Packages Needed:

Scipy
Numpy
Matplotlib
astLib (specifically astStats)
cosmolopy
Python

>python --version
Python 2.7

Python 2.7 (r27:82500, Aug 9 2012, 15:39:06)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-52)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import scipy
>>> import matplotlib
>>> import astLib
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named astLib
>>> import cosmolopy
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named cosmolopy

Installation:

As of right now there is no CausticPy distribution package. The current way to use the code is simply have it in the working directory or PYTHONPATH so it can be sourced.

Scipy, Numpy, and Matplotlib can be installed and built individually; however it is much easier to install via a Python Distribution Package like Anaconda or Enthought Python Distribution. Installation will depend on OS.

ANACONDA (it is best to just to do this for Scipy, Numpy, and Matplotlib):
http://continuum.io/downloads
Download the installer shell script and run it. Say yes to everything. Re-set your python alias to point to the new python that you just installed (in Anaconda/bin)

NUMPY:
Hopefully, this is automatically installed on your system.....You'll need v1.4 or higher

MATPLOTLIB :
http://matplotlib.org/faq/

PyFITS:
http://www.stsci.edu/institute/software_hardware/pyfits/Download

ASTLIB:
astLib can be found at http://astlib.sourceforge.net/ The installation page has instructions. The astLib.astStats module is the only one used in CausticPy . GUNZIP and UNTAR and INSTALL: sudo python setup.py install

COSMOLOPY:
Cosmolopy can be found at http://roban.github.io/CosmoloPy/ or https://pypi.python.org/pypi/CosmoloPy The download and installation page has instructions and basic usage. The distance module is used for both luminosity and angular diameter distance calculations. GUNZIP and UNTAR and INSTALL: sudo python setup.py install

NOTE: I could not get ASTLIB or COSMOLOPY installed on my MAC OSX (10.6.8 w/Python 2.7.5). So I just copied the files in
/Users/christoq/Downloads/astLib-0.7.0/build/lib.macosx-10.6-universal-2.6/astLib/
and
/Users/christoq/Downloads/roban-CosmoloPy-f23ea37/cosmolopy
to my working directory. I then changed my python code to import the needed library functions directly. For example, since we use Cosmolopy's distance function, I import distance instead of cosmolopy.distance. Similarly, I import astStat instead of from astLib import astStat.

CausticMass.py Calling the code and inputs:

Takes as input the following:
1. An N column table of galaxy data with the first 3 columns being ra,dec,z (in that order) that has been imported as a numpy array. This is the default.
or
2. Two arrays, one with the projected radius in Mpc and the other the peculiar velocity (i.e., the observed redshift minus the redshift of the cluster). Since this is not the default, you need to specify GAL_R = [array] and GAL_V = [array] in your call to CausticMass.

For example:
python> c = Caustic()
>good_flag = c.run_caustic(galaxydata) # where galaxydata is a 2D numpy array with ra,dec,z for the galaxies

or

python> good_flag = c.run_caustic(galaxydata, gal_r = rdata, gal_v = vdata) #where galaxydata is a 2D array which can have anything in the same order as the rdata and vdata numpy arrays (i.e., velocity errors, magnitude errors, etc). rdata and vdata are 1D arrays with the list of galaxy projected radii in Mpc and galaxy peculiar velocities in km/s. These velocities need to be cosmologically corrected (i.e,. in the rest-frame of the cluster).

If you have magnitudes, you can include these as an addition N-dimensional array (e.g., 1D for r-band mags or 5D for SDSS u,g,r,i,z mags). The call then looks like:

python> good_flag = c.run_caustic(galaxydata,gal_mags=galaxymags) where galaxymags is your N dimensional numpy array. In a future version, the code will allow you to then specify Nbright = ngal and Nlimit = maglimit in your calls.

If you already have some idea of membership, you can specify a 1D array gal_memberflag that is filled with 0s (non-member) or 1s (members). The membership is only used in the initial velocity dispersion calibration calculation. All galaxies in your original list are still used in the phase-space density and caustic calculation.

You can also specify the location and size of the cluster:

python> good_flag = c.run_caustic(galaxydata, clus_ra=180.0, clus_dec=0.0, clus_z=0.1, r200 = 1.5)
~~**Note that the default r200 is set to 2.0 (Mpc), which is probably not what you want.**~~
The code now estimates r200 based on a scaling relation. However, it is still best to provide it with an estimate.

You can also specify the phase-space limits:
rlimit=4,0 (default in Mpc) and vlimit = 3500 (default in km/s). This defines the phase-space size so long as the following boolean is set:
cut_sample = True (default)

The density grid requires some user information ahead of time:
xmax = 6 (in Mpc) must be larger than the maximum projected radius.
ymax = 5000 (in km/s) must be larger than the maximum absolute value of the peculiar velocity in your data.
These maximums define the resolution of your phase-space grid, If you never change these limits, your output profiles are always on the same projected radius grid.
Note that you must specify these values for the code to work properly. If you have no idea what your limits are, choose some very large numbers. This in turn will stretch out your phase-space and cause your caustic surfaces to be squashed.

The kernel stretching in the velocity dimension can be defined as):

q =10 (default)
Note Geller & Diaferio state that the phase space densities are not strongly dependent on the value of this parameter. The default should be fine unless you have some weird velocity errors.

The velocity dispersion calibration step uses a bi-weighted technique. In the 2D phase-space, the code includes the option to use a shifting gapper (i.e., in bins of radius). Gifford et al. (2013) shows this to be an optimal method for velocity dispersion measurements.
gapper = True (default)

The phase-space can be mirrored (as in Gifford et al. 2013):
mirror = True (default)

Finally:
H0 = 100 (default in km/s/Mpc). Currently, the cosmology is fixed LambdaCDM Omega_m = 0.3 Omega_L = 0.7.

CausticMass.py Ouputs:

The outputs are numpy objects (in our example of "c").

c.clus_ra = cluster RA position in decimal degrees. Same if given as input. Otherwise the average of the galaxy RAs for all data you included.
c.clus_dec = cluster DEC position in decimal degrees. Same if given as input. Otherwise the average of the galaxy RAs for all data you included.
c.clus_z = cluster z position in decimal degrees. Same if given as input. Otherwise the average of the zs for all data you included.
c.ang_d = Angular diameter distance to c.clus_z
c.lum_d = Luminosity distance to c.clus_z
c.angle = 1D array (same size as galaxy input) with angular separations projected from the cluster center c.clus_ra, c.clus_dec. In RADIANS.
c.r = 1D array (same size as galaxy input) with angular separations projected from the cluster center c.clus_ra, c.clus_dec. In Mpc. Same is gal_r if given as input.
c.v = array (same size as galaxy input) of peculiar velocities. Same is gal_v if given as input.
c.data_set = After the cuts are applied (cut_sample=True, gapper=True), this is the revised input data galaxydata numpy array. Columns: r, v, followed by the original galaxydata columns
c.x_range = 1D array of projected radii grid positions Mpc (used in the gridded density map)
c.y_range = 1D array of velocity grid positions km/s (used in the gridded density map)
c.img_tot = 2D array of the phase-space density map (with axis x_range and y_range)
c.caustic_profile = 1D array The absolute value of the surface (with axis x_range) km/s
c.caustic_fit = internal use
c.gal_vdisp = Calibration velocity dispersion calculated either from c.data_set. If membership is

specified, than this velocity dispersion uses only those members. km/s
c.memflag = 1D array (same size as data_set) defining galaxies underneath the caustic envelope (1 or 0).
c.Ngal = Galaxies with c.memflag =1 and with projected radii < c.r200_est (same size as data_set).
c.vdisp_gal = Velocity dispersion for those with c.memflag = 1 (i.e., under the caustic surface).
c.r200_est = Radius (in Mpc) at which the average density equals 200 times the LCDM critical density at the redshift of the cluster.
c.M200_est = Mass (solar) within c.r200_est based on the caustic profile. This version of the code uses an NFW model profile with concentrations calibrated according to c.gal_vdisp from Bivano et al. (XXXX). The velocity anisotropy parameter is defined to be 0.2. This version does not use a calibration factor F_beta.

CausticMass.py worked example on Desktop:

I start with a dataset that looks like (90 galaxies total):

RA	DEC	u	g	r	i	z	ID	redshift
194.754	18.815	18.79	17.33	16.71	16.36	16.14	12	0.0811
194.657	19.213	18.81	18.05	17.60	17.23	17.11	12	0.1544
194.767	19.118	19.63	18.11	17.36	16.98	16.76	12	0.0810
195.083	19.164	19.88	18.27	17.47	17.11	16.85	12	0.0629

For a cluster that I know is at:
ra,dec, z = 195.095, 19.131, 0.063 and an estimate of r200 = 0.743 (h=1)

```
>>> import numpy as np
>>> from CausticMass import *
>>> galra,galdec,gal_umag, gal_gmag, gal_rmag, gal_imag, gal_zmag, gal_c4id, \
        gal_z = np.loadtxt('../complete/data/cluster_members2R_12.dat',\
        dtype='float',usecols=(0,1,2,3,4,5,6,7,8),unpack=True)
>>> galaxydata = np.vstack((galra, galdec, gal_z)).T
>>> c = Caustic()
```

Now we are ready to pass this array to the CausticMass code

```
>>>  good_flag = c.run_caustic(galaxydata,r200=0.743, clus_ra=195.095, clus_dec=19.131,clus_z=0.063)

Calculating Density w/Mirrored Data
Calculating initial surface
CausticMass.py:529: RuntimeWarning: divide by zero encountered in log
  if (np.log(newA)-np.log(pastA))/(np.log(newr)-np.log(pastr)) > 3.0:
CausticMass.py:529: RuntimeWarning: invalid value encountered in double_scalars
  if (np.log(newA)-np.log(pastA))/(np.log(newr)-np.log(pastr)) > 3.0:
CausticMass.py:479: RuntimeWarning: invalid value encountered in double_scalars
  return (np.trapz(Ar**2*phir,useri)/np.trapz(phir,useri),Ar)
CausticMass.py:534: RuntimeWarning: divide by zero encountered in log
```

```
  if (np.log(newA)-np.log(pastA))/(np.log(newr)-np.log(pastr)) < -3.0 and pastA != 0:
CausticMass.py:534: RuntimeWarning: invalid value encountered in double_scalars
  if (np.log(newA)-np.log(pastA))/(np.log(newr)-np.log(pastr)) < -3.0 and pastA != 0:
CausticMass.py:557: RuntimeWarning: invalid value encountered in sqrt
  min_func = lambda x,d0:
np.sqrt(2*4*np.pi*4.5e-48*d0*(halo_srad)**2*np.log(1+x/halo_srad)/(x/halo_srad))*3.08e19
CausticMass.py:606: RuntimeWarning: invalid value encountered in divide
  self.f_beta = 0.5*((r2/r200*self.conc)**2)/((1+((r2/r200*self.conc)))**2*np.log(1+((r2/r200*self.conc))))*self.g_b
CausticMass.py:624: RuntimeWarning: invalid value encountered in divide
  self.avg_density = self.massprofile/(4.0/3.0*np.pi*(ri[:self.f_beta.size])**3.0)
r200 estimate:  0.407905492322
M200 estimate:  1.6463727107e+13
```

Ignore the runtime warnings.
Now let's have a look:

```
>>> from pylab import *
>>> plot(c.r,c.v,'o', c='black')
[<matplotlib.lines.Line2D object at 0x2d3bdd0>]
>>> plot(c.x_range, c.caustic_profile, c='red')
[<matplotlib.lines.Line2D object at 0x2e5d290>]
>>> xlim(0,2)
(0, 2)
>>> ylim(-1500,1500)
(-1500, 1500)
>>> show()
```

or

```
>>> savefig('/Users/christoq/caustic_C4_0012.png')
>>> close()
```

To output the surface:

```
>>>f = open('caustic_C4_0012.caustic','w')
>>> for i in range(c.x_range.size):
...         f.write(str(c.x_range[i]) + ' ')
...         f.write(str(c.caustic_profile[i]) + ' ')
...         f.write('\n')
...
>>> f.close()
```

To see some other results:

```
8>>> print c.vdisp_gal
198.640687534
>>> print c.r200_est
0.407905492322
```

CausticMass.py worked example on HPC:

Log into your HPC:
ssh -X flux-login.engin.umich.edu

We'll be writing PBS scripts to call the HPC's scheduler:
Here's an example:
#!/bin/sh
#PBS -S /bin/sh
#PBS -A christoq_flux *My personal cores.*
#PBS -l qos=flux
#PBS -q flux
#PBS -N CMiller *Name in queue (choose any)*
#PBS -l nodes=1:ppn=1,walltime=30:00,pmem=4000mb *Runtime can be less than you need, but not more than walltime + 5min. One node/job is what we use for serial jobs. Default memory is ~2000mb. The memory and ppn (cores) have to match (i.e., if you have 8mb/core but you need 16mb, then ppn = 2.*
#PBS -t 0-99 *Environment variable. This means we are sending 100 jobs. Max is 999. These IDs have an environment variable called $PBS_ARRAYID and we then use an internal look in our program to make sure we can run al N 1000 clusters.*
#PBS -V
#PBS -joe
#
cd /scratch/christoq_flux/giffordw/Millenium
python flux_3Dgal.py $PBS_ARRAYID 10

For example, suppose we have ~2500 clusters with data in the format used in the above desktop example. We would set:
#PBS -t 0-26
and define our code to do 100 in a loop. The code looks like this:
'''
This program loads the C4 cluster catalog and galaxy data files associated with each cluster. It calls the CausticMass.py program which is in development as the master caustic program for our work.

The reason is that the cluster files live in Chris Miller's local drive which can only be accessed via scp.
'''
```
import numpy as np
from CausticMass import *
import pyfits as fits
#from mpl_toolkits.mplot3d import Axes3D
import os
```

```
import sys

#import cluster file (fits format)
cfit = fits.open('/nfs/christoq_ls/C4/sdssdr7/c4_cluster_single.fit')
Nclusters = cfit[1].data.field('KEEP').size

try: #if the user feeds a number at the command line
        ar = np.int(sys.argv[1])
except IndexError:
        ar = 0
for u in range(100):#Nclusters):
        i = 100*ar+u
        if cfit[1].data.field('Z_BIWT')[i] == 0 or cfit[1].data.field('KEEP')[i] == 0:
                continue
        try:
                #columns: 0:RA  1:DEC  2:umag  3:gmag  4:rmag  5:imag  6:zmag  7:ID  8:redshift
                data_gal = np.genfromtxt('/nfs/christoq_ls/C4/sdssdr7/data/cluster_members2R_'+str(i)+'.dat')
        except:
                print 'No cluster file',str(i)
                continue
        data_gal =
np.vstack((data_gal[:,0],data_gal[:,1],data_gal[:,8],data_gal[:,2],data_gal[:,3],data_gal[:,4],data_gal[:,5],data_gal[:,6],data_gal[:,7])).T
         c = Caustic()
         good_flag = c.run_caustic(galaxydata,r200=cfit[1].data.field('rho200_rv1500')[i], clus_ra=cfit[1].data.field('ra_bcg')[i],
        clus_dec=cfit[1].data.field('dec_bcg')[i],  \
              clus_z=cfit[1].data.field('z_biwt')[i],rlimit=5)
        f = open('/nfs/christoq_ls/C4/sdssdr7/data/caustics/cluster_caustics2R_'+str(i)+'.dat', 'w')
        for j in range(c.x_range.size):
                f.write(str(c.x_range[j]) + ' ')
                f.write(str(c.caustic_profile[j]) + ' ')
                f.write('\n')
        f.close()
```

Note that we are loop in u over 100 clusters for each call to this program. If we have 2550 clusters, we call this program 26 times (i.e., 26 PBS scheduled jobs). Each time, we are looping over cluster IDs from 100*$PBS_ARRAYID + u (or i)

We are getting the cluster positions (zs and r200s) from a fits table that lists these and other data in the same order as 100*$PBS_ARRAYID + u

The data files are identified with the same ID as their row (or ID) in the FITs table
data_gal = np.genfromtxt('/nfs/christoq_ls/C4/sdssdr7/data/cluster_members2R_'+str(i)+'.dat')

```
[christoq@flux-login2 caustics]$ ls
c4_caustics.py  C4_caustics.sh  CausticMass.py
[christoq@flux-login2 caustics]$ qsub C4_caustics.sh
10760936[].nyx.engin.umich.edu
[christoq@flux-login2 caustics]$ qstat -u christoq
[christoq@flux-login2 caustics]$ qdel 10760936[]
```