Introduction to Gravitational Lensing and QLens

Quinn E. Minor

I have written this tutorial to be readable for someone at the undergraduate or beginning graduate student level, with the goal of helping you get started with gravitational lensing concepts and QLens. If you are at a higher level than this, I hope the following will still be useful in getting you started with lens modeling in QLens; you can just skip the parts that cover theory which is already familiar. This tutorial only covers modeling of point images, although QLens is capable of modeling pixellated images as well—I simply haven't had a chance to include it in the tutorial yet. If there are any sections that are confusing, or if you have questions that aren't covered here, please don't hesitate to email me (qeminor@gmail.com) and let me know.

Contents

1 Getting Started with Basic Lens Models

Before getting started, I highly recommend reading a basic introduction to gravitational lensing that discusses lensing by a point mass. An excellent choice for this is James Hartle's textbook *Gravity*, Chapter 11, which works out the lensing equation for a point mass and solves it to find the image positions. In the following, I will assume you are familiar with the concept of an Einstein ring and Einstein radius, but no other knowledge is assumed.

1.1 Point Mass Lens

The first thing you can do with QLens is to play with a point mass lens (in other words, a lens "galaxy" where all the mass is concentrated at a single point). Since this is usually the first model encountered when being introduced to gravitational lensing, it is a good starting point.

Open up qlens and type "help lens". This will display a list of the available lens models—each model describes a lens with a certain mass density profile. Now type "help lens ptmass". This gives you the format required for creating a point mass lens. The first argument b is the Einstein radius (in arcseconds). The second and third arguments give the coordinates of the point mass (if you omit those arguments, it automatically places the point mass at the origin; this is generally true for arguments listed in square brackets, they default to zero if you don't specify them).

To try it out, type "lens ptmass 5". You can verify that this lens has been created by simply typing "lens". This always lists all the current lens models (if there is more than one lens loaded, they are superimposed on each other). It should list lens 0 as a point mass with $b = 5$.

The lens software will always define a grid where images are to be searched for. If you type "grid", it will show the current grid size (type "help grid" to understand the output). When the grid gets created, QLens is configured to automatically center the grid on the first lens model you enter. In addition, it automatically sets the grid size based on the Einstein radius of the first lens model (more specifically, the radius of the grid is set to $1.9R_e$). To see this, type "mkgrid" and then "grid". You will see the grid size has changed to the more optimal size. If you specify the grid dimensions manually (e.g. "grid -20 20 -20 20"), the automatic grid sizing feature immediately turns off.[1] If your lens is complicated or highly asymmetric, QLens has a more sophisticated way to search for the optimal grid size, using the "autogrid" command. However, in general you shouldn't have to use autogrid in most cases, as the Einstein radius is usually sensible for setting the grid size.

Next, let's look at the critical curves of this lens. Critical curves denote regions where image magnifications become very large, in other words, where images are stretched out the most. Type "plotcrit" and the critical curves and caustics will be found and plotted to the screen (assuming you have gnuplot

[1] You can turn the automatic grid sizing on/off manually with the command "autogrid_from_Re on/off"; likewise, the auto-centering is toggled with "autocenter on/off".

installed properly). We'll talk about caustics later, but in any case, you will always see one plot showing the critical curves and one plot showing the caustics. In this case, the caustic plot will look really weird, because it is basically a single point for a point mass (there is machine error, which gives it the weird appearance, but notice the scale is tiny!). You can close the caustic window, we don't care about that right now.

Note that there is only one critical curve in this case. Compare the radius of the critical curve to the Einstein radius—they should be the same. This makes sense, since images become smeared near the Einstein radius (hence the Einstein ring). For a symmetric lens, there is always one image in between and outside critical curves. Hence, in this case there should always be two images for a point mass—one inside the critical curve, and one outside.

Keep in mind that the scale of each axis in the plot is in terms of angular separation in arcseconds (which are the units b was defined in), rather than a physical distance. This is always the case when making plots in QLens.

Now let's play around a bit. To place a source point, you can use the "findimg" command. Type "help findimg" to get a sense of how to use it. (By the way, you don't have to explicitly use the "mkgrid" command before using these commands; if the grid hasn't been created yet, QLens will automatically do so. The only exception is if you manually change the grid dimensions, in which case you must explicitly remake the grid with "mkgrid".) Next, try "findimg 1 0". It should list two images, both with $y = 0$ or something close to it. One image should be outside the critical curve, and one inside. Note the magnifications—a magnification of 3 indicates that the image will look three times brighter (or, if the image was resolved by a telescope, it would look three times larger in size compared to if there was no lens present).

Now try placing a source point closer to the origin. Do you expect the magnification to get larger or smaller? Will the images be closer or further from the critical curve? (Think in terms of approaching an Einstein ring.) Try it out, e.g. with "findimg 0.1 0" and see if it matches your prediction.

It is nice to be able to plot the images to the screen too. You can do this by typing "plotimg" instead of "findimg" (e.g. "plotimg 0.1 0"). This makes one plot with the images & critical curves and one plot showing the source & caustics. Note that here, the source plot looks weird since the caustic is basically a point—just close that plot.

If you have already studied the analytic solution for the images from a point mass lens (the Hartle textbook I mentioned above is a good choice), it would be a good idea at this point to make sure the image positions match the formula you learned before. To do this, you can use the formula for the image positions to calculate the image positions from a source at beta = 1. (This is equivalent to "findimg 1 0" where the r-direction points along the x-axis in this case.) Do the image positions match the x-coordinates you find from the findimg command? Make sure they do before you proceed. Unfortunately, it's pretty rare that lens models have a simple analytic solution like the point mass lens does, except in very special cases. This is why we use lens software like QLens to solve the lens equation numerically.

Finally, to move on to the next part, you can clear the lens model to start over by typing 'lens clear'. (You can verify that it's cleared by typing 'lens'; it should say there are no lens models specified.)

1.2 Power-Law Lens Model

Obviously real galaxies are not concentrated at a single point (if they were, we would all be one-dimensional!). So for a more accurate gravitational lens, we should try a model that is extended in space. The important quantity to describe such a lens is the 2-dimensional mass density of the lens (also called the projected mass density), denoted by the function $\kappa(r)$ where r is the angular separation from the center of the lens. (In fact, κ is dimensionless, because it is defined as the mass density divided by the average density within the Einstein radius.) One particularly simple example is the power-law model,

$$\kappa = \frac{k}{r^\alpha} \tag{1}$$

where k is a constant that determines the mass scale of the lens, and α can be any real number. The greater we make α, the steeper the slope of the density profile becomes and the more centrally concentrated the lens is. Of course, most galaxies do not look perfectly circular; most have some ellipticity to them. This is incorporated by making the replacement $r \rightarrow \xi = \sqrt{x^2 + y^2/q^2}$, where q is called the axis ratio. With this replacement, the mass contours of the kappa profile are ellipses; an axis ratio $q = 0.5$ means the minor axis is half as long as the major axis. Also, in this example the density becomes infinite at the galaxy center, whereas a realistic galaxy may have a constant-density core in the middle. This is incorporated in the following profile, called the "softened power-law model":

$$\kappa = \frac{k}{(\xi^2 + s^2)^{\alpha/2}} \tag{2}$$

where s gives the core size. Note that $s = 0$ and $q = 1$ gives the power-law model quoted above. In QLens, this profile is actually re-written as follows:

$$\kappa = \frac{2-\alpha}{2} \cdot \frac{b^\alpha}{(q\xi^2 + s^2)^{\alpha/2}} \tag{3}$$

We call this the "alpha model". While it might look unnecessarily complicated, the profile is written this way so that b is a good approximation to the average Einstein radius (and b is exactly equal to the Einstein radius if $q = 0$, $s = 0$). In most applications, we set $\alpha = 1$, which is called the "isothermal" model and usually provides a good fit for galaxy-scale lenses.

Type "help lens alpha" to see how the alpha model is created. Create a model with $b = 10$, $\alpha = 1$, $s = 0.1$ and $q = 1$. Plot the critical curves and caustics. Note that this time there are two critical curves—the inner critical curve is created by the presence of the small core. The outer curve is called the tangential critical curve because it stretches images tangentially (think Einstein ring), whereas the

inner curve is called the radial critical curve because is stretches images along the radial direction. How many images do you expect to be produced based on the number of critical curves?

To test your prediction, place a source at (3,0) and find/plot the images. Make sure the number of images matches what you expect. Now try placing a source at (8.5,0) and note the number of images. What happened? Notice the radius of the caustic curve. The caustic curves enclose regions with differing numbers of images produced: in this case, if a source lies inside the caustic, three images are produced, whereas if it lies outside the caustic, only one image is produced. Physically, this makes sense because if the source is too far away, the lensing is weak enough that multiple images are not produced. Finally, note the magnifications of the images for a source inside the caustic. You should see that the central image is *very* demagnified; in fact, it appears so small that is almost never seen in any lens (with a few notable exceptions). For this reason, this is normally called a "double" image configuration, since only two images are seen.

Finally, try placing a source at (15,0). What happened? Shouldn't one image have been found? The problem is that the image produced lies outside the grid where images are searched for. If you type "grid" it will show the current grid size. Try making the grid size larger, e.g. changing the grid size to (-25,25) x (-25,25). You can do this by typing "grid 25 25" (the long version is "grid -25 25 -25 25"). You will need to remake the grid explicitly by typing "mkgrid", and then try placing the source at (15,0) again. This time you should find the image shows up. Since we're normally interested in multiple imaging (or "strong lensing" we don't care about the single images, and making the grid this large means poor grid resolution over the region where multiple images are produced. So this is usually not necessary, but occasionally if you're missing an image you may want to try making the grid larger to see if it shows up.

Next, things get more interesting if the lens has some ellipticity. Clear the lens configuration and create an isothermal model with $b = 10$, $s = 0.1$ and $q = 0.7$. (Remember, isothermal means $\alpha = 1$.) Note the difference in the number of caustics this time. The inner caustic is called the "astroid caustic" Following the pattern, how many images do you expect to find if you place a source inside the astroid caustic? (Hint: the total number of images is always odd, as long as the lens model is reasonably smooth.) To test it out, place a source at (0.5,0). Plot the images so you can see this clearly. Again, the central image is very demagnified, so this is called a "quad configuration" after the four images that are visible. Note that since the lens is no longer symmetric, now there can be more than one image between and outside each critical curve. You should verify that if you place a source between the two caustics, you will get a double image configuration. However, the two images no longer align with the center of the lens galaxy (along the same line), as they did for a spherical lens. This is another indicator that we have an asymmetric lens.

What happens when a source is near the edge of a caustic? To check this out, place a source point at (0.6,0.7) and plot the images. This source is just inside the astroid caustic, so we get a quad configuration. Notice, though, that

two of the images are close together and straddle the tangential critical curve. Note that the magnification of these images is quite large. What will happen as the source crosses the caustic? Try placing a source point at (0.68,0.7). Note the two image points are now quite close together, and their magnifications are large and nearly equal in magnitude. Finally place a point at (0.7,0.7). You can see that as the source moves outward, the images come closer together and when the source crosses the caustic, they merge and disappear. Now we are left with a double image configuration.

One takeaway point from this is that when images occur close together, they must be straddling a critical curve and their magnifications will be large. Unless the grid has good enough resolution, QLens won't be able to distinguish the two images. To deal with this, for any grid cell that contains a critical curve, QLens splits it into smaller cells (each cell splits into a 4x4 subgrid). Type "plotgrid" to see how the grid looks. Note that any cell containing a critical curve (along with its neighbors, for the first splitting) has been split twice. This allows close images to be found. However, if a source point was really close to a caustic, you might have to split the cells further to resolve the images. In QLens, all the grid parameters can be adjusted—the number of splittings, the number of cells in the radial and tangential directions, the minimum cell size, etc. These are more advanced features and you probably won't have to mess with them, but you can type "settings" or "help settings" to see the various settings you can play with.

One last point to make about the grid. Try placing a source at (0,0). You'll see a quad configuration, but what happened to the central image? The default grid is called a radial grid, meaning the grid cells are spaced along the radial and tangential directions. However, each cell is a quadralateral, which means the innermost cells don't go all the way to the origin; there's a small area around the origin that's not covered by the grid. Again, you will never have to care about that, because the central image is not observed. On a side note however, you can also switch to a Cartesian grid. Type "gridtype cartesian" and then "mkgrid". Then if you put a source at (0,0), it will find all the images. Type "plotgrid" and you will see what the Cartesian grid looks like. Generally however, as long as the lens is roughly elliptical in shape, the radial grid is better to use.

1.3 Adding External Shear

Having an elliptical mass distribution is not the only way to get an asymmetric lens configuration. Even if a lensing galaxy is very spherical in shape, there are often nearby galaxies that belong to the same group or cluster as the primary lens. Those nearby galaxies can perturb images through their gravity, warping their observed shapes—this warping is called shear (in particular, we call this *external* shear since it is not caused by the primary lens galaxy). In fact, galaxies don't even need to be close to the primary lens to add external shear—they simply have to be somewhere along our line-of-sight to the lens. As long as the galaxies perturbing the lens are not too terribly close, external shear can be modeled simply as a quadrupole term in the gravitational potential (if you have

taken upper level electromagnetism, you may recall that a charge distribution can be expressed as a multipole expansion; the quadrupole term is the next term after the dipole). This external shear can be added to any given lens model in QLens.

To see how this works, let's clear the lens configuration and create an isothermal model with $b = 10$, $s = 0.3$, and $q = 1$. Note that since the model is symmetric, there is only one caustic curve and the critical curves are circles. Now, let's add some external shear. Type "help lens shear" to get an idea of the format. Gamma gives the magnitude of shear, and generally is between 0 and 0.3 for realistic lens models. Type "lens shear 0.1" to add external shear. If you type "lens" you will see that the isothermal lens is listed as lens 0, and the shear is listed as lens 1. (In general you can add as many additional lenses as you like, but for lens modeling we will stick to at most two or three lenses.) Now plot the critical curves. You can see that the shear has disturbed the symmetry of the lens enough that now an astroid caustic has appeared, and the critical curves are now elliptical rather than circular. Thus, shear by nearby galaxies can also create quad image configurations. Some amount of external shear is nearly always present in a realistic lens, so it will have to be included in lens modeling.

Of course, the direction of the shear usually won't coincide perfectly with the x/y directions of a camera image—it will usually be at some angle. To simulate this, delete the present shear model by typing "lens clear 1"—this removes lens 1. Now type "lens shear 0.1 30". This creates external shear at an angle of 30 degrees to the x-axis. Plot the critical curves and you will see they are inclined at a 30-degree angle, and so are the caustics (of course, the gnuplot graph often doesn't show the same length scale for the x- and y-axis, so you may have to play with the window size to see the correct angle). Generally speaking, when modeling a lensed image we'll have to constrain the angle of the shear in addition to its magnitude.

Note that we have entered the external shear as a separate lens model. When fitting lens models, however, we only want to model the external shear in the vicinity of the lens galaxy, and thus we want the shear's center point to always coincide with that of the lens galaxy. In other words, we want the shear term to be "anchored" to the lens, so that their centers always coincide. There are a couple ways to accomplish this; the easiest way is to re-enter the isothermal lens model as follows:

lens alpha 10 1 0.3 1 0 0 0 shear=0.1 30

After entering the above, now type "lens". Note that the external shear is still listed as a separate lens model, but after printing the parameters it says "(anchored to lens 0)". This tells you that the two lenses are fixed together, so if you move the lens galaxy around, the shear will automatically go with it. You can add external shear to any lens model using this format. (For the sake of completeness: the more general way to anchor lenses is using the "fit anchor" command; this can anchor any lens to any other lens, but usually the external

shear is the only model that needs to be anchored like this.)

The core size of a lensing galaxy is often very difficult to constrain from gravitational lens data. As a result, the most common lens model simply fixes the core size to zero—this is called the "singular isothermal sphere" model. To see this, clear the lens model and type "lens alpha 10 1 0 1" and plot the critical curves. Note the caustic graph looks weird because the caustic is basically a point, rather than a curve—as a result, multiple imaging occurs even if a source is far away. This is because the lens is singular at the origin, similar to a point mass (surely an idealization, but often a decent enough approximation). Just as with a point mass, the lens has only one critical curve. Next, let's add shear with magnitude 0.1. Note that the caustic point has now opened up into an astroid caustic curve! A singular isothermal sphere plus external shear is one of the most common lens models used for modeling gravitational lenses—it is simple enough that usually all the parameters can be constrained. Whether the model is *good* enough depends on the situation. For example, often the lens galaxy is elliptical enough that modeling it as a sphere ($q = 1$) is simply not accurate enough. In that case, one may be able to constrain the axis ratio of the lens galaxy in addition to the shear and other parameters. This is called the "singular isothermal ellipsoid + external shear" model.

2 Redshifts and Cosmology in QLens

2.1 Redshifts and Physical Units for Lens Properties

As we have mentioned, QLens works mainly with dimensionless quantities to describe a lens. The lens is described by the scaled projected mass density κ, which is dimensionless; likewise, positions in the lens and source planes are described in terms of angular separations (in arcseconds), rather than physical distances. As long as we are not interested in physical units (mass, distance etc.), knowing the distance to the lens or source objects is not required.

However, there are many instances in which we *are* interested in physical units—after all, we are usually trying to better understand the physical properties of the lens and/or the sources. For example, we might want to know the actual projected mass within the Einstein radius, or perhaps we are interested in the physical size of the Einstein radius in the lens plane. Or we might be interested in modeling the observed time delay between different images, which obviously should have units of time (QLens uses days for time delay units). In order to calculate these things, we must provide information about the distance to the lens and source objects.

For most extragalactic objects, we cannot measure their distances directly; what we can measure is the redshift of the object's light due to the expansion of the Universe. Translating the redshift into a distance requires assumptions about the cosmology describing the Universe, and in technical terms this is done by solving the Friedmann equations (see Barbara Ryden's excellent undergraduate textbook "Introduction to Cosmology" to learn more about this). In QLens

we assume a flat Universe with the so-called concordance ΛCDM cosmology, with default values for the Hubble parameter H_0 describing the current expansion rate of the universe, and the present-day matter density Ω_m, although these can be changed by the user if desired.[2]

To enter in the redshift of the lens, use "zlens = #", and likewise we enter the source redshift using "zsrc = #". (By default, QLens assumes $zlens = 0.5$ and $zsrc = 2$.) Once you have entered the redshifts, try creating a lens using the "alpha" model like we did in section 1.2. Now type the command "einstein 0", which will calculate the Einstein radius of lens 0. Note that the Einstein radius is given in arcseconds, but also in kpc (kiloparsecs), which is a physical distance in the lens plane. Also shown is the projected mass contained within the Einstein radius in units of solar masses. Now try changing *zlens* or *zsrc*, and enter "einstein 0" again. You should find that the physical quantities have now changed, since they depend on the assumed distance to the lens and source, while the angular units remain the same, since these are directly observed. You might also be interested in plotting the mass contained within radii other than the Einstein radius, which can be done using the "plotmass" command.

If you are interested in finding the time delays between images, use the command "time_delays on". Now if you plot images, QLens will display the time delay between images (in units of days) in addition to the positions and magnifications. Again, this is dependent on the distances, so changing the redshifts will scale the observed time delays.

2.2 Dealing with Multiple Source Points at Different Red Shifts

If you are only going to be modeling lenses with a single source object in the forseeable future (which is typical for lensed quasars), you may want to skip this section for now and move on. If you plan to model cluster lenses, however, I recommend reading this section carefully, since more than one lensed source is very common for cluster lenses.

There may come a time when you are modeling a lens with more than one source object, each of which is at a *different* redshift (a common situation for galaxy cluster lenses). This can be a bit tricky, because in QLens, the dimensionless mass density κ of each lens (defined by the parameters we enter) is already scaled in terms of the average density within the Einstein radius—but the Einstein radius depends on the redshift of a particular source! So if we want to simulate sources at different redshifts, which source redshift do we use to define our kappa for the lens? You cannot simply start with one *zsrc* value, define your kappa, and then change it to another, because this would actually *change* the mass of the lens. It's a subtle point, so let me put this another way. Having

[2] By default the Hubble constant is assumed to be $H_0 = 70$ km/s/Mpc, and $\Omega_m = 0.3$. To change these values, use "h0 = #" and "omega_m = #". In QLens, "h0" is actually given by h where the Hubble constant $H_0 = 100h$ km/s/Mpc. Thus by default QLens uses $h = 0.7$. Incidentally, if one is modeling time delays, it is possible to vary the Hubble parameter and try to constrain its value, but this feature is beyond the scope of this tutorial.

more than one source redshift wouldn't be a problem if we defined our lens in terms of its actual, *physical* mass and size, since these are totally independent of redshift. But since in QLens we actually define each lens by redshift-scaled quantities (in order to get dimensionless units), we need to be careful when we are lensing multiple sources at different redshifts.

When you are modeling image data (which we cover in Section 5), this is actually easy to deal with, because the redshift for each source can be specified explicitly in the data file, and these are independent of *zrsc*. The scaled lens quantities for each lens (κ, etc.) are (by default) still defined by *zsrc*, so there is no confusion. However, you might want to be able to simulate different source redshifts for a given lens—this is useful, for example, when plotting critical curves for different redshifts, or for generating simulated data. How can this be done?

In QLens, there is actually another redshift you can enter called *zrc_ref*, which we call the *reference* source redshift. This is actually the redshift QLens uses to define the dimensionless scaled lens quantities (κ, gravitational potential, etc.). By default, this is automatically set to be equal to whatever value you enter for *zsrc*. However, if you change *zsrc_ref* explicitly before defining your lens, it can now be different from *zsrc* (another way to do this is with the command "auto_zsrc_scaling off"). Now you will be able to change *zsrc* and find the resulting image positions, without changing the lens in any way. Keep in mind, however, that when you plot the critical curves, these will be defined according to the reference source redshift *zsrc_ref*, while the image positions you generate will be determined by the source redshift *zsrc*.

3 Reading Input Scripts

When you get into lens modeling, you won't want to do everything on the command line—it will be much easier to write scripts that can be read by QLens. If you save the commands to a text file, for example *script.in*, then you can run QLens by typing "qlens script.in" and it will execute all the commands in the file. (You can also type "qlens -q script.in" if you want QLens to quit immediately after running the input script.) Another way to run an input script is to simply open QLens and type "read script.in" (or whatever you call the file). You can also place comments in the input file by prefacing them with "#"—comments can be placed after commands or on separate lines. The demo files *demo1.in* and *demo2.in* are included with QLens, so you can try this out by loading those input scripts.

By the way, if you have typed a bunch of commands at the command line and would like to save them to a script so you can run them later, you can do so by entering the command "write <filename>". This saves the entire command history to a text file.

4 Plotting in PDF or Postscript Format

It is possible to plot to PDF or postscript files from within QLens. From QLens, if you type "terminal" (or simply "term"), it should say "Terminal: text". This means that if you plot to files, it will simply make text files. However, you can change this to PDF by typing "term pdf". Now if you plot to a file, e.g. "plotimg images.pdf" or "plotimg source.pdf images.pdf", it will make a PDF file. All the other plotting commands behave similarly. If you get an error from gnuplot, it may not be properly configured for making PDF plots–in that case, you can try to make postscript files instead by typing "term ps".

There are a few plotting options you can play with. For example, the command "plot_title <title>" will set the plot title. You can also change the point size and type using the "ptsize" or "pttype" commands (shorthand for these commands: "ps" or "pt"). If you don't want to show the critical curves, you can type "show_cc off"; if you only want to plot the image plane, type "plot_srcplane off". You can play with the font size using the "fontsize" command. It is also possible to set a specific plotting range for the x/y coordinates; type "help plotimg" for information on how to do this.

5 Lens Modeling

5.1 Chi-square optimization

Before proceeding to lens modeling, a good introduction to how lens model fitting is done for point-images is in Section B.4.6 of the *Saas Fee Lectures on Gravitational Lensing* by Chris Kochanek, available at the following link:

http://arxiv.org/abs/astro-ph/0407232

I would recommend reading through this section carefully before moving on to lens modeling. If you are unfamiliar with the concept of a chi-square test, a good introduction is given in "An Introduction to Error Analysis" by John Taylor (chapter 12), which is an excellent undergraduate-level textbook. We will also describe a bit about the chi-square function at the beginning of Section 5.3.2 of this tutorial.

A sample lens modeling script is included with QLens in the file *alphafit.in*. In this script we find a best-fit model to simulated data using chi-square optimization (i.e. we try to minimize the chi-square function). Read through the comments in this script to get a feel for how the fitting is done. Image data is loaded using the 'imgdata' commands, whereas fitting to this data is handled with the 'fit' commands. You can type 'help imgdata' and 'help fit' to learn about all of these commands. The simulated image data used in this script is given in the file *alphafit.dat*. Look at this file to get a feel for the format required for the image data.

To vary the lens model parameters, enter a lens model as you normally would,

except with 'fit lens' rather than just 'lens'. On the next line you must specify which parameters to vary, entering either 0 or 1 for each parameter (0 means fix the parameter, 1 means vary it). When the fit is started, the parameters you entered will be taken as the initial point in parameter space. For certain fit methods (MCMC or nested sampling) it will also be required to enter prior ranges for each parameter, but for the default fit method (the downhill simplex method, called simply "simplex" in QLens) , which minimizes the chi-square function) this is not required. The initial guess for the source point is specified using the command "fit sourcept".

Particularly if you are trying to minimize the chi-square function, a lot depends on your initial guess for the parameters. To get a sense of this, go ahead and run the script; I have placed a "pause" command before the fit procedure begins. When you get the command line, type "fit" and it will tell you the current lens parameters, along with which parameters are being varied, and the initial parameters for the source point. To see if you are in the right parameter region, type "fit plotimg". This will plot the data points, along with the image points resulting from the current lens and source parameters. [3] If the data and image look dramatically different, you may want to play around with the parameters before starting the fit (of course, you can always try the fit first and see what happens—in this script, I picked the initial parameters sufficiently close so the fit should work out of the box).

You can change the source point using the "fit sourcept" command. To change the lens parameters, you could of course delete a lens and create it again (or just modify and re-run the script), but during this exploratory phase it is usually easier to use the "lens update" command. Suppose you want to change the "alpha" lens. If you type "lens update 0 ...", it will update all the parameters for lens 0 to whatever values you type in. (Note that the external shear is listed as lens 1, so to change the shear parameters do "lens update 1 ..."). Then you can type "fit plotimg" and see if you got an improvement.

A few things to look out for: if two images in the data are close together, then you know those images should straddle a critical curve. So you will have a better chance of finding a good fit if you can get the critical curve to run between the two data images. Often you can accomplish this by rotating the lens, translating its center coordinates, or changing the Einstein radius (the b parameter). Also, to get two images close together, your source point should be near one of the edges (the "fold") of the astroid caustic. Whereas if three images are close together, your source should be near one of the corners (the "cusp") of the caustic. Changing the external shear is similar to changing the ellipticity of the lens, but it does behave a bit differently: in particular, a large amount of shear tends to push images away from the critical curve. The only way to develop an intuition for this is to take the time to play around with it.

When you are ready, type 'c' to continue and run the optimization. If you want to see how the fit looks, you must type "fit use_bestfit" to adopt the best-

[3] If you are modeling more than one source point, you can type "fit plotimg src=#" to compare the images and data for a specific source point (using the number for the image set listed by the "imgdata" command).

fit parameters (this is already done in the script). You can try "fit plotimg" again to see if the resulting images are close to where they should be. If it looks reasonably close, type 'c' again and it will do the image plane chi-square fit, which takes longer. The resulting images should fit the data very closely if all goes well. If you are happy with the fit, you can type "fit save_bestfit" to save the fit information to a file, whose name has the format "<label>.bestfit". A few additional files will be generated as well, whose purpose is not important here but will be described in Section5.3.5.

There are two different chi-square functions you can choose from to model the image positions (both described in the Kochanek reference above): the source plane chi-square, which is fast and well-behaved but does not accurately reflect the errors in the image positions; and the image plane chi-square, which is slow but accurate. (To turn on the image plane chi-square, use "imgplane_chisq on".) The source plane chi-square also has the advantage that the best-fit source position can be solved for analytically while calculating the chi-square, so the coordinates for the source point(s) are not varied as free parameters, making the optimization even faster.[4] Initially you may have better success using the source-plane chi-square to narrow down the viable parameter space, but it is usually best to use the image-plane chi-square for your final fit, since it is more accurate and reproduces the parameter errors consistently.[5]

Now that you have a feel for the fitting procedure, it is a good idea to try it out on some simulated data (although you may want to read through the next section first for additional tips on optimization). For practice, I have included a series of four files that each contain image data for a simulated quad lens. (They are called *testdata0.dat*, *testdata1.dat*, etc., and for each data file there is a file with an "answer key" to check your fit afterword, labeled *testdata*_key.dat*. The first two datafiles contain no external shear, while the others do.) Using the script (*alphafit.in*) as your guide, try to make your own script and fit to this data to recover the correct lens model. You will want to make educated guesses about the lens parameters, as described above (you can use the comments in the example script as your guide). It may take a fair amount of trial and error the first time around.

[4] The disadvantage of solving for the source position(s) analytically is that you don't get an estimate of the uncertainty in the source position, nor can you investigate possible degeneracies between the source position and other parameters. It is possible to vary the source position as a free parameter, even if you are using the source plane chi-square; to do this, enter "analytic_bestfit_src off".

[5] It is possible to make the source plane chi-square more accurate by using magnification information, as described in the Kochanek reference above. This is done using the command "chisqmag on". Be warned that this form of the chi-square is only accurate when we are already close to the best-fit point, so it is usually better not to use it for the initial fit. By default, however, QLens switches to this mode during repeats (although you can disable this with "chisqmag_on_repeats off", but I don't recommend it). Keep in mind that these settings are only relevant if the image plane chi-square is turned off.

5.2 Downhill Simplex Method: Important Details and Tips

The default algorithm used for optimizing the chi-square is called the downhill simplex method (also called the Nelder-Mead method). It starts by defining a simplex in the parameter space—which is a higher-dimensional analogue of a triangle—and, at each step, tries to move the vertex which has the highest chi-square value. A few other actions can occur, for example, shrinking or expanding the simplex along one or more directions (like an amoeba) depending on how rapidly the chi-square is changing in the region the simplex is in. The simplex stops when all the vertices return a chi-square value that are similar to within a specified tolerance set by the variable *chisqtol*.

However, just because the simplex stops doesn't *always* mean it has found a true local minimum of the chi-square function. Sometimes it is only a minimum along a few directions, or perhaps it's in a narrow valley or an inflection point. For this reason, after simplex finishes, it is always a good idea to minimize again, using the previous best-fit point, to make sure it returns the same point. You can do this manually ("fit use_bestfit" followed by "fit run"), but a better way is to use the *nrepeat* command. If you set "nrepeat 1", it will automatically run the optimizer again once. By default *nrepeat* is set to 1, although I think two is probably even better (in any case, I would not recommend setting it to zero). if it converges to a different point the second time around, I would run the optimizer repeatedly to see if it converges to a good chi-square value. This situation may occur if there is degeneracy among the parameters, i.e. no unique solution, so if this happens, it is a good idea to try a few different starting points to see if you arrive at the same best-fit point each time.

What sets the initial size of the simplex? For each parameter, QLens has a default initial "stepsize" which sets the (initial) length of one of the edges of the simplex. However, the stepsizes can be customized as well using the "fit stepsizes" command. Type "help fit stepsizes" to read how this command works. Likewise, you can restrict the allowed range of any given parameter using "fit plimits"; QLens achieves this by returning a large "penalty" value for the chi-square if the simplex strays outside the parameter limits you have defined.

In addition to the downhill simplex algorithm, QLens includes an additional method for minimizing the chi-square function known as Powell's method. In most applications, the downhill simplex method is more robust and converges to a minimum more quickly than Powell's method does, which is why it is set as the default fit method. Nevertheless, there are certain situations where Powell's method performs better, and converges quickly if one is already close to a minimum. It may be worth trying Powell's method if the downhill simplex method fails and the starting chi-square value is not too high.

5.3 Simulated Annealing and Monte Carlo Methods

Once you have gotten the hang of chi-square optimization, you will want to explore ways of boosting your chances of finding a best-fit model. In addition, in many cases there may not be a unique best-fit point; there might be a range

of possible solutions (particularly if you have chosen to vary too many parameters!). For these cases we need algorithms that can better explore the parameter space.

QLens has different methods for fitting data that broadly fall into two categories: 1) chi-square optimizers, which includes the downhill simplex method (the default) and Powell's method; and 2) Monte Carlo methods, which includes nested sampling and the T-Walk algorithm. Finally, there is a hybrid approach called "simulated annealing" which can be added to the downhill simplex method. The key idea behind Monte Carlo methods is to add a "random walk" element to explore the parameter space.

5.3.1 Simulated Annealing

We will begin with a description of simulated annealing. One of the disadvantages of chi-square optimization is that often the chi-square function has several local minima that do not constitute a good fit—for example, there may be a parameter region where the positions of two of the images are well-fit, but the others are not; or the image positions may be well-fit, but the fluxes or time delays (if being modeled) are not. Often these local "traps" are easy for the minimizer to fall into, whereas the true best-fit region is a needle in a haystack. The problem is that by always stepping downhill, we will never find the global minimum if there is a shallower local minimum nearer to the starting point.

To overcome this issue, we should allow some probability of moving uphill rather than downhill. In simulated annealing, this is done by adding a random value to the chi-square at each vertex of the simplex, and at each subsequent new vertex. This allows for the possibility of stepping uphill (because we are fooling the simplex into thinking it is downhill!). The magnitude of the random term added to the chi-square is determined by the *temperature*; the higher the temperature, the greater variation we are introducing to the apparent chi-square values and hence greater leaps will be taken around the parameter space. This allows us to explore a larger search area compared to simply stepping downhill every time.

Of course, if we stay at a high temperature, the simplex will keep taking big leaps and will never settle to a minimum. Thus, the best approach is to "cool" the temperature after a certain number of iterations; this is done by multiplying the temperature by a constant factor, called the *cooling factor* (a typical value would be 0.9 or 0.95), until reaching some specified final temperature. Every time the temperature is reduced, the simplex returns to the best-fit point attained at the previous temperature setting and starts again. After the final temperature is reached, a final run is performed with a temperature of zero, which is identical to simply minimizing, so the simplex settles on a local minimum. If one cools slowly enough, and allows enough iterations at each temperature, the global minimum chi-square value can be attained in principle. Of course, cooling slowly means the algorithm will take a long time to finish, so some balance has to be chosen there.

To see simulated annealing in action, open the script "alphafit_anneal.in"

which also models the same data file as "alphafit.in", but this time starts from a point where simple minimization fails. With the right settings, however, simulated annealing finds the best-fit region eventually. Commands for setting the starting temperature, cooling factor, final temperature, and number of iterations (both during annealing and for the final T=0 run) are given, and tips for choosing their values are given in the comments.

One final tip: you will have greater odds of success using simulated annealing if you can restrict the parameter range to avoid getting stuck in unrealistic regions of parameter space. QLens already disallows unphysical values of certain parameters, for example the axis ratio q (so $q > 1$ or $q \leq 0$ are not allowed for any lens galaxy). However, there may be further restrictions you can impose. For example, if you are varying the slope α of the density profile, you can restrict the range to $[0.5, 2]$ knowing that slopes steeper or shallower than this range are very unlikely (in fact the slope is typically close to 1 in actual lenses). Another example would be to restrict the range of the center of the lens, particularly if you can estimate the position of the lens galaxy center from the observed stellar light profile. To set parameter limits, use "fit plimits" right before running the fit.

5.3.2 Monte Carlo Samplers: Introduction to Bayesian Statistics

If we are interested in more than simply a best-fit point, we need a way to "map" out the parameter space. QLens has two Monte Carlo samplers that accomplish this, nested sampling and T-Walk. First we need a bit of statistical background to understand the general idea.

What are we really doing when we minimize the chi-square function? Typically, random measurement error of some observable (e.g. photon counts, for large numbers of photons) follows a normal (also known as Gaussian) distribution, the ubiquitous "bell curve" in statistics. The likelihood of observing a value x is given by $L(x) \propto \exp(-(x-\bar{x})/2\sigma^2)$, where $\bar{x}$ is the mean value of x observed after many measurements, while σ is the standard deviation in x. (I write $\propto$, meaning "proportional to", because there is also a coefficient in front that I've left out, which normalizes the likelihood so that the total probability of getting any value x is equal to 1; this coefficient is not important here.) When we are modeling the data, however, the expected mean value $\bar{x}$ (and occasionally also σ) usually depends on the parameters of the model we are trying to constrain. Thus, we can write down the likelihood of observing the data x given a particular set of model parameters M, as $L(x|M) \propto \exp(-\chi^2/2)$ where $\chi^2 = (x - \bar{x}(M))/\sigma^2$. The chi-square function χ^2 is therefore given by $\chi^2 = -2 \log L$, that is, twice the (negative) log of the likelihood function.[6]

What we are really doing, then, is to find the model parameter values M that maximize the likelihood function, in other words, maximize the probability

[6] If there is more than one observed data point, x_i, then the likelihood of observing the entire data set is simply the likelihood of observing data point x_1, times the likelihood of observing x_2, times the likelihood of observing x_3, and so on. But multiplying the individual likelihoods is the same as *adding* the exponents, i.e. adding the individual chi-squares together.

of producing the observed data x given our model. The parameter values that maximize the likelihood are what we define as our best-fit point. But this is in fact equivalent to *minimizing* the chi-square function, hence the origin of chi-square (or least-squares) fitting.

However, we may have additional information about the model parameters beyond the data we are analyzing. We might know ahead of time the viable range of some of the parameters, or we might have information from previous observations about the probability of a model parameter taking certain values. Incorporating this *prior information* is the idea behind Bayesian statistics. Instead of focusing on the likelihood of the data D given the model parameters M, which we write as $L(D|M)$, in Bayesian statistics we turn this around and infer the probability of the model parameters M given the data we observe. This is called the *posterior* probability distribution, written as $P(M|D)$. The posterior is related to the likelihood function by Bayes' theorem:

$$P(M|D) \propto L(D|M)P(M) \tag{4}$$

In this equation, $P(M)$ encompasses the prior information on each parameter in the form of a probability distribution. The simplest prior is to have a uniform probability within some range, and zero probability outside that range (called a flat or uniform prior). There are more sophisticated priors one can choose, but the point is that a prior of some kind is required. While it may be disturbing that our prior can influence the inferred best-fit parameters, keep in mind that if there is a lot of data to constrain the model, the prior won't matter so much. The less data we have, the more the prior can influence the inferred probability in each parameter.

In Bayesian statistics, therefore, our goal is usually to map out the posterior probability distribution to constrain the model parameters. Monte Carlo samplers do this by producing a set of points in parameter space (through various methods) whose distribution follows the posterior distribution as closely as possible. If we are interested in the posterior probability of one or two parameters alone, then we must integrate (or *marginalize*) over all values that the other parameters can take. This is easily done by binning the points in the parameters we are interested in, i.e. making a histogram for each parameter. The width of the resulting *marginal* posterior for each parameter then gives us an idea of the uncertainty in our estimated best-fit parameter values.

With some background in hand, let us now look at how these methods are used in QLens.

5.3.3 Nested Sampling and T-Walk

We start with nested sampling. Open the file "alphafit_nest.in" and read through the comments carefully. Procedurally, the biggest difference compared to the chi-square optimizers is that we are required to give a prior range in each parameter we are varying after putting in the lens and source models. By default, QLens assumes a flat prior in each parameter. The initial "guess" values you

put in for the lens and source are not actually used by the nested sampling routine, only the prior range is used here. After you hit "continue" and the nested sampling begins, it will take some time to finish (this occurs when the "test" statistic reaches 2). The details of how nested sampling works is complicated, and not essential to follow at this point; the important point is that nested sampling outputs a series of points that can be binned to approximate the posterior distribution. The number of points that get produced is controlled by *n_mcpoints*, whose default value is 1000. If you increase n_mcpoints, the posterior will be better sampled and you improve your chances of finding peaks in the parameter space. On the other hand, more points means it will take longer to finish. After the nested sampling finishes, the posterior is plotted using the utility *mkdist*, described in the next section.

Running the T-Walk sampler is the same procedure, demonstrated in "alphafit_twalk.in". So how are the two methods different? T-Walk is an example of a Markov Chain Monte Carlo (MCMC) sampler, which advances a number of "chains" of points through parameter space. The convergence criterion for the chains is determined by something called the R-statistic, whose required final value is set by the variable *mcmctol*. The closer mcmctol is set to 1, the longer the chains will run and the more points will be produced by the chains (1.05 or 1.01 are typical tolerance values for the R-statistic). However, as the T-Walk progresses, you can plot the posterior even though the chains have not yet finished. This can be useful, but be careful not to plot the posterior too early on, before the posterior is being well sampled. Typically I don't bother to plot the posteriors before R has gone below 1.1 at the very least.

Nested sampling and T-Walk each have their strengths and drawbacks. Both are good at sampling *multimodal* distributions, where there is more than one separate peak in the posterior. If the posterior is significantly degenerate, i.e. there are long ridges rather than distinct peaks, nested sampling is more adept at sampling such distributions, particularly if the degenerate regions are curved in the parameter space. If you suspect there are strong degeneracies, perhaps because you have varied too many model parameters, nested sampling is probably the better choice. On the other hand, T-Walk can be more adept at locating hard-to-find narrow peaks in the parameter space. It is also more parallelizable, meaning it can be run in parallel with many processors, because a large number of chains can be used and the chains are divided among the processors. However this is a more advanced feature and requires compiling QLens with MPI, which is outside the scope of this tutorial.

5.3.4 Plotting Posterior Distributions in the Model Parameters

After using either nested sampling or T-Walk to map out the parameter space, we need to make histograms of the resulting chain of points in order to plot the posterior probability distribution in each parameter. This is accomplished with the utility *mkdist*, which is included with QLens. To see how this works, run "alphafit_nest.in" until the nested sampling run finishes. Then exit QLens (or else open another terminal), and from the command prompt type

mkdist alpha_nest -n40 -P

This tells mkdist to process the nested sampling data under the label "alpha_nest", which is in the directory "chains_alpha_nest", and make histograms in each parameter, using 40 bins in each case. If you get a Python error, it probably means Python can't find the Python script used to do the plotting, which is called "GetDistPlots.py". In this case you may need to add the QLens directory to your PYTHONPATH variable, which you can do from the prompt by typing "export PYTHONPATH=$PYTHONPATH:<qlens_directory>". Even better, this line should be added to your shell login script. Contact me if you are having trouble with this (qeminor@gmail.com).

The ranges in each parameter for the histogram are chosen automatically based on where the binned values are greater than a chosen threshold (default value is 3×10^{-3}, but this can be customized). The '-P' argument tells mkdist to output the posteriors to a PDF. You can view the resulting file "alpha_nest.pdf". If you prefer to see smoothed histograms that approximate the underlying probability distribution, add the argument '-S'.[7] The optimal number of bins takes a bit of experimentation: if you use too many bins, the histograms will look noisy, whereas if you use too few bins, the histograms may end up looking wider than they really are (because the bins are too wide). The latter issue is particularly dangerous if you are smoothing the histograms, because the smoothing can hide the fact that the posteriors' widths are being inflated due to oversized bins. When in doubt, try doubling the number of bins to make sure the posterior widths are not affected.

The 1-dimensional posteriors thus plotted give you a good idea of the estimated errors in each parameter. However, sometimes we want more information than this; for example, we might want to know if there is some degeneracy or correlation between a pair of parameters, which can give us physical insight. Hence, we would like to plot 2-d posteriors for any pair of parameters. To accomplish this, type

mkdist alpha_nest -n40 -N30 -SP

Here, the '-N30' argument tells mkdist to plot 2-d posteriors with 30×30 bins for each pair of parameters. This can take several minutes to finish, especially if the number of parameters is large and if the number of points produced by the Monte Carlo sampler is large. After some time, the file "alpha_nest_2D.pdf" will be created, which plots all the 2-d posteriors. However, a more useful version is also plotted, "alpha_nest_tri.pdf" which organizes both the 1-d and 2-d posteriors together in an easy-to-follow manner.

How can we quantify the width of the posteriors in each parameter? A typical figure is to quote the parameter range that covers 68% of the total

[7] For convenience, multiple letter arguments can be combined, e.g. you can type '-SP' to combine the above two options in a single argument.

probability covered by the posterior, centered around the point that cuts the total probability (area under the curve) in half. This is known as a standard error or 68% *confidence interval*[8] (the proper Bayesian term would be *credible* interval, but this terminology is less often used). If you run mkdist with the argument '-e', the mean and standard errors are calculated and displayed for each parameter.

The procedure described above is exactly the same as the procedure used for MCMC chains produced by the T-Walk algorithm. However, with the T-Walk there is one additional factor to consider: the points produced by the chains initially tend to be junk, as it takes a bit of time to sample the posterior well. Thus, it is best to discard points from this initial "burn-in" phase of the MCMC. By default, mkdist will discard the first 10% of points from each chain produced by T-Walk. However, depending on the situation, more points may need to be discarded to avoid spurious peaks and noise in the posterior. If the posterior already looks fairly smooth, don't worry about it. Otherwise, the '-c' argument controls how many points get cut, so you can experiment with cutting, say, 200 vs. 800 or more points. Obviously if you cut too many points, you will have fewer points to bin and the histogram will be choppier, so you have to find the right balance.

5.3.5 Plotting Posteriors After a Chi-Square Optimization

What if we have only found a best-fit model by minimizing the chi-square function (using downhill simplex or Powell's method)? Is there any way to plot the resulting posterior using mkdist? Technically no, since we have not mapped out the parameter space. However, if we are confident that we have found a unique best-fit point, the posterior can be approximated as a normal distribution around this point, and by calculating derivatives of the posterior at this point, we can plot approximate posterior distributions. This is done using something called a Fisher matrix analysis, the details of which are not important at this stage (the same analysis is used to calculate the parameter standard errors that are output at the end of a QLens chi-square optimization run). When you type "fit save_bestfit", the information necessary to do this is saved in the form of a matrix (the so-called parameter covariance matrix) along with the best-fit parameter values, which are then used to plot the posteriors. To do this, run mkdist with the '-f' argument, for example,

```
mkdist alphafit -f -P
```

It may take a few minutes to plot all the posteriors, since both 1-d and 2-d posteriors are plotted automatically. Again, keep in mind that the Fisher matrix approximation is only reasonable if there is a unique best-fit point (i.e.

[8] 68% is chosen because in many cases—if the probability follows a normal distribution—this range can be found by taking the standard deviation around the mean of the points being binned, often referred to as 1-sigma. If you take *twice* the standard deviation (2-sigma), this covers 95% of the total probability and hence is referred to as the 95% confidence interval.

there are no severe degeneracies present), and the fit is strongly preferred over neighboring regions, resulting in a normal distribution. With this caveat in mind, the Fisher matrix analysis can still be useful for visualizing parameter errors and correlations without having to take the time for a full Monte Carlo sampling of the parameter space.

6 Modeling Pixel Data with Point Images

Thus far we have discussed modeling gravitational lenses in which the images are effectively point-like. This is true for lensed quasars and supernovae since they are too small to be resolved even in the largest telescopes, but is not true for lensed galaxies, which produce extended images. For lensed galaxies, therefore, we should analyze the pixellated camera image of the lens system and try to reconstruct the source galaxy as well as the lens parameters. QLens can do this, although this kind of modeling is not covered in this tutorial. However, even if you are modeling a full pixel image of a gravitational lens, it is often a good idea to first treat the images as point-like and do an initial fit using a point source. To do this, we can load the pixel image into QLens and then find the centroid of each lensed image, reducing it to a set of point images, which we then model as described above.

We do this as follows. First, typically the pixel data will be in the form of a FITS file. To read FITS files, QLens must be compiled with the "cfitsio" library. In the following we will assume QLens has been compiled this way.

To load the FITS file, type "sbmap loadimg <file_name.fits>". You can check that the image has been loaded correctly by typing "sbmap plotdata". By default, the scale of the x/y axes are determined by whatever the "grid" command is set to. However, if you know the pixel size (e.g. in arcseconds), you can type "fits_pixel_size = <size>" before loading the image, and then it will scale the axes using this pixel size.

To find the centroid of each image, we use the "imgdata add_from_centroid ..." command. You must specify a window that contains each image; for example, if the image lies in a window running from $x = 0.5$ to $x = 1.2$, and from $y = -0.3$ to $y = 0.3$, then you find the centroid as follows: "imgdata add_from_centroid 0.5 1.2 -0.3 0.3". After finding the centroid, this is added as a data point image, as you can verify this by typing "imgdata". Every time you add another centroid, it will add another image point. The error in the position is simply determined by the pixel size. To see if the centroid is roughly in the right place, you can type "imgdata plot sbmap". This plots the centroid points superimposed with the pixel map so you can compare them. Once you have all the centroids and they look good, you can fit to the data using the same method as in the previous section.

Be careful: if there is any significant pixel noise in the image, this can screw up the centroid calculation. A way to mitigate this is to define a minimum threshold for surface brightness, where pixels that are below the stated surface brightness threshold are not included in the calculation. This way, if you set

the threshold high enough to where the signal dominates over the noise, then the centroid will be more accurate. You can do this by typing "sb_threshold = <threshold>". Also, note that the total flux of each image is also calculated—I would only recommend using this flux in your fit if the image is sufficiently compact. Otherwise, the overall flux may differ enough from the flux near the centroid that it will throw off your fit, particularly since the estimated error in the flux (due to pixel noise) may significantly underestimate the true error in that case.

Once you get a decent fit to the centroids, you can proceed to do a full pixel-based analysis, modeling the source as a pixellated object. QLens can do this using the "sbmap" commands, but I haven't documented them in this tutorial. If you are interested in pixel-based modeling, you can start with the help documentation in QLens, and email me with any questions you have.