動態マクロ経済学

Week 6

佐藤健治
sato@eco.osakafu-u.ac.jp

2020/6/12

準備運動：IPython を起動してください

```
import numpy as np
import matplotlib.pyplot as plt
rng = np.random.default_rng(100)   # 100 は乱数の種

A = rng.choice(range(20), size=(3, 3))
A

B = rng.normal(size=(3,3))
B

x = rng.choice(range(10), 3)
x
```

numpy.matrix は非推奨

- NumPy には `Matrix` 型というのがあって，行列を表現するのに使われていました。今は推奨されておりません。比較的最近出た書籍とかでも使っているものがありますが，自分で書くコードには使わないでください。
  - https://numpy.org/doc/1.18/reference/generated/numpy.matrix.html
- 普通に `array` 型を使うとよいです。
  - 2 次元配列が行列

行列演算

数学で習う行列の掛け算は `@` を使う。

```
A + B
A - B
A @ B
A @ x
```

クイズ: 次のコードは何を計算している？

```
A * B
A / B
A + x
```

落とし穴

- NumPy の 1 次元配列（ベクトル）は，状況に応じて行ベクトル・列ベクトルになったりする。
  - `A @ x` の `x` は列ベクトルのように振る舞う。
  - `A + x`（数学的な対応物のない式）の `x` は行ベクトルを 3 つ重ねた行列のように振る舞う。
- 行列の演算をメインとした計算では，数式上は列ベクトルと想定することが多いので，ベクトルを 2 次元配列（1 列だけからなる行列）にしておくと分かりやすいです。

```
x.shape = (3, 1)
A @ x
```

```
array([[225],
       [129],
       [127]])
```

落とし穴 (2)

- `@` はスカラーとの積として使えないので，スカラー時系列も 1×1 行列の系列とみなすほうが何かと便利。
- 次のコードは失敗する。

```
B = np.array([[1], [0]])
eps = np.array([1, 2])
B @ eps[0]
```

- eps の形状を変えた次のコードはうまくいく。

```
eps.shape = (*eps.shape, 1, 1)
B @ eps[0]
```

ベクトル時系列の表現

- ベクトル時系列のデータは表形式で表現できる。

	x	y	z
$t=0$	x_0	y_0	z_0
$t=1$	x_1	y_1	z_1
$t=2$	x_2	y_2	z_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$

- これは行列で表示できる。

$$\begin{bmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

- ベクトルの変化と捉えると....

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}, \quad \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}, \quad \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}, \quad \cdots$$

時系列を 3 次元配列で表現する

```
Y = np.arange(9)
Y.shape = (3, 3, 1)
Y
```

```
array([[[0],
        [1],
        [2]],

       [[3],
        [4],
        [5]],

       [[6],
        [7],
        [8]]])
```

```
Y[0]
```

```
array([[0],
       [1],
       [2]])
```

```
Y[1]
```

```
array([[3],
       [4],
       [5]])
```

- 行列とベクトルの演算が頻繁に現れる線形時系列モデルのシミュレーションでは 3 次元配列の表現が一番使いやすい（と思う）。
  - 各時点のデータが列ベクトルになる。
  - 全部を 2 次元配列にしておくと `@` が使いやすい。
- 可視化やデータの保存には行列形式（表形式）が自然なので，行き来できるようにする。

```
X = Y.squeeze()
X
X.shape = (*X.shape, 1)
X
```

線形時系列モデル

どういう問題を扱いたいか？

- エンダース『実証のための計量時系列分析』にならって，Samuelson (1939) のモデルを紹介する。
  - 動機づけのため。メイントピックより発展的な内容。
- GDP の動学モデルとして以下のようなモデルを考える。

$$
\begin{aligned}
y_t &= c_t + i_t \\
c_t &= \eta y_{t-1} + \varepsilon_{ct} \\
i_t &= \beta(c_t - c_{t-1}) + \varepsilon_{it}
\end{aligned}
$$

  - y は GDP, c は消費，i は設備投資。1 つ目の式は，自由な閉鎖経済の国民所得勘定の恒等式に対応する。
  - 2 つ目の式は消費関数
  - 3 つ目の式は「加速度原理」と呼ばれる式。消費の増分が設備投資を引き起こす。

式変形

$$\begin{bmatrix} 1 & -1 & -1 \\ 0 & 1 & 0 \\ 0 & -\beta & 1 \end{bmatrix} \begin{bmatrix} y_t \\ c_t \\ i_t \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ \eta & 0 & 0 \\ 0 & -\beta & 0 \end{bmatrix} \begin{bmatrix} y_{t-1} \\ c_{t-1} \\ i_{t-1} \end{bmatrix} + \begin{bmatrix} 0 \\ \varepsilon_{ct} \\ \varepsilon_{it} \end{bmatrix}$$

$$\Downarrow$$

$$\underbrace{\begin{bmatrix} y_t \\ c_t \\ i_t \end{bmatrix}}_{\boldsymbol{y}_t} = \underbrace{\begin{bmatrix} 1 & 1+\beta & 1 \\ 0 & 1 & 0 \\ 0 & \beta & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ \eta & 0 & 0 \\ 0 & -\beta & 0 \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} y_{t-1} \\ c_{t-1} \\ i_{t-1} \end{bmatrix}}_{\boldsymbol{y}_{t-1}} + \underbrace{\begin{bmatrix} 1 & 1+\beta & 1 \\ 0 & 1 & 0 \\ 0 & \beta & 1 \end{bmatrix} \begin{bmatrix} 0 \\ \varepsilon_{ct} \\ \varepsilon_{it} \end{bmatrix}}_{\boldsymbol{\varepsilon}_t}$$

線形時系列モデル

$$\boldsymbol{y}_t = \boldsymbol{A}\boldsymbol{y}_{t-1} + \varepsilon_t$$

- 関心のある時系列は $\boldsymbol{y}$
- ε は外的なショック。（平均ゼロ，分散均一，予測不可能）
- 行列とベクトルの掛け算と足し算で構成される「漸化式」
  → 線形時系列モデル
- ベクトル時系列の今の値が自分自身の過去の値で決まっているモデル。VAR モデル（Vector AutoRegressive モデル）

推定とシミュレーション

- $\{\boldsymbol{y}_t\} \longrightarrow \boldsymbol{A}$
  - 推定。
  - 推定結果は，経済構造の解釈や，後に述べるシミュレーションに使う。
- $\boldsymbol{A} \longrightarrow \{\boldsymbol{y}_t\}$
  - シミュレーション。
  - $\boldsymbol{A}$ は推定する。あるいは，パラメータの経済的な意味と他の実証分析から外生的なパラメータを設定することもある。カリブレーション
  - 「投資を刺激するショックが起こると，GDP はどの程度変化するか？」

重要な性質 (1)

$$\boldsymbol{y}_t = \boldsymbol{A}^n \boldsymbol{y}_{t-n} + \underbrace{\left(\varepsilon_t + \boldsymbol{A}\varepsilon_{t-1} + \boldsymbol{A}^2\varepsilon_{t-2} + \cdots \boldsymbol{A}^{n-1}\varepsilon_{t-n+1}\right)}_{\text{平均ゼロ}}$$

- 現在の $\boldsymbol{y}_t$ が大昔の影響を受けずにゼロの周りをウロウロするなら，$\boldsymbol{A}^n \to \boldsymbol{O}$ (零行列) が成り立っている。
  - $\boldsymbol{A}^n \to \boldsymbol{O}$: 行列 $\boldsymbol{A}$ の「**安定性**」
  - $\boldsymbol{A}$ の固有値の絶対値が全部 1 未満であればよい。
- GDP，消費，投資はゼロの周りをウロウロするわけはないのだけど，基準となる水準からの変動だと見ればよい。
  - 変動がおおよそゼロ，という状態は経済の「均衡」に対応

重要な性質 (2) と，今後の見通し

- シミュレーションができるモデルは，過去から未来に進行するということ。
  - $\boldsymbol{y}_{t-1} \longrightarrow \boldsymbol{y}_t$
  - この前提が崩れるとシミュレーションできない。
  - 線形であれ，非線形であれ，この形式だとシミュレーションは簡単。
  - 推定は難しい。非線形だともっと難しい。
- 経済モデルは「将来の予想」が重要。
  - $\boldsymbol{y}_{t+1} \longrightarrow \boldsymbol{y}_t$
  - 将来の予想から今が決まるモデルであっても，シミュレーションは $\boldsymbol{y}_{t-1} \longrightarrow \boldsymbol{y}_t$ のモデルを作った後に実行する。
  - Blanchard-Khan の方法，ダイナミックプログラミング
  - この授業はどこまで行けるのか？！

AR モデル

- 上のモデルをシミュレーションしてもいいのだけど，もう少し汎用的なモデルを紹介する。
- AR (2) 過程

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + \varepsilon_t$$

  - ε_t は「ホワイトノイズ」
  - y_t は株価の変化率みたいな変数だと思えばよい。

状態空間表現

- 次のような変数を導入する。

$$\boldsymbol{x}_t = \begin{bmatrix} y_t \\ y_{t-1} \end{bmatrix}$$

- 先程の AR(2) の式が次の行列方程式と一致することを確認せよ。

$$\begin{aligned} \boldsymbol{x}_t &= \boldsymbol{A}\boldsymbol{x}_{t-1} + \boldsymbol{B}\varepsilon_t \\ y_t &= \boldsymbol{C}\boldsymbol{x}_t \end{aligned}$$

ただし，

$$\boldsymbol{A} = \begin{bmatrix} a_1 & a_2 \\ 1 & 0 \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad \boldsymbol{C} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

解の表現

- 計算は省略するが，y_t は次のように書ける。

$$y_t = \boldsymbol{C}\boldsymbol{A}^n \boldsymbol{x}_{t-n} + \sum_{k=0}^{n-1} \boldsymbol{C}\boldsymbol{A}^k \boldsymbol{B}\varepsilon_{t-k},$$

  - ここでも $\boldsymbol{A}^n$ が重要な働きをする。
  - $\boldsymbol{A}$ の固有値が全部絶対値 1 未満なら，$\boldsymbol{A}^n \to \boldsymbol{O}$ となって，この AR(2) 過程は「よい振る舞い」（弱定常性）を示す。
- コンピュータシミュレーションには逐次的な計算のほうがよいので，この表現は使わない。

係数行列

- 次のように定義する。

```
A = np.array([[0.6, 0.3],
              [1.0, 0.0]])
B = np.array([[1.0],
              [0.0]])
C = np.array([[1.0, 0.0]])
```

- 大かっこの数に注意。
- AR(2) の表現に変換してみてください。

安定性

```
E, V = np.linalg.eig(A)
np.abs(E)
```

```
array([0.9244998, 0.3244998])
```

```
np.linalg.matrix_power(A, 200)
```

```
array([[0.00000011, 0.00000004],
       [0.00000012, 0.00000004]])
```

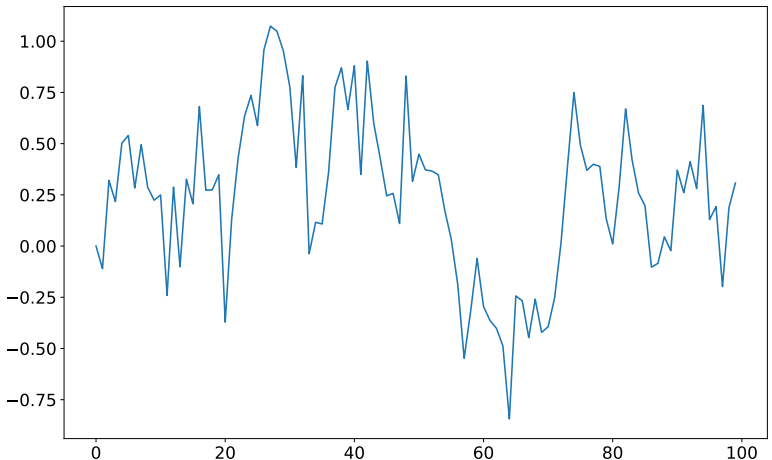
シミュレーションのコード

```
T = 100
x, y = np.empty((T, 2, 1)), np.empty((T, 1, 1))
rng = np.random.default_rng(123)
eps = rng.normal(loc=0, scale=0.3, size=(T, 1, 1))

x[0] = np.array([[0.0],
                 [0.0]])
y[0] = np.array([[0.0]])

for t in range(1, T):
    x[t] = A @ x[t-1] + B @ eps[t]
    y[t] = C @ x[t]
```

```
plt.plot(y.squeeze())
plt.show()
```



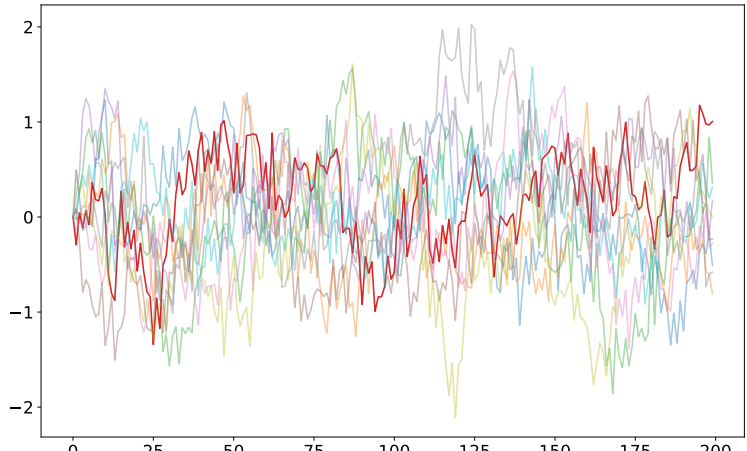
たくさんのシミュレーション

```
for i in range(10):
    T = 200
    x, y = np.empty((T, 2, 1)), np.empty((T, 1, 1))
    eps = rng.normal(loc=0, scale=0.3, size=(T, 1, 1))
    x[0] = np.array([[0.0], [0.0]])
    y[0] = np.array([[0.0]])

    for t in range(1, T):
        x[t] = A @ x[t-1] + B @ eps[t]
        y[t] = C @ x[t]

    plt.plot(y.squeeze(), alpha=(1 if i==3 else 0.4))

plt.show()
```


2
1
0
−1
−2
0
25
50
75
100
125
150
175
200

推定について一言

- AR モデルの推定に使える statsmodels ライブラリの使い方をテキストの方で簡単に説明しています。
- ACF, PACF の見方や推定についての詳細は，エンダースや沖本『経済・ファイナンスデータの計量時系列分析』などを参照

課題

- これまではコピペで完了する課題が多かったのですが，少しずつ負荷を上げていきます。
- 課題のポイント
  - `rng.normal(loc=0, scale=0.3, size=(T, 1, 1))` のパラメータ `loc` と `scale` を変化させたときに，AR(2) モデルのシミュレーション結果はどのように変わるか。
  - グラフの見た目とこれらのパラメータはどのような関係にあるか。いろいろなシミュレーション結果の観察と，テキスト（4 章）に書かれている数学的な結果を見比べて議論してください。
  - $\boldsymbol{A}^n \to \boldsymbol{O}$ とならない係数を持つモデルをシミュレーションするとどうなるか。例示する。