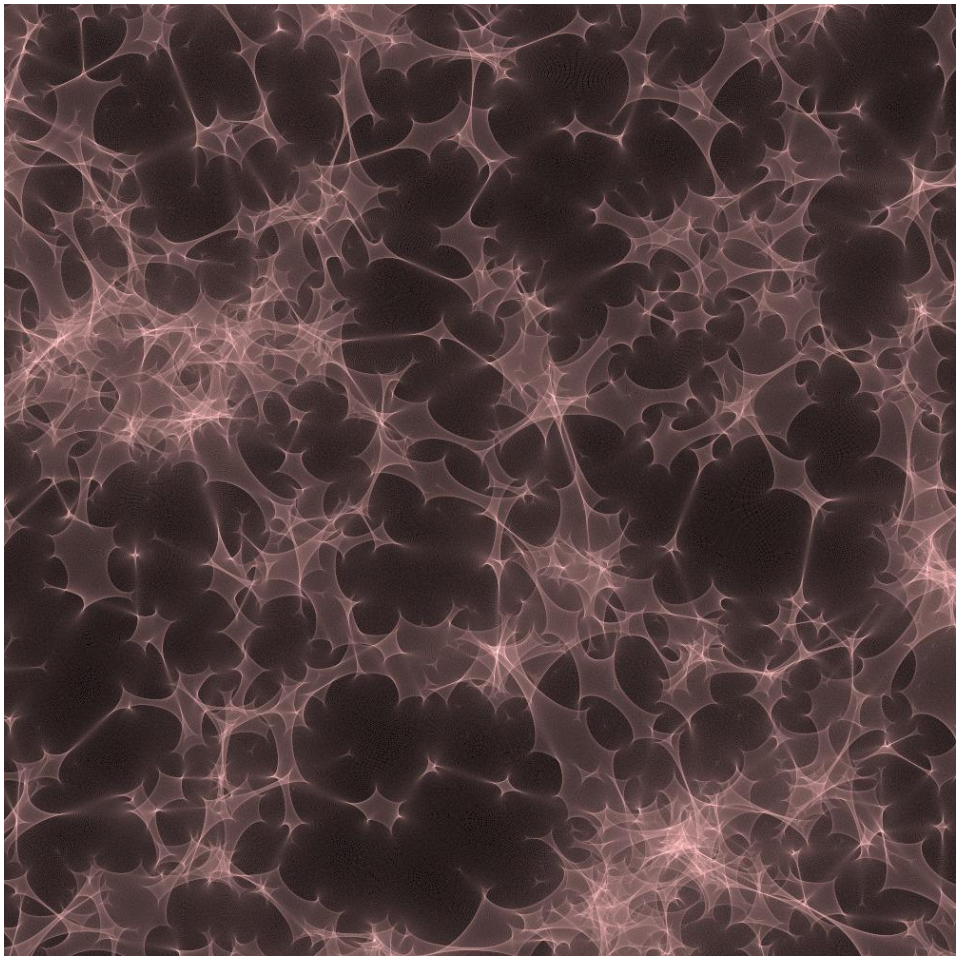
ugr Universidad de Granada

Inverse Ray Shooting Tutorial

Jorge Jiménez Vicente
Dpto. Física Teórica y del Cosmos
Universidad de Granada
Spain

ugr Universidad de Granada

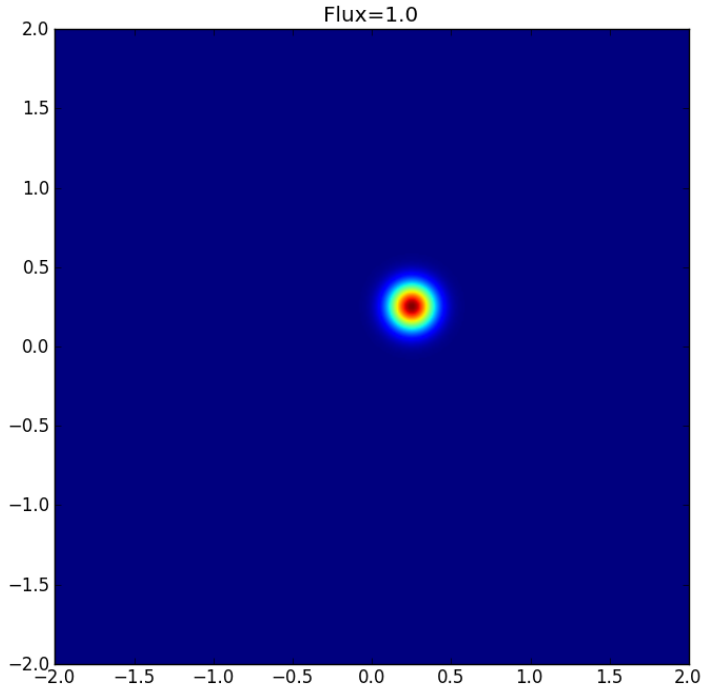
Final goal

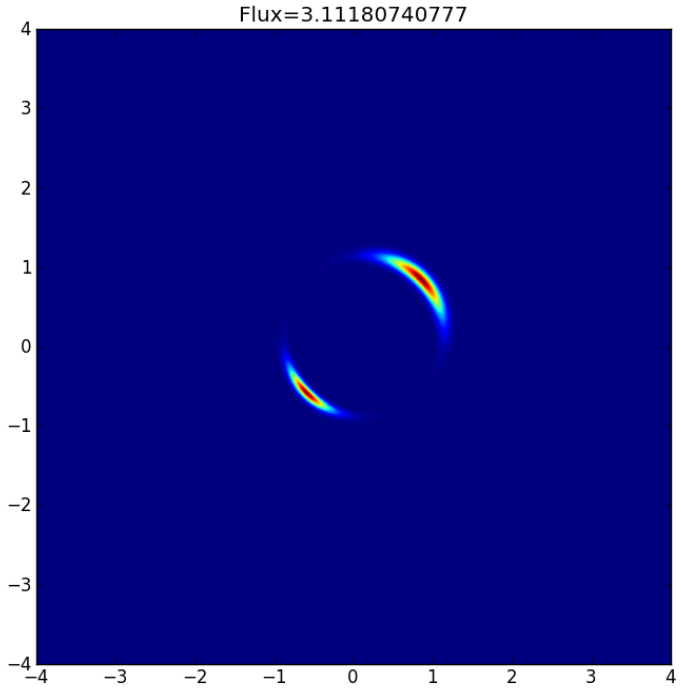


Session I

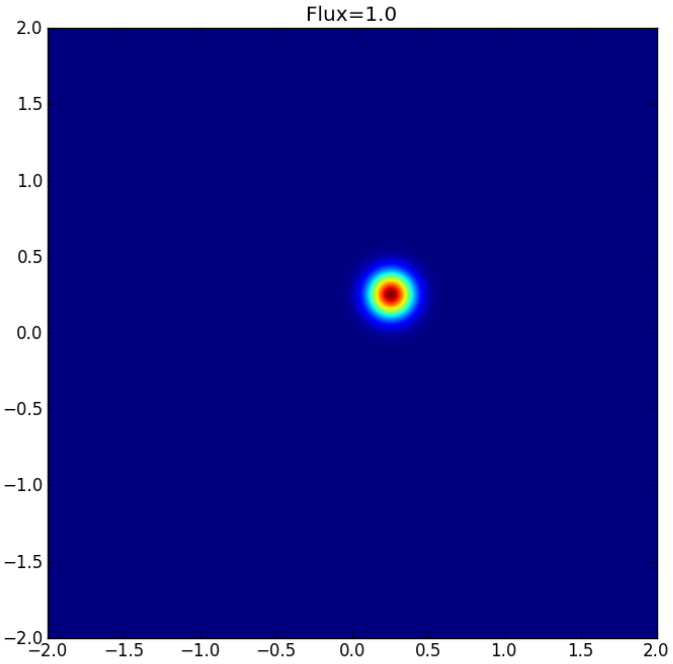
- Introduction to Python
- Solving the lens equation
  - Ray shooting basics
- Image/s of a simple lens.
- (Playing around with lenses and sources)

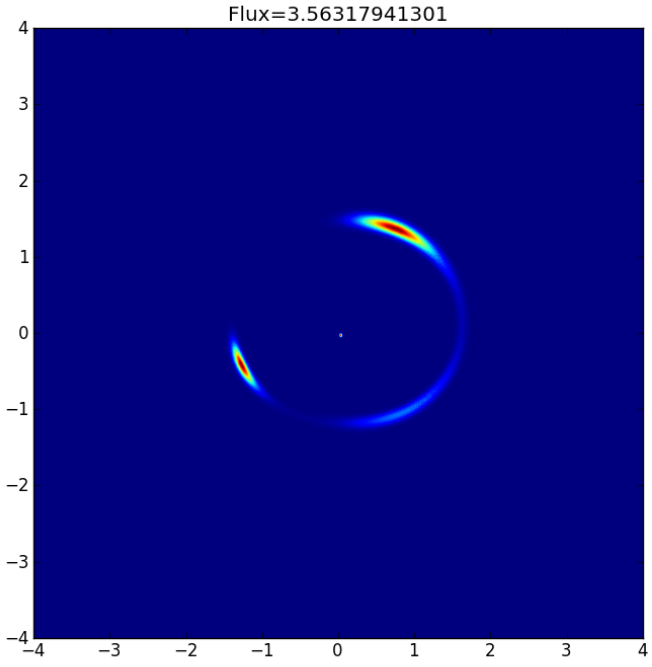
Today's goal #1



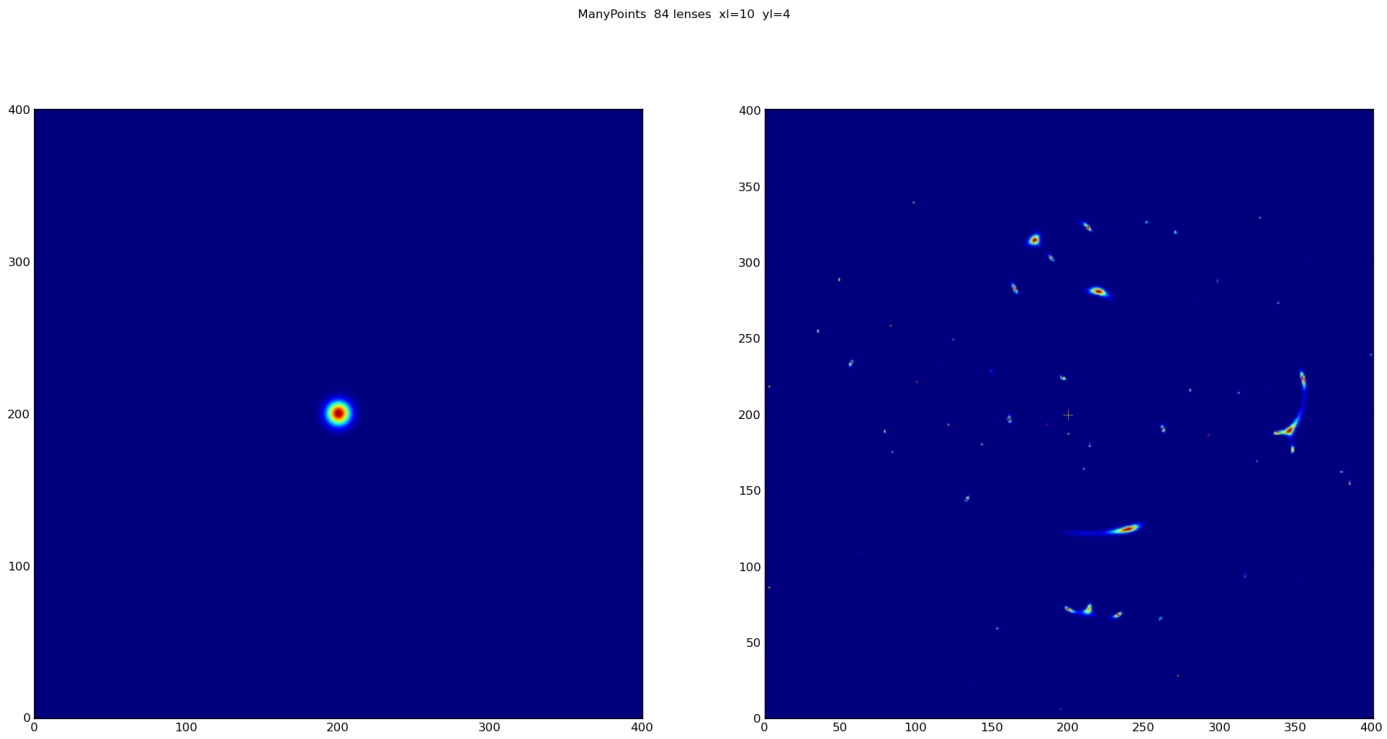


Today's goal #2





Today's goal #3



Python

- We will use python for this tutorial.
- Why?
  - It is free !!
  - It is easy to read/write code !! (dynamic types, interpreted, …)
  - It is versatile/powerful (with some libraries)
  - It is becoming widely used!!
  - It is funny (Got it's name from Monty Python's!!)
  - I have chosen it !!!

Python data types

- a = 1 ← Int
- a = 1.3 ← Float
- a = “Hallo!” ← String
- a = [1,3,4,”Hallo”] ← List
  - a[0] = 7 ← First element
  - a[-1] = “Bye” ← Last element
  - a[1:-1] → [3,4] Slice
- a = [[1,2],[3,4]] ← List of lists (not array!!)
- a = {“mass”:3.2,”vel”:3.2e5,3:4}
  - a[“mass”] = 6.7
- a = (1,3,5,”siesta”) ← Tuple (Non mutable !!!)
  - ~~a[0]=5~~

Python flow control

- Code blocks are indicated in python via indentation level → No braces or alike → Try: from __future__ import braces
- Four spaces per level is customary
- Loops: ← Nested loops can be very inefficient in python !!!

```
for i in range(2,5):
    print i, "bottles of wine are too much"
```

- If …else:

```
if (a[i] == 7):
    print a[i], "is equal to seven"
    a[i] += 1
elif (a[i] == 5):
    print a[i], "is five"
    a[i] -= 1
else:
    print a[i], "is not 5 or 7"
```

- While:

```
i=0
while( i < 10 ):
    print i
    i += 1
```

Read/Write files and command line arguments

- Command line arguments are stored in variable sys.argv from module sys.
- To use them:
  - import sys
  - Use sys.argv
- File I/O
  - Open a file with open for writing, reading or both
    - f=open(filename,'r+')
  - You can read file with f.read(), f.readline() or loop over elements on f
  - You can write to the file with f.write ← Remember to convert to string first with str

Python functions/modules

- Python has many built-in funcions:
  - abs(), help(), raw_input(), etc…
- Modules can be loaded for more fun…ctions:
  - import random → funcs are called via random.funcname → print random.random()
  - import random as r → funcs are called via r.funcname → print r.random()
  - from random import * ← random() ← BEWARE !!
  - from random import random

Python function/modules (II)

- Of course, you can create your own functions/modules
- A function is created like this:

```
def factorial( n ):
    if n <1:    # base case  <- Inline comment
        return 1
    else:
        return n * factorial( n - 1 )
```

- Several functions can be defined on the same file “module.py” an imported as any other module.

Arrays and Numpy

- The Numpy library is very convenient to use/manipulate arrays:
  - It uses arrays in a very easy/intuitive way
  - It uses arrays in a very efficient way ← Speed !!
- Import the module with "import numpy as np"
- Arrays are ordered C-like (last index runs faster → Closer last index are closer in memory) ← **IDL and fortran users beware !!!!**
- We can convert lists and tuples into arrays: a=np.array(a)
- We can create them from scratch as zeros, ones, linspace,etc..
- Particularly useful (for working with 2D images) is mgrid:
  - y,x=np.mgrid[0:3,0:3] →

```
y→ array([[0, 0, 0],
          [1, 1, 1],
          [2, 2, 2]])
x→ array([[0, 1, 2],
          [0, 1, 2],
          [0, 1, 2]])
```

Numpy (II)

- We can operate with every element of the array with a single instruction:
  - c=a+b , c=a*b, c=a/b, c=np.exp(a), etc..
- Numpy tries to “broadcast” smaller arrays to make them fit.
- Arrays multiply on a per element basis. If we want matrix multiplication we need matrix objects instead.← BEWARE
- Matrices and arrays can be converted with the *array* or *matrix* functions.
- We will not use matrices during this tutorial

Plots/Images with matplotlib

- Matplotlib is now the standard plotting library in python.
- It is easy to use but powerful and versatile.
- The most useful sub-module is pyplot. It can be loaded with:

  import matplotlib.pyplot as plt
- Then plotting is as simple as:

  plt.plot(x,y) ← Lots of options, labels, etc..
- We can make subplots with

  plt.subplot(ncol,nrow,active)
- For an image, we display it with:

  plt.imshow(image)

Pyfits

- In the near future we would like to read/write fits files.
- For this tutorial we shall be happy with:

```
import pyfits as pf
a=pf.getdata('filein.fits') ← Read data
pf.writeto('fileout.fits',a) → Write data
```

Zen of python

- Condenses the essence of the guiding principles of python:

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The lens equation

- Using the right units for the angles in the lens and source planes, the lens equation can be written as:

$$\mathbf{y = x - \alpha(x)}$$

Where (y1,y2) and (x1,x2) are the coordinates at the source and lens plane respectively. α is the "scaled/reduced deflection angle".

- This is a highly non-linear equation which can only be solved analyticaly in a few cases.

Inverse Ray Shooting

- A simple way to invert the lens equation (know the position, distortion, magnification, of the image(s) of the source) is by shooting rays backwards from the observer to the lens.
- Light rays are reversible, so we can do it.
- Following the rays backwards we only care about the rays we are really interested in.
- Indeed this “ray tracing” procedure is very common in computer graphics.

Inverse/Backwards Ray Shooting

Idea of “backward ray tracing”:

IRS program

- Divide the image plane into pixels.
- From the center of each pixel (xi1,xi2) shoot a ray backwards.
- Calculate the deflection at the lens (α1, α2)
  - The deflection angles α1, α2 in the horizontal and vertical axis are different for every point at the image plane !!
- Calculate the position at the source plane where the ray hits via the lens equation:

$$yi1=xi1-\alpha1(xi1,xi2)$$

$$yi2=xi2-\alpha2(xi1,xi2)$$

- Assign to the pixel corresponding to coordinates (xi1,xi2) in the image plane the brightness of pixel with coordinates (yi1,yi2) in the source plane.

First contact: No lens to Point lenses

- The “empty” lens.
  - No deflection ➔ y=x ➔ Image plane identical to source plane.
  - It may look stupid but will help to set up the general structure of the program irrespective of complexity in source and/or lens.
- Parameters of the program:
  - Size and number of pixels of the image (xl,nx)
  - Size and number of pixels of the source (yl,ny)
  - Position of the source (ys1,ys2)
  - Position of the lens (x1l, x2l)
- Play around with these parameters to get familiar with them and see their effect on the program.
- The “point source” lens at pos. (x1l,x2l).
  - The deflection angle is (for mass M)
    - α1=M (x1-x1l)/((x1-x1l)^2+(x2-x2l)^2)
    - α2=M (x2-x2l)/((x1-x1l)^2+(x2-x2l)^2)

Sources

- **A circular homogeneous source:**

```
def circ(ny,rad,x1=0.0,y1=0.0):
    x,y=np.mgrid[0:ny,0:ny]
    r2=(x-x1-ny/2)**2+(y-y1-ny/2)**2
    a=(r2<rad**2)
    return a/a.sum()
```

- **A circular gaussian source:**

```
def gcirc(ny,rad,x1=0.0,y1=0.0):
    x,y=np.mgrid[0:ny,0:ny]
    r2=(x-x1-ny/2)**2+(y-y1-ny/2)**2
    a=np.exp(-r2*0.5/rad**2)
    return a/a.sum()
```

Make it faster

- The program irs_0.0.py is the basic IRS program.
  http://www.ugr.es/~jjimenez/IAC-WS2012/S1/irs_0.0.py
- Nested loops in python are not very efficient.
- To make the loop over rays more efficient we should exploit numpy arrays:
  - Use mgrid to generate x and y coordinates for the rays.
  - Operate on entire numpy arrays to deflect rays.
  - Then we loop over the indices that hit the source plane.