Introduction to Gravitational Lensing

With python examples

Massimo Meneghetti

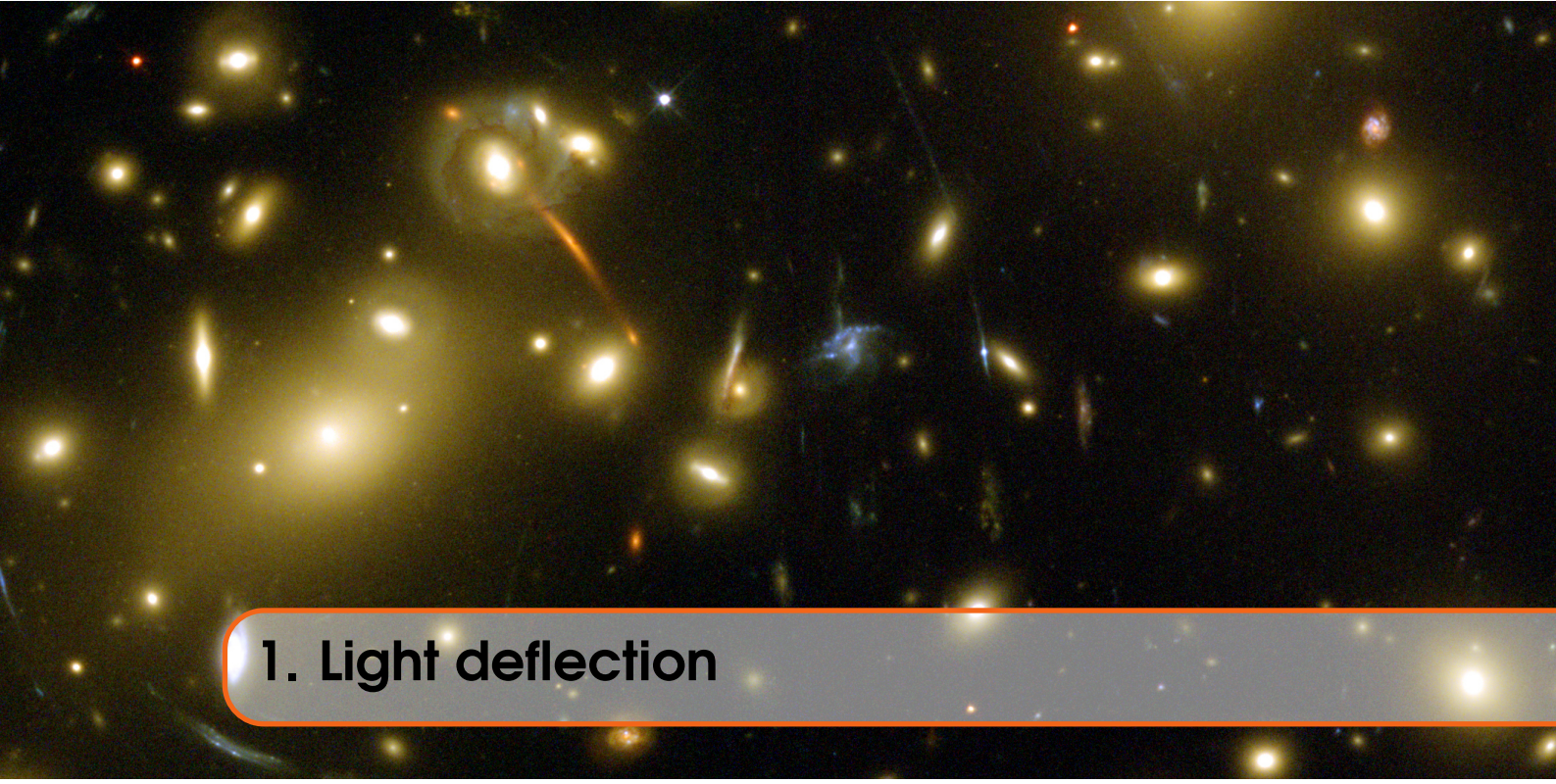
PUBLISHED BY MASSIMO MENEGHETTI

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.

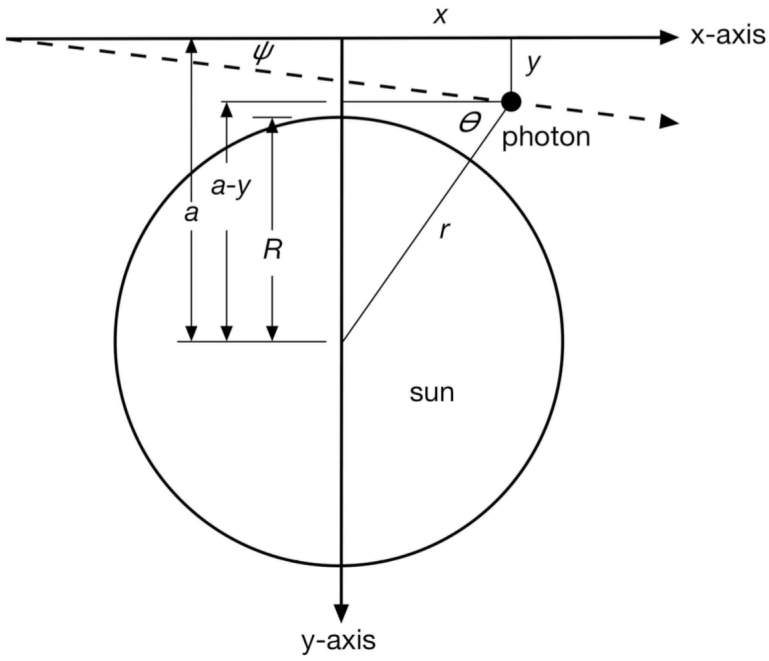


Figure 1.1.1: *Schematic view of a photon grazing the surface of the Sun (from V. J. Stenger, 2013).*

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

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

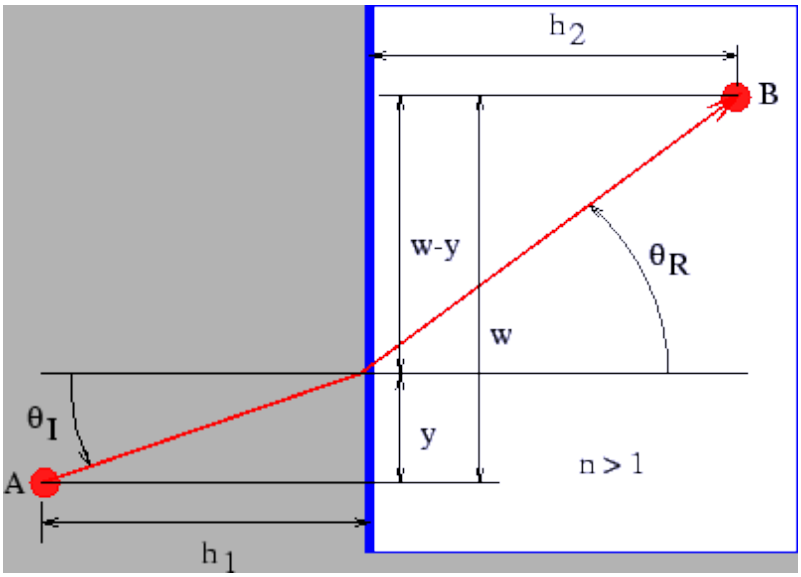


Figure 1.2.1: *Definition sketch for deriving Snell's law of refraction from Fermat's principle. The shaded area has refractive index $n > 1$*

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. ■

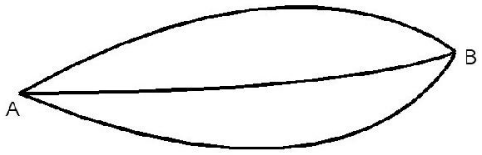
Taking inspiration from the Exercise above, we attempt to treat the deflection of light in a general relativity framework as a refraction problem. We need an refractive index n because Fermat's principle says that light will follow the path which makes extremal the travel time,

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

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

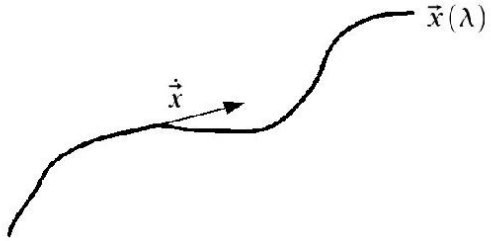
$$\left|\frac{\mathrm{d}\vec{x}}{\mathrm{d}\lambda}\right| = |\dot{\vec{x}}| = (\dot{\vec{x}}^2)^{1/2} \,. \tag{1.24}$$

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,

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

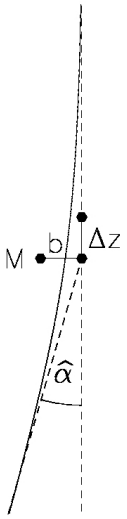
or, in other words, the integral over the "pull" of the gravitational potential perpendicular to the light path. Note that $\vec{\nabla}\Phi$ points away from the lens center, so $\hat{\vec{\alpha}}$ points in the same direction.

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.

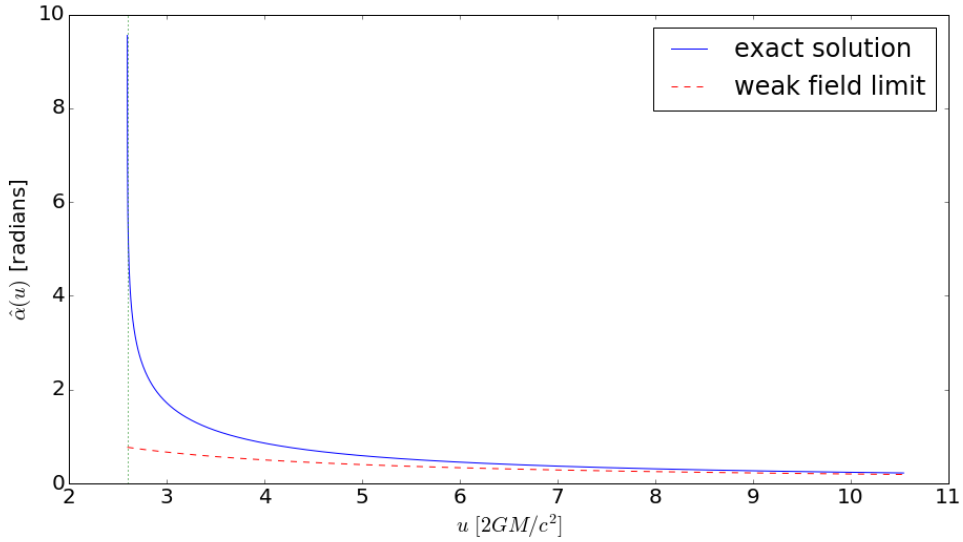


Figure 1.2.2: *Deflection angle by a compact lens as a function of the photon impact parameter. Shown are the exact solution of the geodesic equations for a Schwarzschild metric (solid line) and the solution in the weak field approximation (dashed line). The dotted vertical line shows the impact parameter,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

$$\vec{\tilde{\alpha}}(\vec{k}) = 2\pi\tilde{\kappa}(\vec{k})\vec{\tilde{K}}(\vec{k})$$

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

$$\vec{K}(\vec{x}) = \frac{1}{\pi}\frac{\vec{x}}{|\vec{x}|^2}$$

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/np.pi
        ky=ky/norm/np.pi
        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 = 2.0/(self.kappa.shape[0])/(np.pi)**2*\
                fftengine.fftshift(fftengine.ifftn(2.0*\
                np.pi*density_ft*kernelx_ft))
        alphay = 2.0/(self.kappa.shape[0])/(np.pi)**2*\
                fftengine.fftshift(fftengine.ifftn(2.0*\
                np.pi*density_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.

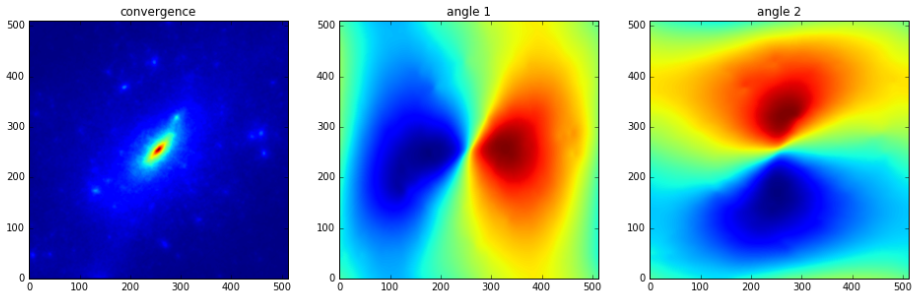


Figure 1.5.1: *Left panel: the surface density (convergence) map of the lens. Middle and right panels: maps of the two components of the deflection angles.*

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 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")
```

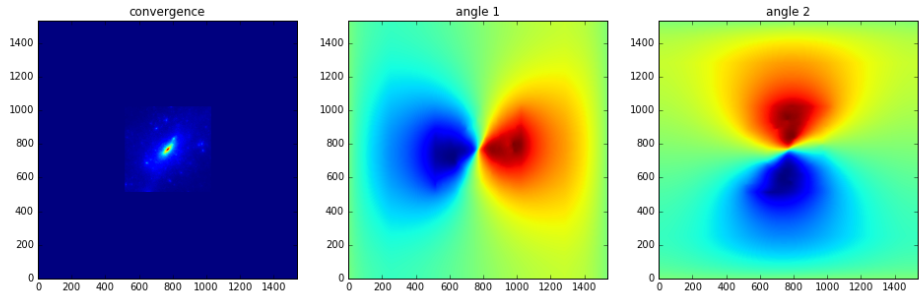


Figure 1.5.2: *The figure shows the same maps as in Fig. 1.5.1, but with zero-padding. Indeed, as shown in the left panel, the lens is surrounded by a frame of zeros, and the deflection angle maps are computed on an area which has double the size of the maps in Fig. 1.5.1.*

```
ax[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")
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..

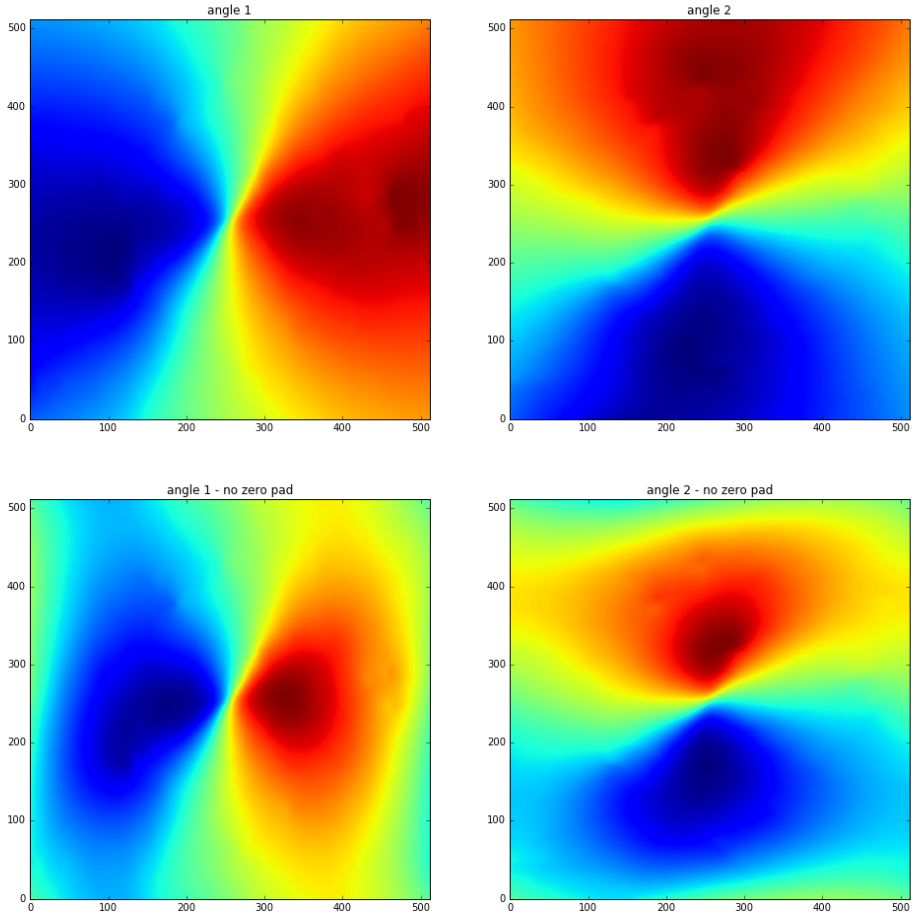
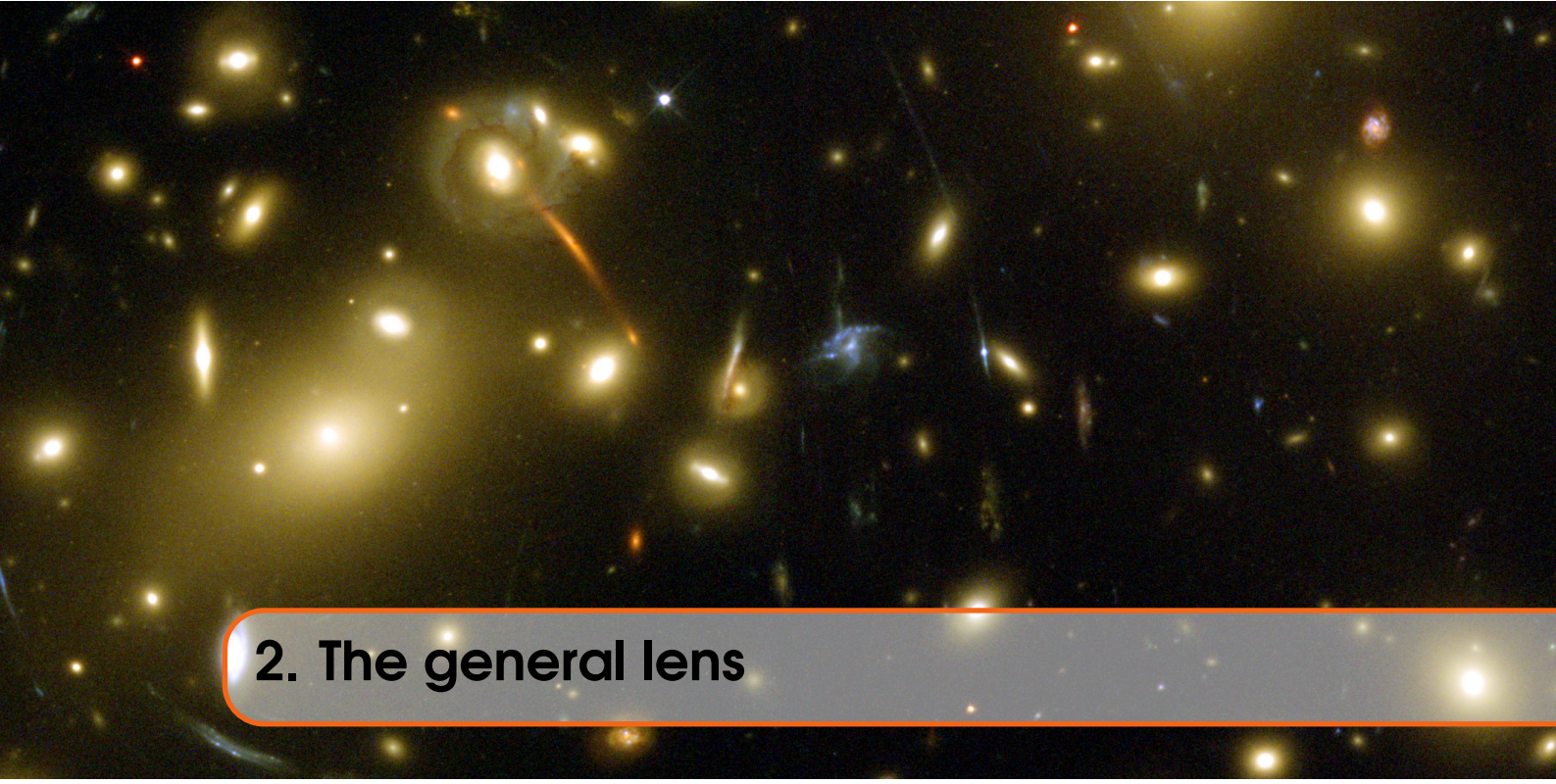


Figure 1.5.3: *The upper panels show the same two maps displayed in the middle and right panels of Fig. 1.5.2, which have been cropped to match the original size of the input convergence map. The bottom panels show the maps obtained without padding, for comparison.*



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 lenses and sources in our own galaxy). Hogg (1999) presents a concise summary of cosmological

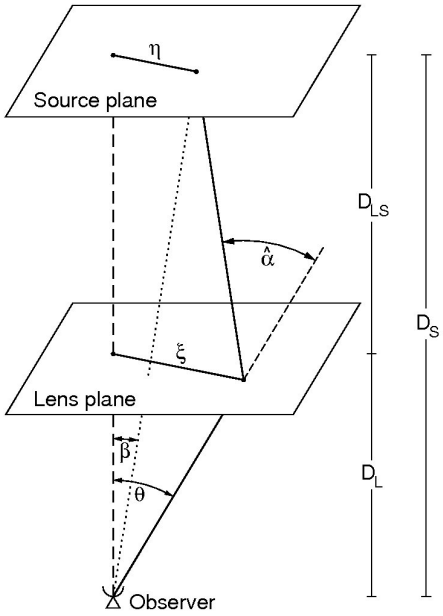


Figure 2.1.1: *Sketch of a typical gravitational lensing system. Figure from Bartelmann and Schneider (2001).*

distances. More in-depth discussions can be found 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**:

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

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

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

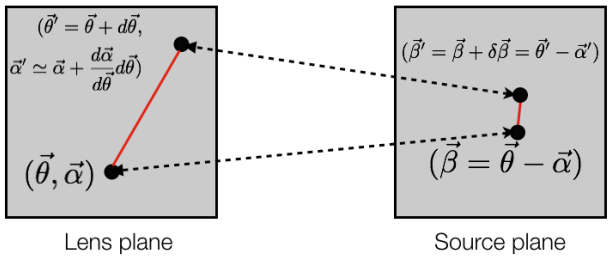


Figure 2.3.1: *Linear mapping between the lens and the source plane, assuming a slowly varying deflection angle.*

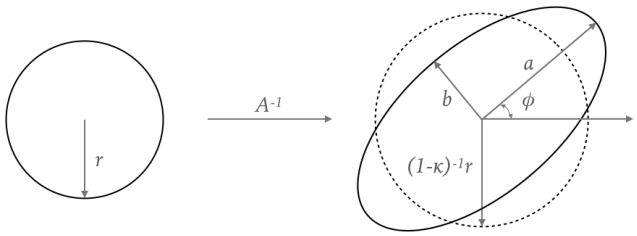


Figure 2.3.2: *Distortion effects due to convergence and shear on a circular source.*

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.

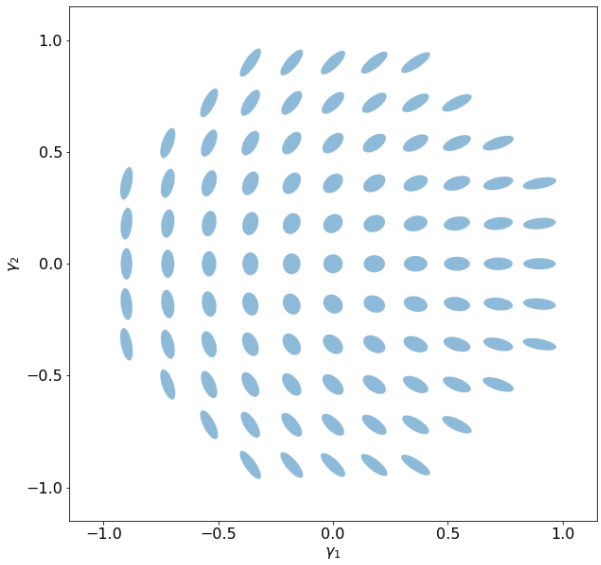


Figure 2.3.3: *Orientation of the images of a circular source for different values of γ_1 and γ_2.*

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 Occurrence of images

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*.

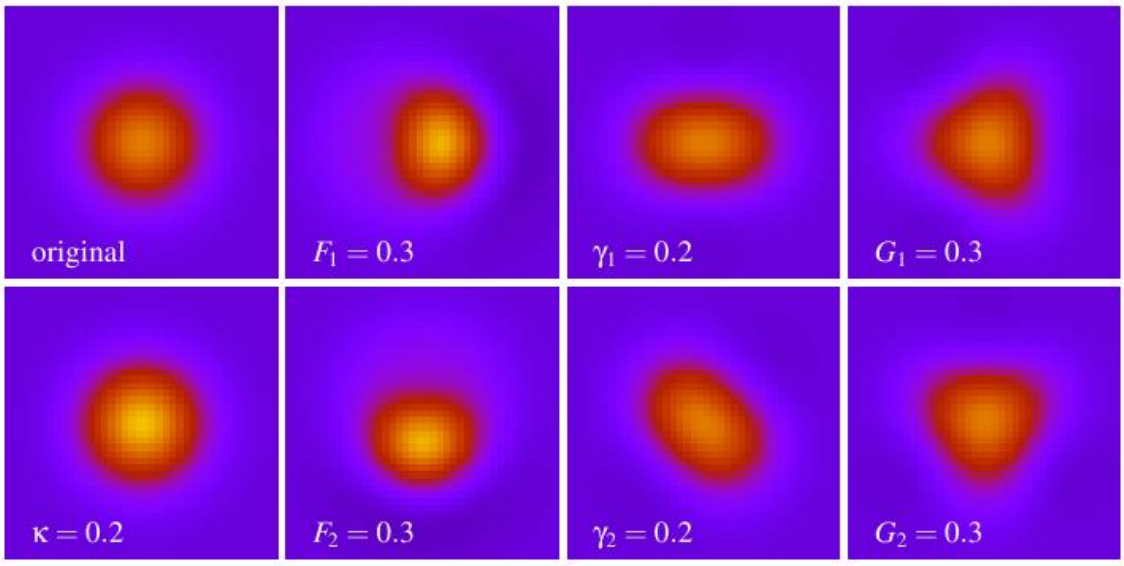


Figure 2.5.1: *First and second order distortions on the image of a circular source. The unlensed source is shown in the top left panel. The convergence simply changes the size (bottom left panel). While the shear deforms the image such that it becomes elliptical (third column of panels from the left),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

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

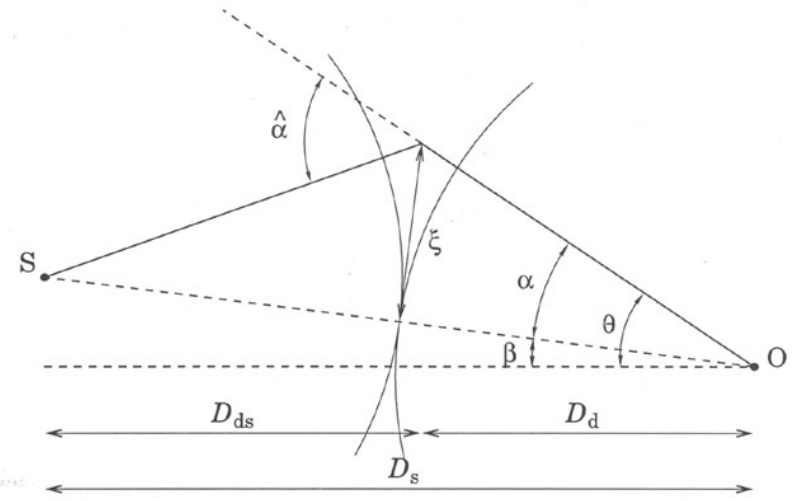


Figure 2.6.1: *Illustration of the geometrical time delay.*

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

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

The quantities

$$D_{\Delta t} = (1+z_L)\frac{D_S D_L}{D_{LS}} \tag{2.81}$$

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.

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

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

We can distinguish between three types of image:

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 magnification;

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

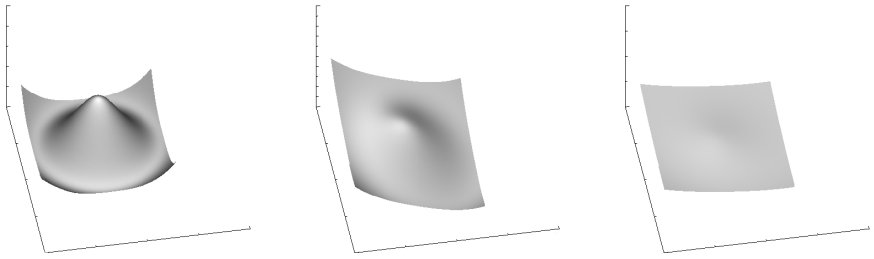


Figure 2.6.2: *Time delay surfaces of an axially symmetric lens for three different source positions. Right panel: source and lens are perfectly aligned along the optical axis; middle panel: the source is no more aligned with the lens. Its projected position on the lens plane is moved along the line $x_1 = x_2$; right panel: the source is moved to an even larger angular distance from the optical axis.*

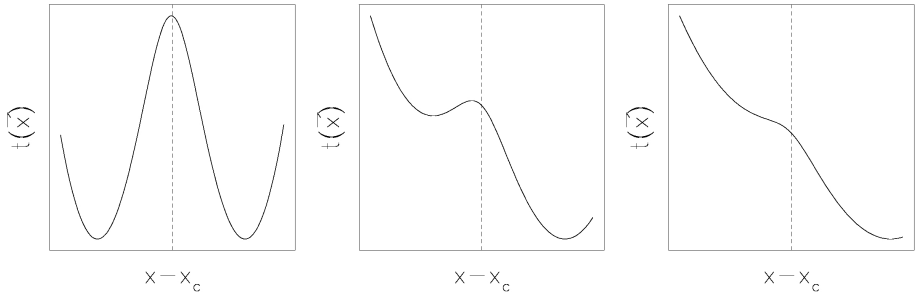


Figure 2.6.3: *Profiles of the time delay surfaces displayed in Fig. (2.6.2) along the line $x_1 = x_2$.*

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 magnification. The interpretation of a negative μ is that the parity of the image is flipped compared to the source;
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.

Since the Hessian matrix describes the local curvature of the time delay surface, the smaller is the curvature along one direction at the position where the image forms, the larger is its magnification along the same direction. We display in Fig. (2.6.2) some examples of the time delay surface for a general axially symmetric lens with core. The density profile of this lens scales with radius as r^{-2} outside the core. The surfaces are plotted for three different source position $\vec{\beta}$: in the left panel the source and the lens are perfectly aligned along the optical axis passing through the lens center ($\vec{y} = 0$ and $\vec{\theta} = 0$); in the middle and right panel, the source is moved far away, increasing its angular distance from the optical axis. In order to better see where the minima and the maxima arise, we show in Fig. (2.6.3) the profile along the line $x_1 = x_2$ of the same surfaces. When the source and the lens are perfectly aligned, the minima of the time delay surface are located on a ring and the maximum is at the lens center. The source therefore is mapped to a ring image of type I (the so called *Einstein Ring*) and to a central type III image. This last one is generally demagnified, since the curvature of the time delay surface here is large for density profiles peaked at the lens center.

As the source is moved far away from the optical axis, the time delay surface deforms. In particular, the ring breaks, leading to the formation of a minimum and of a saddle point. Three

images therefore arise. In the case displayed in the middle panel of Fig. (2.6.2), the type I image at the minimum and the type II image at the saddle point are stretched in the tangential direction, since the local curvature of the time delay surface is small in that direction. This explains the formation of tangential arcs in galaxy clusters. However, as the source is moved to even larger angular distances from the optical axis, the saddle point and the maximum move much closer to each other, while the minimum follows the source. The local curvature of the time delay surface in the radial direction becomes smaller between the saddle point and the maximum as they get closer. The images arising at this two points therefore are stretched towards each other. Then a radial image forms. When the saddle point and the maximum point touch, two images disappear and only the image arising at the minimum of the time delay surface remains (see right panels of Fig. (2.6.2) and Fig. (2.6.3)).

Here follows a number of other important properties of the 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;
- 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 continously 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)
```

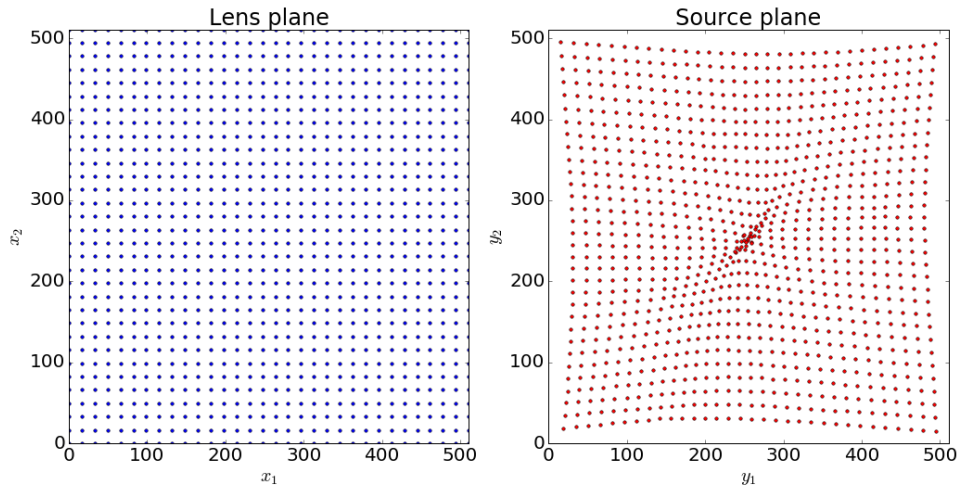


Figure 2.7.1: *Ray-tracing through a regular grid on the lens plane (left panel). The arrival positions of the light-rays on the source plane are shown in the right panel. The lens is the same used in Sect. 1.5.2.*

```
# 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 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])))
        pix=1 # pixel scale (now using pixel units)
        #Compute Laplace operator in Fourier space = -4*pi*l*l
        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.

2.8 To be done

- use the relation between the potential and the convergence to derive the potential from the mass distribution
- use the potential to derive, via differentiation, the properties of the lens: convergence, shear, deflection angles, magnification
- apply lensing distortions to first and second order to a circular source.

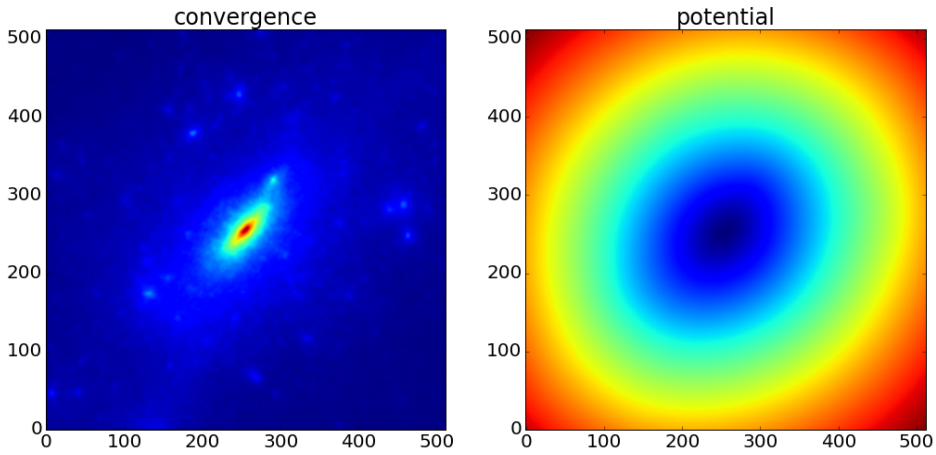
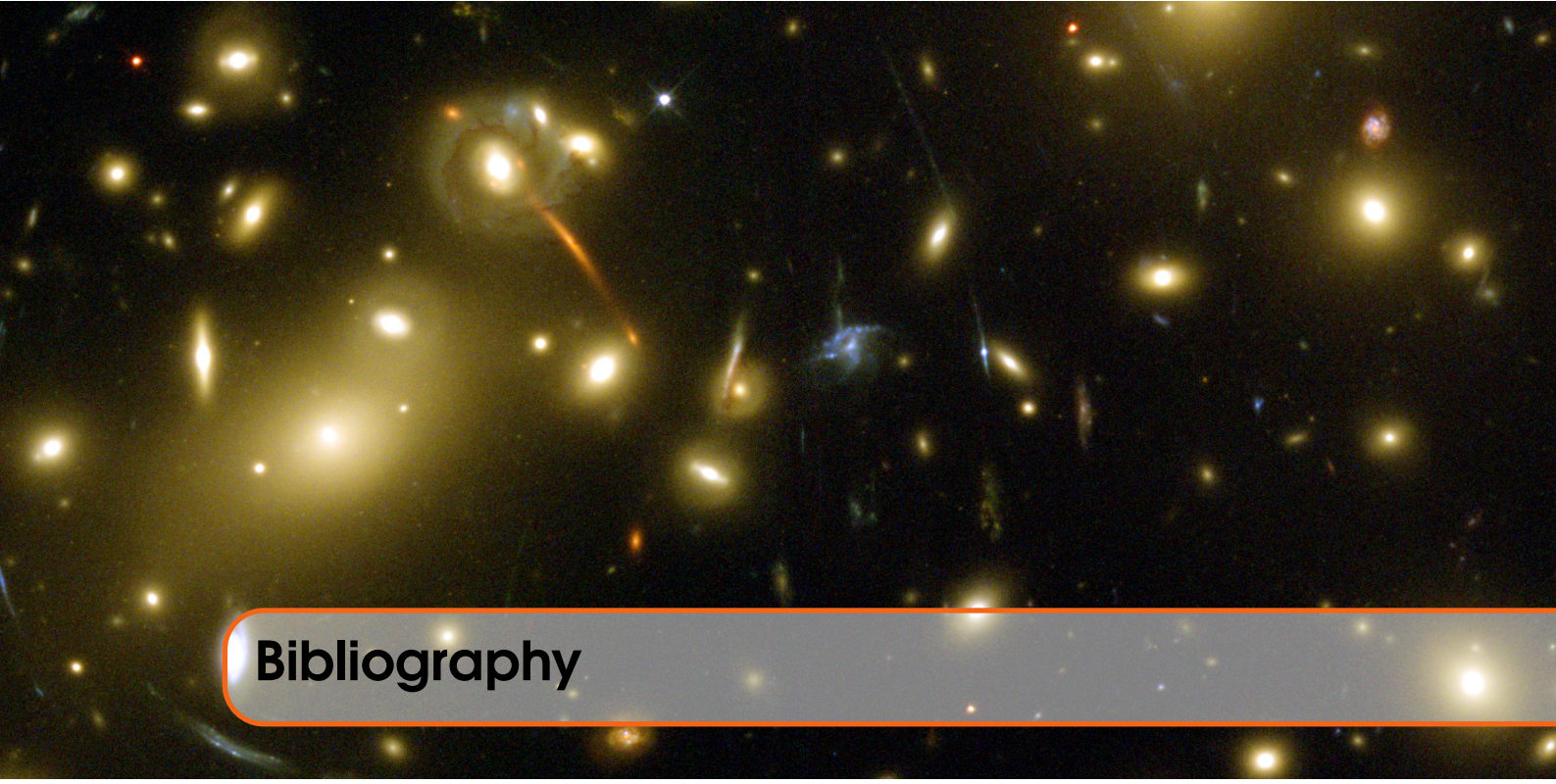


Figure 2.7.2: *Maps of the convergence and of the lensing potential maps for the same lens used in Sect. 1.5.2*

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 16).

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 16).

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 28).

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 15).

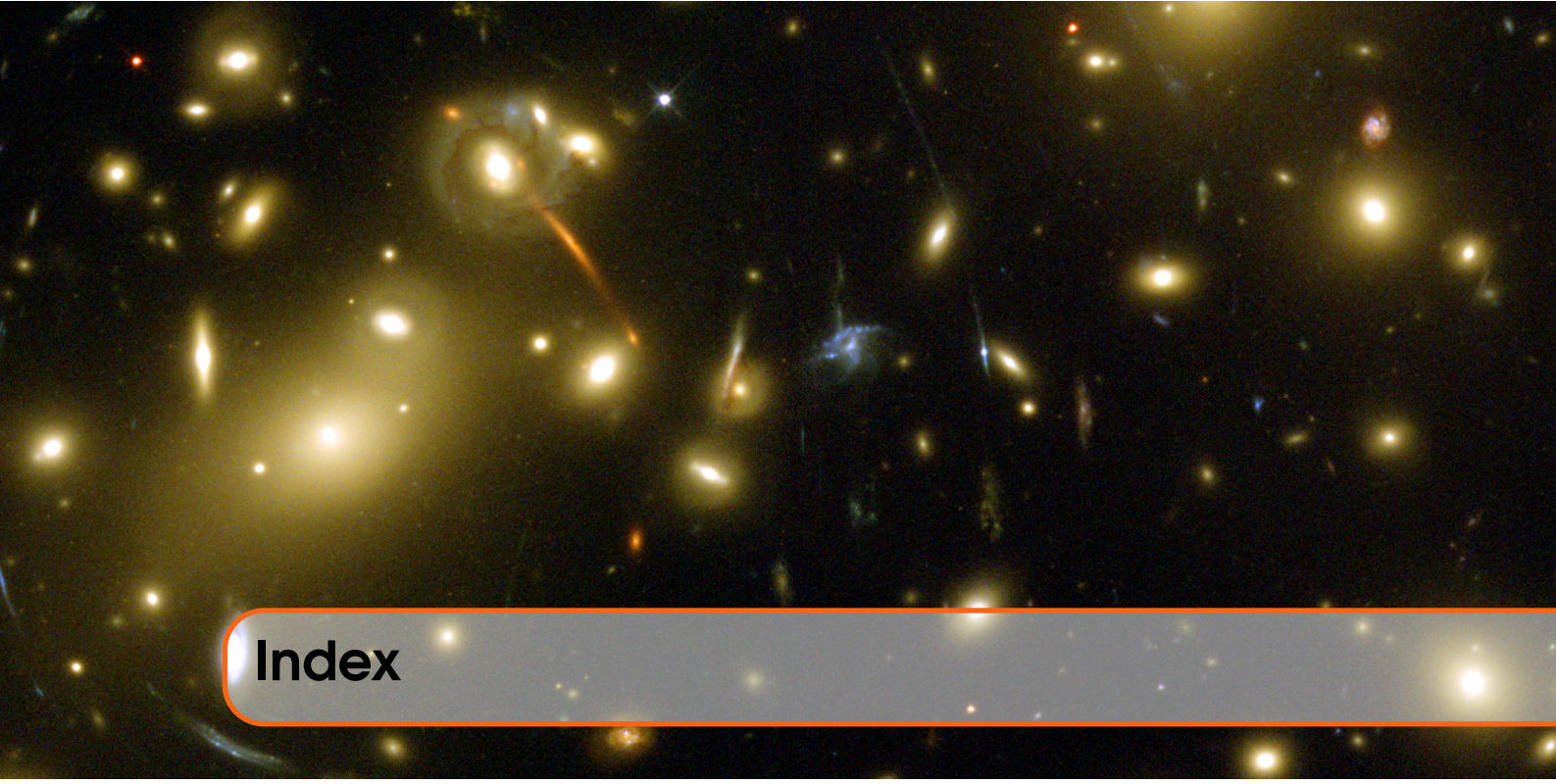
Cooley, James W and John W Tukey (1965). "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of computation* 19(90), pages 297–301 (cited on page 17).

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 15).

Hogg, D. W. (1999). "Distance measures in cosmology". In: *ArXiv Astrophysics e-prints*. eprint: astro-ph/9905116 (cited on page 27).

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 16).

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 14).



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.