caustics

Differentiable modelling of binary and triple lens events

Fran Bartolić

(*Frahn Bart-oh-leech*)

University of St Andrews

25th international microlensing conference

fbartolic

Context

- Modeling microlensing events is very difficult
- Too few researchers relative to scale of current and future datasets and the effort required to model any given event
- **Scientific results** in microlensing are **highly sensitive** to computational methods and assumptions that go into those methods
- There’s been very little methods development, **novel methods** from stats and ML **are under-utilised**

What’s difficult about microlensing? Everything!

- **Three big problems:**

  1. **Fast and accurate computation of magnification for extended limb-darkened sources**
     - Need $\gtrsim 10^6$ likelihood evaluations for MCMC class methods
  2. **Searching for and comparing different models**
     - Multiple competing hypotheses for any given dataset. How to find (and rank) the most probable ones?
  3. **Exploring plausible values of parameters within a small neighbourhood of the parameter space.**
     - How to obtain accurate parameter uncertainties for a single “solution”?

Gradients of the likelihood -> much more information about parameter space

- Gradients -> **local geometry** of the likelihood (χ^2)
- Enable use of gradient based optimization and sampling methods:
  - faster **MLE estimation** + exact **Hessians** (parameter covariance matrix), **Hamiltonian Monte Carlo**, **Variational Inference**…
- Modern **probabilistic programming** and **ML** libraries all use gradient based optimisers or MCMC samplers

Three ways of differentiating a function

1. **Symbolic differentiation** (pen & paper, `Mathematica`, `SymPy`)

   - $\frac{\mathrm{d}}{\mathrm{d}x}\cos x = -\sin x$

2. **Numerical differentiation** (finite differences)

   - $f'(x) \approx \frac{f(x + h/2) - f(x - h/2)}{h}$

3. **Automatic differentiation** (differentiate through computer code, say C++ or Python)

   - `jax.grad(jax.numpy.sin)(x)`

Automatic differentiation (AD)

- **Key idea:**
  - A **computer program** implementing a differentiable function $f : \mathbb{R}^n \to \mathbb{R}^m$ is a composition of elementary operations such as **multiplication**, **addition**, **trig. functions**, etc.
  - **Chain rule** from calculus -> if you can differentiate each step, you can differentiate the whole
  - The program could be something like a **neural network** (pile of liner algebra) or it could be an **entire physics simulator**
- AD is the only way to compute derivatives of scalar functions with lots of inputs
  - In ML "lots" can mean **millions** or **billions** of **parameters!**
  - **Deep Learning unimaginable without AD (backpropagation)**

Automatic differentiation (AD)

- Can't just take an off-the shelf C++ code and do AD, **need to rewrite the code** from scratch using a **specialised AD library**
  - **Examples from astronomy**: `exoplanet` (**transits, RV, TTVs**), `starry` (**occultations**), `exojax` (**exoplanet atmospheres**), `dLux` (**differentiable optics**) …
- **Popular AD libraries**: `Tensorflow`, `PyTorch`, `Aesara` and `JAX` (Python), `Eigen` (C++), `Enzyme` (LLVM)

JAX

- **Not just an AD library**
- Write Python code but it gets **JIT compiled** to **XLA** (low level language) on the fly
  - -> **C like speeds** possible while writing code which looks like Python!
  - -> Same code works on **CPUs, GPUs** and **TPUs**!
- Coding a complicated physics model in JAX is not easy, lots of caveats

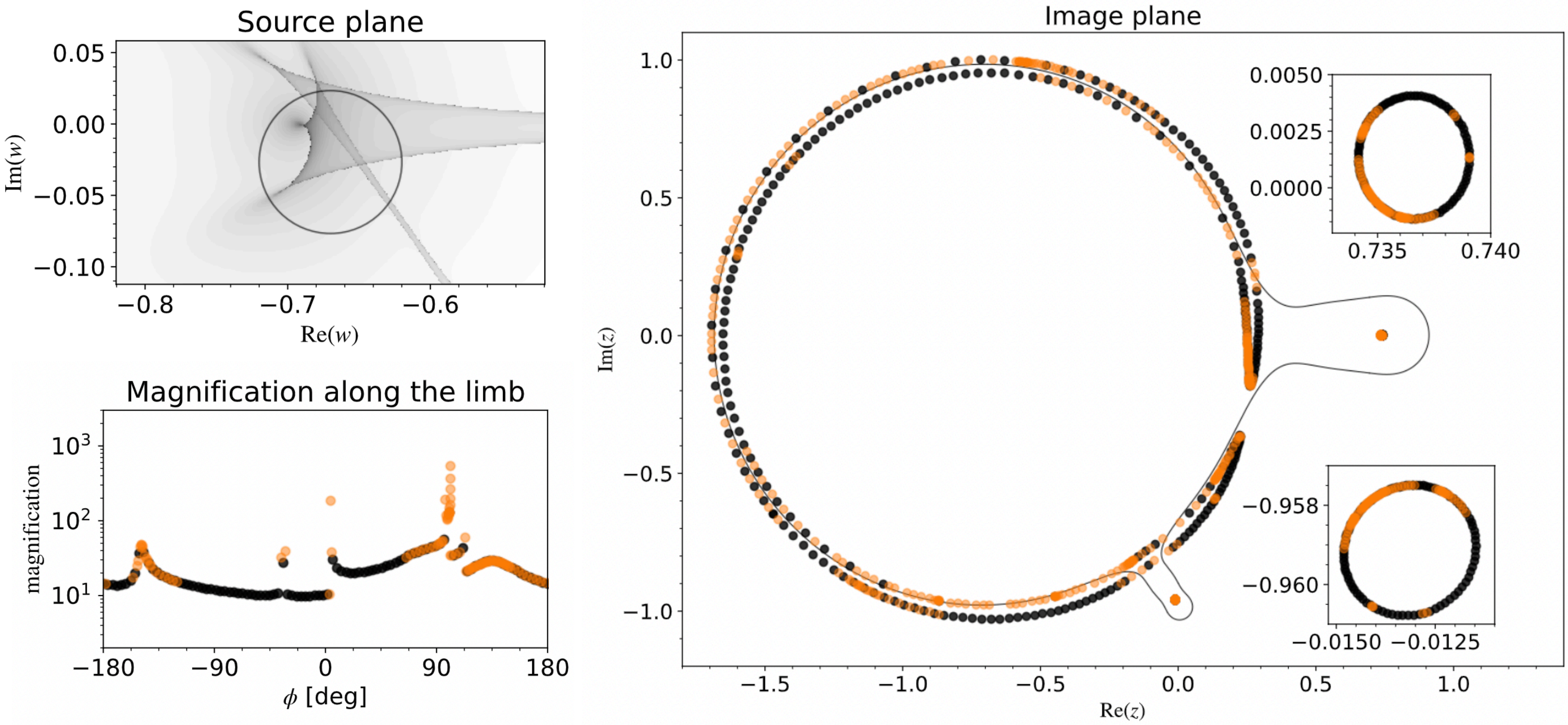
Building a differentiable microlensing code

- I didn't really understand how other codes worked so I started building my own
    - This turned out to be **very hard**, do not recommend!
- The result is `caustics` : https://github.com/fbartolic/caustics
- `caustics` **builds on previous work**:
    - Kuang et. al. 2021 (arXiv:2102.09163)
    - Dominik 1998 (arXiv:astro-ph/9804059)
    - Bozza et. al. 2018 (arXiv:1805.05653)
    - Cassan 2017 (arXiv:1703.03600)

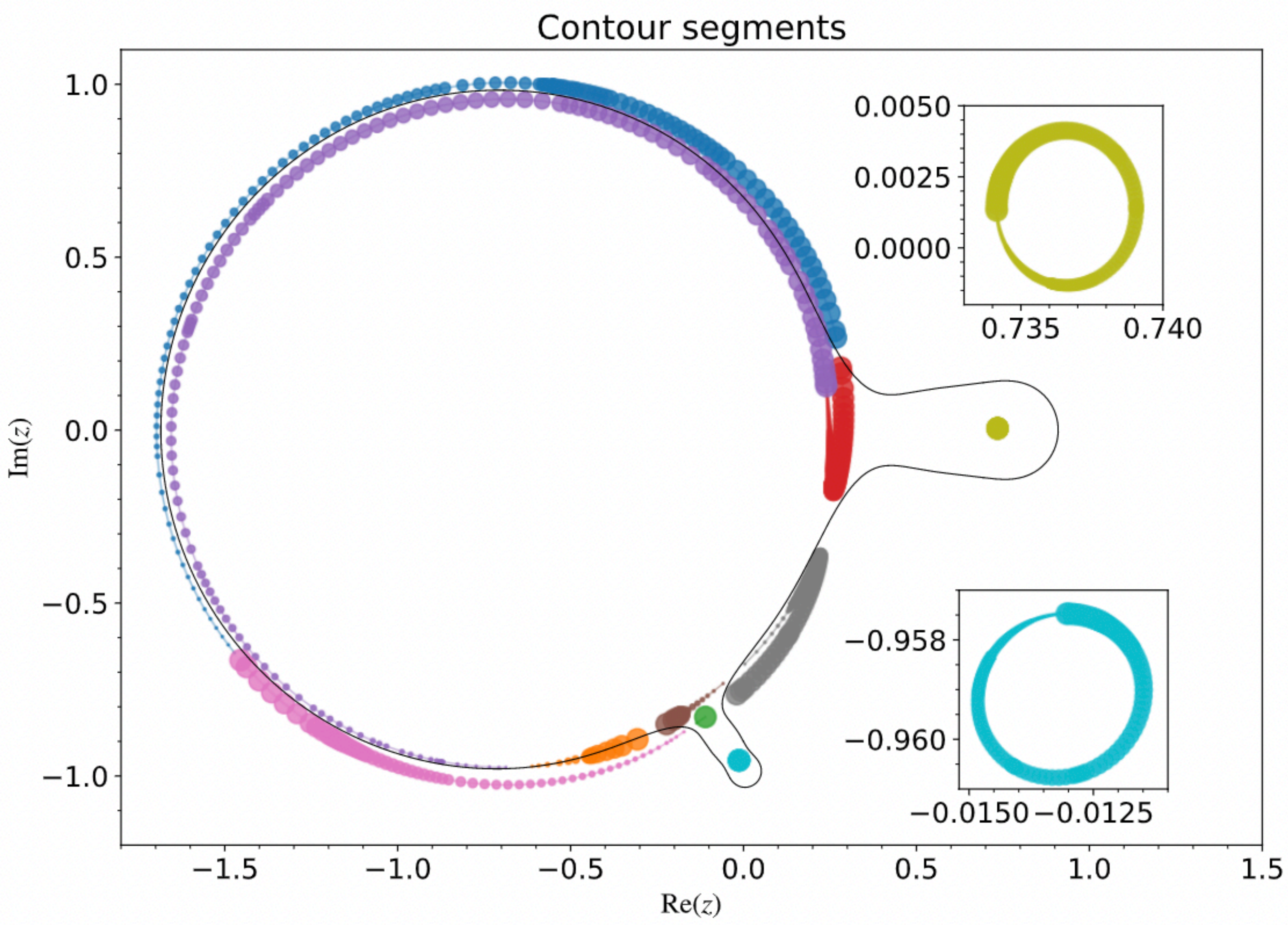
`caustics` in a nutshell

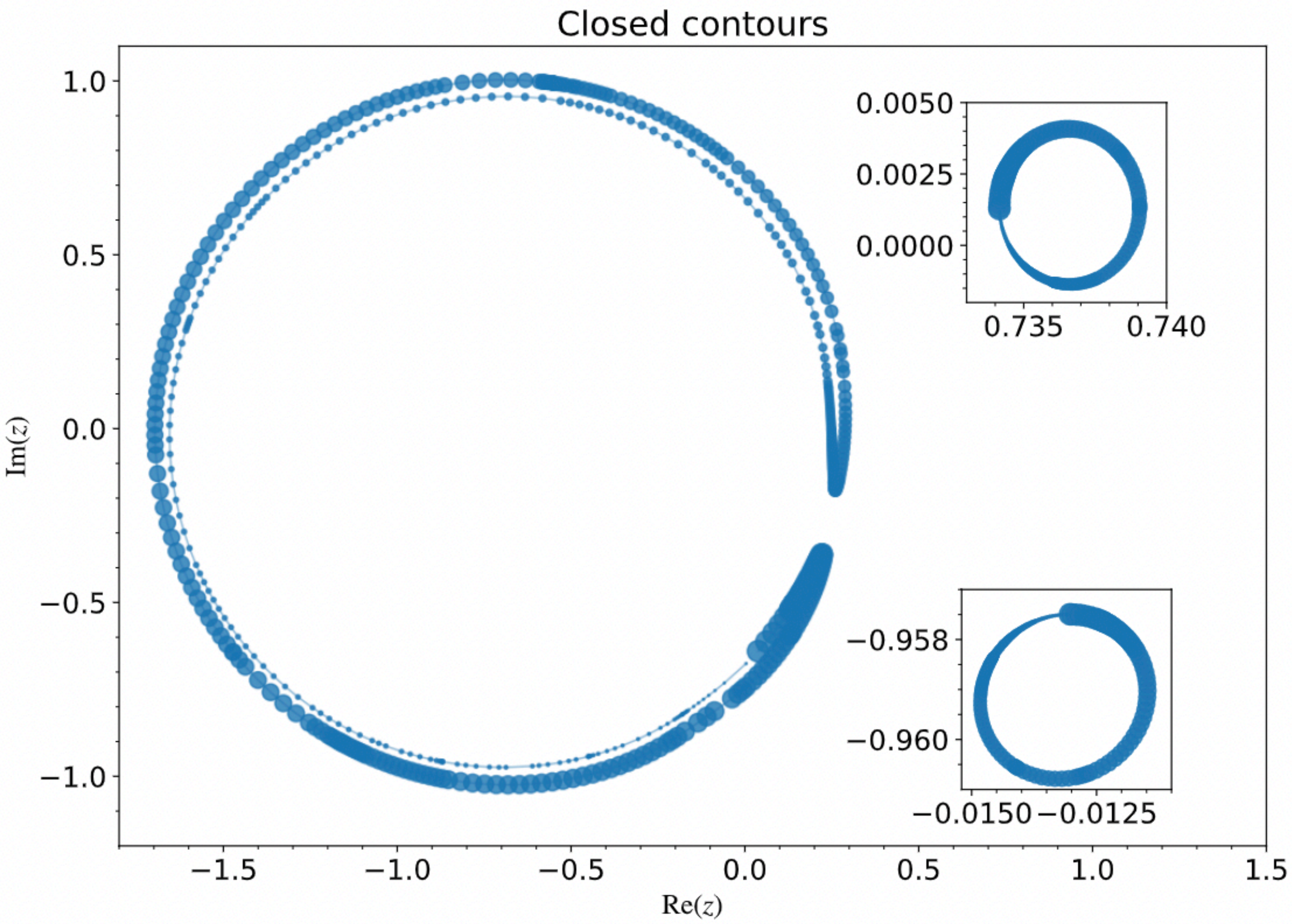
- Support for **single**, **binary** and **triple** lensing (extended sources and limb-darkening)
- **Differentiable Aberth-Ehrlich complex polynomial root solver** (https://hal.archives-ouvertes.fr/hal-03335604)
- **Contour integration** algorithm adapted from Kuang et. al. 2021 with important changes
- Full support for **AD**, **cost of gradient evaluation 3-5X the cost of magnification evaluation**
- **Triple lens magnification only ~2X more expensive than binary lens magnification, limb darkening ~8X more expensive than uniform brightness**
- Up to 10X slower than **VBBinaryLensing** for uniform brightness mag., roughly the same cost for limb-darkening, lots of **room for improvement**

Contour integration

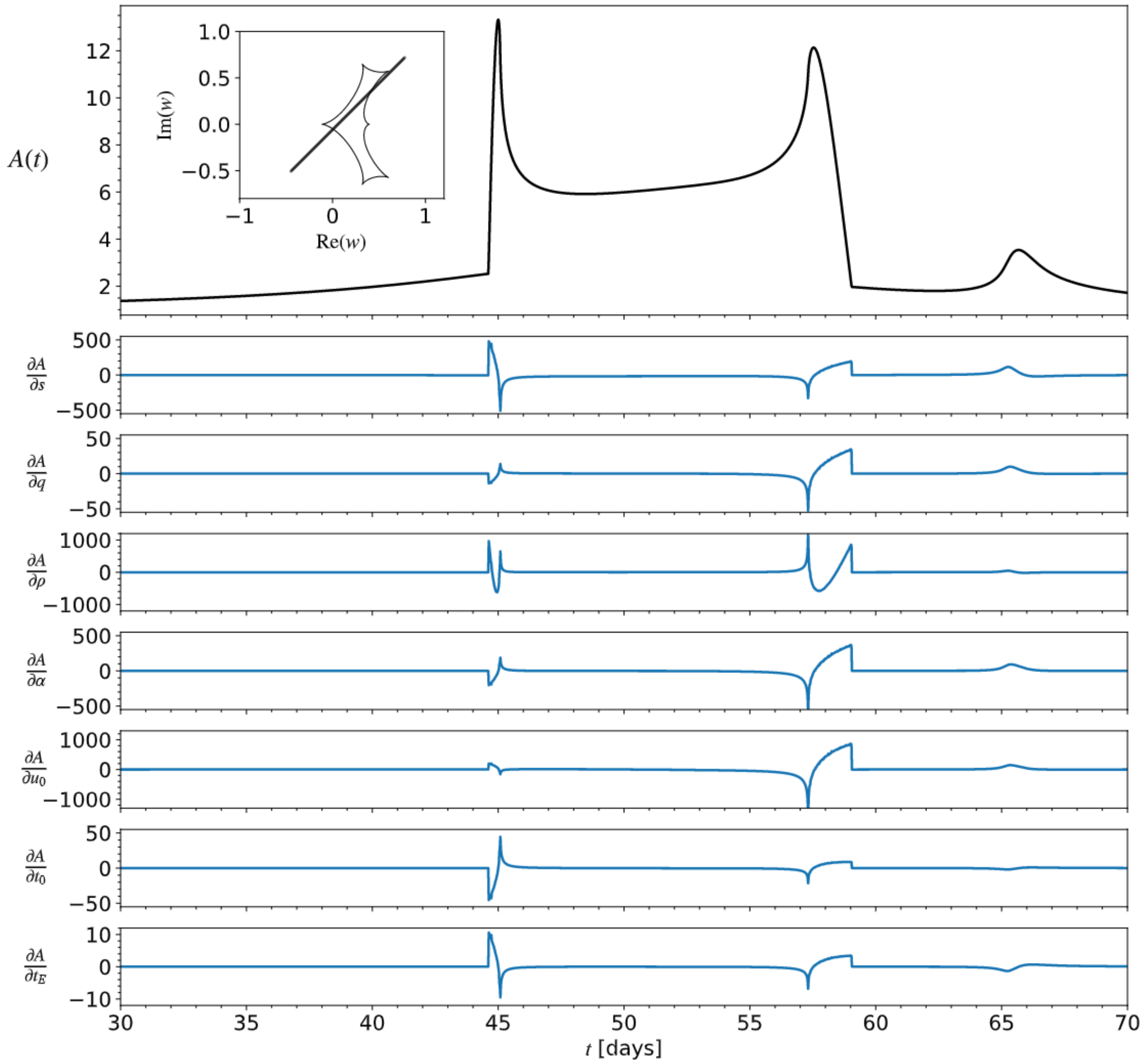


Connecting the dots…





It works!



$A(t)$
Im(w)
Re(w)
$\frac{\partial A}{\partial s}$
$\frac{\partial A}{\partial q}$
$\frac{\partial A}{\partial \rho}$
$\frac{\partial A}{\partial \alpha}$
$\frac{\partial A}{\partial u_0}$
$\frac{\partial A}{\partial t_0}$
$\frac{\partial A}{\partial t_E}$
t [days]

Next steps

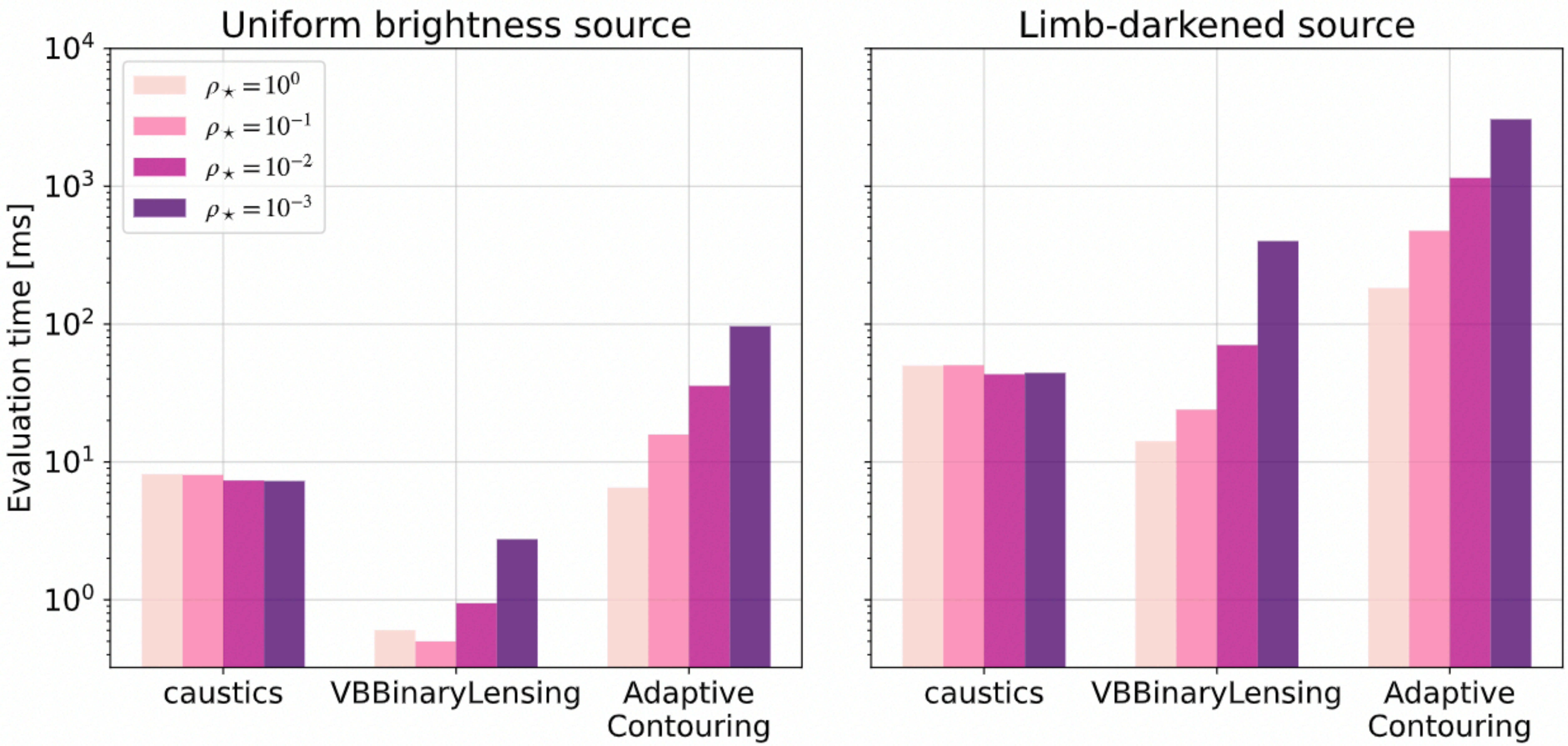
- **Test the code on real world problems!**
- Test to switch between hexadecapole and full calculation doesn’t work for triple lenses at the moment
- More tests for triple lensing
- **Better error control** -> need to differentiate through `while` loops
- **Are gradient based methods actually useful?** If not, what does that imply about gradient-free methods?
- **Astrometric microlensing** -> need a few extra lines of code
- **Arbitrary brightness profiles** -> model stellar spots

Summary

- Differentiable modeling of microlensing light curves for the first time ever
- **First fast triple lens code**
- Looking for feedback from the community!
- Check out the code on GitHub, **contribute**!
- **IMO, effort invested into methods development for microlensing should be 10X more than it is today**

fb90@st-andrews.ac.uk fbartolic

Additional slides



Uniform brightness source
Limb-darkened source
Evaluation time [ms]
10^4
10^3
10^2
10^1
10^0
$\rho_\star = 10^0$
$\rho_\star = 10^{-1}$
$\rho_\star = 10^{-2}$
$\rho_\star = 10^{-3}$
caustics
VBBinaryLensing
Adaptive Contouring

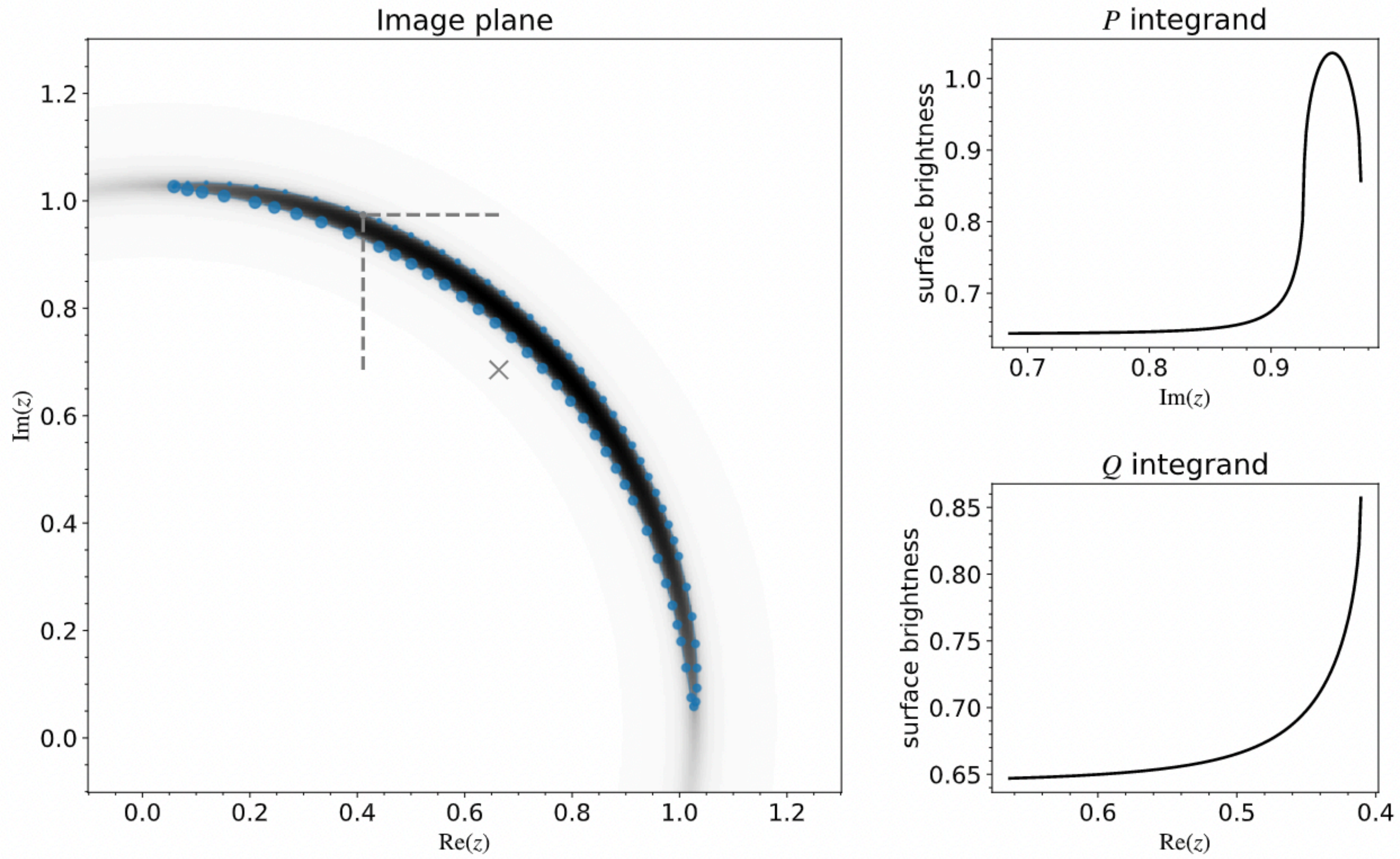
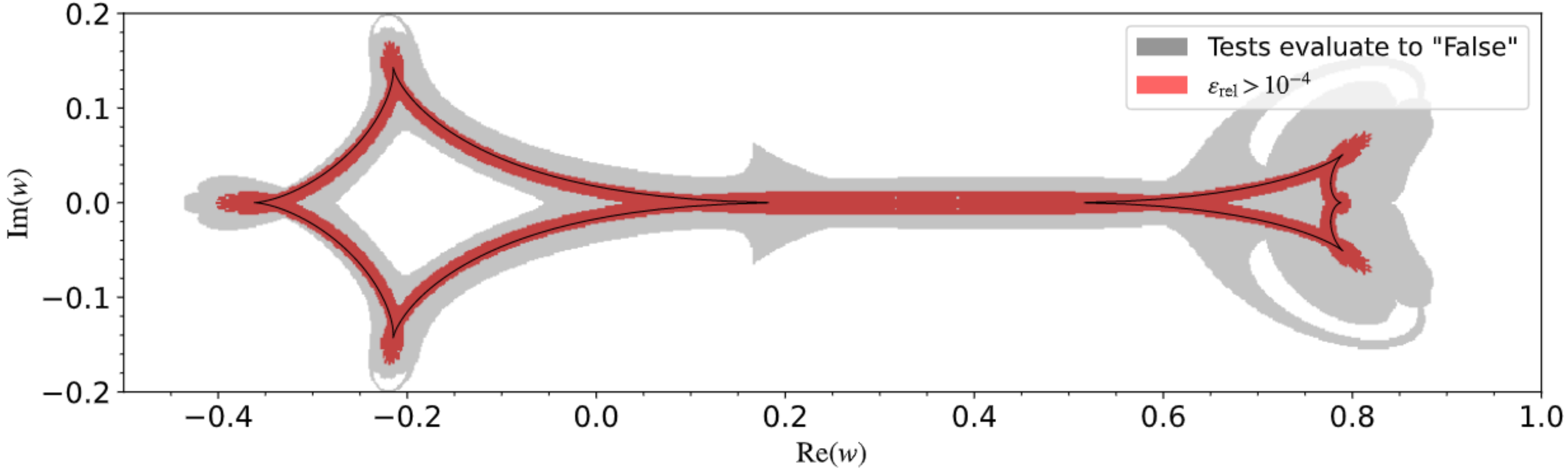


Image plane
Im(z)
Re(z)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
P integrand
surface brightness
1.0
0.9
0.8
0.7
0.7
0.8
0.9
Im(z)
Q integrand
surface brightness
0.85
0.80
0.75
0.70
0.65
0.6
0.5
0.4
Re(z)


Tests evaluate to "False"
$\varepsilon_{\rm rel} > 10^{-4}$
0.2
0.1
0.0
−0.1
−0.2
Im(w)
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
Re(w)