TSC_CM_different_prior

Unknown Author

January 30, 2014

Author: Karen Yin-Yee Ng

This program calculates the CM frame relic position contraints

In [85]:
```
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 [86]:
```
from wcs_ICRS import wcs_ICRS as WICRS
```

load homebrew modules

```
In [87]: from plotmod_dict import *
         import cosmo
```

Part I

Initialization !!!!!!

```
In [88]: #---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) ]
         SE_centroid = ['01h02m38.38s', '-49d16m37.64s']
         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 [89]: 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
```

calculate the separation of the CM from the two relics

In [90]:
```
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 [91]:
```
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)
```

use ΛCDM

In [92]:
```
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)
```

```
NW relic separation from CM is 0.47281510798 Mpc
```

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

Part II

How to do the relic calculation

1 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.

1.1 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

1.2 the velocities of the relics

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

- the observed shock velocity which is 4300 km/s for the NW relic only which from Lindner et al. should be in the CM frame.
- the simulation output for estimating the merger velocity in the CM frame but I am not sure if the standard output, e.g. relative merger velocity is what we want

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

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

1.3 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 = (3.086*10^13)

In [93]:
```
data['v_avg'] = (data['v_3d_obs'] + data['v_3d_col']) * \
                ( SE_mean_mass)/(NW_mean_mass + SE_mean_mass)/ 2.
data['v_relic'] = pd.DataFrame([rand.randn() * 800 + \
                               4300 for i in range(data.shape[0])])
```

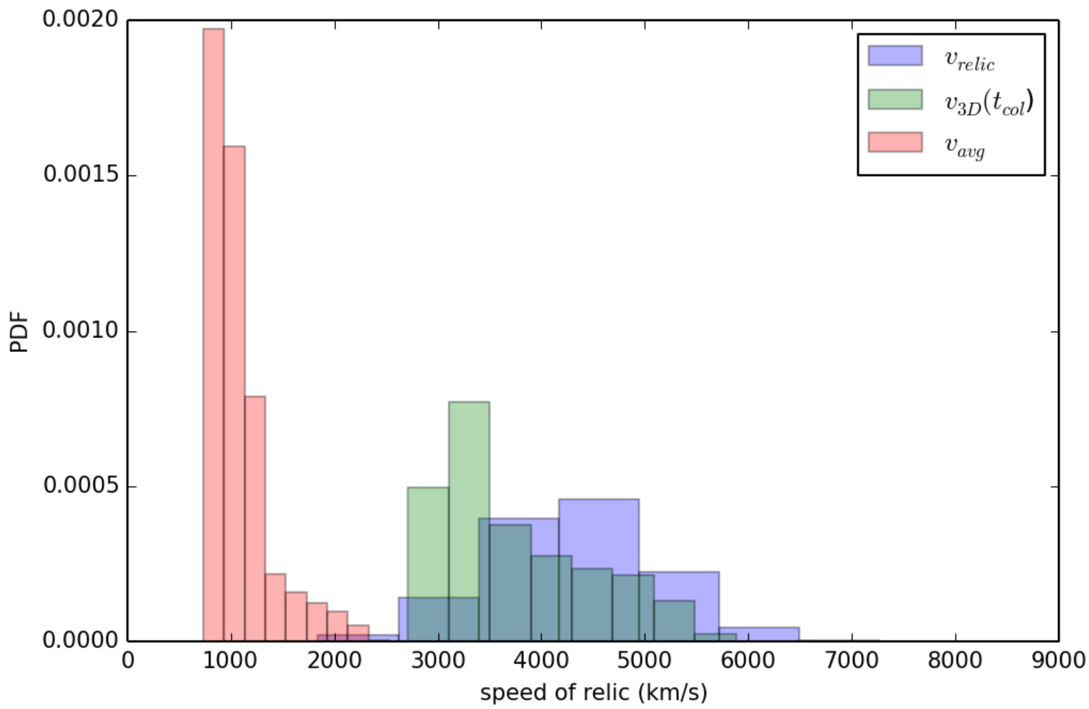
In [94]:
```
# convert m to Mpc , Myrs to seconds
unitconversion = 60 * 60 * 365 * 24 * 1e9 / (1e13 * 1e6 * 3.086)
```

In [95]:
```
j1, j2, j3 = plt.hist(data['v_relic'], histtype = 'bar',
                      label = r'$v_{relic}$', alpha=0.3, normed=True)
j1, j2, j3 = plt.hist(data['v_3d_col'], histtype = 'bar',
                      label = r'$v_{3D}(t_{col}$)', alpha=0.3, normed=True)
j1, j2, j3 = plt.hist(data['v_avg'], histtype = 'bar',
                      label = r'$v_{avg}$', alpha=0.3, normed=True)
```

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

Out [95]:

```
<matplotlib.legend.Legend at 0x10b01d250>
```



In [95]:

```
```

In [96]:

```
data['r_proj0_min'] = (data['v_avg'] * data['TSM_0'] *
                   np.cos(data['alpha'] / 180*np.pi)) * unitconversion
data['r_proj1_min'] = (data['v_avg'] * data['TSM_1'] *
                    np.cos(data['alpha'] / 180*np.pi)) * unitconversion
data = data[~np.isnan(data['r_proj0_min'])]
data = data[~np.isnan(data['r_proj1_min'])]
```

In [97]:

```
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'])]
```

In [98]:

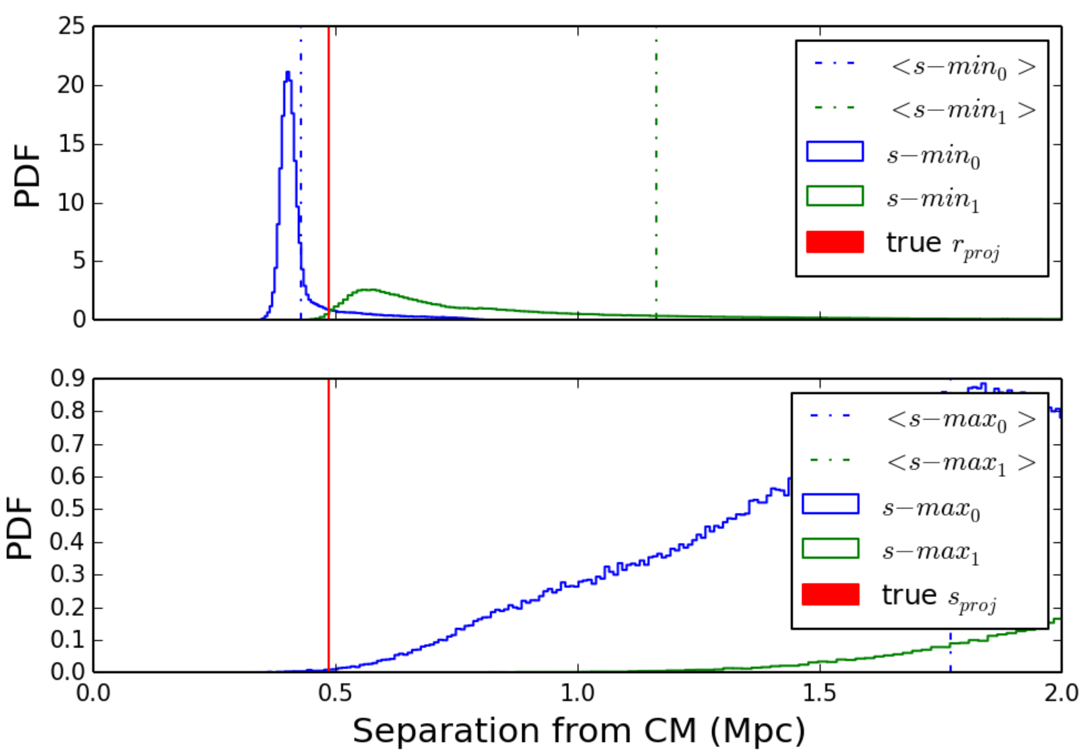
```
ax1 = plt.subplot(211)
j1, j2, j3 = ax1.hist(data['r_proj0_min'][data['r_proj0_min'] < 10],
                      histtype='step', bins=100, label=r'$s-min_0$',
                      normed=True)
j1, j2, j3 = ax1.hist(data['r_proj1_min'][data['r_proj1_min'] < 10],
                      histtype='step', label=r'$s-min_1$',
                      bins = 1000, normed = True)
ax1.axvline(data['r_proj0_min'][data['r_proj0_min'] < 10].mean(),
```

```
              color='blue', ls='-.',label=r'$<s-min_0>$')
ax1.axvline(data['r_proj1_min'][data['r_proj1_min'] < 10].mean(),
              color='green',
              ls='-.',label=r'$<s-min_1>$')
ax1.axvspan(rlower, rupper, color='red', label=r'true $r_{proj}$')
#ax1.set_xlabel('Separation from CM (Mpc)')
#plt.title(r'Lower bound of projected relic location assuming'+\
#          ' $v_{relic} = v_{avg}$')
ax1.legend(loc='upper right')
ax1.set_ylabel('PDF', size=15)
ax1.set_xlim([0, 2])
plt.setp(ax1.get_xticklabels(), visible=False)

ax2 = plt.subplot(212, sharex=ax1)
j1, j2, j3 = ax2.hist(data['r_proj0_max'][data['r_proj0_max'] < 10],
                      histtype='step', bins=500, label=r'$s-max_0$',
                      normed=True)
j1, j2, j3 = ax2.hist(data['r_proj1_max'][data['r_proj1_max'] < 10],
                      histtype='step', label=r'$s-max_1$',
                      bins = 500, normed = True)
ax2.axvline(data['r_proj0_max'][data['r_proj0_max'] < 10].mean(),
              color='blue', ls='-.',label=r'$<s-max_0>$')
ax2.axvline(data['r_proj1_max'][data['r_proj1_max'] < 10].mean(),
              color='green', ls='-.',label=r'$<s-max_1>$')
ax2.axvspan(rlower, rupper, color='red', label=r'true $s_{proj}$')

#plt.title(r'Upper bound of projected relic location assuming'+\
#          ' $v_{relic} \sim 4300 $ km s$^{-1} $')
ax2.legend(loc='upper right')
ax2.set_xlabel('Separation from CM (Mpc)', size=15)
ax2.set_ylabel('PDF', size=15)
#ax2.savefig('r_relic_max.pdf',bbox_inches='tight')
ax2.set_xlim([0, 2])

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

In [99]:
```
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 0.48431510798, 0.48431510798
```

Part III

Apply radio relic polarization prior

In [100]:
```
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
```

In [101]:
```
r_data['v_avg'] = (r_data['v_3d_col'] + r_data['v_3d_obs']) / 2.
```

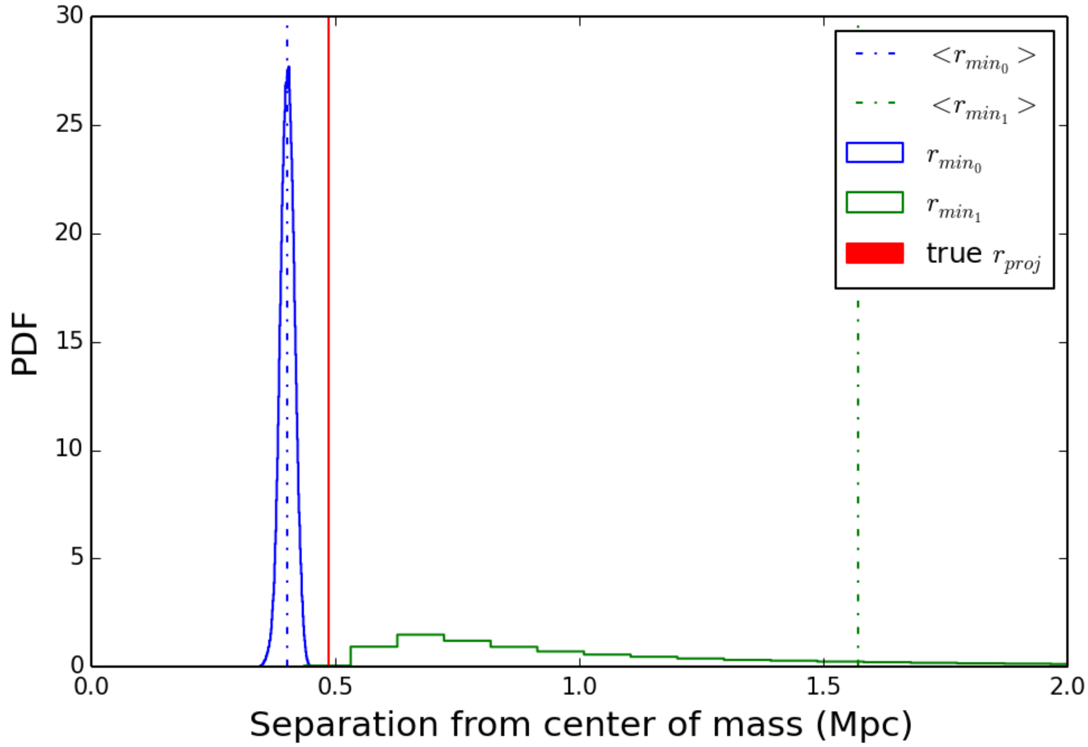
In [102]:
```
r_data['r_proj0_v2'] = (r_data['v_avg'] * r_data['TSM_0'] *
                   np.cos(r_data['alpha'] / 180*np.pi)) * unitconversion  * \
                 (SE_mean_mass)/(NW_mean_mass + SE_mean_mass)
r_data['r_proj1_v2'] = (r_data['v_avg'] * r_data['TSM_1'] *
                    np.cos(r_data['alpha'] / 180*np.pi)) * unitconversion  * \
                    (SE_mean_mass)/(NW_mean_mass + SE_mean_mass)

j1, j2, j3 = plt.hist(r_data['r_proj0_v2'][r_data['r_proj0_v2'] < 10],
                      histtype='step', bins=100, label=r'$r_{min_0 }$',
                      normed=True)
j1, j2, j3 = plt.hist(r_data['r_proj1_v2'][r_data['r_proj1_v2'] < 10],
                      histtype='step', label=r'$r_{min_1}$',
                      bins = 100, normed = True)
plt.axvspan(rlower, rupper, color='red', label=r'true $r_{proj}$')
#plt.title(r'Projected relic location assuming'+\
#          ' $v_{relic} \sim v_{3D}(t_{avg}) $ ')

plt.xlim(0, 2)
plt.ylabel('PDF', size=15)
plt.xlabel('Separation from center of mass (Mpc)', size=15)
plt.axvline(r_data['r_proj0_v2'][r_data['r_proj0_v2'] < 10].mean(),
            color='blue', ls='-.',label=r'$<r_{min_0}>$')
plt.axvline(r_data['r_proj1_v2'][r_data['r_proj1_v2'] < 10].mean(),
            color='green',
            ls='-.',label=r'$<r_{min_1}>$')
plt.legend(loc='upper right')
```

Out [102]:

```
<matplotlib.legend.Legend at 0x10b0140d0>
```



In [103]:
```
r_data['v_relic'] = pd.DataFrame([rand.randn() * 800 + \
                              4300 for i in range(r_data.shape[0])])
```

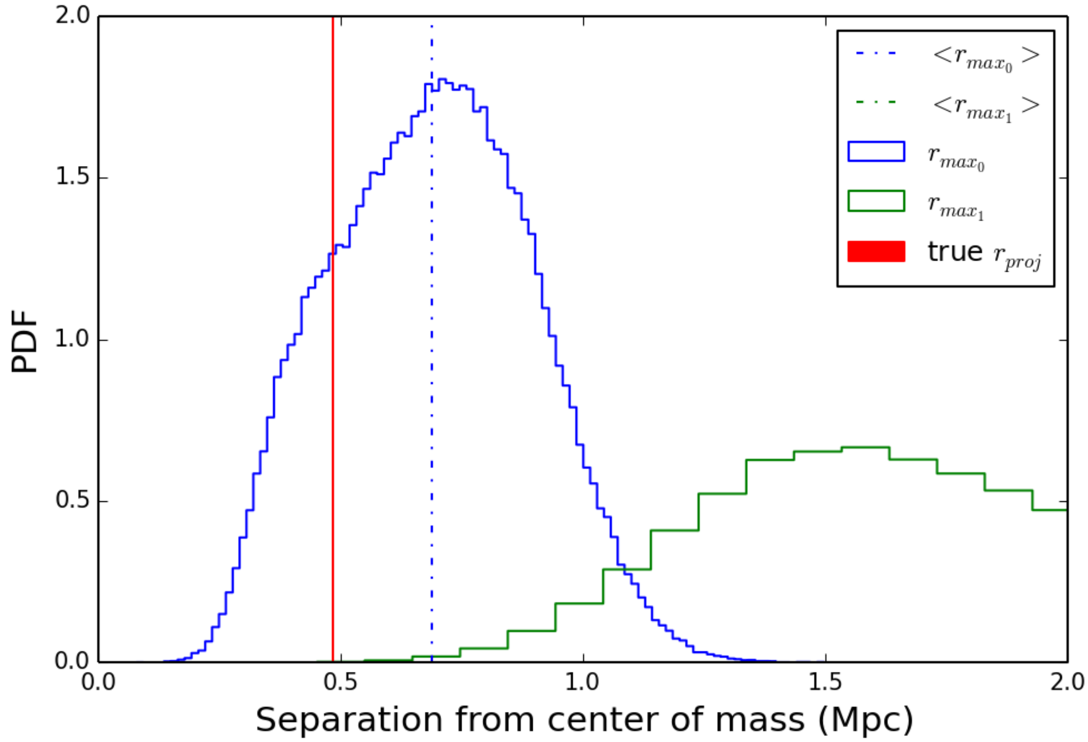
In [104]:
```
r_data['r_proj0_v2'] = (r_data['v_relic'] * r_data['TSM_0'] *
                   np.cos(r_data['alpha'] / 180*np.pi)) * unitconversion  * \
                (SE_mean_mass)/(NW_mean_mass + SE_mean_mass)
r_data['r_proj1_v2'] = (r_data['v_relic'] * r_data['TSM_1'] *
                    np.cos(r_data['alpha'] / 180*np.pi)) * unitconversion  * \
                    (SE_mean_mass)/(NW_mean_mass + SE_mean_mass)

j1, j2, j3 = plt.hist(r_data['r_proj0_v2'][r_data['r_proj0_v2'] < 10],
                      histtype='step', bins=100, label=r'$r_{max_0}$',
                      normed=True)
j1, j2, j3 = plt.hist(r_data['r_proj1_v2'][r_data['r_proj1_v2'] < 10],
                      histtype='step', label=r'$r_{max_1}$',
                      bins=100, normed=True)

plt.xlim(0, 2.0)
plt.ylabel('PDF', size=15)
plt.xlabel('Separation from center of mass (Mpc)', size=15)
plt.axvspan(rlower, rupper, color='red', label=r'true $r_{proj}$')
plt.axvline(r_data['r_proj0_v2'][r_data['r_proj0_v2'] < 10].mean(),
            color='blue', ls='-.', label=r'$<r_{max_0}>$')
plt.axvline(r_data['r_proj1_v2'][r_data['r_proj1_v2'] < 10].mean(),
            color='green', ls='-.', label=r'$<r_{max_1}>$')
plt.legend(loc='best')
```

Out [104]:

```
<matplotlib.legend.Legend at 0x10a20b790>
```



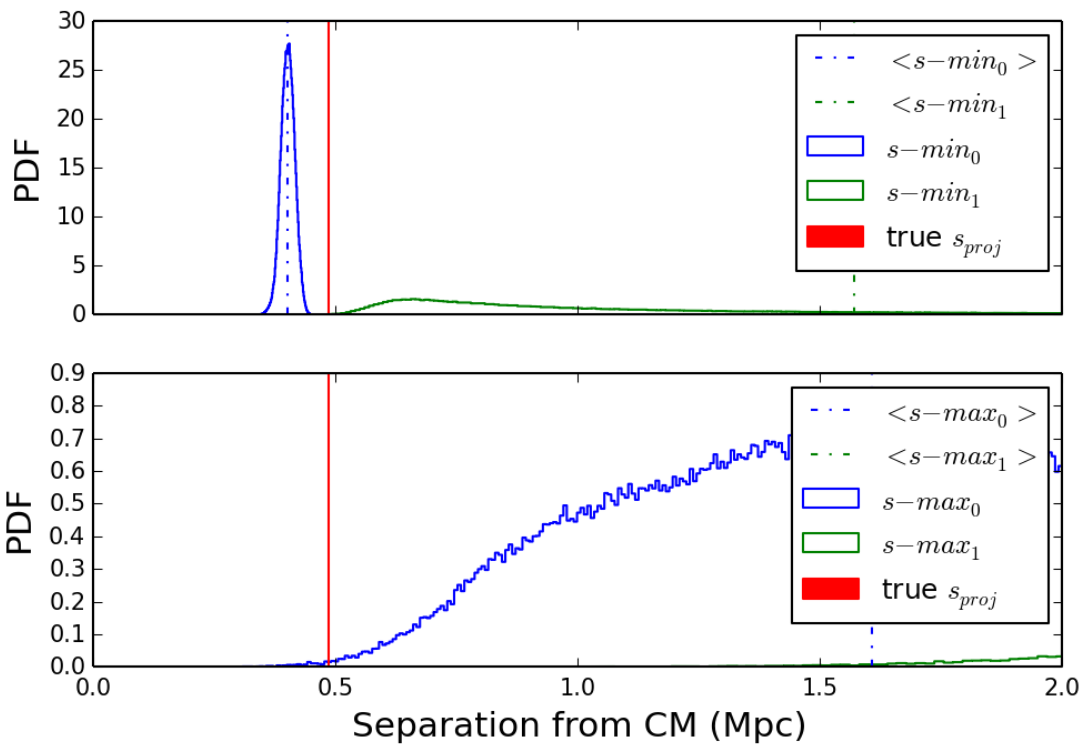
And the relic outruns the NW subcluster so the speed of the NW subcluster in the CM frame is the lower bound

In [105]:
```
ax1 = plt.subplot(211)
j1, j2, j3 = ax1.hist(r_data['r_proj0_min'][r_data['r_proj0_min'] < 10],
                      histtype='step', bins=100,
                      label=r'$s-min_0$',normed=True)
j1, j2, j3 = ax1.hist(r_data['r_proj1_min'][r_data['r_proj1_min'] < 10],
                      histtype='step', label=r'$s-min_1$',
                      bins = 1000, normed = True)
ax1.axvline(r_data['r_proj0_min'][r_data['r_proj0_min'] < 10].mean(),
            color='blue', ls='-.',label=r'$<s-min_0>$')
ax1.axvline(r_data['r_proj1_min'][r_data['r_proj1_min'] < 10].mean(),
            color='green',ls='-.',label=r'$<s-min_1}>$')
ax1.axvspan(rlower, rupper, color='red', label=r'true $s_{proj}$')
#ax1.set_xlabel('Separation from CM (Mpc)')
#plt.title(r'Lower bound of projected relic location assuming'+\
#          ' $v_{relic} = v_{avg}$')
ax1.legend(loc='upper right')
ax1.set_ylabel('PDF', size=15)
ax1.set_xlim([0, 2])
plt.setp(ax1.get_xticklabels(), visible=False)

ax2 = plt.subplot(212, sharex=ax1)
j1, j2, j3 = ax2.hist(r_data['r_proj0_max'][r_data['r_proj0_max'] < 10],
                      histtype='step', bins=500,
                      label=r'$s-max_0$', normed=True)
j1, j2, j3 = ax2.hist(r_data['r_proj1_max'][r_data['r_proj1_max'] < 10],
                      histtype='step', label=r'$s-max_1$',
                      bins = 500, normed = True)
ax2.axvline(r_data['r_proj0_max'][r_data['r_proj0_max'] < 10].mean(),
            color='blue', ls='-.',label=r'$<s-max_0>$')
ax2.axvline(r_data['r_proj1_max'][r_data['r_proj1_max'] < 10].mean(),
            color='green', ls='-.',label=r'$<s-max_1>$')
ax2.axvspan(rlower, rupper, color='red', label=r'true $s_{proj}$')
```

```
#plt.title(r'Upper bound of projected relic location assuming'+\
#           ' $v_{relic} \sim 4300 $ km s$^{-1} $')
ax2.legend(loc='upper right')
ax2.set_xlabel('Separation from CM (Mpc)', size=15)
ax2.set_ylabel('PDF', size=15)
#ax2.savefig('r_relic_max.pdf',bbox_inches='tight')
ax2.set_xlim([0, 2])

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



2 Sort the different arrays to get the extreme values as bounds

3 First from the data with default prior

In [106]:
```
data.columns
```

Out [106]:

```
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_avg', u'v_relic', u'r_proj0_min',
u'r_proj1_min', u'r_proj0_max', u'r_proj1_max'], dtype='object')
```

In [107]:
```
print "true location of relic from CM is\n" + \
" {0:.2f} Mpc".format(r_proj_NW.value)
```

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

In [108]:
```
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
0.34 Mpc < r_relic < 4.65 Mpc
```

In [109]:
```
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
0.40 Mpc < r_relic < 67305552.25 Mpc
```

In [110]:
```
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
0.34 Mpc < r_relic < 3.65 Mpc
```

In [111]:
```
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
0.44 Mpc < r_relic < 67305552.25 Mpc
```