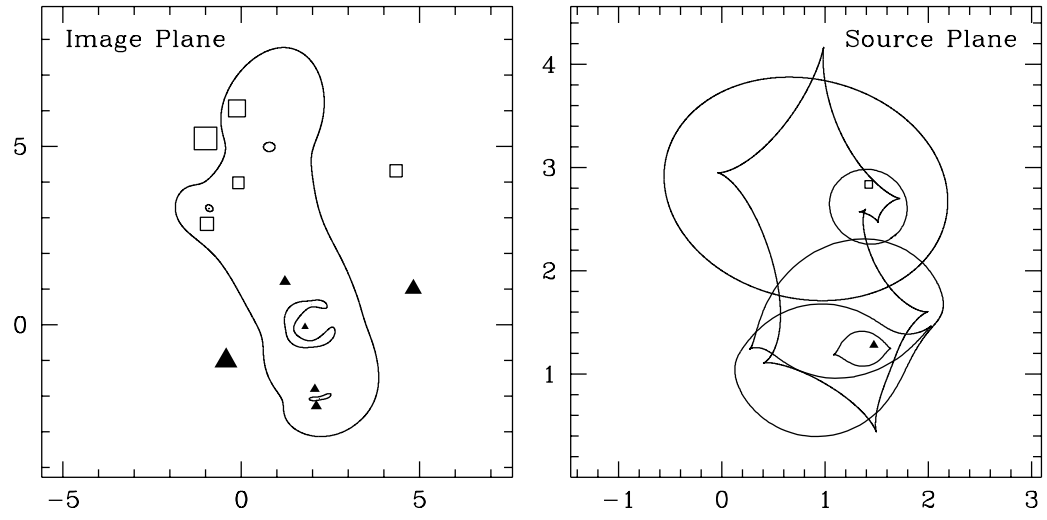
gravlens 1.06

Software for Gravitational Lensing

Chuck Keeton

Version 9 January 2004



Acknowledgements

The techniques and code presented here have been under development for several years, and many people have contributed to the work. Chris Kochanek has offered a tremendous number of ideas, and has thoroughly tested each feature as it was added to the code. Joanne Cohn, Jose Muñoz, David Rusin, Brian McLeod, and Joseph Lehár have put the code through numerous real-world tests, uncovering many species of bugs and offering helpful suggestions for the code and documentation. My thanks to them, and to all of the other users who have offered comments at various times.

Support for this work has been provided by ONR-NDSEG grant N00014-93-I-0774, NSF grant AST-9407122, NASA ATP grant NAG5-4062, Steward Observatory, and Hubble Fellowship grant HST-HF-01141.01-A from the Space Telescope Science Institute, which is operated by the Association of Universities for Research in Astronomy, Inc., under NASA contract NAS5-26555.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction and outline

The `gravlens` package comprises two stand-alone applications that offer somewhat different capabilities: `gravlens` includes a wide range of basic lensing calculations, while `lensmodel` adds many routines for modeling strong lenses. (A package for computing lens statistics is under development.) Two companion papers present the conceptual foundation of the code ([Kee01b]) and a catalog of mass models for lensing ([Kee01a]). This manual explains technical details of the applications as follows:

- Chapter 1: Describes conventions in the code and manual, and reviews the basic lens theory needed to understand how the code works.
- Chapter 2: Reviews the development history, giving changes from previous versions of the software.
- Chapter 3: Describes ellipsoidal lens models, including the "canonical" lens models available in the code (§3.2 and Table 3.2). Also discusses how to combine multiple ellipsoidal lens models into composite models that offer a great deal of complexity and generality.
- Chapter 4: Describes how to run the `gravlens` application, which includes all the basic capabilities of the code for standard lensing calculations. The features in `gravlens` form the foundation of the entire code package, so this is a very important chapter.
- Chapter 5: Reviews general strategies for modeling strong lenses, many of which are implemented in the `lensmodel` application.
- Chapter 6: Describes how to use `lensmodel` to model strong lenses.
- Chapter 7: Offers a `lensmodel` tutorial with step-by-step examples.

- Chapter 8: Lists warnings and error messages that you might encounter.
- Chapter 9: Lists some known limitations of the current version of the software.

1.2 Conventions

The applications are controlled by a command-driven, user-friendly interface. In this manual, `unixprompt%` represents the Unix system prompt, while `>` represents the prompt of the application interface. Most commands take command-line arguments, and these are denoted by

```
> command <arg1> <arg2> [arg3]
```

where angle brackets (`<arg1>`) denote required arguments and square brackets (`[arg3]`) denote optional arguments. Some commands will prompt you to enter further information, and this case is denoted by

```
> command <args...>
    <further input goes here>
```

You can run any of the `gravlens` applications interactively and enter commands at the prompt, for example:

```
unixprompt% gravlens
> help
...
> quit
unixprompt%
```

Or you can collect the commands into an input file, for example `foo.in` containing

```
help
...
quit
```

and then run the application with the command

```
unixprompt% gravlens foo.in
```

The code accepts multiple input files, for example

```
unixprompt% gravlens foo1.in foo2.in [...]
```

Regardless of whether you run the code interactively or with input files, you can include blank lines or comment lines beginning with `#`. The ability to include blank lines and comments applies to all data files as well.

The applications offer rudimentary online help via the `help` command. The command

```
> help
```

lists all the available commands with brief descriptions, while the command

```
> help <command>
```

gives more detailed help on a particular command.

There are a number of commands use a loop to repeat a calculation for a range of values of some parameter. Such commands have the form:

`foo` ⟨*x lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩

Performs some operation repeatedly, looping over x from `lo` to `hi` in the specified number of steps.

Important note: If `steps` > 0 the loop takes linear steps, but if `steps` < 0 the loop takes *logarithmic* steps. For example, the command

```
> foo 2.0 10.0 5
```

would yield a loop with $x = 2, 4, 6, 8, 10$. By contrast, the command

```
> foo 2.0 10.0 -5
```

would yield a loop with $x = 2, 2.99, 4.47, 6.69, 10$.

Each application gives you complete control over its operation via a set of internal variables, which are discussed in the relevant chapters. Each variable has a default value that gives reasonable operation, so you can run without having to set any values. But if you wish to fiddle, use the `set` command to view or change the current value of a variable:

`set` ⟨*variable*⟩ = ⟨*value*⟩

Sets the specified variable to the specified value.

`set` [*variable*]

Shows current variable values. If *variable* is specified, shows only its value; otherwise shows all variable values.

In this manual, the default value of each variable is given in square brackets (`[...]`) as part of the variable's definition.

To end the program use the `quit` command.

1.3 A review of lens theory

A thorough discussion of lens theory is given in the book by [SEF92]; I summarize the relevant aspects here. Note that lensing calculations are all two-dimensional calculations based on projected densities and potentials; in this manual, r is always a projected radius, $r = \sqrt{x^2 + y^2}$. Suppose a source at angular position $\mathbf{u}$ emits a light ray that passes a foreground mass distribution (the lens) with impact parameter $\mathbf{x}$ before arriving at the observer. The light ray is deflected by the gravitational field of the lens, which is assumed to occupy a small fraction of the total path length (the "thin lens" approximation).[1] Compared to an undeflected ray, the deflected ray has a longer travel time because it has a longer geometric length and because it passes through a gravitational potential well. The extra light travel time is

$$\tau(\mathbf{x}) = \frac{1+z_l}{c}\,\frac{D_{ol}D_{os}}{D_{ls}}\left[\frac{1}{2}\left|\mathbf{x}-\mathbf{u}\right|^2 - \phi(\mathbf{x})\right], \tag{1.1}$$

where z_l is the redshift of the lens, and D_{ol}, D_{os}, and D_{ls} are angular diameter distances from the observer to the lens, from the observer to the source, and from the lens to the source, respectively. The potential ϕ is the two-dimensional gravitational potential that is related to the surface mass density $\Sigma(\mathbf{x})$ of the lens by the Poisson equation

$$\nabla^2\phi(\mathbf{x}) = 2\kappa(\mathbf{x})\,, \tag{1.2}$$

where $\kappa(\mathbf{x}) = \Sigma(\mathbf{x})/\Sigma_{cr}$ is the surface mass density in units of the critical surface density for lensing,

$$\Sigma_{cr} = \frac{c^2}{4\pi G}\,\frac{D_{os}}{D_{ol}D_{ls}}\ . \tag{1.3}$$

By Fermat's principle, images form at stationary points of the time delay surface, or at solutions of the equation

$$\mathbf{u} = \mathbf{x} - \nabla\phi(\mathbf{x})\,. \tag{1.4}$$

This is the general form of the gravitational lens equation (for a single lens plane). It relates the source position $\mathbf{u}$ to the image position $\mathbf{x}$ via the deflection angle $\alpha = \nabla\phi$.

The lens equation shows that image positions are completely determined by the deflection function, which is given by the first derivatives of the potential ϕ. Inverting the Poisson equation and differentiating gives the deflection in terms of an integral over the surface mass density,

$$\alpha(\mathbf{x}) = \frac{1}{\pi}\int \frac{\mathbf{x}-\mathbf{y}}{\left|\mathbf{x}-\mathbf{y}\right|^2}\,\kappa(\mathbf{y})\,d\mathbf{y}\,. \tag{1.5}$$

The lens also distorts and amplifies the image(s) in a manner described by the magnification tensor, which is the Jacobian of the lens mapping,

$$\mu \equiv \left(\frac{\partial\mathbf{u}}{\partial\mathbf{x}}\right)^{-1} = \begin{bmatrix} 1-\phi_{,xx} & -\phi_{,xy} \\ -\phi_{,xy} & 1-\phi_{,yy} \end{bmatrix}^{-1}, \tag{1.6}$$

[1] The code currently assumes a single lens plane.

where subscripts denote partial differentiation, $\phi_{,ij} \equiv \partial^2\phi/\partial x_i \partial x_j$. Generically, a strong lens has one or more "critical curves" in the image plane along which $\det \mu^{-1} = 0$, which map to "caustics" in the source plane. Caustics are important because they are places where the number of images changes, and a source near a caustic is highly amplified and distorted.

For any calculation involving just the locations and brightnesses of images (which includes most lensing applications), we can completely specify the lensing properties of a given mass distribution by giving the first and second derivatives of the potential ϕ. If we also need the relative time delays (such as for lensing measurements of the Hubble constant H_0), then from eq. (1.1) we need the potential ϕ itself. The companion paper ([Kee01a]) reviews the potentials, deflections, and magnifications for an extensive collection of lens models.

A word about units. The time delay equation (1.1) is written for image and source positions $\mathbf{x}$ and $\mathbf{u}$ in angular units. However, the lens equation (1.4) is more general and makes no explicit assumptions about units. If L is the basic unit of length, then the image and source positions have dimension L. The scaled surface density κ is dimensionless. Then by eq. (1.2) the potential ϕ has dimension L^2, and hence the deflection $\nabla\phi$ has dimension L. When modeling observed lenses, it may be natural to take L to be an angular unit (such as arcseconds). When studying theoretical lenses there may be a natural length scale, for example in microlensing the Einstein ring radius of the lens star.

Chapter 2

Development History

- **v1.0 – 14 Feb 2001**

  First general release.

- **v1.01, v1.02 – 11 July 2001**

  Small additions, including `clus3` models and the `verbose` variable; minor internal tweakings.

- **v1.03 – 7 Aug 2001**

  Added elliptical versions of `nfwcusp` models; extended `cusp` models to handle $n = 3$; added the `version` command; upgraded the curve fitting algorithm to handle multiple families of curves (see p. 63). *Note that the curve fitting algorithm is likely to change further; contact me for more information.*

- **v1.04 – 4 June 2002**

  Added `alphapot` and `nuker` models; added `calcRein` command; added `vertmode` variable (thanks to Chris Kochanek); fixed a small bug in the use of `omitcore`; added more precision to `devauc` models (the k constant was changed from 7.67 to 7.66925001; thanks to Paul Schechter); fixed a typo in the manual in the equations for `softened power law` models (thanks to David Rusin).

- **v1.05 – 10 Feb 2003**

  Added `magtensor`, tt mock1 and `mock2` commands; added `maxshear` and `NGALMAX` variables; improved use of `chiperpoint` variable; updated normalization of `cusp` model class; added `monopole`, `mpole`, and `intshr` models for multipole series; also added `boxypot` model class.

- **v1.06 – 9 Jan 2004**

  Add `autogrid`, `plotkappa`, `SBmap1`, `SBmap2`, `findimg2`, and `findimg3` cmmands; added `jaffe`, `nfwpot`, and `fourmode` model classes; modified `nfwcusp` model class to use a fitting formula for the case $\gamma = 1.5$; modified `mock1` and `mock2` commands to add more capabilities; made small changes to `plotdef0` command and `convrg` model class.

Chapter 3

Mass Models

To solve the gravitational lens equation (1.4) we need to know the lensing properties (mainly the deflection and magnification) corresponding to a given mass distribution. The magnification components are derivatives of the deflection, so evaluating the deflection is the key step. Unfortunately, for a general mass distribution the deflection integral (eq. 1.5) cannot be evaluated analytically. If the mass distribution has ellipsoidal symmetry, though, the integral can be simplified and in some cases evaluated analytically. The `gravlens` package includes analytic or simple numerical solutions for a canonical set of ellipsoidal lens models. Moreover, it allows you to combine the canonical models in arbitrary ways to obtain an enormous variety of composite lens models representing, for example, galaxy groups, galaxies or clusters with substructure, stars with planetary systems, etc.

This chapter describes the canonical models included in the code, which encompasses all models that are currently used to study lensing phenomena; some of the material in this chapter is repeated from the companion catalog of mass models ([Kee01a]). The basic equations for the lensing properties of ellipsoidal mass distributions are given in §3.1. The canonical models are defined in §3.2. Combining canonical models to produce much more general composite models is discussed in §3.3. The choice of appropriate mass models is discussed in §3.4. Finally, Tables 3.2–3.3 summarize the canonical models, and §3.5 reviews some technical details of the models. See the catalog for a detailed discussion of all of the canonical models, including known analytic results for their lensing properties.

3.1 Basic equations

The general expression for the deflection in terms of a 2-d integral over the mass distribution is given in eq. (1.5). If the mass distribution has circular symmetry, the situation simplifies considerably. The deflection vector is purely radial and has amplitude given by the 1-d integral

$$\phi_{,r}(r) = \frac{2}{r}\int_0^r u\,\kappa(u)\,du = \frac{1}{\pi\Sigma_{cr}}\,\frac{M(r)}{r}\,, \tag{3.1}$$

which is often easily evaluated. This function can be integrated to obtain the potential or differentiated to obtain the magnification. The catalog ([Kee01a]) gives the circular deflection for all of the canonical models.

More generally, we want to consider mass distributions with elliptical symmetry, i.e. with a surface density of the form

$$\kappa = \kappa(\xi)\,, \quad \text{where} \quad \xi^2 = x^2 + y^2/q^2\,, \tag{3.2}$$

where ξ is the ellipse coordinate and q is the projected axis ratio. (This is the functional form expressed in a coordinate system with the mass distribution centered at the origin and oriented with its major axis along the x-axis.) The elliptical symmetry makes it possible to reduce the deflection integral to a 1-d integral (e.g., [Sch90]). We can then write the lensing properties as a set of 1-d integrals,

$$\begin{aligned}
\phi(x,y) &= \frac{q}{2}\, I(x,y) && (3.3)\\
\phi_{,x}(x,y) &= q\, x\, J_0(x,y) && (3.4)\\
\phi_{,y}(x,y) &= q\, y\, J_1(x,y) && (3.5)\\
\phi_{,xx}(x,y) &= 2\, q\, x^2\, K_0(x,y) + q\, J_0(x,y) && (3.6)\\
\phi_{,yy}(x,y) &= 2\, q\, y^2\, K_2(x,y) + q\, J_1(x,y) && (3.7)\\
\phi_{,xy}(x,y) &= 2\, q\, x\, y\, K_1(x,y) && (3.8)
\end{aligned}$$

where the integrals are

$$\begin{aligned}
I(x,y) &= \int_0^1 \frac{\xi}{u}\, \frac{\phi_{,r}\left(\xi(u)\right)}{\left[1-(1-q^2)u\right]^{1/2}}\, du && (3.9)\\
J_n(x,y) &= \int_0^1 \frac{\kappa\left(\xi(u)^2\right)}{\left[1-(1-q^2)u\right]^{n+1/2}}\, du && (3.10)\\
K_n(x,y) &= \int_0^1 \frac{u\,\kappa'\left(\xi(u)^2\right)}{\left[1-(1-q^2)u\right]^{n+1/2}}\, du && (3.11)\\
\text{where} \quad \xi(u)^2 &= u\left(x^2 + \frac{y^2}{1-(1-q^2)u}\right) && (3.12)
\end{aligned}$$

and $\kappa'(\xi^2) = d\kappa(\xi^2)/d\xi^2$. Note from eq. (3.9) the potential can be written as an integral over the *circular* deflection function $\phi_{,r}$ from eq. (3.1), but $\phi_{,r}$ must be evaluated at the appropriate ellipse coordinate $\xi(u)$. Also, the deflection and magnification integrals are written here in terms of $\kappa(\xi^2)$ because the canonical models generally depend on ξ^2 rather than ξ alone.

For reference, if you need third-order derivatives of the potential, they are:

$$\phi_{,xxx}(x,y) = 2\, q\, x\left[2x^2\, L_0(x,y) + 3K_0(x,y)\right] \tag{3.13}$$

$$\phi_{,xxy}(x,y) = 2\,q\,y\left[2x^2\,L_1(x,y) + K_1(x,y)\right] \tag{3.14}$$

$$\phi_{,xyy}(x,y) = 2\,q\,x\left[2y^2\,L_2(x,y) + K_1(x,y)\right] \tag{3.15}$$

$$\phi_{,yyy}(x,y) = 2\,q\,y\left[2y^2\,L_3(x,y) + 3K_2(x,y)\right] \tag{3.16}$$

where the additional integrals are

$$L_n(x,y) = \int_0^1 \frac{u^2\,\kappa''\big(\xi^2(u)\big)}{[1-(1-q^2)u]^{n+1/2}}\,du \tag{3.17}$$

For some of the canonical models these integrals can be evaluated analytically to obtain the lensing properties (see the catalog, [Kee01a]). For the remaining models the code either evaluates the integrals numerically or offers an approximate analytic solution, or both.

3.2 Canonical lens models

Table 3.2 lists the models defined in the code and gives brief summaries, while Tables 3.2 and 3.3 define the models in terms of the surface mass densities or potentials. §3.5 gives some notes about different models, while the catalog ([Kee01a]) gives a more detailed discussion of their properties. Note that for each models that requires numerical integrals, the code provides a corresponding tabulated model (e.g., `nfw` and `nfwT`); see §4.3 for a discussion of tabulated models. For some of these models, the code also provides approximate solutions (e.g., `devauc` and `devaucA`).

The code uses 10 parameters to describe an ellipsoidal mass distribution:

p[1]	=	$\mathcal{M}$	=	mass scale
(p[2], p[3])	=	(x_0, y_0)	=	galaxy position
(p[4], p[5])	=	(e, θ_e)	=	ellipticity parameters
(p[6], p[7])	=	(γ, θ_γ)	=	external shear parameters
(p[8], p[9])	=	(s, a)	=	misc., often scale radii
p[10]	=	α	=	misc., often a power law index

Here $e = 1 - q$ is the standard ellipticity. The position angles θ_e and θ_γ are always expressed in observers' coordinates, i.e. in degrees measured East of North. The ellipticity and shear parameters can be given in Cartesian rather rather than polar coordinates; see §4.2. The galaxy position and ellipticity/shear parameters have definitions that are independent of the density profile; they essentially specify the coordinate frame and are used in exactly the same way in every model. By contrast, the mass scale, scale radii, and miscellaneous parameter α specify the density profile; their use is specific to the model and is given in Tables 3.2 and 3.3.

3.3 Composite lens models

Mass models with ellipsoidal symmetry already provide much more generality than purely circular models. But they are still not fully general. Simple ellipsoidal models are not sufficient to fit

many observed strong lenses (e.g., [KKS97]); in many of these cases the departure from ellipsoidal symmetry can be attributed to tidal perturbations from galaxies near the main lens galaxy (e.g., [HB94]; [KK97]; [KCBL97]; [KHB$^+$97]; [Ton98]). In other cases, the symmetry may be broken by substructure in the lensing mass distribution; examples are spiral galaxies with disks and bulges inside halos (e.g., [MFP97]; [KK98]), and microlensing by binary stars or stars with planets (e.g., [MP91]; [GL92]).

All of these examples share the feature that the general mass model can be written as a combination of ellipsoidal models. Galaxy groups, binary stars, or stars with planets can all be expressed as a combination of multiply ellipsoidal mass distributions centered at different positions. Angular substructure such as isophote twists in elliptical galaxies or the disk+halo construction in spiral galaxies can be modeled using mass distributions with different profiles and ellipticities that are all centered on the same point (e.g., [KK98]; [KFI$^+$00]). Another possibility is combining mass distributions with fixed ellipticity centered at the same point to obtain a new model with a density profile different from any of the canonical models (also see [SW91]). Examples of this include the pseudo-Jaffe and King models (different combinations of isothermal ellipsoids; see the catalog, [Kee01a]), as well as the combinations of ellipsoids that the code uses to obtain an approximation for the exponential disk (see §3.5).

The code allows composite models composed of arbitrary combinations of ellipsoidal mass distributions, opening the door to an enormous variety of mass models that you can use.

3.4 Choosing a mass model

Selecting the class of models to use for a particular application is a key part of the modeling process. When modeling lenses produced by galaxies, a simple and useful place to start is an isothermal model, i.e. a model with density $\rho \propto r^{-2}$ and a flat rotation curve. Spiral galaxy rotation curves (e.g., [RFT78]; [RFT80]), stellar dynamics of elliptical galaxies (e.g., [RdZC$^+$97]), X-ray halos of elliptical galaxies ([Fab89]), models of some individual lenses (e.g., [Koc95]; [CKMK01]), and lens statistics (e.g., [MR93]; [Koc93]; [Koc96]) are all consistent with roughly isothermal profiles. However, an isolated isothermal ellipsoid rarely yields a good quantitative fit to observed lenses (e.g., [KKS97]; [WM97]; [KKF98]). In general, adding parameters to the radial profile of the galaxy fails to produce a good fit, but adding parameters to the angular structure of the potential dramatically improves the fit (e.g., [KK97]; [KKS97]). The additional angular structure comes from the tidal perturbations of objects near the main lens galaxy or along the line of sight. In other words, the fact that few galaxies are truly isolated means that lens models generically require two independent sources of angular structure: an ellipsoidal galaxy plus external perturbations. The combination of angular terms can make it difficult to disentangle the shape of the galaxy and the nature of the external

perturbations, and it is extremely important to understand any degeneracies between the two sources of angular structure before drawing conclusions from the models (see [KKS97]).

To move beyond isothermal models and explore other radial profiles, softened power law lens models have traditionally been very popular. However, these models have flat cores, while early-type galaxies have cuspy luminosity distributions (e.g., [FTA$^+$97]), and dark matter halos in cosmological simulations have cuspy mass distributions (e.g., [NFW96]; [NFW97]; [MGQ$^+$98]; [MQG$^+$99]). The lack of central or "odd" images in most observed galaxy lenses also limits the extent to which galaxies can have flat cores (e.g., [WN93]; [RM01]). Hence [CKMK01] argue that softened power law models are outdated and should be replaced with a family of cuspy lens models. The `gravlens` code includes several such models, including traditional NFW ([NFW96]; [NFW97]), Hernquist ([Her90]), and de Vaucouleurs ([dV48]) models, as well as two families of models with general cusps (see Table 3.2). It will be interesting to determine the extent to which galaxy and cluster lenses are consistent with a unified family of cuspy halo models. The ability to use a single profile across a wide range of masses may be limited, however, because it is expected that the baryons significantly modify the dark matter distribution, especially on galaxy mass scales (e.g., [BFFP86]; [Dub94]).

`none`	:	no mass
`convrg`	:	Convergence from a uniform mass sheet
`clus3`	:	3rd order cluster terms (à la [BF99])
`fourmode`	:	Simple Fourier mode $\cos(\mathbf{k}\cdot\mathbf{x}+\varphi)$
`monopole`	:	General monopole term
`mpole`	:	General multipole term
`intshr`	:	Internal shear quadrupole
`ptmass`	:	Point mass
`alphapot`	:	Softened power law potential
`alpha`	:	Softened power law density
`alphaT`	:	Softened power law density (tabulated)
`pjaffe`	:	Pseudo-Jaffe (truncated isothermal) model
`king`	:	King model
`boxypot`	:	Boxy power law potential
`devauc`	:	de Vaucouleurs $r^{1/4}$ model (numerical)
`devaucT`	:	de Vaucouleurs $r^{1/4}$ model (tabulated)
`devaucA`	:	de Vaucouleurs $r^{1/4}$ model (approximate)
`jaffe`	:	Jaffe model (numerical)
`jaffeT`	:	Jaffe model (tabulated)
`hern`	:	Hernquist model (numerical)
`hernT`	:	Hernquist model (tabulated)
`nfw`	:	NFW elliptical density (numerical)
`nfwT`	:	NFW elliptical density (tabulated)
`nfwpot`	:	NFW elliptical potential
`nfwcusp`	:	Cuspy NFW model (numerical) – circular only
`nfwcuspT`	:	Cuspy NFW model (tabulated) – circular only
`cusp`	:	Cusp model (numerical)
`cuspT`	:	Cusp model (tabulated)
`nuker`	:	Nuker model (numerical)
`unidisk`	:	Uniform density disk
`expdisk`	:	Exponential disk (numerical)
`expdiskT`	:	Exponential disk (tabulated)
`expdiskA`	:	Exponential disk (approximate)
`kuzdisk`	:	Kuzmin disk
`tab`	:	General tabulated model

Table 3.1 Canonical lens models.

Model	Definition	p[1]	p[8]	p[9]	p[10]
convrg	$\kappa = \kappa_0$ (uniform)	κ_0	–	–	–
clus3	$\phi = \frac{r^3\sigma}{4}[\sin(\theta-\theta_\sigma) + \sin 3(\theta-\theta_\sigma)]$	–	σ	θ_σ	–
fourmode	$\kappa = \delta\kappa\, \cos(ex+\varphi)$	$\delta\kappa$	–	–	φ
monopole	$\kappa = \langle\kappa\rangle\ (r/R_0)^{1-n}$	$\langle\kappa\rangle$	R_1	R_2	n
mpole	$\phi = -\frac{r^n\epsilon}{m}\cos\left[m\left(\theta-\theta_m-\frac{\pi}{2}\right)\right]$	–	–	m	n
intshr	$\phi = \frac{b^4\gamma_{\rm int}}{2r^2}\ \cos 2(\theta-\theta_{\rm int})$	b	–	–	–
ptmass	$\kappa = \pi\, R_E^2\, \delta(\mathbf{x})$	R_E	–	–	–
alphapot	$\phi = b\,\left(s^2+\xi^2\right)^{\alpha/2}$	b	s	–	α
alpha, alphaT	$\kappa = \frac{1}{2}\,(b')^{2-\alpha}\ \left[(s')^2+\zeta^2\right]^{\alpha/2-1}$	b'	s'	–	α
pjaffe	$\kappa = \frac{b'}{2\sqrt{(s')^2+\zeta^2}} - \frac{b'}{2\sqrt{(a')^2+\zeta^2}}$	b'	s'	a'	–
king	$\kappa = \frac{2.12b'}{\sqrt{0.75r_s^2+\zeta^2}} - \frac{1.75b'}{\sqrt{2.99r_s^2+\zeta^2}}$	b'	r_s	–	–
boxypot	$\phi = b\, r^\alpha\,[1-\epsilon\cos 2(\theta-\theta_\epsilon)]^\beta$	b	–	β	α
devauc, devaucT, devaucA	$\kappa = \frac{b}{2\mathcal{N}R_e}\ \exp\left[-k(\xi/R_e)^{1/4}\right]$	b	R_e	–	–
jaffe, jaffeT	(see eq. 3.28)	κ_s	r_s	–	–
hern, hernT	(see eq. 3.33)	κ_s	r_s	–	–

Table 3.2 – Canonical lens models. Column 2: the potential (ϕ) or density (κ) as a function of the elliptical radius ξ or ζ in coordinates aligned with the major axis of the ellipse. Some models are written in terms of $\xi = \left(x^2+y^2/q^2\right)^{1/2}$, in which case scale lengths are expressed on the major axis. Others are written in terms of $\zeta = \left[(1-\epsilon)x^2+(1+\epsilon)y^2\right]^{1/2}$ where ϵ is related to the axis ratio by $q^2 = (1-\epsilon)/(1+\epsilon)$; in this case scale lengths are expressed on an intermediate axis. Columns 3–6 show how the code parameters relate to the model parameters; "–" means that the code parameter is not used. Explicit formulas for the lensing properites of each model are given in the companion catalog of mass models ([Kee01a]).

Model	Definition	p[1]	p[8]	p[9]	p[10]
nfw, nfwT	(see eq. 3.36)	κ_s	r_s	–	–
nfwpot	(see eq. 3.38)	κ_s	r_s	–	–
nfwcusp, nfwcuspT	(see eq. 3.40)	κ_s	r_s	–	γ
cusp, cuspT	(see eq. 3.43)	b	r_s	n	γ
nuker	(see eq. 3.45)	κ_b	α	β	γ
unidisk	$\kappa = \begin{cases} q^{-1}\,\kappa_0 & \text{if } \xi < R_d \\ 0 & \text{else} \end{cases}$	κ_0	R_d	–	–
expdisk, expdiskT, expdiskA	$\kappa = q^{-1}\,\kappa_0\,\exp\left[-\xi/R_d\right]$	κ_0	R_d	–	–
kuzdisk	$\kappa = q^{-1}\,\kappa_0\,r_s^3\,\left(r_s^2+\xi^2\right)^{-3/2}$	κ_0	r_s	–	–

Table 3.3 – More canonical lens models.

3.5 Appendix: Notes on the canonical models

This section offers technical details for some of the models in Table 3.2. See the catalog ([Kee01a]) for a full explanation of the models, including explicit formulas for their lensing properties.

Fourier modes: A general Fourier mode in the density can be written as $\kappa = \delta\kappa \cos(\mathbf{k}\cdot\mathbf{x} + \varphi)$, where $\delta\kappa$ is the amplitude, $\mathbf{k}$ is the wavevector, and φ is the phase. In coordinates aligned with the wavevector, this is just $\kappa = \delta\kappa \cos(kx + \varphi)$. In the code, the ellipticity and position angle parameters are used to specify the amplitude and direction of the wavevector.

Multipole terms: In some cases it may be convenient to write the lens model using a multipole series (e.g., [Koc02]). This can be done by combining a general monopole term with any number of higher-order multipole terms. The general `monopole` model is defined in terms of the density,

$$\kappa(r) = \langle\kappa\rangle \left(\frac{r}{R_0}\right)^{1-n}, \tag{3.18}$$

where the model is normalized by the mean density $\langle\kappa\rangle$ in an annulus from R_1 to R_2, and the scale radius is taken to be the midpoint of the annulus, $R_0 = (R_1 + R_2)/2$. A general multipole term can be specified with the `mpole` model defined in terms of the potential,

$$\phi(r,\theta) = -\frac{\epsilon_m r^n}{m} \cos\left[m\left(\theta - \theta_m - \frac{\pi}{2}\right)\right], \tag{3.19}$$

where the orientation angle θ_m is written here as an observer's position angle, i.e., measured East of North. With $n = 2$ and $m = 2$ this is equivalent to an external shear term. In the code, the amplitude ϵ_m and orientation angle θ_m are specified by the parameters `p[4]` and `p[5]`.

A specific multipole term of interest is the shear arising from mass interior to the Einstein ring. This `intshr` model is defined in terms of the potential,

$$\phi(r,\theta) = \frac{b^4\,\gamma_{\rm int}}{2r^2} \cos 2(\theta - \theta_{\rm int})\,. \tag{3.20}$$

The parameters $\gamma_{\rm int}$ and $\theta_{\rm int}$ are specified by the shear parameters `p[6]` and `p[7]`, but in two different modes specified by the variable `intshrmode`. If `intshrmode` = 1 then p[6] = $\gamma_{\rm int}$ and p[7] = $\theta_{\rm int}$. However, if `intshrmode` = 2 then the internal shear is specifed in relation to the total shear. Specifically, p[6] = $f_{\rm int}$ and p[7] = $\Delta\theta_\gamma$ where

$$f_{\rm int} = \frac{\gamma_{\rm int}}{\gamma_{\rm int} + \gamma_{\rm ext}}\,, \tag{3.21}$$

$$\Delta\theta_\gamma = \theta_{\rm int} - \theta_{\rm ext}\,, \tag{3.22}$$

where "int" and "ext" refer to the internal and external shear, respectively.

External perturbations: Objects near the main lens galaxy or along the line of sight often perturb the lensing potential. If the perturbation is weak it may be sufficient to expand the perturbing potential as a Taylor series and keep only a few terms. In a coordinate system centered on

the lens galaxy, the expansion to 3rd order can be written as (see [Koc91b]; [BF99])

$$\phi \approx \phi_0 + \mathbf{b}\cdot\mathbf{x} + \frac{r^2}{2}\left[\kappa_0 + \gamma\,\cos 2(\theta-\theta_\gamma)\right] + r^3\Big[\frac{\sigma}{4}\,\sin(\theta-\theta_\sigma) - \frac{\delta}{6}\,\sin 3(\theta-\theta_\delta)\Big] + \ldots \qquad (3.23)$$

(The direction angles $(\theta_\gamma, \theta_\sigma, \theta_\delta)$ are written here as observers' position angles, i.e., measured East of North.) The 0th order term ϕ_0 represents an unobservable zero point of the potential and can be dropped. The 1st order term $\mathbf{b}\cdot\mathbf{x}$ represents an unobservable uniform deflection and can also be dropped. The 2nd order term κ_0 represents the convergence from the perturbing mass and is equivalent to a uniform mass sheet with density $\Sigma/\Sigma_{cr} = \kappa_0$. The only observable effect of this term is to rescale the time delay(s) by $1-\kappa_0$, which leads to the "mass sheet degeneracy" (e.g., [FGS85]); hence this term is often omitted from lens models and introduced *a posteriori* using independent mass constraints (see, e.g., [BF99]). However, if desired you can include this term explicitly using the `convrg` model class. The 2nd order term γ represents an external tidal shear with strength γ and direction θ_γ; you can include this term using the standard shear parameters (see §3.2). The 3rd order term σ arises from the gradient of the surface density $\kappa(\mathbf{x})$ of the perturber; it has an amplitude $\sigma = |\nabla\kappa|$ and a direction equal to the direction of $\nabla\kappa$. The 3rd order term δ arises from the $m=3$ multipole moment of the perturbing mass. To avoid an explosion of parameters, it may be interesting to use a "restricted" 3rd order cluster with $\theta_\delta = \theta_\sigma$ and $\delta = -3\sigma/2$, relations which are exact for a singular isothermal sphere perturber and are probably reasonable for other perturbers (see [BF99]; [KFI+00]). The `clus3` model class implements this restricted 3rd order cluster. Note that `clus3` models use the parameters `p[8]` $= \sigma$ and `p[9]` $= \theta_\sigma$, so these two parameters are not scale lengths.

The **softened power law** and related models (including the **isothermal**, $\alpha = -1$, **pseudo-Jaffe**, and **King** model ellipsoids) have a slightly different normalization in the code than in the catalog. In the catalog the surface density for softened power law models is written as

$$\kappa(x,y) = \frac{b^{2-\alpha}}{2\left(s^2 + x^2 + y^2/q^2\right)^{1-\alpha/2}}\ , \qquad (3.24)$$

while in the code it is written as

$$\kappa(x,y) = \frac{(b')^{2-\alpha}}{2\left[(s')^2 + (1-\epsilon)x^2 + (1+\epsilon)y^2\right]^{1-\alpha/2}}\ , \qquad (3.25)$$

where ϵ is related to the axis ratio by $q^2 = (1-\epsilon)/(1+\epsilon)$. The only difference between the two normalizations is in the b and s parameters; the two normalizations are related by

$$\frac{b'}{b} = \frac{s'}{s} = q\sqrt{\frac{2}{1+q^2}}\ . \qquad (3.26)$$

The `boxypot` model can have isopotential contours that are boxy or disky rather than pure ellipses (see [CCRP01], [ZP01]). The parameter β controls the boxiness or diskiness, with $\beta = 1/2$

corresponding to strict elliptical symmetry. The parameter ϵ is related to the flattening of the isopotential contours.

The two constants in **de Vaucouleurs** models are $k = 7.66925001$ and the normalization factor $\mathcal{N} = 20160/k^8$

The **Jaffe model** [Jaf83] is a 3-d density distribution with the form

$$\rho = \frac{\rho_s}{(r/r_s)^2(1+r/r_s)^2} \ , \tag{3.27}$$

where r_s is a scale length and ρ_s is a characteristic density. The projected surface mass density has the form

$$\kappa(r) = \kappa_s \left[\frac{\pi}{x} + 2\,\frac{1-(2-x^2)\mathcal{F}(x)}{1-x^2}\right], \tag{3.28}$$

where $x = r/r_s$, $\kappa_s = \rho_s\, r_s/\Sigma_{crit}$, and

$$\mathcal{F}(x) = \begin{cases} \frac{1}{\sqrt{x^2-1}}\tan^{-1}\sqrt{x^2-1} & (x>1) \\ \frac{1}{\sqrt{1-x^2}}\tanh^{-1}\sqrt{1-x^2} & (x<1) \\ 1 & (x=1) \end{cases} \tag{3.29}$$

A useful technical result is the derivative of this function:

$$\mathcal{F}'(x) = \frac{1-x^2\mathcal{F}(x)}{x(x^2-1)} \ . \tag{3.30}$$

The circular deflection is

$$\text{circular:} \quad \phi_{,r} = 2\,\kappa_s\, r_s\, \left[\pi - 2\,x\,\mathcal{F}(x)\right], \tag{3.31}$$

where again $x = r/r_s$. The code defines an elliptical model by using the surface mass density $\kappa(\xi)$, i.e. replacing the polar radius r with the ellipse coordinate ξ in eq. (3.28). The elliptical model cannot be solved analytically.

The **Hernquist model** [Her90] is a 3-d density distribution proposed to describe the light distribution of early-type galaxies. It has the form

$$\rho = \frac{\rho_s}{(r/r_s)(1+r/r_s)^3} \ , \tag{3.32}$$

where r_s is a scale length and ρ_s is a characteristic density. The projected surface mass density has the form

$$\kappa(r) = \frac{\kappa_s}{(x^2-1)^2}\, \left[-3 + (2+x^2)\mathcal{F}(x)\right], \tag{3.33}$$

where $x = r/r_s$, $\kappa_s = \rho_s\, r_s/\Sigma_{crit}$, and $\mathcal{F}$ is given by eq. (3.29). The circular deflection is

$$\text{circular:} \quad \phi_{,r} = 2\,\kappa_s\, r_s\, \frac{x[1-\mathcal{F}(x)]}{x^2-1} \ , \tag{3.34}$$

where again $x = r/r_s$. The code defines an elliptical model by using the surface mass density $\kappa(\xi)$, i.e. replacing the polar radius r with the ellipse coordinate ξ in eq. (3.33). The elliptical model cannot be solved analytically.

The **NFW model** arises from cosmological N-body simulations (e.g., [NFW96]; [NFW97]) which suggest that dark matter halos can be described by a "universal" density profile with the form

$$\rho = \frac{\rho_s}{(r/r_s)(1+r/r_s)^2} \ , \tag{3.35}$$

where r_s is a scale length and ρ_s is a characteristic density. [Bar96] gives the lensing properties of the (spherical) NFW model. The projected surface mass density has the form

$$\kappa(r) = 2\,\kappa_s\,\frac{1-\mathcal{F}(x)}{x^2-1}\,, \tag{3.36}$$

where $x = r/r_s$, $\kappa_s = \rho_s\, r_s/\Sigma_{crit}$, and the $\mathcal{F}$ is given by eq. (3.29). The circular deflection is

$$\text{circular:} \quad \phi_{,r} = 4\,\kappa_s\, r_s\,\frac{\ln(x/2)+\mathcal{F}(x)}{x} \ , \tag{3.37}$$

where again $x = r/r_s$. The code defines an elliptical model by using the surface mass density $\kappa(\xi)$, i.e. replacing the polar radius r with the ellipse coordinate ξ in eq. (3.36). The elliptical model cannot be solved analytically. As an alternative, the code also includes an NFW model with elliptical symmetry in the potential,

$$\phi = 2\,\kappa_2\, r_s^2\,\left[\ln^2\frac{\xi}{2} - \text{arctanh}^2\sqrt{1-\xi^2}\right], \tag{3.38}$$

which allows a fully analytic solution (see [GK02]; [MBM03]).

Some more recent simulations (e.g., [MGQ$^+$98]; [MQG$^+$99]) have suggested that the inner cusp of the NFW profile is too shallow, so several studies ([JS00]; [WTS01]; [KM01]) have considered a generalized NFW-type profile of the form

$$\rho = \frac{\rho_s}{(r/r_s)^\gamma(1+r/r_s)^{3-\gamma}} \ , \tag{3.39}$$

where r_s is a scale length. I call this a **"cuspy NFW" model** because it is like the NFW model but has a central cusp $\rho \propto r^{-\gamma}$. The projected surface density cannot be computed analytically even for a spherical halo. For a spherical model, the surface density and deflection can be written as (see the catalog, [Kee01a])

$$\kappa(r) = 2\,\kappa_s\, r_s\, x^{1-\gamma}\,\left[(1+x)^{\gamma-3} + (3-\gamma)\int_0^1 dy\,(y+x)^{\gamma-4}\left(1-\sqrt{1-y^2}\right)\right], \tag{3.40}$$

$$\phi_{,r} = 4\,\kappa_s\, r_s\, x^{2-\gamma}\ \times \tag{3.41}$$

$$\left\{\ \frac{1}{3-\gamma}\ {}_2F_1[3-\gamma, 3-\gamma; 4-\gamma; -x] + \int_0^1 dy\,(y+x)^{\gamma-3}\,\frac{1-\sqrt{1-y^2}}{y}\ \right\},$$

where $x = r/r_s$, $\kappa_s = \rho_s\, r_s/\Sigma_{cr}$, and ${}_2F_1$ is the hypergeometric function. The catalog [Kee01a] gives simple analytic expressions for $\gamma = 0, 1, 2$; [OLS03] give a fitting formula for κ for the case $\gamma = 1.5$.

Elliptical versions of the model are implemented in the code but can be time consuming because they require double integrals.

To obtain a general **cuspy model** that is more amenable to lensing, [MnKK00] introduce a model with a profile of the form

$$\rho = \frac{\rho_s}{(r/r_s)^\gamma [1+(r/r_s)^2]^{(n-\gamma)/2}} \ , \tag{3.42}$$

where again r_s is a scale length. The density scales as $\rho \propto r^{-\gamma}$ for $r \ll r_s$, and $\rho \propto r^{-n}$ for $r \gg r_s$; in other words, γ and n are the logarithmic slopes at small and large radii, respectively. This model is a subset of the models whose physical properties were studied by [Zha96]. The central cusp must have $\gamma < 3$ for the mass to be finite. For $(\gamma, n) = (1, 4)$ this is a pseudo-Hernquist model, for $(1, 3)$ it is a pseudo-NFW model, and for $(2, 4)$ it is a singular pseudo-Jaffe model. Compared with eq. (3.39), replacing $(1 + r/r_s)$ with $\sqrt{1+(r/r_s)^2}$ does not greatly change the profile shape but does make it possible to solve the spherical model analytically ([MnKK00])

$$\kappa(r) = \kappa_s \, B\left(\frac{n-1}{2}, \frac{1}{2}\right) \left(1+x^2\right)^{(1-n)/2} {}_2F_1\left[\frac{n-1}{2}, \frac{\gamma}{2}; \frac{n}{2}; \frac{1}{1+x^2}\right], \tag{3.43}$$

$$\begin{aligned} \phi_{,r} = \frac{\kappa_s r_s}{x}\Bigg\{ & 2B\left(\frac{n-3}{2}, \frac{3-\gamma}{2}\right) \\ & -B\left(\frac{n-3}{2}, \frac{1}{2}\right)\left(1+x^2\right)^{(3-n)/2} {}_2F_1\left[\frac{n-3}{2}, \frac{\gamma}{2}; \frac{n}{2}; \frac{1}{1+x^2}\right]\Bigg\}, \end{aligned} \tag{3.44}$$

where $x = r/r_s$, $\kappa_s = \rho_s \, r_s/\Sigma_{cr}$, ${}_2F_1$ is the hypergeometric function, $B(a, b) = \Gamma(a)\Gamma(b)/\Gamma(a + b)$ is the Euler beta function, and eq. (3.44) is not valid for $n = 3$. See the catalog ([Kee01a]) for expressions for the case $n = 3$. The elliptical model $\kappa(\xi)$ can be computed numerically with eqs. (3.3)–(3.8). Note that in the software the mass parameter is specified as $b = 2\pi\kappa_s$.

Many early-type galaxies have surface brightness profiles that can be modeled as a **Nuker law** ([LAB+95]; [BGF+96]),

$$I(r) = 2^{(\beta-\gamma)/\alpha} \, I_b \left(\frac{r}{r_b}\right)^{-\gamma} \left[1 + \left(\frac{r}{r_b}\right)^\alpha\right]^{(\gamma-\beta)/\alpha}, \tag{3.45}$$

where γ and β are inner and outer power law indices, respectively, r_b is the radius where the break in the power law occurs, α gives the sharpness of the break, and I_b is the surface brightness at the break radius. If the luminosity distribution has circular symmetry and the mass-to-light ratio is Υ, the lensing deflection is ([Kee02])

$$\phi_{,r} = \frac{2^{1+(\beta-\gamma)/\alpha}}{2-\gamma} \, \kappa_b \, r_b \left(\frac{r}{r_b}\right)^{1-\gamma} {}_2F_1\left[\frac{2-\gamma}{\alpha}, \frac{\beta-\gamma}{\alpha}; 1 + \frac{2-\gamma}{\alpha}; -\left(\frac{r}{r_b}\right)^\alpha\right], \tag{3.46}$$

where $\kappa_b = \Upsilon I_b/\Sigma_{\rm cr}$ is the surface mass density at the break radius in units of the critical density for lensing, and ${}_2F_1$ is the hypergeometric function. If the stellar distribution has ellipsoidal symmetry,

the lensing deflection must be computed numerically. *Note: At present the code does not have the ability to specify all of the Nuker parameters, so all* `nuker` *models have* $r_b = 1$.

An **exponential disk** viewed in projection has elliptical symmetry, but the elliptical model cannot be solved analytically. However, it can be approximated fairly well as a sum of Kuzmin disks (see below),

$$\kappa = q^{-1}\,\kappa_0\,\exp\left[-\xi/R_d\right] \simeq \sum_{j=1}^{11} q^{-1}\,\kappa_j\,s_j^3\,\left(s_j^2+\xi^2\right)^{-3/2} \tag{3.47}$$

where the weights and scale lengths are given in Table 3.4. The code uses this approximate solution.

j	κ_j/κ_0	s_j/R_d
1	0.008079	0.110297
2	−0.028496	0.156972
3	0.045732	0.239749
4	−0.027869	0.383171
5	0.287620	0.647549
6	−0.628801	1.030629
7	2.048180	1.599728
8	0.563387	2.553228
9	−1.415574	3.833604
10	0.042244	6.305107
11	0.105497	9.816998

Table 3.4 – Weights and scale lengths used for approximating an exponential disk as a sum of Kuzmin disks.

Chapter 4

Basic Lensing Calculations: The `gravlens` Kernel

This chapter describes `gravlens`, which is not only a stand-alone application for basic lensing calculations but also a kernel around which more sophisticated applications are built (e.g., `lensmodel` in Chapter 6). In other words, the `gravlens` application provides the foundation for all of the applications in the full `gravlens` package. Before reading this chapter, review the conventions of the code and manual in §1.2.

The heart of the entire code package is a fully general algorithm for solving the lens equation. The algorithm, described in the companion paper ([Kee01b]), involves tiling the image and source planes and using the tiles to determine the number and approximate positions of all lensed images associated with a given source. It requires no assumptions about the symmetry of the lensing potential, so it can be used with aribtrarily complicated mass models. For (nearly) any calculation you must specify the details of the tiling (§4.1) and the parameters of the mass model (§4.2). If your mass model requires numerical integrals, you may wish to speed up the calculations by using interpolation tables (§4.3). You can control the numerical solution of the lens equation (§4.4), including ways to handle "catastrophic" images (§4.5). For applications that require an assumption about the cosmology, you can specify cosmological parameters and the source and lens redshifts (§4.6). There are a few more miscellaneous parameters (§4.7). After specifying the parameters that control how the code operates, you can use the code to perform tests of its numerical accuracy (§4.8), to make plots of lensing quantities (§4.9), and to perform simple lensing calculations like determining the lensed images of a given sources or computing microlensing light curves (§4.10). Step-by-step examples are given in §4.11.

4.1 Specifying the tiling

The first task is to specify the tiling of the image and source planes. The code lays down a polar grid in the image plane centered on the main lens galaxy. It takes the points to be vertices of the image plane tiles, and then maps these to the source plane. You can use a set of variables to specify

the properties of the image plane grid. You can also select between two gridding modes based on what knowledge you have of the morphology of the critical curves.

4.1.1 Variables

`rscale [1.0]`	Radial scale of image plane grid. Used mainly in the `lensmodel` application (see §6.2).
`gridlo1 [0.0]`, `gridhi1 [2.0]`	Radial extent of image plane grid: lower and upper limits (in units of `rscale`, see §6.2).
`gridlo2 [0.0]`, `gridhi2 [360.0]`	Angular extent of image plane grid: lower and upper limits (as astronomical position angles, measured in degrees East of North).
`ngrid1 [20]`, `ngrid2 [20]`	Dimensions of the top grid, in radius and angle.
`nsubg1 [2]`, `nsubg2 [2]`	Dimensions of each subgrid, in radius and angle.
`gridflag [1]`	Flag for whether or not to compute the grid. (A few calculations do not require the grid, so omitting it saves time.)
`checkgaps [1]`	Flag for whether to look for images in the gaps in the source plane (see the companion paper [Kee01b], especially Figure 4). Checking the gaps requires 2×2 subgrids.
`maxlev [3]`	Maximum level of subgrid recursion near critical lines. The top grid is level 1, the first subgrid is level 2, and so on.
`gallev [3]`	Maximum level of subgrid recursion near galaxies other than the primary lens galaxy.
`imglev [3]`	Maximum level of subgrid recursion near images. Valid only when *not* using the catalog option (see below).
`crittol [1.0e-6]`	Tolerance for finding critical curves.

4.1.2 Commands

`gridmode` ⟨*mode*⟩

Specifies the type of tiling to use (see below). The default is a standard polar grid (mode 1).

`autogrid` [*mode*] [μ_{max}]

Automatically sets the grid range to encompass the full multiply-imaged region. Finds the critical radii and caustics, then sets `gridhi1` to be the radius of the smallest circle in the image plane where everything is singly-imaged. Uses the specified grid parameters, subject to the mode argument: (1) linear spacing for finding critical radii and for final grid; (2) logarithmic spacing for finding critical radii, but linear spacing for final grid; (3) logarithmic

spacing for finding critical radii and for final grid. If the optional μ_{max} argument is set, it finds the smallest circle where all images have magnification $\mu < \mu_{max}$.

4.1.3 Discussion

The code always uses a polar grid centered on the main lens galaxy. The default values of the range and resolution variables were chosen to work for most simple calculations, but you may want to change them depending on your problem. Note that if you use a singular lens model such as a point mass, you must explicitly set `gridlo1` to be non-zero in order to avoid the central singularity. (This is not a problem with other "singular" lens models because the code automatically imposes a small but finite core radius.) The code offers two modes for determining where to put grid zones:

- `gridmode 1`: A standard polar grid, with zones spaced equally in radius and angle. Recursive sub-gridding is used to increase the resolution near the critical lines. A sample standard grid with two levels of sub-gridding is shown in Figure 4.1. (Note that the critical curves are found within tolerance `crittol` only when they are plotted with `plotcrit`, see §4.9.)

- `gridmode 2`: A "critical curve" grid, in which the critical curves are used to determine where to place the radial zones. This mode is useful if you need very good resolution of the critical curves and you know that they have a relatively simple morphology. Suppose you know that there are two critical curves, a tangential critical curve $r_{tan}(\theta)$ and a radial critical curve $r_{rad}(\theta)$, and each curve is a simple loop with only one value of r for every θ. At each angle θ, the code uses numerical root finding to determine $r_{rad}(\theta)$ and $r_{tan}(\theta)$ within tolerance `crittol`. It then uses a set of radial zones equally spaced between `gridlo1` and $r_{rad}(\theta)$, another set between $r_{rad}(\theta)$ and $r_{tan}(\theta)$, and a third set between $r_{tan}(\theta)$ and `gridhi1`. The cost is the extra computation for the numerical root finding, but the benefit is much better resolution near the critical curves without sub-gridding. A sample "critical curve" grid is shown in Figure 4.1.

The code uses recursive sub-gridding to increase the resolution near important locations. There are two types of important locations. One is near the critical curves, because that is where the lens mapping folds over on itself (see the companion paper [Kee01b], especially Figure 3). The code generates a subgrid in any tile that contains a critical curve (where the magnification μ changes sign across the tile), to a maximum depth given by `maxlev`. The other important location is near any galaxy other than the main lens galaxy, because the extra galaxy may have its own critical curves that need to be resolved. The code generates a subgrid in any tile that contains a galaxy other than the main lens galaxy, to a maximum depth given by `gallev`. (The top grid is a polar grid centered on the main lens galaxy, so its core is automatically well resolved.) The dimensions of each subgrid are given by `nsubg1` and `nsubg2`; you should probably leave these at the default values (`nsubg1 =`

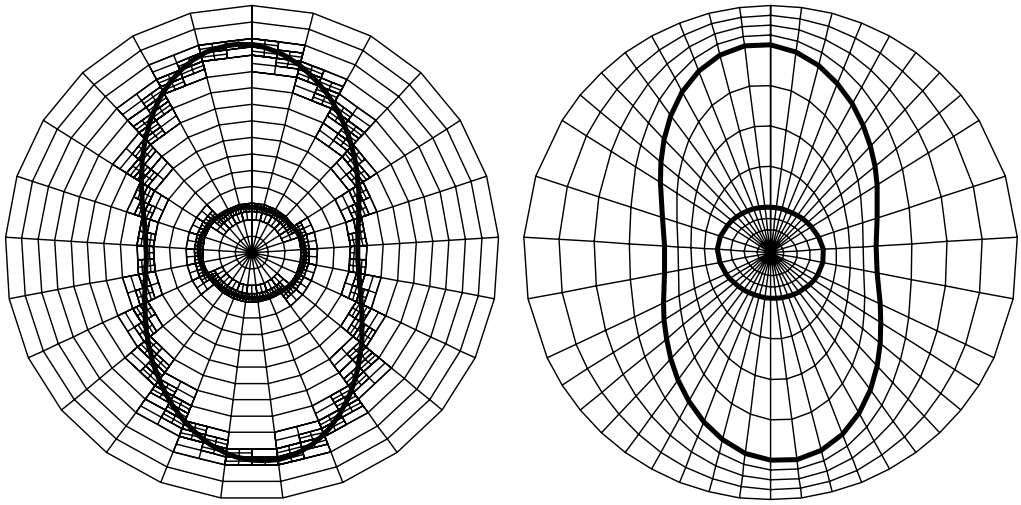


Figure 4.1 – Sample image plane grids. The light lines indicate the grid, while the heavy curves indicate the critical curves. *Left*: a standard polar grid with two levels of sub-gridding to increase the resolution near the critical curves. *Right*: an equivalent "critical curve" grid in which the code uses the critical curves to determine the locations of the radial grid zones.

`nsubg2 = 2`), because that allows the code to handle gaps in the source plane grid created by the sub-gridding (see the companion paper [Kee01b], §3).

The code can also use sub-gridding to increase the resolution in regions that it searches for images, to prevent the (unlikely) situation of having two images within a single tile. The variable `imglev` gives the maximum depth of the grid near images. Note that sub-gridding near images works only when you are *not* using the source catalog to find images (see §4.4). (When the catalog is used, once the catalog is generated it cannot be modified to add subgrids.)

The `autogrid` command is useful when examining lens models with very different scales, because it ensures that the grid covers the full multiply-imaged region. That way you don't have to know *a priori* the size of the critical curves and caustics.

4.1.4 Examples

The following commands specify the standard polar grid in Figure 4.1.

```
gridmode 1
set ngrid1 = 15
set ngrid2 = 25
set maxlev = 3
startup 1 1
  alpha 1.4 0.0 0.0 0.3 10.0 0.1 -12.1 0.2 0.0 1.0
```

```
  0 0 0 0 0 0 0 0 0 0
plotcrit mode1.crit
plotgrid mode1.grid
```

The following commands specify the "critical curve" grid in Figure 4.1.

```
gridmode 2
set ngrid1 =  5
set ngrid2 = 25
set maxlev = 1
startup 1 1
  alpha 1.4 0.0 0.0 0.3 10.0 0.1 -12.1 0.2 0.0 1.0
  0 0 0 0 0 0 0 0 0 0
plotcrit mode2.crit
plotgrid mode2.grid
```

4.2 Specifying the mass model

4.2.1 Variables

`NGALMAX [20]` Maximum number of galaxies allowed in a single model.

`galcoords [1]` Whether to use Cartesian (1) or polar (2) coordinates for galaxy positions. The position of the main lens galaxy is *always* given in Cartesian coordinates. Polar coordinates for any remaining galaxies are centered on the main lens galaxy.

`shrcoords [2]` Whether to use Cartesian (1) or polar (2) coordinates for angular structure parameters (shear and ellipticity). The polar angle is not quite standard, as explained below.

`intshrmode [1]` Specifies the mode in which the internal shear terms are defined (see §3.5)

`potflag [1]` Flag for whether or not to compute the lensing potential.

`nobflip [0]` Used only in the `lensmodel` application to control how the b parameter behaves in lens modeling (see §6.5).

`maxshear [1.0]` Maximum allowed value of the external shear; used mainly in the `lensmodel` application to prevent the shear from becoming unphysically large.

4.2.2 Commands

`startup` ⟨*# gals per mass model*⟩ ⟨*# of mass models*⟩

Allows you to specify the mass model(s) by giving the number of models, the number of galaxies in each model, and the parameters for all the galaxies.

`startup` ⟨*name of startup file*⟩

Allows you to specify the mass models(s) from a file.

`listmodels`

Lists and explains all mass models defined in the code.

4.2.3 Discussion

As discussed in Chapter 3, the code uses composite models made up of multiple "galaxies." Each galaxy is represented by one of the canonical models and is specified by up to 10 parameters. Thus a complete specification of a mass model includes the number of galaxies (N_{gal}) together with the parameters for each galaxy. The maximum number of galaxies you can use in a single model is specified by the variable `NGALMAX`.

For generality, `startup` allows you to specify more than one mass model, for example to feed several initial models into an optimization routine; the only command in `gravlens` that uses more than one of the `startup` models is `checkmod`. (§6.6 discusses how `lensmodel` uses the additional models.)

The `startup` command is structured as follows:

```
> startup <Ngal> <Nmod>
    <model type for model #1, galaxy #1>  <10 params for this galaxy>
    <model type for model #1, galaxy #2>  <10 params for this galaxy>
    ...
    <model type for model #2, galaxy #1>  <10 params for this galaxy>
    <model type for model #2, galaxy #2>  <10 params for this galaxy>
    ...
    <flags; see below>
```

See §3.2 and Table 3.2 for a summary of the canonical models and the parameters they use. See below for some examples. Note that some of the canonical models do not use all 10 parameters, but you must still give all 10. The `flags` mentioned here are included for the sake of the `lensmodel` application; they consist of a string of integers (0 or 1) which specify whether a particular parameter is allowed to vary when fitting a model to an observed lens (see §6.1). For `gravlens` runs they are not used, but they must be included.

As an alternative, you can place the mass model specification in a file and read it with the command

```
> startup <file>
```

Here the startup file must contain the following:

```
<Ngal> <Nmod>
<model type for model #1, galaxy #1>  <10 params for this galaxy>
<model type for model #1, galaxy #2>  <10 params for this galaxy>
...
<model type for model #2, galaxy #1>  <10 params for this galaxy>
<model type for model #2, galaxy #2>  <10 params for this galaxy>
...
<flags as above>
```

Calling `startup` not only specifies the mass model, it also produces the initial tiling of the image and source planes. Hence any variables that affect the tiling should be modified only *before* calling `startup`; see §4.1.

The position of the main lens galaxy is *always* given in Cartesian coordinates. The positions of any remaining galaxies, however, can be given either in Cartesian coordinates or on polar coordinates centered on the main lens galaxy. Use the `galcoords` flag to choose the coordinates.

The angular structure (ellipticity and shear) parameters can be given either in Cartesian coordinates or (modified) polar coordinates. For the ellipticity, the Cartesian coordinates (e_c, e_s) and the polar coordinates (e, θ_e) are related by

$$e_c = e \cos 2\theta_e \tag{4.1}$$

$$e_s = e \sin 2\theta_e \tag{4.2}$$

There are similar relations for the shear parameters (γ_c, γ_s) and (γ, θ_γ). The factor of 2 means that the angle is not a standard polar angle, but it is inserted so that the shear angle θ_γ points toward the mass concentration producing the shear.

The variable `potflag` specifies whether or not to compute the lensing potential ϕ. Often computing ϕ requires evaluating a numerical integral even for models with an analytic deflection (see Chapter 3). Many lensing calculations do not require the potential itself, so setting `potflag=0` can save calculation time. Note that you must set `potflag=1` if you want to compute lensing time delays.

4.2.4 Examples

To specify a model that has a single galaxy represented by a singular isothermal ellipsoid with ellipticity $e = 0.4$ and PA $= 78.1°$ you would use:

```
> startup 1 1
    alpha 1.1 -0.3 0.2 0.4 78.1 0.0 0.0 0.0 0.0 1.0
    0 0 0 0 0 0 0 0 0 0
```

If the model requires not only that galaxy but also a nearby cluster with an NFW profile, you might use:

```
> set galcoords = 1
> startup 2 1
    alpha 1.1 -0.3 0.2 0.4 78.1 0.0 0.0 0.0 0.0 1.0
    nfw   0.6 10.9 1.6 0.0  0.0 0.0 0.0 6.8 0.0 0.0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
```

You could also put this information in an input file. For example, you could create `foo.start` containing

```
2 1
alpha 1.1 -0.3 0.2 0.4 78.1 0.0 0.0 0.0 0.0 1.0
nfw   0.6 10.9 1.6 0.0  0.0 0.0 0.0 6.8 0.0 0.0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

and then load this file with the command

```
> set galcoords = 1
> startup foo.start
```

4.3 Using tabulated models

For mass models that do not have analytic solutions, the code can compute the lensing properties using numerical integrals (see Chapter 3). Because this can be time-consuming, the code offers the ability to tabulate the numerical solutions and then interpolate in them to obtain much faster approximate solutions. This section discusses how to use tabulated models; a sample run in §4.11 illustrates the technique and shows how to check the accuracy of the interpolated solutions.

4.3.1 Variables

(None.)

4.3.2 Commands

`maketab` ⟨*file*⟩ ⟨*e lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨*r lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨θ *steps*⟩

Compute the lensing properites on the specified grid of ellipticity, radius, and angle, and write them to the specified file. The output file is a binary file.

`loadtab` ⟨*file*⟩

Load lensing properties tabulated with the `maketab` command.

4.3.3 Discussion

The implementation of tabulated models rests on two assumptions:

- The mass model has a single scale length s so that the surface mass density has the form $\kappa(x,y;s) = \kappa(x/s, y/s)$. Then in general the potential has the form $\phi(x,y;s) = s^2\, f(x/s, y/s)$. This makes it possible to tabulate the lensing properites for a single value s and then trivially rescale to any other value s':

$$\begin{aligned}
\phi(x,y;s') &= \phi\left(\frac{xs}{s'}, \frac{ys}{s'}; s\right) \times \left(\frac{s'}{s}\right)^2 \\
\phi_{,i}(x,y;s') &= \phi_{,i}\left(\frac{xs}{s'}, \frac{ys}{s'}; s\right) \times \left(\frac{s'}{s}\right) \\
\phi_{,ij}(x,y;s') &= \phi_{,ij}\left(\frac{xs}{s'}, \frac{ys}{s'}; s\right)
\end{aligned} \tag{4.3}$$

- The surface mass density scales linearly with a mass parameter $\mathcal{M}$ so that $\kappa(x,y;\mathcal{M}) = \mathcal{M}\bar{\kappa}(x,y)$. Then all of the lensing properties also scale linearly with $\mathcal{M}$, so it is possible to tabulate them for a single value $\mathcal{M}$ and then rescale to any other value $\mathcal{M}'$.

With these scalings plus the ability to arbitrarily translate and rotate the model, we can obtain full generality by tabulating the lensing properties as functions of position and ellipticity only. (This breaks down for models with two scale lengths, but so far I have not encountered any models with two scale lengths that need numerical solutions.)

The `maketab` command tabulates the lensing properties on a grid of ellipticities and a polar grid of positions and saves the results in a binary file. Both grids can be made either linear or logarithmic; my beta testers recommend using linear grids in ellipticity and polar angle, and a logarithmic radial grid. The `loadtab` command reads the properties from a file written with `maketab` and prepares to use them for interpolation.

There are two ways to use the tabulated results. The code offers a general `tab` model class, for which it interpolates in the lensing properties that were loaded with the *first* `loadtab` command. The code also offers special "tabulated" model classes for some of the models (namely `alphaT`, `nfwT`, `cuspT`, and `devaucT`; see Table 3.2 for definitions). The two approaches are roughly equivalent, with the following caveats:

- You must use the `alphaT` and `devaucT` model classes for tabulated softened power-law and de Vaucouleurs models, respectively, because their scaling is a little different from eq. (4.3) above.

- You must use the special model classes if you use more than one tabulated model at a time, so the code selects properly among the tabulated models you have loaded.

Except for these exceptions, the `tab` model class is general and thus useful for using tabulated results with models other than those specially defined in the code.

Note that the tabulated model results must be loaded *before* any `startup` command that uses tabulated models.

4.3.4 Examples

Here is an example of a how to produce the tabulated results for a de Vaucouleurs model:

```
startup 1 1
  devauc 1.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 1.0
  0 0 0 0 0 0 0 0 0 0
maketab devauc.tab 0.0 0.1 3 0.01 10.0 -101 101
```

Here is how you would specify to use the tabulated results:

```
loadtab devauc.tab
startup 1 1
  devaucT 5.2 0.1 -0.3 0.05 40.0 0.0 0.0 4.0 0.0 0.0
  0 0 0 0 0 0 0 0 0 0
```

An sample run in §4.11 shows how you could check the accuracy of the lensing properties interpolated from the tabulated results.

Here is how you could use the general `tab` model class with an NFW model. First tabulate the NFW model results:

```
startup 1 1
  nfw 1.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 1.0
  0 0 0 0 0 0 0 0 0 0
maketab nfw.tab 0.0 0.2 5 0.01 10.0 -101 101
```

Now load and use the tabulated results:

```
loadtab nfw.tab
startup 1 1
  tab 6.1 -1.8 0.9 0.15 -31.8 0.0 0.0 3.1 0.0 0.0
  0 0 0 0 0 0 0 0 0 0
```

(Once again, you cannot use the `tab` model class with softened power law or de Vaucouleurs models because they have non-standard scalings.)

You can use multiple tabulated models at the same time, but you must use the specially defined model classes so the code knows which set of tabulated results to use for each model. Suppose you tabulated de Vaucouleurs and NFW results as above. You could use them together as follows:

```
loadtab devauc.tab
loadtab nfw.tab
startup 2 1
  devaucT 5.2 0.1 -0.3 0.05 40.0 0.0 0.0 4.0 0.0 0.0
  nfwT 6.1 -1.8 0.9 0.15 -31.8 0.0 0.0 3.1 0.0 0.0
  0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0
```

4.4 Solving the lens equation

4.4.1 Variables

`xtol [1.0e-6]` Tolerance on image positions in numerical root finding.

`catalog [0]` Flag for whether to use a catalog of source tiles.

4.4.2 Commands

(None.)

4.4.3 Discussion

The code uses the tilings of the image and source planes to make sure it finds all of the images of a given source **u**; see the companion paper ([Kee01b]). There are two ways of searching the source plane tiling to find all of the tiles that cover **u**:

- `catalog=0`: For each source position **u**, the code searches the *entire* source plane tiling to look for the tiles that cover **u**.
- `catalog=1`: Before searching for any images, the code first scans the source plane tiling and builds a catalog indicating which tiles cover which source positions. It can then use the catalog to quickly find the tiles that cover any given source position **u**.

The cost of using the catalog is the time required to build it, but the benefit is the ability to solve the lens equation without searching the entire tiling for every source. The time to build the catalog is comparable to the time to search the entire tiling a few times. Thus, if only a few source positions are examined for a given lens model (such as when you model an observed lens with a single source), using the catalog might slow you down. If, however, a large number of sources positions must be examined for each lens model (such as when you compute microlensing light curves, lensing cross sections, and so forth), using the catalog will be *much* faster.

Once it identifies the tiles containing images, the code uses a 2-d numerical root finder to solve the lens equation and find the image positions **x** corresponding to a given source position. The `xtol`

variable specifies the position tolerance in the root finder. The numerical algorithm is relatively efficient, so using a small tolerance (such as 10^{-6}) does not require excessive computation time.

4.5 Handling catastrophic images

4.5.1 Variables

`omitcore [0.0]` Size of omitted regions around galaxies.

`omitcrit [500.0]` Magnification of omitted band around critical curves.

4.5.2 Commands

(None.)

4.5.3 Discussion

The code uses a recursive tiling scheme to obtain good resolution near important regions of the lens mapping (see the companion paper, [Kee01b]), but even so the resolution is always finite. This fact can cause the code to have trouble with two types of images that lie near critical curves.

The first troublesome images are those in the cores of lens galaxies. A general theorem states that any smooth mass distribution must have an odd number of images (see [SEF92], §5.4). In most cases, however, one of the images lies near the high-density core of the lens galaxy and is highly demagnified, which explains why observed lenses usually have just two or four images (see [RM01] for a recent analysis). In the code, the only mass model *not* susceptible to the odd-image theorem is a point mass (see Chapter 3); for isothermal and related power law models, the code imposes a small but finite core radius to keep things well-behaved. As a result, in its default mode the code always looks for and uses the faint "core" image(s). The finite resolution of the tiling can cause the code to get hung up and waste a lot of time trying to pin down such images. If you get a lot of warning messages about a "non-converged image," this might be the problem.

You often don't care about core images because they are so rarely observed, and it is a shame to waste run time on something that is irrelevant. So the code offers a way to exclude core images from consideration. The variable `omitcore` tells the code to create a small region around the core of each lens galaxy that is *never* searched for images. In other words, the code simply ignores any model images within `omitcore` of a galaxy. (This variable also affects the behavior of `lensmodel` by telling the code to ignore "observed" core images; see §6.2.)

The other troublesome images are phantom images near critical curves. The finite resolution of the tiling means that the folding of the image plane at fold caustics is not perfectly resolved. This can lead the tiling algorithm to mistakenly think that there is (at least one) image extremely

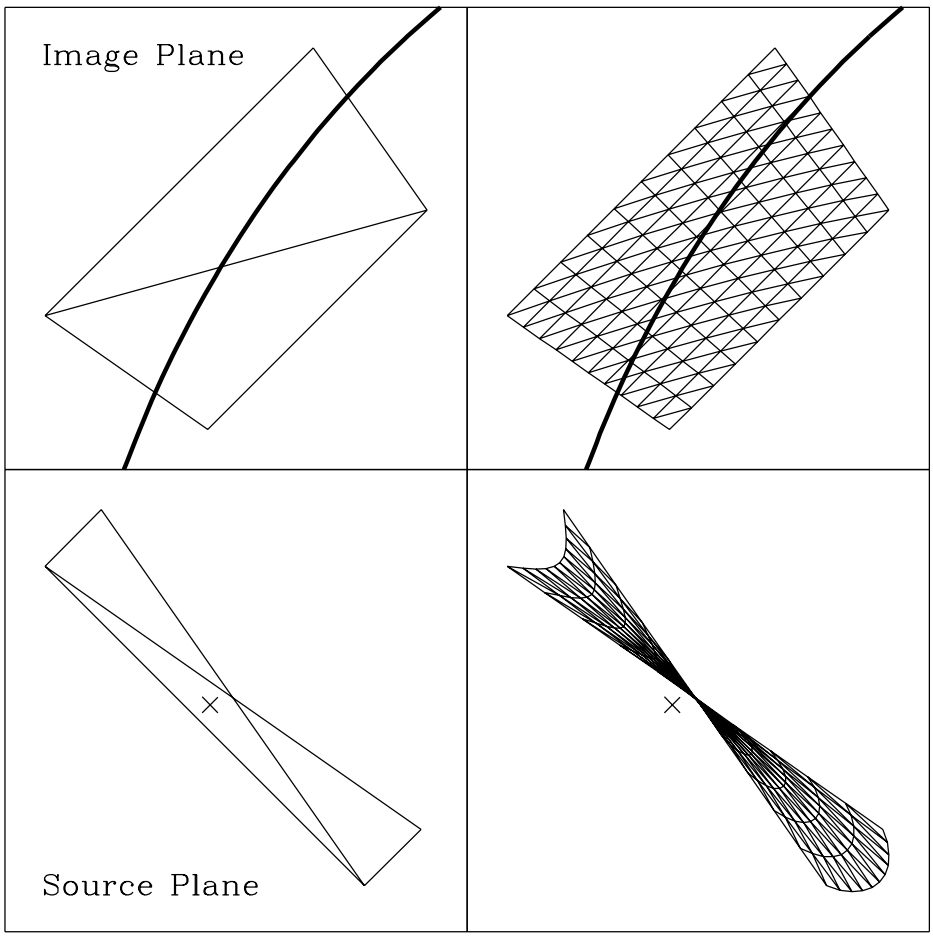


Figure 4.2 – An explanation of phantom images. The light lines are portions of tilings, the heavy lines are portions of critical curves, and the cross in the source plane is a sample source. With a low-resolution tiling (left), the fold causes the source plane tiles to overlap, and *both* of them cover the sample source. Hence the tiling algorithm thinks that there should be two images near the critical curve. With a higher-resolution tiling (right), the fold is much better resolved and the putative images are revealed to be phantoms.

close to a critical curve, when indeed there is none; see Figure 4.2 for an explanation. The code then wastes time searching for a phantom image, only to decide that it does not exist. (This is the other situation that produces a warning about a "non-converged image.") Because phantom images always occur very close to a critical curve, the code defines a thin, high-magnification band around each critical curve that is never searched for images. The variable `omitcrit` specifies the magnification threshold: the code does not search regions of the image plane where the magnifiation is higher than `omitcrit`.

4.6 Specifying the cosmology

4.6.1 Variables

`omega [1.0]`, `lambda [0.0]` — Cosmological parameters Ω_M and Ω_Λ.

`hval [1.0]`, `hvale [0.1]` — Hubble constant and its uncertainty, in terms of the parameter $h = H_0/(100 \text{ km s}^{-1}\text{Mpc}^{-1})$.

`zlens [-1]`, `zsrc [-1]` — The lens and source redshifts, which are used to compute the cosmology-dependent time delay factor t_0. A negative value means that the redshift is not specified.

`tscale [1.0]` — The cosmology-dependent time delay factor t_0 (see eq. 24 in [Kee01b]). Supersedes `zlens` and `zsrc`.

4.6.2 Commands

(None.)

4.6.3 Discussion

At present the cosmology information is used only to compute time delays. In `gravlens`, time delays are used only in `plottdel` (see §4.9); they are used more extensively in `lensmodel` (see Chapters 5 and 6).

4.7 Miscellaneous

4.7.1 Variables

`seed [-15]` — Seed for the random number generator. It should be a negative integer.

`maxrgstr [5]` — Maximum number of registration definitions. Used in `lensmodel`, but not used in `gravlens` (see §6.3).

`inttol [1.0e-6]` — Tolerance for numerical integrals. It is a relative (absolute) error tolerance — *per step* in the integration variable — for quantities greater than (less than) unity.

4.7.2 Commands

`version`

Displays the current version of the software.

4.8 Checking the code

4.8.1 Variables

(None.)

4.8.2 Commands

`checkder` ⟨*outbase*⟩ ⟨*rmin*⟩ ⟨*rmax*⟩ ⟨*Nsamp*⟩ [*h*]

Uses numerical derivatives to test whether the potential, deflection, and magnification of a model are self-consistent. The code picks `Nsamp` random points in an annulus between radii `rmin` and `rmax` (centered on the main lens galaxy). To test point (x, y) it compares the model deflection $\phi_{,x}(x, y)$ with the numerical derivative $[\phi(x+h/2, y)-\phi(x-h/2, y)]/h$, plus analogous techniques for $\phi_{,y}$ and the magnification components $\phi_{,ij}$. It writes the results to the file *outbase*`.dat`, including the absolute and relative errors for $\phi_{,i}$ and $\phi_{,ij}$. It then summarizes the results into error histograms (fraction of points versus log(err)) and writes the histograms to *outbase*`.abs` and *outbase*`.rel`. There are histograms for each component $\phi_{,i}$ and $\phi_{,ij}$, and for the average over all six components. Finally, the code makes a crude text plot to show you the average histograms.

`checkmod` ⟨*outbase*⟩ ⟨*rmin*⟩ ⟨*rmax*⟩ ⟨*Nsamp*⟩

Similar to `checkder`, except that it checks the potential, deflection, and magnification for one model against another model. Used to check whether two models have the same lensing properties, for example to test the accuracy of interpolated lensing properties (see `maketab` and `loadtab`) against the numerical integrals. The output is the same as `checkder`. (An example is given in §4.11.)

`checkmag` ⟨*outname*⟩ ⟨*δu lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩

Solves the lens equation and checks the total magnification against an analytic solution. Used to test how well the code performs near a cusp. If the cusp is at position u_c, the code checks sources at $u_c \pm \delta u$ for the specified range of δu. The output is written to the file *outname*. Full analytic solutions exist only for SIS+shear or ptmass+shear lens models with the shear along the axes.

4.9 Making pictures

4.9.1 Variables

(None.)

4.9.2 Commands

`plotgrid` ⟨*file*⟩ [*mode*]

Writes image and source plane grids to the specified file. Comments at the top of the file explain how to plot the grids. The `mode` flag indicates whether to show quadrilaterals (1) or triangles (2). The tiling algorithm lays down quadrilaterals and breaks each into two triangles because triangles are the only polygons that are guaranteed to remain convex when mapped from the image plane to the source plane; see the companion paper [Kee01b]. Plotting the quadrilaterals shows essentially all of the grid information with less clutter than the triangles. Most of the figures in this manual show just the quadrilaterals.

`plotcrit` ⟨*file*⟩

Writes critical curves and caustics to the specified file. Comments at the top of the file explain how to plot the critical lines. The code uses an algorithm to connect the dots in a way that should work even for critical curves with complicated morphologies, for example multiple galaxies with disjoint critical curves. (Basically, the code tries to connect nearest neighbor points on the critical curves, and maps the critical curve connections to the caustics.) However, the algorithm is not fool-proof, and you may occasionally get bizarre results. These will be obvious by having connections between points that clearly should not be connected.

`plotkappa` ⟨*file*⟩ ⟨*file type*⟩ ⟨*x lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨*y lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩

Plots a κ map. The file type is as follows: (1) plain text; (2) binary; (3) FITS.

`plotdef0` ⟨*file*⟩ ⟨*r lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨θ *steps*⟩

Plots the monopole deflection versus radius. It uses the specified number of θ steps to compute the average over azimuth.

`plotdef1` ⟨*file*⟩ ⟨*x lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨*y lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩

Computes the lensing properties (potential, deflection, and magnification) on the specified Cartesian grid and writes them to a file for plotting. The output file begins with the limits of the grid, followed by a list of $(x; y; \phi; \phi_{,x}; \phi_{,y}; \phi_{,xx}; \phi_{,yy}; \phi_{,xy}; \phi_{,yx})$ for each point. In this list, the x loop is outside the y loop:

```
for (ix=1;ix<=nx;ix++) { for (iy=1;iy<=ny;iy++) { <ix,iy> } }
```

`plotdef2` ⟨*file*⟩ ⟨θ *lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨*r lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩

The same as `plotdef1`, except on a polar grid. The output is still $\phi_{,x}$ and $\phi_{,y}$, etc. (as opposed to $\phi_{,r}$ and $\phi_{,\theta}$), but the quantities are computed at particular values of (r, θ) rather than (x, y). In the output file the θ loop is outside the r loop:

```
for (ith=1;ith<=nth;ith++) { for (ir=1;ir<=nr;ir++) { <ith,ir> } }
```

`plottdel` ⟨*file*⟩ ⟨*src u*⟩ ⟨*src v*⟩ ⟨*x lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨*y lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩

Similar to `plotdef1`, but also computes the time delay surface for the specified source position. If `tscale` is set (see §4.6), the delays are given in units if h^{-1} days, assuming image positions in arcseconds. If `tscale` is not set, the delays are given in units of the time delay scale t_0.

`plotmag` ⟨*file*⟩ ⟨*u lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨*v lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩

Plots the number of images and the total magnification (for a point source) as a function of the source position (u, v). In the output file the u loop is outside the v loop:

```
for (iu=1;iu<=nu;iu++) { for (iv=1;iv<=nv;iv++) { <iu,iv> } }
```

4.10 Simple lensing calculations

4.10.1 Variables

(None.)

4.10.2 Commands

`calcRein` [*mode*] [*file*]

Computes Einstein radii for *spherical* models. Mode 0 means use the startup model (default). Mode N means use a parameter grid with N dimensions; after the `calcRein` command you must give `igal`, `iparm`, `lo`, `hi`, and `steps` the each axis of the parameter grid. Writes results to the specified file (or to the screen as a default.

`findimg` ⟨*u*⟩ ⟨*v*⟩

Finds locations and magnifications of all images corresponding to a (point) source at position (u, v).

`findimg2` ⟨*u lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨*v lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨*outname*⟩

Finds the images corresponding to (point) sources on the specified Cartesian grid.

`findimg3` ⟨*inname*⟩ ⟨*outname*⟩

Finds the images corresponding to (point) sources listed in the input file.

`findsrc` ⟨*x*⟩ ⟨*y*⟩

Finds the source corresponding to an image at (x, y), then finds all the sister images corresponding to that source.

`ellsrc` ⟨*file*⟩ ⟨u_0⟩ ⟨v_0⟩ ⟨e⟩ ⟨*PA*⟩ ⟨*a lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨θ *steps*⟩

Places ellipses in the source plane and maps them to the image plane. The ellipses are centered at (u_0, v_0) and have ellipticity e and orientation PA. Ellipses are plotted with the specified range of semi-major axis a, using the specified number of angular steps. Comments at the top of the file explain how to plot the images.

`ellimg` ⟨*file*⟩ ⟨x_0⟩ ⟨y_0⟩ ⟨e⟩ ⟨PA⟩ ⟨a lo⟩ ⟨hi⟩ ⟨$steps$⟩ ⟨θ $steps$⟩

Places ellipses in the image plane, maps them to the source plane, and then finds all the images of those sources. The initial image plane ellipses are centered at (x_0, y_0) and have ellipticity e and orientation PA. Ellipses are plotted with the specified range of semi-major axis a, using the specified number of angular steps. Comments at the top of the file explain how to plot the images. *NOTE: Compare the* `ellsrc/ellimg` *pair to the* `findsrc/findimg` *pair. One command of each pair places objects in the source plane and maps them to the image plane, while the other places objects in the image plane, maps backwards to the source plane, and then maps forewards to the image plane again to find sister images. However, the* `ellsrc/ellimg` *pair is reversed relative to the* `findsrc/findimg` *pair.*

`SBmap1` ⟨*inname*⟩ ⟨*intype*⟩ ⟨u lo⟩ ⟨hi⟩ ⟨v lo⟩ ⟨hi⟩ ⟨*outname*⟩ ⟨*outtype*⟩ ⟨x lo⟩ ⟨hi⟩ ⟨$steps$⟩ ⟨y lo⟩ ⟨hi⟩ ⟨$steps$⟩

Reads a surface brightness map from the specified input file, puts it in the specified region of the source plane, then lenses it. The units are preserved.

`SBmap2` ⟨x lo⟩ ⟨hi⟩ ⟨$steps$⟩ ⟨y lo⟩ ⟨hi⟩ ⟨$steps$⟩ ⟨*outname*⟩ ⟨*outtype*⟩ ⟨*Nsrc*⟩ [*inname*]

Generates a lensed surface brightness map from an array of analytic sources. The resulting map has units of $L_\odot$ arcsec^{-2} (or something equivalent), so to compute the integrated flux you would sum the pixels and multiply by the area of each pixel.

`magtensor` ⟨x⟩ ⟨y⟩

Computes the magnification tensor at the specified point.

`mock1` ⟨*outname*⟩ ⟨*number of sources*⟩ [*minimum number of images*] [*bias mode*] [ν] [R_{max}]

Generates mock lenses by picking random sources and solving the lens equation. Uses the specified *number* of sources. The sources are distributed uniformly in the smallest circle circumscribing the caustics. The optional 3rd argument can be used to specify the minimum number of images required for a lens to be stored. The optional 4th and 5th arguments refer to the calculation of magnification bias (use `help mock1` for details). The optional 6th argument can be used to override the size of the search circle in the source plane.

`mock2` ⟨*outname*⟩ ⟨*density of sources*⟩ [*minimum number of images*]

Similar to `mock1`, but uses the specified *density* of sources.

`lightcrv` ⟨*file*⟩ ⟨u_1⟩ ⟨v_1⟩ ⟨u_2⟩ ⟨v_2⟩ ⟨# *steps*⟩

Computes the microlensing light curve (for a point source) as the source traverses the line from (u_1, v_1) to (u_2, v_2) in the specified number of steps. Writes the results to the specified file.

4.11 Sample runs

Here is a simple example of a run that defines a model with a singular isolated ellipsoid with mass parameter $b = 1$, centered at $(0.1, 0.2)$, with ellipticity $e = 0.1$ and PA $= 30°$. The run plots some lensing properties, then finds some images, and finally computes a light curve. This example illustrates running the code interactively.

```
unixprompt% gravlens
Creating rscale = 1.000000e+00 [Sets radial scale of grid]
Creating gridlo1 = 0.000000e+00 [Grid radius: lower limit (units of rscale)]
Creating gridhi1 = 2.000000e+00 [Grid radius: upper limit (units of rscale)]
Creating gridlo2 = 0.000000e+00 [Grid angle:  lower limit]
Creating gridhi2 = 3.600000e+02 [Grid angle:  upper limit]
Creating ngrid1 = 2.000000e+01 [dimension of top grid (radius)]
Creating ngrid2 = 2.000000e+01 [dimension of top grid (angle )]
Creating nsubg1 = 2.000000e+00 [dimension of sub grid (radius)]
Creating nsubg2 = 2.000000e+00 [dimension of sub grid (angle )]
Creating gridflag = 1.000000e+00 [Flag for whether to compute grid]
Creating omitcore = 0.000000e+00 [Size of omitted region around galaxies]
Creating omitcrit = 5.000000e+02 [Magnification of omitted band around crit crv]
Creating checkgaps = 1.000000e+00 [Flag for checking gaps in source plane grid]
Creating maxlev = 3.000000e+00 [Deepest level of subgrid recursion]
Creating gallev = 3.000000e+00 [Deepest level of subgrid near galaxies]
Creating crittol = 1.000000e-06 [Tolerance for finding critical curves]
Creating galcoords = 2.000000e+00 [1->cartesian, 2->polar]
Creating shrcoords = 2.000000e+00 [1->cartesian, 2->polar]
Creating maxrgstr = 5.000000e+00 [Maximum number of registration definitions]
Creating potflag = 1.000000e+00 [Compute lensing potential?]
Creating xtol = 1.000000e-06 [Tolerance for src2img]
Creating inttol = 1.000000e-06 [Tolerance for numerical integrals]
Creating catalog = 1.000000e+00 [Flag for using catalog of source tiles]
Creating seed = -1.500000e+01 [Seed for randum number generator]
Creating omega = 1.000000e+00 [Matter density Omega_Matter]
Creating lambda = 0.000000e+00 [Cosmological constant Omega_Lambda]
Creating hval = 1.000000e+00 [H_0 in units of 100 km/s/Mpc]
Creating hvale = 1.000000e-01 [Uncertainty in H_0 in units of 100 km/s/Mpc]
Creating tscale = 1.000000e+00 [The cosmology-dependent time delay factor]
Creating zlens = -1.000000e+00 [The lens redshift]
Creating zsrc = -1.000000e+00 [The source redshift]
> gridmode 2
Setting maxlev = 1.000000e+00
> startup 1 1
Enter galaxies for model #1:
  alpha 1.0 0.1 0.2 0.2 30.0 0.0 0.0 0.0 0.0 1.0
```

```
Enter what to vary for each galaxy:
  0 0 0 0 0 0 0 0 0 0
Note: looking for 2 critical lines.
> plotgrid foo.grid
Writing grid to foo.grid
> plotcrit foo.crit
Writing critical curves and caustics to foo.crit
> plotdef1 foo.def1 -1.0 1.0 21 -1.0 1.0 21
Writing lensing properties to foo.def1
> plotdef2 foo.def2 0.0 90.0 3 0.0 2.0 21
Writing lensing properties to foo.def2
> findimg 0.25 0.10
Source u =  2.500000e-01, v =  1.000000e-01
Found 3 images with positions and magnifications:
   9.996830e-02  2.000251e-01  4.633074e-08
  -4.573846e-01  7.733234e-01 -2.550845e+00
   1.263745e+00 -3.215508e-03  5.898054e+00
> findimg 0.15 0.20
Source u =  1.500000e-01, v =  2.000000e-01
Found 5 images with positions and magnifications:
   9.999041e-02  2.000010e-01  4.090783e-08
  -5.481404e-01  8.822342e-01 -5.416573e+00
   2.850334e-01 -7.716133e-01 -9.012801e+00
  -6.117861e-01 -5.108209e-01  1.124872e+01
   1.110033e+00  6.125100e-01  5.976329e+00
> lightcrv foo.lcrv -0.2 -0.6 0.4 0.3 100
Writing lightcrv to foo.lcrv
Step 0
Step 10
Step 20
Step 30
Step 40
Step 50
Step 60
Step 70
Step 80
Step 90
Step 100
> quit
unixprompt%
```

Alternatively, you could put all of the commands into an input file, for example `foo.in` containing:

```
gridmode 2
startup 1 1
```

```
  alpha 1.0 0.1 0.2 0.2 30.0 0.0 0.0 0.0 0.0 1.0
  0 0 0 0 0 0 0 0 0 0
plotgrid foo.grid
plotcrit foo.crit
plotdef1 foo.def1 -1.0 1.0 21 -1.0 1.0 21
plotdef2 foo.def2 0.0 90.0 3 0.0 2.0 21
findimg 0.25 0.10
findimg 0.15 0.20
lightcrv foo.lcrv -0.2 -0.6 0.4 0.3 100
quit
```

You could then run

```
unixprompt% gravlens foo.in
```

to obtain the same results as the interactive run.

Finally, here is an example of how to use tabulated models (`maketab` and `loadtab`) and to check the accuracy of the interpolated results (`checkmod`). Suppose you want to tabulate a `devauc` model because it does not have an analytic solution. First you tabulate the lensing properties with an input file such as:

```
set inttol = 1.0e-8
set gridflag = 0
startup 1 1
  devauc 1.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 1.0
  0 0 0 0 0 0 0 0 0 0
maketab devauc.tab 0.0 0.1 3 0.01 10.0 -101 101
quit
```

(This input file shows how to use a negative value for `r steps` to obtain a logarithmic grid.) Now you do a run in which you load the tabulated results and then test the accuracy of the interpolation:

```
loadtab devauc.tab
set inttol = 1.0e-8
set gridflag = 0
startup 1 2
  devauc  5.2 0.1 -0.3 0.05 40.0 0.0 0.0 4.0 0.0 0.0
  devaucT 5.2 0.1 -0.3 0.05 40.0 0.0 0.0 4.0 0.0 0.0
  0 0 0 0 0 0 0 0 0 0
checkmod devauc 0.0 10.0 1000
quit
```

(This input file shows how to use `startup` to enter two mass models with one galaxy each, `startup 1 2`, as opposed to a single mass model with two galaxies, `startup 2 1`.) When `checkmod` is finished it shows you a simple text plot of the error histograms:

```
Error histograms: fraction versus log(err)

              absolute                          relative
  1.0 |                             1.0 |
  0.9 |                             0.9 |
  0.8 |                             0.8 |
  0.7 |                             0.7 |
  0.6 |           *                 0.6 |
  0.5 |                             0.5 |           *
  0.4 |                             0.4 |              *
  0.3 |                             0.3 |
  0.2 |        *     *              0.2 |
  0.1 |                             0.1 |        *
  0.0 |--*--*-- -- -- --*--*--*--    0.0 |--*--*-- -- -- --*--*--*--
        <= -6 -5 -4 -3 -2 -1  =>           <= -6 -5 -4 -3 -2 -1  =>
```

Note that the first and last points in the plot refer to all errors outside the range of the histogram (i.e., $\leq 10^{-7}$ or ≥ 1). You may want to use this technique to determine the resolution of the ellipticity, radial, and angular grids you need to use in `maketab` in order to achieve the desired accuracy.

Chapter 5

Strategies for Modeling Strong Lenses

This chapter discusses general issues related to modeling strong lenses. Further discussion appears in the companion paper ([Kee01b]). A lens usually consists of some number of point-like images and/or extended images such as host galaxy arcs or an Einstein ring; the code allows you to use some or all of these data to constrain lens models:

- Point images provide constraints from relative positions, fluxes, and time delays (§5.1). The position constraints can sometimes be used to directly solve for some of the model parameters (§5.2). (This material is discussed in detail in the companion paper; the sections here merely review the main points.)
- Extended images provide numerous constraints that can help break common degeneracies in lens models, but some care is needed to use the constraints efficiently (§§5.3 and 5.4).

Several common degeneracies plague lens modeling, especially when using only point images; §5.5 discusses the degeneracies and ways to identify and understand them. This chapter has a general and theoretical bent; Chapter 6 describes how to use the `lensmodel` application to actually do the modeling, and Chapter 7 leads you through some sample `lensmodel` runs.

5.1 Point images

In many lenses there are multiple images of a given source, and the basic data are the positions and fluxes of the images. If there are measured time delays between the images, they can be used to determine the Hubble constant H_0 and to further constrain the lens models. See §4 of the companion paper ([Kee01b]) for a detailed discussion of how to use point image data to constrain lens models, including definitions of all of the χ^2 goodness-of-fit terms. This section briefly reviews the methods and discusses a few places where the code differs slightly from the discussion in the paper.

The χ^2 for the image positions can be evaluated either in the image plane or on the source plane; see §4.1 of the paper. The source plane χ^2 can be convenient because it does not require the code to

solve the lens equation, but it is only approximately valid as a measure of the ability of the model to fit the data. As a result, the source plane χ^2 is useful for initial modeling to find the appropriate region of parameter space to explore. However, for refining models to find the true best-fit model and the range of models consistent with the data, it is preferable to use the image plane χ^2. See the paper for more discussion, and Chapter 7 for examples. In the code, you can select which χ^2 definition to use with the variable `chimode` (see §6.6).

The use of image fluxes is discussion in §4.2 of the paper, but the code adds one additional feature: signed fluxes to represent the parities of the images. In general, different images of a given source have different partial parities. An image at a local minimum of the time delay surface has two positive partial parities, an image at a saddle point has one positive and one negative, and an image at local maximum has two negative partial parities. If we define the parity be the product of the two partial parities, standard image configurations have the following properties (e.g., [SEF92]):

- In a double lens, one image has positive parity and the other negative.
- In a quadruple lens, as you move around in azimuth around the main lens galaxy the parity always flips sign between images.
- (The faint central image has positive parity.)

These parities provide a very fast way to check whether a particular lens model is viable: if the model parities do not match the observed parities, you can immediately exclude the model without taking the time to solve the lens equation to compute the position and flux χ^2 contributions.

You can specify the parity of an image by putting a sign on the flux f: use $f_{obs,i} > 0$ for images with positive parity, and $f_{obs,i} < 0$ for images with negative parity. In its default mode, the code expects signed fluxes, and it does the parity check as the first step in evaluating models. However, when you first begin to model a lens you may not know what region of parameter space produces models with the right parity. If you pick a random region, you are likely to have models with the wrong parity; in this case the code will try a few models, decide that it cannot find any models with the proper parities, and just stop. To avoid this problem, you can use the `checkparity` variable to tell the code *not* to perform the parity check (see §6.2). In other words, turn the parity check off when you first begin modeling a lens, but once you have a reasonable model turn the parity check on to speed up the run.

Finally, if there are N time delays measured between the images, they can be used to determine the Hubble constant H_0 and to provide $N - 1$ additional constraints on the lens model. See §4.3 in the paper for a discussion of time delays. Note that the code assumes that the time delays have independent Gaussian errors, which may not be true; the paper discusses some possible consequences of this assumption.

5.2 Linear parameters and constraints

With reliable data it may be possible to solve for some of the parameters directly, without having to include them in the numerical optimization. See §4.3 in the paper for a detailed discussion of how to use "linear constraints" to solve for "linear parameters."

5.3 Curve fitting

The number of constraints provided by extended images, essentially the number of independent pixels, can far outnumber the constraints provided by discrete images. However, the constraints can be hard to use because you do not know *a priori* which parts of the extended images are supposed to map into each other. As a result, you must self-consistently build a model of the intrinsic source distribution. There are several algorithms for doing this (e.g., [KBLN89]; [KN92]; [WKN96]), but they are all computationally expensive.

There are two new fast techniques for modeling extended images: *curve fitting* applies to separated structures such as arcs, and is discussed in this section; *ring fitting* applies to partial or complete Einstein rings, and is discussed in §5.4.

The idea of curve fitting derives from the fact that surface brightness is conserved in lensing (see [SEF92]). Thus with extended arcs, contours of constant surface brightness in different arcs must be images of each other. If you take a point on a given contour, all other images of its source must lie on contours of the same surface brightness. Hence we can constrain the models by requiring that curves map into each other. In practice our curves are not strict surface brightness contours, if for no reason other than smearing by the PSF. However, the geometrical idea of matching curves is still useful because it incorporates constraints from thin arcs and slightly resolved images, without requiring the computational effort of building a full model of the intrinsic source. (A paper with a more detailed discussion and sample applications is in the works.)

The basic idea of matching two curves A and B is as follows. Take curve A, map it to the source plane to find the intrinsic source, and map that back to the image plane to find all the other images of the curve. One of those images, call it A', should be near curve B. A useful χ^2 is the perpendicular distance between A' and B, integrated along the length of the curves.

There are practical complications because the curves can have different lengths and different sampling intervals. To handle the different lengths, let the shorter curve be the "test" curve to be compared to the longer "reference" curve. To handle the different samplings of the curve, compare *points* on the test curve with *segments* of the reference curve. Consider a point $\mathbf{y}$ on the test curve and find the nearest segment of the reference curve; the geometry is shown in Figure 5.1a. Let $\mathbf{y}'$ be the projection of test point $\mathbf{y}$ onto the segment connecting reference points $\mathbf{x}_1$ and $\mathbf{x}_2$; simple

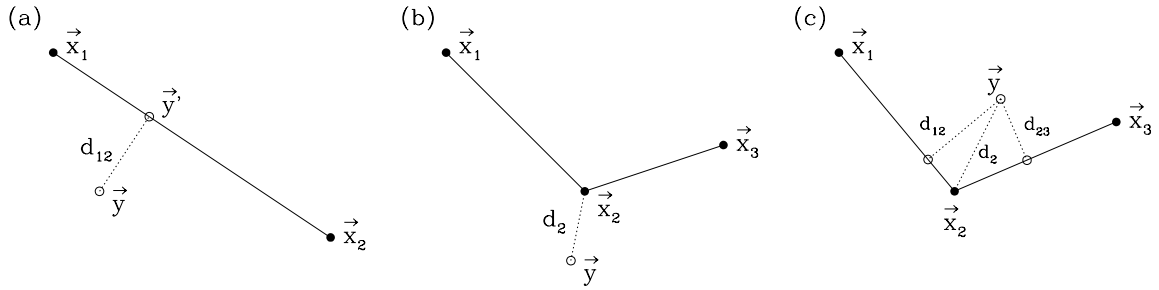


Figure 5.1 – Geometries for curve fitting. Panel (a) shows the simplest geometry for comparing a point $\mathbf{y}$ on the test contour with a segment $\mathbf{x}_1$–$\mathbf{x}_2$ of the reference contour. Panels (b) and (c) show complications that occur when $\mathbf{y}$ has n perpendicular projection onto the nearest contour segments (panel b), or when $\mathbf{y}$ is near a point and two contour segments (panel c).

geometry gives

$$\mathbf{y}'_{12}(\mathbf{y}) = [1 - \xi_{12}(\mathbf{y})]\ \mathbf{x}_1 + \xi_{12}(\mathbf{y})\,\mathbf{x}_2\,, \tag{5.1}$$

$$\text{where} \qquad \xi_{12}(\mathbf{y}) = \frac{(\mathbf{y} - \mathbf{x}_1)\cdot(\mathbf{x}_2 - \mathbf{x}_1)}{|\mathbf{x}_2 - \mathbf{x}_1|^2}\,. \tag{5.2}$$

There is a *proper* projection, meaning that the projection lies on the segment between the two references points, only if ξ is in the range $0 \le \xi \le 1$. In this case, the perpendicular distance from $\mathbf{y}$ to the line segment is then

$$d_{12}(\mathbf{y}) = |\mathbf{y} - \mathbf{y}'_{12}(\mathbf{y})|\,. \tag{5.3}$$

In addition to the simple case shown in Figure 5.1a, there are three possible complications. If the reference curve bends and the test point is outside the bend, there may be no proper projection (Figure 5.1b); while if the test point is inside the bend, there can be two proper projections (Figure 5.1c). Finally, if the test point lies beyond the end of the reference curve there will be no proper projection (not shown). All of these cases can be handled by considering not only projections onto segments, but also the distance from the test point to the nearest reference point. Thus the χ^2 contribution for test point j is

$$\chi^2_{test,j} = \min_i \left(\frac{|\mathbf{y}_j - \mathbf{x}_i|^2}{\sigma_i^2}, \frac{d_{i-1,i}(\mathbf{y}_j)^2}{\sigma_{i-1}\sigma_i} \right), \tag{5.4}$$

where i runs over the reference points, and for each segment we take the errorbar to be the geometric mean of the errorbars at the endpoints. If there are multiple reference curves, find the smallest contribution among all the curves. The total χ^2 for the curve constraints is just the sum over all the test points,

$$\chi^2_{crv} = \sum_j \chi^2_{test,i}\,. \tag{5.5}$$

This definition reverts to the standard image plane χ^2 for point sources if each "curve" consists of only a single point.

Sometimes the curve fitting algorithm will try to reduce the total χ^2_{crv} by changing the lens model to eliminate some points. For example, suppose you have an extended source where part of the source is doubly-imaged and part is quadruply-imaged. In some cases the code may be able to get a lower total χ^2_{crv} by making a qualitatively *poorer* fit that has *fewer* points. You can avoid this problem by using as the curve statistic $\bar{\chi}^2_{crv} \equiv \chi^2_{crv}/N_{crv}$ where N_{crv} is the number of points. You can specify this behavior using the variable `chiperpoint` (see §6.2).

There are a couple of limitations to this simple curve fitting technique. First, suppose the uncertainty σ varies greatly along the curve. The funny situation may arise where it may be better for the χ^2 if the test point is matched to a distant segment that has a large uncertainty, as opposed to a near segment with a small uncertainty. I don't know how likely this problem is, and I haven't tried to remedy it. Second, the technique described here does not use any information about the orientation of closed curves. Curve fitting will receive more attention when it is rigorously applied to an observed lens; please let me know if you have good ideas or applications!

5.4 Ring fitting

The ring fitting technique for modeling Einstein rings is introduced and applied to three readl lenses by [KKM01]. The basic idea is as follows. For an observed Einstein ring, create a series of spokes emanating from a point inside the ring (such as the center of the lens galaxy). If the reference point is (x_0, y_0), the spokes are

$$x(\lambda;\varphi) = x_0 - \lambda \sin\varphi\,, \tag{5.6}$$

$$y(\lambda;\varphi) = y_0 + \lambda \cos\varphi\,, \tag{5.7}$$

where φ is the azimuth angle of the spoke and λ is a parameter along the spoke. Along each spoke, find the location λ_0 of the peak surface brightness of the ring. The points (φ, λ_0) form a curve in the image plane that contains most of the information about the shape of the observed ring; henceforth I refer to this set of points as the ring. [KKM01] show that this ring depends only on the position and angular shape of the source, not on its radial profile. This result requires only the assumption that the radial profile is monotonic, which is true for the host galaxies that we expect to form optical and infrared Einstein rings. Hence the shape of the ring provides powerful constraints on the lens model and on the shape of the source.

There are two other types of constraints you can get from a ring. If you trace the peak surface brightness as a function of azimuth along the ring, local *maxima* are positions that map to the center of the source, while local *minima* are positions where the ring crosses the critical curve of the lens model.

The code allows all three types of constraints with a goodness of fit statistic

$$\chi^2_{ring} = \sum_{\varphi} \frac{(\lambda_{obs} - \lambda_{mod})^2}{\sigma^2_\lambda} + \sum_{\text{maxima}} \frac{|\mathbf{x}_{obs} - \mathbf{x}_{mod}|^2}{\sigma^2_x} + \sum_{\text{minima}} \frac{|\mathbf{x}_{obs} - \mathbf{x}_{mod}|^2}{\sigma^2_x} \ . \qquad (5.8)$$

Here λ_{obs} and λ_{mod} are the observed and model ring positions on each spoke φ, respectively, and σ_λ is the uncertainty in the observed value. Also, $\mathbf{x}_{obs}$ and $\mathbf{x}_{mod}$ are the observed and model positions of the flux maxima and minima along the ring, with position uncertainty σ_x.

See [KKM01] for a much more detailed discussion of ring fitting, together with sample modeling performed using the `lensmodel` code.

5.5 Common degeneracies

There are two types of degeneracies that often affect lens models. The first relates to the radial density profile of the lens galaxy. If the lensed images lie at approximately the same distance from the center of the galaxy, they constrain the total enclosed mass but not how that mass is distributed. As a result, the lens might be consistent with a range of density profiles (e.g., [Koc91a]; [WP94]; [KK97]). A good strategy for handling the profile degeneracy is to study and understand a set of models with a fixed profile, and then examine other profiles; see [KF99] and [CKMK01] for examples of this approach.

The second degeneracy relates to the angular structure of the lensing potential. The lensed images usually constrain the net quadrupole moment of the potential. However, the net quadrupole moment can have contributions from the ellipticity of the lens galaxy as well as tidal shear from the surrounding environment. Models with only one of these two compoents (ellipticity or shear, but not both) are well constrained but usually do not give good fits (e.g., [KKF98]). Adding the second component dramatically improves the fit, but there may end up being a significant degeneracy between the shear and the ellipticity (e.g., [KKS97], especially Figure 8). If you need to use both ellipticity and shear, it is important to understand whether there is a degeneracy between them, or whether your data are good enough to break the degeneracy.

Another possible degeneracy arises when you explicitly model the sources of external tidal perturbations. If the source of the perturbations is relatively close to the lens, then approximating the perturbations as an external shear may not be appropriate, and you may need to put down a mass distribution representing the perturber. In this case, the location and physical properties of the perturber may be poorly constrained. [KK97] present examples for PG 1115+080, although [IFK$^+$98] show that improved data reduce the degeneracies; [Cha99] gives examples for Q 0957+561. The best strategy here is to vary the properties of the perturber and understand how any degeneracies affect conclusions drawn from the models.

Chapter 6

Modeling Strong Lenses: The `lensmodel` Application

This chapter describes the `lensmodel` application, which takes the capabilities of the `gravlens` kernel and supplements them with routines to make it easy to fit models to observed lens systems. Since `lensmodel` operates in the same way as `gravlens` and uses many of its commands and variables, you should read Chapter 4 before this chapter. The companion paper ([Kee01b]) and Chapter 5 give a detailed discussion of the modeling strategies that are implemented in `lensmodel`. Chapter 7 leads you step-by-step through some modeling examples.

6.1 Overview

A crucial step in any run is specifying the mass model with the `startup` command; see §4.2 to review this command. One difference from `gravlens` is that `lensmodel` does use the flags that you specify after giving the model parameters. These flags tell the optimization routine which parameters are allowed to vary when searching for a best fit. For example, if you are fitting a singular isothermal ellipsoid to an observed lens and you want to fix both the position and PA of the lens galaxy while letting the mass and ellipticity of the galaxy vary, you might use:

```
> startup 1 1
    alpha 1.1 -0.3 0.2 0.4 78.1 0.0 0.0 0.0 0.0 1.0
    1 0 0 1 0 0 0 0 0 0
```

In the flags, the 1's mean that the mass parameter (`p[1]`) and the ellipticity (`p[4]`) are allowed to vary, while the lens position (`p[2]` and `p[3]`), PA (`p[5]`), and other parameters are not. The flags apply in the same order as the parameters.

Because the code allows multiple galaxies in a model, the full set of parameters in a model can be thought of as an array `p[1..Ngal][1..10]`. In this manual we refer to a particular model parameter as an element in this array, e.g. `p[igal][iparm]`. This picture of a parameter array relates to the `startup` command as follows:

```
> startup Ngal 1
    model1 p[1][1] p[1][2] p[1][3] p[1][4] p[1][5] p[1][6] p[1][7] ...
    model2 p[2][1] p[2][2] p[2][3] p[2][4] p[2][5] p[2][6] p[2][7] ...
    ...
    modelN p[N][1] p[N][2] p[N][3] p[N][4] p[N][5] p[N][6] p[N][7] ...
   [array of flags for what to vary]
```

The central function of the `lensmodel` application is to compute χ^2 for a set of models. The main constraints on models come from lens data, and these are explained in §6.2. If you have data from different sources, you can have the code optimize the registration between the data sets (§6.3). In some cases you can have the code solve analytically for some of the model parameters (§6.4). You can also specify explicit constraints on parameters (§6.5). You can control the way the code searches the parameter space for good models (§§6.6 and 6.7). Table 6.1 summarizes the basic capabilities of `lensmodel`. Table 6.2 lists some χ^2 values that indicate errors during runs, and Table 6.3 describes the output files from various runs.

When optimizing models you need to be aware of any degeneracies among parameters; review §5.5 for a brief discussion of some common degeneracies and how to handle them.

6.2 Specifying the lens data

6.2.1 Variables

`rscale [1.0]` Distance of farthest image from the lens galaxy — *set automatically in the code*. (Also see §4.1.)

`fluxweight [1.0]` Weight applied to flux constraints in χ^2.

`checkparity [1]` Flag for whether to check image parities.

`nopointchi [0]` Flag for whether to ignore point image.

`chiperpoint [0.0]` For curve constraints, compute total (0) or per point (>0) curve χ^2. If nonzero, it specifies the expected mean number of images and imposes a penalty if the number of images is too small.

`omitcore [0.0]` Size of omitted regions around galaxies; see §4.5.

6.2.2 Commands

`data` ⟨*file*⟩

Loads lens data from the specified file. Assumes isotropic position uncertainties. The form of the data file is discussed below.

`data2` ⟨*file*⟩

Feature	Capability
Point images	Position, flux, and time delay data. Image plane or source plane χ^2. Can handle multiple sources.
Arcs	Curve fitting.
Rings	Ring fitting with elliptical sources.
General data	Allows a floating registration between different types of data (e.g., radio and optical).
Mass models	Includes numerous circular and elliptical models, and allows arbitrary combinations of them.
Parameters	Gives full control over which parameters vary. Can produce arbitrary parameter surveys. Allows external constraints on parameter ranges. Allows relations between parameters to be imposed. Can use linear parameters and constraints.
Cosmology	Allows arbitrary values of Ω_M, Ω_Λ, and H_0. Can determine H_0 from time delays.
Output	Goodness of fit. Properties of mass model. Critical curves and caustics. Plots of time delay surface, potential, etc.

Table 6.1 – Features of `lensmodel`.

Similar to `data`, but allows error ellipses for the position uncertainties. The form of the data file is discussed below.

`crvmax` [*max # of curves*] [*max # of segments in a curve*] [*max # of points on a segment*]

Checks or updates the maximum dimensions of curve data files.

`crvstatus`

Prints information about the curve data that have been loaded.

`crvpts` ⟨*index*⟩ ⟨*mode*⟩ [*list of files*]

Loads data for curve constraints. The *index* specifies which family of curves you are loading (see below). The *mode* specifies which type of points you are loading: 1→loads reference points, 2→loads test points. The form of the data files and the definition of χ^2 are discussed below.

`ringmax` ⟨*max # of rings*⟩ ⟨*max # of rays on a ring*⟩

Sets maximum dimensions of ring data files.

`ringdat` [*list of files*]

Loads data for Einstein ring constraints.

6.2.3 Point images

When modeling lenses with discrete image, the code allows constraints from the positions and fluxes of the images (see §5.1, and §4 of the companion paper [Kee01b]), as well as the position of the lens galaxy and its optical properties like effective radius, PA, and ellipticity. In addition, if time delays are known then `lensmodel` can use them to compute H_0 and to constrain the models (see below). If you do not wish to use a particular constraint, you must still include it in the data file but you can give it a large errorbar. One exception: if time delays are not known, tell the code not to use them by setting their errorbars to zero.

You can tell the code to ignore the point images entirely by setting `nopointchi=1`.

If the position errorbars are isotropic, use the `data` command to load a data file with the following form:

```
 Ngal                # number of galaxies
[for each galaxy:]
 x y sigma(x)        # position and errorbar
 R_e sigma(R_e)      # effective radius and errorbar
 PA  sigma(PA)       # PA and errorbar
 e   sigma(e)        # ellipticity and errorbar

 Nsrc                # number of distinct sources
[for each source:]
 Nimg                # number of images of this source
```

```
[for each image of this source, including central image:]
 x y flux sigma(x) sigma(flux) tdel sigma(tdel)
```

You can use blank lines or comment lines beginning with `#` to separate parts of the file. If you want to use error ellipses for the position uncertainties, use the `data2` command to load a data file with the following form:

```
 Ngal
[for each galaxy:]
 x y sigma(a) sigma(b) sigma(pa)
 R_e sigma(R_e)
 PA  sigma(PA)
 e   sigma(e)

 Nsrc
[for each source:]
 Nimg
[for each image of this source, including central image:]
 x y flux sigma(a) sigma(b) sigma(pa) sigma(flux) tdel sigma(tdel)
```

Here each error ellipse is described by a semi-major axis $\sigma(a)$, a semi-minor axis $\sigma(b)$, and a position angle σ(pa) measured in degrees East of North.

Three comments are in order. First, in its default mode the code finds the faint central image that exists in all non-singular lens models but is rarely observed. There are two ways to handle such core images (see Chapter 7 for an example). You can include estimated data for the core image. For example, you might give it a position near the lens galaxy, with a large error bar to prevent the position from contributing to the position χ^2. If you have an upper limit on the flux of the central image you can include that as the flux error bar; otherwise give it a large flux error bar. This is the approach to take if you want to use the lack of a central image to constrain the core of the lens galaxy. Alternatively, you can tell the code to ignore core images altogether using the `omitcore` variable; see §4.5 for a discussion. Note that the code uses `omitcore` to decide whether each image in the data file is a core image or not. *Hence* `omitcore` *must be set before you load the lens data using* `data` *or* `data2`*.*

Second, the code uses signed fluxes for the images, where the sign indicates the parity (see §5.1). In a 2-image lens, the outer image has positive parity and the inner image has negative parity. In a 4-image lens there are two positive images and two negative images; the parity alternates as you move around a circle centered on the lens galaxy. The code uses the parity as a first test to determine whether a model is viable; if a model does not produce the correct parities, the code can quickly reject the model without having to solve the lens equation, thus saving computation time. This feature can be annoying, however, when you first start to model a lens. Your initial guess for

parameter values will likely produce a model that cannot reproduce the parities, and the code will simply reject your model without telling you how to improve it. You can turn the parity check off by setting `checkparity=0`. In general, you probably want to omit the parity check for initial modeling. Once you get a reasonable first guess for a model, however, you probably want to turn the parity check back on because it can speed up the run.

Finally, there is an important technicality of the `startup`, `data`, and `data2` commands. When the code reads the lens data, it sets the `rscale` variable equal to the maximum separation of an image from the lens galaxy: `rscale` $= \max_i \left|\mathbf{x}_i - \mathbf{x}_{gal}\right|$. The variables that set the radial extent of the grid (`gridlo1` and `gridhi1`) are then given in units of `rscale`. This wrinkle is included to make sure that the image plane grid is always big enough to cover all of the images. For example, if `gridhi1=2` then the image plane grid will extend to twice the image separations, no matter how large or small those separations are. That way you can run `lensmodel` on any lens system without having to manually adjust the extent of the grid every time. Note, though, that this linking between `data`/`data2` (which set `rscale`) and `startup` (which uses it to generate the grid) means that you should always call `data`/`data2` before `startup`.

Examples

Here is a sample data file for a 4-image lens with isotropic position uncertainties, to be loaded with the `data` command:

```
1                     # number of galaxies
 0.42 -1.31 0.05      # position: x y sigma
 0.0 10000.0          # R_eff sigma
 0.0 10000.0          # PA    sigma
 0.0 10000.0          # ellip sigma

1                     # number of sources

5                     # number of images for source #1
#
#  x       y     flux   sig(x)  sig(f)   tdel sig(t)
#
-0.588  -1.934   401.0   0.01    20.0    0.0   0.0    # image A1
-0.721  -1.530  -362.0   0.01    18.1    0.0   0.0    # image A2
 0.000   0.000   156.0   0.01     7.8    0.0   0.0    # image B
 1.361  -1.635   -58.0   0.01     2.9    0.0   0.0    # image C
 0.42   -1.31      0.1  10000.  10000.   0.0   0.0    # central image
```

(Note that comments are indicated by `#`; you can include comments as detailed as you like.) A more complicated lens might have two sources, each of which is doubly-imaged. The position uncertainties

might be described by error ellipses (as opposed to isotropic errorbars). Here is a sample data file, to be loaded with the `data2` command:

```
1                              # number of galaxies
 -0.199 0.092 0.006 0.006 0.   # position: x y sig(a) sig(b) sig(pa)
  0.75  10000.0                # R_eff sigma
  61.0  10000.0                # PA    sigma
  0.58  10000.0                # ellip sigma

2                              # number of sources

3                              # number of images of source #1
#
#  x       y     flux   sig(a) sig(b) pa  sig(f)  tdel sig(t)
#
 0.0000  0.0000   621   0.0014 0.0010 41.  7.0     0.0 0.6  # A1
-0.3091  0.1274  -172   0.0017 0.0010 -3.  3.0    11.7 0.6  # B1
-0.199   0.092    0.01  1000.  1000.   0.  1000.   0.0 0.0  # central

3                              # number of images of source #2
-0.0010  0.0008   379   0.0011 0.0009 -7.  5.0     0.0 0.0  # A2
-0.3106  0.1274  -104   0.0018 0.0015 71.  3.0     0.0 0.0  # B2
-0.199   0.092    0.01  1000.  1000.   0.  1000.   0.0 0.0  # central
```

This data file also shows how to use the time delay constraints: simply make non-zero the time delay error for any image whose time delay you want to use.

6.2.4 Time delays and H_0

See §4.3 of the companion paper ([Kee01b]) for a discussion of how to use time delays to determine the Hubble constant H_0. The model time delay factors into two pieces (see eq. 23 in the paper),one of which depends only on the model and the other of which (t_0) depends only on cosmology. You can specify the cosmology and the lens and source redshifts so the code can compute the time delay scale t_0 and thus express delays in h^{-1} days (assuming image positions in arcseconds); see §4.6. Alternatively, you can compute the time delay scale t_0 yourself and specify its value using the `tscale` variable. Setting `tscale` supersedes `zlens` and `zsrc`.

Finally, you can specify a Hubble constant `hval` and its uncertainty `hvale` (see §4.6), which are used as prior assumptions on $h = H_0/(100 \text{ km s}^{-1}\text{Mpc}^{-1})$ (see the paper, especially eqs. 27 and 28). Using `hvale=0` forces the code to use the specified value `hval` for h. Alternatively, using a large value for `hvale` allows the code to adopt the value for h that gives the best fit to the time delays.

To trace χ^2 versus h, use the `varyh` command (see §6.7).

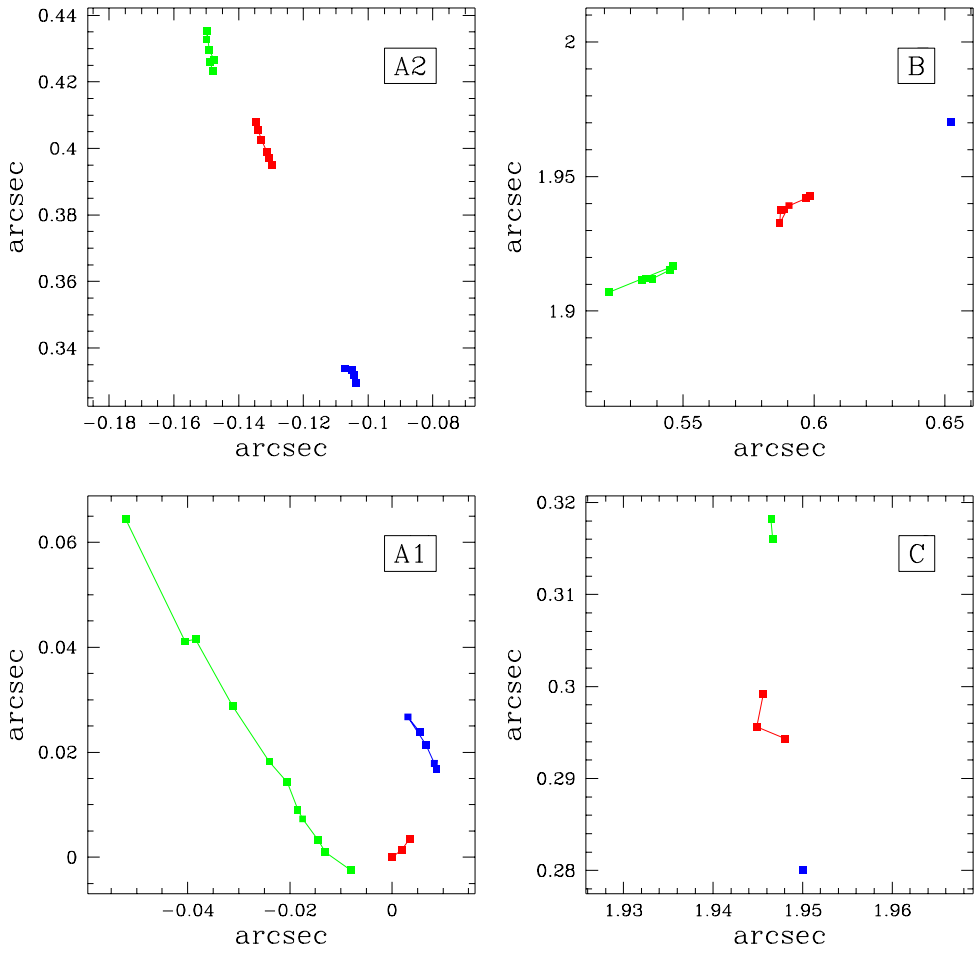


Figure 6.1 – Curve data for the 4-image lens MG J0414+0534 (see [RGM+00]; data courtesy E. Ros and J. Muñoz).

6.2.5 Curves

See §5.3 for a general discussion of how the code uses constraints from curves. The code allows you to have multiple families of curves. Curve segments in a given family should map into each other, but segments from different families should not. For example, Figure 6.1 shows curves used to model the 4-image lens MG J0414+0534 (see [RGM+00]). The red curves all form one family, the green curves form a second family, and the blue curves form a third family. In other words, all of the red curves should map into each other, the green curves should map into each other, and so forth. No

red curve segment should map into any green or blue curve segment, and so forth.

Create a different file for each segment in each curve family. For example, for Figure 6.1 you might have the following files:

red family	:	`A1.1.dat`, `A2.1.dat`, `B.1.dat`, `C.1.dat`
green family	:	`A1.2.dat`, `A2.2.dat`, `B.2.dat`, `C.2.dat`
blue family	:	`A1.3.dat`, `A2.3.dat`, `B.3.dat`, `C.3.dat`

The form of each curve data file is as follows:

```
N           # number of points in this file
x1 y1 sigma1
x2 y2 sigma2
...
xN yN sigmaN
```

Note that the errorbars `sigmaj` must be included in the file, but they are used only for the reference curves (and not for the test curves).

To load the curve data, use the `crvpts` command. You must specify the index of the curve family you are loading. You must also specify a *mode* to indicate the type of points: mode 1 load the reference curve(s), while mode 2 loads the test curve(s). For example, suppose you want to load just the red family of curves, and you want to use image A1 as the test curve and all of the images as the reference curves:

```
> crvpts 1 1 A1.1.dat A2.1.dat B.1.dat C.1.dat
> crvpts 1 2 A1.1.dat
```

To load all three families of curves:

```
> crvpts 1 1 A1.1.dat A2.1.dat B.1.dat C.1.dat
> crvpts 1 2 A1.1.dat
> crvpts 2 1 A1.2.dat A2.2.dat B.2.dat C.2.dat
> crvpts 2 2 A1.2.dat
> crvpts 3 1 A1.3.dat A2.3.dat B.3.dat C.3.dat
> crvpts 3 2 A1.3.dat
```

Use the `crvstatus` command to check the status of the curve data you have loaded.

Note that the `crvpts` command is set up so that you can use exactly the same data files for both the reference curve(s) and the test curve(s). Thus if you have two curve segments A and B in the files `fooA.dat` and `fooB.dat`, you could give the commands:

```
> crvpts 1 fooA.dat fooB.dat   # load them as reference curves
> crvpts 2 fooA.dat fooB.dat   # load them as test curves
```

In this case the χ^2_{crv} statistic would quantify the ability of A to map into B and of B to map into A.

The code has an upper limit on the number of curve families you can load, on the number of curve segments per family, and on the number of points per segment. Use the `crvmax` command with no arguments to check these limits. Use `crvmax` with its three arguments to change the limits.

Sometimes the curve fitting algorithm will try to reduce the total χ^2_{crv} by changing the lens model to eliminate some points. For example, suppose you have an extended source where part of the source is doubly-imaged and part is quadruply-imaged. In some cases the code may be able to get a lower total χ^2_{crv} by making a qualitatively *poorer* fit that has *fewer* points. You can avoid this problem by using as the curve statistic $\bar{\chi}^2_{crv} \equiv \chi^2_{crv}/N_{crv}$ where N_{crv} is the number of points. Specify this behavior by setting `chiperpoint` > 0. The value of `chiperpoint` specifies the expected mean number of images, and there is a penalty imposed if the number of images is too small.

Examples

Here is a sample curve data file, to be loaded with the `crvpts` command:

```
  9   # number of points in this file
  3.918765  3.585372  0.01
  3.276968  5.600562  0.01
  2.554720  6.119629  0.01
  1.686298  6.843378  0.01
  1.275909  6.705071  0.01
 -0.662859  6.066982  0.01
 -0.812853  4.424905  0.01
  0.497114  3.585157  0.01
  2.603029  2.688660  0.01
```

6.2.6 Rings

See §5.4 for a general discussion of how the code uses constraints from Einstein rings. To load the ring data, use the `ringdat` command. You may use more than one Eintein ring. Put each ring into its own file, and load them all with `ringdat`. For example, suppose you have an optical ring with data in `oring.dat` and a radio ring with data in `rring.dat`. If you want to use just the radio ring, use the command

```
> ringdat rring.dat
```

However, if you want to load both rings, use the command

```
> ringdat rring.dat oring.dat
```

The form of the ring data file is as follows:

```
 N           # number of rays on ring
[for each ray:]
 x0 y0 phi lambda d(lambda) d(phi) flag
```

The ray is defined as

$$x(\lambda) = x_0 - \lambda \sin\varphi\,, \tag{6.1}$$

$$y(\lambda) = y_0 + \lambda \cos\varphi\,, \tag{6.2}$$

so φ is a position angle, i.e. given in degrees East of North. The value for λ in the data file gives the location of the ring along that ray, with uncertainty $d\lambda$. The `flag` has the following values:

- `flag=0`: normal ring point.
- `flag=1`: local maximum in the flux along the ring.
- `flag=2`: local minimum in the flux along the ring.

§5.4 discusses how the various types of points are used. The angle uncertainty $d\varphi$ is used only for points that are local maxima or minima in the flux along the ring.

In order to model a ring you must specify the parameters for the source. The only important parameters are the position and shape — the ring fitting technique is insensitive to the radial profile of the source (see §5.4 and [KKM01]). Specify the ring parameters as part of the lens model using the `ring` model class:

```
ring <iring> <u0> <v0> <e> <PA> 0 0 0 0 0
```

Here `iring` specifies which ring you are talking about; the code can handle more than one ring. The source position is given by (u_0, v_0), and the shape by (e, PA). The ring source parameters can be optimized or held fixed just as any other model parameters.

Examples

(?? Add an example! ??)

6.3 Specifying data registrations

6.3.1 Variables

`maxrgstr [5]` Maximum number of registration definitions. (Also see §4.7.)

6.3.2 Commands

(None.)

6.3.3 Discussion

In some cases you might have different sets of data for which the relative registrations are poorly known. For example, you may have observed point images at radio wavelengths, but the lens galaxy and an Einstein ring at optical wavelengths (e.g., B 1608+656, [MFD$^+$95]). In each observation (radio and optical) the errorbars on the *relative* positions may be small, but the uncertainty in registering the radio images relative to the optical galaxy may be considerably larger. For such cases, you can tell the code to let the registration between the radio and optical maps "float" and be optimized as part of the model.

In order to allow the registration to be optimized, it must be specified as part of the lens model. Do this by using the model class `register`, which has the following form:

```
register <datatype> <xshift> <yshift> <index> 0 0 0 0 0 0
```

Note that the final six parameters (zeros in this example) are ignored. The `datatype` parameter specifies the type of data to which the registration applies: `datatype=1` means point data, `datatype=2` means curve data, and `datatype=3` means ring data. The amount of shift applied to the data is given by `xshift` and `yshift`; the shift is applied such that if a data file gives a data point at position (x_i, y_i), the code treats the data point as though it were at (x_i + `xshift`, y_i + `yshift`).

Finally, `index` specifies which entry of a particular data type receives the shift. For example, if you have point images for two sources

```
register 1 <xshift> <yshift> 1 0 0 0 0 0 0
```

specifies a shift that applies to the images for source #1, while

```
register 1 <xshift> <yshift> 2 0 0 0 0 0 0
```

specifies a shift that applies to the images for source #2. The same thing applies to ring data if you have multiple rings. The value `index=0` is special, and it indicates that the specified shift applies to *all* data of the given type. In other words,

```
register 3 <xshift> <yshift> 0 0 0 0 0 0 0
```

indicates to apply the same shift to all Einstein rings.

When you specify the parameters of a model, the `register` specifications must come *after* all of the mass components. For example,

```
> startup 4 1
  alpha ...
  nfw ...
  register ...
  register ...
```

is valid, but

```
> startup 3 1
  alpha ...
  register ...
  nfw ...
  register ...
```

is invalid because a mass component (`nfw`) comes after a `register`.

6.4 Using linear parameters and constraints

6.4.1 Variables

`linparms [0]` Flag for whether to fix linear parameters using linear constraints.

6.4.2 Commands

(None.)

6.4.3 Discussion

The basic ideas behind linear parameters and constraints are discussed in §4.4 of the companion paper ([Kee01b]). Candidate linear parameters are the galaxy mass parameters (`p[j][1]` for various `j`) and the external shear parameters. The code currently allows the following modes of linear parameters; recall that they all require the selected images to be fit *exactly*:

- `linparms=0`: No action.
- `linparms=1`: Uses the first two images of the first source to determine the amplitude and direction of the external shear.
- `linparms=2`: Uses the first two images of the first source to determine the amplitude of the external shear and the mass parameter of the main lens galaxy (`p[1][1]`).
- `linparms=3`: Uses the first two images of the first source to determine the mass parameters of the first two galaxies (`p[1][1]` and `p[2][1]`). *Requires a model with at least two galaxies.*

6.5 Controlling the parameters

6.5.1 Variables

`nobflip [0]` Flag for whether to prevent sign flips in the b parameters. If `nobflip=1`, the code rejects any model where a b parameter has changed signs relative to the `startup` model.

6.5.2 Commands

`plimits` ⟨*file*⟩

Lets you limit the range of model parameters.

`pmatch` ⟨*file*⟩

Lets you specify relations between model parameters.

6.5.3 Discussion

The central step in lens modeling is varying the model parameters to find the best fit. With `plimits` and `pmatch` you can control how the parameters vary. The two commands are similar, with the difference that `plimits` involves *absolute limits* on parameter ranges, while `pmatch` involves *relations* between parameters.

The code takes `plimits` constraints and adds a χ^2 term to penalize the model when parameters exceed a given range. You specify the contrained parameters and their ranges in a `plimits` input file with the following form:

```
 N (# of plimits constraints)
[for each constraint:]
 igal iparm value sigma
```

For each `plimits` constraint the code adds a χ^2 term of the form

$$\chi^2_{plimits} = \frac{(\mathtt{p[igal][iparm]} - \mathrm{value})^2}{\sigma^2} \ . \tag{6.3}$$

The code uses `pmatch` to enforce relations between model parameters. You specify the relations in a `pmatch` input file with the following form:

```
 N (# of pmatch constraints)
[for each constraint:]
 igal iparm jgal jparm ratio sigma
```

For each `pmatch` constraint with $\sigma = 0$ the code forces

$$\mathtt{p[igal][iparm]} = \mathrm{ratio} \times \mathtt{p[jgal][jparm]} \, . \tag{6.4}$$

For each `pmatch` constraint with $\sigma \neq 0$ the code adds a χ^2 term of the form

$$\chi^2_{pmatch} = \frac{(\mathtt{p[igal][iparm]} - \mathrm{ratio} \times \mathtt{p[jgal][jparm]})^2}{\sigma^2} \ . \tag{6.5}$$

It is important to understand the difference between these two types of constraints. The $\sigma = 0$ constraints represent parameter relations that are explicitly imposed on the models; `p[igal][iparm]`

is not allowed to vary separately from `p[jgal][jparm]`, but is always forced to the obey the specified relation. By contrast, the $\sigma \neq 0$ constraints are included only as χ^2 penalties; the two parameters are allowed to vary independently, but the model is penalized if they do not satisfy the specified relation.

Direct control over the b parameters is provided with the variable `nobflip`. When you are using complicated mass models with multiple components, the code may try to improve the fit by giving one of the components a negative mass. To prevent this possibility, set `nobflip=1` to tell the code to reject (with a large χ^2 penalty) any model where a b parameter has changed sign. The `nobflip` actually works in reverse, too: if you are using a composite model where you actually want one of the components to have a negative mass, [1] the code will prevent the corresponding b from becoming positive. In other words, `nobflip` simply prevents b from changing signs. The signs are always compared with the `startup` model.

6.5.4 Examples

We recently introduced models for Q 0957+561 comprising two concentric pseudo-Jaffe ellipsoids to allow radial and angular substructure similar to that observed in the main lens galaxy (see [KFI$^+$00]). A typical example of such a model might be:

```
set galcoords = 2
startup 2 1
  pjaffe 2.1 1.9 3.2 0.21 39.2 0.0 0.0 0.0   0.51 0.0
  pjaffe 1.8 0.0 0.0 0.49 57.8 0.0 0.0 1.23 30.0  0.0
  1 1 1 1 1 0 0 0 1 0
  1 0 0 1 1 0 0 1 0 0
```

In this example the two pseudo-Jaffe components have different ellipticities and orientations, so in the total model e and PA vary with radius to produce an "isophote twist." We wanted to consider models without a twist, so we used `pmatch` to force the two components to have the same ellipticity and orientation using the following `pmatch` file:

```
2   # number of pmatch constraints
2 4 1 4 1.0 0.0   # fix e2  to match e1
2 5 1 5 1.0 0.0   # fix PA2 to match PA1
```

If we wanted to constrain the resulting model to have an orientation in the range $30° < \mathrm{PA} < 70°$, we could use `plimits` with an input file as follows:

```
1   # number of plimits constraints
1 5 50.0 20.0   # limit PA to be 50 +/- 20
```

[1] This situation might arise when you combine several mass models to mimic a more complicated profile. For example, a pseudo-Jaffe model is a combination of two softened isothermal models where one component has a negative mass (see Chapter 3.)

Alternatively, suppose we wanted to allow the isophote twist but to constrain the two orientations to be in the range $30° < (\mathrm{PA}_1, \mathrm{PA}_2) < 70°$. Then we would not use `pmatch`, but we would use `plimits` with the following input file:

```
2   # number of plimits constraints
1 5 50.0 20.0   # limit PA1 to be 50 +/- 20
2 5 50.0 20.0   # limit PA2 to be 50 +/- 20
```

6.6 Controlling the optimization

6.6.1 Variables

`chimode [1.0]`	Which position χ^2 to use: 0 → source plane, 1 → image plane, otherwise mixed.
`srcmode [1]`	Flag for whether to optimize source position.
`optmode [1]`	Which optimization algorithm to use: 1 → amoeba, 2 → Powell.
`vertmode [1]`	Controls how the vertices for the initial simplex are determined: 1 → along parameter axes, 2 → random directions (see below).
`xfloor [0.005]`, `xceil [0.1]`	Smallest and largest length variations in initial simplex.
`ftol [1.0e-4]`	Tolerance for optimization routine.
`restart [1]`	Number of times to run optimization routine.
`upenalty [1.0e-4]`	Penalty for being in wrong source region.
`tempfiles [1]`	Flag specifying whether or not to write temporary files during optimization; see Table 6.3.

6.6.2 Commands

(None.)

6.6.3 Discussion

You can choose to have the position χ^2 evaluated in the image plane or the source plane, as discussed in §5.1. Setting `chimode=0` indicates the source plane χ^2, while setting `chimode=1` indicates the image plane χ^2. Any other value for `chimode` indicates a mixed use: the code evaluates the source plane χ^2, and if $\chi^2 <$ `chimode` the code then computes the image plane χ^2. You might be able to use this to speed up runs: first evaluate the fast source plane χ^2, and only if the model looks good do you evaluate the slower but more robust image plane χ^2. However, I have not explored this feature very thoroughly.

Note that if you use just the source plane χ^2, the code doesn't need to solve the lens equation and hence you don't need to use the tiling algorithm. Thus, you can speed up the run by turning off the tiling; to do so, set `gridflag=0` (see §4.1).

Use `srcmode` to tell the code whether or not to optimize the source position when using the image plane χ^2. Optimizing the source gives the lowest χ^2 but requires more function evaluations. If you set `srcmode=0`, the code maps each image $\mathbf{x}_i$ to its source $\mathbf{u}_i$ and whichever of these sources gives the lowest χ^2.

Use `optmode` to select between the optimization algorithms amoeba and Powell, which are presented by [PTVF92].

To begin the optimization, the code must set up a "simplex" consisting of $N+1$ vertices in the N-dimensional space of parameters that vary. If you specify more than one model in the `startup` command (see §4.2), the code uses up to $N+1$ of those models as initial vertices.[2] If you specify fewer than $N+1$ models, the code sets the remaining vertices automatically by taking steps away from the initial model. For the automatic vertices, the code uses the χ^2 to estimate an appropriate length scale

$$\delta \sim \left[\frac{\chi^2_{pos}}{\sum_i 1/\sigma^2_{x,i}} \right]^{1/2}, \tag{6.6}$$

and it uses this scale to determine appropriate steps in each parameter.[3] You can use the variables `xfloor` and `xceil` to specify the smallest and largest length variations the code can use to set up the initial simplex.

In default mode (`vertmode` = 1), the automatic vertices are determined by taking steps along the parameter axes. Suppose the startup model is $\mathbf{p}_0$, the parameter space has 4 dimensions, and the step sizes used to set up the initial simplex are δ_i. Then the additional vertices would be:

$$\mathbf{p}_1 = \mathbf{p}_0 + (\delta_1, 0, 0, 0) \tag{6.7}$$

$$\mathbf{p}_2 = \mathbf{p}_0 + (0, \delta_2, 0, 0) \tag{6.8}$$

$$\mathbf{p}_3 = \mathbf{p}_0 + (0, 0, \delta_3, 0) \tag{6.9}$$

$$\mathbf{p}_4 = \mathbf{p}_0 + (0, 0, 0, \delta_4) \tag{6.10}$$

However, if you set `vertmode` = 2 then rather than stepping along the unit vectors the code will pick random directions:

$$\mathbf{p}_1 = \mathbf{p}_0 + (a_{11}\delta_1, a_{12}\delta_2, a_{13}\delta_3, a_{14}\delta_4) \tag{6.11}$$

$$\mathbf{p}_2 = \mathbf{p}_0 + (a_{21}\delta_1, a_{22}\delta_2, a_{23}\delta_3, a_{24}\delta_4) \tag{6.12}$$

[2]The exception is the `reopt` command, in which the code uses each of the `startup` models to start a separate optimization. See §6.7 for details.

[3]Note that the `randomize` is different from the other optimization commands, and it does *not* use this technique to set up the initial simplex; see §6.7.

$$\mathbf{p}_3 = \mathbf{p}_0 + (a_{31}\delta_1, a_{32}\delta_2, a_{33}\delta_3, a_{34}\delta_4) \quad (6.13)$$

$$\mathbf{p}_4 = \mathbf{p}_0 + (a_{41}\delta_1, a_{42}\delta_2, a_{43}\delta_3, a_{44}\delta_4) \quad (6.14)$$

where the a_{ij} are random numbers between -1 and 1. This random vertex mode may help the code get out of local minima during restarts.

The optimization algorithm varies the parameters (the vertices in the simplex) to look for the best fit. The algorithm stops when the *fractional* difference between the χ^2 at all vertices is less than `ftol`. If the model gives a perfect fit ($\chi^2 \approx 0$), the algorithm stops when all vertices have $\chi^2 <$ `ftol`. There is a tradeoff to consider when setting `ftol`. Small values of `ftol` require many function evaluations before they "converge" to the final fit. However, with large values of `ftol` the optimization may stop before it reaches a true minimum in the χ^2 surface — for example, it may get hung up as it tries to work its way down a narrow valley. I have found that `ftol` $= 10^{-4}$ seems to work well.

A good way to check whether the optimization is getting hung up before it reaches a true minimum is to restart the code from the "final" model and see if it reconverges to the same (or a very similar) model. If you restart manually and discover that the code *is* stopping prematurely, use the `restart` variable to tell the code to do the restart automatically. Note that `restart` is a bit of a misnomer, because it is really the total number of runs, not the number of restarts.

Finally, when the code optimizes the source position, the source may cross a caustic and enter the wrong region of the source plane. For example, when modeling a 4-image lens, the source may accidentally step outside the astroid caustic and into the 2-image region (or vice versa). The code uses the variable `upenalty` to define a penalty function that helps force the source back into the correct region of the source plane. You should not need to adjust `upenalty`.

6.7 Optimizing the model

6.7.1 Variables

`verbose [1]` Specifies the extent to which status reports are written to the screen; `verbose` $=$ `1` means everything is written, while larger values means that some messages are omitted (see below).

6.7.2 Commands

`optimize` [*outbase*]

Varies model parameters to find the best fit to the constraints. Writes the results to the files *outbase*`.dat` and *outbase*`.start`. The default value of *outbase* is `best`.

`minimize` [*outbase*]

Same as `optimize`.

`reopt` ⟨*outbase*⟩ [*starting index*] [*ending index*]

Re-optimizes models given in the `startup` command. Writes the results to the files *outbase*`.dat` and *outbase*`.startN`. Runs only between the specified starting and ending indices (default 1 and N).

`randomize` ⟨*number of times to run*⟩ ⟨*outbase*⟩

Generates a set of models with random values of the parameters that are allowed to vary, and then runs the optimization. Can be run multiple times to check whether the "converged" models are robust. After you give the `randomize` command, the code prompts you to specify the ranges for the parameters that vary.

`modelgrid` ⟨*number of dimensions*⟩ ⟨*outbase*⟩

Compute χ^2 for a discrete grid of models. After you give the `modelgrid` command, the code prompts you to specify the parameters and their ranges on the grid.

`varyone` ⟨*igal*⟩ ⟨*iparm*⟩ ⟨*lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨*outbase*⟩

Computes 1-d slice of χ^2 surface by tracing χ^2 versus the specified parameter over the specified range.

`varytwo` ⟨*igal*⟩ ⟨*iparm*⟩ ⟨*lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨*jgal*⟩ ⟨*jparm*⟩ ⟨*lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨*outbase*⟩

Computes 2-d slice of χ^2 surface by tracing χ^2 versus the specified parameters over the specified ranges.

`varyh` ⟨*h lo*⟩ ⟨*hi*⟩ ⟨*steps*⟩ ⟨*outbase*⟩

Computes χ^2 versus H_0 by tracing χ^2 versus h over the specified range.

6.7.3 Discussion

Armed with the data and at least one startup model, the `lensmodel` application is set to compute χ^2 for a variety of models. There are several ways to do this. For fully automated optimization, use `optimize`. With this command, the code varies all of the model parameters you flagged as variable (in `startup`), using the Numerical Recipes [PTVF92] algorithm amoeba or powell to find the best-fit model. This command can be dangerous, though, if there are degenaracies among the model parameters (see §5.5). *Be extremely wary of running a full optimization without first understanding any parameter degeneracies that may be present!*

For each command that involves looping over models (`reopt`, `randomize`, `modelgrid`, `varyone`, `varytwo`, and `varyh`) you can use the `verbose` variable to specify the extend of status reports written to the screen. In the most verbose setting (`verbose` = `1`), the code prints a status report for each step in the loop, plus some messages from the optimization routine as it operates. If you are doing large loops where each step is fast, the time to print the status reports may slow you down and you may wish to turn them off. For `verbose` = 2 you get medium verbosity; the optimization routine

becomes silent, but the code still tells you about each step in the loop. For `verbose` = 3 even the loop steps are silent, so you get very few status reports.

To compute χ^2 for a particular model (no optimization of parameters), simply run `optimize` with all the parameters flagged to be fixed.

To trace out χ^2 versus a particular model parameter (say `p[igal][iparm]`), use `varyone`. For each value of the parameter, the code varies all the other parameters that you flagged as variable (in `startup`). Note that you need not manually flag `p[igal][iparm]` as fixed; the code does that automatically. Thus you can do back-to-back runs first holding `p[igal][iparm]` fixed and then `p[jgal][jparm]`, for example:

```
data mydata
startup 1 1
  alpha 1.1369 0.5270 -1.3378 0.4382 79.05 0.0 0.0 0.0 0.0 1.0
  1 0 0 1 1 0 0 0 0 0
varyone 1 4 0.0  0.6 21 vary_e
varyone 1 5 0.0 90.0 21 vary_PA
```

The first `varyone` run traces $\chi^2(e)$ while the PA and mass vary, and the second run traces χ^2(PA) while e and the mass vary.

The command `varyh` is similar to `varyone`, but for the Hubble constant.

For two-dimensional cuts of parameter space, use the `varytwo` command. For higher-dimensional cuts, use the `modelgrid` command. These commands compute χ^2 on a parameter grid with any dimension you choose (up to 10), again allowing other parameters to vary if desired. For example, you could use `varytwo` to plot χ^2 as a function of lens position where the lens mass, ellipticity, and PA are optimized at every lens position:

```
startup 1 1
  alpha 1.0 -0.492 0.193 0.1 10.0 0.0 0.0 0.0 0.0 1.0
  1 0 0 1 1 0 0 0 0 0
varytwo 1 2 -0.6 -0.4 21 1 3 0.1 0.3 21 vary_pos
```

In this example, x_{gal} varies from -0.6 to -0.4 in 21 steps, and y_{gal} varies from 0.1 to 0.3 in 21 steps. As another example, you could use `modelgrid` to plot χ^2 as a function of the mass parameters for a model with three spherical galaxies, where the galaxy positions are allowed to vary:

```
startup 3 1
  alpha 0.416 -0.129 -0.945 0.0 0.0 0.0 0.0 0.0 0.0 1.0
  alpha 0.253 -0.652 -0.742 0.0 0.0 0.0 0.0 0.0 0.0 1.0
  alpha 0.285 -0.624 -1.491 0.0 0.0 0.0 0.0 0.0 0.0 1.0
  0 1 1 0 0 0 0 0 0 0
  0 1 1 0 0 0 0 0 0 0
```

```
  0 1 1 0 0 0 0 0 0 0
modelgrid 3 vary_b3
  1 1 0.1 0.8 8  # b for galaxy 1
  2 1 0.1 0.5 5  # b for galaxy 2
  3 1 0.1 0.5 5  # b for galaxy 3
```

The `varyone` command is useful for computing error bars on fitted parameters; for a parameter p, trace $\chi^2(p)$ and look for the region where χ^2 changes by some fixed amount (e.g., $\Delta\chi^2 = 1$). The `varytwo` and `modelgrid` commands are useful for making cuts of parameter space to look for degeneracies, and for examining the error ellipse for degenerate parameters.

There is an important detail about `varyone` and `varytwo`. During the run, when the code starts to examine a new value for the parameter(s) in question, it uses a neighboring model as the starting point. For example, if you are running `varyone` with the ellipticity from 0.1 to 0.3 in 21 steps, when the code starts to examine $e = 0.23$ it will look at the model with $e = 0.22$ to get the starting point. This feature is helpful when the model changes significantly during the course of the run. However, it has one drawback. Sometimes the code gets confused and fails to find a good model for a particular value of the parameter(s). The neighbor technique means that the bad model can corrupt all subsequent models. The effect is obvious as a sharp discontinuity in the χ^2 values. On the occasions when I have encountered this problem, I have found that increasing the number of models (so there is less distance between neighbors) often eliminates the problem. Another option is to switch to `modelgrid`, which does not use the neighbor technique.

Another trick with the parameter survey commands is to set `restart` to be something greater than 1 (see §6.6). This tells the code that for each parameter value it should run the optimization routine once and then restart from the "converged" model and run again to make sure that the routine didn't get hung up in a local minimum or a narrow valley. Using `restart=2` helps make the optimization more robust, especially in more complicated lens models.

At times you may want to generate random models as starting points for the optimization. For example, when you first begin modeling a system you may have no idea what parameter values are appropriate. Or, when using complicated models you may want to run a series of tests to check that your "converged" model is not just a local minimum of the χ^2 function. Use `randomize` to generate and run random models. The code asks you to specify ranges for the parameters that vary. For example, suppose you want to run a model with a singular isothermal ellipsoid and an external shear; suppose you want to fix the galaxy position and try random values of the mass, ellipticity and its PA, and shear and its PA:

```
startup 1 1
  alpha 1.0 -0.492 0.193 0.1 10.0 0.1 10.0 0.0 0.0 1.0
  1 0 0 1 1 1 1 0 0 0
randomize 10 myrun
```

```
  0.5  2.0  # b range
  0.0  0.5  # ellip range
-90.0 90.0  # PA range
  0.0  0.2  # shear range
-90.0 90.0  # PA range
```

(Note that you do not have to specify which parameters are active in `randomize`; the code uses the vary flags given in the `startup` command to identify them. The parameter ranges are entered in the same order as the vary flags.) The code creates an initial simplex whose vertices are random points in the parameter box you specified, and uses this simplex to start the optimization routine. You can repeat the process multiple times (10 in this example) to check whether the converged models all agree. Note that if you do several `randomize` runs, you should change the seed for the random number generator each time (see §4.7) so you don't get the same sequence of "random" models for each run.

You can re-optimize a set of models with the `reopt` command. For example, suppose you use `varyone` to trace out χ^2 versus some parameter `p[igal][iparm]`. Later, you update the lens data, so you need to recompute χ^2. You could run `varyone` again, but it would be faster to re-optimize the old models. The previous `varyone` run produced a `startup` file, say `old.startN`, containing all of the previous models. To reoptimize those models, use the commands:

```
data newdata
startup old.startN
reopt new
```

The `reopt` command has optional arguments specifying the range of model indices you want to run. If you wanted to leave the first 10 models untouched and optimize models starting with number 11, use the command:

```
reopt new 11
```

This option might be useful to run `reopt` in several pieces.

The code uses large errors of χ^2 to indicate errors that occur during modeling. The large values allow the code to handle the errors without crashing, and they should drive the model back into acceptable regions. The error values are given in Table 6.2. The errors are often explained in the working file `chitmp.dat`.

Each type of optimization command produces a specific set of output files; see Table 6.3 for a description of the files. See Chapter 7 for a step-by-step discussion of sample runs.

χ^2	Error
1e8	failed to initialize model because parameters out of bounds
1e9	failed parity test; see §§5.1 and 6.2
1e10	model has too many images
1e11	model has too few images

Table 6.2 – χ^2 values that indicate specific errors. The total χ^2 may be slightly higher than these values (due to contributions from, say, galaxy or `plimits` constraints), but the order of magnitude should still indicate the error.

Command	File	Comments
Working files[†]	`chitmp.dat`	models tested during optimization
	`crvpts.tmp`	points used with curve constraints
	`ringpts.tmp`	points used with ring constraints
`optimize` [*outbase*]	*outbase*`.dat`	a summary of the results for the best model
or `minimize` [*outbase*]	*outbase*`.start`	a `startup` file containing the best model
	`best.sm`	SuperMongo macros to plot the best model
	`grid.dat`	the grids for the best model
	`crit.dat`	the critical curves for the best model
`reopt` ⟨*outbase*⟩	*outbase*`.dat`	the params for the re-optimized models
	outbase`.startN`	a `startup` file for the models
`varyone` ⟨*outbase*⟩	*outbase*`.dat`	the params for the models
or `varyh` ⟨*outbase*⟩	*outbase*`.chi`	the χ^2 for the models[‡]
or `varytwo` ⟨*outbase*⟩	*outbase*`.best`	a summary of the best model
or `modelgrid` ⟨*outbase*⟩	*outbase*`.start`	a `startup` file for the best model
or `randomize` ⟨*outbase*⟩	*outbase*`.startN`	a `startup` file for all models

Table 6.3 – Summary of output files for the various optimization commands.

[†] The working files can be large (especially `chitmp.dat`). Setting `tempfiles=0` tells the code not to write them.

[‡] The χ^2 file is not written for `modelgrid` or `randomize`. The form of the χ^2 file differs for different optimization commands, and is described at the top of the file.

Chapter 7

A `lensmodel` Tutorial

This chapter leads you step by step through examples of modeling the lens PG 1115+080 (e.g., [KK97]; [IFK+98]). The chapter is organized in what I think of as a logical sequence for lens modeling:

- §7.1, gather the data.
- §7.2, run 0^{th} order models to get a general idea what the models should be like.
- §7.3, run models with ellipticity but no shear.
- §7.4, run ellipticity+shear models and understand degeneracies.
- §7.5, replace the shear with a model for the group of galaxies.
- §7.6, consider other, more complicated classes of models.
- §7.7, use the time delay to constrain the Hubble constant H_0.

7.1 Data

Here is a sample data file, `pg1115.dat`, containing data for the lens PG 1115+080 from [IFK+98]:

```
1                       # 1 lens galaxy
 -0.382 -1.344 0.003    # position
 0.59 1000.             # R_eff observed: 0.59 +/- 0.06
 0.0  1000.             # PA unconstrained in observations
 0.0  1000.             # observed e < 0.07 at 1sigma

1                       # 1 source
5                       # 5 images of the source
 -1.328 -2.037  3.88  0.003  0.78  0.0 0.0  # A1
 -1.478 -1.576 -2.51  0.003  0.50  0.0 0.0  # A2
  0.341 -1.960 -0.65  0.003  0.13  0.0 0.0  # B
  0.0    0.0    1.0   0.003  0.20  0.0 0.0  # C
 -0.383 -1.345  0.0   1000.  0.05  0.0 0.0  # central

# note: use flux errorbars of 20%
```

The file includes contraints from the galaxy position, the quasar image positions,[1] and the image flux ratios. The position constraints are usually reliable within the quoted astrometric errorbars. The flux ratios are often less reliable; they may vary systematically due to intrinsic variability in the quasar combined with the lensing time delay; and they may vary randomly because of microlensing by objects in the lens galaxy (e.g., [MS98]). These uncertainties are probably more significant for optical fluxes than for radio fluxes. For these reasons, I have set the flux uncertainties to 20%.

The file includes data for the effective radius, PA, and ellipticity of the main lens galaxy. The effective radius and ellipticity are useful if using a de Vaucouleurs constant mass-to-light ratio galaxy, but they are less useful for more general dark matter models because there is no *a priori* reason to think that the dark halo must resemble the luminosity distribution. There is evidence that the projected mass and light distributions tend to be aligned ([KKF98]); so the lens galaxy PA might provide a helpful constraint, but it is not yet clear how reliable it is. I have set large errorbars so the code does not use any of these constraints.

The file includes a guess for the faint central image, which has not been observed. I have assumed that the central image will be near the center of the lens galaxy, but sets a large position uncertainty; and I have constrained it to have a flux <5% of the C image at 1σ. However, in the modeling I usually tell the code to ignore the central image (using `omitcore`).

This data file does not include the time delay; see §7.7 for an example of time delays and the Hubble constant.

[1] With isotropic position uncertainties. See §6.2 for a discussion of how to use error ellipses.

7.2 Preliminary models

The first step is to determine the basic parameters for models. For these preliminary models, I often use the source plane χ^2 so I can move through this process quickly. When you use the source plane χ^2, you don't need the tiling (because the code doesn't need to solve the lens equation), so you can set `gridflag=0` to turn of the tiling and speed up the runs.

I often begin by taking a singular isothermal ellipsoid (SIE), fixing the lens at the observed position, and varying the b parameter (the lens mass) together with the ellipticity and orientation. Fixing the lens simply reduces the number of free parameters on a first pass. If you do not know the orientation of the lens, you may want to use `varyone` to explicitly examine a set of orientations, as in this example:

```
# explore PAs, optimizing b and ellipticity;
# fix lens at observed position
set omitcore    = 0.05  # don't care about core image; must go *before* data
data pg1115.dat
set chimode     = 0     # preliminary: use source plane chi^2
set checkparity = 0     # don't worry about parities
set gridflag    = 0     # don't need the tiling
startup 1 1
  alpha 1.0 -0.382 -1.344 0.03 10.0 0.0 0.0 0.0 0.0 1.0
  1 0 0 1 1 0 0 0 0 0
varyone 1 5 -90.0 90.0 19 prelim1
quit
```

The code writes a file called `prelim1.chi` which contains the χ^2 as a function of PA, with the best model summarized in the file `prelim1.best`. (The `varyone` command specifies that the base name for the output files is `prelim1`.) The "best-fit" is very bad, with $\chi^2 = 2616.5$, but it's a place to start.

Next, use the model from `prelim1` to start a new run where you optimize the PA, again using the source plane χ^2 for speed:

```
# now optimize PA
set omitcore    = 0.05
data pg1115.dat
set chimode     = 0
set checkparity = 0
set gridflag    = 0
startup 1 1
  alpha 1.121597 -0.382 -1.344 0.3539533 70.0 0.0 0.0 0.0 0.0 1.0
  1 0 0 1 1 0 0 0 0 0
optimize
quit
```

Allowing the PA to vary yields $\chi^2 = 799.8$. Now the best-fit model is summarized in the file `best.dat`, because that is the file used by the `optimize` command (see Table 6.3). The parameters are:

```
alpha 1.134208e+00 -3.820000e-01 -1.344000e+00 3.498284e-01 6.637265e+01 \\
      0.0 0.0 0.0 0.0 1.0
```

7.3 Elliptical models

Now it's time to get serious about elliptical models, letting the lens position vary and using the image plane χ^2 to see how well the models really do:

```
# start with the best preliminary model and optimize
# all parameters, using the image plane chi^2
set omitcore = 0.05
data pg1115.dat
startup 1 1
  alpha 1.134208 -0.382 -1.344 0.3498284 66.37265 0.0 0.0 0.0 0.0 1.0
  1 1 1 1 1 0 0 0 0 0
optimize
quit
```

Now the best-fit model is

```
alpha 1.124237e+00 -3.779179e-01 -1.362499e+00 3.801617e-01 6.651718e+01 \\
      0.0 0.0 0.0 0.0 1.0
```

and it has $\chi^2 = 784.7$. Note we are now using the image plane χ^2, which is computed with directly observable quantities and is thus better than the source plane χ^2 for indicating how well the model agrees with the data (see §5.1).

Look at the file `best.dat` to get a detailed summary of the model, including the χ^2 broken down into various components, the source parameters, and a summary of how the observed images compare to model images. For example, the line

```
Source #1: (u,v) = (-0.38943, -1.18302), fsrc = 0.2998, h = 1.0000
```

gives the position and flux of the source. (The h parameter indicates the value of the Hubble constant H_0 when time delays are included; see §7.7.)

If you want to view the model graphically, you can use a set of SuperMongo macros in the file `best.sm`, which is automatically generated by the code. Figure 7.1 illustrates the best-fit SIE model for PG 1115+080.

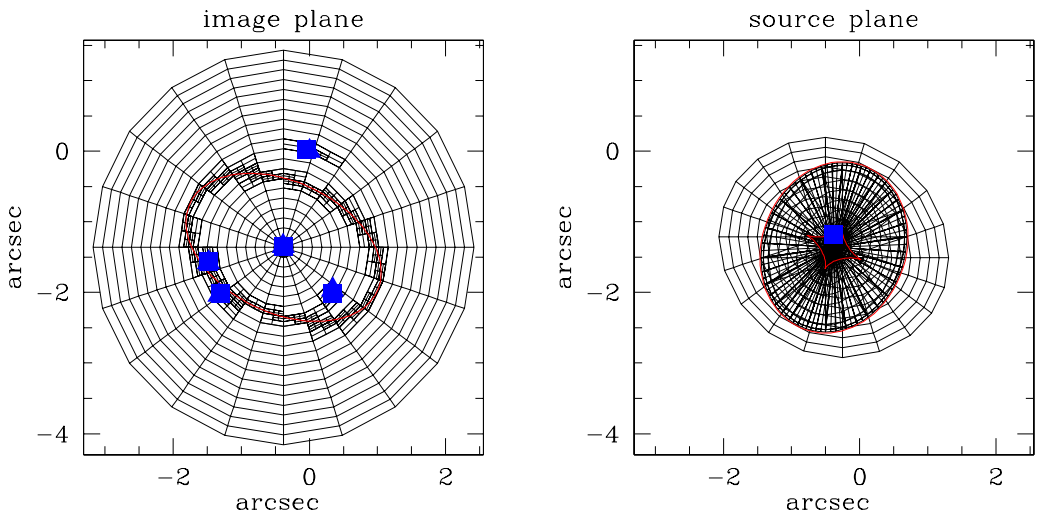


Figure 7.1 – The best-fit SIE model for PG 1115+080. The left-hand panel shows the image plane, with the tiling in black, the critical curve in red, and the images in blue. The right-hand panel source the source plane, with the tiling again in black, the caustics in red, and the source in blue. This figure was produced using the SuperMongo macros in the file `best.sm` that is generated by the code. (Run SuperMongo, run the macros, and execute the macro `plot`.)

7.4 Models with external shear

The SIE model gives a very bad fit, $\chi^2 = 784.7$, so clearly it is oversimplified. To make the model more complicated, the best place to start is the angular structure of the lens model because many lenses seem to require an external tidal perturbation in addition to an ellipsoidal galaxy (e.g., [KKS97]), and because in PG 1115+080 we know there is a group of galaxies around the lens that perturb the lens model (e.g., [KCBL97]; [Ton98]).

When you add a shear you should proceed carefully, because there may be a strong degeneracy between ellipticity and shear that you need to understand (e.g., [KKS97]). Here's one way to proceed. At first you don't know the amplitude or orientation of the shear, nor do you know the ellipticity and orientation of the galaxy in the presence of a shear. The first thing I would do is pick arbitrary value for the ellipticity and shear and try to find the optimum orientations:

```
# first fix ellip/shear and use varytwo to examine
# a range of PAs; use source plane chi^2
set omitcore    = 0.05
data pg1115.dat
set chimode     = 0
set checkparity = 0
set gridflag    = 0
```

```
set restart     = 2
startup 1 1
  alpha 1.124095e+00 -3.777015e-01 -1.362442e+00 0.1 0.0 0.1 0.0 0.0 0.0 1.0
  1 1 1 0 0 0 0 0 0 0
varytwo 1 5 -90.0 90.0 19 1 7 -90.0 90.0 19 run1
quit
```

Note that for now I am using the source plane χ^2 for speed. Also note that I am using the `restart` variable to help ensure that the code finds the minimum of the χ^2 function and doesn't get hung up in a local minimum or a narrow valley (see §6.6).

Next I would take the result of the previous run and use it to start a run where I optimize the orientations, but still keeping the ellipticity and shear fixed:

```
# optimize the PAs, with ellip/shear fixed;
# use source plane chi^2
set omitcore    = 0.05
data pg1115.dat
set chimode     = 0
set checkparity = 0
set gridflag    = 0
set restart     = 2
startup 1 1
  alpha 1.146101e+00 -3.650053e-01 -1.353517e+00 0.1 80.0 0.10 60.0 0.0 0.0 1.0
  1 1 1 0 1 0 1 0 0 0
varytwo 1 4 0.0 0.5 11 1 6 0.0 0.2 11 run2
quit
```

Now that I have optimized all the other parameters with the ellipticity and shear fixed, I would let the ellipticity/shear vary and optimize everything. Here I would use the image plane χ^2:

```
# optimize all; use image plane chi^2
set omitcore = 0.05
data pg1115.dat
startup 1 1
  alpha 1.135596e+00 -3.682313e-01 -1.342225e+00 0.2 -8.867558e+01 \\
        0.1 4.820297e+01 0.0 0.0 1.0
  1 1 1 1 1 1 1 0 0 0
optimize
quit
```

The best-fit model is

```
alpha 1.137422e+00 -3.683225e-01 -1.341511e+00 1.859513e-01 -8.830340e+01 \\
      9.997982e-02 4.945917e+01 0.0 0.0 1.0
```

which has a small-ish ellipticity $e = 0.19$ and a moderate shear $\gamma = 0.10$. The shear has improved the fit substantially: now we get $\chi^2 = 34.3$.

I mentioned above that you want to be wary of a degeneracy between the ellipticity and shear. So I would go back and use `varytwo` to examine the ellipticity/shear plane and look for such a degeneracy:

```
# now look for ellip/shear degeneracy;
# use image plane chi^2
set omitcore = 0.05
data pg1115.dat
set restart  = 2
startup 1 1
  alpha 1.137422e+00 -3.683225e-01 -1.341511e+00 1.859513e-01 -8.830340e+01 \\
        9.997982e-02 4.945917e+01 0.0 0.0 1.0
  1 1 1 1 1 1 1 0 0 0
varytwo 1 4 0.0 0.5 26 1 6 0.0 0.2 21 run4
quit
```

Figure 7.2 shows contours of χ^2 from this run, which show a degeneracy such that models can have a larger ellipticity with a smaller shear, or vice versa. The minimum of the χ^2 function is well-defined, but it is still important to keep the degeneracy in mind when interpreting models.

7.5 Models with the group

The perturbation from the group may not well modeled as an external shear. To make the model more complicated and realistic, the next logical step is to add a mass distribution representing the group. The first thing you might try is a singular isothermal sphere (SIS) representing the common group halo, because it is a simple but not unrealistic mass distribution that adds only three parameters to the total model. (You can try more complicated models later if you want.) You need two parameters for the position of the group, which I usually specify in polar coordinates relative to the lens galaxy, plus one parameter for the mass scale of the group (the b parameter of the SIS).

The first step is to find some reasonable model that includes the group, which you will later optimize. You can get some idea of a relevant model using the models with external shear: the shear angle gives an idea of the direction to the group, and the shear amplitude gives an idea of the group mass. The shear angle was 50°, but because an external shear is degenerate to changing the angle by ±180° the direction to the group could be around 50° or around −130°. The observed group is southwest of the lens galaxy ([KCBL97]; [Ton98]), so guess −130°.

Now you want to get a rough idea of the group position. Here's a run where I use `varytwo` to examine a set of positions, optimizing the lens galaxy and the group mass (b), and using the source

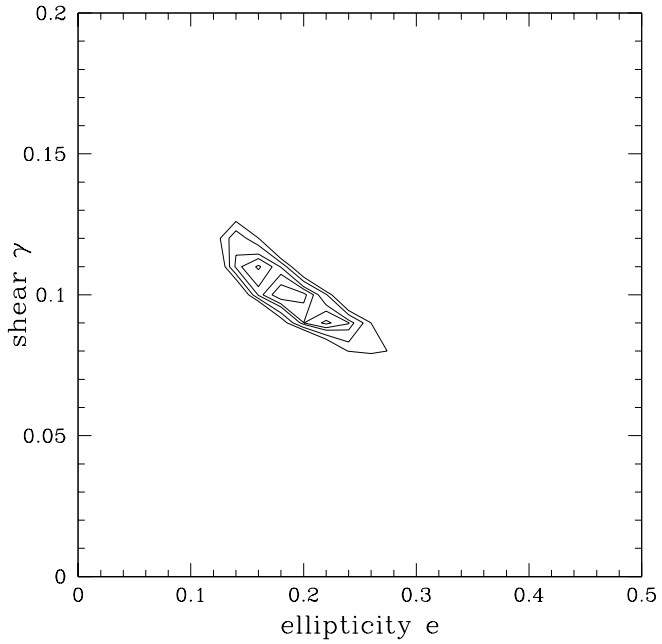


Figure 7.2 – χ^2 contours for PG 1115+080 in the plane of ellipticity e and shear γ. The contours are drawn at $\Delta\chi^2 = 2.30$, 4.61, 6.17, 9.21, and 11.8, which correspond to the 1σ, 90%, 2σ, 99%, and 3σ confidence intervals for two parameters (see [PTVF92]). The contours at low $\Delta\chi^2$ are not well resolved because the plane is sparsely sampled (a 26×21 grid); if I were running "for real" I would sample the plane more densely to remove this effect. Nevertheless, there is a clear degeneracy between the ellipticity and shear.

plane χ^2 since this is just a first cut. In the specification of the model parameters, the first `alpha` model refers to the lens galaxy, while the second refers to the group:

```
# use varytwo to try a range of group positions,
# using source plane chi^2 for speed
set omitcore    = 0.05
data pg1115.dat
set chimode     = 0
set checkparity = 0
set restart     = 2
set gridflag    = 0
set galcoords   = 2  # give group position in polar coordinates
startup 2 1
  alpha 1.138809e+00 -3.683466e-01 -1.340800e+00 1.750000e-01 -8.795244e+01 \\
        0.0 0.0 0.0 0.0 1.0
  alpha 4.0 20.0 -120.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
  1 1 1 1 1 0 0 0 0 0
```

```
  1 0 0 0 0 0 0 0 0 0
varytwo 2 2 10.0 20.0 11 2 3 -140.0 -100.0 11 run1
quit
```

Next I would take the best model from this run and use it to start a run where I optimize everything including the group position; now I use the χ^2 to run "for real":

```
# optimize using image plane chi^2
set omitcore  = 0.05
data pg1115.dat
set galcoords = 2
startup 2 1
  alpha 1.028937e+00 -3.842081e-01 -1.345091e+00 4.412798e-02 2.412278e+01 \\
        0.0 0.0 0.0 0.0 1.0
  alpha 2.114593e+00 1.000000e+01 -1.120000e+02 0.0 0.0 0.0 0.0 0.0 0.0 1.0
  1 1 1 1 1 0 0 0 0 0
  1 1 1 0 0 0 0 0 0 0
optimize
quit
```

The best-fit model has $\chi^2 = 3.59$, so it is quite a good fit to the data. The parameters are:

```
 alpha 1.033458e+00 -3.816041e-01 -1.343972e+00 3.030961e-02 4.081453e+01 \\
       0.0 0.0 0.0 0.0 1.0
 alpha 2.115115e+00 1.032799e+01 -1.137792e+02 0.0 0.0 0.0 0.0 0.0 0.0 1.0
```

Given the galaxy/environment degeneracy seen in the shear models, it's worth looking for degeneracies in groups models. Here is a run using `varytwo` and the image plane χ^2 to determine the range of group positions that give good fits:

```
set omitcore  = 0.05
data pg1115.dat
set galcoords = 2
set restart   = 2
startup 2 1
  alpha 1.033458e+00 -3.816041e-01 -1.343972e+00 3.030961e-02 4.081453e+01 \\
        0.0 0.0 0.0 0.0 1.0
  alpha 2.115115e+00 1.032799e+01 -1.137792e+02 0.0 0.0 0.0 0.0 0.0 0.0 1.0
  1 1 1 1 1 0 0 0 0 0
  1 1 1 0 0 0 0 0 0 0
varytwo 2 2 5.0 20.0 31 2 3 -130.0 -100.0 31 vary_group
quit
```

Figure 7.3 shows χ^2 contours versus the group position from this run. There is a range of group positions that yield good fits, although the range is surprisingly small given that the galaxy dominates

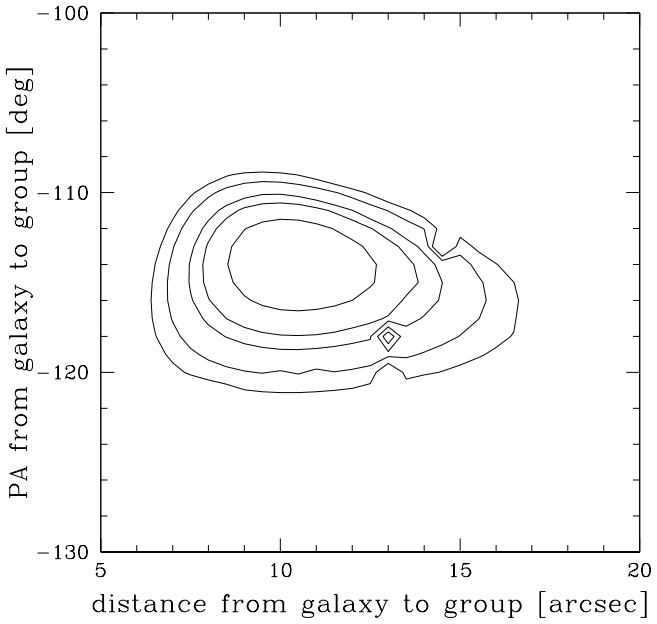


Figure 7.3 – χ^2 contours for PG 1115+080 in the plane of the group position (in polar coordinates). The contours are drawn at $\Delta\chi^2 = 2.30$, 4.61, 6.17, 9.21, and 11.8, which correspond to the 1σ, 90%, 2σ, 99%, and 3σ confidence intervals for two parameters (see [PTVF92]). There are a few funny points, like $(13.0, -118.0)$, where the blip in the χ^2 surface suggests that the code did not converge to the global minimum. If I were running "for real" I would work to clean up such points, perhaps running those models separately and then modifying the χ^2 data file with the updated results.

the lensing potential and the group merely perturbs it. The range of acceptable models must be considered as part of the systematic uncertainties in the lens model.

7.6 Other models

You may want to consider other more complicated model to identify other systematic model uncertainties. For example, you should consider other radial profiles for the main lens galaxy (see [KK97]; [IFK+98]; [KKM01]). You should also consider other models for the group. [KK97] examine models where the group halo is more concentrated than an isothermal sphere. It would also be interesting to examine more complicated models for the group — including the known member galaxies instead of just a common dark matter halo, and allowing the extended halo to depart from spherical symmetry. All of these models could be run in `lensmodel` using the techniques already discussed.

7.7 Using time delays to determine the Hubble constant

Time delays in PG 1115+080 were first measured by [SBB$^+$97], and the standard values are taken from the re-analysis of the data by [Bar97]. Image C is the leading image, followed by the A1/A2 pair and then image B. The time delay between images C and B is the best measured, $\Delta t_{CB} = 25.0 \pm 1.6$ days, and this is the value I use.

You can include time delays in the lens data file if you want to use them to determine the Hubble constant H_0 and further constrain the models. Use the sixth and seventh columns in the data file to specify the delays and their uncertainties. The code ignores any time delay whose errorbar is set to zero. Here is a data file for PG 1115+080 with the B−C time delay; it is identical to the data file in §7.1 above except for the time delay:

```
1                       # 1 lens galaxy
 -0.382 -1.344 0.003    # position
 0.59 1000.             # R_eff observed: 0.59 +/- 0.06
 0.0  1000.             # PA unconstrained in observations
 0.0  1000.             # observed e < 0.07 at 1sigma


1                       # 1 source
5                       # 5 images of the source
 -1.328 -2.037  3.88  0.003  0.78   0.0 0.0  # A1
 -1.478 -1.576 -2.51  0.003  0.50   0.0 0.0  # A2
  0.341 -1.960 -0.65  0.003  0.13  25.0 1.6  # B
  0.0    0.0    1.0   0.003  0.20   0.0 1.6  # C
 -0.383 -1.345  0.0   1000.  0.05   0.0 0.0  # central

# note: use flux errorbars of 20%
```

First I want to determine the value of H_0 implied by the best-fit group model from §7.5. For simplicity, I put the model parameters in a `startup` file called `group.start` (see §4.2). I then do the following run:

```
set omitcore = 0.05
data pg1115_tdel.dat
set galcoords = 2
#
set omega  = 0.3
set lambda = 0.7
set zlens  = 0.31
set zsrc   = 1.72
set hvale  = 1.0e6
#
```

```
startup group.start
optimize
quit
```

Note that this input file specifies the cosmology as well as the lens and source redshifts; the code uses these values to compute the time delay scale factor t_0 (see eq. 23 of the companion paper [Kee01b]). The input file also includes `set hvale=1.0e6`, which tells the code that to use the time delay to determine H_0 with no prior assumption on the value (see §4.3 of the companion paper). When the run is complete, look at the file `best.dat` to get the results. There is a line

```
tscale = 3.255131e+01
```

which gives you the value of the scale factor t_0, in h^{-1} days, for the cosmology and redshifts you specified. The value for H_0 is given by the value of $h = H_0/(100\,\mathrm{km\ s^{-1}Mpc^{-1}})$ in the line

```
Source #1: (u,v) = (1.51556, -2.09162), fsrc = 0.1702, h = 0.4528
```

In other words, this model gives $H_0 = 45\,\mathrm{km\ s^{-1}Mpc^{-1}}$, which is very low compared with standard values, but it is what you get with isothermal+group models for PG 1115+080 ([IFK+98]).

The value for H_0 does not include any errorbars. There is a contribution due to uncertainties in the measured time delay, plus a contribution due to uncertainties in the lens model. The uncertainties in this particular model class (SIE galaxy plus SIS group) can be determined from the run at the end of §7.5, because that run indicates how much you can vary the model parameters and still get an acceptable fit. See [KK97] and [IFK+98] for a detailed discussion of how to determine the uncertainties in H_0 that are associated with uncertainties in the lens model. Finally, §7.6 discusses other classes of models that should be considered, and you should examine how they affect the inferred value of H_0. See [KK97], [IFK+98], and [KKM01] for a discussion of how changing the radial profiles of the galaxy and group affect H_0.

Chapter 8

Warnings and Error Messages

Here is an incomplete list of warnings and error messages that you might encounter while running the code.

- *"Non-converged image"*: The code is having trouble with images very close to critical curves; see §4.5.
- *"Too many steps in numerical integration"*: The code is having trouble with a numerical integral. I hope you don't encounter this one! If you do, contact me with a detailed description of the type of model you were trying to run and the context — including input and data files would be good.
- *"With a point mass at the origin, gridlo1 must be >0"*: The deflection diverges at the location of a point mass, so you must make sure that the code does not try to evaluate the model at that point. The problem is most likely if you have a point mass at the origin; if you have `gridlo1=0`, the code must evaluate the origin to construct the grid. In this situation, simply set `gridlo1` to something small but non-zero (e.g., `gridlo1=1.0e-4`). (If you have a point mass that is not at the origin, chances are low that the code will need to evaluate the model at the exact position of the point mass.)
- *"Grid has more than 180 degrees per angular step"*: The tiling algorithm does not work properly if the grid has more than 180° per angular step, because then some of the tiles may not be convex. Increase the angular resolution. (You would encounter this error only when trying to run with a very low resolution grid, e.g. `ngrid2=2`. You might think of doing this when doing modeling with the source plane χ^2, because then you don't need to waste time computing the tiling. When you don't need the tiling, it is better to turn it off altogether by setting `gridflag=0`; see §4.1.)

Chapter 9

Known Limitations

In general, I believe that the code works correctly. It has been used for several years by different people for different applications, and we have not uncovered any fundamental errors. Nevertheless, I make no guarantee of its accuracy. I encourage you to perform your own tests, and to watch out for results that do not make sense. Please feel free to contact me about any questions you have; my contact information is kept current on the code's website, available via the website for the CfA/Arizona Space Telescope Lens Survey at `http://cfa-www.harvard.edu/castles`.

Here is a list of some limitations that I know exist in the current version of the code.

- *Single lens plane*: The code cannot handle multiple lens planes. Allowing additional lens planes should not be difficult; I just haven't gotten around to it yet.

- *Maps of extended images*: The only existing features for modeling extended images are "ring fitting" for (optical) Einstein rings and "curve fitting" for arcs (see Chapter 5). In both cases you must process a map of extended images to extract data in the form that the code can use. The code cannot take a full map of exteneded images and do pixel-by-pixel modeling.

- *Resolving the cores of galaxies*: The tiling of the image plane uses a polar grid centered on the first lens galaxy (see §4.1), so the grid resolution on this galaxy is intrinsically very good. The same does not hold, however, for mass distributions centered at other positions. I have not come up with a tiling scheme that can *automatically* give comparable resolution to two galaxies simultaneously. The code does use recursive sub-tiling to increase the resolution near additional galaxies (see the companion paper, [Kee01b]), and while the resolution is improved it is still not as good as on the first galaxy. You should check to make sure the resolution is good enough for your purposes; for example, if a second galaxy is near enough that it might produce additional images, you should make sure that its critical curves are well resolved. You can control the resolution by changing the resolution of the top grid, or by controlling the recursive (using the `maxlev` and `gallev` variables; see §4.1).

Chapter 10

Index

All variables and commands,
in alphabetical order:

Bibliography

[Bar96] M. Bartelmann. "Arcs from a universal dark–matter halo profile". *A&A*, 313:697, 1996.

[Bar97] R. Barkana. "Analysis of time delays in the gravitational lens PG 1115+080". *ApJ*, 489:21, 1997.

[BF99] G. Bernstein and P. Fischer. "Values of H_0 from the gravitational lens 0957+561". *AJ*, 118:14, 1999.

[BFFP86] G. Blumenthal, S. Faber, R. Flores, and J. Primack. "Contraction of dark matter galactic halos due to baryonic infall". *ApJ*, 301:27, 1986.

[BGF+96] Y.-I. Byun, C. J. Grillmair, S. M. Faber, E. A. Ajhar, A. Dressler, J. Kormendy, T. R. Lauer, D. Richstone, and S. Tremaine. "The centers of early-type galaxies with HST. II. Empirical models and structural parameters". *AJ*, 111:1889, 1996.

[CCRP01] V. F. Cardone, S. Capozziello, V. Re, and E. Peidipalumbo. "Gravitational lensing potential reconstruction in quadruply imaged systems". *A&A*, 379:72, 2001.

[Cha99] K.-H. Chae. "New modeling of the lensing galaxy and cluster of Q0957+561: Implications for the global value of the Hubble constant". *ApJ*, 524:582, 1999.

[CKMK01] J. D. Cohn, C. S. Kochanek, B. A. McLeod, and C. R. Keeton. "Constraints on galaxy density profiles from strong gravitational lensing: The case of B1933+503". *ApJ*, 554:1216, 2001.

[Dub94] J. Dubinski. "The effect of dissipation on the shapes of dark halos". *ApJ*, 431:617, 1994.

[dV48] G. de Vaucouleurs. "Title?". *Ann d'Ap*, 11:247, 1948.

[Fab89] G. Fabbiano. "X-rays from normal galaxies". *ARA&A*, 27:87, 1989.

[FGS85] E. E. Falco, M. V. Gorenstein, and I. I. Shapiro. "On model-dependent bounds on H_0 from gravitational images: Application to Q0957+561A,B". *ApJ*, 289:L1, 1985.

[FTA+97] S. M. Faber, S. Tremaine, E. A. Ajhar, Y.-I. Byun, A. Dressler, K. Gebhardt, C. Grillmair, J. Kormendy, T. R. Lauer, and D. Richstone. "The centers of early-type galaxies with HST. IV. Central parameter relations". *AJ*, 114:1771, 1997.

[GK02] G. Golse and J.-P. Kneib. "Pseudo elliptical lensing mass model: Application to the NFW mass distribution". *A&A*, 390:821, 2002.

[GL92] A. Gould and A. Loeb. "Discovering planetary systems through gravitational microlenses". *ApJ*, 396:104, 1992.

[HB94] D. W. Hogg and R. D. Blandford. "The gravitational lens system B1422+231: Dark matter, superluminal expansion, and the Hubble constant". *MNRAS*, 268:889, 1994.

[Her90] L. Hernquist. "An analytical model for spherical galaxies and bulges". *ApJ*, 356:359, 1990.

[IFK+98] C. D. Impey, E. E. Falco, C. S. Kochanek, J. Lehár, B. A. McLeod, H.-W. Rix, C. Y. Peng, and C. R. Keeton. "An infrared Einstein ring in the gravitational lens PG 1115+080". *ApJ*, 509:551, 1998.

[Jaf83] W. Jaffe. "A simple model for the distribution of light in spherical galaxies". *MNRAS*, 202:995, 1983.

[JS00] Y. P. Jing and Y. Suto. "The density profiles of the dark matter halo are not universal". *ApJL*, 529:L69, 2000.

[KBLN89] C. S. Kochanek, R. D. Blandford, C. R. Lawrence, and R. Narayan. "The Ring Cycle: An interative lens reconstruction tehnique applied to MG 1131+0456". *MNRAS*, 238:43, 1989.

[KCBL97] T. Kundić, J. G. Cohen, R. D. Blandford, and L. M. Lubin. "Keck spectroscopy of the gravitational lens system PG 1115+080: Redshifts of the lensing galaxies". *AJ*, 114:507, 1997.

[Kee01a] C. R. Keeton. A catalog of mass models for gravitational lensing, 2001. Preprint (astro-ph/0102341).

[Kee01b] C. R. Keeton. Computational methods for gravitational lensing, 2001. Preprint (astro-ph/0102340).

[Kee02] C. R. Keeton. Lensing and the centers of distant early-type galaxies, 2002. ApJ submitted.

[KF99] L. V. E. Koopmans and C. D. Fassnacht. "A determination of H_0 with the CLASS gravitational lens B1608+656. II. Mass models and the Hubble constant from lensing". *ApJ*, 527:513, 1999.

[KFI+00] C. R. Keeton, E. E. Falco, C. D. Impey, C. S. Kochanek, J. Lehár, B. A. McLeod, H.-W. Rix, J. A. Muñoz, and C. Y. Peng. "The host galaxy of the lensed quasar Q0957+561". *ApJ*, 542:74, 2000.

[KHB+97] T. Kundić, D. W. Hogg, R. D. Blandford, J. G. Cohen, and L. M. Lubin. "The external shear acting on gravitational lens B1422+231". *AJ*, 114:2276, 1997.

[KK97] C. R. Keeton and C. S. Kochanek. "Determining the Hubble constant from the gravitational lens PG 1115+080". *ApJ*, 487:42, 1997.

[KK98] C. R. Keeton and C. S. Kochanek. "Gravitational lensing by spiral galaxies". *ApJ*, 495:157, 1998.

[KKF98] C. R. Keeton, C. S. Kochanek, and E. E. Falco. "The optical properties of gravitational lens galaxies as a probe of galaxy structure and evolution". *ApJ*, 509:561, 1998.

[KKM01] C. S. Kochanek, C. R. Keeton, and B. A. McLeod. "The importance of Einstein rings". *ApJ*, 547:50, 2001.

[KKS97] C. R. Keeton, C. S. Kochanek, and U. Seljak. "Shear and ellipticity in gravitational lenses". *ApJ*, 482:604, 1997.

[KM01] C. R. Keeton and P. Madau. "Lensing constraints on the cores of massive dark matter halos". *ApJL*, 549:L25, 2001.

[KN92] C. S. Kochanek and R. Narayan. "LensClean: An algorithm for inverting extended, gravitationally lensed images with application to the radio ring PKS 1830−211". *ApJ*, 401:461, 1992.

[Koc91a] C. S. Kochanek. "The implications of lenses for galaxy structure". *ApJ*, 373:354, 1991.

[Koc91b] C. S. Kochanek. "Systematic effects in lens inversions: $\aleph_1$ exact models for 0957+561". *ApJ*, 382:58, 1991.

[Koc93] C. S. Kochanek. "The analysis of gravitational lens surveys. I. Selection functions and ambiguous candidates". *ApJ*, 419:12, 1993.

[Koc95] C. S. Kochanek. "Evidence for dark matter in MG 1654+134". *ApJ*, 445:559, 1995.

[Koc96] C. S. Kochanek. "Is there a cosmological constant?". *ApJ*, 466:638, 1996.

[Koc02] C. S. Kochanek. "What do gravitational lens time delays measure?". *ApJ*, 578:25, 2002.

[LAB+95] T. R. Lauer, E. A. Ajhar, Y.-I. Byun, A. Dressler, S. M. Faber, C. Grillmair, J. Kormendy, D. Richstone, and S. Tremaine. "The centers of early-type galaxies with HST. I. An observational survey". *AJ*, 110:2622, 1995.

[MBM03] M. Meneghetti, M. Bartelmann, and L. Moscardini. "Cluster cross-sections for strong lensing: analytic and numerical lens models". *MNRAS*, 340:105, 2003.

[MFD+95] S. T. Myers, C. D. Fassnacht, S. G. Djorgovski, R. D. Blandford, K. Matthews, G. Neugebauer, T. J. Pearson, A. C. S. Readhead, J. D. Smith, D. J. Thompson, D. S. Womble, I. W. A. Browne, P. N. Wilkinson, S. Nair, N. Jackson, I. A. G. Snellen, G. K. Miley, A. G. de Bruyn, and R. T. Schilizzi. "1608+656: A quadruple-lens system found in the CLASS gravitational lens survey". *ApJL*, 447:L5, 1995.

[MFP97] A. H. Maller, R. A. Flores, and J. R. Primack. "Inclination effects in spiral galaxy gravitational lensing". *ApJ*, 486:681, 1997.

[MGQ+98] B. Moore, F. Governato, T. Quinn, J. Stadel, and G. Lake. "Resolving the structure of cold dark matter halos". *ApJL*, 499:L5, 1998.

[MnKK00] J. A. Muñoz, C. S. Kochanek, and C. R. Keeton. Cuspy mass models for gravitational lenses, 2000. in preparation.

[MP91] S. Mao and B. Paczyński. "Gravitational microlensing by double stars and planetary systems". *ApJL*, 374:L37, 1991.

[MQG+99] B. Moore, T. Quinn, F. Governato, J. Stadel, and G. Lake. "Cold collapse and the core catastrophe". *MNRAS*, 310:1147, 1999.

[MR93] D. Maoz and H.-W. Rix. "Early-type galaxies, dark halos, and gravitational lensing statistics". *ApJ*, 416:425, 1993.

[MS98] S. Mao and P. Schneider. "Evidence for substructure in lens galaxies?". *MNRAS*, 295:587, 1998.

[NFW96] J. F. Navarro, C. S. Frenk, and S. D. M. White. "The structure of cold dark matter halos". *ApJ*, 462:563, 1996.

[NFW97] J. F. Navarro, C. S. Frenk, and S. D. M. White. "A universal density profile from hierarchical clustering". *ApJ*, 490:493, 1997.

[OLS03] M. Oguri, J. Lee, and Y. Suto. "Arc statistics in triaxial dark matter halos: Testing the collisionless cold dark matter paradigm". *ApJ*, 599:7, 2003.

[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge Univ. Press, New York, second edition, 1992.

[RdZC+97] H.-W. Rix, P. T. de Zeeuw, C. M. Carollo, N. Cretton, and R. P. van der Marel. "Dynamical modeling of velocity profiles: The dark halo around the elliptical galaxy NGC 2434". *ApJ*, 488:702, 1997.

[RFT78] V. C. Rubin, W. K. Ford, and N. Thonnard. "Extended rotation curves of high-luminosity spiral galaxies. IV. systematic dynamical properties, Sa → Sc". *ApJL*, 225:L107, 1978.

[RFT80] V. C. Rubin, W. K. Ford, and N. Thonnard. "Rotational properties of 21 Sc galaxies with a large range of luminosities and radii, from NGC 4605 ($R = 4$ kpc) to UGC 2885 ($R = 122$ kpc)". *ApJ*, 238:471, 1980.

[RGM+00] E. Ros, J. C. Guirado, J. M. Marcaide, M. A. Pérez-Torres, E. E. Falco, J. A. Muñoz, A. Alberdi, and L. Lara. "VLBI imaging of the gravitational lens MG J0414+0534". *A&A*, 362:845, 2000.

[RM01] D. Rusin and C.-P. Ma. "Constraints on the inner mass profiles of lensing galaxies from missing odd images". *ApJL*, 549:L33, 2001.

[SBB+97] P. L. Schechter, C. D. Bailyn, R. Barr, R. Barvainis, C. M. Becker, G. M. Bernstein, J. P. Blakeslee, S. J. Bus, A. Dressler, E. E. Falco, R. A. Fesen, P. Fischer, K. Gebhardt, D. Harmer, J. N. Hewitt, J. Hjorth, T. Hurt, A. O. Jaunsen, M. Mateo, D. Mehlert, D. O. Richstone, L. S. Sparke, J. R. Thorstensen, J. L. Tonry, G. Wegner, D. W. Willmarth, and G. Worthey. "The quadruple gravitational lens PG 1115+080: Time delays and models". *ApJL*, 475:L85, 1997.

[Sch90] T. Schramm. "Realistic elliptical potential wells for gravitational lens models". *A&A*, 231:19, 1990.

[SEF92] P. Schneider, J. Ehlers, and E. E. Falco. *Gravitational Lenses*. Springer, Berlin, 1992.

[SW91] P. Schneider and A. Weiss. "A practical approach to (nearly) elliptical gravitational lens models". *A&A*, 247:269, 1991.

[Ton98] J. L. Tonry. "Redshifts of the gravitational lenses B1422+231 and PG 1115+080". *AJ*, 115:1, 1998.

[WKN96] S. Wallington, C. S. Kochanek, and R. Narayan. "LensMEM: A gravitational lens inversion algorithm using the Maximum Entropy Method". *ApJ*, 465:64, 1996.

[WM97] H. J. Witt and S. Mao. "Probing the structure of lensing galaxies with quadruple lenses: the effect of 'external' shear". *MNRAS*, 291:211, 1997.

[WN93] S. Wallington and R. Narayan. "The influence of core radius on gravitational lensing by elliptical galaxies". *ApJ*, 403:517, 1993.

[WP94] J. Wambsganss and B. Paczyński. "Parameter degeneracy in models of the quadruple lens system Q2237+0305". *AJ*, 108:1156, 1994.

[WTS01] J. S. B. Wyithe, E. L. Turner, and D. N. Spergel. "Gravitational lens statistics for generalized NFW profiles: Parameter degeneracy and implications for self-interacting cold dark matter". *ApJ*, 555:504, 2001.

[Zha96] H.-S. Zhao. "Analytic models for galactic nuclei". *MNRAS*, 278:488, 1996.

[ZP01] H.-S. Zhao and D. Pronk. "Systematic uncertainties in gravitational lensing models: a semi-analytical study of PG 1115+080". *MNRAS*, 320:401, 2001.