In [1]:
```
import numpy as np
from astropy import units as u
from astropy.coordinates import SkyCoord
import scipy as sp
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
import emcee
import corner
%matplotlib inline
```

In [2]:
```
def cart2pol(x, y):
    rho = np.sqrt(x**2 + y**2)
    phi = np.arctan2(y, x)
    return(rho, phi)

def pol2cart(rho, phi):
    x = rho * np.cos(phi)
    y = rho * np.sin(phi)
    return(x, y)
```

Using polar coordinates centered on the lens galaxy, the combined lens potential can be written as

$$\phi(r,\theta) = brf(\theta) + \frac{r^2}{2}(\gamma_c \cos 2\theta + \gamma_s \sin 2\theta)$$

where

$$f(\theta) = [1 - \epsilon \cos 2(\theta - \theta_0)]^{1/2}$$

The deflection vector $\nabla\phi$ has cartesian components

$$\nabla_x \phi = \frac{b}{f(\theta)}[\cos\theta - \epsilon\cos(\theta - 2\theta_0)] + \gamma_c r\cos\theta + \gamma_s r\sin\theta$$

$$\nabla_y \phi = \frac{b}{f(\theta)}[\sin\theta - \epsilon\sin(\theta - 2\theta_0)] + \gamma_s r\cos\theta - \gamma_c r\sin\theta$$

The gravitational lens equation has the form

$$\vec{u} = \vec{x} - \nabla\phi(\vec{x})$$

which is really a set of two equations.

$$u = x - \nabla_x \phi(\vec{x})$$
$$v = y - \nabla_y \phi(\vec{x})$$

We need a penalty function χ^2 that will determine the parameters $b, \epsilon, \gamma_c, \gamma_s, \theta_0$ in the lens potential ϕ and the parameters u, v, the position of the source. Let $\vec{x}_i, \sigma_i,\ i = 0, 1, 2, 3$ be the positions and uncertainties of the four images. Based on Keeton (2010, Gen.Rel.Grav., 42, 2151) we will define our χ^2 function to be in the source plane. This eliminates the need for solving the lens equation (which is computationally expensive), and is a fine apprixmation given how small our uncertainties are. Let $\vec{\mu}_i = (\mu_i, \nu_i)$ be the position of the source as calculated from the lens equation using the ith image position. Then we can define our penalty function to be

$$\chi^2 = \sum_{i=0}^{3} \frac{1}{\sigma_i^2} (\vec{u} - \vec{\mu}_i)^2$$

In [3]:
```
def f(theta, eps, theta0): # defining f(ϑ)
    return (1 - eps * np.cos(2 * (theta - theta0)))**(1/2)

def phi(rtheta, b, eps, gc, gs, theta0): # ϕ
    r, theta = rtheta
    return b * r * f(theta, eps, theta0) + r**2 / 2 * ( gc * np.cos(2*theta) + 
gs * np.sin(2*theta))

def dxphi(rtheta, b, eps, gc, gs, theta0): # deflection vector x component
    r, theta = rtheta
    return b / f(theta, eps, theta0) * (np.cos(theta) - eps * np.cos( theta - 
2 * theta0)) \
        + gc * r * np.cos(theta) + gs * r * np.sin(theta)


def dyphi(rtheta, b , eps, gc, gs, theta0): #deflection vector y component
    r, theta = rtheta
    return b / f(theta, eps, theta0) * (np.sin(theta) - eps * np.sin( theta - 
2 * theta0)) \
        + gs * r * np.cos(theta) - gc * r * np.sin(theta)


def calcsource(params, rtheta): # calculates the source given the ϕ-params and 
an image position
    b, eps, gc, gs, theta0, _, _ = params
    r, theta = rtheta
    x, y = pol2cart(r, theta)
    mu = x - dxphi(rtheta, b, eps, gc, gs, theta0)
    nu = y - dyphi(rtheta, b, eps, gc, gs, theta0)
    return (mu, nu)


def lnprob(params, rthetasigma): # our χ^2 function
    b, eps, gc, gs, theta0, u, v = params
    r, theta, sigma = rthetasigma
    if b<0 or not 0<=eps<1 or not 0<=theta0<2*np.pi: return -np.inf
    chi2 = 0
    for i in range(len(r)):
        mu, nu = calcsource(params, (r[i], theta[i]))
        chi2 += ((u - mu)**2 + (v - nu)**2) / sigma[i]**2
    return -chi2 / 2
```

In [4]:
```
def mcmc_model(data, nwalk=20, nburn=100000, nmain=200000, lensnum=None):
    ndim = 7

    # randomly generating starting points
    p0 = np.zeros((nwalk,ndim))
    for iwalk in range(nwalk):
        p0[iwalk,0] = np.random.uniform()
        p0[iwalk,1] = np.random.uniform()
        p0[iwalk,2] = np.random.uniform()
        p0[iwalk,3] = np.random.uniform()
        p0[iwalk,4] = np.random.uniform(low=0, high=2*np.pi)
        p0[iwalk,5] = np.random.uniform()
        p0[iwalk,6] = np.random.uniform()

    sampler = emcee.EnsembleSampler(nwalk,ndim,lnprob, args=(data,))

    # burn-in run
    pos,prob,state = sampler.run_mcmc(p0,nburn)
    sampler.reset()

    # main run
    res = sampler.run_mcmc(pos,nmain)
    samples = sampler.chain.reshape((-1,ndim))

    # creating the corner plot
    fig = corner.corner(samples,show_titles=True,labels=('b','ϵ','γc','γs','θ
0','u','v'), title_fmt='.5f')
    params = samples[-1,:]
    fig.suptitle('Lens {0}; $\chi^2 = {{{1:.3f}}}$'.format(lensnum, lnprob(par
ams, data)), fontsize=22)

    return params, fig
```

In [5]:
```
def plot_chi2(x, y, b, eps, gc, gs, theta0, u, v): # the chi^2 function for th
e contour plot
    rtheta = cart2pol(x,y)
    return (x - dxphi(rtheta, b, eps, gc, gs, theta0) - u)**2 + (y - dyphi(rth
eta, b, eps, gc, gs, theta0) - v)**2

def plot_model(data, params, sqrlim=None, lensnum=None):
    # handling inputs
    data = data[0:2].T
    xyimg = np.array(pol2cart(data[:,0], data[:,1]))
    b, eps, gc, gs, theta0, u, v = params

    # creating and evaluating points for the contour plot
    if sqrlim==None:
        sqrlim = max(xyimg.flatten(), key=lambda x:abs(x)) * 1.3
    levels = np.arange(-8, 0, 0.5)
    x = np.linspace(-sqrlim, sqrlim, 500)
    y = np.linspace(-sqrlim, sqrlim, 500)
    xx,yy = np.meshgrid(x,y)
    zz = plot_chi2(xx, yy, b, eps, gc, gs, theta0, u, v)

    # plotting the contour plot and the images on top of it
    fig, ax = plt.subplots(figsize=(14,14))
    c = ax.contour(xx, yy, np.log10(zz), levels=levels, zorder=0)
    divider = make_axes_locatable(ax)
    cax = divider.append_axes("right", size="5%", pad=0.1)
    cb = plt.colorbar(c, cax=cax)
    ax.set_aspect('equal', adjustable='box')
    imgs = ax.scatter(xyimg[0], xyimg[1], color='k', marker=(5,1,0), s=300, zo
rder=1, label='Gilman et al. 2019')

    # beautification
    ax.legend(loc='upper right', fontsize=16)
    ax.set_xlabel('$x$', fontsize=22)
    ax.set_ylabel('$y$', fontsize=22)
    cb.set_label(r'$\log ( \, \chi^2 = (x - \Delta_x \phi - u)^2 + (y - \Delta
_y \phi - v)^2 \, )$', fontsize=18)
    ax.set_title('Model and Observed Images; Lens {}'.format(lensnum), fontsiz
e=22)

    return fig
```

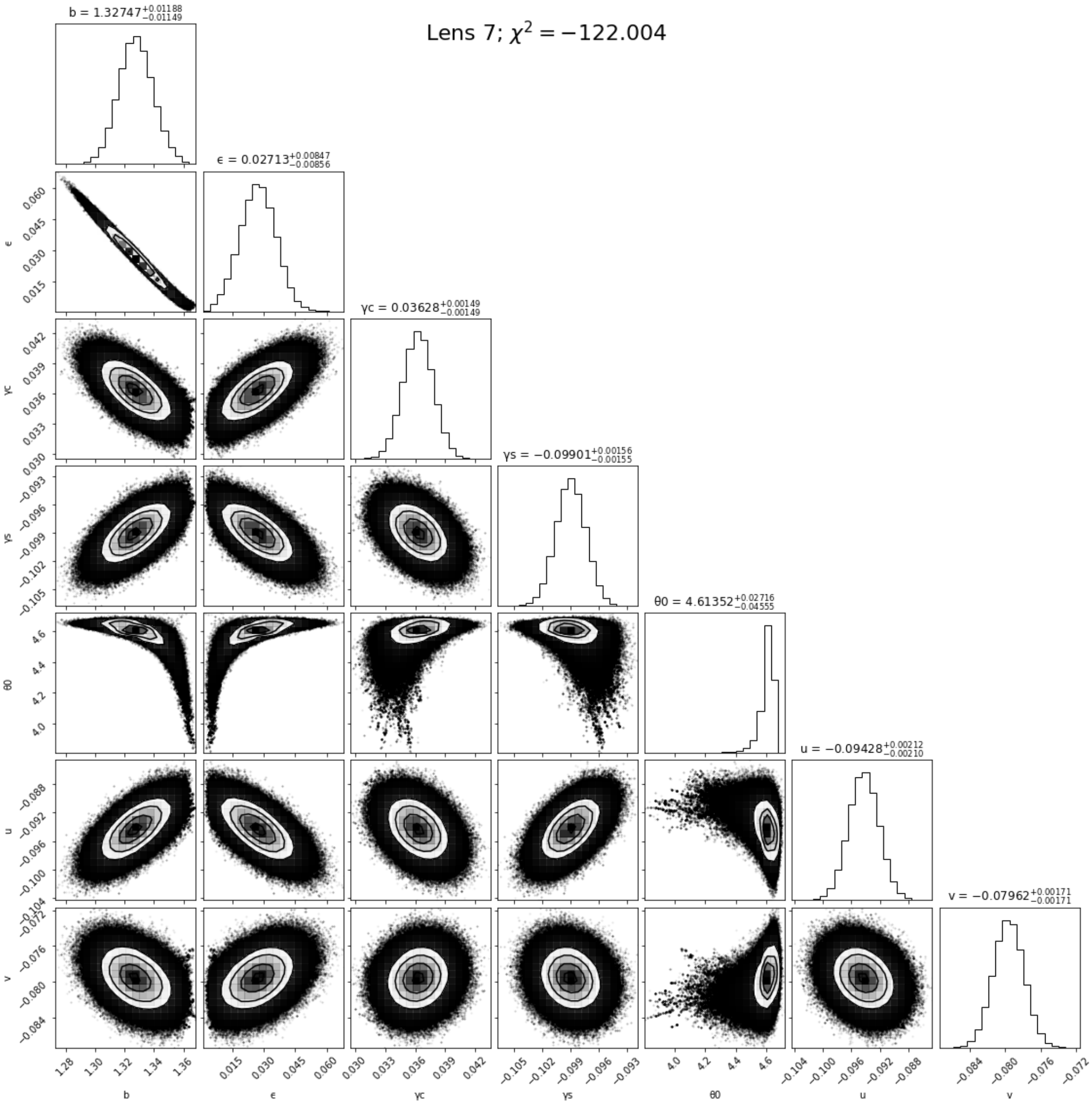
Here we import the data from the file lensData.txt, in which the data is stored as described

For each lens there should be 11 numbers First we have the uncertainty, then we have the dRA and dDec for the lensing galaxy Then we have the dRA and dDec for each image (4 images each) If we have 3 lenses, we should have 33 numbers in our chain, and so on...

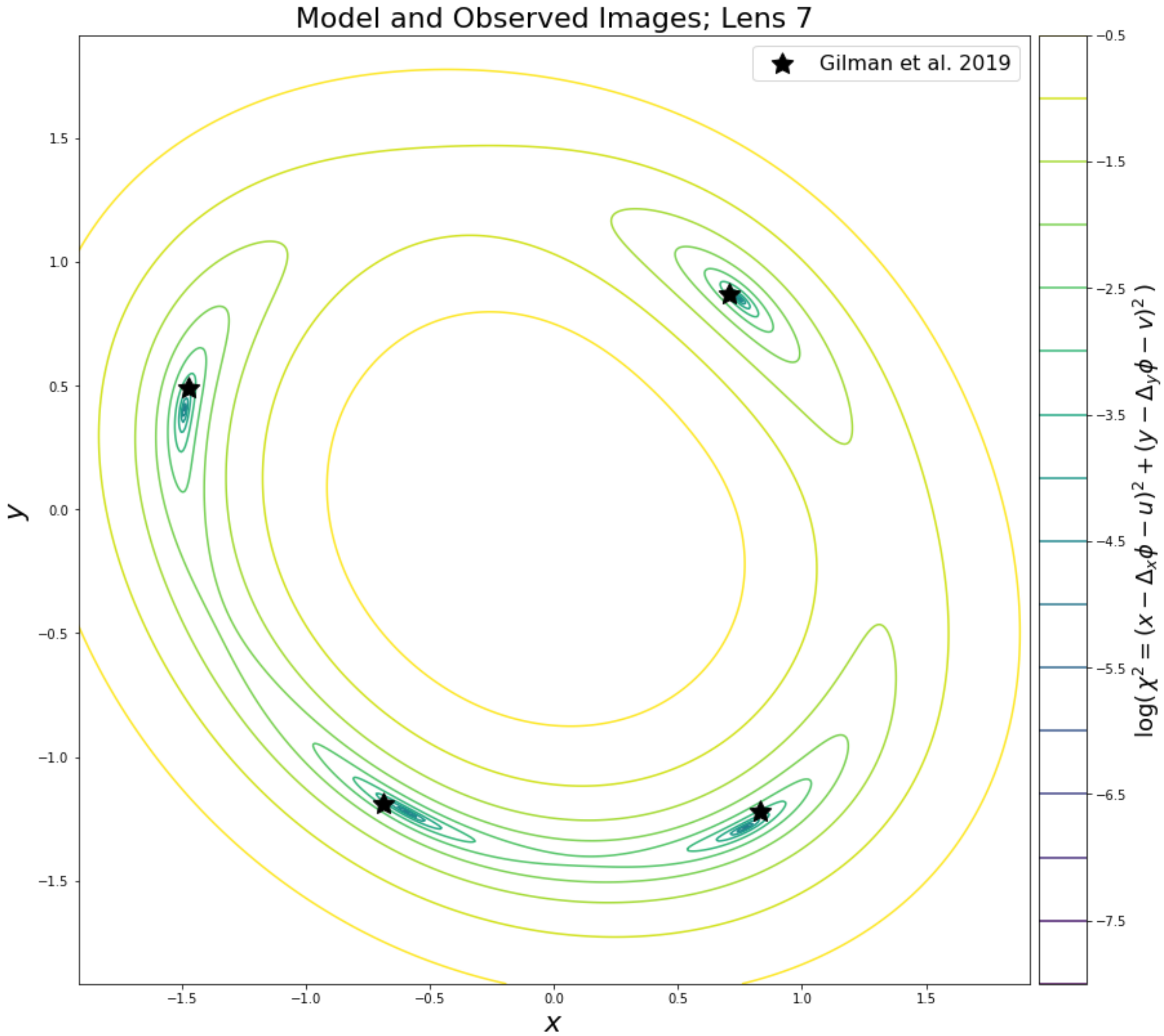
In [6]:

```python
file1 = open('lensData.txt','r')
Lines = file1.read().replace('\n',' ');


a= Lines.split(" ")

numLenses = 8
numImages = 4
oldData = np.zeros((numLenses,3,numImages))
newData = np.zeros((numLenses,3,numImages))

i=0
j=0
k=0
count=0;
while (i< numLenses):
    uncert= a[count]
    count=count+1;
    xChange=a[count]
    count=count+1
    yChange=a[count]
    count=count+1
    while (j< numImages):
        oldData[i][0][j] = float(a[count]) - float(xChange)
        count=count+1;
        oldData[i][1][j] = float(a[count]) - float(yChange)
        count=count+1;
        oldData[i][2][j]= float(uncert);
        j=j+1;
    i=i+1;
    j=0


i=0
j=0
k=0

while (i<numLenses):
    while (j<numImages):
        newData[i][2][j] = oldData[i][2][j]
        newData[i][0][j],newData[i][1][j] = cart2pol(oldData[i][0][j], oldData
[i][1][j])
        j=j+1;
    i=i+1;
    j=0

#We now have our data stored as polar coordinates.
#In our array, we have a 3D array of length [8(number of lenses)][3(numer of p
aramters)][4(number of images)]
#In which we store the values for r and theta for each image, and then the unc
ertainty of the image
```

In [7]:
```
for i in range(8)[-1:]:
    try: params, corfig = mcmc_model(newData[i], lensnum=i)
    except: params, corfig = mcmc_model(newData[i], lensnum=i)
    plt.savefig('plots/corner_{}.png'.format(i), format='png', dpi=300)
    plt.show()

    contfig = plot_model(newData[i], params, lensnum=i)
    plt.savefig('plots/contour_{}.png'.format(i), format='png', dpi=300)
    plt.show()
```



Lens 7; χ² = −122.004
b = 1.32747 +0.01188 −0.01149
ϵ = 0.02713 +0.00847 −0.00856
γc = 0.03628 +0.00149 −0.00149
γs = −0.09901 +0.00156 −0.00155
θ0 = 4.61352 +0.02716 −0.04555
u = −0.09428 +0.00212 −0.00210
v = −0.07962 +0.00171 −0.00171
b
ϵ
γc
γs
θ0
u
v



Model and Observed Images; Lens 7
Gilman et al. 2019
y
x
$\log(\chi^2 = (x - \Delta_x\phi - u)^2 + (y - \Delta_y\phi - v)^2)$