radvel Documentation

Release 1.0.5

BJ Fulton, Erik Petigura, and Sarah Blunt

Jan 06, 2018

Contents

Welcome to the documentation for `radvel`, a Python package for modeling of radial velocity time series data.

Please cite the folloing DOI if you make use of RadVel in any publication. Contents:

CHAPTER 1

Getting Started

1.1 Installation

Install `radvel` using pip:

```
$ pip install radvel
```

Make sure that `pdflatex` is installed and in your system's path. You can get `pdflatex` by installing the TexLive package or other LaTeX distributions. By default it is expected to be in your system's path, but you may specify a path to pdflatex using the `--latex-compiler` option at the `radvel report` step.

1.2 Example Fit

Test your installation by running through one of the included examples. We will use the `radvel` command line interface to execute a multi-planet, multi-instrument fit.

The `radvel` binary should have been automatically placed in your system's path by the `pip` command (see *Installation*). If your system can not find the `radvel` executable then try running `python setup.py install` from within the top-level `radvel` directory.

First lets look at `radvel --help` for the available options:

```
$ radvel --help
usage: RadVel [-h] [--version] {fit,plot,mcmc,derive,bic,table,report} ...

RadVel: The Radial Velocity Toolkit

optional arguments:
  -h, --help            show this help message and exit
  --version             Print version number and exit.

subcommands:
  {fit,plot,mcmc,derive,bic,table,report}
```

Here is an example workflow to run a simple fit using the included *HD164922.py* example configuration file. This example configuration file can be found in the `example_planets` subdirectory on the GitHub repository page.

Perform a maximum-likelihood fit. You almost always will need to do this first:

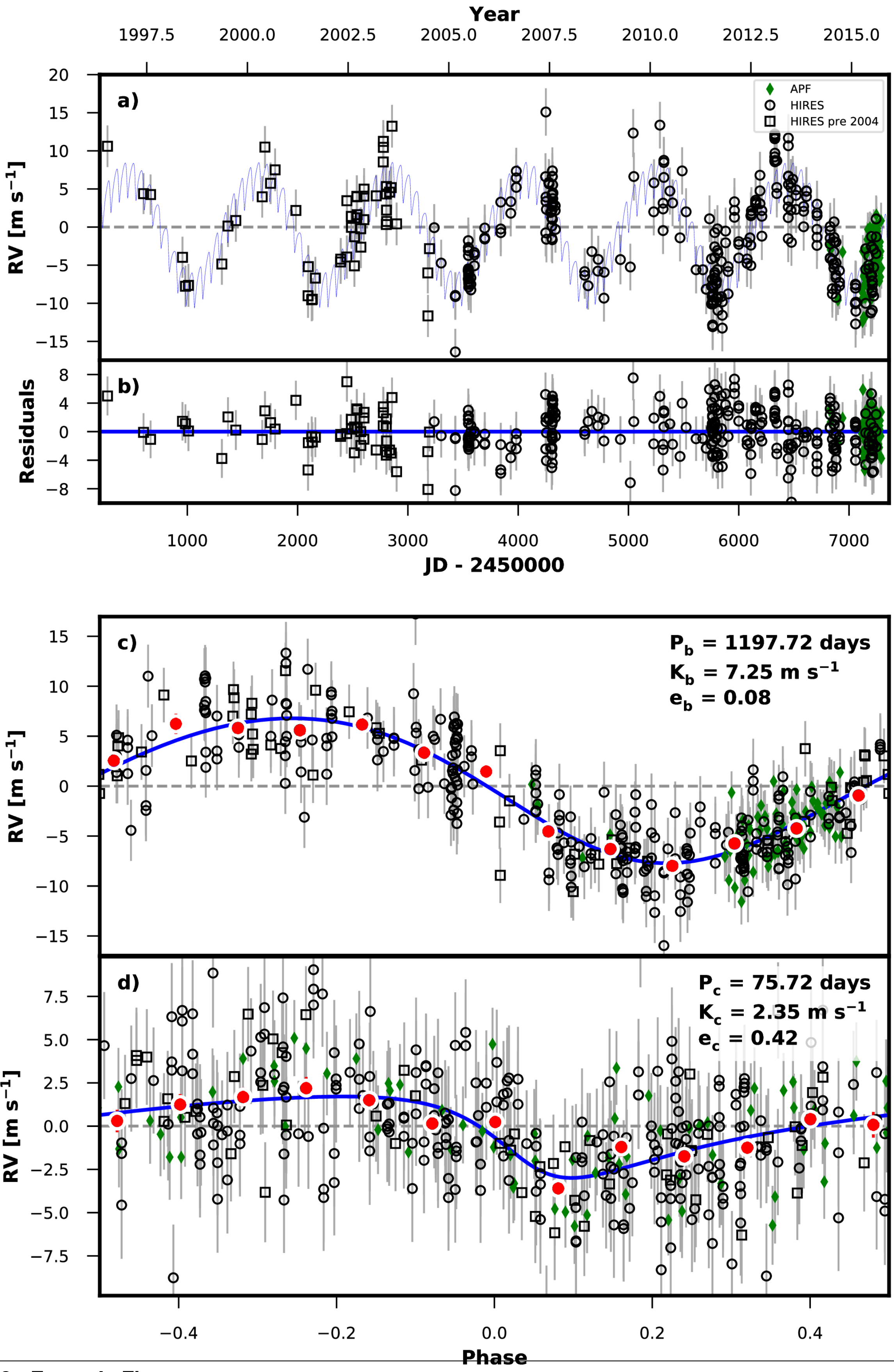
```
$ radvel fit -s /path/to/radvel/example_planets/HD164922.py
```

By default the results will be placed in a directory with the same name as your planet configuration file (without *.py*, e.g. *HD164922*). You may also specify an output directory using the `-o` flag.

After the maximum-likelihood fit is complete the directory should have been created and should contain one new file: *HD164922/HD164922_post_obj.pkl*. This is a `pickle` binary file that is not meant to be human-readable but lets make a plot of the best-fit solution contained in that file:

```
$ radvel plot -t rv -s /path/to/radvel/example_planets/HD164922.py
```

This should produce a plot named *HD164922_rv_multipanel.pdf* that looks something like this.


Year
1997.5
2000.0
2002.5
2005.0
2007.5
2010.0
2012.5
2015.0
a)
APF
HIRES
HIRES pre 2004
RV [m s^{-1}]
b)
Residuals
JD - 2450000
c)
P_b = 1197.72 days
K_b = 7.25 m s^{-1}
e_b = 0.08
d)
P_c = 75.72 days
K_c = 2.35 m s^{-1}
e_c = 0.42
Phase

Next lets perform the Markov-Chain Monte Carlo (MCMC) exploration to assess parameter uncertainties.

```
$ radvel mcmc -s /path/to/radvel/example_planets/HD164922.py
```

Once the MCMC chains finish running there will be another new file called *HD164922_mcmc_chains.csv.tar.bz2*. This is a compressed csv file containing the parameter values and likelihood at each step in the MCMC chains.

Now we can update the RV time series plot with the MCMC results and generate the full suite of plots.

```
$ radvel plot -t rv corner trend -s /path/to/radvel/example_planets/HD164922.py
```

Then create a LaTeX document and corresponding PDF to summarize the results.

```
$ radvel report -s /path/to/radvel/example_planets/HD164922.py
```

The report PDF will be saved as *HD164922_results.pdf*. It should contain a table reporting the parameter values and uncertainties, a table summarizing the priors, the RV time-series plot, and a corner plot showing the posterior distributions for all free parameters.

1.3 Optional Features

Combine the measured properties of the RV time-series with the properties of the host star defined in the setup file to derive physical parameters for the planetary system. Have a look at the *epic203771098.py* example setup file to see how to include stellar parameters.

```
$ radvel derive -s /path/to/radvel/example_planets/HD164922.py
```

Generate a corner plot for the derived parameters. This plot will also be included in the summary report if available.

```
$ radvel plot -t derived -s /path/to/radvel/example_planets/HD164922.py
```

Perform a model comparison testing models with progressively fewer planets. If this is run a new table will be included in the summary report.

```
$ radvel bic -t nplanets -s /path/to/radvel/example_planets/HD164922.py
```

Generate and save only the TeX code for any/all of the tables.

```
$ radvel table -t params priors nplanets -s /path/to/radvel/example_planets/HD164922.
↪py
```

CHAPTER 2

Advanced Usage

These tutorials give some examples in the use of the underlying `radvel` API. They are also available as interactive iPython notebooks in the *tests* subdirectory of the radvel package.

2.1 K2-24 Fitting & MCMC

Using the K2-24 (EPIC-203771098) dataset, we demonstrate how to use the `radvel` API to:

- perform a max-likelihood fit
- do an MCMC exploration of the posterior space
- plot the results

Perform some preliminary imports:

```
import os

import matplotlib
import numpy as np
import pylab as pl
import pandas as pd
from scipy import optimize

import corner

import radvel
import radvel.plotting
```

Define a function that we will use to initialize the `radvel.Parameters` and `radvel.RVModel` objects

```
def initialize_model():
    time_base = 2420
    params = radvel.Parameters(2,basis='per tc secosw sesinw logk') # number of␣
↪planets = 2
```

```
    params['per1'] = radvel.Parameter(value=20.885258)
    params['tc1'] = radvel.Parameter(value=2072.79438)
    params['secosw1'] = radvel.Parameter(value=0.01)
    params['sesinw1'] = radvel.Parameter(value=0.01)
    params['logk1'] = radvel.Parameter(value=1.1)
    params['per2'] = radvel.Parameter(value=42.363011)
    params['tc2'] = radvel.Parameter(value=2082.62516)
    params['secosw2'] = radvel.Parameter(value=0.01)
    params['sesinw2'] = radvel.Parameter(value=0.01)
    params['logk2'] = radvel.Parameter(value=1.1)
    mod = radvel.RVModel(params, time_base=time_base)
    mod.params['dvdt'] = radvel.Parameter(value=-0.02)
    mod.params['curv'] = radvel.Parameter(value=0.01)
    return mod
```

Define a simple plotting function to display the data and model.

```
def plot_results(like):
    fig = pl.figure(figsize=(12,4))
    fig = pl.gcf()
    fig.set_tight_layout(True)
    pl.errorbar(
        like.x, like.model(t)+like.residuals(),
        yerr=like.yerr, fmt='o'
        )
    pl.plot(ti, like.model(ti))
    pl.xlabel('Time')
    pl.ylabel('RV')
    pl.draw()
```

Load up the K2-24 data. In this example the RV data is stored in an CSV file

```
path = os.path.join(radvel.DATADIR,'epic203771098.csv')
rv = pd.read_csv(path)

t = np.array(rv.t)
vel = np.array(rv.vel)
errvel = rv.errvel
ti = np.linspace(rv.t.iloc[0]-5,rv.t.iloc[-1]+5,100)
```

2.1.1 Circular Orbits

Use the function we just defined to initialize a model object and add a few additional parameters into the `radvel.likelihood.Likelihood` object that are not associated with the Keplerian orbital model but still needed to calculate a likelihood.

```
mod = initialize_model()
like = radvel.likelihood.RVLikelihood(mod, t, vel, errvel)
like.params['gamma'] = radvel.Parameter(value=0.1)
like.params['jit'] = radvel.Parameter(value=1.0)
```

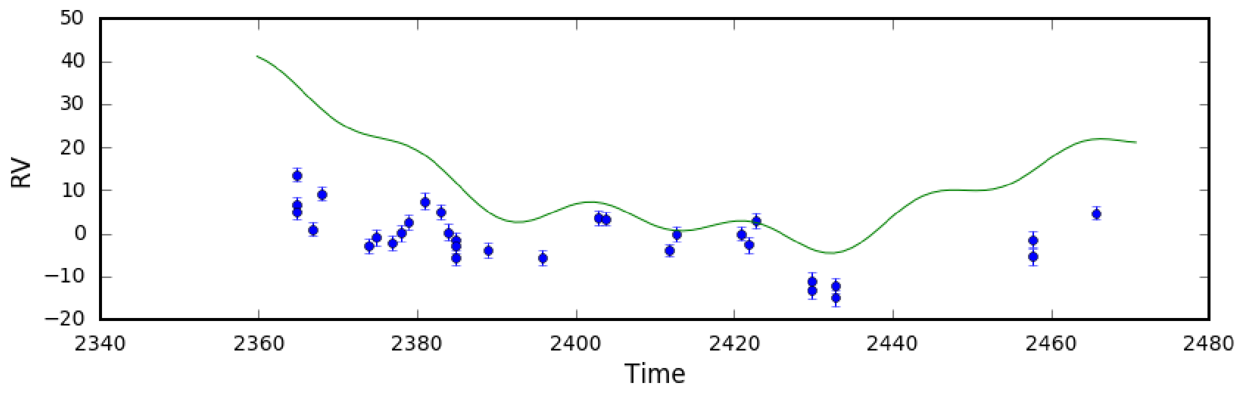
Choose which parameters to vary or fix. By default, all `radvel.Parameter` objects will vary, so you only have to worry about setting the ones you want to hold fixed.

```
like.params['secosw1'].vary = False
like.params['sesinw1'].vary = False
like.params['secosw2'].vary = False
like.params['sesinw2'].vary = False
like.params['per1'].vary = False
like.params['per2'].vary = False
like.params['tc1'].vary = False
like.params['tc2'].vary = False
print(like)
```

```
parameter                    value      vary
per1                       20.8853     False
tc1                        2072.79     False
secoswl                       0.01     False
sesinw1                       0.01     False
logk1                          1.1      True
per2                        42.363     False
tc2                        2082.63     False
secosw2                       0.01     False
sesinw2                       0.01     False
logk2                          1.1      True
dvdt                         -0.02      True
curv                          0.01      True
gamma                          0.1      True
jit                              1      True
```

Plot the initial model

```
pl.figure()
plot_results(like)
```



Well that solution doesn't look very good. Now lets try to optimize the parameters set to vary by maximizing the likelihood.

Initialize a `radvel.Posterior` object and add some priors

```
post = radvel.posterior.Posterior(like)
post.priors += [radvel.prior.Gaussian( 'jit', np.log(3), 0.5)]
post.priors += [radvel.prior.Gaussian( 'logk2', np.log(5), 10)]
post.priors += [radvel.prior.Gaussian( 'logk1', np.log(5), 10)]
post.priors += [radvel.prior.Gaussian( 'gamma', 0, 10)]
```

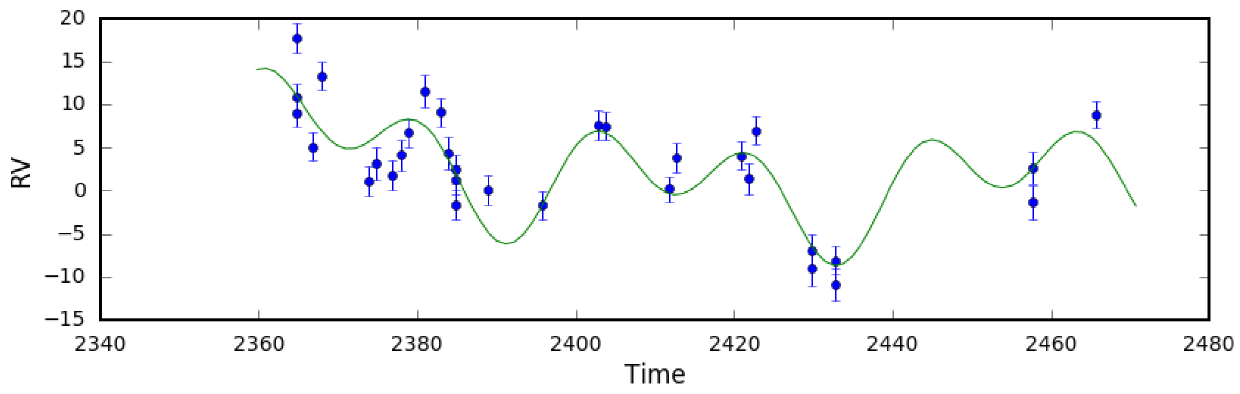
Maximize the likelihood and print the updated posterior object

```
res  = optimize.minimize(
    post.neglogprob_array,     # objective function is negative log likelihood
    post.get_vary_params(),    # initial variable parameters
    method='Powell',           # Nelder-Mead also works
    )

plot_results(like)             # plot best fit model
print(post)
```

```
parameter                     value      vary
per1                        20.8853      False
tc1                         2072.79      False
secosw1                        0.01      False
sesinw1                        0.01      False
logk1                       1.56037       True
per2                         42.363      False
tc2                         2082.63      False
secosw2                        0.01      False
sesinw2                        0.01      False
logk2                       1.80937       True
dvdt                     -0.0364432       True
curv                    -0.00182455       True
jit                         2.62376       True
gamma                       2.62376       True

Priors
------
Gaussian prior on jit, mu=1.09861228867, sigma=0.5
Gaussian prior on logk2, mu=1.60943791243, sigma=10
Gaussian prior on logk1, mu=1.60943791243, sigma=10
Gaussian prior on gamma, mu=0, sigma=10
```



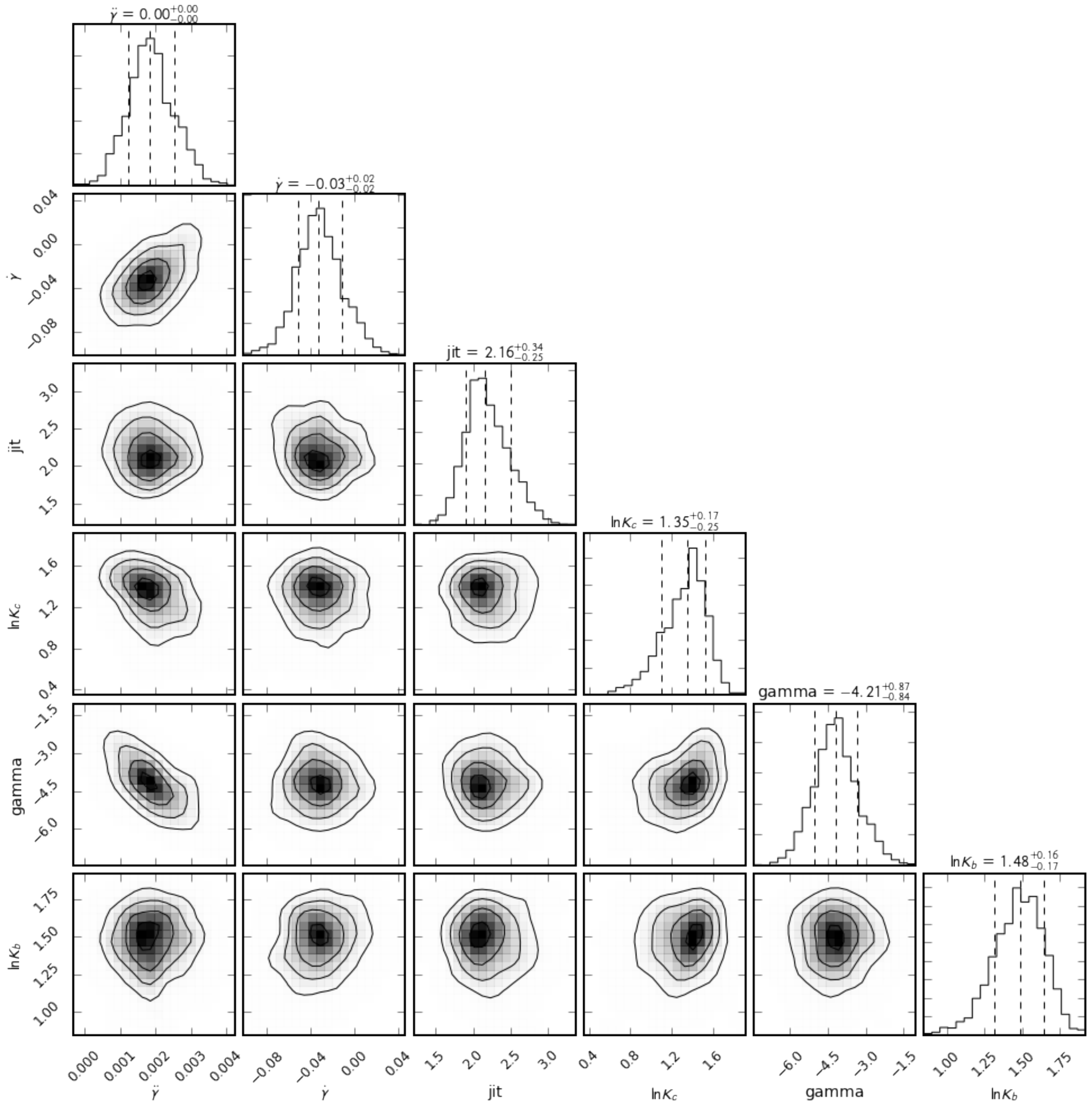
That looks much better!

Now lets use Markov-Chain Monte Carlo (MCMC) to estimate the parameter uncertainties. In this example we will run 1000 steps for the sake of speed but in practice you should let it run at least 10000 steps and ~50 walkers. If the chains converge before they reach the maximum number of allowed steps it will automatically stop.

```
df = radvel.mcmc(post,nwalkers=20,nrun=1000)
```

Make a corner plot to display the posterior distributions.

```
radvel.plotting.corner_plot(post, df)
```



2.1.2 Eccentric Orbits

Allow `secosw` and `sesinw` parameters to vary

```
like.params['secosw1'].vary = True
like.params['sesinw1'].vary = True
like.params['secosw2'].vary = True
like.params['sesinw2'].vary = True
```

Add an `EccentricityPrior` to ensure that eccentricity stays below 1.0. In this example we will also add a

Gaussian prior on the jitter (`jit`) parameter with a center at 2.0 m/s and a width of 0.1 m/s.

```
post = radvel.posterior.Posterior(like)
post.priors += [radvel.prior.EccentricityPrior( 2 )]
post.priors += [radvel.prior.Gaussian( 'jit', np.log(2), np.log(0.1))]
```

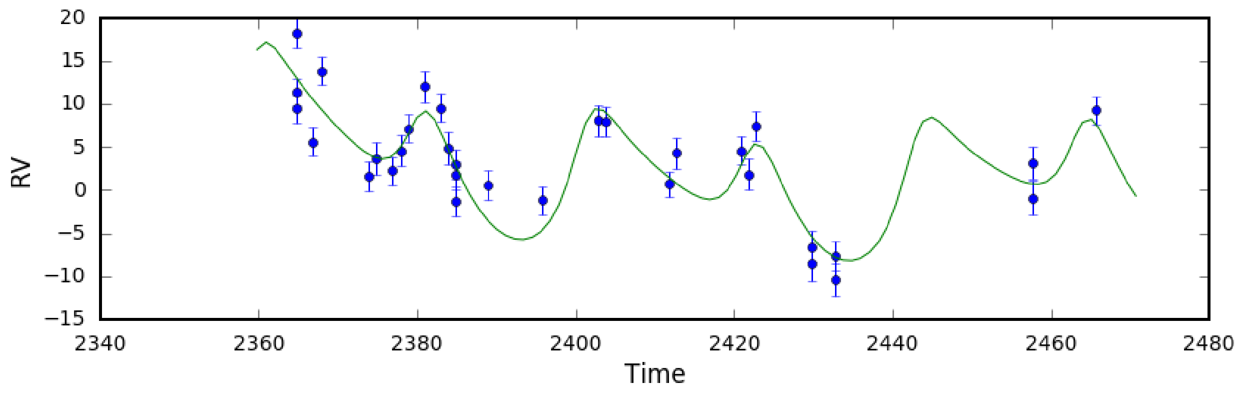
Optimize the parameters by maximizing the likelihood and plot the result

```
res  = optimize.minimize(
    post.neglogprob_array,
    post.get_vary_params(),
    method='Nelder-Mead',)

plot_results(like)
print(post)
```

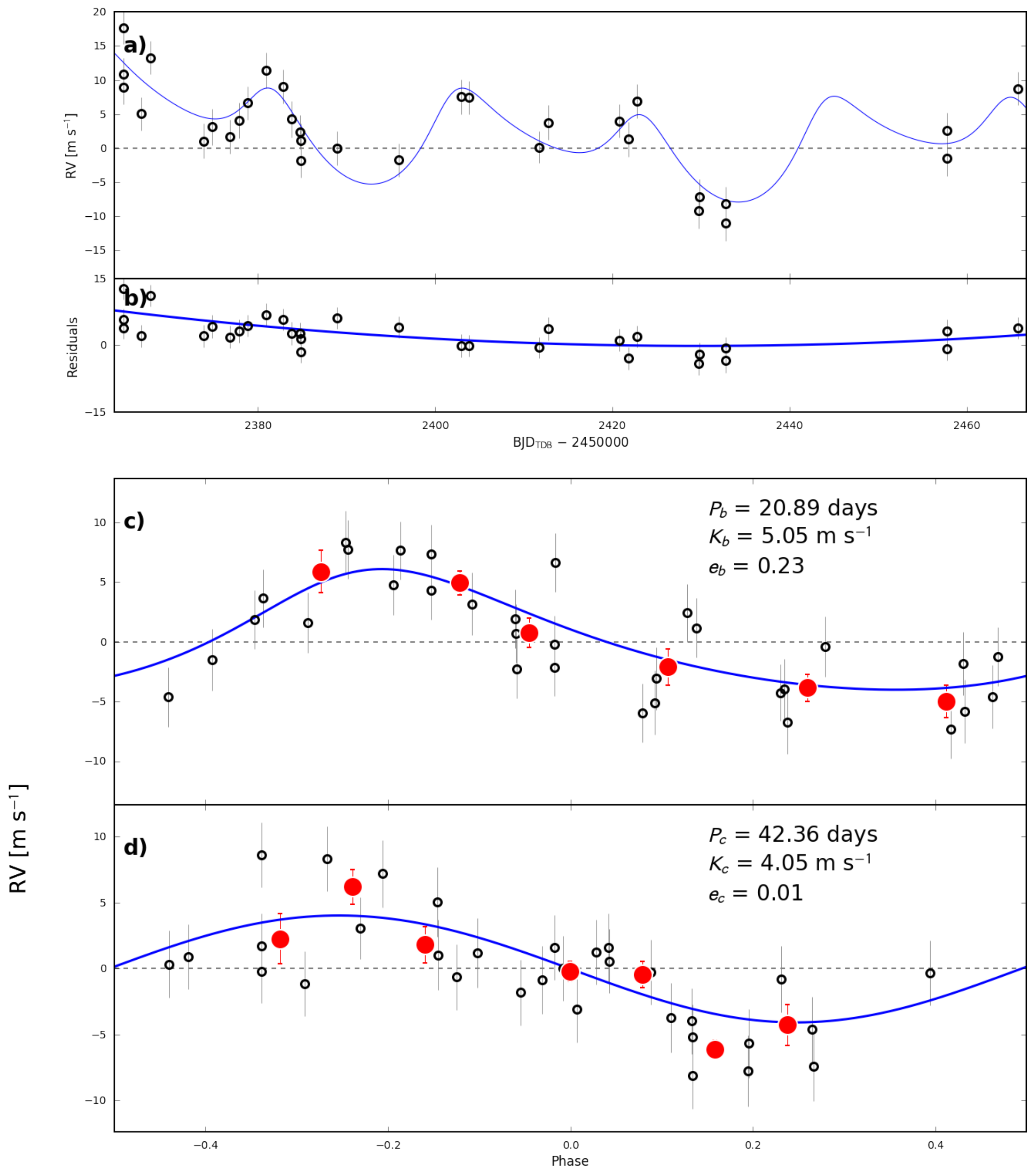
```
parameter                     value      vary
per1                        20.8853      False
tc1                         2072.79      False
secosw1                    0.389104       True
sesinw1                    0.059227       True
logk1                       1.65139       True
per2                         42.363      False
tc2                         2082.63      False
secosw2                    0.194769       True
sesinw2                   -0.422685       True
logk2                        1.6278       True
dvdt                      -0.027433       True
curv                     0.00152703       True
gamma                      -4.38996       True
jit                          2.2025       True

Priors
------
e1 constrained to be < 0.99
e2 constrained to be < 0.99
Gaussian prior on jit, mu=0.6931471805599453, sigma=-2.3025850929940455
```



Plot the final solution

```
radvel.plotting.rv_multipanel_plot(post)
```


a)
b)
c)
d)
RV [m s^{-1}]
Residuals
BJD$_{TDB}$ − 2450000
P_b = 20.89 days
K_b = 5.05 m s^{-1}
e_b = 0.23
P_c = 42.36 days
K_c = 4.05 m s^{-1}
e_c = 0.01
RV [m s^{-1}]
Phase

CHAPTER 3

API

3.1 The RV Model

class radvel.model.**Parameter**(*value=None*, *vary=True*, *mcmcscale=None*)
Object to store attributes of each orbital parameter

value
float – value of parameter.

vary
Bool – True if parameter is allowed to vary in MCMC fits, false if fixed.

mcmcscale
float – step size to be used for MCMC fitting

class radvel.model.**Parameters**(*num_planets*, *basis='per tc secosw sesinw logk'*, *planet_letters=None*)
Object to store the orbital parameters.

Parameters to describe a radial velocity orbit stored as an OrderedDict

Parameters

- **num_planets** (*int*) – Number of planets in model
- **basis** (*string*) – parameterization of orbital parameters. See `radvel.basis.Basis` for a list of valid basis strings.
- **planet_letters** (*Dictionary[optional]*) – custom map to match the planet numbers in the Parameter object to planet letters. Default {1: 'b', 2: 'c', etc.}. The keys of this dictionary must all be integers.

basis
radvel.Basis – Basis object

planet_parameters
list – orbital parameters contained within the specified basis

num_planets
int – number of planets in the model

Examples

```
>>> import radvel
# create a Parameters object for a 2-planet system with
# custom planet number to letter mapping
>>> params = radvel.Parameters(2, planet_letters={1:'d', 2:'e'})
```

tex_labels(*param_list=None*)
Map Parameters keys to pretty TeX code representations.

Parameters **param_list** (*list*) – (optional) Manually pass a list of parameter labels

Returns dictionary mapping Parameters keys to TeX code

Return type dict

class radvel.model.**RVModel**(*params*, *time_base=0*)
Generic RV Model

This class defines the methods common to all RV modeling classes. The different RV models, having different parameterizations inherit from this class.

3.2 Keplerian Orbits

radvel.kepler.**kepler**(*inbigM*, *inecc*)
Solve Kepler's Equation

Parameters

- **inbigM** (*array*) – input Mean anomaly
- **inecc** (*array*) – eccentricity

Returns array

Return type eccentric anomaly

radvel.kepler.**rv_drive**(*t*, *orbel*, *use_c_kepler_solver=True*)
RV Drive

Parameters

- **t** (*array of floats*) – times of observations
- **orbel** (*array of floats*) – [per, tp, e, om, K]. Omega is expected to be in radians
- **use_c_kepler_solver** (*bool*) – (default: True) If True use the Kepler solver written in C, else use the Python/NumPy version.

Returns (array of floats): radial velocity model

Return type rv

radvel.orbit.**timeperi_to_timetrans**(*tp*, *per*, *ecc*, *omega*, *secondary=False*)
Convert Time of Periastron to Time of Transit

Parameters

- **tp** (*float*) – time of periastron
- **per** (*float*) – period [days]
- **ecc** (*float*) – eccentricity
- **omega** (*float*) – argument of peri (radians)
- **secondary** (*bool*) – calculate time of secondary eclipse instead

Returns time of inferior conjuntion (time of transit if system is transiting)

Return type float

`radvel.orbit.`**`timetrans_to_timeperi`**(*tc*, *per*, *ecc*, *omega*)
Convert Time of Transit to Time of Periastron Passage

Parameters

- **tc** (*float*) – time of transit
- **per** (*float*) – period [days]
- **ecc** (*float*) – ecceсntricity
- **omega** (*float*) – longitude of periastron (radians)

Returns time of periastron passage

Return type float

`radvel.orbit.`**`true_anomaly`**(*t*, *tp*, *per*, *e*)
Calculate the true anomoly for a given time, period, eccentricity.

Parameters

- **t** (*array*) – array of times in JD
- **tp** (*float*) – time of periastron, same units as t
- **per** (*float*) – orbital period in days
- **e** (*float*) – eccentricity

Returns true anomoly at each time

Return type array

3.3 Orbital Parameter Basis Sets

class `radvel.basis.`**`Basis`**(**args*)
Object that knows how to convert between the various Keplerian bases

Parameters

- **name** (*str*) – basis name
- **num_planets** (*int*) – number of planets

synth_params
str – name of synth basis

Note: Valid basis functions:

'per tp e w k' (The synthesis basis)

'per tc secosw sesinw logk'

'per tc secosw sesinw k'

'per tc ecosw esinw k'

'per tc e w k'

'logper tc secosw sesinw k'

'logper tc secosw sesinw logk'

---

from_synth (*params_in*, *newbasis*, ***kwargs*)

Convert from synth basis into another basis

Convert instance of Parameters with parameters of a given basis into the synth basis

Parameters

- **params_in** (*radvel.Parameters or pandas.DataFrame*) – radvel.Parameters object or pandas.Dataframe containing orbital parameters expressed in current basis
- **newbasis** (*string*) – string corresponding to basis to switch into
- **keep** (*Optional[bool]*) – keep the parameters expressed in the old basis, else remove them from the output dictionary/DataFrame

Returns dict or dataframe with the parameters converted into the new basis

to_any_basis (*params_in*, *newbasis*)

Convenience function for converting Parameters object to an arbitraty basis

Parameters

- **params_in** (*radvel.Parameters*) – radvel.Parameters object expressed in current basis
- **newbasis** (*string*) – string corresponding to basis to switch into

Returns radvel.Parameters object expressed in the new basis

to_synth (*params_in*, ***kwargs*)

Convert to synth basis

Convert Parameters object with parameters of a given basis into the synth basis

Parameters

- **params_in** (*radvel.Parameters or pandas.DataFrame*) – radvel.Parameters object or pandas.Dataframe containing orbital parameters expressed in current basis
- **noVary** (*Optional[bool]*) – if True, set the 'vary' attribute of the returned Parameter objects to '' (used for displaying best fit parameters)

Returns parameters expressed in the synth basis

Return type *Parameters* or DataFrame

3.4 Likelihood Functions

class `radvel.likelihood.`**`Likelihood`**(*model*, *x*, *y*, *yerr*, *extra_params=[]*, *decorr_params=[]*, *decorr_vectors=[]*)

Generic Likelihood

class `radvel.likelihood.`**`RVLikelihood`**(*model*, *t*, *vel*, *errvel*, *suffix=''*, *decorr_vars=[]*, *decorr_vectors=[]*)

RV Likelihood

The Likelihood object for a radial velocity dataset

Parameters

- **`model`** (`radvel.model.RVModel`) – RV model object
- **`t`** (*array*) – time array
- **`vel`** (*array*) – array of velocities
- **`errvel`** (*array*) – array of velocity uncertainties
- **`suffix`** (*string*) – suffix to identify this Likelihood object useful when constructing a *CompositeLikelihood* object.

`errorbars`()

Return uncertainties with jitter added in quadrature.

Returns uncertainties

Return type array

`logprob`()

Return log-likelihood given the data and model. Priors are not applied here.

Returns Natural log of likelihood

Return type float

`residuals`()

Residuals

Data minus model

`radvel.likelihood.`**`loglike_jitter`**(*residuals*, *sigma*, *sigma_jit*)

Log-likelihood incorporating jitter

See equation (1) in Howard et al. 2014. Returns loglikelihood, where sigma**2 is replaced by sigma**2 + sigma_jit**2. It penalizes excessively large values of jitter

Parameters

- **`residuals`** (*array*) – array of residuals
- **`sigma`** (*array*) – array of measurement errors
- **`sigma_jit`** (*float*) – jitter

Returns log-likelihood

Return type float

3.5 The Posterior Object

class radvel.posterior.**Posterior**(*likelihood*)

Posterior object

Posterior object to be sent to the fitting routines. It is essentially the same as the Liklihood object, but priors are applied here.

Parameters

- **likelihood** (radvel.likelihood.Likelihood) – Likelihood object
- **params** (radvel.model.Parameters) – parameters object

Note: Append *radvel.prior.Prior* objects to the Posterior.priors list to apply priors in the likelihood calculations.

bic()

Calculate the Bayesian information criterion

Returns BIC

Return type float

logprob()

Log probability

Log-probability for the likelihood given the list of priors in *Posterior.priors*.

Returns log probability of the likelihood + priors

Return type float

logprob_array(*param_values_array*)

Log probability for parameter vector

Same as *self.logprob*, but will take a vector of parameter values. Useful as the objective function for routines that optimize a vector of parameter values instead of the dictionary-like format of the *radvel.model.Parameters* object.

Returns log probability of the likelihood + priors

Return type float

writeto(*filename*)

Save posterior object to pickle file.

Parameters **filename** (*string*) – full path to outputfile

radvel.posterior.**load**(*filename*)

Load posterior object from pickle file.

Parameters **filename** (*string*) – full path to pickle file

3.6 Priors

class radvel.prior.**EccentricityPrior**(*num_planets*, *upperlims=0.99*)

Physical eccentricities

Prior to keep eccentricity between 0 and a specified upper limit.

Parameters

- **num_planets** (*int or list*) – Planets to apply the eccentricity prior. If an integer is given then all planets with indexes up to and including the specified integer will be included in the prior. If a list is given then the prior will only be applied to the specified planets.
- **upperlims** (*float or list of floats*) – List of eccentricity upper limits to assign to each of the planets. If a float is given then all planets must have eccentricities less then this value. If a list of floats is given then each planet can have a different eccentricity upper limit.

class radvel.prior.**Gaussian**(*param*, *mu*, *sigma*)
Gaussian prior

Guassian prior on a given parameter.

Parameters

- **param** (*string*) – parameter label
- **mu** (*float*) – center of Gaussian prior
- **sigma** (*float*) – width of Gaussian prior

class radvel.prior.**HardBounds**(*param*, *minval*, *maxval*)
Prior for hard boundaries

This prior allows for hard boundaries to be established for a given parameter.

Parameters

- **param** (*string*) – parameter label
- **minval** (*float*) – minimum allowed value
- **maxval** (*float*) – maximum allowed value

class radvel.prior.**PositiveKPrior**(*num_planets*)
K must be positive

A prior to prevent K going negative. Be careful with this as it can introduce a bias to larger K values.

Parameters **num_planets** (*int*) – Number of planets. Used to ensure K for each planet is positive

class radvel.prior.**SecondaryEclipsePrior**(*planet_num*, *ts*, *ts_err*)
Secondary eclipse prior

Implied prior on eccentricity and omega by specifying measured secondary eclipse time

Parameters

- **planet_num** (*int*) – Number of planet with measured secondary eclipse
- **ts** (*float*) – Secondary eclipse midpoint time. Should be in the same units as the timestamps of your data.
- **ts_err** (*float*) – Uncertainty on secondary eclipse time

3.7 Maximum Likelihood Fitting

radvel.fitting.**maxlike_fitting**(*post*, *verbose=True*)
Maximum Likelihood Fitting

Perform a maximum likelihood fit.

Parameters

- **`post`** (*`radvel.Posterior`*) – Posterior object with initial guesses
- **`verbose`** (*`bool`*) – (optional) Print messages and fitted values?

Returns: radvel.Posterior [Posterior object with parameters] updated their maximum likelihood values

`radvel.fitting.`**`model_comp`**(*post*, *verbose=False*)
Model Comparison

Fit for planets adding one at a time. Save results as list of posterior objects.

Parameters

- **`post`** (*`radvel.Posterior`*) – posterior object for final best-fit solution with all planets
- **`verbose`** (*`bool`*) – (optional) print out statistics

Returns

List of dictionaries with fit statistics. Each value in the dictionary is a tuple with the statistic value as the first element and a description of that statistic in the second element.

Return type list of dictionaries

3.8 MCMC Fitting

`radvel.mcmc.`**`convergence_check`**(*samplers*, *maxGR*, *minTz*, *minsteps*)
Check for convergence Check for convergence for a list of emcee samplers

Parameters

- **`samplers`** (*`list`*) – List of emcee sampler objects
- **`maxGR`** (*`float`*) – Maximum G-R statistic for chains to be deemed well-mixed and halt the MCMC run
- **`minTz`** (*`int`*) – Minimum Tz to consider well-mixed
- **`minsteps`** (*`int`*) – Minimum number of steps per walker before convergence tests are performed

`radvel.mcmc.`**`draw_models_from_chain`**(*mod*, *chain*, *t*, *nsamples=50*)
Draw Models from Chain

Given an MCMC chain of parameters, draw representative parameters and synthesize models. :param mod: RV model :type mod: radvel.RVmodel :param chain: pandas DataFrame with different values from MCMC

chain

Parameters

- **`t`** (*`array`*) – time range over which to synthesize models
- **`nsamples`** (*`int`*) – number of draws

Returns 2D array with the different models as different rows

Return type array

`radvel.mcmc.`**`gelman_rubin`**(*pars0*, *minTz*, *maxGR*)

Gelman-Rubin Statistic Calculates the Gelman-Rubin statistic and the number of independent draws for each parameter, as defined by Ford et al. (2006) (http://adsabs.harvard.edu/abs/2006ApJ…642..505F). The chain is considered well-mixed if all parameters have a Gelman-Rubin statistic of <= 1.03 and >= 1000 independent draws. History:

2010/03/01 - Written: Jason Eastman - The Ohio State University 2012/10/08 - Ported to Python by BJ Fulton - University of Hawaii,

Institute for Astronomy

2016/04/20 - Adapted for use in radvel. Removed "angular" parameter.

Parameters

- **`pars0`** (*`array`*) – A 3 dimensional array (NPARS,NSTEPS,NCHAINS) of parameter values
- **`minTz`** (*`int`*) – minimum Tz to consider well-mixed
- **`maxGR`** (*`float`*) – maximum Gelman-Rubin statistic to consider well-mixed

Returns

tuple containing: ismixed (bool): Are the chains well-mixed? gelmanrubin (array): An NPARS element array containing the

Gelman-Rubin statistic for each parameter (equation 25)

Tz (array): An NPARS element array containing the number of independent draws for each parameter (equation 26)

Return type (tuple)

`radvel.mcmc.`**`mcmc`**(*post*, *nwalkers=50*, *nrun=10000*, *ensembles=8*, *checkinterval=50*, *burnGR=1.03*, *maxGR=1.01*, *minTz=1000*, *minsteps=1000*)

Run MCMC Run MCMC chains using the emcee EnsambleSampler :param post: radvel posterior object :type post: radvel.posterior :param nwalkers: (optional) number of MCMC walkers :type nwalkers: int :param nrun: (optional) number of steps to take :type nrun: int :param ensembles: (optional) number of ensembles to run. Will be run

in parallel on separate CPUs

Parameters

- **`checkinterval`** (*`int`*) – (optional) check MCMC convergence statistics every *checkinterval* steps
- **`burnGR`** (*`float`*) – (optional) Maximum G-R statistic to stop burn-in period
- **`maxGR`** (*`float`*) – (optional) Maximum G-R statistic for chains to be deemed well-mixed and halt the MCMC run
- **`minTz`** (*`int`*) – (optional) Minimum Tz to consider well-mixed
- **`minsteps`** (*`int`*) – (optional) Minimum number of steps per walker before convergence tests are performed

Returns DataFrame containing the MCMC samples

Return type DataFrame

3.9 Pipeline Driver Functions

Driver functions for the radvel pipeline.These functions are meant to be used only withthe *cli.py* command line interface.

`radvel.driver.`**`bic`**(*args*)
Compare different models and comparative statistics

Parameters **`args`** (*`ArgumentParser`*) – command line arguments

`radvel.driver.`**`derive`**(*args*)
Derive physical parameters from posterior samples

Parameters **`args`** (*`ArgumentParser`*) – command line arguments

`radvel.driver.`**`fit`**(*args*)
Perform maximum-likelihood fit

Parameters **`args`** (*`ArgumentParser`*) – command line arguments

`radvel.driver.`**`load_status`**(*statfile*)
Load pipeline status

Parameters **`statfile`** (*`string`*) – name of configparser file

Returns configparser.RawConfigParser

`radvel.driver.`**`mcmc`**(*args*)
Perform MCMC error analysis

Parameters **`args`** (*`ArgumentParser`*) – command line arguments

`radvel.driver.`**`plots`**(*args*)
Generate plots

Parameters **`args`** (*`ArgumentParser`*) – command line arguments

`radvel.driver.`**`report`**(*args*)
Generate summary report

Parameters **`args`** (*`ArgumentParser`*) – command line arguments

`radvel.driver.`**`save_status`**(*statfile*, *section*, *statevars*)
Save pipeline status

Parameters

- **`statfile`** (*`string`*) – name of output file
- **`section`** (*`string`*) – name of section to write
- **`statevars`** (*`dict`*) – dictionary of all options to populate the specified section

`radvel.driver.`**`tables`**(*args*)
Generate TeX code for tables in summary report

Parameters **`args`** (*`ArgumentParser`*) – command line arguments

3.10 Utility Functions

`radvel.utils.`**`Msini`**(*K*, *P*, *Mtotal*, *e*, *Msini_units='earth'*)
Calculate Msini

Calculate Msini for a given K, P, stellar mass, and e

Parameters

- **`K`** (*`float`*) – Doppler semi-amplitude [m/s]
- **`P`** (*`float`*) – Orbital period [days]
- **`Mtotal`** (*`float`*) – Mass of star + mass of planet [Msun]
- **`e`** (*`float`*) – eccentricity
- **`=`** (*`Msini_units`*) –

Returns Msini [units = Msini_units]

Return type float

`radvel.utils.`**`date2jd`**(*date*)

Convert datetime object to JD"

Parameters **`date`** (*`datetime.datetime`*) – date to convert

Returns Julian date

Return type float

`radvel.utils.`**`density`**(*mass*, *radius*, *MR_units='earth'*)

Parameters

- **`mass`** (*`float`*) – mass, units = MR_units
- **`radius`** (*`float`*) – radius, units = MR_units
- **`MR_units`** – (optional) units of mass and radius. Must be 'earth', or 'jupiter' (default 'earth').

Returns density (g/cc)

`radvel.utils.`**`geterr`**(*vec*, *angular=False*)

Calculate median, 15.9, and 84.1 percentile values for a given vector."

Parameters

- **`vec`** (*`array`*) – vector, usually an MCMC chain for one parameter
- **`angular`** (*`bool`*) – (optional) Is this an angular parameter? if True vec should be in radians. This will perform some checks to ensure proper boundary wrapping.

Returns 50, 15.9 and 84.1 percentiles

Return type tuple

`radvel.utils.`**`jd2date`**(*jd*)

Convert JD to datetime.datetime object

Parameters **`jd`** (*`float`*) – Julian date

Returns calendar date

Return type datetime.datetime

`radvel.utils.`**`semi_amplitude`**(*Msini*, *P*, *Mtotal*, *e*, *Msini_units='jupiter'*)

Compute Doppler semi-amplitude

Parameters

- **`Msini`** (*`float`*) – mass of planet [Mjup]

- **P** (*float*) – Orbital period [days]
- **Mtotal** (*float*) – Mass of star + mass of planet [Msun]
- **e** (*float*) – eccentricity
- **Msini_units** (*string*) – Units of returned Msini. Must be 'earth', or 'jupiter' (default 'jupiter').

Returns Doppler semi-amplitude [m/s]

radvel.utils.**sigfig**(*med*, *errlow*, *errhigh=None*)

Format values with errors into an equal number of signficant figures.

Parameters

- **med** (*float*) – median value
- **errlow** (*float*) – lower errorbar
- **errhigh** (*float*) – upper errorbar

Returns (med,errlow,errhigh) rounded to the lowest number of significant figures

Return type tuple

radvel.utils.**working_directory**(**args*, ***kwds*)

Do something in a directory

Function to use with *with* statements.

Parameters **dir** (*string*) – name of directory to work in

Example

```
>>> with workdir('/temp'):
    # do something within the /temp directory
```

3.11 Plotting

radvel.plotting.**add_anchored**(**args*, ***kwargs*)

Parameters

- **s** (*string*) – Text.
- **loc** (*str*) – Location code.
- **pad** (*float, optional*) – Pad between the text and the frame as fraction of the font size.
- **borderpad** (*float, optional*) – Pad between the frame and the axes (or *bbox_to_anchor*).
- **prop** (*matplotlib.font_manager.FontProperties*) – Font properties.

radvel.plotting.**corner_plot**(*post*, *chains*, *saveplot=None*)

Make a corner plot from the output MCMC chains and a posterior object.

Parameters

- **post** (*radvel.Posterior*) – Radvel posterior object

- **chains** (*DataFrame*) – MCMC chains output by radvel.mcmc
- **saveplot** (*str, optional*) – Name of output file, will show as interactive matplotlib window if not defined.

Returns None

radvel.plotting.**corner_plot_derived_pars** (*chains*, *planet*, *saveplot=None*)
Make a corner plot from the output MCMC chains and a posterior object.

Parameters

- **chains** (*DataFrame*) – MCMC chains output by radvel.mcmc
- **planet** (*Planet object*) – Planet configuration object
- **(Optional[string]** (*saveplot*) – Name of output file, will show as interactive matplotlib window if not defined.

Returns None

radvel.plotting.**correlation_plot** (*post*, *outfile=None*)
Correlation plot

Plot parameter correlations.

Parameters

- **post** (*radvel.Posterior*) – Radvel Posterior object
- **outfile** (*string*) – name of output multi-page PDF file

Returns None

radvel.plotting.**rv_multipanel_plot** (*post*, *saveplot=None*, *telfmts={}*, *nobin=False*, *yscale_auto=False*, *yscale_sigma=3.0*, *nophase=False*, *epoch=2450000*, *uparams=None*, *phase_ncols=None*, *phase_nrows=None*, *legend=True*, *rv_phase_space=0.08*)
Multi-panel RV plot to display model using post.params orbital parameters.

Parameters

- **post** (*radvel.Posterior*) – Radvel posterior object. The model plotted will be generated from post.params
- **saveplot** (*string, optional*) – Name of output file, will show as interactive matplotlib window if not defined.
- **nobin** (*bool, optional*) – If True do not show binned data on phase plots. Will default to True if total number of measurements is less then 20.
- **yscale_auto** (*bool, optional*) – Use matplotlib auto y-axis scaling (default: False)
- **yscale_sigma** (*float, optional*) – Scale y-axis limits to be +/- yscale_sigma*(RMS of data plotted) if yscale_auto==False
- **telfmts** (*dict, optional*) – dictionary of dictionaries mapping instrument code to plotting format code.
- **nophase** (*bool, optional*) – Will omit phase-folded plots if true
- **epoch** (*float, optional*) – Subtract this value from the time axis for more compact axis labels (default: 245000)
- **uparams** (*dict, optional*) – parameter uncertainties, must contain 'per', 'k', and 'e' keys.

- **phase_ncols** (*int, optional*) – number of columns in the phase folded plots. Default behavior is 1.
- **phase_nrows** (*int, optional*) – number of columns in the phase folded plots. Default is nplanets.
- **legend** (*bool, optional*) – include legend on plot? (default: True)
- **rv_phase_space** (*float, optional*) – verticle space between rv plot and phase-folded plots (in units of fraction of figure height)

Returns current matplotlib figure object list: list of axis objects

Return type figure

`radvel.plotting.`**`trend_plot`**(*post*, *chains*, *nwalkers*, *outfile=None*)

MCMC trend plot

Make a trend plot to show the evolution of the MCMC as a function of step number.

Parameters

- **post** (*radvel.Posterior*) – Radvel Posterior object
- **chains** (*DataFrame*) – MCMC chains output by radvel.mcmc
- **nwalkers** (*int*) – number of walkers used in this particular MCMC run
- **outfile** (*string*) – name of output multi-page PDF file

Returns None

3.12 LaTeX Report

class `radvel.report.`**`RadvelReport`**(*planet*, *post*, *chains*, *compstats=None*)

Radvel report Class to handle the creation of the radvel summary PDF :param planet: planet configuration object loaded in :type planet: planet object :param *kepfit.py* using *imp.load_source* post: :type *kepfit.py* using *imp.load_source* post: radvel.posterior :param radvel.posterior object containing the best-fit parameters in: :param post.params chains: output DataFrame from a :type post.params chains: DataFrame :param *radvel.mcmc* run:

`compile`(*pdfname*, *latex_compiler='pdflatex'*, *depfiles=[]*)

Compile radvel report Compile the radvel report from a string containing TeX code and save the resulting PDF to a file. :param pdfname: name of the output PDF file :type pdfname: string :param latex_compiler: path to latex :type latex_compiler: string :param depfiles: list of file names of dependencies needed for

LaTex compilation (e.g. figure files)

`figtex`(*infile*, *caption=''*)

Generate TeX for figure Generate TeX to insert a figure into the report :param infile: file name of figure :type infile: string :param caption: (optional) figure caption :type caption: string

Returns TeX code

Return type string

`texdoc`()

TeX for entire document TeX code for the entire output results PDF :returns: TeX code for report :rtype: string

class radvel.report.**TexTable**(*report*)

LaTeX table Class to handle generation of the LaTeX tables within the summary PDF. :param report: radvel report object :type report: radvel.report.RadvelReport

comp_table(*statsdict*)

Model comparisons Compare models with increasing number of planets :returns: String containing TeX code for the model comparison table :rtype: string

prior_summary()

Summary of priors Summarize the priors in separate table within the report PDF. :returns: String containing TeX code for the prior summary table :rtype: string

tex(*tabtype='all'*, *compstats=None*)

TeX code for table :returns: TeX code for the results table in the radvel report. :rtype: string

CHAPTER 4

Indices and tables for Python code

- genindex
- modindex
- search

Python Module Index

Index