


(/)

コミュニティ

キーワードを入

ユーザ登録 (/signup?redirect\_to=%2Fkenmatsu4%2Fitems%2Fa0c703762a2429e21793)

ログイン (/login?redirect\_to=%2Fkenmatsu4%2Fitems%2Fa0c703762a2429e21793)



@kenmatsu4 (/kenmatsu4) 2015年11月19日に更新

# 超訳 PyMC3 Tutorial（マルコフ連鎖モンテカルロ法フレームワーク）その1

Python(/tags/Python) MCMC(/tags/MCMC)

統計学(/tags/%E7%B5%B1%E8%A8%88%E5%AD%A6)

ベイズ統計 (/tags/%E3%83%99%E3%82%A4%E3%82%BA%E7%B5%B1%E8%A8%88%E5%AD%A6)

マルコフ連鎖(/tags/%E3%83%9E%E3%83%AB%E3%82%B3%E3%83%95%E9%80%A3%E9%8E%96%E3%83%A2%E3%83%B3%E3%83%86%E3%82%AB%E3%83%AB%E3%83%AD%E6%B3%95)

76

この記事は最終更新日から1年以上が経過しています。

Pythonでマルコフ連鎖モンテカルロ法(MCMC)を実行できるライブラリ、PyMC3のチュートリアルの記事を書いてみました。タイトルにあるように、原文をそのままではなく意識を超えた「超訳」です 🙇

## 原文のURL

[http://pymc-devs.github.io/pymc3/getting\\_started/](http://pymc-devs.github.io/pymc3/getting_started/) ([http://pymc-devs.github.io/pymc3/getting\\_started/](http://pymc-devs.github.io/pymc3/getting_started/))

## イントロダクション（だいたい省略）

確率的プログラミング(Probabilistic programming : PP)は柔軟なベイズ統計モデルをプログラムで行うことを可能にします。

PyMC3は新しいオープンソースの確率プログラミングフレームワークで、No-U-Turn Sampler (NUTS; Hoffman, 2014)や、ハミルトニアンモンテカルロ法 (HMC; Duane, 1987)のパラメーターの自己チューニングなど、次世代のマルコフ連鎖モンテカルロ法(MCMC)が使えることが特徴です。

NUTSはまた、幾つかのセルフチューニングストラテジーを持っており、ハミルトニアンモンテカルロ法のチューニング可能なパラメーターを順応的にセットします。なので、ユーザーはアルゴリズムをうまく動かすためのチューニングに関する特別な知識は必要ではありません。

現在、HMCが使える確率プログラミングパッケージはPyMC3, Stan (Stan Development Team, 2014), とR用のLaplacesDemonだけです。

PyMC3はpure Pythonで書くことができ、その裏ではTheanoが使われています。ここでは、モデルのコードからCのコードが生成され、さらにそれがコンパイルされて機械語になるので、パフォーマンスが高いです。



イントロダクション（だいたい省略）

インストール

線形回帰の例

テストデータの生成

モデルの特定(Model Specification)

モデルフィッティング

最大事後確率化法 (Maximum a posteriori methods)

サンプリングの方法 (Sampling methods)

勾配ベースのサンプリング法 (Gradient-based sampling methods)

事後分布の分析(Posterior analysis)

ここでは、PyMC3を使って一般的なベイズ統計推定を解き、問題の推定を行うことについての入門をご紹介します。

まず最初にPyMC3の使い方についての基礎を下記のexampleで説明します。

- インストール
- データ生成
- モデルの決定
- モデルフィッティング
- 事後分布の分析

2つのケーススタディをつかって、モデルの定義と、より洗練されたモデルにフィットする方法を説明します。

最後に、どうやって PyMC3を拡張するかということと、下記のいくつかの有用な特徴について説明します。

- 一般化線形モデルサブパッケージ
- カスタマイズされた分布、Transformation
- バックエンドのストレージについて

## インストール

Python version: 2.7以降、もしくは3.4以降 (3.4以降がおすすめ)

対応OS

Mac OSX, Linux, Windows

(Anaconda Python Distribution を使うと簡単に導入できる。)

PyMC3は `pip` でインストールできます。

```
pip install git+https://github.com/pymc-devs/pymc3
```

PyMC3は他のパッケージにも依存していますが、`pip`を使うことで自動的にインストールされます。(Theano, NumPy, SciPy, and Matplotlibなど) PyMC3をフル活用するならPandas と Patsyもインストールしたほうがいいでしょう。これらは自動にインストールされないで、下記の`pip`コマンドを使って各自でインストールしてください。

```
pip install patsy pandas
```

PyMC3のソースコードはGitHub(<https://github.com/pymc-devs/pymc3>)にあります。  
(<https://github.com/pymc-devs/pymc3>)%E2%80%8B%E3%81%AB%E3%81%82%E3%82%8A%E3%81%BE%E3%81%99%E3%80%82)

## 線形回帰の例

モデル定義の例として、パラメーターの事前分布が正規分布であるようなベイズ線形回帰を使います。

被説明変数は $Y$ で、期待値  $\mu$  の正規分布からの観測値です。説明変数は2つで  $X_1$  と

$X_2$  です。

$$Y \sim N(\mu, \sigma^2)$$

$$\mu = \alpha + \beta_1 X_1 + \beta_2 X_2$$

ベイズアンモデルを構成するので、未知の変数には事前分布を割り当てる必要があります。今回は無情報事前分布として、 $\beta_1, \beta_2$  には平均ゼロ、分散100の正規分布を使います。 $\sigma$  には半正規分布（0で折り返した正規分布）を適用します。

$$\alpha \sim N(0, 100)$$

$$\beta_i \sim N(0, 100)$$

$$\sigma \sim |N(0, 1)|$$

## テストデータの生成

numpyのrandomモジュールを使った乱数から生成したテストデータでシミュレーションができます。それで、pymc3を使ってその乱数生成モデルで使われる対応するパラメータを推定して取り出してみます。なので、意図的にデータ生成はPyMC2のモデル構造とほぼ一致させています。

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Intialize random number generator
np.random.seed(123)

# True parameter values
alpha, sigma = 1, 1
beta = [1, 2.5]

# Size of dataset
size = 100

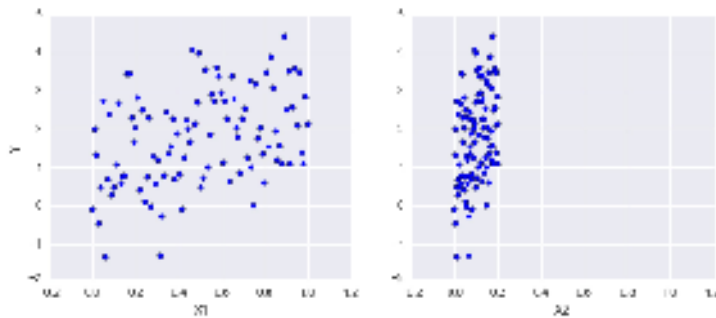
# Predictor variable
X1 = np.linspace(0, 1, size)
X2 = np.linspace(0,.2, size)

# Simulate outcome variable
Y = alpha + beta[0]*X1 + beta[1]*X2 + np.random.randn(size)*sigma

%matplotlib inline

fig, axes = plt.subplots(1, 2, sharex=True, figsize=(10,4))
axes[0].scatter(X1, Y)
axes[1].scatter(X2, Y)
axes[0].set_ylabel('Y'); axes[0].set_xlabel('X1'); axes[1].set_xlabel('X2')
);
```

テストデータをmatplotlibでグラフ化します。



(<https://camo.qiitusercontent.com/75e60ff144752232f33cf43d5022381bd448db5b/68747470733a2f2f1696974612d696d6167652d73746f72652e73332e616d617a6f6e6177732e636f6d2f302f35303637302f64303761663633372d383162652d326330652d326661392d6166653732313963316638312e706e67>)

## モデルの特定(Model Specification)

pymc2でのモデル特定は、文法が統計的な表記と近いので簡単です。

ほとんどの箇所、Pythonのコードの位置行一行が、その統計モデル表記と一致するのです。

まず初めに、pymc3のコンポーネントをインポートします。

```
from pymc3 import Model, Normal, HalfNormal
```

さて、実際にモデルを作ってみます。

まず全体を下記に記載しますので、そのあと一つ一つ解説します。

```
basic_model = Model()

with basic_model:

    # Priors for unknown model parameters
    alpha = Normal('alpha', mu=0, sd=10)
    beta = Normal('beta', mu=0, sd=10, shape=2)
    sigma = HalfNormal('sigma', sd=1)

    # Expected value of outcome
    mu = alpha + beta[0]*X1 + beta[1]*X2

    # Likelihood (sampling distribution) of observations
    Y_obs = Normal('Y_obs', mu=mu, sd=sigma, observed=Y)
```

最初の行は

```
basic_model = Model()
```

です。この行は、乱数を格納するModelオブジェクトを生成しています。

withステートメントで囲むことで、その中ではモデルのインスタンスを省略できます。

```
with basic_model:
```

with句で作られるこのインデントブロックは、コンテキストマネージャーを生成しており、`basic_model` をコンテキストとして、インデントの終わりまでノスベテのステートメントで使用されています。すべてのPyMC3オブジェクトは、このwithステートメントの中にインデントで表されたコードブロックで使われます。このcontext managerの書き方がなければ、それぞれの変数に`basic_model`を明示してなければならなくなります。その場合、もし新しい確率変数を `with model:` ステートメントで作らなければ、その変数が追加される明示的なモデルがないために、エラーとなるでしょう。

最初の3つの文を見てみましょう。

```
alpha = Normal('alpha', mu=0, sd=10)
beta = Normal('beta', mu=0, sd=10, shape=2)
sigma = HalfNormal('sigma', sd=1)
```

これらは確率変数を生成します。1つ目は回帰係数 $\alpha$ で、平均0、標準偏差10の正規分布に従う変数となります。 $\beta$ も同じく平均0、標準偏差10の正規分布に従う変数です。

これらは確率的です。なぜなら一部は確率変数の依存グラフツリー状の親によって確定されており、その他の一部は確率的であるからです。(訳注: 依存グラフツリー⇒グラフィカルモデルのようなもの?)

Normalクラスのコンストラクタは正規事前分布に従う確率変数を生成する時に使われます。

第一引数は確率変数の名前です。基本的にPythonコード上で戻り値を格納する変数名と名前を合わせてください。PyMCの計算結果から変数を取り出す時にこれを使用することがあります。

残りの必須引数は確率分布に必要なパラメーターの指定です。この正規事前確率の場合、ハイパーパラメーターとしてmu: 平均、sd: 標準偏差を指定してmodelに設定します。

一般的に分布のパラメーターはその位置や分布の形、散らばりの具合を決めるものです。

一般的に使われているBeta(ベータ分布), Exponential(指数分布), Categorical(カテゴリカル分布[多変数ベルヌーイ]), Gamma(ガンマ分布), Binomial(二項分布)等様々な分布をPyMC3でも利用することができます。

変数betaにはNormalコンストラクタの引数にshapeが追加されています。これが設定されるとパラメータはベクトルとして定義され、この例の場合betaは2次元ベクトルになります。引数shapeはすべての分布で使用することができます。しかしスカラーにする場合はデフォルトが1なので特に指定する必要はありません。

数値は整数(int)、配列、タプル、多次元配列が使えます。shape=(5,7)とすると5x7の確率変数の行列となります。

確率分布、サンプリング方法、その他PyMC3の機能に関する詳細なヘルプはhelp()で確認することができます。

```
help(Normal) #try help(Model), help(Uniform) or help(basic_model)
```

out

```

Help on class Normal in module pymc3.distributions.continuous:

class Normal(pymc3.distributions.distribution.Continuous)
|   Normal log-likelihood.
|
|   .. math::
|
|   ight\}
|
|   Parameters
|   -----
|   mu : float
|       Mean of the distribution.
|   tau : float
|       Precision of the distribution, which corresponds to
|       :math:`1/\sigma^2` (tau > 0).
|   sd : float
|       Standard deviation of the distribution. Alternative parameterizati
on.
|
|   .. note::
|   - :math:`E(X) = \mu`
|   - :math:`Var(X) = 1/\tau`
|
|   Method resolution order:
|       Normal
|       pymc3.distributions.distribution.Continuous
|       pymc3.distributions.distribution.Distribution
|       builtins.object
|
|   Methods defined here:
|
|   __init__(self, mu=0.0, tau=None, sd=None, *args, **kwargs)
|
|   logp(self, value)
|
|   random(self, point=None, size=None, repeat=None)
|
|   -----
|   Methods inherited from pymc3.distributions.distribution.Distribution:
|
|   __getnewargs__(self)
|
|   default(self)
|
|   get_test_val(self, val, defaults)
|
|   getattr_value(self, val)
|
|   -----
|   Class methods inherited from pymc3.distributions.distribution.Distribu
tion:
|
|   dist(*args, **kwargs) from builtins.type
|
|   -----
|   Static methods inherited from pymc3.distributions.distribution.Distrib
ution:
|
|   __new__(cls, name, *args, **kwargs)
|
|   -----
|   Data descriptors inherited from pymc3.distributions.distribution.Distr

```

```
__dict__:
    dictionary for instance variables (if defined)
__weakref__:
    list of weak references to the object (if defined)
```

事前分布の定義にあたり、下記のコードはoutcomeの期待値muを線形の関係として生成しています。

```
mu = alpha + beta[0]*X1 + beta[1]*X2
```

これは決定論的確率変数を作ります。これは親となる値によって完全にその値が決まることを暗に示しています。つまり、親となる値から継承されるに当たっては不確実性がないということです。

ここではmuは単に切片alphaと2つの係数（傾き）betaと2つの説明変数をかけたものの、和でできているということです。PyMC3の確率変数とデータは任意に加減乗除をたがいにすることができ、それを元に新しい確率変数を作ることができます。

この表記法は、高いモデルの表現性をもたらします。sum, sin expなどの数学関数と、dotsやinvなどの線形代数関数なども使うことができます。

このモデルの最後の行で定義されているY\_obsは、データセットから得られた標本分布です。

```
Y_obs = Normal('Y_obs', mu=mu, sd=sigma, observed=Y)
```

これは確率変数の特殊ケースで、観測された確率（observed stochastic）と呼ばれ、データから得られたモデルの尤度を表します。これは通常の確率変数と同じであるけれどもそれが既に観測されていて、それが変数に代入されている点が異なります。観測されているので不変となり、モデルを推定するアルゴリズムの中でも値が変化しません。

データはnumpy.ndarray or pandas.DataFrameのオブジェクトを使って代入が可能です。

ここで注意が必要なのは、これまでの事前分布と違い、この正規分布Y\_obsのパラメーターは固定値ではなく、決定的オブジェクトmuと確率変数sigmaからできていることです。

これは、尤度と2つの確率変数の間に親子関係を生成します。

## モデルフィッティング

モデルが完全に設計できたので、次のステップはモデル中の未知の変数に対して、事後分布の推定量を得る事です。理想的には事後分布の推定量を解析的に解けるとよいのですが、多くの簡単ではないモデルではそれは難しいです。

そこで2つのアプローチを考えます。それらの妥当性はモデルの構造や、分析のゴールに依存します。



- 最大事後確率点(maximum a posteriori[MAP] point) を最適化法で見つける。
- マルコフ連鎖モンテカルロ法(Markov Chain Monte Carlo [MCMC]) を使って、事後分布からサンプルを生成してそのサマリーを計算する。

## 最大事後確率化法 (Maximum a posteriori methods)

---

最大事後確率とは...

- モデルを推定するもの
- 事後分布の最大値
- 一般的には数値最適化法を使って得ることができる。

ものです。

これはほとんどの場合高速に、かつ簡単に実行できますが、しかし点でのパラメータの推定値しか得ることができません。さらに、もし最頻値が分布の代表値でない場合、偏りが出る可能性があります。PyMC3はfind\_MAPでこの機能を実現しています。

下記でこのページで取り上げたモデルのMAP値を算出しています。このMAPはパラメータを点として返しており、それはPythonのDictionaryで変数に名前付けをしてNumPyのarrayでパラメータ値を保管しています。

```
from pymc3 import find_MAP

map_estimate = find_MAP(model=basic_model)

print(map_estimate)

out

{'sigma_log': array(0.119287649834956), 'beta': array([ 1.46791595,  0.293
58319]), 'alpha': array(1.01366409951285)}
```

デフォルトでは、find\_MAPはBroyden-Fletcher-Goldfarb-Shanno (BFGS)という最適化アルゴリズムを使ってLog事後確率の最大化点を導出しています。しかし、ほかの最適化手法もオプションとして、scipy.optimizeから選択することができます。下記では例として、Powell's methodをMAPの導出に使っています。

```
from scipy import optimize

map_estimate = find_MAP(model=basic_model, fmin=optimize.fmin_powell)

print(map_estimate)

out

{'sigma_log': array(0.1181510683418693), 'beta': array([ 1.51426781,  0.03
520891]), 'alpha': array(1.0175522115056725)}
```

MAP推定量がいつでも妥当かというそうではないことが、特に最頻値が極端な場合に重要です。サンプルサイズが少ないことにより、局所的に高い確率密度が存在し、全体としては確率が低いような場合、高次元の事後確率ではちょっとした問題になります。これは分散パラメーターにランダム効果があるような階層モデルでよく発生します。もし、個々のグループの平均がすべて一緒に、分散などのスケールパラメーターがほぼゼロであれば、たとえこのような小さなスケールパラメーターの確率が小さくても、グループの平均が極端にお互い近くなるので、事後確率は無限に近い密度を持ちます。

MAP推定量を得るための多くのテクニックは局所最適を得る方法でしかない（たいていの場合それで十分だが）。しかし、マルチモーダルな（双峰性をもつ）事後分布に対しては、それぞれの異なる最頻値が無視できないくらい違う場合、ひどく失敗する可能性がある。

## サンプリングの方法 (Sampling methods)

MAPを探すことは、モデルの未知のパラメーターの推定値を得るための高速で簡単な方法ですが、これには制限があります。（というのも、MAP推定量から得られる推定量の不確実性に関する推定量がないからです。

その代わりに、マルコフ連鎖モンテカルロ法(MCMC)などのシミュレーションベースの方法を値のマルコフ連鎖を得るために使うことができます。これは、とある条件を満たしていれば事後分布から得られたサンプルと見分けがつかません。

PyMC3を使って事後確率のサンプルを生成するMCMCサンプリングを実行するには、特定のMCMCアルゴリズムに対応したstep methodを決める必要があります。（メトロポリス法、スライスサンプリング、No-U-Turn Sampler (NUTS)など）

PyMC3のstep\_methodsには下記のものがあります。

- NUTS
- Metropolis
- Slice
- HamiltonianMC
- BinaryMetropolis

これらのStep MethodはPyMC3では自動に決まるが、手動で変更することもできます。このstep methodの自動選択はモデルの中のそれぞれの変数のタイプに応じて下記のように設定されます。

- 二値変数(Boolean)には BinaryMetropolisが割り当てられる
- 離散変数 Metropolisが割り当てられる
- 連続変数 NUTSが割り当てられる

この自動選択機能はサンプリングの実施前に手動で再設定することで、任意の変数のサブセットにも上書き適用することができます。

## 勾配ベースのサンプリング法 (Gradient-based sampling methods)

PyMC3は標準的なサンプリングアルゴリズム (アダプティブメトロポリス法、アダプティブスライスサンプリングなど) が実装されているが、PyMC3で使うのに最適は方法はNo-U-Turn サンプラーです。

NUTSは、多くの連続なパラメーターを持ち、ほかのMCMCアルゴリズムではシミュレーションがとて遅くなるようなモデルで特に有用です。この手法は、対数事後分布の密度関数の勾配をベースとして、どこ領域が高確率であるか、という情報を利用して使います。大きな問題を扱う場合に、この手法は今までのサンプリング方法よりも劇的に収束を早くすることができます。

PyMC3は、事後分布の密度関数を微分することによってモデルの勾配を解析的に求める、という処理にTheanoを内部的に使っています。

NUTSはまた、ハミルトニアンモンテカルロ法のチューニングパラメーターを順応的にセットするために、いくつかの自動チューニング機能を持っています。微分不可な確率変数 (つまり、離散変数) には、NUTSは使えません。しかし、微分不可な変数をモデルがもっていても、そのモデルの中の微分可能な変数に対しては適用することができます。

NUTSは尺度行列パラメーター (スケーリング マトリクス パラメーター) を必要とします。これは、メトロポリスヘイスティングス法における提案分布の分散パラメーターに類似したものです。ただし、NUTSではいくらか違う使われ方をします。この行列は、NUTSがある方向に過大に、もしくは過小にジャンプしないように、分布のラフな形を与えるものです。

効果的なサンプルを実行し、妥当な値を得るためにこの尺度パラメータを決めることは重要です。多くの観測できない確率変数があるモデルや、事後分布の形状が正規分布とかなり異なるようなモデルで特に有効です。適切に設定されていない尺度パラメーターではNUTSの実行速度が極端に下がることがあります。完全に停止してしまうこともあるでしょう。

サンプリングのための妥当な開始点を設定することも、効果的なサンプリングのために重要です。幸運なことに、NUTSは尺度パラメーターに対して良い推測をすることができます。

もしパラメーター空間のある点をNUTSに渡した場合 (パラメーターの名前をキーとしたdictionary型オブジェクトで、find\_MAPが返す値と同じフォーマット)、それが対数事後分布の局所的な曲率 (curvature) を (ヘッセ行列の対角成分として) 良い尺度ベクトルの検討を付ける点として見るができる。この時結果のは良い値になることが多い。【訳自信なし】

(If you pass a point in parameter space (as a dictionary of variable names to parameter values, the same format as returned by find\_MAP) to NUTS, it will look at the local curvature of the log posterior-density (the diagonal of the Hessian matrix) at that point to make a guess for a good scaling vector, which often results in a good value.)

このMAP推定量は、サンプリングを実施するための良い点として使えることが多いです。

そのほかに自分でベクトルや、尺度行列をNUTSに与えることもできますが、これはアドバンストな使い方です。もし、特定の点においてヘッセ行列を独自の尺度行列として修正したいなら、find\_hessian やfind\_hessian\_diagが使えます。ここで扱っている基本的な線形回帰の例basic\_modelでは、MAPを開始点と尺度点として使った事後分布から、NUTSを使ってサンプリングを行い、2000のデータを生成し、そのグラフを書いています。

これもまた、モデルのコンテキストの中で実施する必要があります。(つまりwith basic\_modelの中)

```
from pymc3 import NUTS, sample

with basic_model:

    # obtain starting values via MAP
    start = find_MAP(fmin=optimize.fmin_powell)

    # draw 2000 posterior samples
    trace = sample(2000, start=start)

out

Assigned <class 'pymc3.step_methods.nuts.NUTS'> to alpha
Assigned <class 'pymc3.step_methods.nuts.NUTS'> to beta
Assigned <class 'pymc3.step_methods.nuts.NUTS'> to sigma_log
[-----100%-----] 2000 of 2000 complete in 5.2 sec
/Users/xxxxxxxxx/anaconda/lib/python2.7/site-packages/theano/gof/cmodule.py:327: RuntimeWarning: numpy.ndarray size changed, may indicate binary incompatibility
rval = __import__(module_name, {}, {}, [module_name])
```

ここで使われているsample 関数は指定されたStep手法で、与えられたいてレーション回数実行し、サンプリング結果がその順序も保持されて含まれているオブジェクトTraceを返します。このTraceオブジェクトはdictionary型のような形でデータが保持されており、変数（パラメーター）の名称を指定することで、値のリストがnumpy.arrayで取得することができます。

配列の最初の次元は、サンプリング インデックスを表します。後ろ部分の変数のシェイプ（配列の次元構造）を表します。変数alpha の最後の5つの値を見たいときは、下記のようにします。

```
trace['alpha'][-5:]

out

array([ 1.21912128,  1.21912128,  1.34578668,  1.01553719,  0.81090504])
```

もし、sigma に対してデフォルトの設定であるNUTSではなく、スライスサンプリングを適用したいときはsample関数の引数stepに指定します。

```
from pymc3 import Slice

with basic_model:

    # obtain starting values via MAP
    start = find_MAP(fmin=optimize.fmin_powell)

    # instantiate sampler
    step = Slice(vars=[sigma])

    # draw 5000 posterior samples
    trace = sample(5000, step=step, start=start)
```

```
out
```

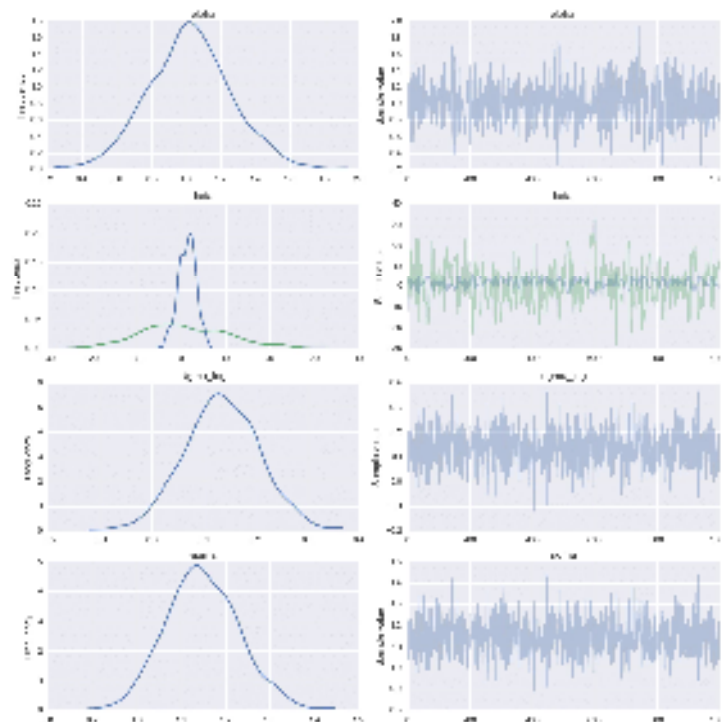
```
Assigned <class 'pymc3.step_methods.nuts.NUTS'> to alpha
Assigned <class 'pymc3.step_methods.nuts.NUTS'> to beta
[-----100%-----] 5000 of 5000 complete in 17.9 se
c
```

## 事後分布の分析(Posterior analysis)

PyMC3は出力されたサンプリングの結果を分析するために、プロットを描いたり、サマリを表示したりする機能があります。事後分布からのサンプルをプロットするにはtraceplotを使います。

```
from pymc3 import traceplot

traceplot(trace[4000:]);
```



(<https://camo.qiitausercontent.com/beff9f44d54cd061e11a1723d07d777f5b2ea3f5/68747470733a2f2f1696974612d696d6167652d73746f72652e73332e616d617a6f6e61777332e636f6d2f302f35303637302f31313061663064632d653639642d663566382d356363352d6564616533333134333464382e706e67>)

グラフの左側の列は、それぞれの確率変数の限界 (marginal) 事後分布の平滑化したヒストグラム (カーネル密度推定を使ったもの) です。右側の列は、生成した順番にプロットしたマルコフ連鎖のサンプルのグラフです。

2つの説明変数の係数に対応している2次元ベクトル変数のbetaについては、対応したそれぞれ2つのヒストグラムとトレースが表示されています。

さらに、summary関数は事後分布の統計量の共通した結果をテキストベースで表示

します。

```
from pymc3 import summary
```

```
summary(trace[4000:])
```

out



けん まつ (/kenmatsu4) @kenmatsu4 (/kenmatsu4)

会社員です。データ解析なことや、統計学なこと、機械学習などについて書いています。【今まで書いた記事一覧】<http://qiita.com/kenmatsu4/items/623514c61166e34283bb>  
(/kenmatsu4) [English Blog] <http://kenmatsu4.tumblr.com>

フォロー

ストック

いいね

76 (http://

(<https://qiita.com/kenmatsu4/items/a0c703762a2429e21793/likers>)

Tweet

56



Toot (<https://qiitadon.com/share?text=%E8%B6%85%E8%A8%B3+PyMC3+Tutorial+%EF%BC%88%E3%83%9E%E3%83%AB%E3%82%B3%E3%83%95%E9%80%A3%E9%8E%96%E3%83%A2%E3%83%B3%E3%>)

55

alpha:

| Mean  | SD    | MC Error | 95% HPD interval |
|-------|-------|----------|------------------|
| 1.022 | 0.242 | 0.008    | [0.579, 1.515]   |

Posterior quantiles:

| 2.5   | 25    | 50    | 75    | 97.5  |
|-------|-------|-------|-------|-------|
| 0.558 | 0.864 | 1.021 | 1.172 | 1.511 |

beta:

| Mean  | SD    | MC Error | 95% HPD interval  |
|-------|-------|----------|-------------------|
| 1.343 | 1.981 | 0.139    | [-2.470, 5.452]   |
| 0.788 | 9.781 | 0.689    | [-17.230, 21.432] |

Posterior quantiles:

| 2.5     | 25     | 50     | 75    | 97.5   |
|---------|--------|--------|-------|--------|
| -2.882  | -0.060 | 1.486  | 2.676 | 5.076  |
| -17.056 | -6.275 | -0.076 | 7.779 | 21.918 |

sigma\_log:

| Mean  | SD    | MC Error | 95% HPD interval |
|-------|-------|----------|------------------|
| 0.136 | 0.072 | 0.003    | [0.001, 0.278]   |

Posterior quantiles:

| 2.5    | 25    | 50    | 75    | 97.5  |
|--------|-------|-------|-------|-------|
| -0.002 | 0.088 | 0.134 | 0.185 | 0.277 |

sigma:

| Mean  | SD    | MC Error | 95% HPD interval |
|-------|-------|----------|------------------|
| 1.148 | 0.083 | 0.003    | [1.001, 1.321]   |

Posterior quantiles:

| 2.5   | 25    | 50    | 75    | 97.5  |
|-------|-------|-------|-------|-------|
| 0.998 | 1.092 | 1.143 | 1.203 | 1.319 |

超訳 PyMC3 Tutorial (マルコフ連鎖モンテカルロ法フレームワーク) その2 「Case study 1: 確率的ボラティリティモデル」  
(<http://qiita.com/kenmatsu4/items/a0c703762a2429e21793>)につづく。



この記事は以下の記事からリンクされています

- MCMCについて整理してみた。 (/shogiai/items/bab2b915df2b8dd6f6f2#\_reference-deba52f0bbb50229f3f4) からリンク 約2年前
- 超訳 PyMC3 Tutorial (マルコフ連鎖モンテカルロ法フレームワーク) その2 「Case study 1: 確率的ボラティリティモデル」 (/kenmatsu4/items/41f22e51671da97da15a#\_reference-b8c2b7d93ba817b81d2c) からリンク 約2年前
- 今までの投稿記事のまとめ (統計学/機械学習/数学 etc) (/kenmatsu4/items/183020c058feac6a779b#\_reference-213375f45443a0baf4d4) からリンク 1年以上前