(/kenmatsu4) @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２ 「Case study 1: 確率的ボラティリティモデル」

Python(/tags/Python) MCMC(/tags/MCMC)
統計学(/tags/%E7%B5%B1%E8%A8%88%E5%AD%A6)
ベイズ統計学 (/tags/%E3%83%99%E3%82%A4%E3%82%BA%E7%B5%B1%E8%A8%88%E5%AD%A6)
マルコフ連鎖モンテカルロ法 (/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)

⚠ 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「超訳」その２「Case study 1: 確率的ボラティリティモデル」です。
今回も要所要所で原文のニュアンスを基に超訳した部分があります。🤔

原文のURL
http://pymc-devs.github.io/pymc3/getting_started/#case-study-1-stochastic-volatility (http://pymc-devs.github.io/pymc3/getting_started/#case-study-1-stochastic-volatility)

コードはGithubにもまとめてあります。
(https://github.com/matsuken92/Qiita_Contents/blob/master/General/case_study1_StochasticVolatirity.ipynb (https://github.com/matsuken92/Qiita_Contents/blob/master/General/case_study1_StochasticVolatirity.ipynb))
⇒ 動かすために一部改変したのでこちらにアップしました。

Case study 1: 確率的ボラティリティモデル

もっと現実的な問題を題材としてPyMC3の使い方を理解するために、株式市場のボラティリティが時間に応じて変わる、というような確率ボラティリティに関するケーススタディを取り扱います。市場からの収益の分布は全く正規分布に従っていません。なので、ここからボラティリティのサンプリングをすることはなかなか難しいです。

ここで扱う例は、400以上のパラメータを持ち、そのためメトロポリスヘイスティングス法のような一般的なサンプリングアルゴリズムを使った場合に自己相関が高くてうまくいかない、というような例です。代わりに NUTS を使うことで劇的に効率が改善します。

モデルについて

資産価格は時間によって変化します(日々の収益の分散として現れている)。とある期間における収益は変化がとても大きく、一方、他の期間ではとても落ち着いていたりします。確率的ボラティリティモデルは、時間によって変化する潜在ボラティリティ変数に対応したモデルです。下記は、NUTSの論文(Hoffman 2014, p. 21) に書かれているものと似たモデルです。

$$\sigma \sim \exp(50)$$
$$\nu \sim \exp(.1)$$
$$s_i \sim N(s_{i-1}, \sigma^{-2})$$
$$\log(y_i) \sim t(\nu, 0, \exp(-2s_i))$$

ここで y は、自由度不明(パラメータν で表される)のステューデントのt分布でモデリングされる日々の収益の時系列を表します。また、スケールパラメーターは潜在プロセス s によって決定されます。 個々のs_i は、日々の潜在対数ボラティリティプロセスの対数ボラティリティです。(4つ目の式のt分布の関数、3つ目の引数に表れている。)

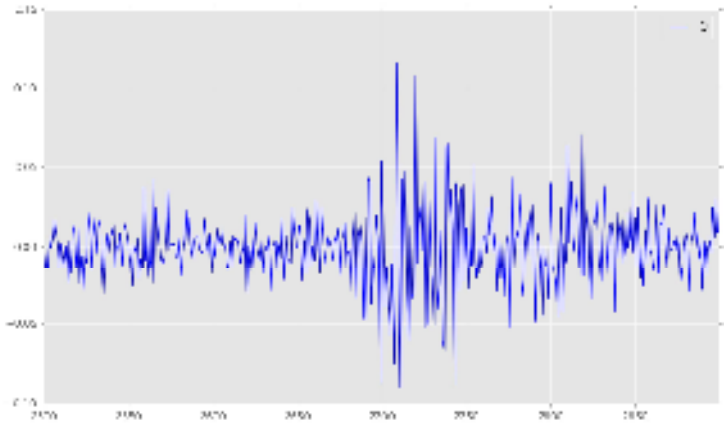
データについて

ここで扱うデータは2008年の金融危機の時のS&P 500の日々の収益データです。

（※ 訳注：Githubに上がっているS&P 500のデータが少し変わってしまっていたので、訳者にて本ページのコードは実行可能とするために、Tutorialオリジナルから少し書き換えたものにしています。基本的にこのままipython notebookにコピペで実行可能なはずです。）

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
returns= pd.read_csv("https://raw.githubusercontent.com/pymc-devs/pymc3/ma
ster/pymc3/examples/data/SP500.csv",
                     header=-1, parse_dates=True)[2500:2900]

plt.style.use('ggplot')
returns.columns =['S&P500']
returns.plot(figsize=(12,7), c="b")
plt.show()
```



(https://camo.qiitausercontent.com/220226da01f4adfd83f91bbfa40bf54ec9e351ad/68747470733a2f2f71696974612d696d6167652d73746f72652e73332e616d617a6f6e61777732e636f6d2f302f35303637302f65306331633666322d353338322d613262372d383562372d333666633262646461666337 2e706e67)

モデルの特定

線形回帰モデルの例と同様に、PyMC3でのモデル特定は、統計的なモデル特定とほぼ同じ表記が使えます。このモデルはいくつかの新しい分布を使います：

- ν とσ に使われている `Exponential` 分布（指数分布）
- 収益に使われているステューデントのt分布
- 潜在ボラティリティに使用されているガウシアンランダムウォーク

PyMC3では、 `Exponential` のような常に正の値をとる変数は、対数の形式に変換されます。これによってサンプリングが頑健（ロバスト）になります。
（訳注：値域が広がるからでしょうか。求むコメント。）

この裏で、 `variableName_log` と呼ばれる制限のない（訳注：多分正の値をとる、という制限がなくなる）空間の変数がモデルのサンプリングのために追加されています。このモデルには、指数事前分布のパラメータとして自由度 `nu` と、ボラティリティプロセスのスケールパラメータ `sigma` が使用されています。

`Beta` や `Uniform`の ような、上側とした側の両方に制限があるような事前分布をもつ確率変数もまた、対数オッズの変換によって制限のない形に変換されます。
PyMC2におけるモデル特定とは違い、通常はモデル特定時に変数に初期値は与える必要はないけれども、引数 `testval` をつかって初期値を任意の分布に与えることもできます。(以降、"test value"と呼ぶ。)これは、分布のtest valueのデフォルト値（通常はその分布の平均、中央値、最頻値が使われる）の上書きをしており、ある値が異常値である場合にそれを除外し、正常値のみを使用したい、という時などに使われることが多いです。
この、分布のTest valueはデフォルトではサンプリングやOptimizationの初期値としてもつかわれますが、こちらも簡単に上書きすることができます。
潜在ボラティリティの `s` は `GaussianRandomWalk` で事前分布が与えられます。名前が示しているように、 `GaussianRandomWalk` は、引数 `shape` によって特徴づけられる、長さ `n` の正規ランダムウォークからなるベクトル値を持つ分布です。

ランダムウォークのinnovationの尺度である `sigma` は、通常、分布したinnovationのprecisionの単位で特定されます。【訳注：訳せませんでした・・・求むヘルプ】これはスカラーでもベクトルでもよいです。
The scale of the innovations of the random walk, sigma, is specified in terms of the precision of the normally distributed innovations and can be a scalar or vector.

```
from pymc3 import Exponential, T, exp, Deterministic, Model, sample, NUTS,
 find_MAP, traceplot
from pymc3.distributions.timeseries import GaussianRandomWalk

with Model() as sp500_model:
    nu = Exponential('nu', 1./10, testval=5.)
    sigma = Exponential('sigma', 1./.02, testval=.1)
    s = GaussianRandomWalk('s', sigma**-2, shape=len(returns))
    volatility_process = Deterministic('volatility_process', exp(-2*s))
    r = T('r', nu, lam=1/volatility_process, observed=returns['S&P500'])
```

対数ボラティリティ過程 `s` を `exp(-2*s)` によってボラティリティ過程に変換していることに注意します。このexp 関数はNumpyの関数というよりはTheanoの関数です。TheanoはNumpyよりも数学関数の大きいサブセットを持っています。
`sp500_model` という名前で宣言された `Model` は最初コンテキストマネージャーとして登場していることに注意します。

フィッティング

事後分布からのサンプルのグラフを描く前に、比較的確率の高い点を探してそれを使うことによって、適切な初期値とすることが推奨されています。全ての変数における事後分布最大化（MAP）により求められた点は、
分布から無限の密度を持つ１点に退化したと考えられます。

しかし、 `log_sigma` と `nu` を決めれば、もうそれは退化ではありませんので、
`log_sigma` と `nu` をデフォルト値で固定して、ボラティリティ過程 `s` の値を変数としてMAPを求めます。（ `sigma` には `testval=.1` と設定していたことを思い出しましょう）

`scipy.optimize` パッケージにある、高次元かつ、400もの確率変数（そのうち多くは `s`)を扱うのにより効果的な記憶制限BFGS (L-BFGS) オプティマイザーを使います。

サンプリングを行うために、短いイニシャル起動を実施します。（高い確率のボリュームゾーンに置くために）そのあと、そのサンプリングトレースの最後の点
`trace[-1]` を初期値として再度サンプリングをスタートします。

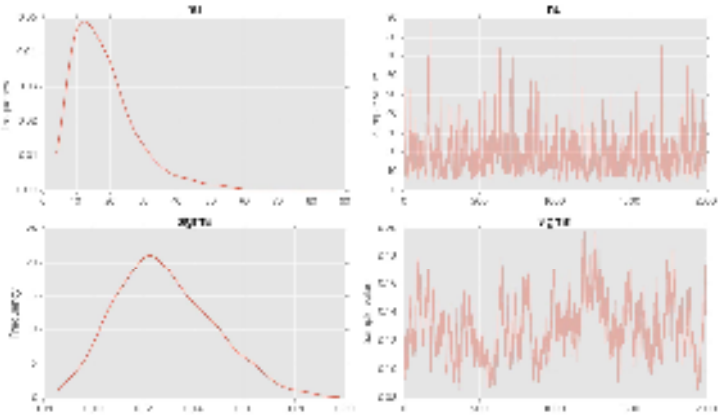
NUTは新しい点をベースに尺度パラメーターを再計算します。この場合、よりよいサンプリング尺度となるためにサンプリングが早くなります。

```
import scipy
with sp500_model:
    start = find_MAP(vars=[s], fmin=scipy.optimize.fmin_l_bfgs_b)
    step = NUTS(scaling=start)
    trace = sample(100, step, progressbar=False)

    # Start next run at the last sampled position.
    step = NUTS(scaling=trace[-1], gamma=.25)
    trace = sample(2000, step, start=trace[-1], progressbar=False)
```

nu と sigma のサンプリング結果のトレース図を下記で見ることができます。

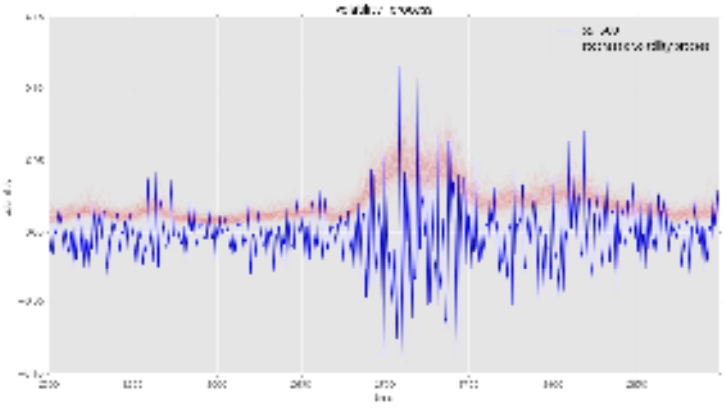
```
traceplot(trace, [nu, sigma]);
```



(https://camo.qiitausercontent.com/db688fd2356b62c1a008e101a2c367a94184776c/68747470733a2f2f71696974612d696d6167652d73746f72652e73332e616d617a6f6e6177732e636f6d2f302f35303637302f65393839356165612d353061662d613064662d363961662d303465326362386232323131362e706e67)

最後に、サンプリングされたボラティリティパスの点を使って分布を、元データの上にプロットします。
Matplotlibの plot 関数の引数 alpha を使って、少し透過させて描画しています。
そのため、点の多い個所は色が濃く表示されています。

```
fig, ax = plt.subplots(figsize=(15, 8))
returns.plot(ax=ax)
ax.plot(returns.index, 1/np.exp(trace['s',::30].T), 'r', alpha=.03);
ax.set(title='volatility_process', xlabel='time', ylabel='volatility');
ax.legend(['S&P500', 'stochastic volatility process'])
```



(https://camo.qiitausercontent.com/9eaacc9cdb5a8d073e83b117c0247786f5be31c5/68747470733a2f2f71696974612d696d6167652d73746f72652e73332e616d617a6f6e6177732e636f6d2f302f35303637302f33643430663732652d346534632d623162622d613866662d396635636266631303963623 42e706e67)

このケーススタディで、2008年の金融危機の時期のボラティリティの増加を、このモデルで正しく推定できました。さらに、高次元かつ、依存構造のあるランダムウォーク分布であるため非常に複雑なモデルですが、PyMC3に実装されたNUTSは、うまく、そして簡単に事後分布を推定することができました。

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

会社員です。データ解析的なことや、統計学的なこと、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 【English Blog】 http://kenmatsu4.tumblr.com

フォロー

関連ページ

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
http://qiita.com/kenmatsu4/items/a0c703762a2429e21793
(http://qiita.com/kenmatsu4/items/a0c703762a2429e21793)

この記事は以下の記事からリンクされています

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-d0d3f232714e878c9045) からリンク 1年以上前