La Forge Documentation

Release 1.0.2

Jeffrey Shafiq Hazbboun

Dec 01, 2022

CONTENTS:

1 La Forge **1**
1.1 Features 2

2 Indices and tables **41**

Python Module Index **43**

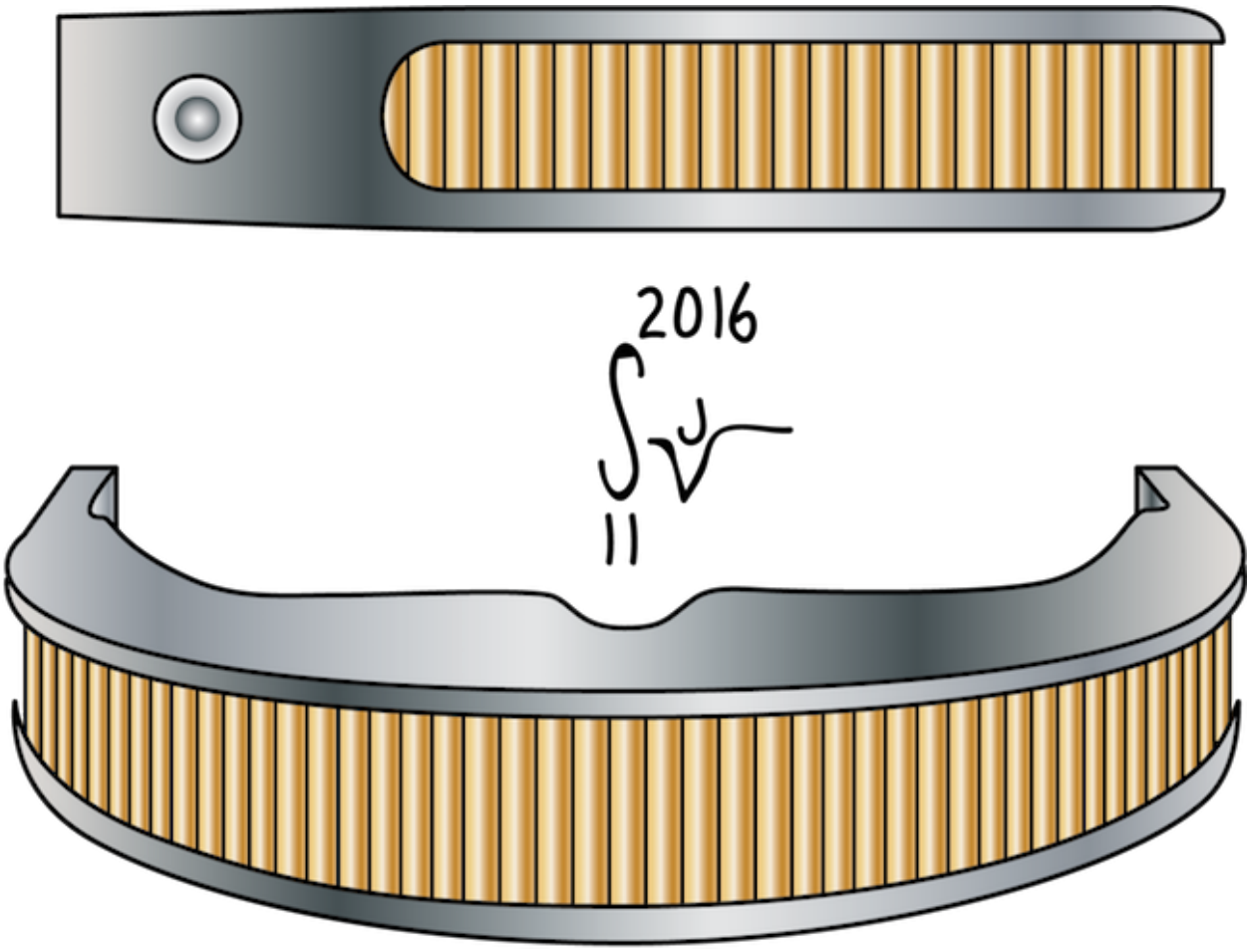
Index **45**

CHAPTER

ONE

LA FORGE

Pulsar Timing Array Bayesian Data Visualization



Graphic Credit: Stewart Vernon, via Deviant Art

Python package for conveniently plotting results from pulsar timing array bayesian analyses. Many of the functions are best used with enterprise outputs.

La Forge is available on PyPI:

```
pip install la-forge
```

- Free software: MIT license
- Documentation: https://la-forge.readthedocs.io.

1.1 Features

- Sweep up Bayesian analysis MCMC chains along with sampling info.
- Allow easy retrieval of various samples from chains.
- Support for saving chains as *HDF5* files.
- Call chains with parameter names.
- Plot posteriors easily.
- Reconstruct Gaussian process realizations using posterior chains.
- Plot red noise power spectral density.
- Separate consituent models of a hypermodel analysis.

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

1.1.1 Installation

Stable release

To install La Forge, run this command in your terminal:

```
$ pip install la-forge
```

This is the preferred method to install La Forge, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for La Forge can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/Hazboun6/la_forge
```

Or download the tarball:

```
$ curl  -OL https://github.com/Hazboun6/la_forge/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

1.1.2 Tutorial #1 Basic Core Usage

```
import la_forge.core as co
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import numpy as np
import json
```

Loading a Chain Directory

```
chaindir = '/Users/hazboun/software_development/la_forge/tests/data/chains/ng12p5yr_pint_
↪be/'
```

```
c0 = co.Core(chaindir=chaindir,
             label='NG12.5-year Pint Bayes Ephem Tests')
```

```
print('Number of parameters: \t',len(c0.params))
print('Chain shape: \t\t',c0.chain.shape)
print('Burn: \t\t\t' , c0.burn)
```

```
Number of parameters:        102
Chain shape:                 (6500, 102)
Burn:                        1625
```

```
c0.params[:10]
```

```
['B1855+09_red_noise_gamma',
 'B1855+09_red_noise_log10_A',
 'B1953+29_red_noise_gamma',
 'B1953+29_red_noise_log10_A',
 'J0023+0923_red_noise_gamma',
 'J0023+0923_red_noise_log10_A',
 'J0030+0451_red_noise_gamma',
 'J0030+0451_red_noise_log10_A',
 'J0340+4130_red_noise_gamma',
 'J0340+4130_red_noise_log10_A']
```

Access Parameters as Keywords

Retrieve a single parameter's samples, post burn-in.

```
c0('gw_log10_A')
```

```
array([-14.56648872, -14.56648872, -14.56648872, ..., -14.59112768,
       -14.59112768, -14.59112768])
```

Retrieve a parameter's samples with no burn-in.

```
c0('gw_log10_A',to_burn=False)
```

```
array([-14.60087212, -14.60087212, -14.60087212, ..., -14.59112768,
       -14.59112768, -14.59112768])
```

Retrieve multiple parameters' samples, post burn-in.

```
c0(['J1944+0907_red_noise_gamma',
    'J1944+0907_red_noise_log10_A',
    'J2010-1323_red_noise_gamma',
    'J2010-1323_red_noise_log10_A'])
```

```
array([[  1.25097717, -15.11251267,   4.60928109, -14.5367517 ],
       [  1.25097717, -15.11251267,   4.60928109, -14.5367517 ],
       [  1.25097717, -15.11251267,   4.60928109, -14.5367517 ],
       ...,
       [  1.59585533, -19.7683706 ,   4.99654344, -19.22860778],
       [  1.59585533, -13.91543988,   4.99654344, -19.22860778],
       [  1.59585533, -13.91543988,   4.99654344, -19.22860778]])
```

Parameter Statistics

Retrieve multiple parameters' 68% credible intervals

```
c0.credint(['J1909-3744_red_noise_gamma',
            'J1909-3744_red_noise_log10_A'],
           interval=68)
```

```
array([[  0.56467805,   4.55240021],
       [-17.62162001, -13.92578082]])
```

Retrieve single parameter's 95% upper limit

```
c0.credint('gw_log10_A',interval=95,onesided=True)
```

```
-14.561571336129667
```

Retrieve multiple parameters' median values

```
c0.median(['J1909-3744_red_noise_gamma',
           'J1909-3744_red_noise_log10_A'])
```

```
array([  1.78939389, -14.42312396])
```

```
c0.median('J1909-3744_red_noise_gamma')
```

```
1.789393894714808
```

Set the burn in as an integer

```
c0.set_burn(600)
c0.burn
```

```
600
```

Set the burn in as a fraction of the chain length

```
c0.set_burn(0.5)
c0.burn
```

```
3250
```

Get the *maximum a postori* index

```
c0.map_idx
```

```
2349
```

Get the *maximum a postori* values

```
c0.map_params[:20]
```

```
array([  4.6528546 , -14.24849806,   2.41262458, -12.7869943 ,
         0.56952096, -13.04800569,   6.18038659, -14.97272291,
         4.94995876, -15.20737186,   1.08256527, -13.31662541,
         6.19081461, -16.02476786,   4.36074626, -16.36712724,
         0.99736931, -13.79920502,   4.5261634 , -15.47236191])
```

Retrieve a *maximum a postori* dictionary and save it as a noise file.

```
with open('noise_file.json','w')as fout:
    json.dump(c0.get_map_dict(),fout)
```

Jump Proposal Acceptance

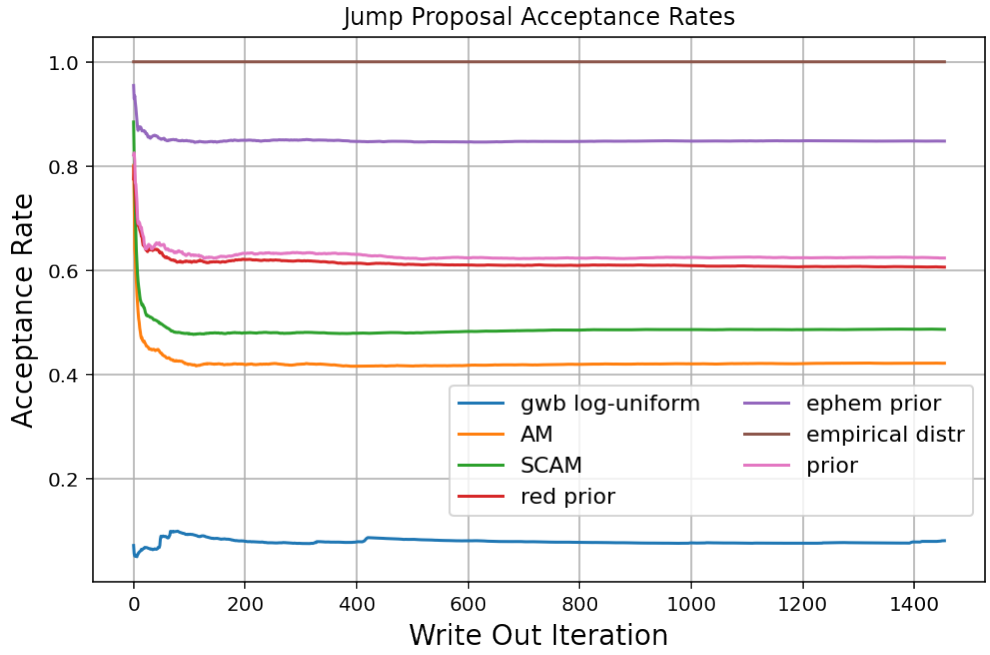
Plot the jump proposal acceptance for all of the sampled proposals.

```
plt.figure(figsize=[8,5])


for ii,ky in enumerate(c0.jumps.keys()):
    if ii>=9:
        ls='--'
    else:
        ls='-'
    if (ky=='jumps') or (ky=='DEJump_jump'):
        pass
    else:
        if ky[0]=='c':
            lab = 'SCAM' if 'SCAM' in ky else 'AM'
        elif ky=='DEJump_jump':
```

(continues on next page)

(continued from previous page)

```
            lab = 'DEJump'
        else:
            lab = ' '.join(np.array(ky.split('_'))[2:-1])
            if 'gwb' in lab:
                lab = 'gwb log-uniform'
        if lab == 'DEJump':
            deL = c0.jumps[ky].size
            jL = c0.jumps['covarianceJumpProposalAM_jump'].size

            nums = np.linspace(jL-deL,jL,deL)
            plt.plot(nums,c0.jumps[ky],label=lab,ls=ls,lw=1.5)
        else:
            plt.plot(c0.jumps[ky],label=lab,ls=ls,lw=1.5)

plt.grid()
plt.legend(loc=[0.4,0.12],ncol=2,fontsize=11)
plt.ylabel('Acceptance Rate',fontsize=14)
plt.xlabel('Write Out Iteration',fontsize=14)
plt.title('Jump Proposal Acceptance Rates')
plt.show()
```



Fractional breakdown of various jump proposals

```
c0.jump_fractions
```

```
{'draw_from_red_prior': 0.071,
 'covarianceJumpProposalAM': 0.11,
 'draw_from_empirical_distr': 0.071,
 'draw_from_gwb_log_uniform_distribution': 0.071,
 'draw_from_prior': 0.036,
```

(continues on next page)

(continued from previous page)

```
 'draw_from_ephem_prior': 0.071,
 'DEJump': 0.36,
 'covarianceJumpProposalSCAM': 0.21}
```

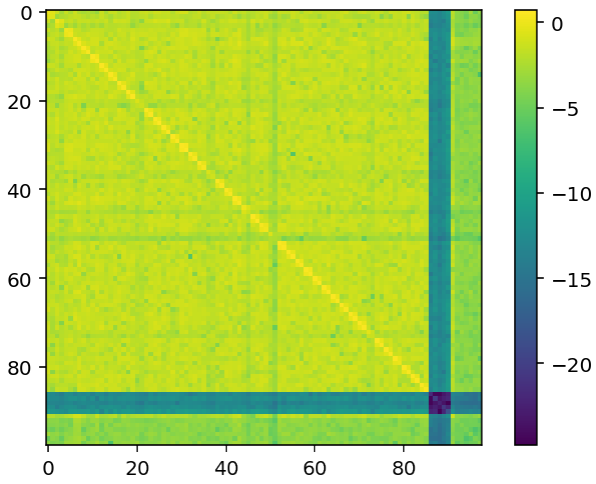
Runtime Information

```
print(c0.runtime_info[:960])
```

```
system : Linux
node : compute-105.mycluster
release : 3.10.0-1160.42.2.el7.x86_64
version : #1 SMP Tue Sep 7 14:49:57 UTC 2021
machine : x86_64

enterprise_extensions v2.3.3
enterprise v3.2.1.dev30+gffe69bf,  Python v3.9.7
================================================================================================

Signal Name                                 Signal Class                    no. Parameters
================================================================================================
B1855+09_marginalizing_linear_timing_model TimingModel                       0

params:
------------------------------------------------------------------------------------------------
↪
B1855+09_red_noise                          FourierBasisGP                  2

params:
B1855+09_red_noise_log10_A:Uniform(pmin=-20, pmax=-11)
B1855+09_red_noise_gamma:Uniform(pmin=0, pmax=7)
```

Parameter Covariance Matrix

```
plt.imshow(np.log10(abs(c0.cov)))
plt.colorbar()
plt.show()
```



1.1.3 Tutorial #2 Plotting Posteriors

```
import la_forge.core as co
import la_forge.diagnostics as dg
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import numpy as np
```

```
coredir = '/Users/hazboun/software_development/la_forge/tests/data/cores/'
```

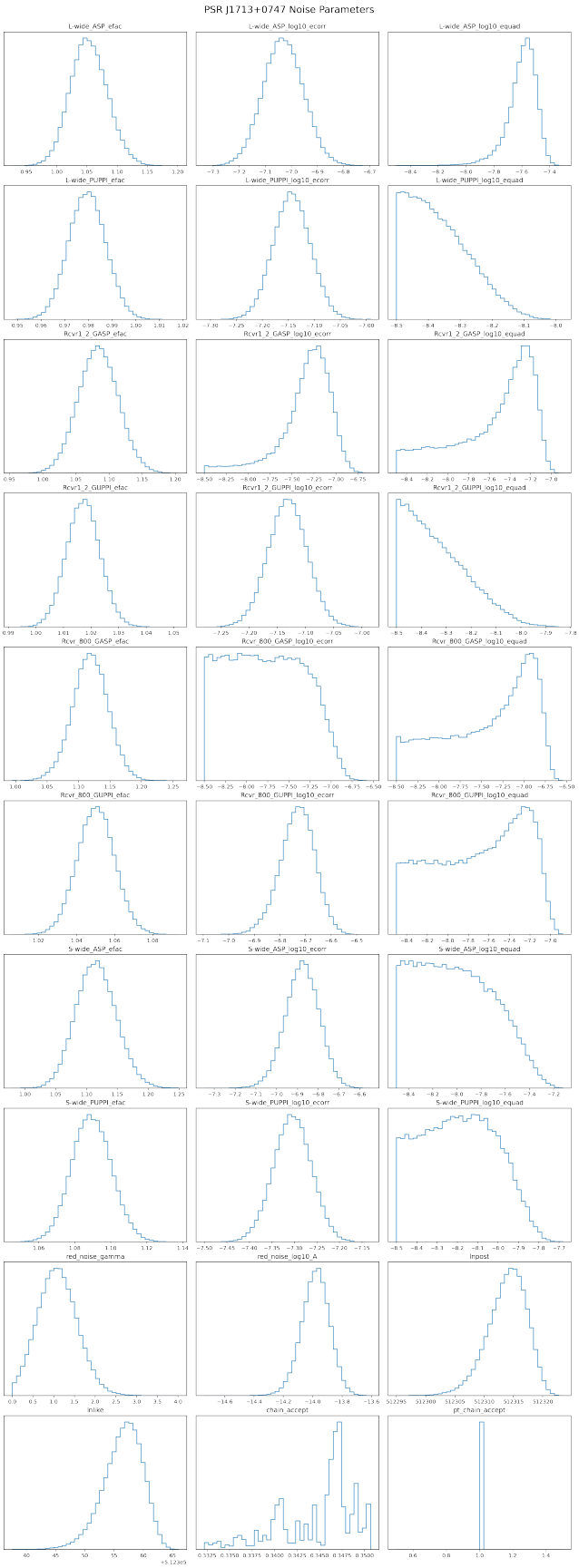
```
c0 = co.Core(corepath=coredir+'J1713+0747_plaw_dmx.core',
             label='NG12.5yr Noise Run: Power Law Red Noise')
```

```
Loading data from HDF5 file....
```

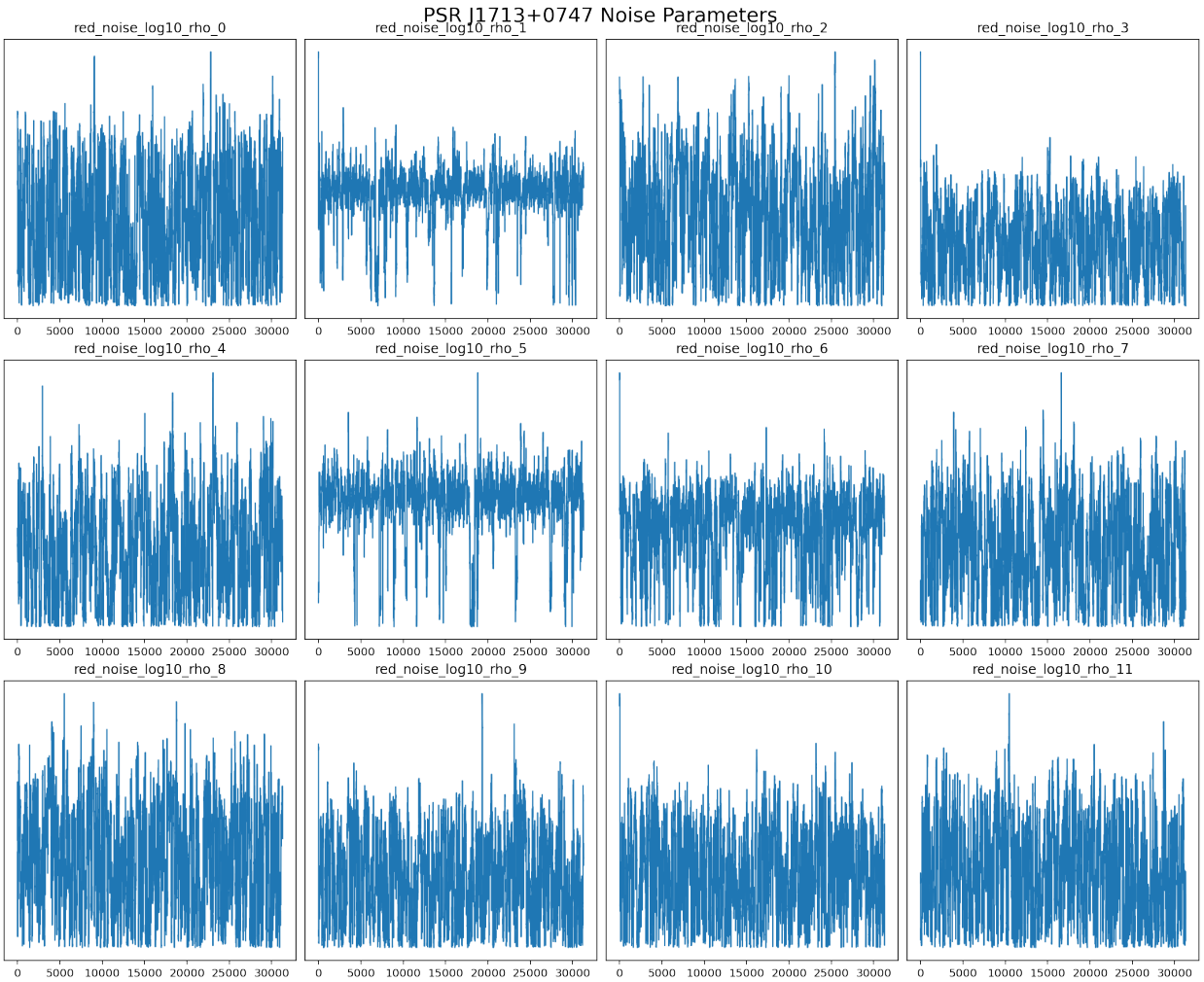
```
c1 = co.Core(corepath=coredir+'J1713+0747_fs_dmx.core',
             label='NG12.5yr Noise Run: Free Spectral Red Noise')
```

```
Loading data from HDF5 file....
```

```
dg.plot_chains(c0)
```

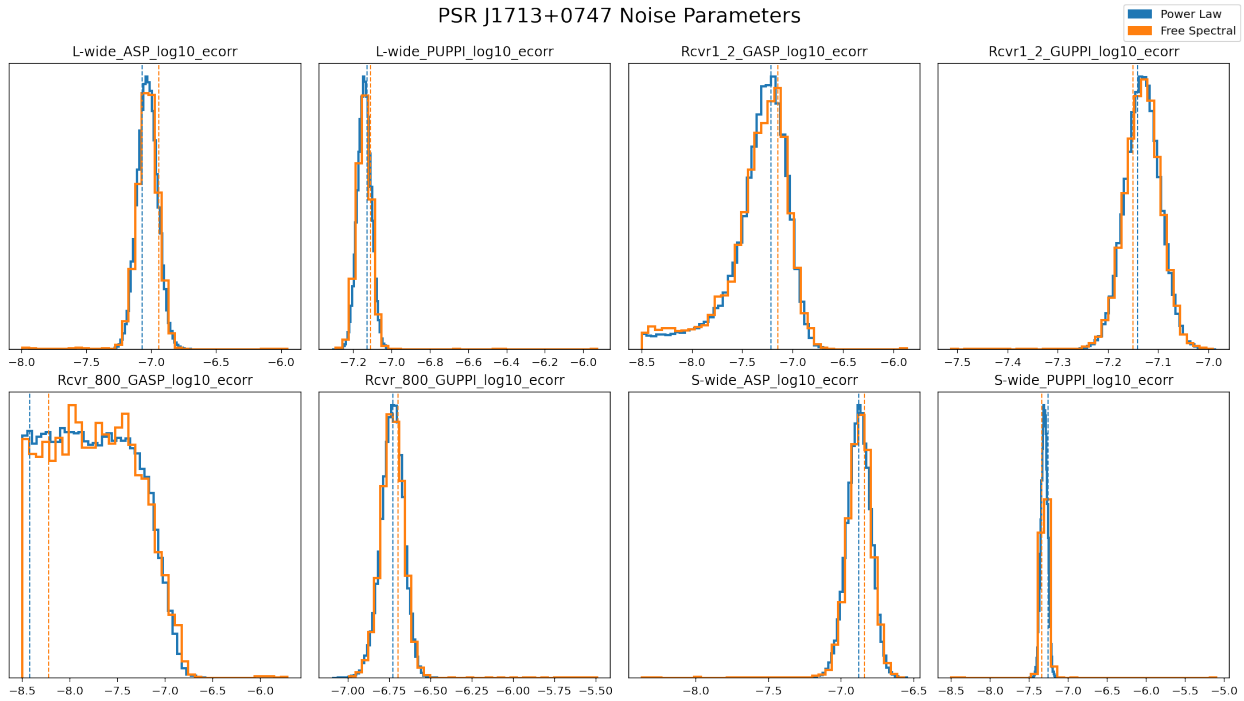

PSR J1713+0747 Noise Parameters
L-wide_ASP_efac
L-wide_ASP_log10_ecorr
L-wide_ASP_log10_equad
L-wide_PUPPI_efac
L-wide_PUPPI_log10_ecorr
L-wide_PUPPI_log10_equad
Rcvr1_2_GASP_efac
Rcvr1_2_GASP_log10_ecorr
Rcvr1_2_GASP_log10_equad
Rcvr1_2_GUPPI_efac
Rcvr1_2_GUPPI_log10_ecorr
Rcvr1_2_GUPPI_log10_equad
Rcvr_800_GASP_efac
Rcvr_800_GASP_log10_ecorr
Rcvr_800_GASP_log10_equad
Rcvr_800_GUPPI_efac
Rcvr_800_GUPPI_log10_ecorr
Rcvr_800_GUPPI_log10_equad
S-wide_ASP_efac
S-wide_ASP_log10_ecorr
S-wide_ASP_log10_equad
S-wide_PUPPI_efac
S-wide_PUPPI_log10_ecorr
S-wide_PUPPI_log10_equad
red_noise_gamma
red_noise_log10_A
lnpost
lnlike
chain_accept
pt_chain_accept

```
c1.set_burn(0)
dg.plot_chains(c1,
               hist=False,
               pars=c1.params[24:36],
               ncols=4)
```




```
ecorr_pars = [p for p in c0.params if 'ecorr' in p]
```

```
dg.plot_chains([c0,c1],
               plot_map=True,
               ncols=4,
               pars=ecorr_pars,
               title_y=1.05,
               legend_labels=['Power Law', 'Free Spectral'],
               linewidth=2)
```



1.1.4 Tutorial #3 Plotting Red Noise Spectra

```
import la_forge.core as co
import la_forge.rednoise as rn
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import numpy as np
```

```
coredir = '/Users/hazboun/software_development/la_forge/tests/data/cores/'
```

```
c0 = co.Core(corepath=coredir+'J1713+0747_plaw_dmx.core',
             label='NG12.5yr Noise Run: Power Law Red Noise')
```

```
Loading data from HDF5 file....
```

```
c1 = co.Core(corepath=coredir+'J1713+0747_fs_dmx.core',
             label='NG12.5yr Noise Run: Free Spectral Red Noise')
```

```
Loading data from HDF5 file....
```

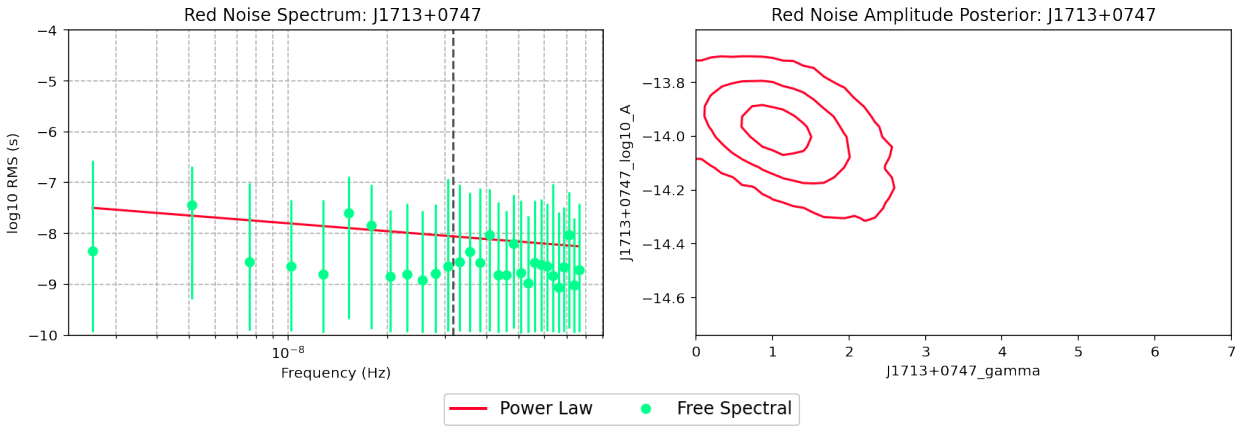
```
rn.plot_rednoise_spectrum('J1713+0747',
                          [c0,c1],
                          rn_types=['_red_noise','_red_noise'])
```

```
Plotting Powerlaw RN Params:Tspan = 12.4 yrs, 1/Tspan = 2.6e-09
Red noise parameters: log10_A = -13.97, gamma = 1.02
```

(continues on next page)

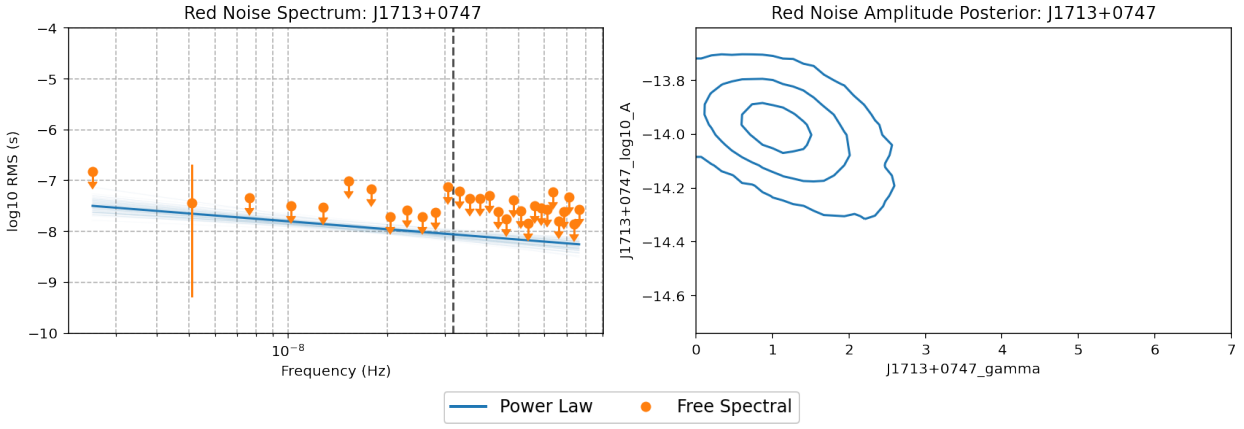
(continued from previous page)

```
Plotting Free Spectral RN Params:Tspan = 12.4 yrs    f_min = 2.6e-09
```



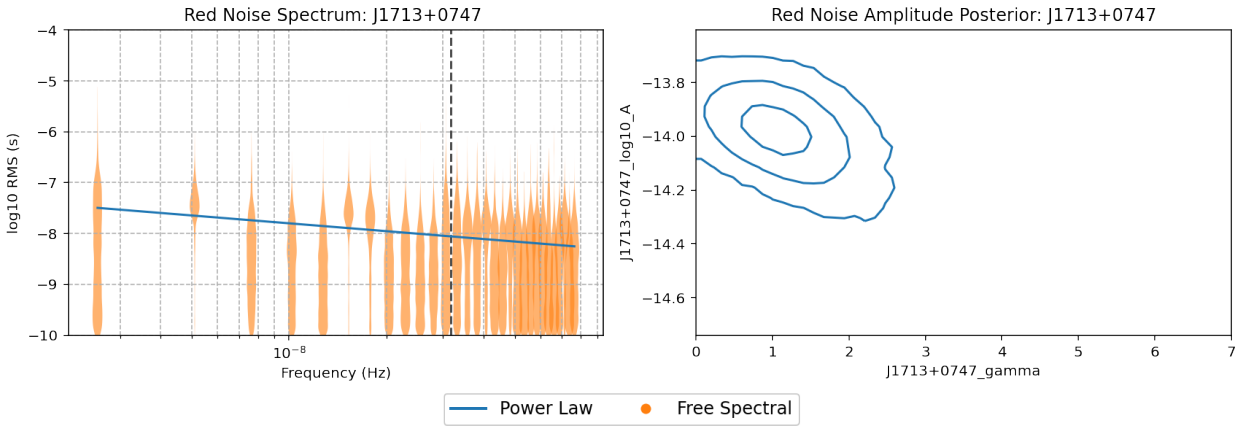

```
rn.plot_rednoise_spectrum('J1713+0747',
                          [c0,c1],
                          free_spec_ul=True,
                          rn_types=['_red_noise','_red_noise'],
                          Colors=['C0','C1'],
                          n_plaw_realizations=100)
```

```
Plotting Powerlaw RN Params:Tspan = 12.4 yrs, 1/Tspan = 2.6e-09
Red noise parameters: log10_A = -13.97, gamma = 1.02
Plotting Free Spectral RN Params:Tspan = 12.4 yrs    f_min = 2.6e-09
```




```
rn.plot_rednoise_spectrum('J1713+0747',
                          [c0,c1],
                          rn_types=['_red_noise','_red_noise'],
                          free_spec_violin=True,
                          Colors=['C0','C1'])
```

```
Plotting Powerlaw RN Params:Tspan = 12.4 yrs, 1/Tspan = 2.6e-09
Red noise parameters: log10_A = -13.97, gamma = 1.02
Plotting Free Spectral RN Params:Tspan = 12.4 yrs    f_min = 2.6e-09
```



1.1.5 Tutorial #4 HyperModel Cores

```
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'
%matplotlib inline
import numpy as np

import json, pickle, copy
```

```
import la_forge.diagnostics as dg
import la_forge.core as co
from la_forge.rednoise import plot_rednoise_spectrum, plot_free_spec
from la_forge.utils import epoch_ave_resid
```

Loading Chains

```
psrname = 'J1713+0747'
chaindir = '/Users/hazboun/software_development/la_forge/tests/data/chains/adv_noise_
↪J1713+0747/'
```

Us the `core.HyperModelCore` to load up the chains from an `enterprise_extensions.hypermodel.HyperModel` analysis. The code automatically looks for a dictionary containing the parameters in each model at `'./model_params.json'`, but you can also provide one with a keyword argument.

```
cH=co.HyperModelCore(label=f'PSR {psrname}, DM Model Selection',
                     chaindir=chaindir)
```

The `HyperModelCore` has most of the same atrributes in the base `Core` class that are useful for looking at the chains.

```
len(cH.param_dict.keys())
```

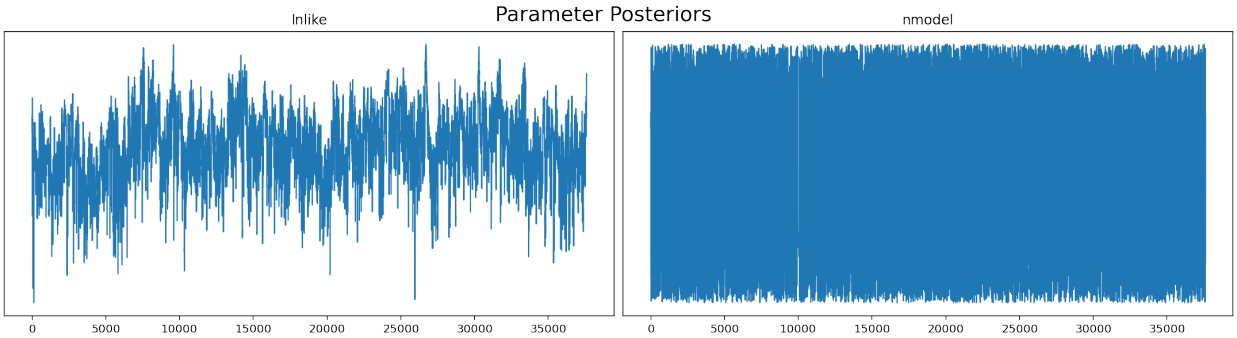
```
5
```

Here we return the model with the largest number of samples.

```
vals,bins=np.histogram(cH.get_param('nmodel'),
                       bins=[-0.5+ii for ii in range(len(cH.param_dict.keys()))])
np.argmax(vals)
```

```
0
```

```
cH.set_burn(3000)
dg.plot_chains(cH,hist=False,pars=['lnlike','nmodel'],ncols=2)
```




```
def odds_ratio(nmodel,models=[0,1]):
    top = np.logical_and(nmodel>models[1]-0.5,nmodel<models[1]+0.5)
    bottom = np.logical_and(nmodel>models[0]-0.5,nmodel<models[0]+0.5)
    return np.sum(top)/np.sum(bottom)
```

```
odds_ratio(cH('nmodel'),models=[0,2])
```

```
0.9796767230032117
```

Noise Flower Plots

One useful figure we look at for model selection anlayses is a radial histogram that plots the time spent in each of the possible models. We call these "noise flowers".

Running advanced noise analysis tools leads to a set of labels being saved for the models being analyzed in a given chain directory under `'model_labels.json'`. This is a list of labels that should help us keep track of the various models being analyzed. If these are not descriptive enough, feel free to edit the nested list.

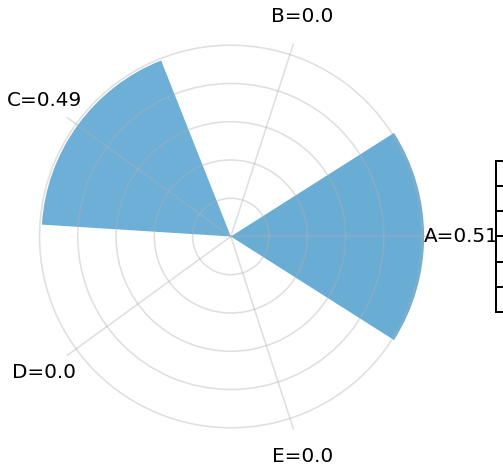
One can construct a list of your choosing as well.

```
with open(chaindir+'/model_labels.json' , 'r') as fin:
    model_labels= json.load(fin)

model_labels
```

```
[['A', 'periodic', True, True, True, 'periodic', 4],
 ['B', 'periodic', True, True, True, 'sq_exp', 4],
 ['C', 'periodic', True, True, True, 'periodic', 4.4],
 ['D', 'periodic', True, True, True, 'sq_exp', 4.4],
 ['E', 'periodic', True, True, False, None, None]]
```

There is a `noise_flower` function in `la_forge.diagnostics` that takes as input a `HyperModelCore`, along with various options, in order to fill out the table with useful information about the models being analyzed.

```
dg.noise_flower(cH,
                colLabels=['', 'DM GP','Dip 1','Dip 2','Chrom', 'Kernel','Idx'],
                cellText=model_labels,
                colWidths=[0.06,0.22,0.14,0.14,0.18,0.21,0.09])
```



PSR J1713+0747
Noise Model Selection

	DM GP	Dip 1	Dip 2	Chrom	Kernel	Idx
A	periodic	True	True	True	periodic	4
B	periodic	True	True	True	sq_exp	4
C	periodic	True	True	True	periodic	4.4
D	periodic	True	True	True	sq_exp	4.4
E	periodic	True	True	False		

Single model cores

A `core.HyperModelCore` object has a method to return **only** the samples from a particular model.

The individual cores are pulled out using the `model_core` method and an integer that gives which model you'd like returned.

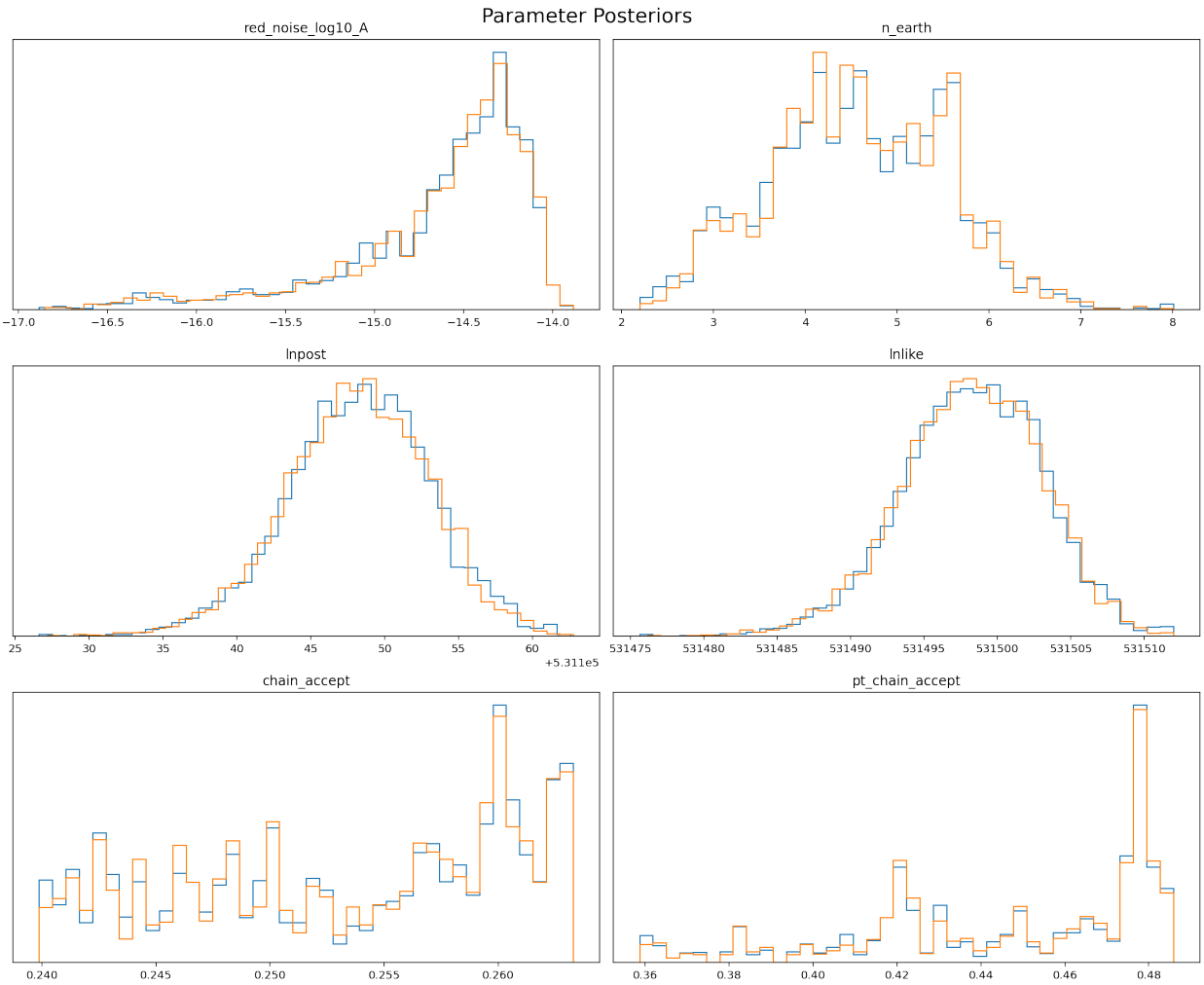
```
c0=cH.model_core(0)
c2=cH.model_core(2)
```

These cores are now individual core instances that can be saved as individual `HDF5` files.

```
type(c0)
```

```
la_forge.core.Core
```

```
dg.plot_chains([c0,c2],pars=c0.params[-6:],ncols=2)
```



1.1.6 Tutorial #5 Visualizing Gaussian Process Realizations

The purpose of thise notebook is to look at the output from the noise analysis and try to understand various aspects of what the model selection and data are telling us.

```
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'
%matplotlib inline
import numpy as np
import json, pickle, copy
```

```
import la_forge.diagnostics as dg
import la_forge.core as co
from la_forge.rednoise import plot_rednoise_spectrum, plot_free_spec
from la_forge.utils import epoch_ave_resid
```

Load Chains

You'll want to point to the chains for the pulsar you wish to investigate.

```
psrname = 'J1713+0747'
chaindir = '/Users/hazboun/software_development/la_forge/tests/data/chains/adv_noise_
↪J1713+0747/'
```

Here we load up a set of chains from an `enterprise.hypermodel.HyperModel` analysis.

```
cH=co.HyperModelCore(label='PSR {}, DM Model Selection',
                     chaindir=chaindir)
```

The `HyperModelCore.model_core` method returns only the samples, and paranmeters, for one particular model.

```
c0=cH.model_core(0)
```

1.1.7 Gaussian Process Realizations

Most of the advanced noise models that we are unleashing on ur various pulsars are various iterations of Gaussian processes. These are meant to model stochastic processes in data and while there are often functions that are used to describe them, they are inherently realization dependent. In order to get a feeling for how well the GPs are fitting the data we use `enterprise` to make realizations of the GPs.

We start by importing a few functions, retrieving the pulsar and making the PTA object.

```
from enterprise_extensions.models import model_singlepsr_noise
```

```
from la_forge.gp import Signal_Reconstruction as gp
```

```
filepath = '/Users/hazboun/nanograv_detection/12p5yr/noise_model_selection/'
filepath += '{0}_ng12p5yr_v3_nodmx_ePSR.pkl'.format(psrname)
```

```
with open(filepath,'rb') as fin:
    psr=pickle.load(fin)

with open(chaindir+'/model_kwargs.json' , 'r') as fin:
    model_kwargs=json.load(fin)
```

You shouldn't need this next cell, but there are some of the original model_kwarg dictionaries with spurious entries that need to be deleted. If you get an error when calling `model_singlepsr_noise` and it matches one of the these kwargs, try running this cell to delete them.

```
pta = model_singlepsr_noise(psr, **model_kwargs['0'])
```

```
len(pta.param_names), len(c0.params[:-4])
```

```
(41, 41)
```

The `Signal_Reconstruction` class take an `enterprise.pulsar.Pulsar` object, an `enterprise.signal_base.PTA` object and a `la_forge.core.Core` object as inputs. (One can alternatively use a chain array and burn value for the latter.)

```
sr=gp(psr, pta, core=c0)
```

The gp_types atribute will tell you which Gaussian process signals are available in this PTA. Additionally there are a number of other options one can use for the gp_type flag that goes into a signal reconstruction. These include ['achromatic_rn','DM','FD','all']. Also any of the timing parameter perturbations can also be called.

```
sr.gp_types
```

```
['linear_timing_model', 'red_noise', 'dm_gp', 'chrom_gp']
```

Every pulsar has a list of the timing parameters that are fit with the linearized timing model. These are also modeled as Gaussian process and can be retrieved with the same functions.

```
psr.fitpars
```

```
['Offset',
 'ELONG',
 'ELAT',
 'F0',
 'F1',
 'DM',
 'DM1',
 'DM2',
 'PMELONG',
 'PMELAT',
 'PX',
 'PB',
 'T0',
 'A1',
 'OM',
 'ECC',
 'M2',
 'FD1',
 'FD2',
 'FD3',
 'FD4',
 'FD5',
 'KOM',
 'KIN',
 'JUMP1',
 'JUMP2',
 'JUMP3']
```

```
# parameter indices to pull from chain. Change `size` flag for more or less.
# first one picks the "most likely values"
idxs = np.argsort(c0.get_param('lnpost',to_burn=False))[::-1][:20]

# this one just picks random values. Should be broadly the same as above if well␣
↪converged
# idxs = np.random.randint(sr.burn, sr.chain.shape[0],size=100)
```

The `reconstruct_signal` method

There are few useful options for regaining GPs using this method. Here is the docstring:

```
Parameters
----------
gp_type : str, {'achrom_rn','gw','DM','none','all',timing parameters}
    Type of gaussian process signal to be reconstructed. In addition
    any GP in `psr.fitpars` or `Signal_Reconstruction.gp_types` may be
    called.
    ['achrom_rn','red_noise'] : Return the achromatic red noise.
    ['DM'] : Return the timing-model parts of dispersion model.
    [timing parameters] : Any of the timing parameters from the linear
        timing model. A list is available as `psr.fitpars`.
    ['timing'] : Return the entire timing model.
    ['gw'] : Gravitational wave signal. Works with common process in
        full PTAs.
    ['none'] : Returns no Gaussian processes. Meant to be used for
        returning deterministic signal.
    ['all'] : Returns all Gaussian processes.

det_signal : bool
    Whether to include the deterministic signals in the reconstruction.

mlv : bool
    Whether to use the maximum likelihood value for the reconstruction.

idx : int, optional
    Index of the chain array to use.
```

In particular you can choose which GP signals to return. To return a single gp choose from the list `sr.gp_types`. If you want the dispersion measure elements of the DM model you can use `'DM'`. This returns either the DMX GP or the DM1 and DM2 GPs.

To get all of the deterministic signals, but no GP use `reconstruct_signal(gp_type='none',det_signal=True, ...)`.

The next cell gives the timing model components of the dispersion model + any deterministic models (DM, CW, ...). The only determinsitic model this pulsar has is the solar wind, but if you add DM dips or something this is the flag to use. You get all of them at the same time. One can construct these signals separately but it would take different code.

```
DM = np.array([sr.reconstruct_signal(gp_type='DM',det_signal=True, idx=idx)[psrname]
               for idx in idxs])
```

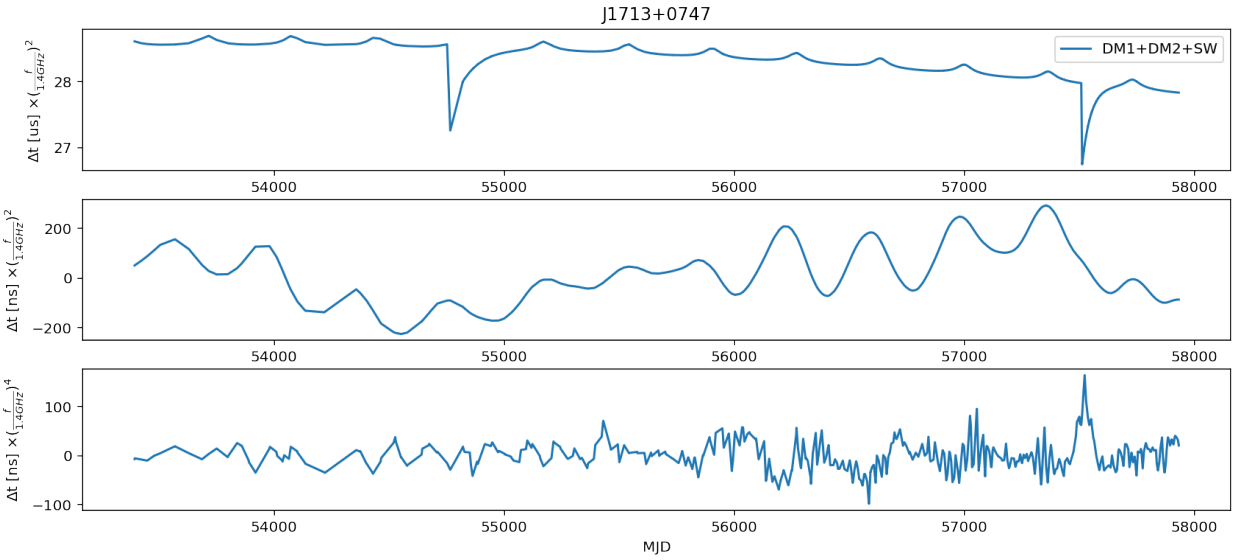
The next three cells return realizations of the DM GP and the Chromatic GP.

```
dm_gp = np.array([sr.reconstruct_signal(gp_type='dm_gp', idx=idx)[psrname]
                  for idx in idxs])
```

```
chrom_gp = np.array([sr.reconstruct_signal(gp_type='chrom_gp', idx=idx)[psrname]
                     for idx in idxs])
```

```
mn_DM = np.mean(DM,axis=0)
mn_dmgp = np.mean(dm_gp,axis=0)
mn_chrom_gp = np.mean(chrom_gp,axis=0)
```

```
fig = plt.figure(figsize=[14,6])

####### First Plot ########
fig.add_subplot(311)
plt.plot(psr.toas/(24*3600),mn_DM*(psr.freqs/1400)**2*1e6,label='DM1+DM2+SW')
plt.legend()
plt.ylabel(r'$\Delta$t [us] $\times (\frac{f}{1.4 GHz})^2$')
plt.title(psrname)

####### Second Plot ########
fig.add_subplot(312)
plt.plot(psr.toas/(24*3600),mn_dmgp*(psr.freqs/1400)**2*1e9,label='DM GP 1')
plt.ylabel(r'$\Delta$t [ns] $\times (\frac{f}{1.4 GHz})^2$')

####### Third Plot ########
fig.add_subplot(313)
plt.plot(psr.toas/(24*3600),mn_chrom_gp*(psr.freqs/1400)**4*1e9,label='Chrom GP')
plt.ylabel(r'$\Delta$t [ns] $\times (\frac{f}{1.4 GHz})^4$')
plt.xlabel('MJD')
plt.show()
```



Plot DMX

Use the DMX values from the data release as a comparison of how well the DM GP models are matching the changes in the dispersion measure.

```
#Load DMX values
dtypes = {'names': ('DMXEP', 'DMX_value', 'DMX_var_err',
                    'DMXR1', 'DMXR2', 'DMXF1',
                    'DMXF2', 'DMX_bin'),
          'formats': ('f4','f4','f4','f4','f4','f4','f4','U6')}
dmx = np.loadtxt('/Users/hazboun/nanograv_detection/12p5yr/noise_model_selection/dmx/{0}_
↪NANOGrav_12yv3.dmx'.format(psrname),
```

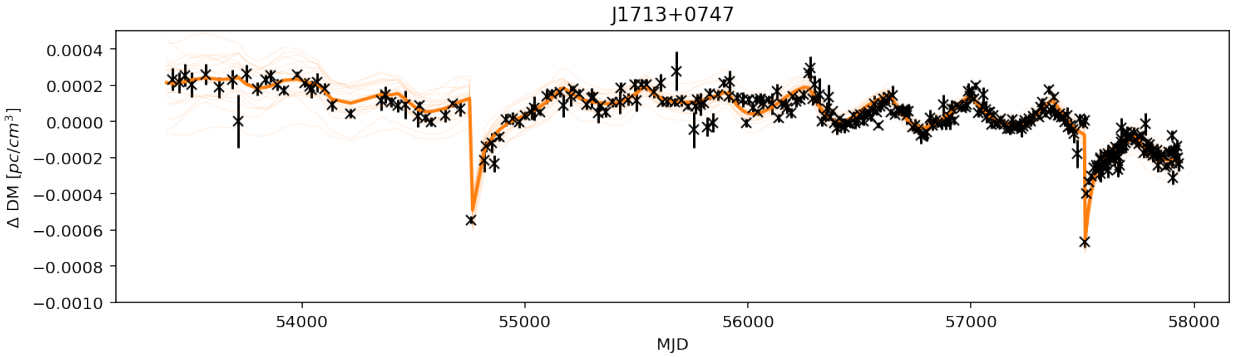
(continues on next page)

(continued from previous page)

```
                skiprows=4,
                dtype=dtypes)
```

```
# Convert signals into units of DM [pc/cm^3]
dm_units = (dm_gp  + DM)*psr.freqs[np.newaxis,:]**2*2.41e-4
dm_mean = (mn_DM + mn_dmgp )*psr.freqs**2*2.41e-4
```

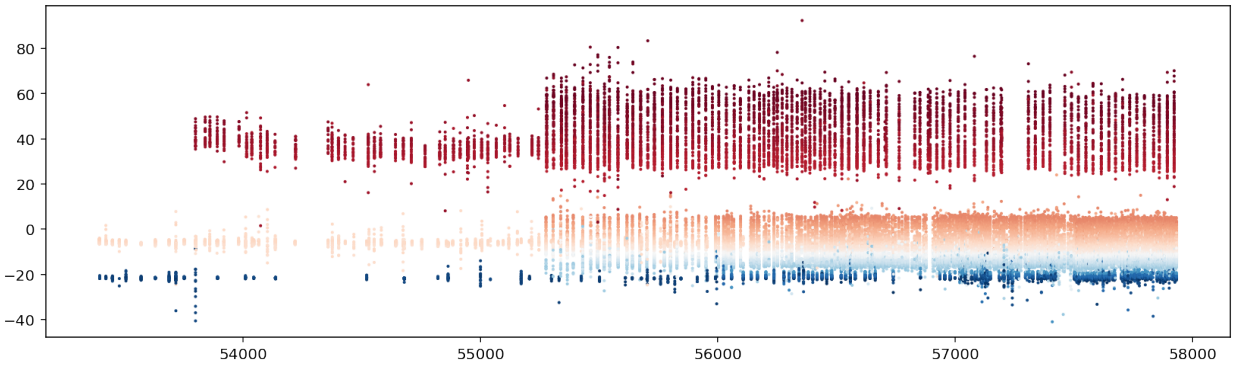
```
plt.figure(figsize=[12,3])
for dm in dm_units:
    plt.plot(psr.toas/(24*3600),dm-dm.mean(),linewidth=0.2,alpha=0.3,color='C1')

plt.plot(psr.toas/(24*3600),dm_mean-dm_mean.mean(),linewidth=2,color='C1')
plt.errorbar(x=dmx['DMXEP'],
             y=dmx['DMX_value']-dmx['DMX_value'].mean(),
             yerr=dmx['DMX_var_err'],
             marker='x',color='k',linestyle='none')


plt.ylim(-0.0010,0.0005)
plt.xlabel('MJD')
plt.ylabel(r'$\Delta$ DM [$pc/cm^3$]')
plt.title(psrname)
plt.show()
```



Looking at Residuals

Uncorrected residuals will look really bad since we have stripped all of the DMX parameters and DM1/DM2 are set to zero initially.

```
sec_to_day = 24*3600
plt.figure(figsize=[14,4])
plt.scatter(x=psr.toas/sec_to_day,
            y=(psr.residuals)*1e6,
            s=1,
            c=psr.freqs,cmap='RdBu')
# plt.xlim(57000,58000)
plt.show()
```



The GPs are natively returned as delays in seconds so we can subtract them from the residuals to see what residuals `Enterprise` is actually calculating during the analysis. The following function calculates the epoch averaged TOAs after subtracting the given `correction` GPs.

```
resids,msks=epoch_ave_resid(psr, correction= mn_DM + mn_dmgp + mn_chrom_gp) #This is all␣
↪the chromatic GPs, DM1/DM2 + DMGP + ChromGP
```

This shows the two masks created for the different receivers. This allows us to plot by frequency.

```
masks = list(msks.keys())
masks
```

```
['Rcvr_800', 'Rcvr1_2', 'L-wide', 'S-wide']
```

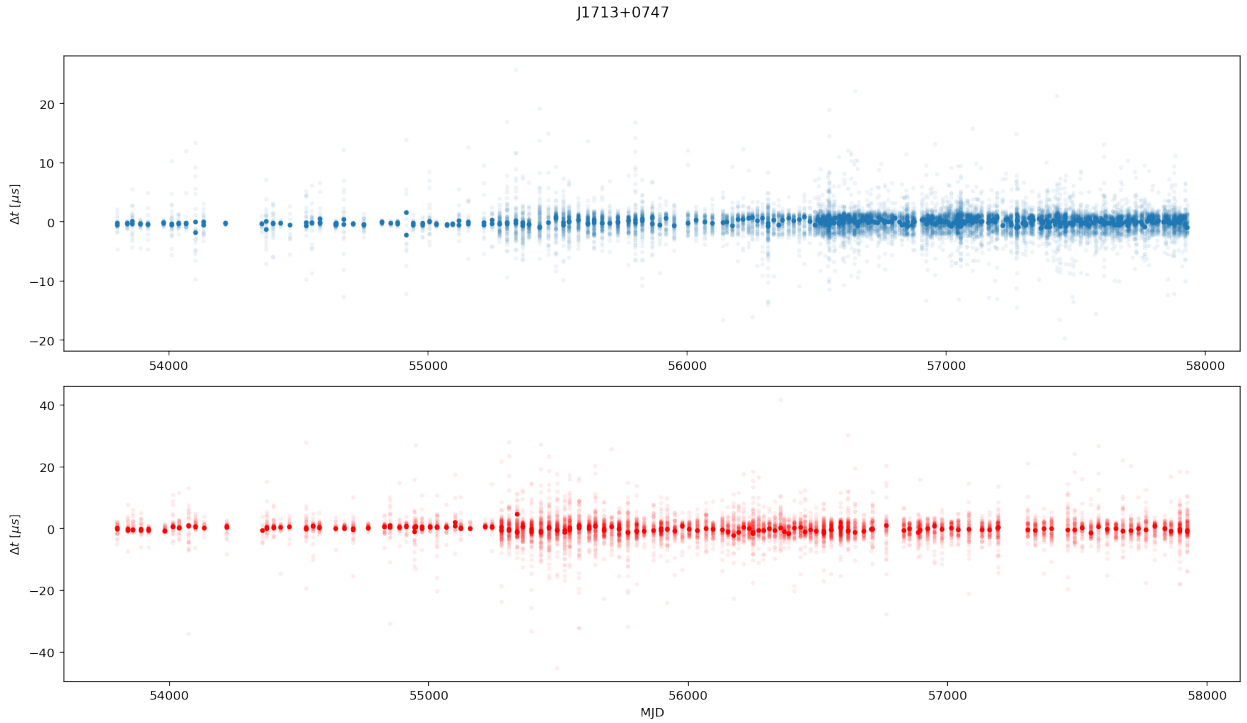
```
all_chrgp = mn_DM + mn_dmgp + mn_chrom_gp
```

```
sec_to_day = 24*3600
fig=plt.figure(figsize=[14,8])

#--------- 1st Plot -------------#
fig.add_subplot(211)
high_rec = 'Rcvr1_2'

plt.scatter(x=resids[high_rec][:,0]/sec_to_day,
            y=resids[high_rec][:,1]*1e6-np.mean(resids[high_rec][:,1]*1e6),
            s=8,c='C0')
plt.scatter(x=psr.toas[msks[high_rec]]/sec_to_day,
            y=((psr.residuals-all_chrgp)[msks[high_rec]]-(psr.residuals-all_
↪chrgp)[msks[high_rec]].mean())*1e6,
            s=6,
            c='C0',alpha=0.05)
plt.ylabel(r'$\Delta t$ [$\mu s$]')

#--------- 2nd Plot -------------#
fig.add_subplot(212)
low_rec = 'Rcvr_800'
plt.scatter(x=resids[low_rec][:,0]/sec_to_day,
            y=resids[low_rec][:,1]*1e6-np.mean(resids[low_rec][:,1]*1e6),
            s=8, c='red')
plt.scatter(x=psr.toas[msks[low_rec]]/sec_to_day,
```

(continues on next page)

(continued from previous page)

```
            y=((psr.residuals-all_chrgp)[msks[low_rec]]-(psr.residuals-all_
↪chrgp)[msks[low_rec]].mean())*1e6,
            s=6,
            c='red',alpha=0.05)#psr.freqs,cmap='RdBu')

plt.ylabel(r'$\Delta t$ [$\mu s$]')
plt.xlabel('MJD')
fig.suptitle(psrname,y=1.01)
fig.tight_layout()
plt.show()
```



Here we just plot the residuals along with the scattering GP to see if there is interesting that is missed by the current modeling.

```
everything = np.array([sr.reconstruct_signal(gp_type='all',det_signal=True,␣
↪idx=idx)[psrname] for idx in idxs])
```

```
mn_all = np.mean(everything, axis=0)
```

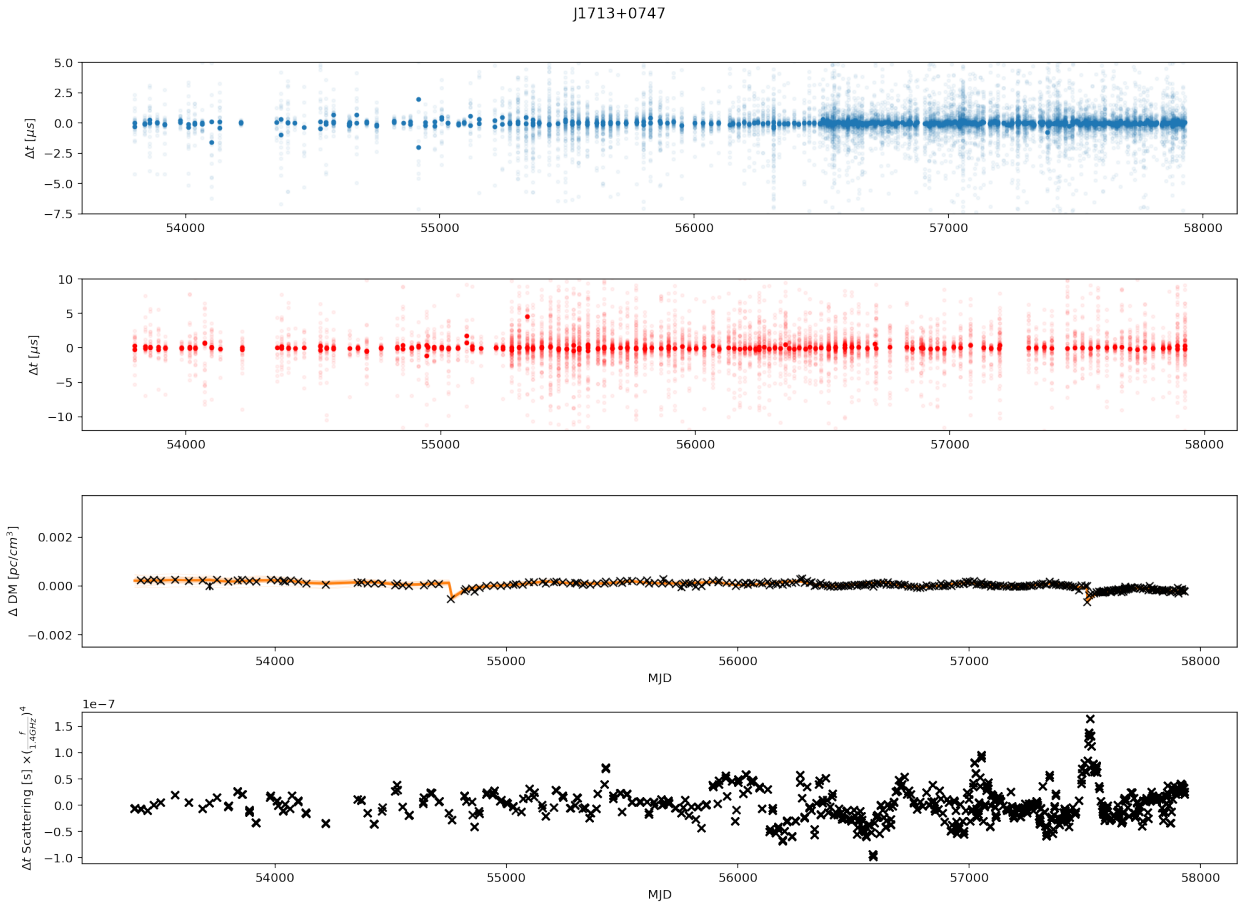
```
resids2,msks=epoch_ave_resid(psr, correction= mn_all) #This is all te chromatic GPs, DM1/
↪DM2 + DMGP + ChromGP
```

```
sec_to_day = 24*3600
fig=plt.figure(figsize=[14,10])
fig.add_subplot(411)
plt.scatter(x=resids2['Rcvr1_2'][:,0]/sec_to_day,
            y=resids2['Rcvr1_2'][:,1]*1e6,
            s=8,c='C0')
```

(continues on next page)

(continued from previous page)

```
plt.scatter(x=psr.toas[psr.flags['fe']=='Rcvr1_2']/sec_to_day,
            y=(psr.residuals-mn_all)[psr.flags['fe']=='Rcvr1_2']*1e6,
            s=6,
            c='C0',alpha=0.05)
plt.ylim(-7.5,5)
plt.ylabel(r'$\Delta t$ [$\mu s$]')

fig.add_subplot(412)

plt.scatter(x=resids2['Rcvr_800'][:,0]/sec_to_day,
            y=resids2['Rcvr_800'][:,1]*1e6,
            s=8,c='red')
plt.scatter(x=psr.toas[psr.flags['fe']=='Rcvr_800']/sec_to_day,
            y=(psr.residuals-mn_all)[psr.flags['fe']=='Rcvr_800']*1e6,
            s=6,
            c='red',alpha=0.05)
plt.ylim(-12,10)

plt.ylabel(r'$\Delta t$ [$\mu s$]')

fig.add_subplot(413)

for dm in dm_units:
    plt.plot(psr.toas/(24*3600),dm-dm.mean(),linewidth=0.2,alpha=0.3,color='C1')

plt.plot(psr.toas/(24*3600),dm_mean-dm_mean.mean(),linewidth=2,color='C1')
plt.errorbar(x=dmx['DMXEP'],
             y=dmx['DMX_value']-dmx['DMX_value'].mean(),
             yerr=dmx['DMX_var_err'],
             marker='x',color='k',linestyle='none')

plt.ylim(-0.0025,0.0037)
plt.xlabel('MJD')
plt.ylabel(r'$\Delta$ DM [$pc/cm^3$]')

fig.add_subplot(414)
plt.plot(psr.toas/sec_to_day, mn_chrom_gp*(psr.freqs/1400)**4,'x',color='k')
plt.ylabel(r'$\Delta t$ Scattering [s] $\times (\frac{f}{1.4GHz})^4$')
plt.xlabel('MJD')

fig.suptitle(psrname,y=1.01)
fig.tight_layout(pad=1.01)
plt.show()
```



1.1.8 la_forge

la_forge package

Submodules

la_forge.core module

class la_forge.core.**Core**(*chaindir=None*, *corepath=None*, *burn=0.25*, *label=None*, *fancy_par_names=None*, *chain=None*, *params=None*, *pt_chains=False*, *skiprows=0*)

Bases: `object`

An object that stores the parameters and chains from a bayesian analysis. Currently configured specifically for posteriors produced by *PTMCMCSampler*.

Parameters

chaindir
[str] Directory with chains and file with parameter names. Currently supports chains as {'chain_1.txt','chain.fit'} and parameters as {'pars.txt','params.txt','pars.npy'} . If chains are stored in a FITS file it is assumed that the parameters are listed as the column names.

label
[str] Name of the core.

burn
[int, optional] Number of samples burned from beginning of chain. Used when calculating statistics and plotting histograms.

fancy_par_names
[list of str] List of strings provided as names to be used when plotting parameters. Must be the same length as the parameter list associated with the chains.

chain
[array, optional] Array that contains samples from an MCMC chain that is samples x param in shape. Loaded from file if dir given as *chaindir*.

params
[list, optional] List of parameters that corresponds to the parameters in the chain. Is loaded automatically if in the chain directory given above.

corepath
[str] Path to an already saved core. Assumed to be an *hdf5* file made with *la_forge*.

pt_chains
[bool] Whether to load all higher temperature chains from a parallel tempering (PT) analysis.

skiprows
[int] Number of rows to skip while loading a chain text file. This effectively acts as a burn in, which can not be changed once the file is loaded (unless loade again). Useful when dealing with large chains and loading multiple times.

`credint`(*param*, *onesided=False*, *interval=68*)

Returns credible interval of parameter given.

Parameters

param
[str, list of str]

onesided
[bool, optional] Whether to calculate a one sided or two sided credible interval. The onesided option gives an upper limit.

interval: float, optional
Width of interval in percent. Default set to 68%.

`get_map_dict`()

Return a dictionary of the max a postori values for the parameters in the core. The keys are the appropriate parameter names.

`get_map_param`(*param*)

Returns maximum a posteri value of samples for the parameter given.

`get_param`(*param*, *to_burn=True*)

Returns array of samples for the parameter given.

param can either be a single list or list of strings.

`get_param_credint`(*param*, *onesided=False*, *interval=68*)

Returns credible interval of parameter given.

Parameters

param
[str, list of str]

onesided
[bool, optional] Whether to calculate a one sided or two sided credible interval. The onesided option gives an upper limit.

interval: float, optional
Width of interval in percent. Default set to 68%.

`get_param_median`(*param*)

Returns median of parameter given.

`property map_idx`

Maximum a posteri parameter values. From burned chain.

`property map_params`

Return all Maximum a posteri parameters.

`median`(*param*)

Returns median of parameter provided. Can be given as a string or list of strings.

`save`(*filepath*)

Save Core object as HDF5.

`set_burn`(*burn*)

Set number of samples to burn.

Parameters

burn
[int, float] An integer designating the number of samples to remove from beginning of chain. A float between 0 and 1 can also be used, which will estimate the integer value as a fraction of the full array length.

`set_fancy_par_names`(*names_list*)

Set fancy_par_names.

`set_rn_freqs`(*freqs=None*, *Tspan=None*, *nfreqs=30*, *log=False*, *partimdir=None*, *psr=None*, *freq_path='fourier_components.txt'*)

Set gaussian process red noise frequency array.

Parameters

freqs
[list or array of floats] List or array of frequencies used for red noise gaussian process.

Tspan
[float, optional] Timespan of the data set. Used for calculating frequencies. Linear array is calculated as *[1/Tspan, … ,nfreqs/Tspan]*.

nfreqs
[int, optional] Number of frequencies used for red noise gaussian process.

log
[bool, optional] Whether to use a log-linear space when calculating the frequency array.

partimdir
[str, optional] Directory with pulsar data (assumed the same for *tim* and *par* files.) Calls the *utils.get_Tspan()* method which loads an *enterprise.Pulsar(psr,partimdir)* and extracts the timespan.

psr
[str, optional] Puslar name, used when get the time span by loading *enterprise.Pulsar()* as in the documentation of *partimdir* above. It is assumed that there is only one par and tim file in the directory with this pulsar name in the file name.

freq_path
[str, optional] Path to a txt file containing the rednoise frequencies to be used.

Returns

Array of red noise frequencies.

class la_forge.core.**HyperModelCore**(*label=None*, *param_dict=None*, *chaindir=None*, *burn=0.25*, *corepath=None*, *fancy_par_names=None*, *chain=None*, *params=None*, *pt_chains=False*, *skiprows=0*)

Bases: *Core*

A class to make cores for the chains made by the enterprise_extensions HyperModel framework.

model_core(*nmodel*)

Return a core that only contains the parameters and samples from a single HyperModel model.

class la_forge.core.**TimingCore**(*chaindir=None*, *burn=0.25*, *label=None*, *fancy_par_names=None*, *chain=None*, *params=None*, *pt_chains=False*, *tm_pars_path=None*)

Bases: *Core*

A class for cores that use the enterprise_extensions timing framework. The Cores for timing objects need special attention because they are sampled in a standard format, rather than using the real parameter ranges. These Cores allow for automatic handling of the parameters.

get_param(*param*, *to_burn=True*, *tm_convert=True*)

Returns array of samples for the parameter given. Will convert timing parameters to physical units based on *TimingCore.tm_pars_orig* entries. Will also accept shortened timing model parameter names, like *PX*.

param can either be a single list or list of strings.

mass_function(*PB*, *A1*)

Computes Keplerian mass function, given projected size and orbital period.

Parameters

PB
[float] Orbital period [days]

A1
[float] Projected semimajor axis [lt-s]

Returns

mass function
Mass function [solar mass]

mass_pulsar()

Computes the companion mass from the Keplerian mass function. This function uses a Newton-Raphson method since the equation is transcendental.

la_forge.core.**load_Core**(*filepath*)

la_forge.diagnostics module

la_forge.gp module

class la_forge.gp.**Signal_Reconstruction**(*psrs*, *pta*, *chain=None*, *burn=None*, *p_list='all'*, *core=None*)

Bases: object

Class for building Gaussian process realizations from enterprise models.

reconstruct_signal(*gp_type='achrom_rn'*, *det_signal=False*, *mlv=False*, *idx=None*, *condition=False*, *eps=1e-16*)

Parameters

gp_type
[str, {'achrom_rn','gw','DM','none','all',timing parameters}] Type of gaussian process signal to be reconstructed. In addition any GP in *psr.fitpars* or *Signal_Reconstruction.gp_types* may be called.

['achrom_rn','red_noise'] : Return the achromatic red noise.

['DM'] : Return the timing-model parts of dispersion model.

[timing parameters] : Any of the timing parameters from the linear timing model. A list is available as *psr.fitpars*.

['timing'] : Return the entire timing model.

['gw'] : Gravitational wave signal. Works with common process in full PTAs.

['none'] : Returns no Gaussian processes. Meant to be used for returning only a deterministic signal.

['all'] : Returns all Gaussian processes.

det_signal
[bool] Whether to include the deterministic signals in the reconstruction.

mlv
[bool] Whether to use the maximum likelihood value for the reconstruction.

idx
[int, optional] Index of the chain array to use.

Returns

wave
[array] A reconstruction of a single gaussian process signal realization.

sample_params(*index*)

sample_posterior(*samp_idx*, *array_params=['alphas', 'rho', 'nE']*)

la_forge.rednoise module

la_forge.rednoise.**determine_if_limit**(*vals*, *threshold=0.1*, *minval=-10*, *lower_q=0.3*)

Function to determine if an array or list of values is sufficiently
separate from the minimum value.

Parameters

vals
[array or list]

threshold: float
Threshold above *minval* for determining whether to count as twosided interval.

minval: float
Minimum possible value for posterior.

lower_q: float
Percentile value to evaluate lower bound.

la_forge.rednoise.**get_Tspan**(*pulsar*, *filepath=None*, *fourier_components=None*, *datadir=None*)

Function for getting timespan of a set of pulsar dataself.

Parameters

pulsar
[str]

filepath
[str] Filepath to a *txt* file with pulsar name and timespan in two columns. If supplied this file is used to return the timespan.

fourier_components
[list or array] Frequencies used in gaussian process modeling. If given *1/numpy.amin(fourier_components)* is retruned as timespan.

datadir
[str] Directory with pulsar data (assumed the same for *tim* and *par* files.) Calls the *utils.get_Tspan()* method which loads an *enterprise.Pulsar()* and extracts the timespan.

la_forge.rednoise.**get_rn_freqs**(*core*)

Get red noise frequency array from a core, with error message if noise array has not been included.

la_forge.rednoise.**plot_free_spec**(*core*, *axis*, *parname_root*, *prior_min=None*, *ci=95*, *violin=False*, *Color='k'*, *Fillstyle='full'*, *plot_ul=False*, *verbose=True*, *Tspan=None*)

Plots red noise free spectral parameters in units of residual time. Determines whether the posteriors should be considered as a fit a parameter or as upper limits of the given parameter and plots accordingly.

Parameters

core
[list] *la_forge.core.Core()* object which contains the posteriors for the relevant red noise parameters to be plotted.

axis
[matplotlib.pyplot.Axis] Matplotlib.pyplot axis object to append various red noise parameter plots.

parname_root
[str] Name of red noise free spectral coefficient parameters.

prior_min
[float, None, 'bayes'] Minimum value for uniform or log-uniform prior used in search over free spectral coefficients. If 'bayes' is used then a gorilla_bf calculation is done to determine if confidence interval should be plotted.

verbose
[bool, optional]

Color
[str, optional] Color of the free spectral coefficient markers.

Fillstyle
[str, optional] Fillstyle for the free spectral coefficient markers.

Tspan
[float, optional] Timespan of the data set. Used for converting amplitudes to residual time. Calculated from lowest red noise frequency if not provided.

la_forge.rednoise.**plot_powerlaw**(*core*, *axis*, *amp_par*, *gam_par*, *verbose=True*, *Color='k'*, *Linestyle='-'*, *n_realizations=0*, *Tspan=None*, *to_resid=True*)

Plots a power law line from the given parmeters in units of residual time.

Parameters

core
[list] *la_forge.core.Core()* object which contains the posteriors for the relevant red noise parameters to be plotted.

axis
[matplotlib.pyplot.Axis] Matplotlib.pyplot axis object to append various red noise parameter plots.

amp_par
[str] Name of red noise powerlaw amplitude parameter.

gam_par
[str] Name of red noise powerlaw spectral index parameter (gamma).

verbose
[bool, optional]

n_realizations
[int, optional] Number of realizations to plot.

Color
[list, optional] Color to make the plot.

Tspan
[float, optional] Timespan of the data set. Used for converting amplitudes to residual time. Calculated from lowest red noise frequency if not provided.

la_forge.rednoise.**plot_rednoise_spectrum**(*pulsar*, *cores*, *show_figure=True*, *rn_types=None*, *plot_2d_hist=True*, *verbose=True*, *Tspan=None*, *title_suffix=''*, *freq_yr=1*, *plotpath=None*, *cmap='gist_rainbow'*, *n_plaw_realizations=0*, *n_tproc_realizations=1000*, *n_bplaw_realizations=100*, *Colors=None*, *bins=30*, *labels=None*, *legend=True*, *legend_loc=None*, *leg_alpha=1.0*, *Bbox_anchor=(0.5, -0.25, 1.0, 0.2)*, *freq_xtra=None*, *free_spec_min=None*, *free_spec_ci=95*, *free_spec_violin=False*, *free_spec_ul=False*, *ncol=None*, *plot_density=None*, *plot_contours=None*, *add_2d_scatter=None*, *bplaw_kwargs={}*, *return_plot=False*, *excess_noise=False*, *levels=(0.39346934, 0.86466472, 0.988891)*)

Function to plot various red noise parameters in the same figure.

Parameters

pulsar
[str]

cores
[list] List of *la_forge.core.Core()* objects which contain the posteriors for the relevant red noise parameters to be plotted.

Tspan
[float, optional] Timespan of the data set. Used for converting amplitudes to residual time. Calculated from lowest red noise frequency if not provided.

show_figure
[bool]

rn_types
[list {'','_dm_gp','_chrom_gp','_red_noise'}] List of strings to choose which type of red noise parameters are used in each of the plots.

plot_2d_hist
[bool, optional] Whether to include two dimensional histogram of powerlaw red noise parameters.

verbose
[bool, optional]

title_suffix
[str, optional] Added to title of red noise plot as: 'Red Noise Spectrum: ' + pulsar + ' ' + title_suffix

freq_yr
[int , optional] Number of 1/year harmonics to include in plot.

plotpath
[str, optional] Path and file name to which plot will be saved.

cmap
[str, optional] Color map from which to cycle plot colrs, if not given in Colors kwarg.

n_plaw_realizations
[int, optional] Number of powerlaw realizations to plot.

n_tproc_realizations
[int, optional] Number of T-process realizations to plot.

Colors
[list, optional] List of colors to cycle through in plots.

labels
[list, optional] Labels of various plots, for legend.

legend_loc
[tuple or str, optional] Legend location with respect to Bbox_anchor.

leg_alpha
[float, optional] Opacity of legend background.

Bbox_anchor
[tuple, optional] This is the bbox_to_anchor value for the legend.

`la_forge.rednoise.plot_tprocess(core, axis, alpha_parname_root, amp_par, gam_par, Color='k', n_realizations=100, Tspan=None)`

Plots a power law line from the given parmeters in units of residual time.

Parameters

core
[list] *la_forge.core.Core()* object which contains the posteriors for the relevant red noise parameters to be plotted.

axis
[matplotlib.pyplot.Axis] Matplotlib.pyplot axis object to append various red noise parameter plots.

alpha_parname_root
[str] Root of the t-process coefficient names, i.e. for J1713+0747_red_noise_alphas_0 give: 'J1713+0747_red_noise_alphas'.

amp_par
[str] Name of red noise powerlaw amplitude parameter.

gam_par
[str] Name of red noise powerlaw spectral index parameter (gamma).

n_realizations
[int, optional] Number of realizations to plot.

Color
[list, optional] Color to make the plot.

Tspan
[float, optional] Timespan of the data set. Used for converting amplitudes to residual time. Calculated from lowest red noise frequency if not provided.

la_forge.slices module

`class la_forge.slices.SlicesCore(label=None, slicedirs=None, pars2pull=None, params=None, corepath=None, fancy_par_names=None, verbose=True, burn=0.25, parfile='pars.txt')`

Bases: *Core*

A class to make a la_forge.core.Core object that contains a subset of parameters from different chains. Currently this supports a list of strings for multiple columns of a given txt file or a single string.

Parameters

la_forge.slices.**get_col**(*col*, *filename*)

la_forge.slices.**get_idx**(*par*, *filename*)

la_forge.slices.**store_chains**(*filepaths*, *idxs*, *verbose=True*)

la_forge.utils module

la_forge.utils.**bayes_fac**(*samples*, *ntol=200*, *logAmin=-18*, *logAmax=-12*, *nsamples=100*, *smallest_dA=0.01*, *largest_dA=0.1*)

Computes the Savage Dickey Bayes Factor and uncertainty. Based on code in enterprise_extensions. Expanded to include more options for when there are very few samples at lower amplitudes.

Parameters

- **samples** – MCMC samples of GWB (or common red noise) amplitude
- **ntol** – Tolerance on number of samples in bin
- **logAmin** – Minimum log amplitude being considered.
- **logAmax** – Maximum log amplitude being considered.

Returns

(bayes factor, 1-sigma bayes factor uncertainty)

la_forge.utils.**compute_rho**(*log10_A*, *gamma*, *f*, *T*)

Converts from power to residual RMS.

la_forge.utils.**epoch_ave_resid**(*psr*, *correction=None*, *dt=10*)

Epoch averaged residuals organized by receiver.

Parameters

psr

[*enterprise.pulsar.Pulsar*]

correction

[array, optional] Numpy array which gives a correction to the residuals. Used for adding various Gaussian process realizations or timing model perturbations.

dt

[float] Coarse graining time [sec]. Sets filter size for TOAs.

Returns

fe_resids

[dict of arrays] Dictionary where each entry is an array of epoch averaged TOAS, residuals and TOA errors. Keys are the various receivers.

fe_mask

[dict of arrays] Dictionary where each entry is an array that asks as a mask for the receiver used as a key.

la_forge.utils.**figsize**(*scale*)

la_forge.utils.**getMax2d**(*samples1*, *samples2*, *weights=None*, *smooth=True*, *bins=[40, 40]*, *x_range=None*, *y_range=None*, *logx=False*, *logy=False*, *logz=False*)

Function to return the maximum likelihood values by interpolating over a two dimensional histogram made of two sets of samples.

Parameters

samples1, samples2
[array or list] Arrays or lists from which to find two dimensional maximum likelihood values.

weights
[array of floats] Weights to use in histogram.

bins
[list of ints] List of 2 integers which dictates number of bins for samples1 and samples2.

x_range
[tuple, optional] Range of samples1

y_range
[tuple, optional] Range of samples2

logx
[bool, optional] A value of True use log10 scale for samples1.

logy
[bool, optional] A value of True use log10 scale for samples2.

logz
[bool, optional] A value of True indicates that the z axis is in log10.

la_forge.utils.**get_Tspan**(*pulsar*, *datadir*)

Returns timespan of a pulsars dataset by loading the pulsar as an *enterprise.Pulsar()* object.

Parameters

pulsar
[str]

datadir
[str] Directory where *par* and *tim* files are found.

la_forge.utils.**get_params_2d_mlv**(*core*, *par1*, *par2*)

Convenience function for finding two dimensional maximum likelihood value for any two parameters.

la_forge.utils.**get_rn_noise_params_2d_mlv**(*core*, *pulsar*)

Convenience function to find 2d rednoise maximum likelihood values.

la_forge.utils.**powerlaw**(*freqs*, *log10_A=-16*, *gamma=5*)

la_forge.utils.**quantize_fast**(*toas*, *residuals*, *toaerrs*, *dt=0.1*)

Function to quantize and average TOAs by observation epoch. Used especially for NANOGrav multiband data.

Based on [3].

Parameters

toas
[array]

residuals
[array]

toaerrs
[array]

dt
[float] Coarse graining time [sec].

la_forge.utils.**rn_power**(*amp*, *gamma=None*, *freqs=None*, *T=None*, *sum_freqs=True*)

Calculate the power in a red noise signal assuming the P=A^2(f/f_yr)^-gamma form.

la_forge.utils.**set_publication_params**(*param_dict=None*, *scale=0.5*)

la_forge.utils.**weighted_quantile**(*values*, *quantiles*, *sample_weight=None*, *values_sorted=False*, *old_style=False*)

Very close to numpy.percentile, but supports weights. NOTE: quantiles should be in [0, 1]! [From Max Ghenis via Stack Overflow: https://stackoverflow.com/questions/21844024/weighted-percentile-using-numpy]

Parameters

values
[numpy.array] The data.

quantiles
[array-like] Many quantiles needed.

sample_weight
[array-like] Samples weights. The same length as *array*.

values_sorted
[bool] If True, then will avoid sorting of initial array.

old_style: bool
If True, will correct output to be consistent with numpy.percentile.

Returns

computed quantiles
[numpy.array]

Module contents

Top-level package for La Forge.

1.1.9 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/Hazboun6/la_forge/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

La Forge could always use more documentation, whether as part of the official La Forge docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Hazboun6/la_forge/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *la_forge* for local development.

1. Fork the *la_forge* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/la_forge.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv la_forge
$ cd la_forge/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 la_forge tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/Hazboun6/la_forge/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests.test_la_forge
```

Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

1.1.10 Credits

Development Leads

- Jeffrey Hazboun <jeffrey.hazboun@gmail.com>

Contributors

- Kristina Islo
- Aaron Johnson
- Sarah Vigeland

1.1.11 History

1.0.2 (2022-07-19) Fixes a delimiter bug for pulling in priors.txt.

1.0.1 (2021-10-21) Fixes a bug that did not allow parameter dictionary I/O in HyperModelCores

1.0.0 (2021-10-18) Capabilities to save as HDF5 files. Added full documentation and filled out the testing suite.

0.4.0 (2021-09-24) Added CI testing suite and cleaned up functions.

0.3.0 (2020-10-28) New docs.

0.2.0 (2020-02-13) Cleaned up various functionality and added more docs.

0.1.0 (2018-09-21)*

- First release on PyPI.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

l

la_forge, 36
la_forge.core, 25
la_forge.gp, 29
la_forge.rednoise, 30
la_forge.slices, 33
la_forge.utils, 34

INDEX

B

bayes_fac() (*in module la_forge.utils*), 34

C

compute_rho() (*in module la_forge.utils*), 34
Core (*class in la_forge.core*), 25
credint() (*la_forge.core.Core method*), 26

D

determine_if_limit() (*in module la_forge.rednoise*), 30

E

epoch_ave_resid() (*in module la_forge.utils*), 34

F

figsize() (*in module la_forge.utils*), 34

G

get_col() (*in module la_forge.slices*), 34
get_idx() (*in module la_forge.slices*), 34
get_map_dict() (*la_forge.core.Core method*), 26
get_map_param() (*la_forge.core.Core method*), 26
get_param() (*la_forge.core.Core method*), 26
get_param() (*la_forge.core.TimingCore method*), 28
get_param_credint() (*la_forge.core.Core method*), 26
get_param_median() (*la_forge.core.Core method*), 27
get_params_2d_mlv() (*in module la_forge.utils*), 35
get_rn_freqs() (*in module la_forge.rednoise*), 30
get_rn_noise_params_2d_mlv() (*in module la_forge.utils*), 35
get_Tspan() (*in module la_forge.rednoise*), 30
get_Tspan() (*in module la_forge.utils*), 35
getMax2d() (*in module la_forge.utils*), 34

H

HyperModelCore (*class in la_forge.core*), 28

L

la_forge
    module, 36
la_forge.core
    module, 25
la_forge.gp
    module, 29
la_forge.rednoise
    module, 30
la_forge.slices
    module, 33
la_forge.utils
    module, 34
load_Core() (*in module la_forge.core*), 28

M

map_idx (*la_forge.core.Core property*), 27
map_params (*la_forge.core.Core property*), 27
mass_function() (*la_forge.core.TimingCore method*), 28
mass_pulsar() (*la_forge.core.TimingCore method*), 28
median() (*la_forge.core.Core method*), 27
model_core() (*la_forge.core.HyperModelCore method*), 28
module
    la_forge, 36
    la_forge.core, 25
    la_forge.gp, 29
    la_forge.rednoise, 30
    la_forge.slices, 33
    la_forge.utils, 34

P

plot_free_spec() (*in module la_forge.rednoise*), 30
plot_powerlaw() (*in module la_forge.rednoise*), 31
plot_rednoise_spectrum() (*in module la_forge.rednoise*), 31
plot_tprocess() (*in module la_forge.rednoise*), 33
powerlaw() (*in module la_forge.utils*), 35

Q

quantize_fast() (*in module la_forge.utils*), 35

R

reconstruct_signal()

(*la_forge.gp.Signal_Reconstruction method*), 29
rn_power() (*in module la_forge.utils*), 36

S

sample_params() (*la_forge.gp.Signal_Reconstruction method*), 29
sample_posterior() (*la_forge.gp.Signal_Reconstruction method*), 29
save() (*la_forge.core.Core method*), 27
set_burn() (*la_forge.core.Core method*), 27
set_fancy_par_names() (*la_forge.core.Core method*), 27
set_publication_params() (*in module la_forge.utils*), 36
set_rn_freqs() (*la_forge.core.Core method*), 27
Signal_Reconstruction (*class in la_forge.gp*), 29
SlicesCore (*class in la_forge.slices*), 33
store_chains() (*in module la_forge.slices*), 34

T

TimingCore (*class in la_forge.core*), 28

W

weighted_quantile() (*in module la_forge.utils*), 36