Time series models are ubiquitous in science, arising in any situation where researchers seek to understand how a system’s behaviour changes over time. A key problem in time series modelling is inference; determining properties of the underlying system based on observed time series. For both statistical and mechanistic models, inference involves finding parameter values, or distributions of parameters values, which produce outputs consistent with observations. A wide variety of inference techniques are available and different approaches are suitable for different classes of problems. This variety presents a challenge for researchers, who may not have the resources or expertise to implement and experiment with these methods. PINTS (Probabilistic Inference on Noisy Time Series — https://github.com/pints-team/pints) is an open-source (BSD 3-clause license) Python library that provides researchers with a broad suite of non-linear optimisation and sampling methods. It allows users to wrap a model and data in a transparent and straightforward interface, which can then be used with custom or pre-defined error measures for optimisation, or with likelihood functions for Bayesian inference or maximum-likelihood estimation. Derivative-free optimisation algorithms — which work without harder-to-obtain gradient information — are included, as well as inference algorithms such as adaptive Markov chain Monte Carlo and nested sampling, which estimate distributions over parameter values. By making these statistical techniques available in an open and easy-to-use framework, PINTS brings the power of these modern methods to a wider scientific audience.

**Keywords:** Time series models; non-linear optimisation; MCMC sampling; nested sampling; Bayesian inference; Python

**Funding statement:** M.C., G.R.M. and D.J.G. acknowledge support from the UK Biotechnology and Biological Sciences Research Council [BBSRC grant number BB/P010008/1]; M.R., S.G. and D.J.G. gratefully acknowledge research support from the UK Engineering and Physical Sciences Research Council Cross-Disciplinary Interface Programme [EPSRC grant number EP/I017909/1]; C.L.L. acknowledges support from the Clarendon Scholarship Fund, the EPSRC and the UK Medical Research Council (MRC) [EPSRC grant number EP/L016044/1]; B.L. acknowledges support from the UK Engineering and Physical Sciences Research Council [EPSRC grant number EP/F500394/1]; and S.G. and G.R.M. acknowledge support from the Wellcome Trust & Royal Society [Wellcome Trust grant numbers 101222/Z/13/Z and 212203/Z/18/Z].

(1) Overview

**Introduction**

Time series models are common in science, where they are used to describe the dynamics of system behaviours. In many cases, these models are non-linear and impossible to solve analytically, so that the forward problem (predicting the model output for a given set of parameters) is computationally hard. For such models, there is no single method which can reliably solve the inverse problem of estimating parameter values from a noisy time trace. Much like there is a variety of forward models, there is a diversity of approaches for parameter inference. Further, it is often unclear which approach to apply when, meaning that researchers are required to implement a range of methods before successfully fitting their model to data. PINTS is a software framework that allows users to easily trial and apply different inference methods to their
problem. The inference methods supplied by PINTS fall into two broad categories: optimisers, which attempt to find a single best parameter vector, and samplers, which aim to estimate a probability distribution over parameter values that are compatible with observed results. Users are expected to already have a forward model (for example, a simulation) at their disposal, which they make available to PINTS by writing a simple Python wrapper. They then define a Problem (a forward model plus a data set), on which either an ErrorMeasure (for optimisation) or a LogPDF (for optimisation and sampling) is defined. Currently available optimisers include CMA-ES [6], XNES [5], SNES [21], and Particle Swarm Optimisation (PSO) [12]. Sampling methods include Random Walk Metropolis Markov chain Monte Carlo (MCMC) [13, 15], adaptive covariance MCMC [10], Population MCMC [8], Differential Evolution [23], DREAM [25], emcee [3], Hamiltonian MCMC [17], and MALA [4]. In addition, elliptoidal [16] and rejection nested samplers [22] are provided. Convenience plotting methods are provided to quickly visualise the results, as well as diagnostic tools to inspect the validity of the results. An example of an optimisation problem and its solution using PINTS is shown in Figure 1.

PINTS was developed as a community effort by researchers in electrochemistry, cardiac electrophysiology, and statistics, to compare different methods for solving inverse problems in a common framework. It features a clean and transparent object-orientated API that is assumed expensive) model, as well as a wide variety of optimisers and samplers. In contrast to the PINTS Python API, Dakota uses either a C++ or file input/output process for communication between user models and the library, and does not provide options for specifying either the error measures or the log-likelihoods of the inverse problem. However, Dakota has some features not yet available in PINTS, such as the option to train a surrogate model, useful for very expensive model evaluations.

Other software packages that enable parameter inference and sampling for ODE models include BioBayes [26], ABC-SysBio [14], SYSBIONS [9], PyMC3 [20], and

![Figure 1:](image)

**Figure 1:** (Left) An experimentally measured noisy time series, and a simulated one. (Right) An example of an optimisation procedure with PINTS. Note that the actual simulation code is omitted from the example model wrapper at the top; this is the user-provided part, and can be written in Python or any other language that can interface with Python, allowing computationally heavy forward simulations to be handled entirely outside of PINTS. This image, and the full example code, can also be found in the PINTS repository.
Stan [2]. These packages use either a common model description format (for example, SBML) or their own language (for example, Stan’s probabilistic programming language) to specify the model, presenting additional learning hurdles for a user and often restricting the class of models which can be fit. By contrast, PINTS aims to be as general as possible to support a wider variety of models (for example, PDEs). PyMC3 [20] does provide a similar generic model interface to PINTS but, as with the other packages, specialises in one sampling method, whereas PINTS aims to support a wide variety of methods with the assumption that no one sampling method is suitable for all models of interest. BCM [24] offers both a generic interface (via C++) and a wide variety of samplers, but does not supply any likelihood functions and is unfortunately largely undocumented.

Implementation and architecture
PINTS is designed around two core ideas: 1. PINTS should work with a wide range of time series models, and make no demands on how they are implemented other than a minimal input/output interface. 2. It is assumed that model evaluation (simulation) is the most costly step in any optimisation or sampling routine.

The decision to use Python fits both these criteria: Python interfaces well with C and C++, which are typically used for high-performance simulation, and any performance hit of using the high-level, easy to read and write language Python is overshadowed by simulation time.

Defining an optimisation or sampling problem
All optimisers operate on a callable ErrorMeasure object that describes a function to minimise or on a callable LogPDF object that describes a probability density function (PDF) to maximise. Similarly, all samplers start from a callable LogPDF, so that the same probability function can be used with both optimisers and samplers. The natural logarithm of the PDF is used for computational efficiency and accuracy, and we allow the probability density to be unnormalised (i.e. its integral does not have to sum to 1). Figure 2 shows how a user-defined model can be wrapped in a PINTS ForwardModel and combined with time points and measured values to create a Problem from which several standard ErrorMeasures and LogPDFs can be created. For inference in a Bayesian context, a LogPosterior class and several LogPrior distributions are provided. If a given LogPDF or ErrorMeasure cannot be constructed from PINTS classes, users can also define their own classes.

Implementation of optimisers and samplers
Most PINTS samplers and optimisers are implemented using a so-called ask-and-tell interface, inspired by the Python implementation of CMA-ES [6] (https://github.com/CMA-ES/pycma). In this framework, the details of solving the forward problem are partitioned away from the rest of the sampling or optimising algorithm. For each iteration the following steps are undertaken (Figure 3): first, the user calls ask() to obtain one or more parameter values from their chosen method — these values are typically vectors generated stochastically conditional on an internal system state; second, the user solves the forward model and generates a score for each parameter vector, for example, an error measure or (unnormalised) posterior probability; third, the user calls tell() to pass the score back to the method, which can then update its internal state and finish the iteration. For example, in

![Diagram](image-url)

**Figure 2:** An overview of the main PINTS classes used to define error measures (for optimisation) and PDFs (for optimisation or sampling). Users write a wrapper class for their model, making it available to pints, and must provide the experimental data using any Python sequence structure (for example, a list or a NumPy array). With these ingredients, a (single or multi-output) problem can be defined that can then be used with any of the available error measure or likelihood classes. Alternatively, users implement their own ErrorMeasure or LogPDF, which allows for further customisation and for problems other than time series problems to be solved.
Quality control

PINTS has three levels of testing: unit testing, functional testing, and comparative testing. Unit tests are used to test the functionality of simple (deterministic) methods, and to check that complex (pseudo-random) methods run without raising exceptions. All the unit tests are available to be run by a user to ensure the software is working correctly. Continuous integration is carried out using Travis CI (Ubuntu Trusty distribution with Python versions 2.7, 3.4, 3.5 and 3.6, and OS/X with Python version 2.7) and AppVeyor (Windows Server 2011 R2 with Python versions 2.7, 3.4, 3.5 and 3.6). PINTS uses Flake8 linter tests to ensure that contributed code conforms to best practices and code coverage tests to ensure that all code is sufficiently tested and documented. Functional testing of the methods is performed separately and is used to test the method’s behaviour from different (pseudo-random) initial conditions. Analysis of functional checking is done both visually and statistically (for example, if recent results deviate significantly from previous results this indicates the possible introduction of a bug). In our current solution, functional checking is run automatically but the results are interpreted by the PINTS developers. Work is ongoing to develop an automated analysis system so that these functional tests can be included in the continuous integration pipeline. Finally, in comparative testing a number of problems are set up and solved with different methods in order to compare the solutions they return and evaluate their performance. Performance testing is partially automated, but tests are initiated and analysed by the developer.

(2) Availability

Operating system

PINTS uses no functions specific to any operating system (OS) and so can run on any OS that provides Python. Optional parallelisation is provided that uses the Python multiprocessing module, which works best on UNIX-based systems (for example, Linux and OS/X), but runs on Windows with slightly reduced performance.

Programming language

PINTS requires Python 2.7 or higher, or Python 3.4 or higher.

Additional system requirements

PINTS has a minimal disk space footprint (approximately 2MB) and can be run on single-processor devices or headless on multi-processor machines (for example, via ssh).

Dependencies

PINTS uses the NumPy (version 1.8 or higher) and SciPy (version 0.14 or higher, [11]) libraries extensively. The default optimisation method is CMA-ES, for which the cma package (version 2 or higher) is used. The remaining
optimisation and sampling methods require no further dependencies. Finally, Matplotlib (version 1.5 or higher [7]) is required for installation, but it is possible to use PINTS without Matplotlib or with a different plotting library.

List of contributors
Michael Clerx, Sanmitra Ghosh, Ben Lambert, Chon Lok Lei, Martin Robinson.

Software location
Name: GitHub (release v0.2.2)
Persistent identifier: https://github.com/pints-team/pints/releases/tag/v0.2.2
Licence: BSD 3-clause
Publisher: Pints team
Version published: 0.2.2
Date published: 30/04/2019

Code repository
Name: GitHub (develop)
Persistent identifier: https://github.com/pints-team/pints
Licence: BSD 3-clause
Date published: 16/05/17

Language
English.

(3) Reuse potential
Detailed documentation is provided on using PINTS with user-supplied models (see, for example, the writing-a-model example on the GitHub repository). While PINTS was designed primarily with biological and electrochemical problems in mind, there is nothing to prohibit its use on time series models from other fields. Similarly, while the implemented ErrorMeasure and LogPDF classes were chosen to work well with time series problems, PINTS can be used outside this setting (in fact, utility functions fmin and curve_fit are provided specifically for this purpose). Users are also free to create their own ErrorMeasure or LogPDF classes that do not rely on PINTS ForwardModel or Problem classes, which allows PINTS to be used beyond its intended scope of time-series problems. A link to the full API documentation can be found on the GitHub repository, which also contains a list of examples for all PINTS’ main features (https://github.com/pints-team/pints/tree/master/examples). We have found that these examples, rather than our API documentation, serve to kick-start any new project based on PINTS.

Contact and support
We welcome questions, suggestions, bug reports, and user contributions via the GitHub repository, which acts as a central communication platform for PINTS. A detailed guide on contributing to PINTS is also available there. A Research Software Engineering group has recently been established at Oxford university, through which we hope to provide long-term support. In addition, we are happy to respond to questions via e-mail or other means of communication.

Competing Interests
The authors have no competing interests to declare.

Author Contributions
Michael Clerx and Martin Robinson authors contributed equally to this manuscript.

References


Submitted: 07 November 2018 Accepted: 05 July 2019 Published: 19 July 2019

Copyright: © 2019 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. See http://creativecommons.org/licenses/by/4.0/.

Journal of Open Research Software is a peer-reviewed open access journal published by Ubiquity Press

OPEN ACCESS