High Precision Pulsar Timing with PINT a new software package

Jing Luo

University of Toronto

March 21, 2019
PyGamma19

Outline

- Pulsars
- Pulsar timing with PINT
  - Pulse time of arrival (TOA)
  - Modeling TOAs
  - Updating timing model
  - Applications
- Future plans

Pulsar (What is a pulsar?)

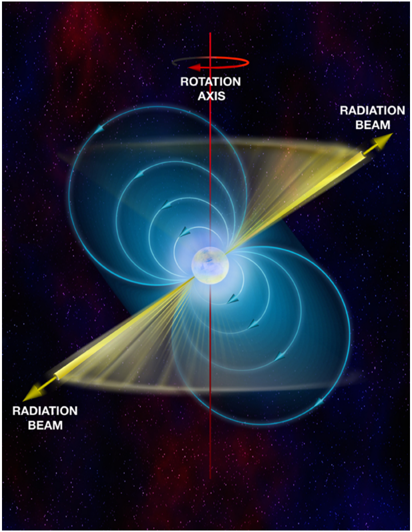
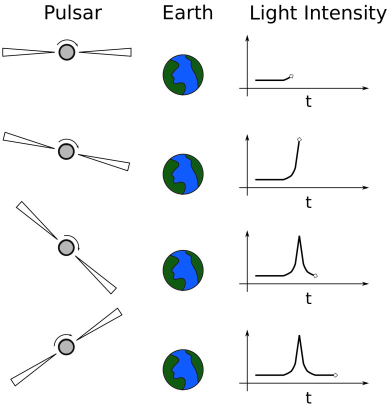


Image credit: Bill Saxton, NRAO/AUI/NSF

- Pulsar = Pulse + Star
- Pulsars are rapidly rotating neutron stars.
- Pulsars have strong beamed radiation from their magnetic poles.
- Radiations are wide band, some of them have Gamma Ray emission.

Pulsar (Light house effect)

Neutron star's bright beamed radiation sweeps past our line of sight once every rotation, similar to a light house.



Pulsar Data

Pulsar data are periodic pulsed signals

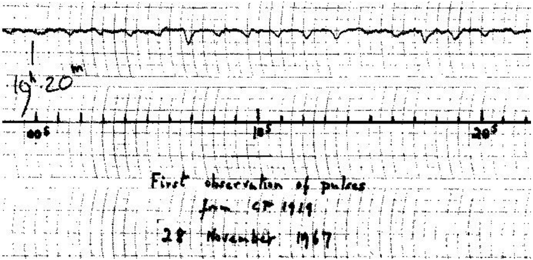


Figure 2: The pulsar discovery data

Pulsar Data in pop culture

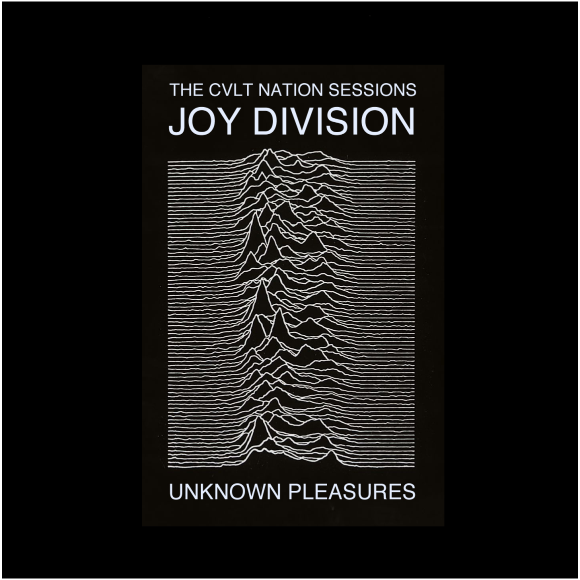


Figure 3: Pulses from CP1919 aliened with their period

Pulsar data at high energy

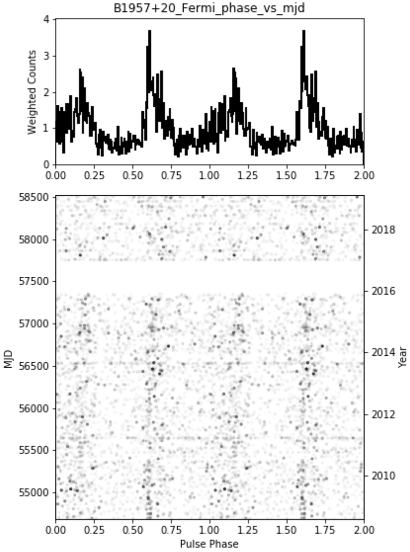


Figure 4: An example of PSR b1957+20 Fermi data.

A lof of astrophysical effects can change the pulse period?

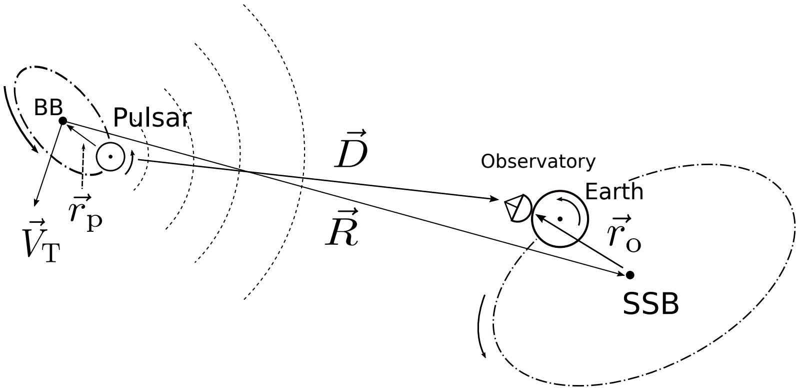


Figure 5: The geometry between Pulsar and Observatory

Pulsar Timing with PINT

Pulsar Timing

Pulsar timing is a technique of modeling astrophysical phenomena (e.g., pulsar emission and propagation) via the pulse time of arrivals (TOAs)

- Pulsar timing process

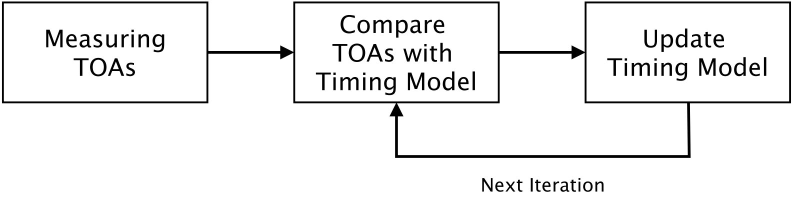


Figure 6

PINT (PINT Is Not Tempo3)

PINT is a Python based pulsar timing software.

- Independent developed from traditional timing software (TEMPO and TEMPO2).
- Highly Object oriented and modularized.
- Utilizing the well-debugged and widely used packages (`Astropy`, `Numpy`).
- Utilizing the unittest scheme.
- Well documented code.
- Using modern version control (git/github).

Code Architecture

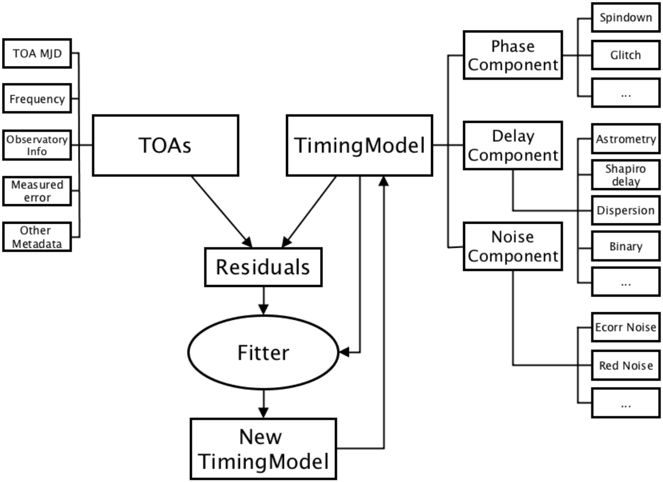


Figure 7: PINT code architecture

TOA and its Metadata

- Information of a TOA measurement

Name	Description
TOA MJD	Measured TOA in the format of MJD
TOA error	TOA measurement errors
Frequency	Observing frequency
Observatory	The observatory where the TOA is produced
Receiver	The receiver that made the TOA observation
Backend	The backend instrument that recodes the data
...	...

How do we organize this information?

PINT (TOA Module)

- The time is stored by:
  Astropy Time object and NumPy longdouble type in MJD format.
- Precision of time:
  1 ns (the precision to detect GWs.)
- TOAs and metadata are storaged in an Astropy Table.

<Table length=62>

idx	index	mjd	mjd_float	error	freq	obs	flags	tdb	tdbld
			d	us	MHz				
0	0	53478.2858714	53478.2858714	21.71	1949.609	gbt	{'clkcorr': <Quantity 2.7592372626226786e-05 s>, 'ddm': 0.0, 'format': 'Princeton'}	53478.2866143	53478.2866143
1	1	53483.2767052	53483.2767052	21.95	1949.609	gbt	{'clkcorr': <Quantity 2.761836126569327e-05 s>, 'ddm': 0.0, 'format': 'Princeton'}	53483.2774481	53483.2774481
2	2	53489.4683898	53489.4683898	29.95	1949.609	gbt	{'clkcorr': <Quantity 2.7662287640724238e-05 s>, 'ddm': 0.0, 'format': 'Princeton'}	53489.4691327	53489.4691327

Pre-process TOAs

1. Get the target time scale of TOAs.
   - Barycentric dynamic time (TDB).
2. Compute the observatory Coordinates at each TOA
   - Solar system barycentric reference system. International Celestial Reference System (ICRS), International Celestial Reference Frame (ICRF2) realization, is chosen for PINT.

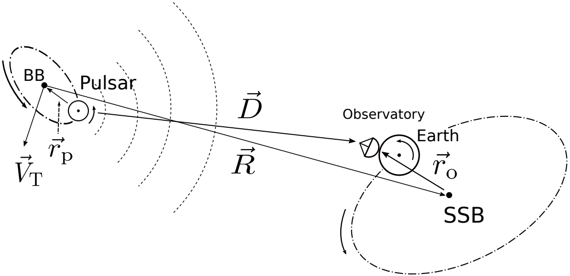


Figure 8

Clock Correction (get target timescale)

- Time scale TOAs are recorded:
  Observatory UTC (for Ground-based observatory)
  TT (for spacecrafts)
- Time scale needed:
  Barycentric Dynamic time (TDB)

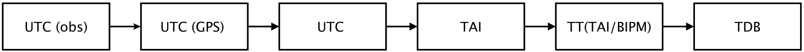


Figure 9

`Astropy` Time object helps the converting from (Universal Coordinate Time)UTC $\rightarrow$ TDB.
PINT makes the transform form UTC(obs) $\rightarrow$UTC and TT(BIPM) correction.

PINT (Observatory Module)

Observatory module functionality

- Provides a unified API for all the observatories (Ground-based and space-based).

In [4]:

```
import pint.toa as toa
toa.get_observatory('gbt')
```

- Calculates the clock corrections.
- Calculates the observatory position and velocity in ICRS.
  - Earth orientation is handled by IERS earth orientation parameters (EOP).
  - Earth location in ICRS is handled by JPL ephemeris.

PINT reading TOAs

pint.toas.get_TOAs() function handles:

- Reading TOAs from a file(e.g., .TIM)
- Applying clock corrections.
- Computing TDB and observatroy location.

```
>>> import pint.toa as toa
>>> tim = "NGC6440E.tim"
>>> t = toa.get_toas(tim)
INFO: Applying clock corrections. [pint.toa]
INFO: Getting IERS params and computing TDBs. [pint.toa]
INFO: Computing TDB columns. [pint.toa]
INFO: Computing observatory positions and velocities. [pint.toa]
INFO: Compute positions and velocities of observatories and Earth (planets = F
se), using DE421 ephemeris [pint.toa]
INFO: Adding columns ssb_obs_pos ssb_obs_vel obs_sun_pos [pint.toa]
```

Figure 10

Modeling TOAs

A timing model includes:

- Pulsar Rotation
- Pulse propagation time

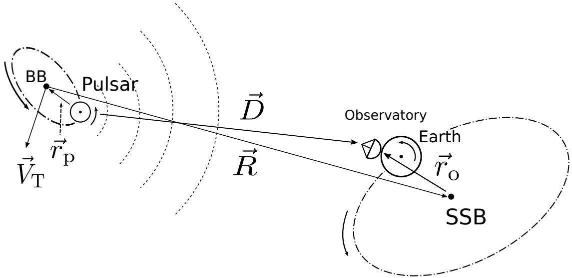


Figure 11

Modeling pulsar rotation

Under the pulsar co-moving frame

$$N(t_\mathrm{e}) = N_0 + \nu_0(t_\mathrm{e} - t_0) + \frac{1}{2}\dot{\nu}_0(t_\mathrm{e} - t_0)^2 + \frac{1}{6}\ddot{\nu}_0(t_\mathrm{e} - t_0)^3 + \ldots, \quad (1)$$

Notation	Parameter	Description
N_0	...	Reference phase
t_0	...	Reference time
t_e	...	Pulse emission time
ν_0	**F0**	Spin frequency
$\dot{\nu}_0$	**F1**	Time derivative of spin frequency
$\ddot{\nu}_0$	**F2**	Second order spin frequency derivative

However, the pulses are observed at the observatory.

$$t_\mathrm{e} = t_\mathrm{obs} - \Delta, \quad (2)$$

Δ is pulse propagation time.

Timing Model (Pulse Propagation)

Modeling pulse propagation time Δ

- Classic
  - Pulsar - Observatory astrometry
  - Interstellar space
  - Pulsar system
- General Relativity
  - Shapiro delay
  - Einstein delay
  - Post Keplerian pulsar motion
  - Gravitational waves

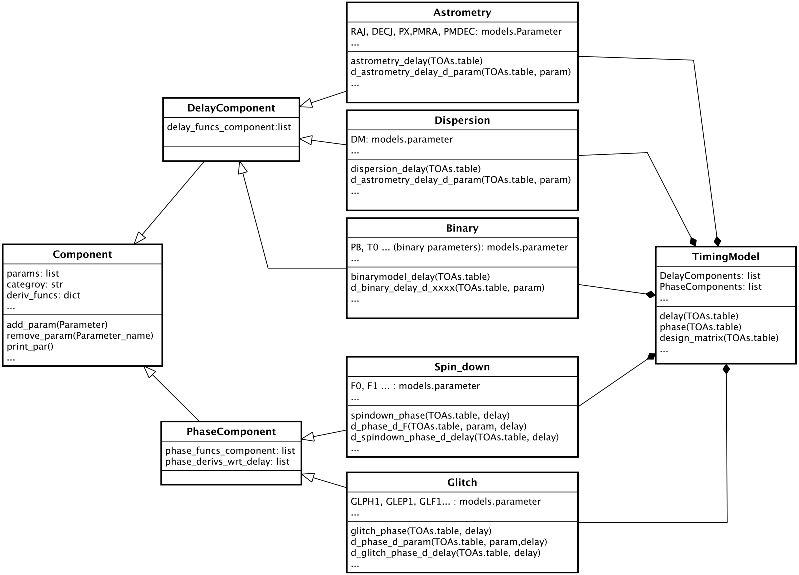
PINT (Models Module)

How to handle the complicated models:
Object Oriented and Modularized

Features of models module

- TimingModel object
  - Centralizes all the model information.
  - Unified API.
- Model components
  - Independently implemented components.
- Model builder
  - Automatically builds TimingModel object from `.par` file

PINT (Models module)



PINT example

```
>>> import pint.models as models
>>> import pint.toa as toa
>>> # Read TOAs
>>> t = toa.get_TOAs("NGC6440E.tim",usepickle=False)
>>> # Read model
>>> m = models.get_model("NGC6440E.par")
>>> # compute phase
>>> phase = m.phase(t)
```

Figure 12: This code example illustrates how to get pulse phase via PINT

Update Timing model

Time residual:

$$R_{\rm time} \equiv t_{\rm obs} - t_{\rm model} \tag{3}$$

In practical:
We compare the model predicted phase to the nearest integer phase number.

$$\begin{aligned} R_{\rm phase} &= N(t_{\rm obs}) - N_i(t_{\rm obs}) \quad (4) \\ R_{\rm time} &= R_{\rm phase}/\nu_0. \quad (5) \end{aligned}$$

The timing model can be updated by fitting the residuals using different fitting method (e.g. Least Square fitting).

PINT (Fitter Module)

A timing model can be updated using different fitting method.

Features of fitter module

- Unified API.
- Object oriented design for fitting algorithm.
- Easy to change fitting method.

Fitter Name	Algorithm	Comments
PowellFitter	Scipy Powell minimizing	...
WlsFitter	Weighted least square fitting	...
GLSFitter	Generalized least square fitting	Noise model incorporated

Table 1: PINT implemented fitting algorithms

PINT update timing model

```
>>> import pint.models
>>> import pint.toa
>>> import pint.residuals as r
>>> import pint.fitter
>>> # Initialize PINT TimingModel object using a Tempo/TEMPO2 style parameter file
>>> m = pint.models.get_model("NGC6440E.par")
>>> # Initialize PINT TOAs object using a Tempo/TEMPO2 style TOAs file
>>> t = pint.toa.get_toas("NGC6440E.tim")
>>> # Create the residuals with a less accurate model
>>> rst = r.resids(t, m).time_resids
>>> # Print out the rms of the residuals.
>>> print("RMS in time is", rst.std().to(u.us))
    RMS in time is 1099.12298871 us
>>> # Updating the model.
>>> # Initialize Fitter object with TimingModel object and TOAs object
>>> f = pint.fitter.WlsFitter(t, m)
>>> # Fit the data and update the model.
>>> chi2 = f.fit_toas()
>>> print("Chi square is : ", chi2)
    Chi square is : 59.5743132766
>>> print("RMS in time is", f.resids.time_resids.std().to(u.us))
    RMS in time is 33.3342862167 us
```

Figure 13: This code example illustrates how to update a timing model using PINT

Applications

What can we learn from pulsar timing? It provides an accurate timing model.

- Pulsar characteristics
- Solar system dynamics
- Pulsar system dynamics
  - Companion stars
  - General relativity tests
- Interstellar medium
- Gravitational waves
- ...

PTAs

A pulsar timing array (PTA) is a set of pulsars which is analyzed to search for correlated Gravitational Wave signatures in the pulse arrival times.

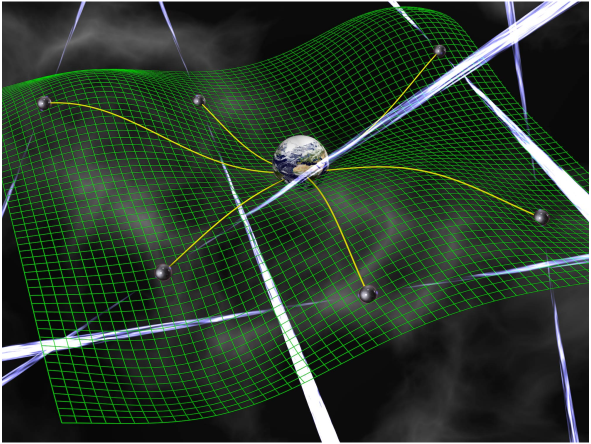


Figure 14: Credit: David Champion

NanoGrav

the North American Nanohertz Observatory for Gravitational Waves (NANOGrav) uses the most sensitive radio telescopes to detective GWs via pulsar timing.

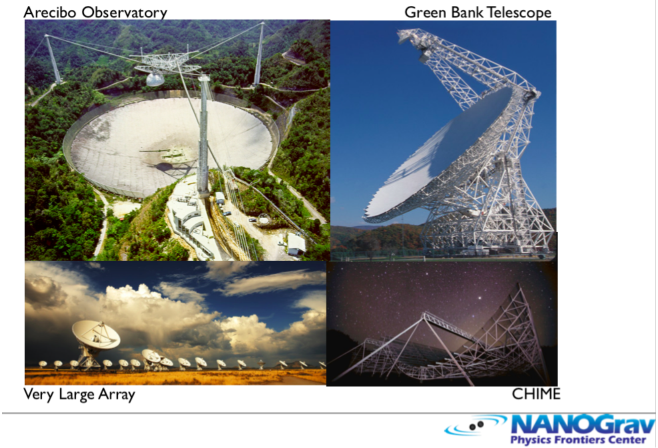


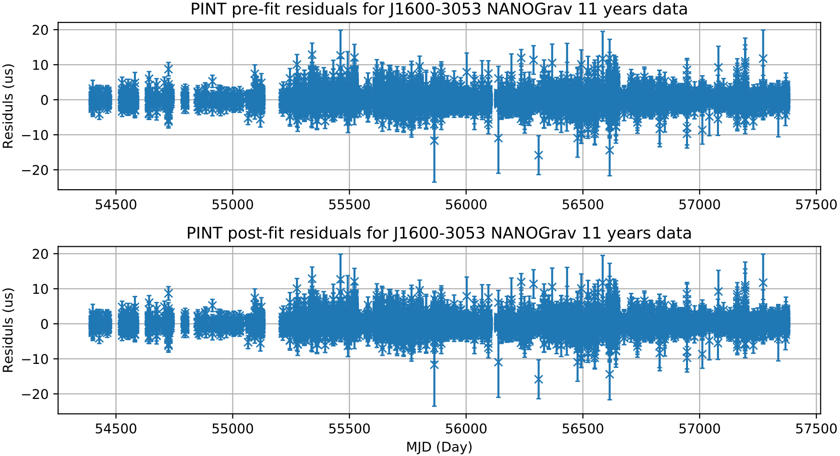
Figure 15: Credit: NanoGrav Telescopes

Current Status

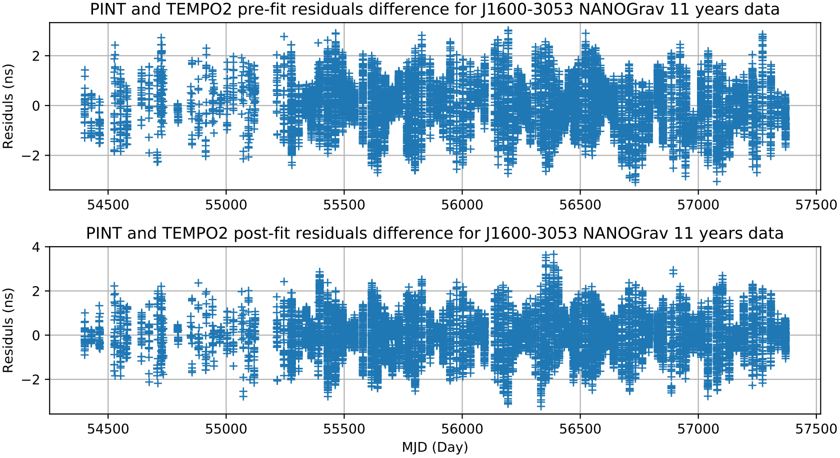
- PINT is able to process NANOGrav 11-year data set.
- A set of high energy data analysis scripts and tools are provided.
- PINT package has been adopted by:
  - NANOGrav collaboration
  - NASA's NICER mission
- PINT is incorporated with gravitational wave analysis pipelines.
- PINT is used to analyze NANOGrav 12.5-year data.

NANOGrav 11-year data example

Result of PSR J1600-3053 NANOGrav 11-year data
WRMS : 0.9610 μs, NTOAs: 12433, χ^2:12265.50



Compare with existing software (PINT vs TEMPO2)



The Future

- PINT enables interactive pulsar data analysis and facilitates the pulsar tools development platform. (a lot of pulsar timing data analysis tools will be based on PINT)
- PINT will be used as the major data analysis tool for NANOGrav 14-year data.
- PINT always welcome new features, new observatories, and Pull request on Github: https://github.com/nanograv/PINT