Lab_7

October 18, 2015

1 Laboratory #7: Solving partial differential equations using an explicit, finite difference method.

Lin Yang & Susan Allen & Carmen Guo

1.1 Contents

1.2 List of Problems

- Problem 7: Finite difference form of equations
- Problem 8: Dispersion relation for D-grid
- Problem 9: Accuracy of the approximation on various grids

2 1. Objectives

2.1 1.1 Sections 3 through 6

When you have completed these sections you will be able to:

- find the dispersion relation for a set of differential equations (the "real" dispersion relation).
- find the dispersion relation for a set of difference equations (the numerical dispersion relation).
- describe a leap-frog scheme
- construct a predictor-corrector method
- use the given differential equations to determine unspecified boundary conditions as necessary
- describe a staggered grid
- state one reason why staggered grids are used
- explain the physical principle behind the CFL condition
- find the CFL condition for a linear, explicit, numerical scheme
- state one criteria that should be considered when choosing a grid

2.2 1.2 Section 7

TBA

3 2. Readings

These are the suggested readings for this lab. For more details about the books and papers, click on the reference link.

- **Rotating Navier Stokes Equations**
  - Pond and Pickard, 1983, Chapters 3,4 and 6
- **Shallow Water Equations**
  - Gill, 1982, Section 5.6 and 7.2 (not 7.2.1 etc)
- **Poincaré Waves**
  - Gill, 1982, Section 7.3 to just after equation (7.3.8), section 8.2 and 8.3
- **Introduction to Numerical Solution of PDE's**
  - Press et al, 1992, Section 17.0
- **Waves**
  - Cushman-Roision, 1994, Appendix A

```
In [1]: from IPython.display import Image
        import IPython.display as display
        # import plotting package and numerical python package for use in examples later
        import matplotlib.pyplot as plt
        # make the plots happen inline
        % matplotlib inline
        # import the numpy array handling library
        import numpy as np
        # import the quiz script
        from numlabs.lab7 import quiz7 as quiz
        # import the pde solver for a simple 1-d tank of water with a drop of rain
        from numlabs.lab7 import rain
        # import the dispersion code plotter
        from numlabs.lab7 import accuracy2d
        # import the 2-dimensional drop solver
        from numlabs.lab7 import interactive1
        # import the 2-dimensional dispersion relation plotter
        from numlabs.lab7 import dispersion_2d
```

4 3. Physical Example, Poincaré Waves

One of the obvious examples of a physical phenomena governed by a partial differential equation is waves. Consider a shallow layer of water and the waves on the surface of that layer. If the depth of the water is much smaller than the wavelength of the waves, the velocity of the water will be the same throughout the depth. So then we can describe the state of the water by three variables: $u(x, y, t)$, the east-west velocity of the water, $v(x, y, t)$, the north-south velocity of the water and $h(x, y, t)$, the height the surface of the water is deflected. As specified, each of these variables are functions of the horizontal position, (x, y) and time t but, under the assumption of shallow water, not a function of z.

In oceanographic and atmospheric problems, the effect of the earth's rotation is often important. We will first introduce the governing equations including the Coriolis force (Full Equations). However, most of the numerical concepts can be considered without all the complications in these equations. We will also consider two simplier sets; one where we assume there is no variation of the variables in the y-direction (No variation in y) and one where, in addition, we assume that the Coriolis force is negligible (Simple Equations).

The solution of the equations including the Coriolis force are Poincaré waves whereas without the Coriolis force, the resulting waves are called shallow water gravity waves.

The remainder of this section will present the equations and discuss the dispersion relation for the two simplier sets of equations. If your wave theory is rusty, consider reading Appendix A in Cushman-Roisin, 1994.

4.0.1 Full Equations

The linear shallow water equations on an f-plane over a flat bottom are

(Full Equations, Eqn 1)

$$\frac{\partial u}{\partial t} - fv = -g\frac{\partial h}{\partial x}$$

(Full Equations, Eqn 2)

$$\frac{\partial v}{\partial t} + fu = -g\frac{\partial h}{\partial y}$$

(Full Equations, Eqn 3)

$$\frac{\partial h}{\partial t} + H\frac{\partial u}{\partial x} + H\frac{\partial v}{\partial y} = 0$$

where

- $\vec{u} = (u, v)$ is the horizontal velocity,
- f is the Coriolis frequency,
- g is the acceleration due to gravity,
- h is the surface elevation, and
- H is the undisturbed depth of the fluid.

We will return to these equations in section 7. Full Equations.

4.0.2 No variation in y

To simplify the problem assume there is no variation in y. This simplification gives:

(No variation in y, first eqn)

$$\frac{\partial u}{\partial t} - fv = -g\frac{\partial h}{\partial x}$$

(No variation in y, second eqn)

$$\frac{\partial v}{\partial t} + fu = 0$$

(No variation in y, third eqn)

$$\frac{\partial h}{\partial t} + H\frac{\partial u}{\partial x} = 0$$

4.0.3 Simple Equations

If we consider waves in the absence of the earth's rotation, $f = 0$, which implies $v = 0$ and we get

$$\frac{\partial u}{\partial t} = -g\frac{\partial h}{\partial x}$$

$$\frac{\partial h}{\partial t} + H\frac{\partial u}{\partial x} = 0$$

These simplified equations give shallow water gravity waves. For example, a solution is a simple sinusoidal wave:

(wave solution- h)

$$h = h_0 \cos(kx - \omega t)$$

(wave solution- u)

$$u = \frac{h_0 \omega}{kH} \cos(kx - \omega t)$$

where h_0 is the amplitude, k is the wavenumber and ω is the frequency (See Cushman-Roisin, 1994 for a nice review of waves in Appendix A).

Substitution of (wave solution- h) and (wave solution- u) back into the differential equations gives a relation between ω and k. Confirm that

(Analytic Dispersion Relation)

$$\omega^2 = gHk^2,$$

which is the dispersion relation for these waves.

4.0.4 No variation in y

Now consider $f \neq 0$.

By assuming

$$h = h_0 e^{i(kx-\omega t)}$$

$$u = u_0 e^{i(kx-\omega t)}$$

$$v = v_0 e^{i(kx-\omega t)}$$

and substituting into the differential equations, eg, for (No variation in y, first eqn)

$$-iwu_0 e^{i(kx-\omega t)} - fv_0 e^{i(kx-\omega t)} + ikgh_0 e^{i(kx-\omega t)} = 0$$

and cancelling the exponential terms gives 3 homogeneous equations for u_0, v_0 and h_0. If the determinant of the matrix derived from these three equations is non-zero, the only solution is $u_0 = v_0 = h_0 = 0$, NO WAVE! Therefore the determinant must be zero.

4.0.5 Quiz: Find the Dispersion Relation

What is the dispersion relation for 1-dimensional Poincare waves?

A) $\omega^2 = f^2 + gH(k^2 + \ell^2)$

B) $\omega^2 = gHk^2$

C) $\omega^2 = f^2 + gHk^2$

D) $\omega^2 = -f^2 + gHk^2$

In the following, replace 'x' by 'A', 'B', 'C' or 'D' and run the cell.

```
In [2]: print (quiz.dispersion_quiz(answer = 'x'))
```

```
Acceptable answers are 'A', 'B', 'C' or 'D'
```

5 4. Numerical Solution

5.0.1 Simple Equations

Consider first the simple equations with $f = 0$. In order to solve these equations numerically, we need to discretize in 2 dimensions, one in space and one in time. Consider first the most obvious choice, shown in Figure Unstaggered Grid.

```
In [3]: Image(filename='images/nonstagger.png',width='40%')
```

```
Out[3]:
```

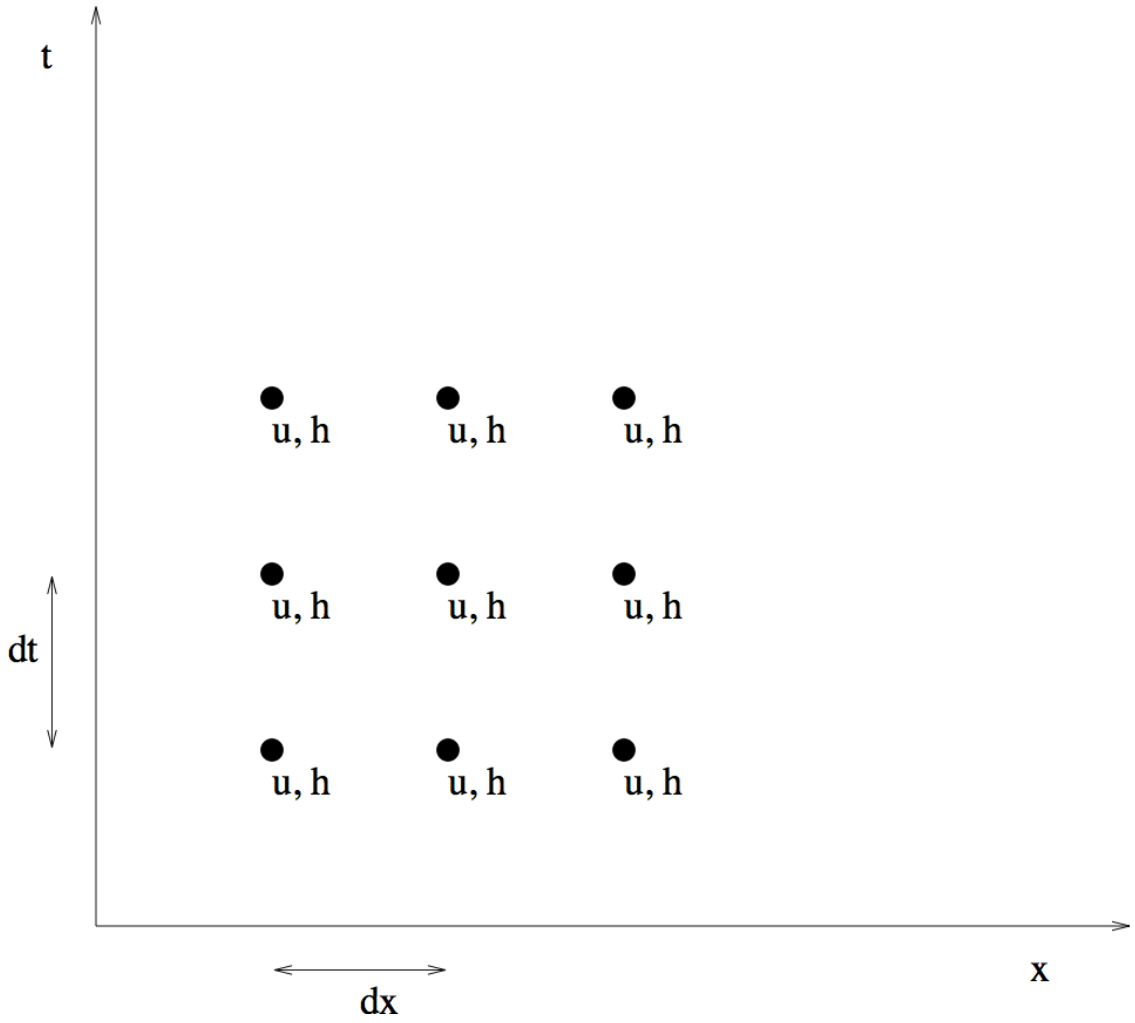


Figure Unstaggered Grid.
We will use centred difference schemes in both x and t. The equations become:
(Non-staggered, Eqn One)

$$\frac{u(t+dt,x)-u(t-dt,x)}{2dt}+g\frac{h(t,x+dx)-h(t,x-dx)}{2dx}=0$$

(Non-staggered, Eqn Two)

$$\frac{h(t+dt,x)-h(t-dt,x)}{2dt}+H\frac{u(t,x+dx)-u(t,x-dx)}{2dx}=0$$

We can rearrange these equations to give $u(t+dt,x)$ and $h(t+dt,x)$. For a small number of points, the resulting problem is simple enough to solve in a notebook.

For a specific example, consider a dish, 40 cm long, of water 1 cm deep. Although the numerical code presented later allows you to vary the number of grid points, in the discussion here we will use only 5 spatial points, a distance of 10 cm apart. The lack of spatial resolution means the wave will have a triangular shape. At $t=0$ a large drop of water lands in the centre of the dish. So at $t=0$, all points have zero velocity and zero elevation except at $x=3dx$, where we have

$$h(0,3dx)=h_0=0.01cm$$

A centred difference scheme in time, such as defined by equations (Non-staggered, Eqn One) and (Non-staggered, Eqn Two), is usually refered to as a *Leap frog scheme*. The new values, $h(t+dt)$ and $u(t+dt)$ are equal to values two time steps back $h(t-dt)$ and $u(t-dt)$ plus a correction based on values calculated one time step back. Hence the time scheme "leap-frogs" ahead. More on the consequences of this process can be found in section 8.4 Computational Mode.

As a leap-frog scheme requires two previous time steps, the given conditions at $t=0$ are not sufficient to solve (Non-staggered, Eqn One) and (Non-staggered, Eqn Two). We need the solutions at two time steps in order to step forward.

5.1 4.1 Predictor-Corrector to Start

In section 4.2.2 of Lab 2, predictor-corrector methods were introduced. We will use a predictor-corrector based on the forward Euler scheme, to find the solution at the first time step, $t=dt$. Then the second order scheme (Non-staggered, Eqn One), (Non-staggered, Eqn Two) can be used.

Using the forward Euler Scheme, the equations become

$$\frac{u(t+dt,x)-u(t,x)}{dt}+g\frac{h(t,x+dx)-h(t,x-dx)}{2dx}=0$$

$$\frac{h(t+dt,x)-h(t,x)}{dt}+H\frac{u(t,x+dx)-u(t,x-dx)}{2dx}=0$$

1. Use this scheme to predict u and h at $t=dt$.

2. Average the solution at $t=0$ and that predicted for $t=dt$, to estimate the solution at $t=\frac{1}{2}dt$. You should confirm that this procedure gives:

$$u(\frac{dt}{2})=\begin{cases}0 & x=3dx\\ (-gh_0dt)\,/\,(4dx) & x=2dx\\ (gh_0dt)\,/\,(4dx) & x=4dx\end{cases}$$

$$h(\frac{dt}{2})=\begin{cases}h_0 & x=3dx\\ 0 & x\neq 3dx\end{cases}$$

3. The corrector step uses the centred difference scheme in time (the leap-frog scheme) with a time step of $dt/2$ rather than dt. You should confirm that this procedure gives:

$$u(dt)=\begin{cases}0 & x=3dx\\ (-gh_0dt)\,/\,(2dx) & x=2dx\\ (gh_0dt)\,/\,(2dx) & x=4dx\end{cases}$$

$$h(dt)=\begin{cases}0 & x=2dx,4dx\\ h_0-\left(gHdt^2h_0\right)\,/\,\left(4dx^2\right) & x=3dx\end{cases}$$

Note that the values at $x=dx$ and $x=5dx$ have not been specified. These are boundary points and to determine these values we must consider the boundary conditions.

5.2 4.2 Boundary Conditions

If we are considering a dish of water, the boundary conditions at $x=dx,5dx$ are those of a wall. There must be no flow through the wall.

$$u(dx)=0$$

$$u(5dx)=0$$

But these two conditions are not sufficient; we also need h at the walls. If $u = 0$ at the wall for all time then $\partial u/\partial t = 0$ at the wall, so $\partial h/\partial x = 0$ at the wall. Using a one-sided difference scheme this gives

$$\frac{h(2dx) - h(dx)}{dx} = 0$$

or

$$h(dx) = h(2dx)$$

and

$$\frac{h(4dx) - h(5dx)}{dx} = 0$$

or

$$h(5dx) = h(4dx)$$

which gives the required boundary conditions on h at the wall.

5.3 4.3 Simple Equations on a Non-staggered Grid

1. Given the above equations and boundary conditions, we can find the values of u and h at all 5 points when $t = 0$ and $t = dt$.

2. From (Non-staggered, Eqn One) and (Non-staggered, Eqn Two), we can find the values of u and h for $t = 2dt$ using $u(0, x)$, $u(dt, x)$, $h(0, x)$, and $h(dt, x)$.

3. Then we can find the values of u and h at $t = 3dt$ using $u(dt, x)$, $u(2dt, x)$, $h(dt, x)$, and $h(2dt, x)$.

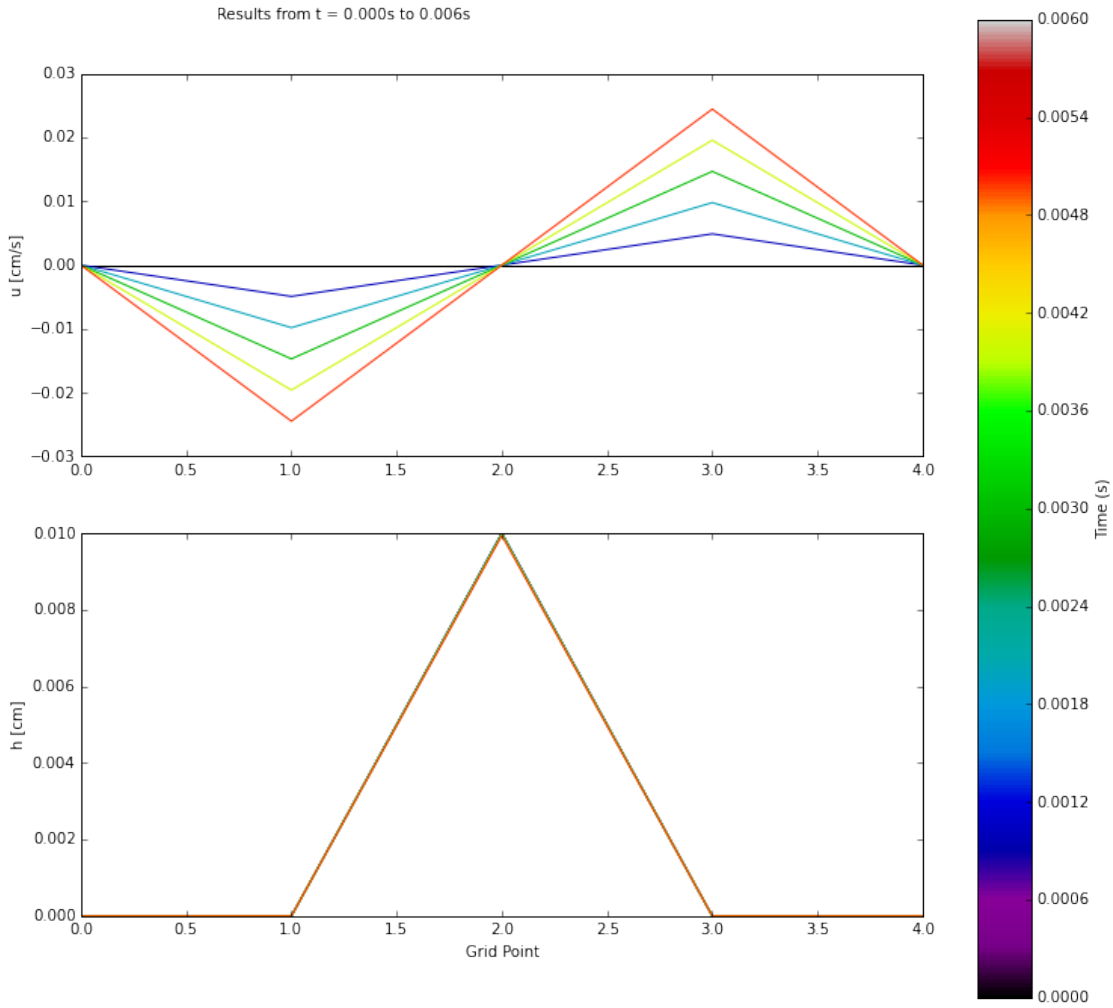
We can use this approach recursively to determine the values of u and h at any time $t = n * dt$. The python code that solves this problem is provided in the file rain.py. It takes two arguments, the first is the number of time steps and the second is the number of horizontal grid points.

The output is two coloured graphs. The color represents time with black the earliest times and red later times. The upper plot shows the water velocity (u) and the lower plot shows the water surface. To start with the velocity is 0 (black line at zero across the whole domain) and the water surface is up at the mid point and zero at all other points (black line up at midpoint and zero elsewhere)

Not much happens in 6 time-steps. Do try longer and more grid points.

```
In [4]: rain.rain([6, 5])
```

```
/Users/phil/miniconda3/lib/python3.4/site-packages/matplotlib/collections.py:590: FutureWarning: elemen
  if self._edgecolors == str('face'):
```



If you want to change something in the script (say the colormap I've chosen, spectral, doesn't work for you), you can edit rain.py in an editor or spyder. To make it take effect here though, you have to reload rain. See next cell for how to. You will also need to do this if you do problem one or other tests by changing rain.py but running in a notebook.

```
In [5]: import importlib
        importlib.reload(rain)
```

```
Out[5]: <module 'numlabs.lab7.rain' from '/Users/phil/repos/numeric/numlabs/lab7/rain.py'>
```

5.4 4.4 Staggered Grids

After running the program with different numbers of spatial points, you will discover that the values of u are always zero at the odd numbered points, and that the values of h are always zero at the even numbered points. In other words, the values of u and h are zero in every other column starting from $u(t, dx)$ and $h(t, 2dx)$, respectively.

A look at (Non-staggered, Eqn One) and (Non-staggered, Eqn Two) can help us understand why this is the case:

$u(t + dt, x)$ is dependent on $h(t, x + dx)$ and $h(t, x - dx)$,

but $h(t, x + dx)$ is in turn dependent on u at $x + 2dx$ and at x,

and $h(t, x - dx)$ is in turn dependent on u at $x - 2dx$ and at x.

Thus, if we just look at u at a particular x, $u(x)$ will depend on $u(x + 2dx)$, $u(x - 2dx)$, $u(x + 4dx)$, $u(x - 4dx)$, $u(x + 6dx)$, $u(x - 6dx)$, ... but not on $u(x + dx)$ or $u(x - dx)$. Therefore, the problem is actually decoupled and consists of two independent problems: one problem for all the u's at odd numbered points and all the h's at even numbered points, and the other problem for all the u's at even numbered points and all the h's at odd numbered points, as shown in Figure Unstaggered Dependency.

In [6]:
```
Image(filename='images/dependency.png',width='50%')
```

Out[6]:

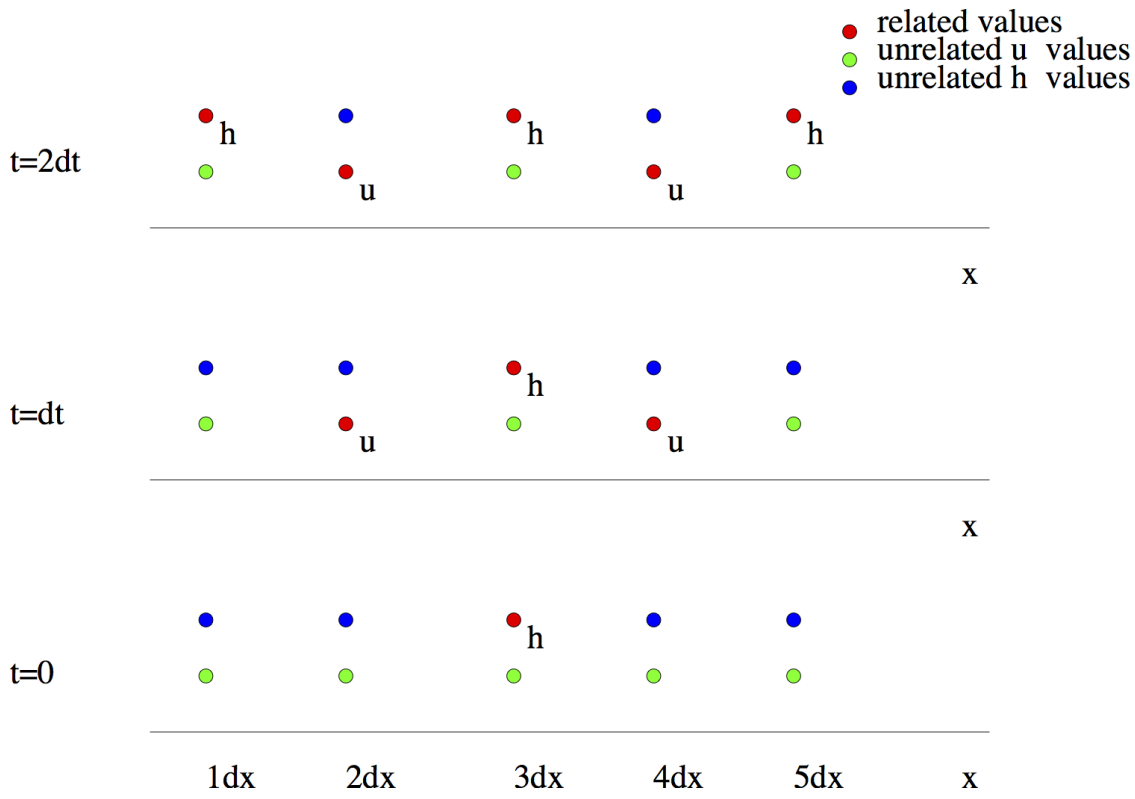


Figure Unstaggered Dependency

In either problem, only the variable that is relevant to that problem will be considered at each point. So for one problem, if at point x we consider the u variable, at $x + dx$ and $x - dx$ we consider h. In the other problem, at the same point x, we consider the variable h.

Now we can see why every second u point and h point are zero for *rain*. We start with all of $u(dx), h(2dx), u(3dx), h(4dx), u(5dx) = 0$, which means they remain at zero.

Since the original problem can be decoupled, we can solve for u and h on each decoupled grid separately. But why solve two problems? Instead, we solve for u and h on a single staggered grid; whereas before we solved for u and h on the complete, non-staggered grid. Figure Decoupling shows the decoupling of the grids.

In [7]:
```
Image(filename='images/decoupling.png',width='50%')
```

Out[7]:

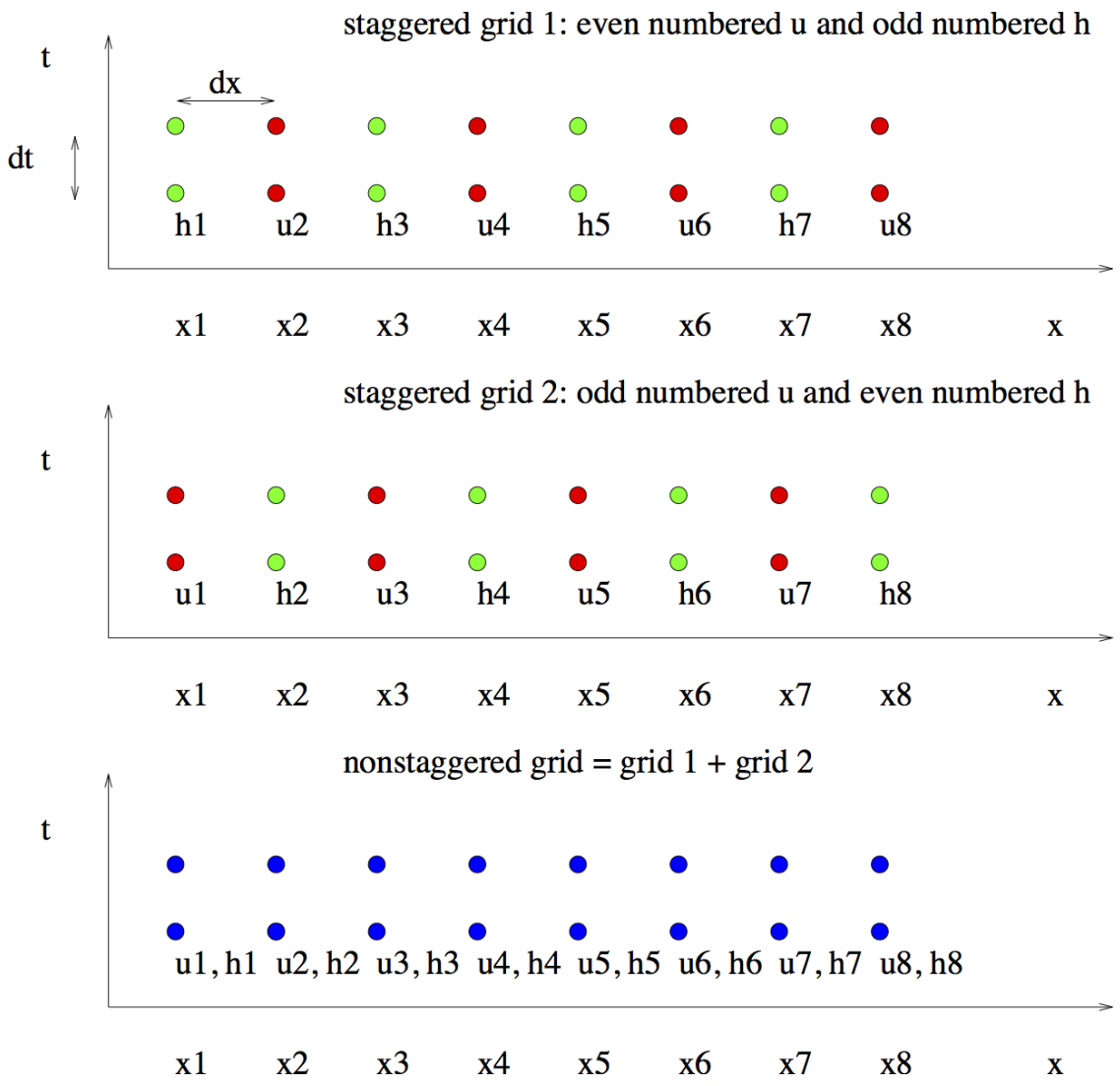


Figure Decoupling: The two staggered grids and the unstaggered grid. Note that the unstaggered grid has two variables at each grid/time point whereas the staggered grids only have one.

Now consider the solution of the same problem on a staggered grid. The set-up of the problem is slightly different this time; we are considering 4 spatial points in our discussion instead of 5, shown in Figure Staggered Grid. We will also be using h_i and u_i to denote the spatial points instead of $x = dx * i$.

```
In [8]: Image(filename='images/stagger.png',width='50%')
```

```
Out[8]:
```

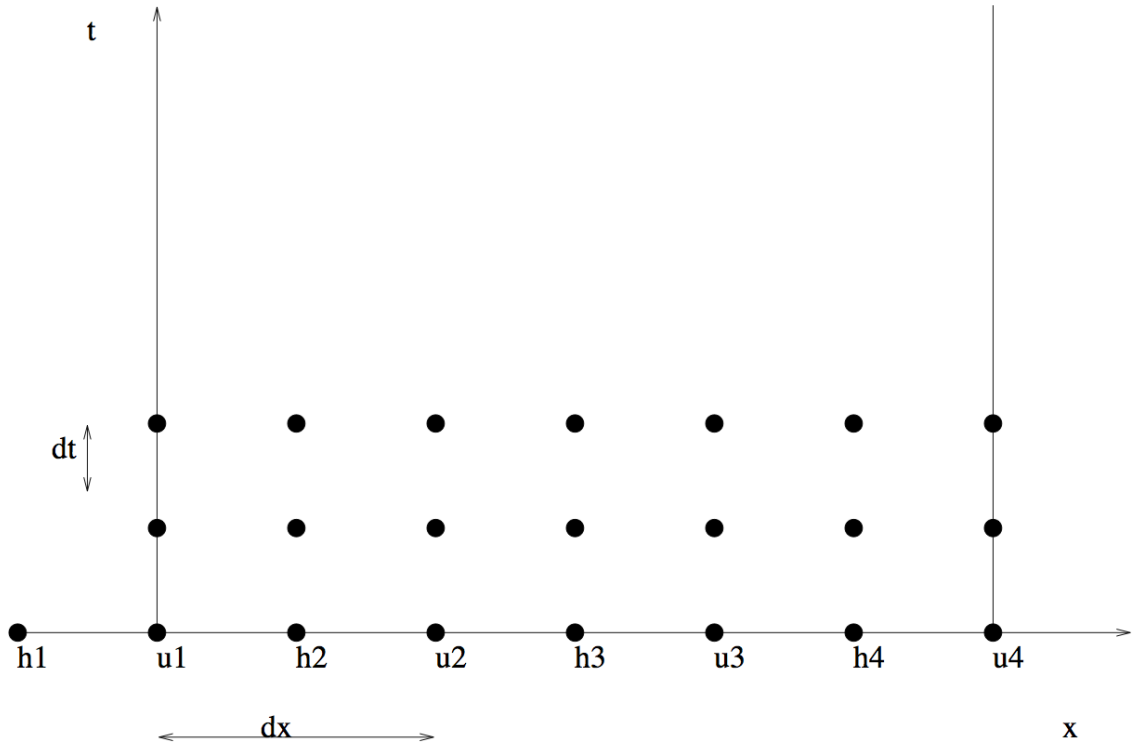


Figure Staggered Grid: The staggered grid for the drop in the pond problem.

The original equations, boundary and initial conditions are changed to reflect the staggered case. The equations are changed to the following:

(Staggered, Eqn 1)

$$\frac{u_i(t+dt) - u_i(t-dt)}{2dt} + g\frac{h_{i+1}(t) - h_i(t)}{dx} = 0$$

(Staggered, Eqn 2)

$$\frac{h_i(t+dt) - h_i(t-dt)}{2dt} + H\frac{u_i(t) - u_{i-1}(t)}{dx} = 0$$

The initial conditions are: At $t = 0$ and $t = dt$, all points have zero elevation except at h_3, where

$$h_3(0) = h_0$$

$$h_3(dt) = h_3(0) - h_0 H g \frac{dt^2}{dx^2}$$

At $t = 0$ and $t = dt$, all points have zero velocity except at u_2 and u_3, where

$$u_2(dt) = -h_0 g \frac{dt}{dx}$$

$$u_3(dt) = -u_2(dt)$$

This time we assume there is a wall at u_1 and u_4, so we will ignore the value of h_1. The boundary conditions are:

$$u_1(t) = 0$$

$$u_4(t) = 0$$

5.5 *Problem One*

Modify *rain.py* to solve this problem (Simple equations on a staggered grid). Submit your code and a final plot for one case.

6 5. Stability: the CFL condition

In the previous problem, $dt = 0.001s$ is used to find u and h. If you increase dt by 100 times, and run *rain.py* on your staggered grid code again, you can see that the magnitude of u has increased from around 0.04 to 10^8! Try this by changing $ dt = 0.001 $ to $ dt = 0.1 $ in the code and compare the values of u when run with different values of dt. This tells us that the scheme we have been using so far is unstable for large values of dt.

To understand this instability, consider a spoked wagon wheel in an old western movie (or a car wheel with a pattern in a modern TV movie) such as that shown in Figure Wheel.

```
In [9]: Image(filename='images/wheel_static.png',width='35%')
```

```
Out[9]:
```

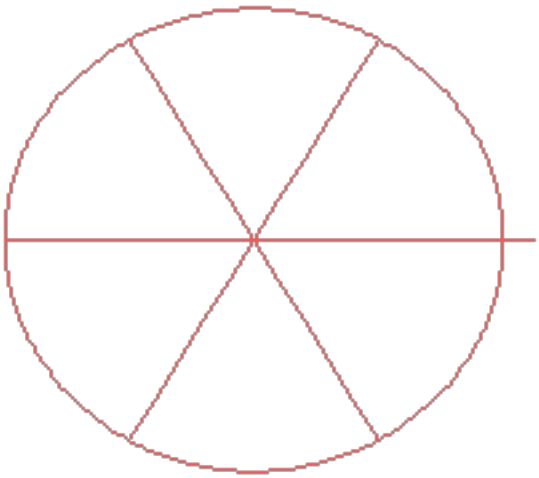


Figure Wheel: A spoked wagon wheel.

Sometimes the wheels appear to be going backwards. Both TV and movies come in frames and are shown at something like 30 frames a second. So a movie discretizes time. If the wheel moves just a little in the time step between frames, your eye connects the old position with the new position and the wheel moves forward − a single frame is shown in Figure Wheel Left.

```
In [10]: Image(filename='images/wheel_left.png',width='35%')
```

```
Out[10]:
```

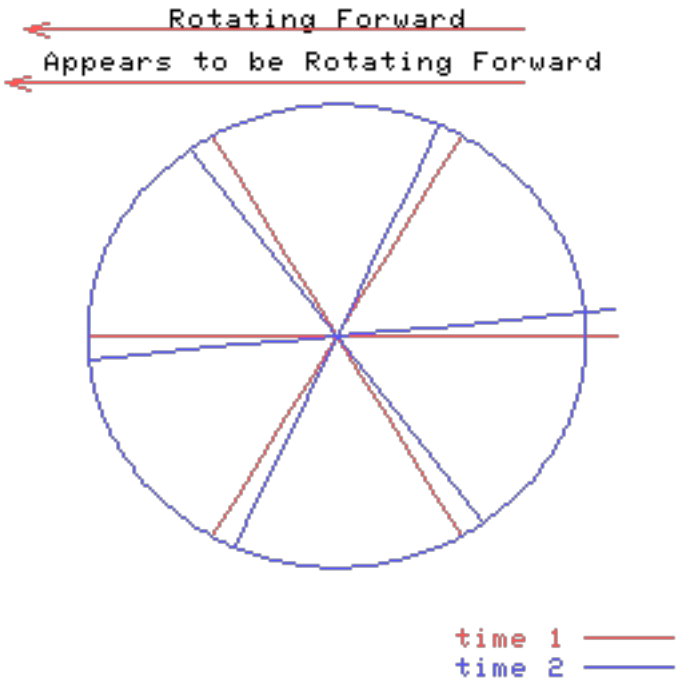


Figure Wheel Left: The wheel appears to rotate counter-clockwise if its speed is slow enough.

```
In [11]: vid = display.YouTubeVideo("hgQ66frbBEs", modestbranding=1, rel=0, width=500)
         display.display(vid)
```

```
<IPython.lib.display.YouTubeVideo at 0x1090cfd30>
```

However, if the wheel is moving faster, your eye connects each spoke with the next spoke and the wheel seems to move backwards − a single frame is depicted in Figure Wheel Right.

```
In [12]: Image(filename='images/wheel_right.png',width='35%')
```

```
Out[12]:
```

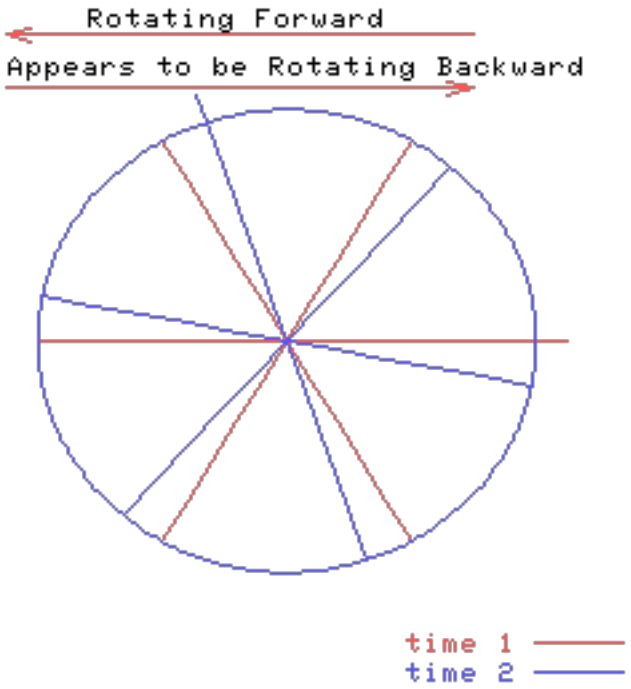


Figure Wheel Right: When the wheel spins quickly enough, it appears to rotate clockwise!

```
In [13]: vid = display.YouTubeVideo("w8iQIwX-ek8", modestbranding=1, rel=0, width=500)
         display.display(vid)
```

```
<IPython.lib.display.YouTubeVideo at 0x1090ecb38>
```

In a similar manner, the time discretization of any wave must be small enough that a given crest moves less than half a grid point in a time step. Consider the wave pictured in Figure Wave.

```
In [14]: Image(filename='images/wave_static.png',width='65%')
```

```
Out[14]:
```

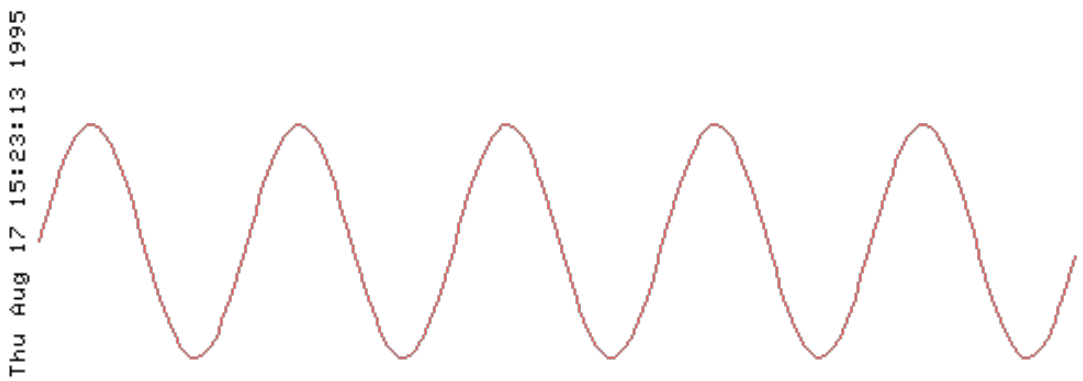


Figure Wave: A single frame of the wave.

If the wave moves slowly, it seems to move in the correct direction (i.e. to the left), as shown in Figure Wave Left.

```
In [15]: Image(filename='images/wave_left.png',width='65%')
```

Out[15]:

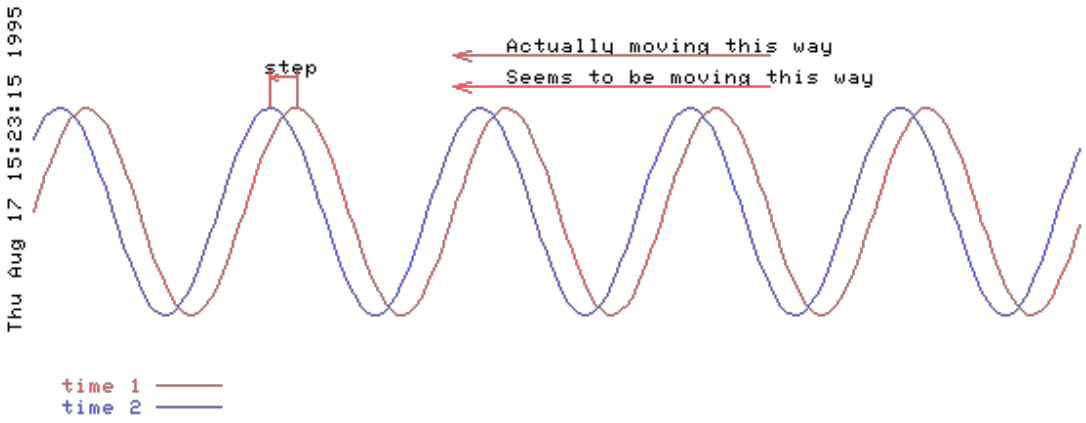


Figure Wave Left: The wave moving to the left also appears to be moving to the left if it's speed is slow enough.

```
In [16]: vid = display.YouTubeVideo("CVybMbfYRXM", modestbranding=1, rel=0, width=500)
         display.display(vid)
```

```
<IPython.lib.display.YouTubeVideo at 0x1090ec160>
```

However if the wave moves quickly, it seems to move backward, as in Figure Wave Right.

In [17]: Image(filename='images/wave_right.png',width='65%')

Out[17]:

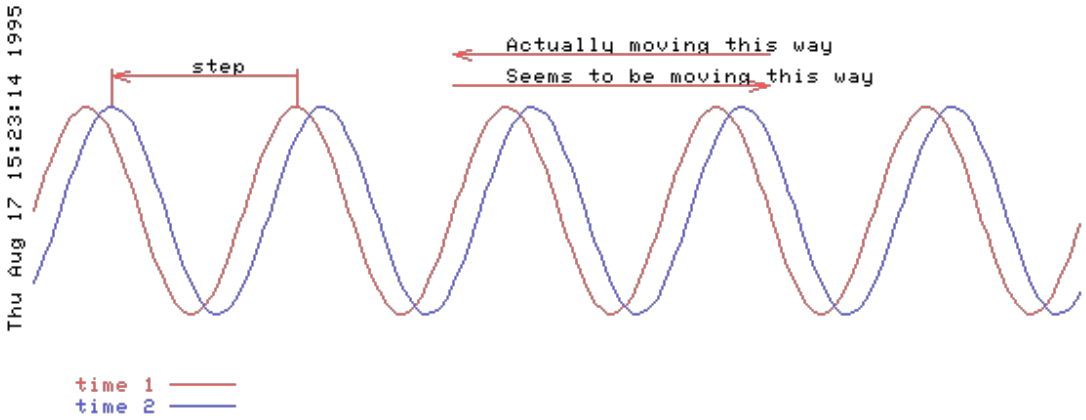


Figure Wave Right: If the wave moves too rapidly, then it appears to be moving in the opposite direction.

In [18]:
```
vid = display.YouTubeVideo("al2VrnkYyD0", modestbranding=1, rel=0, width=500)
display.display(vid)
```

```
<IPython.lib.display.YouTubeVideo at 0x10d23f898>
```

In summary, an explicit numerical scheme is unstable if the time step is too large. Such a large time step does not resolve the process being modelled. Now we need to calculate, for our problem, the maximum value of the time step is for which the scheme remains stable. To do this calculation we will derive the dispersion relation of the waves *in* the numerical scheme. The maximum time step for stability will be the maximum time step for which all waves either maintain their magnitude or decay.

Mathematically, consider the equations (Staggered, Eqn 1) and (Staggered, Eqn 2). Let $x = md$ and $t = p\,dt$ and consider a solution

(u-solution)

$$\begin{aligned} u_{mp} = & \quad \mathcal{R}e\{\mathcal{U}\exp[i(kx-\omega t)]\} \\ = & \quad \mathcal{R}e\{\mathcal{U}\exp[i(kmd-\omega p\,dt)]\} \\ h_{mp} = & \quad \mathcal{R}e\{\mathcal{H}\exp[i(k[x-dx/2]-\omega t)]\} \\ = & \quad \mathcal{R}e\{\mathcal{H}\exp[i(k[m-1/2]d-\omega p\,dt)]\} \end{aligned}$$

where $\mathcal{R}e$ means take the real part and $\mathcal{U}$ and $\mathcal{H}$ are constants. Substitution into (Staggered, Eqn 1) and (Staggered, Eqn 2) gives two algebraic equations in $\mathcal{U}$ and $\mathcal{H}$ which can be written:

$$\begin{bmatrix} -\sin(\omega dt)/dt & 2g\sin(kd/2)/d \\ 2H\sin(kd/2)/d & -\sin(\omega\,dt)/dt \end{bmatrix} \begin{bmatrix} \mathcal{U} \\ \mathcal{H} \end{bmatrix} = 0.$$

where $\exp(ikd) - \exp(-ikd)$ has been written $2i\sin(kd)$ etc. In order for there to be a non-trivial solution, the determinant of the matrix must be zero. This determinant gives the dispersion relation

(Numerical Dispersion Relation)

$$\frac{\sin^2(\omega\,dt)}{dt^2} = 4gH\frac{\sin^2(kd/2)}{d^2}$$

Which can be compared to the (Analytic Dispersion Relation), the "real" dispersion relation. In particular, if we decrease both the time step and the space step, $dt \to 0$ and $d \to 0$, (Numerical Dispersion Relation) approaches (Analytic Dispersion Relation). The effect of just the discretization in space can be found by letting just $dt \to 0$ which gives

(Continuous Time, Discretized Space Dispersion Relation)

$$\omega^2 = 4gH\frac{\sin^2(kd/2)}{d^2}$$

The "real" dispersion relation, (Analytic Dispersion Relation), and the numerical dispersion relation with continuous time, (Continuous Time, Discretized Space Dispersion Relation), both give ω^2 positive and therefore ω real. However, this is not necessarily true for the numerical dispersion relation (Numerical Dispersion Relation). What does a complex ω mean? Well, go back to (u-solution). A complex $\omega = a + ib$ gives $u \propto \exp(-iat)\exp(bt)$. The first exponential is oscillatory (like a wave) but the second gives exponential growth if b is positive or exponential decay if b is negative. Obviously, for a stable solution we must have $b \leq 0$. So, using (Numerical Dispersion Relation) we must find ω and determine if it is real.

Now, because (Numerical Dispersion Relation) is a transcendental equation, how to determine ω is not obvious. The following works:

- Re-expand $\sin(\omega\, dt)$ as $(\exp(i\omega\, dt) - \exp(-i\omega\, dt))/2i$.
- Write $\exp(-i\omega\, dt)$ as λ and note that this implies $\exp(i\omega\, dt) = 1/\lambda$. If $\omega dt = a + ib$ then $b = ln|\lambda|$. For stability the magnitude of λ must be less than one.
- Write $4gH\sin^2(kd/2)/d^2$ as q^2, for brevity.
- Substitute in (Numerical Dispersion Relation) which gives:

  $$-(\lambda - 1/\lambda)^2 = 4q^2dt^2$$

  or

  $$\lambda^4 - 2(1 - 2q^2dt^2)\lambda^2 + 1 = 0$$

  or

  (Lambda Eqn)

  $$\lambda = \pm\left(1 - 2q^2dt^2 \pm 2qdt\left(q^2dt^2 - 1\right)^{1/2}\right)^{1/2}$$

A plot of the four roots for λ is shown below in Figure Roots.

In [19]:
```
Image(filename='images/allmag.png',width='45%')
```

Out[19]:

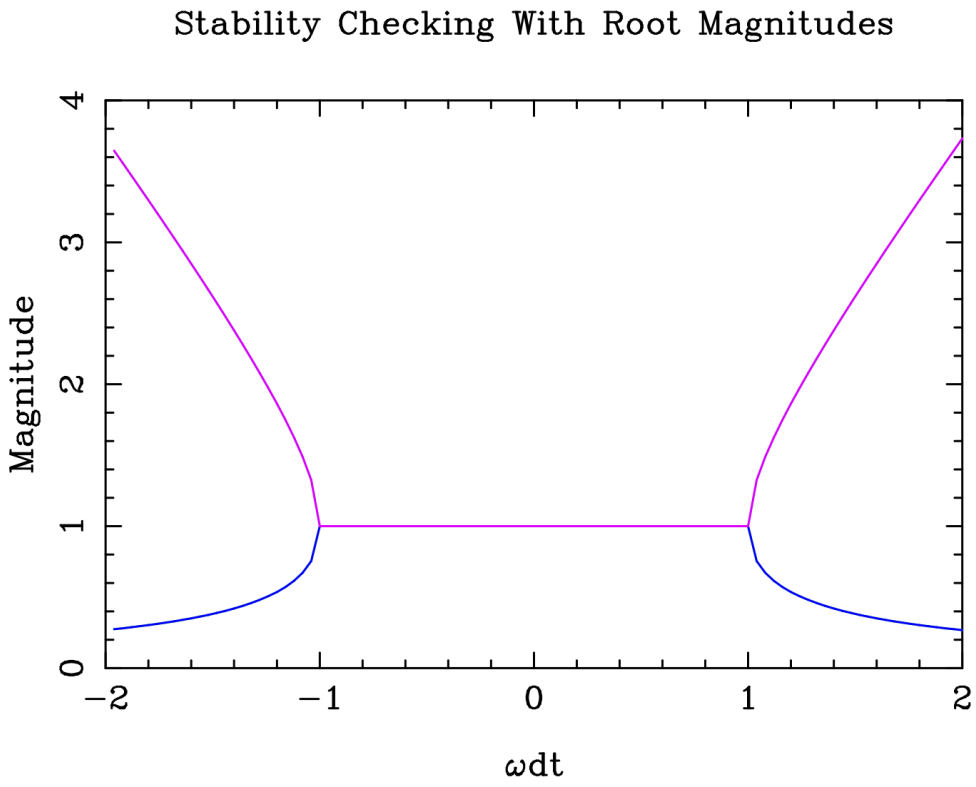


Figure Roots: Magnitude of the four roots of λ as a function of qdt (not ωdt).

The four roots correspond to the "real" waves travelling to the right and left, as well two *computational modes* (see Section 8.4 Computational Mode for more information). The plots for the four roots overlap, so it is most helpful to view separate plots for each of the roots.

```
In [20]: Image(filename='images/multimag.png',width='60%')
```

```
Out[20]:
```

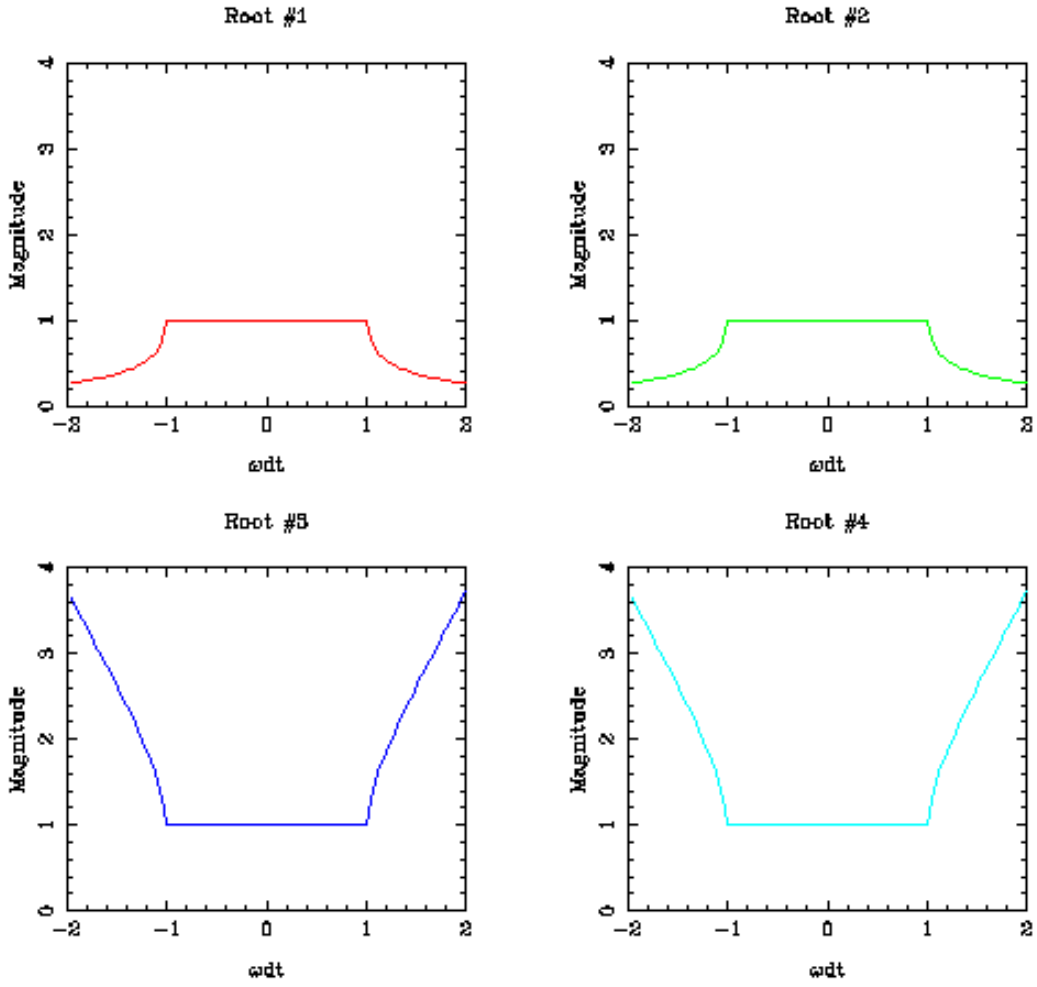


Figure Separate Roots: Magnitude of the four roots of λ as a function of qdt (not ωdt).

Now for stability λ must have a magnitude less than or equal to one. From Figure Roots, it is easy to see that this is the same as requiring that $|qdt|$ be less than 1.0.

Substituting for q

$$1 > q^2 dt^2 = \frac{4gH}{d^2} \sin^2(kd/2) dt^2$$

for all k.

The maximum wavenumber that can be resolved by a grid of size d is $k = \pi/d$. At this wavenumber, the sine takes its maximum value of 1. So the time step

$$dt^2 < \frac{d^2}{4gH}$$

For this case ($f = 0$) the ratio of the space step to the time step must be greater than the wave speed $\sqrt{gH}$, or

$$d/dt > 2\sqrt{gH}.$$

This stability condition is known as **the CFL condition** (named after Courant, Friedrich and Levy).

On a historical note, the first attempts at weather prediction were organized by Richardson using a room full of human calculators. Each person was responsible for one grid point and passed their values to

neighbouring grid points. The exercise failed dismally, and until the theory of CFL, the exact reason was unknown. The equations Richardson used included fast sound waves, so the CFL condition was

$$d/dt > 2 \times 300\text{m/s}.$$

Richardson's spatial step, d, was too small compared to dt and the problem was unstable.

6.1 *Problem Two*

a) Find the CFL condition (in seconds) for dt for the Python example in Problem One. Test your value.

b) Find the CFL condition (in seconds) for dt for the Python example in *rain.py*, ie., for the non-staggered grid. Test your value.

7 6. Accuracy

A strong method to determine the accuracy of a scheme is to compare the numerical solution to an analytic solution. The equations we are considering are wave equations and so we will compare the properties of the waves. Wave properties are determined by the dispersion relation and we will compare the *numerical dispersion relation* and the exact, continuous *analytic dispersion relation*. Both the time step and the space step (and as we'll see below the grid) affect the accuracy. Here we will only consider the effect of the space step. So, consider the numerical dispersion relation assuming $dt \to 0$ (reproduced here from (Continuous Time, Discretized Space Dispersion Relation))

$$\omega^2 = 4gH\frac{\sin^2(kd/2)}{d^2}$$

with the exact, continuous dispersion relation (Analytic Dispersion Relation)

$$\omega^2 = gHk^2$$

We can plot the two dispersion relations as functions of kd, The graph is shown in Figure Simple Accuracy.

```
In [21]: Image(filename='images/simple_accuracy.png',width='50%')
```

```
Out[21]:
```

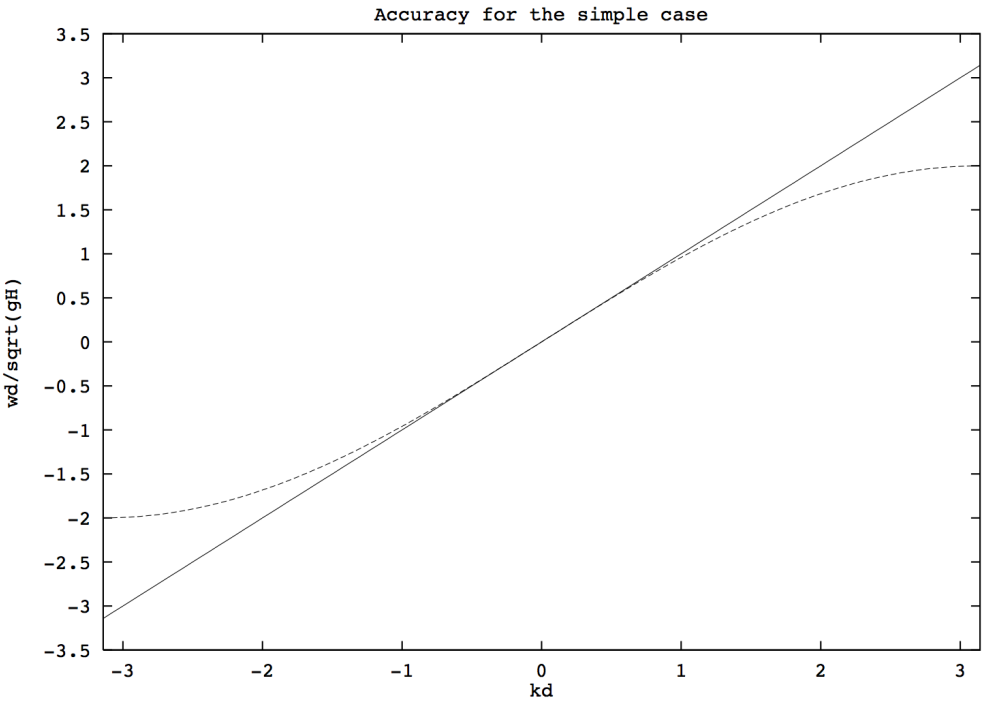


Figure Simple Accuracy

We can see that the accuracy is good for long waves (k small) but for short waves, near the limit of the grid resolutions, the discrete frequency is too small. As the phase speed is ω/k, the phase speed is also too small and most worrying, the group speed $\partial\omega/\partial k$ goes to zero!

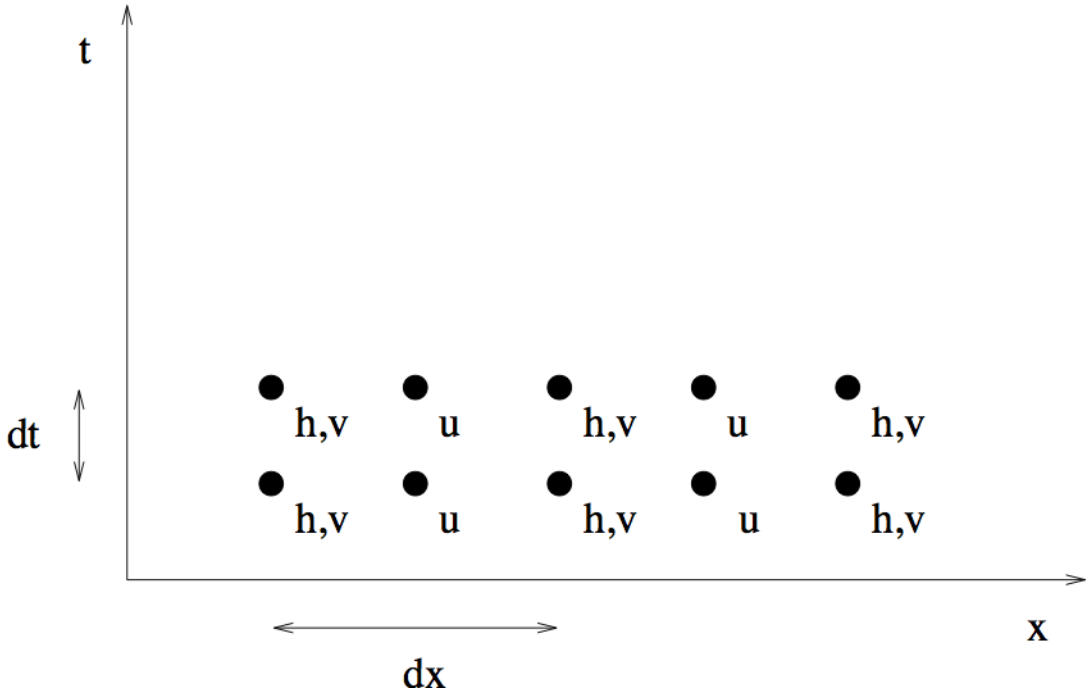
7.1 6.1 Choosing a Grid

7.1.1 No variation in y

For the simple case above, there is little choice in grid. Let's consider the more complicated case, $f \neq 0$. Then $v \neq 0$ and we have to choose where on the grid we wish to put v. There are two choices:

```
In [22]: Image(filename='images/simple_grid1.png',width='50%')
```

```
Out[22]:
```



In [23]: Image(filename='images/simple_grid2.png',width='50%')

Out[23]:

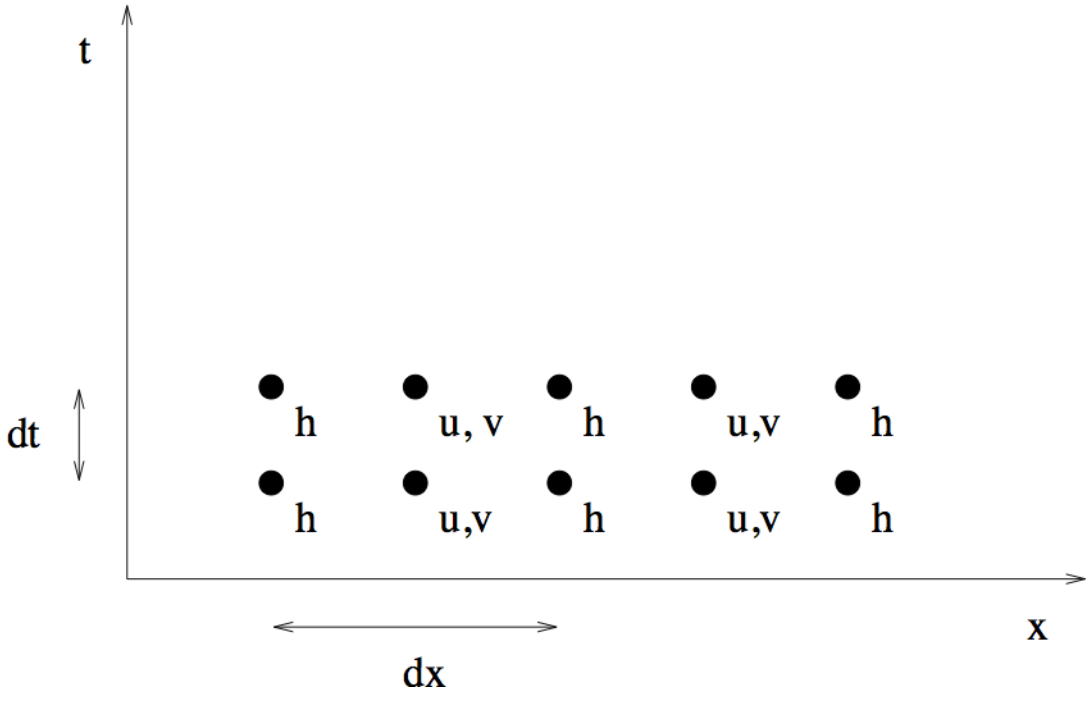


Figure Two Simple Grids
For each of these, we can calculate the discrete dispersion relation discussed above.
For grid 1

$$\omega^2 = f^2 \cos^2(\frac{kd}{2}) + \frac{4gH \sin^2(\frac{kd}{2})}{d^2}$$

7.2 *Problem Three*

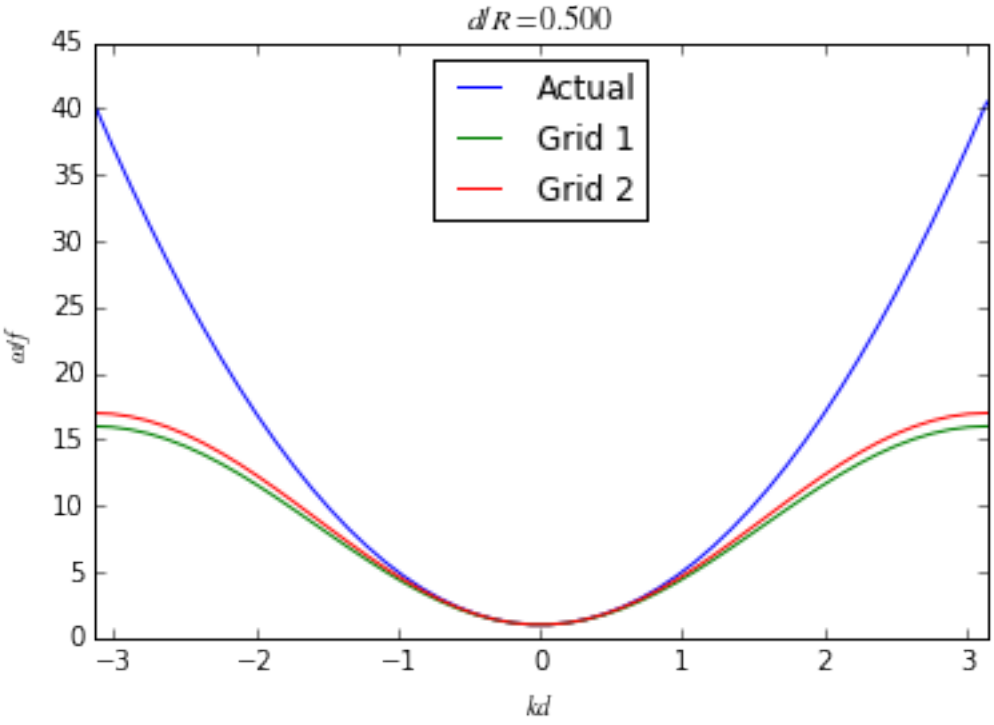
Show that for grid 2

$$\omega^2 = f^2 + \frac{4gH \sin^2(\frac{kd}{2})}{d^2}$$

We can plot the two dispersion relations as a function of k given the ratio of d/R, where d is the grid size and R is the Rossby radius which is given by

$$R = \frac{\sqrt{gH}}{f}.$$

```
In [24]: accuracy2d.main(0.5)
```



7.3 *Problem Four*

Which grid gives the best accuracy for $d = R/2$? Explain in what ways it is more accurate.

7.4 *Problem Five*

Modify *rain.py* to solve equations (No variation in y, first eqn), (No variation in y, second eqn) and (No variation in y, third eqn) on the most accurate grid.

8 7. Full Equations

In order to solve the full equations (Full Equations, Eqn 1), (Full Equations, Eqn 2) and (Full Equations, Eqn 3) numerically, we need to discretize in 3 dimensions, two in space and one in time. Mesinger and Arakawa, 1976 introduced five different spatial discretizations.

Consider first the most obvious choice an unstaggered grid or Arakawa A grid, shown in Figure Arakawa A Grid. We might expect, from the studies above, that an unstaggered grid may not be the best choice. The grid A is not two de-coupled grids because of weak coupling through the Coriolis force. However, we will see that this grid is not as accurate as some of the staggered grids (B, C, D and E).

In [25]:
```
Image(filename='images/grid1.png',width='50%')
```

Out[25]:

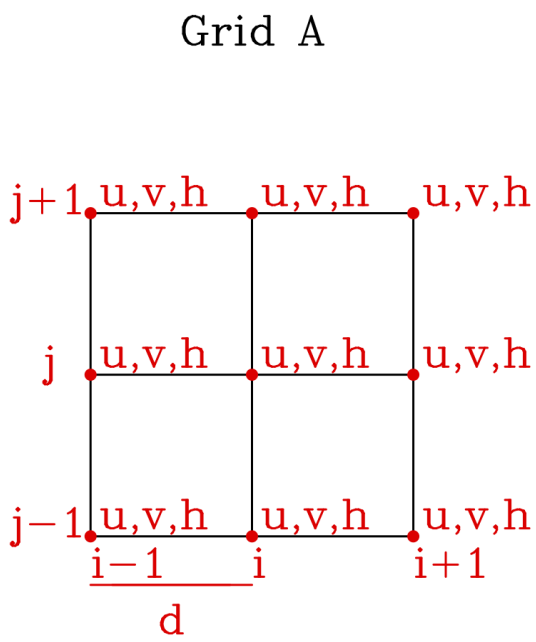


Figure Arakawa A Grid.

As the problem becomes more complicated, we need to simplify the notation; hence define a discretization operator:

$$(\delta_x \alpha)_{(m,n)} \equiv \frac{1}{2d}(\alpha_{m+1,n} - \alpha_{m-1,n})$$

where d is the grid spacing in both the x and y directions. Note that this discretization is the same centered difference we have used throughout this lab.

The finite difference approximation to the full shallow water equations on the A grid becomes:

$$\frac{\partial u}{\partial t} = -g\delta_x h + fv$$

$$\frac{\partial v}{\partial t} = -g\delta_y h - fu$$

$$\frac{\partial h}{\partial t} = -H(\delta_x u + \delta_y v)$$

As before, consider a centre difference scheme (leap-frog method) for the time step as well, so that

$$\frac{\partial u}{\partial t}(t) = \frac{u(t+1) - u(t-1)}{2dt}$$

Putting this together with the spatial scheme we have:

$$\frac{u(t+1) - u(t-1)}{2dt} = -g\delta_x h(t) + fv(t)$$

$$\frac{v(t+1) - v(t-1)}{2dt} = -g\delta_y h(t) - fu(t)$$

$$\frac{h(t+1) - h(t-1)}{2dt} = -H(\delta_x u(t) + \delta_y v(t))$$

Each of these equations can be rearranged to give $u(t+1)$, $v(t+1)$ and $h(t+1)$, respectively. Then given the values of the three variables at every grid point at two times, (t and $t-1$), these three equations allow you to calculate the updated values, the values of the variables at $t+1$. Once again, the following questions arise regarding the scheme:

- **Is it stable?**
- **Is it accurate?**

8.1 7.1 Stability

To determine the stability of the scheme, we need to repeat the analysis of section 5. Stability for the 2 spatial dimensions used here. The first step is to assume a form of the solutions:

$$\begin{aligned} z_{mnp} = & \quad \mathcal{R}e\{\mathcal{Z}\exp[i(kx + \ell y - \omega t)]\} \\ = & \quad \mathcal{R}e\{\mathcal{Z}\exp[i(kmd + \ell nd - \omega p\, dt)]\} \end{aligned}$$

where z represents any of u, v and h and we have let $x = md$, $y = nd$ and $t = p\,dt$. Substitution gives three algebraic equation in $\mathcal{U}$, $\mathcal{V}$ and $\mathcal{H}$:

$$\begin{bmatrix} -i\sin(\omega dt)/dt & -f & ig\sin(kd)/d \\ f & -i\sin(\omega dt)/dt & ig\sin(\ell d)/d \\ iH\sin(kd)/d & iH\sin(\ell d)/d & -i\sin(\omega\, dt)/dt \end{bmatrix} \begin{bmatrix} \mathcal{U} \\ \mathcal{V} \\ \mathcal{H} \end{bmatrix} = 0.$$

Setting the determinate to zero gives the dispersion relation:
(Full Numerical Dispersion Relation)

$$\frac{\sin^2(\omega\, dt)}{dt^2} = f^2 + \frac{gH}{d^2}\left(\sin^2(kd) + \sin^2(\ell d)\right)$$

Still following section 5. Stability, let $\lambda = \exp(i\omega\, dt)$ and let $q^2 = f^2 + gH/d^2\left(\sin^2(kd) + \sin^2(\ell d)\right)$, substitution into (Full Numerical Dispersion Relation) gives

$$-(\lambda - 1/\lambda)^2 = 4q^2 dt^2$$

or equation (Lambda Eqn) again. For stability λ must be less than 1, so

$$1 > q^2 dt^2 = dt^2 \left(f^2 + gH/d^2 \left(\sin^2(kd) + \sin^2(\ell d) \right) \right)$$

The sines take their maximum values at $k = \pi/(2d)$ and $\ell = \pi/(2d)$ giving

$$dt^2 < \frac{1}{f^2 + 2gH/d^2}$$

This is the CFL condition for the full equations on the Arakawa A grid. Note that the most unstable mode moves at 45^o to the grid.

8.2 *Problem Six*

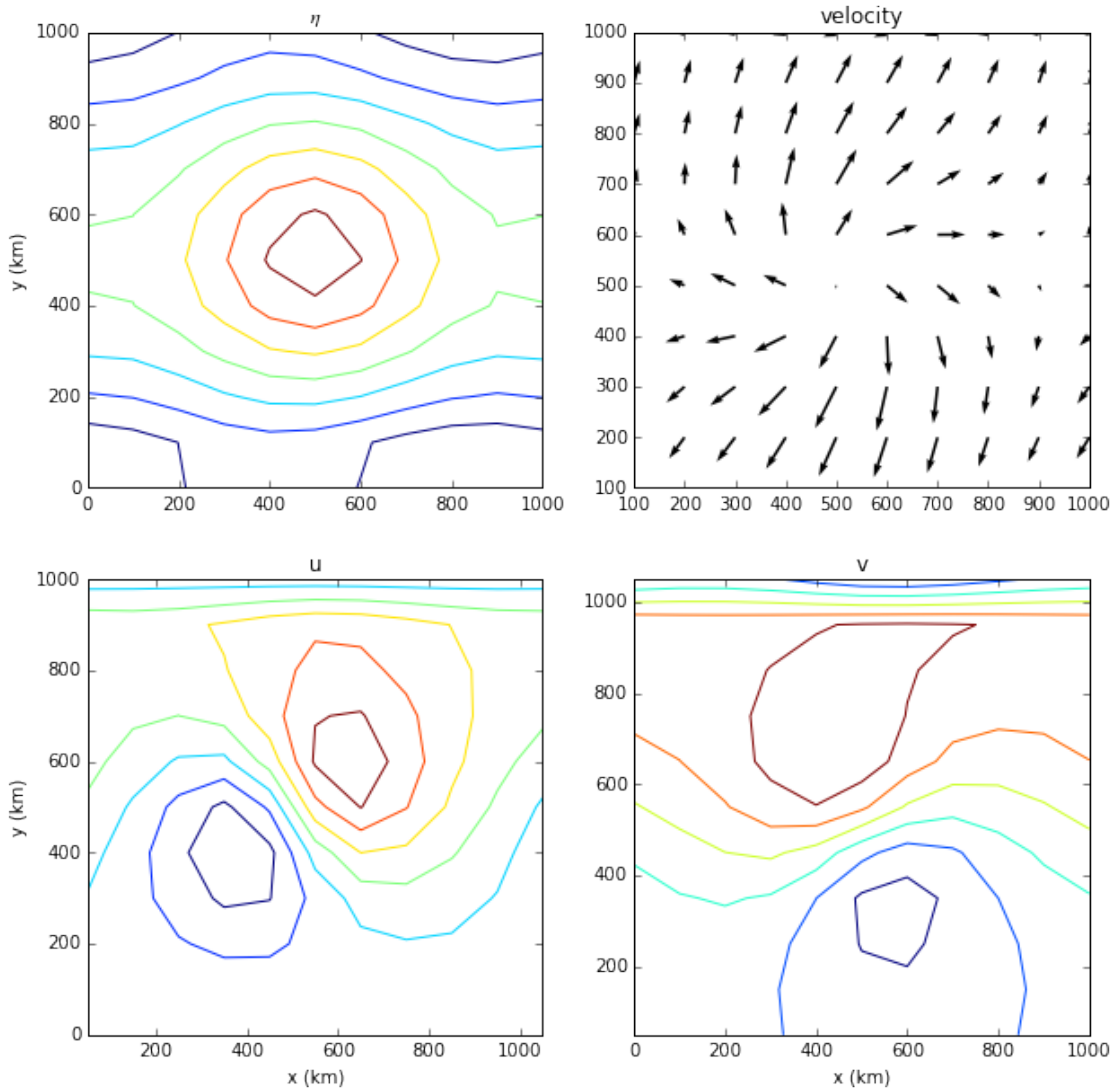
Use the interactive example below to investigate stability of the various grids. Calculate the stability for each of the other 4 grids. Find the dt for stability (to one significant figure) given $f = 1 \times 10^{-4} \mathrm{s}^{-1}$, $g = 10$ m s^{-2}, $H = 4000$ m and $dx = 20$ km. Is it the same for all four grids? Why not?

```
In [26]: # grid A is grid 1, grid B is grid 2 and and grid C is grid 3
         # ngrid is the number of grid points in x and y
         # dt is the time step in seconds
         # T is the time plotted is seconds to 4*3600 is 4 hours
         interactive1.interactive1(grid=3, ngrid=11, dt=150, T=4*3600)
```

```
/Users/phil/miniconda3/lib/python3.4/site-packages/matplotlib/collections.py:650: FutureWarning: elemen
  if self._edgecolors_original != str('face'):
/Users/phil/miniconda3/lib/python3.4/site-packages/matplotlib/collections.py:590: FutureWarning: elemen
  if self._edgecolors == str('face'):

dx/dt 666.667

sqrt(g*H) 100.000
```



8.3 7.2 Accuracy

To determine the accuracy of the spatial discretization, we will compare the *numerical dispersion relation* (Full Numerical Dispersion Relation) for $dt \to 0$

$$\omega^2 = f^2 + gH\frac{\sin^2(kd)}{d^2} + gH\frac{\sin^2(\ell d)}{d^2}$$

with the exact, *continuous dispersion relation*

(Full Analytic Dispersion Relation)

$$\omega^2 = f^2 + gH(k^2 + \ell^2)$$

We can plot the two dispersion relations as functions of k and ℓ, given the ratio of d/R, where d is the grid size and R is the Rossby radius defined in the previous section. For example, the exact ω and its discrete approximation, using Grid A and $d/R = 1/2$, can be compared in Figure Accuracy Comparison.

In [27]: `Image(filename='images/accuracy_demo.png',width='60%')`

Out[27]:

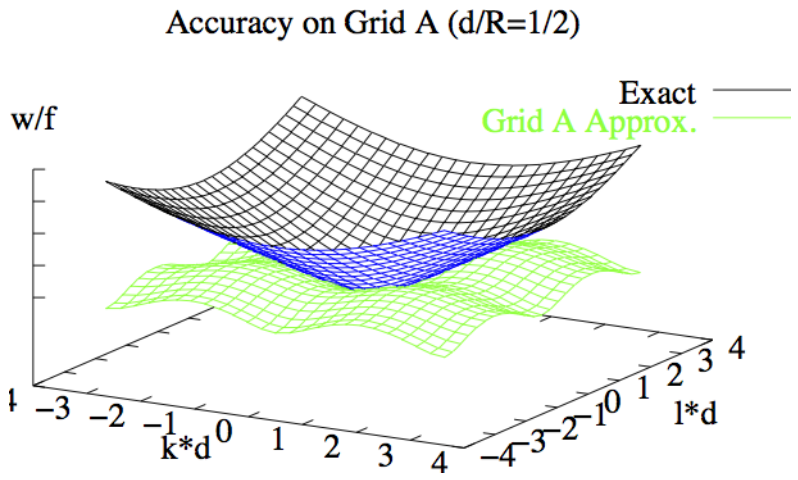


Figure Accuracy Comparison: A comparison of the exact ω and the discrete approximation using Grid A and with $d/R = 1/2$.

It is easy to see that the Grid A approximation is not accurate enough. There are a number of other possibilities for grids, all of which *stagger* the unknowns; that is, different variables are placed at different spatial positions as discussed in section 4.4 Staggered Grids.

Below are four which are known as Mesinger and Arakawa B, C, D and E grids. Note that E is simply a rotation of grid B by 45°.

In [28]: `Image(filename='images/fourgrids.png', width='80%')`

Out[28]:

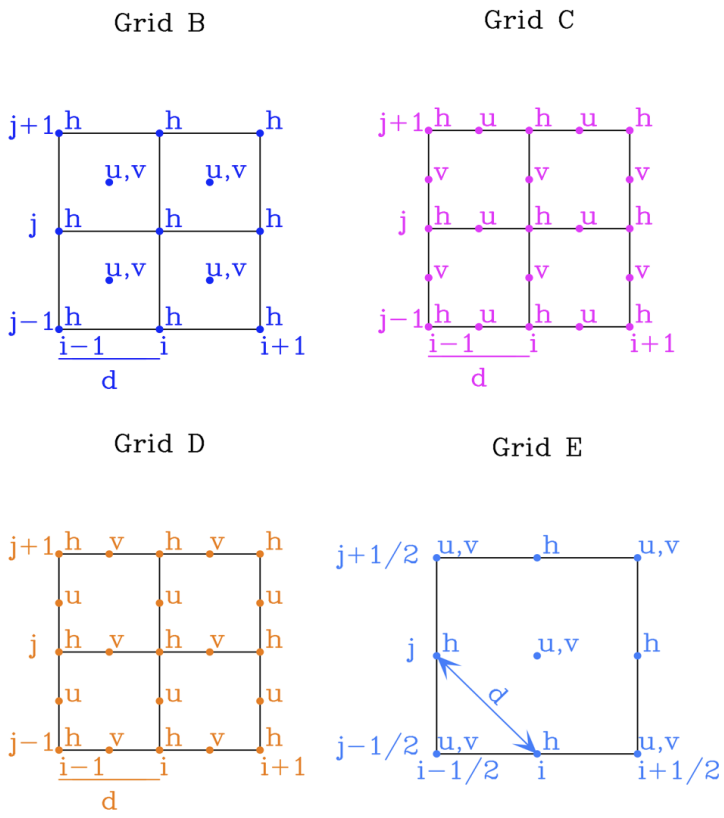


Figure Four More Grids.

To work with these grids, we must introduce an averaging operator, defined in terms of *half-points* on the grid:

$$\overline{\alpha}^{x}_{mn} = \frac{\alpha_{m+\frac{1}{2},n} + \alpha_{m-\frac{1}{2},n}}{2}$$

and modify the difference operator

$$(\delta_x \alpha)_{mn} = \frac{\alpha_{m+\frac{1}{2},n} - \alpha_{m-\frac{1}{2},n}}{d}$$

8.4 *Problem Seven*

A. For grid B, write down the finite difference form of the shallow water equations.

B. For grid C, write down the finite difference form of the shallow water equations.

C. For grid D, write down the finite difference form of the shallow water equations.

The dispersion relation for each grid can be found in a manner analgous to that for the A grid. For the B grid the dispersion relation is

$$\left(\frac{\omega}{f}\right)^2 = 1 + 4\left(\frac{R}{d}\right)^2 \left(\sin^2\frac{kd}{2}\cos^2\frac{\ell d}{2} + \cos^2\frac{kd}{2}\sin^2\frac{\ell d}{2}\right)$$

and for the C grid it is

$$\left(\frac{\omega}{f}\right)^2 = \cos^2\frac{kd}{2}\cos^2\frac{\ell d}{2} + 4\left(\frac{R}{d}\right)^2\left(\sin^2\frac{kd}{2} + \sin^2\frac{\ell d}{2}\right)$$

8.5 *Problem Eight*

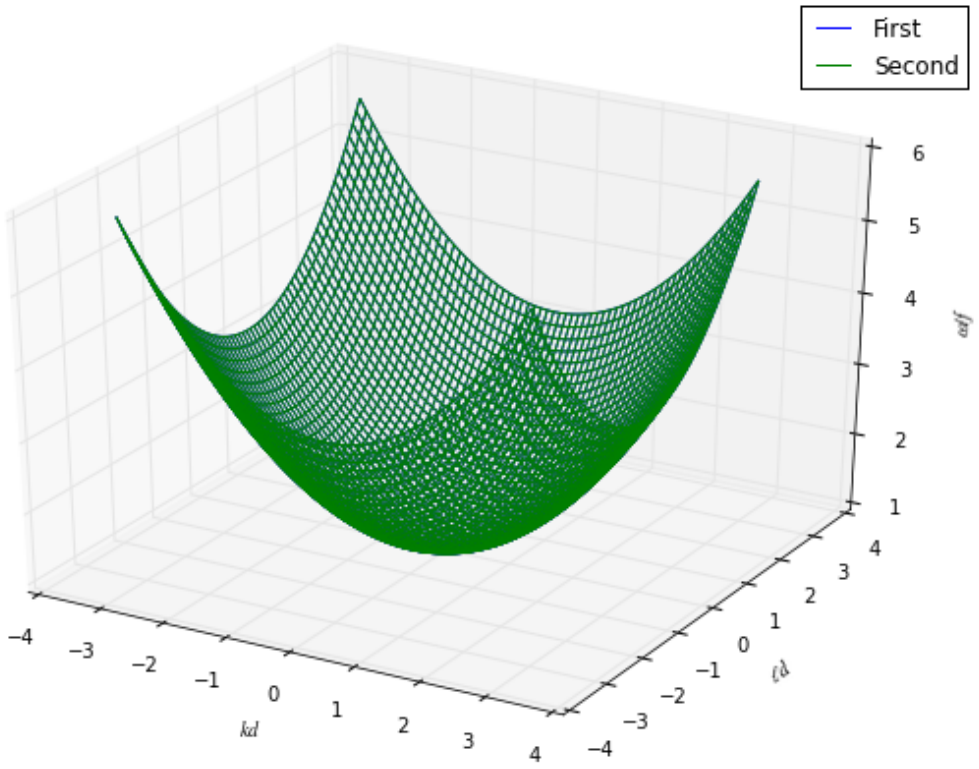
Find the dispersion relation for the D grid.

In the interactive exercise below, you will enter the dispersion for each of the grids. Study each plot carefully for accuracy of phase and group speed.

```
In [29]: def disp_analytic(kd, ld, Rod=0.5):
             Omegaof = 1 + Rod**2 * (kd**2 + ld**2)
             return Omegaof
         # define disp_A, disp_B, disp_C here and run the cell
```

```
In [30]: # replace the second disp_analytic with one of your numerical dispersion functions, e.g. disp_
         dispersion_2d.dispersion_2d(disp_analytic, disp_analytic, Rod=0.5)
```

```
/Users/phil/miniconda3/lib/python3.4/site-packages/matplotlib/collections.py:590: FutureWarning: elemen
  if self._edgecolors == str('face'):
```



8.6 *Problem Nine*

A. For $R/d = 2$ which grid gives the most accurate solution? As well as the closeness of fit of ω/f, also consider the group speed (gradient of the curve). The group speed is the speed at which wave energy propagates.

B. For $R/d = 0.2$ which grid gives the most accurate solution? As well as the closeness of fit of ω/f, also consider the group speed (gradient of the curve). The group speed is the speed at which wave energy propagates.

9 8. Details

9.1 8.1 Starting the Simulation Full Equations

The *leap-frog scheme* requires values of u, v and h at step 1 as well as at step 0. However, the initial conditions only provide starting values at step 0, so we must find some other way to obtain values at step 1. There are various methods of obtaining the second set of starting values, and the code used in this laboratory uses a *predictor/corrector method* to obtain values at step 1 from the values at step 0. For the simple equations this process was discussed in section 4.1 Predictor-Corrector to Start. For the full equations the procedure goes as follows:

- the solution at step 1 is predicted using a *forward Euler step*:

$$\begin{aligned} u(1) &= u(0) + dt(fv(0) - g\delta_x h(0)) \\ v(1) &= v(0) + dt(-fu(0) - g\delta_y h(0)) \\ h(1) &= h(0) + dt(-H(\delta_x u(0) + \delta_y v(0))) \end{aligned}$$

- then, step $\frac{1}{2}$ is estimated by averaging step 0 and the predicted step 1:

$$\begin{aligned} u(1/2) &= 1/2(u(0) + u(1)) \\ v(1/2) &= 1/2(v(0) + v(1)) \\ h(1/2) &= 1/2(h(0) + h(1)) \end{aligned}$$

- finally, the step 1 approximation is corrected using leap frog from 0 to 1 (here, we use only a half time-step $\frac{1}{2}dt$):

$$\begin{aligned} u(1) &= u(0) + dt(fv(1/2) - g\delta_x h(1/2)) \\ v(1) &= v(0) + dt(-fu(1/2) - g\delta_y h(1/2)) \\ h(1) &= h(0) + dt(-H(\delta_x u(1/2) + \delta_y v(1/2))) \end{aligned}$$

9.2 8.2 Initialization

The initial conditions used for the stability demo for the full equations are Poincare waves as described in the physical example in Section 3 Physical Example, Poincare Waves.

Assuming a surface height elevation

$$h = \cos(kx + \ell y)$$

equations (Full Eqns, Eqn 1,Full Eqns, Eqn 2) give

$$\begin{aligned} u = & \quad \frac{-1}{H(k^2 + \ell^2)} \left(k\omega \cos(kx + \ell y) + f\ell \sin(kx + \ell y)\right) \\ v = & \quad \frac{1}{H(k^2 + \ell^2)} \left(-\ell\omega \cos(kx + \ell y) + fk \sin(kx + \ell y)\right) \end{aligned}$$

where ℓ and k are selected by the user. It is assumed $g = 9.8\text{m/s}^2$, $H = 400\text{m}$ and $f = 10^{-4}/\text{s}$. The value of the frequency ω is given by the dispersion relation, (Full Analytic Dispersion Relation).

9.3 8.3 Boundary Conditions

The boundary conditions used for the stability demo for the full equations are *"periodic"* in both x and y. Anything which propagates out the left hand side comes in the right hand side, *etc.* This condition forces a

periodicity on the flow, the wavelengths of the simulated waves must be sized so that an integral number of waves fits in the domain.

Specifically, for a $m \times n$ grid, the boundary conditions for the variable h along the right and left boundaries are

$$\begin{aligned} h(i=1,j) &= \quad h(i=m-1,j) \text{ for } j=2 \text{ to } n-1 \\ h(i=m,j) &= \quad h(i=2,j) \text{ for } j=2 \text{ to } n-1 \end{aligned}$$

and along the top and bottom boundaries

$$\begin{aligned} h(i,j=1) &= \quad h(i,j=n-1) \text{ for } i=2 \text{ to } m-1 \\ h(i,j=n) &= \quad h(i,j=2) \text{ for } i=2 \text{ to } m-1. \end{aligned}$$

The conditions for u and v are identical.

9.4 8.4 Computational Mode

In section 5. Stability it was determined that there are four oscillatory modes. Consider letting $dt \to 0$ in (Lambda Eqn). Two of the roots give $+1$ and two roots give -1.

Consider the variable u at time $p\,dt$ and one time step later at time $(p+1)\,dt$.

$$\frac{u_{m(p+1)}}{u_{mp}} = \frac{\mathcal{R}e\{\mathcal{U}\exp[i(kmd - \omega(p+1)\,dt)]\}}{\mathcal{R}e\{\mathcal{U}\exp[i(kmd - \omega p\,dt)]\}} = \exp(i\omega\,dt) = \lambda$$

Thus, λ gives the ratio of the velocity at the next time step to its value at the current time step. Therefore, as the time step gets very small, physically we expect the system not to change much between steps. In the limit of zero time step, the value of u or h should not change between steps; that is λ should be 1.

A value of $\lambda = -1$ implies u changes sign between each step, no matter how small the step. This mode is not a physical mode of oscillation, but rather a *computational mode*, which is entirely non-physical. It arises from using a centre difference scheme in time and not staggering the grid in time. There are schemes that avoid introducing such spurious modes (by staggering in time), but we won't discuss them here (for more information, see Mesinger and Arakawa, 1976 [Ch. II]). However, the usual practice in geophysical fluid dynamics is to use the leap-frog scheme anyway (since it is second order in time) and find a way to keep the computational modes "small", in some sense.

For a reasonably small value of dt, the computational modes have $\lambda \approx -1$. Therefore, these modes can be eliminated almost completely by averaging two adjacent time steps. To understand why this is so, think of a computational mode $\hat{u}_{mp}$ at time level p, which is added to its counterpart, $\hat{u}_{m(p+1)} \approx -\hat{u}_{mp}$ at the next time step: *their sum is approximately zero!* For the code in this lab, it is adequate to average the solution in this fashion only every 101 time steps (though larger models may need to be averaged more often). After the averaging is performed, the code must be restarted; see Section on Starting.

10 Glossary

Poincare waves: These are waves that obey the dispersion relation $\omega^2 = f^2 + k^2c^2$, where c is the wave speed, k is the magnitude of the wavenumber vector, f is the Coriolis frequency, and ω is the wave frequency.

CFL condition: named after Courant, Friedrichs and Levy, who first derived the relationship. This is a stability condition for finite difference schemes (for propagation problems) that corresponds physically to the idea that the continuous domain of dependence must contain the corresponding domain of dependence for the discrete problem.

dispersive wave: Any wave whose speed varies with the wavenumber. As a consequence, waves of different wavelengths that start at the same location will move away at different speeds, and hence will spread out, or *disperse*.

dispersion relation: For dispersive waves, this relation links the frequency (and hence also the phase speed) to the wavenumber, for a given wave. See also, *dispersive wave*.

dispersive wave: Any wave whose speed varies with the wavenumber. As a consequence, waves of different wavelengths that start at the same location will move away at different speeds, and hence will spread out, or *disperse*.

dispersion relation: For dispersive waves, this relation links the frequency (and hence also the phase speed) to the wavenumber, for a given wave. See also, *dispersive wave*.

staggered grid: This refers to a problem with several unknown functions, where the discrete unknowns are not located at same grid points; rather, they are *staggered* from each other. For example, it is often the case that in addition to the grid points themselves, some uknowns will be placed at the center of grid cells, or at the center of the sides of the grid cells.

leap-frog scheme: This term is used to refer to time discretization schemes that use the centered difference formula to discretize the first time derivative in PDE problems. The resulting difference scheme relates the solution at the next time step to the solution *two* time steps previous. Hence, the even- and odd-numbered time steps are linked together, with the resulting computation performed in a *leap-frog* fashion.

periodic boundary conditions: Spatial boundary conditions where the value of the solution at one end of the domain is required to be equal to the value on the other end (compare to dirichlet boundary values, where the the solution at both ends is fixed at a specific value or values). This enforces periodicity on the solution, and in terms of a fluid flow problem, these conditions can be thought of more intuitively as requiring that any flow out of the one boundary must return through the corresponding boundary on the other side.

computational mode: When performing a modal analysis of a numerical scheme, this is a mode in the solution that does not correspond to any of the "real" (or physical) modes in the continuous problem. It is an artifact of the discretization process only, and can sometimes lead to spurious computational results (for example, with the leap-frog time stepping scheme).

11 References

Cushman-Roisin, B., 1994: Introduction to Geophysical Fluid Dynamics, Prentice Hall.

Gill, A.E., 1982: Atmosphere-Ocean Dynamics, Academic Press, Vol. 30 International Geophysics Series, New York.

Mesinger, F. and A. Arakawa, 1976: Numerical Methods Used in Atmospheric Models,GARP Publications Series No.˜17, Global Atmospheric Research Programme.

Pond, G.S. and G.L. Pickard, 1983: Introductory Dynamic Oceanography, Pergamon, Great Britain, 2nd Edition.

Press, W.H., S.A. Teukolsky, W.T. Vetterling and B.P. Flannery, 1992: Numerical Recipes in FORTRAN: The Art of Scientific Computing, Cambridge University Press, Cambridge, 2n Edition.

In [ ]: