Ecole Normale Supérieure de Cachan

Rapport de stage
Licence 3

Institut d'Astrophysique de Paris

Microlentilles gravitationnelles binaires : Algorithmes de minimisation et méthodes bayésiennes

---

Auteur :
Maxime Garnier

Sous la direction de :
Dr. Arnaud Cassan

23 Juin 2014

1 Informations générales

1.1 L'Institut d'Astrophysique de Paris

Photographie du bâtiment de l'IAP

Actuellement dirigé par Francis Bernardeau, l'Institut d'Astrophysique de Paris (IAP) est une Unité Mixte de Recherche (UMR 7095) de l'université Pierre et Marie Curie (UPMC - Paris VI) et du Conseil National de la Recherche Scientifique (CNRS). Il a été fondé en 1936 par le ministre de l'Éducation Nationale de l'époque Jean Zay sous l'impulsion essentielle de Jean Perrin. Depuis 2005, l'IAP est un Observatoire des Sciences de l'Univers (OSU) ce qui lui confère le titre d'Ecole Interne de l'UPMC.

Au moment de mon stage, l'IAP comprend 52 enseignants-chercheurs, chercheurs et ingénieurs de recherche, ainsi que 29 doctorants et 24 post-doctorants, pour un personnel total d'environ 132 personnes.

Ses thèmes de recherche recouvrent les demaines de l'astrophysique et de la physique théorique, incluant la cosmologie, la relativité générale, la formation des grandes structures, l'astrophysique des hautes énergies, l'origine et l'évolution des galaxies, la physique stellaire et les planètes extrasolaires.

L'IAP n'est pas seulement un centre important de recherche, il prend part à de multiples activités d'enseignement et de formation. En effet, il héberge le Master 2 *"Astronomie, Astrophysique et Ingénierie Spatiale"* et est l'antenne UPMC de l'École doctorale d'astronomie et d'astrophysique d'Île-de-France (ED 127).

1.2 L'équipe physique stellaire, planétaire et planètes extra-solaires

L'équipe physique stellaire, planétaire et planètes extra-solaires est l'équipe intégrée durant ce stage. Dirigée par Arnaud Cassan, ses thèmes de recherches incluent la recherche de planètes extrasolaires *via* les méthodes des microlentilles gravitationnelles, des transits et des vitesses radiales ainsi que la physique du Soleil et des autres étoiles.

Le travail réalisé lors de ce stage a nécessité l'étroite collaboration d'Arnaud Cassan et de Clément Ranc (son doctorant).

2 Optimisation de l'ajustement de courbes de lumières obtenue par effet de microlentille gravitationelle

2.1 Généralités

La déviation de rayons lumineux par un corps massif est prévu par la Relativité Générale d'Einstein en 1915, et a été mise en évidence pour la première fois lors d'une éclipse solaire de 1919. En 1936, Einstein montre qu'une étoile peut servir de loupe gravitationnelle, et produire plusieurs images d'une autre étoile lointaine; c'est l'effet de lentille gravitationnelle. Appliqués à des lentilles composées d'une étoile ou d'un système planétaire, ces résultats montrent qu'il est possible de détecter des planètes extra-solaires. Dans ce cas, on parle de *microlentille gravitationnelle*.

Ces événements sont rares car environ une étoile sur un million produit un événement de microlentille à un instant donné en direction du Centre Galactique. Les observations se font en direction du Bulbe Galactique ou des nuages de Magellan (pour la recherche de matière noire). La configuration usuelle d'une observation est présentée sur la Fig. 1. Si $D_{\rm L}$ et $D_{\rm S}$ sont respectivement les distances entre l'observateur et la lentille et entre l'observateur et la source on a typiquement $D_{\rm L} \approx 1$ - 8 kpc et $D_{\rm S} \approx 8$ kpc avec pc = 3,2616 a.l.. Le parsec, de symbole pc, est une unité de mesure des distances plus pratique à utiliser

que l'année-lumière. Les principales collaborations de suivi d'évènements de microlentilles sont OGLE (Optical Gravitational Lensing Experiment), PLANET (Probing Lensing Anomalies NETwork) et MOA (Microlensing Observations in Astrophysics).

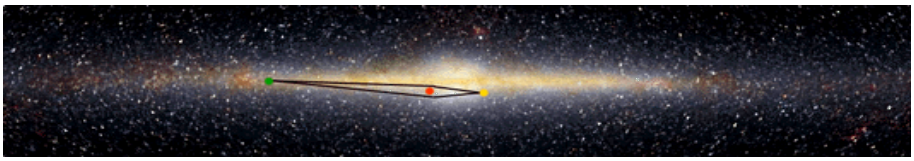


Figure 1: Configuration typique d'une observation (d'après le site web de la collaboration MOA). Point vert : observateur, point rouge : lentille et point jaune : source.

Cette technique a déjà permis de découvrir des planètes extra-solaires, la première découverte a eu lieu en 2003 (OGLE 2003–BLG–235/MOA 2003–BLG–53), la première super-Terre glacée en 2005, OGLE 2005-BLG-390Lb (5 fois la masse de la Terre, à 2,3 Unité Astronomiques) et près de 30 planètes (géantes gazeuses, super-Terres, Neptunes). La planète la moins massive découverte avec cette méthode fait seulement trois fois la Terre. Elle a aussi permis d'estimer l'abondance moyenne des planètes, pour des orbites entre 0.5 et 10 UA et des masses entre 5 fois la Terre et 10 fois Jupiter. En effet, Cassan *et al.* (2012) ont montré par une étude statistique qu'il existait au moins une planète par étoile dans la Voie Lactée.

2.2 Modélisation

La relation entre la position de la source (étoile en arrière-plan) et ses images (on montre qu'il y en a 3 ou 5 selon la position de la source, le nombre d'images est noté $N_{\rm im}$ par la suite) est donnée par l'*équation des lentilles binaires* si la lentille est composée de deux objets (*ex.* étoile + planète).

$$\zeta = z - \frac{1}{1+q}\left(\frac{1}{\overline{z}} + \frac{q}{\overline{z}+s}\right) \tag{1}$$

où s est la *séparation projetée sur le plan du ciel* entre les deux objets de la lentille et q est le rapport de masse.

Cette équation utilise un formalisme complexe pour repérer la position de la source ζ et celle de ses images z dans les plan-lentille et plan-source. Hautement non-linéaire, cette équation n'est pas soluble analytiquement.

Les grandeurs ζ , z et s sont exprimées en unité du rayon d'Einstein $R_{\rm E} = D_{\rm L}\sqrt{\frac{D_S - D_L}{D_S}\frac{4GM}{c^2}}$[1].

On peut définir une amplification, μ, pour une source ponctuelle (l'observateur reçoit plus de lumière qu'il "ne devrait") :

$$\mu = \sum_{j=1}^{N_{\rm im}} \frac{1}{|1 - |W_2(z_j)|^2|} \tag{2}$$

$$W_k(z) = \frac{1}{1+q}\left(\frac{1}{z^k} + \frac{q}{(z+s)^k}\right) \tag{3}$$

On définit les caustiques (*resp.* courbes critiques) comme étant le lieu des points ζ (*resp.* z) où l'amplification est formellement infinie. Elles ne dépendent que de s et q. En 1990, Witt (1990) montre qu'elles sont paramétrées par ϕ qui varie dans $[0; 2\pi]$ selon l'équation :

$$\frac{1}{1+q}\left(\frac{1}{z^2} + \frac{q}{(z+s)^2}\right) = e^{-i\phi} \tag{4}$$

[1]Les grandeurs d'Einstein, indicées E, sont relatives au cas des lentilles uniques, et c est la célérité de la lumière dans le vide, M la masse totale de la lentille et G la constante de gravitation de Newton.

On montre qu'il y a **bijection** $z \leftrightarrow \zeta$ sur les caustiques/courbes critiques. Les équations des lentilles (Eq. (1)) et de Witt (Eq. (4)) se résolvent en les mettant sous la forme d'une équation polynomiale de degré 5 et 4 respectivement. Un exemple de caustiques/courbes critiques est donné sur la Fig. 2.

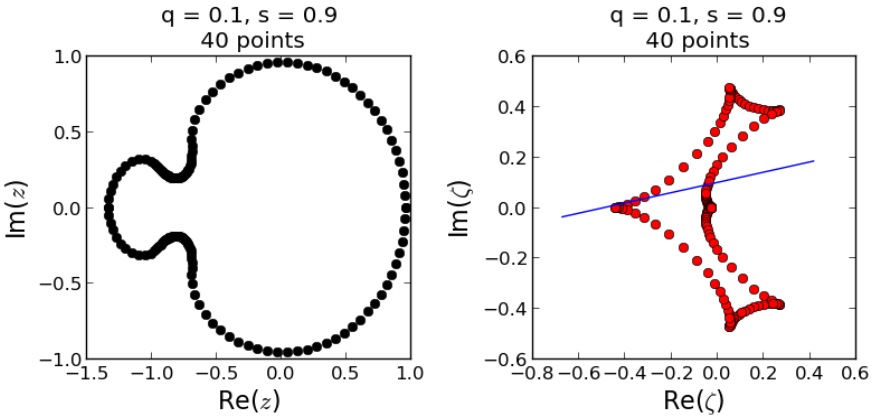
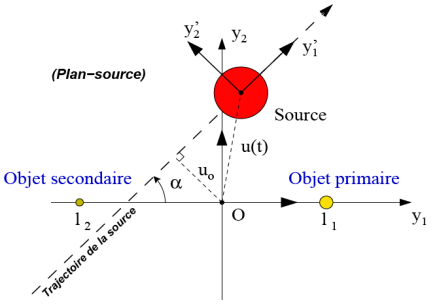
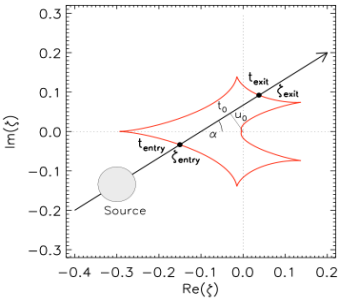


Figure 2: A gauche : courbe critique. A droite : caustique. $s = 0.9\ R_{\rm E}$ et $q = 0.1$

Les courbes de lumière représentent l'évolution temporelle de la magnitude observée en fonction du temps. La magnitude est une échelle *inversée* du flux qui est lié à l'amplification par la relation $\phi = \mu(t)\phi_s + \phi_b$, où ϕ_s est le flux de la source et ϕ_b le flux de "blending" (*i.e.* le flux des autres objets proches). Ainsi, pour pouvoir les modéliser, il faut paramétrer le mouvement de la source. Deux choix sont possibles : une paramétrisation "classique" et une paramétrisation "caustic-crossing" ("traversée de caustique") introduite par Cassan en 2008.



Paramétrisation classique. *Cf.* thèse A. Cassan (2005).



Paramétrisation "Caustic-crossing". *Cf.* A. Cassan (2008).

$$\zeta(t) = \left(\frac{t - t_0}{t_{\rm E}} + i u_0 + \delta\zeta(t) \right)$$

$$\zeta(t) = \frac{\zeta_{\rm out} - \zeta_{\rm in}}{t_{\rm out} - t_{\rm in}}(t - t_{\rm in}) + \zeta_{\rm in} + \delta\zeta(t)$$

Les effets de parallaxe (*i.e.* les effets dûs à la rotation de la Terre autour du Soleil) sont représentés par $\delta\zeta(t)$. Ce paramètre est surtout utile pour modéliser les évènements longs. L'avantage de la seconde paramétrisation est qu'elle utilise les paramètres $t_{\rm in}$ et $t_{\rm out}$ qui peuvent être facilement contraints par la courbe de lumière.
On peut aisément étendre ces résultats au cas d'une source étendue dans l'approximation hexadecapolaire où on approxime un disque par 13 points judicieusement placés (*cf.* A. Gould, 2008).

3 Travail réalisé

A terme, l'objectif du travail amorcé ici est de permettre l'optimisation de l'ajustement des courbes de lumière. Cette stratégie d'optimisation est basée sur le calcul des dérivées analytiques de l'amplification

par rapport aux paramètres du modèle. En notant θ un paramètre quelconque, les expressions générales sont les suivantes :

$$\frac{\partial \mu_j}{\partial \theta} = \frac{2\varepsilon}{\left(1-|W_2|^2\right)^2} \mathfrak{Re}\left[\overline{W_2}\,\frac{\partial W_2}{\partial \theta}\right] \tag{5}$$

$$\frac{\partial \mu_j}{\partial \theta} = \frac{-4}{\left|1-|W_2|^2\right|^3} \mathfrak{Re}\left[\overline{W_2} W_3 \left(\frac{\partial \zeta}{\partial \theta} - \overline{W_2}\overline{\frac{\partial \zeta}{\partial \theta}}\right)\right] \tag{6}$$

On utilise l'une ou l'autre expression selon les cas.

Le travail réalisé a consisté en l'implémentation de ces expressions pour les deux paramétrisations, pour une source ponctuelle ou étendue, leur test (erreurs de calcul, domaine de validité) et la vérification de ces calculs. La majeure partie des programmes a été rédigée dans le langage Python.

3.1 Implémentation des dérivées analytiques

L'implémentation numérique que j'ai effectuée est la suivante : on calcule la position du point-source, on calcule ses images, on calcule les dérivées $\partial\mu_j/\partial\theta$ puis on les somme car chaque image contribue à l'amplification totale. Dans la paramétrisation classique, cette implémentation ne pose pas de problèmes.

Cependant, pour la paramétrisation "caustic-crossing", il a fallu que je construise une **bijection** $\phi \leftrightarrow z \leftrightarrow \zeta$. En effet, l'Eq. (4) étant de degré 4, elle possède 4 solutions, il a donc fallu remarquer que chaque solution appartient à une branche *différente* des caustiques et utiliser l'unicité $z \leftrightarrow \zeta$ sur ces courbes.

J'ai mis en place cette construction et les courbes résultantes sont tracées sur la Fig. 3.

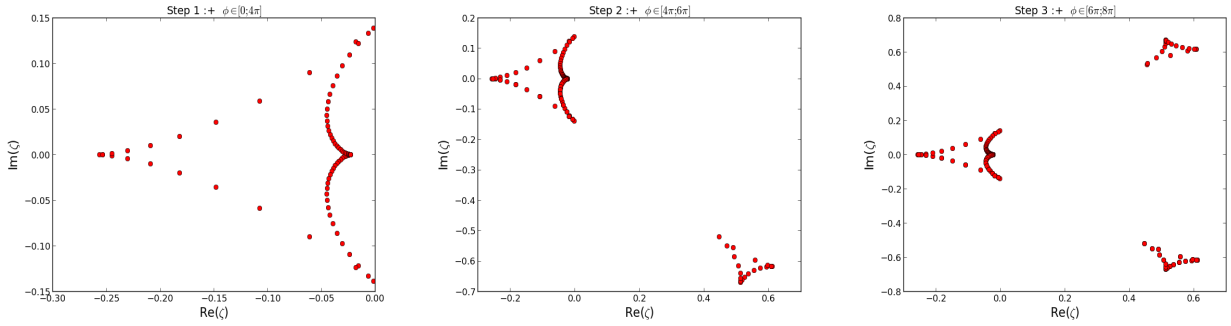


Figure 3: Construction progressive des caustiques.

J'ai implémenté et tracé ces dérivées analytiques dans les deux paramétrisations (classique et "caustic-crossing") pour une source ponctuelle et pour une source étendue dans l'approximation hexadecapolaire (voir Fig. 4).

3.2 Test des dérivées analytiques

Avant de continuer, il a fallu que je m'assure de la validité des expressions analytiques. Pour cela, on les a comparées aux dérivées numériques $\Delta\mu/\Delta\theta$ où $\Delta\theta \ll 1$. Encore une fois, la réalisation de ce test ne pose pas de problèmes particulier pour la paramétrisation classique. Dans la paramétrisation "caustic-crossing", il a fallu faire attention au fait que la plus petite variation possible pour $\theta = \phi_{\text{in/out}}$ était la valeur de l'échantillonnage des caustiques, $d\phi$. (figures).

Le second test a consisté en la linéarisation de l'amplification : $\mu(\theta+d\theta) \approx \mu(\theta)+\frac{\partial\mu}{\partial\theta}d\theta$ et en la comparaison de cette approximation avec la valeur exacte $\mu(\theta + d\theta)$. Ce second test permet d'estimer le domaine de validité de l'approximation de l'amplification.

J'ai effectué ces deux tests pour les deux paramétrisations pour une source ponctuelle.

Sur la Fig. 4 sont tracés trois exemples de courbes obtenues par les différents programmes.

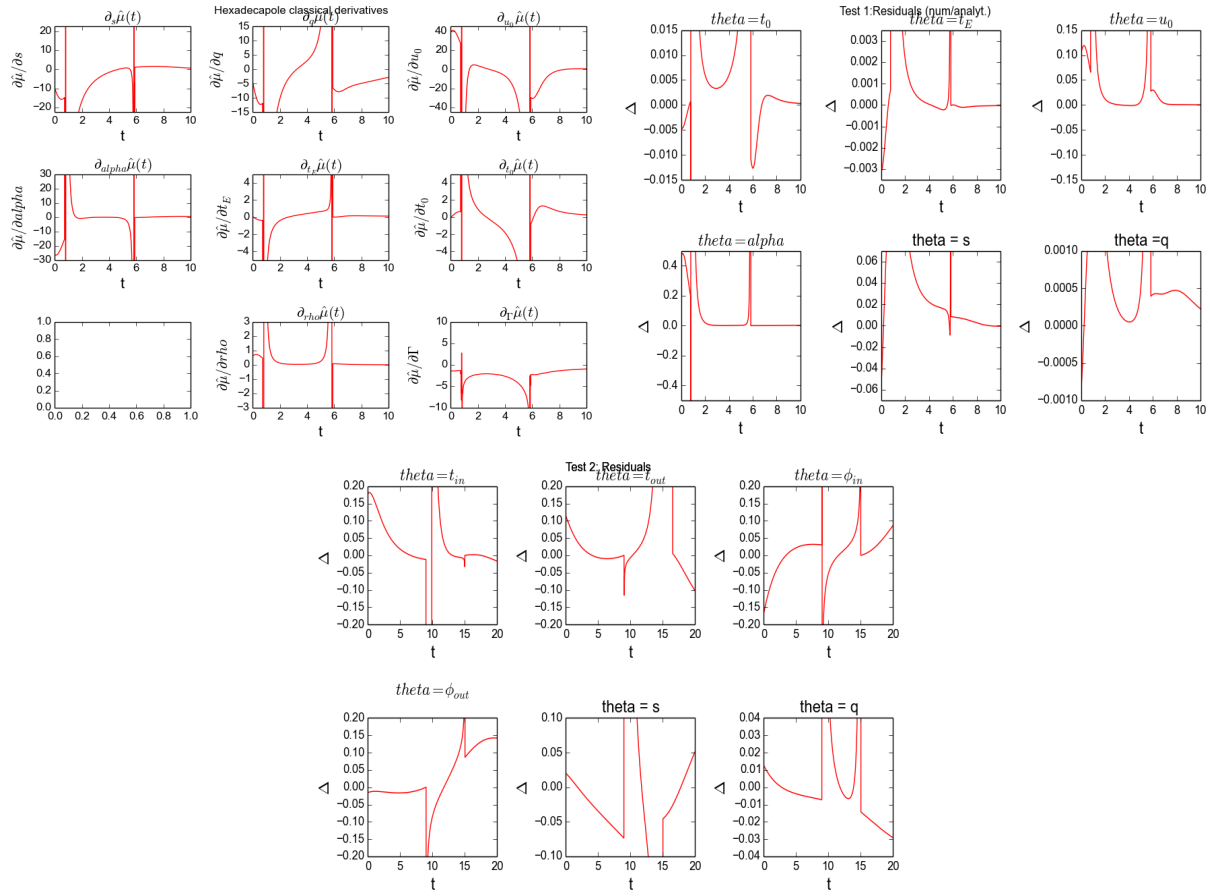


Figure 4: En haut à gauche, tracé des dérivées analytiques pour une source étendue ($\rho = 0.005$) dans la paramétrisation classique ($s = 1.1$, $q = 0.1$, $u_0 = 0.1$, $t_0 = 6.0$, $t_\mathrm{E} = 9.0$, $\alpha = 0.2$). En haut à droite, comparaison numérique/analytique pour une source ponctuelle dans la paramétrisation classique (même trajectoire). En bas, résidus de la linéarisation de l'amplification pour une source ponctuelle dans la paramétrisation "caustic-crossing" ($s = 1.1$, $q = 0.1$, $\phi_\mathrm{in} = 17.0$, $\phi_\mathrm{out} = 7.0$, $t_\mathrm{in} = 9.0$, $t_\mathrm{out} = 15.0$).

3.3 Stratégie d'optimisation

La première étape pour envisager une optimisation a été la génération de données "expérimentales". Pour ce faire, on introduit des erreurs gaussiennes. On écrit que le point expérimental à chaque date t suis une densité de probabilité gaussienne centrée sur la valeur prévue par le modèle. J'ai réalisé cette opération. (graphe si place...)

Pour trouver le meilleur modèle correspondant aux données, plusieurs méthodes d'optimisation ont été envisagées mais elles reposaient toutes sur des méthodes bayésiennes. Cette approche repose sur le *théorème de Bayes* qui permet de déterminer la probabilité *a posteriori* d'un jeu de paramètres en connaissant sa probabilité a priori (ou, en anglais, *prior*) et sa vraisemblance $\mathcal{L}$ [2]. Certains priors ont été déterminés par Cassan *et al.* (2010). Pour maximiser la probabilité *a posteriori*, on a envisagé deux possibilités. La première est la réalisation d'un algorithme *Markov Chain Monte-Carlo* (MCMC) et la seconde est l'application d'une optimisation par méthode de Newton. Cependant, les deux méthodes souffrent du fait que les paramètres introduits jusqu'à présent étaient corrélés. On peut envisager de les décorréler en orthogonalisant l'espace des paramètres *via* un procédé de Gram-Schmidt comme suggéré par Kains (2010).

J'ai donc mis en place un programme qui orthonormalise l'espace des paramètres pour chaque choix

[2] Dans le cadre d'erreurs gaussiennes, minimiser le χ^2 et maximiser la vraisemblance $\mathcal{L}$ sont des approches équivalentes car $\mathcal{L} = \exp(-\chi^2/2)$.

de paramétrisation[3].

Par ailleurs, j'ai mis en place la méthode de minimisation du χ^2 par méthode de Newton via la routine Python `scipy.optimize.minimize` avec la méthode `Newton-CG` qui allie méthode de Newton et du Gradient Conjugué. Pour cela, il a fallu calculer les dérivées du χ^2 par rapport aux différents paramètres grˆace aux dérivées analytiques de l'amplification μ. Là encore, il a fallu réaliser deux programmes différents.

D'un point de vue purement numérique, il a fallu optimiser la recherche des racines de l'équation des lentilles. Pour ce faire, j'ai interfacé une routine écrite en Fortran par Skowron et Gould (2012) en Python grâce au module `f2py`. Des tests ont été menés et cette routine est beaucoup plus rapide que la routine générale `numpy.roots()` de Python[4].

4 Conclusion et perspectives

Ce stage m'a permis de vivre une première expérience de recherche avec ses hauts, ses bas et ses illuminations. En effet, on apprend que tout travail de recherche prend du temps et que l'organisation est importante, les résultats n'étant pas connus à l'avance. Un des principaux bénéfice de ce stage à été l'apprentissage de cette organisation ainsi que de la hiérarchisation des programmes informatiques qui tendaient à s'allonger avec le temps. Il a aussi fallu que je construise des "petits" programmes afin de visualiser les caustiques, courbes critiques, trajectoires et images de sources étendues. J'ai ainsi appris beaucoup du point de vue informatique (interfaçage, travail en réseau, ...). Il ne faut pas oublier qu'il y a eu une phase de réflexion sur l'orientation à donner au projet, j'ai effectué quelques recherches bibliographiques qui m'ont permis de mieux comprendre le domaine de l'optimisation numérique (différents algorithmes et méthodes) et de la validation statistique de modèles.

Ce stage a aussi permis une insertion, voire une immersion, dans un laboratoire de recherche avec ses nombreuses propositions de conférences, tout en cotoyant des chercheurs, d'autres stagiaires (L3, M2) et des doctorants dans des domaines différents. Particulèrement, les conférences auxquelles j'ai assisté balayaient un large spectre de spécialités, de la gravité modifiée (MOND, physique théorique) aux trous noirs en passant par l'astrophysique des hautes énergies, la cosmologie (modèle ΛCDM incluant matière et énergie noires), et, bien sûr, les exoplanètes. J'ai particulièrement apprécié les conférences hebdomadaires données par les doctorants dans le cadre du YMCA (Young and Motivated Cluster of Astronomers) car elles présentent un travail en cours de réalisation par de jeunes chercheurs. Cette émulation, scientifique ou non, permanente, m'a beaucoup plu et me confirme dans mon choix d'orientation vers la recherche.
Mon stage se prolongera quelques temps après la date officielle ce qui me permettra de commencer à mettre en place l'algorithme MCMC et de finaliser l'optimisation par méthode de Newton. Une fois que les programmes seront prêts, on pourra envisager de les tester sur des données expérimentales réelles.

5 Remerciements

Je tiens à remercier tout particulièrement Arnaud Cassan pour m'avoir proposé ce stage et m'avoir fait confiance pour travailler en autonomie, Clément Ranc, son doctorant, pour ses multiples discussions fructueuses et pleines d'informations, pour ses conseils et son aide. Je souhaite aussi remercier vivement les doctorants et stagiaires de l'IAP de m'avoir si bien intégré. Une mention spéciale est cependant nécessaire pour Jill, étudiante en stage de M2, partenaire de labeurs informatiques, pour sa bonne humeur, ses conseils et ses explications.

6 Bibliographie succinte

- Cassan, A. : *L' effet de microlentille gravitationnelle dans la recherche de planètes extra-solaires et dans le sondage d'atmosphères d'étoiles géantes du Bulbe*, (thèse) 2005, *An alternative parameterisa-*

[3]En effet, il a fallu traiter séparément les deux paramétrisations car en "caustic-crossing", $\zeta(t)$ dépend explicitement de $\zeta_{\rm in}$ et $\zeta_{\rm out}$ qui ne sont pas de "vrais" paramètres car liés à $\phi_{\rm in}$ et $\phi_{\rm out}$.

[4]Skowron et Gould ont en effet prévu une méthode spécifique à l'équation des lentilles.

tion for binary-lens caustic-crossing events, Astronomy and Astrophysics, 2008, *Bayesian analysis of caustic-crossing microlensing events*, Astronomy and Astrophysics,(*et al.*) 2010, *One or more bound planets per Milky Way star from microlensing observations*, Nature, (*et al.*) 2012.

- Gould, A., *Hexadecapole Approximation in Planetary Microlensing*, The Astrophysical Journal, 2008, *General complex polynomial root solver*, (with Skowron, J.) 2012.
- Kains, N., *The Solar System in Perspective : from Debris Discs to Extrasolar Planets*, (thèse) 2010.

7 Annexe

Cette partie a pour but d'expliciter les programmes que j'ai écrit durant ce stage ainsi que leur fonctionnement. Je m'efforce de fournir tous les éléments permettant leurs réutilisation future par d'autres personnes que moi. Certains programmes sont encore en cours d'écriture, mais leurs états au 23 Juin sont détaillés.

A Routines de recherche des racines d'un polynôme à coefficients complexes

A.1 numpy.roots *vs* cmplx_roots_sg.f90

Etant donné que l'objectif visé est l'optimisation du code, il faut chercher à améliorer sa rapidité. Un des points sur lesquels on peut influer est la résolution des équations polynômiales. En effet, on résout de très nombreuses fois le polynôme associé à l'équation des lentilles (au moins une fois par pas de temps pour une source ponctuelle et 13 fois pour une source étendue sans l'approximation hexadecapolaire). Ainsi, la fonction `numpy.roots()` commence à montrer ses limites.
Une autre solution est d'utiliser la routine mise en oeuvre par Skowron et Gould qui permet de résoudre beaucoup plus rapidement des polynômes à coefficients complexes. Les auteurs ont même écrit une routine spécialement faite pour l'équation des lentilles.
La figure suivante récapitule les différents temps d'éxécution des routines `numpy.roots()`, `cmplx_roots_gen()` et `cmplx_roots_5()` (routine spécialement écrite pour l'équation des lentilles).

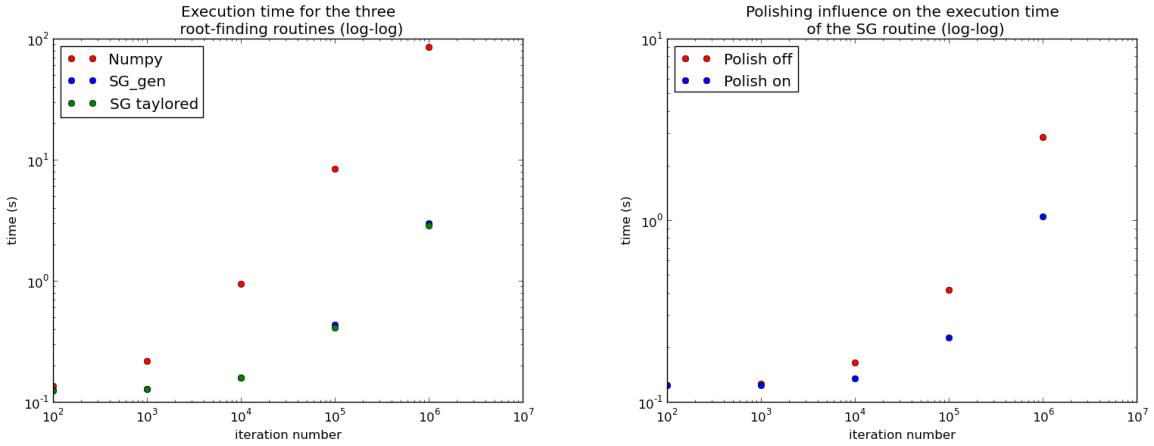


Figure 5: Comparaison de différentes routines. A gauche, comparaison `numpy.roots()`, `cmplx_roots_gen` et `cmplx_roots_5`. A droite comparaison `cmplx_roots_5` avec et sans polish.

La figure de droite compare la même routine avec activation ou non de l'option `polish` .
Le test a été effectué sur le même polynôme d'ordre 5 pour les trois routines.

A.2 Interfaçage Python - Fortran *via* f2py

La routine de Skowron et Gould présente un large gain de temps. Cependant elle a été écrite en Fortran. Ainsi, pour pouvoir l'utiliser, il a fallu interfacer les langages Python et Fortran grâce à un outil appelé `f2py`. `f2py` permet de créer une bibliothèque (**.so** pour "shared object") Python à partir de la routine Fortran.
Avant toute chose, il faut vérifier que `python-dev` est installé, sinon une erreur du type : "il manque Python.h" sera renvoyée.

Puis il faut compiler le fichier Fortran `cmplx_roots_sg.f90` (routine de Skowron et Gould) avec la commande :

`f2py -c cmplx_roots_sg.f90 -m cmplx_roots`

où l'argument après le -m donne le nom de la bibliothèque créée. Cette commande créé le fichier : `cmplx_roots.so` qu'il faut appeler dans Python avec `import cmplx_roots`.

Le programme `interf_solver.py` utilise la routine de Skowron et Gould en Python. Il faut faire attention à deux points :

- Le type des tableaux dans la routine Fortran est `complex*16` (8 bits pour la partie réelle et 8 bits pour la partie imaginaire) donc il faut mettre les éléments de tableaux python en complex128 (64 octets donc 8 bits pour la partie réelle et 8 bits pour la partie imaginaire). Par exemple : `array = numpy.array([0., 0., 0.,], dtype = np.complex128)`.
- L'ordre des tableaux demandé par `numpy.roots()` et différent de celui de `cmplx_roots_sg.f90`. En effet, pour `numpy`, le premier élément du tableau est le coefficient de degré le plus élevé alors que pour Skowron et Gould c'est le terme de degré le plus bas.

B Inversion numérique de l'équation des lentilles binaires

Comme on l'a vu, l'Eq. (1) peut se mettre sous la forme d'une équation polynomiale complexe de degré 5 qu'on peut aisément résoudre numériquement grâce à des routines intégrées comme `numpy.roots()` pour Python. Cependant, il faut faire attention en résolvant cette équation car quand la source est en dehors des régions intérieures aux caustiques, elle n'a que trois images alors que le degré du polynôme est toujours 5. Ainsi, il faut inclure une étape de vérification des solutions dans le code, c'est-à-dire qu'à chaque pas de temps, on vérifie si le ζ associé a chaque z trouvé est égal au ζ initial à une précision de 10^{-5} près.

Ceci est réalisé dans la fonction `lense_equation_roots(zeta, s, q, eps)` :

```
def lense_equation_roots(zeta, s, q, eps):


# Definition of the polynom coefficients
    coefs = [ (1+q)**2 * (s+zeta.conjugate()) * zeta.conjugate(), \
(1+q)*(s*(q-abs(zeta)**2 * (1+q)) + (1+q)*((1+2*s**2) - abs(zeta)**2 + 2*s*zeta.
    conjugate())* zeta.conjugate()), \
(1+q)*(s**2 * q - s*(1+q)*zeta + (2*s+s**3 * (1+q) + s**2 * (1+q)*zeta.conjugate())
    *zeta.conjugate() - 2*abs(zeta)**2 * (1+q)*(1+s**2+s*zeta.conjugate())), \
-(1+q)*(s*q + s**2 *(q-1)*zeta.conjugate() + (1+q+s**2 *(2+q))*zeta + abs(zeta)**2
    *(2*s*(2+q)+s**2 *(1+q)*(s+zeta.conjugate()))), \
-s*(1+q)*((2+s**2)*zeta + 2*s*abs(zeta)**2) - s**2 *q , \
-s**2 * zeta ]

    res = []

    # Find the roots
    roots = np.roots(coefs)

    # Use the lense equation to compute the zeta associated with each root
    zeta_prime = [lense_equation(roots[i], s, q) for i in xrange(0, len(roots))]
```

```
    delta = [abs(zeta - zeta_prime[i]) for i in xrange(0, len(roots))]

    # Verification of the true solutions. eps = 1e-5 (Cf Clement Ranc).
    for j in xrange(0, len(roots)):
        if delta[j] < eps:
            res.append(roots[j])


    return res
```

Listing 1: Routine d'inversion de l'équation des lentilles avec `numpy.roots`.

Et la même fonction mais utilisant le solver de Skowron et Gould :

```
def lense_equation_roots(zeta, s, q, eps):

  #If false sets the polish off. If True sets the polish on.

  a = bool(0)
  res = []

  # Initialize the array of the roots
  roots = np.array([0., 0., 0., 0., 0.], dtype = np.complex128)


    coefs = np.array([-s**2 * zeta, -s*(1+q)*((2+s**2)*zeta + 2*s*abs(zeta)**2) - s
        **2 *q, \
 -(1+q)*(s*q + s**2 *(q-1)*zeta.conjugate() + (1+q+s**2 *(2+q))*zeta + abs(zeta)**2
     *(2*s*(2+q)+s**2 *(1+q)*(s+zeta.conjugate()))),\
 (1+q)*(s**2 * q - s*(1+q)*zeta +\
 (2*s+s**3 * (1+q) + s**2 * (1+q)*zeta.conjugate())*zeta.conjugate() - 2*abs(zeta)
    **2 * (1+q)*(1+s**2+s*zeta.conjugate())),\
 (1+q)*(s*(q-abs(zeta)**2 * (1+q)) + (1+q)*((1+2*s**2) - abs(zeta)**2 + 2*s*zeta.
    conjugate())* zeta.conjugate()),\
 (1+q)**2 * (s+zeta.conjugate()) * zeta.conjugate() ], dtype = np.complex128)


  # Call the Skowron and Gould routine.
  # The roots array contains the roots of the polynom


  cmplx_roots.cmplx_roots_5(roots, coefs, a)

  # Compute the zeta associated to the roots
  zeta_prime = [lense_equation(roots[i], s, q) for i in xrange(0, len(roots))]

  # Compute the difference between zetas
  delta = [abs(zeta - zeta_prime[i]) for i in xrange(0, len(roots))]

  # Keep only the "real" images.
  for j in xrange(0, len(roots)):
    if delta[j] < eps:
      res.append(roots[j])


  return res
```

Listing 2: Routine d'inversion de l'équation des lentilles avec `cmplx_roots_sg.f90`.

C Liste des programmes - Notice d'utilisation

Cette section a pour vocation à expliciter les programmes qui peuvent s'avérer utiles pour la suite ainsi que la manière de s'en servir.

1. `caustic_unique.py` : Échantillonnage des caustiques pour la paramétrisation “caustic-crossing”.
2. `caustics.py` : Tracé de la trajectoire de la source, des caustiques et des courbes critiques.
3. `PS_magnification.py` : Tracé de courbes d’amplification pour une source ponctuelle.
4. `hexadecapole_magnification.py` : Tracé de courbes d’amplification pour une source étendue dans l’approximation hexadecapolaire.
5. `test_derivatives.py` : Tracés et tests des dérivées pour une source ponctuelle dans les deux paramétrisations.
6. `hexadecapole_derivatives_full.py` : Tracés des dérivées pour une source étendue (approximation hexadecapolaire) dans les deux paramétrisations.
7. `exp_data.py` : Simulation de données expérimentales avec erreurs gaussiennes.
8. `ESBL_opt_ortho.py` : Orthonormalisation de l’espace des paramètres pour la paramétrisation classique. La même tâche est réalisée pour la seconde paramétrisation dans `ESBL_opt_ortho_cc.py`.
9. `ESBL_opt_newton_bis.py` : Minimisation du χ^2 par la méthode Newton-CG pour la paramétrisation classique. Pour la paramétrisation “caustic-crossing” utiliser le programme avec `._cc.py`.

C.1 Échantillonnage des caustiques pour la paramétrisation “caustic-crossing”

Le programme `caustique_unique.py` réalise la bijection entre le paramètre ϕ de Witt, z et ζ. Ce programme ne nécessite aucune modification particulière. Afin de faire varier l’échantillonnage des caustiques (variable `dphi`) faire varier, le nombre d’itérations maximal `iter_max`. On n’a pas jugé nécessaire de copier ici le proramme étant donné que sa méthode de construction a été détaillée.

Pour l’utiliser dans un autre programme, écrire le code :

```
import cmath
import numpy as np
from caustic_unique import *

#--------------------------#
#---------Functions--------#
#--------------------------#


phi_data, dphi, z_cc, zeta_c = make_equiv(s,q)

e = int(phi_in/dphi)

z_in, zeta_in =  z_cc[e], zeta_c[e]

f = int(phi_out/dphi)

z_out, zeta_out =  z_cc[f], zeta_c[f]
```

Listing 3: Appel des fonctions nécessaires à l’échantillonnage des caustiques.

Attention : n’appeler ces fonctions qu’une fois en début de programme !

La liste `phi_data` contient toutes les valeurs de ϕ échantillonnées, `dphi` est le pas d’échantillonnage, `z_cc` la liste des z décrivant les courbes critiques et `zeta_c` la liste des ζ décrivant les caustiques.

Pour utiliser la version du programme mettant en œuvre la routine de Skowron et Gould `cmplx_roots_gen`, remplacer `caustic_unique` par `caustic_unique_solv`.

C.2 Tracé de la trajectoire de la source, des caustiques et des courbes critiques

Le programme `caustics.py` trace simplement sur la même figure les courbes critiques (à gauche) et les caustiques (à droite) associées à une configuration (s, q) donnée. Sur la partie droite est aussi tracée la trajectoire de la source. Il Il suffit alors seulement de modifier les paramètres adéquats (α, u_0, t_0, $t_{\rm E}$) ou ($t_{\rm in}$, $t_{\rm out}$, $\phi_{\rm in}$, $\phi_{\rm out}$). Bien sûr, pour la trajectoire de la source dans la paramétrisation "caustic-crossing", il faut au préalable échantillonner les caustiques comme décrit précédemment.

C.3 Tracé de courbes d'amplification pour une source ponctuelle

Le programme `PS_magnification.py` trace en sortie la courbe d'amplification pour une source ponctuelle. Choisir la valeur des paramètres au début du programme et ajuster la valeur de `t_max` (maximum de l'intervalle de temps). Pour choisir la paramétrisation classique, mettre le *flag* `param` à 0 et pour la paramétrisation "caustic-crossing", le mettre à 1. [5]

C.4 Tracé de courbes d'amplification pour une source étendue dans l'approximation hexadecapolaire

Le programme `hexadecapole_magnification.py` fonctionne de la même manière que le précédent. Il faut simplement penser à fixer le rayon de la source par la variable `rho` quelle que soit la paramétrisation choisie.

En sortie, le programme trace la courbe d'amplification dans le cadrant supérieur et les résidus vis-à-vis de la source ponctuelle[6] (hexadecapôle - monopôle).

Ce programme **ne met pas** en place le *limb-darkening* (assombrissement centre-bord des étoiles). Les changements à effectuer dans ce programme (et les autres) sont mineurs car la formule de l'amplification $\hat{\mu}$ a déjà été dérivée dans le paragraphe "Effets de taille finie - Approximation hexadecapolaire".

C.5 Implémentation et test des dérivées analytiques pour une source ponctuelle

Le programme `test_derivatives.py` est un des plus importants. En effet, il calcule et trace les dérivées analytiques pour les deux paramétrisations, mais effectue aussi les deux tests déjà cités (vérification vis-à-vis des dérivées numériques et linéarisation de l'amplification). L'exécution de ce programme affiche quatre figures. La première contient le tracé des dérivées par rapport aux différents paramètres de la paramétrisation choisie *via* le flag `param` alors que la seconde et la troisième tracent respectivement la différence (non relative et sans valeur absolue) des dérivées numériques et des dérivées analytiques et la différence entre valeur exacte de $\mu(\theta + d\theta)$ et approximation linéaire $\mu(\theta) + \frac{\partial \mu}{\partial \theta} d\theta$. Enfin, la quatrième trace cette même différence mais en relatif (on divise le résultat par $\mu(\theta + d\theta)$), les échelles de cette figure sont donc à multiplier par 100 pour avoir des résultats en pourcentage.

Le cœur de ce programme réside dans les fonctions `classical(z, t, s, q, u_0, alpha, t_E, t_0)` et `caustic_crossing(z, t, s, q, zeta_in, zeta_out, z_in, z_out, t_in, t_out)` qui calculent les dérivées à *un instant donné*. La contribution de chaque image est sommée dans ces fonctions. `z` est ici un `array` (ou `list`) contenant les affixes des images du point-source considéré, sa dimension est donc 3 ou 5. `t` est la valeur du temps (discrétisé).

Les dérivées numériques de l'amplification sont obtenues par une petite variation d'*un paramètre à la fois* puis par le calcul de l'amplification induite par le nouveau jeu de paramètres.

C.6 Tracés des dérivées pour une source étendue dans l'approximation hexadecapolaire

Le choix de la paramétrisation se fait, là encore, *via* le flag `param`.

[5] Ce flag est présent dans tous les programmes et il fonctionne de la même manière partout.

[6] Aussi appelé monopôle.

Ce programme trace les dérivées analytiques de l'amplification par rapport aux même paramètres que dans le cas ponctuel et ajoute les dérivées par rapport aux deux nouveaux paramètres ρ, le rayon de la source (en unité du rayon d'Einstein), et Γ, le paramètre de *limb-darkening*[7].

Attention, le fonctionnement de ce programme diffère légèrement du précédent. L'inversion de l'équation des lentilles n'est plus réalisée dans le programme principal mais dans les routines
`hexadeca_classical(zeta_0, t, w, p, s, q, u_0, alpha, t_E, t_0)` et
`hexadeca_caustic_crossing(zeta_0, t, w, p, s, q, zeta_in, zeta_out, z_in, z_out, t_in, t_out)`.
A chaque pas de temps, dans le programme principal, seule la nouvelle position du *centre de la source* ζ_0 (argument `zeta_0`) est calculée puis, un appel à l'une ou l'autre des routines permet de calculer les dérivées pour w (argument `w`) et p (argument `p`) donnés. Puis, il suffit juste de sommer ces différents termes avec les coefficients déjà calculés.

Remarque : Il peut arriver que la courbe de lumière présente des amplifications négatives au niveau des traversées de caustiques car en développant l'expression de l'amplification totale $\hat{\mu}$, on trouve un coefficient négatif devant a_0.

C.7 Simulation de données expérimentales avec erreurs gaussiennes

Le programme `exp_data.py` permet de simuler et tracer des données expérimentales avec erreurs gaussiennes dans les deux paramétrisations et pour les deux types de source, ponctuelle ou étendue. A chaque pas de temps, on détermine l'amplification prévue par le modèle choisi puis on lui ajoute un nombre tiré selon une loi normale $\mathcal{N}(0,1)$ pondéré par un paramètre `sigma_mes` qui, multiplié par 100, donne l'erreur de mesure en pourcentage.

Remarque : L'erreur de mesure (en pourcentage) est ici la même pour tous les points ce qui engendre des barre d'erreurs plus grandes pour les fortes valeurs de l'amplification que pour les faibles valeurs. Or, en pratique, sur les courbes de lumière, on s'efforce de faire l'inverse. Il faudra donc penser à bien régler ce paramètre pour faire de vrai tests. Ne pas oublier de réduire le nombre d'itérations de la boucle en temps (variable `i_max`) pour avoir une courbe expérimentale vraisemblable.

C.8 Orthogonalisation de l'espace des paramètres

Comme mentionné précédemment, il a fallu séparer les deux paramétrisations dans deux programmes différents `ESBL_opt_ortho.py` pour la paramétrisation classique et `ESBL_opt_ortho_cc.py` pour la paramétrisation "caustic-crossing".

Les programmes contiennent trois fonctions nécessaires à l'implémentation du procédé de Gram-Schmidt,
`dot_prod(x, y, N, sigma)` qui implémente le produit scalaire de KAINS, `proj(x, y, N, sigma)` qui réalise l'opération de projection de `x` sur `y` et `gram_schmidt(U, N, I, sigma)` qui renvoie la matrice des vecteurs orthonormalisés. `x` et `y` sont des vecteurs (`list`), N est le nombre d'itérations en temps (à remplacer par `i_max` lors de l'appel de la fonction) et `sigma` la liste des incertitudes de mesure. `U` est la matrice des vecteurs P_i à orthonormaliser et I la dimension de l'espace des paramètres ((à remplacer par `m` lors de l'appel de la fonction)).

La démarche adoptée consiste à calculer les dérivées de l'amplification par rapport à chaque paramètre pour chaque pas de temps, de construire la matrice regroupant les vecteurs P_i puis de la transmettre à la fonction `gram_schmidt()` qui va renvoyer la matrice des vecteurs B_i. Puis le programme calcule les matrices de passage T et S qui seront indispensables pour la suite.

Dé-commenter les dernières lignes pour "tracer" les matrices T et S.

C.9 Optimisation par méthode Newton-CG

Les programmes réalisent la minimisation du χ^2 par méthode de Newton-CG **dans l'espace des paramètres corrélés** sont `ESBL_opt_Newton_bis.py` pour la paramétrisation classique et `ESBL_opt_Newton_bis_cc.py` pour la paramétrisation "caustic-crossing".

[7] A noter, pour calculer cette dérivée, il n'y a pas besoin d'introduire Γ dans le calcul de l'amplification. *Cf.* expression analytique

La partie la plus importante de ce programme est la fonction `chi_square(parameters, param, data, sigma, m, i_max)` qui renvoie, dans cet ordre, la valeur du χ^2 et sa matrice jacobienne (ses dérivées) pour un jeu de paramètres et une courbe expérimentale donnés. La liste `parameters` est le vecteur des paramètres, `param` le flag présent dans tous les programmes, `data` la liste des données expérimentales; `sigma` la liste des incertitudes de mesure, `m` la dimension de l'espace des paramètres et `i_max` le nombre d'itération de la boucle en temps (nombre de mesures). Pour la paramétrisation "caustic-crossing", les changements sont mineurs.

Puis, pour minimiser le χ^2, on applique la routine `scipy.optimize.minimize`. On envisagera d'étendre ce programme à la maximisation de la probabilité bayésienne *a posteriori*.

Les premiers résultats ne sont pas encore très probants car l'algorithme de minimisation ne converge pas tout le temps, ou ne réalise qu'une itération. En effet, la méthode de Newton souffre des divergences des éléments de matrice des matrices jacobienne et hessienne sur les caustiques; il faut donc trouver une méthode pour exclure ces points. Il faudra ensuite réaliser la minimisation dans l'espace des paramètres non corrélés.