JAVELIN

Overview
Source
Commits
Branches
Pull requests
Pipelines
Downloads

nye17 / JAVELIN

Overview

HTTPS https://bitbucket.org/nye17/javelin

Last updated	2017-12-01
Website	http://www.astronomy.ohi...
Language	Python
Access level	Read

0 Open PRs
2 Branches

W

Recent activity

1 commit
Pushed to nye17/javelin
fa69a1c formatting.
nye17 · yesterday

1 commit
Pushed to nye17/javelin
1aeb032 added reference t...
nye17 · yesterday

3 commits
Pushed to nye17/javelin
4de58e1 added diskmodel
f075b22 flow: Merged <fea...
10068d3 flow: Closed <feat...
nye17 · 2017-11-22

1 commit
Pushed to nye17/javelin
559174b finish thin disk
nye17 · 2017-11-22

1 commit
Pushed to nye17/javelin
afc13c3 moar
nye17 · 2017-11-22

1 commit
Pushed to nye17/javelin
1d7476a fixed >>> blocks.
nye17 · 2017-11-22

1 commit
Pushed to nye17/javelin
52409b2 added readme for ...
nye17 · 2017-11-22

1 commit
Pushed to nye17/javelin
9844f95 moar.
nye17 · 2017-11-22

1 commit
Pushed to nye17/javelin
7249e41 fixed a typo in math.
nye17 · 2017-11-22

JAVELIN

What is JAVELIN

JAVELIN stands (reluctantly) for **J**ust **A**nother **V**ehicle for **E**stimating **L**ags **I**n **N**uclei. As a version of our *SPEAR* algorithm written in Python to provide more flexibility in both functionality and visualization. You can use **JAVELIN** to model quasar variability using different covariance functions (Zu et al. 2013), and measure emission line lags using either spectroscopic light cruves (Zu et al. 2011) or photometric light curves (Zu et al. 2016).

Changelog

Please send an email to `yingzu AT sjtu.edu.cn` if you have any questions.

> *Version 0.33* includes the new Disk_Model method from Mudd et al. 2017
>
> *Version 0.32alpha* now has the capability to keep some parameters fixed during MCMC sampling
>
> *Version 0.3* features photometric capabilities

Install JAVELIN

Prerequisites

JAVELIN requires

1. Fortran Compiler (>F90)
2. Python (>2.5)
3. Numpy (>1.4)
4. Scipy (>0.1)
5. Matplotlib (>1.0)

We strongly recommend that you have the `Lapack` and `Atlas` libraries installed on the system, although they are not necessary. It requires no extra effort to install them, as many systems either come with LAPACK and BLAS pre-installed (MAC), or have them conveniently in software repositories (Linux distributions). They provide native support for fast solving linear systems inside JAVELIN, otherwise JAVELIN will simply install a subset of the the `Lapack` source from scratch.

Installation

To download the package, you can either go to the `Downloads` tab for stable releases or directly pull the cutting-edge version using `mercurial`. We strongly suggest you clone the repo for the latest version, as we are updating the package on a regular basis.

You can install JAVELIN by the standard Python package installation procedure:

```
$ python setup.py config_fc --fcompiler=intel  install
```

or if you want to install the package to a specified directory `JAVELINDIR`:

```
$ python setup.py config_fc --fcompiler=intel install
```

where `config_fc --fcompiler=intel` tells Python to use the *intel fortran compiler* to compile Fortran source codes. You can also specifies other fortran compilers that are available in your system, e.g.,:

```
$ python setup.py config_fc --fcompiler=gnu95 install
```

uses `GFortran` as its Fortran compiler.

Caution!

Note that the short names for Fortran compilers may vary from system to system, you can check the list of available Fortran compilers in your system using:

```
$ python setup.py config_fc --help-fcompiler
```

and you can find them in the `Fortran compilers found:` section of the output.

Caution!

If you installed Python from Conda that comes with the Accelerate framework, multithreading may not work for you, the solution is to set the environmental variable `VECLIB_MAXIMUM_THREADS` to `1` in your terminal:

```
$ export VECLIB_MAXIMUM_THREADS=1
```

Test Installation

After installing JAVELIN, navigate into the `examples` directory:

```
$ cd javelin/examples/
```

you can try:

```
$ python demo.py test
```

to make sure the code works (i.e., no error's reported). Also, to test whether the graphics work, you can try:

```
$ python plotcov.py
```

which exactly reproduces Figure 1 in Zu et al. (2013).

1 commit
Pushed to nye17/javelin
e0b0167 ughhhhh
nye17 · 2017-11-22

1 commit
Pushed to nye17/javelin
052180d moar.
nye17 · 2017-11-22

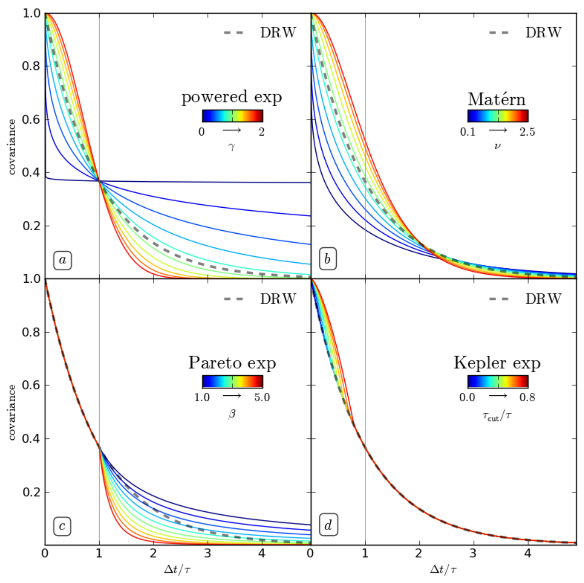

Fig. 1 : Illustration of four continuum models available in JAVELIN.

Demonstration

Here we briefly explain how to use JAVELIN to caculate the spectroscopic and photometric line lags for the AGN hosted by an imaginary Loopdeloop galaxy, where two emission lines are observed, Ylem and Zing. If you are already familiar with the Zu et al. (2011) paper, feel free to skip to the next section. Every file and script referred to here can be found inside `examples` directory:

```
$ cd javelin/examples
```

To give you an idea of how things work in JAVELIN, let us first go through several figures illustrating the underlying methodology of JAVELIN. You can also show the figures below locally by running:

```
$ python demo.py show
```

on the command line.

In our RM models, we assume the quasar variability on scales longer than a few days can be well described by a Damped Random Walk (DRW) model, and the emission line light curves are simply the lagged, smoothed, and scaled versions of the continuum light curve. Fig. 1 shows the true light curves for the continuum, the Ylem, the Zing, and the broadband Yelm lines. In particular, the Ylem (Zing) light curve is lagged by 100 (250) days, scaled by a factor of 0.5 (0.25), and smoothed by a top hat of width 2 (4) days, from the continuum light curve. The continuum light curve is generated from the DRW model with a time scale 400 days, a variability amplitude of sigma=3, and a mean of 10.0 (arbitratry flux units). Thus, for spectroscopic RM we have two parameters for the continuum DRW model, sigma and tau, and three parameters for each emission line model --- the lag t, the width of the tophat smoothing function w, and the flux scaling factor s; for photometric RM we have an additional parameter — *alpha* describing the ratio between the

two continua, one off and one on the line flux.

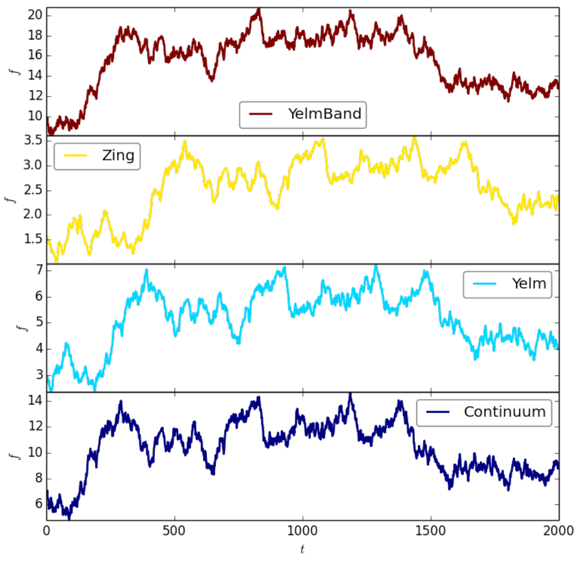

Fig. 2: True light curves of loopdeeloop (from top to bottom: the Yelm band flux, the Zing emission line, the Ylem emission line, and the continuum).

In practice, what we could observe are down-sampled and noisy versions of the true light curves, sometimes with seasonal gaps because of the conflict with our Sun's schedule, as shown by Fig. 3.

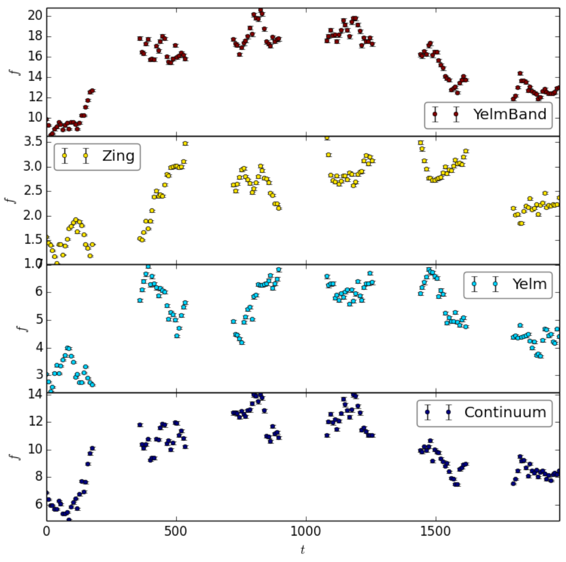

Fig. 3: Same as Fig. 2, but observed versions.

To directly derive lags from those sparse light curves is hard with traditional cross-correlation based methods. JAVELIN makes it much less formidable, by incorporating the statistical properties of the continuum light curve into the lag determination, keeping track of all the correlations of the model, and self-consistently removes the light curve mean. The first step is to build a continuum model to determine the DRW parameters of the continuum light curve. Fig. 4 shows

the posterior distribution of the two DRW parameters of the continuum variability as calculated from JAVELIN using MCMC chains,

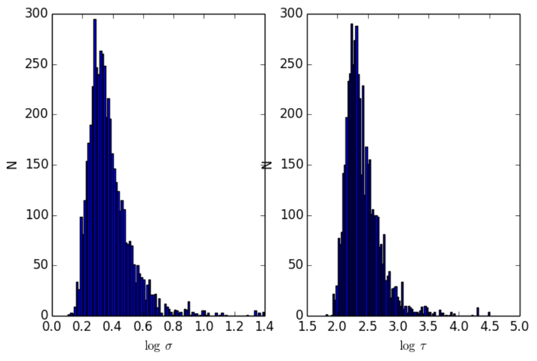

Fig. 4: Posterior distributions of the DRW parameters based on fits to the continuum light curve.

Once we derive the posteriors of the DRW parameters, we then have a pretty good idea of how much the continuum light curves in unobserved epochs should vary relative to observed epochs, i.e., we know how to statistically interpolate the continuum light curve. To measure the lag between the continuum and the Ylem light curve, JAVELIN then tries to interpolate the continuum light curve based on the posteriors derived in Fig. 4, and then shifts, smooths, and scales each continuum light curve to compare to the observed Ylem light curve. After doing this many many times in a MCMC run, JAVELIN finally derives the posterior distribution of the lag t, the tophat width w, and the scale factor s of the emission line, along with updated posteriors for the timescale tau and the amplitude sigma of the continuum, as shown in Fig. 5.

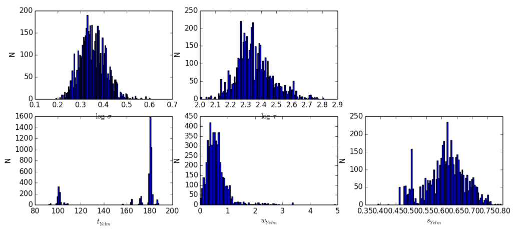

Fig. 5: Posterior distributions of the emission line lag t, tophat width w, and the scale factor s for the Ylem light curve (bottom). The top two panels show the updated posteriors for tau and sigma.

However, we can see two peaks for the lag distribution in Fig. 5, which is caused by the 180-day seasonal gaps in the two light curves - JAVELIN found that it is much easier to shift the continuum by 180 days to compare to the line light curve - there is no overlap between the two, therefore no objection from the data!

Fortunately, we also have observations of the Zing light curve. Although equally sparsely sampled and having the same gaps, the mere existence of the Zing light curve makes it impossible for JAVELIN to shift the continuum by 180 days TWICE to compare to the two line light curves! After another MCMC run, JAVELIN is able to eliminate the second peak at 180 days and solve the lags for both emission lines simultaneously, as shown

in Fig. 6.

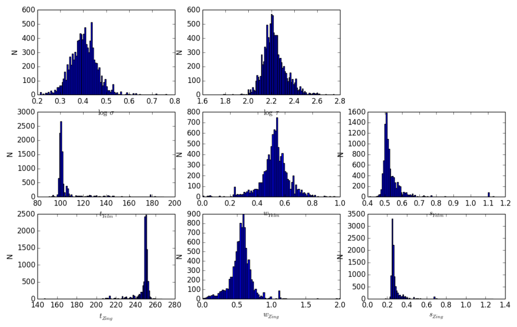

Fig. 6: As in Fig. 5, but after running JAVELIN for both two line light curves plus the continuum simultaneously.

Finally, we want to know what the best--fit parameters from the last MCMC run look like. It is generally very hard to visualize the fit for the traditional cross-correlation methods, but JAVELIN is exceptionally good at this - after all what it has been doing is to interpolate and align light curves, so why not for the best-fit parameters? Fig. 7 compares the best-fit light curves and the observed ones shown earlier in Fig. 3. Apparently JAVELIN does a great job of recovering the true light curves (compare to Fig. 2). Remember, however, that these show the weighted mean of light curves consistent with the data and the dispersion of those light curves --- they are not a particular realizations of a single light curve.

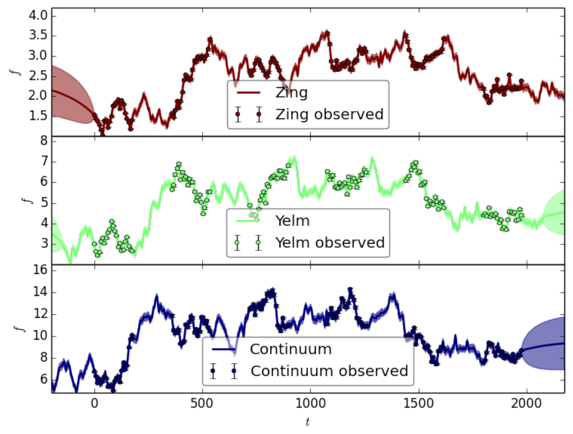

Fig. 7: Comparison between the simulated light curves as computed from the best-fit parameters, and the observed light curves.

For the photometric line light curve, just to demonstrate the photometric RM function of JAVELIN, we place a hard limit on range of lags during MCMC searching, so that the 180 ambiguity won't happen. Fig. 8 shows the posterior probability disgtribution of parameters in the photometric RM model

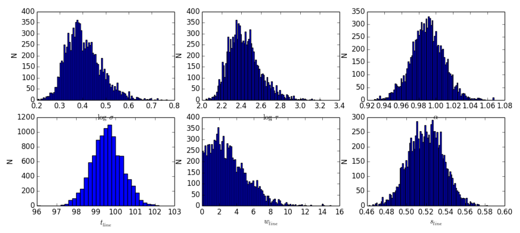

Fig. 8: As in Fig. 5, but after running JAVELIN for the Yelm band light curves plus the continuum using the photometric RM model.

Usage

To use JAVELIN, it is useful to have some a priori knowledge of Python, but not necessary. Here we will walk you through the actual procedures outlined in the last section. In this section, we will manipulate the files in two different terminals, one is the usual Unix command line marked by "$" in the beginning, one is the Python terminal started with ">>>".

Running JAVELIN is Easy

Lag determination can usually be done by JAVELIN within a few lines of codes. The following is a quick example of inferring lag using spectroscopic light curves.

Starting from the data files in the `examples/dat` directly:

```
$ cd javelin/examples/dat
```

Fire up a Python terminal (iPython is strongly recommened!),:

```
$ python
Python 2.7.2+ (default, Jan 20 2012, 23:05:38)
[GCC 4.6.2] on linux2
Type "help", "copyright", "credits" or "license" for m
>>>
```

and do

```
>>>from javelin.zylc import get_data
>>>from javelin.lcmodel import Cont_Model, Rmap_Model,
```

to load the necessary modules, then:

```
>>>c = get_data(["con.dat"])
>>>cmod = Cont_Model(c)
>>>cmod.do_mcmc()
```

to fit the continuum data, then:

```
>>>cy = get_data(["con.dat", "yelm.dat"])
>>>cymod = Rmap_Model(cy)
>>>cymod.do_mcmc(conthpd=cmod.hpd)
```

to fit the continuum+line data, where `Rmap_Model` is the the spectroscopic reverberation mapping~(RM) model. The results can be shown by:

```
>>>cymod.show_hist()
```

as the 1D posterior distributions of model parameters, including the lag t.

And finally, to do photometric RM using the continuum+line-band data, do:

```
>>>cyb = get_data(["con.dat", "yelmband.dat"])
>>>cybmod = Pmap_Model(cyb)
>>>cybmod.do_mcmc(conthpd=cmod.hpd)
```

where `Pmap_Model` is the two-band photometric RM model (there is also a one-band photometric RM model `SPmap_Model` available). Again, the results can be shown by:

```
>>>cybmod.show_hist()
```

For the more patient users, now I will go through each step in detail, starting from the supported data files.

Reading Light Curves

JAVELIN can work on two types of light curve files, the first one is the typical 3-column file like `con.dat`, `yelm.dat`, `zing.dat`, and `yelmband.dat` in the current directory. If you do:

```
$ head -n 3 con.dat
```

to show the first 3 rows of the continuum light curve file `con.dat`:

```
0.00000    6.75749    0.06846
8.00400    6.37599    0.06364
16.00800    5.74500    0.05907
```

where the 1st, 2nd, and 3rd columns are *the observing epoch*, *the light curve value*, and *the measurement uncertainty*, respectively. Since the basic data unit in JAVELIN is a `LightCurve` object, you need to read the data files through a function into the `LightCurve` object. Open a Python terminal in the `dat` directory and then do:

```
>>>from javelin.zylc import get_data
>>>javdata1 = get_data(["con.dat", "yelm.dat"], names=
```

to load the continuum light curve `con.dat` and the Yelm light curve `yelm.dat` into a `LightCurve` object called `javdata1`, with `names` as "Continuum" and "Yelm". The brackets `[]` tell JAVELIN that the two light curves should be analyzed in one set, and if you want to look at the light curves in figures just run:

```
>>>javdata1.plot()
```

Note that in Python you have to keep the parentheses even when no arguments are needed.

The second type of file JAVELIN uses is a slight variant of the 3-column format, like `loopdeloop_con.dat`, `loopdeloop_con_y.dat`, `loopdeloop_con_y_z.dat`, and `loopdeloop_con_yb.dat` in the current directory. As suggested by the names of these files, since JAVELIN usually works on several light curves simultaneously, it is useful (at least to me) to keep different set of data files separated (similar to the brackets used in the reading of 3-column files).

Imagine you want to fit two light curves, the first one should always be the continuum light curves and the second one the line light curve. If the continuum light curve has 5 data points while the line light curve has 4, the data file should be like (text after # are comments, not part of the file)

```
2                       # number of light curves, cont
5                       # number of data points in the
461.5  22.48    0.36    # each light curve entry consi
490.6  20.30    0.30
520.3  19.59    0.56
545.8  20.11    0.15
769.6  21.12    1.20
4                       # number of data points in the
545.8   9.82    0.23
890.4  11.86    0.58
949.4  10.55    0.87
988.6  11.06    0.27
```

To read the second type of file, simply do:

```
>>>javdata2 = get_data("loopdeloop_con_y.dat", names=['
```

Note right now there are only brackets from the `names`, but a single string for the input file. Given `loopdeloop_con_y.dat` is just another version of packing `con.dat` and `yelm.dat` together, `javdata` and `javedata2` are equivalent to each other. You can varify this by doing `javdata2.plot()`.

Constraining Continuum Variability

We can use JAVELIN to model the continuum variability, or as shown in the last section, for RM we need to fit the continuum light curve alone first to derive set priors on the DRW parameters for the second step of lag fitting. Since for now we only work on the continuum model, we can load the continuum light curve either by:

```
>>>javdata3 = get_data(["con.dat",], names=["Continuum"
```

or by:

```
>>>javdata3 = get_data("loopdeloop_con.dat", names=["Co
```

Note that the brakets are still needed even for loading a single light curve.

After loading the data, we need to set up a continuum model. In JAVELIN, the light curve models are described in the `javelin.lcmodel` module, for now we need to initiate the `Cont_Model` class:

```
>>>from javelin.lcmodel import Cont_Model
>>>cont = Cont_Model(javdata3)
```

By default, `Cont_Model` will model the light curve as a DRW process, but you are also specify models like matern, —*pow_exp*, —*kepler_exp*, etc (see details in Zu et al. 2011). However, currently the spectroscopic and photometric RM models —*Rmap_Model* and —*(S)PMap_Model* do not yet support continuum covariance models other than DRW.

Without exploring any further options, you can simply run:

```
>>>cont.do_mcmc(fchain="mychain0.dat")
```

to start a MCMC analysis and the chain will be saved into "mychain0.dat" file. By default, the chain will go through 5000 iterations for a burn-in period, and then another 5000 iterations for the actual chain. JAVELIN uses the kick-ass MCMC sampler named emcee introduced by Dan Foreman-Mackey et al (2012). `emcee` works by randomly releasing numerous `walkers` at every possible corner of the parameter space, which then collaboratively sample the posterior probability distributions, so you do not need to tell the sampler where to start. The number of `walkers`, the number of burn-in iterations, and the number of sampling iterations for each `walker` are specified by `nwalker` (default: 100), `nchain` (default: 50), and `nburn` (default: 50), respectively. For examples, if you want to double the chain length of both burn-in and sampling periods (well, you do not want to do it right now):

```
>>>cont.do_mcmc(nwalkers=100, nburn=100, nchain=100, f
```

The default values of `nwalker`, `nchain`, and `nburn` would usually be enough for fitting continuum or fitting continuum+one line, but the required values would rise quickly with the number of lines if you are doing fitting with muliple lines. So, whenever you find the MCMC chain does not converge well --- JAVELIN fail to find a unique combination of solutions but a broad lag distribution, try to increase these three parameters.

After sampling, you can check the 1D posterior distributions of tau and sigma:

```
>>>cont.show_hist(bins=100)
```

which looks like Fig. 4.

The output `fchain` is simply a two-column txt file with the first column log(sigma) and the second one log(tau), both natural logs. You can also store the log likelihoods as a separate chain in `flogp`.

Older chains can be reloaded for analysis by:

```
>>>cont.load_chain("mychain0.dat")
```

and the highest posterior density (HPD) intervals can be retrieved by:

```
>>>cont.get_hpd()
>>>conthpd = cont.hpd
>>>print(conthpd)
[[ 0.363  3.923]
  [ 0.518  4.29 ]
  [ 0.737  4.743]]
```

which is a 3x2 array with the three elements of the first (second) column being the 18%, 50%, and 84% values for log sigma (log tau). `cont.hpd` here is exactly what we are after in this subsection, as will become apparently below, to provide useful constraints on the DRW parameters to help determining lags,

Spectroscopic RM: Fitting the Continuum and one line (Yelm)

First, we need to load the necessary light curves files, in this

First, we need to load the necessary light curves files, in this case, both the continuum and the Ylem light curves, into a `LightCurve` object, which is simply the `javdata1` or the `javdata2` we created earlier. Also, we need to construct a model, this time a Continuum+Line model, which is called a `Rmap_Model` in JAVELIN:

```
>>>from javelin.lcmodel import Rmap_Model
>>>rmap1 = Rmap_Model(javdata1)
```

Remember that we need the results from fitting the continuum as priors on the DRW parameters in finding lags,

```
>>>rmap1.do_mcmc(conthpd=conthpd, fchain="mychain1.dat"
```

where `conthpd` is the HPD interval array we obtained from last subsection and `fchain` is again the file name for the output chain.

There are several interesting options that you can use to to tweak the MCMC sampler (you can always check the source for the full argument list):

```
>>>rmap1.do_mcmc(conthpd=conthpd, lagtobaseline=0.3, la
```

In particular, `lagtobaseline` indicates that a logarithmic prior is applied to logarithmically penalize lag values larger than `lagtobaseline` times the baseline of the continuum light curve (default: 0.3). `laglimit` gives the boundaries beyond which lag values are forbidden. The default is `baseline`, meaning no lags larger than the observation baseline (total span of the light curves), and its non-default value could only be a list of 2-element lists, indicating the range of the possible lag values for each emission line. In particular, after a first run with `laglimit=baseline`, you can use the results to narrow the boundaries for the new run with a higher convergence MCMC search. For example, you can narrow down the boundaries to between 100 and 200 days and rerun a finer MCMC search:

```
>>>rmap1.do_mcmc(conthpd=conthpd, fchain="mychain1_fin
```

where `laglimit` is a list that is comprised of a single 2-element list because we have only one emission line here.

The `emcee` sampler does multi-threading, so if your system has multiple cores, you should run the above command with `threads` set to the number of cores to speed things up:

```
>>>rmap1.do_mcmc(conthpd=conthpd, fchain="mychain1_fin
```

The other chain length related parameters are the same as in the continuum case.

After running the MCMC analysis, the 1D posterior distributions can be shown with:

```
>>>rmap1.show_hist()
```

which then looks like Fig. 5.

The output `fchain` file is comprised of 2+3*n columns, where n is the number of emission lines. Thus here we have 5 columns, with each column as, from left to right:

```
log(sigma), log(tau), lag, width, scale
```

and the number of columns augments by 3 for every additional emission line. Again, you can also store the log likelihoods as a separate chain using `flogp`. You can play with the `fchain` file in any way you like, but JAVELIN provides several tools to start with, for example,:

```
>>>rmap1.load_chain("mychain1.dat")
```

for reloading the chain file,:

```
>>>rmap1.break_chain([[100, 200],])
```

for abandoning the chain segments where the lag value is outside of [100, 200], and:

```
>>>rmap1.restore_chain()
```

to restore to the original untrimmed chain.

Usually the lag finding ends here if the 1D posterior distribution of lag shows a single peak, but sometimes you may want to fit two emission lines simultaneously to improve the results, as in our example of how fitting multiple lines eliminates seasonal aliasing problems.

Spectroscopic RM: Fitting the Continuum and two lines (Yelm and Zing)

The extrapolation from using one emission line to using two is rather trivial. Read the light curves by:

```
>>>javdata4 = get_data(["con.dat", "yelm.dat", "zing.da
```

set the model by:

```
>>>rmap2 = Rmap_Model(javdata4)
```

and lastly, run the models using MCMC:

```
>>>rmap2.do_mcmc(conthpd=conthpd, fchain="mychain2.dat"
```

if you have two cpus available.

In the loopdeeloop example here, the false peak seen in the last subsection should be largely eliminated, as shown by the 1D posteriors:

```
>>>rmap2.show_hist()
```

which looks like Fig. 6.

To isolate the peaks in the chain, you can do (assuming both peaks land between 100 and 300 days):

```
>>>rmap2.break_chain([[100, 300],[100, 300]])
```

Now you can retrieve and print out the HPD intervals for the double emission-line model fit:

```
>>>rmap2.get_hpd()
>>>rmap2hpd = rmap2.hpd
```

and the medians can be obtained by:

```
>>>par_best = rmap2hpd[1,:]
>>>print(par_best)
array([ 0.592, 4.262, 127.169, 0.525, 1.024, 254.262, (
```

which shows the median values for log(sigma), log(tau), lag_yelm, width_yelm, scale_yelm, lag_zing, width_zing, and scale_zing, respectively.

To make the story more completely, you can draw the best-fit light curves on top of the observed ones as shown in Fig. 7.:

```
>>>javdata_best =  rmap2.do_pred(par_best)
>>>javdata_best.plot(set_pred=True, obs=javdata4)
```

Spectroscopic RM in JAVELIN is Highly Extensible

If you have more than three light curves for the same objects at the same period, you also plug the additional lines in JAVELIN in the same way, simply by feeding a longer list of light curves to `get_data` and constructing a new `Rmap_Model`. The estimation will improve a lot if the additional emission lines have drastically different lags. However, the estimation may also become worse if the additional light curves are intrinsically noisy or the uncertainties are overly underestimated.

Another important issue in fitting more than two line is, as mentioned earlier in the manual, the default values for `nwalkers`, `nchain`, and `nburn` may not be adequate because you have a rapid increase in the dimensionality of the problem. Therefore, try to increase the values of these parameters whenever you find the MCMC does not converge well.

Photometric RM: Two-Band or One-Band

The Photometric RM module is as easy to use as the Spectroscopic one:

```
>>>javdata5 = get_data("loopdeloop_con_yb.dat", names=
>>>pmap = Pmap_Model(javdata5)
```

for loading the Two-Band Photometric RM model, and:

```
>>>javdata6 = get_data("yelmband.dat", names=["YelmBan
>>>spmap = SPmap_Model(javdata6)
```

For the Two-Band method, the procedure is similar to the Spectroscopic RM, where we constrain the continuum variability first and use that as prior information for the second step of calling `Pmap_Model`. However, for the One-Band method, since we do not have independent continuum information, we directly fit the single broad band light curve without using `conthpd`. You can either look into the `demo.py` code under `example` dir, or check the source code `lcmodel.py` under `javelin` dir for details.

Disk RM Model of Mudd et al. 2017

The Disk_Model object is developed recently by Mudd et al. 2017 , and what it does is take in a series of continuum light curves at known wavelengths/effective wavelengths, and find the best-fitting thin disk model for the data. The thin disk assumes that the size of the accretion disk scales as

$$R_\lambda = R_0\left(\frac{\lambda}{\lambda_0}\right)^\beta,$$

where R_λ is the disk size at wavelength λ and R_0 is the disk size at a reference wavelength λ_0. We can instead write this in terms of a time delay t between the emission at two wavelengths as

$$t = \frac{R_0}{c}\left[\left(\frac{\lambda}{\lambda_0}\right)^{\beta} - 1\right],$$

where again R_0 is the disk size at a reference wavelength λ_0 and t is the time delay between a feature at wavelength λ_0 and λ. More information on the model and further references can be found in Mudd et al. 2017.

For a usage example, you can find an example script at

```
javelin/examples/thindisk/test_thindisk.py
```

Let's say you have a driving light curve "driver.dat", and then three other continuum light curves "wave2.dat", "wave3.dat", and "wave4.dat", measured at wavelengths of 2000A, 4000A, 5000A, and 8000A. You can use the "get_data()" method to read your light curves right into your Disk_Model object, the only difference here compared to the other models is that you also need to specify the wavelengths of the light curves in addition to reading in the light curves with "get_data()".:

```
>>>disk1 = Disk_Model(get_data(["driver.dat", "wave2.da
```

Note that the "get_data()" method takes in a list of file names as before (or a singular file with multiple light curves formatted as directed in a previous example) and a list of names, but it is the Disk_Model object instance that requires the "effwave" parameter as well, which is a list or array of wavelengths for the light curve. Note that the Disk Model will always treat the first light curve in the list as the driver. You can then run your model exactly as in other RM models:

```
>>>disk1.do_mcmc(nwalkers=100, nburn=100, nchain=500,
```

In this example, the "thin_disk_chain.dat" will wind up being a 50000 line text file with 10 columns and fburn will be a 10000 line text file with 10 columns. These will correspond to the DRW model amplitude (sigma) and timescale (tau), same as for all of the previous examples on this page. Rather than providing a lag, tophat width, and tophat scale for each light curve, however, the thin disk model fits for the R_0 and β parameters. These can then be used to predict the time lags based on the effective wavelengths of the other light curves based on the earlier equations in this section. Each new light curve still needs a tophat width and scale, however. This means that each new light curve in the model adds 2 parameters rather than 3, and starts with 4 parameters (sigma, tau, R_0, and β). Thus, with three light curves being fit to our driver, we expect 10 parameters- sigma, tau, R_0, β, width_wave2, scale_wave2, width_wave3, scale_wave3, width_wave4, and scale_wave4. The R_0 parameter in the chain will correspond to the effective accretion disk size at a reference wavelength corresponding to that of "driver.dat".

Additional useful tools that might be helpful are the fact that, like with previous Model objects in JAVELIN, you may choose to supply DRW parameters at the outset with the "conthpd" parameter in "do_mcmc()", the ability to fix model parameters using a "fixed" array indicating which parameters are allowed to change and a supply of initial walker values in this case as "p_fix". Even though the lags are not fit for directly, you may still place boundaries on parameter spaces for each light curve using the "lagtobaseline" or "laglimit" parameters as with the "RMap_Model" before. Options unique to the Thin Disk object are the ability to set a top hat minimum width (in the same units as your light curve), as well as place limits on the R_0 and β parameters when initializing the "Disk_Model" object. Using our previous example,:

```
>>>disk1 = Disk_Model(get_data(["driver.dat", "wave2.da
```

would not allow top hat widths less than 0.1, R_0 outside the range of [-5, 5], or β outside the range of [-2, 2]. Again, note that the units on the top hat minimum width and R_0 are the same as those in your light curves.

Additional Information

Please refer to the JAVELIN source code for all the modules and their arguments (the code is in my humble opinion semi-well-documented).

Citation

You are welcome to use and modify JAVELIN, however please acknowledge its use either as is or with modifications with a citation to

Zu, Y., Kochanek, C.S., Kozlowski, S., & Udalski, A. 2013, ApJ, 765, 106

for quasar optical variability studies,

Zu, Y., Kochanek, C.S., & Peterson, B.M. 2011, ApJ, 735, 80

for spectroscopic reverberation mapping,

Zu, Y., Kochanek, C.S., Kozlowski, S., & Peterson, B.M. 2016,