pyjags Documentation

Release 1.2.2

Tomasz Miąsko

Mar 31, 2017

Contents

CHAPTER 1

Documentation

Getting Started

Prerequisites

- JAGS
- C++11 compiler
- Python 2.7, >= 3.2
- NumPy >= 1.7

Installation

Stable release from pip (recommended):

```
pip install pyjags
```

Development version:

```
pip install git+https://github.com/tmiasko/pyjags.git
```

Using setup.py after cloning the repository:

```
git clone --recursive https://github.com/tmiasko/pyjags.git
python setup.py install
```

The setup.py script uses pkg-config to locate the JAGS library. If JAGS is installed in some non-standard location, then you may need to configure pkg-config to pickup correct metadata file. For example, if JAGS have been configured with `--prefix=/opt/`, then JAGS metadata file would be located in `/opt/lib/pkgconfig/`. This path can be included in pkg-config search path as follows:

```
export PKG_CONFIG_PATH=/opt/lib/pkgconfig/:$PKG_CONFIG_PATH
```

Example

Simple linear regression:

```
import pyjags
import numpy as np

np.random.seed(0)
np.set_printoptions(precision=1)

N = 500
a = 70
b = 4
sigma = 50

x = np.random.uniform(0, 100, size=N)
y = np.random.normal(a + x*b, sigma, size=N)

code = '''
model {
    for (i in 1:N) {
        y[i] ~ dnorm(alpha + beta * x[i], tau)
    }
    alpha ~ dunif(-1e3, 1e3)
    beta ~ dunif(-1e3, 1e3)
    tau <- 1 / sigma^2
    sigma ~ dgamma(1e-4, 1e-4)
}
'''

model = pyjags.Model(code, data=dict(x=x, y=y, N=N), chains=4)
samples = model.sample(5000, vars=['alpha', 'beta', 'sigma'])

def summary(samples, varname, p=95):
    values = samples[varname]
    ci = np.percentile(values, [100-p, p])
    print('{:<6} mean = {:>5.1f}, {}% credible interval [{:>4.1f} {:>4.1f}]'.format(
      varname, np.mean(values), p, *ci))

for varname in ['alpha', 'beta', 'sigma']:
    summary(samples, varname)
```

```
adapting: iterations 1000 of 1000, elapsed 0:00:03, remaining 0:00:00
sampling: iterations 5000 of 5000, elapsed 0:00:17, remaining 0:00:00
alpha  mean =  64.2, 95% credible interval [56.9 71.3]
beta   mean =   4.0, 95% credible interval [ 3.9  4.1]
sigma  mean =  49.4, 95% credible interval [46.9 52.1]
```

API Reference

pyjags module

This is a convenience module that imports all names from submodules. Equivalent to

```
from pyjags.model import *
from pyjags.modules import *
```

pyjags.model

class pyjags.model.**Model**(*code=None*, *data=None*, *init=None*, *chains=4*, *adapt=1000*, *file=None*, *encoding='utf-8'*, *generate_data=True*, *progress_bar=True*, *refresh_seconds=None*, *threads=1*, *chains_per_thread=1*)

High level representation of JAGS model.

chains
int – A number of chains in the model.

Note: In JAGS arrays are indexed from 1. On the other hand Python uses 0 based indexing. It is important to keep this in mind when providing data to JAGS and interpreting resulting samples. For example, what in JAGS would be x[4,2,7] in Python is x[3,1,6].

Note: The JAGS supports data sets where some of observations have no value. In PyJAGS those missing values are described using numpy MaskedArray. For example, to create a model with observations x[1] = 0.25, x[3] = 0.75, and observation x[2] missing, we would provide following data to Model constructor:

```
>>> {'x': np.ma.masked_array(data=[0.25, 0, 0.75], mask=[False, True, False])}
{'x': masked_array(data = [0.25 -- 0.75],
             mask = [False  True False],
       fill_value = 1e+20)
}
```

From JAGS version 4.0.0 it is also possible to monitor variables that are not completely defined in the description of the model, e.g., if y[i] is defined only for y[3], then y[1], and y[2] will have missing values for all iterations in all chains. Those missing values are also represented using numpy MaskedArray.

Create a JAGS model and run adaptation steps.

Parameters

- **code** (*str or bytes, optional*) – Code of the model to load. Model may be also provided with file keyword argument.
- **file** (*str, optional*) – Path to the model to load. Model may be also provided with code keyword argument.
- **init** (*dict or list of dicts, optional*) – Specifies initial values for parameters. It can be either a dictionary providing initial values for parameters used as keys, or a list of dictionaries providing initial values separately for each chain. If omitted, initial values will be generated automatically.

Additionally this option allows to configure random number generators using following special keys:

- '.RNG.name' str, name of random number generator
- '.RNG.seed' int, seed for random number generator
- '.RNG.state' array, may be specified instead of seed, shape of array depends on particular generator used

- **data** (*dict, optional*) – Dictionary with observed nodes in the model. Keys are variable names and values should be convertible to numpy arrays with shape compatible with one used in the model.

  The numpy.ma.MaskedArray can be used to provide data where some of observations are missing.

- **generate_data** (*bool, optional*) – If true, data block in the model is used to generate data.
- **chains** (*int, 4 by default*) – A positive number specifying number of parallel chains.
- **adapt** (*int, 1000 by default*) – An integer specifying number of adaptations steps.
- **encoding** (*str, 'utf-8' by default*) – When model code is provided as a string, this specifies its encoding.
- **progress_bar** (*bool, optional*) – If true, enables the progress bar.
- **threads** (*int, 1 by default*) – A positive integer specifying number of threads used to sample from model. Using more than one thread is experimental functionality.
- **chains_per_thread** (*int, 1 by default*) – A positive integer specifying a maximum number of chains sampled in a single thread. Takes effect only when using more than one thread.

update (*iterations*)

Updates the model for given number of iterations.

sample (*iterations*, *vars=None*, *thin=1*, *monitor_type='trace'*)

Creates monitors for given variables, runs the model for provided number of iterations and returns monitored samples.

Parameters

- **iterations** (*int*) – A positive integer specifying number of iterations.
- **vars** (*list of str, optional*) – A list of variables to monitor.
- **thin** (*int, optional*) – A positive integer specifying thinning interval.

Returns Sampled values of monitored variables as a dictionary where keys are variable names and values are numpy arrays with shape: (dim_1, dim_n, iterations, chains). dim_1, ..., dim_n describe the shape of variable in JAGS model.

Return type dict

adapt (*iterations*)

Run adaptation steps to maximize samplers efficiency.

Returns True if achieved performance is close to the theoretical optimum.

Return type bool

variables
Variable names used in the model.

state
Values of model parameters and model data for each chain.

See also:

parameters, *data*

parameters
Values of model parameters for each chain. Includes name of random number generator as '.RNG.name' and its state as '.RNG.state'.

data
Model data. Includes data provided during model construction and data generated as part of data block.

pyjags.modules

pyjags.modules.**version**()
JAGS version as a tuple of ints.

```
>>> pyjags.version()
(3, 4, 0)
```

pyjags.modules.**get_modules_dir**()
Return modules directory.

pyjags.modules.**set_modules_dir**(*directory*)
Set modules directory.

pyjags.modules.**list_modules**()
Return a list of loaded modules.

pyjags.modules.**load_module**(*name*, *modules_dir=None*)
Load a module.

Parameters

- **name** (*str*) – A name of module to load.
- **modules_dir** (*str, optional*) – Directory where modules are located.

pyjags.modules.**unload_module**(*name*)
Unload a module.

CHAPTER 2

JAGS Documentation

- JAGS manual and examples

CHAPTER 3

Similar projects

- PyStan
- PyMC
- PyMC3

CHAPTER 4

Indices and tables

- genindex
- search

Python Module Index

p

Index