Lecture 5

March 15, 2016

1 Image distortions at second order

Lens mapping at second order include a second order term in the expansion of the deflection angles. The mapping equations become:

$$\beta_1 = A_{11}\theta_1 + A_{12}\theta_2 + \frac{1}{2}(D_{111}\theta_1^2 + 2D_{112}\theta_1\theta_2 + D_{221}\theta_2^2)$$

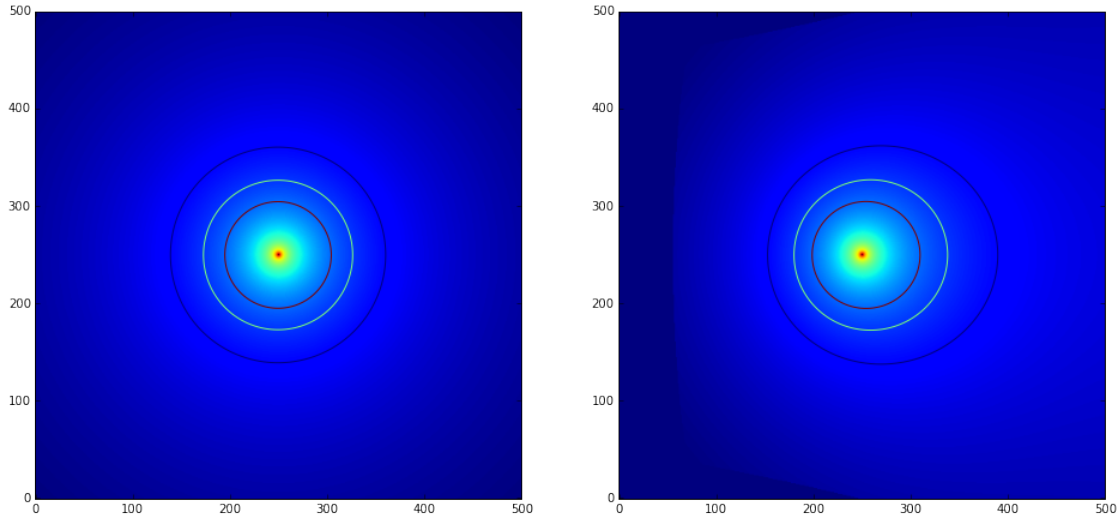
$$\beta_2 = A_{22}\theta_2 + A_{12}\theta_1 + \frac{1}{2}(D_{222}\theta_2^2 + 2D_{221}\theta_1\theta_2 + D_{112}\theta_1^2)$$

In [2]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import galflex
gal1 = galflex.Sersic(1.0, 4, N=500, flux=10.0, q=1.0, phi=0.0, cx=0.0,cy=0.0, re=80.0)
gal = galflex.Sersic(1.0, 4, N=500, flux=10.0, q=1.0, phi=0.0, cx=0.0,cy=0.0, re=80.0)
gal.lens(kap=0.0, gamma1=0.0, gamma2=0.0, f1=0.5,f2=0.0,g1=0.0,g2=0.0)
#gal.plot()
fig,ax=plt.subplots(1,2,figsize=(16,8))
ax[0].imshow(gal1.image,origin='lower')
ax[0].contour(gal1.image,levels=[0.00005,0.00006,0.00007])
ax[1].imshow(gal.image,origin='lower')
ax[1].contour(gal.image,levels=[0.00005,0.00006,0.00007])
```

```
/Users/massimo/anaconda/envs/python2/lib/python2.7/site-packages/matplotlib/collections.py:650: FutureWa
  if self._edgecolors_original != str('face'):
```

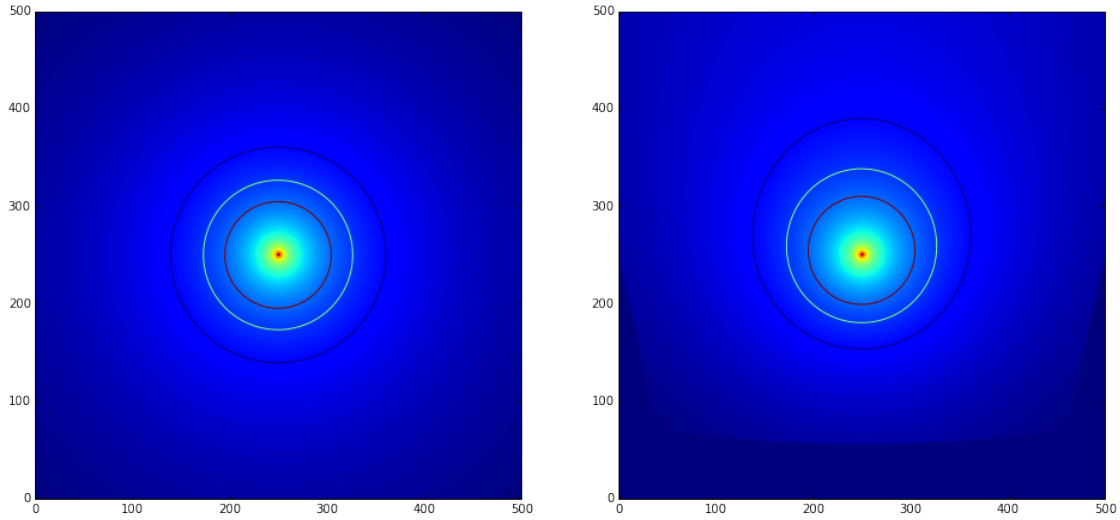
Out[2]: <matplotlib.contour.QuadContourSet instance at 0x10c218b90>

```
/Users/massimo/anaconda/envs/python2/lib/python2.7/site-packages/matplotlib/collections.py:590: FutureWa
  if self._edgecolors == str('face'):
```



In [3]:
```
import galflex
gal1 = galflex.Sersic(1.0, 4, N=500, flux=10.0, q=1.0, phi=0.0, cx=0.0,cy=0.0, re=80.0)
gal = galflex.Sersic(1.0, 4, N=500, flux=10.0, q=1.0, phi=0.0, cx=0.0,cy=0.0, re=80.0)
gal.lens(kap=0.0, gamma1=0.0, gamma2=0.0, f1=0.0,f2=0.5,g1=0.0,g2=0.0)
#gal.plot()
fig,ax=plt.subplots(1,2,figsize=(16,8))
ax[0].imshow(gal1.image,origin='lower')
ax[0].contour(gal1.image,levels=[0.00005,0.00006,0.00007])
ax[1].imshow(gal.image,origin='lower')
ax[1].contour(gal.image,levels=[0.00005,0.00006,0.00007])
```

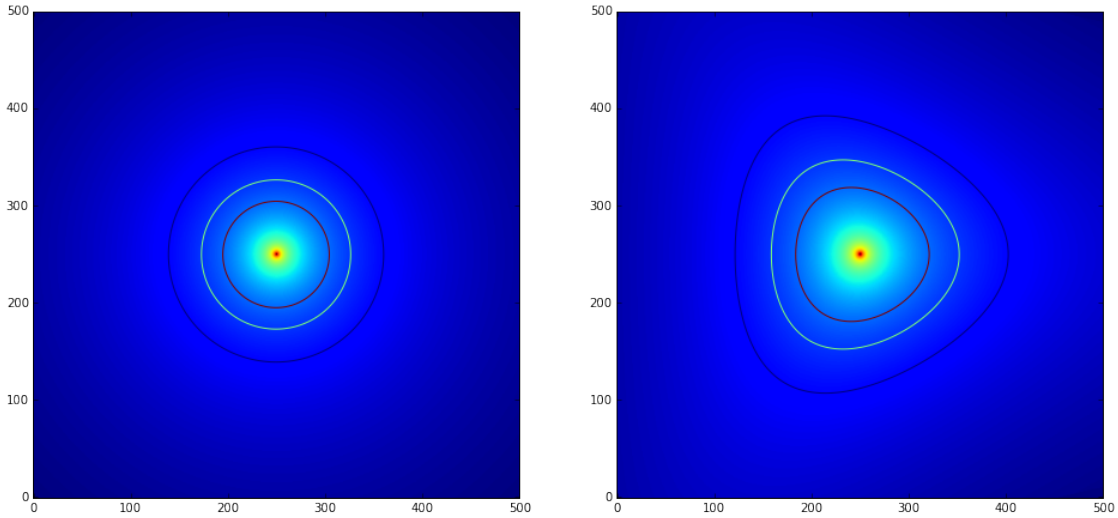
Out[3]: <matplotlib.contour.QuadContourSet instance at 0x10e2b3200>



In [4]:
```
import galflex
gal1 = galflex.Sersic(1.0, 4, N=500, flux=10.0, q=1.0, phi=0.0, cx=0.0,cy=0.0, re=80.0)
```

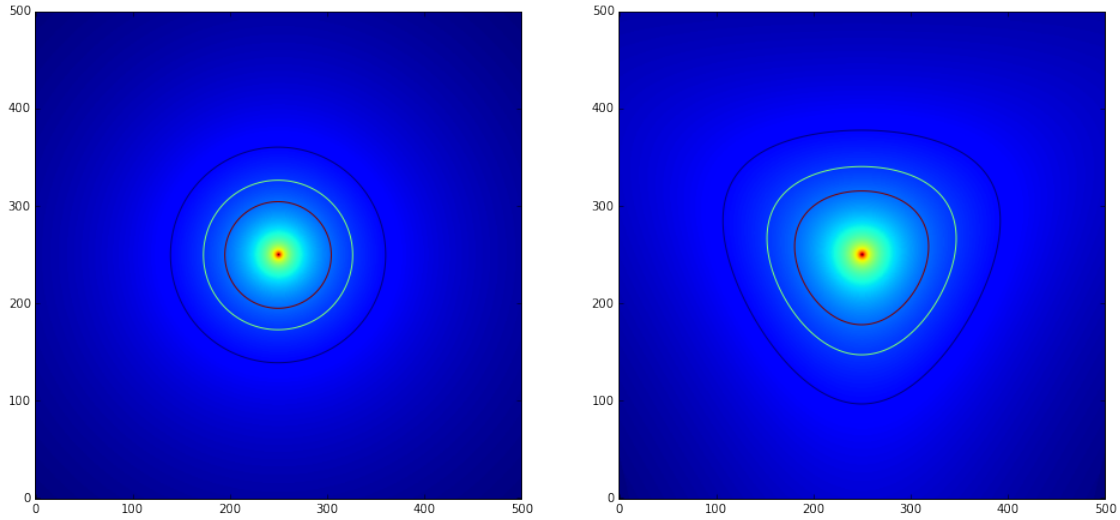
```
gal = galflex.Sersic(1.0, 4, N=500, flux=10.0, q=1.0, phi=0.0, cx=0.0,cy=0.0, re=80.0)
gal.lens(kap=0.2, gamma1=0.0, gamma2=0.0, f1=0.0,f2=0.0,g1=0.5,g2=0.0)
#gal.plot()
fig,ax=plt.subplots(1,2,figsize=(16,8))
ax[0].imshow(gal1.image,origin='lower')
ax[0].contour(gal1.image,levels=[0.00005,0.00006,0.00007])
ax[1].imshow(gal.image,origin='lower')
ax[1].contour(gal.image,levels=[0.00005,0.00006,0.00007])
```

Out[4]: <matplotlib.contour.QuadContourSet instance at 0x10e5ba320>



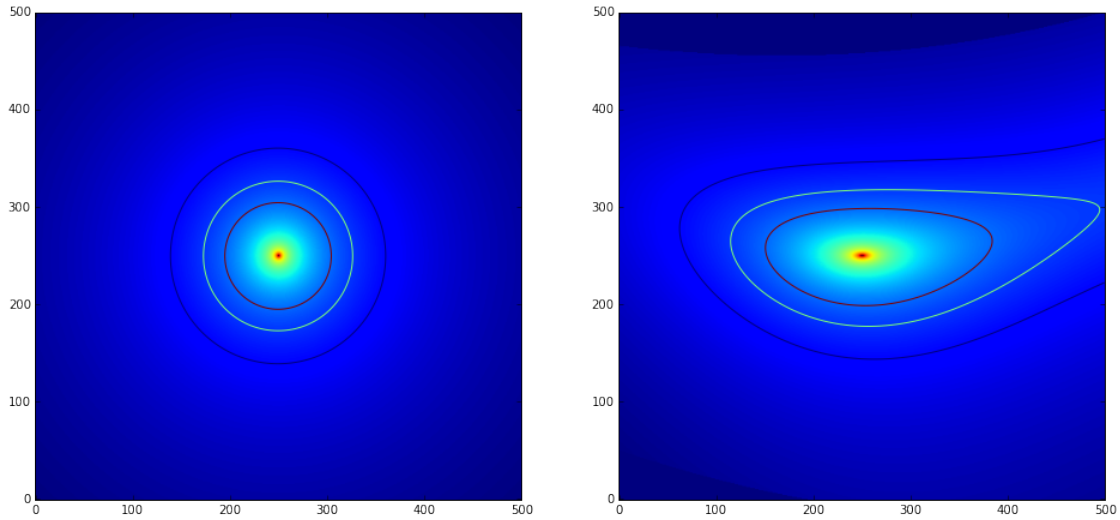
In [5]:
```
import galflex
gal1 = galflex.Sersic(1.0, 4, N=500, flux=10.0, q=1.0, phi=0.0, cx=0.0,cy=0.0, re=80.0)
gal = galflex.Sersic(1.0, 4, N=500, flux=10.0, q=1.0, phi=0.0, cx=0.0,cy=0.0, re=80.0)
gal.lens(kap=0.2, gamma1=0.0, gamma2=0.0, f1=0.0,f2=0.0,g1=0.0,g2=0.5)
#gal.plot()
fig,ax=plt.subplots(1,2,figsize=(16,8))
ax[0].imshow(gal1.image,origin='lower')
ax[0].contour(gal1.image,levels=[0.00005,0.00006,0.00007])
ax[1].imshow(gal.image,origin='lower')
ax[1].contour(gal.image,levels=[0.00005,0.00006,0.00007])
```

Out[5]: <matplotlib.contour.QuadContourSet instance at 0x11243a320>



In a realistic case, lensing effects at various orders are mixed together. The following example illustrates what happens in the case of convergence, shear and flexions acting simultaneously.

```
In [6]: import galflex
        gal1 = galflex.Sersic(1.0, 4, N=500, flux=10.0, q=1.0, phi=0.0, cx=0.0,cy=0.0, re=80.0)
        gal = galflex.Sersic(1.0, 4, N=500, flux=10.0, q=1.0, phi=0.0, cx=0.0,cy=0.0, re=80.0)
        gal.lens(kap=0.2, gamma1=0.3, gamma2=0.0, f1=0.2,f2=0.0,g1=0.0,g2=0.5)
        #gal.plot()
        fig,ax=plt.subplots(1,2,figsize=(16,8))
        ax[0].imshow(gal1.image,origin='lower')
        ax[0].contour(gal1.image,levels=[0.00005,0.00006,0.00007])
        ax[1].imshow(gal.image,origin='lower')
        ax[1].contour(gal.image,levels=[0.00005,0.00006,0.00007])

Out[6]: <matplotlib.contour.QuadContourSet instance at 0x112dc3098>
```



2 Time delays

The time-delay surface is

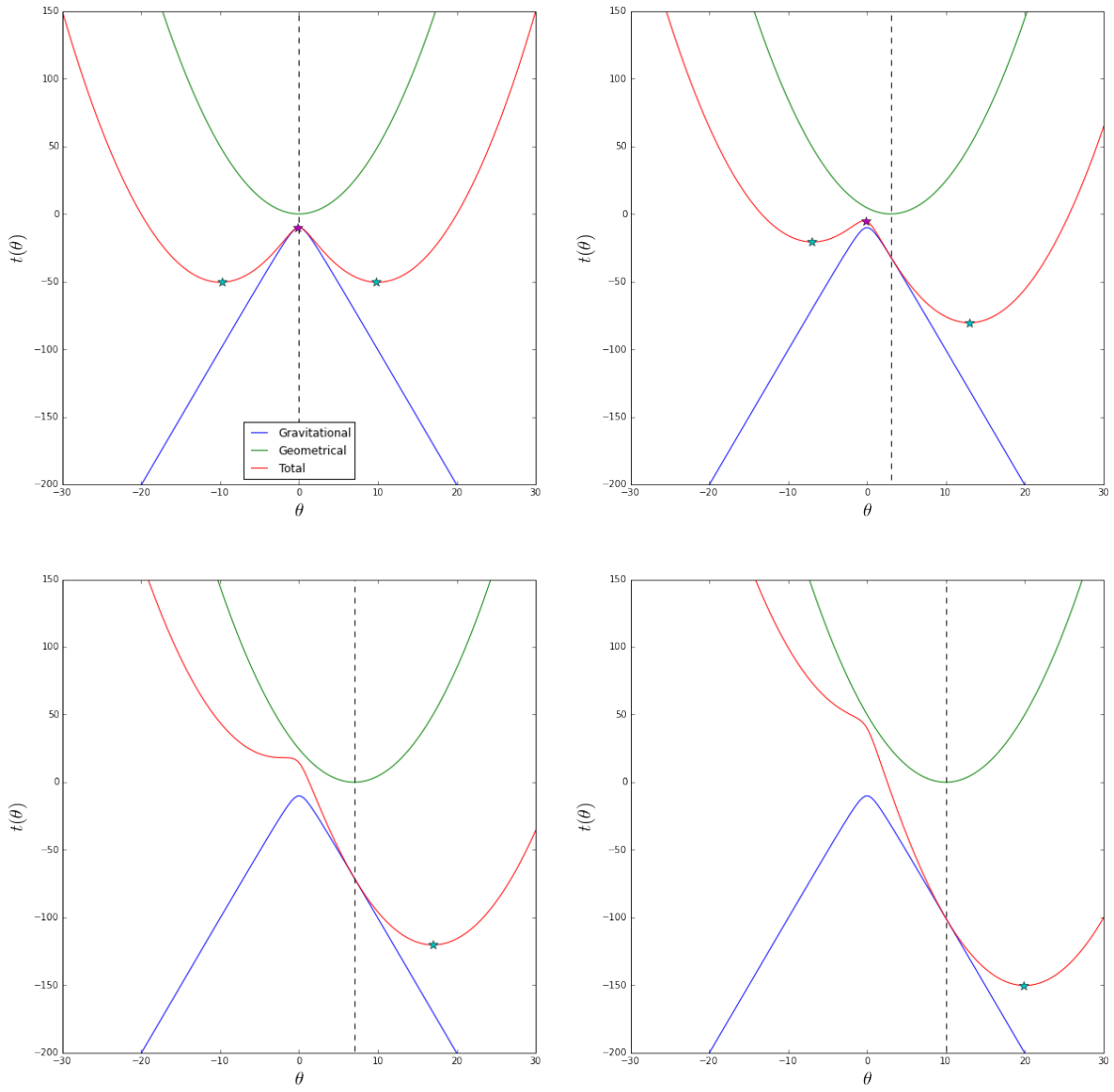
$$t(\vec{\theta}) \propto \left[\frac{1}{2}(\vec{\theta}-\vec{\beta})^2 - \hat{\Psi}(\vec{\theta})\right]$$

The two terms entering this equations are called the geometrical and the gravitational time delays, respectively.

In the follwoing example, we reduce the problem to one dimension. Instead of dealing with a time-delay surface, we consider a time-delay function (by simpling substituting the vectors $\vec{\theta}$ and $\vec{\beta}$ with two scalars. We will change the relative position of source and lens to see how the time-delay function changes.

In [36]:
```
import numpy as np

theta=np.linspace(-200.0,200.0,1000)

def t_grav_func(theta,theta_c=1.0):
    # the lensing potential of a softened-isothermal-sphere
    return -np.sqrt(theta*theta+theta_c*theta_c)*10

theta_c=1.0
t_grav=t_grav_func(theta,theta_c)


def t_geom_func(theta,beta):
    # the geometrical time-delay
    return 0.5*(theta-beta)**2

# the position of the source
beta=[0,3,7,10]

# plot results
%matplotlib inline
fig,ax=plt.subplots(2,2,figsize=(20,20))
for i in range(len(beta)):
    ix=int(i/2)
    iy=i-ix*2

    # compute the time-delay as  a function of \theta
    t_geom=t_geom_func(theta,beta[i])

    t_total=t_grav+t_geom

    ax[ix,iy].plot(theta,t_grav,label='Gravitational')
    ax[ix,iy].plot(theta,t_geom,label='Geometrical')
    ax[ix,iy].plot(theta,t_total,label='Total')
    ax[ix,iy].set_xlim([-30,30])
    ax[ix,iy].set_ylim([-200,150])
    x=[beta[i],beta[i]]
    y=[-1500,1000]
    ax[ix,iy].plot(x,y,'--',color='black')
    if (ix == 0 and iy ==0):
        ax[ix,iy].legend(loc='best')
```

```
        # find local minima and maxima of the function and mark their positions on the curve
        i_min= (np.diff(np.sign(np.diff(t_total))) > 0).nonzero()[0] + 1 # local min
        i_max= (np.diff(np.sign(np.diff(t_total))) < 0).nonzero()[0] + 1 # local max
        ax[ix,iy].plot(theta[i_min],t_total[i_min],'*',markersize=10)
        ax[ix,iy].plot(theta[i_max],t_total[i_max],'*',markersize=10)
        ax[ix,iy].set_xlabel(r'$\theta$',fontsize=20)
        ax[ix,iy].set_ylabel(r'$t(\theta)$',fontsize=20)

fig.savefig("/Users/massimo/Dropbox/Lensing Course/figs/timedelay_1D.png")
```



For $\beta < 7$, the source has three images, occurring at two local minima and one maximum of $t(\theta)$. While the minima locations are symmetrical with respect to the center of the lens when $\beta = 0$, the symmetry is broken otherwise. When the source moves away from the lens center, one minimum follows the source, while the maximum and the other minimum approach each other on the opposite side of the lens. Notice that the $t(\theta)$ becomes increasingly flat in between these two images. Given that the curvature is proportional to $\det A = \mu^{-1}$, this implies that the magnification of these two images increases as they approach each other.

When $\beta \sim 7$, the maximum and one of the minima touch each other and disappear. At this time, $t(\theta)$ is flat, indicating that the point where the two images vanish is a critical point. Thus, the source has reached the caustic. We just learn that caustics separate regions of the source plane with different image multiplicities.

As β becomes increasingly larger, the only remaining image tends to approach the source: at large distance from the lens, we will not notice and lensing effect, even in terms of image shift.