dart$_{b}oardDocumentation$

Release latest

Mar 12, 2021

User Guide

dart_board makes it easy to model the formation and evolution of binary stars. It acts as a statistical wrapper around rapid binary evolution codes.

This documentation will provide a guide to using our code, but for details on the statistical approach, some tests, and various applications, check out our paper.

CHAPTER 1

a basic example

To generate a population of high mass X-ray binaries (HMXB), use the following example

```
import numpy as np
import dart_board
import pybse

# Set up the sampler
pub = dart_board.DartBoard("HMXB",
                           evolve_binary=pybse.evolve,
                           nwalkers=320)

# Initialize the walkers
pub.aim_darts()

# Run the sampler
pub.throw_darts(nburn=20000, nsteps=100000)

# Save the chains
np.save("HMXB_posterior_chains.npy", pub.chains)
```

CHAPTER 2

getting started

You will want to start with the installation guide to get dart_board running. Until we have our quickstart tutorial completed, the documentation we provide below is probably the best way to learn how to use dart_board for your needs. For more concrete examples, we provide the source code used to generate all the examples and figures in our paper. If you have problems, feel free to label an issue.

2.1 Installation Guide

2.1.1 Installing dart_board

Currently, dart_board can only be installed from source. Assuming you have python, pip, and git installed, use the code snippet below to install dart_board:

```
git clone https://github.com/astroJeff/dart_board.git
cd dart_board
pip install .
```

2.1.2 python bindings for BSE

dart_board does not come with a black box rapid binary evolution code, however we provide instructions for getting set up with bse. All examples and tutorials assume you have bse set up for use.

There are two separate pieces to using BSE within dart_board. First, we need to create python bindings for BSE using the compiler `f2py. Check out this website for documentation on `f2py`. Second, we need to create a separate python library, `pyBSE`, that serves as a wrapper around the bse executable created with f2py.

- Start by downloading BSE (not SSE) from Jarrod Hurley's website. Place the downloaded tarball (`bse.tar`) into dart_board/pyBSE/
- Change directories to dart_board/pyBSE/ and unpack the tarball you just downloaded:

```
cd pyBSE
tar -xvf bse.tar
```

You should now see a bunch of new files that contains the rapid binary evolution files from SSE/BSE.

- Move to the parent dart_board directory and apply the included BSE patch file:

```
cd ..
patch -s -p0 < BSE_updates.patch
```

This will update all the BSE files to interface correctly with dart_board.

- Change directories back to pyBSE and make the bse executable:

```
cd pyBSE
make pybse
```

- Now, we need to create the pyBSE python library so that it is accessible to any directory. We have included a setup.py script for use with pip:

```
pip install -e .
```

2.1.3 python dependencies

- numpy
- scipy
- matplotlib
- pickle
- astropy
- emcee
- corner (optional)

2.2 Using dart_board for your needs

This Page is Under Construction

dart_board can be used with relatively little effort for two broadly defined tasks: modeling binary populations and fitting individual systems. We provide broad instruction on the usage for each of these cases below. We will assume that dart_board and pybse have both been installed on your system.

2.2.1 Modeling populations

One may wish to generate populations of stellar binaries, a task which essentially covers any exercise previously performed in the literature by the technique labeled "binary population synthesis." In this case, the likelihood function for dart_board is the indicator function: if the model parameters produce a system of interest, it has a likelihood of unity, the likelihood is zero. Model parameters for priors are based on distributions commonly used in the binary population synthesis literature.

First, our basic example to generate a population of high mass X-ray binaries:

```
import numpy as np
import dart_board
import pybse

# Set up the sampler
pub = dart_board.DartBoard("HMXB",
                           evolve_binary=pybse.evolve,
                           nwalkers=320)

# Initialize the walkers
pub.aim_darts()

# Run the sampler
pub.throw_darts(nburn=20000, nsteps=100000)

# Save the chains
np.save("HMXB_posterior_chains.npy", pub.chains)
```

Let's go over this line-by-line. We start by importing our necessary libraries:

```
import numpy as np
import dart_board
import pybse
```

We will need numpy for manipulating the resulting set of posterior samples, which exists as a numpy array. dart_board is the library created upon installation, and pybse is the library that provides python bindings for bse.

Next, initialize the DartBoard object with:

```
# Set up the sampler
pub = dart_board.DartBoard("HMXB",
                           evolve_binary=pybse.evolve,
                           nwalkers=320)
```

The first argument `"HMXB"` is the only required argument, as this tells the likelihood function within dart_board to only select for high mass X-ray binaries. Currently there are only a handful of possible options, but we are working on this. It is simple to add your own to `dart_board/posterior.py`. The next argument `evolve_binary=pybse.evolve` tells dart_board the name of the rapid binary evolution function. The default is pybse, but if you wish to add your own, check out our guide for importing your rapid binary evolution script to dart_board. We also tell dart_board to use 320 walkers. Consult the function definition for a guide to the many other optional arguments.

Now, we must initialize the walkers in an initial part of the parameter space. This is done through an iterative process that could probably be improved, but in practice is rarely a bottleneck. Although this function must be called, it requires no arguments:

```
# Initialize the walkers
pub.aim_darts()
```

Next, we can actually run the sampler:

```
# Run the sampler
pub.throw_darts(nburn=20000, nsteps=100000)
```

We have told the sampler to run for a burn-in of 20000 and then save the next 100000 steps.

The results of the sampler are saved as part of the DartBoard class. Principal among these are the chains which contain the posterior samples. we will save these to data for further use in post-processing with:

```
# Save the chains
np.save("HMXB_posterior_chains.npy", pub.chains)
```

2.2.2 Modeling individual systems

The only difference between modeling populations and modeling individual systems is the inclusion of observational constraints. This is allowed through a `kwargs` argument passed to the DartBoard upon initialization. For instance, if we want to model the formation of the LMC high mass X-ray binary LMC X-3, we want to add constraints on the black hole mass ($M_1 = 6.98 \pm 0.56\ M_\odot$), the donor mass ($M_2 = 3.63 \pm 0.57\ M_\odot$), and the orbital period ($P_{\rm orb} = 1.7 \pm 1$ days). Our new code now looks like:

```
import numpy as np
import dart_board
import pybse

system_kwargs = {"M1" : 6.98, "M1_err" : 0.56, "M2" : 3.63, "M2_err" : 0.57, "P_orb"␣
↪: 1.7, "P_orb_err" : 0.1}

pub = dart_board.DartBoard("BHHMXB",
                           evolve_binary=pybse.evolve,
                           nwalkers=320,
                           system_kwargs=system_kwargs)

pub.aim_darts()

pub.throw_darts(nburn=20000, nsteps=10000)

np.save("LMC-X3_posterior_chains.npy", pub.chains)
```

Notice that the only things that changed are the creation of system_kwargs, which is then passed as an argument when DartBoard is initialized, and the conversion of `"HMXB"` to `"BHHMXB"`, to specify that we only care about high mass X-ray binaries with black hole accretors.

Note that uncertainties (assumed to be Gaussian) must be provided along with any observational values, otherwise the likelihood function is ill-defined.

A variety of observational constraints can be included. Look at the function definition for posterior.posterior_properties for a list and description of the available options.

2.2.3 Writing your own prior, likelihood, and posterior functions

This section is under construction.

2.3 The DartBoard Class

2.4 Using Star Formation Histories

Under Construction

2.5 Importing your own rapid binary evolution code

Under Construction

The default rapid binary evolution code is BSE by Hurley et al. (2002). We provide python bindings for BSE for integration with dart_board, see the installation guide for instructions.

CHAPTER 3

copyright and licensing

If you use dart_board in your research, please cite our paper.

CHAPTER 4

changelog

Version 1.0.0 - Feb. 2018

Python Module Index

d

Index

D