TSC_CM_different_prior

Unknown Author

February 8, 2014

Author: Karen Yin-Yee Ng

This program calculates the CM frame relic position contraints

In [469]:
```
from __future__ import division
%autosave 60
%load_ext nbtoc
%nbtoc
from astropy import wcs
import astropy.coordinates as coord
import astropy.units as unit
from astropy.coordinates import Angle
import astropy.io.fits
from astropy.cosmology import FlatLambdaCDM
from astropy.coordinates import ICRS
import numpy as np

import time
import numpy.random as rand
import pdb
import pandas as pd
import sys
tstart = time.time()
#sys.path.append('/afs/sapphire.physics.ucdavis.edu/home/karenyng/Documents/Research_c
```

```
Autosaving every 60 seconds
The nbtoc extension is already loaded. To reload it, use:
  %reload_ext nbtoc
```

In [470]:
```
from wcs_ICRS import wcs_ICRS as WICRS
```

load homebrew modules

In [471]:
```
from plotmod_dict import *
import cosmo
```

Part I

Initialization !!!!!!

In [472]:
```
#---initializations-----------------------------------------------------------
data_path = "/Users/karenyng/Documents/Research/code"+\
            "/TSM/mercury_elGo/Feb_data/"
prefix = data_path + "ElGordo_"
oprefix = data_path + "polar_"
index = [ str(i) for i in range(20) ]

NW_relic = ICRS('01h02m46s -49d14m43s',
                unit=(unit.degree, unit.degree))
SE_relic = ICRS('01h03m01s -49d17m14s',
                unit=(unit.degree, unit.degree))
#Histogram bins
N_bins_2d = 130
N_bins_1d = 200
N_bins_TSM = 45
N_bins_alpha = 90

#specify the number of iterations
Iter = 500000
#Iter = 475000
par=['m_1',
     'm_2',
     'z_1',
     'z_2',
     'd_proj',
     'v_rad_obs',
     'alpha',
     'v_3d_obs',
     'd_3d',
     'v_3d_col',
     'd_max',
     'TSM_0',
     'TSM_1',
     'T',
     'prob']

fitspath = '/Users/karenyng/Documents/Research/code/'+\
           'ElGordo-Dynamics-Paper/Analysis/HSTlensing/'
fits = fitspath + 'header.fits'
getwcs = WICRS('01h03m01s -49d17m14s', fitsname=fits)
w = getwcs.wcs
```

load pickles into suitable formats

In [473]:
```
data = load_pickles_to_df(par, prefix, index)
```

```
converting entry m_1 to units of 1e14 m_sun
converting entry m_2 to units of 1e14 m_sun
```

1 Calculate CM coordinates in pixel then convert back to WCS coordinates

In [492]:
```
# these values r from El Gordo WL paper latest arxiv version 2
NW_pix = np.array
NW = ICRS('01h02m50.60s -49d15m04.5s')
SE = ICRS('01h02m56.31s -49d16m23.2s')
[NW_pix, SE_pix] = w.wcs_world2pix(np.array([[NW.ra.deg, NW.dec.deg],
                                             [SE.ra.deg, SE.dec.deg]]), 1)
```

In [494]:
```
### the following values are from James via private communication
### and are what we use for our MCMAC

#NW_pix = np.array([2966.6336, 5431.9614])
#SE_pix = np.array([2954.5809, 3152.1517])
#[NW_wcs, SE_wcs] = w.wcs_pix2world(np.array([NW_pix, SE_pix]), 1)
```

In [495]:
```
#NW = ICRS('01h02m51.68s -49d15m04.40s')
#SE = ICRS('01h02m38.38s -49d16m37.64s')
#[NW_pix, SE_pix] = w.wcs_world2pix(np.array([[NW.ra.deg, NW.dec.deg],
                                    #         [SE.ra.deg, SE.dec.deg]]), 1)
```

Just use the mean of the mass blobs as the center of mass location

In [524]:
```
cm_pix_x = (NW_pix[0] * NW_mean_mass + SE_pix[0] * SE_mean_mass) / \
(NW_mean_mass + SE_mean_mass)
cm_pix_x
```

Out [524]:
```
3133.7272010974093
```

In [526]:
```
cm_pix_y = (NW_pix[1] * NW_mean_mass + SE_pix[1] * SE_mean_mass) / \
(NW_mean_mass + SE_mean_mass)
cm_pix_y
```

Out [526]:
```
4725.4838968134009
```

In [529]:
```
NW_pix
```

Out [529]:
```
array([ 3142.04012987,  5551.36265773])
```

In [512]:
```
NW_mean_mass = data['m_1'].mean()
SE_mean_mass = data['m_2'].mean()
cm_pix = (NW_pix * NW_mean_mass + SE_pix * SE_mean_mass) /\
        (NW_mean_mass + SE_mean_mass)
[cm_wcs, junk] = w.wcs_pix2world(np.array([cm_pix, cm_pix]), 1)
cm_wcs = ICRS(ra=cm_wcs[0], dec=cm_wcs[1],
              unit=(unit.degree, unit.degree))
SE_relic_CM_sep = cm_wcs.separation(SE_relic)
NW_relic_CM_sep = cm_wcs.separation(NW_relic)
print cm_wcs
```

```
<ICRS RA=15.72101 deg, Dec=-49.26060 deg>
```

In [505]:
```
[NW_relic_pix, SE_relic_pix] = w.wcs_world2pix(np.array([[NW_relic.ra.deg,
                                                         NW_relic.dec.deg],
                                                        [SE_relic.ra.deg,
                                                         SE_relic.dec.deg]]), 1)
```

2 Compare all the relevant coordinates side by side

In [655]:
```
print NW_relic_pix, SE_relic_pi
```

```
[ 3636.50083636  6418.49706285] [ 2949.2945903   2262.66993921]
```

Something went wrong with the coordinates, let's try to visualize

In [664]:
```
dec_axis = np.arange(-49.30, -49.19, 0.02)
ra_axis = np.arange(15.68, 15.76, 0.01)
print dec_axis
print ra_axis

def ra_deg_to_wcs(deg):
    deg = deg / 15.
    [h, m] = str(deg).split(".")
    [m, s] = str(float("." + m) * 60 ).split(".")
    s = '{0:.1f}'.format( float("." + s) * 60.)
    return h + 'h' + m + 'm' + s + 's'

def dec_deg_to_wcs(deg):
    [d, m] = str(deg).split(".")
    [m, s] = str(float("." + m) * 60).split(".")
    s = '{0:.1f}'.format( float("." + s) * 60.)
    return d + 'h' + m + 'm' + s + 's'

raAXIS = [ra_deg_to_wcs(ra) for ra in ra_axis]
decAXIS = [dec_deg_to_wcs(dec) for dec in dec_axis]
print raAXIS
print decAXIS
```

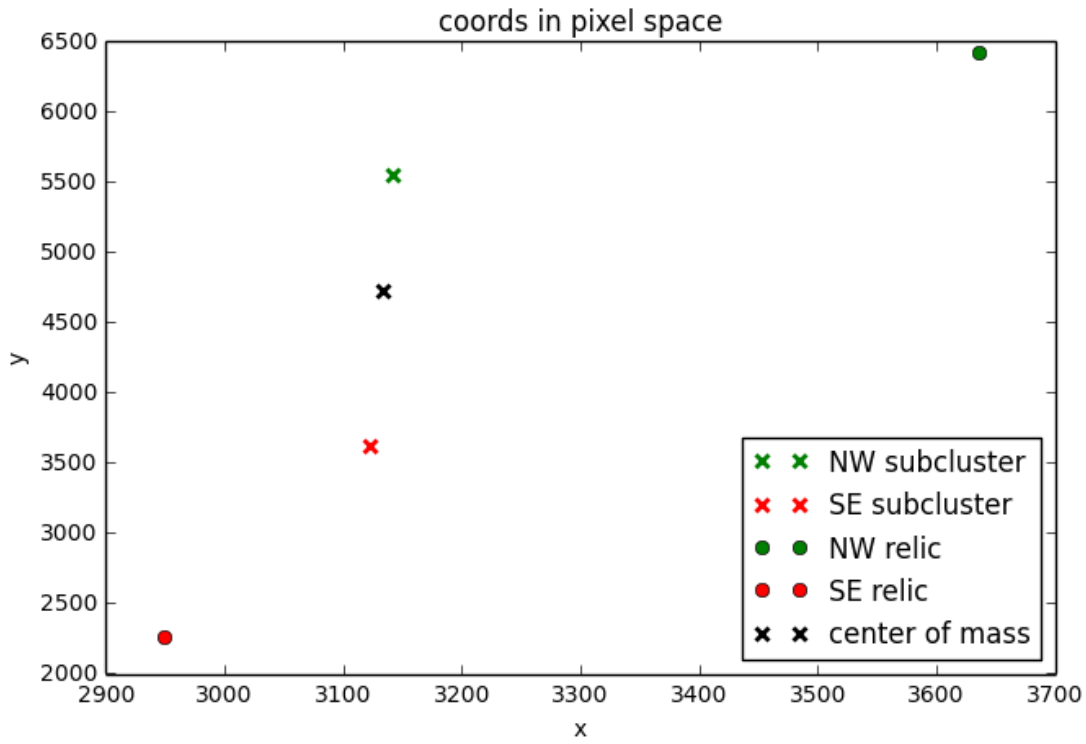
```
[-49.3  -49.28 -49.26 -49.24 -49.22 -49.2 ]
[ 15.68  15.69  15.7   15.71  15.72  15.73  15.74  15.75  15.76]
['1h2m43.2s', '1h2m45.6s', '1h2m48.0s', '1h2m50.4s', '1h2m52.8s',
'1h2m55.2s', '1h2m57.6s', '1h3m0.0s', '1h3m2.4s']
['-49h18m0.0s', '-49h16m48.0s', '-49h15m36.0s', '-49h14m24.0s',
'-49h13m12.0s', '-49h12m0.0s']
```

In [665]:
```
plt.plot(NW_pix[0], NW_pix[1], "gx", label="NW subcluster",
         markeredgewidth=2)
plt.plot(SE_pix[0], SE_pix[1], "rx", label="SE subcluster",
         markeredgewidth=2)
plt.plot(NW_relic_pix[0], NW_relic_pix[1], "go", label="NW relic")
plt.plot(SE_relic_pix[0], SE_relic_pix[1], "ro", label="SE relic")
plt.plot(cm_pix[0], cm_pix[1], "kx", label="center of mass",
         markeredgewidth=2)
plt.legend(loc="lower right")
ylim_min, ylim_max = plt.ylim()
```

```
#plt.ylim(-49.3, -49.2)
#plt.xlim(15.69, 15.76)

plt.xlabel('x')
plt.ylabel('y')
plt.title('coords in pixel space')
```

Out [665]:

```
<matplotlib.text.Text at 0x115210850>
```



In [656]:

```
print 'NW_pix = {0} '.format(NW_pix)
print 'SE_pix = {0} '.format(SE_pix)
print 'NW_relic_pix = {0}'.format(NW_relic_pix)
print 'SE_relic_pix = {0}'.format(SE_relic_pix)
print 'cm_pix = {0} '. format(cm_pix)
```

```
NW_pix = [ 3142.04012987  5551.36265773]
SE_pix = [ 3122.60864656  3620.86983129]
NW_relic_pix = [ 3636.50083636  6418.49706285]
SE_relic_pix = [ 2949.2945903   2262.66993921]
cm_pix = [ 3133.7272011   4725.48389681]
```

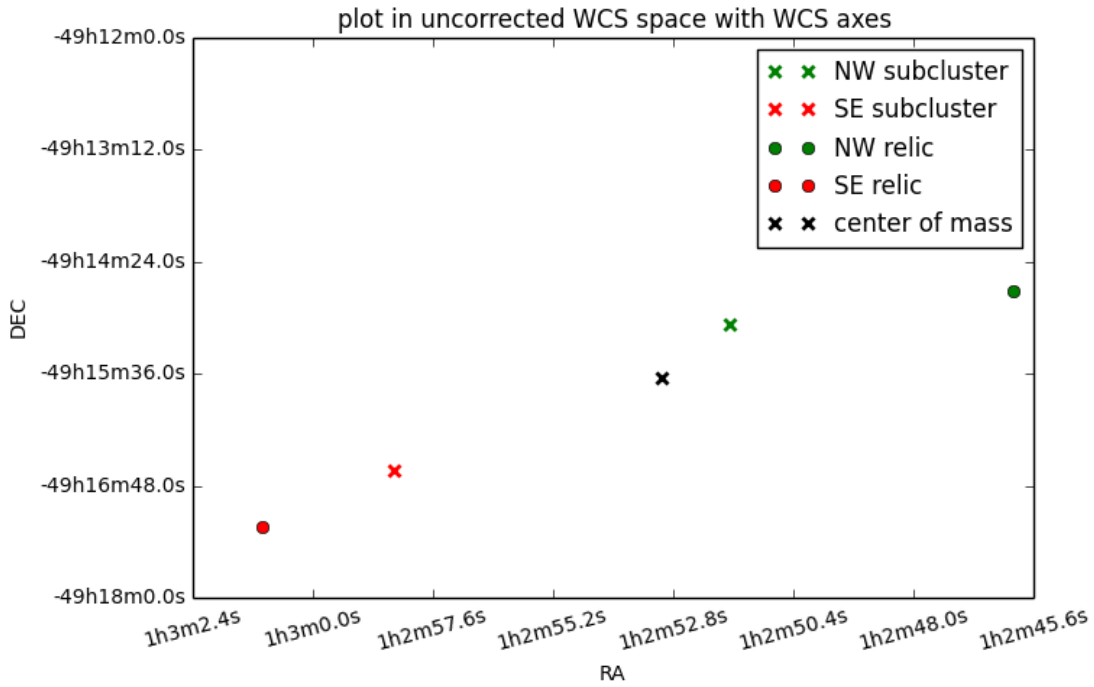
In [666]:

```
plt.plot(NW_wcs[0], NW_wcs[1], "gx", label="NW subcluster",
         markeredgewidth=2)
plt.plot(SE_wcs[0], SE_wcs[1], "rx", label="SE subcluster",
         markeredgewidth=2)
plt.plot(NW_relic.ra.deg, NW_relic.dec.deg, "go", label="NW relic")
plt.plot(SE_relic.ra.deg, SE_relic.dec.deg, "ro", label="SE relic")
plt.plot(cm_wcs.ra.deg, cm_wcs.dec.deg, "kx", label="center of mass",
         markeredgewidth=2)
```

```
a = plt.gca()
a.get_xaxis().get_major_formatter().set_useOffset(False)
a.get_yaxis().get_major_formatter().set_useOffset(False)

plt.ylim(-49.3, -49.2)
xlim_min, xlim_max = plt.xlim()
plt.xlim(xlim_max, xlim_min)
plt.ylim(-49.3, -49.2)
plt.xlim(15.76, 15.69)
plt.title('plot in uncorrected WCS space with WCS axes')

a = plt.gca()
a.set_xticklabels(raAXIS, rotation = 15)
a.set_yticklabels(decAXIS)
plt.xlabel("RA")
plt.ylabel("DEC")
plt.legend(loc = 'best')
```

Out [666]:

```
<matplotlib.legend.Legend at 0x1152068d0>
```



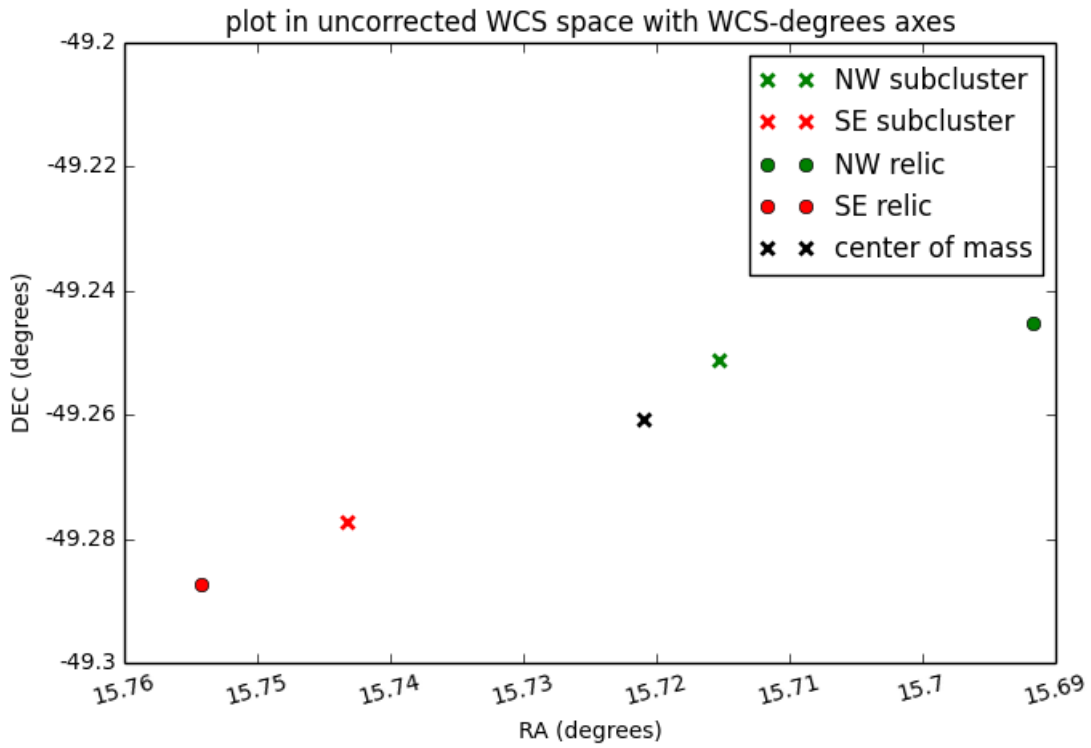
In [667]:

```
plt.plot(NW_wcs[0], NW_wcs[1], "gx", label="NW subcluster",
         markeredgewidth=2)
plt.plot(SE_wcs[0], SE_wcs[1], "rx", label="SE subcluster",
         markeredgewidth=2)
plt.plot(NW_relic.ra.deg, NW_relic.dec.deg, "go", label="NW relic")
plt.plot(SE_relic.ra.deg, SE_relic.dec.deg, "ro", label="SE relic")
plt.plot(cm_wcs.ra.deg, cm_wcs.dec.deg, "kx", label="center of mass",
         markeredgewidth=2)
a = plt.gca()
a.get_xaxis().get_major_formatter().set_useOffset(False)
a.get_yaxis().get_major_formatter().set_useOffset(False)

plt.ylim(-49.3, -49.2)
xlim_min, xlim_max = plt.xlim()
plt.xlim(xlim_max, xlim_min)
```

```
plt.ylim(-49.3, -49.2)
plt.xlim(15.76, 15.69)
plt.title('plot in uncorrected WCS space with WCS-degrees axes')
plt.xlabel("RA (degrees)")
plt.ylabel("DEC (degrees)")
plt.legend(loc = 'best')

a = plt.gca()
a.set_xticklabels(ra_axis, rotation = 15)
a.set_yticklabels(dec_axis)
```

Out [667]:

```
[<matplotlib.text.Text at 0x1162b0050>,
 <matplotlib.text.Text at 0x1162aae10>,
 <matplotlib.text.Text at 0x11794a690>,
 <matplotlib.text.Text at 0x11794ab90>,
 <matplotlib.text.Text at 0x117947310>,
 <matplotlib.text.Text at 0x117947a50>]
```



In []:

```
NW = ICRS(NW_wcs[0], NW_wcs[1], unit = (unit.deg, unit.deg))
SE = ICRS(SE_wcs[0], SE_wcs[1], unit = (unit.deg, unit.deg))
```

use ΛCDM

In []:

```
cosmo = FlatLambdaCDM(H0=70, Om0=0.3)
r_proj_SE = cosmo.angular_diameter_distance(.87) * \
            SE_relic_CM_sep.rad
r_proj_NW = cosmo.angular_diameter_distance(.87) * \
            NW_relic_CM_sep.rad

print "SE relic separation from CM is {0}".format(r_proj_SE)
```

```
print "NW relic separation from CM is {0}".format(r_proj_NW)
```

Now repeat the calculation for the separation of the NW subcluster to the center of mass

In []:
```
NW_subclstr_sep = cm_wcs.separation(NW).rad * \
                  cosmo.angular_diameter_distance(.87)
print NW_subclstr_sep
```

Which is consistent with the picture given by Lindner et al. 's paper Fig 1.

Part II

How to do the relic calculation

3 Info from Lindner et al. 2013

> "shock speed can be interpreted as an upper limit on the collision speed"

implies that this a relative speed between the two subclusters?

This is different than saying that the shock speed is in the CM frame.

3.1 the position of the center of mass and thus the separations of the relics to the CM

- using the full realizations takes up a lot of time for doing conversion between coordinates and calculating the physical separation since we have half a million realizations and the code is not vectorized - python loops are slow
- using the mean location of the center of mass which is a lot faster but will not fold the uncertainties in — maybe I should rewrite this part using Cython

3.2 the velocities of the relics

we do not know the time evolution of the speed but we can assume an average velocity to be either:

- the collisional velocity of the NW subcluster
- the collision velocity of the NW subcluster times a slow-down factor

In the center of mass frame, the speed of m_1 and m_2 are related by:

$$v_{2,CM} = -\frac{m_1}{m_2} v_{1,CM}$$

– (1)

but in our simulation, we only know about the relative velocities of the two subclusters,

$v_{rel} = v_1 - v_2$ – (2) which is true in any frame

Plug (1) in (2) to get

$$v_{1,CM} = \left(\frac{m_2}{m_1 + m_2}\right) v_{rel}$$
$$v_{2,CM} = -\left(\frac{m_1}{m_1 + m_2}\right) v_{rel}$$

3.3 Relic calculation - v.1 - just getting a feeling of how the distances compare

- using a mean location for the center of mass
- using NW $v_{relic} = 4300$ km/s ± 800 km/s assuming this is in CM frame

Is this shock speed in 3D or 2D??? it makes a difference - should be in 3D. . . Calculation of the simulated separation for relic from the CM:

$$s_{relic} = v_{relic_i}\, t_{tsc_j} \cos(\alpha)$$

There will be 2 outputs from the calculations just from simulation since there are two TSCs.Unit conversion is done to convert units of (km /s * Gyr) to Mpc

Mpc / km = $60 \times 60 \times 365 \times 24 \times 1e9/(1e13 \times 1e6 \times 3.086)$

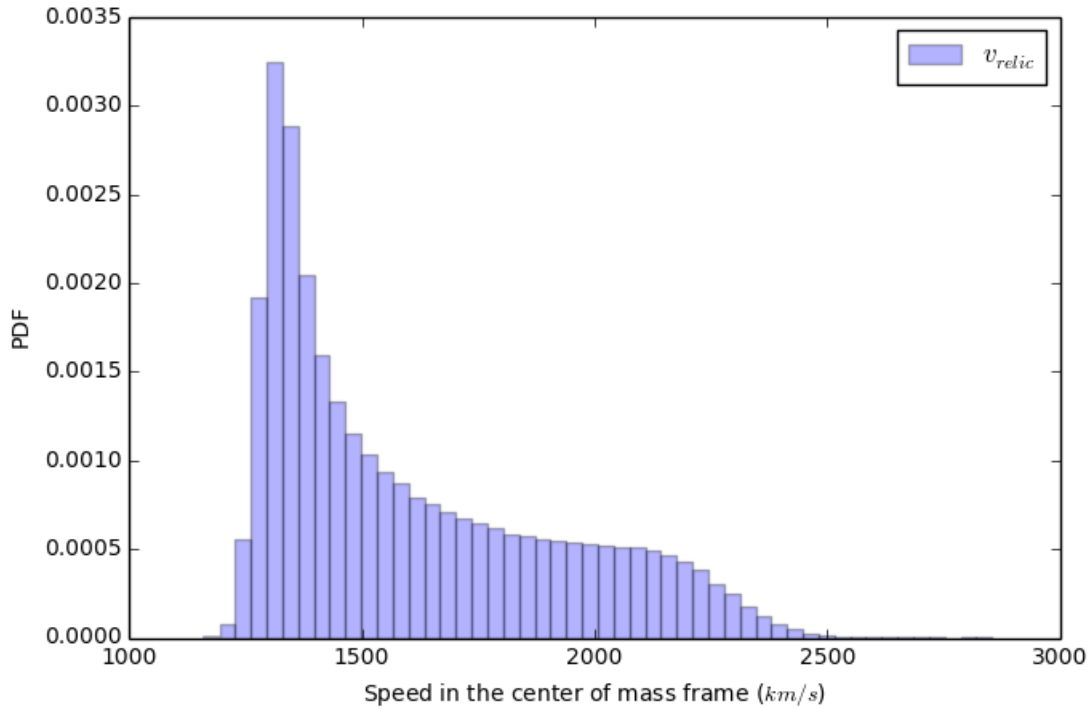
3.4 Initialize the velocities

In [530]:
```
data['v_relic'] = data['v_3d_col'] * SE_mean_mass /\
                  (NW_mean_mass + SE_mean_mass)
```

3.5 See if the speed distribution looks alright

In [531]:
```
j1, j2, j3 = plt.hist(data['v_relic'], histtype = 'bar', bins=50,
                      label = r'$v_{relic}$', alpha=0.3, normed=True)
plt.ylabel('PDF')
plt.xlabel(r'Speed in the center of mass frame ($km / s$)')
plt.legend(loc = 'best')
```

Out [531]:
```
<matplotlib.legend.Legend at 0x10e18e850>
```



3.6 Initialize the lower bound of the predicted location

$$x_{NW} = \frac{m_{SE}}{m_{NW}} x_{SE}$$

$$x_{rel} = x_{NW} - x_{SE}$$

$$x_{NW} = \frac{m_{SE}}{m_{NW} + m_{SE}} x_{rel}$$

In [533]:
```
# convert m to Mpc , Myrs to seconds
unitconversion = 60 * 60 * 365 * 24 * 1e9 / (1e13 * 1e6 * 3.086)

data['r_proj0_min'] = data['d_3d'] * \
                        np.cos(data['alpha'] / 180. * np.pi) * \
                        (SE_mean_mass) / (SE_mean_mass + NW_mean_mass)
data['r_proj1_min'] = data['d_3d'] * \
                        np.cos(data['alpha'] / 180. * np.pi) * \
                        (SE_mean_mass) / (SE_mean_mass + NW_mean_mass)

data = data[~np.isnan(data['r_proj0_min'])]
data = data[~np.isnan(data['r_proj1_min'])]
```

3.7 Initialize the upper bound of the predicted location

In [534]:
```
data['r_proj0_max'] = (data['v_relic'] * data['TSM_0'] *
                       np.cos(data['alpha'] / 180*np.pi)) * unitconversion
data['r_proj1_max'] = (data['v_relic'] * data['TSM_1'] *
                        np.cos(data['alpha'] / 180*np.pi)) * unitconversion
```

```
data = data[~np.isnan(data['r_proj0_max'])]
data = data[~np.isnan(data['r_proj1_max'])]
```

3.8 Include the uncertainty of the width of the relic

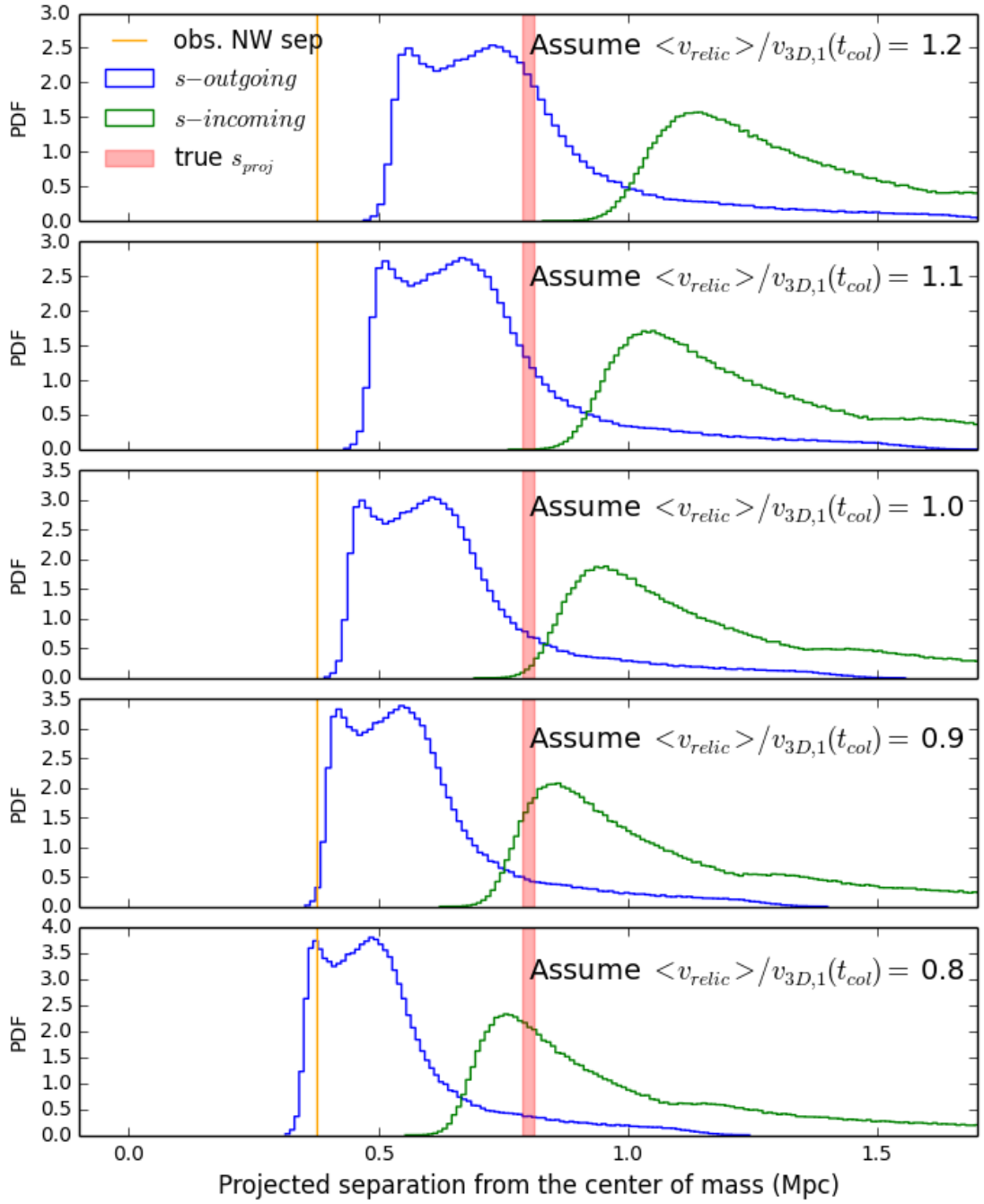
In [535]:
```
rupper = r_proj_NW.value - 23 / 2 / 1000
rlower = r_proj_NW.value + 23 / 2 / 1000
print "range is {0}, {1}".format(rupper, rlower)
```

```
range is 0.788629498264, 0.811629498264
```

In [617]:
```
def make_hist(this_ax, data1, data2, data1_mean, data2_mean, beta, i):
    j1, j2, j3 = this_ax.hist(data1,
                      histtype='step', bins=100, label=r'$s-outgoing$',
                      normed=True)
    j1, j2, j3 = this_ax.hist(data2,
                      histtype='step', label=r'$s-incoming$',
                      bins = 1000, normed = True)
    #this_ax.axvline(data1_mean, color='blue', ls='-.',label=r'$<s-outgoing>$')
    #this_ax.axvline(data2_mean, color='green', ls='-.',label=r'$<s-incoming>$')
    this_ax.axvspan(rlower, rupper, color='red', label=r'true $s_{proj}$', alpha=0.3)
    this_ax.axvline(NW_subclstr_sep.value, color='orange', label=r'obs. NW sep')
    this_ax.set_xlim(-0.1, 1.7)
    this_ax.set_ylabel('PDF')
```

3.9 Create plot with varying beta ratio

In [623]:
```
beta = np.arange(1.2, 0.7, -0.1)
f, ax = plt.subplots(beta.size, sharex=True)

for i in range(beta.size):
    # initialize the different ratio of velocities
    temp1 = data['v_3d_col'] * data['TSM_0'] * \
            np.cos(data['alpha'] / 180 * np.pi) * beta[i] * \
            unitconversion * SE_mean_mass / \
            (SE_mean_mass + NW_mean_mass)
    data1 = temp1[temp1 < 10]
    data1_mean = np.mean(data1)
    temp2 = data['v_3d_col'] * data['TSM_1'] * \
            np.cos(data['alpha'] / 180 * np.pi) * beta[i] * \
            unitconversion * SE_mean_mass / \
            (SE_mean_mass + NW_mean_mass)
    data2 = temp2[temp2 < 10]
    data2_mean = np.mean(data2)
    make_hist(ax[i], data1, data2, data1_mean, data2_mean, beta[i], i)

f.subplots_adjust(hspace=0.1)
f.set_size_inches(8, beta.size * 2)
plt.xlabel('Projected separation from the center of mass (Mpc)', size = 12)
#ax[0].legend(bbox_to_anchor=(1.05, 0), loc='lower left', borderaxespad=0.)
ax[0].legend(bbox_to_anchor=(0.005, 0.1), loc='lower left', frameon=False)

# text explanation
for i in range(beta.size):
    plt.text(.8, 3.5 + 4.45 * i, r"Assume $<v_{relic}> / v_{3D,1}(t_{col}) =$" + \
                  " {0}".format(beta[beta.size - i - 1]), va='top', size = 14)

plt.savefig("default_prior_bounds.pdf", bbox_inches="tight")
```



turns out we need to use proper bin size !Let us zoom in on the part where the relic position is

Part III

Apply radio relic polarization prior

In [605]:
```
radio_polar_prior
```

Out [605]:
```
<function plotmod_dict.radio_polar_prior>
```

In [538]:
```
radiomask1, count1 = radio_polar_prior(data['alpha'])
radiomask = radiomask1
r_data= data.copy(deep=True)
r_data = r_data[radiomask]
print 'length of data after applying prior is'+\
        ' {0}'.format(len(r_data['d_3d']))
```

```
length of data after applying prior is 268971
```

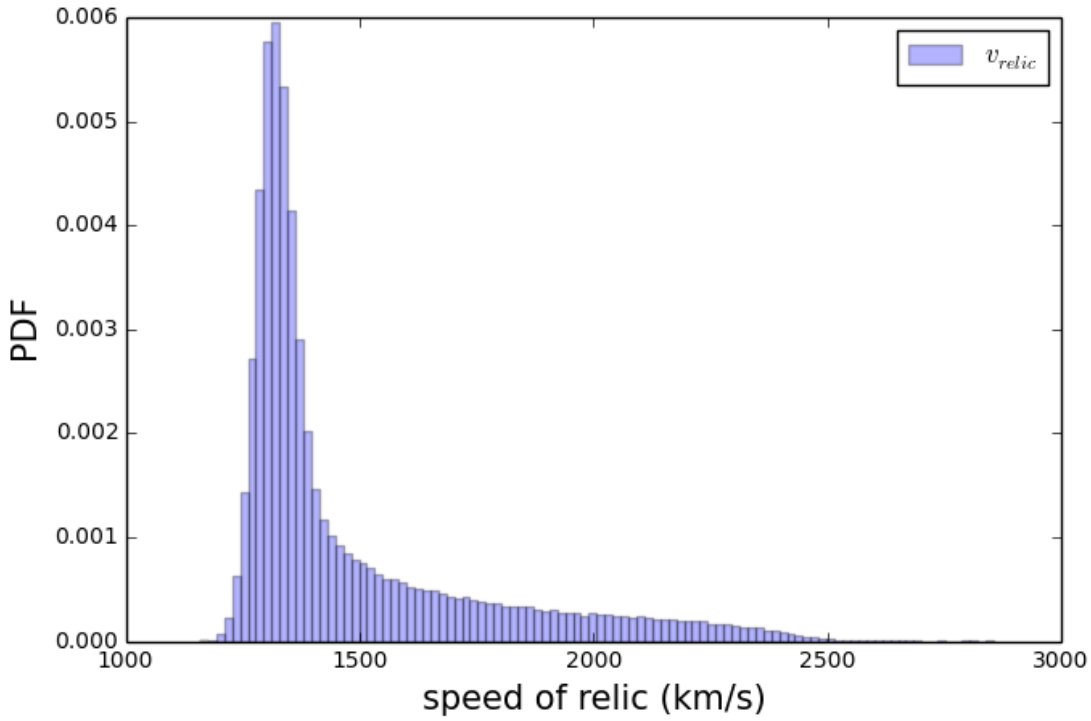
3.10 Initialize the speeds for the data with polarization data

In [539]:
```
r_data['v_relic'] = r_data['v_3d_col'] * SE_mean_mass / \
                    (SE_mean_mass + NW_mean_mass)
```

3.11 See if the speed distribution looks alright

In [608]:
```
j1, j2, j3 = plt.hist(r_data['v_relic'], histtype = 'bar',
                      label = r'$v_{relic}$', alpha=0.3, normed=True, bins = 100)

plt.xlabel('speed of relic (km/s)', size=15)
plt.ylabel('PDF', size=15)
plt.legend(loc = 'best')
```

Out [608]:
```
<matplotlib.legend.Legend at 0x1168a0c50>
```



3.12 Calculate the predicted separation

In [541]:
```
r_data.columns
```

Out [541]:
```
Index([u'm_1', u'm_2', u'z_1', u'z_2', u'd_proj', u'v_rad_obs',
u'alpha', u'v_3d_obs', u'd_3d', u'v_3d_col', u'd_max', u'TSM_0',
u'TSM_1', u'T', u'prob', u'v_relic', u'r_proj0_min', u'r_proj1_min',
u'r_proj0_max', u'r_proj1_max'], dtype='object')
```

In [542]:
```
r_data['TSM_0'][r_data['TSM_0']<6].mean()
```

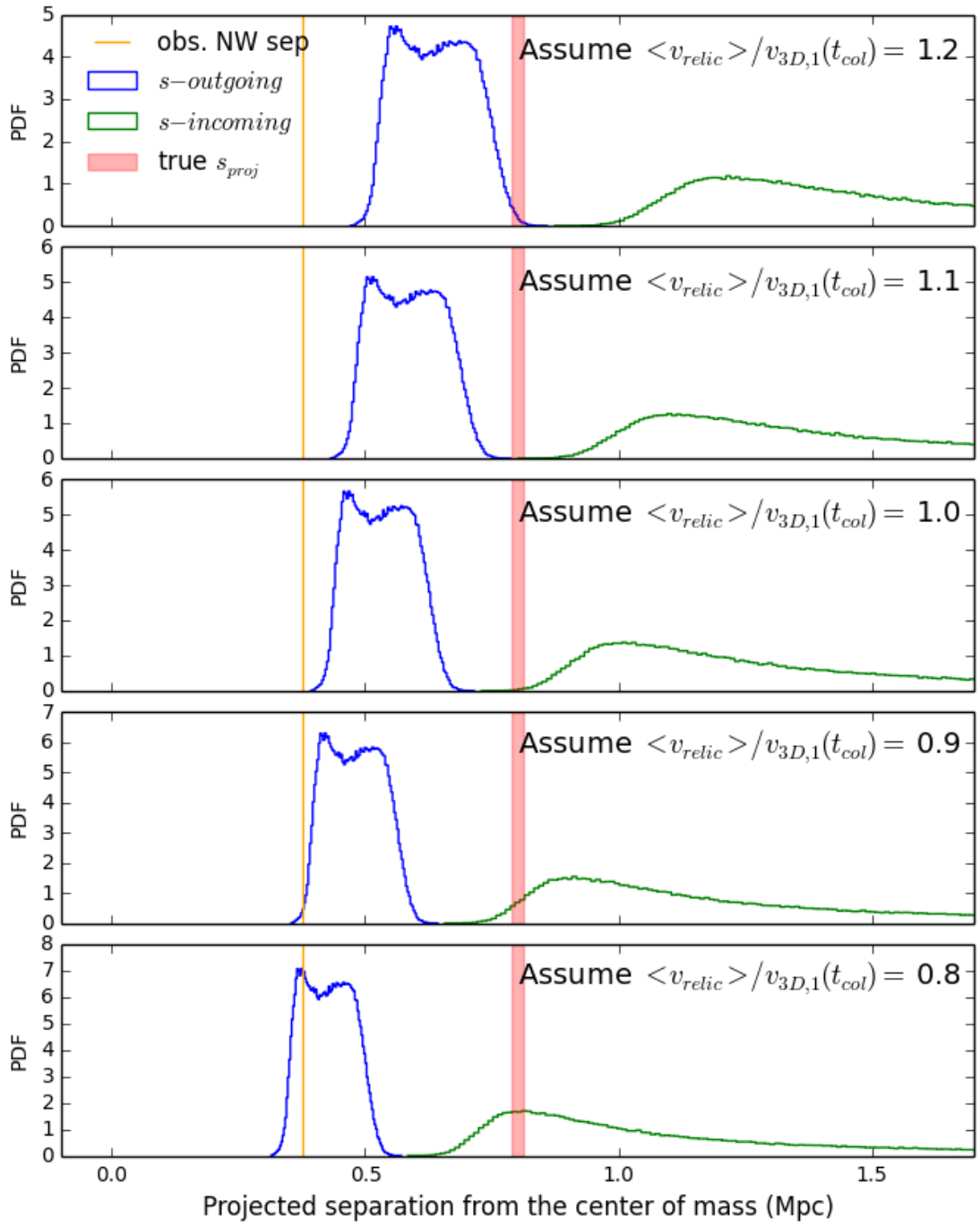
Out [542]:
```
0.40473912377000321
```

In [543]:
```
r_data['TSM_1'][r_data['TSM_1']<6].mean()
```

Out [543]:
```
1.3572542421114591
```

In [621]:
```
beta = np.arange(1.2, 0.7, -0.1)
f, ax = plt.subplots(beta.size, sharex=True)

for i in range(beta.size):
    # initialize the different ratio of velocities
    temp1 = r_data['v_3d_col'] * r_data['TSM_0'] * \
            np.cos(r_data['alpha'] / 180 * np.pi) * beta[i] *\
```

```
            unitconversion * SE_mean_mass /\
            (SE_mean_mass + NW_mean_mass)
    r_data1 = temp1[temp1 < 10]
    r_data1_mean = np.mean(r_data1)
    temp2 = r_data['v_3d_col'] * r_data['TSM_1'] * \
            np.cos(r_data['alpha'] / 180 * np.pi) * beta[i] *\
            unitconversion * SE_mean_mass /\
            (SE_mean_mass + NW_mean_mass)
    r_data2 = temp2[temp2 < 10]
    r_data2_mean = np.mean(r_data2)
    make_hist(ax[i], r_data1, r_data2, r_data1_mean,
              r_data2_mean, beta[i], i)

f.subplots_adjust(hspace=0.1)
f.set_size_inches(8, beta.size * 2)
plt.xlabel('Projected separation from the center of mass (Mpc)', size = 12)
#ax[0].legend(bbox_to_anchor=(1.05, 0), loc='lower left', borderaxespad=0.)
ax[0].legend(bbox_to_anchor=(0.005, 0.1),loc='lower left', frameon=False)

for i in range(beta.size):
    plt.text(.8, 7.5 + 8.75 * i,
                 r"Assume $<v_{relic}> / v_{3D, 1}(t_{col}) =$" + \
                 " {0}".format(beta[beta.size - i - 1]), va='top', size = 14)

plt.savefig("polar_prior_bounds.pdf", bbox_inches="tight")
```


obs. NW sep
$s-outgoing$
$s-incoming$
true s_{proj}
Assume $<v_{relic}>/v_{3D,1}(t_{col}) = 1.2$
Assume $<v_{relic}>/v_{3D,1}(t_{col}) = 1.1$
Assume $<v_{relic}>/v_{3D,1}(t_{col}) = 1.0$
Assume $<v_{relic}>/v_{3D,1}(t_{col}) = 0.9$
Assume $<v_{relic}>/v_{3D,1}(t_{col}) = 0.8$
PDF
Projected separation from the center of mass (Mpc)

Part IV

Sort the different arrays to get the extreme values as bounds

4 First from the data with default prior

In [545]:
```
data.columns
```

Out [545]:
```
Index([u'm_1', u'm_2', u'z_1', u'z_2', u'd_proj', u'v_rad_obs',
u'alpha', u'v_3d_obs', u'd_3d', u'v_3d_col', u'd_max', u'TSM_0',
u'TSM_1', u'T', u'prob', u'v_relic', u'r_proj0_min', u'r_proj1_min',
u'r_proj0_max', u'r_proj1_max'], dtype='object')
```

In [546]:
```
print "true location of relic from CM is\n" + \
" {0:.2f} Mpc".format(r_proj_NW.value)
```

```
true location of relic from CM is
 0.80 Mpc
```

In [547]:
```
print "the bounds with default priors for outgoing scenario is\n" + \
"{0:.2f} Mpc < r_relic < {1:.2f} Mpc".format(data['r_proj0_min'].min(),
                                            data['r_proj0_max'].max())
```

```
the bounds with default priors for outgoing scenario is
0.37 Mpc < r_relic < 1.56 Mpc
```

In [548]:
```
print "the bounds with default priors for incoming scenario is\n" + \
"{0:.2f} Mpc < r_relic < {1:.2f} Mpc".format(data['r_proj1_min'].min(),
                                            data['r_proj1_max'].max())
```

```
the bounds with default priors for incoming scenario is
0.37 Mpc < r_relic < 43616847.30 Mpc
```

In [549]:
```
print "the bounds with polarization priors for outgoing scenario is\n" + \
"{0:.2f} Mpc < r_relic < {1:.2f} Mpc".format(r_data['r_proj0_min'].min(),
                                            r_data['r_proj0_max'].max())
```

```
the bounds with polarization priors for outgoing scenario is
0.37 Mpc < r_relic < 0.72 Mpc
```

In [550]:
```
print "the bounds with polarization priors for incoming scenario is\n" + \
"{0:.2f} Mpc < r_relic < {1:.2f} Mpc".format(r_data['r_proj1_min'].min(),
                                            r_data['r_proj1_max'].max())
```

the bounds with polarization priors for incoming scenario is
0.37 Mpc < r_relic < 43616847.30 Mpc