TempoGravity Documentation

Amodio Carleo

June 29, 2025

1 Overview

`TempoGravity` is the most easy-to use Python-based script meticulously crafted for executing Bayesian parameter estimation on pulsar timing data. It leverages the power of the `emcee` library, a renowned Markov Chain Monte Carlo (MCMC) sampler, to explore the multi-dimensional parameter space of pulsar timing models. At its core, the script establishes a crucial interface with the widely-used `tempo2` pulsar timing package. This connection allows `TempoGravity` to dynamically generate pulsar parameter files (`.par`), execute `tempo2` to compute timing residuals for these parameters against observed Times of Arrival (TOAs), and subsequently calculate the likelihood of each parameter set. A key strength of this script lies in its flexibility regarding prior information: it can incorporate standard uniform priors derived directly from the uncertainties listed in the `.par` file and also allows for the inclusion of additional, potentially more restrictive, uniform priors or definitions of theoretical parameters through equations provided in a separate input file. This hybrid approach to prior specification offers users greater control and the ability to test specific astrophysical hypotheses, as well as alternative theories of gravity through specific equations for the post-Keplerian (PK) parameters.

2 Features

- `.par` **File Parsing:** Reads and interprets pulsar parameters, their initial values, and associated uncertainties from a standard `tempo2` `.par` file. This forms the baseline for the parameters to be explored.
- `.tim` **File Handling:** Processes timing data from a standard `tempo2` `.tim` file, extracting crucial information such as Modified Julian Dates (MJDs), TOA uncertainties, and observatory/backend identifiers necessary for residual calculation and uncertainty scaling.
- **Prior Extraction from** `.par`**:** Automatically extracts uniform priors for parameters marked with a fitting flag of `1` in the `.par` file. The width of these priors is determined by a user-defined multiplier (`delta`) applied to the listed uncertainties, creating a range around the initial parameter value.

- **External Prior and Equation Specification:** Supports an optional external file (`theory_priors.txt`) to define additional uniform priors and complex theoretical parameter relationships via mathematical equations. Priors specified in this file take precedence over those derived from the `.par` file, providing a mechanism for incorporating external knowledge or focusing the parameter space exploration.

- **Derived Parameter Computation:** Utilizes the `sympy` library to symbolically evaluate equations provided in the `theory_priors.txt` file. This allows for the calculation of derived parameters (e.g., companion mass, orbital inclination) based on the sampled fundamental parameters during each MCMC step.

- **MCMC Sampling with `emcee`:** Employs the `emcee` library to perform efficient MCMC sampling of the parameter posterior distribution. This method explores the parameter space probabilistically, yielding a set of samples that represent the relative probability of different parameter combinations given the data and priors.

- `tempo2` **Interface:** Seamlessly integrates with `tempo2` by dynamically generating temporary `.par` files for each MCMC sample, executing `tempo2` with the relevant `.tim` file, and capturing the resulting timing residuals.

- **Log-Likelihood Calculation:** Computes the log-likelihood of each parameter set based on the calculated timing residuals and their corresponding uncertainties, including the application of `T2EFAC` corrections derived from the MCMC sample. The likelihood function quantifies how well the model (defined by the parameters) fits the observed data. The log-likelihood is calculated as:

$$\ln \mathcal{L} = -\frac{1}{2} \sum_{i=1}^{N} \left(\frac{r_i}{\sigma_i \cdot \mathrm{EFAC}_i} \right)^2 - \sum_{i=1}^{N} \ln(\sigma_i \cdot \mathrm{EFAC}_i)$$

  where:

  - N is the total number of TOAs
  - r_i is the i-th timing residual
  - σ_i is the reported uncertainty for the i-th TOA from the `.tim` file
  - EFAC_i is the EFAC parameter corresponding to the backend i that recorded the i-th TOA. The EFACs are sampled parameters in the MCMC.

- **Posterior Visualization:** Generates insightful corner plots using `corner.py`, visualizing the 1D marginalized probability distributions for individual parameters and the 2D joint probability distributions for pairs of parameters. Users can specify a subset of parameters to plot for clarity.

- **Posterior Sample Output:** Saves the complete set of MCMC samples for all sampled and derived parameters to a text file (`full_posterior_samples.txt`). This allows for post-processing, detailed analysis,

or further use of the posterior distribution.

- **Best-Fit Value Reporting:** Calculates and prints the median and 1σ uncertainties (derived from the 16th and 84th percentiles) for both sampled and derived parameters, providing a concise summary of the MCMC results.
- **Robust Error Handling:** Includes basic error handling for `tempo2` execution failures, automatically saving the problematic `.par` files to a designated directory (`failed_pars`) for later inspection and debugging.

3 Dependencies

This script relies on several key Python libraries to perform its tasks:

- `numpy`: Essential for numerical operations, array manipulation, and mathematical functions.
- `math`: Provides standard mathematical functions.
- `subprocess`: Used to run external commands, specifically the `tempo2` executable.
- `os`: For interacting with the operating system, such as managing file paths and directories.
- `emcee`: The core library for running the MCMC sampler.
- `corner`: Used to generate the corner plots for visualizing the posterior distribution.
- `matplotlib`: The plotting library used by `corner.py`.
- `multiprocessing`: Enables parallel execution of the MCMC sampling across multiple CPU cores, significantly speeding up the process.
- `tempfile`: Used to create temporary files for the `.par` and `tempo2` output, ensuring a clean working environment.
- `ast`: (Abstract Syntax Trees) Potentially used for parsing or evaluating expressions, although `sympy` is the primary tool for equations here.
- `warnings`: For issuing non-fatal warnings during execution.
- `sympy`: A powerful library for symbolic mathematics, used here to define and evaluate theoretical parameter equations.
- `random`: For generating random numbers, used in initializing walker positions.
- `re`: Regular expression operations, likely used for parsing specific patterns in input files.

In particular, the code has been test with the following versions: `numpy 2.2.3`, `corner 2.2.3` and `matplotlib 3.10.1`. Crucially, the script is designed to work in conjunction with the `tempo2` pulsar timing software. Therefore, `tempo2` must be installed on your system and its executable must be accessible from the command line environment where you run the Python script. Without a functional `tempo2` installation, the script will not be able to compute residuals and thus cannot perform the likelihood calculations necessary for the MCMC.

4 Installation

To set up and run `TempoGravity`, follow these steps:

1. **Install Python:** Ensure you have a compatible version of Python 3.x installed on your system. You can download it from the official Python website or use a package manager.

2. **Install Python Dependencies:** Open your terminal or command prompt and install the required Python libraries using the `pip` package installer:

   ```
   >> pip install numpy emcee corner matplotlib sympy
   ```

   This command will download and install the latest versions of these libraries and their own dependencies.

3. **Install Tempo2:** Install the `tempo2` pulsar timing package. The installation process for `tempo2` can vary depending on your operating system and preferred method (e.g., compiling from source, using a package manager). Please consult the official `tempo2` documentation for detailed, platform-specific installation instructions. After installation, verify that the `tempo2` executable is in your system's PATH by typing `tempo2` in your terminal; it should either run or display a help message. If not, you may need to manually add the `tempo2` binary directory to your system's PATH environment variable.

5 Input Files

5.1 `.par` File

The primary input parameter file is a standard `tempo2 .par` file. This file defines the pulsar's timing model, including parameters like spin frequency (`F0`), spin-down rate (`F1`), astrometric parameters (`RAJ`, `DECJ`, `PMRA`, `PMDEC`), binary parameters (`PB`, `A1`, `ECC`, `OM`, `T0`), and various other model components. The script reads these parameters and, importantly, their associated uncertainties (typically in the 4th column). Parameters that have a fitting flag set to `1` in the 3rd column are automatically identified as potential candidates for sampling in the MCMC. The script includes specific parsing logic to correctly handle the sexagesimal formats for `RAJ` and `DECJ`, as well as the structure of `JUMP` and `T2EFAC` lines.

5.2 `.tim` File

The timing data is provided in a standard `tempo2` `.tim` file. This file lists the observed Times of Arrival (TOAs) for the pulsar. Each TOA entry typically includes the observation frequency, the TOA itself (usually in MJD), the uncertainty on the TOA, the telescope or observatory, and the backend used for the observation. The script parses this file to obtain the TOA MJDs and their uncertainties. The backend information is particularly important as it is used in conjunction with the `T2EFAC` parameters sampled by the MCMC to correctly scale the uncertainties for the likelihood calculation.

5.3 `theory_priors.txt` (Optional)

This optional text file provides a powerful way to augment the prior information and introduce theoretical relationships between parameters. Each non-commented, non-empty line in this file should follow one of two formats:

- **Uniform Prior Definition:**

  ```
  PARAMETER_NAME lower_bound upper_bound
  ```

  This format allows you to specify a strict uniform prior range for any parameter, whether it's a standard parameter from the `.par` file or a new parameter you wish to introduce. If a parameter listed here also has a fitting flag in the `.par` file, the prior defined in `theory_priors.txt` will *override* the prior derived from the `.par` file uncertainty. This is useful for incorporating external constraints or testing specific hypotheses. For example:

  ```
  PB 0.12345 0.12355
  GAMMA 0.001 0.005
  ```

- **Theoretical Parameter Equation:**

  ```
  DERIVED_PARAMETER_NAME = equation
  ```

  This format allows you to define new parameters that are not directly sampled but are instead calculated as a function of the sampled parameters. The `equation` should be a valid mathematical expression that can be parsed and evaluated by the `sympy` library. The variables in the equation should correspond to the names of the parameters being sampled (either from the `.par` file or defined with uniform priors in this

file). For example, to calculate the mass of a binary companion (`MC`) based on the sampled pulsar mass (`MP`), orbital period (`PB`), and projected semi-major axis (`A1`), you could include:

```
MC = (MP*PB)**(1/3) / (2*pi*G_NEWTON)**(1/3)
```

In this example, `MP`, `PB`, `A1`, π, and G_{Newton} would need to be either sampled parameters or defined as constants accessible within the script or the `theory_priors.txt` file (though typically fundamental constants like π and G would be defined within the script as they are). The script uses `sympy` to handle the symbolic evaluation of these equations for each MCMC sample.

Lines starting with `#` are treated as comments and are ignored by the parser. Empty lines are also skipped.

6 Priors

Prior probability distributions are a fundamental component of Bayesian inference, representing our knowledge or assumptions about the parameters before considering the observed data. In `TempoGravity`, uniform priors are used for all sampled parameters. The script provides two mechanisms for defining these uniform priors:

6.1 Priors from the `.par` file

For parameters listed in the input `.par` file that have their fitting flag set to `1` (in the third column), the script automatically constructs a uniform prior distribution. The center of this prior is the parameter's initial value given in the `.par` file. The width of the uniform prior is determined by the parameter's uncertainty (in the fourth column) multiplied by the user-defined `delta` value. The prior range for such a parameter P with initial value V_P and uncertainty σ_P is $[V_P - \delta \times \sigma_P, V_P + \delta \times \sigma_P]$, where δ is a user-defined multiplicative factor (see below). This approach leverages the existing uncertainty information in the `.par` file to define a reasonable initial search space for the MCMC. About priors for `JUMP` and `T2EFAC` parameters, they are treated as uniform distributions, based on the contents of the `.par` file. For `T2EFAC` parameters, which scale the TOA uncertainties for each backend, the prior is always set as:

$$\text{T2EFAC}_{\text{backend}} \sim \mathcal{U}(0.1,\ 2.0)$$

This prior is applied if a corresponding `T2EFAC` line exists in the `.par` file, *regardless of the presence of the fit flag 1.* If no such line is found for a given backend, the code assumes a fixed EFAC of 1.0 without performing any sampling. For `JUMP` parameters, which model phase offsets for selected subsets of TOAs, each *active* `JUMP` (i.e., with fit flag set to 1) is identified and assigned a prior of the form:

$$\text{JUMP}_{\text{backend}} \sim \mathcal{U}\left(v - \delta \cdot 0.1|v|,\ v + \delta \cdot 0.1|v|\right)$$

where v is the JUMP value specified in the `.par` file. Notice that the assumed format of the `.tim` file is:

Name, Freq, TOA, uncertainty, telescope, -i, backend

therefore the name of the backend is extracted from column 7 of the `.tim` file, i.e. **backend = parts[6]** by default. Therefore, if the backend location is different, please change this number in the `load_residulas()` function.

6.2 Priors from `theory_priors.txt`

The optional `theory_priors.txt` file provides a way to specify additional uniform priors explicitly. For any parameter listed in this file with a defined lower and upper bound (using the format `PARAMETER_NAME lower_bound upper_bound`), a uniform prior is set for that parameter over the specified range.

Precedence: It is important to note that priors defined in `theory_priors.txt` take precedence over those derived from the `.par` file. If a parameter is listed in both the `.par` file with a fitting flag of `1` and in the `theory_priors.txt` file with explicit bounds, the prior from `theory_priors.txt` will be used for the MCMC sampling. This allows users to override the default `.par`-based priors with more specific or restrictive priors based on external information or assumptions.

6.3 Application in MCMC

These defined uniform priors are used in two key ways during the MCMC sampling:

1. **Walker Initialization:** The initial positions of the `emcee` walkers are generated by randomly sampling a value for each parameter from within its defined uniform prior range. This ensures that the starting points of the MCMC chains are within the plausible parameter space according to the specified priors.

2. **Log-Probability Evaluation:** Within the `log_prob` function, before calculating the likelihood, the script checks if the current parameter values being proposed by `emcee` fall within their respective uniform prior bounds. If any parameter value is outside its defined prior range, the `log_prob` function immediately returns $-\infty$ (negative infinity). This effectively assigns zero probability to any parameter combination outside the prior volume, ensuring that the MCMC sampler explores only the region of parameter space allowed by the priors.

Theoretical parameters defined by equations in `theory_priors.txt` do not have priors themselves, as their values are deterministically calculated from the sampled parameters. The priors apply only to the parameters that are directly sampled by the MCMC.

7 Usage

Once the script and its dependencies (including `tempo2`) are installed, you can run it from your terminal. The basic command structure is as follows:

```
>> python3 TempoGravity your_pulsar.par your_timing_data.tim [theory_priors.txt]
```

- `your_pulsar.par`: Replace this with the actual path to your pulsar parameter file (a standard `tempo2 .par` file). This file contains the initial timing model and parameter uncertainties.
- `your_timing_data.tim`: Replace this with the actual path to your pulsar timing data file (a standard `tempo2 .tim` file). This file contains the observed TOAs.
- `[theory_priors.txt]`: This argument is optional. If you have a file containing additional uniform priors or theoretical parameter equations, provide its path here. If you do not wish to use this feature, simply omit this argument.

For example, if your files are named `J1713+0747.par`, `J1713+0747.tim`, and you have a theory file named `my_binary_priors.txt` in the same directory, you would run:

```
>> python3 TempoGravity J1713+0747.par J1713+0747.tim my_binary_priors.txt
```

If you only want to use the information from the `.par` and `.tim` files (i.e., no additional theory priors or derived parameters), you would run:

```
>> python3 TempoGravity J1713+0747.par J1713+0747.tim
```

The script will then proceed to set up the MCMC, run the sampler, and output the results and plots.

8 Script Functions

The script is organized into several functions, each responsible for a specific task in the pulsar timing and MCMC process:

- `hms_to_seconds(hms)`: Converts a Right Ascension string in the format "hh:mm:ss.sssss" into a total number of seconds.
- `seconds_to_hms(seconds)`: Performs the reverse conversion, taking a total number of seconds and formatting it back into the "hh:mm:ss.sssss" string format suitable for writing to a `.par` file.
- `dms_to_arcseconds(dms)`: Converts a Declination string in the format "dd:mm:ss.sssss" (or "dd mm ss.sssss") into a total number of arcseconds. Handles the sign of the declination correctly.

- `arcseconds_to_dms(arcseconds)`: Converts a total number of arcseconds back into the "dd:mm:ss.sssss" string format for writing to a `.par` file, preserving the sign and formatting.
- `read_theory_priors(theory_priors_file)`: Parses the optional `theory_priors.txt` file to read uniform priors and equations.
- `get_priors(par_file, delta, theory_priors_file=None)`: Extracts priors from the `.par` file and combines them with (overriding) priors from the theory file. Identifies sampled and theoretical parameters.
- `compute_derived_parameters(sampled_params, theoretical_params)`: Computes the values of derived parameters based on the current sample of sampled parameters and the defined equations. Handles potential errors like division by zero or non-real results.
- `sample_from_priors(priors)`: Generates an initial position for a single MCMC walker by sampling randomly from the defined uniform priors.
- `write_new_par_file(par_file, new_par_file, sampled_params, theoretical_params)`: Creates a temporary `.par` file with the parameter values from the current MCMC sample (both sampled and derived). Preserves the original file structure.
- `run_tempo2(par_file, tim_file, output_file="residuals.dat")`: Executes the `tempo2` command to generate residuals. Saves failed `.par` files to a `failed_pars` directory.
- `compute_likelihood(residuals, uncertainties)`: Calculates the log-likelihood of the given residuals and uncertainties, including EFAC corrections.
- `load_residuals(residual_file, tim_file, new_par_file)`: Loads residuals from the `tempo2` output file and matches them with uncertainties from the `.tim` file, applying `T2EFAC` corrections.
- `log_prob(theta, par_file, tim_file, param_names, theoretical_params, priors)`: The core log-probability function for `emcee`. It updates the `.par` file, runs `tempo2`, loads residuals, and returns the log-likelihood. Includes prior checks and handling for invalid derived parameters.
- `run_emcee(par_file, tim_file, n_samples, n_walkers, n_threads, delta, theoretical_params)`: Sets up and runs the `emcee` sampler. Initializes walker positions and manages the multiprocessing pool.
- `plot_posterior(posterior, derived_posterior, param_names)`: Generates a corner plot of the posterior distribution for the `selected_params`.

9 Output

Upon successful execution, the script generates several outputs:

- `residuals.dat`: (temporary) Output file from `tempo2` containing residuals.
- `sample_1.par`, `sample_2.par`, **... (up to 5)**: Copies of the first few generated `.par` files are saved in the current directory for debugging.
- `failed_pars/failed_.par`: If `tempo2` fails for a specific parameter set, the corresponding `.par` file is saved in a `failed_pars` subdirectory.
- `full_posterior_samples.txt`: A text file containing all the MCMC samples for both sampled and derived parameters. The first line is a header listing the parameter names.
- **Terminal Output**: Prints information about sampled and derived parameters, CPU threads, MCMC progress, best-fit values, execution time, and any warnings or errors.
- **Corner Plot**: A graphical display of the 1D and 2D marginalized posterior distributions for the `selected_params`.

10 Error Handling and Debugging

The script incorporates several mechanisms to handle potential errors and aid in debugging:

- **File Operation Errors:** `try...except` blocks catch errors during file reading/writing.
- `tempo2` **Execution Errors:** Checks `tempo2` return code, prints error output, and saves failed `.par` files to `failed_pars`.
- **Invalid Uncertainties:** Checks for non-positive uncertainties and handles potential mathematical errors in likelihood calculation.
- **Mismatched TOAs/Residuals:** Checks if residual MJDs match TOA MJDs in the `.tim` file.
- **Missing** `T2EFAC`**:** Checks for `T2EFAC` values for each backend and assumes 1.0 if missing (with a warning).
- **Invalid Derived Parameters:** Uses `sympy` to detect issues like division by zero or non-real results during equation evaluation and rejects the sample.
- **Negative** `T2EFAC` **in** `.par` **Writing:** Explicitly checks for negative sampled `T2EFAC` values and rejects the iteration if found.
- **Temporary File Cleanup:** Attempts to remove temporary files after use.

11 Example

To illustrate the usage, let's assume you have the following files in your working directory: `my_pulsar.par`, `my_timing_data.tim`, and an optional `my_theory.txt` file.

To run the script using all three files:

```
>> python3 TempoGravity.py my_pulsar.par my_timing_data.tim my_theory.txt
```

Or, without the theory priors file:

```
>> python3 TempoGravit.py my_pulsar.par my_timing_data.tim
```

The script will then print information, run the MCMC, print results, and display the corner plot.

12 `TempoGravity` vs. `TEMPONEST`

`TEMPONEST` and `TempoGravity` are both tools designed for Bayesian analysis of pulsar timing data, but they take fundamentally different approaches in how they perform inference and integrate with the pulsar timing engine TEMPO2. `TEMPONEST` is a sophisticated, purpose-built tool that uses the MultiNest nested sampling algorithm to explore the full posterior distribution of timing model and noise parameters. It tightly integrates with `Tempo2` via a plugin interface, allowing it to directly query `Tempo2`'s timing predictions and design matrix at each sample point. This tight coupling enables it to model both deterministic parameters (like spin and orbital elements) and stochastic components such as red timing noise and dispersion measure variations. Crucially, `TEMPONEST` also computes the Bayesian evidence for each model it samples, which allows it to perform principled model comparison — for example, to decide whether the inclusion of a red noise component is justified. Its design prioritizes statistical rigor, enabling fully joint inference over all relevant parameters with options for analytic marginalization of linear terms.

By contrast, `TempoGravity` is a lightweight, Python-based framework that uses Markov Chain Monte Carlo (MCMC) via the emcee package. Rather than using a plugin or shared memory interface to `Tempo2`, it calls `Tempo2` externally as a subprocess for each sample, passing it an updated parameter file and reading back timing residuals. This makes the integration looser but easier to audit and modify. `TempoGravity` does not compute Bayesian evidence, so it cannot formally compare different models, but it excels in flexibility. Notably, it allows users to define symbolic expressions for derived parameters (such as relativistic post-Keplerian parameters) via sympy, making it particularly useful for theory-testing applications. All priors are handled as uniform distributions, typically extracted from .par files or an optional theory priors file, which also defines the algebraic relationships among parameters.

In essence, `TEMPONEST` is designed for comprehensive statistical analysis and large-scale timing projects where

full Bayesian inference and model selection are required. `TempoGravity`, on the other hand, is better suited to exploratory or theory-driven investigations where ease of customization, symbolic manipulation, and transparency are more important than formal model comparison.

12.1 Core Algorithmic Differences

MCMC (as in `TempoGravity` with `emcee`): MCMC methods construct a Markov chain in parameter space such that, after a "burn-in" phase, the samples drawn from the chain are representative of the target posterior probability distribution $P(\theta|D, I) \propto L(D|\theta, I)P(\theta|I)$, where θ are the parameters, D is the data, I is background information, L is the likelihood, and $P(\theta|I)$ is the prior. The chain moves by proposing new parameter values and accepting or rejecting them based on the Metropolis-Hastings ratio, which compares the posterior probability at the proposed and current locations. The `emcee` library implements an affine-invariant ensemble sampler, where an ensemble of walkers proposes new positions by interacting with other walkers in the ensemble. This can help in efficiently sampling posteriors with complex correlations.

Nested Sampling (as in `TEMPONEST`): Nested Sampling, in contrast, is designed to calculate the Bayesian Evidence, $Z = \int L(\theta)P(\theta)d\theta$, which is the integral of the likelihood over the prior volume. It achieves this by sampling from a series of nested likelihood contours. The algorithm maintains a set of "live points" initialized from the prior. In each iteration, the live point with the lowest likelihood $L_{\min}$ is replaced by a new point sampled from the prior volume subject to the constraint that its likelihood must be greater than $L_{\min}$. This process iteratively shrinks the prior volume while exploring regions of increasing likelihood. The Evidence is computed by summing the likelihoods of the "discarded" points, weighted by the estimated prior volume they represent. Samples from the posterior distribution are obtained as a byproduct of this process.

12.2 Primary Goals and Outputs

MCMC (`TempoGravity`): The primary goal is parameter estimation. The main output is a set of samples from the posterior distribution. From these samples, one can calculate marginalized posterior distributions for individual parameters, estimate best-fit values (e.g., median), and determine uncertainties (e.g., using percentiles). While the Evidence can be estimated from MCMC samples, it is not a direct or typically highly accurate output.

Nested Sampling (`TEMPONEST`): The primary goal is Bayesian model comparison, which relies on the accurate calculation of the Bayesian Evidence. The Evidence quantifies how well a given model is supported by the data, integrated over all possible parameter values within that model. Nested Sampling provides the Evidence directly as part of its computation. Samples from the posterior distribution are also produced, allowing for parameter estimation, but this is often considered a secondary output compared to the Evidence.

12.3 Handling of Complex Posteriors

MCMC (`TempoGravity`): While ensemble samplers like `emcee` are more robust than basic random-walk MCMC, they can still face challenges with highly multimodal posterior distributions. Walkers might converge to only one mode, potentially missing other significant regions of high probability in the parameter space. Diagnosing convergence and ensuring all modes are explored can require careful tuning and analysis.

Nested Sampling (`TEMPONEST`): Nested Sampling is generally more effective at exploring multimodal posteriors. By sampling from likelihood contours, it naturally discovers and characterizes different modes as it progresses towards higher likelihood regions. This makes it a preferred method when there is a possibility of multiple distinct solutions in the parameter space.

12.4 Computational Considerations

MCMC (`TempoGravity`): The computational cost for parameter estimation with MCMC generally scales reasonably well with the number of parameters, especially with parallelization. The number of steps required for convergence can vary depending on the posterior's complexity.

Nested Sampling (`TEMPONEST`): While efficient for Evidence calculation, the computational cost of Nested Sampling can become significant in very high-dimensional parameter spaces compared to MCMC focused solely on parameter estimation. The number of "live points" needed often increases with dimension to adequately sample the shrinking prior volume.

12.5 Implications for Pulsar Timing Analysis

The choice between using an MCMC approach (like your script) and a Nested Sampling approach (like `TEMPONEST`) in pulsar timing depends on the specific scientific question. (a) If the primary goal is to obtain precise estimates and uncertainties for the parameters of a known timing model (e.g., refining binary parameters), a well-tuned MCMC can be very efficient. (b) On the other hand, if the goal is to compare different timing models (e.g., deciding whether the data favors a model with a companion over a single-pulsar model, or comparing different gravitational wave background models), the Bayesian Evidence provided by Nested Sampling is the key metric, making tools like `TEMPONEST` more suitable. (c) For complex posterior landscapes with multiple plausible parameter regions (multimodality), Nested Sampling can offer a more robust exploration and a more reliable estimate of the total posterior volume (related to the Evidence).