Lecture 18

June 6, 2016

1 Elliptical lenses: the SIE

The SIE can be derived from the SIS by making the substitution:

$$\xi \Rightarrow \sqrt{\xi_1^2 + f^2\xi_2^2}$$

so that

$$\Sigma(\xi) = \frac{\sigma_v^2}{2G}\frac{\sqrt{f}}{\sqrt{\xi_1^2 + f^2\xi_2^2}}$$

With this substitution, $\Sigma(\xi)$ will be constant on ellipses with minor axis ξ and major axis $\xi/\sqrt{f}$, oriented such that the major axis is along the ξ_2 axis.

By choosing $\xi_0 = \xi_{0,SIS}$ as reference scale, and using polar coordinates, we obtain

$$\kappa(x,\varphi) = \frac{\sqrt{f}}{2x\Delta(\varphi)}$$

where

$$\Delta(\varphi) = \sqrt{\cos\varphi^2 + f^2\sin\varphi^2}$$

The lensing potential can be found by solving the Poisson equation:

$$\frac{\partial^2\psi}{\partial x^2} + \frac{1}{x}\frac{\partial\psi}{\partial x} + \frac{1}{x^2}\frac{\partial\psi}{\partial\varphi} = 2\kappa = \frac{\sqrt{f}}{\Delta(\varphi)}$$

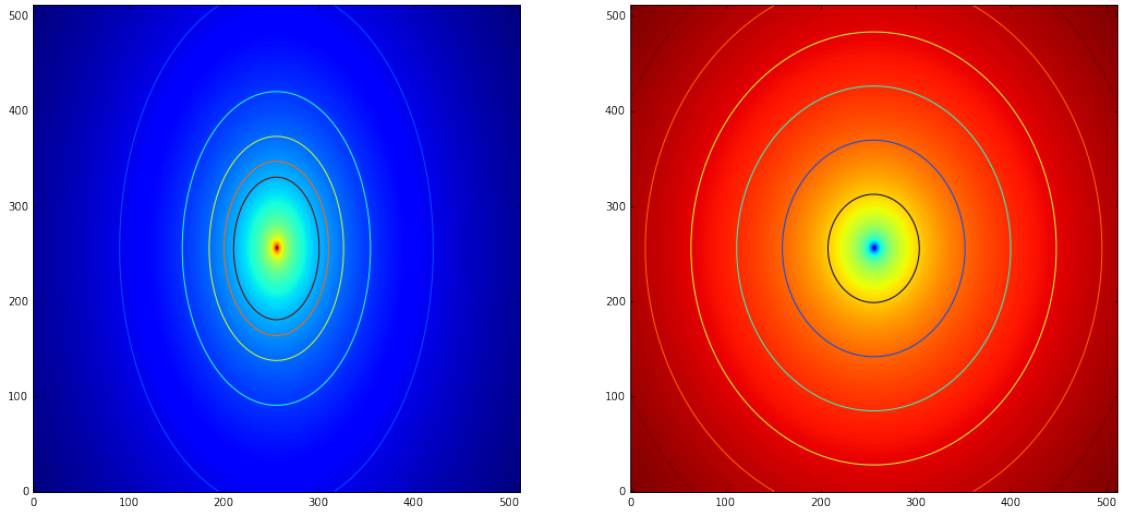
Making the ansatz $\psi(x,\varphi) := x\tilde{\psi}(\varphi)$ and using Green's method, we find

$$\psi(x,\varphi) = x\frac{\sqrt{f}}{f'}\left[\sin\varphi\arcsin\left(f'\sin\varphi\right) + \cos\varphi\,\mathrm{arcsinh}(f'/f\cos\varphi)\right]$$

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from matplotlib.colors import LogNorm, SymLogNorm
        %matplotlib inline

        def delta(f,phi):
            return np.sqrt(np.cos(phi)**2+f**2*np.sin(phi)**2)

        def kappa_ell(x,phi,f):
            return(np.sqrt(f)/2.0/x/delta(f,phi))

        def gamma_ell(x,phi,f):
            return(-kappa_ell(x,phi,f)*np.cos(2.0*phi),-kappa_ell(x,phi,f)*np.sin(2.0*phi))
```

```
def mu_ell(x,phi,f):
    ga1,ga2=gamma_ell(x,phi,f)
    ga=np.sqrt(ga1*ga1+ga2*ga2)
    return 1.0/(1.0-kappa-ell(x,phi,f)-ga)/(1.0-kappa-ell(x,phi,f)+ga)

def kappa_map(xk,yk,f):

    mappa=[]
    for i in range(xk.size):
        for j in range(yk.size):
            phi=np.arctan2(yk[j],xk[i])
            x=np.sqrt(xk[i]*xk[i]+yk[j]*yk[j])
            mappa.append(kappa_ell(x,phi,f))
    mappa=np.array(mappa)
    mappa=mappa.reshape([xk.size,yk.size],order='F')

    return(np.abs(mappa))

def psi_tilde(phi,f):
    fp=np.sqrt(1.0-f**2)
    return np.sqrt(f)/fp*(np.sin(phi)*np.arcsin(fp*np.sin(phi))+np.cos(phi)*np.arcsinh(fp/f*np.

def psi_ell(x,phi,f):
    psi=x*psi_tilde(phi,f)
    return psi

def pot_map(xk,yk,f):

    mappa=[]
    for i in range(xk.size):
        for j in range(yk.size):
            phi=np.arctan2(yk[j],xk[i])
            x=np.sqrt(xk[i]*xk[i]+yk[j]*yk[j])
            mappa.append(psi_ell(x,phi,f))
    mappa=np.array(mappa)
    mappa=mappa.reshape([xk.size,yk.size],order='F')

    return(np.abs(mappa))

f=0.6
xmin=-2
xmax=2
ymin=-2
ymax=2
npix=512

xk=np.linspace(xmin,xmax,npix)
yk=np.linspace(ymin,ymax,npix)

fig,ax=plt.subplots(1,2,figsize=(18,8))

kappa=kappa_map(xk,yk,f)
ax[0].imshow(kappa,norm=LogNorm(),origin='low')
cs=ax[0].contour(kappa,levels=[0.1,0.3,0.5,0.7,0.9,1.1])
```

```
pot=pot_map(xk,yk,f)
ax[1].imshow(pot,norm=LogNorm(),origin='low')
cs=ax[1].contour(pot)
```



The deflection angle can be derived as usual by taking the gradient of the lensing potential. It is convenient to opreate in polar coordinates, so that

$$\frac{\partial}{\partial x_1} = \cos\varphi \frac{\partial}{\partial x} - \frac{\sin\varphi}{x}\frac{\partial}{\partial \varphi}$$

and

$$\frac{\partial}{\partial x_1} = \sin\varphi \frac{\partial}{\partial x} + \frac{\cos\varphi}{x}\frac{\partial}{\partial \varphi}$$

Then, we obtain

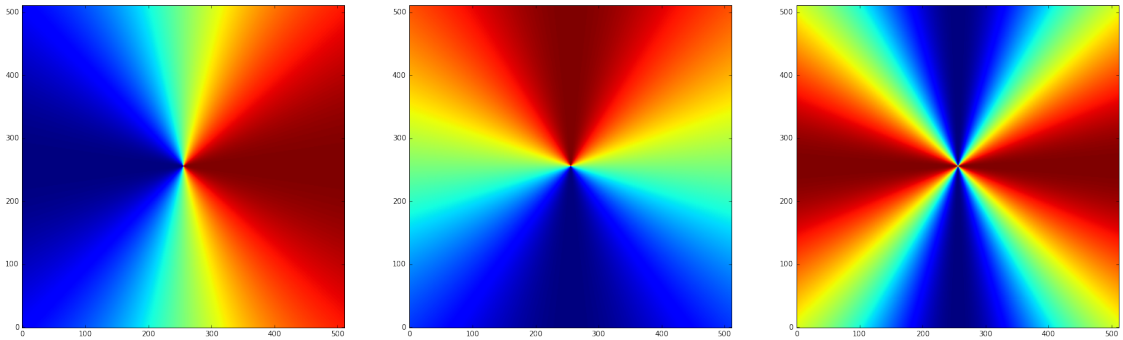
$$\vec{\alpha}(\vec{x}) = \vec{\nabla}\psi = \frac{\sqrt{f}}{f'}\left[\text{arcsinh}\left(\frac{f'}{f}\cos\varphi\right), \arcsin(f'\sin\varphi)\right]$$

As found for the SIS, the deflection angle of the SIE does not depend on x. The figures below show how the two components α_1 and α_2, and the absolute value α depend on the phase φ.

```
In [2]: def alpha_ell(phi,f):
            fp=np.sqrt(1.0-f**2)
            a1=np.sqrt(f)/fp*np.arcsinh(fp/f*np.cos(phi))
            a2=np.sqrt(f)/fp*np.arcsin(fp*np.sin(phi))
            return a1,a2

        def alpha_map(xk,yk,f):

            mappax=[]
            mappay=[]
            for i in range(xk.size):
                for j in range(yk.size):
                    phi=np.arctan2(yk[j],xk[i])
                    a1,a2=alpha_ell(phi,f)
```

```
            mappax.append(a1)
            mappay.append(a2)
    mappax=np.array(mappax)
    mappay=np.array(mappay)
    mappax=mappax.reshape([xk.size,yk.size],order='F')
    mappay=mappay.reshape([xk.size,yk.size],order='F')

    return np.array(mappax),np.array(mappay)

a1,a2=alpha_map(xk,yk,f)
fig,ax=plt.subplots(1,3,figsize=(27,8))
ax[0].imshow(a1,origin='low')
ax[1].imshow(a2,origin='low')
ax[2].imshow(np.sqrt(a1*a1+a2*a2),origin='low')
```

Out[2]: <matplotlib.image.AxesImage at 0x1023f0690>



The further step is the derivation of the shear components. These can be derived by means of the derivatives of the deflection angle:

$$\gamma_1 = \frac{1}{2}\left(\frac{\partial \alpha_1}{\partial x_1} - \frac{\partial \alpha_2}{\partial x_2}\right) \gamma_2 = \frac{\partial \alpha_1}{\partial x_2}$$

Using the differential operators in polar coordiantes and the results above, we find that

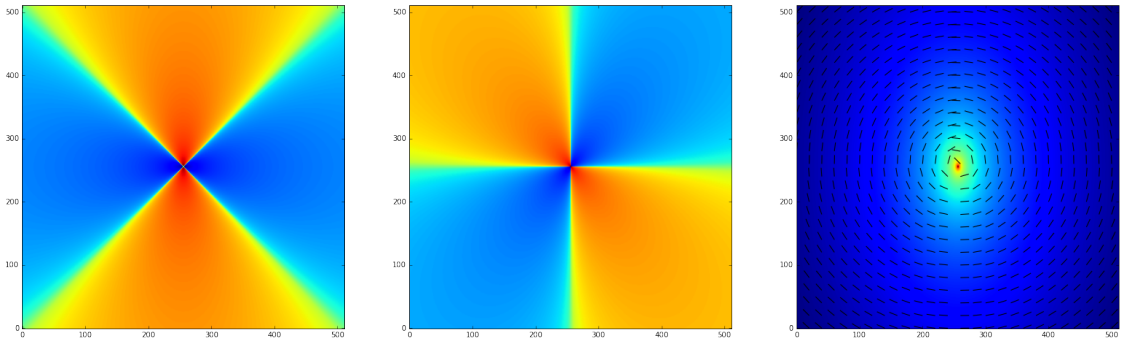
$$\gamma_1 = -\frac{\sqrt{f}}{2x\Delta(\varphi)} \cos 2\varphi = -\kappa \cos 2\varphi \gamma_2 = -\frac{\sqrt{f}}{2x\Delta(\varphi)} \sin 2\varphi = -\kappa \sin 2\varphi$$

or, in other words, $\gamma = \kappa$ as for the SIS. Shown below are the maps of the two components of the shear, as well as the map of its absolute value.

```
In [3]: def gamma_map(xk,yk,f):

            mappax=[]
            mappay=[]
            for i in range(xk.size):
                for j in range(yk.size):
                    phi=np.arctan2(yk[j],xk[i])
                    x=np.sqrt(xk[i]*xk[i]+yk[j]*yk[j])
                    ga1,ga2=gamma_ell(x,phi,f)
                    mappax.append(ga1)
                    mappay.append(ga2)
            mappax=np.array(mappax)
```

```
        mappay=np.array(mappay)
        mappax=mappax.reshape([xk.size,yk.size],order='F')
        mappay=mappay.reshape([xk.size,yk.size],order='F')

        return np.array(mappax),np.array(mappay)

    ga1,ga2=gamma_map(xk,yk,f)
    fig,ax=plt.subplots(1,3,figsize=(27,8))
    ax[0].imshow(ga1,origin='low',norm=SymLogNorm(linthresh=0.01))
    ax[1].imshow(ga2,origin='low',norm=SymLogNorm(linthresh=0.01))
    ax[2].imshow(np.sqrt(ga1*ga1+ga2*ga2),origin='low',norm=LogNorm())

    #shear pattern
    pixel_step=20
    x,y = np.meshgrid(np.arange(0,ga1.shape[1],pixel_step),np.arange(0,ga1.shape[0],pixel_step))

    #Translate shear components into sines and cosines
    cos_2_phi = -ga1 / np.sqrt(ga1**2 + ga2**2)
    sin_2_phi = ga2 / np.sqrt(ga1**2 + ga2**2)

    #Compute stick directions
    cos_phi = np.sqrt(0.5*(1.0 + cos_2_phi)) * np.sign(sin_2_phi)
    sin_phi = np.sqrt(0.5*(1.0 - cos_2_phi))

    #Fix ambiguity when sin_2_phi = 0
    cos_phi[sin_2_phi==0] = np.sqrt(0.5*(1.0 + cos_2_phi[sin_2_phi==0]))

    ax[2].quiver(x,y,cos_phi[x,y],sin_phi[x,y],headwidth=0,units="height",scale=x.shape[0],color="b
```

Out[3]: <matplotlib.quiver.Quiver at 0x110320f50>



Once we have the shear, we can compute the lensing Jacobian. After some math, we can see that

$$A = \begin{bmatrix} 1-\kappa-\gamma_1 & -\gamma_2 \\ \gamma_2 & 1-\kappa+\gamma_1 \end{bmatrix} = \begin{bmatrix} 1-2\kappa\sin^2\varphi & \kappa\sin 2\varphi \\ \kappa\sin 2\varphi & 1-2\kappa\cos^2\varphi \end{bmatrix}$$

and that

$$\lambda_t = 1-\kappa-\gamma = 1-2\kappa\lambda_r = 1-\kappa+\gamma = 1$$

It turns out that, as the SIS, the SIE does not have a critical line, being the radial magnification always unity. The tangential critical line is the ellipse where

$$\kappa = \frac{1}{2}$$

meaning that

$$\vec{x}_t(\varphi) = \frac{\sqrt{f}}{\Delta(\varphi)} \left[\cos\varphi, \sin\varphi\right] \ .$$

The critical line for the same lens used in the examples above is shown in the right panel below. This line can be mapped onto the source plane using the lens equation, to obtain the tangential caustic:

$$y_{t,1} = \frac{\sqrt{f}}{\Delta(\varphi)} \cos\varphi - \frac{\sqrt{f}}{f'} \text{arcsinh}\left(\frac{f'}{f}\cos\varphi\right)$$

$$y_{t,2} = \frac{\sqrt{f}}{\Delta(\varphi)} \sin\varphi - \frac{\sqrt{f}}{f'} \arcsin(f'\sin\varphi)$$

As for the SIS, we can now search for the $\{cut\}$*:*

$$\vec{y}_c = \lim_{x\to 0} \vec{y}(x,\varphi) = -\vec{\alpha}(\varphi)$$

Thus,

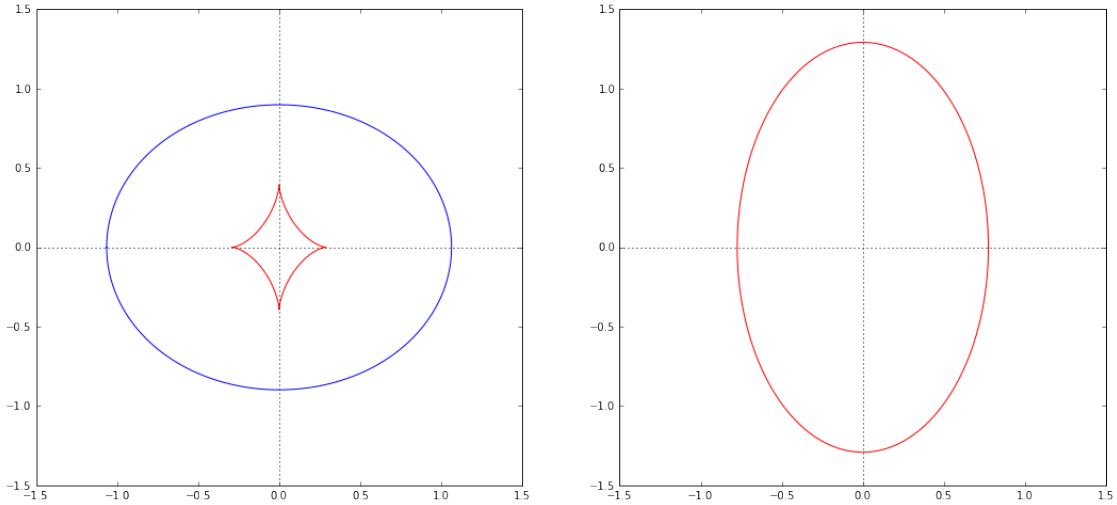
$$y_{t,1} = -\frac{\sqrt{f}}{f'} arcsinh\left(\frac{f'}{f}\cos\varphi\right)$$

$$y_{t,2} = -\frac{\sqrt{f}}{f'} \arcsin(f'\sin\varphi)$$

Both the caustic and the cut are shown below in the left panel (red and blue curves, respectively).

```
In [4]: def cut_ell(f,phi_min=0,phi_max=2.0*np.pi):
            phi=np.linspace(phi_min,phi_max,1000)
            y1,y2=alpha_ell(phi,f)
            return -y1,-y2

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

        def showaxes(ax):
            x=[-100,100]
            y=[0,0]
            xlim=ax.get_xlim()
            ylim=ax.get_ylim()
            ax.set_xlim(xlim)
            ax.set_ylim(ylim)
            ax.plot(x,y,':',color='black')
            ax.plot(y,x,':',color='black')
            return

        def tan_caustic(f,phi_min=0,phi_max=2.0*np.pi):
            phi=np.linspace(phi_min,phi_max,1000)
            delta=np.sqrt(np.cos(phi)**2+f**2*np.sin(phi)**2)
            a1,a2=alpha_ell(phi,f)
            y1=np.sqrt(f)/delta*np.cos(phi)-a1
            y2=np.sqrt(f)/delta*np.sin(phi)-a2
            return y1,y2
```

```
def tan_cc(f,phi_min=0,phi_max=2.0*np.pi):
    phi=np.linspace(phi_min,phi_max,1000)
    delta=np.sqrt(np.cos(phi)**2+f**2*np.sin(phi)**2)
    r=np.sqrt(f)/delta
    x1=r*np.cos(phi)
    x2=r*np.sin(phi)
    return(x1,x2)

fig,ax=plt.subplots(1,2,figsize=(18,8))
y1_cut,y2_cut=cut_ell(f)
y1_cau,y2_cau=tan_caustic(f)
x1_cc,x2_cc=tan_cc(f)

ax[0].set_xlim([-1.5,1.5])
ax[0].set_ylim([-1.5,1.5])
ax[1].set_xlim([-1.5,1.5])
ax[1].set_ylim([-1.5,1.5])
ax[0].plot(y1_cut,y2_cut,'-',color='blue')
ax[0].plot(y1_cau,y2_cau,'-',color='red')
ax[1].plot(x1_cc,x2_cc,'-',color='red')
showaxes(ax[0])
showaxes(ax[1])
```



The cut and the caustic intercept the y_1 and the y_2 axes in points that are symmetric with respect to the center of the lens. These points have coordinates

$$y_{c,1}(\varphi = 0, \pi) = \pm s_{1,c} y_{c,2} = 0$$

and

$$y_{c,1} = 0 y_{c,2}(\varphi = \pi/2, -\pi/2) = \pm s_{2,c}$$

for the cut, and

$$y_{t,1}(\varphi = 0, \pi) = \pm s_{1,t} y_{t,2} = 0$$

and

$$y_{t,1} = 0 y_{t,2}(\varphi = \pi/2, -\pi/2) = \pm s_{2,t}$$

for the caustic.
The plot below shows s_1 *and* s_2 *vary as a function of* f *for both the cut and the caustic. Clearly*

$$s_{1,c} > s_{1,t}$$

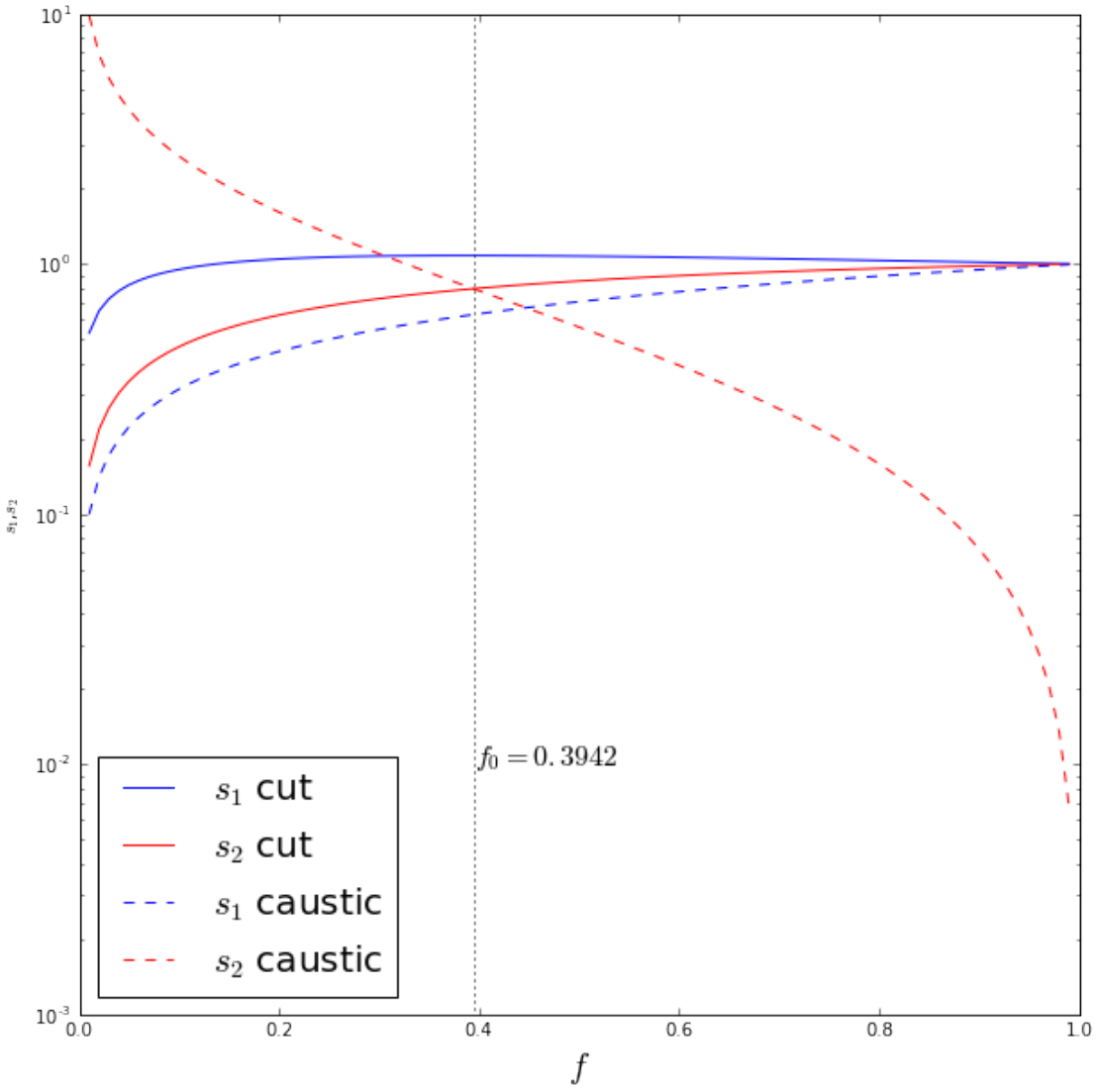
for any f*, while there exists a value* $f = f_0 = 0.3942$ *such that*

$$s_{2,c} \leq s_{2,t} \text{ for } f \leq f_0 s_{2,c} > s_{2,t} \text{ for } f > f_0$$

```
In [5]: def cut_intercept_y1(f):
            s1,y2_tmp=alpha_ell(0.0,f)
            return s1

        def cut_intercept_y2(f):
            y1_tmp,s1=alpha_ell(np.pi/2.0,f)
            return s1

        def cau_intercept_y1(f):
            a1,a2=alpha_ell(np.pi/2.0,f)
            s1=np.sqrt(f)-a1
            return s1

        def cau_intercept_y2(f):
            a1,a2=alpha_ell(np.pi/2.0,f)
            s1=np.sqrt(f)/f-a2
            return s1

        f_arr=np.linspace(0.01,0.99,100)
        s1_cut=cut_intercept_y1(f_arr)
        s2_cut=cut_intercept_y2(f_arr)
        s1_cau=cau_intercept_y1(f_arr)
        s2_cau=cau_intercept_y2(f_arr)

        print s1_cut.size, s2_cut.size

        fig,ax=plt.subplots(1,1,figsize=(10,10))
        ax.plot(f_arr,s1_cut,'-',color='blue',label='$s_1$ cut')
        ax.plot(f_arr,s2_cut,'-',color='red',label='$s_2$ cut')
        ax.plot(f_arr,s1_cau,'--',color='blue',label='$s_1$ caustic')
        ax.plot(f_arr,s2_cau,'--',color='red',label='$s_2$ caustic')
        ylim=ax.get_ylim()
        ax.plot([0.3942,0.3942],ylim,':',color='black')
        ax.set_yscale('log')
        ax.legend(loc='best',fontsize=20)
        ax.text(0.3942,1e-2,'$f_0=0.3942$',fontsize=16)
        ax.set_xlabel('$f$',fontsize=20)
        ax.set_ylabel('$s_1,s_2$')
```

```
100 100
```

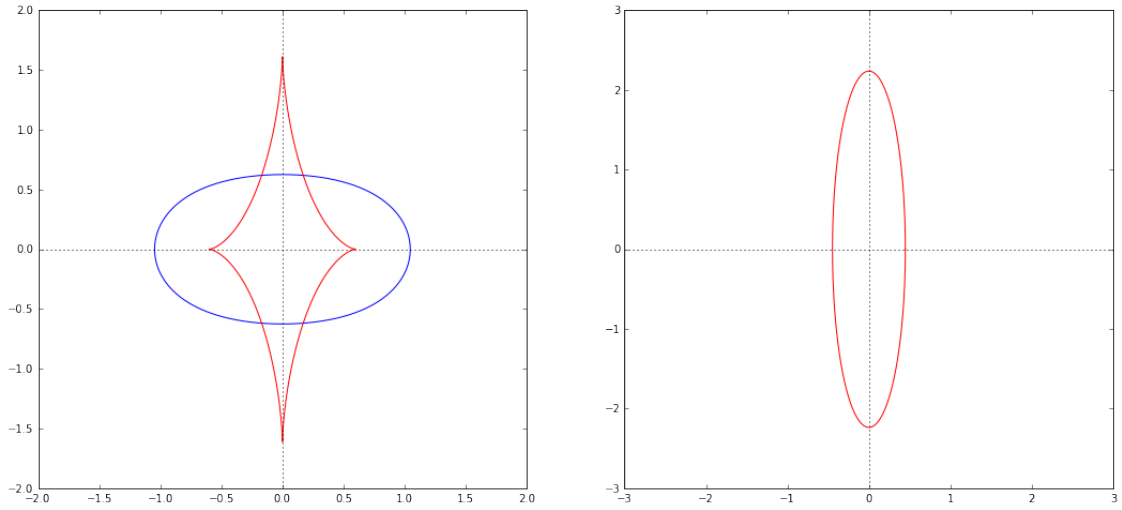
```
Out[5]: <matplotlib.text.Text at 0x10f93a850>
```



Thus, when the ellipticity exceeds a certain value, the caustic is no longer fully contained into the cut. This will impact the image multiplicity, as it will be discussed below.

```
In [6]: f_small=0.2
        fig,ax=plt.subplots(1,2,figsize=(18,8))
        y1_cut,y2_cut=cut_ell(f_small)
        y1_cau,y2_cau=tan_caustic(f_small)
        x1_cc,x2_cc=tan_cc(f_small)

        ax[0].set_xlim([-2,2])
        ax[0].set_ylim([-2,2])
        ax[1].set_xlim([-3,3])
        ax[1].set_ylim([-3,3])
        ax[0].plot(y1_cut,y2_cut,'-',color='blue')
        ax[0].plot(y1_cau,y2_cau,'-',color='red')
        ax[1].plot(x1_cc,x2_cc,'-',color='red')
```

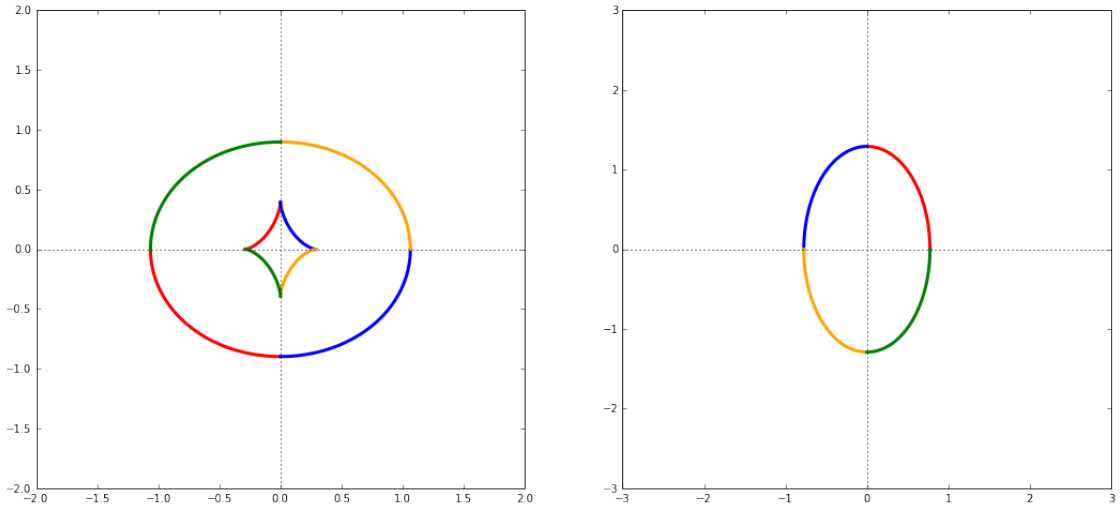
```
showaxes(ax[0])
showaxes(ax[1])
```



It is important to note how the mapping between the lens and the source plance takes place:

```
In [7]: f=0.6
        # first quadrant in the lens plane
        fig,ax=plt.subplots(1,2,figsize=(18,8))
        y1_cut,y2_cut=cut_ell(f,0.,np.pi/2.0)
        y1_cau,y2_cau=tan_caustic(f,0.,np.pi/2.0)
        x1_cc,x2_cc=tan_cc(f,0.,np.pi/2.0)

        ax[0].plot(y1_cut,y2_cut,'-',color='red',lw=3)
        ax[0].plot(y1_cau,y2_cau,'-',color='red',lw=3)
        ax[1].plot(x1_cc,x2_cc,'-',color='red',lw=3)

        # second quadrant in the lens plane
        y1_cut,y2_cut=cut_ell(f,np.pi/2.0,np.pi)
        y1_cau,y2_cau=tan_caustic(f,np.pi/2.0,np.pi)
        x1_cc,x2_cc=tan_cc(f,np.pi/2.0,np.pi)

        ax[0].plot(y1_cut,y2_cut,'-',color='blue',lw=3)
        ax[0].plot(y1_cau,y2_cau,'-',color='blue',lw=3)
        ax[1].plot(x1_cc,x2_cc,'-',color='blue',lw=3)

        # third quadrant in the lens plane
        y1_cut,y2_cut=cut_ell(f,np.pi,1.5*np.pi)
        y1_cau,y2_cau=tan_caustic(f,np.pi,1.5*np.pi)
        x1_cc,x2_cc=tan_cc(f,np.pi,1.5*np.pi)

        ax[0].plot(y1_cut,y2_cut,'-',color='orange',lw=3)
        ax[0].plot(y1_cau,y2_cau,'-',color='orange',lw=3)
        ax[1].plot(x1_cc,x2_cc,'-',color='orange',lw=3)

        # fourth quadrant in the lens plane
        y1_cut,y2_cut=cut_ell(f,1.5*np.pi,2*np.pi)
```

```
y1_cau,y2_cau=tan_caustic(f,1.5*np.pi,2*np.pi)
x1_cc,x2_cc=tan_cc(f,1.5*np.pi,2*np.pi)

ax[0].plot(y1_cut,y2_cut,'-',color='green',lw=3)
ax[0].plot(y1_cau,y2_cau,'-',color='green',lw=3)
ax[1].plot(x1_cc,x2_cc,'-',color='green',lw=3)



ax[0].set_xlim([-2,2])
ax[0].set_ylim([-2,2])
ax[1].set_xlim([-3,3])
ax[1].set_ylim([-3,3])

showaxes(ax[0])
showaxes(ax[1])
```



Note the left-right mapping rule for sources near the caustic and the diagonal rule for sources near the cut (of course assuming that the lens major axis is oriented along the x_2 axis). This is important for understanding the image parity rule and to understand how to locate counter images.

2 Multiple images

Kovner, Bartelmann & Schneider (1994) discuss an approach to find the images of a source lensed by the SIE. Beginning with the lens equation, and multiplying y_1 by $\cos\varphi$ and y_2 by $\sin\varphi$, the lens equation can be reduced to the one-dimensional equation

$$x = y_1 \cos\varphi + y_2 \sin\varphi + \tilde{\psi}(\varphi)$$

By reinserting this equation into the lens equation, we obtain

$$F(\varphi) = \left[y_1 + \frac{\sqrt{f}}{f'} arcsinh\left(\frac{f'}{f} \cos\varphi \right) \right] \sin\varphi - \left[y_2 + \frac{\sqrt{f}}{f'} \arcsin(f' \sin\varphi) \right] \cos\varphi = 0$$

Now the problem of finding the images of the source at (y_1, y_2) reduces to the problem of finding the zeros of $F(\varphi)$. Indeed, once φ has been determined, it can be inserted in the equation above to obtain x.

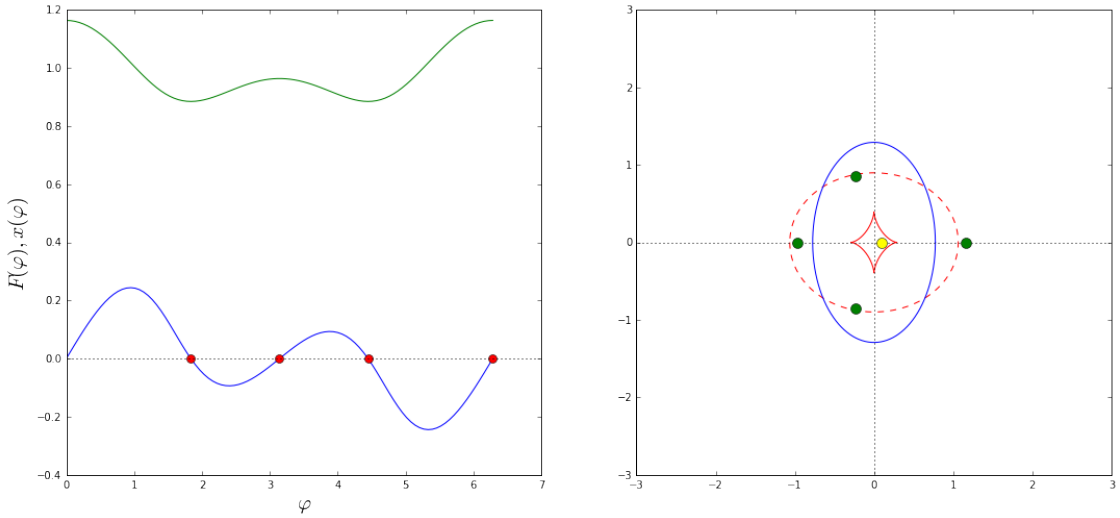
The solutions cannot be found analytically: a root finding algorithm such as the Brent's method must be employed.

```
In [8]: from scipy.optimize import fsolve, newton,brenth, brentq
        from sympy import *
        import math

        def x_ima(y1,y2,phi,f):
            x=y1*np.cos(phi)+y2*np.sin(phi)+(psi_tilde(phi,f))
            return x

        def phi_ima(y1,y2,f,checkplot=True,verbose=True):
            def phi_func(phi):
                a1,a2=alpha_ell(phi,f)
                func=(y1+a1)*np.sin(phi)-(y2+a2)*np.cos(phi)
                return func

            U=np.linspace(0.,2.0*np.pi+0.001,100)
            c = phi_func(U)
            s = np.sign(c)
            phi=[]
            xphi=[]
            for i in range(100-1):
                if s[i] + s[i+1] == 0: # opposite signs
                    u = brentq(phi_func, U[i], U[i+1])
                    z = phi_func(u)
                    if np.isnan(z) or abs(z) > 1e-3:
                        continue
                    x=x_ima(y1,y2,u,f)
                    if (x>0):
                        phi.append(u)
                        xphi.append(x)
                    if (verbose):
                        print('found zero at {}'.format(u))
                        if (x<0):
                            print 'discarded because x is negative ({})'.format(x)
                        else:
                            print 'accepted because x is positive ({})'.format(x)
            if (checkplot):
                phi_=np.linspace(0.,2.0*np.pi,100)
                ax[0].plot(phi_,phi_func(phi_),label=r'£F(\varphi)')
                ax[0].plot(phi_,x_ima(y1,y2,phi_,f),label=r'£x(\varphi)')
                #ax[0].plot(phi_,psi_tilde(phi_,f)-1)
                ax[0].plot(phi,phi_func(phi),'o',markersize=8)
                ax[0].set_xlabel(r'£\varphi£',fontsize=20)
                ax[0].set_ylabel(r'£F(\varphi),x(\varphi)£',fontsize=20)

            return(np.array(xphi),np.array(phi))

        fig,ax=plt.subplots(1,2,figsize=(18,8))

        y1=0.1
        y2=0.0
        f=0.6
```

```
x,phi=phi_ima(y1,y2,f)
#x=x_ima(y1,y2,phi,f)

x1_ima=x*np.cos(phi)
x2_ima=x*np.sin(phi)

y1_cut,y2_cut=cut_ell(f)
y1_cau,y2_cau=tan_caustic(f)
x1_cc,x2_cc=tan_cc(f)

ax[1].set_xlim([-3,3])
ax[1].set_ylim([-3,3])
ax[1].plot(y1_cut,y2_cut,'--',color='red')
ax[1].plot(y1_cau,y2_cau,'-',color='red')
ax[1].plot(x1_cc,x2_cc,'-',color='blue')
showaxes(ax[0])
showaxes(ax[1])

ax[1].plot(y1,y2,'o',markersize=10,color='yellow')
ax[1].plot(x1_ima,x2_ima,'o',markersize=10,color='green')
```

```
found zero at 1.83632144709
accepted because x is positive (0.884932671162)
found zero at 3.14159265359
accepted because x is positive (0.963726774488)
found zero at 4.44686386009
accepted because x is positive (0.884932671162)
found zero at 6.28318530718
accepted because x is positive (1.16372677449)
```

Out[8]: [<matplotlib.lines.Line2D at 0x119a1ab10>]

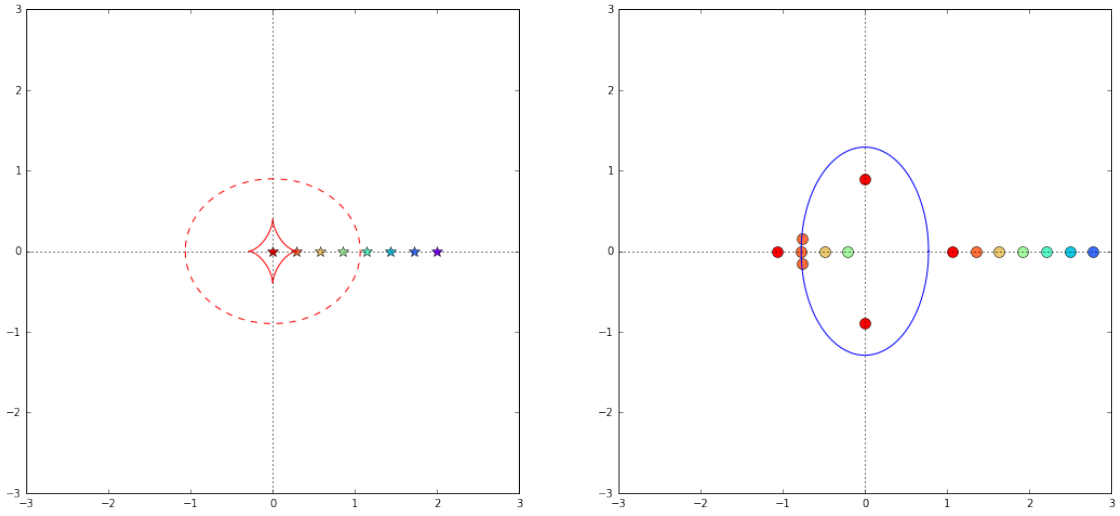


The two curves in the left panel show the functions $F(\varphi)$ and $x(\varphi; y_1, y_2)$. The zeros of F are marked in red and the corresponding images are shown in the right panel with green circles. Shown are also the caustic (solid-red), the cut (dashed-red) and the critical line (solid-blue). The source position is indicated by the yellow dot.

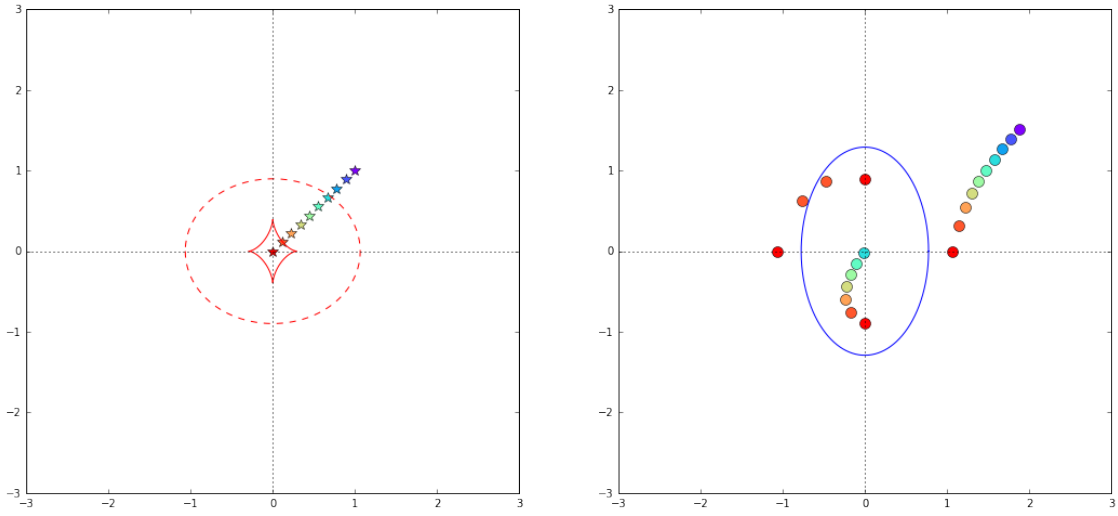
Note that some zeros of F are actually spurious solutions and must be rejected. We recongnize them by means of the sign of x: whenever it is negative, we reject the solution.

We can now study the multiple image configurations arising from sources at different positions. In the following examples, we consider the case $f > f_0$. We show below the images of sources at different positions, including sources on the cusps and on the folds of the caustic.

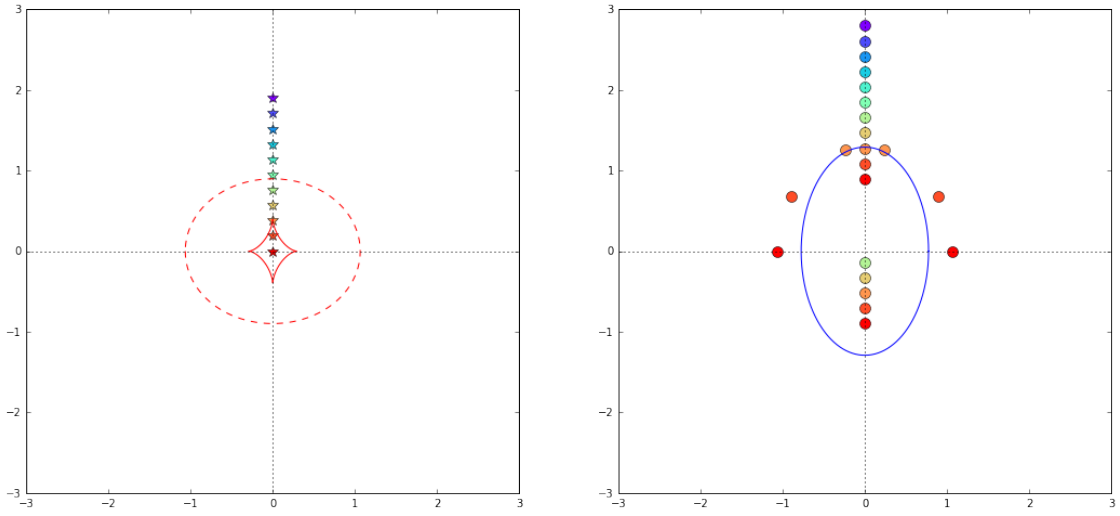
In [9]:
```
from matplotlib.pyplot import cm

fig,ax=plt.subplots(1,2,figsize=(18,8))

# f>f0
f=0.6

# source on the y1 axis, moving from outside the cut towards the center of the lens
y1=np.linspace(2,0,8)
y2=np.zeros(y1.size)

color=iter(cm.rainbow(np.linspace(0,1,y1.size)))
for i in range(y1.size):
    c=next(color)
    x,phi=phi_ima(y1[i],y2[i],f,checkplot=False,verbose=False)

    x1_ima=x*np.cos(phi)
    x2_ima=x*np.sin(phi)

    ax[0].plot(y1[i],y2[i],'*',markersize=10,color=c)
    ax[1].plot(x1_ima,x2_ima,'o',markersize=10,color=c)

y1_cut,y2_cut=cut_ell(f)
y1_cau,y2_cau=tan_caustic(f)
x1_cc,x2_cc=tan_cc(f)

ax[1].set_xlim([-3,3])
ax[1].set_ylim([-3,3])
ax[0].set_xlim([-3,3])
ax[0].set_ylim([-3,3])
ax[0].plot(y1_cut,y2_cut,'--',color='red')
ax[0].plot(y1_cau,y2_cau,'-',color='red')
ax[1].plot(x1_cc,x2_cc,'-',color='blue')
showaxes(ax[0])
showaxes(ax[1])
```



In [10]:
```
fig,ax=plt.subplots(1,2,figsize=(18,8))

# f>f0
f=0.6

# source on the diagonal, moving from outside the cut towards the center of the lens
y1=np.linspace(1.0,0,10)
y2=y1

color=iter(cm.rainbow(np.linspace(0,1,y1.size)))
for i in range(y1.size):
    c=next(color)
    x,phi=phi_ima(y1[i],y2[i],f,checkplot=False,verbose=False)

    x1_ima=x*np.cos(phi)
    x2_ima=x*np.sin(phi)

    ax[0].plot(y1[i],y2[i],'*',markersize=10,color=c)
    ax[1].plot(x1_ima,x2_ima,'o',markersize=10,color=c)

y1_cut,y2_cut=cut_ell(f)
y1_cau,y2_cau=tan_caustic(f)
x1_cc,x2_cc=tan_cc(f)

ax[1].set_xlim([-3,3])
ax[1].set_ylim([-3,3])
ax[0].set_xlim([-3,3])
ax[0].set_ylim([-3,3])
ax[0].plot(y1_cut,y2_cut,'--',color='red')
ax[0].plot(y1_cau,y2_cau,'-',color='red')
ax[1].plot(x1_cc,x2_cc,'-',color='blue')
showaxes(ax[0])
showaxes(ax[1])
```



In [11]:
```
fig,ax=plt.subplots(1,2,figsize=(18,8))

# f>f0
f=0.6

# source on the y2 axis, moving from outside the cut towards the center of the lens
y2=np.linspace(1.9,0,11)
y1=np.zeros(y2.size)


color=iter(cm.rainbow(np.linspace(0,1,y1.size)))
for i in range(y1.size):
    c=next(color)
    x,phi=phi_ima(y1[i],y2[i],f,checkplot=False,verbose=False)

    x1_ima=x*np.cos(phi)
    x2_ima=x*np.sin(phi)

    ax[0].plot(y1[i],y2[i],'*',markersize=10,color=c)
    ax[1].plot(x1_ima,x2_ima,'o',markersize=10,color=c)

y1_cut,y2_cut=cut_ell(f)
y1_cau,y2_cau=tan_caustic(f)
x1_cc,x2_cc=tan_cc(f)

ax[1].set_xlim([-3,3])
ax[1].set_ylim([-3,3])
ax[0].set_xlim([-3,3])
ax[0].set_ylim([-3,3])
ax[0].plot(y1_cut,y2_cut,'--',color='red')
ax[0].plot(y1_cau,y2_cau,'-',color='red')
ax[1].plot(x1_cc,x2_cc,'-',color='blue')
showaxes(ax[0])
showaxes(ax[1])
```



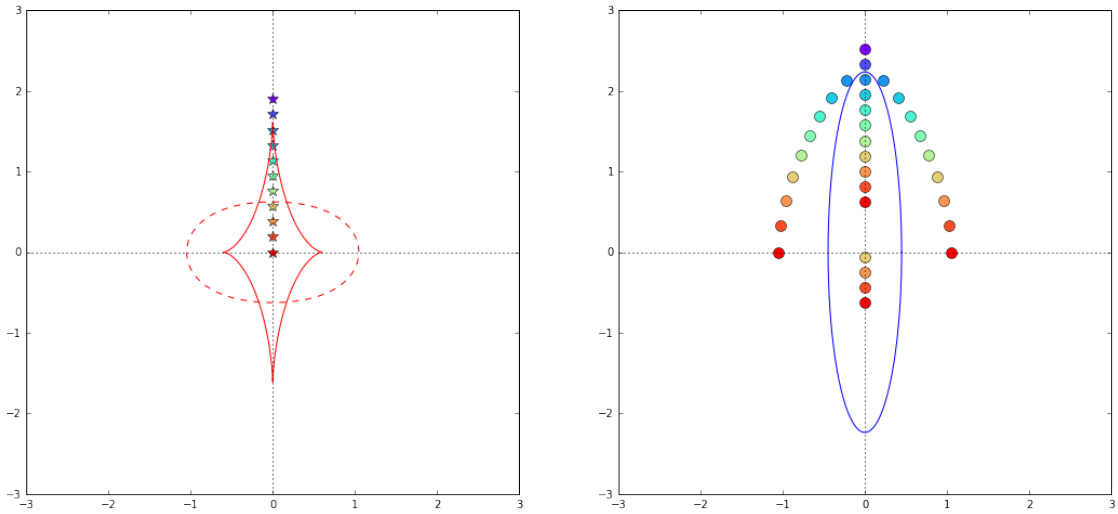
The examples shown so far illustrate that in the case $f < f_0$:

- *a source outside the cut always produces one image;*
- *inside the cut, but outside the caustic, it produces two images. The image multiplicity increases by one only because of the singularity at the center of the lens*
- *inside the caustic, it produces four images (thus, the multiplicity changes by two when crossing the caustic)*

For $f > f_0$, the upper and the lower cusps of the caustic become "naked". The regions outside the cut but inside the caustic have multiplicity three. In this case, the lens can produce one, two, three, or four images. Check this out in the example below.

```
In [12]: fig,ax=plt.subplots(1,2,figsize=(18,8))

         # f<f0
         f=0.2

         # source on the y2 axis, moving from outside the cut towards the center of the lens
         y2=np.linspace(1.9,0,11)
         y1=np.zeros(y2.size)


         color=iter(cm.rainbow(np.linspace(0,1,y1.size)))
         for i in range(y1.size):
             c=next(color)
             x,phi=phi_ima(y1[i],y2[i],f,checkplot=False,verbose=False)

             x1_ima=x*np.cos(phi)
             x2_ima=x*np.sin(phi)

             ax[0].plot(y1[i],y2[i],'*',markersize=10,color=c)
             ax[1].plot(x1_ima,x2_ima,'o',markersize=10,color=c)

         y1_cut,y2_cut=cut_ell(f)
```

```
y1_cau,y2_cau=tan_caustic(f)
x1_cc,x2_cc=tan_cc(f)

ax[1].set_xlim([-3,3])
ax[1].set_ylim([-3,3])
ax[0].set_xlim([-3,3])
ax[0].set_ylim([-3,3])
ax[0].plot(y1_cut,y2_cut,'--',color='red')
ax[0].plot(y1_cau,y2_cau,'-',color='red')
ax[1].plot(x1_cc,x2_cc,'-',color='blue')
showaxes(ax[0])
showaxes(ax[1])
```



Some comments on the magnification and parity:

- μ *is*

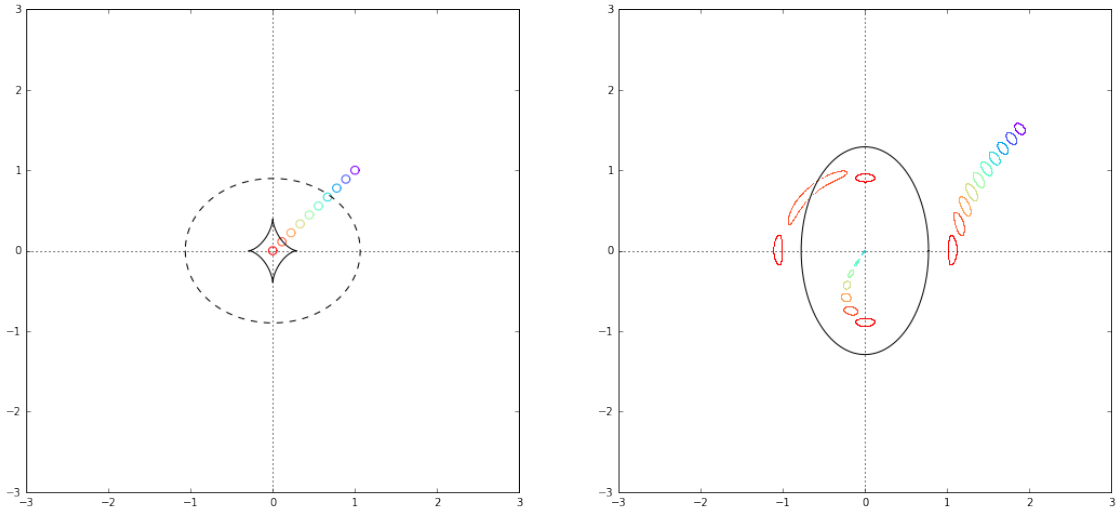
$$\mu = \frac{1}{1-2\kappa} ,$$

meaning that $\mu > 0$ *for* $\kappa < 0.5$. *Therefore, the parity of the images is positive outside the critical line, negative otherwise. * the magnification of the images forming near the center of the lens is very small* $|\mu| \sim 0$, *being* κ *divergent * the magnification of the images near the critical line is divergent, of course...*

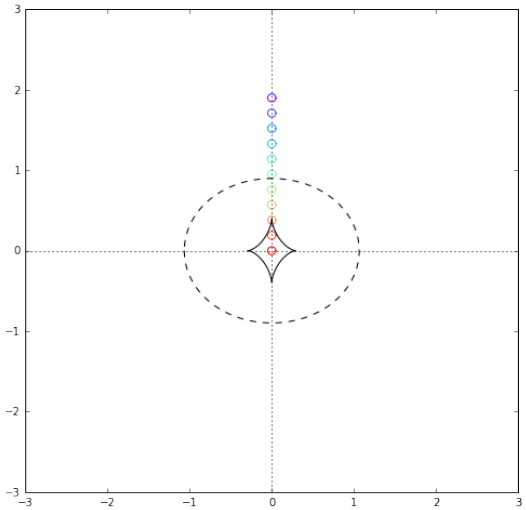
Magnification and distortion are better understood using extended sources. In the examples below, we implement the reconstruction of the images of a circular source. The code can be changed to see how the results are affected by the size of the source or by its shape.

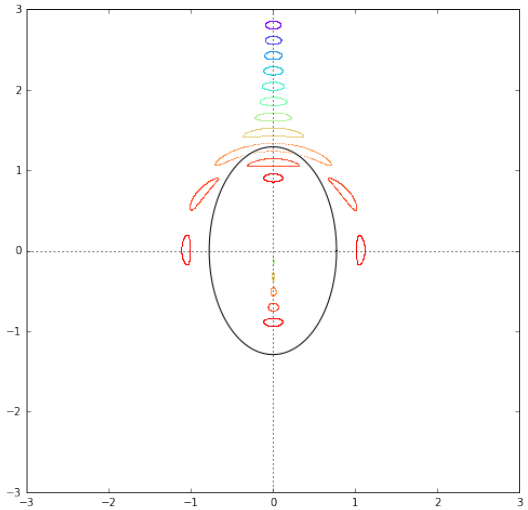
In [13]:
```
# extended source (y1c=y2c)

def ext_source(yc1,yc2,rs):
    phi=np.linspace(0,2.*np.pi,360)
    y1=yc1+rs*np.cos(phi)
    y2=yc2+rs*np.sin(phi)
    return(y1,y2)
```

```
fig,ax=plt.subplots(1,2,figsize=(18,8))

y1c=np.linspace(1.0,0,10)
y2c=y1c
rs=0.05
f=0.6

color=iter(cm.rainbow(np.linspace(0,1,y1c.size)))


for j in range(y1c.size):
    c=next(color)
    y1,y2=ext_source(y1c[j],y2c[j],rs)
    ax[0].plot(y1,y2,'-',color=c)

    for i in range(y1.size):
        x,phi=phi_ima(y1[i],y2[i],f,checkplot=False,verbose=False)
        x1=x*np.cos(phi)
        x2=x*np.sin(phi)
        ax[1].plot(x1,x2,',',color=c)

y1_cut,y2_cut=cut_ell(f)
y1_cau,y2_cau=tan_caustic(f)
x1_cc,x2_cc=tan_cc(f)

ax[1].set_xlim([-3,3])
ax[1].set_ylim([-3,3])
ax[0].set_xlim([-3,3])
ax[0].set_ylim([-3,3])

ax[0].plot(y1_cut,y2_cut,'--',color='black')
ax[0].plot(y1_cau,y2_cau,'-',color='black')
ax[1].plot(x1_cc,x2_cc,'-',color='black')
showaxes(ax[0])
showaxes(ax[1])
```

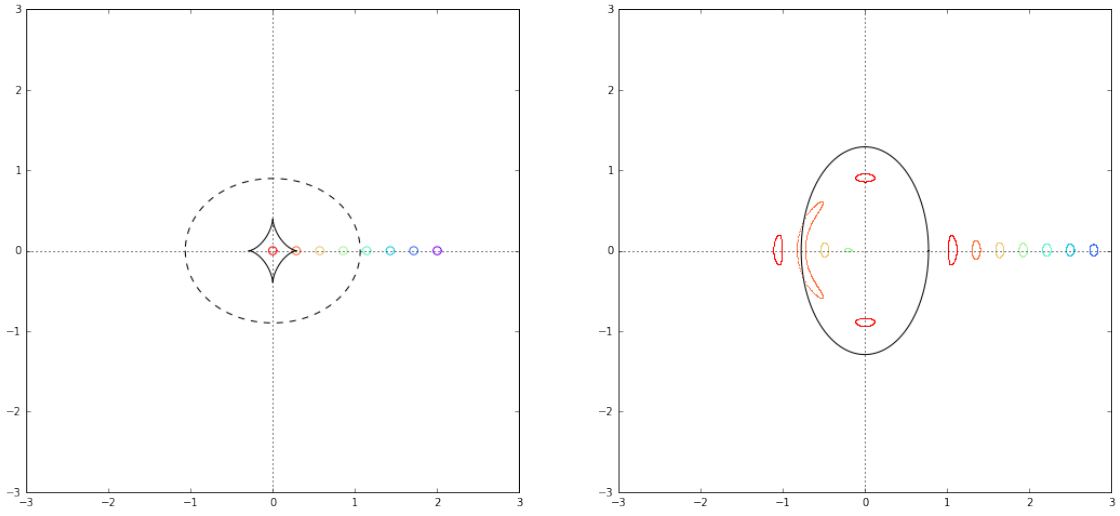


In [14]:
```
# extended source (y1c=0)

fig,ax=plt.subplots(1,2,figsize=(18,8))


y2c=np.linspace(1.9,0,11)
y1c=np.zeros(y2c.size)

color=iter(cm.rainbow(np.linspace(0,1,y1c.size)))



for j in range(y1c.size):
    c=next(color)
    y1,y2=ext_source(y1c[j],y2c[j],0.05)
    ax[0].plot(y1,y2,'-',color=c)

    for i in range(y1.size):
        x,phi=phi_ima(y1[i],y2[i],f,checkplot=False,verbose=False)
        x1=x*np.cos(phi)
        x2=x*np.sin(phi)
        ax[1].plot(x1,x2,',',color=c)


ax[1].set_xlim([-3,3])
ax[1].set_ylim([-3,3])
ax[0].set_xlim([-3,3])
ax[0].set_ylim([-3,3])

ax[0].plot(y1_cut,y2_cut,'--',color='black')
ax[0].plot(y1_cau,y2_cau,'-',color='black')
ax[1].plot(x1_cc,x2_cc,'-',color='black')
showaxes(ax[0])
showaxes(ax[1])
```

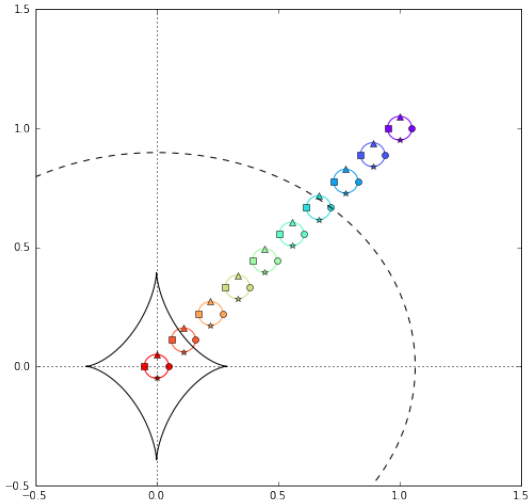


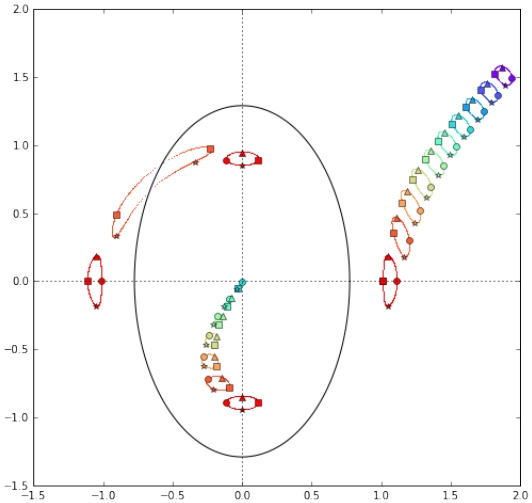


In [15]:
```
# extended source (y2c=0)

fig,ax=plt.subplots(1,2,figsize=(18,8))


y1c=np.linspace(2,0,8)
y2c=np.zeros(y1c.size)

color=iter(cm.rainbow(np.linspace(0,1,y1c.size)))



for j in range(y1c.size):
    c=next(color)
    y1,y2=ext_source(y1c[j],y2c[j],0.05)
    ax[0].plot(y1,y2,'-',color=c)

    for i in range(y1.size):
        x,phi=phi_ima(y1[i],y2[i],f,checkplot=False,verbose=False)
        x1=x*np.cos(phi)
        x2=x*np.sin(phi)
        ax[1].plot(x1,x2,',',color=c)


ax[1].set_xlim([-3,3])
ax[1].set_ylim([-3,3])
ax[0].set_xlim([-3,3])
ax[0].set_ylim([-3,3])

ax[0].plot(y1_cut,y2_cut,'--',color='black')
ax[0].plot(y1_cau,y2_cau,'-',color='black')
ax[1].plot(x1_cc,x2_cc,'-',color='black')
showaxes(ax[0])
showaxes(ax[1])
```



```
In [16]: def source_spots(yc1,yc2,rs):
             phi=np.array([0,np.pi/2.,np.pi,1.5*np.pi])
             y1=yc1+rs*np.cos(phi)
             y2=yc2+rs*np.sin(phi)
             return(y1,y2)

         fig,ax=plt.subplots(1,2,figsize=(18,8))

         y1c=np.linspace(1.0,0,10)
         y2c=y1c
         rs=0.05
         f=0.6

         color=iter(cm.rainbow(np.linspace(0,1,y1c.size)))


         for j in range(y1c.size):
             c=next(color)
             y1,y2=ext_source(y1c[j],y2c[j],rs)
             y1s,y2s=source_spots(y1c[j],y2c[j],rs)
             markers=['o','^','s','*']
             ax[0].plot(y1,y2,'-',color=c)
             for i in range(len(markers)):
                 ax[0].plot(y1s[i],y2s[i],str(markers[i]),color=c)

             for i in range(y1.size):
                 x,phi=phi_ima(y1[i],y2[i],f,checkplot=False,verbose=False)
                 x1=x*np.cos(phi)
                 x2=x*np.sin(phi)
                 ax[1].plot(x1,x2,',',color=c)

             for i in range(y1s.size):
                 x,phi=phi_ima(y1s[i],y2s[i],f,checkplot=False,verbose=False)
                 x1=x*np.cos(phi)
                 x2=x*np.sin(phi)
                 ax[1].plot(x1,x2,str(markers[i]),color=c)


         y1_cut,y2_cut=cut_ell(f)
         y1_cau,y2_cau=tan_caustic(f)
         x1_cc,x2_cc=tan_cc(f)

         ax[1].set_xlim([-1.5,2])
         ax[1].set_ylim([-1.5,2])
         ax[0].set_xlim([-0.5,1.5])
         ax[0].set_ylim([-0.5,1.5])

         ax[0].plot(y1_cut,y2_cut,'--',color='black')
         ax[0].plot(y1_cau,y2_cau,'-',color='black')
         ax[1].plot(x1_cc,x2_cc,'-',color='black')
         showaxes(ax[0])
         showaxes(ax[1])
```


1.5
1.0
0.5
0.0
−0.5
−0.5
0.0
0.5
1.0
1.5


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

In [ ]: