification of the version in **3a**.

Problem 4: Implement a `Maple` procedure that computes the unique polynomial (the Lagrange interpolating polynomial) of degree $n-1$ that passes through $[x_i, f(x_i)],\ i = 1 \cdots n$. Note that all of the x_i are assumed to be distinct. The procedure should have the header

```
polyinterp := proc(ldata::list(list), var::name) ...
```

`polyinterp` must return a polynomial in `var`; do not assume, for example, that `var` will always be "x". A sample invocation of `polyinterp` and the resulting output is:

```
> polyinterp([ [0,1], [1,6], [2,4], [3,0] ],'x');

       3      2
  5/6 x  - 6 x  + 61/6 x + 1
```

Prepare the procedure definition (adequately documented and with as much error-checking as possible) in the `Maple` source file (plain-text file) `~/hw2/a4/polyinterp`. Your routine will be tested with my own input. Note that I wrote (will write) this procedure in class; you are free to copy what I did there verbatim. However, you are encouraged to implement the procedure on your own, working from the basic mathematical description, also covered in class. Finally, my version does not exit with an error message if the x_i are not distinct; *yours MUST do so.*

Problem 5: (for 555 credit, *optional* for 410 students) From the following 3 dimension-full physical constants (values given in *SI* units):

- *Newton's gravitational constant:* $G = 6.673 \times 10^{-11}\,\mathrm{k}g^{-1}\mathrm{m}^3\mathrm{s}^{-2}$
- *Speed of light:* $c = 2.998 \times 10^8\,\mathrm{m\,s}^{-1}$
- *Planck's reduced constant:* $\hbar = 1.0546 \times 10^{-34}\,\mathrm{k}g\,\mathrm{m}^2\,\mathrm{s}^{-1}$

it is possible to compute a fundamental mass, length, time, density etc. known as the Planck mass, Planck length, Planck time, Planck density etc. More precisely, for any physical attribute with dimension

$$M^{\alpha_1}\, L^{\alpha_2}\, T^{\alpha_3} \tag{5.1}$$

where M, L, and T have the dimensions of mass, length, and time respectively, and the α_i are real constants, the associated Planck quantity has the same dimensions, and is generically given by

$$c^{\beta_1}\, \hbar^{\beta_2}\, G^{\beta_3} \tag{5.2}$$

for some to-be-determined real constants β_i. For example:

$$L \sim c^{-3/2}\, \hbar^{1/2}\, G^{1/2}$$

where the $\sim$ denotes "has the same dimensions". In *SI* units, then, the Planck length is 1.616×10^{-34}m.

Write a `Maple` procedure called `planck` that accepts algebraic expressions of the form (5.1) and returns the corresponding Planck quantity (5.2). You should first extend the `type` procedure to recognize a new type `MLTdim` that is any expression precisely of the form (5.1) (with *constant* α_i). Thus, for example,

```
> type(M, MLTdim);

            true

> type(M^2/(L*T), MLTdim);

            true

> type(2*L, MLTdim);

            false

> type(M^p1, MLTdim);

            false
```

Once you have extended `type` appropriately, note that you can use `Maple`'s type-checking facility by using a header of the form:

```
   planck := proc(x::MLTdim)
```

Finally, extend the floating-point evaluation routine, `evalf` so that it recognizes the constants `G, c` and `hbar` and returns their *SI* values (without dimensions) as given above. Prepare all of the procedures you write in a single file called `~/hw2/a5/planck`. Typical output from `planck` should look like this:

```
> # Input definitions from file `planck`
> read planck;

> planck(L);
                           1/2  1/2
                       hbar    G
                       ------------
                            3/2
                           c
> evalf(%);
                                  -34
                     .1616058223*10
```

Test your implementation thoroughly; it will be evaluated using input of my own design.