

Time Series Analysis in Python with statsmodels

Wes McKinney¹ Josef Perktold² Skipper Seabold³

¹Department of Statistical Science
Duke University

²Department of Economics
University of North Carolina at Chapel Hill

³Department of Economics
American University

10th Python in Science Conference, 13 July 2011

What is statsmodels?

- A library for statistical modeling, implementing standard statistical models in Python using NumPy and SciPy
- Includes:
 - Linear (regression) models of many forms
 - Descriptive statistics
 - Statistical tests
 - Time series analysis
 - ...and much more

What is Time Series Analysis?

- Statistical modeling of time-ordered data observations
- Inferring structure, forecasting and simulation, and testing distributional assumptions about the data
- Modeling dynamic relationships among multiple time series
- Broad applications e.g. in economics, finance, neuroscience, signal processing...

Talk Overview

- Brief update on statsmodels development
- Aside: user interface and data structures
- Descriptive statistics and tests
- Auto-regressive moving average models (ARMA)
- Vector autoregression (VAR) models
- Filtering tools (Hodrick-Prescott and others)
- Near future: Bayesian dynamic linear models (DLMs), ARCH / GARCH volatility models and beyond

Statsmodels development update

- We're now on GitHub! Join us:

`http://github.com/statsmodels/statsmodels`

- Check out the slick Sphinx docs:

`http://statsmodels.sourceforge.net`

- Development focus has been largely **computational**, i.e. writing correct, tested implementations of all the common classes of statistical models

- Major work to be done on providing a nice integrated **user interface**
- We **must** work together to close the gap between R and Python!
- Some important areas:
 - Formula framework, for specifying model design matrices
 - Need integrated rich statistical data structures (**pandas**)
 - Data visualization of results should always be a few keystrokes away
 - Write a “Statsmodels for R users” guide

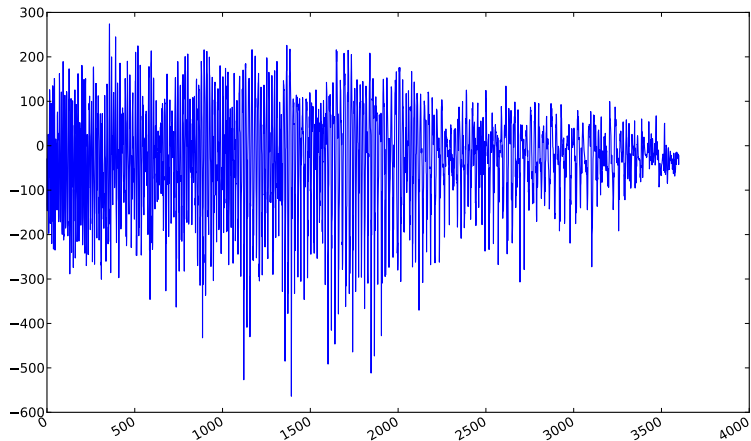
Aside: statistical data structures and user interface

- While I have a captive audience...
- **Controversial fact:** **pandas** is the only Python library *currently* providing data structures matching (and in many places exceeding) the richness of R's data structures (for statistics)
 - Let's have a BoF session so I can justify this statement
- Feedback I hear is that end users find the fragmented, incohesive set of Python tools for data analysis and statistics to be confusing, frustrating, and certainly not compelling them to use Python...
 - (Not to mention the packaging headaches)

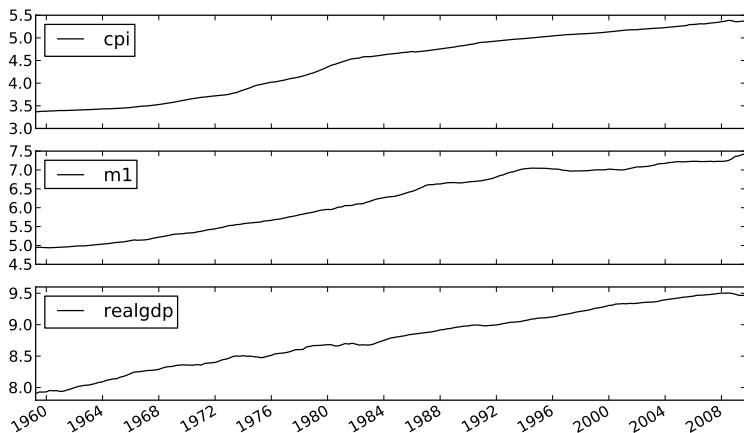
Aside: statistical data structures and user interface

- We need to “commit” **ASAP** (not 12 months from now) to a high level data structure(s) as the “primary data structure(s) for statistical data analysis” and communicate that clearly to end users
 - Or we might as well all start programming in R...

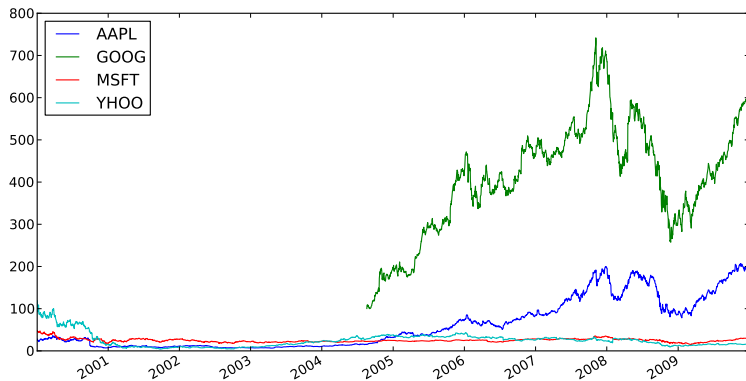
Example data: EEG trace data



Example data: Macroeconomic data



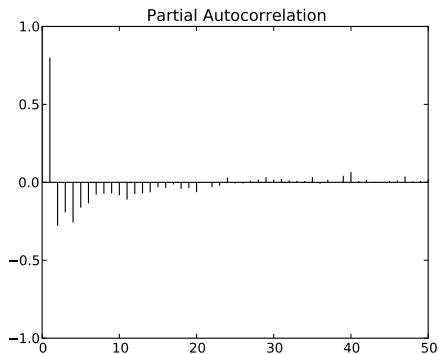
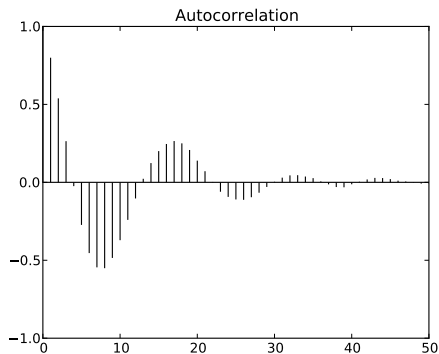
Example data: Stock data



Descriptive statistics

- Autocorrelation, partial autocorrelation plots
- Commonly used for identification in ARMA(p,q) and ARIMA(p,d,q) models

```
acf = tsa.acf(eeg, 50)  
pacf = tsa.pacf(eeg, 50)
```



Statistical tests

- Ljung-Box test for zero autocorrelation
- Unit root test for cointegration (Augmented Dickey-Fuller test)
- Granger-causality
- Whiteness (iid-ness) and normality
- See our conference paper (when the proceedings get published!)

Autoregressive moving average (ARMA) models

- One of most common univariate time series models:

$$y_t = \mu + a_1 y_{t-1} + \dots + a_k y_{t-p} + \epsilon_t + b_1 \epsilon_{t-1} + \dots + b_q \epsilon_{t-q}$$

where $E(\epsilon_t, \epsilon_s) = 0$, for $t \neq s$ and $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$

- Exact log-likelihood can be evaluated via the Kalman filter, but the “conditional” likelihood is easier and commonly used
- statsmodels has tools for simulating ARMA processes with known coefficients a_i , b_i and also estimation given specified lag orders

```
import scikits.statsmodels.tsa.arima_process as ap
ar_coef = [1, .75, -.25]; ma_coef = [1, -.5]
nobs = 100
y = ap.arma_generate_sample(ar_coef, ma_coef, nobs)
y += 4 # add in constant
```

ARMA Estimation

- Several likelihood-based estimators implemented (see docs)

```
model = tsa.ARMA(y)
result = model.fit(order=(2, 1), trend='c',
                  method='css-mle', disp=-1)

result.params
# array([ 3.97, -0.97, -0.05, -0.13])
```

- Standard model diagnostics, standard errors, information criteria (AIC, BIC, ...), etc available in the returned `ARMAResults` object

Vector Autoregression (VAR) models

- Widely used model for modeling multiple (K -variate) time series, especially in macroeconomics:

$$Y_t = A_1 Y_{t-1} + \dots + A_p Y_{t-p} + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \Sigma)$$

- Matrices A_i are $K \times K$.
- Y_t must be a *stationary* process (sometimes achieved by differencing). Related class of models (VECM) for modeling nonstationary (including cointegrated) processes

Vector Autoregression (VAR) models

```
>>> model = VAR(data); model.select_order(8)
```

```
VAR Order Selection
```

```
=====
```

	aic	bic	fpe	hqic
0	-27.83	-27.78	8.214e-13	-27.81
1	-28.77	-28.57	3.189e-13	-28.69
2	-29.00	-28.64*	2.556e-13	-28.85
3	-29.10	-28.60	2.304e-13	-28.90*
4	-29.09	-28.43	2.330e-13	-28.82
5	-29.13	-28.33	2.228e-13	-28.81
6	-29.14*	-28.18	2.213e-13*	-28.75
7	-29.07	-27.96	2.387e-13	-28.62

```
=====
```

* Minimum

Vector Autoregression (VAR) models

```
>>> result = model.fit(2)
>>> result.summary() # print summary for each variable
<snip>
Results for equation m1
=====
              coefficient      std. error  t-stat  prob
-----
const          0.004968         0.001850   2.685  0.008
L1.m1          0.363636         0.071307   5.100  0.000
L1.realgdp    -0.077460         0.092975  -0.833  0.406
L1.cpi        -0.052387         0.128161  -0.409  0.683
L2.m1          0.250589         0.072050   3.478  0.001
L2.realgdp    -0.085874         0.092032  -0.933  0.352
L2.cpi         0.169803         0.128376   1.323  0.188
=====
<snip>
```

Vector Autoregression (VAR) models

```
>>> result = model.fit(2)
>>> result.summary() # print summary for each variable
<snip>
```

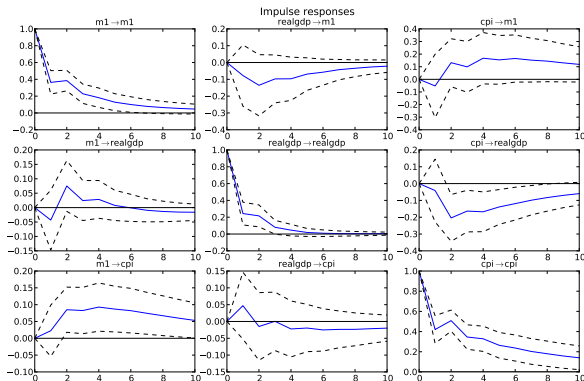
Correlation matrix of residuals

	m1	realgdp	cpi
m1	1.000000	-0.055690	-0.297494
realgdp	-0.055690	1.000000	0.115597
cpi	-0.297494	0.115597	1.000000

VAR: Impulse Response analysis

- Analyze systematic impact of unit “shock” to a single variable

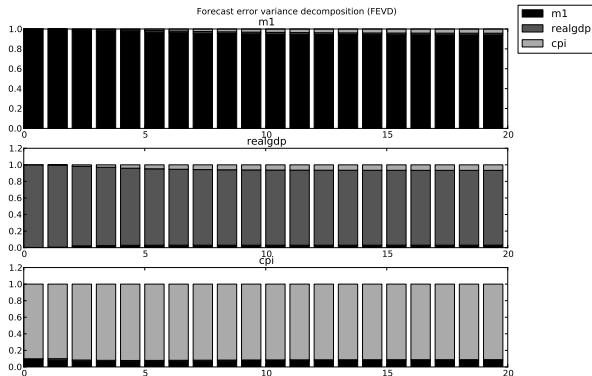
```
irf = result.irf(10)  
irf.plot()
```



VAR: Forecast Error Variance Decomposition

- Analyze contribution of each variable to forecasting error

```
fevd = result.fevd(20)
fevd.plot()
```

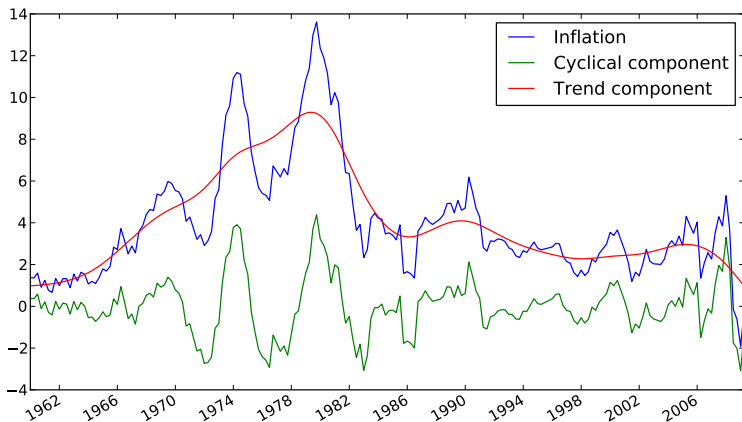


VAR: Statistical tests

```
In [137]: result.test_causality('m1', ['cpi', 'realgdp'])
Granger causality f-test
=====
      Test statistic      Critical Value      p-value      df
-----
           1.248787           2.387325           0.289   (4, 579)
=====
H_0: ['cpi', 'realgdp'] do not Granger-cause m1
Conclusion: fail to reject H_0 at 5.00% significance level
```

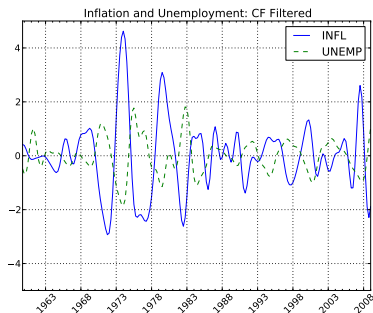
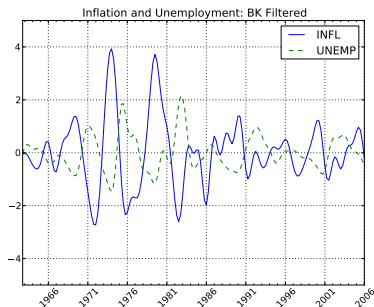
Filtering

- Hodrick-Prescott (HP) filter separates a time series y_t into a trend τ_t and a cyclical component ζ_t , so that $y_t = \tau_t + \zeta_t$.



Filtering

- In addition to the HP filter, 2 other filters popular in finance and economics, Baxter-King and Christiano-Fitzgerald, are available
- We refer you to our paper and the documentation for details on these:



Preview: Bayesian dynamic linear models (DLM)

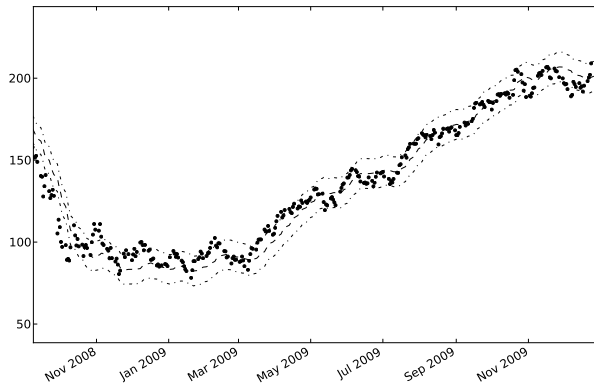
- A state space model by another name:

$$y_t = F_t' \theta_t + \nu_t, \quad \nu_t \sim \mathcal{N}(0, V_t)$$
$$\theta_t = G \theta_{t-1} + \omega_t, \quad \omega_t \sim \mathcal{N}(0, W_t)$$

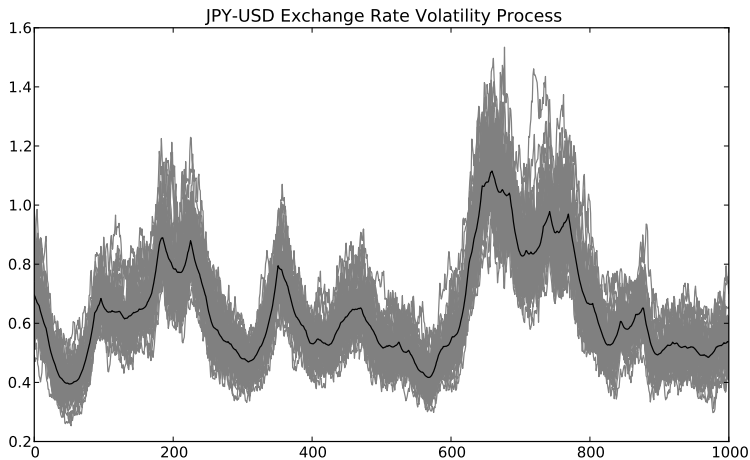
- Estimation of basic model by Kalman filter recursions. Provides elegant way to do time-varying linear regressions for forecasting
- Extensions: multivariate DLMs, stochastic volatility (SV) models, MCMC-based posterior sampling, mixtures of DLMs

Preview: DLM Example (Constant+Trend model)

```
model = Polynomial(2)
dlm = DLM(close_px['AAPL'], model.F, G=model.G, # model
          m0=m0, C0=C0, n0=n0, s0=s0, # priors
          state_discount=.95) # discount factor
```



Preview: Stochastic volatility models



Future: sandbox and beyond

- ARCH / GARCH models for volatility
- Structural VAR and error correction models (ECM) for cointegrated processes
- Models with non-normally distributed errors
- Better data description, visualization, and interactive research tools
- More sophisticated Bayesian time series models

Conclusions

- We've implemented many foundational models for time series analysis, but the field is very broad
- User interface can and should be much improved
- Repo: <http://github.com/statsmodels/statsmodels>
- Docs: <http://statsmodels.sourceforge.net>
- Contact: pystatsmodels@googlegroups.com