Running `TripleAstroLightCurve` from a Local Build on macOS

This guide shows how to reproduce Valerio’s working setup: build the C++ extension with CMake, keep the compiled library inside the checkout, and point Python (or notebooks) at the repo.

These instructions assume a Conda environment and have been tested on macOS from step 2 onwards without `compilers` installed (you possibly don't need it for macOS) and using `brew install cmake`. Step 1 is my best guess at the easiest way to make sure you have all the required dependencies.

1. Install build dependencies in the Conda env

In the environment you use for notebooks, install the tooling from `conda-forge`:

```
conda install -c conda-forge cmake pybind11 compilers
```

- `cmake` – drives the build process
- `pybind11` – headers + CMake configuration
- `compilers` – provides `clang` / `g++` if Apple’s tools are missing

If any package is already present, Conda will skip it.

2. Configure and build the extension

From the repository root:

```
cd /path/to/VBMicrolensing
mkdir -p build/cmake-local
cd build/cmake-local

# Tell CMake where pybind11’s CMake package lives
cmake -Dpybind11_DIR="$(python -m pybind11 --cmakedir)" ../..

# Compile the extension (Release or Debug as preferred)
cmake --build . --config Release
```

The build emits `VBMicrolensing.so` in `build/cmake-local/` .

> Need different precision/options? Add `-D…` flags to the first `cmake` command.

3. Copy the built library into the package

`VBMicrolensing/__init__.py` expects the compiled module in the same directory. Copy it in:

```
cp VBMicrolensing.so ../../VBMicrolensing/
```

The repo now contains:

```
VBMicrolensing/
  __init__.py
  VBMicrolensing.so   ← freshly built extension
  data/
  …
```

4. Point Python at the checkout

One-off shell session

```
export PYTHONPATH="/path/to/VBMicrolensing${PYTHONPATH:+:$PYTHONPATH}"
python - <<'PY'
import math
import VBMicrolensing

VBM = VBMicrolensing.VBMicrolensing()
VBM.astrometry = True
VBM.SetObjectCoordinates("17:51:40.2082 -29:53:26.502")

params = [
    math.log(0.9), math.log(0.028997), 0.1, 0.261799, math.log(0.01), math.log(20), 0.0
    math.log(1.5), math.log(0.003270), 0.785398,
    0.1, 0.1, -3, -2, 0.12, 5.15,
]
print("Calling TripleAstroLightCurve …")
VBM.TripleAstroLightCurve(params, [-5.0, 0.0, 5.0])
print("Call completed.")
PY
```

If you see “Call completed.” nothing crashed and the setup matches Valerio’s (presumably).

Make it permanent for notebooks

1. Locate the environment’s `site-packages` directory (for example: `python -c "import site; print(site.getsitepackages())"`).
2. Create a `.pth` file there, such as `vbm_dev.pth` , containing a single line with the repo path:

```
echo "/path/to/VBMicrolensing" > /Users/username/miniconda3/envs/vbmic/lib/python3.
```

Any Python process launched from that Conda environment (including Jupyter) will automatically import from the checkout.

5. Run the full script

```python
import numpy as np
import VBMicrolensing
import matplotlib.pyplot as plt

# Initialize VBMicrolensing instance
VBM = VBMicrolensing.VBMicrolensing()
VBM.RelTol = 1e-3
VBM.Tol = 1e-3
VBM.astrometry = True

# System parameters
s12 = 0.9
q2 = 0.028997
u0 = 0.1
alpha = 0.261799  # radians
rho = 0.01
tE = 20
t0 = 0
s23 = 1.5
q3 = 0.003270
psi = 0.785398  # radians

# Astrometric parameters
piEN = 0.1
piEE = 0.1
muS_N = 0.1
muS_E = 0.1
piS = 0.1
thetaE = 1.0

# Time array
num_points = 10
tmin = -50
tmax = 50
t = np.linspace(t0 + tmin, t0 + tmax, num_points).tolist()

# Parameter list
params = [
    np.log(s12), np.log(q2), u0, alpha, np.log(rho), np.log(tE), t0,
    np.log(s23), np.log(q3), psi,
    piEN, piEE, muS_N, muS_E, piS, thetaE
]

result = VBM.TripleAstroLightCurve(params, t)
```

With the environment configured, the 10 000-point script (above) should execute cleanly. If it still segfaults, double-check:

- The compiled `VBMicrolensing.so` sits in `VBMicrolensing/`.
- The checkout is on `PYTHONPATH` (or in a `.pth` file).
- The Conda env has the same toolchain you used during the build.

Once those conditions hold, `TripleAstroLightCurve` behaves exactly like Valerio reported.