Computational Physics
Lecture 8

sieversj@ukzn.ac.za

git clone https://github.com/ukzncompphys/lecture8_2015.git

Old Tutorial Problems

- Write a function that will shift an array by an arbitrary amount using a convolution (yes, I know there are easier ways to do this). The function should take 2 arguments - an array, and an amount by which to shift the array. Plot a gaussian that started in the centre of the array shifted by half the array length. (10)

- The correlation function f★g is $\int f(x)g(x+y)$. Through a similar proof, one can show f★g=ift(dft(f)*conj(dft(g))). Write a routine to take the correlation function of two arrays. Plot the correlation function of a Gaussian with itself. (10)

- Using the results of part 1 and part 2, write a routine to take the correlation function of a Gausian (shifted by an arbitrary amount) with itself. How does the correlation function depend on the shift? Does this surprise you? (10)

- The circulant (wrap-around) nature of the dft can sometimes be problematic. Write a routine to take the convolution of two arrays *without* any danger of wrapping around. You may wish to add zeros to the end of the input arrays. (10)

Old Tutorial Problems 2

- Complete the complex definition to support -,*, and / (__sub__, __mul__, and __div__). Recall that a/b = a*conj(b)/(b*conj(b)). Show from a few sample cases that your functions work. (10)

- Next lecture we will look at n-body simulations. In preparation, write a class that contains masses and x and y positions for a collection of particles. The class should also contain a dictionary that can contain options. Two entries in the dictionary should be the # of particles and G (gravitational constant). The class should also contain a method that calculates the potential energy of every particle, sum(G $m_1 m_2 / r_{12}$) . (10)

Reminder: Advection

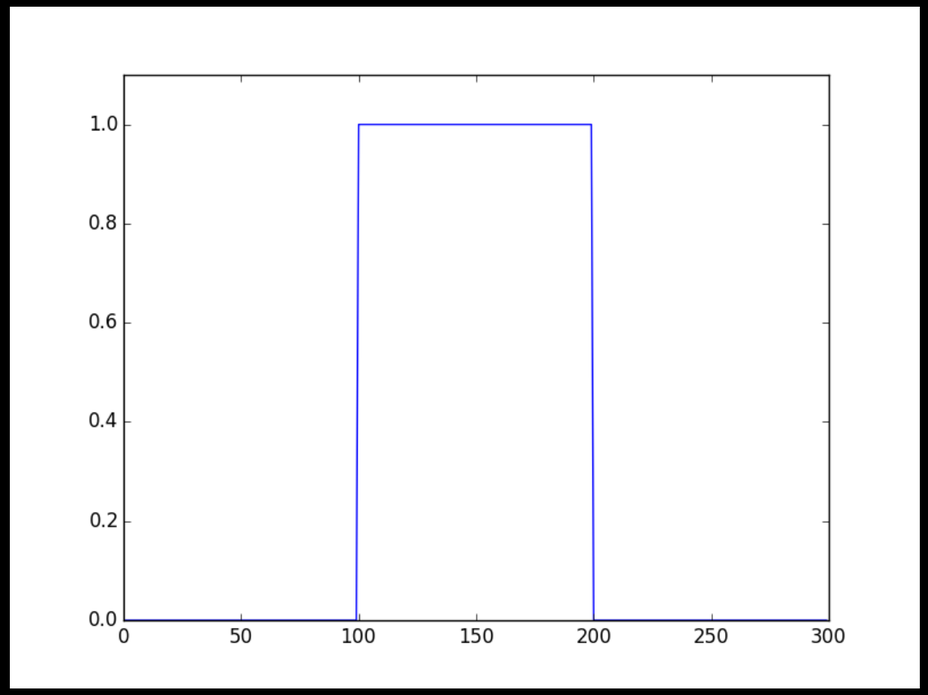
- Advection equation: $\partial f/\partial t+u\partial f/\partial x=0$
- Trial: f=f(ut-x): then uf' +u(-f')=0. check
- So, any function of (ut-x) will solve this equation.
- So, if we watch the spot in the function at x_0 when t=0, then at time=t, the position will be: ut-x=0-x_0, or x=x_0+ut. Information moves with velocity u.

Finite Volume Advection

```
#simple_advect_finite_volume.py
import numpy
from matplotlib import pyplot as plt
n=300
rho=numpy.zeros(n)
rho[n/3:(2*n/3)]=1
v=1.0
dx=1.0
x=numpy.arange(n)*dx

plt.ion()
plt.clf()
plt.plot(x,rho)
```

Left: set up initial conditions. Density is 1 in the middle third of region, zero otherwise. Below left: initial density plotted. Bottom: advection code.




```
dt=1.0
for step in range(0,50):
    #take the difference in densities
    drho=rho[1:]-rho[0:-1]
    #update density.  We haven't said what happens at
    #cell 0 (since cell -1 doesn't exist), ignore for now
    rho[1:]=rho[1:]-v*dt/dx*drho
    plt.clf()
    plt.plot(x,rho)
    plt.draw()
```

Conservation Equation

- If a quantity is conserved, time rate of change in a volume is equal to net flow into/out volume.
- If conserved quantity is ρ and velocity is u then flow out of region is $\rho_+ u_+$ and flow in is $\rho_- u_-$. Net flux is then $-\partial(\rho u)/\partial x$.
- Equation then become $\partial\rho/\partial t = -\partial(\rho u)/\partial x$, or $\partial\rho/\partial t+\partial(\rho u)/\partial x=0$
- If a quantity is created, then we pick up extra term for rate of creation:
- now $\partial\rho/\partial t=-\partial\rho/\partial x + q$, where q is the creation rate.

Euler Equations

- Now we're set to derive equations of fluid mechanics.
- The full fluid equations (Navier-Stokes) include forces from viscosity
- We will make approximation that viscosity is negligible
- Further, we will assume no energy flows between pieces of fluid (this is usually quite a good approximation)
- Leaves us with Euler equations. What equations should we have?

Mass Conservation

- Generally, no matter is created/destroyed, so mass is strictly conserved.
- Mass conservation becomes $\partial\rho/\partial t+\partial(u\rho)\partial x=0$
- Note that if you had source/sink of matter, it would appear as an extra term

Momentum

- Momentum is ρu. So conservation equation is $\partial(\rho u)/\partial t+\partial(\rho u^2)/\partial x=0$
- Velocity appears squared, so equation is nonlinear
- Fluid pressure will exert a force, so force term must be added.
- Force on right side of a packet is $-P_+$, force on left is $+P_-$, so total net force is difference, limit is $-\partial P/\partial x$. This force has to go into momentum equation.
- Momentum equation: $\partial(\rho u)/\partial t+\partial(\rho u^2)/\partial x=-\partial P/\partial x$
- Conservation form: rewrite as $p=\rho u$, get $\partial p/\partial t+\partial(pu+P)/\partial x=0$

Energy

- Two pieces of energy - internal thermal energy and bulk kinetic.
- Call total energy (thermal+kinetic) per unit mass E.
- Energy creation rate from pressure is power, or force * velocity
- Gives $\partial(\rho E)/\partial t+\partial(u\rho E)/\partial x=-\partial(uP)/\partial x$
- Rewrite into conservation form: $\partial(\rho E)/\partial t+\partial(u\rho E+uP)/\partial x=0$

Euler So Far

- $\partial\rho/\partial t+\partial(u\rho)\partial x=0 \quad \partial p/\partial t+\partial(pu+P)/\partial t=0 \quad \partial(\rho E)/\partial t+\partial(u\rho E+uP)/\partial x=0$
- Three equations, how many unknowns? Solution needs velocity, density, energy, and pressure.
- So, need one more equation. Normally done by specifying a relation between pressure and energy. This is called an *equation of state*.
- Classic EoS is gamma law, $P\sim\rho^{\gamma}$. For ideal gas, $e=3/2\ nkT$, pressure is nKT, so $P=2/3\rho e$ (where $e=E-1/2\rho u^2$ is the thermal energy).

Derivation of γ

- Let's compress a volume of gas and see how energy changes.
- $dE=-PdV$. $E=aPV$ (where $a=3/2$ for ideal gas)
- $ad(PV)=-PdV$. $aVdP+aPdV=-PdV$
- $dP(aV)=-dV(P(1+a))$, $adP/P=-(1+a)dV/V$.
- $\log(P)\sim-(1+a)/a\log(V)$. $P\sim V^{-(1+a)/a}$. Density $\sim 1/V$, so $P\sim\rho^{1+1/a}$. The index is usually called γ (gamma). For ideal gas, a is 3/2, so $\gamma=1+2/3=5/3$.

Euler Equations with EoS

- We can now write down Euler equations in conservation form with EoS
- $E=1/2u^2+e$, $\rho e=P/(\gamma-1)$. So $P=\rho(\gamma-1)(E-1/2u^2)$
- $\partial Q/\partial t+\partial(f(Q))/\partial x=0$
- $Q=[\rho,\rho u,\rho E]$, $f(Q)=[\rho u, \rho u^2+P, \rho uE+uP]$
- using momentum $p=\rho u$: $Q=[\rho,p,\rho E]$, $f(Q)=[p,pu+P,pE+uP]$

Reminder: Time Steps

- Smaller time step normally more accurate.
- Let's look at solution for some different time steps.
- What happened?
- Behaviour of sharp features often very important - in practice, run test problems with known solutions to verify behaviour.

```
#advect_finite_volume_timestep.py
dt=1.0
big_rho=numpy.zeros(n+1)
big_rho[1:]=rho
del rho  #we can delete the to save space
oversamp=10 #let's do finer timestamps
dt_use=dt/oversamp
for step in range(0,150):

    big_rho[0]=0
    for substep in range(0,oversamp):
        drho=big_rho[1:]-big_rho[0:-1]
        big_rho[1:]=big_rho[1:]-v*dt_use/dx*drho

    plt.clf()
    plt.axis([0,n,0,1.1])
    plt.plot(x,big_rho[1:])
    plt.draw()
```

Reminder: Stability

$$\rho_j^{new}=\rho_j-(\rho_j-\rho_{j-1})vdt/dx$$

- You can learn a lot by plugging in sine waves.
- If $\rho_j=exp(ikj)$, ρ_j^{new}=what? define a=vdt/dx
- $\rho_j^{new}=exp(ikj)-a(exp(ikj)-exp(ik(j-1)) = exp(ikj)-a(exp(ikj)-exp(-ik)exp(ikj))$
- $\rho_j^{new}=exp(ikj)*[1-a(1-exp(-ik))]$
- If quantity in [] gets bigger than unity, solution will grow with time. Our code would be *unstable* - this is bad!

Reminder: CFL Condition (a=vdt/dx)

- Look at 1-a(1-exp(-ik)). 1-exp(-ik) is bounded by (0,2)
- if 0, []=1, solution always stable.
- if 2, then []=1-2a can have magnitude >1 for sufficiently large a.
- By construction, a is positive, so can't get []>1. But can get []<-1: 1-2a<-1, 2<2a, or a>1.
- For stability, a<=1, or dt<dx/v. In words, dt has to be shorter than crossing time for cell.
- This is called the Courant–Friedrichs–Lewy (CFL) condition. vdt/dx is the Courant number.

System of PDE's

- Let's take a system of 2 equations with constant coefficients:
- $\partial f/\partial t + c_{11}\partial f/\partial x + c_{12}\partial g/\partial x = 0$ and $\partial g/\partial t + c_{21}\partial f/\partial x + c_{22}\partial g/\partial x = 0$
- Solution to 1-D advection was h(x-ut), so let's guess solution is $f=v_1h(ut-x)$, $g=v_2h(ut-x)$
- Plug in: system is then $uv_1h'-c_{11}v_1h'-c_{12}v_2h'=0$ and $uv_2h'-c_{21}v_1h'-c_{22}v_2h'=0$
- h' drops out, and we're left with system: $uv=Cv$

PDE Systems, Ctd:

- System uv=Cv is just eigenvalue problem. Get a solution for each eigenvector/eigenvalue pair, where propagation speed is eigenvalue.
- When eigenvalues are real, system is called *hyperbolic*, solutions of form h(x-ut). Information propagates at finite speed.
- When eigenvalues are imaginary, system is *elliptical,* solutions of form h(x-iut). You might expect treatment in numerical solvers to be different.
- Do you think fluid equations should be elliptical or hyperbolic?

CFL Condition Revisited

- Euler equations give us a system of 3 coupled equations.
- This means 3 eigenvalues. For CFL condition, want time step stable for largest velocity eigenvalue.
- What do you think the three eigenvalues are? You should be able to guess from physical intuition. (recall that the speed of sound $c_s^2=\gamma P/\rho$)

Aside: Stiff Equations

- We get one eigenvalue for fluid velocity, and 2 for velocity ±speed of sound.

- If $c_s >> u$, then CFL means timestep has to be tiny compared to natural one from fluid velocity. When eigenvalues diverge like this, equations are called *stiff*. Different computational techniques required.

- Incompressible fluid mechanics - limit where $c_s >> u$. Fluid has time to move out of the way. Otherwise it would compress.

- Techniques to solve stiff equations are different. If you hit a stiff set, look them up. Always check if your system is stiff!

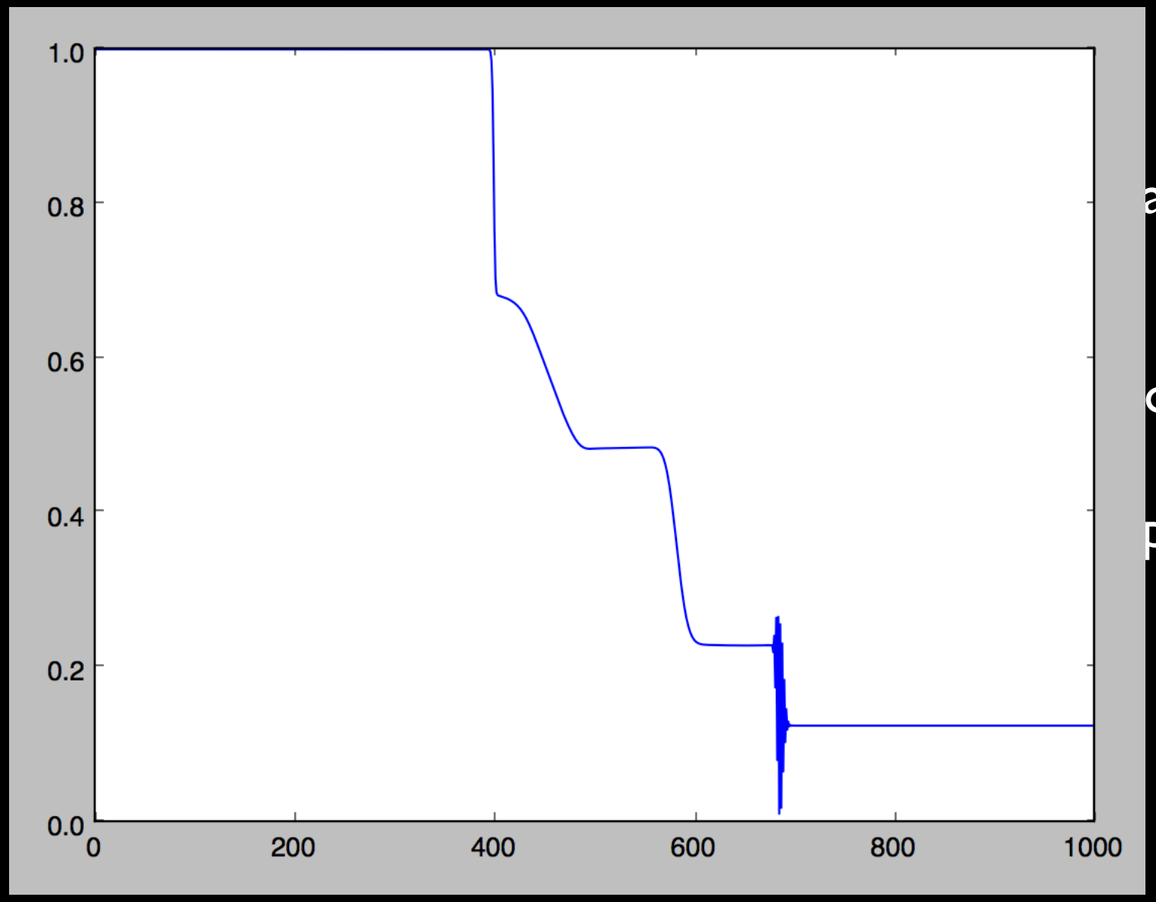
Structure of a 1-D Fluid Code

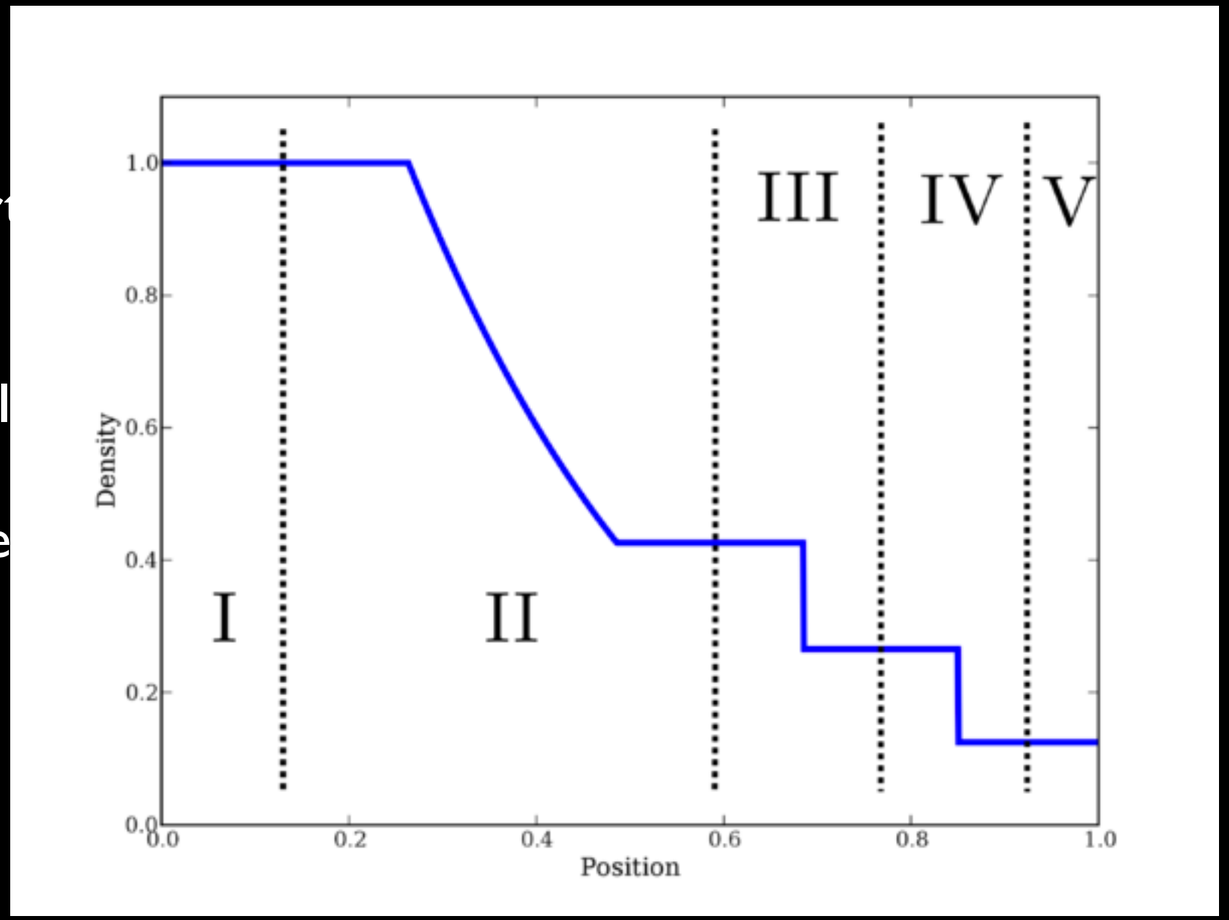
- First, do boundary conditions
- If we use density, momentum, total energy as variables (the conservation quantities) then need to calculate velocity
- Now need to calculate pressure
- Next calculate gradients - we use upwind 1st order scheme, where I flow with my velocity
- Calculate CFL timestep
- Finally, update density, momentum, Energy

Shock Tube

- Classic testing problem is a shock tube: start with a density/pressure jump in the middle, with velocity=0.
- What should this look like? let's run hydro1d.py
- What answer *should* look like from wikipedia:

Shock Tube


1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000


I
II
III
IV
V
Density
Position
0.0
0.2
0.4
0.6
0.8
1.0

Tutorial Problems

- Look at hydro1d.py. you'll see an assert guaranteed to fail at the end of get_bc, the boundary condition routine. Why did I do this? (5)

- Further on in hydro1d, where the derivatives are getting calculated, there's a factor of 1/2 in the pressure gradient. Why? (5)

- Finally, look at the time step calculator. Right now it doesn't implement the CFL condition. Put in a proper timestep calculator. This should find the globally smallest stable timestep and return the input times this value. So, if global CFL limit is 0.3 and we pass in 0.1, the return value should be 0.03 (10).