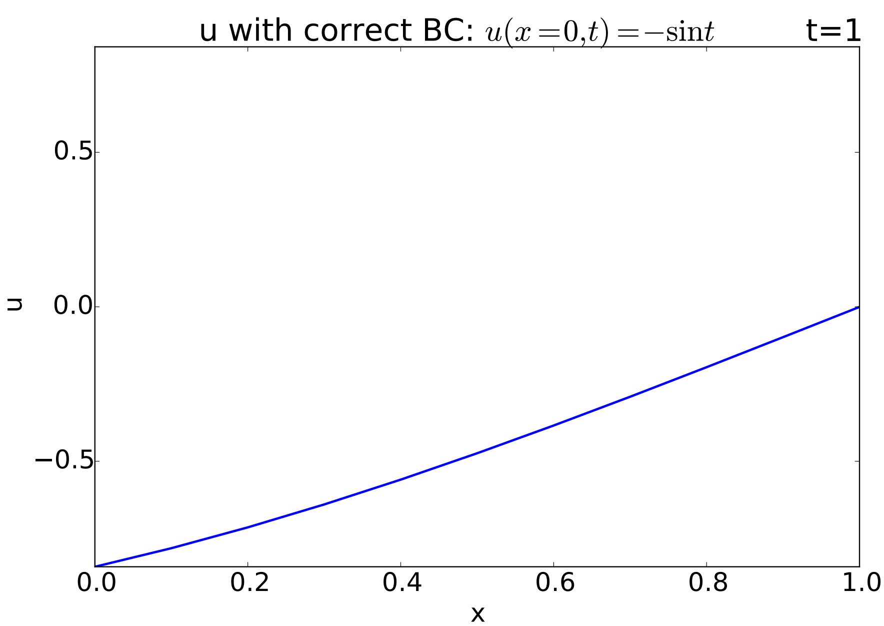
Numerical Relativity - PHY 6938

Solutions to HW 8

1. a) In this problem we are trying to solve the PDE $\partial_t u + \partial_x u = 0$. This is a single mode with speed 1 moving to the right. I.e. on the left boundary ($x = 0$) we need to specify a boundary condition to say what is coming in. On the right boundary ($x = 1$) nothing can come in since the mode is moving to the right, so no BC is needed. In fact imposing one at $x = 1$ may be incompatible with well-posedness and lead to instabilities.
Thus the program `advection1.py` is incorrect! It never changes the value of u at both boundaries. The reason is that `advection1.py` uses `D0` which simply does nothing at both ends. This is the same as imposing the BCs $u(0,t) = 0$ and $u(1,t) = \sin(1)$. I.e. it imposes BCs at both ends! If you run `advection1.py` you will see that u develops growing spikes with the highest possible wavenumber of the grid. This is the hallmark of an instability, which is caused by using an ill-posed method.
If we use `Dp` instead of `D0` things get worse. Now we are imposing $u(1,t) = \sin(1)$ at the side where there should be no BC, because `Dp` does nothing at the right side. But on the left side we use $\partial_t u + \partial_x u = 0$ because there `Dp` works just fine. I.e. on the left side we are not imposing a BC, even though we should.
If we use `Dm` instead of `D0` the program works fine. Now we are using $\partial_t u + \partial_x u = 0$ on the right side because `Dm` works well there. While on the left we are using $u(0,t) = 0$, because we never change u there. Thus we get the case where the inital sinus function simply moves to the right, and nothing is coming in.
b) We should impose a BC only on the left side at $x = 0$.
c) In order to obtain $\sin(x - t)$ we should impose $u(0,t) = \sin(-t)$ on the left side and set the initial u to $u(x,0) = \sin(x)$ (as done already).
d) The new modified program and its output at $t = 1$ is below:



```
# import some packages
from __future__ import print_function
import numpy as np

######################
# define some functions:
######################

# right deriv
def Dp(u, dx, du):
    for i in range(0, len(u)-1):
        du[i] = (u[i+1] - u[i])/dx

# left deriv
def Dm(u, dx, du):
    for i in range(1, len(u)):
        du[i] = (u[i] - u[i-1])/dx

# centered deriv
def D0(u, dx, du):
    for i in range(1, len(u)-1):
        du[i] = (u[i+1] - u[i-1])/(2.0*dx)

# RHS of $\partial_t u$
def eval_rhs(u, dx, du):
    Dm(u, dx, du)
    return -du

# u(t+dt) = u(t) + \partial_t u * dt
def calc_unew(u, rhs, dt):
    return u + rhs * dt

# nonsense BC at both ends
def set_BC_alt(u, dx, du, t, dt):
    u[0] = np.sin(-t)
    im = len(u)
    u[im-1] = np.sin(dx*(im-1) - t)

# BC at the side x=0 where we have incoming modes
def set_BC(u, dx, du, t, dt):
    u[0] = np.sin(-t)

# print colums with data at time t
def pr_timeframe(t, x, u):
    print("# time =", t)
    for i in range(0, len(u)):
        print(x[i], u[i])
    print()


###############
# main program:
###############

# grid: 11 points from 0 to 1, i.e. x[0]=0, ..., x[10]=1
x = np.linspace(0, 1, 11)
dx = x[1]-x[0]  # grid spacing
dt = 0.5*dx     # time step

# du will contain deriv of u, initialze to 0
du = np.zeros(len(x))

# initial u
t = 0.0
u = np.sin(x)
set_BC(u, dx, du, t, dt)
pr_timeframe(t, x, u)

timesteps = 20
```

```
# loop over time steps, and print results
for n in range(0, timesteps):
    # make time step, using simple Euler method
    t = t + dt
    rhs = eval_rhs(u, dx, du)
    u = calc_unew(u, rhs, dt)
    set_BC(u, dx, du, t, dt)
    # print colums with data
    pr_timeframe(t, x, u)
```