TempoNest2-doc Documentation

Release 1

A. Berthereau, L. Lentati

Dec 14, 2022

SECTIONS:

TEMPONEST2 is a profile domain pulsar timing analysis method (see Lentati et al., 2017). Python2 based replacement for TempoNest, GPU accelerated (CPU implemented).

The TempoNest2 method fully modelizes the impulsion by using shapelet profiles before sampling the likelihood in the parameters space. By modeling the impulsion, the method can include profile variations from scattering or jitter. The sampling is performed by an Hamiltonian Monte-Carlo method and can be used either on GPU or CPU.

Examples

> The Example directories include ipython notebooks with two analyses, including the TOAs, the parfile and the observations.

CHAPTER

ONE

QUICK START

1.1 Requirements

You will need the following libraries installed :

- PSRCHIVE (see the Installation guide)
- PTMCMCSampler (see the github repository)
- Libstempo (see the repository)

1.2 Installation

You will need to install **Ellipsis**. Instructions for compiling **Ellipsis** can be found in GHS/Ellipsis/README.md. This results in a libellipsis.a file. You then need to create a shared object (.so) file from this file by running:

```
gcc -shared -o libghs.so libellipsis.a -Wl `
```

The location of libghs.so then needs to be added to your LD_LIBRARY_PATH in order for python to find it by :

```
echo LD_LIBRARY_PATH=$LIBRARY_PATH:/path_to_libghs.so/
```

TempoNest 2 can then be installed by running:

```
python setup.py install
```

CHAPTER

TWO

API REFERENCES

You will find below the API reference for TempoNest2 class : Likelihood.

class Likelihood(*useGPU=False*)

__init__(*useGPU=False*)

Initialize the Likelihood class.

Parameters

useGPU (*boolean, optional*) – True to use the GPU for the sampling, or False to use CPU.

loadPulsar(*parfile*, *timfile*, *ToPickle=False*, *FromPickle=False*, *root='Example'*, *iters=1*, *usePreFit=False*)

Load the pulsar using libstempo.

Parameters

- **parfile** (*string*) – Path to the parfile used by libstempo.
- **timfile** (*string*) – Path to the file containing the TOA used by libstempo.
- **ToPickle** (*boolean, optional*) – True to save the profile and info into ProfData.pickle in root folder.
- **FromPickle** (*boolean, optional*) – True to load a pickle object containing profile and info from the file ProfData.pickle in root folder.
- **root** (*string, optional*) – Path to the folder where files will be saved or loaded..
- **iters** (*int, optional*) – Tempo2 numbers of fitting iterations when using libstempo.
- **usePreFit** (*boolean, optional*) – True to use the prefit parameters from libstempo.

TScrunch(*doplot=True*, *channels=None*, *ChanSep=None*, *FreqRange=None*, *FromPickle=False*, *ToPickle=False*, *FromTM=False*)

Scrunch fully in time and in frequency according the parameters channels (or FreqRange).

Parameters

- **doplot** (*boolean, optional*) – True to display the fully time scrunched profile.
- **channels** (*int, optionnal*) – Split the data in the specified number of channels.
- **Freqrange** (*list, optional*) – Split the data into specific frequency range provided, e.g: FreqRange=[[0,900],[900,1800]].
- **ToPickle** (*boolean, optional*) – True to save the profile and info into root-TScrunch-NCHANC.pickle in root folder, where NCHAN is the number of channels after data reduction.
- **FromPickle** (*boolean, optional*) – True to load from a saved profile named root-TScrunch-NCHANC.pickle in root folder.

- **FromTM** (*boolean, optional*) – True to use the previous fit to align profile or use PSRCHIVE default centering

getInitialParams(*MaxCoeff=1*, *fitNComps=1*, *RFreq=1400*, *polyorder=0*, *parameters=None*, *pmin=None*, *pmax=None*, *x0=None*, *cov_diag=None*, *burnin=1000*, *outDir='./Initchains/'*, *sampler='pal'*, *resume=False*, *incScattering=False*, *mn_live=500*, *doplot=False*)

Initial estimate of the mean profile parameters (Phase, Width and NCoeff for each frequency components) using either PTMCMC or MULTINEST sampler and compute the profile model for the mean profile.

Parameters

- **MaxCoeff** (*int, optional*) – The number of shapelets used in profile mode
- **fitNComps** (*int, optional*) – The number of components in the profile model which is different of the MaxCoeff.
- **RFreq** (*float, optional*) – Center frequency of the observations in MHz.
- **polyorder** (*int, optional*) – Degree of the polynomial to fit the shapelet coefficients to the frequencies.
- **parameters** (*list, optional*) – Contains the parameters (phase, width and Ncoeff for each frequency components) to fit. If not provided, a list will be created.
- **pmin** (*list, optional*) – The lower bound for multinest sampling for each parameter. Dimension should be 3xfitNComps
- **pmax** (*list, optional*) – The upper bound for multinest sampling for each parameter. Dimension should be 3xfitNComps.
- **x0** (*list, optional*) – First point to start the sampling, dimension of the parameters (3xfitNComps)
- **cov_diag** (*list, optional*) – Initial covariance of model parameters for PTMCMC, the covariance matrix is built inside the function.
- **burnin** (*int, optional*) – Number of burning point for the sampling.
- **outDir** (*string, optional*) – Path for saving the result.
- **sampler** (*string, optional*) – Name of the sampler to use, 'pal' (PTMCMC) or 'multinest'.
- **resume** (*boolean, optional*) – True to use the saved result from a previous sampling.
- **incScattering** (*boolean*) – True to include the scattering during the estimation.
- **mn_live** (*int, optional*) – Number of multinest living points.
- **doplot** (*boolean, optional*) – To plot the profile model at the end of the process.

FFTInitialLogLike(*x*)

Estimate the loglike of the input parameters or ML shapelet coefficients and their errors based on the input parameters.

Parameters

x (*list*) – Vector of the parameters to estimate.

Returns

- **MLCoeff, MLerrs** (*tuple(arrays)*) – Maximum Likelihood of the shapelet components and their errors, only if the class attribute returnVal is set to 1.
- **loglike** (*float*) – Loglike of the ML parameters, only if the class attribute returnVal is set to 0.

PreComputeFFTShapelets(*interpTime=1*, *MeanBeta=0.1*, *ToPickle=False*, *FromPickle=False*, *doplot=False*, *useNFBasis=0*)

Precompute the FFT shapelet profile for the interpolated TOA to reduce the computing time during the sampling phase, the upperindex defined the lenght of each interpolated profile.

Parameters

- **interpTime** (*float, optional*) – Step for interpolation in ns.
- **MeanBeta** (*float, optional*) – The mean width of the impulsion for the pulse modelization.
- **ToPickle** (*boolean, optional*) – True to save the array of precomputed shapelets.
- **FromPickle** (*boolean, optional*) – True to use the saved shapelets previously computed.
- **useNFBasis** (*float, optional*) – If >0, the upperindex is ignored and then the bin's number is equal to 2*useNFBasis for each profile.

getInitialPhase(*doplot=True*, *ToPickle=False*, *FromPickle=False*)

Update the mean phase by computing the ML of the phase using the full data.

Parameters

- **doplot** (*boolean, optional*) – True to plot the log-likelihood of the phase.
- **ToPickle** (*boolean, optional*) – True to save the mean phase in a pickle object.
- **FromPickle** (*boolean, optional*) – True to use the previously computed mean phase from a pickle object.

FFTPhaseLike(*x*)

Compute the loglike of the phase vector x.

Parameters
x (*list*) – Vector phase of all the data.

Returns
loglike – Loglike of the input vector.

Return type
float

calculateGHSHessian(*diagonalGHS=False*)

Compute the Hessian matrix for GHS.

Parameters
diagonalGHS (*Boolean*) – Return a diagonal matrix based on the EVD.

Returns

- **x0** (*Array*) – Starting points for the GHS.
- **cov_diag** (*Array*) – Diagonal covariant matrix of the parameters.
- **M** (*Array*) – Normalized complex eigen vector, if diagonalGHS is True, then just a 1D array [1].
- **hess_dense** (*Array*) – Hessian of the likelihood.

callGHS(*resume=False*, *nburn=100*, *nsamp=100*, *feedback_int=100*, *seed=-1*, *max_steps=10*, *dim_scale_fact=0.4*)

Call the Guided Hamiltonian Sampler to perform sampling of the pulsar parameters.

Parameters

- **resume** (*boolean, optional*) – To use the previous sampling results saved in the "extract.dat" file.
- **nburn** (*int, optional*) – Number of burnt samples.
- **nsamp** (*int, optional*) – Number of wanted samples.
- **feedback_int** (*int, optional*) – Number of steps between each feedback printed on the screen.
- **seed** (*int, optional*) – The seed used to initiate the random number generator in GHS.
- **max_step** (*int, optional*) – The maximal number of step done during the leapfrog of the hamiltonian sampling.
- **dim_scale_fact** (*float, optional*) – Dimensionality scale factor.

addPNoise(*Fit=True*, *ML=None*, *write=True*)

Add the profiles noise to the model parameters.

Parameters

- **Fit** (*boolean, optional*) – True to fit the profile noise during the sampling process.
- **ML** (*array(float),optional*) – The maximum likelihood values.
- **write** (*boolean, optional*) – True to write the updated parameter into the GHS extract file.

addPAmps(*Fit=True*, *ML=None*, *Dense=False*, *write=True*)

Add the profile amplitudes to the model parameters.

Parameters

- **Fit** (*boolean, optional*) – True to fit the profile amplitudes during the sampling process.
- **ML** (*array(float), optional*) – The maximum likelihood values.
- **Dense** (*boolean, optional*) – True to put the amplitude into the dense parameters for the sampling.
- **write** (*boolean, optional*) – True to write the updated parameter into the GHS extract file.

addPhase(*Fit=True*, *ML=numpy.nan*, *write=True*)

Add the phase to the parameter of the model.

Parameters

- **Fit** (*boolean, optional*) – True to fit the phase during the sampling process.
- **ML** (*array(float), optional*) – The maximum likelihood values. If not provided, the model will use the mean phase.
- **write** (*boolean, optional*) – True to write the updated parameter into the GHS extract file.

addLinearTM(*Fit=True*, *ML=numpy.array*, *write=True*)

Add the linear timing parameters to the model.

Parameters

- **Fit** (*boolean, optional*) – True to fit the TM parameters during the sampling process.

- **ML** (*array, optional*) – The maximum likelihood values.
- **write** (*boolean, optional*) – True to write the updated parameter into the GHS extract file

addProfile(*Fit=True*, *ML=numpy.array*, *write=True*)

Add the profiles to the model parameters

Parameters

- **Fit** (*boolean, optional*) – True to fit the profiles during the sampling process.
- **ML** (*array(float), optional*) – The maximum likelihood values of the profiles.
- **write** (*boolean, optional,*) – True to write the updated parameter into the GHS extract file.

addScatter(*FitScatter=True*, *FitFreqScale=False*, *MLScatter=None*, *MLFreqScale=None*, *mode='parfile'*, *writeScatter=True*, *writeFreqScale=True*, *RefFreq=1*, *Prior=0*, *StepSize=0*)

Add scattering to the pulse model.

Parameters

- **FitScatter** (*boolean, optional*) – True to fit the scattering during the sampling process, it includes the parameter into the Dense parameters.
- **FitFreqScale** (*boolean, optional*) – True to fit the frequency scale parameter during the sampling process.
- **MLScatter** (*array(float), optional*) – Contains the maximum of likelihood value of the scatter parameter.
- **MLFreqScale** (*array(float), optional*) – Contains the maximum of likelihood value of the frequency scale parameter.
- **mode** (*string, optional*) – Choose the mode to apply the scattering : 'parfile', 'flag', 'time'. The 'parfile' mode uses the parfile '**SX_**' prefix to apply the scattering, the 'flag' uses the flags in the TOAs, and time applies the scattering to every observation.
- **writeScatter** (*boolean, optional*) – True to write the scattering parameter into the GHS extract file.
- **writeFreqScale** (*boolean, optional*) – True to write the frequency scale parameter into the GHS extract file.
- **RefFreq** (*float, optional*) – Reference frequency in GHz.
- **Prior** (*float, optional*) – Prior of the scatter parameter.
- **StepSize** (*int, optional*) – The step used for the fit, the hessian value during the sampling will be set to 2^(-stepsize).

addEQUAD(*FitSignal=True*, *FitPrior=True*, *MLSignal=None*, *MLPrior=None*, *mode='flag'*, *flag='sys'*, *model=None*, *Dense=None*, *writeSignal=True*, *writePrior=True*)

Add the EQUAD parameter to the model.

Parameters

- **FitSignal** (*boolean, optional*) – True to fit the EQUAD signal during the sampling process.
- **FitPrior** (*boolean, optional*) – True to fit the prior of the EQUAD signal during the sampling process .

- **MLSignal** (*array(float), optional*) – Contains the maximum of likelihood values of the signal.
- **MLPrior** (*array(float), optional*) – Contains the maximum of likelihood values of the signal.
- **mode** (*string*) – Set the mode to add the ECORR, can either be 'flag' or 'global'. The 'flag' option apply the ECORR to the corresponding flag.
- **flag** (*string*) – The flag used for applying the noise.
- **Dense** (*boolean, optional*) – True to include the parameter into the Dense parameters for the sampling.
- **writeSignal** (*boolean, optional*) – True to write the EQUAD signal into the GHS extract file.
- **writePrior** (*boolean, optional*) – True to write the EQUAD prior into the GHS extract file.

addECORR(*FitSignal=True*, *FitPrior=True*, *MLSignal=None*, *MLPrior=None*, *mode='flag'*, *flag='sys'*, *model=None*, *Dense=None*, *writeSignal=True*, *writePrior=True*)

Add the ECORR noise parameter to the model.

Parameters

- **FitSignal** (*boolean, optional*) – True to fit the ECORR signal during the sampling process
- **FitPrior** (*boolean, optional*) – True to fit the prior of the ECORR signal during the sampling process
- **MLSignal** (*array(float), optional*) – Contains the maximum of likelihood values of the signal.
- **MLPrior** (*array(float), optional*) – Contains the maximum of likelihood values of the signal.
- **mode** (*string*) – Set the mode to add the ECORR, can either be 'flag' or 'global'. The 'flag' option apply the ECORR to the corresponding flag.
- **flag** (*string*) – The flag used for applying the noise.
- **Dense** (*boolean, optional*) – True to include the parameter into the Dense parameters for the sampling.
- **writeSignal** (*boolean, optional*) – True to write the ECORR signal into the GHS extract file.
- **writePrior** (*boolean, optional*) – True to write the ECORR prior into the GHS extract file.

addBaselineNoise(*FitAmpPrior=True*, *FitSpecPrior=True*, *MLAmpPrior=None*, *MLSpecPrior=None*, *writeAmpPrior=True*, *writeSpecPrior=True*, *BaselineNoiseRefFreq=2*, *BaselineNoisePrior=None*)

Add baseline noise to model's parameters.

Parameters

- **FitAmpPrior** (*boolean, optional*) – True to fit the prior of the amplitude during the sampling process.
- **FitSpecPrior** (*boolean, optional*) – True to fit the prior of the spectral index during the sampling process.

- **MLAmpPrior** (*float, optional*) – Contains the maximum of likelihood value of the prior.
- **MLSpecPrior** (*float*) – Contains the maximum of likelihood value of the prior.
- **writeAmpPrior** (*boolean, optional*) – True to write the amplitude prior into the GHS extract file.
- **writeSpecPrior** (*boolean, optional*) – True to write the spectral prior into the GHS extract file.
- **BaselineNoiseRefFreq** (*float, optional*) – The reference frequency for the baseline noise.
- **BaselineNoisePrior** (*array, optional*) – Prior of the baseline noise.

CHAPTER

THREE

TUTORIALS

This is a guide you can follow to start a basic pulsar analysis with TempoNest2. If you want more details on the functions below, see *API references*. More detailed examples can be found in the Examples provided in the original github repository.

3.1 Initialization

To perform the analysis, you will need the pulsar observations, a .tim file and a .par file. You will need to instanciate a Likelihood object for each analysis. The parameter *useGPU* allows you to choose to perform the sampling on GPU or CPU (default is False).

```
import TempoNest as tn
#For CPU-uses
lfunc = tn.Lilekihood(useGPU=False)

#For GPU-uses
lfunc = tn.Lilekihood(useGPU=True)

#Load the data
lfunc.loadPulsar("parfile.par", "timfile.tim", root='./results/-', iters=1)
```

3.2 Scrunch and get initial profile model

To scrunch fully in time, you can use the function **TScrunch**, and indicates the frequency resolution you want to keep for the analysis. The **getInitialParams** function will fit the pulse phase, the width and the shapelets needed to modelize the pulse.

```
lfunc.TScrunch(doplot = False, channels = 1, FromPickle = False, FromTM = True)
lfunc.getInitialParams(pmin = [-0.5, -2, 0], pmax = [0.5, -1, 2], MaxCoeff = 100,␣
↪resume=True, outDir = './Init/', sampler='multinest', incScattering = False, mn_live =␣
↪1000,  fitNComps = 1, doplot = False)
```

3.3 Precompute the shapelet interpolation matrix

To reduce the computing time, TempoNest2 interpolates the shapelet profiles (see L. Lentati, 2016). These shapelet profiles will then be fitted during the sampling by adjusting the model parameters.

```
lfunc.PreComputeFFTShapelets(interpTime = 2, MeanBeta = lfunc.MeanBeta, doplot=False)
```

3.4 Set up the model

You need to add parameters to the model before the sampling. The example below includes adding the profile noises, the profile amplitudes, the phase, the timing parameters and the profiles. If you do not add these parameters to the model, they will not be fitted during the sampling and may lead to a bad performance.

```
#Add some parameters to the model
lfunc.addPNoise(Fit = True, write=True)
lfunc.addPAmps(Fit = True, Dense=False, write=True)
lfunc.addPhase(Fit = True, ML=np.array([0]))
lfunc.addLinearTM(Fit = True)
lfunc.addProfile(Fit = True, ML = lfunc.MLShapeCoeff[1:,0].flatten())
```

INDEX

Symbols

A

C

F

G

L

P

T