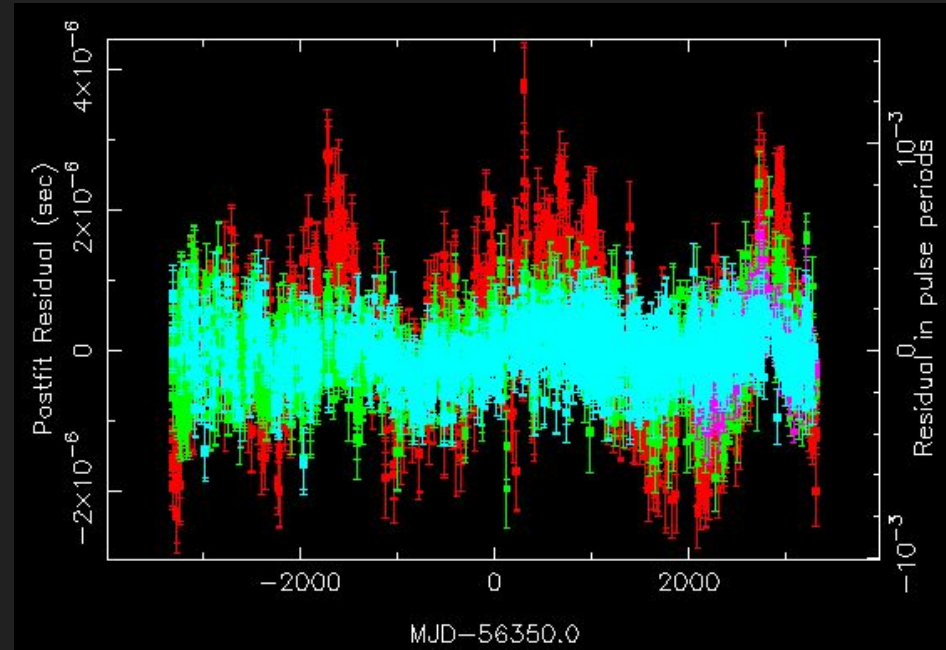


# What is Enterprise?

Is it good?

# Noise modelling

- We will be reading in .par and .tim files into Enterprise to model the noise
- **White** noise
  - Uncorrelated in time
- **Red** noise
  - Time-correlated
  - Frequency-independent
- **Dispersion** measure variations
  - Time-correlated
  - $1/\text{Frequency}^{**2}$  scaling
- Other noise processes including
  - Exponential dip events
  - Chromatic/band noise
- Soon after we will search for gravitational waves too!



# Enterprise in a nutshell

- Inference tool for pulsar noise analysis and GW searches
- Challenge:
  - Strong covariance between noise processes and GWB
  - Need to search for processes simultaneously
- Identify noise processes in individual pulsars
- Marginalise (analytically or numerically) over nuisance parameters
  
- Most commonly used tool for nHz-frequency GW searches

# Single-pulsar noise analysis

Pure Enterprise!

# Imports explained

“Signals” contains everything needed to construct a noise model for a pulsar:

Priors, noise model functions, selection methods, and lots of useful utilities

The sampler. We can use lots of samplers through Bilby later

A custom model function we define

```
import os
import sys
import numpy as np
from enterprise import constants as const
from enterprise.pulsar import Pulsar
from enterprise.signals import signal_base
from enterprise.signals import white_signals
from enterprise.signals import gp_signals
from enterprise.signals import parameter
from enterprise.signals import selections
from enterprise.signals import gp_priors
from enterprise.signals import deterministic_signals
from enterprise.signals import gp_bases
# from enterprise_extensions import blocks
from PTMCMCSampler.PTMCMCSampler import PTSampler as ptmcmc
```

Pulsar object holds data, e.g: ToAs, ephem, flags

```
@signal_base.function
def chrom_exp_decay(toas, freqs, log10_Amp=-7, sign_param=-1.0,
                   t0=54000, log10_tau=1.7, idx=2, ref_freq=1400):
    """
    Chromatic exponential-dip delay term in TOAs.
    :param t0: time of exponential minimum [MJD]
    :param tau: 1/e time of exponential [s]
    :param log10_Amp: amplitude of dip
    :param sign_param: sign of waveform
    :param idx: index of chromatic dependence
    :param ref_freq: reference frequency in MHz
    :return wf: delay time-series [s]
    """
    t0 *= const.day
    tau = 10**log10_tau * const.day
    ind = np.where(toas > t0)[0]
    wf = 10**log10_Amp * np.heaviside(toas - t0, 1)
    wf[ind] *= np.exp(-(toas[ind] - t0) / tau)

    return np.sign(sign_param) * wf * (ref_freq / freqs) ** idx
```

# Setting up a single-pulsar noise analysis in python

- Load in some data for your pulsar(s)

```
# Read in pulsar name, data directory, and chain number
psrname = sys.argv[1]
chainnum = sys.argv[2]
dir = sys.argv[3]
datadir = os.path.abspath("./data/" + str(dir))

# load in pulsar data
pfile = os.path.join(datadir, psrname + ".par")
tfile = os.path.join(datadir, psrname + ".tim")
psr = Pulsar(pfile, tfile, ephem='DE440')
```

Read arguments from  
command-line e.g.

> python singlePsrNoise.py J1643-1224 1 uwl

- Creating a noise model
  - Choose **selections** for noise parameters (e.g. backend-dependent)
  - Choose **priors** for parameters (can be a constant, which you read-in from a file)
  - Choose noise **models**, and apply selections and priors
  - Sum all noise models, and apply total noise model to a “**PTA**” object containing your pulsar(s)
- Set up sampler (e.g. PTMCMC, or something else via Bilby)

# Timing model

- No selection required because it applies to every ToA
- No prior required as these are determined from a timing model fit
- Marginalise over linear timing model
- Timing model parameters can be sampled

```
○  
# define useful selections  
by_backend = selections.Selection(selections.by_backend)  
no_selection = selections.Selection(selections.no_selection)  
  
"""  
Choose whether to marginalise or sample the timing model  
"""  
tm = gp_signals.MarginalizingTimingModel(use_svd=True)
```

(defining some  
selections for later)

Choosing to  
marginalise over  
timing model

# White noise

- Selection, priors, and models

```
"""
Define white noise model
"""
# EFAC "MeasurementNoise" can add equad, but only t2equad - we want the tnequad
efac_prior = parameter.Uniform(0.01, 10.0)
wn = white_signals.MeasurementNoise(
    efac=efac_prior,
    selection=by_backend)

# EQUAD - TempoNest definition: sigma = sqrt((efac*sigma_0)**2 + (tnequad)**2)
log10_equad_prior = parameter.Uniform(-10, -5)
wn += white_signals.TNEquadNoise(
    log10_tnequad=log10_equad_prior,
    selection=by_backend)

# ECORR - we will swap to "white_signals.EcorrKernelNoise" later
log10_ecorr_prior = parameter.Uniform(-10, -5)
wn += gp_signals.EcorrBasisModel(
    log10_ecorr=log10_ecorr_prior,
    selection=by_backend)
```

Uniform linear prior

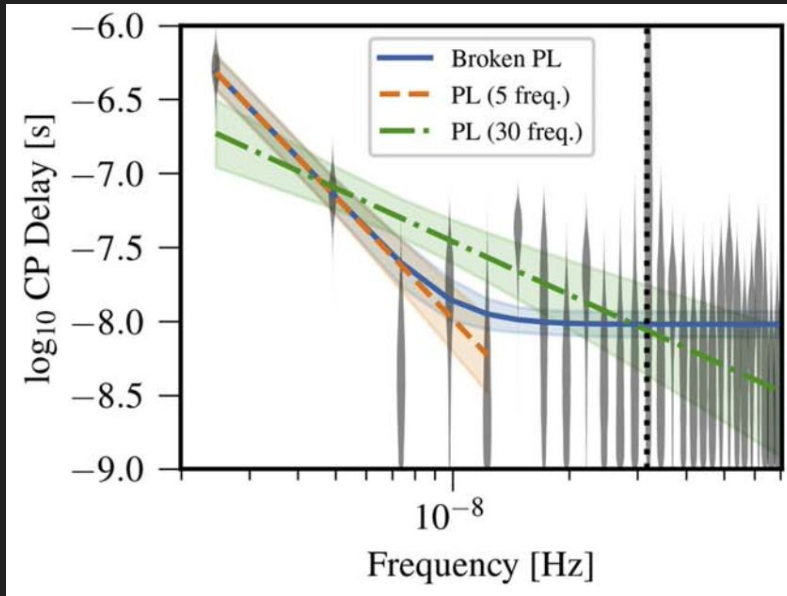
Uniform log prior

Selections are by\_backend. One white noise parameter per "-group" flag



# Red and DM noise

- Number of Fourier components
- Powerlaw vs “broken-powerlaw”?



```
"""
Define red noise model
"""
log10_A_prior = parameter.Uniform(-20, -11)
gamma_prior = parameter.Uniform(0, 7)
# powerlaw
rn_model = gp_priors.powerlaw(log10_A=log10_A_prior,
                              gamma=gamma_prior)

# # broken powerlaw
# lfb_prior = parameter.Uniform(-10, -7) # log10 transition frequency
# kappa_prior = 0.1 # smoothness of transition (Default = 0.1)
# delta_prior = 0 # slope for frequencies > f_break
# rn_model = gp_priors.broken_powerlaw(log10_A=log10_A_prior,
#                                     gamma=gamma_prior,
#                                     delta=delta_prior,
#                                     log10_fb=lfb_prior,
#                                     kappa=kappa_prior)

components = 30
rn = gp_signals.FourierBasisGP(rn_model, components=components,
                              selection=no_selection, name='red')
# rn = blocks.red_noise_block(components=30) # The easy way

"""
Define DM noise model
"""
log10_A_dm_prior = parameter.Uniform(-20, -11)
gamma_dm_prior = parameter.Uniform(0, 7)
dm_model = gp_priors.powerlaw(log10_A=log10_A_dm_prior,
                              gamma=gamma_dm_prior)
dm_basis = gp_bases.createfourierdesignmatrix_dm(nmodes=components)
components = 30
dm = gp_signals.BasisGP(dm_model, dm_basis, name='dm')
# dm = blocks.dm_noise_block(components=30) # The easy way
```

# Define the total model

Total model is the sum of components

$$s = tm + wn + dm + rn$$

Define exponential dip parameter priors based on previous studies.

The “waveform” of this model is what we defined at the start

Add some exponential dip parameters to the model for pulsars, if using the PPTA-DR2 component

```
"""
Define total model by summing all components
"""
# define model
s = tm + wn + dm + rn

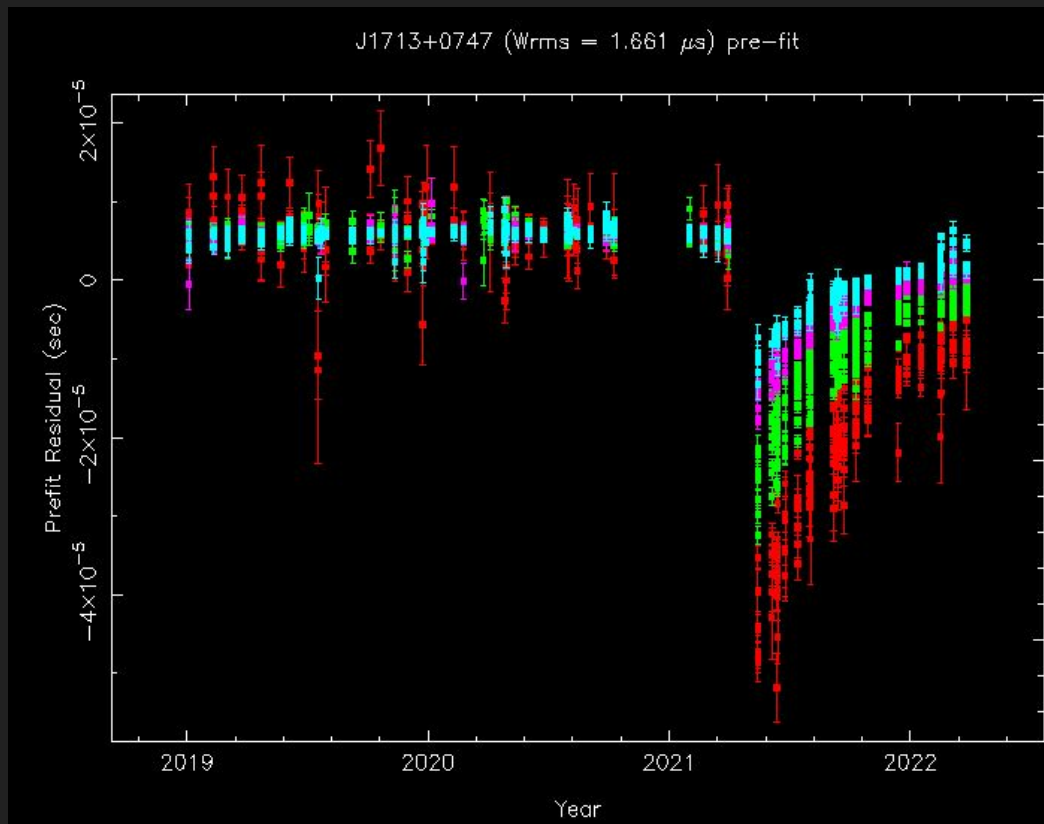
"""
Add special noise model components for some pulsars
"""
# Define exponential dip parameters for 0437, 1643, and 1713
if psr.name == 'J1713+0747' and psr.toas.min() < 57500*86400:
    expdip = True
    num_dips = 2
    idx = [parameter.Uniform(1.0, 3.0), parameter.Uniform(0.0, 2.0)]
    tmin = [54650, 57400] # centred 54750 and 57510
    tmax = [54850, 57600]
elif psr.name == 'J0437-4715' and psr.toas.min() < 57100*86400:
    expdip = True
    num_dips = 1
    idx = [parameter.Uniform(-1.0, 2.0)]
    tmin = [57000]
    tmax = [57200]
elif psr.name == 'J1643-1224' and psr.toas.min() < 57100*86400:
    expdip = True
    num_dips = 1
    idx = [parameter.Uniform(-2.0, 0.0)]
    tmin = [57000]
    tmax = [57200]
else:
    expdip = False

# Add exponential dipoles to model
if expdip:
    name = ['exp_{0}'].format(ii+1) for ii in range(num_dips)]

    for idip in range(num_dips):
        t0_exp = parameter.Uniform(tmin[idip], tmax[idip])
        log10_Amp_exp = parameter.Uniform(-10, -2)
        log10_tau_exp = parameter.Uniform(0, 2.5)
        # Define chromatic exponential decay waveform
        wf = chrom_exp_decay(log10_Amp=log10_Amp_exp,
                             t0=t0_exp, log10_tau=log10_tau_exp,
                             sign_param=-1.0, idx=idx[idip])
        expdip = deterministic_signals.Deterministic(wf, name=name[idip])
    s += expdip
```

# Exponential dip events

- Changes to the pulsar magnetosphere.
- Four observed in PPTA-DR2
- J1643-1224
  - High-frequency first
- J1713+0747
  - Two events near  $\text{freq}^{-2}$
  - Third (right) deleted from PPTA-DR3
- J0437-4715
  - Low-frequency first,  $\text{freq}^{-1}$



# Form a “PTA” and sample

The PTA object has useful functions for model selection, setting up the sampler, and holding models with multiple pulsars

$10^6$  -  $10^7$  samples depending on complexity of data set and model

Common partial PTA:  $N \sim 1e7$  (fixed white)

Common full PTA with correlations:

$N \sim 10^8$  (fixed white)

Save parameters to file for plotting later

```
"""
Set up your PTA likelihood and your sampler
"""
# set up PTA
pta = signal_base.PTA(s(psr))

# set initial parameters drawn from prior
x0 = np.hstack([p.sample() for p in pta.params])
ndim = len(x0)

# sampler for N steps
N = int(1e6)

# initial jump covariance matrix
cov = np.diag(np.ones(ndim) * 0.01**2)

# output directory
outdir = datadir + '/chains/singlePsrNoise/' + psrname + "_" + chainnum

# Use PTMCMC sampler. We can easily update this to use e.g. Bilby instead
sampler = ptmcmc(ndim, pta.get_lnlikelihood, pta.get_lnprior, cov,
                 outDir=outdir, resume=False)

# Print parameter names and write to a file
print(pta.param_names)
filename = outdir + "/pars.txt"
if os.path.exists(filename):
    os.remove(filename)
with open(filename, "a") as f:
    for par in pta.param_names:
        f.write(par + '\n')

# Sample! The sampler parameters can be left as default.
sampler.sample(x0, N, SCAMweight=30, AMweight=15, DEweight=50)
```

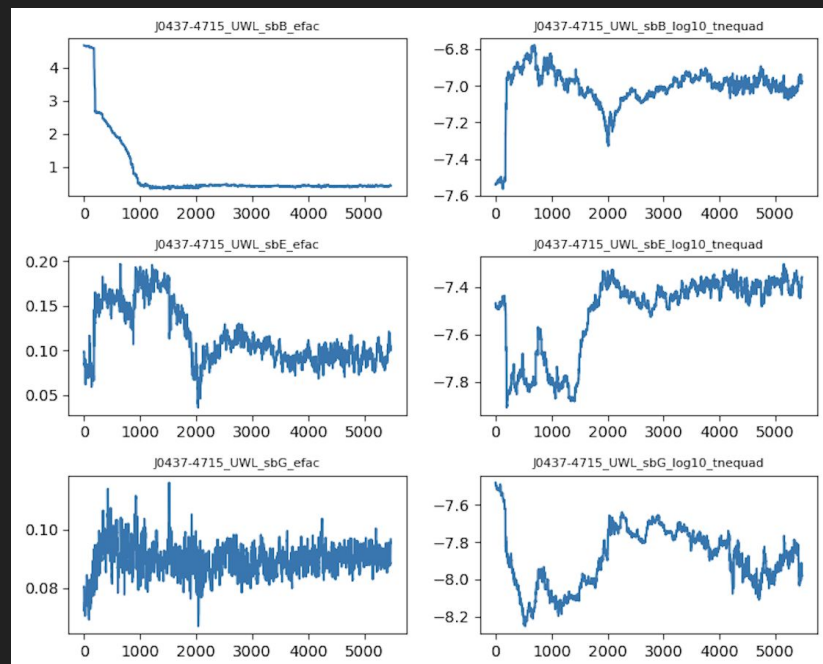
Bilby can replace these lines

# Post-processing

- Create plots of the chains
- “Burn” and “thin”
  - Get rid of first N samples, and then take every  $i^{\text{th}}$ , where  $i$  is the autocorrelation length of the chain
- Save parameters to .json noise files
  - These can be read-in later, to set constant priors for future runs.  
E.g. for fixed white-noise analyses

## makeNoise.py

- Combines multiple chains (if any), burns, plots, and makes noise .json files.
- Note: Does not thin the chains (action item!)



```
{  
  "J1125-6014_UWL_CASPSR_20CM_efac": 0.5328980996565253,  
  "J1125-6014_UWL_CASPSR_20CM_log10_tnequad": -7.310371507414705,  
  "J1125-6014_UWL_CASPSR_40CM_efac": 1.287952621705652,  
  "J1125-6014_UWL_CASPSR_40CM_log10_tnequad": -7.298227141620199,  
}
```

# Noise model comparison

With hypermodel

(Following `singlePsrNoiseComparison.py` in `/fred/oz002/dreardon/ppta_dr3/pipeline/dr3/ppta-dr3` )

```
> python singlePsrNoiseComparison.py J1643-1224 1 test
```

# Noise model comparison

- Unlike nested samplers, the mcmc sampler does not provide the Bayesian evidence. Instead, model comparison is treated as a parameter estimation problem through a “hypermodel parameter”, *nmodel*
- The *nmodel* parameter toggles between two competing models, and its posterior distribution is a measure of the “odds ratio”, which is the Bayes factor multiplied by a prior odds
  - Note: if there is very strong evidence for one model, the alternative may never be sampled. The number of samples gives only a lower-limit on the Bayes factor

# Single-pulsar noise model comparison

- Follow the single-pulsar analysis, but add an alternative model
  - In this case, chromatic noise with scaling  $f^{-4}$

```
"""
Define chromatic noise model
"""
log10_A_chrom_prior = parameter.Uniform(-20, -11)
gamma_chrom_prior = parameter.Uniform(0, 7)
chrom_model = utils.powerlaw(log10_A=log10_A_chrom_prior,
                             gamma=gamma_chrom_prior)

idx = 4 # Define freq-idx scaling
chrom_basis = gp_bases.createfourierdesignmatrix_chromatic(nmodes=components,
                                                           idx=idx)

components = 30
chrom = gp_signals.BasisGP(chrom_model, chrom_basis, name='chrom')
```

```
"""
Define total model by summing all components
"""
# define models
s = tm + wn + dm + rn
```

Alternative model

```
"""
Set up alternative model
"""
s2 = s + chrom
```

Same total model as before



# Form a “Hypermodel” and sample

- Now we use `enterprise_extensions` for the first time

```
from enterprise_extensions import hypermodel
```

- The “hypermodel” is convenient for setting up the sampler

We create a dictionary containing two PTAs, one for each model, and then pass this to the hypermodel. It adds the “nmodel” parameter for us.

It has some shortcuts for setting up the sampler too

```
"""
Set up your PTA likelihood and your sampler
"""
# set up PTA and add it to a hypermodel
nmodels = 2
pta = dict.fromkeys(np.arange(nmodels))
pta[0] = signal_base.PTA(s(psr))
pta[1] = signal_base.PTA(s2(psr))
hyper_model = hypermodel.HyperModel(pta)

# set initial parameters drawn from prior
x0 = hyper_model.initial_sample()
ndim = len(x0)

# sampler for N steps
N = int(1e6)

# output directory:
outdir = datadir + '/chains/singlePsrNoiseComp/' + psrname + "_" + chainnum

# Use PTMCMC sampler. We can easily update this to use e.g. Bilby instead
sampler = hyper_model.setup_sampler(outdir=outdir, resume=False)

# Print parameter names and write to a file
print(hyper_model.param_names)
filename = outdir + "/pars.txt"
if os.path.exists(filename):
    os.remove(filename)
with open(filename, "a") as f:
    for par in hyper_model.param_names:
        f.write(par + '\n')

# Sample! The sampler parameters can be left as default.
sampler.sample(x0, N, SCAMweight=30, AMweight=15, DEweight=50)
```

# Common noise

(Following `commonNoise.py` in `/fred/oz002/dreardon/ppta_dr3/pipeline/dr3/ppta-dr3` )

```
> python commonNoise.py 1 test
```

# Common noise analysis

- Now we want to read in multiple .par and .tim files and set up a pulsar for each one

```
# Read in data directory, and chain number
chainnum = sys.argv[1]
dir = sys.argv[2]
datadir = os.path.abspath("./data/" + str(dir))

psrnames = ['J1125-6014', 'J1600-3053', 'J1643-1224']

# parfiles = sorted(glob.glob(datadir + '/*.par'))
# timfiles = sorted(glob.glob(datadir + '/*.tim'))
parfiles = []
timfiles = []
for psrname in psrnames:
    parfiles.append(datadir + '/' + psrname + '.par')
    timfiles.append(datadir + '/' + psrname + '.tim')

psrs = []
for p, t in zip(parfiles, timfiles):
    print("loading..", p, t)
    psr = Pulsar(p, t, ephem='DE440')
    if psr.name in psrnames:
        psrs.append(psr)
```

Select a few pulsars

Uncomment to use all pulsars


Add each pulsar to a list

# Common noise analysis

- We also want to fix white-noise parameters, so we read in the noise files

```
noisefiles = sorted(glob.glob(datadir+'/noiseFiles/*.json'))
noisedict = {}
for nf in noisefiles:
    with open(nf, 'r') as f:
        noisedict.update(json.load(f))
```

Save all noise parameters to one dictionary



# Choosing a common noise process

- Use the `enterprise_extensions` shortcut

```
from enterprise_extensions.blocks import common_red_noise_block
```

- Powerlaw or “broken” powerlaw? Correlations or none?

```
"""
Define common red noise
"""
# Powerlaw no correlations
crn = common_red_noise_block(psd='powerlaw', prior='log-uniform',
                             components=30, orf=None, name='common')
# # Broken powerlaw no correlations
# crn = common_red_noise_block(psd='broken_powerlaw', prior='log-uniform',
#                               components=30, orf=None, name='common')
# # Powerlaw Hellings-Downs
# crn = common_red_noise_block(psd='powerlaw', prior='log-uniform',
#                               components=30, orf='hd', name='gwb')
# # Powerlaw Monopole
# crn = common_red_noise_block(psd='powerlaw', prior='log-uniform',
#                               components=30, orf='monopole', name='monopole')
# # Powerlaw Dipole
# crn = common_red_noise_block(psd='powerlaw', prior='log-uniform',
#                               components=30, orf='dipole', name='dipole')
"""
Define total model by summing all components
"""
# define model
s = tm + wn + dm + rn + crn
```

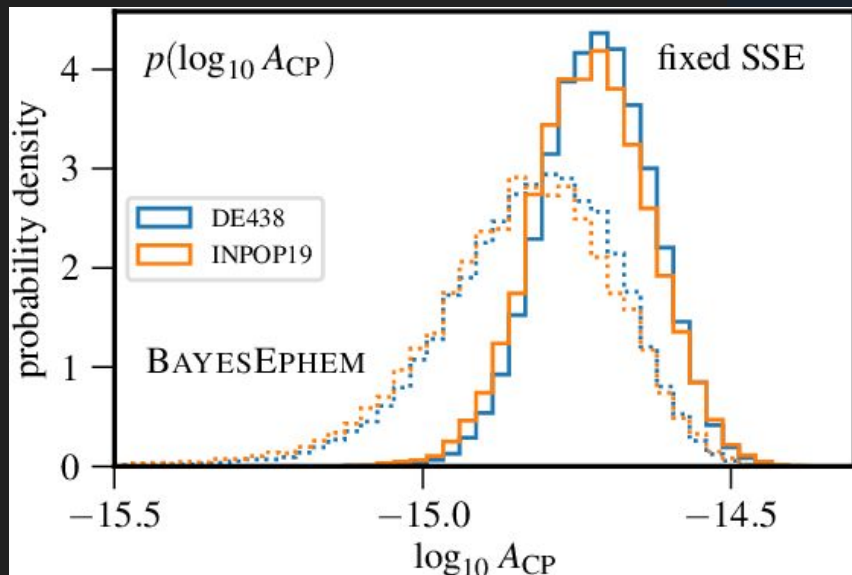
H-D, monopole,  
dipole, or none

Remember to  
add to the total  
model

# Bayesephem?

- Do we want to fit for perturbations in the masses of major planets and of the orbit of jupiter?

```
# Add ephemeris model?  
bayesephem = False  
if bayesephem:  
    s += deterministic_signals.PhysicalEphemerisSignal(use_epoch_toas=True,  
                                                       model='setIII_1980')
```



NANOGrav 12.5yr

# Loop through pulsars and set constant priors

- For each pulsar, add its model to a list

```
model = []
for psr in psrs:
    # Define exponential dip parameters for 0437, 1643, and 1713
    if psr.name == 'J1713+0747' and psr.toas.min() < 57500*86400:
    elif psr.name == 'J0437-4715' and psr.toas.min() < 57100*86400:
    elif psr.name == 'J1643-1224' and psr.toas.min() < 57100*86400:
    else:

    # Add exponential dipoles to model
    if expdip:
        name = ['exp_{0}'].format(ii+1) for ii in range(num_dips)]

        s2 = s
        for idip in range(num_dips):
            model.append(s2(psr))
    else:
        model.append(s(psr))

    """
    Set up your PTA likelihood and your sampler
    """
    # set up PTA
    pta = signal_base.PTA(model)
    pta.set_default_params(noisedict)
```

Append to  
model list

(code collapsed for  
readability)

- Then set constant parameters to the values from the noise dictionary


Launching to OzStar



- Example slurm script below
- Can also launch multiple jobs with enterprise.engage
  - Multiple chains simultaneously for multiple pulsars

```
#!/bin/bash
#SBATCH --job-name=1_J0437-4715
#SBATCH --output=/fred/oz002/dreardon/ppta_dr3/pipeline/dr3/enterprise/jobs/J0437-4715_1_%J.out
#SBATCH --ntasks=1
#SBATCH --time=48:00:00
#SBATCH --mem-per-cpu=5g
```


“%J” makes the job  
name unique



```
ml anaconda3/2021.05
conda init bash
source ~/.bashrc
conda activate ent15y
```

enterprise.engage script can iterate chain  
number and/or pulsar name

```
cd /fred/oz002/dreardon/ppta_dr3/pipeline/dr3/enterprise
python singlePsrNoiseComparison.py J0437-4715 1 uwl
```



# The plan!

(Discussion time)

- Step 0: Finalise dataset
  - Outliers in J0437-4715
  - Any more jumps?
  - How many FD parameters per pulsar? Any other missing parameters?
- Step 1: Which pulsars have red noise and DM variations?
- Step 2: Determine white noise parameters using best model
  - Which pulsars and which groups need ECORRs?
  - How many components for the red noise? Test with broken powerlaw?
  - Set DM components based on highest frequency =  $1/(N \text{ days})$
- Step 3: First common noise search
  - Fixed white noise
  - Red noise for all pulsars? Same priors and number of components?
- Step 4: Advanced noise modelling
  - Searching for any additional processes (e.g. missing JUMPs, chromatic, excess low-freq)
  - Searching for correlations and adjusting individual pulsar noise models
    - Changing number of components, and adding extra noise terms
- Step 5: Validation via sky/phase scrambles

Steps 1 - 3 are easy to do using the tools from this workshop!

Step 4 may require some new selections or model functions, and Bilby

# Further reading

- NANOGrav 12.5 year analysis:  
<https://iopscience.iop.org/article/10.3847/2041-8213/abd401/pdf>
- Bayesian inference intro:  
<https://arxiv.org/pdf/1809.02293.pdf>
- PPTA-DR2 analysis:  
<https://arxiv.org/pdf/2107.12112.pdf>
- NANOGrav 12.5 year analysis github page and notebooks:  
[https://github.com/nanograv/12p5yr\\_stochastic\\_analysis](https://github.com/nanograv/12p5yr_stochastic_analysis)
- enterprise\_extensions models file:  
[https://github.com/nanograv/enterprise\\_extensions/blob/master/enterprise\\_extensions/models.py](https://github.com/nanograv/enterprise_extensions/blob/master/enterprise_extensions/models.py)



# Workshop 2.0

20th Sept 2022

# Where are we currently?

- See also Andrew's slides:  
[https://docs.google.com/presentation/d/19mqlgaSa-5Gwet3Zs2p2QdAsmuHuhwEegIRlzfzI2\\_c/edit#slide=id.p](https://docs.google.com/presentation/d/19mqlgaSa-5Gwet3Zs2p2QdAsmuHuhwEegIRlzfzI2_c/edit#slide=id.p)
- Data set nearly completed (except J0437..)
- Noise models nearly complete (except J0437..)
- Common noise analyses started (without J0437..)
  
- New data and lots of updated scripts to use:

**`/fred/oz002/dreardon/ppta_dr3/pipeline/dr3/enterprise/workshop/ppta-dr3`**

# Single pulsar noise model results

- Likely candidates for chromatic noise:
  - J0437, J0613, J1017, J1045, J1600, J1643, J1939
- Likely candidates for low-frequency band noise:
  - As above, plus J1909-3744, J1713+0747
- Just use ECORR for everything
  - White noise will be fixed at small value if not significant
- Use maximum likelihood noise model
- Components. Currently suggesting:
  - 240 days for red and chromatic noise
  - 60 days for DM and band noise
- Optimised for best pulsars - e.g. J1909 had a definite peak here, in uwl and dr2

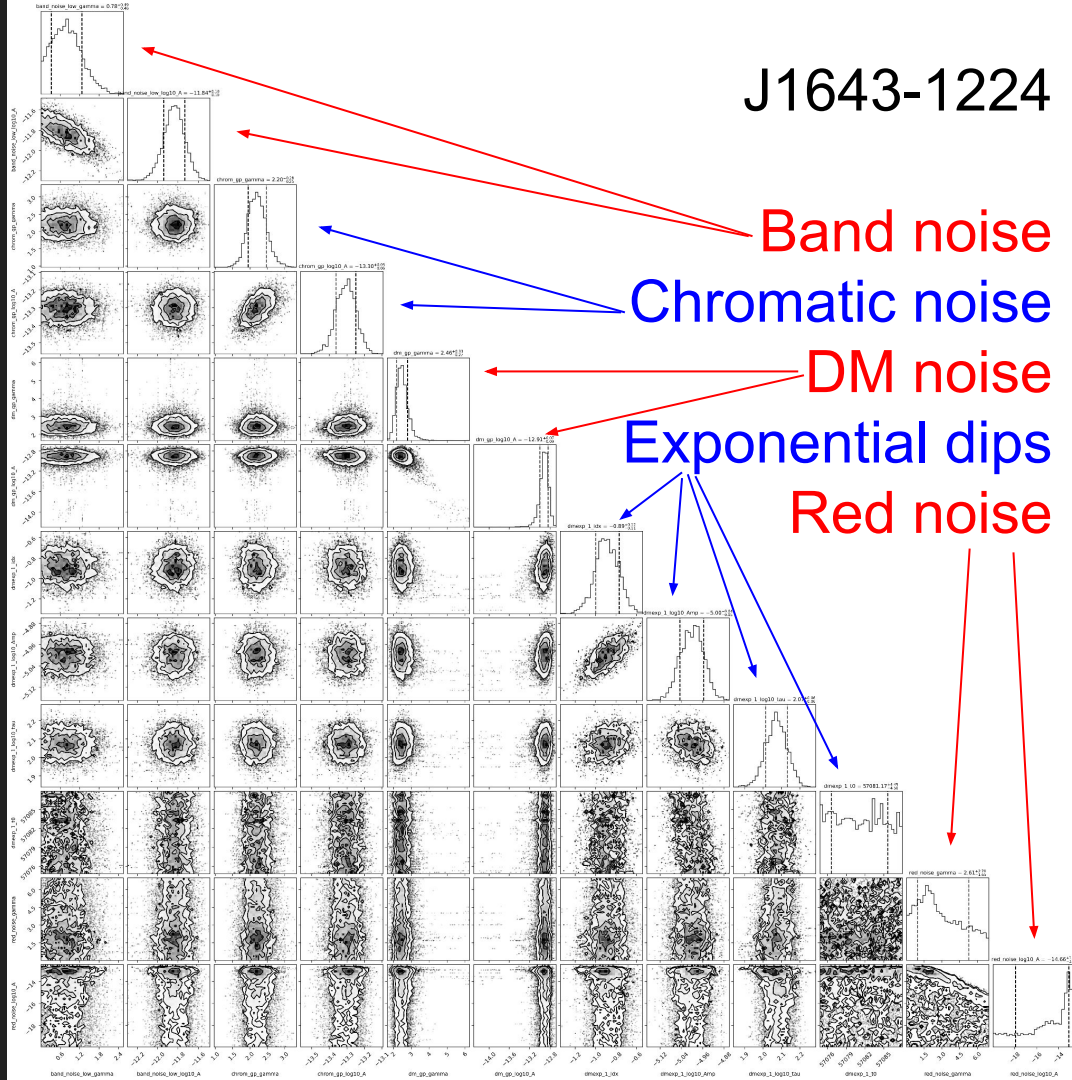


# Latest single-pulsar noise modelling

makeNoise.py

will make corner plots for everything except white noise parameters

## J1643-1224



# Testing noise models

tempo2 general2 plugin

```
$ tempo2 -output general2 -outfile J1643-1224.out -s "{file} {sat} {bat} {freq} {pre} {post}
{posttn} {err} {tndm} {tndmerr} {tnrn} {tnrnerr}\n" -f J1643-1224.par ../../J1643-1224.tim
```

Where to write the results

String instructions: What do you want printed, and how?

Input .par and .tim files. Temponest noise models in .par file

**enterprise\_to\_tnest.py** to make the par files

Then use, **make\_plots.py** to show the results. Add path to data in the first lines:

```
27
28     output_dir = '/Users/dreardon/Desktop/ppta-dr3/dr2/'
29     output_files = sorted(glob.glob(output_dir + 'J*.out'))
30     par_files = sorted(glob.glob(output_dir + 'J*.par'))
31
```

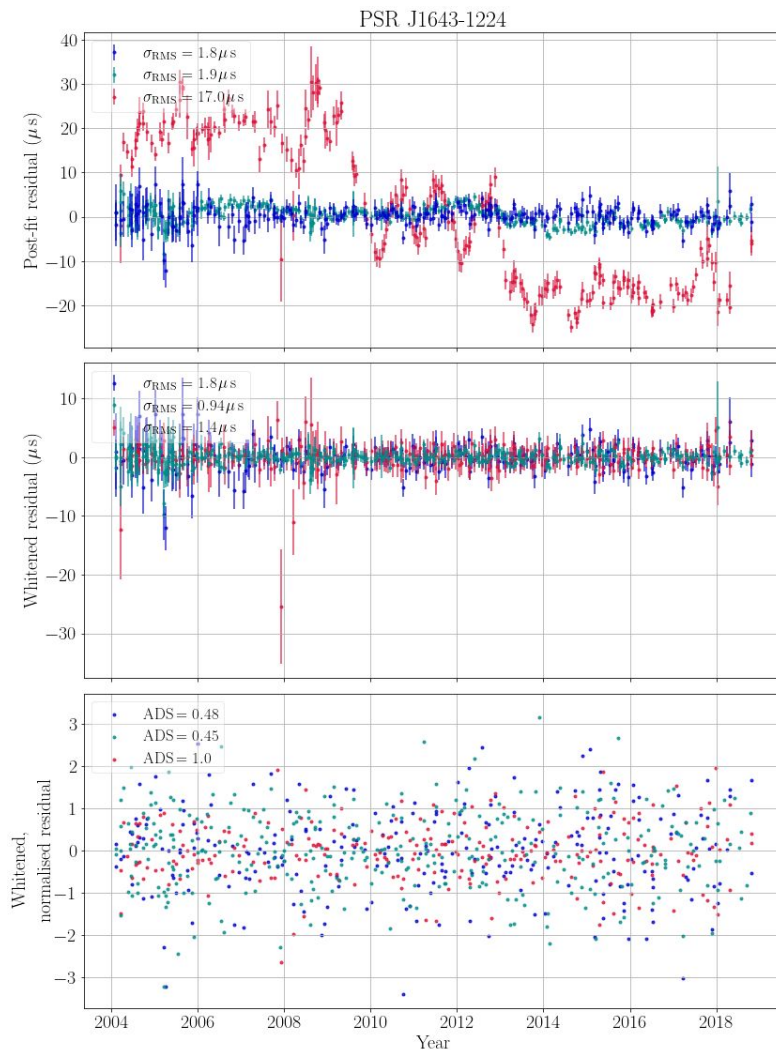
# Frequency-averaged residuals (with ECORR)

Right: Residuals with noise subtracted for J1643

`make_plots.py` output (*send path to Daniel*)

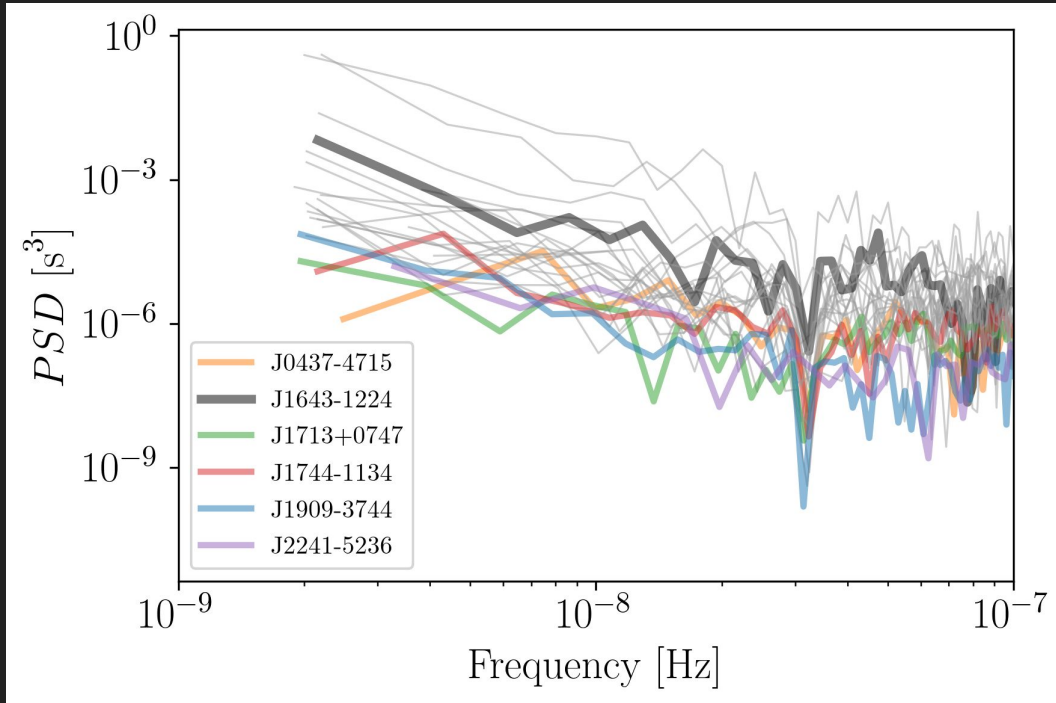
Passes whiteness and Gaussianity

- Fix white noise parameters at these values



# Power spectra visualised

- Andrew Zic produced Cholesky spectra for our pulsars using the dr2 portion

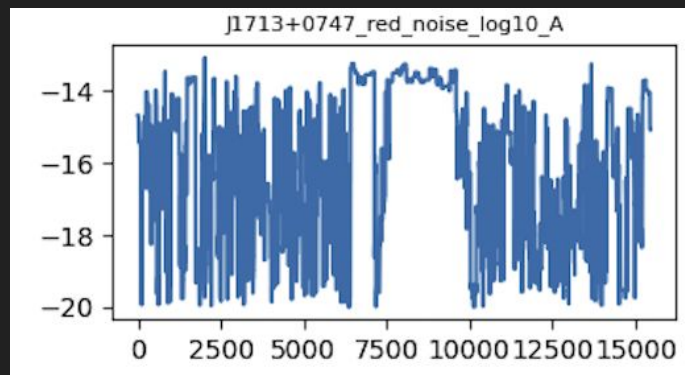
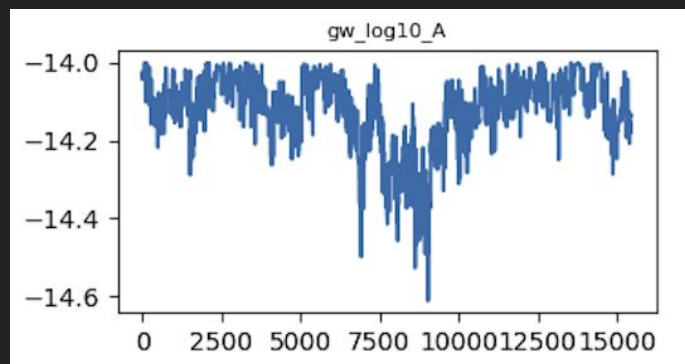


# DM subtraction

- Ryan realised that we need to subtract the  $DM(t)$  before doing optimal statistic work
- Also useful for accelerating our common noise search as we have  $2*N_{psr}$  fewer parameters!
- Running now (right)

```
./make_dmoff.sh J1909-3744 dr2
```

- Creates  
./data/dr2/noiseFiles\_maxlike/tnest/J1909-3744\_dmo.par  
and J1909-3744\_dmo.tim
- Note: All noise parameters removed from \_dmo.par



## Single pulsar recipe

Or singlePulsarNoise\_reddm.py



```
$ python singlePulsarNoise.py JXXXX-XXXX 1 dr2
```

(launch multiple with enterprise.engage)

Or uwl/all



```
$ python makeNoise.py JXXXX-XXXX dr2
```

```
$ python enterprise_to_tnest.py dr2
```

```
$ ./make_dmoff.sh JXXXX-XXXX dr2
```

makeNoise.py also does the corner plots now

Then we try **commonNoise\_os.py**

# Suggested action items:

- Daniel, Ryan, Andrew:
  - Finalise dataset. Check all pulsars for outliers using current best noise models
  - Daniel: Check ECORRs for all pulsars, using fully-averaged dataset. DR2 + UWL J0437
  - Andrew: Using normalised residuals, find outliers in UWL J0437
  - Ryan: Investigate UWL J1022, J2124, J2145
- Andrew:
  - Get break frequencies from power spectra
- Matt:
  - Take a script, e.g. singlePulsarNoise.py and add Bilby compatibility (e.g. importing minimum functions from bilby\_warp)
- Atharva, Axl, Rowina, Valentina, anyone else:
  - Single-pulsar noise analyses: **Red+DM only**. Band, chromatic, and components investigation
  - Fixed white noise for “all” dataset. Output general2
  - DM-subtracted datasets for Red + DM only
  - Common noise runs using DM-subtracted

# Other things to do

- Make a >1GHz (or 960MHz for UWL sub-band) dataset?
  - More compact, and probably no chromatic or band noise!
- Make dr3e?
  - ToAs and noise model unchained from previous analyses - extends to lower frequencies