LCA vs beta distribution

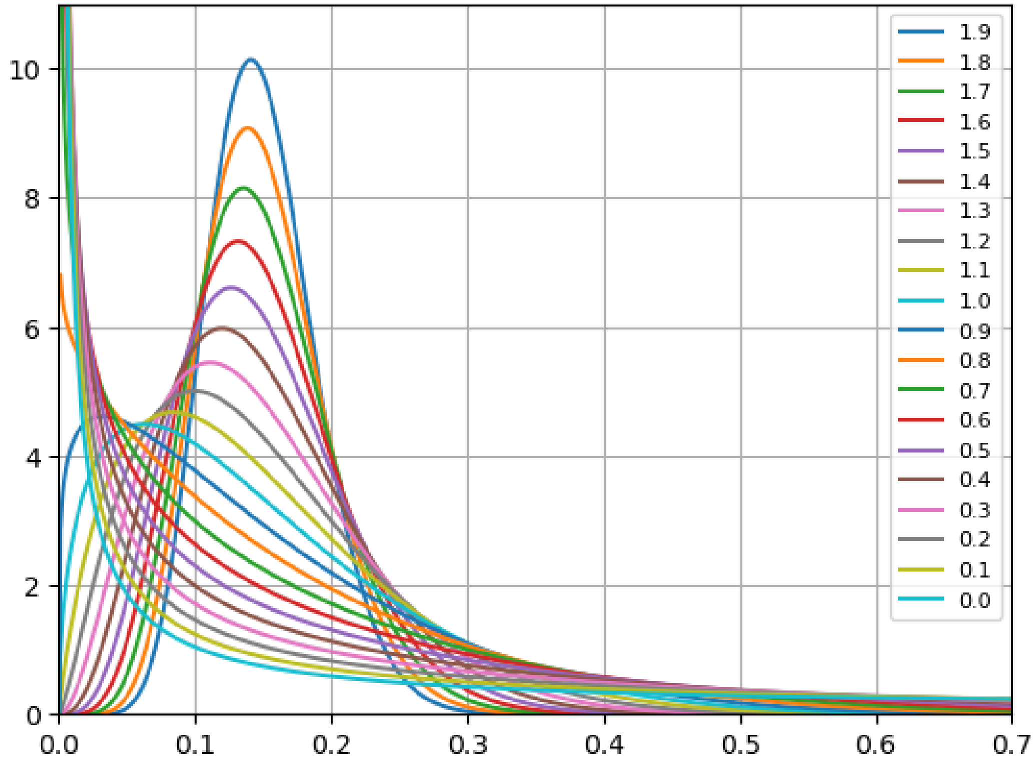
August 9, 2023

1 Fitting beta distributions to LCA data

Can I scan the parameter space?

[1]:
```
import numpy as np
import matplotlib.pyplot as pl
from astropy.table import Table
from glob import glob
import h5py

from scipy.stats import beta
from scipy.interpolate import interp1d
%matplotlib inline
```

[2]:
```
# This is what some beta distributions look like for a fixed
# mean value
fbar = 0.15
f = np.arange(0,1,0.001)
for s in np.arange(0., 2, 0.1)[::-1]:
    p = beta(fbar*(10**s),(1-fbar)*(10**s))
    pl.plot(f, p.pdf(f),label='{:.1f}'.format(s))
pl.xlim(0,0.7)
pl.ylim(0,11)
pl.legend(loc=1, prop={'size': 8})
pl.grid()
```



[3]:
```
# Read in the big table
info = Table.read('lcaSummary.fits')
# Get rid of junk
junk = [172, 465,235]
info = info[np.isin(info['orbitid'],junk,invert=True)]

info2 = Table.read('y6_tno_color.fits')    # Has dynamical info

d2 = {row['MPC']:row['CLASS'] for row in info2}
info.add_column([d2[row['MPC']] for row in info],name='Class')
d2 = {row['MPC']:row['i'] for row in info2}
info.add_column([d2[row['MPC']] for row in info],name='i')
d2 = {row['MPC']:row['a'] for row in info2}
info.add_column([d2[row['MPC']] for row in info],name='a')

print(info.columns)

# Add free inclinations where we have them
ift = Table.read('y6_ifree.fits')
d2 = {}
for row in ift:
    try:
```

```
        f = float(row['Ifree'])
    except:
        f=99.
    d2[row['MPC']]=f
ifree = []
for row in info:
    k = row['MPC']
    if k in d2:
        ifree.append(d2[k])
    else:
        ifree.append(99.)
info.add_column(ifree, name='Ifree')
```

```
<TableColumns names=('orbitid','MPC','nGood','nBad','dChisq','Hr','sigmaHr','g-
r','sigma_g-r','r-i','sigma_r-i','r-z','sigma_r-
z','LCA_16','LCA_50','LCA_84','Class','i','a')>
```

[4]:

```
# Read all files

lca = {tno:Table.read('NoY/object{:03d}.noY.hdf5'.
 ↪format(tno),path='samples')['lca'] \
      for tno in info['orbitid']}
```

[5]:

```
# Define a bunch of subsets
subset = {}
ckbo = np.logical_and(info['Class']=='Classical', info['Ifree'].data<5.)
hcut = np.logical_and(info['Hr']>6,info['Hr']<8.2)
# Get the h-restricted ckbo's
subset['CC']= np.logical_and(ckbo,hcut)

# Everything else in H range
subset['All hot'] = np.logical_and(hcut, ~ckbo)

# Just the resonants
subset['Resonant'] = np.logical_and(hcut, info['Class']=='Resonant')

# And scattering
subset['Scattering'] = np.logical_and(hcut, info['Class']=='Scattering')

# And hot classicals
use = np.logical_and(hcut, info['Class']=='Classical')
subset['HC'] = np.logical_and(use, info['Ifree'].data>=5.)
# Split into large/small halves
subset['HC_faint'] = np.logical_and(subset['HC'],info['Hr'].data>=7.4)
subset['HC_bright'] = np.logical_and(subset['HC'],info['Hr'].data<7.4)
```

```python
# And detached:
subset['Detached'] = np.logical_and(hcut, info['Class']=='Detached')

# Subdivide resonant:
tmp = np.logical_and(info['a']>39, info['a']<40)
subset['3:2'] = np.logical_and(subset['Resonant'], tmp)
subset['not3:2']=np.logical_and(subset['Resonant'], ~tmp)

tmp = np.logical_and(info['a']>47.5, info['a']<48)
subset['2:1'] = np.logical_and(subset['Resonant'], tmp)

subset['R_inner'] = np.logical_and(subset['Resonant'], info['a']<45)
subset['R_outer'] = np.logical_and(subset['Resonant'], info['a']>=45)
subset['R_low_i'] = np.logical_and(subset['Resonant'], info['i']<15)
subset['R_high_i'] = np.logical_and(subset['Resonant'], info['i']>=15)

subset['CC_low_i'] = np.logical_and(subset['CC'], info['Ifree']<2.12)
subset['CC_high_i'] = np.logical_and(subset['CC'], info['Ifree']>=2.12)
subset['HC_low_i'] = np.logical_and(subset['HC'], info['Ifree']<25)
subset['HC_high_i'] = np.logical_and(subset['HC'], info['Ifree']>=25)


subset['Big']=info['Hr']<6

for t in subset:
    print(t, np.count_nonzero(subset[t]), np.median(info['Hr'][subset[t]]))
```

```
CC 95 7.175874983897794
All hot 598 7.436893976643992
Resonant 170 7.560077027204757
Scattering 34 7.569595548908758
HC 261 7.381186643007863
HC_faint 126 7.669435347728221
HC_bright 135 7.036403192290921
Detached 133 7.392057657432252
3:2 49 7.528898033685008
not3:2 121 7.561713074909067
2:1 16 7.597450268873207
R_inner 107 7.580324957343896
R_outer 63 7.517654666277968
R_low_i 83 7.519047097837339
R_high_i 87 7.614991129246528
CC_low_i 47 7.175874983897794
CC_high_i 48 7.179356419835406
HC_low_i 129 7.379260058251811
HC_high_i 132 7.38625875767924
Big 36 5.4331575984703555
```

[6]:
```python
# How many Big are Cold?
print('Big and cold:',np.count_nonzero(np.logical_and(subset['Big'],ckbo)))
# Which classes are the Big objects from?
np.unique(info['Class'][subset['Big']],return_counts=True)
```

```
Big and cold: 2
```

[6]:
```
(<Column name='Class' dtype='str10' length=4>
  Classical
   Detached
   Resonant
 Scattering,
 array([14, 11,  6,  5]))
```

[7]:
```python
# Routines
def logpbeta(a,b,use):
    p = beta(a,b)
    out = 0.
    for k in info['orbitid'][use]:
        out += np.log(np.mean(p.pdf(lca[k])))
    return out

def maplogp(fmean,s,use):
    logp = np.zeros( (len(s),len(fmean)))
    for i,n in enumerate(10**s):
        for j,f in enumerate(fmean):
            a = n*f
            b = n-a
            logp[i,j]=logpbeta(a,b,use)
    return logp

def argmaxNd(p):
    # Find indices to maximum element of an n-dimensional array
    return np.unravel_index(p.argmax(), p.shape)

def plot2d(fmean,s,logp,txt):
    prob = np.exp(logp - np.max(logp))
    pl.imshow(prob, origin='lower', interpolation='nearest', cmap='viridis',
         extent=(np.min(fmean),np.max(fmean),
                 np.min(s),np.max(s)),aspect='auto')
    pl.text(0.17,1.3,txt, color='w')
    pl.xlabel('Mean LCA')
    pl.ylabel('Sharpness')
    pl.colorbar()
    return

def plot1d(fmean,logp,txt,ax=None, **kwargs):
```

```python
    # Make 1d plot of marginalized distribution of f, and report stats
    prob = np.sum(np.exp(logp - np.max(logp)), axis=0)
    prob /= np.sum(prob)  # Normalize to unity
    mid,lower,upper = stats1d(fmean,prob)
    print('{:11s}: {:.4f} + {:4f} - {:.4f}  {:.4f}--{:.4f}'.format(txt,
                        mid,upper-mid,mid-lower,lower,upper))
    if ax is None:
        pl.plot(fmean,prob,label=txt,**kwargs)
    else:
        ax.plot(fmean,prob,label=txt,**kwargs)
    return
```

[8]:
```python
def stats1d(x,p):
    '''Return maximum likelihood and 68% CL bounds on parameter x given
    p(x) probability distribution.  p does not need to be normalized.
    p[j] is taken to be pdf evaluated at x[j].'''
    # Calculate the dx interval between each point.  Assume first and
    # last are the same as their neighbors
    dx = np.zeros_like(x)
    x_split = 0.5*(x[:-1]+x[1:])  # midpoints between p sample points
    dx[1:-1] = x_split[1:] - x_split[:-1]
    # Duplicate end widths
    dx[0] = dx[1]
    dx[-1] = dx[-2]
    x_top = np.append(x_split, x[-1]+0.5*dx[-1])
    cdf = np.cumsum(p*dx)
    cdf /= cdf[-1]  # Normalize CDF

    # Strip places where CDF is very flat
    tmp = np.where(cdf[1:]-cdf[:-1]>1e-8)[0]
    keep = slice(max(0,tmp[0]), min(len(p),tmp[-1]+1))
    xx = x[keep]
    pp = p[keep]
    x_top = x_top[keep]
    cdf = cdf[keep]
    ### print(cdf) ###
    # Strip bounding zeros
    ii = interp1d(cdf, x_top,kind='linear',fill_value='extrapolate')
    # Search for 68% interval that is narrowest
    dp = 0.68
    dx_best = ii(dp)-ii(0.)
    ll_best = 0.
    for ll in np.arange(0.001,1-dp,0.001):
        dx = ii(ll+dp)-ii(ll)
        ###print('...',ll,ii(ll),ii(ll+dp),dx)
        if dx < dx_best:
            dx_best = dx
```

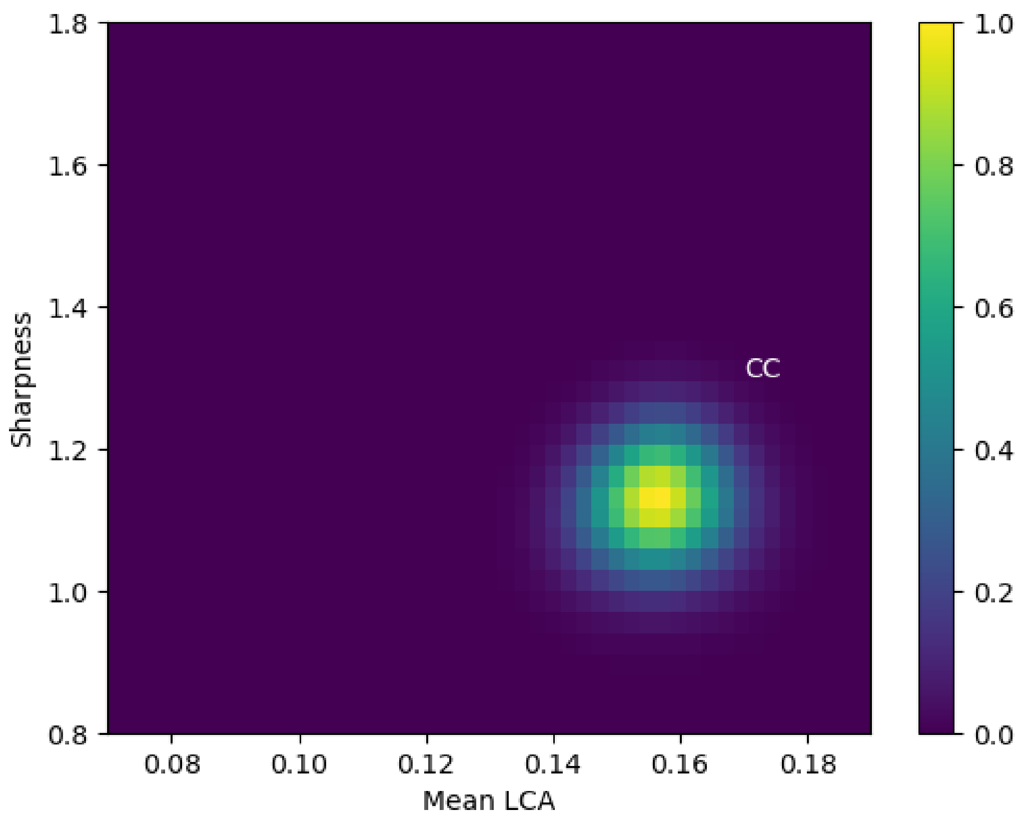
```
            ll_best = ll
    lower = ii(ll_best)
    upper = ii(ll_best+dp)
    ### print('ll_best',ll_best,lower,upper) ####
    # Find max
    i = np.argmax(pp)
    if i==0 or i==len(pp)-1:
        # max is at bound
        mid = xx[i]
    else:
        # Quadratic interp about highest value
        xfit = xx[i-1:i+2] - xx[i]
        pfit = pp[i-1:i+2]
        A = np.vstack( [np.ones_like(xfit),xfit,xfit*xfit]).T
        coeffs = np.linalg.solve(A,pfit)
        mid = xx[i] - coeffs[1]/coeffs[2]/2.
        ### print(xx,pp,mid,coeffs) ###
    return mid, lower, upper
```

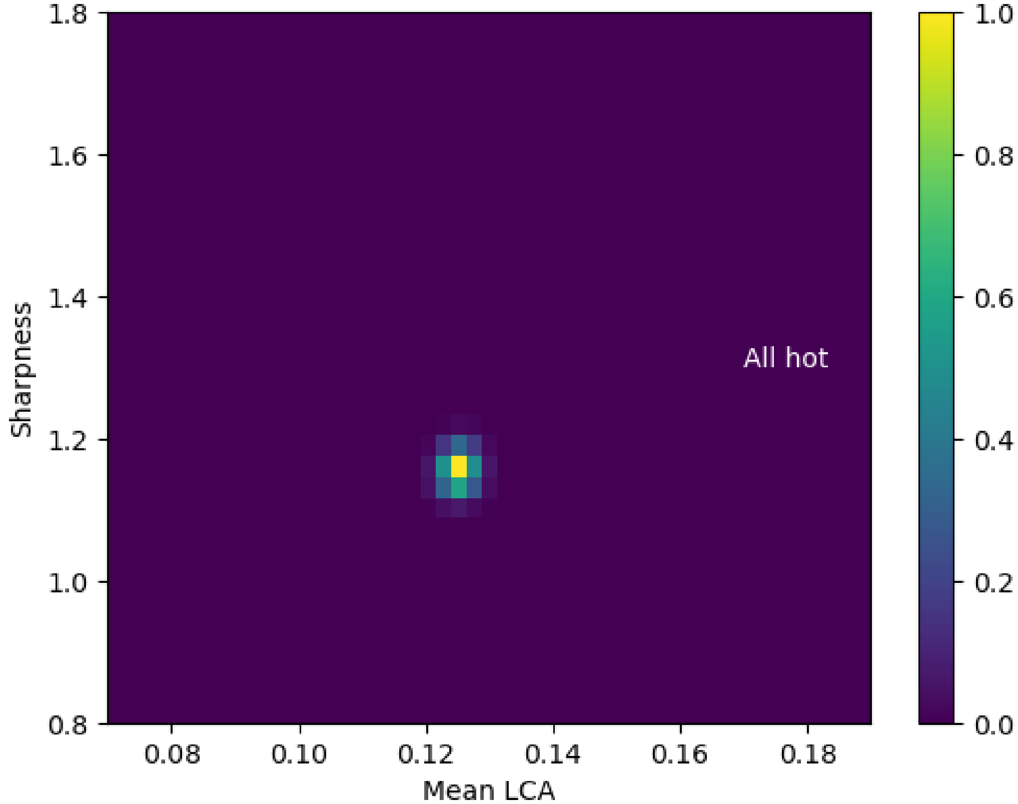
[ ]:
```
### SKIP THIS CELL IF YOU ALREADY HAVE THE logp_tno.npz FILE
### THIS IS THE SLOW PART!

# Create a standard grid of beta parameters
fmean = np.arange(0.06,0.24,0.0025)
s = np.arange(0.5,2.,0.03)

# Get logp for all params for each TNO
logp = {}
use = np.zeros(len(info),dtype=bool)
for i,tno in enumerate(info['orbitid']):
    print("doing",tno)
    use[i]=True
    logp[tno] = maplogp(fmean,s,use)
    use[i] = False

# Save all those grids
np.savez('logp_tno',**{str(k):v for k,v in logp.items()}, fmean=fmean, s=s)
```

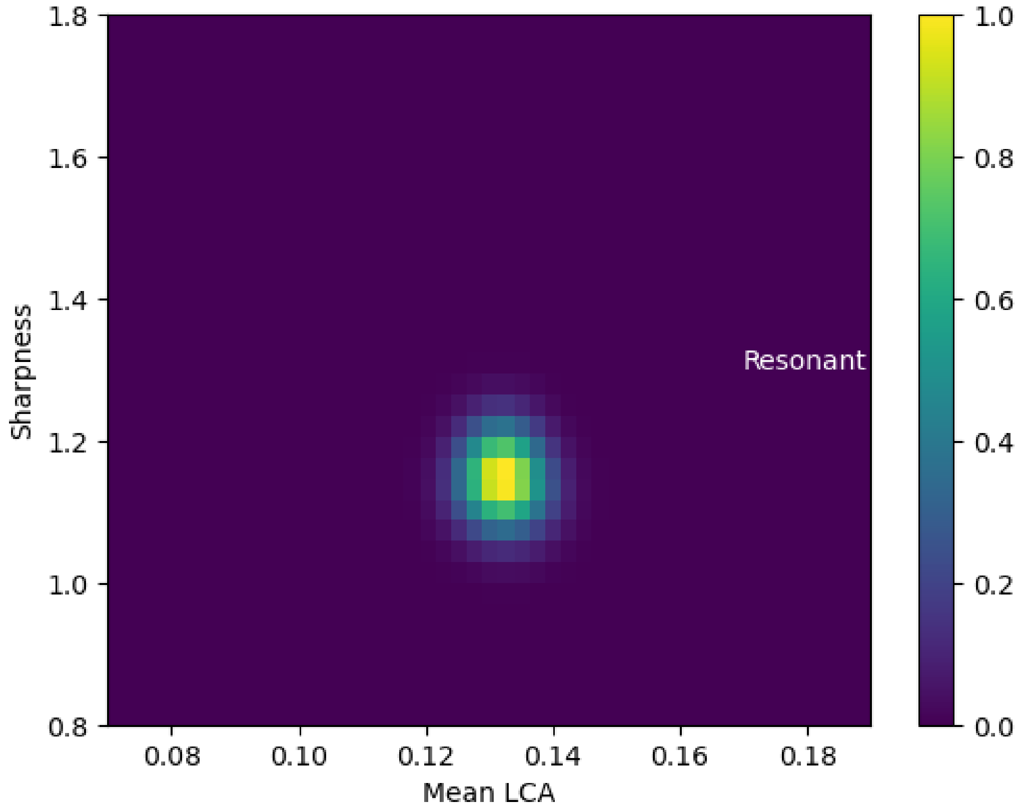
[11]:
```
### CAN SKIP THIS CELL IF YOU JUST RAN THE ONE BEFORE
### THIS IS WHERE WE RECONSTRUCT THE
# Retrieve saved grid data here
tmp = np.load('logp_tno.npz')
logp = {}
for k in tmp:
    if k=='fmean':
        fmean = tmp[k]
    elif k=='s':
```

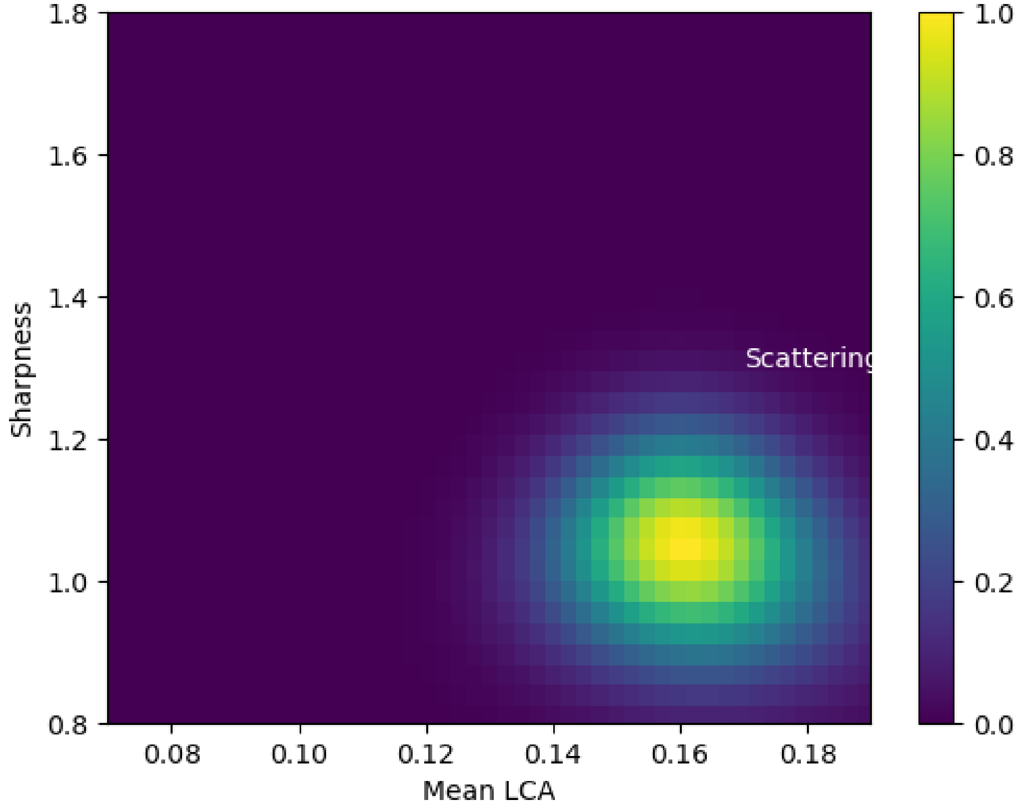
```
        s = tmp[k]
    else:
        logp[int(k)]=tmp[k]
```

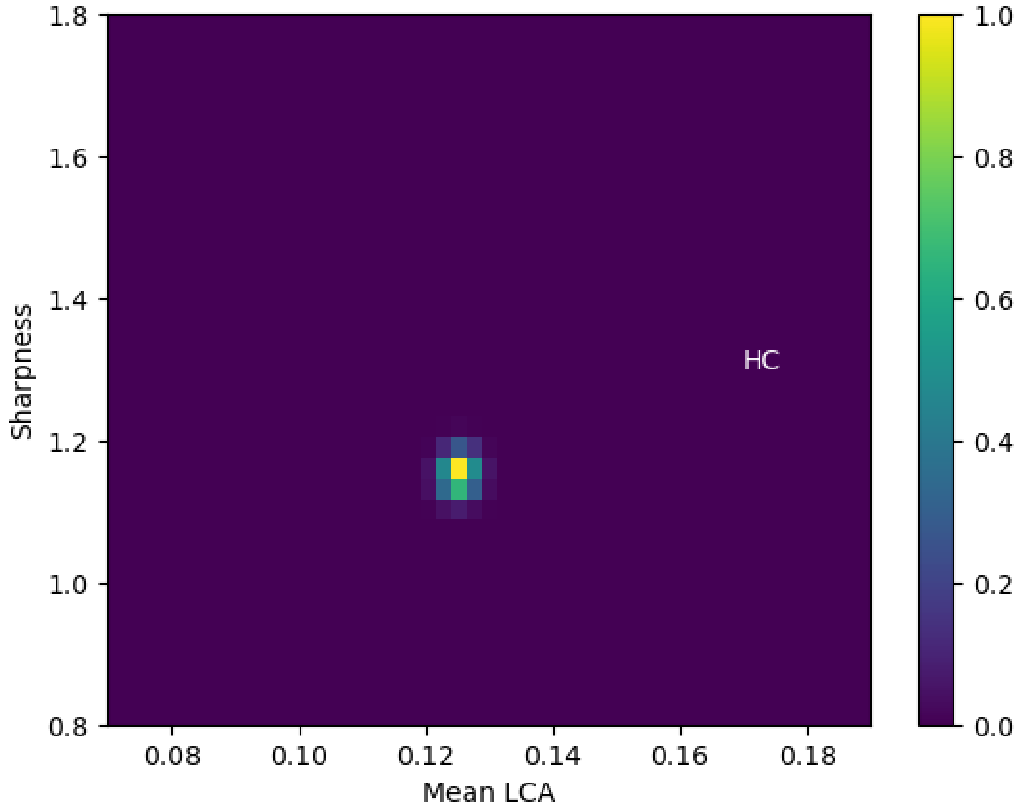
[12]:
```
# Calculate and plot likelihoods for each subset
logp_sets = {}
for k,v in subset.items():
    tnos = info['orbitid'][v]
    tmp = logp[tnos[0]]
    for t in tnos[1:]:
        tmp += logp[t]
    pl.figure()
    plot2d(fmean,s,tmp,k)
    pl.ylim(0.8,1.8)
    pl.xlim(0.07,0.19)
    logp_sets[k] = tmp.copy()

# And save them
np.savez('logp_lca',**logp_sets)
```

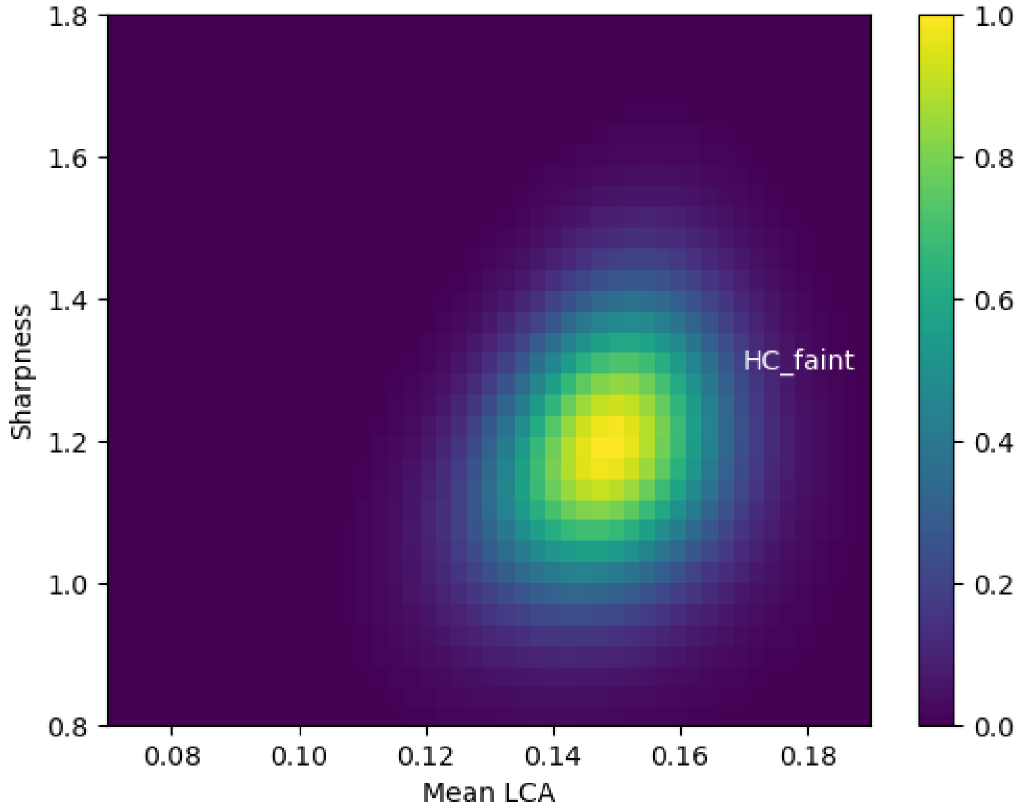


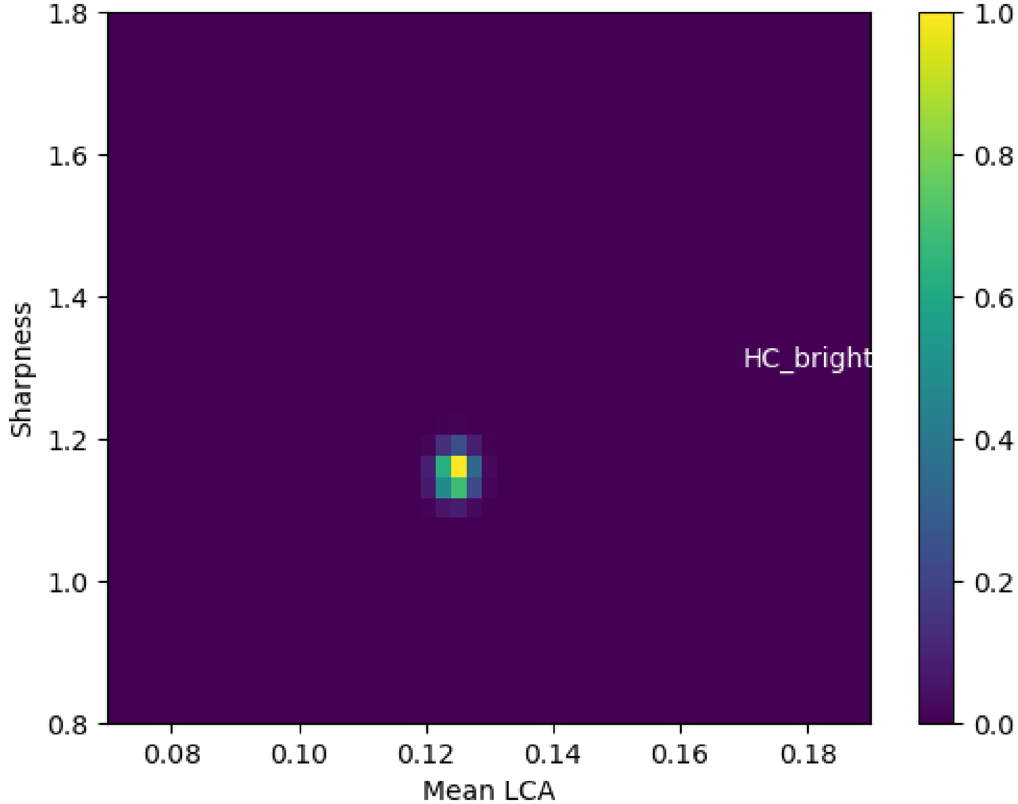

All hot
Sharpness
Mean LCA
1.8
1.6
1.4
1.2
1.0
0.8
0.08
0.10
0.12
0.14
0.16
0.18
1.0
0.8
0.6
0.4
0.2
0.0


1.8
1.6
1.4
1.2
1.0
0.8
Sharpness
Resonant
0.08
0.10
0.12
0.14
0.16
0.18
Mean LCA
1.0
0.8
0.6
0.4
0.2
0.0

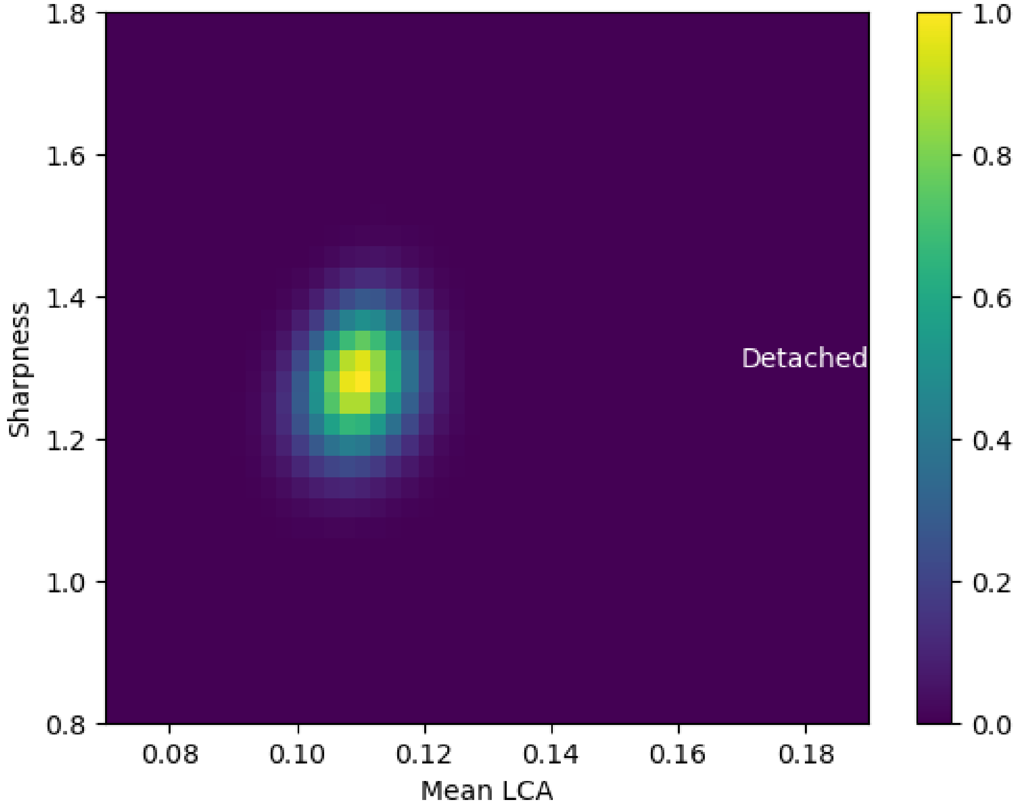

Sharpness
Mean LCA
Scattering
1.8
1.6
1.4
1.2
1.0
0.8
0.08
0.10
0.12
0.14
0.16
0.18
1.0
0.8
0.6
0.4
0.2
0.0


1.8
1.6
1.4
1.2
1.0
0.8
Sharpness
HC
0.08
0.10
0.12
0.14
0.16
0.18
Mean LCA
1.0
0.8
0.6
0.4
0.2
0.0

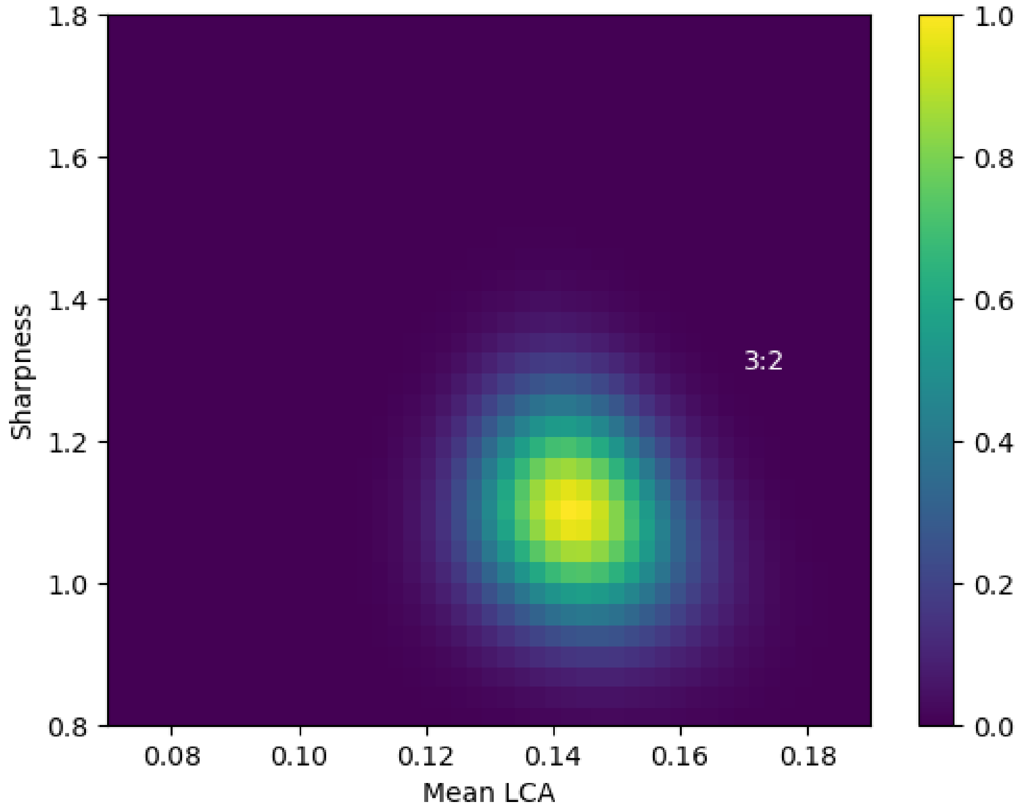

1.8
1.6
1.4
1.2
1.0
0.8
Sharpness
0.08
0.10
0.12
0.14
0.16
0.18
Mean LCA
HC_faint
1.0
0.8
0.6
0.4
0.2
0.0



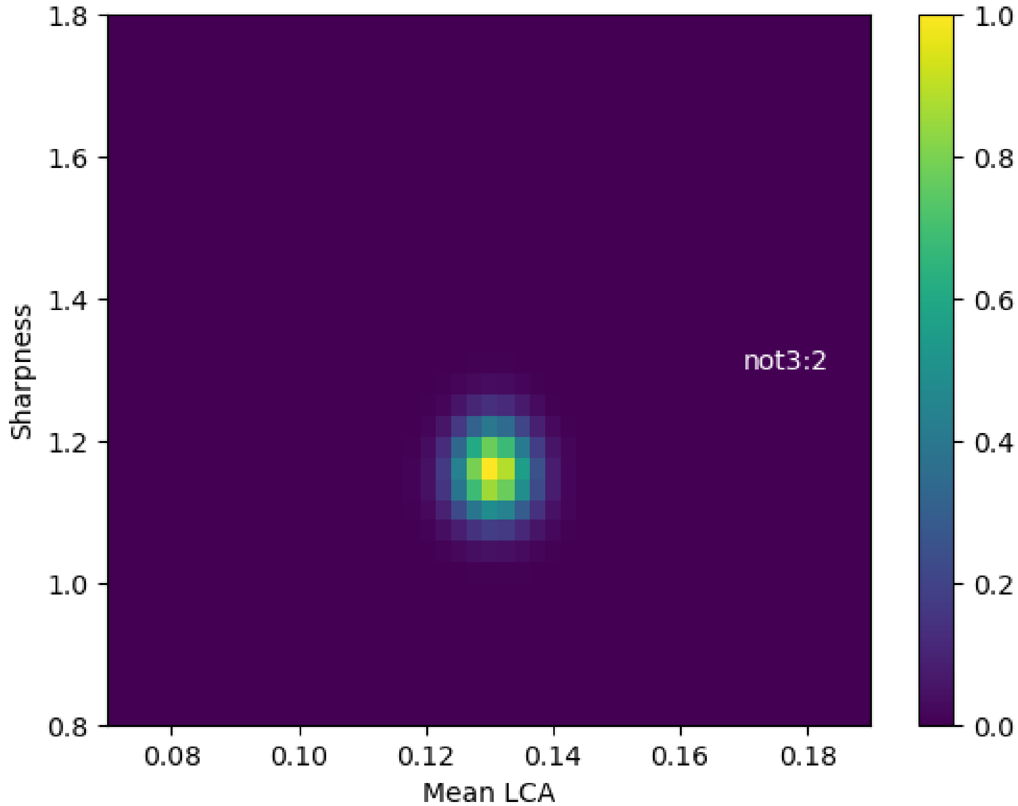
1.8
1.6
1.4
1.2
1.0
0.8
Sharpness
0.08
0.10
0.12
0.14
0.16
0.18
Mean LCA
HC_bright
1.0
0.8
0.6
0.4
0.2
0.0

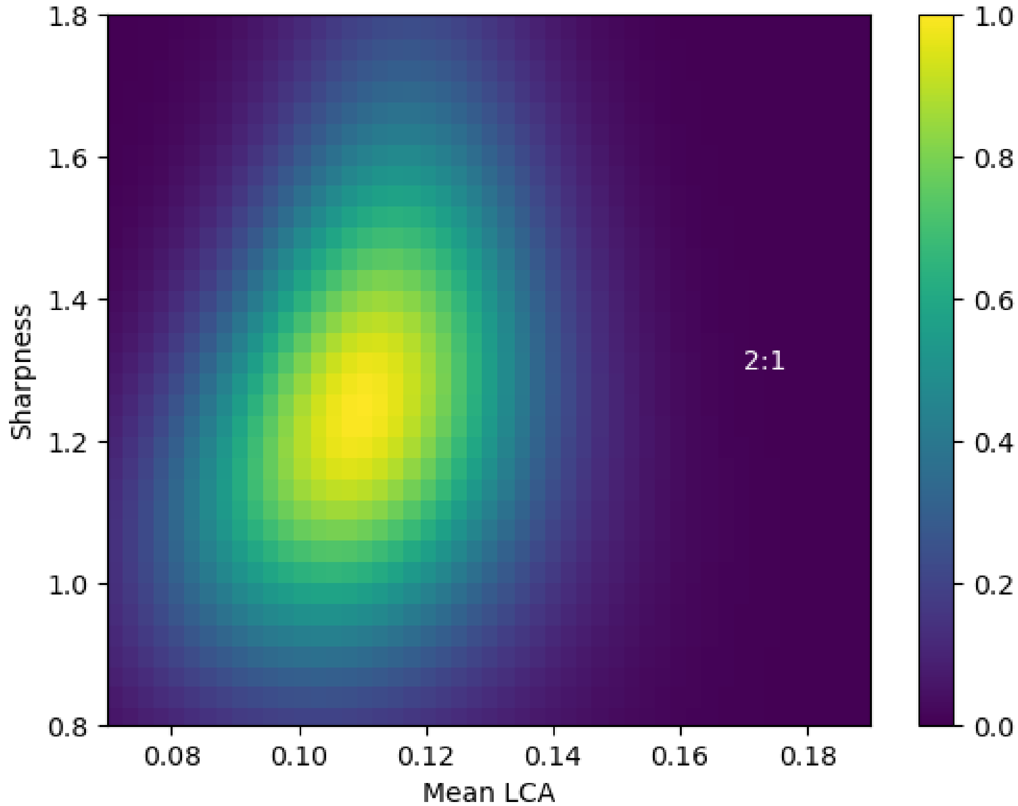


Sharpness
1.8
1.6
1.4
1.2
1.0
0.8
0.08
0.10
0.12
0.14
0.16
0.18
Mean LCA
Detached
1.0
0.8
0.6
0.4
0.2
0.0

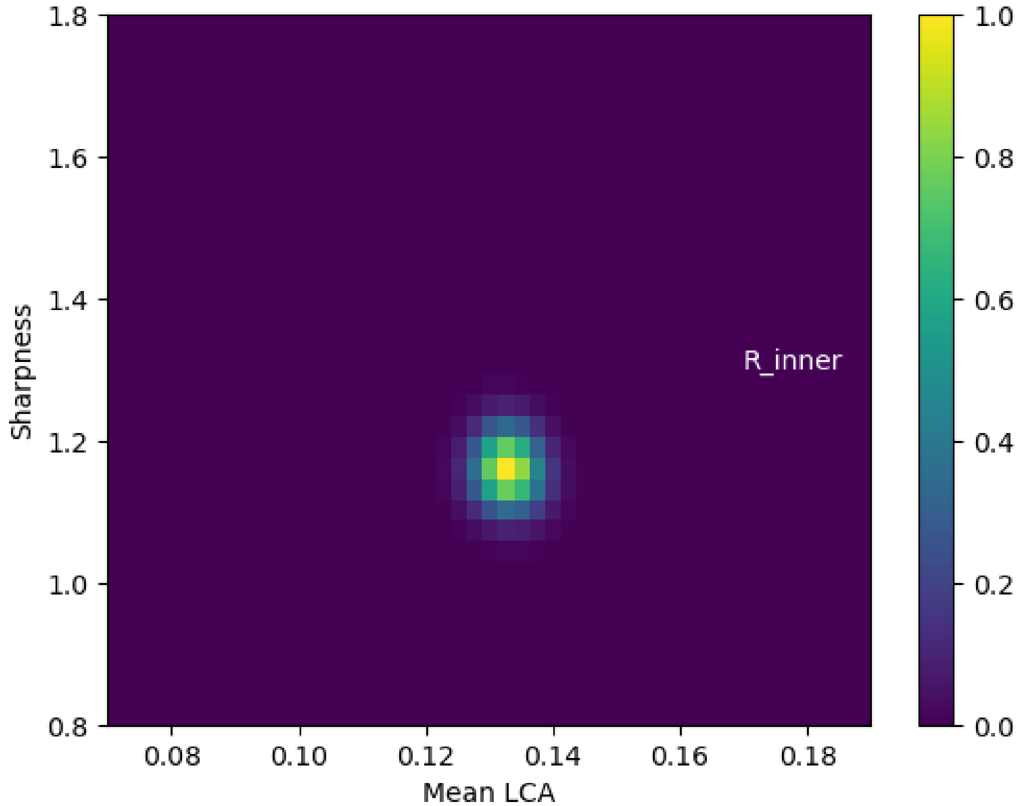


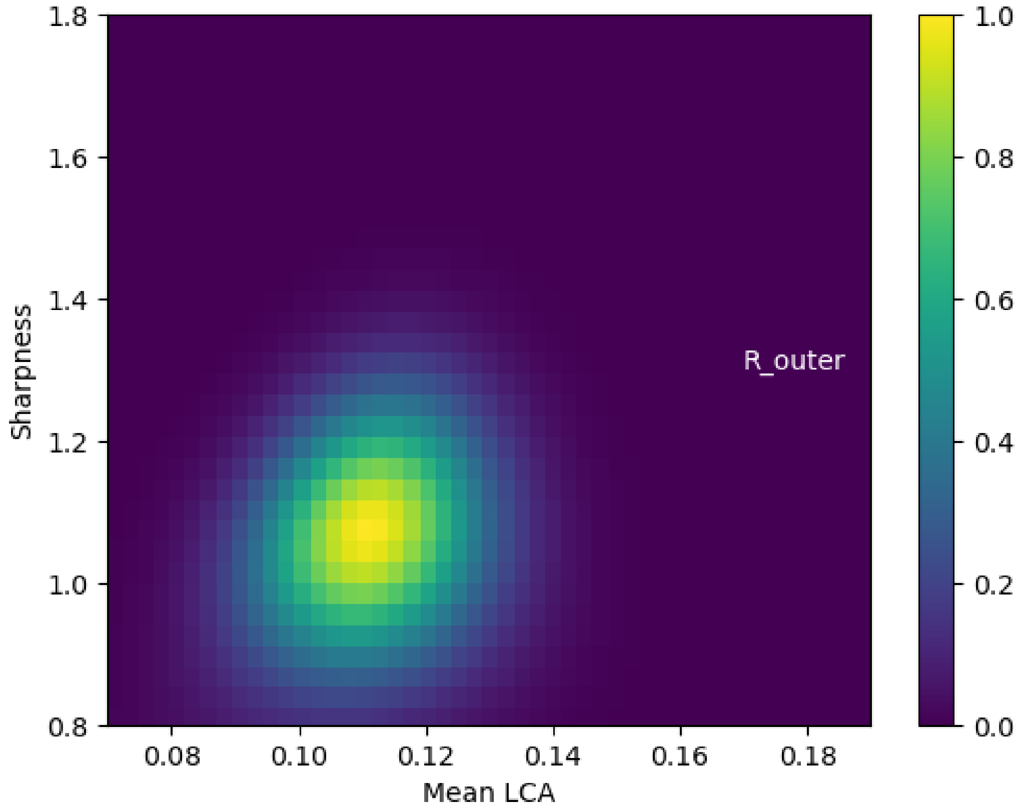
1.8
1.6
1.4
1.2
1.0
0.8
Sharpness
0.08
0.10
0.12
0.14
0.16
0.18
Mean LCA
3:2
1.0
0.8
0.6
0.4
0.2
0.0

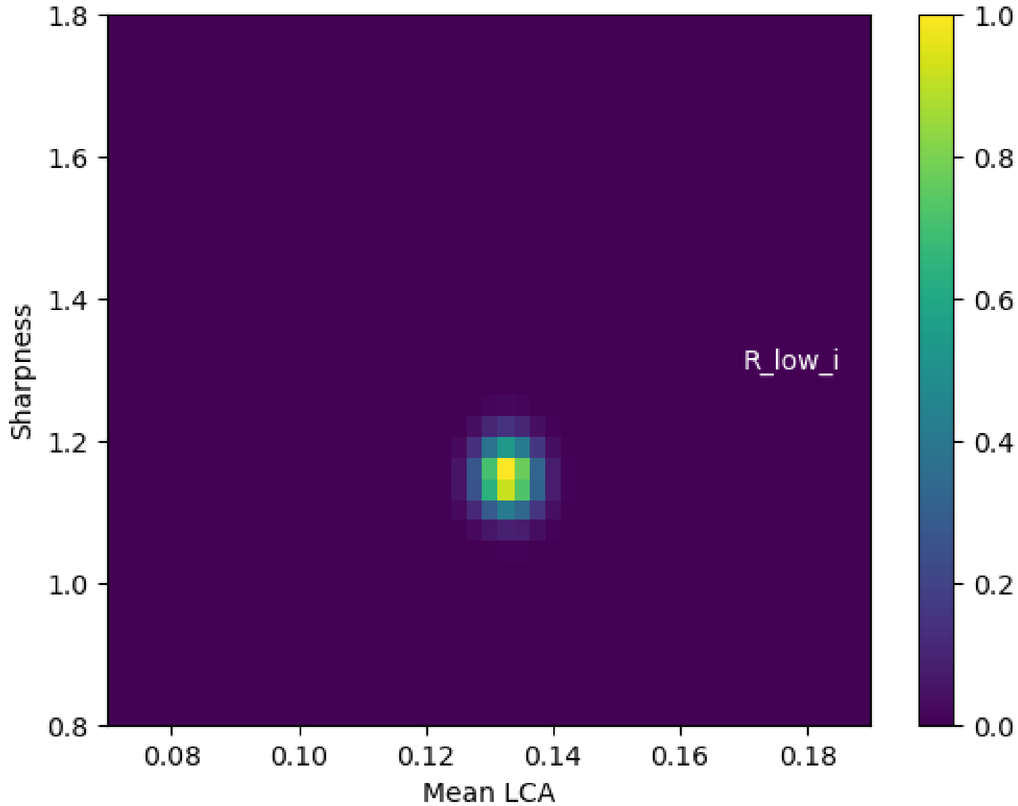

1.8
1.6
1.4
1.2
1.0
0.8
Sharpness
not3:2
0.08
0.10
0.12
0.14
0.16
0.18
Mean LCA
1.0
0.8
0.6
0.4
0.2
0.0



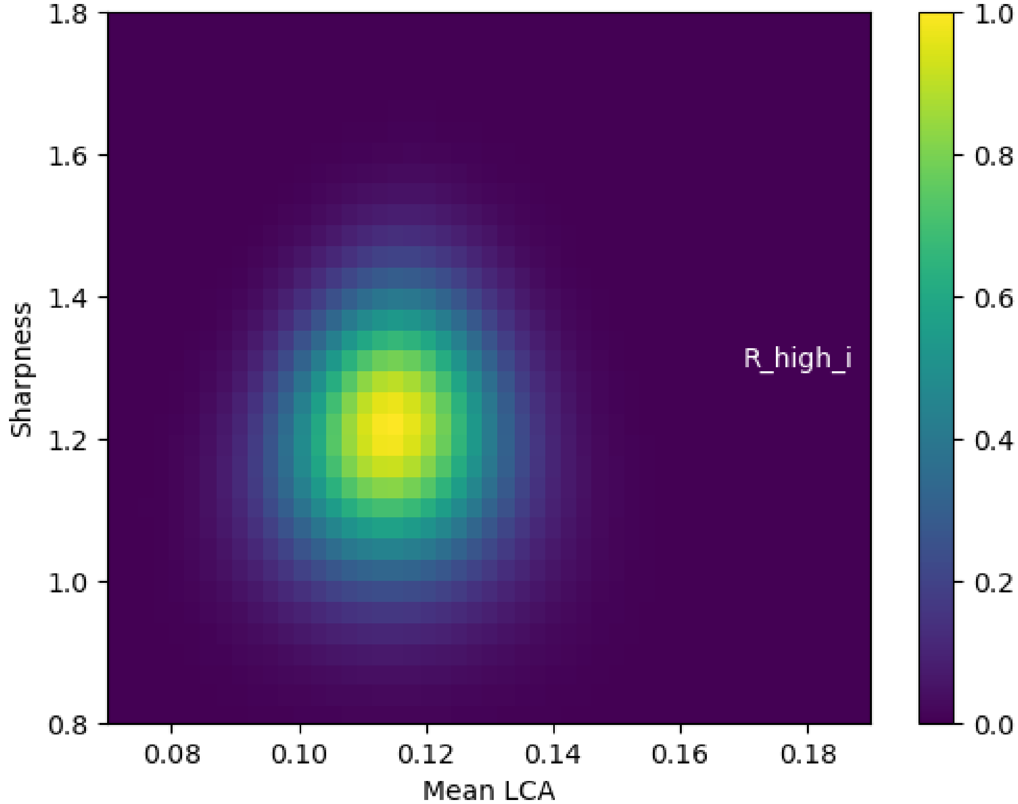
1.8
1.6
1.4
1.2
1.0
0.8
Sharpness
0.08
0.10
0.12
0.14
0.16
0.18
Mean LCA
2:1
1.0
0.8
0.6
0.4
0.2
0.0

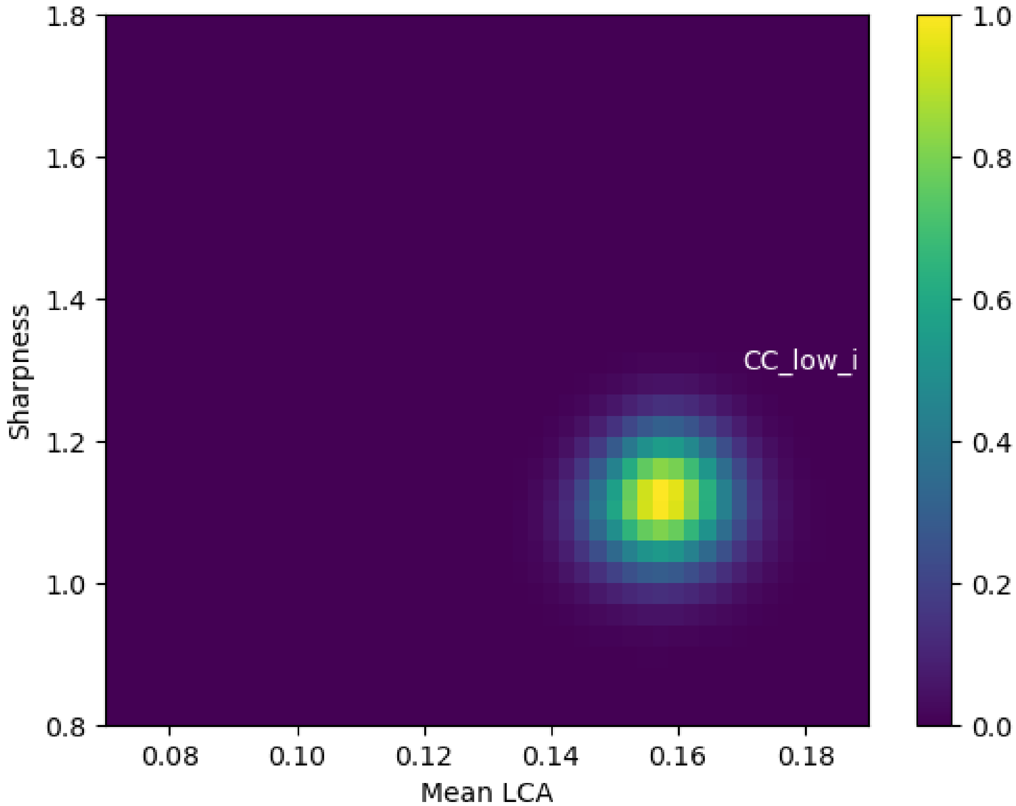

R_inner
Sharpness
Mean LCA
1.8
1.6
1.4
1.2
1.0
0.8
0.08
0.10
0.12
0.14
0.16
0.18
1.0
0.8
0.6
0.4
0.2
0.0


R_outer
Sharpness
Mean LCA
1.8
1.6
1.4
1.2
1.0
0.8
0.08
0.10
0.12
0.14
0.16
0.18
1.0
0.8
0.6
0.4
0.2
0.0

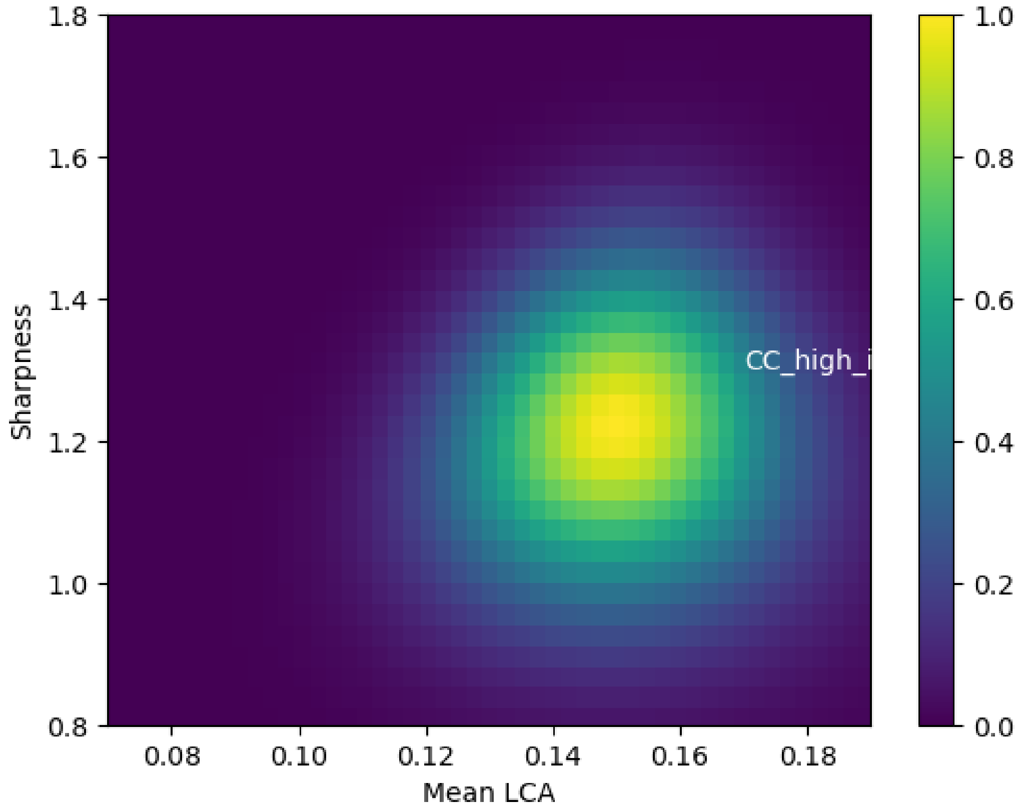


1.8
1.6
1.4
1.2
1.0
0.8
Sharpness
R_low_i
0.08
0.10
0.12
0.14
0.16
0.18
Mean LCA
1.0
0.8
0.6
0.4
0.2
0.0

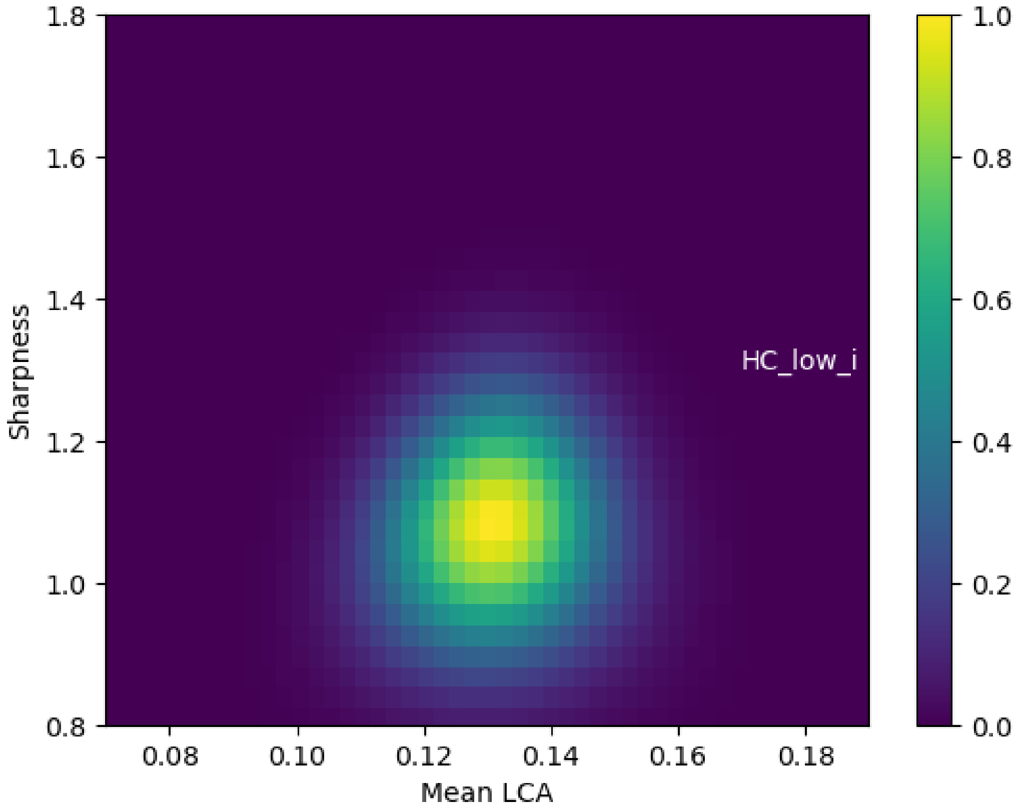

R_high_i
Sharpness
Mean LCA
1.8
1.6
1.4
1.2
1.0
0.8
0.08
0.10
0.12
0.14
0.16
0.18
1.0
0.8
0.6
0.4
0.2
0.0



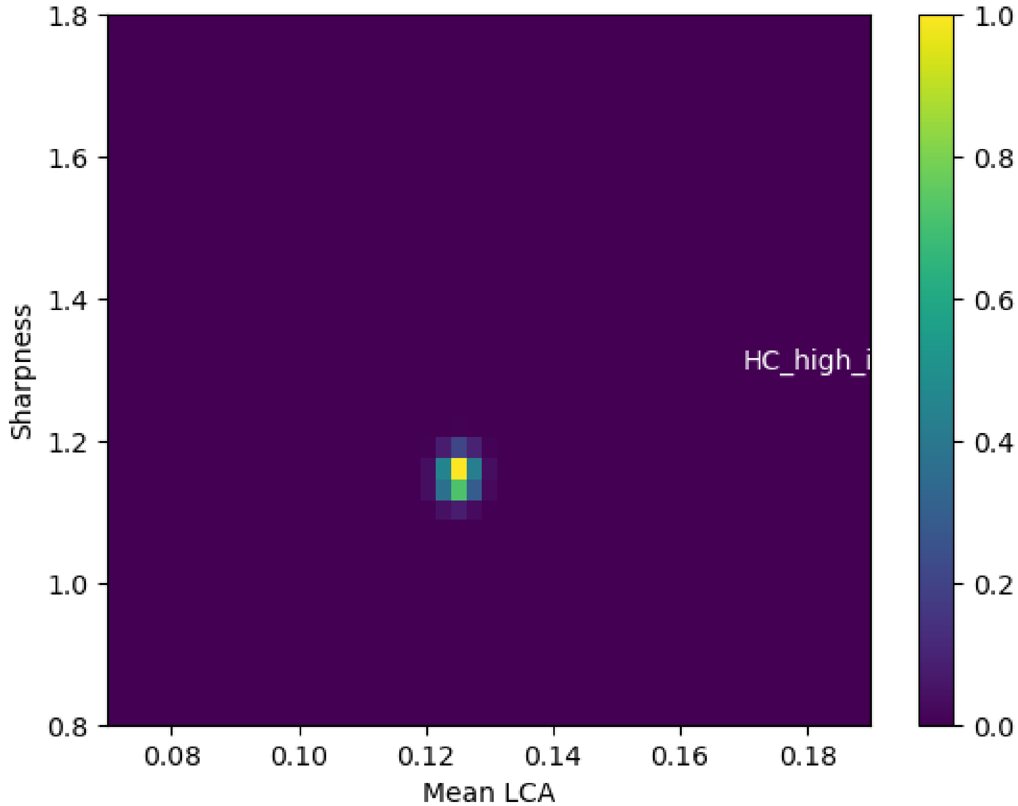
1.8
1.6
1.4
1.2
1.0
0.8
Sharpness
CC_low_i
0.08
0.10
0.12
0.14
0.16
0.18
Mean LCA
1.0
0.8
0.6
0.4
0.2
0.0

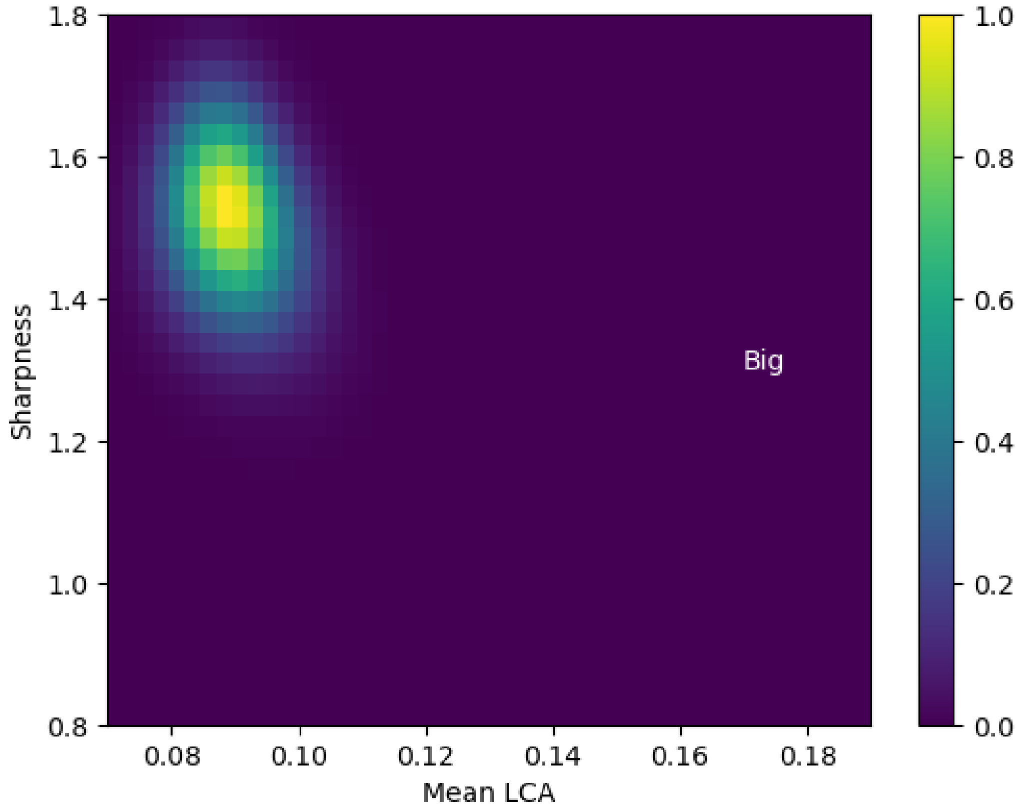


1.8
1.6
1.4
1.2
1.0
0.8
Sharpness
CC_high_i
0.08
0.10
0.12
0.14
0.16
0.18
Mean LCA
1.0
0.8
0.6
0.4
0.2
0.0



1.8
1.6
1.4
1.2
1.0
0.8
Sharpness
0.08
0.10
0.12
0.14
0.16
0.18
Mean LCA
HC_low_i
1.0
0.8
0.6
0.4
0.2
0.0

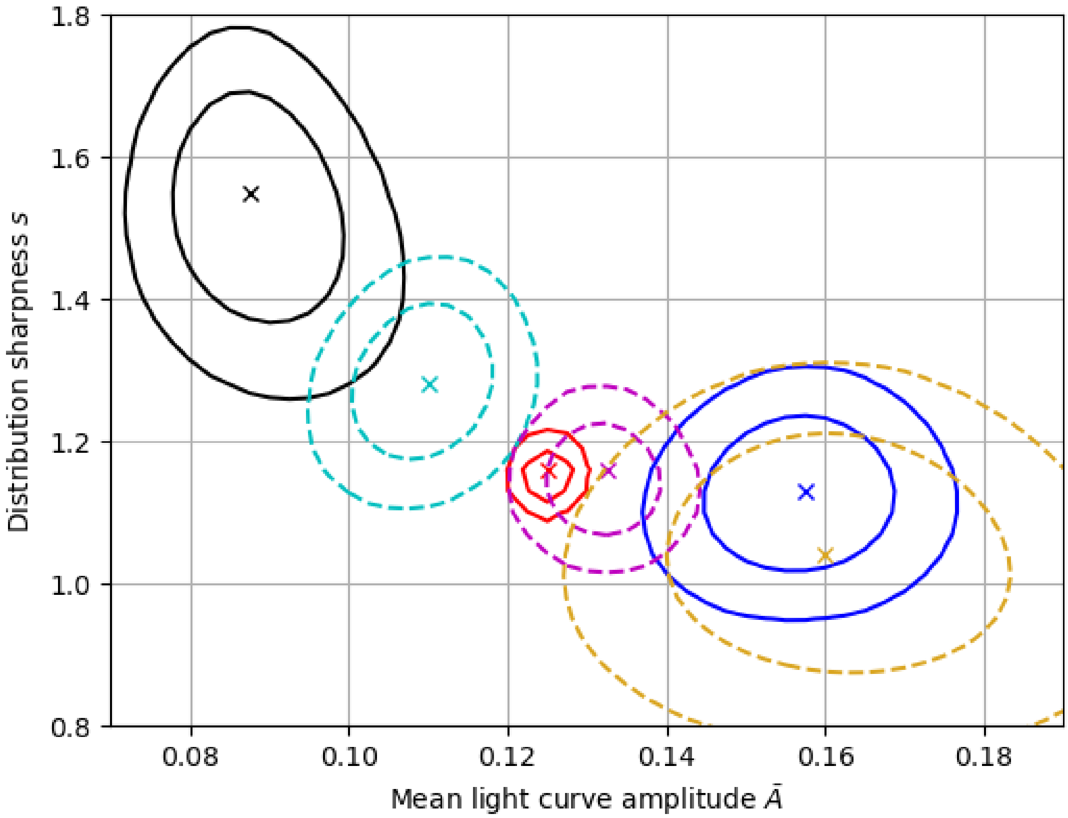

1.8
1.6
1.4
1.2
1.0
0.8
Sharpness
HC_high_i
0.08
0.10
0.12
0.14
0.16
0.18
Mean LCA
1.0
0.8
0.6
0.4
0.2
0.0



[13]:
```python
# Try making percentile contour plots
def contour(fmean,s,logp,color='k',ax=None, **kwargs):
    "Draw contours containing 68 and 95% of some probability"
    '''Using https://stackoverflow.com/questions/37890550/python-plotting-
    percentile-contour-lines-of-a-probability-distribution'''
    p = np.exp(logp-np.max(logp))
    p /= np.sum(p)
    t = np.linspace(0, np.max(p), 100)
    integral = ((p >= t[:, None, None]) * p).sum(axis=(1,2))
    f = interp1d(integral, t)
    try:
        t_contours = f(np.array([0.95,0.68]))
    except:
        t_contours = f(np.array([0.95]))

    if ax is None:
        pl.contour(fmean, s, p, t_contours, colors=color,
            **kwargs)
    else:
        ax.contour(fmean, s, p, t_contours, colors=color,
            **kwargs)
```

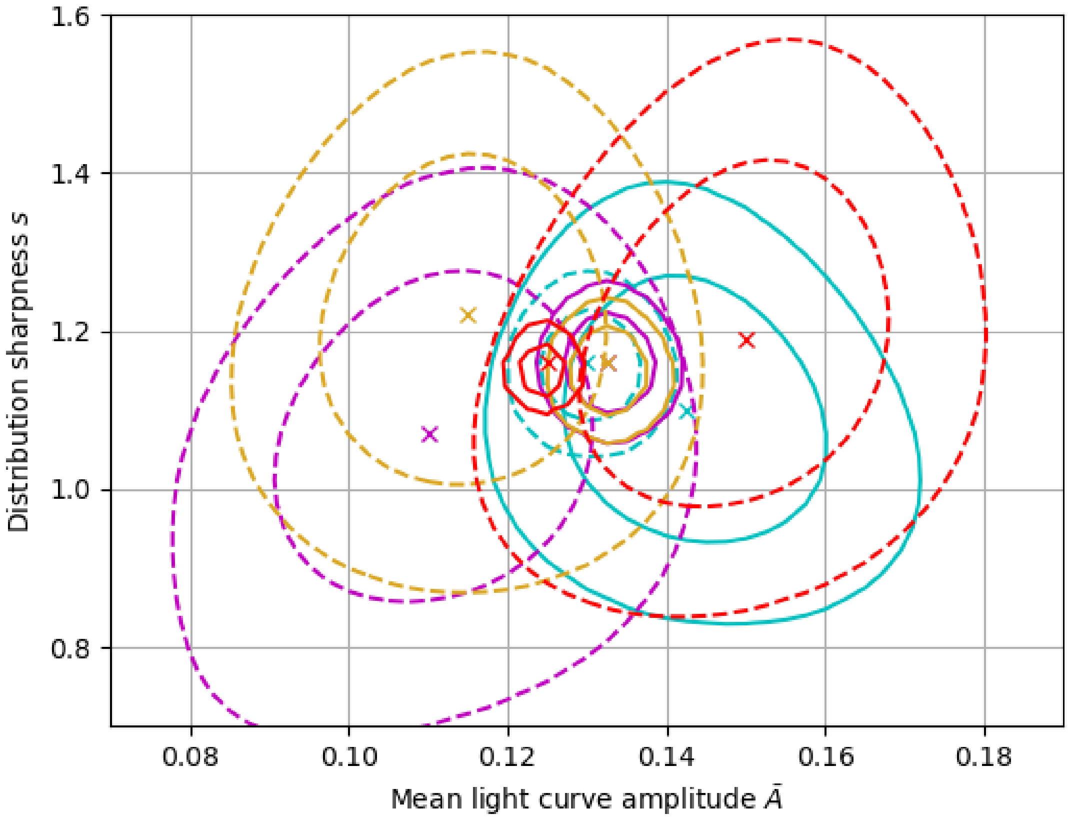
```python
    # Plot max likelihood point
    ij = np.unravel_index(p.argmax(), p.shape)
    if ax is None:
        pl.plot(fmean[ij[1]],s[ij[0]],marker='x',color=color)
    else:
        ax.plot(fmean[ij[1]],s[ij[0]],marker='x',color=color)
    return
```

[14]:
```python
plotinfo = {'CC':('b','-'),
            'HC':('r','-'),
            'Big':('k','-'),
            'Detached':('c','--'),
            'Resonant':('m','--'),
            'Scattering':('goldenrod','--'),
            '3:2':('c','-'),
            'not3:2':('c','--'),
            'R_inner':('m','-'),
            'R_outer':('m','--'),
            'R_low_i':('goldenrod','-'),
            'R_high_i':('goldenrod','--'),
            'HC_bright':('r','-'),
            'HC_faint':('r','--')}

for k in ('CC','HC','Big','Detached','Resonant','Scattering'):
    color,linestyle = plotinfo[k]
    contour(fmean,s,logp_sets[k],color=color,linestyles=linestyle)
pl.xlim(0.07,0.19)
pl.ylim(0.8,1.8)
pl.xlabel(r'Mean light curve amplitude $\bar A$')
pl.ylabel(r'Distribution sharpness $s$')
pl.grid()
```



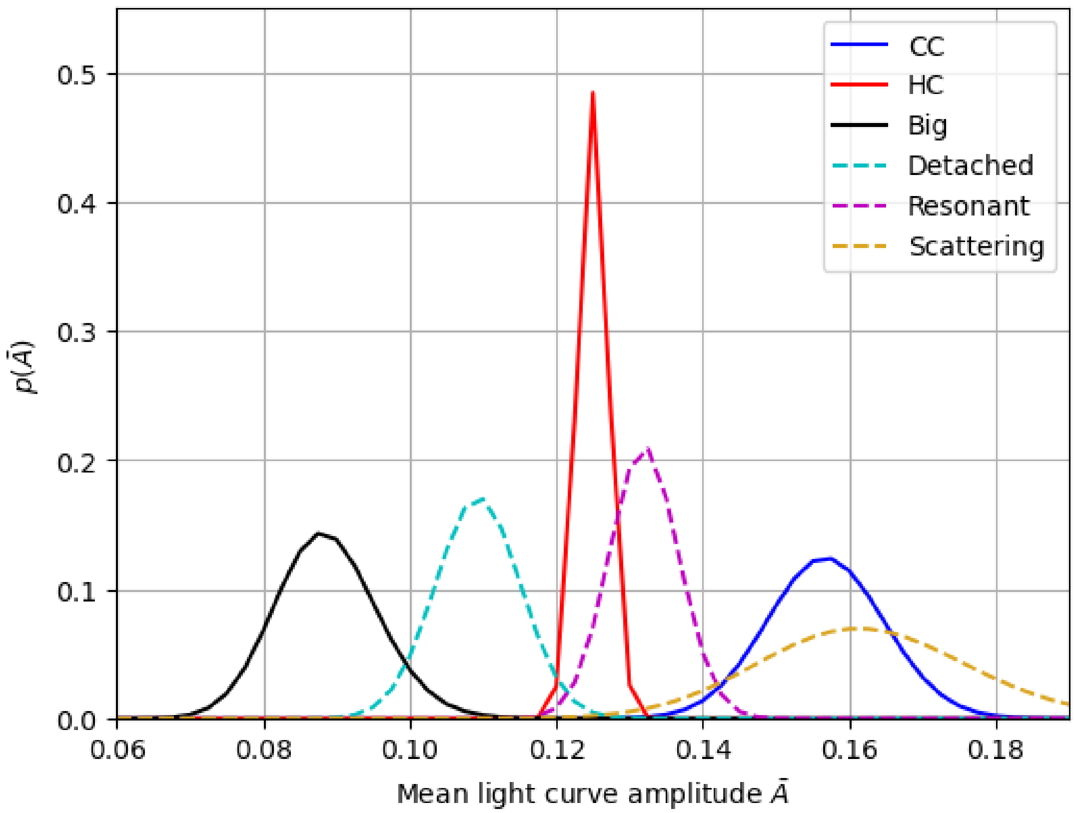
[15]:
```
for k in ('3:2','not3:2','R_inner','R_outer',
          'R_high_i','R_low_i','HC_bright','HC_faint'):
    print(k)
    color,linestyle = plotinfo[k]
    contour(fmean,s,logp_sets[k],color=color,linestyles=linestyle)
pl.xlim(0.07,0.19)
pl.ylim(0.7,1.6)
pl.xlabel(r'Mean light curve amplitude $\bar A$')
pl.ylabel(r'Distribution sharpness $s$')
pl.grid()
```

```
3:2
not3:2
R_inner
R_outer
R_high_i
R_low_i
HC_bright
HC_faint
```



[16]:
```python
for k in ('CC','HC','Big','Detached','Resonant','Scattering'):
    color,linestyle = plotinfo[k]
    plot1d(fmean,logp_sets[k],k,color=color,linestyle=linestyle)
pl.xlim(0.06,0.19)
pl.ylim(0,0.55)
pl.xlabel(r'Mean light curve amplitude $\bar A$')
pl.ylabel(r'$p(\bar A)$')
pl.legend()
pl.grid()
```

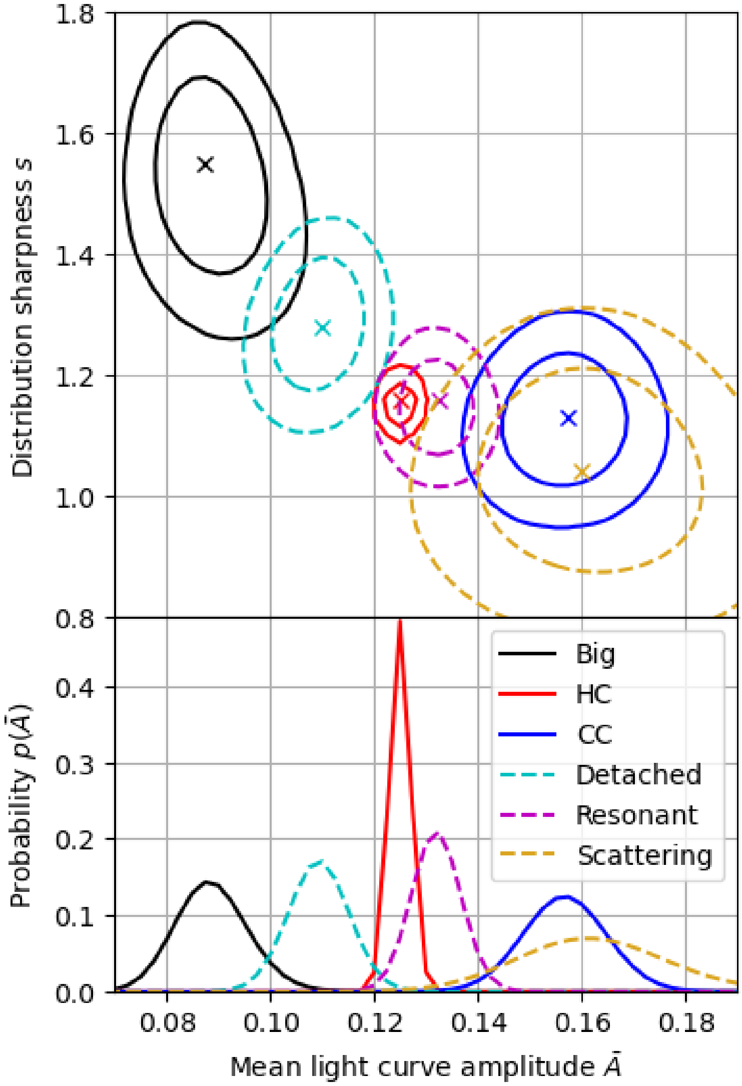
```
CC         : 0.1567 + 0.008015 - 0.0080  0.1487--0.1647
HC         : 0.1250 + 0.003343 - 0.0013  0.1237--0.1283
Big        : 0.0881 + 0.007021 - 0.0069  0.0813--0.0952
Detached   : 0.1093 + 0.006041 - 0.0056  0.1037--0.1154
Resonant   : 0.1320 + 0.004295 - 0.0052  0.1268--0.1363
Scattering : 0.1613 + 0.014956 - 0.0136  0.1476--0.1762
```



[17]:
```
# Production combined contour / projected figure
fig,ax=pl.subplots(2,1,sharex=True,height_ratios=(1,0.618), figsize=(4,6))
pl.subplots_adjust(hspace=0.0,left=0.15,right=0.96, top=0.96)
for k in ('Big','HC','CC','Detached','Resonant','Scattering'):
    color,linestyle = plotinfo[k]
    plot1d(fmean,logp_sets[k],k,ax=ax[1],color=color,linestyle=linestyle)
    contour(fmean,s,logp_sets[k],ax=ax[0],color=color,linestyles=linestyle)
ax[1].set_xlim(0.07,0.19)
ax[0].set_ylim(0.8,1.8)
ax[1].set_ylim(0.,0.49)
ax[1].set_xlabel(r'Mean light curve amplitude $\bar A$')
ax[0].set_ylabel(r'Distribution sharpness $s$')
ax[1].set_ylabel(r'Probability $p(\bar A)$')

ax[0].grid()
ax[1].grid()
ax[1].legend(loc=1)
pl.savefig('abar.pdf')
```

```
Big         : 0.0881 + 0.007021 - 0.0069   0.0813--0.0952
HC          : 0.1250 + 0.003343 - 0.0013   0.1237--0.1283
```

```
CC          : 0.1567 + 0.008015 - 0.0080  0.1487--0.1647
Detached    : 0.1093 + 0.006041 - 0.0056  0.1037--0.1154
Resonant    : 0.1320 + 0.004295 - 0.0052  0.1268--0.1363
Scattering : 0.1613 + 0.014956 - 0.0136  0.1476--0.1762
```



[18]:
```
def dchisq(logp1, logp2):
    '''Return difference in -2 log p between forcing common params and
    allowing 2 different params'''
    return 2*(np.max(logp1)+np.max(logp2) -np.max(logp1+logp2))

tmp = ('CC','HC','Big','Detached','Resonant','Scattering')
for i,k1 in enumerate(tmp):
    for k2 in tmp:
```

```
        if k1==k2:
            continue
        print('{:11s} {:11s} {:5.2f}'.
↪format(k1,k2,dchisq(logp_sets[k1],logp_sets[k2])))
```

```
CC          HC          16.38
CC          Big         33.82
CC          Detached    27.76
CC          Resonant     7.24
CC          Scattering   0.42
HC          CC          16.38
HC          Big         19.60
HC          Detached    10.07
HC          Resonant     2.16
HC          Scattering   9.06
Big         CC          33.82
Big         HC          19.60
Big         Detached     7.40
Big         Resonant    21.76
Big         Scattering  28.47
Detached    CC          27.76
Detached    HC          10.07
Detached    Big          7.40
Detached    Resonant    12.01
Detached    Scattering  18.20
Resonant    CC           7.24
Resonant    HC           2.16
Resonant    Big         21.76
Resonant    Detached    12.01
Resonant    Scattering   4.86
Scattering  CC           0.42
Scattering  HC           9.06
Scattering  Big         28.47
Scattering  Detached    18.20
Scattering  Resonant     4.86
```

[19]:
```
# Calculate evidence ratios for pairs of subsets being equal.
# Need a normalized prior over (fbar, s) to get this.  I'll assume flat prior
# over the range we're plotting
prior = np.ones_like(logp_sets['CC'])
drop = np.logical_or(fmean<0.06, fmean>0.19)
prior[:,drop] = 0.
drop = np.logical_or(s<0.7, s>1.9)
prior[drop,:] = 0.
prior /= np.sum(prior)

def logEvidenceRatio(logp1, logp2, prior):
```

```python
    '''Return difference in -2 log p between forcing common params and
    allowing 2 different params.
    The log evidence ratio will be:
    log( sum (p1 * prior) * sum(p2*prior)) - log(sum(p1*p2*prior))'''

    p1 = np.exp(logp1-np.max(logp1))
    p2 = np.exp(logp2-np.max(logp2))
    # Note that the rescalings will cancel out of the ratio
    return np.log( np.sum(p1*prior)) + np.log(np.sum(p2*prior)) \
            - np.log(np.sum(p1*p2*prior))

tmp = ('CC','HC','Big','Detached','Resonant','Scattering')
for i,k1 in enumerate(tmp):
    for k2 in tmp:
        if k1==k2:
            continue
        print('{:11s} {:11s} {:5.2f}'.format(k1,k2,
                                             logEvidenceRatio(logp_sets[k1],
                                                              logp_sets[k2],
                                                              prior)))
```

```
CC          HC           4.33
CC          Big         13.79
CC          Detached    10.63
CC          Resonant     0.22
CC          Scattering  -2.25
HC          CC           4.33
HC          Big          6.21
HC          Detached     0.99
HC          Resonant    -3.53
HC          Scattering   1.51
Big         CC          13.79
Big         HC           6.21
Big         Detached     0.54
Big         Resonant     7.58
Big         Scattering  11.62
Detached    CC          10.63
Detached    HC           0.99
Detached    Big          0.54
Detached    Resonant     2.37
Detached    Scattering   6.55
Resonant    CC           0.22
Resonant    HC          -3.53
Resonant    Big          7.58
Resonant    Detached     2.37
Resonant    Scattering  -0.18
Scattering  CC          -2.25
```

```
Scattering  HC           1.51
Scattering  Big         11.62
Scattering  Detached     6.55
Scattering  Resonant    -0.18
```

[20]:
```
# Look at Delta chisq between some splits
print('3:2/not: {:.2f}'.format(dchisq(logp_sets['3:2'],logp_sets['not3:2'])))
print('inner/outer: {:.2f}'.
  ↪format(dchisq(logp_sets['R_inner'],logp_sets['R_outer'])))
print('high/low i: {:.2f}'.
  ↪format(dchisq(logp_sets['R_low_i'],logp_sets['R_high_i'])))
print('Bright/faint HC: {:.2f}'.
  ↪format(dchisq(logp_sets['HC_bright'],logp_sets['HC_faint'])))
print('HC High/low i: {:.2f}'.
  ↪format(dchisq(logp_sets['HC_high_i'],logp_sets['HC_low_i'])))
print('CC High/low i: {:.2f}'.
  ↪format(dchisq(logp_sets['CC_high_i'],logp_sets['CC_low_i'])))
```

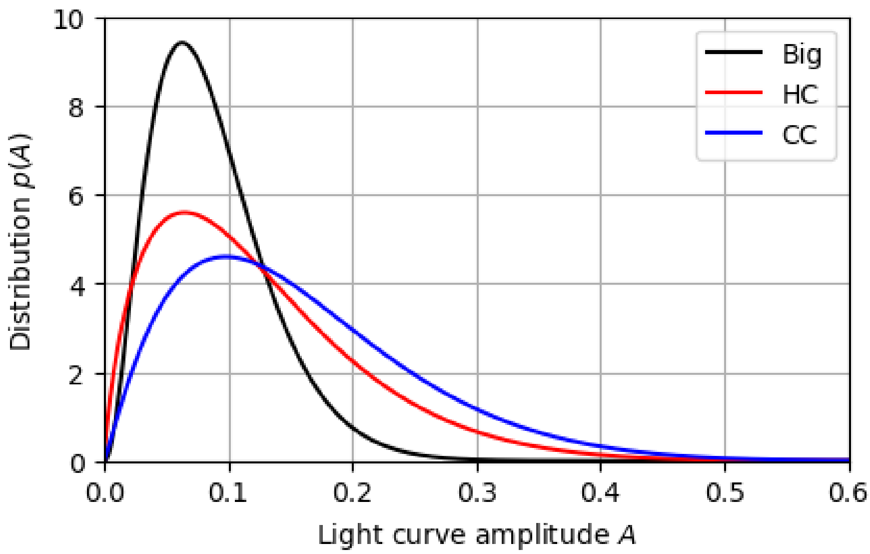
```
3:2/not: 1.48
inner/outer: 3.16
high/low i: 2.40
Bright/faint HC: 3.46
HC High/low i: 0.73
CC High/low i: 0.47
```

[21]:
```
# And look at evidence ratios between some splits
print('3:2/not: {:.2f}'.format(logEvidenceRatio(logp_sets['3:
  ↪2'],logp_sets['not3:2'],prior)))
print('inner/outer: {:.2f}'.
  ↪format(logEvidenceRatio(logp_sets['R_inner'],logp_sets['R_outer'],prior)))
print('high/low i: {:.2f}'.
  ↪format(logEvidenceRatio(logp_sets['R_low_i'],logp_sets['R_high_i'],prior)))
print('Bright/faint HC: {:.2f}'.
  ↪format(logEvidenceRatio(logp_sets['HC_bright'],logp_sets['HC_faint'],prior)))
print('HC High/low i: {:.2f}'.
  ↪format(logEvidenceRatio(logp_sets['HC_high_i'],logp_sets['HC_low_i'],prior)))
print('CC High/low i: {:.2f}'.
  ↪format(logEvidenceRatio(logp_sets['CC_high_i'],logp_sets['CC_low_i'],prior)))
```

```
3:2/not: -2.17
inner/outer: -1.06
high/low i: -1.48
Bright/faint HC: -0.85
HC High/low i: -2.48
CC High/low i: -1.70
```

[22]:
```python
# Make a plot of the distributions of LCA at best fits
# for 3 subsets
pl.figure(figsize=(5,3))
f = np.arange(0,1,0.001)
for k in 'Big','HC','CC':
    ij = argmaxNd(logp_sets[k])
    fbar = fmean[ij[1]]
    ss = s[ij[0]]
    print(k,fbar,ss)
    p = beta(fbar*(10**ss),(1-fbar)*(10**ss))
    color,linestyle = plotinfo[k]
    pl.plot(f, p.pdf(f), color=color, linestyle=linestyle, label=k)
pl.xlim(0,0.6)
pl.ylim(0,10)
pl.xlabel(r'Light curve amplitude $A$')
pl.ylabel(r'Distribution $p(A)$')
pl.legend()
pl.grid()
pl.savefig('bestfit_q.pdf')
```

```
Big 0.08750000000000002 1.550000000000001
HC 0.12500000000000006 1.1600000000000006
CC 0.15750000000000008 1.1300000000000006
```



[24]:
```python
# Find minimal 90% credible region for each TNO's LCA under flat prior
def findcl(vals, cl=0.9):
    tmp = np.sort(vals)
    nrange = int(np.floor(cl*len(tmp)))
```

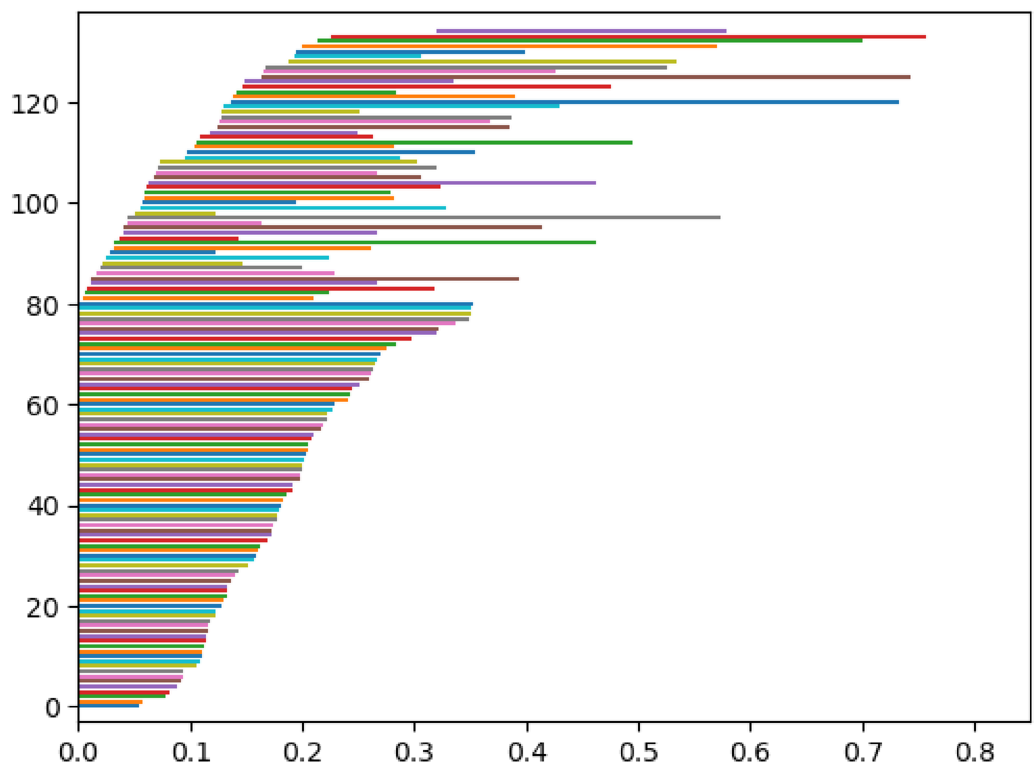
```
    width = tmp[nrange:] - tmp[:-nrange]
    i0 = np.argmin(width)
    return tmp[i0],tmp[i0+nrange]
```

[25]:
```
lca_90pct = {i:findcl(lca[i]) for i in info['orbitid']}
```

[26]:
```
# Make bar chart for variability levels of the brighter half of HC
count = 0
hl = np.array([lca_90pct[i] for i in info['orbitid'][subset['HC_bright']]])
hl[:,0] = np.where(hl[:,0]<0.005, 0, hl[:,0])
print(hl.shape)
indices = np.lexsort((hl[:,1],hl[:,0]))
hl = hl[indices]
for i in range(hl.shape[0]):
    pl.plot( hl[i], [count,count])
    count += 1
pl.xlim(0,0.85)
pl.ylim(-3,hl.shape[0]+3)
```

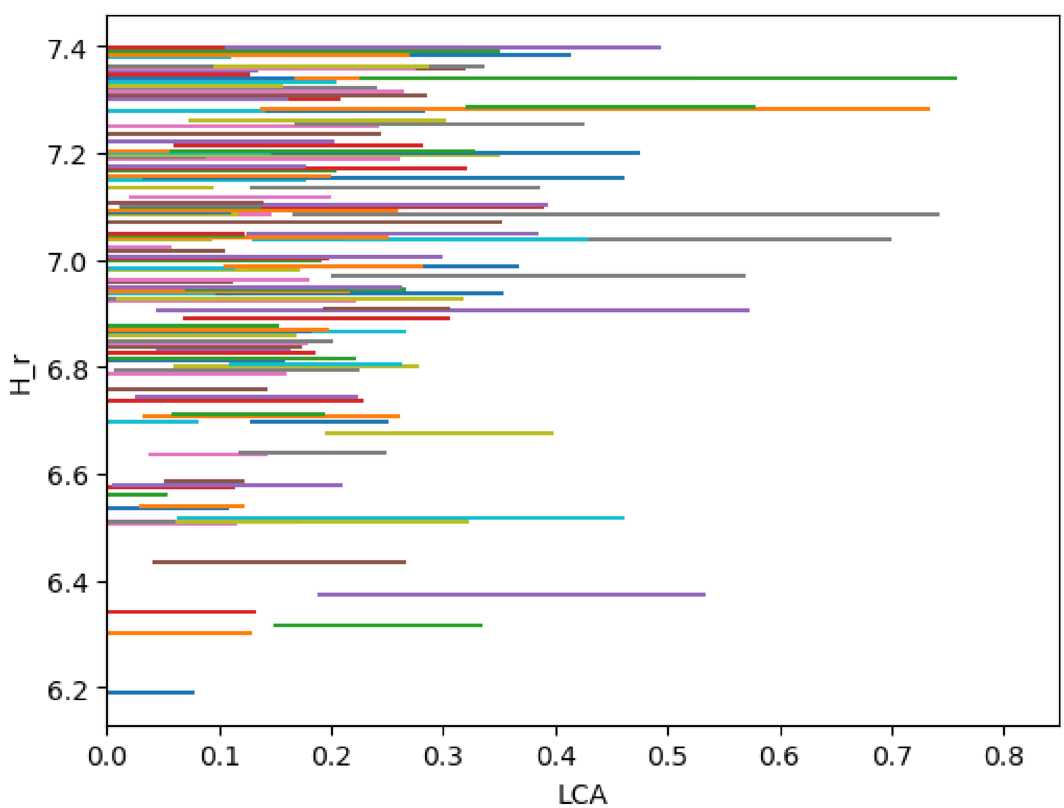
```
(135, 2)
```

[26]:
```
(-3.0, 138.0)
```



[27]:
```
# This time ordered by H
count = 0
#k = 'Big'
k = 'HC_bright'
hl = np.array([lca_90pct[i] for i in info['orbitid'][subset[k]]])
indices = np.argsort(info['Hr'][subset[k]])
hl = hl[indices]
Hr = info['Hr'][subset[k]][indices]
for i in range(hl.shape[0]):
    pl.plot( hl[i], [Hr[i],Hr[i]])
    count += 1
pl.xlim(0,0.85)
#pl.ylim(3,6.1)
pl.xlabel('LCA')
pl.ylabel('H_r')
```

[27]: Text(0, 0.5, 'H_r')

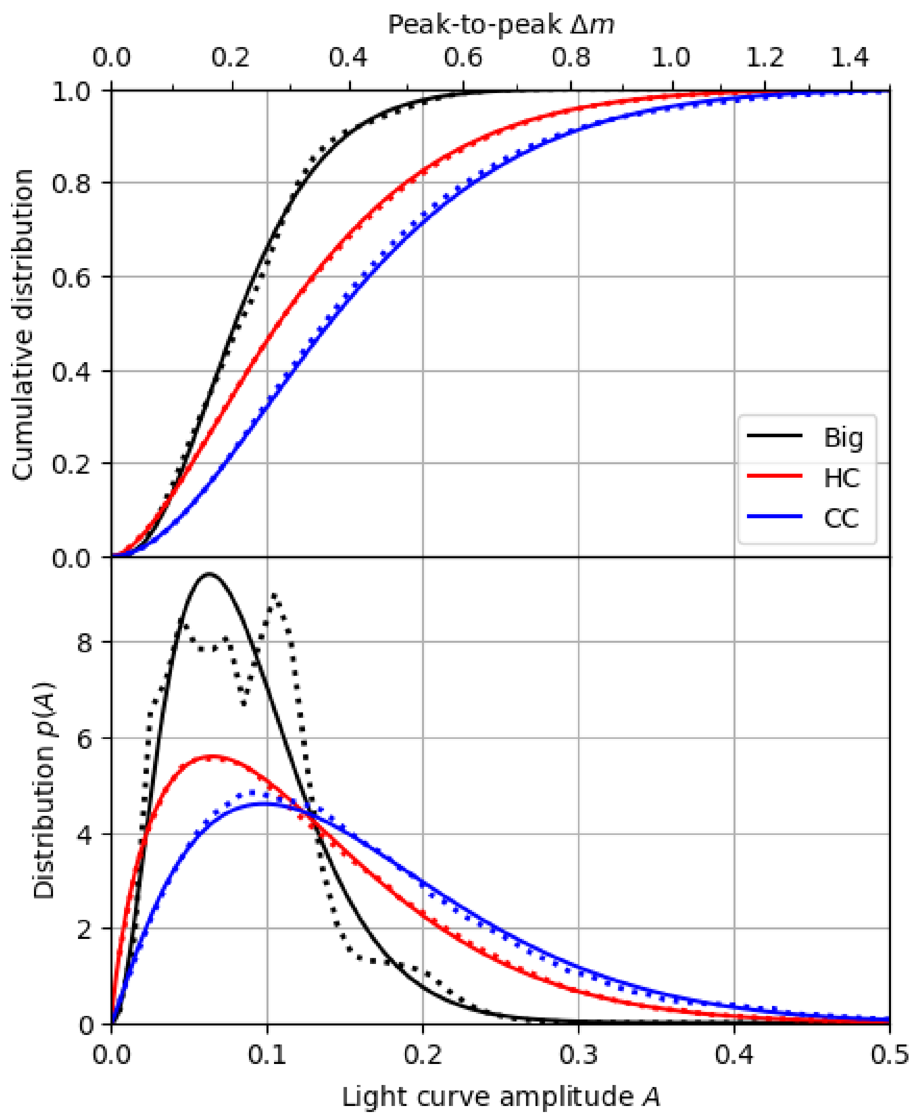


[28]:
```
# Try making a plot of CDFs and PDFs that are fit compared to the posterior␣
↪distribution
# for 3 subsets
```

```
fig,ax = pl.subplots(2,1,sharex=True, figsize=(5,6))
pl.subplots_adjust(hspace=0.0,left=0.15,right=0.96, top=0.92)
f = np.arange(0,1,0.001)
delta_mag = 2.5*np.log10((1+f)/(1-f))
for k in 'Big','HC','CC':
    ij = argmaxNd(logp_sets[k])
    fbar = fmean[ij[1]]
    ss = s[ij[0]]
    print(k,fbar,ss)
    p = beta(fbar*(10**ss),(1-fbar)*(10**ss))
    color,linestyle = plotinfo[k]
    ax[0].plot(f, p.cdf(f), color=color, linestyle=linestyle, label=k)
    ax[1].plot(f, p.pdf(f), color=color, linestyle=linestyle, label=k)

    # Make a posterior distribution
    bins = np.linspace(0,1.,101)
    post = np.zeros(len(bins)-1)
    nTNO = np.count_nonzero(subset[k])
    for tno in info['orbitid'][subset[k]]:
        p_A = p.pdf(lca[tno])
        p_A /= (np.sum(p_A)*nTNO)
        post += np.histogram(lca[tno],bins=bins,weights=p_A)[0]
    ax[0].plot(bins[1:],np.cumsum(post),color=color,linestyle=":",lw=2)
    ax[1].plot(0.5*(bins[1:]+bins[:-1]),post/0.01,color=color,linestyle=":
 ↪",lw=2)
ax[0].set_xlim(0,0.5)
ax[0].set_ylim(0,1)
ax[1].set_ylim(0,9.8)
ax[1].set_xlabel(r'Light curve amplitude $A$')
ax[1].set_ylabel(r'Distribution $p(A)$')
ax[0].set_ylabel(r'Cumulative distribution')
ax[0].legend(loc=4)
ax[1].grid()
ax[0].grid()
# Ticks along top axis giving Delta M
def dm(a):
    aa = np.array(a)
    return 2.5*np.log10((1+aa)/(1-aa))
def aa(dm):
    out = 10**(0.4*dm)
    return (out-1)/(out+1)
ax2 = ax[0].twiny()
topticks = np.arange(0.,1.5,0.2)
ax2.set_xlim(ax[0].get_xlim())
ax2.set_xticks(aa(topticks),major=True)
ax2.set_xticks(aa(np.arange(0.1,1.6,0.2)),minor=True)
ax2.set_xticklabels(['{:.1f}'.format(f) for f in topticks])
```

```
ax2.set_xlabel(r'Peak-to-peak $\Delta m$')
pl.savefig('pdfcdf.pdf')
```

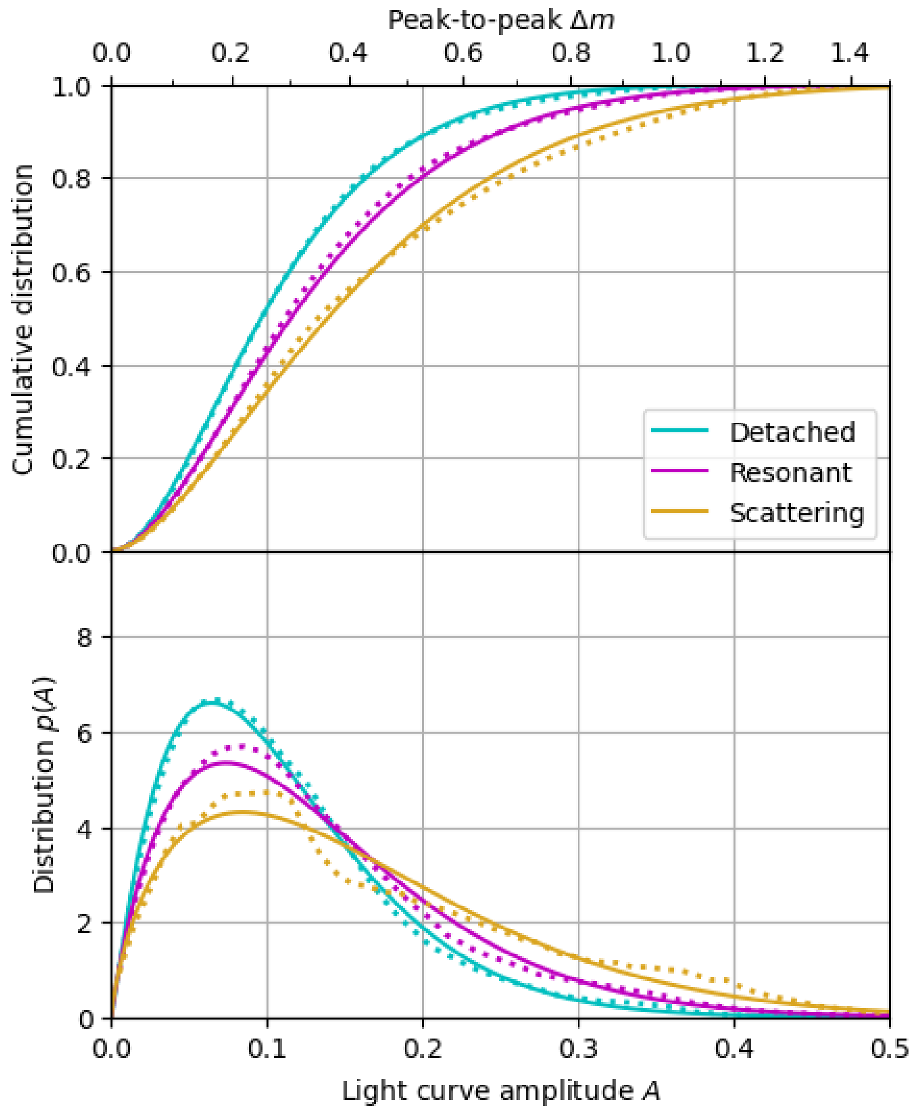
```
Big 0.08750000000000002 1.550000000000001
HC 0.12500000000000006 1.1600000000000006
CC 0.15750000000000008 1.1300000000000006
```



[29]:
```
# Try making a plot of CDFs and PDFs that are fit compared to the posterior␣
↪distribution
# for 3 subsets
fig,ax = pl.subplots(2,1,sharex=True, figsize=(5,6))
pl.subplots_adjust(hspace=0.0,left=0.15,right=0.96, top=0.92)
```

```
f = np.arange(0,1,0.001)
delta_mag = 2.5*np.log10((1+f)/(1-f))
for k in 'Detached','Resonant','Scattering':
    ij = argmaxNd(logp_sets[k])
    fbar = fmean[ij[1]]
    ss = s[ij[0]]
    print(k,fbar,ss)
    p = beta(fbar*(10**ss),(1-fbar)*(10**ss))
    color,linestyle = plotinfo[k]
    ax[0].plot(f, p.cdf(f), color=color, linestyle='-', label=k)
    ax[1].plot(f, p.pdf(f), color=color, linestyle='-', label=k)

    # Make a posterior distribution
    bins = np.linspace(0,1.,101)
    post = np.zeros(len(bins)-1)
    nTNO = np.count_nonzero(subset[k])
    for tno in info['orbitid'][subset[k]]:
        p_A = p.pdf(lca[tno])
        p_A /= (np.sum(p_A)*nTNO)
        post += np.histogram(lca[tno],bins=bins,weights=p_A)[0]
    ax[0].plot(bins[1:],np.cumsum(post),color=color,linestyle=":",lw=2)
    ax[1].plot(0.5*(bins[1:]+bins[:-1]),post/0.01,color=color,linestyle=":",lw=2)
ax[0].set_xlim(0,0.5)
ax[0].set_ylim(0,1)
ax[1].set_ylim(0,9.8)
ax[1].set_xlabel(r'Light curve amplitude $A$')
ax[1].set_ylabel(r'Distribution $p(A)$')
ax[0].set_ylabel(r'Cumulative distribution')
ax[0].legend(loc=4)
ax[1].grid()
ax[0].grid()
# Ticks along top axis giving Delta M
def dm(a):
    aa = np.array(a)
    return 2.5*np.log10((1+aa)/(1-aa))
def aa(dm):
    out = 10**(0.4*dm)
    return (out-1)/(out+1)
ax2 = ax[0].twiny()
topticks = np.arange(0.,1.5,0.2)
ax2.set_xlim(ax[0].get_xlim())
ax2.set_xticks(aa(topticks),major=True)
ax2.set_xticks(aa(np.arange(0.1,1.6,0.2)),minor=True)
ax2.set_xticklabels(['{:.1f}'.format(f) for f in topticks])
ax2.set_xlabel(r'Peak-to-peak $\Delta m$')
pl.savefig('pdfcdfdrs.pdf')
```

```
Detached 0.11000000000000004 1.2800000000000007
Resonant 0.13250000000000006 1.1600000000000006
Scattering 0.1600000000000001 1.0400000000000005
```



[ ]: