# PHYS 210: Introduction to Computational Physics Fall 2009 Homework 4

**Due: Wednesday, November 18, 11:59 PM**

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

*Please refer to the preamble for Homework 3, which is generally applicable to this Homework as well. In addition, please note the following.*

1. The following assignment requires you to write 1 `octave` function, `wave`, which is to be coded in the file `/phys210/$LOGNAME/hw4/a1/wave.m`.
2. As with the previous homework, there is an associated instructor-supplied "driver" script that you will use to generate the final results for the question.

   In this case the script is

   ```
   /home/phys210/octave/hw4/twave.m
   ```

   Once again, you can use this script as a guide / template for the design of your own script to test your implementation of `wave` as you code it.

   **Warning!!** If you *do* modify the script for your own purposes, ensure that you give it a *different* name. That is, the script `twave` that you must eventually execute to complete the homework *must* be the one defined in `/home/phys210/octave/hw4`, not your own version.
3. **Important!!** To ensure that you can execute the driver script from within `octave` (executing on `hyper`), add the following line to your ~`/.octaverc` file.

   ```
   addpath('/home/phys210/octave/hw4')
   ```

   This needs to be done *before* you start `octave` if you want the driver script to be available in that session.

   ```
   % This code is based in part on '<whatever>.m', written by M.W. Choptuik
   ```
4. As discussed in the e-mail that I sent on November 6, you are again free to use code that I have written to solve other problems as a template / guide for your implementation of `wave`. However, should you use that code in a substantial way (e.g. should you copy some specific file and then modify it to produce your `wave.m`), then, following usual academic protocol, you should acknowledge that fact in a comment near the top of the source file. For example
5. **Loopless coding**: Attempt to solve the problem using as few `for` or `while` loops as possible, but, as discussed below, if that proves difficult, feel free to at least start (and perhaps end) with a version that uses loops and references to individual array elements.

**PROBLEM 1:** *The Wave Equation*

## 1.1 Mathematical specification

The wave equation is a partial differential equation (PDE) that plays an extremely important role in mathematics and physics, and most of you will encounter it (and variations of it) many times in your future studies. Although some of the mathematics below may be unfamiliar to many of you, the finite difference solution of the (simple) wave equation that will be described is quite straightforward, and not significantly more involved than the solution of the advection equation that we have discussed in class.

The simplest wave equation is the following PDE in one spatial variable, $x$, and time, $t$, for a scalar function, $u(t,x)$:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}\,, \tag{1}$$

where $c$ is a positive constant whose significance we will discuss momentarily.

We will solve (1) on the domain

$$0 \le x \le 1 \qquad 0 \le t \le t_{\max}\,, \tag{2}$$

where $t_{\max}$ is the final time to which we wish to solve the equation.

In order to compute a specific solution of (1) we must provide both *initial conditions* and *boundary conditions.* Since the PDE (1) is second order in time (and in analogy to our treatment of the oscillator equation in the previous problem), for the initial conditions, we have to give values for the wave variable $u$ *and* its first time derivative at $t = 0$. That is, we must specify two *functions* $u_0(x)$ and $v_0(x)$ such that

$$\begin{aligned} u(0,x) &= u_0(x)\,, && (3)\\ \frac{\partial u}{\partial t}(0,x) &= v_0(x)\,. && (4)\end{aligned}$$

For boundary conditions, we will choose *fixed* or *Dirichlet* conditions—demanding that there be no wave motion at the end points, $x = 0$ and $x = 1$, of the spatial domain at any time:

$$\begin{aligned} u(t,0) &= 0\,, && (5)\\ u(t,1) &= 0\,. && (6)\end{aligned}$$

We note that we must then ensure that the initial condition (3) is compatible with the boundary conditions, i.e. we must have

$$u_0(0) = u_0(1) = 0\,. \tag{7}$$

If you wish to have a physical picture for this setup, imagine a string stretched snugly between two fixed attachment points. Assuming that the string is sufficiently elastic (i.e. so that it does not dissipate energy as it vibrates), then sufficiently small transverse displacements of the string (i.e. motions perpendicular to the line defined by the equilibrium configuration of the string) will be described by the above set of equations.

As you might expect (or know), the constant $c$ represents the speed at which disturbances in the wave variable propagate. As we discussed for the case of the advection equation, we can always choose a system of units in which this constant becomes 1, and since it is convenient both mathematically and computationally to do so, we will hereafter assume, without any loss of generality, that $c = 1$. Thus (1) becomes

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}\,. \tag{8}$$

Paralleling our study of the advection equation, if we momentarily consider the solution of (8) on an infinite spatial domain, $-\infty < x < \infty$, and for $t \ge 0$, we can write down the *general solution* of (8):

$$u(t,x) = l(x+t) + r(x-t)\,. \tag{9}$$

Here $l$ and $r$ are two *arbitrary* functions, each of which depends on a single independent variable. I will leave it as an exercise for you to convince yourself that (9) *does* satisfy (8)—if you are unable to do so, please see me during office hours or in one of the labs for an explanation.

Now, consider the case where $u(t,x)$ is such that $r(t,x) \equiv 0$. Then we have

$$u(t,x) = l(x+t)\,, \tag{10}$$

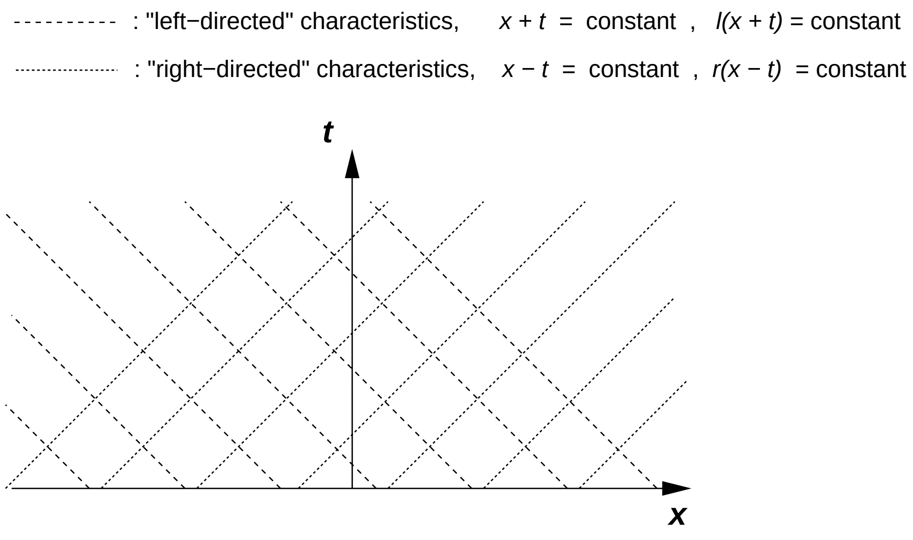
which is the same general solution that we derived for the advection equation, and thus represents propagation of the initial wave profile, $l(x)$, (i.e. $u(t,0) = l(x)$) *leftwards* with unit speed. Again, we identify the lines $t = -x + \text{const.}$ as *characteristics* of the wave equation, and for a solution of the form (10), the wave amplitude, $u$, is *constant* along these characteristics.

Similarly, if we assume that $u(t,x)$ is such that $l(t,x) \equiv 0$, we have

$$u(t,x) = r(x-t) \tag{11}$$

which now represents propagation of the initial profile, $r(x)$, *rightwards* with unit speed. For the wave equation there is thus a second set of characteristics defined by $t = +x + \text{const.}$ along which $u$ is constant for solutions of the form (11).

Fig. 1 provides a pictorial summary of our discussion of the general solution of the wave equation and its characteristic structure.



: "left−directed" characteristics, $x + t$ = constant , $l(x + t)$ = constant
: "right−directed" characteristics, $x - t$ = constant , $r(x - t)$ = constant
t
x

Figure 1: Characteristics of the wave equation: $u_{tt} = u_{xx}$. Signals (disturbances) travel along the characteristics (dashed and dotted lines).

Returning now to the solution of the wave equation on the (finite) domain given by (2), we note that we can use the notion of the left- and right-moving "profile" functions, $l$ and $r$, as a convenient way of specifying the initial conditions (3) and (4). First, for the initial value of the wave variable $u$ itself, we simply superimpose the two profile functions:

$$u(0,x) = l(x) + r(x)\,. \tag{12}$$

Second, to specify the first time derivative of $u$ at the initial time, we observe that since as a function of space *and* time we have $l = l(x+t)$ and $r = r(x-t)$, we get

$$\frac{\partial u}{\partial t}(0,x) = l'(x) - r'(x)\,, \tag{13}$$

where the prime ($'$) denotes ordinary differentiation of the functions $l$ and $r$ with respect to their single independent variable. Again, if you have difficulty understanding this last equation, be sure to ask me for a more detailed explanation.

In our numerical solution of the wave equation using finite difference techniques, we will use this technique of specifying (initially) left-moving and right-moving profiles to provide initial conditions. Moreover, we will restrict attention to one specific profile shape, namely that of a "gaussian", so that we will have

$$l(x) \quad = \quad a_l \exp\left[-\left(\frac{x - x_{0l}}{\delta_l}\right)^2\right]\,, \tag{14}$$

$$r(x) \quad = \quad a_r \exp\left[-\left(\frac{x-x_{0r}}{\delta_r}\right)^2\right] . \tag{15}$$

Here $a_l$, $a_r$, $x_{01}$, $x_{0r}$, $\delta_l$ and $\delta_r$ are to be viewed as parameters which control the amplitude, $a$, the effective width, $\delta$, and the "center point", $x_0$, of each profile.

For example, Fig. 2 shows the initial profile, $u(0,x)$, for the case $a_l = 1.0$, $x_{0l} = 0.45$, $\delta_l = 0.05$, $a_r = 0.5$, $x_{0r} = 0.55$ and $\delta_r = 0.1$ (which, not coincidentally, is the case implemented in the driver script for this problem `twave.m`).

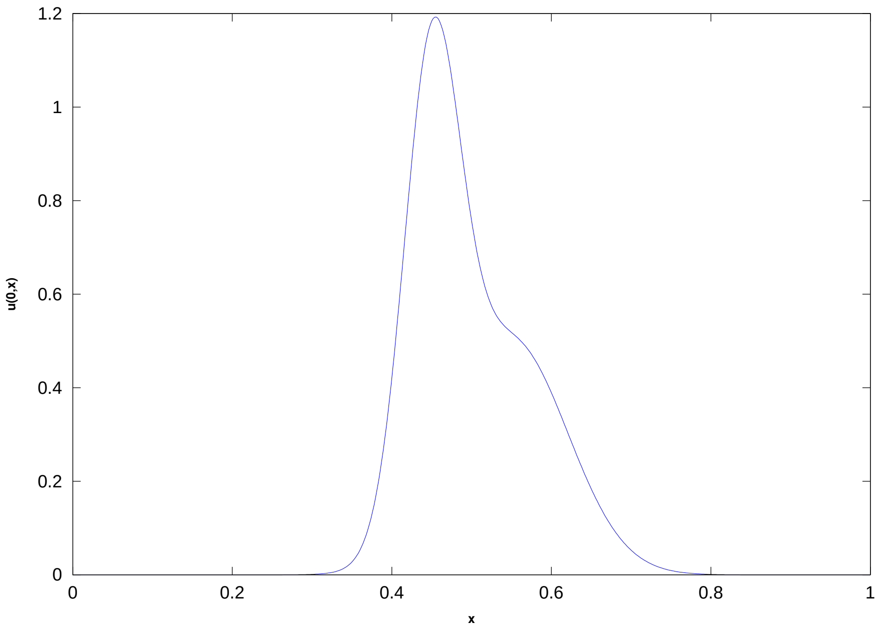


Figure 2: Typical initial data for the wave equation where both the initially left- and right-moving profiles, $l(x)$ and $r(x)$ are gaussians.

Let us now proceed to a discussion of the solution of (1)–(6) using finite difference techniques. (Note that the description of the mesh set-up is precisely the same as that which was discussed in the context of the advection equation, but is repeated here for the sake of completeness, as well as for the benefit of those of you who may not have taken complete notes in class!)

As always, we begin the discretization process by replacing the continuum domain (2) with a finite difference grid, $(t^n, x_j)$. For simplicity, as well as from considerations of accuracy, this mesh will have constant spacings, $\Delta t$ and $\Delta x$, respectively, in the two coordinate directions $t$ and $x$ (i.e. the grid is uniform). Moreover, the nature of solutions of wave equations such as (1) is such that if one uses the same order of finite difference approximation for both the time and space derivatives, then it is natural to choose the spacings $\Delta t$ and $\Delta x$ to be roughly of the same size. We thus introduce the parameter, $\lambda$, often known as the Courant number

$$\lambda \equiv \frac{\Delta t}{\Delta x}\,, \tag{16}$$

and note that when we perform convergence tests in which the mesh spacings are varied, $\lambda$ will be kept fixed. This means that even though superficially there seem to be two mesh parameters, $\Delta t$ and $\Delta x$, in reality the grid will be characterized by a *single* discretization scale, $h$, which we can conveniently identify with $\Delta x$. In particular, we will then always have

$$O(\Delta x) = O(\Delta t) = O(h)\,, \tag{17}$$

$$O(\Delta x^2) = O(\Delta t^2) = O(h^2)\,, \tag{18}$$

etc. Additionally, following our usual practice, we will specify the number of spatial points, $n_x$, using a level parameter, $\ell$, such that

$$n_x = 2^\ell + 1\,. \tag{19}$$

The spatial coordinates, $x_j$, of the grid are then given by

$$x_j = (j-1)\Delta x, \quad j = 1, 2, \ldots, n_x\,, \tag{20}$$

where

$$\Delta x = \frac{x_{\max} - x_{\min}}{n_x - 1} = \frac{1}{n_x - 1}\,, \tag{21}$$

and we have used the fact that we are solving the wave equation on the spatial interval $0 \le x \le 1$.

Similarly, the discrete time coordinates, $t^n$ are given by

$$t^n = (n-1)\Delta t, \quad j = 1, 2, \ldots, n_t\,, \tag{22}$$

where from (16) we have

$$\Delta t = \lambda \Delta x\,, \tag{23}$$

and there is thus an implicit assumption that we have chosen $\lambda$ and $t_{\max}$ so that $\Delta t$ as computed from (23) is consistent with the value that could also be calculated using

$$\Delta t = \frac{t_{\max}}{n_t - 1}\,. \tag{24}$$

Assuming that this *is* the case, the number of discrete time values ("time steps"), $n_t$ then becomes a *derived* quantity.

Having defined the finite difference mesh, we introduce the discrete approximation to the continuum solution of (1), denoting it by $u_j^n$:

$$u_j^n \equiv u((n-1)\Delta t, (j-1)\Delta x)\,. \tag{25}$$

We then discretize the second time and space derivatives in (1) using the standard $O(h^2)$ centred finite difference approximation for a second derivative (the same one used in the previous problem) to get our basic set of difference equations for the wave equation

$$\frac{u_j^{n+1} - 2u_j^n + u_j^{n-1}}{\Delta t^2} = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2}\,, \quad j = 2, 3, \ldots, n_x - 1, \quad n+1 = 3, 4, \ldots, n_t\,. \tag{26}$$

As was the case for the non-linear oscillator, we view (26) as defining algebraic equations for the advanced time unknowns

$$u_j^{n+1}\,, \quad j = 2, 3, \ldots, n_x - 1 \tag{27}$$

but in this instance I will leave it to you to do the simple algebra that solves (26) for $u_j^{n+1}$ (you will need the result in your implementation of the `octave` function `wave` that is prototyped below).

The system of equations (26) must be supplemented with discrete versions of the initial conditions (3)–(4) and the boundary conditions (5)–(6). Discrete boundary conditions are easy to "derive" in this case: we simply have

$$u_1^n = 0 \quad n = 1, 2, \ldots n_t\,, \tag{28}$$

$$u_{n_x}^n = 0 \quad n = 1, 2, \ldots n_t\,. \tag{29}$$

The discrete initial conditions are a little trickier to implement. Again, in precise analogy with our treatment of the Van der Pols equation, we have adopted a three-time-level finite difference approximation, and thus must use the continuum initial conditions (3)–(4), or equivalently (12)–(13) to initialize $u_j^1$ and $u_j^2$ for $j = 1, 2, \ldots n_x$. We also must ensure that the initialization is consistent with the boundary conditions—i.e. we must have $u_1^1 = u_{n_x}^1 = 0$.

Appropriate values of $u_j^1$ follow directly from (3)

$$u_j^1 = u_0(x_j)\,. \tag{30}$$

The tricky bit is working out sufficiently accurate values for $u_j^2$. Once more paralleling the development in the previous problem, we proceed by Taylor series expansion in $t$ about $t = 0$.

$$u_j^2 \equiv u(\Delta t, x_j) = u(0, x_j) + \Delta t \frac{\partial u}{\partial t}(0, x_j) + \frac{1}{2}\Delta t^2 \frac{\partial^2 u}{\partial t^2}(0, x_j) + O(\Delta t^3)\,. \tag{31}$$

We eliminate the second time derivative using the wave equation (1) itself to get

$$u_j^2 \equiv u(\Delta t, x_j) = u(0, x_j) + \Delta t \frac{\partial u}{\partial t}(0, x_j) + \frac{1}{2}\Delta t^2 \frac{\partial^2 u}{\partial x^2}(0, x_j) + O(\Delta t^3)\,. \tag{32}$$

Then, using (3) and (4) and neglecting the $O(\Delta t^3)$ and higher order terms, we have

$$u_j^2 = u_0(x_j) + \Delta t\, v_0(x_j) + \frac{1}{2}\Delta t^2 u_0''(x_j)\,, \tag{33}$$

where the prime ($'$) denotes ordinary differentiation (recall that $u_0$ is a function of only one independent variable).

In terms of the initially left- and right-moving profiles, $l(x)$ and $r(x)$, we can show that (30) and (33) become

$$u_j^1 = l(x_j) + r(x_j)\,, \tag{34}$$

$$u_j^2 = l(x_j) + r(x_j) + \Delta t\,[l'(x_j) - r'(x_j)] + \frac{1}{2}\Delta t^2\,[l''(x_j) + r''(x_j)]\,. \tag{35}$$

Now, if your eyes have completely glazed over by this point, the good news is that you won't have to concern yourself with all of these gory details concerning the proper initialization of the grid function values. Rather, as detailed below, you will simply need to call instructor-supplied `octave` functions that will return appropriate values for $u_j^1$ and $u_j^2$.

### 1.2 The problem *per se*

Make the directory `hw4/a1`, and within that directory create an `octave` source file, `wave.m`, that defines the function `wave` having the following header

```
function [x t u] = wave(tmax, level, lambda, al, x0l, deltal, ar, x0r, deltar)
```

The input arguments to `wave` are as follows

- `tmax`: $t_{\max}$, as defined above.
- `level`: The discretization level, $\ell$, as discussed above. Again, this defines the number of discrete spatial coordinates, $n_x = 2^\ell + 1$, and the mesh spacing $\Delta x = h = 1/(n_x - 1)$.
- `lambda`: The ratio of the temporal mesh spacing to the spatial mesh spacing, $\lambda = \Delta t/\Delta x$. You should always compute with a value of $\lambda < 1$ and in order to make it easy to ensure that `tmax` and `lambda` are consistent (see equations (23) and (24)), I suggest that you use $\lambda = 1/2$.
- `al, x0l, deltal`: The parameters, $a_l$, $x_{0l}$ and $\delta_l$ that define the initially left-moving gaussian profile (14).
- `ar, x0r, deltar`: The parameters, $a_r$, $x_{0r}$ and $\delta_r$ that define the initially right-moving gaussian profile (15).

*As usual, you do not have to do any error checking of the input arguments.*

Your implementation of `wave` must define the 3 output arguments as follows:

- `x`: A row vector of length $n_x$ defining the discrete spatial coordinates, $x_j$.
- `t`: A row vector of length $n_t$ defining the discrete time coordinates, $t^n$.
- `u`: A $n_t \times n_x$ matrix (two dimensional array) containing the complete finite difference solution, $u_j^n$.

**Implementation notes**

1. As mentioned above, you can use the following two `octave` functions to set the values $u_j^1$ and $u_j^2$
   (a) `function [u1] = waveu1(x, delt, al, x0l, deltal, ar, x0r, deltar)`
   (b) `function [u2] = waveu2(x, delt, al, x0l, deltal, ar, x0r, deltar)`

   For both functions, the input argument `x` is a row vector of length $n_x$ containing the discrete x coordinates, $x_j$, and `delt` is the time spacing $\Delta t$. All other input arguments are as per the discussion of the input arguments to `wave`. The output arguments are row vectors of length $n_x$ containing the values $u_j^1$ and $u_j^2$, respectively, as defined by (34) and (35).

2. Again, you can use the driver script `twave.m` located in `/home/phys210/octave/hw4/` as the basis for your own script and/or interactive experimentation to develop, debug and convergence test your implementation of `wave`. In particular, note that `twave.m` illustrates the use of the `octave` plotting function, `mesh`, that produces a "surface plot" ("3D plot") of a function defined on a mesh such as $(t^n, x_j)$. Although you may find it useful to generate your own surface plots while you are implementing `wave`, you should also consider using `plot` to view the grid function $u_j^n$ at specific (single) discrete times, $t^n$

3. There is a somewhat subtle point that you must be aware of if you use different values for the parameters defining the gaussian profiles (14) and (15). The observation is the following: although we have stressed above that the initial conditions must be consistent with the boundary conditions, which state that $u$ must vanish at $x = 0$ and $x = 1$ at all times, in reality the gaussians (14) and (15) only vanish in the limits $x \to \pm\infty$. In practice, provided that the centers $x_0$ and widths $\delta$ are chosen carefully, the values of $l(x)$ and $r(x)$ at the spatial boundaries will be so small that, for all intents and purposes, they can be considered to be 0. This is the case for the parameters defining the initial profile, $u(0, x)$, that is illustrated in Fig. 2. However, if you do *not* choose $x_0$ and/or $\delta$ appropriately (for either $l(x)$ or $r(x)$) then the initial data may effectively become discontinuous. The subsequent evolution will then tend to look rather strange, and the solution will certainly not converge as $h \to 0$ as expected.

4. Finally, as mentioned in the instruction page, you should attempt to code `wave` using as few `for` or `while` loops as possible—but don't go overboard! (my solution, for example, has one `for` loop). If you find this difficult, go ahead and write a version of `wave` that uses loops, but once you have it working, and assuming you have time, try to code another version (without destroying the working code!) that doesn't use so many loops. Such a strategy has the advantage that you will have a working code to generate results that you can use to test the new program.

When you are satisfied that you have implemented `wave` correctly, execute the driver `twave` which should produce output as follows:

```
>> twave
Running wave at level 6
Generating surface plot and saving it as wave-6.ps
This will take some time (about 15 seconds).  Please be patient!
Type 'Enter' to continue:
Checking code convergence
Running wave at level 6
Running wave at level 7
Running wave at level 8
Running wave at level 9
Running wave at level 10
Scaled RMS values of soln differences: ...

Type 'Enter' to continue:

Done!!
```

Again, some output which is part of the solution has been suppressed. You should also see one nifty surface plot appear on your screen as the script executes.

**File Inventory for directory** `hw4/a1`

1. `wave.m`
2. `wave-6.ps`