Pippi 2.0 manual (arXiv:1206.2245v3)
This version Nov 17 2016; earlier version published as EPJ Plus 127:138, 2012

Pippi – painless parsing, post-processing and plotting of posterior and likelihood samples

Pat Scott

Department of Physics, Imperial College London, London SW7 2AZ, UK
e-mail: p.scott@imperial.ac.uk

Abstract. Interpreting samples from likelihood or posterior probability density functions is rarely as straightforward as it seems it should be. Producing publication-quality graphics of these distributions is often similarly painful. In this short note I describe **pippi**, a simple, publicly-available package for parsing and post-processing such samples, as well as generating high-quality PDF graphics of the results. Pippi is easily and extensively configurable and customisable, both in its options for parsing and post-processing samples, and in the visual aspects of the figures it produces. I illustrate some of these using an existing supersymmetric global fit, performed in the context of a gamma-ray search for dark matter. Here I also outline new features introduced in pippi 2.0, including hdf5 support, out of core processing for extremely large datasets, flexible data cuts, per-observable binning, and inline post-processing with arbitrary Python expressions directly from the input `pip` file. Pippi can be downloaded and followed at http://github.com/patscott/pippi.

1 Introduction

Many applications in physics and astronomy require sampling from a probability distribution. Examples include parameter estimation for supersymmetry [1,2,3,4,5,6,7], cosmology [8,9] and cosmic ray propagation [10,11]. A range of sophisticated optimisation and exploration algorithms, and corresponding public codes, exist for doing just this. These include Markov-chain Monte Carlos (MCMCs; see e.g. [12]), nested sampling [13,14], genetic algorithms (e.g. [15]) and differential evolution [16]. However, the set of public tools available for analysing samples produced by these algorithms is somewhat smaller, and less developed. Here I describe **pippi**, a simple public code for analysing a set of samples from a likelihood or posterior probability density function (PDF). This updated note serves as an announcement of the public release of pippi 2.0, common documentation of its workings for papers relying on it (e.g. [17]), and a basic manual for prospective users.

Public codes do exist for this purpose; the best known are getdist, shipped as part of CosmoMC [8], and its various derivatives. Getdist requires the purchase and installation of Matlab, whereas pippi produces native pdfLaTeX output with Python and the open-source Ruby package ctioga2[1]. The resulting plots contain fully embedded LaTeX text and graphics, and are of very high visual quality. Python rewrites of getdist also exist, and produce similarly high-quality output to pippi. Apart from the extensive suite of options it offers, pippi differs from those codes in that it is not a translation or rewrite of getdist, and uses interpolation between binned samples rather than a contouring algorithm to produce colour maps; it thus provides a fully independent way to construct distributions from samples. It has been extensively tested against getdist, and the resulting distributions agree well.

Similar functions are also available as ROOT macros within RooStats [18]. These produce characteristically ugly ROOT figures and require a C++ driver program or implementation within a ROOT session. Barrett [19] and Superplot [20] provide additional alternatives.

In the following I briefly describe how pippi works, and give some examples of results produced with it. I will use the term 'chain' to refer to a set of samples produced by an arbitrary sampler, not just an MCMC.

2 Functions

Pippi consists of 6 core functions. Options are specified via an ASCII `.pip` input file passed as a command-line argument.

[1] http://ctioga2.rubyforge.org/

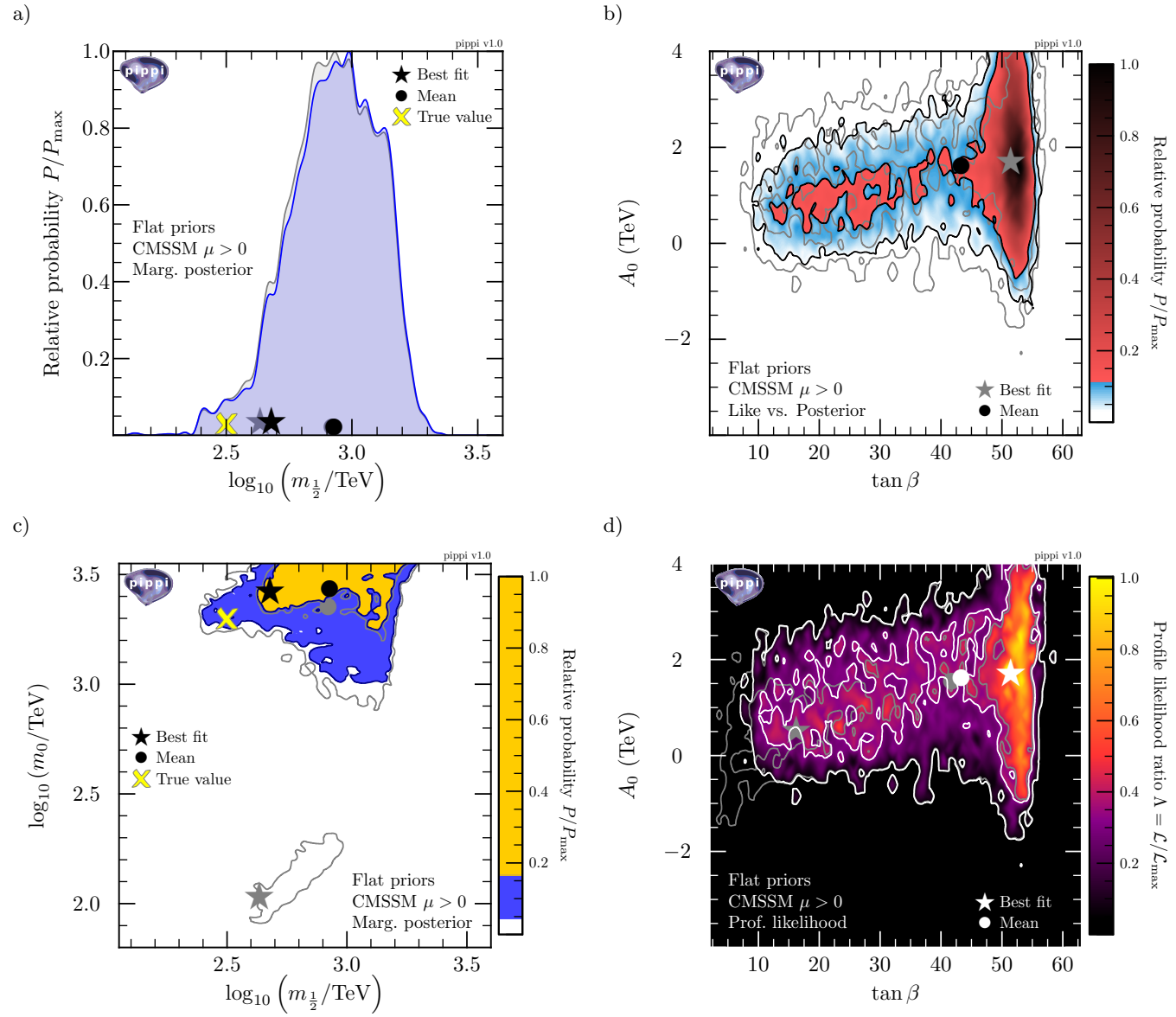

Fig. 1. Plots of posterior PDFs and profile likelihoods for a sample CMSSM chain taken from [3]. All points with $m_0 < 1\,\mathrm{TeV}$ have been heavily down-weighted; grey lines in subplots a, c and d show the corresponding distribution / contours for the original chain, without down-weighting. a) 1D marginalised posterior PDF of the parameter $m_{1/2}$, shown on a log scale. Here a mock "true point" is plotted for illustration alongside the real best fit and posterior mean. b) 2D posterior PDF of the parameters A_0 and $\tan\beta$ with 68% and 95% credible regions. The corresponding profile likelihood contours are shown in grey. c) The 2D marginalised posterior PDF of the parameters m_0 and $m_{1/2}$, showing the corresponding posterior PDF contours from the original chain in grey. A mock "true value" is again plotted. d) 2D profile likelihoods for the parameters A_0 and $\tan\beta$, comparing the 68% and 95% confidence regions obtained in the original and down-weighted chains.

pippi merge simply reads two or more chains, checks them for basic compatibility (number of columns, data types, etc.), and outputs a single concatenated chain to stdout.

pippi pare post-processes a chain, using a user-supplied function $F(\theta)$ for operating on a single sample θ. Pippi will dynamically load a python module M whose name is passed as a command-line argument, find F within it, and use $F(\theta)$ to operate on each point θ in the chosen chain. It will then output the resulting post-processed chain to stdout. The only thing required of the user is to write F, which implements the actual desired physics. F takes as input a vector containing the parameter and observable values of a single sample, and returns the post-processed parameter and observable values for that sample. The returned sample (and hence the final post-processed chain) need not contain the same parameters and observables as the original chain, nor even the same number of them. M

may contain any number of other routines, which may e.g. open a data file and initialise a new likelihood component or observable to be added to the chain. As of pippi 2.0, post-processing is also available via pippi parse.

pippi probe prints the names of the available data records in an hdf5 point database, along with the generalised column index that each will be mapped to.

pippi parse automatically bins a chain, then profiles its likelihood and/or marginalises its posterior PDF over relevant parameters. Options include which parameters or combinations of parameters to profile/marginalise over, by how much each parameter or observable should be rescaled, whether it must be binned and displayed in terms of its actual value or logarithm, and the range of values that the bins should encompass. The number of bins into which samples are sorted is individually configurable for each parameter or observable. The final resolution with which bin centres will be interpolated between in the output data files is also configurable. Either linear interpolation or curvature-minimising splines can be employed for this. Unlike other parsing programs, the option to smooth the output distributions is explicitly excluded, as this amounts to modifying the underlying chain; a similar effect can be achieved whilst preserving the underlying data using interpolation. Parse has the ability to work with an essentially arbitrary chain format, with multiplicities and likelihoods, χ^2 values or $\pm$log-likelihoods located in any column of the chain.

Similarly, it can also import data from point databases in hdf5 format. In this case, the user typically needs to specify in their input `.pip` file which named data records in the `.hdf5` file correspond to which parameters, observables, likelihoods, multiplicities and related quantities, in much the same way as different columns in ASCII chain files must be identified with specific quantities. When importing results from `.hdf5` files, pippi operates in out of core mode, reading *only* the observables and parameters requested for plotting into memory, rather than the entire file. This essentially allows arbitrarily large datasets to be plotted; in extreme cases, this can be achieved by restricting single `.pip` files to deal with only one observable or pair of observables. As of pippi 2.0, parse is also able to compute new observables and arbitrary transformations of old observables on the fly, by reading Python code embedded directly in a user's `.pip` file. The results of these on-the-fly calculations can be saved and plotted like any other observable, and used in further calculations in the same run of pippi. They can also be used together with the new ability to cut samples on the value of any parameter or observable to apply any arbitrarily complicated cuts to the underlying dataset before plotting.

pippi script writes shell scripts for plotting a parsed chain with ctioga2. Either 1D or 2D distributions can be plotted, including comparison of two chains, or comparison of profile likelihoods and posterior PDFs. 1D plots may be presented as histograms or interpolated distributions. 2D plots may have arbitrary confidence contours, shading and a colour bar. Axis labels and all other annotations can be specified directly in true LaTeX. The best fit and posterior mean may be plotted on or excluded from different plots, and corresponding legends and keys can be automatically drawn and placed. A reference point (and key) can be specified and plotted in terms of any combination of parameters and/or observables. A by-line can be placed in the top right of the figure, and a PDF logo or other image can even be included. All aspects of the colour scheme, markers, gradients, transparencies and line drawing can be modified by choosing a different built-in scheme, or easily writing one's own scheme in a few short lines of Python code.

pippi plot runs the plotting scripts created in a script operation, and organises the resulting PDF files according to the specified pip file.

If pippi is invoked with only the name of a pip file, the parse, script and plot functions are automatically performed in this order. Chains, intermediate and final files can all be arranged automatically into any combination of different or identical directories, using any combination of relative or absolute paths. Missing paths are created automatically.

3 Examples

In Fig. 1 I show some example plots created from the chain included in the pippi distribution, which comes originally from [3]. This chain is based on a global fit to the Constrained Minimal Supersymmetric Standard Model (CMSSM), and was created using SuperBayeS v1.35 [2], with all likelihood components turned on. Here I have used the pare function of pippi to reduce the likelihoods and posterior weightings of all points in the chain with values of the parameter $m_0 < 1\,\text{TeV}$, so as to remove the area at low m_0 known as the stau co-annihilation region. The resulting 1D marginalised posterior PDF for the parameter $m_{1/2}$ is shown for the chain processed by pippi pare (a 'pared chain') in blue in Fig. 1a, alongside the corresponding marginalised posterior for the original chain in grey. The equivalent 2D distribution in the m_0, $m_{1/2}$ plane is given directly below in Fig. 1c, with the 68% and 95% credible contours from the pared chain plotted in colour, and the contours corresponding to the original chain in grey. The best-fit and posterior mean are plotted in each case, in grey for the original and black for the pared chain. For the sake of illustration, I have also added a fictional "true value" to the these two plots.

Fig. 1b compares the posterior PDF (coloured) to the profile likelihood (grey) in the pared chain, this time in the A_0, $\tan\beta$ plane. In this case I have employed a visual scheme with a gradient fill for the 2D marginalised posterior. Similarly in Fig. 1d, where I compare the profile likelihood in the pared (colour) and original (grey) chains in the A_0, $\tan\beta$ plane, using yet another built-in visual scheme.

An example pip file for creating these and other plots is included in the pippi distribution. The Python function used with pippi pare to effect the $m_0 > 1\,\text{TeV}$ post-processing cut is also included. As of pippi 2.0, this example also includes the new features like data cuts and inline post-processing, so the final plots look a little different to Fig. 1.

4 Summary

Pippi can automatically bin, marginalise and profile sets of posterior or likelihood samples, or post-process them using functions easily defined by the user. It produces clean, visually-appealing plots in native PDF format, with a minimum of effort and maximal flexibility. Pippi 2.0 depends on Python v2.7 or later, ctioga2 v0.8 or later, SciPy, NumPy (v0.9.0 or later to use the spline interpolation option) and bash. It requires essentially no installation beyond unpacking a tarball and adding the new directory to the shell PATH variable. The latest incarnation of pippi can always be found at http://github.com/patscott/pippi.

Acknowledgements

I thank Antje Putze and Christoph Weniger for helpful comments during development of pippi, and Christoph for contributing draft code for supporting hdf5 inputs.

References

1. B. C. Allanach, C. G. Lester, Phys. Rev. D **73**, 1, 015013 (2006), arXiv:hep-ph/0507283, 10.1103/PhysRevD.73.015013
2. R. Trotta, F. Feroz, M. Hobson, et al., JHEP **12**, 24 (2008), arXiv:0809.3792, 10.1088/1126-6708/2008/12/024
3. P. Scott, J. Conrad, J. Edsjö, et al., JCAP **1**, 31 (2010), arXiv:0909.3300, 10.1088/1475-7516/2010/01/031
4. Y. Akrami, P. Scott, J. Edsjö, et al., JHEP **4**, 57 (2010), arXiv:0910.3950, 10.1007/JHEP04(2010)057
5. Y. Akrami, C. Savage, P. Scott, et al., JCAP **4**, 12 (2011), arXiv:1011.4318, 10.1088/1475-7516/2011/04/012
6. Y. Akrami, C. Savage, P. Scott, et al., JCAP **7**, 2 (2011), arXiv:1011.4297, 10.1088/1475-7516/2011/07/002
7. P. Bechtle, T. Bringmann, K. Desch, et al., JHEP **6**, 98 (2012), arXiv:1204.4199, 10.1007/JHEP06(2012)098
8. A. Lewis, S. Bridle, Phys. Rev. D **66**, 10, 103511 (2002), arXiv:astro-ph/0205436, 10.1103/PhysRevD.66.103511
9. M. J. Mortonson, H. V. Peiris, R. Easther, Phys. Rev. D **83**, 4, 043505 (2011), arXiv:1007.4205, 10.1103/PhysRevD.83.043505
10. A. Putze, L. Derome, D. Maurin, A&A **516**, A66 (2010), arXiv:1001.0551, 10.1051/0004-6361/201014010
11. R. Trotta, G. Jóhannesson, I. V. Moskalenko, et al., ApJ **729**, 106 (2011), arXiv:1011.0037, 10.1088/0004-637X/729/2/106
12. W. H. Press, S. A. Tuekolsky, W. T. Vetterling, et al., *Numerical Recipes*, 3rd edn. (Cambridge University Press, 2007)
13. J. Skilling, in R. Fischer, R. Preuss, U. V. Toussaint, eds., *American Institute of Physics Conference Series*, vol. 735, 395–405 (2004)
14. F. Feroz, M. P. Hobson, M. Bridges, MNRAS **398**, 1601 (2009), arXiv:0809.3437, 10.1111/j.1365-2966.2009.14548.x
15. P. Charbonneau, ApJS **101**, 309 (1995), 10.1086/192242
16. R. Storn, K. Price, J. Global Optimization **11**, 4, 341 (1997), 10.1023/A:1008202821328
17. P. Scott, C. Savage, J. Edsjö, et al., JCAP **11**, 57 (2012), arXiv:1207.0810, 10.1088/1475-7516/2012/11/057
18. L. Moneta, K. Belasco, K. Cranmer, et al., PoS(**ACAT2010**)057 (2010), arXiv:1009.1003
19. S. Liem (2016), arXiv:1608.00990
20. A. Fowlie, M. H. Bardsley, Eur. Phys. J. Plus **131**, 391 (2016), arXiv:1603.00555, 10.1140/epjp/i2016-16391-0