

Python and Asset Pricing

Reading Group Presentation

Simina Pușcașu

June 12, 2020



Table of contents

1. Python Scientific Libraries

Overview

NumPy

SciPy

Pandas

Numba

2. Object-oriented programming

3. Application

Python Scientific Libraries

Main libraries:

- [NumPy](#)
- [SciPy](#)
- [Pandas](#)
- [Numba](#)
- [Matplotlib](#) (not included in this presentation)

* Text in red contains links to resources.

- How to: `import numpy as np`
- Scope: fast array processing
- Data structure: `np.array()`
- Similar functionalities to Matlab:
[See here a comprehensive list of equivalent expressions](#)

- How to:
`from scipy import subpackage 1, ..., subpackage n`
- Scope: algorithms and functions built on NumPy
- Subpackages must be imported separately

- `constants` : Physical and mathematical constants
- `integrate` : Integration and ODE solvers
- `interpolate` : Interpolation and smoothing splines
- `linalg` : Linear algebra
- `optimize` : Optimization and root-finding routines
- `stats` : Statistical distributions and functions

- Optimization
 - local: `minimize(...,method = '...')`
 - global: `differential_evolution()`
- Root finding
 - 1-D: `bisect()`, `newton()`, `fixed_point()`
 - n-D: `root()`
- Linear programming
 - `linprog()`
- Other (legacy) functions:
 - `fmin()`, `fsolve()`

- How to: `import pandas as pd`
- Scope: data analysis (handling)
- Data structures: `pd.Series()`, `pd.DataFrames()`
- Easily translates to R
- Similar functionalities in Stata
- Related libraries:
 - statsmodels
 - linearmodels
 - scikit-Learn

- How to: `import statsmodels.api as sm`
- Scope: statistical models, tests, data exploration
- Useful models are implemented as classes or modules:
 - class `sm.OLS()`
 - module `sm.tsa` with classes such as :
 - `.ar_model.AutoReg()`
 - `.arima_model.ARMA()`
 - `.regime_switching.markov_regression.MarkovRegression()`
 - module `statsmodels.tsa.statespace` with corresponding classes and methods
- Check [User guide](#) for full list of functionalities

- How to: `from linearmodels.ModuleName import ModelName`
- Scope: complements statsmodels with
 - **panel data models** (including Fama-MacBeth routine)
 - **IV estimators**
 - **factor asset pricing models** (including GMM)
 - **system regressions** (SUR, SURE, 3SLS, GMM)

- How to: `from sklearn import` subpackage
- Scope:
 - Supervised learning
 - Unsupervised learning
 - Model selection and evaluation
 - Inspection
 - Visualization
 - Data transformation and importing

- `linear_model`
 - `linear_model.LinearRegression()`
 - `linear_model.Ridge()`
 - `linear_model.Lasso()`
 - `LogisticRegression()`
- `svm`
- `neighbors`
- `tree`

- Scope: Python code \rightarrow fast-running optimized code
- Works for functions and classes
- Key aspect: infers input type to speed things up

```
from numba import jit
import random

@jit(nopython=True) # decorator
def monte_carlo_pi(nsamples):
    acc = 0
    for i in range(nsamples):
        x = random.random()
        y = random.random()
        if (x ** 2 + y ** 2) < 1.0:
            acc += 1
    return 4.0 * acc / nsamples
```

Object-oriented programming

- In Python everything is an object.
- Examples of objects:
 - functions
 - modules (scripts)
 - variables such as integers, strings...
- An object is characterized by
 - type (string, integer, function, class etc.)
 - identity (its address in memory)
 - content (data)
 - methods (functions/ callable attributes)

- A **Class** acts as a blueprint for creating your own objects
- An **object** is therefore an instance of a class
- Objects will have their own content and can use the methods provided in the class definition
- Attributes of objects = $\{content, methods\}$

Application

$$x_{t+1} = \rho x_t + \varphi_x \sigma_t \varepsilon_{t+1}^x \quad (1)$$

$$g_{t+1}^c = \mu + x_t + \sigma_t \varepsilon_{t+1}^c \quad (2)$$

$$g_{t+1}^d = \mu_d + \phi x_t + \varphi_d \sigma_t \varepsilon_{t+1}^d + \varphi_{dc} \sigma_t \varepsilon_{t+1}^c \quad (3)$$

$$\sigma_{t+1}^2 = \sigma^2 + \nu_1(\sigma_t^2 - \sigma^2) + \varphi_s \varepsilon_{t+1}^\sigma \quad (4)$$

$$x_{t+1} = \rho x_t + \varphi_x \sigma_t \varepsilon_{t+1}^x \quad (1)$$

$$g_{t+1}^c = \mu + x_t + \sigma_t \varepsilon_{t+1}^c \quad (2)$$

$$g_{t+1}^d = \mu_d + \phi x_t + \varphi_d \sigma_t \varepsilon_{t+1}^d + \varphi_{dc} \sigma_t \varepsilon_{t+1}^c \quad (3)$$

$$\sigma_{t+1}^2 = \sigma^2 + \nu_1 (\sigma_t^2 - \sigma^2) + \varphi_s \varepsilon_{t+1}^\sigma \quad (4)$$

$$r_{t+1}^c = \kappa_0 + \kappa_1 w c_{t+1} - w c_t + g_{t+1}^c \quad (5)$$

$$w c_t = A_0 + A_1 x_t + A_2 \sigma_t^2 \quad (6)$$

$$r_{t+1}^d = \kappa_0^m + \kappa_1^m p d_{t+1} - p d_t + g_{t+1}^d \quad (7)$$

$$p d_t = A_0^m + A_1^m x_t + A_2^m \sigma_t^2 \quad (8)$$

- Build LLR model class
- Instances of the class: BY2004 with and without stochastic volatility for various parametrizations
- Class methods:
 - Simulate the stochastic variables
 - Produce log-linear approximation solution
 - Produce projection method solution (implemented: the collocation method as in Pohl, Schmedders and Wilms (2018))
 - Compute asset pricing moments
 - *Methods for visualizing model output

- Build LLR model class
- Instances of the class: BY2004 with and without stochastic volatility for various parametrizations
- Class methods:
 - Simulate the stochastic variables
 - Produce log-linear approximation solution
 - Produce projection method solution (implemented: the collocation method as in Pohl, Schmedders and Wilms (2018))
 - Compute asset pricing moments
 - *Methods for visualizing model output

- The time invariant Euler equation for consumption is:

$$0 = \int \left[1 - \exp \left(\theta \left(\log \delta + \left(1 - \frac{1}{\psi} \right) \Delta c(s'|s) + z(s') - \log(\exp(z(s)) - 1) \right) \right) \right] df_{\epsilon}$$

- $s = (x, \sigma^2)$
- $s' = (x, \sigma^2; \varepsilon)$
- z as wc ratio

- Approximate the wc ratio using Chebyshev polynomial series,
 $\Lambda_k = T_k\left(\frac{2x-a-b}{b-a}\right)$, $\Lambda_h = T_h\left(\frac{2\sigma^2-c-d}{d-c}\right)$, with $x \in [a, b]$, $\sigma^2 \in [c, d]$:

$$\hat{z}(x, \sigma^2; \alpha) = \sum_{k=0}^n \sum_{h=0}^n \alpha_{kh} \Lambda_k(x) \Lambda_h(\sigma^2)$$

- $T_0(x) = 1$, $T_1(x) = x$, $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$

$$\hat{z}(x, \sigma^2; \alpha) = \sum_{k=0}^n \sum_{h=0}^n \alpha_{kh} \Lambda_k(x) \Lambda_h(\sigma^2)$$

- The coefficients in α are obtained by imposing that the Euler equation holds at each $s_{ij} = (x_i, \sigma_j^2)$ pair with $i, j \in \{1, \dots, n\}$ (approximation nodes) with $s'_{ij}(x_i, \sigma_j^2; \epsilon)$ as next period state:

$$0 = \int \left[1 - \exp \left(\theta \left(\log \delta + \left(1 - \frac{1}{\psi} \right) \Delta c(s'_{ij} | s_{ij}) + \hat{z}(s'_{ij}, \alpha) - \log(\exp(\hat{z}(s_{ij}, \alpha)) - 1) \right) \right) \right] df_{\epsilon}$$

$$0 = \int \left[1 - \exp \left(\theta \left(\log \delta + \left(1 - \frac{1}{\psi} \right) \Delta c(s'_{ij} | s_{ij}) + \hat{z}(s'_{ij}, \alpha) - \log(\exp(\hat{z}(s_{ij}, \alpha)) - 1) \right) \right) \right] df_{\varepsilon}$$

- Finally, the integral above is computed using Gauss-Hermite quadrature of degree \mathcal{L} over each of the 4 normal shocks:

$$0 = \sum_{\ell=1}^{(\mathcal{L})^4} \left[1 - \exp \left(\dots \right) \right] w_{\ell}(\varepsilon)$$

- A similar procedure is performed for the pd ratio. Details in Judd (1998) or in the appendix of Pohl, Schmedders and Wilms (2018)

- My Python code
- The original Matlab code for PSW(2018) (downloads code on click)
- 1:1 translation of Matlab code to Python of PSW(2018)