Introduction to Gravitational Lensing

With python examples

Massimo Meneghetti

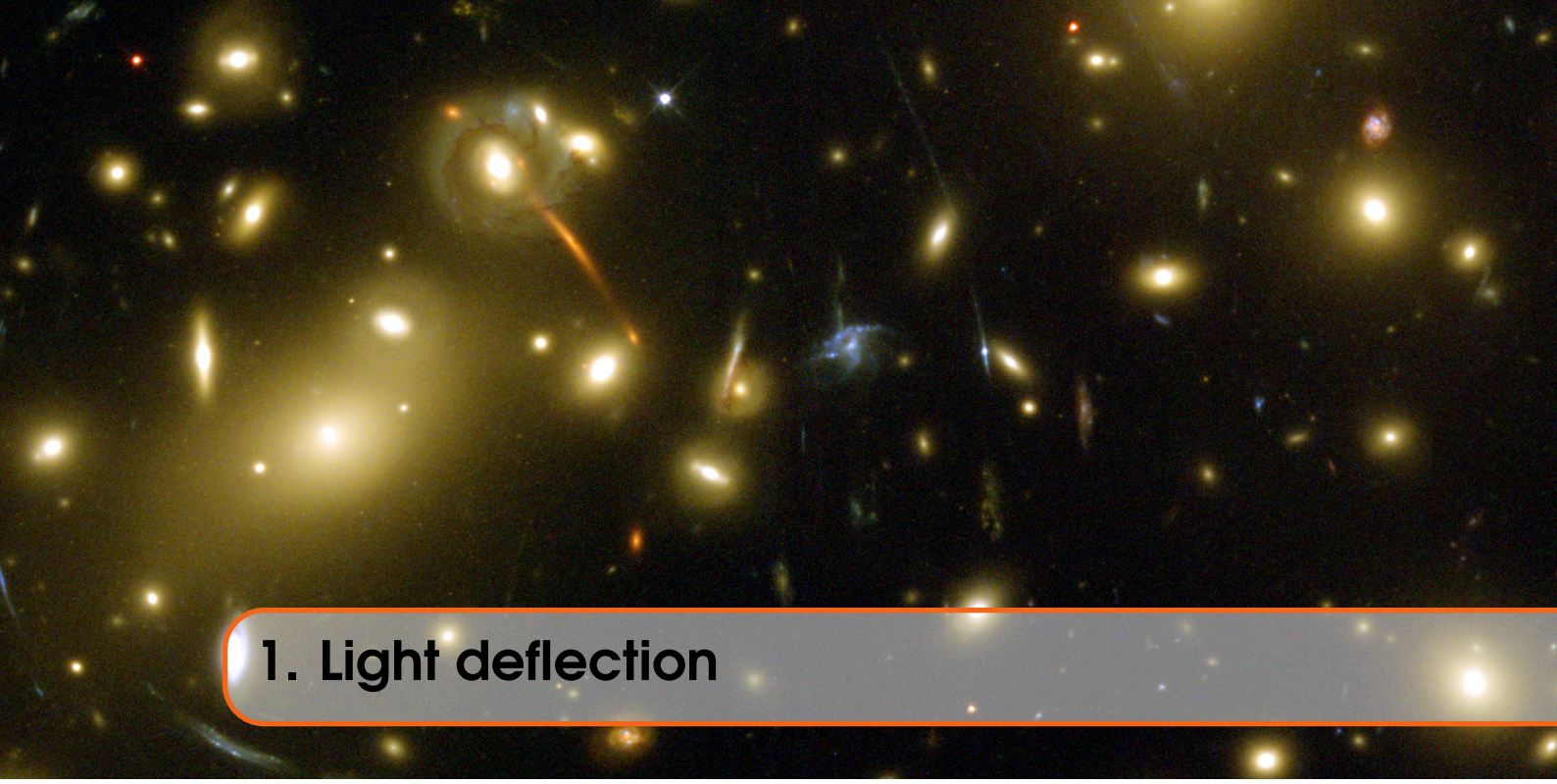
PUBLISHED BY MASSIMO MENEGHETTI

HTTP://PICO.BO.ASTRO.IT/ MASSIMO/TEACHING.HTML

First printing, March 2017

Contents

Part One: Generalities



1. Light deflection

1.1 Deflection of a light corpuscle

The idea that light could be bent by gravity was mentioned by Isaac Newton in a note at the end of *Optiks*, pusblished in 1704. Further calculations were made about a century later by the German astronomer Johann Georg Von Soldner (1776-1833), who ended up quantifying that the deflection of a photon grazing the surface of the sun would amount to about 0.9".

What were the assumptions under which this result was obtained? We should first of all introduce the framework within which the idea was proposed. This is the so called "Corpuscolar Theory of Light", which assumes that photons are not mass-less.

In this framework, the derivation of the deflection angle of a photon by a body with mass M is rather straightforward. It can be done in many ways, but we re-propose here a simple calculation by Victor J. Stenger (2013), which is based on four ingredients:

- Newton's law of gravity;
- Newton's second law of motion;
- Einstein's principle of equivalence;
- Einstein's special relativity.

Newton's law of gravity says that the gravitational force between two bodies with masses m and M is

$$\vec{F} = \frac{GmM}{r^3}\vec{r}\,, \tag{1.1}$$

where r is the distance between the bodies, and G is the gravitational constant.

Newton's second law of motion states that

$$\vec{F} = \frac{d\vec{p}}{dt} = m\vec{a} \tag{1.2}$$

where $\vec{p}$ and a are the momentum and the acceleration of the body with inertial mass m, respectively.

Because of the principle of equivalence, the gravitational mass m in Eq. 1.1 equals the inertial mass m in Eq. 1.2.

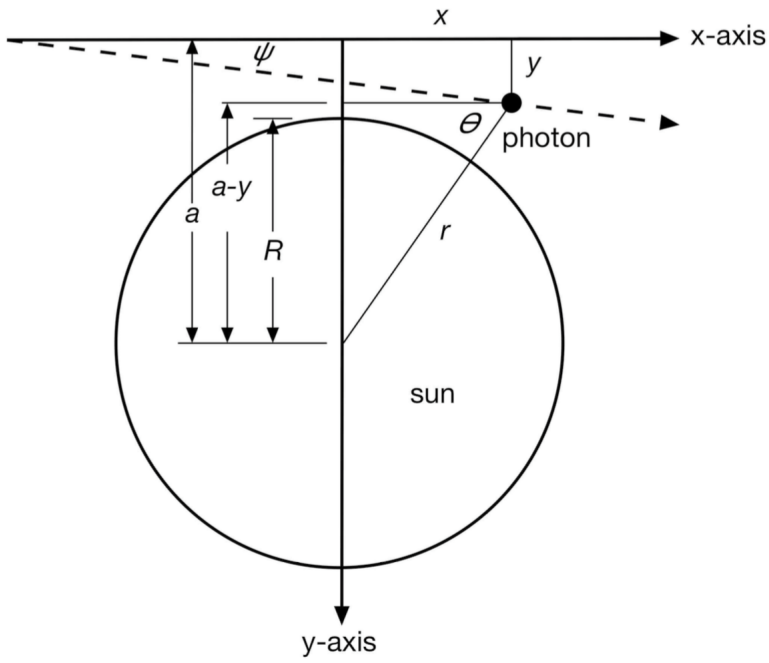


Figure 1.1.1: *Schematic view of a photon grazing the surface of the Sun (from V. J. Stenger, 2013).*

From Einstein's special relativity, we have that the inertial mass of a photon with energy E is E/c^2, where c is the speed of light.

Let assume that a photon with initial momentum $\vec{p}$ grazes the surface of the Sun, as shown in Fig. 1.1.1. The photon travels along the x-axis, while the y-axis was chosen to pass through the center of the sun, whose mass is M and whose radius is R. Let a be the impact parameter of the photon, i.e. the minimal distance of the un-deflected trajectory of the photon from the center of the Sun. When the photon is at the position (x,y), the distance from the Sun is

$$r = \sqrt{x^2 + (a-y)^2}\,. \tag{1.3}$$

Let's assume that the momentum of the photon does not change significantly along its path. The components of the gravitational force acting on the photon are

$$\begin{aligned} F_x = \frac{dp}{dt}\cos\theta &= \frac{GMp}{c[x^2+(a-y)^2]}\cos\theta = \frac{GMp}{c}\frac{x}{[x^2+(a-y)^2]^{3/2}}\,, \\ F_y = \frac{dp}{dt}\sin\theta &= \frac{GMp}{c[x^2+(a-y)^2]}\sin\theta = \frac{GMp}{c}\frac{a-y}{[x^2+(a-y)^2]^{3/2}}\,. \end{aligned} \tag{1.4}$$

Now, let's assume that $dx = cdt$. We can then write:

$$\frac{dp_i}{dt} = \frac{dp_i}{dx}\frac{dx}{dt} = c\frac{dp_i}{dx}\,, \tag{1.5}$$

which allows to re-write Eqs. 1.4 as

$$\begin{aligned} \frac{dp_x}{dx} &= \frac{GMp}{c^2}\frac{x}{[x^2+(a-y)^2]^{3/2}}\,, \\ \frac{dp_y}{dx} &= \frac{GMp}{c^2}\frac{a-y}{[x^2+(a-y)^2]^{3/2}}\,. \end{aligned} \tag{1.6}$$

These equations allow us to calculate by how much does the momentum change along the x and the y axes as the x coordinate of the photon changes. Along the $x-$axis:

$$\Delta p_x = \frac{GMp}{c^2}\int_{-\infty}^{\infty}\frac{x}{[x^2+(a-y)^2]^{3/2}}dx = 0\,. \tag{1.7}$$

Thus, the photon momentum is un-changed along the x-axis. On the contrary, along the y-axis, the photon momentum changes by

$$\begin{aligned}\Delta p_y &= \frac{GMp}{c^2}\int_{-\infty}^{\infty}\frac{a-y}{[x^2+(a-y)^2]^{3/2}}dx \\ &= \frac{GMp}{c^2}\left[\frac{x}{(a-y)\sqrt{x^2+(a-y)^2}}\right]_{-\infty}^{+\infty} \\ &= \frac{2GMp}{c^2}\frac{1}{a-y}\,,\end{aligned} \tag{1.8}$$

which can be used to compute the deflection angle

$$\psi = \frac{\Delta p_y}{p} = \frac{2GM}{c^2}\frac{1}{a-y}\,. \tag{1.9}$$

If the photon impact parameter is $a-y=R_\odot$, Eq. 1.9 reduces to

$$\psi = \frac{\Delta p_y}{p} = \frac{2GM}{c^2R_\odot} \approx 0.875"\,, \tag{1.10}$$

when inserting $M=M_\odot=1.989\times10^{30}$ kg and $R_\odot=6.96\times10^8$ m. Thus, using Newtonian gravity and assuming that photons are light corpuscles, we obtain that a photon grazing the surface of the Sun is deflected by 0.875". We will see shortly that this value is just half of what predicted by Einsten in the framework of his Theory of General Relativity.

1.2 Deflection of light according to General Relativity

1.2.1 Fermat principle and light deflection

Starting from the field equations of general relativity, light deflection can be calculated by studying geodesic curves. It turns out that light deflection can equivalently be described by Fermat's principle, as in geometrical optics. This will be our starting point.

Exercise 1.1 — Derive the Snell's law from Fermat principle. In its simplest form the Fermat's principle says that light waves of a given frequency traverse the path between two points which takes the least time. The speed of light in a medium with refractive index n is c/n, where c is its speed in a vacuum. Thus, the time required for light to go some distance in such a medium is n times the time light takes to go the same distance in a vacuum.

Referring to Fig. 1.2.1, the time required for light to go from A to B becomes

$$t = [\{h_1^2+y^2\}^{1/2}+n\{h_2^2+(w-y)^2\}^{1/2}]/c.$$

We find the minimum time by differentiating t with respect to y and setting the result to zero, with the result that

$$\frac{y}{\{h_1^2+y^2\}^{1/2}} = n\frac{w-y}{\{h_2^2+(w-y)^2\}^{1/2}}.$$

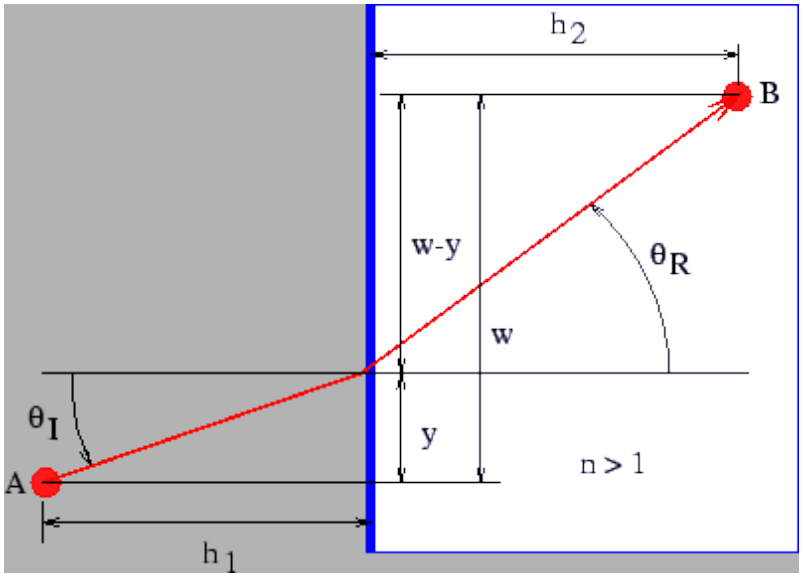


Figure 1.2.1: *Definition sketch for deriving Snell's law of refraction from Fermat's principle. The shaded area has refractive index $n > 1$*

However, we note that the left side of this equation is simply $\sin\theta_I$, while the right side is $n\sin\theta_R$, so that the minimum time condition reduces to

$$\sin\theta_I = n\sin\theta_R$$

We recognize this result as Snell's law. ■

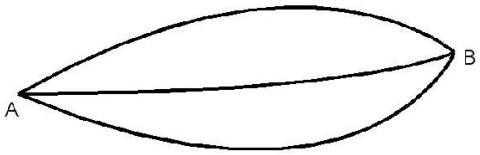
Taking inspiration from the Exercise above, we attempt to treat the deflection of light in a general relativity framework as a refraction problem. We need an refractive index n because Fermat's principle says that light will follow the path which makes extremal the travel time,

$$t_{travel} = \int \frac{n}{c} \mathrm{d}l \,. \tag{1.11}$$

As in geometrical optics, we thus search for the path, $\vec{x}(l)$, for which

$$\delta \int_A^B n(\vec{x}(l))\mathrm{d}l = 0 \,, \tag{1.12}$$

where the starting point A and the end point B are kept fixed.



Deflection in the Minkowski's space-time

In order to find the refractive index, we make a first approximation: we assume that the lens is weak, and that it is small compared to the overall dimensions of the optical system composed of source, lens and observer. With "weak lens", we mean a lens whose Newtonian gravitational potential Φ is much smaller than c^2, $\Phi/c^2 \ll 1$. Note that this approximation is valid in virtually all

cases of astrophysical interest. Consider for instance a galaxy cluster: its gravitational potential is $|\Phi| < 10^{-4}c^2 \ll c^2$. In addition, we also assume that the light deflection occurs in a region which is small enough that we can neglect the expansion of the universe.

In this case, the metric of (locally flat) unperturbed space-time is the Minkowski metric,

$$\eta_{\mu\nu} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix},$$

whose line element is

$$\mathrm{d}s^2 = \eta_{\mu\nu}\mathrm{d}x^\mu \mathrm{d}x^\nu = (\mathrm{d}x^0)^2 - (\mathrm{d}\vec{x})^2 = c^2\mathrm{d}t^2 - (\mathrm{d}\vec{x})^2 \,. \tag{1.13}$$

Now, we consider a weak lens perturbing this metric, such that

$$\eta_{\mu\nu} \to g_{\mu\nu} = \begin{pmatrix} 1+\frac{2\Phi}{c^2} & 0 & 0 & 0 \\ 0 & -(1-\frac{2\Phi}{c^2}) & 0 & 0 \\ 0 & 0 & -(1-\frac{2\Phi}{c^2}) & \\ 0 & 0 & 0 & -(1-\frac{2\Phi}{c^2}) \end{pmatrix}$$

for which the line element becomes

$$\mathrm{d}s^2 = g_{\mu\nu}\mathrm{d}x^\mu \mathrm{d}x^\nu = \left(1+\frac{2\Phi}{c^2}\right)c^2\mathrm{d}t^2 - \left(1-\frac{2\Phi}{c^2}\right)(\mathrm{d}\vec{x})^2 \,. \tag{1.14}$$

■ **Example 1.1 — Schwarzschild metric in the weak field limit.** Assuming a spherically symmetric and static potential, the Einstein's field equations can be solved to obtain the *Schwarzschild metric*. The line element is written in spherical coordinates as

$$\mathrm{d}s^2 = \left(1-\frac{2GM}{Rc^2}\right)c^2\mathrm{d}t^2 - \left(1-\frac{2GM}{Rc^2}\right)^{-1}\mathrm{d}R^2 - R^2(\sin^2\theta\mathrm{d}\phi^2 + \mathrm{d}\theta^2) \,.$$

To obtain a simpler expression, it is convenient to introduce the new radial coordinate r, defined through

$$R = r\left(1+\frac{GM}{2rc^2}\right)^2$$

and the cartesian coordinates $x = r\sin\theta\cos\theta$, $y = r\sin\theta\sin\phi$, and $z = r\cos\theta$, so that $\mathrm{d}l^2 = \mathrm{d}x^2 + \mathrm{d}y^2 + \mathrm{d}z^2$. After some algebra, the metric can then be written in the form

$$\mathrm{d}s^2 = \left(\frac{1-GM/2rc^2}{1+GM/2rc^2}\right)^2 c^2\mathrm{d}t^2 - \left(1+\frac{GM}{2rc^2}\right)^4 (\mathrm{d}x^2+\mathrm{d}y^2+\mathrm{d}z^2) \,.$$

In the weak field limit, $\Phi/c^2 = -GM/rc^2 \ll 1$,

$$\begin{aligned} \left(\frac{1-GM/2rc^2}{1+GM/2rc^2}\right)^2 &\approx \left(1-\frac{GM}{2rc^2}\right)^4 \\ &\approx \left(1-\frac{2GM}{rc^2}\right) \\ &= \left(1+\frac{2\Phi}{c^2}\right) \end{aligned}$$

and

$$\left(1+\frac{GM}{2rc^2}\right)^4 \approx \left(1+2\frac{GM}{rc^2}\right) = \left(1-\frac{2\Phi}{c^2}\right) .$$

Therefore, the Schwarzschild metric in the weak field limit equals

$$\mathrm{d}s^2 = \left(1+\frac{2\Phi}{c^2}\right)c^2\mathrm{d}t^2 - \left(1-\frac{2\Phi}{c^2}\right)\mathrm{d}l^2 ,$$

thus recovering Eq. 1.14. ∎

Effective refractive index

Light propagates at zero eigentime, $\mathrm{d}s = 0$, from which we obtain

$$\left(1+\frac{2\Phi}{c^2}\right)c^2\mathrm{d}t^2 = \left(1-\frac{2\Phi}{c^2}\right)(\mathrm{d}\vec{x})^2 . \tag{1.15}$$

The light speed in the gravitational field is thus

$$c' = \frac{|\mathrm{d}\vec{x}|}{\mathrm{d}t} = c\sqrt{\frac{1+\frac{2\Phi}{c^2}}{1-\frac{2\Phi}{c^2}}} \approx c\left(1+\frac{2\Phi}{c^2}\right) , \tag{1.16}$$

where we have used that $\Phi/c^2 \ll 1$ by assumption. The refractive index is thus

$$n = c/c' = \frac{1}{1+\frac{2\Phi}{c^2}} \approx 1-\frac{2\Phi}{c^2} . \tag{1.17}$$

With $\Phi \leq 0$, $n \geq 1$, and the light speed c' is smaller than in absence of the gravitational potential.

Deflection angle

The refractive index n depends on the spatial coordinate $\vec{x}$ and perhaps also on time t. Let $\vec{x}(l)$ be a light path. Then, the light travel time is

$$t_{travel} \propto \int_A^B n[\vec{x}(l)]\mathrm{d}l , \tag{1.18}$$

and the light path follows from

$$\delta\int_A^B n[\vec{x}(l)]\mathrm{d}l = 0 . \tag{1.19}$$

This is a standard variational problem, which leads to the well known Euler equations. In our case we write

$$\mathrm{d}l = \left|\frac{\mathrm{d}\vec{x}}{\mathrm{d}\lambda}\right|\mathrm{d}\lambda , \tag{1.20}$$

with a curve parameter λ which is yet arbitrary, and find

$$\delta\int_{\lambda_A}^{\lambda_B}\mathrm{d}\lambda\, n[\vec{x}(\lambda)]\left|\frac{\mathrm{d}\vec{x}}{\mathrm{d}\lambda}\right| = 0 \tag{1.21}$$

The expression

$$n[\vec{x}(\lambda)]\left|\frac{\mathrm{d}\vec{x}}{\mathrm{d}\lambda}\right| \equiv L(\dot{\vec{x}},\vec{x},\lambda) \tag{1.22}$$

takes the role of the Lagrangian, with

$$\dot{\vec{x}} \equiv \frac{\mathrm{d}\vec{x}}{\mathrm{d}\lambda} \,. \tag{1.23}$$

Finally, we have

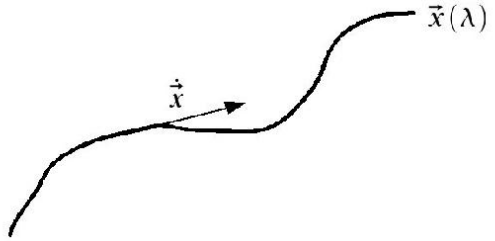
$$\left|\frac{\mathrm{d}\vec{x}}{\mathrm{d}\lambda}\right| = |\dot{\vec{x}}| = (\dot{\vec{x}}^2)^{1/2} \,. \tag{1.24}$$

The Euler equation writes:

$$\frac{\mathrm{d}}{\mathrm{d}\lambda}\frac{\partial L}{\partial \dot{\vec{x}}} - \frac{\partial L}{\partial \vec{x}} = 0 \,. \tag{1.25}$$

Now,

$$\frac{\partial L}{\partial \vec{x}} = |\dot{\vec{x}}|\frac{\partial n}{\partial \vec{x}} = (\vec{\nabla} n)|\dot{\vec{x}}| \,, \frac{\partial L}{\partial \dot{\vec{x}}} = n\frac{\dot{\vec{x}}}{|\dot{\vec{x}}|} \,. \tag{1.26}$$



Evidently, $\dot{\vec{x}}$ is a tangent vector to the light path, which we can assume to be normalized by a suitable choice for the curve parameter λ. We thus assume $|\dot{\vec{x}}| = 1$ and write $\vec{e} \equiv \dot{\vec{x}}$ for the unit tangent vector to the light path. Then, we have

$$\frac{\mathrm{d}}{\mathrm{d}\lambda}(n\vec{e}) - \vec{\nabla} n = 0 \,, \tag{1.27}$$

or

$$n\dot{\vec{e}} + \vec{e}\cdot[(\vec{\nabla} n)\dot{\vec{x}}] = \vec{\nabla} n \,,$$

$$\Rightarrow n\dot{\vec{e}} = \vec{\nabla} n - \vec{e}(\vec{\nabla} n \cdot \vec{e}) \,. \tag{1.28}$$

The second term on the right hand side is the derivative along the light path, thus the whole right hand side is the gradient of n perpendicular to the light path. Thus

$$\dot{\vec{e}} = \frac{1}{n}\vec{\nabla}_\perp n = \vec{\nabla}_\perp \ln n \,. \tag{1.29}$$

As $n = 1 - 2\Phi/c^2$ and $\Phi/c^2 \ll 1$, $\ln n \approx -2\Phi/c^2$, and

$$\dot{\vec{e}} \approx -\frac{2}{c^2}\vec{\nabla}_\perp \Phi \,. \tag{1.30}$$

The total deflection angle of the light path is now the integral over $-\dot{\vec{e}}$ along the light path,

$$\hat{\vec{\alpha}} = \vec{e}_{in} - \vec{e}_{out} = \frac{2}{c^2}\int_{\lambda_A}^{\lambda_B} \vec{\nabla}_\perp \Phi \mathrm{d}\lambda \,, \tag{1.31}$$

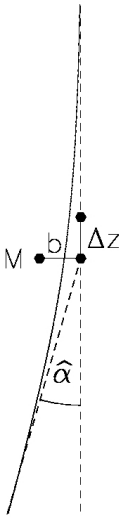
or, in other words, the integral over the "pull" of the gravitational potential perpendicular to the light path. Note that $\vec{\nabla}\Phi$ points away from the lens center, so $\hat{\vec{\alpha}}$ points in the same direction.

Born approximation

As it stands, the equation for $\hat{\vec{\alpha}}$ is not useful, as we would have to integrate over the actual light path. However, since $\Phi/c^2 \ll 1$, we expect the deflection angle to be small. Then, we can adopt the *Born approximation*, familiar from scattering theory, and integrate over the unperturbed light path.

Suppose, therefore, that a light ray starts out into $+\vec{e}_z$-direction and passes a lens at $z = 0$, with impact parameter b. The deflection angle is then given by

$$\hat{\vec{\alpha}}(b) = \frac{2}{c^2}\int_{-\infty}^{+\infty} \vec{\nabla}_\perp \phi \, \mathrm{d}z \tag{1.32}$$



■ **Example 1.2 — Deflection by a point mass.** If the lens is a point mass, then

$$\Phi = -\frac{GM}{r} \tag{1.33}$$

with $r = \sqrt{x^2+y^2+z^2} = \sqrt{b^2+z^2}$, $b = \sqrt{x^2+y^2}$ and

$$\vec{\nabla}_\perp \phi = \begin{pmatrix} \partial_x \Phi \\ \partial_y \Phi \end{pmatrix} = \frac{GM}{r^3}\begin{pmatrix} x \\ y \end{pmatrix} . \tag{1.34}$$

The deflection angle is then

$$\begin{aligned} \hat{\vec{\alpha}}(b) &= \frac{2GM}{c^2}\begin{pmatrix} x \\ y \end{pmatrix}\int_{-\infty}^{+\infty} \frac{\mathrm{d}z}{(b^2+z^2)^{3/2}} \\ &= \frac{4GM}{c^2}\begin{pmatrix} x \\ y \end{pmatrix}\left[\frac{z}{b^2(b^2+z^2)^{1/2}}\right]_0^\infty \\ &= \frac{4GM}{c^2 b}\begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix} , \end{aligned} \tag{1.35}$$

with

$$\begin{pmatrix} x \\ y \end{pmatrix} = b\begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix} \tag{1.36}$$

Notice that $R_s = \frac{2GM}{c^2}$ is the Schwarzschild radius of a (point) mass M, thus

$$|\hat{\vec{\alpha}}| = \frac{4GM}{c^2 b} = 2\frac{R_s}{b} . \tag{1.37}$$

Also notice that $\hat{\vec{\alpha}}$ is linear in M, thus the superposition principle can be applied to compute the deflection angle of an ensemble of lenses. ■

Deflection of light by the Sun's gravitational field

Note that the deflection angle found here in the framework of general relativity is very similar to the result found in the Newtonian limit for a photon grazing the surface of the Sun. However, we find here an extra factor two.

The reason for the factor of 2 difference is that both the space and time coordinates are bent in the vicinity of massive objects — it is four-dimensional space–time which is bent by the Sun.

The famous eclipse expedition of 1919 to Sobral, Brazil, and the island of Principe, in the Gulf of Guinea, led by Eddington, Dyson, and Davidson was a turning point in the history of relativity: it confirmed that masses bend light by the amount that is predicted by General Relativity.

For further reading on the Eddington expedition, we refer the reader to Smith (2015).

1.2.2 Deflection of light in the strong field limit

For the vast majority of gravitational lenses in the universe, the weak field limit holds. However, compact objects such as neutron stars and black holes can also act as lenses. In these cases, the approximations introduced above break down, as photons travel through very strong gravitational fields. In the following, we briefly discuss the deflection angle of a static (i.e. non-rotating) compact lens.

For a general static, spherically symmetric metric in the form

$$\mathrm{d}s^2 = A(R)\mathrm{d}t^2 - B(R)\mathrm{d}R^2 - C(R)(\mathrm{d}\theta^2 + \sin^2\theta\mathrm{d}\phi^2) \tag{1.38}$$

the analysis of the geodesic equations leads to the following expression for the deflection angle:

$$\hat{\alpha} = -\pi + \frac{2G}{c^2}\int_{R_m}^{\infty} u\sqrt{\frac{B(R)}{C(R)[C(R)/A(R)-u^2]}}\mathrm{d}R\,, \tag{1.39}$$

where u is the impact parameter of the unperturbed photon and R_m is the minimal distance of the deflected photon from the lens (Bozza, 2010). It can be shown that

$$u^2 = \frac{C(R_m)}{A(R_m)}\,. \tag{1.40}$$

Note that, in the case of the Schwarzschild metric, $A(R) = 1 - 2GM/Rc^2$, $B(R) = A(R)^{-1}$, and $C(R) = R^2$.

In the weak field limit ($R \geq R_m \gg 2GM/c^2$, i.e. for impact parameters much larger than the lens Schwarzschild radius), Eq. 1.39 reduces to the well know equation

$$\hat{\alpha} = \frac{4GM}{c^2 u}\,. \tag{1.41}$$

The exact solution of Eq.1.39 was calculated by Darwin (1959) to be

$$\hat{\alpha} = -\pi + 4\frac{G}{c^2}\sqrt{R_m/s}F(\varphi,m)\,, \tag{1.42}$$

where $F(\phi,m)$ is the elliptic integral of the first kind, and

$$s = \sqrt{(R_m - 2M)(R_m + 6M)} \tag{1.43}$$

$$m = (s - R_m + 6M)/2s \tag{1.44}$$

$$\varphi = \arcsin\sqrt{2s/(3R_m - 6M + s)} \tag{1.45}$$

Fig.1.2.2 shows how the deflection angle varies as a function of the impact parameter of the photon. At large distances, Eq. 1.42 is well approximated by the solution in the weak field limit. For small impact parameters, the solutions in the strong and in the weak field limit differ significantly. In particular, the deflection angle in Eq. 1.42 diverges for $u = 3\sqrt{3}GM/c^2$ (or $R_m = 3GM/c^2$). Before reaching that point, the deflection angle exceeds 2π, meaning that the photon loops around the lens before leaving it.

1.3 Deflection by an ensemble of point masses

The deflection angle in Eq. 1.37 depends linearly on the mass M. This result was obtained by linearizing the equations of general relativity in the weak field limit. Under these circumstances, the superposition principle holds and the deflection angle of an array of lenses can be calculated as the sum of all contributions by each single lens.

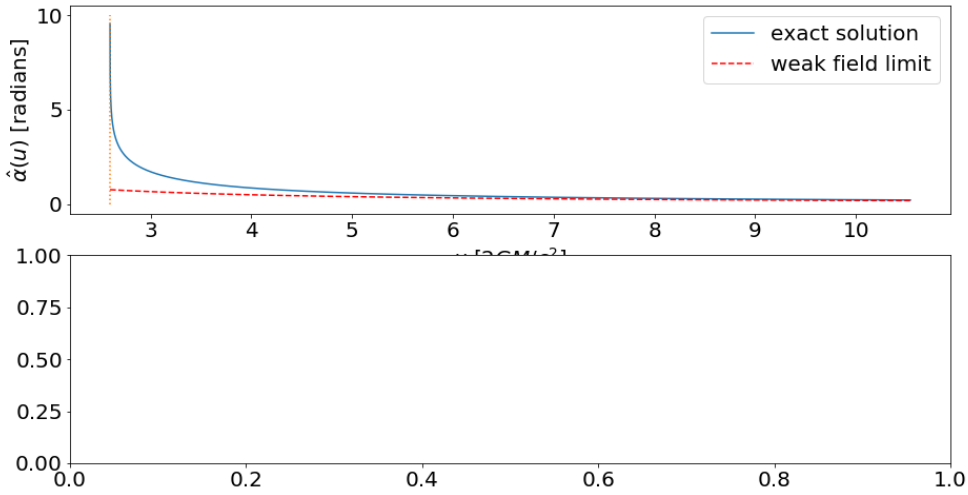


Figure 1.2.2: *Deflection angle by a compact lens as a function of the photon impact parameter. Shown are the exact solution of the geodesic equations for a Schwarzschild metric (solid line) and the solution in the weak field approximation (dashed line). The dotted vertical line shows the impact parameter, $u = 3\sqrt{3}GM/c^2$, for which the exact solution diverges, indicating that the photon keeps looping around the lens.*

Suppose we have a sparse distribution of N point masses on a plane, whose positions and masses are $\vec{\xi}_i$ and M_i, $1 \leq i \leq N$. The deflection angle of a light ray crossing the plane at $\vec{\xi}$ will be:

$$\hat{\vec{\alpha}}(\vec{\xi}) = \sum_i \hat{\vec{\alpha}}_i(\vec{\xi} - \vec{\xi}_i) = \frac{4G}{c^2} \sum_i M_i \frac{\vec{\xi} - \vec{\xi}_i}{|\vec{\xi} - \vec{\xi}_i|^2} \,. \tag{1.46}$$

Note that the formula above is similar to that we would use to compute the gravitational force between point masses on the plane. While the force depends on the inverse squared distance, the deflection angle scales as ξ^{-1}. In the case of many lenses, the computation of the deflection angle using Eq. 1.46 can become very computationally expensive, as it costs $O(N^2)$. However, as it is usually done to solve numerically N-body problems, algorithms employing meshes or hierarchies (such as the so-called tree algorithms (Barnes and Hut, 1986)) can significantly reduce the cost of calculations (e.g. to $O(N \log N)$). For some application of these algorithms in the computation of the deflection angles, we refer the reader to the works of Aubert, Amara, and Metcalf (2007) and Meneghetti et al. (2010).

1.4 Deflection by an extended mass distribution

We now consider more realistic lens models, i.e. three dimensional distributions of matter. Even in the case of lensing by galaxy clusters, the physical size of the lens is generally much smaller than the distances between observer, lens and source. The deflection therefore arises along a very short section of the light path. This justifies the usage of the *thin screen approximation*: the lens is approximated by a planar distribution of matter, the lens plane.

Within this approximation, the lensing matter distribution is fully described by its surface density,

$$\Sigma(\vec{\xi}) = \int \rho(\vec{\xi}, z) \, \mathrm{d}z \,, \tag{1.47}$$

where $\vec{\xi}$ is a two-dimensional vector on the lens plane and ρ is the three-dimensional density.

As long as the thin screen approximation holds, the total deflection angle is obtained by summing the contribution of all the mass elements $\Sigma(\vec{\xi})\mathrm{d}^2\xi$:

$$\vec{\hat{\alpha}}(\vec{\xi}) = \frac{4G}{c^2}\int \frac{(\vec{\xi}-\vec{\xi}')\Sigma(\vec{\xi}')}{|\vec{\xi}-\vec{\xi}'|^2}\,\mathrm{d}^2\xi' \,. \tag{1.48}$$

This equation shows that the calculation of the deflection angle is formally a convolution of the surface density $\Sigma(\vec{\xi})$ with the kernel function

$$\vec{K}(\vec{\xi}) \propto \frac{\vec{\xi}}{|\vec{\xi}|^2} \,. \tag{1.49}$$

This enables the calculation of the deflection angle field in the Fourier space as the product of the Fourier transforms of Σ and K:

$$\tilde{\hat{\alpha}}_i(\vec{k}) \propto \tilde{\Sigma}(\vec{k})\tilde{K}_i(\vec{k}) \,, \tag{1.50}$$

where $\vec{k}$ is the conjugate variable to $\vec{\xi}$ and the tilde denotes the Fourier Transforms. The subscript $i \in [1,2]$ indicates the two components along the two axes on the lens plane (remember that $\hat{\alpha}$ is a vector!). This calculation can be implemented efficiently using the *Fast-Fourier-Transform (FFT)* algorithm (Cooley and Tukey, 1965). Note that this assumes that the integration extends to an infinite domain, while gravitational lenses have finite mass distributions. FFT algorithms implement this feature assuming periodic conditions on the boundaries of the integration domain.

1.5 Python applications

1.5.1 Deflection by a black-hole

In our first python application, we write a script to produce Fig. 1.2.2. A brief python tutorial can be found in Appendix A.

We want to implement the formula in Eq. 1.42. We also need to remind that

$$u^2 = \frac{C(R_m)}{A(R_m)}$$

We will compare the resulting deflection angle to

$$\hat{\alpha} = \frac{4GM}{c^2 u} \tag{1.51}$$

which is the result we obtained in the weak-field limit.

We start by importing some useful packages:

```
from scipy import special as sy  # need special functions for incomplete \\
# elliptic integrals of the first kind
import numpy as np # efficient vector and matrix operations
import matplotlib.pyplot as plt # a MATLAB-like plotting framework
%matplotlib inline # only needed in jupyter notebooks
```

Note that we import the module `special` from `scipy` in order to compute the elliptic integral of the first kind appearing in Eq. 1.42. See https://docs.scipy.org/doc/scipy/reference/special.html.

Our goal is to produce a graph. Let's setup the fonts and the character size

```
font = {'family' : 'normal',
        'weight' : 'normal',
        'size'   : 20}

import matplotlib
matplotlib.rc('font', **font)
```

The task can be completed in several ways. Here we chose to build a class for point black-holes:

```
class point_bh:

    def __init__(self,M):
        self.M=M

    # functions which define the metric.
    def A(self,r):
        return(1.0-2.0*self.M/r)

    def B(self,r):
        return (self.A(r)**(-1))

    def C(self,r):
        return(r**2)

    # compute u from rm
    def u(self,r):
        u=np.sqrt(self.C(r)/self.A(r))
        return(u)

    # functions concurring to the deflection angle calculation
    def ss(self,r):
        return(np.sqrt((r-2.0*self.M)*(r+6.0*self.M)))

    def mm(self,r,s):
        return((s-r+6.0*self.M)/2/s)

    def phif(self,r,s):
        return(np.arcsin(np.sqrt(2.0*s/(3.0*r-6.0*self.M+s))))

    # the deflection angle
    def defAngle(self,r):
        s=self.ss(r)
        m=self.mm(r,s)
        phi=self.phif(r,s)
        F=sy.ellipkinc(phi, m) # using the ellipkinc function
                               # from scipy.special
        return(-np.pi+4.0*np.sqrt(r/s)*F)
```

The class contains several methods which will be used to compute the deflection angle. For example, we implement the functions $A(R)$, $B(R)$, and $C(R)$. These will be used to convert the

minimal distance R_m to u. We also implement the functions s, m, φ, which depend on the mass of the black-hole and on the minimal distance R_m. Finally, we implement the function `defAngle`, which enables to compute the deflection angle using Eq. 1.42. This function uses the method `elipkinc` from `scipy.special` to compute the incomplete elliptic integral of the first kind, $F(\varphi, m)$. Note that φ and m can be passed as `numpy` arrays, i.e. `elipkinc` can return values for a number of couples (φ, m).

Following the same approach, we build another class which deals with point lenses in the weak field limit, i.e. it implements Eq. 1.51:

```
class point_mass:

    def __init__(self,M):
        self.M=M

    # the classical formula
    def defAngle(self,u):
        return(4.0*self.M/u)
```

We can now use the two classes above to build two objects, namely a black-hole lens (employing the exact solution for the deflection angle) and a point mass lens, for which we will adopt the weak-field limit. In both cases, the mass of the lens is fixed to $3M_\odot$. For a mass of this size, the Schwarzchild radius is $R_s \sim 9$km:

```
bh=point_bh(3.0)
pm=point_mass(3.0)
```

We now use the `linspace` method from `numpy` to initialize an vector of minimal distances R_m, which we will use to compute $\hat{\alpha}$. We use the method `u(r)` of `point_bh` to convert R_m into an array of impact parameters u:

```
r=np.linspace(3.0/2.0,10,1000)*2.0*bh.M
u=bh.u(r)/2.0/bh.M
```

The deflection angle as a function of u or R_m can be computed in the cases of the exact solution and in the weak field limit using the method `defAngle` applied to `bh` and `pm`:

```
a=bh.defAngle(r)
b=pm.defAngle(u*2.0*bh.M)
```

Note that u is in units of the Schwarzchild radius and that we have set $G/c^2 = 1$.

Finally, we can produce a nice figure displaying the results of the calculation. We use `matplotlib.pyplot` to do this:

```
# initialize figure and axes
# (single plot, 15" by 8" in size)
fig,ax=plt.subplots(1,1,figsize=(15,8))
# plot the exact solution in ax
ax.plot(u,a,'-',label='exact solution')
# plot the solution in the weak field limit
ax.plot(u,b,'--',label='weak field limit',color='red')
# set the labels for the x and the y axes
ax.set_xlabel(r'$u$ $[2GM/c^2]$')
```

```
ax.set_ylabel(r'$\hat\alpha(u)$ [radians]')
# add the legend
ax.legend()
```

We also want to show the vertical asymptote at $u_{lim} = 3\sqrt{3}/2$:

```
# plot a vertical dotted line at u=3\sqrt(3)/2
x=[np.min(u),np.min(u)]
y=[0,10]
ax.plot(x,y,':')
```

To conclude, we save the figure in a .png file:

```
# save figure in png format
fig.savefig('bhalpha.png')
```

1.5.2 Deflection by an extended mass distribution

In this application, we implement the calculation of the deflection angle field by an extended lens. A two-dimensional map of the lens surface-density is provided by the fits file `kappa_gl.fits` (see the data folder in the github repository). The map was obtained by projecting the mass distribution of a dark matter halo obtained from N-body simulations on a lens plane. To be precise, this is the surface density divided by a constant which depends on the lens and source redshifts (we will talk about this constant in the next lectures). Let's denote this quantity as κ. Accounting for this normalization, the calculation we want to implement is

$$\vec{\alpha}(\vec{x}) = \frac{1}{\pi}\int \kappa(\vec{x}')\frac{\vec{x}-\vec{x}'}{|\vec{x}-\vec{x}'|^2}d^2x' \,.$$

This is a convolution, which can be written in the Fourier Space as

$$\vec{\tilde{\alpha}}(\vec{k}) = \frac{1}{\pi}\tilde{\kappa}(\vec{k})\vec{\tilde{K}}(\vec{k})$$

where $\vec{\tilde{K}}(\vec{k})$ is the Fourier Transform of

$$\vec{K}(\vec{x}) = \frac{\vec{x}}{|\vec{x}|^2}$$

We use the `numpy.fft` module:

```
import numpy as np
import numpy.fft as fftengine
```

We define a class called deflector, where the deflector object is initialized by reading the fits file containing the surface density map of the lens. To deal with the fits files, we need to use the `astropy.io.fits` module.

The class contains some methods to

- build the kernel $K(\vec{x})$;
- compute the deflection angle map by convolving the convergence with the kernel;
- perform the so-called "zero-padding";
- crop the zero-padded maps.

```
import astropy.io.fits as pyfits


class deflector(object):

    # initialize the deflector using a surface density (covergence) map
    # the boolean variable pad indicates whether zero-padding is used
    # or not
    def __init__(self,filekappa,pad=False):
        kappa,header=pyfits.getdata(filekappa,header=True)
        self.kappa=kappa
        self.nx=kappa.shape[0]
        self.ny=kappa.shape[1]
        self.pad=pad
        if (pad):
            self.kpad()
        self.kx,self.ky=self.kernel()

    # implement the kernel function K
    def kernel(self):
        x=np.linspace(-0.5,0.5,self.kappa.shape[0])
        y=np.linspace(-0.5,0.5,self.kappa.shape[1])
        kx,ky=np.meshgrid(x,y)
        norm=(kx**2+ky**2+1e-12)
        kx=kx/norm
        ky=ky/norm
        return(kx,ky)

    # compute the deflection angle maps by convolving
    # the surface density with the kernel function
    def angles(self):
        # FFT of the surface density and of the two components of the kernel
        density_ft = fftengine.fftn(self.kappa,axes=(0,1))
        kernelx_ft = fftengine.fftn(self.kx,axes=(0,1),
                                    s=self.kappa.shape)
        kernely_ft = fftengine.fftn(self.ky,axes=(0,1),
                                    s=self.kappa.shape)
        # perform the convolution in Fourier space and transform the result
        # back in real space. Note that a shift needs to be applied using
        # fftshift
        alphax = 1.0/np.pi/(self.kappa.shape[0])*\
                fftengine.fftshift(fftengine.ifftn(kappa_ft*kernelx_ft))
        alphay = 1.0/np.pi/(self.kappa.shape[0])*\
                fftengine.fftshift(fftengine.ifftn(kappa_ft*kernely_ft))
        return(alphax.real,alphay.real)

    # returns the surface-density (convergence) of the deflector
    def kmap(self):
        return(self.kappa)
```

```
    # performs zero-padding
    def kpad(self):
        # add zeros around the original array
        def padwithzeros(vector, pad_width, iaxis, kwargs):
            vector[:pad_width[0]] = 0
            vector[-pad_width[1]:] = 0
            return vector
        # use the pad method from numpy.lib to add zeros (padwithzeros)
        # in a frame with thickness self.kappa.shape[0]
        self.kappa=np.lib.pad(self.kappa, self.kappa.shape[0],
                              padwithzeros)

    # crop the maps to remove zero-padded areas and get back to the
    # original region.
    def mapCrop(self,mappa):
        xmin=0.5*(self.kappa.shape[0]-self.nx)
        ymin=0.5*(self.kappa.shape[1]-self.ny)
        xmax=xmin+self.nx
        ymax=ymin+self.ny
        mappa=mappa[xmin:xmax,ymin:ymax]
        return(mappa)
```

We can now build a deflector and use it to compute the deflection angles employing the method `angles`:

```
df=deflector('data/kappa_gl.fits')
angx_nopad,angy_nopad=df.angles()
kappa=df.kmap()

import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm, PowerNorm, SymLogNorm
%matplotlib inline

fig,ax = plt.subplots(1,3,figsize=(16,8))
ax[0].imshow(kappa,origin="lower")
ax[0].set_title('convergence')
ax[1].imshow(angx_nopad,origin="lower")
ax[1].set_title('angle 1')
ax[2].imshow(angy_nopad,origin="lower")
ax[2].set_title('angle 2')
```

Note that at this point we have not yet used the zero-padding trick. FFT assumes periodic boundary conditions, meaning that the lens mass distribution is replicated outside the boundaries. Given that the region around the lens considered in this example is relatively small, we expect that the deflection angles will be biased near the borders. The three panels in Fig. 1.5.1 show the maps of the convergence and of the two components of the deflection angles obtained with this setting.

Zero-padding consists of placing zeros all around the convergence map. By doing so, we double the size of the original map, but we expect to increase the accuracy of the calculations near the borders, beacause the periodic conditions are better reproduced in this setting. We activate zero-padding by just setting the variable `pad=True` when initializing the deflector. Fig. 1.5.2 shows

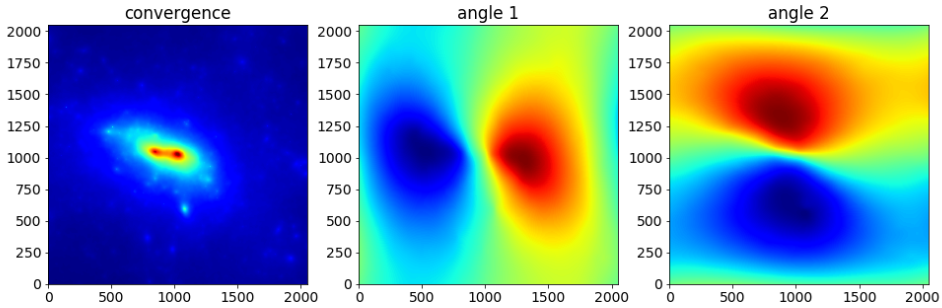


Figure 1.5.1: *Left panel: the surface density (convergence) map of the lens. Middle and right panels: maps of the two components of the deflection angles.*

the zero-padded convergence map and the two new maps of the deflection angle components.

```
df=deflector('data/kappa_gl.fits',True)
angx,angy=df.angles()
kappa=df.kmap()

fig,ax = plt.subplots(1,3,figsize=(16,8))
angx,angy=df.angles()
ax[0].imshow(kappa,origin="lower")
ax[0].set_title('convergence')
ax[1].imshow(angx,origin="lower")
ax[1].set_title('angle 1')
ax[2].imshow(angy,origin="lower")
ax[2].set_title('angle 2')
```

We are not interested in this large area, thus we can get rid of the values outside the lens convergence map by cropping the deflection angle maps. The results are shown in Fig. 1.5.3 and compared to the previous ones. In fact, significant differences are visible along the borders.

```
angx=df.mapCrop(angx)
angy=df.mapCrop(angy)

fig,ax = plt.subplots(2,2,figsize=(16,16))
ax[0,0].imshow(angx,origin="lower")
ax[0,0].set_title('angle 1')
ax[0,1].imshow(angy,origin="lower")
ax[0,1].set_title('angle 2')
ax[1,0].imshow(angx_nopad,origin="lower")
ax[1,0].set_title('angle 1 - no zero pad')
ax[1,1].imshow(angy_nopad,origin="lower")
```

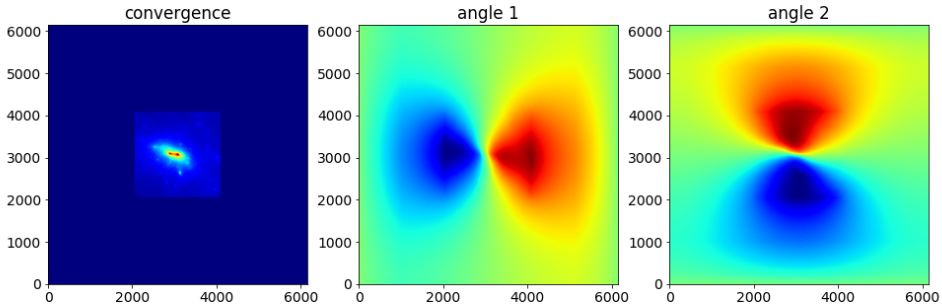


Figure 1.5.2: *The figure shows the same maps as in Fig. 1.5.1, but with zero-padding. Indeed, as shown in the left panel, the lens is surrounded by a frame of zeros, and the deflection angle maps are computed on an area which has double the size of the maps in Fig. 1.5.1.*

```
ax[1,1].set_title('angle 2 - no zero pad')
```

1.6 Problems

Problem 1.1 — Write a python script to produce a figure displaying $\hat{\alpha}(R_m)$ with R_m in the range 9-1000 km for two lenses with mass $M = 3M_\odot$ and $M = 10M_\odot$..

Problem 1.2 — Define a class for an ensemble of point masses. The class should be initialized with two numpy arrays containing the masses and the positions of the lenses. Use the thin screen approximation and write the method to compute the deflection angle at a certain location on the lens plane..

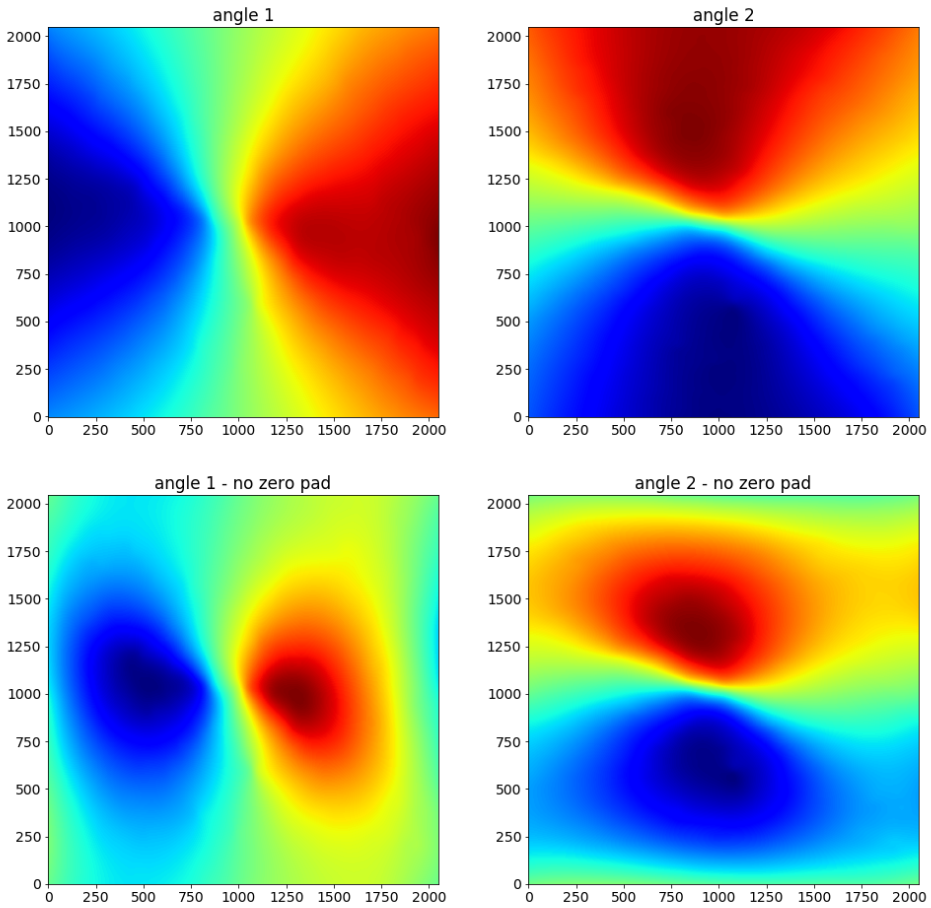
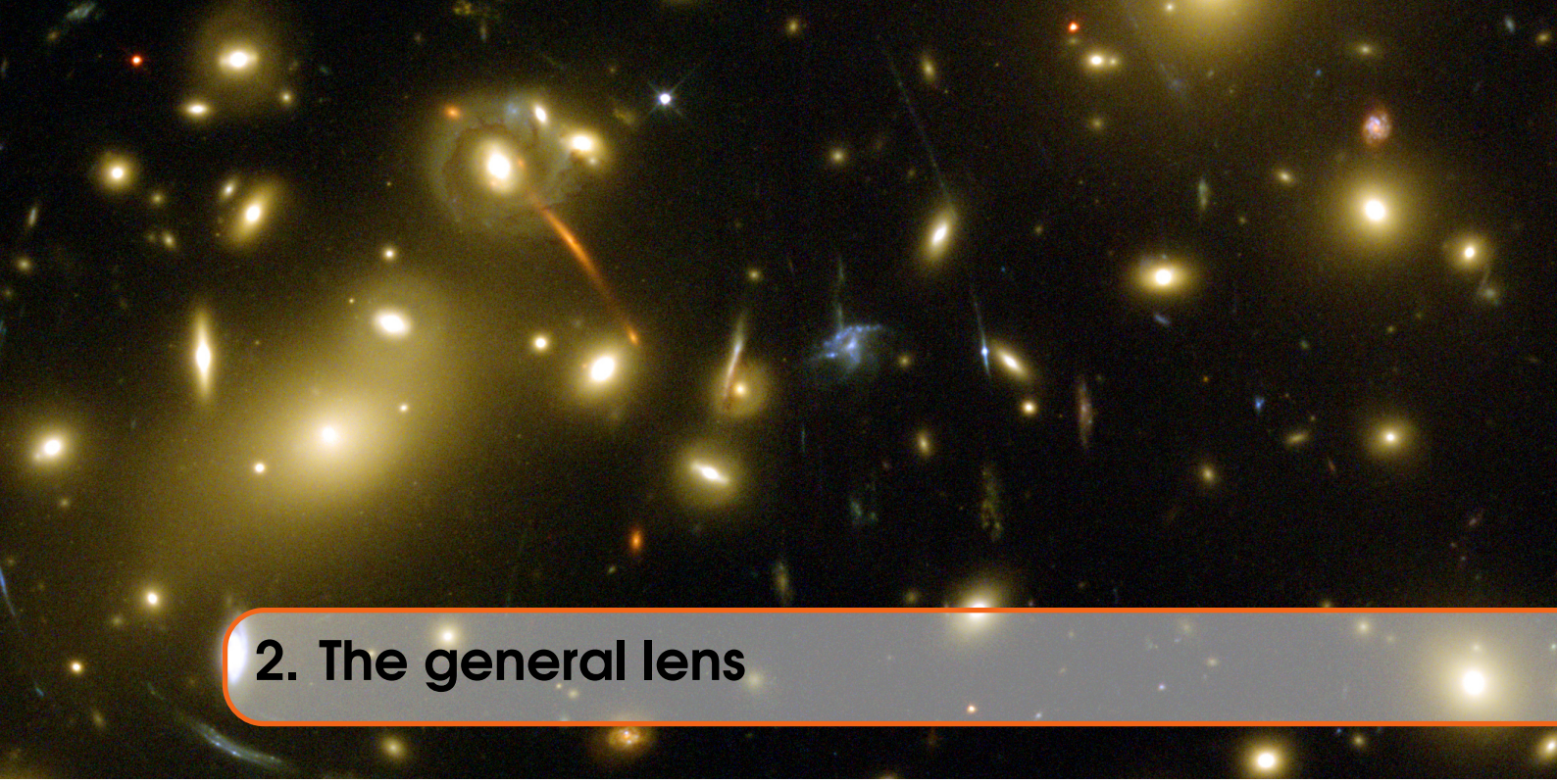


Figure 1.5.3: *The upper panels show the same two maps displayed in the middle and right panels of Fig. 1.5.2, which have been cropped to match the original size of the input convergence map. The bottom panels show the maps obtained without padding, for comparison.*



2. The general lens

2.1 Lens equation

Gravitational lensing is sensitive to the geometry of the universe. In particular, as in refractive phenomena, the amplitude of the lensing effects is heavily dependent on the distances between the observer, the lenses, and the sources. These are in turn related to the curvature and expansion rate of the universe, which suggests that gravitational lensing is indeed a powerful tool for cosmology.

Instrinsic and apparent source position

In this section, we seek a relationship between observed and intrinsic positions of a source in a gravitational lensing event. In absence of the lens, the light emitted by a distant source reaches an observer, who sees the source at a certain position on the sky, $\vec{\beta}$ (in angular units). This is the *intrinsic* position of the source. Instead, when photons are deflected by the gravitational lens, the observer collects them from a different direction, $\vec{\theta}$, which corresponds to the *apparent* (or *observed*) position of the source. We refer to the apparent position of the source as to the *image* position.

In Fig. (2.1.1), we sketch a typical gravitational lens system. A mass is placed at redshift $z_{\rm L}$, corresponding to an angular diameter distance $D_{\rm L}$. This lens deflects the light rays coming from a source at redshift $z_{\rm S}$ (or angular distance $D_{\rm S}$). At the bottom of the diagram, an observer collects the photons from the distant source. The angular diameter distance between the lens and the source is $D_{\rm LS}$.

R The angular diameter distance D_A is defined as the ratio of an object's physical transverse size to its angular size (in radians). Therefore, it is used to convert angular separations in the sky to physical separations on the plane of the sources.

This distance does not increase indefinitely with redshift, but it peaks at $z \sim 1$ and then it turns over. Due to the expansion of the universe the angular diameter distance between z_1 and z_2 (with $z_2 > z_1$) is not found by subtracting the two individual angular diameter distances:

$$D_A(z_1, z_2) \neq D_A(z_2) - D_A(z_1) \tag{2.1}$$

except for those situations where the expansion of the universe can be neglected (i.e. for

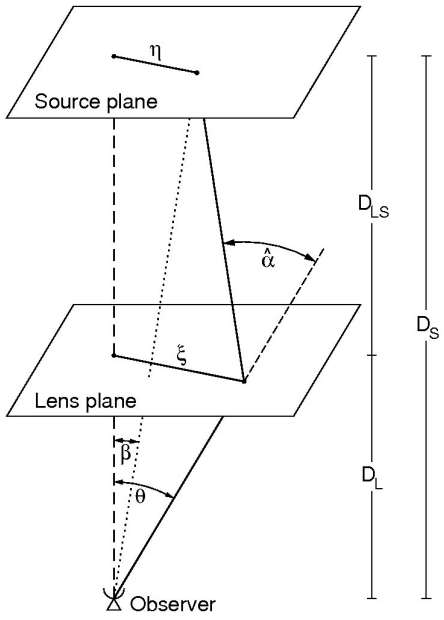


Figure 2.1.1: *Sketch of a typical gravitational lensing system. Figure from Bartelmann and Schneider (2001).*

lenses and sources in our own galaxy). More in-depth discussions can be found in Appendix B and in several cosmology books (see e.g. Weinberg, 1972).

Thin screen approximation

If the physical size of the lens is small compared to the distances $D_{\rm L}$, $D_{\rm LS}$, and $D_{\rm S}$, the extension of the lens along the line-of-sight can be neglected in the calculation of the light deflection. We can assume that this occurs on a plane, called the *lens plane*.

R Given that the apparent position of the source, or image position, originates on this plane, the lens plane is often refereed as the *image plane*.

Similarly, we can assume that all photons emitted by the source originate from the same distance $D_{\rm S}$, meaning that the source lies on a *source plane*. The approximation of the lens and of the source to planar distributions of mass and light, is called *thin screen approximation*.

Relating the intrinsic and apparent positions of the source

We first define an optical axis, indicated in Fig. (2.1.1) by the dashed line, perpendicular to the lens and source planes and passing through the observer. Then we measure the angular positions on the lens and on the source planes with respect to this reference direction.

Consider a source at the intrinsic angular position $\vec{\beta}$, which lies on the source plane at a distance $\vec{\eta} = \vec{\beta} D_{\rm S}$ from the optical axis. The source emits photons (we may now use the term “light rays") that impact the lens plane at $\vec{\xi} = \vec{\theta} D_{\rm L}$, are deflected by the angle $\vec{\hat{\alpha}}$, and finally reach the observer. The amplitude of the deflection is given by Eq. (1.32).

Due to the deflection, the observer receives the light coming from the source as if it was emitted at the apparent angular position $\vec{\theta}$. Note that we have used vectors to identify the source and image positions on the corresponding planes, either in the case of angular and physical positions.

If $\vec{\theta}$, $\vec{\beta}$, and $\vec{\hat{\alpha}}$ are small, the true position of the source and its observed position on the sky are related by a very simple relation, which can be readily obtained from the diagram in Fig. 2.1.1.

This relation is called the *lens equation* and is written as

$$\vec{\theta} D_S = \vec{\beta} D_S + \hat{\vec{\alpha}} D_{LS} \, , \tag{2.2}$$

where D_{LS} is the angular diameter distance between lens and source.

Defining the reduced deflection angle

$$\vec{\alpha}(\vec{\theta}) \equiv \frac{D_{LS}}{D_S} \hat{\vec{\alpha}}(\vec{\theta}) \, , \tag{2.3}$$

from Eq. (2.2), we obtain

$$\vec{\beta} = \vec{\theta} - \vec{\alpha}(\vec{\theta}) \, . \tag{2.4}$$

This equation, called *lens equation*, is apparently very simple. However, $\vec{\alpha}(\vec{\theta})$ can be a complicated function of $\vec{\theta}$, which implies that the equation can only be solved numerically in many cases.

It is very common and useful to write Eq. (2.2) in dimensionless form. This can be done by defining a length scale ξ_0 on the lens plane and a corresponding length scale $\eta_0 = \xi_0 D_S / D_L$ on the source plane. Then, we define the dimensionless vectors

$$\vec{x} \equiv \frac{\vec{\xi}}{\xi_0} \quad ; \quad \vec{y} \equiv \frac{\vec{\eta}}{\eta_0} \, , \tag{2.5}$$

as well as the scaled deflection angle

$$\vec{\alpha}(\vec{x}) = \frac{D_L D_{LS}}{\xi_0 D_S} \hat{\vec{\alpha}}(\xi_0 \vec{x}) \, . \tag{2.6}$$

Carrying out some substitutions, Eq. (2.2) can finally be written as

$$\vec{y} = \vec{x} - \vec{\alpha}(\vec{x}) \, . \tag{2.7}$$

Solving the lens equation

From Eqs.2.4 and 2.7, it is obvious that knowing the intrinsic position of the source and the deflection angle field $\vec{\alpha}(\vec{\theta})$ of the lens, the positions of the image(s) can be be found by solving the lens equation for $\vec{\theta}$. As it will be discussed later on, this can be achieved analytically only for very simple lens mass distributions. Indeed, the equation is typically highly non-linear. When multiple solutions exist, the source is lensed into *multiple images*.

When observing a lens system, the intrinsic position of the source is unknown, while the position of its images can be measured. Then the source intrinsic position can be recovered by assuming a model for the mass distribution of the lens, i.e. by solving the lens equation for $\vec{\beta}$. This is a much easier task, because the lens equation is linear in $\vec{\beta}$: for each image there is a unique solution. Thus, if multiple images of the same source are identified, and the lens mass model is correct, the same solution of the lens equation should be found for all images.

2.2 Lensing potential

An extended distribution of matter is characterized by its *effective lensing potential*, obtained by projecting the three-dimensional Newtonian potential on the lens plane and by properly rescaling it:

$$\hat{\Psi}(\vec{\theta}) = \frac{D_{LS}}{D_L D_S} \frac{2}{c^2} \int \Phi(D_L \vec{\theta}, z) \mathrm{d}z \, . \tag{2.8}$$

The lensing potential satisfies two important properties:

1. **the gradient of $\hat{\Psi}$ is the reduced deflection angle**:

$$\vec{\nabla}_\theta \hat{\Psi}(\vec{\theta}) = \vec{\alpha}(\vec{\theta}) \,. \tag{2.9}$$

Indeed, by taking the gradient of the lensing potential we obtain:

$$\begin{aligned} \vec{\nabla}_\theta \hat{\Psi}(\vec{\theta}) &= D_\mathrm{L} \vec{\nabla}_\perp \hat{\Psi} = \vec{\nabla}_\perp \left(\frac{D_\mathrm{LS}}{D_\mathrm{S}} \frac{2}{c^2} \int \hat{\Phi}(\vec{\theta}, z) \mathrm{d}z \right) \\ &= \frac{D_\mathrm{LS}}{D_\mathrm{S}} \frac{2}{c^2} \int \vec{\nabla}_\perp \Phi(\vec{\theta}, z) \mathrm{d}z \\ &= \vec{\alpha}(\vec{\theta}) \end{aligned} \tag{2.10}$$

Note that, using the dimensionless notation,

$$\vec{\nabla}_x = \frac{\xi_0}{D_\mathrm{L}} \vec{\nabla}_\theta \,. \tag{2.11}$$

We can see that

$$\vec{\nabla}_x \hat{\Psi}(\vec{\theta}) = \frac{\xi_0}{D_\mathrm{L}} \vec{\nabla}_\theta \hat{\Psi}(\vec{\theta}) = \frac{\xi_0}{D_\mathrm{L}} \vec{\alpha}(\vec{\theta}) \,. \tag{2.12}$$

By multiplying both sides of this equation by D_L^2/ξ_0^2, we obtain

$$\frac{D_\mathrm{L}^2}{\xi_0^2} \vec{\nabla}_x \hat{\Psi} = \frac{D_\mathrm{L}}{\xi_0} \vec{\alpha} \,. \tag{2.13}$$

This allows us to introduce the dimensionless counterpart of $\hat{\Psi}$:

$$\Psi = \frac{D_\mathrm{L}^2}{\xi_0^2} \hat{\Psi} \,. \tag{2.14}$$

Substituting Eq. 2.14 into Eq 2.13, we see that

$$\vec{\nabla}_x \Psi(\vec{x}) = \vec{\alpha}(\vec{x}) \,. \tag{2.15}$$

2. **the Laplacian of $\hat{\Psi}$ is twice the *convergence* κ**:

$$\triangle_\theta \hat{\Psi}(\vec{\theta}) = 2\kappa(\vec{\theta}) \,. \tag{2.16}$$

The *convergence* is defined as a dimensionless surface density

$$\kappa(\vec{\theta}) \equiv \frac{\Sigma(\vec{\theta})}{\Sigma_\mathrm{cr}} \quad \text{with} \quad \Sigma_\mathrm{cr} = \frac{c^2}{4\pi G} \frac{D_\mathrm{S}}{D_\mathrm{L} D_\mathrm{LS}} \,, \tag{2.17}$$

where Σ_cr is called the *critical surface density*, a quantity which characterizes the lens system and which is a function of the angular diameter distances of lens and source.
Eq. 2.16 is derived from the Poisson equation,

$$\triangle \Phi = 4\pi G \rho \,. \tag{2.18}$$

The surface mass density is

$$\Sigma(\vec{\theta}) = \frac{1}{4\pi G} \int_{-\infty}^{+\infty} \triangle \Phi \mathrm{d}z \tag{2.19}$$

and

$$\kappa(\vec{\theta}) = \frac{1}{c^2} \frac{D_\mathrm{L} D_\mathrm{LS}}{D_\mathrm{S}} \int_{-\infty}^{+\infty} \triangle \Phi \mathrm{d}z \,. \tag{2.20}$$

Let us now introduce a two-dimensional Laplacian

$$\triangle_\theta = \frac{\partial^2}{\partial\theta_1^2} + \frac{\partial^2}{\partial\theta_2^2} = D_{\rm L}^2\left(\frac{\partial^2}{\partial\xi_1^2} + \frac{\partial^2}{\partial\xi_2^2}\right) = D_{\rm L}^2\left(\triangle - \frac{\partial^2}{\partial z^2}\right) , \tag{2.21}$$

which gives

$$\triangle\Phi = \frac{1}{D_{\rm L}^2}\triangle_\theta\Phi + \frac{\partial^2\Phi}{\partial z^2} . \tag{2.22}$$

Inserting Eq. 2.22 into Eq. 2.20, we obtain

$$\kappa(\vec{\theta}) = \frac{1}{c^2}\frac{D_{\rm LS}}{D_{\rm S}D_{\rm L}}\left[\triangle_\theta\int_{-\infty}^{+\infty}\Phi{\rm d}z + D_{\rm L}^2\int_{-\infty}^{+\infty}\frac{\partial^2\Phi}{\partial z^2}{\rm d}z\right] . \tag{2.23}$$

If the lens is gravitationally bound, $\partial\Phi/\partial z = 0$ at its boundaries and the second term on the right hand side vanishes. From Eqs. 2.8 and 2.14, we find

$$\kappa(\theta) = \frac{1}{2}\triangle_\theta\hat{\Psi} = \frac{1}{2}\frac{\xi_0^2}{D_{\rm L}^2}\triangle_\theta\Psi . \tag{2.24}$$

Since

$$\triangle_\theta = D_{\rm L}^2\triangle_\xi = \frac{D_{\rm L}^2}{\xi_0^2}\triangle_x , \tag{2.25}$$

using dimensionless quantities, Eq. 2.24 reads

$$\kappa(\vec{x}) = \frac{1}{2}\triangle_x\Psi(\vec{x}) \tag{2.26}$$

Integrating Eq. (2.16), the effective lensing potential can be written in terms of the convergence as

$$\Psi(\vec{x}) = \frac{1}{\pi}\int_{\mathbf{R}^2}\kappa(\vec{x}')\ln|\vec{x}-\vec{x}'|{\rm d}^2x' , \tag{2.27}$$

from which we obtain that the scaled deflection angle is

$$\vec{\alpha}(\vec{x}) = \frac{1}{\pi}\int_{\mathbf{R}^2}{\rm d}^2x'\kappa(\vec{x}')\frac{\vec{x}-\vec{x}'}{|\vec{x}-\vec{x}'|} . \tag{2.28}$$

2.3 First order lens mapping

One of the main consequences of gravitational lensing is image distortion. This is particularly evident when the source has an extended size. For example, background galaxies can appear as very long arcs when lensed by galaxy clusters or other galaxies.

The distortion arises because light bundles are deflected differentially. Ideally, the shape of the images can be determined by solving the lens equation for all the points within the extended source. In particular, if the source is much smaller than the angular scale on which the lens deflection angle field changes, the relation between source and image positions can locally be linearized.

This situation is sketched in Fig. 2.3.1. Let us consider a point on the lens (or image) plane at position $\vec{\theta}$, where the deflection angle is $\vec{\alpha}$. If the deflection angle satisfies the above conditions, at the nearby location $\vec{\theta}' = \vec{\theta} + d\vec{\theta}$, the deflection will be

$$\vec{\alpha}' \simeq \vec{\alpha} + \frac{d\vec{\alpha}}{d\vec{\theta}}d\vec{\theta} . \tag{2.29}$$

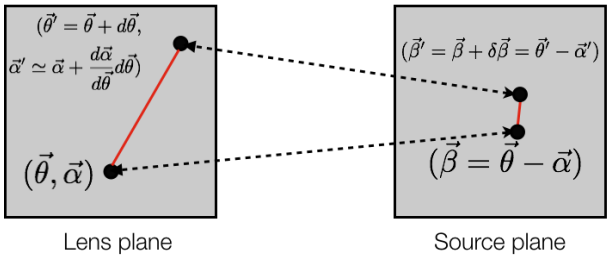


Figure 2.3.1: *Linear mapping between the lens and the source plane, assuming a slowly varying deflection angle.*

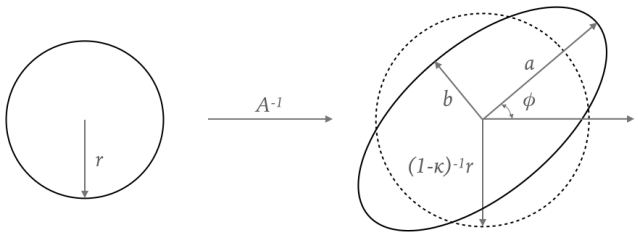


Figure 2.3.2: *Distortion effects due to convergence and shear on a circular source.*

Using the lens equation, the points $\vec{\theta}$ and $\vec{\theta}'$ can be mapped on the points $\vec{\beta}$ and $\vec{\beta}' = \vec{\beta} + d\vec{\beta}$ onto the source plane. Through this mapping, the vector $(\vec{\beta}' - \vec{\beta})$ is given by

$$(\vec{\beta}' - \vec{\beta}) = \left(I - \frac{d\vec{\alpha}}{d\vec{\theta}} \right) (\vec{\theta}' - \vec{\theta}) \, . \tag{2.30}$$

In other words, the distortion of images can be described by the Jacobian matrix

$$A \equiv \frac{\partial \vec{\beta}}{\partial \vec{\theta}} = \left(\delta_{ij} - \frac{\partial \alpha_i(\vec{\theta})}{\partial \theta_j} \right) = \left(\delta_{ij} - \frac{\partial^2 \hat{\Psi}(\vec{\theta})}{\partial \theta_i \partial \theta_j} \right) \, , \tag{2.31}$$

where θ_i indicates the i-component of $\vec{\theta}$ on the lens plane.

Eq. (2.31) shows that the elements of the Jacobian matrix can be written as combinations of the second derivatives of the lensing potential. For brevity, we will use the shorthand notation

$$\frac{\partial^2 \hat{\Psi}(\vec{\theta})}{\partial \theta_i \partial \theta_j} \equiv \hat{\Psi}_{ij} \, . \tag{2.32}$$

We can now split off the isotropic part from the Jacobian, to obtain its traceless part:

$$\left(A - \frac{1}{2} \mathrm{tr} A \cdot I \right)_{ij} = \delta_{ij} - \hat{\Psi}_{ij} - \frac{1}{2}(1 - \hat{\Psi}_{11} + 1 - \hat{\Psi}_{22})\delta_{ij} \tag{2.33}$$

$$= -\hat{\Psi}_{ij} + \frac{1}{2}(\hat{\Psi}_{11} + \hat{\Psi}_{22})\delta_{ij} \tag{2.34}$$

$$= \begin{pmatrix} -\frac{1}{2}(\hat{\Psi}_{11} - \hat{\Psi}_{22}) & -\hat{\Psi}_{12} \\ -\hat{\Psi}_{12} & \frac{1}{2}(\hat{\Psi}_{11} - \hat{\Psi}_{22}) \end{pmatrix} \, . \tag{2.35}$$

This allows us to define the *shear* tensor

$$\Gamma = \begin{pmatrix} \gamma_1 & \gamma_2 \\ \gamma_2 & -\gamma_1 \end{pmatrix}, \tag{2.36}$$

often written in the form of a pseudo-vector $\vec{\gamma} = (\gamma_1, \gamma_2)$, whose components are

$$\gamma_1 = \frac{1}{2}(\hat{\Psi}_{11} - \hat{\Psi}_{22}) \tag{2.37}$$

$$\gamma_2 = \hat{\Psi}_{12} = \hat{\Psi}_{21}\,. \tag{2.38}$$

The shear is manifestly an symmetric, traceless tensor. It quantifies the projection of the gravitational tidal field (the gradient of the gravitational force), which describes distortions of background sources.

The eigenvalues of the shear tensor are

$$\pm\sqrt{\gamma_1^2 + \gamma_2^2} = \pm\gamma\,. \tag{2.39}$$

Thus, there exist a rotation $R(\varphi)$ such that the shear tensor (and therefore the Jacobian) can be written in a diagonal form. Generally, the Jacobian transforms as

$$A \to A' = R(\varphi)^T A R(\varphi) \tag{2.40}$$

where T indicates the transposed matrix. This shows that the shear components transform under rotations as

$$\begin{aligned} \gamma_1 &\to \gamma_1' = \gamma_1\cos(2\varphi) + \gamma_2\sin(2\varphi) \\ \gamma_2 &\to \gamma_2' = -\gamma_1\sin(2\varphi) + \gamma_2\cos(2\varphi)\,, \end{aligned} \tag{2.41}$$

unlike a vector. Since the shear components are invariant under rotations of $\varphi = \pi$ rather that $\varphi = 2\pi$, they form a spin-2 tensor.

We can write the shear tensor as

$$\begin{pmatrix} \gamma_1 & \gamma_2 \\ \gamma_2 & -\gamma_1 \end{pmatrix} = \gamma \begin{pmatrix} \cos 2\phi & \sin 2\phi \\ \sin 2\phi & -\cos 2\phi \end{pmatrix}, \tag{2.42}$$

where we have introduced the angle ϕ, which identifies the direction of the eigenvector of the shear tensor corresponding to the eigenvalue $+\gamma$.

R Note the factor 2 on the angle ϕ, which reminds that the shear component are elements of a 2×2 tensor and not a vector.

The remainder of the Jacobian is

$$\frac{1}{2}\mathrm{tr}A \cdot I = \left[1 - \frac{1}{2}(\hat{\Psi}_{11} + \hat{\Psi}_{22})\right]\delta_{ij} \tag{2.43}$$

$$= \left(1 - \frac{1}{2}\triangle\hat{\Psi}\right)\delta_{ij} = (1-\kappa)\delta_{ij}\,. \tag{2.44}$$

Thus, the Jacobian matrix becomes

$$\begin{aligned} A &= \begin{pmatrix} 1-\kappa-\gamma_1 & -\gamma_2 \\ -\gamma_2 & 1-\kappa+\gamma_1 \end{pmatrix} \\ &= (1-\kappa)\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \gamma\begin{pmatrix} \cos 2\phi & \sin 2\phi \\ \sin 2\phi & -\cos 2\phi \end{pmatrix}. \end{aligned} \tag{2.45}$$

The last equation explains the meaning of both convergence and shear. The convergence determines an isotropic transformation, i.e. the images are only rescaled by a constant factor $1/(1-\kappa)$ in all directions. On the other hand, the shear stretches the intrinsic shape of the source along privileged directions. Specifically, the stretch corresponds to an extra term $+\gamma$ in the direction set by angle ϕ and $-\gamma$ in the perpendicular direction. Indeed, the eigenvalues of the Jacobian matrix are

$$\lambda_t = 1-\kappa-\gamma \tag{2.46}$$
$$\lambda_r = 1-\kappa+\gamma\,. \tag{2.47}$$

2.3.1 Lensing of circular source

Let consider the reference frame where the Jacobian is diagonal. Then,

$$A = \begin{pmatrix} 1-\kappa-\gamma & 0 \\ 0 & 1-\kappa+\gamma \end{pmatrix}. \tag{2.48}$$

Consider a circular source, whose isophotes have equation $\beta_1^2+\beta_2^2=r^2$. The lens equation implies that the points on the source plane satisfying this equation are mapped onto the points (θ_1,θ_2), such that

$$\begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} 1-\kappa-\gamma & 0 \\ 0 & 1-\kappa+\gamma \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}. \tag{2.49}$$

Thus

$$\beta_1 = (1-\kappa-\gamma)\theta_1 \tag{2.50}$$
$$\beta_2 = (1-\kappa+\gamma)\theta_2\,. \tag{2.51}$$

Summing in quadrature, we obtain

$$r^2=\beta_1^2+\beta_2^2=(1-\kappa-\gamma)^2\theta_1^2+(1-\kappa+\gamma)^2\theta_2^2\,, \tag{2.52}$$

which is the equation of an ellipse on the lens plane. Thus, a circular source, which is small enough compared to the typical length-scale over which the lens deflection field varies, like is mapped onto an ellipse when κ and γ are both non-zero, as shown in Fig. (2.3.2).

The semi-major and -minor axes of the ellipse are

$$a=\frac{r}{1-\kappa-\gamma}\quad,\quad b=\frac{r}{1-\kappa+\gamma}\,. \tag{2.53}$$

Obviously, the ellipse reduces to a circle if $\gamma=0$.

As said in the previous section, in an arbitrary reference frame, the ellipse will have its axes aligned with the eigenvectors of the shear tensor. Note that:

- if $\gamma_1>0$ and γ_2=0, then the major axis of the ellipse will be along the θ_1 axis;
- if $\gamma_1=0$ and γ_2>0, then the major axis of the ellipse will form an angle $\pi/4$ with the θ_1 axis;
- if $\gamma_1<0$ and γ_2=0, then the major axis of the ellipse will be perpendicular to the θ_1 axis;
- if $\gamma_1=0$ and γ_2<0, then the major axis of the ellipse will form an angle $3\pi/4$ with the θ_1 axis.

In Fig. 2.3.3, the ellipse orientation is shown for different values of the two components of the shear.

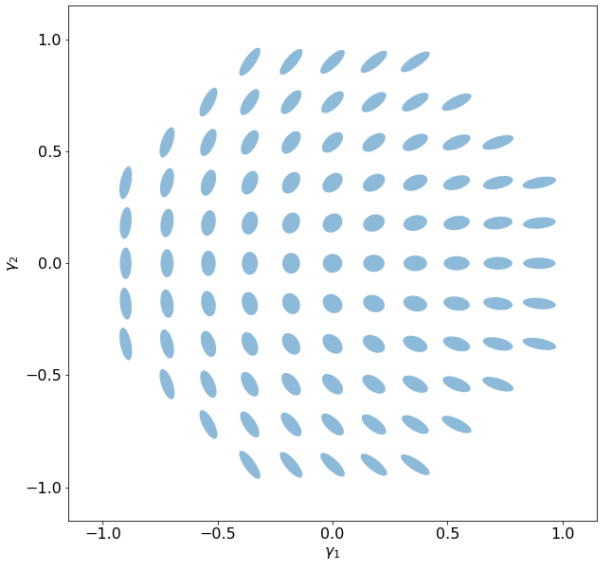


Figure 2.3.3: *Orientation of the images of a circular source for different values of γ_1 and γ_2.*

2.4 Magnification

An important consequence of the lensing distortion is the magnification. Through the lens equation, the solid angle element $\delta\beta^2$ (or equivalently the surface element $\delta\eta^2$ or δy^2) is mapped onto the solid angle $\delta\theta^2$ (or on the surface element $\delta\xi^2$ or δx^2). Because of the Liouville theorem, in absence of emission and absorbtion of photons, the source surface brightness is conserved despite light deflection. Thus, the change of the solid angle under which the source is observed implies that the flux received is magnified (or demagnified).

Given Eq. (2.31), the *magnification* is given by the inverse of the determinant of the Jacobian matrix. For this reason, the matrix $M = A^{-1}$ is called the *magnification tensor*. We therefore define

$$\mu \equiv \det M = \frac{1}{\det A} = \frac{1}{(1-\kappa)^2 - \gamma^2} \,. \tag{2.54}$$

The eigenvalues of the magnification tensor (or the inverse of the eigenvalues of the Jacobian matrix) measure the amplification in the direction of the eigenvectors of the shear tensor. For an axially symmetric lens, these are tangentially and radially oriented with respect to the lens iso-surface density contours. Thus, the quantities

$$\mu_t = \frac{1}{\lambda_t} = \frac{1}{1-\kappa-\gamma} \tag{2.55}$$

$$\mu_r = \frac{1}{\lambda_r} = \frac{1}{1-\kappa+\gamma} \tag{2.56}$$

are often called the *tangential* and *radial* magnification factors.

The magnification is ideally infinite where $\lambda_t = 0$ and where $\lambda_r = 0$. These two conditions define two curves in the lens plane, called the *tangential* and the *radial critical lines*.

2.5 Lensing to the second order

We extend now the lens equation including the second order terms in the expansion of the deflection angle. The lens equation then becomes

$$\beta_i \simeq \frac{\partial \beta_i}{\partial \theta_j}\theta_j + \frac{1}{2}\frac{\partial^2 \beta_i}{\partial \theta_j \partial \theta_k}\theta_j \theta_k \, . \tag{2.57}$$

We introduce the tensor

$$D_{ijk} = \frac{\partial^2 \beta_i}{\partial \theta_j \partial \theta_k} = \frac{\partial A_{ij}}{\partial \theta_k} \, . \tag{2.58}$$

Then, Eq. 2.57 reads

$$\beta_i \simeq A_{ij}\theta_j + \frac{1}{2} D_{ijk}\theta_j \theta_k \tag{2.59}$$

By simple algebra, it can be shown that

$$D_{ij1} = \begin{pmatrix} -2\gamma_{1,1} - \gamma_{2,2} & -\gamma_{2,1} \\ -\gamma_{2,1} & -\gamma_{2,2} \end{pmatrix} , \tag{2.60}$$

and

$$D_{ij2} = \begin{pmatrix} -\gamma_{2,1} & -\gamma_{2,2} \\ -\gamma_{2,2} & 2\gamma_{1,2} - \gamma_{2,1} \end{pmatrix} . \tag{2.61}$$

Thus, the second order lensing effect can be expressed in terms of the derivatives of the shear (or in terms of the third derivatives of the potential).

2.5.1 Complex notation

It is quite useful to use complex notation to map vectors or pseudo-vectors on the complex plane. Indeed, in this case we can also use complex differential operators to write down some relations between the lensing quantities in a very concise way.

In complex notation, any vector or pseudo-vector $v = (v_1, v_2)$ is written as

$$v = v_1 + i v_2 \, . \tag{2.62}$$

Similarly we can define the complex deflection angle $\alpha = \alpha_1 + i\alpha_2$ and the complex shear $\gamma = \gamma_1 + i\gamma_2$.

It is also possible to define some complex differential operators, namely

$$\partial = \partial_1 + i\partial_2 \tag{2.63}$$

and

$$\partial^\dagger = \partial_1 - i\partial_2 \, . \tag{2.64}$$

Using this formalism, we can easily see that

$$\partial\hat{\Psi} = \partial_1\hat{\Psi} + i\partial_2\hat{\Psi} = \alpha_1 + i\alpha_2 = \alpha \, . \tag{2.65}$$

Moreover

$$\partial^\dagger\partial = \partial_1^2 + \partial_2^2 = \triangle \, . \tag{2.66}$$

Thus,

$$\partial^\dagger \partial \hat{\Psi} = \triangle \hat{\Psi} = 2\kappa \,. \tag{2.67}$$

Note that while $\hat{\Psi}$ is a spin-0 scalar field, the application of the ∂ operator gives the deflection angle, i.e. a spin-1 vector field. On the contrary, the $\partial^\dagger$ operator applied to the deflection field gives another spin-0 scalar field (the convergence). Therefore, the ∂ and $\partial^\dagger$ operators can be considered as spin raising and lowering operators.

By applying twice the raising operator, we obtain

$$\frac{1}{2}\partial\partial\hat{\Psi} = \frac{1}{2}\partial\alpha = \gamma \,: \tag{2.68}$$

the shear field is indeed a spin-2 tensor field, which is invariant for rotations by multiples of π.

Note also that

$$\partial^{-1}\partial^\dagger\gamma = \frac{1}{2}\partial^{-1}\partial^\dagger\partial\partial\hat{\Psi} = \partial^\dagger\partial\hat{\Psi} = \kappa \tag{2.69}$$

We can use the raising and lowering operators to define

$$F = \frac{1}{2}\partial\partial^\dagger\partial\hat{\Psi} = \partial\kappa \tag{2.70}$$

$$G = \frac{1}{2}\partial\partial\partial\hat{\Psi} = \partial\gamma \tag{2.71}$$

After some math, it can be shown that

$$F = F_1 + iF_2 = (\gamma_{1,1} + \gamma_{2,2}) + i(\gamma_{2,1} - \gamma_{1,2}) \tag{2.72}$$

and

$$G = G_1 + iG_2 = (\gamma_{1,1} - \gamma_{2,2}) + i(\gamma_{2,1} + \gamma_{1,2}) \,. \tag{2.73}$$

The quantities F and G are called *first and second flexion*, respectively. It is easy to show that D_{ijk} can be written in terms of F and G. Thus, they describe second order distortions of the images of lensed sources. Note that F is a spin-1 vector field. Indeed, it is

$$\vec{F} = \vec{\nabla}\kappa \,. \tag{2.74}$$

Thus, it describes transformations that are invariant under rotations by 2π. For this reason, F stretches the images along one particular direction, introducing asymmetries in their shape. On the contrary, G is a spin-3 tensor field. The transformations described by G are invariant under rotations by $2\pi/3$. This is manifested in the "triangular" pattern in the image shapes, as shown in Fig. 2.5.1.

2.6 Time delay surface

2.6.1 Gravitational and geometrical time delays

The deflection of light rays causes a delay in the travel-time of light between the source and the observer. This time delay has two components:

$$t = t_{\rm grav} + t_{\rm geom} \tag{2.75}$$

The first one is the *gravitational time delay*, also known as the Shapiro delay. It can be derived by comparing the time required to light to travel through a space-time with an effective refractive index and through empty space, by assuming *same trajectories*.

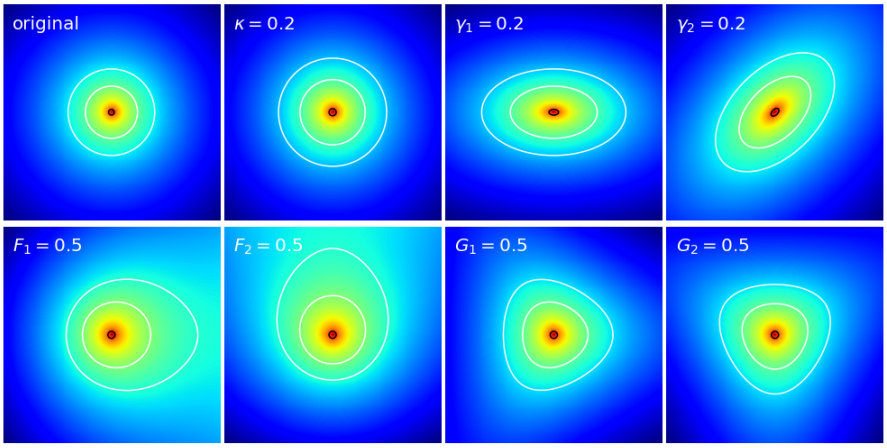


Figure 2.5.1: *First and second order distortions on the image of a circular source. The unlensed source is shown in the top left panel. The convergence simply changes the size (second-left upper panel). While the shear deforms the image such that it becomes elliptical (third and fourth panels on the upper row), the first and the second flexion introduce curvature and other distortions (panels on the bottom).*

Let $n = 1 - 2\Phi/c^2$ be the effective refractive index. We have that

$$t_{\rm grav} = \int \frac{dz}{c'} - \int \frac{dz}{c} = \frac{1}{c}\int (n-1)dz = -\frac{2}{c^3}\int \Phi {\rm d}z \tag{2.76}$$

Using the definition of the lensing potential, this can be written as

$$t_{\rm grav} = -\frac{D_{\rm L}D_{\rm S}}{D_{\rm LS}}\frac{1}{c}\hat{\Psi}\,. \tag{2.77}$$

The second term in the time delay is called *geometrical* and is due to the different path length of the deflected light rays compared to the unperturbed ones. This time delay is proportional to the squared angular separation between the intrinsic position of the source and the location of its image. This result can be derived from the metric, but it can be estimated also through a simple geometrical construction, shown in Fig. 2.6.1. The extra-path of the light in presence of the lens can be written as

$$\Delta l = \xi\frac{\hat{\vec{\alpha}}}{2} = (\vec{\theta}-\vec{\beta})\frac{D_{\rm L}D_{\rm S}}{D_{\rm LS}}\frac{\vec{\alpha}}{2} = \frac{1}{2}(\vec{\theta}-\vec{\beta})^2\frac{D_{\rm L}D_{\rm S}}{D_{\rm LS}}\,, \tag{2.78}$$

and the corresponding time-delay is

$$t_{\rm geom} = \frac{\Delta l}{c} \tag{2.79}$$

Both the time delays occur at the lens position, thus they need to be multiplied by a factor $(1+z_{\rm L})$ for accounting for the expansion of the universe. Then, the total time delay introduced by

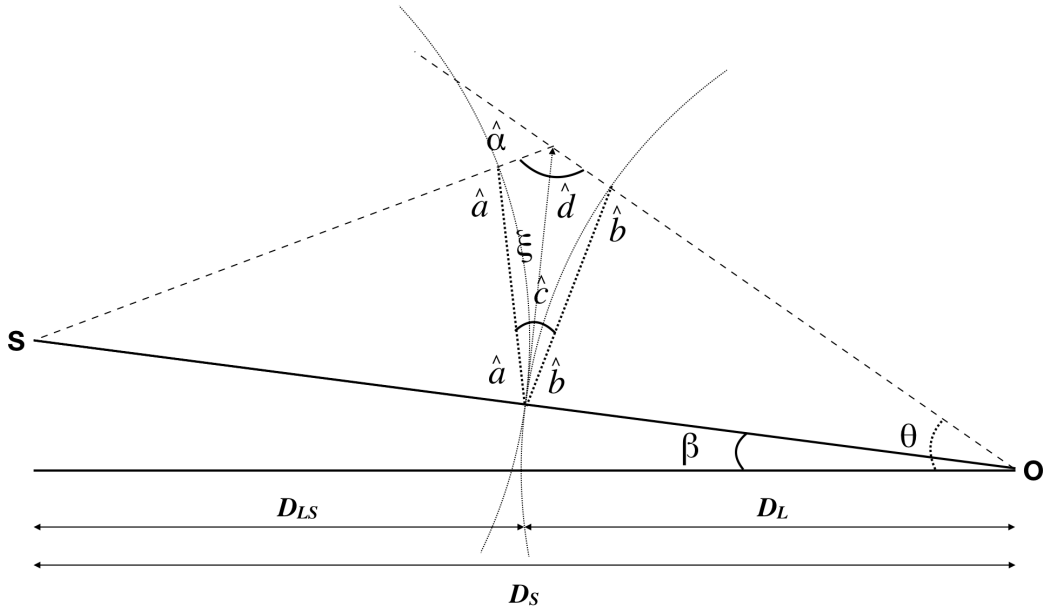


Figure 2.6.1: *Illustration of the geometrical time delay.*

gravitational lensing at the position $\vec{\theta}$ on the lens plane is[1]

$$\begin{aligned} t(\vec{\theta}) &= \frac{(1+z_{\rm L})}{c}\frac{D_{\rm L}D_{\rm S}}{D_{\rm LS}}\left[\frac{1}{2}(\vec{\theta}-\vec{\beta})^2-\hat{\Psi}(\vec{\theta})\right] \\ &= \frac{D_{\Delta t}}{c}\tau(\vec{\theta})\,. \end{aligned} \tag{2.80}$$

The quantities

$$D_{\Delta t} = (1+z_{\rm L})\frac{D_{\rm S}D_{\rm L}}{D_{\rm LS}} \tag{2.81}$$

and

$$\tau(\vec{\theta}) = \frac{1}{2}(\vec{\theta}-\vec{\beta})^2-\hat{\Psi}(\vec{\theta})\,, \tag{2.82}$$

are often called *time delay distance* and *Fermat potential*, respectively.

2.6.2 Multiple images and magnification

Through the effective lensing potential, the lens equation can be written as

$$(\vec{\theta}-\vec{\beta})-\nabla\hat{\Psi}(\vec{\theta}) = \nabla\left[\frac{1}{2}(\vec{\theta}-\vec{\beta})^2-\hat{\Psi}(\vec{\theta})\right] = 0\,. \tag{2.83}$$

Eqs. (2.80) and (2.83) imply that images satisfy the Fermat Principle, $\nabla t(\vec{\theta}) = 0$. Images therefore are located at the stationary points of the time delay surface given by Eq. (2.80). The Hessian matrix of this surface is

$$T = \frac{\partial^2 t(\vec{\theta})}{\partial\theta_i\partial\theta_j} \propto (\delta_{ij}-\hat{\Psi}_{ij}) = A \tag{2.84}$$

[1] The dimensionless form of the time delay can be obtained by multiplying and dividing by the factor $(\xi_0/D_{\rm L})^2$.

Given that the Hessian matrix of the time delay surface coincides with the lensing Jacobian, and that the magnification $\mu = \det A^{-1}$, it is clear that the curvature of the time delay surface at the image position is inversely proportional to the image magnification. In particular, a flat time delay surface implies as infinite magnification, while a large curvature means that the magnification is small.

One can also measure the curvature along a specific direction on the time delay surface. This will provide a way to quantify the image dostortions. Therefore, the shape of the time delay surface near the stationary points will also provide hints on the shape of the images.

We can distinguish between three types of images:

1. type I images arise at the minima of the time delay surface, where the eigenvalues of the Hessian matrix are both positive, hence $\det A > 0$ and $\mathrm{tr} A > 0$. Therefore, they have positive magnifications;
2. type II images arise at the saddle points of the time delay surface, where eigenvalues have opposite signs. Since $\det A < 0$, they have negative magnifications;
3. finally, type III images arise at the maxima of the time delay surface. Here, the eigenvalues are both negative, hence $\det A > 0$ and $\mathrm{tr} A < 0$. These images therefore have positive magnification.

R Note that a negative magnification does not mean that the image is de-magnified! The absolute value if the magnification accounts for how much larger is the solid angle of the image with respect to that of the unlensed source. Thus, the image is de-magnified only if $|\mu| < 1$. Instead, the sign of the magnification is related to the *parity* of the image. The parity determines the orientation of the image with respect to the unlensed source.

2.6.3 Examples

Eq. 2.80 shows that the time delay surface is obtained by summing the paraboloid $\propto (\vec{\theta} - \vec{\beta})^2$, which has a minimum at the position of the source, and the surface $-\hat{\Psi}(\vec{\theta})$. The lensing potential of a centrally concentrated lens has a minimum at the lens center. Thus, because of the negative sign, the function $-\hat{\Psi}$ peaks at the center of the lens, regardless of the position of the source. For simplicity, we choose the reference frame, (θ_1, θ_2) such that the lens center is at $(0,0)$, and we study how the shape of the time delay surface changes as a function the position of the source, $\vec{\beta}$.

Axially symmetric lenses: one dimensional case

We begin with an axially symmetric lens. Let us forget for the moment that the $t(\vec{\theta}|\beta)$ is a surface and consider the azimuthal cut of the surface along an arbitrary direction passing through the center of the lens and the source position. As an example, we consider the potential

$$\hat{\Psi}(\theta) \propto \frac{1}{\sqrt{\theta^2 + \theta_c^2}} \,. \tag{2.85}$$

As we will see in Chapter **??**, this potential corresponds to a cored isothermal lens. The core radius θ_c prevents the potential from diverging for $\theta \to 0$.

In Fig. 2.6.2, we show the geometrical and the gravitational components of the time delay and their combination for a few positions of the source relative to the lens, given by the vertical dashed lines. Given that β is a parameter defining the shape of the time delay surface, we use the notation $t(\theta) \equiv t(\theta, \beta)$. For $\beta = 0$ (upper left panel), the time delay function $t(\theta, 0)$ has a local maximum at $\theta_0 = 0$, and two minima on both sides of the origin, θ_- and θ_+ (of course, in this one-dimensional example there are no saddle points). Thus, the source at $\beta = 0$ has three images, forming at θ_0, θ_- and θ_+, with $\theta_- = -\theta_+$.

We shift the source with respect to the lens along the positive θ axis, and we notice that the symmetry of the time delay function breaks. In the upper-right panel, the maximum is now

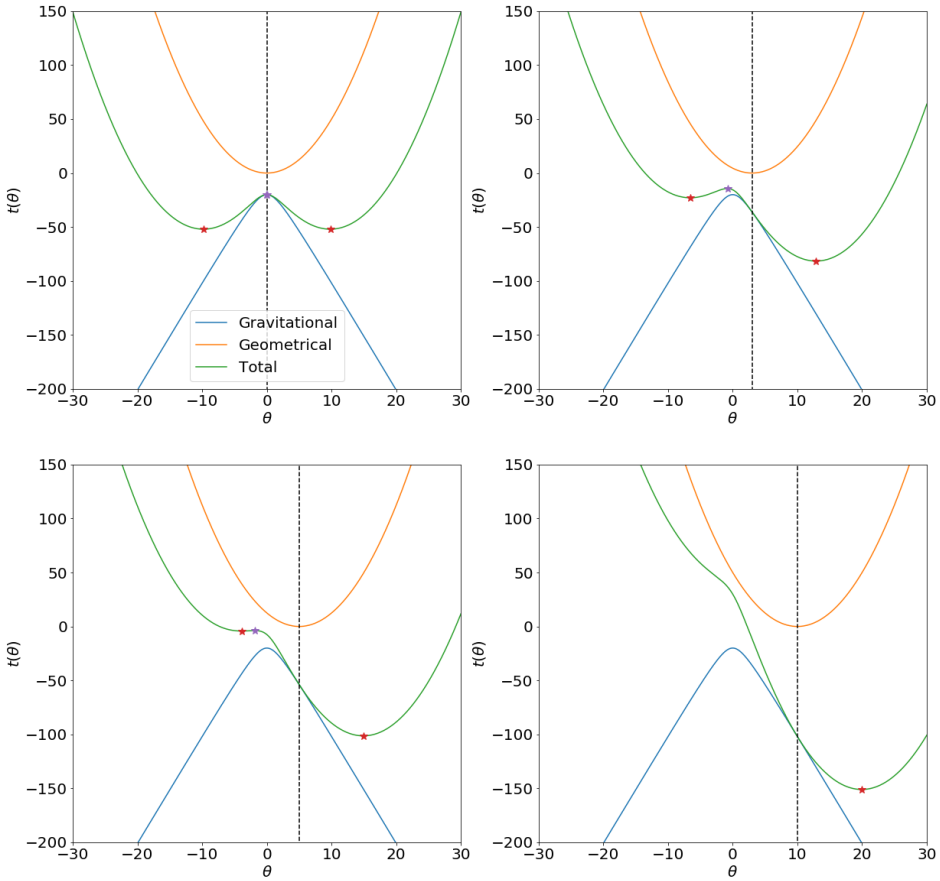


Figure 2.6.2: *One-dimensional time-delay functions for a non-singular isothermal potential. Each panel corresponds to a different position of the source relative to the lens (dashed line). The two components of the time delay are shown separately and then combined. The positions of the images are indicated by stars.*

shifted along the negative θ axis, $\theta_0 < 0$. One of the two minima, θ_+ moves away from the origin, following the source (i.e. along the positive θ axis), while the other moves towards the maximum. Note also that the difference between the time delays of the images θ_0 and θ_-, $t(\theta_0,\beta) - t(\theta_-,\beta)$, is smaller than in the previous case ($\beta = 0$). On the other hand, $t(\theta_+,\beta) < t(\theta_+,0)$.

Let us focus on the curvature of $t(\theta,\beta)$. Clearly, moving the source along the positive θ axis, $t(\theta,\beta)$ flattens off in between θ_- and θ_0. This implies an increasing magnification along the direction connecting the two images. Therefore, these will be stretched towards each other.

Moving the source further away from the lens, we will reach the situation where the two images θ_0 and θ_- will merge. At that point, the function $t(\theta,\beta)$ will only have one minimum, corresponding to the image θ_+. The source will no longer have multiple images (bottom left panel). As $\beta - \rightarrow \infty$, the image θ_+ will tend to coincide with β (bottom right panel).

It is interesting to see how the results are affected by the choice of the lensing potential. The

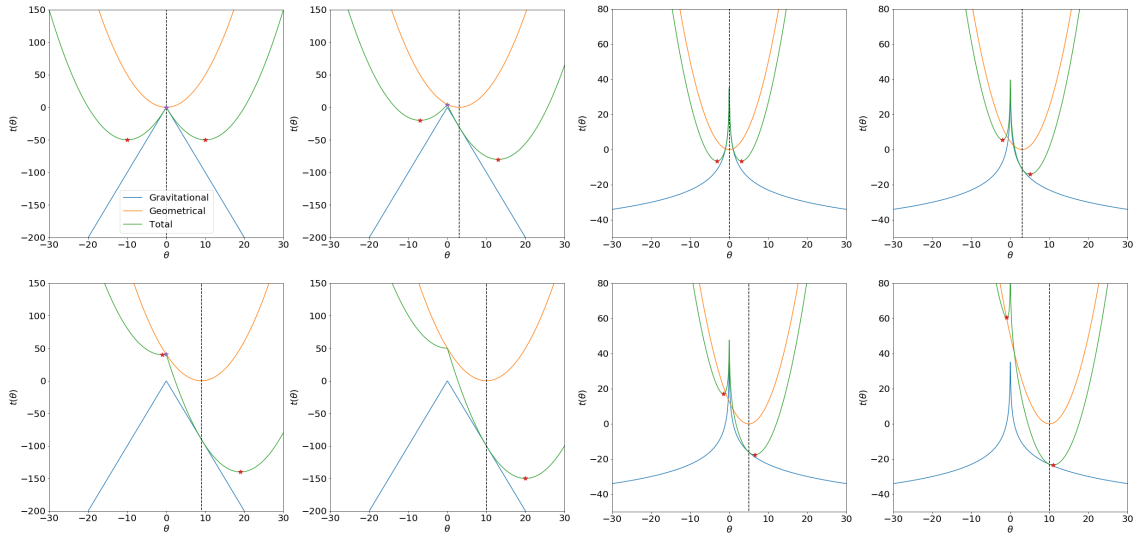


Figure 2.6.3: *One-dimensional time-delay functions for singular-isothermal (left panels) and point-mass (right panels) potentials. Each panel corresponds to a different position of the source relative to the lens (dashed line). The two components of the time delay are shown separately and then combined. The positions of the images are indicated by stars.*

examples in Fig. 2.6.3 show the results obtained by setting the core radius to $\theta_c = 0$ (left four panels) and by using a potential in the form

$$\hat{\Psi} \propto \ln|\theta| \,. \tag{2.86}$$

The second of these cases (right four panels) corresponds to the point-mass lens.

In both cases, the presence of the central singularity makes the function $t(\theta,\beta)$ non continuously deformable. This implies that, for every choice of β, the central image θ_0, if any, will correspond to infinite curvature of the time-delay surface. Consequently, for these images the magnification will be $\mu = 0$. Note that, in the case of the point-mass lens, there are always two minima on the opposite sides of the lens. However, as $\beta \to \infty$, the curvature at $t(\theta_-,\beta)$ becomes increasingly higher, meaning that the image is increasingly de-magnified.

Axially symmetric lenses: two dimensional case

The correct representation of the time delay is through a surface, not a one-dimensional function. In Fig. 2.6.4, we show the two-dimensional analog of Fig 2.6.2, where the time-delay surfaces correspond to several positions of the source along the θ_1 axis (i.e. $\beta_2 = 0$). We also show the projection of the surfaces on the (θ_1,θ_2) planes and the sections of the surfaces along the θ_1 axis ($\theta_2 = 0$).

The upper left panel shows the time-delay surface $t(\vec{\theta},0)$. In this two-dimensional representation, we see that the minima $\vec{\theta}_-$ and $\vec{\theta}_+$ are part of a ring, due to the symmetry properties of the lens. We will see later that this ring is called *Einstein ring*. The central image, $\vec{\theta}_0$, still coincides with the center of the lens. Thus a point-source perfectly aligned with this axially symmetric (circular) lens is imaged into a point at the lens center and into a ring surrounding the lens.

Along the ring, the curvature of the time-delay surface is zero, meaning that the magnification diverges. This condition is met along the lens critical lines. More precisely, the Einstein ring corresponds to the tangential critical line of the lens. Given that the ring is originated by a source at $\vec{\beta} = (0,0)$, this point on the source plane coincides with the lens tangential caustic.

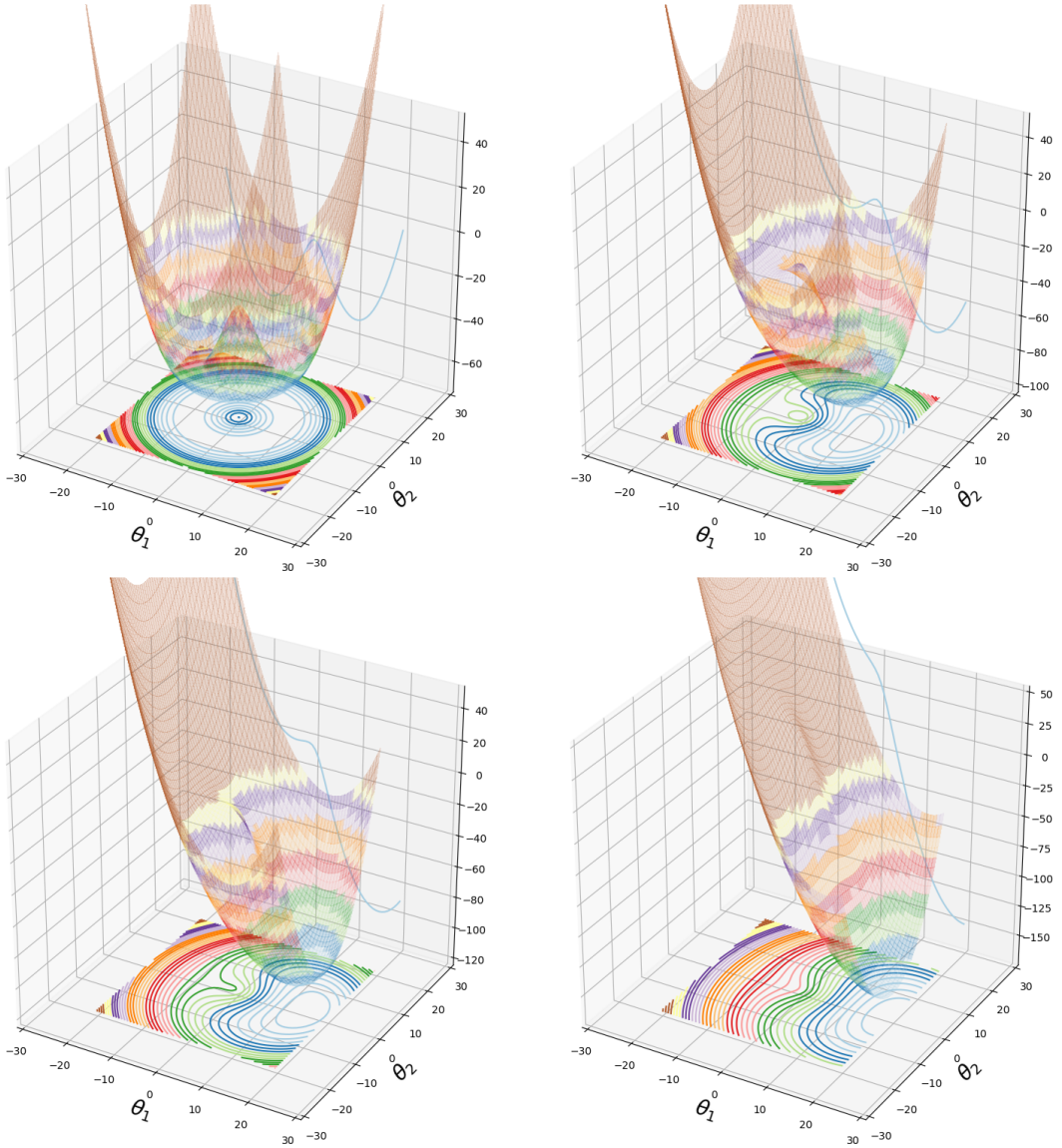


Figure 2.6.4: *Time delay surfaces for the same lens used in Fig. 2.6.2. Different panels correspond to different positions of the source relative to the lens. The surfaces are projected onto the plane* (θ_1, θ_2) *on the bottom of each panel. The blue curves on the vertical plane behind the surfaces show the sections of the surfaces along the* θ_1 *axis at* $\theta_2 = 0$.

Shifting the source with respect to the lens breaks the symmetry of the time-delay surface. In the upper-right panel of Fig. 2.6.4, we notice that the image at $\vec{\theta}_-$ is not a minimum, but a saddle point. As the source moves away from the lens, the saddle and the maximum points approach each other and the curvature of $t(\vec{\theta}, \vec{\beta})$ in between these two stationary points becomes increasingly smaller in the radial direction. When the two images coincide, the time-delay surface is radially flat (bottom left panel): the images $\vec{\theta}_-$ and $\vec{\theta}_0$ merge on another critical line: the radial critical line. The corresponding source position $\vec{\beta} = \vec{\beta}_{rad}$ marks the position of the radial caustic on the source plane. Due to the symmetry of the lens, the radial caustic of an axially symmetric lens is a circle with radius β_{rad}.

For large distances between the lens and the source, only the image $\vec{\theta}_+$ exists, which corresponds to the minimum of the time-delay surface.

As discussed earlier, adopting singular potentials, the time-delay surface becomes non continuously deformable. As a consequence, there are no configurations where the time-delay surface can become radially flat. This implies that these lenses do not have a radial critical line. In the case of the singular-isothermal lens ($\theta_c = 0$), there is a particular distance of the source from the lens center, β_{cut}, for which $\vec{\theta}_- = 0$. This condition defines a circle on the source plane called *cut*. The lens produces two images only if the source lays within the cut. Otherwise there is only one image.

Elliptical potentials

While axially symmetric lenses can produce up to three multiple images, depending on the relative position of the source relative to the lens, elliptical lenses behave differently. We can introduce ellipticity in the potentials considered in any of the previous examples by making the substitution

$$|\theta| \to \sqrt{\frac{\theta_1^2}{1-\varepsilon} + \theta_2^2(1-\varepsilon)}\,. \tag{2.87}$$

The resulting lens have elliptical iso-potential contours with major axes oriented along the θ_2 axis.

R Lenses with elliptical potentials are not elliptical lenses. Indeed, their convergence maps do not have elliptical iso-contours. Instead, these contours typically have dumbbell shapes. Introducing large ellipticity in the potential can even lead to unphysical negative convergence. Lenses with elliptical potentials are dubbed *pseudo-elliptical* lenses.

When combined with the paraboloid describing the usual geometrical time delay, the resulting surface can have up to five stationary points, depending on the potential radial profile and of the relative positions of lens and source.

The examples displayed in Figs. 2.6.5 and 2.6.6 illustrate the case of a lens with cored isothermal potential and ellipticity $\varepsilon = 0.4$. The upper panels of Fig. 2.6.5 show the maps of the lensing potential before and after introducing the ellipticity. The left and the right bottom panels show the critical lines and the caustics of the lens, respectively. Fig. 2.6.6 displays the time delay surfaces corresponding to the source positions marked by the blue dots in the bottom left panel of Fig. 2.6.5. In the upper left panel (1), the source is at $\vec{\beta} = 0$. Such source have five multiple images: one maximum at the lens center, two minima symmetric with respect to the center of the lens along the θ_1 axis, and two saddle points, also symmetric with respect to the center of the lens, but forming along the θ_2 axis. This image configuration is called *Einstein cross*.

In the upper right panel (2), the source is shifted along the positive θ_1 axis. One of the two minima moves in the same direction, while the central maximum moves in the direction opposite to the source. The saddle points also move opposite to the source approaching the maximum and the minimum. Eventually, for even larger separations between the lens and the source, the saddle points, the maximum and the minimum merge, producing and image which is both radially and tangentially magnified. When this happens, the source is close to both the radial and the tangential caustics.

The middle panels show the case of a source moving along the positive θ_2 axis instead. Two minima and one saddle point move in the same direction. The maximum and the other saddle point approach each other along the negative θ_2 axis. As shown by the flatness of the time delay surface in between these two images, they are are radially magnified until the merge and disappear. This indicates that the source has crossed the radial caustic (3). Shifting the source further, also the two minima and the remaining saddle point merge, forming a very elongated, tangentially magnified image. Such configurations occur when the source is the proximity of the cusp of the caustic, while laying within the caustic itself (4). The largest gravitational arcs form in this way.

Finally, in the panels on the bottom the source is shifted along the diagonal in the (θ_1, θ_2) plane. As the source departs from the lens center, one of the minima follows the source. The other

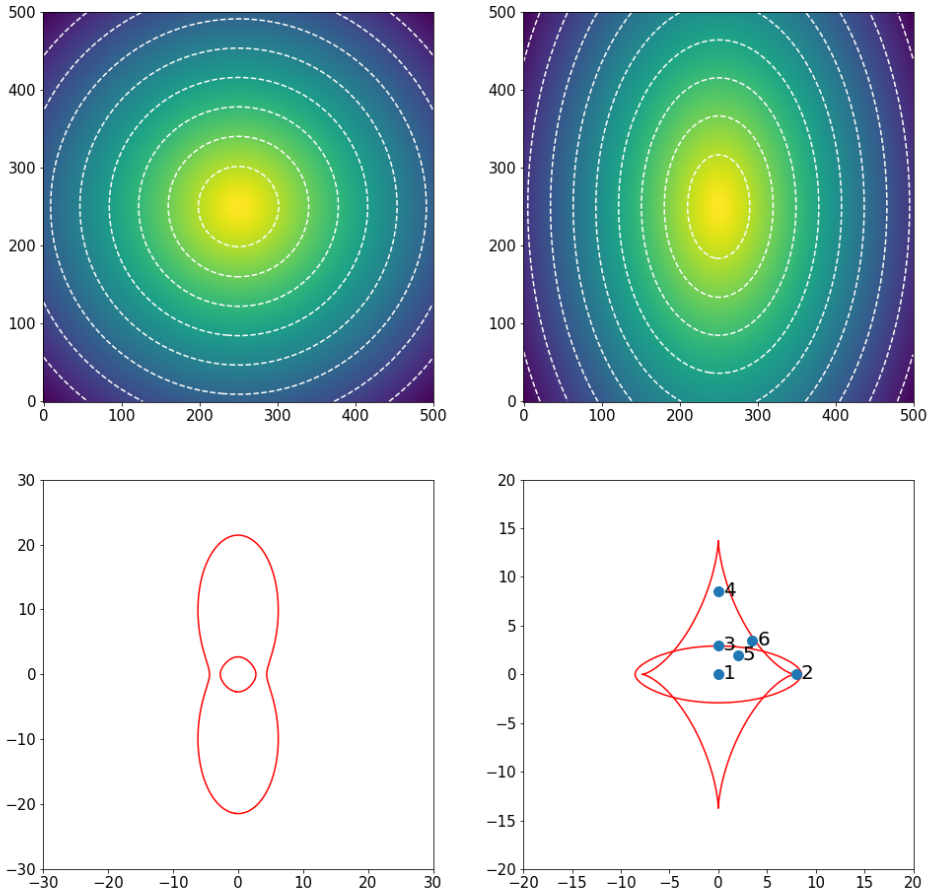


Figure 2.6.5: *Pseudo-elliptical lens. Upper panels: the lensing potential before and after adding an ellipticity $\varepsilon = 0.4$. Bottom panels: critical lines (left) and caustics (right). The blue dots mark the positions of the sources used to generate the time delay surfaces shown in Fig. 2.6.6.*

minimum and one of the saddle points approach each other. The maximum and the other saddle point merge forming a radially elongate image opposite to the source with respect to the lens (5). Again, this happens when the source is located at the radial caustic. Shifting it further, we see that also one of the minima and the remaining saddle point merge, forming a tangentially elongated image. At this point, the source is on the tangential caustic (6).

2.6.4 General considerations

Here follows a number of other important properties of the continuously deformable time-delay surface:

- the height difference at different images of the surface $t(\vec{\theta})$ gives the difference in arrival time between these images. This time delay can be measured if the source is variable, and provides one way of potentially measuring the Hubble constant, as we will discuss in Chapter **??**;

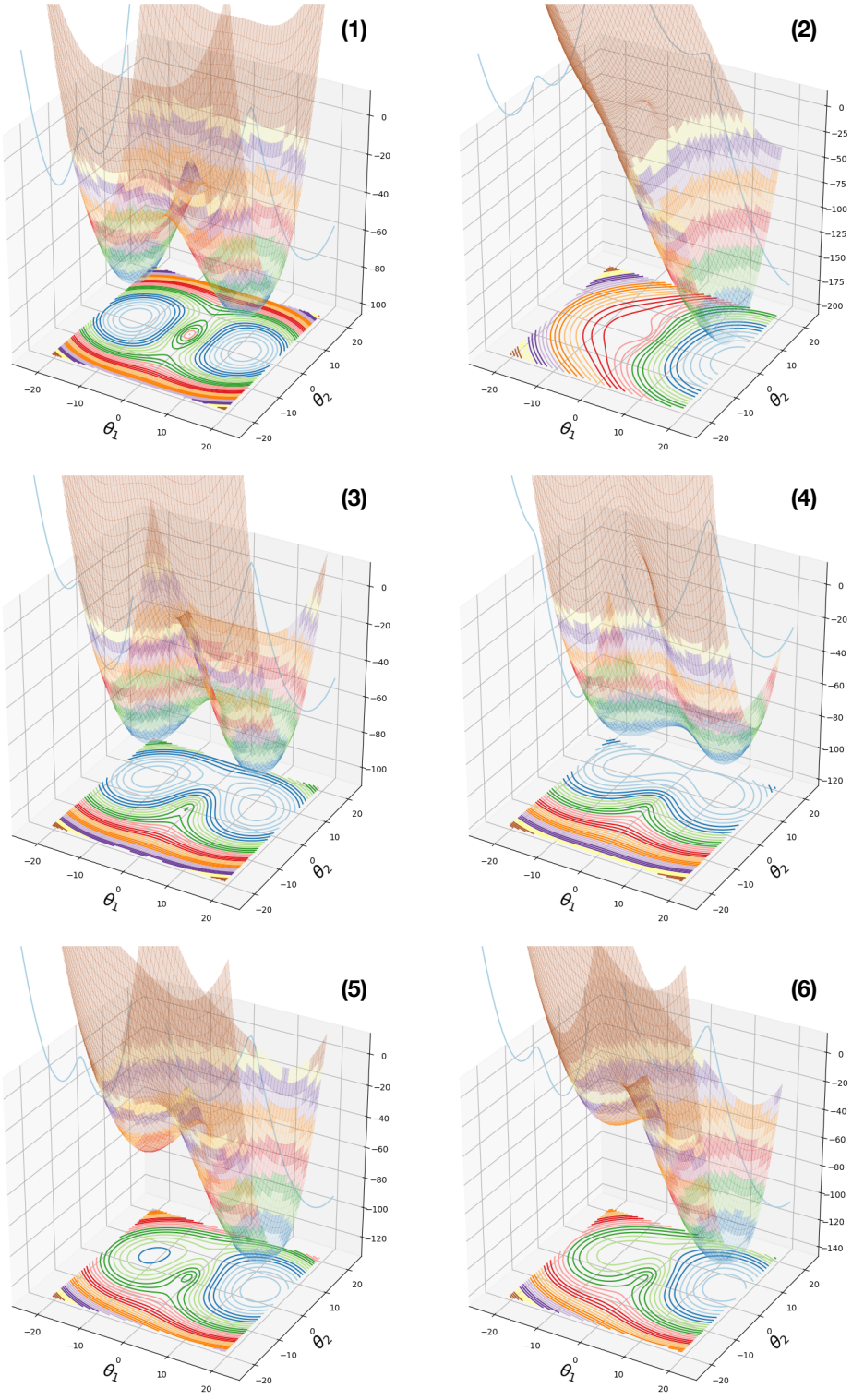


Figure 2.6.6: *Time delay surfaces for a pseudo-elliptical lens with cored isothermal profile. Different panels correspond to different positions of the source relative to the lens. The surfaces are projected onto the plane (θ_1, θ_2) on the bottom of each panel. The blue curves on the vertical plane behind the surfaces show the sections of the surfaces along the θ_1 axis at $\theta_2 = 0$.*

- in absence of the lens, the time-delay surface is a parabola which has a single extremum (a minimum); additional extrema have to come in pairs, thus the total number of images must be odd (as we showed earlier by continuously deforming the time-delay surface);
- when two additional images are formed, they must be a maximum and a saddle point; in between them, the curvature changes from negative to positive, thus it is zero between them; remember that $\det A = 0$ is the condition for having a critical point, where the magnification is (formally) infinite. The critical lines thus separate multiple-image pairs; these pairs merge and desappear (as discussed above) at the critical lines. In other words, the critical lines separate regions of different image multiplicities.

2.7 Python applications

2.7.1 Implementing a ray-tracing algorithm

In this exercise, we will implement a simple ray-tracing algorithm. In ray-tracing, we make use of the lens equation to propagate a bundle of light-rays from the observer position, through a regular grid covering the lens plane, to the source plane. For each ray passing through the position $\vec{x}^{ij}$, we will evaluate the deflection angle $\vec{\alpha}(\vec{x}^{ij})$ and compute the arrival position on the source plane as

$$\vec{y}^{ij} = \vec{x}^{ij} - \vec{\alpha}(\vec{x}^{ij}) \, .$$

In the formula above, (i, j) identify the ray passing through the grid point with indexes i and j along the x_1 and x_2 axes, respectively.

The deflector used in this example is the same of the previous exercise. In particular, we will use the deflection angle maps shown in the upper panels of the figure above to propagate the light rays towards the sources.

We start by creating a mesh, where each grid-point has two coordinates. Suppose coordinates along the x_1 and x_2 axes are represented by the n_{pix}-dimensional vectors $|x_1^i|$ and $|x_2^j|$, with $i, j \in [1, n_{pix}]$ (so that n_{pix} is the number of grid points along one axis on the mesh). The mesh can be created using the `numpy.meshgrid` method, as e.g.

```
npix=angx.shape[0]
x1=np.linspace(0.0,1.0,npix)*(npix-1) # define x1 coordinates
x2=np.linspace(0.0,1.0,npix)*(npix-1) # define x2 coordinates
x1_,x2_=np.meshgrid(x1,x2) # lens plane mesh
```

This will generate two numpy arrays, `x1_` and `x2_`, with size $n_{pix} \times n_{pix}$. In the first, the values on the $i-$th column will be equal to x_1^i; in the second, the values on the $j-$th row will be equal to x_2^j.

We may now implement the lens equation for the two components along x_1 and x_2:

```
y1=x1_-angx
y2=x2_-angy
```

In fact, we could arrive to the same result by using a feature in numpy called *broadcasting*. The term broadcasting describes how numpy treats arrays with different shapes during arithmetic operations. Subject to certain constraints, the smaller array is “broadcast" across the larger array so that they have compatible shapes.

Using this feature, we can write the first component of the lens equation as

$$\begin{bmatrix} y_1^{1,1} & \cdots & y_1^{1,n_{pix}} \\ \vdots & \ddots & \vdots \\ y_1^{n_{pix},1} & \cdots & y_1^{n_{pix},n_{pix}} \end{bmatrix} = B(n_{pix}, n_{pix}) \begin{bmatrix} x_1^1 \\ \vdots \\ x_1^{n_{pix}} \end{bmatrix} - \begin{bmatrix} \alpha_1^{1,1} & \cdots & \alpha_1^{1,n_{pix}} \\ \vdots & \ddots & \vdots \\ \alpha_1^{n_{pix},1} & \cdots & \alpha_1^{n_{pix},n_{pix}} \end{bmatrix}$$

The vector `x_1` is then broadcast to match the size of `alpha_1`‘ (the broadcasting function is here indicated as $B(n_{pix}, n_{pix})$). The result will be to add $|x_1^i|$ to each column of the matrix $-|\alpha_1^{ij}|$.

Computing the coordinates $|y_2^{ij}|$ involves few more steps. Again, using the lens equation, we obtain:

$$\begin{bmatrix} y_1^{1,1} & \cdots & y_1^{n_{pix},1} \\ \vdots & \ddots & \vdots \\ y_1^{1,n_{pix}} & \cdots & y_1^{n_{pix},n_{pix}} \end{bmatrix} = B(n_{pix}, n_{pix}) \begin{bmatrix} x_2^1 \\ \vdots \\ x_2^{n_{pix}} \end{bmatrix} - \begin{bmatrix} \alpha_1^{1,1} & \cdots & \alpha_1^{n_{pix},1} \\ \vdots & \ddots & \vdots \\ \alpha_1^{1,n_{pix}} & \cdots & \alpha_1^{n_{pix},n_{pix}} \end{bmatrix}$$

This equation implements the column-wise addition of $|x_2^i|$ to $-|\alpha_2^{ji}| = |\alpha_2^{ij}|^T$, where T indicates the transposed matrix. The result is $|y_2^{ji}| = |y_2^{ij}|^T$.

The python implementation is quite easy:

```
y1=(x1-angx) # y1 coordinates on the source plane
y2=np.transpose(x2-np.transpose(angy)) # y2 coordinates on the source plane
```

There is not much difference between this approach and the previous one in terms of execution time and memory usage.

This example builds on the deflection angles derived in Sect. 1.5.2, for a numerically simulated dark matter halo. In this case the lens is at redshift $z_L = 0.3$ and the source plane is at $z_S = 2$. The deflection angles are stored in the arrays `angx` and `angy` and the maps contain 512×512 pixels. In order to improve the visualization of the results, we downsample the maps by tracing a lower number of rays through the lens plane. We reduce the number of points on the lens plane mesh by a factor `ndown=16` along the two axes, x_1 and x_2.

```
ndown=16
x1=np.linspace(0.0,1.0,npix/ndown)*(npix-1) # dowsampled x1,x2 coordinates
x2=np.linspace(0.0,1.0,npix/ndown)*(npix-1) #
x1_,x2_=np.meshgrid(x1,x2) # downsampled grid
# now we need to interpolate the defl. angle maps at (x1_,x2_)
# we can use the method map_coordinates from scipy.ndimage
from scipy.ndimage import map_coordinates
# first, we need to reshape x1_ and y1_:
x=np.reshape(x1_,x1_.size)
y=np.reshape(x2_,x2_.size)
# then we interpolate:
angx_=map_coordinates(angx,[[y],[x]],order=1)
angy_=map_coordinates(angy,[[y],[x]],order=1)
# now we reshape the angles back to a mesh
angx_=angx_.reshape((npix/ndown,npix/ndown))
angy_=angy_.reshape((npix/ndown,npix/ndown))
y1=x1_-angx_
y2=x2_-angy_
# or
#y1=(x1-angx_)
#y2=np.transpose(x2-np.transpose(angy_))
```

The result of this calculation is shown in Fig. 2.7.1. In the left panel, we show the regular grid on the lens plane, through which light-rays are traced starting from the observer position. In the right panel, we show the arrival positions of the light-rays on the source plane. We can see that 1) the

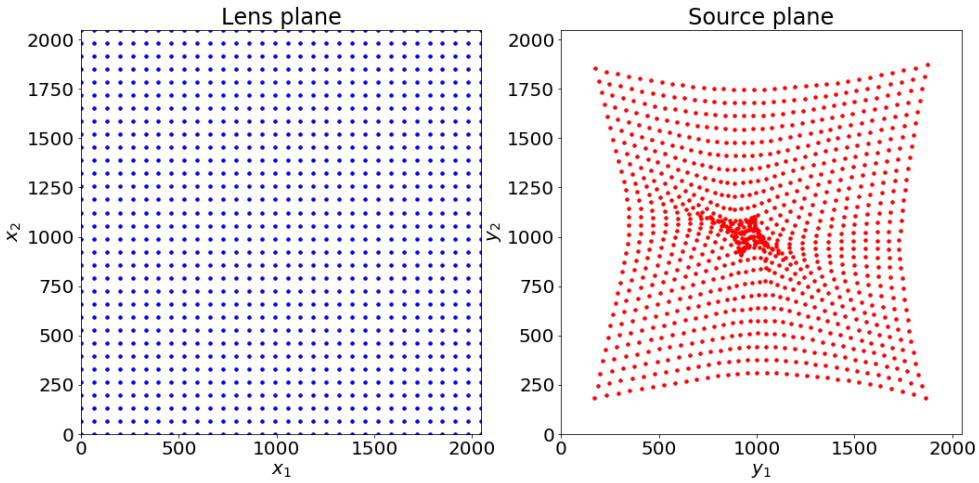


Figure 2.7.1: *Ray-tracing through a regular grid on the lens plane (left panel). The arrival positions of the light-rays on the source plane are shown in the right panel. The lens is the same used in Sect. 1.5.2.*

grid on the source plane is no longer regular, as the consequence of the variations of the deflection angles across the field of view; 2) the source plane is crunched, specially near the center of the lens, where many points are brought very close to each other. This is a manifestation of the lensing magnification: a small area on the source plane is mapped onto a larger area on the lens plane.

2.7.2 Derivation of the lensing potential

Deriving the lensing potential from the lens convergence map requires to solve the Poisson equation in two dimensions (Eqs. 2.26 and 2.27). This can be done numerically by means of Fast-Fourier-Transform.

The Fourier transform of the Laplace operator is

$$\tilde{\triangle}(\vec{k}) = -4\pi^2 k^2$$

where $k^2 = k_1^2 + k_2^2$. Therefore, in Fourier space, the Poisson equation reads

$$-4\pi^2 k^2 \tilde{\Psi}(\vec{k}) = 2\tilde{\kappa}(\vec{k}) \, .$$

The Fourier transform of the lensing potential is then

$$\tilde{\Psi}(\vec{k}) = -\frac{\tilde{\kappa}(\vec{k})}{2\pi^2 k^2} \, .$$

As shown in Sect. 1.5.2, the numerical calculation of the Fourier transforms can be implemented using e.g. the `numpy.fft` module. The following function could be added to the class `deflector` in Sect. 1.5.2:

```
    def potential(self):
        # define an array of wavenumbers (two components k1,k2)
        k = np.array(np.meshgrid(fftengine.fftfreq(self.kappa.shape[0])\
                                 ,fftengine.fftfreq(self.kappa.shape[1])))
```

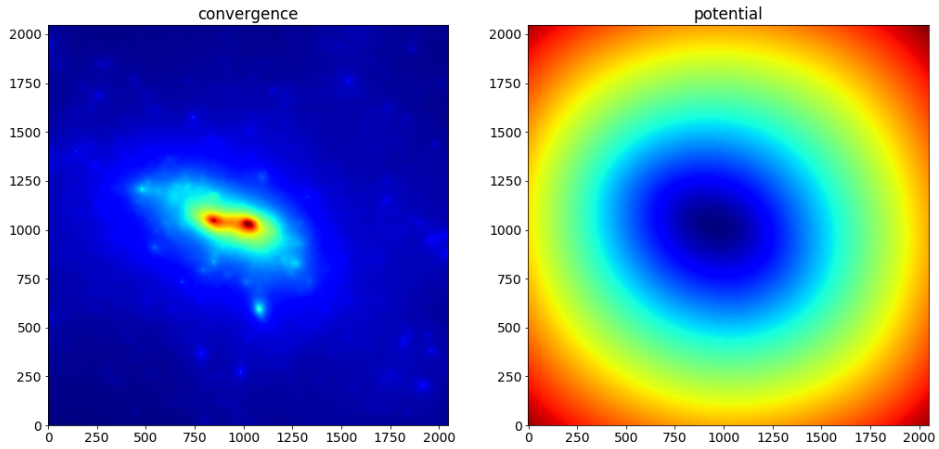


Figure 2.7.2: *Maps of the convergence and of the lensing potential maps for the same lens used in Sect. 1.5.2*

```
        pix=1 # pixel scale (now using pixel units)
        #Compute Laplace operator in Fourier space
        kk = k[0]**2 + k[1]**2
        kk[0,0] = 1.0
        #FFT of the convergence
        kappa_ft = fftengine.fftn(kappa)
        #compute the FT of the potential
        kappa_ft *= - pix**2 / (kk * (2.0*np.pi**2))
        kappa_ft[0,0] = 0.0
        potential=fftengine.ifftn(kappa_ft) #units should be rad**2
        return potential.real
```

We can compute the lensing potential and display the resulting map as follows:

```
pot=df.potential() # compute the potential
kappa=df.mapCrop(kappa) # remove zero-padded region from
                        # convergence and potential maps
pot=df.mapCrop(pot)

# display the results
fig,ax = plt.subplots(1,2,figsize=(17,8))
ax[0].imshow(kappa,origin="lower")
ax[0].set_title('convergence')
ax[1].imshow(pot,origin="lower")
ax[1].set_title('potential')
```

The maps of the convergence and of the lensing potential for the lens considered are shown in Fig. 2.7.2. Clearly, the potential is much smoother than the convergence. This reflects the fact that the convergence is obtained by means of second derivatives of the potential.

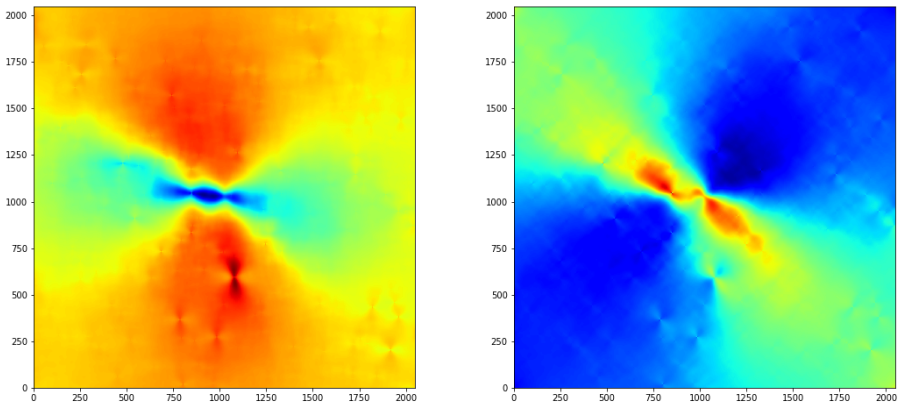


Figure 2.7.3: *Maps of the shear components for the same lens used in Sect. 1.5.2*

2.7.3 Lensing maps

Once the potential is known, it is easy to compute maps of many other properties of the lens. For example, the gradient of $\hat{\Psi}$ is the deflection angle. Thus, we can implement a method to compute $\vec{\alpha}$ which is alternative to that discussed in Sect. 1.5.2.

The python implementation shown here makes usage of the `numpy.gradient` method. Precisely,

```
a2,a1=np.gradient(pot)
```

returns two maps of the deflection angle components α_1 and α_2. Note that, because of the axis convention in python, the derivatives of $\hat{\Psi}$ along the second dimension is given first. We do not display the maps, as they are analogous to those shown e.g. in Fig. 1.5.3.

By computing further gradients of the maps of the two components of the deflection angle, we obtain maps of the second derivatives of the potential. By combining them, we can compute the convergence (which is already known, as it was the input to derive the potential) and the shear components. The python implementation of Eq. 2.38 is as follows:

```
# First we compute the second derivatives of pot
psi12,psi11=np.gradient(a1)
psi22,psi21=np.gradient(a2)
# Then we combine them to form the first and the second component of
# the shear tensor
gamma1=0.5*(psi11-psi22)
gamma2=psi12
```

In Fig. 2.7.3, we show the maps both γ_1 and γ_2.

As discussed in Sect. 2.3.1, the shear introduces an anisotropic distortion of the images of sources. For example, a circular source is mapped onto an elliptical image (in the case of a slowly varying deflection angle). The direction of the axes of the ellipse is given by the angle ϕ in Eq. 2.42, which can be computed using the `arctan2` function:

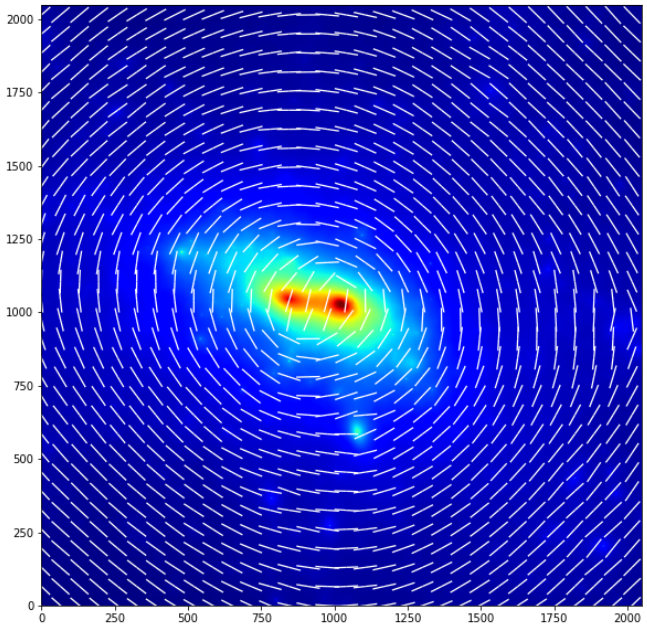


Figure 2.7.4: *Shear orientation overlaid to the convergence map of the same lens used in Sect. 1.5.2*

```
phi=np.arctan2(gamma2,gamma1)/2.0
```

Note that we have to divide by 2 in order to account for the fact that γ is a spin-2 tensor. It is interesting to display the direction into which the shear distorts images and compare it to lens mass distribution. Fig. 2.7.4 shows the direction of the first eigenvector of the shear overlaid to the lens convergence. The code to produce the figure is

```
pixel_step=gamma_1.shape[1]/32+1
x,y = np.meshgrid(np.arange(0,gamma_1.shape[1],pixel_step),
                                np.arange(0,gamma_1.shape[0],pixel_step))
fig,ax=plt.subplots(1,1,figsize=(10,10))
ax.imshow(ka,origin='lower',vmax=3)
ax[1].imshow(kappa,origin='lower',vmax=3)

# showing only the orientation of the shear. This will create two sticks
# departing from the point where the shear is evaluated (x,y) and directed
# in opposite directions
ax[1].quiver(y,x,np.cos(phi[x,y]),np.sin(phi[x,y]),
        headwidth=0,units="height",scale=x.shape[0],color="white")
ax[1].quiver(y,x,-np.cos(phi[x,y]),-np.sin(phi[x,y]),
        headwidth=0,units="height",scale=x.shape[0],color="white")
```

From the maps of the shear, we can derive the maps of the flexions F and G. Each of these quantities have two components, corresponding to the real and to imaginary parts of the complex

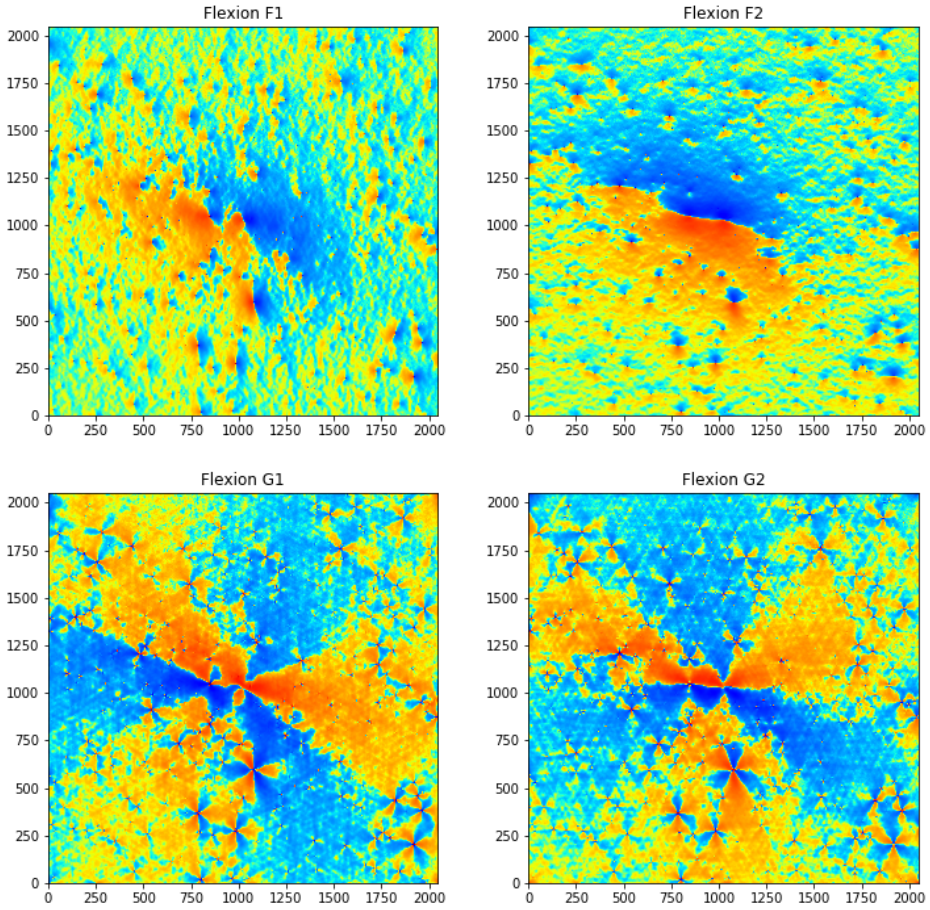


Figure 2.7.5: *Maps of the components of the flexions F and G for the same lens used in Sect. 1.5.2*

quantities in Eqs. 2.72 and 2.73:

```
gamma12,gamma11=np.gradient(gamma_1)
gamma22,gamma21=np.gradient(gamma_2)
F1,F2=gamma11+gamma22,gamma21-gamma12
G1,G2=gamma11-gamma22,gamma21+gamma12
```

Their maps are shown in Fig. 2.7.5. These maps show some interesting features:

- the features in the flexion F maps have dipole symmetry, as expected for a spin-1 field;
- the features in the flexion G maps have triangular symmetry, denoting the spin-3 nature of this field, which is invariant under rotations by $2\pi/3$ radians;
- in both the cases of the flexion F and G, the signal of the small scale structures in the convergence map is amplified (flexion is obtained via third-order derivatives of the lensing potential).

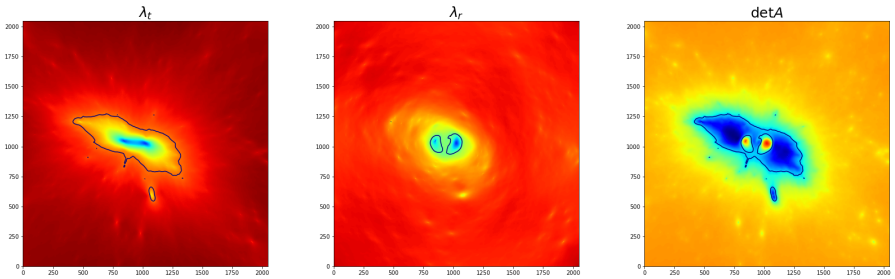


Figure 2.7.6: *The left and the central panel show the maps of the eigenvalues of the lensing Jacobian with overlaid their zero-level contours, i.e. the critical lines. Instead, the right panel shows the the map of* det$A$*, i.e. the product of the two previous maps.*

2.7.4 Critical lines and caustics

Critical lines are defined by the equations

$$\lambda_t = 1-\kappa-\gamma=0 \tag{2.88}$$
$$\lambda_r = 1-\kappa+\gamma=0 \tag{2.89}$$

where $\gamma=(\gamma_1^2+\gamma_2^2)^{1/2}$. There are several methods to identify the points belonging to the critical lines. A simple way to visualize them is draw the zero-level contours in the maps of λ_t and λ_r. Since the λ_t and λ_r are the eigenvalues of the lensing Jacobian, the critical lines correspond also to the zero-level contours in the map of det$A$. The following code implements the search for the critical lines using this method.

```
from matplotlib.colors import LogNorm, PowerNorm, SymLogNorm

gamma=np.sqrt(gamma_1**2+gamma_2**2)
lambdat=1.0-kappa-gamma
lambdar=1.0-kappa+gamma
detA=lambdat*lambdar

fig,ax=plt.subplots(1,3,figsize=(28,8))
ax[0].imshow(lambdat,origin='lower')
ax[0].contour(lambdat,levels=[0.0])
ax[0].set_title('$\lambda_t$',fontsize=25)
ax[1].imshow(lambdar,origin='lower')
ax[1].contour(lambdar,levels=[0.0])
ax[1].set_title('$\lambda_r$',fontsize=25)
ax[2].imshow(detA,origin='lower',norm=SymLogNorm(0.3))
ax[2].contour(detA,levels=[0.0])
ax[2].set_title('$\det A$',fontsize=25)
```

The results are displayed in Fig. 2.7.6. The left and the central panels show the maps of λ_t and λ_r, respectively. In the right panel, we show the map of det$A$. In all panels, we overlay the zero-level contours, i.e. the critical lines. In the left and in the central panels the tangential and the radial critical lines are shown separately, while in the right panel they are displayed simultaneously. Note that these are the critical lines for a specific source redshift. In fact, the convergence map

used to compute the lensing potential is normalized to $z_{s,norm} = 9$[2]. To obtain the critical lines for different source redshifts z_s, we need to rescale both κ and γ by the distance ratio

$$\Xi = \frac{D_{\mathrm{S},norm}}{D_{\mathrm{LS},norm}} \frac{D_{\mathrm{LS}}(z_s)}{D_{\mathrm{S}}(z_s)} , \tag{2.90}$$

where the distances $D_{\mathrm{S},norm}$ and $D_{\mathrm{LS},norm}$ are computed for $z_s = z_{s,norm}$. The code below repeats this operation for 20 equally spaced redshifts between z_l and $z_s = 10$. The corresponding critical lines are shown in Fig. 2.7.7. In order to compute the distances, we have to assume a cosmological model. This can be done by using the `astropy.cosmology` module. Here, we import a pre-defined flat ΛCDM cosmological model with density parameter $\Omega_M = 0.3$ and $\Omega_\Lambda = 0.7$. We assume the Hubble parameter to be $H_0 = 70$ km$/s/$Mpc. The angular diameter distances are computed using the `angular_diameter_distance` and the `angular_diameter_distance_z1z2` methods.

```
from astropy.cosmology import FlatLambdaCDM
cosmo = FlatLambdaCDM(H0=70, Om0=0.3)

zl=0.5
zs_norm=9.0

zs=np.linspace(zl,10.0,20)
dl=cosmo.angular_diameter_distance(zl)
ds=cosmo.angular_diameter_distance(zs)
dls=[]
for i in range(ds.size):
    dls.append(cosmo.angular_diameter_distance_z1z2(zl,zs[i]).value)


ds_norm=cosmo.angular_diameter_distance(zs_norm)
dls_norm=cosmo.angular_diameter_distance_z1z2(zl,zs_norm)

fig,ax=plt.subplots(1,2,figsize=(16,8))
ax[0].imshow(lambdat,origin='lower')
ax[1].imshow(lambdar,origin='lower')
for i in range(ds.size):
    kappa_new=kappa*ds_norm.value/dls_norm.value*dls[i]/ds[i].value
    gamma_new=gamma*ds_norm.value/dls_norm.value*dls[i]/ds[i].value
    lambdat_new=(1.0-kappa_new-gamma_new)
    lambdar_new=(1.0-kappa_new+gamma_new)
    ax[0].contour(lambdat_new,levels=[0.0])
    ax[1].contour(lambdar_new,levels=[0.0])

ax[0].contour(lambdat,levels=[0.0],colors="yellow",linewidths=2)
ax[1].contour(lambdar,levels=[0.0],colors="magenta",linewidths=2)
```

The caustics are the "sources" of the critical lines. In other words, if $\vec{\theta}_c$ defines a set of points belonging to the critical lines, then

$$\vec{\beta}_c = \vec{\theta}_c - \vec{\alpha}(\vec{\theta}_c) \tag{2.91}$$

[2]The lens redshift is $z_l = 0.5$.

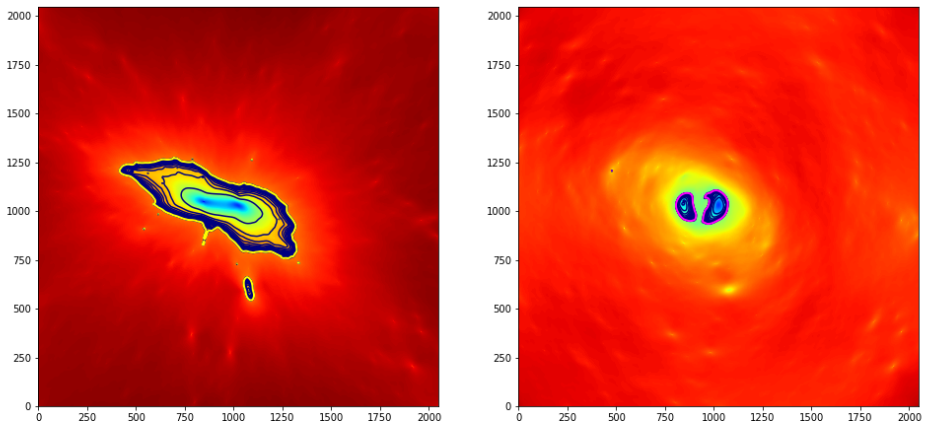


Figure 2.7.7: *Tangential (left) and radial (right) critical lines of the lens for different source redshifts.*

defines a set of points belonging to the caustics.

To proceed to the calculation of the caustics, we need first to gather the deflection angles at the position of the critical points. These can be read from the maps of the deflection angles computed earlier by means of an interpolation. We will use the `map_coordinates` method from the `scipy.ndimage` module. In the following, we will work in pixel units:

```
fig,ax=plt.subplots(1,2,figsize=(18,8))
# first we extract the level-0 contours of the map of detA
cs=ax[0].contour(detA,levels=[0.0])

# then, we take the path of each closed contour
contour=cs.collections[0]
p=contour.get_paths() # p contains the paths of each individual
                                    # critical line

sizevs=np.empty(len(p),dtype=int)

from scipy.ndimage import map_coordinates

# if we found any contour, then we proceed
if (sizevs.size > 0):
    for j in range(len(p)):
        # for each path, we create two vectors containing the x1
        # and x2 coordinates of the vertices
        vs = contour.get_paths()[j].vertices
        sizevs[j]=len(vs)
        x1=[]
        x2=[]
        for i in range(len(vs)):
            xx1,xx2=vs[i]
```

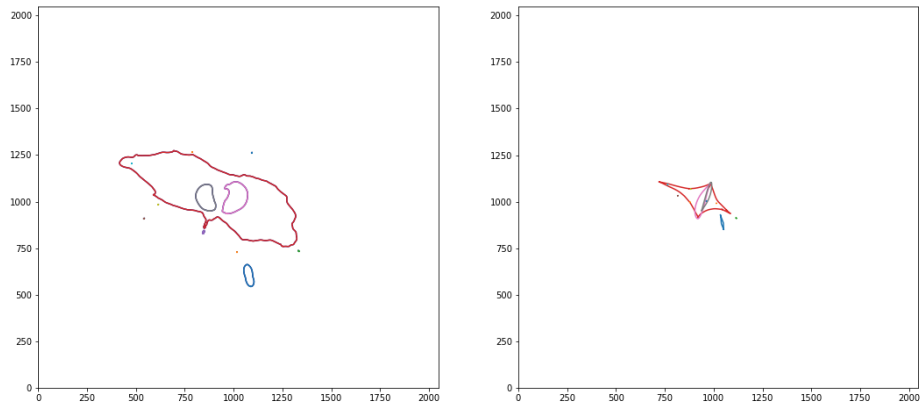


Figure 2.7.8: *Critical lines (left panel) and caustics (right panel) for $z_s = 9$. Different closed critical lines in the left panel have different colors. These can be used to identify the corresponding caustics in the right panel.*

```
            x1.append(float(xx1))
            x2.append(float(xx2))
        # these are the points we want to map back on the source plane.
        # To do that we need to evaluate the deflection angle at their
        # positions using scipy.ndimage.interpolate.map_coordinates
        # we perform a bi-linear interpolation

        a_1=map_coordinates(a1, [[x2],[x1]],order=1)
        a_2=map_coordinates(a2, [[x2],[x1]],order=1)

        # now we can make the mapping using the lens equation:
        y1=x1-a_1[0]
        y2=x2-a_2[0]

        # plot the results!
        ax[0].plot(x1,x2,'-')
        ax[1].plot(y1,y2,'-')

ax[1].set_xlim([0,2048])
ax[1].set_ylim([0,2048])
```

The left and right panels in Fig. 2.7.8 show the critical lines and the caustics on the lens and the source planes, respectively. They display exactly the same region of the sky. The shape of the caustics should be compared to the pattern visible in the right panel of Fig. 2.7.1. Performing ray-tracing, we found that starting from a regular grid of ray positions on the lens plane, we end up with an irregular grid on the source plane, which also covers a smaller area of the sky due to magnification. Now, we can easily see that the arrival positions of light rays on the source plane are clustered around the caustics of the lens.

2.7.5 Shear and flexion

In this example we use Eq. 2.57 to build an application to visualize the lensing distortions due to shear and flexion.

As we have seen, the elements of D, D_{ijk}, are expressed as third derivatives of the lensing potential. These in turn can be written in terms of the flexions F and G. After some math, we find that

$$\begin{aligned} D_{111} &= -2\gamma_{11} - \gamma_{22} = -\frac{1}{2}(3F_1 + G_1) \\ D_{211} &= D_{121} = D_{112} = -\gamma_{21} = -\frac{1}{2}(F_2 + G_2) \\ D_{122} &= D_{212} = D_{221} = -\gamma_{22} = -\frac{1}{2}(F_1 - G_1) \\ D_{222} &= 2\gamma_{12} - \gamma_{21} = -\frac{1}{2}(3F_2 - G_2) \end{aligned} \tag{2.92}$$

Then, the two components of $\vec{\beta}$ are

$$\begin{aligned} \beta_1 &= A_{11}\theta_1 + A_{12}\theta_2 + \frac{1}{2}D_{111}\theta_1^2 + D_{121}\theta_1\theta_2 + \frac{1}{2}D_{122}\theta_2^2 \\ \beta_2 &= A_{21}\theta_1 + A_{22}\theta_2 + \frac{1}{2}D_{211}\theta_1^2 + D_{212}\theta_1\theta_2 + \frac{1}{2}D_{222}\theta_2^2 \end{aligned} \tag{2.93}$$

We consider a circular source centered at $\vec{\beta}_0 = (0,0)$. We assign to this source a surface brightness profile. For this exercise we can choose a generic Sérsic profile (Sérsic, 1963) in the form

$$I_s(\beta) \propto \exp -b_n[(\beta/r_e)^{1/n} - 1] \, . \tag{2.94}$$

The shape parameter b_n is given by the approximated formula (Capaccioli et al., 1989)

$$b_n = 1.992n - 0.3271 \, , \tag{2.95}$$

which is valid for for $0.5 < n < 10$. The parameter r_e is the effective radius of the source and n is the Sersic index.

Since the surface brightness is conserved, we can relate the surface brightness $i(\vec{\theta})$ on the lens and $I_s(\vec{\beta})$ on the source plane using the lens equation:

$$I(\vec{\theta}) = I_s(\vec{\beta}) \, . \tag{2.96}$$

This allows us to straightforwardly reconstruct the image of the source on the lens plane. In fact, the lens equation above allows to map the coordinates $\vec{\theta}$ into the coordinates $\vec{\beta}$ on the plane of the source. This is a ray-tracing procedure. We can indeed cover the lens plane with a grid of coordinates $\vec{\theta}$ and, once we know the corresponding coordinates $\vec{\beta}$, we can read-off the surface brightness at $\vec{\theta}$ from the map of $I_s(\vec{\beta})$.

Note that the lens equation above does not include any shift of the image of the image position (i.e. the source and the image will be at the same position on the plane of the sky, i.e. within our field-of-view).

We start by implementing a class for Sérsic sources. To initialize each instance of this class, we provide the parameters of the brightness profile, namely n and r_e. For this example, no other parameters are necessary to define the appearance of the source. Indeed, we will generate circular sources. However, we also need to specify the size, `size`, and the number of pixels, N, of the image of the source that we want to produce.

```
# import the usual packages: numpy and matplotlib
import numpy as np
import matplotlib.pyplot as plt

class sersic(object):
    def __init__(self,side,N,**kwargs):

        if ('n' in kwargs):
            self.n=kwargs['n']
        else:
            self.n=4

        if ('re' in kwargs):
            self.re=kwargs['re']
        else:
            self.re=50.0

        self.N=N
        self.side=float(side)

        # define the pixel coordinates
        pc=np.linspace(-side/2.,side/2.,self.N)
        self.x1,self.x2 = np.meshgrid(pc,pc)
        y1,y2=self.x1,self.x2
        self.unlensed = self.brightness(y1,y2)

    def brightness(self,y1,y2):
        r = np.sqrt(y1**2+y2**2)
        bn = 1.992*self.n - 0.3271
        return (np.exp(-bn*((r/self.re)**(1.0/self.n)-1.0)))
```

The method `brightness` implements the calculation of the surface brightness on the source plane using the Sérsic profile. As it is now, the brightness is calculated on the regular grid of light rays $(x_1.x_2)$, which covers the lens plane and is propagated to the source plane without deflections. The resulting brightness distribution is recorded in the array `unlensed`.

Now we implement Eq. 2.93. This is done in the function `lens` below

```
    def lens(self,**kwargs):
        if ('kappa' in kwargs):
            self.kappa = kwargs['kappa']
        else:
            self.kappa=0.0

        if ('gamma1' in kwargs):
            self.gamma1 = kwargs['gamma1']
        else:
            self.gamma1=0.0

        if ('gamma2' in kwargs):
```

```
            self.gamma2 = kwargs['gamma2']
        else:
            self.gamma2=0.0

        if ('g1' in kwargs):
            self.g1 = kwargs['g1']
        else:
            self.g1=0.0
        if ('g2' in kwargs):
            self.g2 = kwargs['g2']
        else:
            self.g2=0.0

        if ('f1' in kwargs):
            self.f1 = kwargs['f1']
        else:
            self.f1=0.0
        if ('f2' in kwargs):
            self.f2 = kwargs['f2']
        else:
            self.f2=0.0

        a11=1.0-self.kappa-self.gamma1
        a22=1.0-self.kappa+self.gamma1
        a12=-self.gamma2
        a111=-0.5*(self.g1+3.0*self.f1)
        a222=-0.5*(3.0*self.f2-self.g2)
        a112=-0.5*(self.f2+self.g2)
        a221=-0.5*(self.f1-self.g1)

        y1 = a11*self.x1 + a12*self.x2 + 0.5*a111*self.x1**2 + \
            a112*self.x1*self.x2 + 0.5*a221*self.x2**2
        y2 = a22*self.x2 + a12*self.x1 + 0.5*a222*self.x2**2 + \
            a221*self.x1*self.x2 + 0.5*a112*self.x1**2

        self.lensed=self.brightness(y1,y2)
```

At this point, we can create an instance of the Sérsic source and lens it by choosing the values of the convergence, shear, and F and G flexions. Some results are shown in Fig. 2.5.1.

2.7.6 Full ray-tracing simulation and time delay surface

The procedure outlined above for lensing a Sérsic source can be extended to include also distortions described by higher order terms in the expansion of the deflection angle. By design, all these terms are included in a full ray-tracing simulation employing the algorithm discussed in Sect. 2.7.1.

In this example, we will build an application to visualize the distortions of an elliptical source with Sérsic profile by virtually any lens.The application will also compare the shape and location of the distorted images with the contours of equal time-delay. Compared to the example shown in Sect. 2.7.5, we will extend the class `Sérsic` to allow elliptical shapes with random orientation. We will also use inheritance to build a parent class for generic lenses and child classes for specific

lensing potentials.

We start by importing some useful packages:

```
# import numpy and matplotlib
import numpy as np
import matplotlib.pyplot as plt

# import map_coordinates from scipy
from scipy.ndimage import map_coordinates

# import fits from astropy
import astropy.io.fits as pyfits

# import fft from numpy
import numpy.fft as fftengine
```

The parent class `gen_lens` will contain methods that can be applied to any lens. The initialization function is empty. It contains only one instruction to set the logic variable `pot_exist=False`, meaning that the parent class itself has no potential specified a priori. This will be given in the child class.

```
# the parent class
class gen_lens(object):

    # in the beginning, the class does not have a potential
    def __init__(self):
        self.pot_exists=False
```

The functions to derive convergence, shear, Jacobian determinant (inverse magnification) are part of the generic lens class, because they can work with any potential.

```
    # convergence
    def convergence(self):
        if (self.pot_exists):
            kappa=0.5*(self.a11+self.a22)
        else:
            print ("The lens potential is not initialized yet")

        return(kappa)

    #shear
    def shear(self):
        if (self.pot_exists):
            g1=0.5*(self.a11-self.a22)
            g2=self.a12
        else:
            print ("The lens potential is not initialized yet")
        return(g1,g2)

    # determinant of the Jacobian matrix
    def detA(self):
```

```
        if (self.pot_exists):
            deta=(1.0-self.a11)*(1.0-self.a22)-self.a12*self.a21
        else:
            print ("The lens potential is not initialized yet")
        return(deta)

    # critical lines overlaid to the map of detA, returns a set of
    # contour objects
    def crit_lines(self,ax=None,show=True):
        if (ax==None):
            print ("specify the axes to display the critical lines")
        else:
            deta=self.detA()
            #ax.imshow(deta,origin='lower')
            cs=ax.contour(deta,levels=[0.0],colors='white',alpha=0.0)
            if show==False:
                ax.clear()
        return(cs)

    # plot of the critical lines in the axes ax
    def clines(self,ax=None,color='red',alpha=1.0,lt='-'):
        cs=self.crit_lines(ax=ax,show=False)
        contour=cs.collections[0]
        p=contour.get_paths()
        sizevs=np.empty(len(p),dtype=int)

        no=self.pixel
        # if we found any contour, then we proceed
        if (sizevs.size > 0):
            for j in range(len(p)):
                # for each path, we create two vectors containing
                #the x1 and x2 coordinates of the vertices
                vs = contour.get_paths()[j].vertices
                sizevs[j]=len(vs)
                x1=[]
                x2=[]
                for i in range(len(vs)):
                    xx1,xx2=vs[i]
                    x1.append(float(xx1))
                    x2.append(float(xx2))

                # plot the results!
                ax.plot((np.array(x1)-self.npix/2.)*no,
                        (np.array(x2)-self.npix/2.)*no,lt,color=color,
                        alpha=alpha)

    # plot of the caustics in the axes ax
    def caustics(self,ax=None,alpha=1.0,color='red',lt='-'):
        cs=self.crit_lines(ax=ax,show=True)
```

```
        contour=cs.collections[0]
        p=contour.get_paths() # p contains the paths of each individual
                              # critical line
        sizevs=np.empty(len(p),dtype=int)

        # if we found any contour, then we proceed
        if (sizevs.size > 0):
            for j in range(len(p)):
                # for each path, we create two vectors containing
                # the x1 and x2 coordinates of the vertices
                vs = contour.get_paths()[j].vertices
                sizevs[j]=len(vs)
                x1=[]
                x2=[]
                for i in range(len(vs)):
                    xx1,xx2=vs[i]
                    x1.append(float(xx1))
                    x2.append(float(xx2))

                a_1=map_coordinates(self.a1, [[x2],[x1]],order=1)
                a_2=map_coordinates(self.a2, [[x2],[x1]],order=1)

                # now we can make the mapping using the lens equation:
                no=self.pixel
                y1=(x1-a_1[0]-self.npix/2.)*no
                y2=(x2-a_2[0]-self.npix/2.)*no

                # plot the results!
                ax.plot(y1,y2,lt,color=color,alpha=alpha)
```

Note that all the functions included in the generic lens were discussed in the details in the previous examples.

We can now add the functions to compute the time-delay surface and to plot its contour levels:

```
     # geometrical time delay
    def t_geom_surf(self, beta=None):
        x = np.arange(0, self.npix, 1, float)*self.pixel
        y = x[:,np.newaxis]
        if beta is None:
            x0 = y0 = self.npix / 2*self.pixel
        else:
            x0 = beta[0]+self.npix/2*self.pixel
            y0 = beta[1]+self.npix/2*self.pixel

        return 0.5*((x-x0)*(x-x0)+(y-y0)*(y-y0))


    # gravitational time delay (this will need a potential to be specified):
    def t_grav_surf(self):
```

```
        return -self.pot

    # total time delay
    def t_delay_surf(self,beta=None):
        t_grav=self.t_grav_surf()
        t_geom=self.t_geom_surf(beta)
        td=(t_grav+t_geom)
        return(t_grav+t_geom)

    # display the time delay contours
    def show_contours(self,surf0,ax=None,minx=-25,miny=-25,
                      cmap=plt.get_cmap('Paired'),
                      linewidth=1,fontsize=20,nlevels=40,levmax=100,
                      offz=0.0):
        if ax==None:
            print ("specify the axes to display the contours")
        else:
            minx=minx
            maxx=-minx
            miny=miny
            maxy=-miny
            surf=surf0-np.min(surf0)
            levels=np.linspace(np.min(surf),levmax,nlevels)
            ax.contour(surf, cmap=cmap,levels=levels,
                       linewidth=linewidth,
                       extent=[-self.size/2,self.size/2,
                       -self.size/2,self.size/2])
            ax.set_xlim(minx, maxx)
            ax.set_ylim(miny, maxy)
            ax.set_xlabel(r'$\theta_1$',fontsize=fontsize)
            ax.set_ylabel(r'$\theta_2$',fontsize=fontsize)
            ax.set_aspect('equal')
```

It remains to be specified the child class. This will contain the definition of the lensing potential. We define two of such classes. The first is the elliptical pseudo-isothermal model with core (PSIEc).

The PSIEc potential has the form:

$$\hat{\Psi}(\vec{\theta}) = \frac{\hat{\Psi}_0}{\sqrt{\theta^2 + \theta_c^2}}\,, \tag{2.97}$$

where $\hat{\Psi}_0$ is a normalization factor, $\theta^2 = \theta_1^2/(1-e) + \theta_2^2(1-e)$, and θ_c is the core radius. The lens has iso-potential contours with ellipticity e.

As said, `PSIEc` is a child class of `gen_lens`, and inherits all its methods. For using them, we just need to define a lensing potential. We define it on the map `pot`. The map has a certain dimension, `size`, and number of pixels, `npix`. The function `potential` also computes the first and the second derivatives of the lensing potential. These quantities are used in the `gen_lens` class to derive the lens properties.

```
# psiec: pseudo elliptical isothermal lens with core
class PSIEc(gen_lens):
    def __init__(self,size=100.0,npix=200,**kwargs):

        if ('theta_c' in kwargs):
            self.theta_c=theta_c
        else:
            self.theta_c=0.0

        if ('ell' in kwargs):
            self.ell=ell
        else:
            ell=0.0

        if ('norm' in kwargs):
            norm=norm
        else:
            norm=1.0

        self.size=size
        self.npix=npix
        self.pixel=float(self.size)/float((self.npix-1))
        self.potential()

  # lensing potential and its derivatives
    def potential(self):
        x = np.arange(0, self.npix, 1, float)
        y = x[:,np.newaxis]
        x0 = y0 = self.npix / 2
        no=self.pixel**2
        self.pot_exists=True
        pot=np.sqrt(((x-x0)*self.pixel)**2/(1-self.ell)
                        +((y-y0)*self.pixel)**2*(1-self.ell)
                        +self.theta_c**2)*self.norm
        self.pot=pot#/no
        self.a2,self.a1=np.gradient(self.pot/self.pixel**2)
        self.a12,self.a11=np.gradient(self.a1)
        self.a22,self.a21=np.gradient(self.a2)
```

The last step is to modify the class `Sérsic` to include the full ray-tracing. We will also allow for elliptical sources. The initialization function is written as follows

```
class sersic(object):

    def __init__(self,size,N,gl=None,**kwargs):

        if ('n' in kwargs):
            self.n=kwargs['n']
        else:
```

```
            self.n=4

        if ('re' in kwargs):
            self.re=kwargs['re']
        else:
            self.re=5.0

        if ('q' in kwargs):
            self.q=kwargs['q']
        else:
            self.q=1.0

        if ('pa' in kwargs):
            self.pa=kwargs['pa']
        else:
            self.pa=0.0

        if ('ys1' in kwargs):
            self.ys1=kwargs['ys1']
        else:
            self.ys1=0.0

        if ('ys2' in kwargs):
            self.ys2=kwargs['ys2']
        else:
            self.ys2=0.0

        self.N=N
        self.size=float(size)
        self.df=gl

        # define the pixel coordinates
        pc=np.linspace(-self.size/2.0,self.size/2.0,self.N)
        self.x1, self.x2 = np.meshgrid(pc,pc)
        if self.df != None:
            y1,y2 = self.ray_trace()
        else:
            y1,y2 = self.x1,self.x2

        self.image=self.brightness(y1,y2)
```

In addition to the parameters `N` and `size`, we need to assign to the source a generic lens `gl`. We will use its deflection angles to perform the ray-tracing. Among the arguments that define the intrinsic properties of the source, we include

- the Sérsic index, `n`, and the effective radius, `re`;
- the axis ratio, `q`, and the position angle, `pa`;
- the coordinates of the center of the lens, `ys1, ys2`.

The parameters `N` and `size` define the grid of light rays to be propagated towards the source. The coordinates of the rays are `x1` and `x2`. If `gl` is provided , the deflections are calculated and the

lens equation is applied. The arrival positions of the rays on the source plane have coordinates y1 and y2.

```
    def ray_trace(self):
        px=self.df.pixel#size/(self.df.npix-1)
        x1pix=(self.x1+self.df.size/2.0)/px
        x2pix=(self.x2+self.df.size/2.0)/px

        # compute the deflection angles at the light positions
        # on the lens plane. Use the deflection angles of self.df
        a1 = map_coordinates(self.df.a1,
                             [x2pix,x1pix],order=2)*px
        a2 = map_coordinates(self.df.a2,
                             [x2pix,x1pix],order=2)*px
          # apply the lens equation
        y1=(self.x1-a1) # y1 coordinates on the source plane
        y2=(self.x2-a2) # y2 coordinates on the source plane
        return(y1,y2)
```

Note that the deflection angles are reduced, i.e. they already contain the factor D_{LS}/D_S which accounts for the distances of the lens and of the source.

Finally, the brightness is computed at the coordinates (y1, y2) using the function brightness:

```
    def brightness(self,y1,y2):
        x = np.cos(self.pa)*(y1-self.ys1)+np.sin(self.pa)*(y2-self.ys2)
        y = -np.sin(self.pa)*(y1-self.ys1)+np.cos(self.pa)*(y2-self.ys2)
        r = np.sqrt(((x)/self.q)**2+(y)**2)
        # profile
        bn = 1.992*self.n - 0.3271
        brightness = np.exp(-bn*((r/self.re)**(1.0/self.n)-1.0))
        return(brightness)
```

We implement hear the ellipticity and the rotation of the source by the angle pa.

The following code shows how to use the classes above to produce Fig. 2.7.9.

```
# lens params
kwargs={'theta_c': 2.0, 'norm': 10.0, 'ell': 0.4}
el=psie(size=80,npix=1000,**kwargs)

# size of the source image
size_stamp=150.0
npix_stamp=1000

xmin,xmax=-el.size/2,el.size/2
ymin,ymax=-el.size/2,el.size/2


fig,ax=plt.subplots(1,2,figsize=(14,8))

# sersic source with no lensing
```

```
kwargs={'q': 0.5,'re': 1.0, 'pa': np.pi/4.0, 'n': 1,
                'ys1': beta[0], 'ys2': beta[1]}
se_unlensed=sersic(size_stamp,npix_stamp,**kwargs)

# same source with lensing by the lens el
se=sersic(size_stamp,npix_stamp,gl=el,**kwargs)

# compute the time delay surface for a source at beta

beta=[0,0]
td=el.t_delay_surf(beta=beta)

# draw caustics (on the left) and critical lines (on the right)
el.caustics(ax=ax[0],lt='--',alpha=1.0)
el.clines(ax=ax[1],lt='--',alpha=1.0)

# show unlensed (on the left) and lensed (on the right) images
ax[0].imshow(se_unlensed.image,origin='lower',
             extent=[-se.size/2,se.size/2,
                     -se.size/2,se.size/2],
             cmap='gray_r')
ax[1].imshow(se.image,origin='lower',
             extent=[-se.size/2,se.size/2,
                     -se.size/2,se.size/2],
             cmap='gray_r')

# show contours of the time delay surface

el.show_contours(td,ax=ax[1],minx=xmin,miny=ymin,
                 nlevels=35,levmax=500,fontsize=20)

x0,x1=-20,20
y0,y1=-20,20
ax[0].set_xlim([x0,x1])
ax[0].set_ylim([y0,y1])

fig.tight_layout()
```

We clearly see that the images of the source shown in the left hand panel are located at the stationary points of the time-delay surface.

Our smart implementation of the general lens class, allows us to easily switch between lens models. Aiming at distorting the source using the usual numerically simulated lens, it is sufficient to initialize the lensed sersic model using a deflector built from the lens convergence map. For example, this could be obtained from the `deflector` class shown below:

```
# child class deflector
class deflector(gen_lens):
```

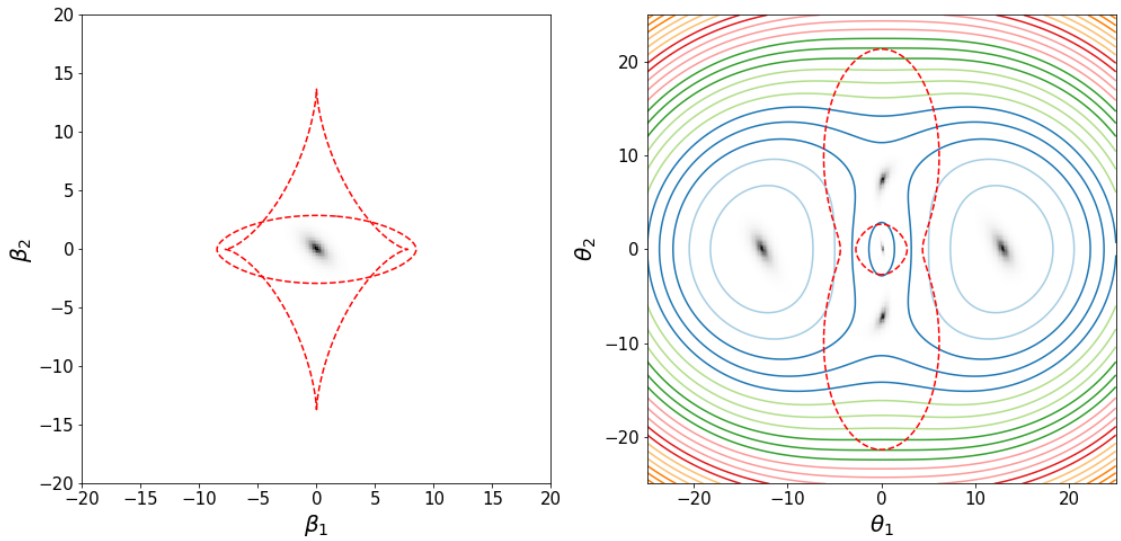


Figure 2.7.9: *Left panel: Caustics of a PSIEc lens. A Sérsic source is placed at the center of the caustics. Right panel: critical lines (dashed) and lensed images of the source shown in the left panel. The colored solid contours show the levels of equal time delay for the same source.*

```
    # initialize the deflector using a surface density (covergence) map
    # the boolean variable pad indicates whether zero-padding is used or not

    def __init__(self,filekappa,pad=False,npix=200,size=100):
        kappa,header=pyfits.getdata(filekappa,header=True)
        self.pixel_scale=header['CDELT2']*3600.0
        self.kappa=kappa
        self.nx=kappa.shape[0]
        self.ny=kappa.shape[1]
        self.pad=pad
        self.npix=npix
        self.size=size
        self.pixel=float(self.size)/float(self.npix-1)
        if (pad):
            self.kpad()
        self.potential()

    # performs zero-padding
    def kpad(self):
        # add zeros around the original array
        def padwithzeros(vector, pad_width, iaxis, kwargs):
            vector[:pad_width[0]] = 0
            vector[-pad_width[1]:] = 0
            return vector
        # use the pad method from numpy.lib to add zeros (padwithzeros)
```

```
        # in a frame with thickness self.kappa.shape[0]
        self.kappa=np.lib.pad(self.kappa, self.kappa.shape[0],
                              padwithzeros)

    # calculate the potential by solving the poisson equation
    def potential_from_kappa(self):
        # define an array of wavenumbers (two components k1,k2)
        k = np.array(np.meshgrid(fftengine.fftfreq(self.kappa.shape[0])\
                                 ,fftengine.fftfreq(self.kappa.shape[1])))
        pix=1 # pixel scale (now using pixel units)
        #Compute Laplace operator in Fourier space = -4*pi*k^2
        kk = k[0]**2 + k[1]**2
        kk[0,0] = 1.0
        #FFT of the convergence
        kappa_ft = fftengine.fftn(self.kappa)
        #compute the FT of the potential
        kappa_ft *= - pix**2 / (kk * (2.0*np.pi**2))
        kappa_ft[0,0] = 0.0
        potential=fftengine.ifftn(kappa_ft) #units should be rad**2
        if self.pad:
            pot=self.mapCrop(potential.real)
        return pot

    # returns the map of the gravitational time delay
    def potential(self):
        no=self.pixel
        x_ = np.linspace(0,self.npix-1,self.npix)
        y_ = np.linspace(0,self.npix-1,self.npix)
        x,y=np.meshgrid(x_,y_)
        potential=self.potential_from_kappa()
        x0 = y0 = potential.shape[0] / 2*self.pixel_scale-self.size/2.0
        x=(x0+x*no)/self.pixel_scale
        y=(y0+y*no)/self.pixel_scale
        self.pot_exists=True
        pot=map_coordinates(potential,[y,x],order=1)
        self.pot=pot*self.pixel_scale**2/no/no
        self.a2,self.a1=np.gradient(self.pot)
        self.a12,self.a11=np.gradient(self.a1)
        self.a22,self.a21=np.gradient(self.a2)
        self.pot=pot*self.pixel_scale**2


    # crop the maps to remove zero-padded areas and get back to the original
    # region.
    def mapCrop(self,mappa):
        xmin=int(self.kappa.shape[0]/2-self.nx/2)
        ymin=int(self.kappa.shape[1]/2-self.ny/2)
        xmax=int(xmin+self.nx)
        ymax=int(ymin+self.ny)
```

```
        mappa=mappa[xmin:xmax,ymin:ymax]
        return(mappa)
```

Here is an example:

```
size=200.0
npix=500
df=deflector('data/kappa_2.fits',True,npix=npix,size=size)

beta=[-30,8]
td=df.t_delay_surf(beta=beta)

xmin,xmax=-df.size/2,df.size/2
ymin,ymax=-df.size/2,df.size/2

fig,ax=plt.subplots(1,2,figsize=(14,8))
kwargs={'q': 0.5,'re': 1.0, 'pa': np.pi/4.0, 'n': 1,'
               ys1': beta[0], 'ys2': beta[1]}


se_unlensed=sersic(size_stamp,npix_stamp,**kwargs)
se=sersic(size_stamp,npix_stamp,gl=df,**kwargs)
df.caustics(ax=ax[0],lt='--',alpha=1.0)
df.clines(ax=ax[1],lt='--',alpha=1.0)
ax[0].imshow(se_unlensed.image,origin='lower',
             extent=[-se_unlensed.size/2,se_unlensed.size/2,
                     -se_unlensed.size/2,se_unlensed.size/2],
             cmap='gray_r')
ax[1].imshow(se.image,origin='lower',
             extent=[-se.size/2,se.size/2,-se.size/2,se.size/2],
             cmap='gray_r')
df.show_contours(td,ax=ax[1],minx=xmin,miny=ymin,nlevels=25,
                              levmax=1600,fontsize=20)
x0,x1=-40,10
y0,y1=-25,25
ax[0].set_xlim([x0,x1])
ax[0].set_ylim([y0,y1])
x0,x1=-80,50
y0,y1=-65,65
ax[1].set_xlim([x0,x1])
ax[1].set_ylim([y0,y1])
fig.tight_layout()
```

The code above produces the figure shown in Fig. 2.7.10. The source is placed near the cusp of the tangential caustic. Being inside the caustic, it produces three images, which are distorted and merge into a tangential arc. Note that the shape of the arc reflects the shape of the levels of equal time delay near the image.

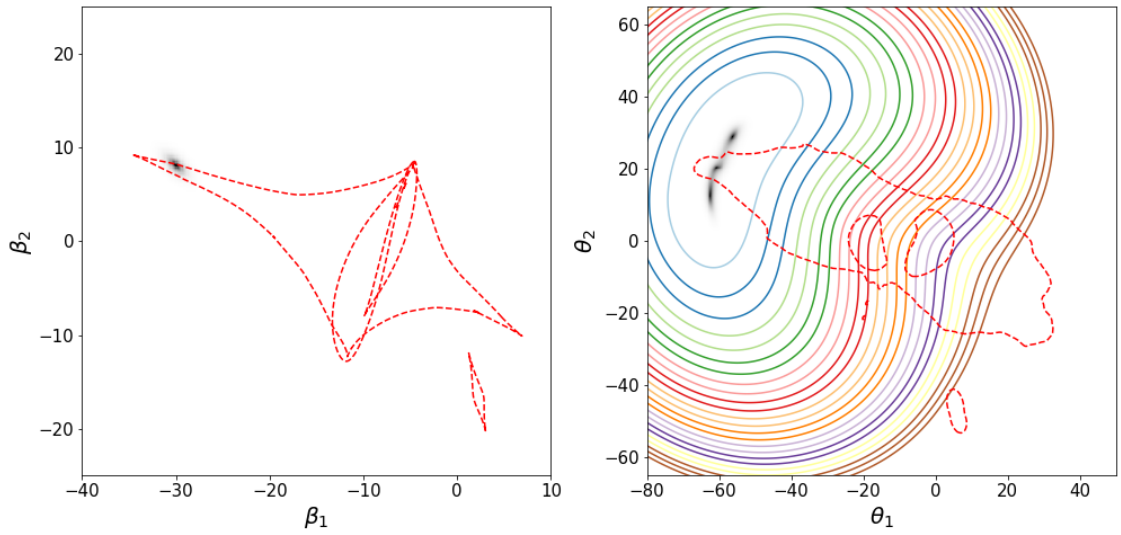
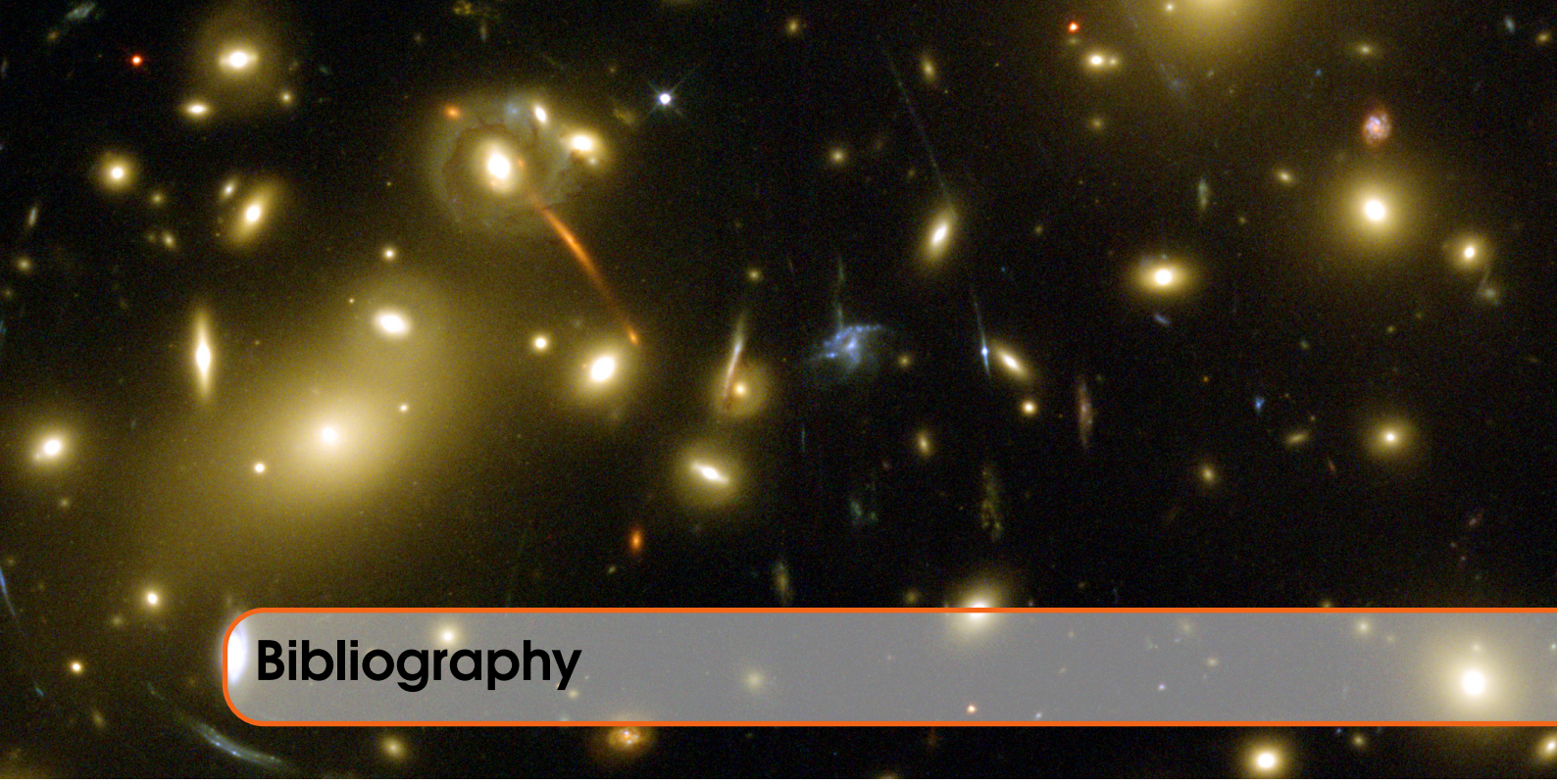


Figure 2.7.10: *Left panel: Caustics of a numerically simulated lens. A Sérsic source is placed near the cusp of the caustics. Right panel: critical lines (dashed) and lensed images of the source shown in the left panel. The colored solid contours show the levels of equal time delay for the same source.*

II Part Two: Applications



Bibliography

Aubert, D., A. Amara, and R. B. Metcalf (2007). "Smooth Particle Lensing". In: *MNRAS* 376, pages 113–124. DOI: 10.1111/j.1365-2966.2006.11296.x. eprint: astro-ph/0604360 (cited on page 18).

Barnes, J. and P. Hut (1986). "A hierarchical O(N log N) force-calculation algorithm". In: *Nature* 324, pages 446–449. DOI: 10.1038/324446a0 (cited on page 18).

Bartelmann, M. and P. Schneider (2001). "Weak gravitational lensing". In: *Phys.Rep.* 340, pages 291–472. DOI: 10.1016/S0370-1573(00)00082-X. eprint: astro-ph/9912508 (cited on page 30).

Bozza, V. (2010). "Gravitational lensing by black holes". In: *General Relativity and Gravitation* 42, pages 2269–2300. DOI: 10.1007/s10714-010-0988-2. arXiv: 0911.2187 [gr-qc] (cited on page 17).

Capaccioli, M. et al. (1989). "Properties of the nova population in M31". In: *AJ* 97, pages 1622–1633. DOI: 10.1086/115104 (cited on page 60).

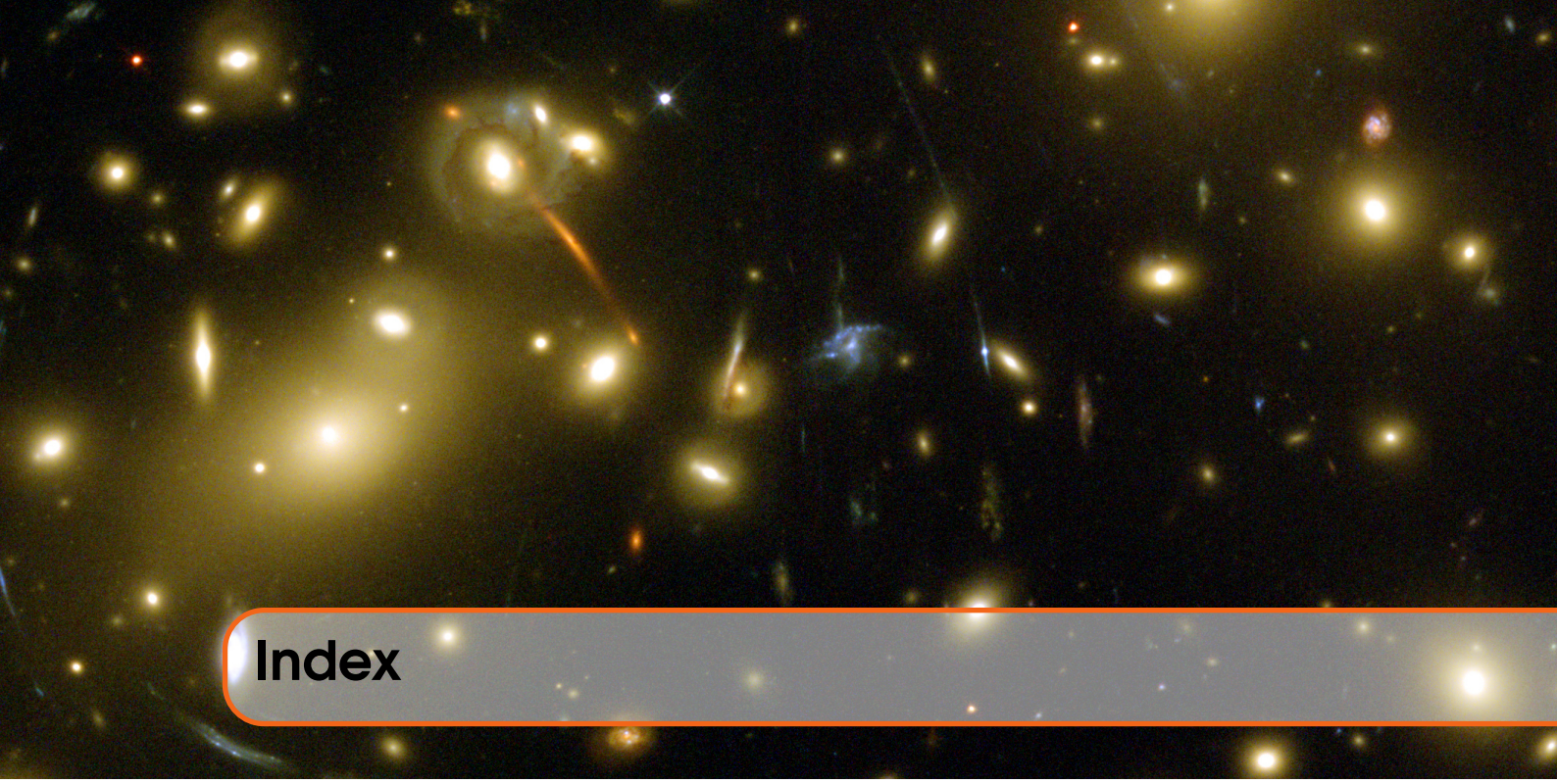
Cooley, James W and John W Tukey (1965). "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of computation* 19(90), pages 297–301 (cited on page 19).

Darwin, C. (1959). "The Gravity Field of a Particle". In: *Proceedings of the Royal Society of London Series A* 249, pages 180–194. DOI: 10.1098/rspa.1959.0015 (cited on page 17).

Meneghetti, M. et al. (2010). "Weighing simulated galaxy clusters using lensing and X-ray". In: *A & A* 514, A93, A93. DOI: 10.1051/0004-6361/200913222. arXiv: 0912.1343 (cited on page 18).

Sérsic, J. L. (1963). "Influence of the atmospheric and instrumental dispersion on the brightness distribution in a galaxy". In: *Boletin de la Asociacion Argentina de Astronomia La Plata Argentina* 6, page 41 (cited on page 60).

Smith, James (2015). "Bending space–time: a commentary on Dyson, Eddington and Davidson (1920) - A determination of the deflection of light by the Sun"s gravitational field". In: 373, page 2039 (cited on page 16).



Index

A. Python tutorial

A.1 Installation

The codes discussed as part of these lectures have been developed and run using Anaconda python 2.7 by Continuum analytics. This is just one of the python distributions available for free and we expect that the codes proposed here should run without problems with any of them.
If the reader opts for the Anaconda distribution, she/he can download the installer, which is available for Windows, Mac OSX, and Linux platforms, from https://www.continuum.io/downloads. Following the installation instructions, python should be ready for usage within few minutes.

A.2 Documentation

There are many resources online and books to learn how to program in python. The list below is just a starting point and does not want to be complete:

- the online official documentation can be found at this url: http://www.python.org/doc;
- several platforms for e-learning propose courses to learn python. For example, Codeacademy offers an excellent course, which can be completed in only 13 hours;
- Google also offers a python class online;
- a more extensive (and practical) guide to python is given by Learn python the hard way

A.3 Running python

Python can be run in several ways:

- from the interactive interpreter: launch “python" in a shell. Quit with Ctrl+D or type “exit()" when finished.
- create your own script with an extension “.py" and run it in a shell by typing “python <script name>.py”
- use an Interactive Development Environment (IDE). These are software which include an editor for coding and capabilities for executing the code. There are several options available (e.g. spyder, Rodeo, etc.)

- We recommend to become familiar with jupyter notebook, which is increasingly popular among python users for sharing code and ideas.

A.4 Your first python code

Try running the code:

```
# your first python code -- this is a comment
print ("Hello World!")
```

Congratulations! You have run your first python code!

A.5 Variables

Variables are names poinint to values or objects. Setting them in python is extremely easy, and you don't need to declare them before:

```
int_var = 4
float_var = 7.89778
boolean_var = True
string_var = "My name is Python"
obj_var=some_class_name(par1,par2)
```

A.6 Strings

String constants can be defined in three ways:

```
single_quotes = 'my name is Python'
double_quotes = "my name is Python"
triple_quotes = """my name is Python
and this is  a multiline  string.""" #This can contain line breaks!
```

Note that you can combine single and double quotes when you want to define strings which contain quotes themselves:

```
double_quotes1 = 'my name is "Python"'
double_quotes2 = "don't"
```

otherwise you have to use backslashes:

```
double_quotes3 = 'don\'t'
\end{python}

Strings can me sliced:
\begin{minted}[bgcolor=bg]{python}
my_name='Massimo Meneghetti'
name = my_name[:7]
surname = my_name[8:]
a_piece_of_my_name=my_name[4:7]
```

You can make many operations with strings. These are objects and have many methods. Check out this url to learn more: https://docs.python.org/2/library/stdtypes.html

Some examples:

- String concatenation:

```
back_to_my_full_name=name+" "+surname
```

- Convert to upper case

```
my_name_uppercase=back_to_my_full_name.upper()
```

The built-in function `str` converts numbers to strings:

```
my_int=2
my_float=2.0
str_int=str(my_int)
str_float=str(my_float)
```

Another way to include numbers in strings:

```
my_string1 = 'My integer is %d.' % my_int
my_string2 = 'My float is %f.' % my_float
my_string3 = 'My float is %3.1f (with only one decimal)' % my_float
```

With several variables, we need to use parentheses:

```
a = 2
b = 67
my_string4 = '%d + %d = %d' % (a, b, a+b)

a = 2
b = 67.3
my_string5 = '%d + %5.2f = %5.1f' % (a, b, a+b)
```

Not only you can convert numbers to string, but you can do the reverse operation:

```
s = '23'
i = int(s)
s = '23'
i = float(s)
```

Strip spaces at beginning and end of a string:

```
stripped = a_string.strip()
```

Replace a substring inside a string:

```
newstring = a_string.replace('abc', 'def')
```

Important note: a Python string is "immutable". In other words, it is a constant which cannot be changed in place. All string operations create a new string. This is strange for a C developer, but it is necessary for some properties of the language. In most cases this is not a problem.

A.7 Lists

A list is a dynamic array of any objects. It is declared with square brackets:

```
a_list = [1, 2, 3, 'abc', 'def']
```

Lists may contain lists:

```
another_list = [a_list, 'abc', a_list, [1, 2, 3]]
```

Note that a_{list} in this case is a pointer.

Access a specific element by index (index starts at zero):

```
elem = a_list[2]
elem2 = another_list[3][1]
```

It's easy to test if an item is in the list:

```
if 'abc' in a_list:
    print 'bingo!'
```

Extracting a part of a list is called slicing:

```
list2 = a_list[2:4] # returns a list with items 2 and 3 (not 4)
```

Other list operations like appending:

```
a_list.append('ghi')
a_list.remove('abc')
```

Other list operations: http://docs.python.org/lib/typesseq.html

A.8 Tuples

A tuple is similar to a list but it is a fixed-size, immutable array. This means that once a tuple has been created, its elements may not be changed, removed, appended or inserted.

It is declared using parentheses and comma-separated values:

```
a_tuple = (1, 2, 3, 'abc', 'def')
```

but parentheses are optional:

```
another_tuple = 1, 2, 3, 'abc', 'def'
```

Tip: a tuple containing only one item must be declared using a comma, else it is not considered as a tuple:

```
a_single_item_tuple = ('one value',)
```

Tuples are not constant lists – this is a common misconception. Lists are intended to be homogeneous sequences, while tuples are hetereogeneous data structures.

In some sense, tuples may be regarded as simplified structures, in which position has semantic value [e.g. (name,surname,age,height,weight)]. For this reason they are immutable, contrary to lists.

A.9 Dictionaries

A Dictionary (or "dict") is a way to store data just like a list, but instead of using only numbers to get the data, you can use almost anything. This lets you treat a dict like it's a database for storing and organizing data.

Dictionaries are initialized using curl brackets:

```
person = {'name': 'Massimo', 'surname': 'Meneghetti'}
```

You can access the elements of the dictionary by using the entry keys:

```
person['name']
```

The keys can also be numbers:

```
person = {'name': 'Massimo', 'surname': 'Meneghetti', 1: 'new data'}
person[1]
```

A.10 Blocks and Indentation

Blocks of code are delimited using indentation, either spaces or tabs at the beginning of lines. This will become clearer in the next sections, when loops will be introduced.

Tip: NEVER mix tabs and spaces in a script, as this could generate bugs that are very difficult to be found.

A.11 IF / ELIF / ELSE

Here is an example of how to implement an IF/ELIF/ELSE loop:

```
if a == 3:
    print 'The value of a is:'
    print 'a=3'

if a == 'test':
    print 'The value of a is:'
    print 'a="test"'
    test_mode = True
else:
    print 'a!="test"'
    test_mode = False
    do_something_else()

if a == 1 or a == 2:
    pass # do nothing
elif a == 3 and b > 1:
    pass
elif a==3 and not b>1:
    pass
else:
    pass
```

A.12 While loops

```
a=1
while a<10:
    print a
    a += 1
```

A.13 For loops

```
for a in range(10):
    print a

my_list = [2, 4, 8, 16, 32]
for a in my_list:
    print a
```

A.14 Functions

Functions can be defined in python as follows:

```
def compute_sum(arg1,arg2):
        # implement function to calculate the sum of two numbers
        res=arg1+arg2
    return(res)
```

The function can be called by typing the function name. If the function returns a value or object, this is assigned to a variable as follows:

```
summa=compute_sum(3.0,7.0)
```

Otherwise, the function can just be called without setting it equal to any variable.

```
c=3

def change_global_c(val):
        # this function change the value of a global variable
    global c
    c=val

change_global_c(10)
```

A.15 Classes

Classes are a way to group a set of functions inside a container. These can be accessed using the . operator. The main purpose of classes is to define objects of a certain type and the corresponding methods. For example, we may want to define a class called 'square', containing the methods to compute the square properties, such as the perimeter and the area. The object is initialized by means of a "constructor":

```
class square:

    #the constructor:
    def __init__(self,side):
            self.side=side

           #area of the square:
    def area(self):
            return(self.side*self.side)

    #perimeter of the square:
    def perimeter(self):
            return(4.0*self.side)
```

We can then use the class to define a square object:

```
s=square(3.0) # a square with side length 3
print s.area()
print s.perimeter()
```

As in other languages (e.g. C++), python supports inheritance. A class can be used as an argument for another class. In this case the new class will inherit the methods of the parent class. For example:

```
class geometricalFigure(object):

    def __init__(self,name):
        self.name=name

    def getName(self):
        print 'this is a %s' % self.name

class square(geometrical_figure):

    #the constructor:
    def __init__(self,side):
        geometricalFigure.__init__(self,'square')
        self.side=side

    #area of the square:
    def area(self):
        return(self.side*self.side)

    #perimeter of the square:
    def perimeter(self):
        return(4.0*self.side)

class circle(geometrical_figure):

    #the constructor:
```

```
    def __init__(self,radius):
        geometricalFigure.__init__(self,'circle')
        self.radius=radius

    #area of the square:
    def area(self):
        return(3.141592653*self.radius**2)

    #perimeter of the square:
    def perimeter(self):
        return(2.0*self.radius*3.141592653)


s=square(3.0)
c=circle(3.0)
s.getName()
c.getName()
```

In the example above, `square` and `circle` are two examples of `geometricalFigure`. They have some specialized methods to compute the area and the perimeter, but both can access the method `getName`, which belongs to `geometricalFigure`, because they have inherited it from the parent class.

A.16 Modules

A module is a file containing Python definitions and statements (constants, functions, classes, etc). The file name is the module name with the suffix .py appended.

Modules can be imported in another script by using the `import` statement:

```
import modulename
```

The functions and statements contained in the module can be accessed using the . operator.

Modules can import other modules. It is customary but not required to place all import statements at the beginning of a module (or script, for that matter).

There is a variant of the import statement that imports names from a module directly into the importing module's symbol table. For example:

```
from modulename import something
```

A.17 Importing packages

Packages can be added to your python distribution by using either the `pip` or `easy_install` utilities. Anaconda has its own utility for installing a (limited) set of supported packages, called `conda`. To learn more, check out https://packaging.python.org/installing/

Packages can be used by importing modules and classes in the code as discussed above.

Some packages that we will use a lot:

- numpy: fundamental package for scientific computing with Python (powerful N-dimensional array object, sophisticated functions, tools for integrating C/C++ and Fortran code, useful linear algebra, Fourier transform, and random number capabilities);

- scipy: provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization;
- matplotlib: a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms;
- astropy: a community effort to develop a single core package for Astronomy in Python and foster interoperability between Python astronomy packages.

Other packages will be introduced in the examples.

B. Cosmological background

We review in this appendix those aspects of the standard cosmological model which are relevant for understanding gravitational lensing and its applications.

The standard model is based on the assumption of the so called *"Cosmological Principle"*, which states that the Universe is homogeneous and isotropic on large scales. The metric properties of the space-time are described by the Robertson-Walker metric, in which hypersurfaces of constant time are homogeneous and isotropic three-spaces, either flat or curved. The time dependence of these hypersurfaces is given by a scale factor, which is a function of time only.

Structures grow from initial density fluctuations via gravitational collapse. The origin of these initial seed perturbations is still unclear. The most common hypothesis is that their statistics is Gaussian. Given that their amplitude remains small for most of their evolution, the growth of the initial density perturbations can be appropriately described by the linear perturbation theory until late stages. After that, the growth of perturbations starts to depart from linearity and enters into the non-linear regime. Then, structure evolution becomes very complicated and numerical methods are required for a realistic description.

B.1 The Robertson-Walker metric

The standard cosmological models is based on the assumption of the *"Cosmological Principle"*. This is the assertion that, on sufficiently large scales (beyond those traced by the large-scale structure of the galaxy distribution), the Universe is homogeneous and isotropic. Originally introduced by Einstein and subsequent relativistic cosmologists without strict empirical justification, the Cosmological Principle is today accepted because it agrees with observations: data concerning radio galaxies, clusters of galaxies, quasars and the microwave background all demonstrate that the level of anisotropy of the universe on very large scales is about one part in 10^5.

The geometrical properties of the space-time are described by a metric. All events in the space-time have one time coordinate $x^0 = ct$, where c is the velocity of light and t is the proper time, and three space coordinates x^1, x^2, x^3. The interval between two events in the space-time can

be written as

$$ds^2 = g_{ij}dx^i dx^j \ , \tag{B.1}$$

where repeated suffixes imply summation and i, j both run from 0 to 3. The tensor g_{ij} is the metric tensor, which describes the space-time geometry.

The most general space-time metric describing a universe in which the Cosmological Principle is obeyed is the *Robertson-Walker* metric. Adopting this metric, Eq. (B.1) can be written as

$$ds^2 = (cdt)^2 + a(t)^2 \left[\frac{dr^2}{1-Kr^2} + r^2(d\theta^2 + \sin\theta^2 d\phi^2) \right] \ , \tag{B.2}$$

where r, θ and ϕ are spherical polar coordinates. As can be easily seen in Eq. (B.2), the distance between two points in space depends on time only through the scale factor $a(t)$, whose form will be shown later. Given that r, θ and ϕ are time independent variables, these coordinates are called *comoving*. The parameter K determines the curvature of spatial hypersurfaces. It is a constant which can be scaled to assume only the values $1, 0$ or -1. The case $K = 0$ corresponds to the flat, Euclidean space, whose properties are familiar. The other two cases, $K = 1$ and $K = -1$, correspond, respectively, to hyperspheres and spaces of constant negative curvature: the first is a closed space, with finite volume and no boundary; the second is an open (hyperbolic) space, i.e. infinite.

B.2 Redshift

If the scale factor $a(t)$ changes with time, i.e. if the universe expands or shrinks, photons which are emitted by a source are redshifted or blueshifted while propagating to the observer.

Consider a luminous source at comoving distance r, emitting photons whose wavelength is λ_e at time t_e, and an observer placed at the origin of the coordinate system ($r = 0$). We define the *redshift* z of the source as

$$z = \frac{\lambda_o - \lambda_e}{\lambda_e} \ , \tag{B.3}$$

where λ_o is the wavelength of radiation from the source measured by the observer at time t_o.

Given that radiation travels along null geodesics in the space-time (i.e. $ds^2 = 0$), we obtain from Eq. (B.2):

$$\int_{t_e}^{t_o} \frac{cdt}{a(t)} = \int_0^r \frac{dr'}{(1-Kr'^2)} = f(r) \ . \tag{B.4}$$

The same result of Eq. (B.4) can be obtained also for light which is emitted by the source at time $t'_e = t_e + \delta t_e$ and received by the observer at time $t'_o = t_o + \delta t_o$:

$$\int_{t'_e}^{t'_o} \frac{cdt}{a(t)} = f(r) \ . \tag{B.5}$$

Eq. (B.4) and (B.5) imply that, if δt_e and δt_o are small,

$$\frac{\delta t_o}{a_o} = \frac{\delta t_e}{a_e} \ , \tag{B.6}$$

where $a_o = a(t_o)$ and $a_e = a(t_e)$. Identifying the inverse of the time intervals t_o and t_e with the inverse frequencies of the observed and emitted radiation, ν_o and ν_e, Eq. (B.6) can be written as

$$\nu_e a_e = \nu_o a_o \ , \tag{B.7}$$

or equivalently,

$$\frac{a_e}{\lambda_e} = \frac{a_o}{\lambda_o} , \tag{B.8}$$

from which

$$1 + z = a_o / a_e . \tag{B.9}$$

This Eq. shows the relationship between the redshift z and the scale factor $a(t)$: if the universe expands and $a_o > a_e$, then the light observed from distant sources is redshifted ($z > 0$).

B.3 The Friedmann Equations

The geometry of space-time, expressed by the metric tensor g_{ij}, is related to the matter content of the universe, expressed by the energy-momentum tensor T_{ij}, through *Einstein's field equations*,

$$R_{ij} - \frac{1}{2} R g_{ij} - \Lambda g_{ij} = \frac{8\pi G}{c^4} T_{ij} , \tag{B.10}$$

where R_{ij} and R are the Ricci tensor and Ricci scalar, respectively.

The term Λ is called the *cosmological constant*. It was introduced by Einstein to enable for static cosmological solutions of the field equations. However, even after the expansion of the universe was observationally established, the cosmological constant refused to die. Its physical meaning can be easily understood by considering Eq. (B.10) in vacuum. In that case, the energy-momentum tensor is

$$T_{ij}^{vac} = -\frac{c^4 \Lambda}{8\pi G} g_{ij} . \tag{B.11}$$

In other words, if the cosmological constant differs from zero, the vacuum has non-zero energy density and pressure.

The energy-momentum tensor of the universe is that of an homogeneous perfect fluid, which is characterized by its density $\rho(t)$ and pressure $p(t)$. For the Robertson-Walker metric the Einstein's equations simplify to the *Friedmann equations*,

$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{8\pi G}{3}\rho - \frac{Kc^2}{a^2} + \frac{\Lambda c^2}{3} \tag{B.12}$$

$$\frac{\ddot{a}}{a} = -\frac{4}{3}\pi G \left(\rho + \frac{3p}{c^2}\right) + \frac{\Lambda c^2}{3} . \tag{B.13}$$

These two equations are not independent: the second can be recovered from the first if one takes the adiabatic expansion of the universe into account, i.e. if we assume that the change of internal energy equals minus the pressure times the change in proper volume,

$$\frac{d}{dt}\left[a^3(t)\rho(t)c^2\right] = -p\frac{da^3(t)}{dt} . \tag{B.14}$$

The time dependence of the scale factor $a(t)$ can then be determined by integrating these differential equations and by fixing its value at one instant of time. We choose $a = 1$ at the present epoch t_0.

B.4 Cosmological parameters

Before going further, it is convenient to define some parameters which will be largely used in the following sections. First of all, we introduce the parameter used to quantify the relative expansion rate of the universe,

$$H(t) \equiv \frac{\dot{a}(t)}{a(t)} \,, \tag{B.15}$$

which is called the *Hubble parameter*. Its value at the present epoch, $H_0 = H(t_0)$, is known as the *Hubble constant*. Current measurements of this quantity roughly fall in the range $H_0 = (50 \div 80)$ km s^{-1} Mpc^{-1}: the *HST Key Project* team find $H_0 = 72 \pm 8$ km s^{-1} Mpc^{-1} **freedman01** while Saha et al. **saha01** using their Cepheid calibration for a sample of galaxies with Type Ia supernovae, find a smaller value, $H_0 = 59 \pm 6$ km s^{-1} Mpc^{-1}. The uncertainty in H_0 is commonly expressed as $H_0 = 100h$ km s^{-1} Mpc^{-1}, with $h = (0.5 \div 0.8)$.

We then define the present *critical density* of the universe as

$$\rho_{0,\mathrm{cr}} \equiv \frac{3H_0^2}{8\pi G} \approx 1.9 \times 10^{-29} h^2 \mathrm{g\ cm^{-3}} \,, \tag{B.16}$$

where G is the gravitational constant.

The density of the universe in units of $\rho_{0,\mathrm{cr}}$ is the *density parameter*,

$$\Omega_0 \equiv \frac{\rho_0}{\rho_{0,\mathrm{cr}}} \,. \tag{B.17}$$

As will be discussed later, the value of Ω_0 is crucial for cosmology.

All the components of the universe contribute to its density: baryonic and non-baryonic matter, radiation and vacuum. As was pointed out in the previous section, a non-null cosmological constant means that the vacuum has a finite density. This density can be written in units of the critical density as

$$\Omega_{0\Lambda} \equiv \frac{\Lambda c^2}{3H_0^2} \,. \tag{B.18}$$

The acquisition of the latest results from high-redshift Type Ia supernovae searches and from microwave background experiments indicate that the total density parameter of the universe is very close to unity and that the largest contribution to it comes from the cosmological constant.

The values of Ω_0 and $\Omega_{0\Lambda}$ are related to the curvature of the spatial hypersurfaces. Using the previous definitions, Eq. (B.12) can be written for $a = a_0 = 1$ as

$$H_0^2(1 - \Omega_0 - \Omega_{0\Lambda}) = -Kc^2 \,, \tag{B.19}$$

from which we see that if $K = 0$, then $\Omega_0 + \Omega_{0\Lambda} = 1$. Moreover, the sign of K is positive if $\Omega_0 + \Omega_{0\Lambda} > 1$, negative otherwise.

Finally, we define the *deceleration parameter*

$$q_0 = -\frac{\ddot{a}a}{\dot{a}^2} \tag{B.20}$$

at $t = t_0$. The sign of this parameter tells us if the universe is in decelerated expansion or not.

B.5 Cosmological distances

Several types of distance between two points can be defined in a curved space-time. Indeed, distance definitions in terms of different measurement prescriptions generally lead to different distances.

The *proper distance* of P from P_0, which we can take as the origin of the polar coordinate system r, ϕ, θ, is the distance measured at time t by a chain of observers connecting P to P_0. From Eq. (B.2), this distance is

$$d_{\rm pr} = \int_0^r \frac{a dr'}{(1-Kr'^2)^{1/2}} = a f(r) \ , \tag{B.21}$$

where a is the scale factor at time t, $a = a(t)$, and the radial function $f(r)$ is

$$f(r) = \begin{cases} \arcsin r & K = 1 \\ r & K = 0 \\ \text{arcsinh } r & K = -1 \end{cases} \ . \tag{B.22}$$

Given the time dependence of the scale factor, also the proper distance changes with time. Therefore, the point P has a radial velocity with respect to P_0 given by

$$v_{\rm r} = \dot{a} f(r) = \frac{\dot{a}}{a} f(r) = H(t) d_{\rm pr} \ . \tag{B.23}$$

This Eq. is called the *Hubble Law*.

The proper distance at the actual time t_0 defines the *comoving distance*

$$d_{\rm c} = f(r) = a^{-1} d_{\rm pr} \ . \tag{B.24}$$

This is the distance on the spatial hyper-surface $t = t_0$ between the world-lines of points P and P_0 comoving with the cosmic flow.

The *angular diameter distance* is defined in analogy to the relation in the Euclidean space between the physical diameter $D_{\rm pr}$ of a source and the angle $\Delta\theta$ that it subtends. From Eq. (B.2), it results in $D_{\rm pr} = ar\Delta\theta$ and the angular diameter distance is then

$$d_{\rm A} = \frac{D_{pr}}{\Delta\theta} = ar \ . \tag{B.25}$$

The distance defined in such a way to preserve the Euclidean inverse-square law for the decrease of luminosity with distance from a source is called *luminosity distance*. If L is the amount of energy emitted by a source in P per unit time and l is the amount of energy received per unit time and per unit area by an observer in P_0, the luminosity distance between these two points is defined as

$$d_{\rm L} = \left(\frac{L}{4\pi l}\right)^{1/2} \ . \tag{B.26}$$

The area of a spherical surface centered on P and passing through P_0 at time t_0 is just $4\pi r^2$. The photons emitted by the source arrive at this surface having been redshifted by the expansion of the universe by a factor a. Also, photon arrival times are delayed by another factor a. Therefore, the flux l is given by

$$l = \frac{L}{4\pi r^2} a^2 \ , \tag{B.27}$$

from which

$$d_{\rm L} = a^{-1} r = a^{-2} d_{\rm A} \ . \tag{B.28}$$

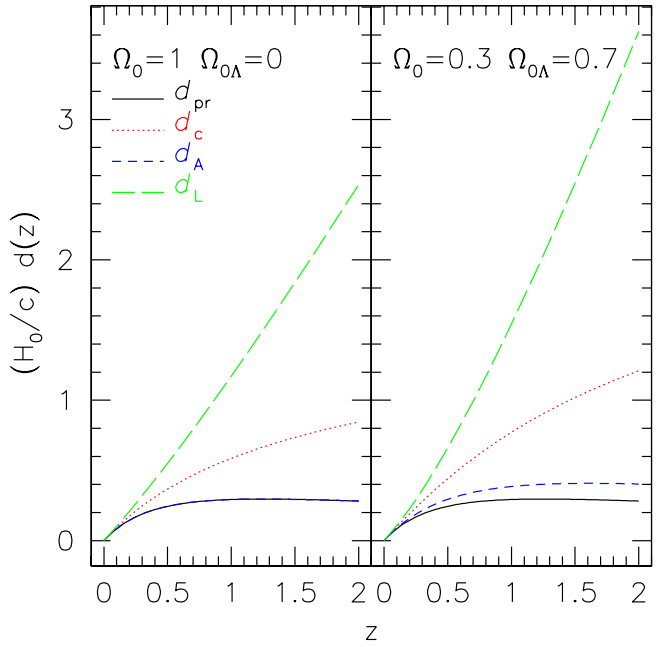


Figure B.5.1: *Four distance measures are plotted as a function of redshift in two cosmological models: a model with $\Omega_0 = 1$ and $\Omega_{0\Lambda} = 0$ (left panel) and a model with $\Omega_0 = 0.3$ and $\Omega_{0\Lambda} = 0.7$ (right panel). These are the proper distance $d_{\rm pr}$ (solid line), the comoving distance $d_{\rm c}$ (dotted line), the angular-diameter distance (short-dashed line) and the luminosity distance (log-dashed line).*

We plot the four distances $d_{\rm pr}$, $d_{\rm c}$, $d_{\rm A}$ and $d_{\rm L}$ as a function of redshift in Fig.(B.5.1). Given that they depend on the curvature of the space time, distances change for different values of Ω_0 and $\Omega_{0\Lambda}$: they are larger for lower cosmic density and higher cosmological constant. The largest differences are at high redshift, while at low redshift all distances follow the Hubble law,

$$\text{distance} = \frac{cz}{H_0} + O(z^2)\ . \tag{B.29}$$

B.6 The Friedmann models

We now introduce the standard cosmological models described by the equations (B.12) and (B.13). They receive their name from A. Friedmann, who derived them in 1922. The main assumptions on which these cosmological models are based are:

- the universe can be approximated by a perfect fluid with some density ρ and pressure p;
- the appropriate equation of state, relating pressure to density, can be cast, either exactly or approximately, in the form
$$p = w\rho c^2 \tag{B.30}$$

The perfect fluid is, in fact, quite a realistic approximation in many situations. For example, if the scale of particle interaction is much smaller than the scale of physical interest, as is the case for the global properties of the universe, the fluid can be treated as perfect. Even the second assumption on the fluid equation of state is quite appropriate in many cases of physical interest.

Ordinary matter, which can be considered as pressureless, is frequently called *dust*, and corresponds to the case $w = 0$. This approximation is valid in general for any non-relativistic gas, which can then be treated as a fluid of dust.

On the other hand, relativistic matter and radiation correspond to the case $w = 1/3$. Inserting these expressions into Eq. (B.14), we obtain:

$$\rho = \begin{cases} \rho_0 a^{-3} = \rho_0 (1+z)^4 & \text{for dust, } p = 0 \\ \rho_0 a^{-4} = \rho_0 (1+z)^3 & \text{for relativistic matter, } p = \frac{1}{3}\rho c^2 \end{cases} . \tag{B.31}$$

The energy density of relativistic matter therefore drops more rapidly than that of ordinary matter. This can be understood by considering a comoving box containing N particles. Let us assume that the box expands, with an expansion factor $a(t)$, and particles within it are neither created nor destroyed. In the case of non-relativistic matter, the density of particles simply changes as the inverse of the box volume, i.e. as a^{-3}. If the particles are relativistic, they behave like photons: not only their number density decreases as a^{-3}, but their wavelength λ is also increased by a factor a. Since the energy per particle is proportional to λ^{-1}, the total energy density decreases as a^{-4}.

The relativistic matter component candidates today are photons and neutrinos. Their cosmic density contribution, $\Omega_{R,0} = \Omega_{\gamma,0} + \Omega_{\nu,0}$, is estimated to be $\sim 3.2 \times 10^{-5} h^{-2}$. This is four orders of magnitude smaller than the today-estimated value of the matter density parameter of the universe, and can thus be considered negligible. At the epoch when the density of the relativistic matter component equaled that of the non-relativistic one, the expansion scale factor was

$$a_{\rm eq} = \frac{\Omega_{R,0}}{\Omega_0} = 3.2 \times 10^{-5} \Omega_0^{-1} h^{-2} . \tag{B.32}$$

Much before that epoch, i.e. for $a \ll a_{\rm eq}$, the expansion of the universe therefore was radiation-dominated.

An important property of the Friedmann models with no cosmological constant is that, if $w > -1/3$, they possess a point in time where $a = 0$, which is called the *Big-Bang singularity*. Indeed, inserting Eq. (B.30) into Eq. (B.12), we obtain

$$H^2(t) = \left(\frac{\dot{a}}{a}\right)^2 = H_0^2 a^{-2} \left[\Omega_{0w} a^{-(1+3w)} + (1 - \Omega_{0w})\right] , \tag{B.33}$$

where $\Omega_{0w} = \rho_{0w}/\rho_{\rm cr}$ is the actual density parameter of the fluid. Suppose that at some generic time t the universe is expanding, so that $\dot{a}(t) > 0$. If $w > -1/3$, from Eq. (B.12) we see that $\ddot{a} < 0$ for all t. The graph describing $a(t)$ therefore is necessarily concave and there must be a time, which we can set as $t = 0$, when $a(0) = 0$ and when the density diverges.

B.6.1 Single component models

We discuss in this Section the solutions of Eq. (B.12) for flat, open and closed models in the simple case of single-component universes.

The solution appropriate to a flat universe, i.e. with $\Omega_{0w} = 1$, without cosmological constant is known as the *Einstein-de Sitter universe*. In this case, integrating Eq. (B.12), we obtain

$$a(t) = \begin{cases} \left(\frac{t}{t_0}\right)^{2/3} & \text{for the matter-dominated universe, } w = 0 \\ \left(\frac{t}{t_0}\right)^{1/2} & \text{for the radiation-dominated universe, } w = \frac{1}{3} \end{cases} . \tag{B.34}$$

A general property of these models therefore is that the expansion parameter a grows indefinitely with time. Moreover, the deceleration parameter is constant and positive in both the cases of matter- and radiation-dominated universes. This means that the expansion in these models is decelerated.

The solutions for curved models are rather more complicated. At early times, they behave in a manner very similar to flat models, and the solutions seen for the Einstein-de Sitter model can be

applied. Then, when $a(t) \gg a(t^\star) = a^\star$, where $a^\star$ is given by

$$a^\star = \left| \frac{\Omega_{0w}}{1-\Omega_{0w}} \right|^{1/(1+3w)} , \tag{B.35}$$

solutions differ. In models with $\Omega_{0w} < 1$ (open universes), for $t \gg t^\star$ the scale factor grows with time as

$$a(t) \simeq a^\star \frac{t}{t^\star} , \tag{B.36}$$

and the deceleration parameter is approximately zero.

In models with $\Omega_w > 1$, at $t = t^\star$ the scale factor reaches a maximum $a_\mathrm{m} = a_\star$. After that time $a(t)$ starts to decrease symmetrically around a_m. At time $t = 2t^\star$ there is another singularity in a symmetrical position with respect to the *Big-Bang*, called the *Big-Crunch*.

Analytical solutions exist for both matter- and radiation-dominated universes. For a more detailed discussion, see **coles02**

B.6.2 Multiple component models

Considering a model universe made of matter, radiation and cosmological constant, Eq. (B.12) becomes

$$H^2(t) = H_0^2 \left[\frac{\Omega_{R,0}}{a^4} + \frac{\Omega_0}{a^3} + \frac{1-\Omega_0-\Omega_{0\Lambda}}{a^2} + \Omega_{0\Lambda} \right] . \tag{B.37}$$

There is generally no simple solution to this Equation. Qualitatively, we can say that at very early times the universe is flat and radiation-dominated. Even the term depending on the cosmological constant is negligible at that time. Therefore, solutions for the Einstein-de Sitter model are appropriate for describing the evolution of the scale factor during this period. Then, after equivalence, matter starts to dominate over radiation and the cosmological constant term becomes increasingly more significant.

Using Eq. (B.37), we can determine the dependence of Ω and Ω_Λ on the scale factor a. For a matter dominated universe, we find

$$\Omega(a) = \frac{8\pi G}{3H^2(a)}\rho_0 a^{-3} = \frac{\Omega_0}{a+\Omega_0(1-a)+\Omega_{0\Lambda}(a^3-a)} , \tag{B.38}$$

$$\Omega_\Lambda(a) = \frac{\Lambda c^2}{3H^2(a)} = \frac{\Omega_{0\Lambda} a^3}{a+\Omega_0(1-a)+\Omega_{0\Lambda}(a^3-a)} . \tag{B.39}$$

Whatever the values of Ω_0 and $\Omega_{0\Lambda}$ are at the present epoch, Eqs. (B.38) and (B.39) show that $\Omega \to 1$ and $\Omega_\Lambda \to 0$ for $a \to 0$. On the other hand, if $\Omega_0 + \Omega_{0\Lambda} \leq 1$, $\Omega \to 0$ and $\Omega_\Lambda \to 1$ monotonically for $a \to \infty$. Therefore, for this kind of models we can define a time t_Λ such that for $a \gg a_\Lambda = a(t_\Lambda)$, the dominant term in Eq. (B.37) is the cosmological constant one. Neglecting all the other terms, the solution of the Friedmann equations at that time is that of the so-called *de Sitter universe*, which is written in the form

$$a(t) \propto \exp\left[\left(\frac{1}{3}\Lambda \right)^{1/2} ct \right] . \tag{B.40}$$

This means that at a given time t_{acc}, the cosmological constant term starts to accelerate the expansion of the universe. This can be explained, if one remembers that the cosmological constant is proportional to the energy density and pressure of vacuum. In particular, from Eq. (B.11) we see that the vacuum has a negative-pressure equation of state:

$$p_\mathrm{vac} = -\rho_\mathrm{vac} c^2 . \tag{B.41}$$

This means that the pressure of vacuum can be interpreted as a source of gravitational repulsion. Therefore, we can assume that, when the matter density is sufficiently low, gravity looses its efficiency to decelerate the expansion of the universe and the vacuum pressure starts to accelerate it.

B.7 Structure formation

B.7.1 Linear growth of density perturbations

The standard model assumes that structure in the universe formed via gravitational collapse from initial density fluctuations. The origin of these fluctuations is still unclear, but the general idea is that they were generated during an inflationary epoch from quantum fluctuations. They are assumed to be uncorrelated and the distribution of their amplitudes is assumed to be Gaussian.

Density perturbations are characterized by the density contrast

$$\delta(\vec{r},a) = \frac{\rho(\vec{r},a) - \overline{\rho}(a)}{\overline{\rho}(a)} \,, \tag{B.42}$$

where $\overline{\rho}$ is the average cosmic density and ρ is the density at the position given by $\vec{r}$.

Until $\delta \ll 1$, it is possible to study the evolution of the density perturbations using the linear approach. First, we consider a universe dominated by baryonic matter, i.e. gas having pressure p. The time evolution of a small density perturbation δ is determined by two opposite forces: the first is the pressure force, which acts such as to cancel the density fluctuation; the second is the gravitational force, which tends to amplify it. If the fluctuation is contained in a region of radius R, the pressure force per unit mass can be written as

$$F_p \approx \frac{pR^2}{M} = \frac{v_s^2}{R} \,, \tag{B.43}$$

where $M = \rho R^3$ is the mass of the fluctuation and $v_s \sim \sqrt{p/\rho}$ is the sound velocity in the fluid. On the other hand, the gravitational force per unit mass is

$$F_g \approx \frac{GM}{R^2} = G\rho R \,. \tag{B.44}$$

The density perturbation can grow only if $F_g \geq F_p$. Equaling Eqs. (B.43) and (B.44), we obtain that this condition is satisfied only if the size of the region where the fluctuation happens is larger than the *Jeans radius*,

$$R_J = \left(\frac{v_s^2}{G\rho}\right)^{1/2} \,. \tag{B.45}$$

Density perturbations on scales $\lambda \geq R_J$ therefore are amplified. Perturbations on scales $\lambda < R_J$ do not grow and propagate as acoustic waves.

Linear theory shows that perturbations on scales larger than the Jeans radius grow like

$$\delta(a) \propto \begin{cases} a^2 & \text{before } a_{\rm eq} \\ a & \text{after } a_{\rm eq} \end{cases} \,, \tag{B.46}$$

as long as the Einstein-de Sitter limit holds. If $\Omega_0 \neq 1$ and $\Omega_{0\Lambda} \neq 0$, this approximation, however, does not apply anymore for $a \gg a_{\rm eq}$. Then, the linear growth of density perturbations is changed according to

$$\delta(a) = \delta_0 a \frac{g'(a)}{g'(1)} \equiv \delta_0 a g(a) \,, \tag{B.47}$$

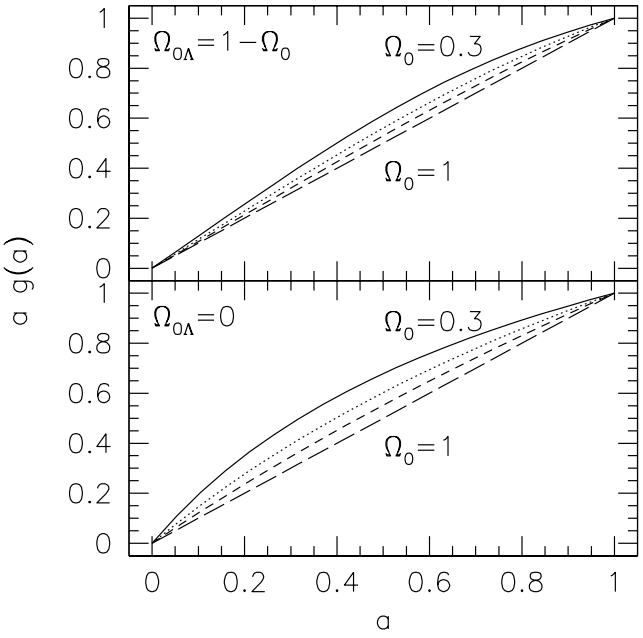


Figure B.7.1: *The growth function $ag(a)$ given in Eqs. (B.47) and (B.48) for different values of Ω_0 between $\Omega_0 = 0.3$ and $\Omega_0 = 1.0$: the solid, dotted, short-dashed and long-dashed lines refer to $\Omega_0 = 0.3, 0.5, 0.7$ and 1, respectively. Curves are displayed for flat models with $\Omega_{0\Lambda} = 1-\Omega_0$ (top panel) and for open models with $\Omega_{0\Lambda} = 0$ (bottom panel).*

where δ_0 is the density contrast linearly extrapolated to the present epoch and $g'(a)$ is the linear growth function, which depends on $\Omega(a)$ and $\Omega_\Lambda(a)$ as given by the fitting function

$$g'(a;\Omega_0,\Omega_{0\Lambda}) = \frac{5}{2}\Omega(a)\left[\Omega^{4/7}(a) - \Omega_\Lambda(a) + \left(1+\frac{\Omega(a)}{2}\right)\left(1+\frac{\Omega_\Lambda(a)}{70}\right)\right]^{-1} . \qquad \text{(B.48)}$$

The growth function $ag(a)$ is shown in Fig. (B.7.1) for a variety of parameters Ω_0 and $\Omega_{0\Lambda}$: the growth rate is constant for the Einstein-de Sitter model ($\Omega_0 = 1$; $\Omega_{0\Lambda} = 0$), while for low Ω_0 models it is higher for $a \ll 1$ and lower for $a \approx 1$. This feature means that structures form earlier in low-density than in high-density models, which will turn out to be of great importance for the further discussion on arc statistics.

Instead of using the size of fluctuations, we can deal with their mass. For a perturbation of density ρ and size R, the mass is defined as

$$M = \frac{4}{3}\pi R^3 \rho \ . \qquad \text{(B.49)}$$

This permits us to define the *Jeans mass*, $M_J \equiv M(R_J) = (4/3)\pi R_J^3 \rho$. Only perturbations of mass $M \geq M_J$ can be amplified by gravity.

Before equivalence, the Jeans mass M_J grows like $M_J \propto a^3$. Then, while the matter is coupled with radiation, it is constant, $M_J(a_{\rm eq}) \sim 10^{16} M_\odot/h$. When radiation and matter decouple, the Jeans mass drops down to $M_J \sim 10^5 M_\odot/h$. This is very important, because the perturbations which initially had mass $M < M_J$ and oscillated without being amplified, can now start to grow.

Density fluctuations can be on scales even larger than the size of the causally connected regions in the universe. This size is called the *cosmological horizon*. It is given by the distance by which a photon can travel in the time t since the Big Bang. Since the appropriate time scale is given by the

inverse Hubble parameter $H^{-1}(a)$, the horizon size is $R_H = cH^{-1}(a)$. The mass inside a sphere with radius R_H is called the *horizon mass*,

$$M_H = \frac{4}{3}\pi R_H^3 \rho \ . \tag{B.50}$$

Considering only the matter contribution to the density ρ, the horizon mass grows like $M_H \propto a^3$ before equivalence, being $M_H(a_{\rm eq}) \sim 10^{15} M_\odot/h$, and like $M_H \propto a^{3/2}$ after equivalence. When a perturbation has $M \leq M_H$ it can experience all the physical processes, like dissipation, which happen in the expanding universe.

Before the epoch $a_{\rm rec}$ of the hydrogen recombination, when matter and radiation are still coupled, the baryonic matter in the universe can be considered as a plasma of photons, electrons and protons. Dissipative effects arise in this plasma mainly due to the diffusion of photons. The size of the region which is affected by the dissipation effects grows with time like $R_S \propto (c^2 \tau_{\gamma e} t)^{1/2}$, where $\tau_{\gamma e}$ is the time scale for the collisions between photons and electron-proton pairs. The *dissipation mass* (or *Silk mass*) is then

$$M_S = \frac{4}{3}\pi R_S^3 \rho \ . \tag{B.51}$$

This mass grows like $M_S \propto a^{9/2}$ before equivalence, when $M_S(a_{\rm eq}) \sim 10^{11} M_\odot/h$, and like $M_S \propto a^{15/4}$ after equivalence. After recombination, the dissipation mass drops to zero. It is important to note that dissipation obliterates the fluctuations on all scales smaller than the Silk mass almost immediately. In a collisional fluid, no structure will therefore be formed on all mass scale less than the Silk mass at recombination, $M_S(a_{\rm rec}) \sim 10^{14} M_\odot/h$. Smaller structures can only form by fragmentation of the larger ones.

The main problem of this model for structure formation is that the cosmic microwave background reveals relative temperature fluctuations of order 10^{-5} on large scales. By the Sachs-Wolfe effect **sachs67** these temperature fluctuations reflect baryonic density fluctuations of the same order of magnitude at recombination. Given that recombination occurs at $a \sim 10^3$, Eq. (B.46) implies that density fluctuations today should only reach a level of 10^{-2}. Instead structure (e.g. galaxies) with $\delta \gg 1$ are observed.

This is one of the strongest arguments for the existence of an additional matter component which does not couple electromagnetically (probably only weakly), i.e. the *dark matter*. If this matter component exists, then fluctuations in that component can grow as soon as it decouples from the cosmic plasma, well before photons decouple from baryons to set the cosmic microwave background free.

We can consider the dark matter as a collisionless fluid. Previous definitions of Jeans mass, horizon mass and dissipation mass can be extended even to this fluid by substituting the sound velocity with the average velocity $v_\star$ of the collisionless particles. In this case fluctuations on mass scales less than the Jeans mass do not propagate as acoustic waves but are damped by the velocity dispersion of particles. This dissipation process is called *free streaming*. Perturbations are completely canceled when their mass is equal to the *free streaming mass*,

$$M_{FS} = \frac{4}{3}\pi R_{FS}^3 \rho_{DM} \ , \tag{B.52}$$

where R_{FS} is the free streaming length,

$$R_{FS} = a \int_0^t \frac{v(t')}{a(t')} {\rm d}t' \ . \tag{B.53}$$

If we assume that dark matter decoupled from the cosmic plasma when it was already non-relativistic (*Cold Dark Matter*), the Jeans mass and the free streaming mass are almost identical

and much lower than the masses of cosmological interest, being $M_J(a_{rec}) \simeq M_{FS}(a_{rec}) \sim 10^5 M_\odot/h$. Structures can thus form starting at the smallest scales. Baryons can then fall into the potential wells of the dark matter structures once they decouple from radiation.

Among the fluctuations of mass $M \leq M_J(a_{\rm rec})$, those which enter the horizon at $a \leq a_{\rm eq}$ cannot grow until radiation ceases dominating the expansion of the universe: before $a_{\rm eq}$, the expansion time-scale $t_{\rm exp} \sim (G\rho_{\rm R})^{-1/2}$ is smaller than the collapse time scale $t_{DM} \sim (G\rho_{\rm DM})^{-1/2}$ of dark-matter fluctuations. In other words, the radiation-driven expansion of the universe prevents dark-matter perturbations from collapsing. Since the time evolution of density perturbations before equivalence is $\delta \propto a^2$, fluctuations of mass $M < M_H(a_{\rm eq})$ which enter the horizon at $a_{\rm enter}$ are suppressed at equivalence by a factor

$$f_{\rm sup} = \left(\frac{a_{\rm enter}}{a_{\rm eq}}\right)^2 . \tag{B.54}$$

B.7.2 Density power spectrum

It is very convenient to think of the linear perturbations as of superpositions of plane waves which evolve independently while the fluctuations are still linear. In other words, we can decompose the density contrast into Fourier modes. This description should be valid only in flat space, but we can use it also for curved models because 1) at early times space can be considered flat in all cosmological models and 2) at late times the interesting scales of the density perturbations are much smaller than the curvature radius of the universe.

We therefore write the density contrast as

$$\delta(\vec{r}) = \int \frac{d^3k}{(2\pi)^3}\, \hat{\delta}(\vec{k}) e^{-i\vec{k}\vec{r}} , \tag{B.55}$$

where $\vec{k}$ is the wavevector and $\hat{\delta}$ denotes the Fourier transform of δ,

$$\hat{\delta}(\vec{k}) = \int d^3r\, \delta(\vec{r}) e^{i\vec{k}\vec{r}} . \tag{B.56}$$

The *Power spectrum* of the density fluctuations is then defined as

$$P(k) \equiv \langle |\hat{\delta}^2(\vec{k})| \rangle . \tag{B.57}$$

It is commonly assumed that the primordial power spectrum has a power-law form,

$$P_i(k) = A_i k^{n_i} . \tag{B.58}$$

Moreover, if we require that the primordial power spectrum is *scale invariant*, i.e. the fluctuations in the gravitational potential are independent of the length scale, the power index must be $n_i = 1$. This power spectrum is then called the *Harrison-Zel'dovich spectrum*, which is compatible with predictions from inflationary models, although they tend to require $n_i \lesssim 1$.

The primordial power spectrum is later modified by all the physical processes which affect the growth of density perturbations. In particular, we must take into account the suppression of the fluctuations which enter the horizon before equivalence. Given that the time $a_{\rm eq}$ when a perturbation of comoving length λ or wavenumber $k = \lambda^{-1}$ enters the horizon is $a_{\rm enter} \propto k^{-1}$ for $a_{\rm enter} \ll a_{\rm eq}$, the suppression factor (B.54) can be written as

$$f_{\rm sup} = \left(\frac{k_0}{k}\right)^2 , \tag{B.59}$$

where k_0 is the wavenumber corresponding to a perturbation which enters the horizon at $a_{\rm eq}$. Combining the primordial spectrum with this suppression of small scale modes, we obtain

$$P(k) \propto \begin{cases} k & \text{for } k \ll k_0 \\ k^{-3} & \text{for } k \gg k_0 \end{cases} . \tag{B.60}$$

Several methods are available for normalizing the power spectrum. The first is based on the measurements of the temperature of the cosmic microwave background. The COBE satellite has measured temperature fluctuations at the *rms* level of $\Delta T/T \sim 1.3 \times 10^{-5}$ at angular scales of $\sim 7^\circ$. If a shape for the power spectrum is assumed, these fluctuations can be translated into an amplitude of $P(k)$. However, due to the large scale of the measurements, this method sets the amplitude on large scale (small k) only.

The second possibility is that based on the local variance of galaxy counts. One then must assume that galaxies are biased tracers of underlying dark matter fluctuations. Conventionally, the variance of galaxy counts $\sigma_{8,\rm gal}$ is measured within spheres of radius 8 h^{-1}Mpc. Recent measurements find $\sigma_{8,\rm gal} \approx 1$. However, in order to set the amplitude of $P(k)$ properly, the correct expression for bias must be known.

Finally, the normalization can be made by using the local abundance of galaxy clusters **white93a**; **eke96**; **viana96** Indeed, galaxy clusters form by gravitational instability from dark matter density perturbations. Their spatial number density can then be used for setting the amplitude of the power spectrum on the scales of the density fluctuations collapsing to form galaxy clusters. This scale is of order $6 \div 10$ Mpc$/h$.

B.7.3 Non-linear evolution

Once the non linear regime is reached by the density perturbations, their evolution is separated from the universal expansion and they start to collapse to form virialized objects. It is very difficult to properly describe the collapse process unless one makes strong assumptions, for example on the symmetry of the perturbations.

The simplest assumption we can make is that the collapse is spherical. In absence of the cosmological constant, the radius R of a mass shell in a spherically symmetric density perturbation evolves according to

$$\frac{d^2R}{dt^2} = -\frac{GM}{R^2} , \tag{B.61}$$

where M is the mass within the mass shell. Integrating this equation, we obtain

$$\frac{1}{2}\left(\frac{dR}{dt}\right)^2 - \frac{GM}{R} = E . \tag{B.62}$$

If the energy $E < 0$, the shell collapses. Using this spherical approximation, the solution of Eq. (B.62) is characterized by

- a maximum expansion, $R = R_{\rm max}$, at $t = t_{\rm max}$, when the perturbation density is $\rho_{\rm p}(t_{\rm max}) = (3\pi/4)^2\overline{\rho}(t_{\rm max}) \approx 5.5\,\overline{\rho}(t_{\rm max})$, where $\overline{\rho}$ is the average density of the unperturbed background;
- a singularity, $R = 0$, at the collapse time $t_{\rm c} = 2t_{\rm max}$, when density ideally goes to infinity at the center.

In fact, when the density is high, small departures from spherical symmetry will result in the formation of shocks and considerable pressure gradients. Heating of the material will occur due to the dissipation of shocks which converts the kinetic energy of collapse into heat. The final result will therefore be an equilibrium state which is not a singular point but some extended configuration with radius $R_{\rm vir}$ and mass M. This happens when the system reaches the virial equilibrium. From

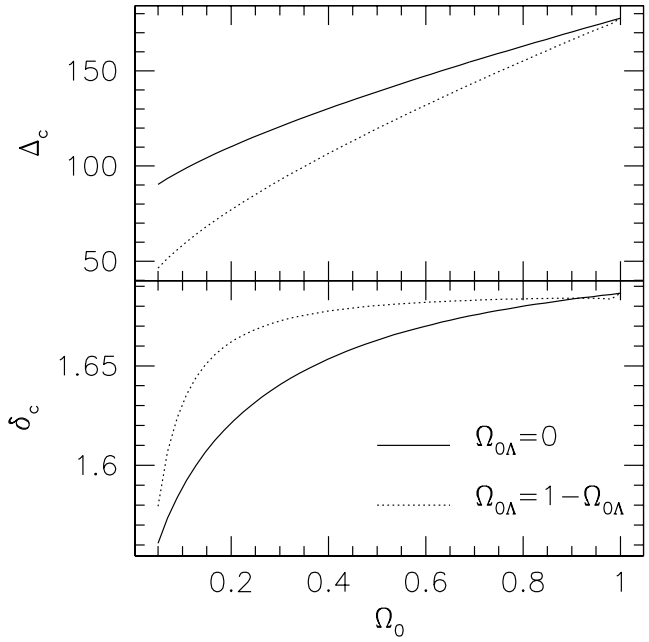


Figure B.7.2: *Lower panel: virial over density of collapsed objects in units of the critical density as a function of Ω_0. Results are plotted for open models with $\Omega_{0\Lambda}=0$ (solid line) and flat models with $\Omega_0+\Omega_{0\Lambda}=1$ (dotted line). Upper panel: linear extrapolation of the density contrast at the collapse time t_c. The solid and the dotted lines are as in the upper panel.*

the virial theorem, $R_{\rm vir}=R_{\rm max}/2$. In an Einstein-de Sitter universe, the perturbation density in units of the critical density at t_c is

$$\Delta_c \equiv \frac{\rho_p(t_c)}{\rho_{cr}(t_c)} = 18\pi^2 \approx 178\ . \tag{B.63}$$

An extrapolation of linear theory would give a density contrast $\delta_c=\delta(t_c)=3(12\pi)^{2/3}/20\approx 1.687$ with a weak dependence on the cosmological parameters.

We show in Fig. (B.7.2) how Δ_c and δ_c for halos collapsing at $z_c=0$ change for a variety of parameters Ω_0 and $\Omega_{0\Lambda}$. The critical overdensity δ_c has only a weak dependence on Ω_0 for both open models with $\Omega_{0\Lambda}=0$ and flat models with $\Omega_0+\Omega_{0\Lambda}=1$. The dependence of Δ_c on the cosmological models is much stronger.

Of course, non-linearity affects the shape of the power spectrum $P(k)$ in a very complicated way. Numerical methods are required for properly evaluating the non-linear power spectrum. Analytic formulae can be obtained only under some strong assumptions. For example, starting from the *ansatz* that the two point correlation functions in the linear and non-linear regimes are related by a general scaling relation **hamilton91** analytic formulae describing the non-linear behavior of $P(k)$ have been derived **jain95**; **peacock96**; **smith02** We show in Fig. (B.7.3) the CDM power spectrum corresponding to an Einstein-de Sitter model with $h=0.5$ and normalized to the local cluster abundance. The solid line shows the results obtained by assuming a primordial Harrison-Zel'dovich spectrum and linear evolution of the density perturbations on all scales. The dashed curve shows the non-linear evolution of the previous spectrum to $z=0$. Non-linearity affects the small-scale (large k) part of the spectrum, because small scale perturbations are the first which enter the non-linear regime.

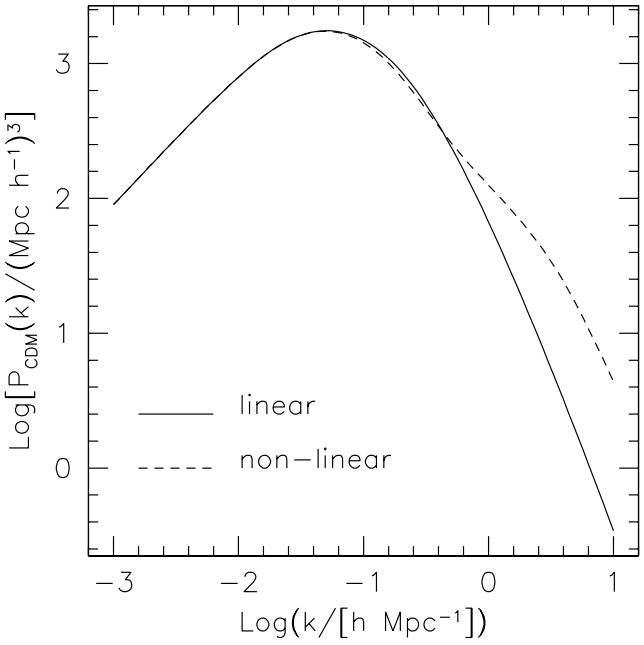


Figure B.7.3: *CDM power spectrum, normalized to the local cluster abundance, for an Einstein-de Sitter universe with $h = 0.5$. The solid curve shows the linear power spectrum, extrapolated to the present time; the dashed line shows its non-linear evolution.*

B.8 Mass function

The mass distribution of dark matter halos undergoing spherical collapse in the framework of CDM models is described by the Press & Schechter function **press74** The number density of collapsed lumps at redshift z with mass in the range $[M, M+dM]$ is

$$n(M,z)dM = \frac{\overline{\rho}}{M} f(\nu) \frac{d\nu}{dM} dM \,, \tag{B.64}$$

where $\overline{\rho}$ is the universe mean density at redshift z. The function f depends only the variable $\nu = \delta_c(z)\sigma_M$ and is normalized such that $\int f(\nu)d\nu = 1$. $\delta_c(z)$ is the linearly extrapolated density contrast of halos collapsed at redshift z (see previous section). The r.m.s. density fluctuation at the mass scale M, σ_M, is given by

$$\sigma_M = \frac{1}{2\pi^2} \int_0^\infty dk\, k^2 P(k) W^2(kR) \,, \tag{B.65}$$

where $W(kR)$ is the Fourier transform of the window function, which describes the shape of the volume from which the collapsing object is accreting material. Finally, R is the comoving size of the fluctuation of mass M.

In their original derivation of the cosmological mass function, Press & Schechter obtained

$$f(\nu) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\nu^2}{2}\right) . \tag{B.66}$$

The mass function has been used in this form for more than a decade. However, the last generation of N-body simulations revealed significant deviations of the original Press & Schechter function from the numerical description of the mass distribution of dark matter halos. Correcting the

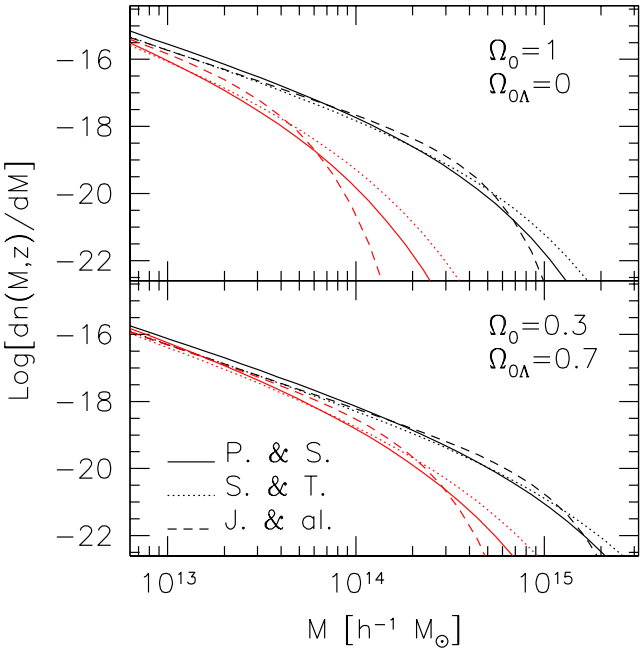


Figure B.8.1: *Differential mass function of dark matter halos. Three different mass functions are showed: the original Press & Schechter function (solid lines), its modification obtained by including non-spherical collapse by Sheth & Tormen (dotted lines) and the mass function derived from numerical simulations by Jenkins et al. (dashed lines). Results are plotted for a flat model with* $\Omega_0 = 0.3$ *and* $\Omega_{0\Lambda} = 0.7$ *and for an open model with* $\Omega_0 = 0.3$ *and* $\Omega_{0\Lambda} = 0$*. Black and red lines show the mass functions of halos at redshift* $z = 0$ *and* $z = 1$*, respectively.*

Press-Schechter approach by incorporating the effects of non-spherical collapse, Sheth & Tormen **sheth99** found

$$f(\nu) = \sqrt{\frac{2A}{\pi}} C \left(1 + \frac{1}{(A\nu^2)^q}\right) \exp\left(-\frac{A\nu^2}{2}\right), \tag{B.67}$$

where $A = 0.707$, $C = 0.3222$ and $q = 0.3$. This equation reduces to the Press & Schechter expression for $A = 1$, $C = 0.5$ and $q = 0$. Fitting the results of N-body simulations, Jenkins et al. **jenkins01** found a formula which is statistically not distinguishable from the Sheth & Tormen one with $A = 0.75$.

We plot in Fig. (B.8.1) the three different mass functions by Press & Schechter, Sheth & Tormen and Jenkins et al. The original Press & Schechter function underpredicts the abundance of large mass halos with respect to the Sheth & Tormen and the Jenkins et al. functions. The last two are very close to each other; there is only a small difference in the very high mass tail.

B.9 Quintessence models

As seen in the previous sections, the cosmological constant is strongly connected to the energy density of vacuum. Actually most favored cosmological models predict that today this vacuum energy density is of the same order of magnitude as the amount of dark and baryonic matter energy density in the universe, $\varepsilon \approx (10^{-3}\text{eV})^4$. This number is tiny in terms of the natural scale of primordial energy density given by the Planck mass $M_p = 1.22 \times 10^{19}$ GeV. A dominant radiation or matter energy density decreases as $\rho \sim M_p^2\, t^{-2}$ and the present age of the universe is $t_0 \approx 1.5 \times 10^{10}$

yr. It is therefore very easy to explain the smallness of the actual matter energy density in terms of the long duration of the cosmological expansion. On the other hand, assuming that the vacuum energy density is constant, it is very difficult to understand why the cosmological constant is so small today.

In order to solve this problem, it has been proposed that even Λ might be very small now because it has been rolling toward zero for a very long time. This idea that the universe contains nearly to homogeneous dark energy, called *Quintessence*, that approximates a time-variable cosmological "constant" arose also in particle physics, through the discussion of phase transition in the early universe and through the search for a dynamical cancellation of the vacuum energy density. We shall not present a rigorous treatment of quintessence models here, but only summarize some basic concepts. For a more quantitative discussion, we refer to the excellent reviews by Peebles & Ratra **peebles02** and by Wetterich **wetterich02**

Dark energy is usually modeled as that of a homogeneous scalar field, Φ. In this case, if spatial curvature can be neglected, the field equation is

$$\ddot{\Phi} + 3\frac{\dot{a}}{a}\dot{\Phi} + \frac{dV}{d\Phi} = 0 \,, \tag{B.68}$$

where V is the potential energy density, which is a function of the field Φ.

The energy-momentum tensor of this homogeneous field is diagonal in the rest frame of an observer moving such that the universe appears isotropic, and its time and space parts along the diagonal define the energy density and pressure,

$$\rho_\Phi = \frac{1}{2}\dot{\Phi}^2 + V(\Phi) \,, \tag{B.69}$$

$$p_\Phi = \frac{1}{2}\dot{\Phi}^2 - V(\Phi) \,. \tag{B.70}$$

The general form of the equation of state relating ρ_Φ to p_Φ is again

$$p_\Phi = w\rho_\Phi c^2 \,, \tag{B.71}$$

with $w < -1/3$. If the scalar field varies slowly in time ($\dot{\Phi}^2 \ll V(\Phi)$), the field energy approximates the effect of the Einstein's cosmological constant with $p_\Phi \simeq -\rho_\Phi c^2$.

This condition is verified during the inflationary epoch but not at its the end, when $V \sim 0$. When this happens, Φ oscillates. The scalar field is supposed to vary rapidly enough to produce the entropy of our universe and the field or the entropy may produce the baryons, leving ρ_Φ small or zero. If, after inflation, the time evolution of ρ_Φ starts to slow down and becomes slower than that of the matter density, there comes a time when ρ_Φ starts to dominate and the universe appears to have a cosmological constant.

There are many and well justified forms for the potential of this slowly evolving field (e.g. Lucchin & Matarrese **lucchin85**). A very simple model assumes a potential of the form

$$V = \frac{Q}{\Phi^\alpha} \,, \tag{B.72}$$

where the constant Q has dimensions of mass raised to the power $\alpha + 4$. Using this potential, the ratio of the mass densities in the scalar field and in the matter or radiation turns out to be

$$\frac{\rho_\Phi}{\rho} \propto t^{4/(2+\alpha)} \,. \tag{B.73}$$

In the limit where the parameter $\alpha = 0$, ρ_Φ is constant and this model is equivalent to the Einstein cosmological constant. Solutions as those obtained by using this potential for $\alpha > 0$ have two properties that seem desiderable. First, they are said to be attractors **ratra88** or trackers **steinhardt99**

meaning that they are asymptotic solutions for a broad range of initial conditions at high redshift. For example, energy distribution becomes nearly homogeneous even when gravity has collected the other matter into non-relativistic clumps. Second, the energy density in the attractor solution decreases less rapidly than that of matter and radiation.