Python for Finance

Risk Measurement

Gabor Nguyen

SOLVENCY ANALYTICS

Outline

Motivation

- In quantitative finance arguably the most important task/challenge is the financial risk measurement and assessment. The majority of the financial engineer's task is connected to this field.
- Risk Measurement for a financial institution is very data intense and computational expensive process.
- Using the adequate, state of the art technologies are key to keep up with the speed of the financial market innovation, and to be able to meet the regulatory requirements.

Types of risk measures

There is a wide range of risk measures that spread through the Market.

- Volatility (σ)
- Value at Risk (VaR)
- Expected Shortfall (ES)
- Conditional Value at Risk (CVaR) - In case of continuous distributions same as ES
- Average Drawdown, Maximum Drawdown
- etc.

VaR and ES

Formal definitions:

1. Value at Risk:

$$VaR_\alpha(X) = -\inf\{x \in \mathbb{R} : F_X(x) > \alpha\} = -F_X^{-1}(\alpha) \qquad (1)$$

2. Expected Shortfall:

$$ES_\alpha(X) =$$

$$-\frac{1}{\alpha}[E(X * 1_{\{X \leq -VaR_\alpha(X)\}}) - VaR_\alpha(X)(\alpha - P(X \leq -VaR_\alpha(X)))] \qquad (2)$$

$$= \frac{1}{\alpha}\int_0^\alpha VaR_p(X)dp \qquad (3)$$

Python Implementation

```
def compute_var(x, alpha):
    return -np.percentile(x, alpha*100.0)
def compute_es(x, alpha):
    x_alpha = -compute_var(x, alpha)
    prct = sum(x<=x_alpha)/float(len(x))
    y = x * (x <= var_alpha)
    es = -1.0/alpha*(y.mean()+
        x_alpha*(alpha-prct))
    return es
```

Outline

Parametric VaR, ES

1. From historical prices we need to calculate the returns:

```
rets = np.log(S) - np.log(S.shift(1))
rets = rets[1:]
```

2. Assuming Normal distribution ($r \sim N(\mu, \sigma^2)$), we need to estimate the parameters:

```
mu = rets.mean()
sigma = rets.std()
```

3. Use the closed form solution to calculate the VaR and ES:

```
var = -scs.norm.ppf(alpha, loc=mu,
                    scale=sigma)
es = scs.norm.pdf(scs.norm.ppf(alpha))*
    1/alpha*sigma-mu
```

Normality assumption

Many influential theories in finance assume the normality of stock returns:

1. *Portfolio theory and CAPM:* Normality of returns implies that only the mean and variance are relevant for portfolio optimization.
2. *Option pricing theory:* The Black and Scholes model, and many other pricing models assume (log-)returns follow a Brownian motion, i.e., are normally distributed over any given time interval.

Is normality a realistic assumption? How to test for it?

Benchmark: Simulated data

Let us simulate 10,000 stock price paths from a Geometric Brownian motion, and save the log returns.

```
def gen_paths(S0,r,sigma, T,M,I):
 dt=float(T)/M
 paths=np.zeros((M+1,I))
 paths[0]=S0
 for t in range(1,M+1):
  rand=np.random.randn(I)
  paths[t]=paths[t-1]*
   np.exp((r-0.5*sigma**2)*dt
   +sigma*np.sqrt(dt)*rand)
 return paths
paths=gen_paths(100,0.05,0.2,1.0,50,10000)
log_returns=np.log(paths[1:]/paths[0:-1])
```

Informal tests of normality

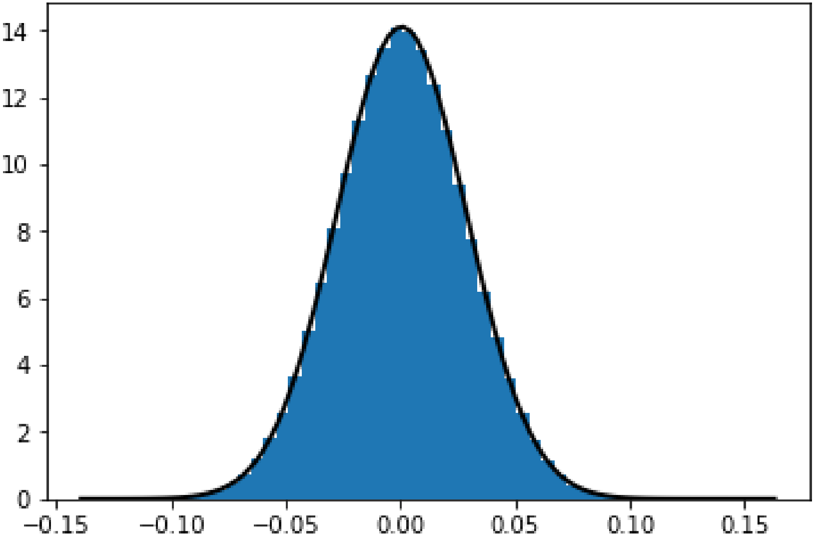
Graphically, one can assess (non-)normality of data via two types of plots:

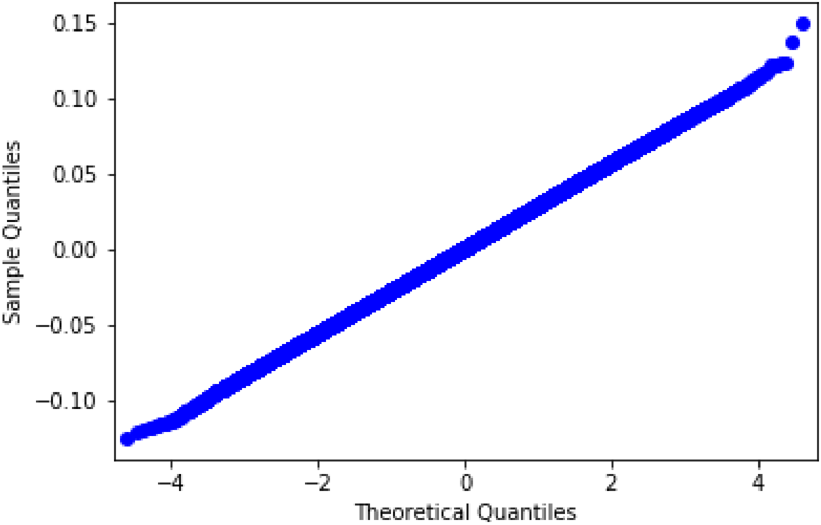
1. Histograms.

```
plt.hist(log_returns.flatten(),bins=100,
  normed=True)
```

2. QQ (quantile-quantile) plots.

```
sm.qqplot(log_returns.flatten())
```


14
12
10
8
6
4
2
0
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15


0.15
0.10
0.05
0.00
−0.05
−0.10
Sample Quantiles
−4
−2
0
2
4
Theoretical Quantiles

Formal tests

- We need to import the scipy.stats sublibrary.

```python
import scipy.stats as scs
```

- Skewness and skewness test (returns test value and p-value).

```python
scs.skew(log_returns.flatten())
scs.skewtest(log_returns.flatten())
```

- Kurtosis (normalized to 0) and kurtosis test (returns test value and p-value).

```python
scs.kurtosis(log_returns.flatten())
scs.kurtosistest(log_returns.flatten())
```

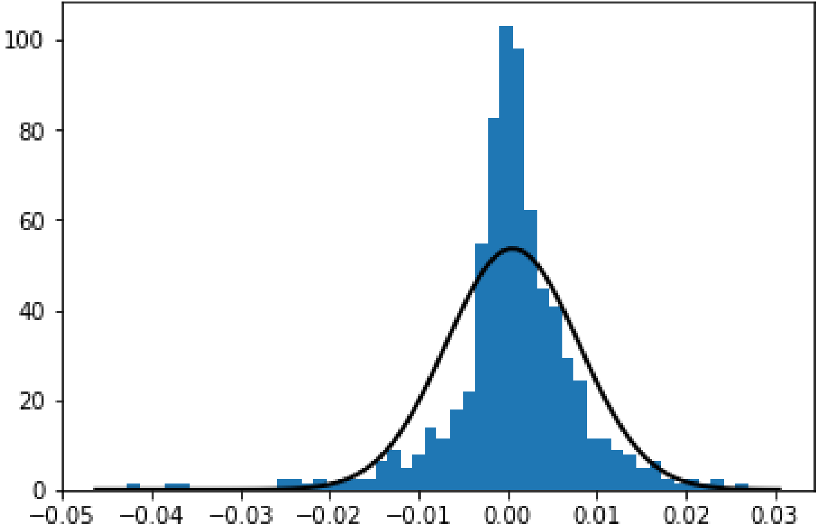
- Catch-all normality test (returns test value and p-value), based on D'Agostino and Pearson (1973).

```python
scs.normaltest(log_returns.flatten())
```

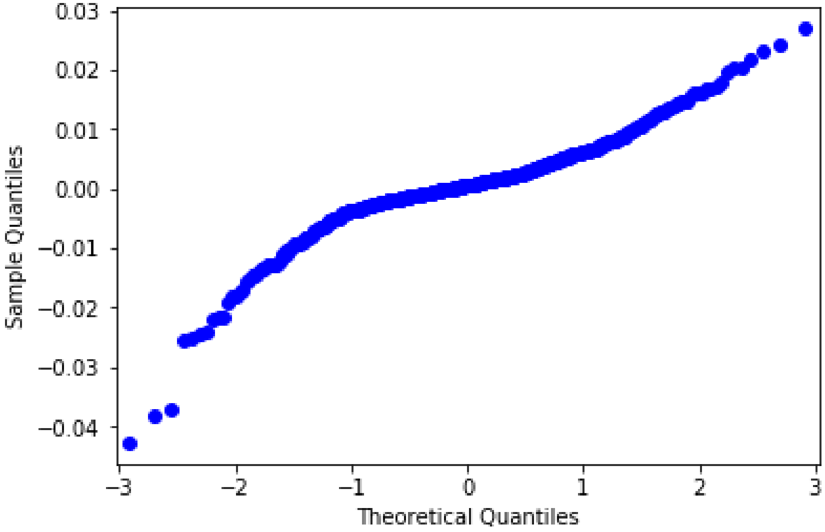
Real-world data: the S&P 500 index

We want to see whether the S&P 500 returns are also normally distributed.

```
plt.hist(sp_returns, bins=50, density=True)
sm.qqplot(sp_returns)
```



100
80
60
40
20
0
−0.05
−0.04
−0.03
−0.02
−0.01
0.00
0.01
0.02
0.03


0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04
Sample Quantiles
−3
−2
−1
0
1
2
3
Theoretical Quantiles

Normality tests on S&P 500 data

The S&P 500 data have negative skewness (i.e., the returns are not symmetric around the mean), and they also have excess kurtosis (fat tails: excessive extreme returns). Therefore, it fails the normality test.

Measure	Value	Test statistic	p-value
Skewness	-0.943	-7.943	1.96e-15
Kurtosis	5.540	8.673	4.19e-18
Normality		138.3	9.18e-31

Outline

Monte Carlo Simulation

1. Similarly to Parametric calculation the Monte Carlo simulations are also driven by an assumed distribution. In case of Normality assumption we can estimate the parameters similarly to the Parametric case.
2. Generating random numbers based on the given distribution
3. Generating return paths
4. Calculate Risk Measures

Built in random number Generators

```
import numpy.random as npr
# Normal distribution
r = npr.normal(loc=mu, scale=sigma, size=(T, N))
R = npr.multivariate_normal(mean, cov, size)
# Log-Normal distribution:
r = npr.lognormal([mean, sigma, size])
# Student-t distribution
r = npr.standard_t(dof, N)
```

Full list: `https://docs.scipy.org/doc/numpy-1.14.0/reference/routines.random.html`

Path generation with pre-generated returns

```
def make_path(R, S0=1, b_return_path=True):
    # R.shape = [T, N]
    R_with_0 = np.zeros((R.shape[0]+1,R.shape[1]))
    R_with_0[1:,:] = R
    cumulative_returns = R_with_0.cumsum(axis=0)
    if b_return_path:
        return np.exp(cumulative_returns)*S0
    else:
        return cumulative_returns
```

Standardizing variables – mean

Let us generate 100 standard normal variables:

```
X=npr.randn(100)
```

Especially in small samples, the sample mean and variance will not be zero and one!

```
X.mean()=-0.128
X.std()=1.015
```

Mean adjustment

First, we can normalize the mean by subtracting it from each element:

```
X1=X-X.mean()
X1.mean()=5.16e-17
X1.std()=1.015
```

This solves the mean issue, not the standard deviation!

Standardizing variables – variance

Variance adjustment

Second, we can normalize the variance by dividing each element with the standard deviation:

```
X2=X1/X.std()
X2.mean()=5.16e-17
X2.std()=1.000
```

Or, directly:

```
X3=(X-X.mean())/X.std()
```

Antithetic variables

Exploiting the feature of a distribution where the transformation of the variable leads to a similar distribution, we can

- significantly speed up the simulation procedure
- reduce the error of the Monte Carlo Simulation

In case of normal distribution:
$X \sim N(0, \sigma^2), Y := -X \Rightarrow Y \sim N(0, \sigma^2)$

VaR with Antithetic variate

```
rets = npr.randn(n)
[pct_up, pct_down] =
    np.percentile(rets, [alpha, 100.0-alpha])
var = -(sigma*(pct_up - pct_down)/2+mu)
```

GARCH model

The **G**eneralised **A**uto**r**egressive **C**onditional **H**eteroskedasticity model, or GARCH:

1. Allows the volatility of returns to be time-dependent, following a AR process.
2. Therefore, it is a suitable model to account for the empirical observation of **volatility clustering**
3. Clustering: Periods of high volatility are likely to be followed also by periods of high volatility.
4. An important note is that the volatility is non-stochastic!
5. Estimation of GARCH is done by maximum likelihood.

GARCH specification

Let stock (index) returns be given by:

$$r_t = \mu + \epsilon_t, \text{ where } \epsilon_t \sim \mathcal{N}\left(0, \sigma_t^2\right) \tag{4}$$

The variance dynamics is:

$$\sigma_t^2 = \omega + \alpha\epsilon_{t-1}^2 + \beta\sigma_{t-1}^2 \tag{5}$$

For the model to remain stationary (variance does not explode) we require:

$$\alpha + \beta < 1. \tag{6}$$

There are four parameters to estimate: μ, ω, α, β.

GARCH in Python

The arch package contains the tools for GARCH analysis. It can be downloaded by pip install (available for Python 2 and 3 as well).

```
from arch import arch.model
```

First we need to define the model that we would like to estimate. Default is GARCH(1,1).

```
am = arch_model(returns)
```

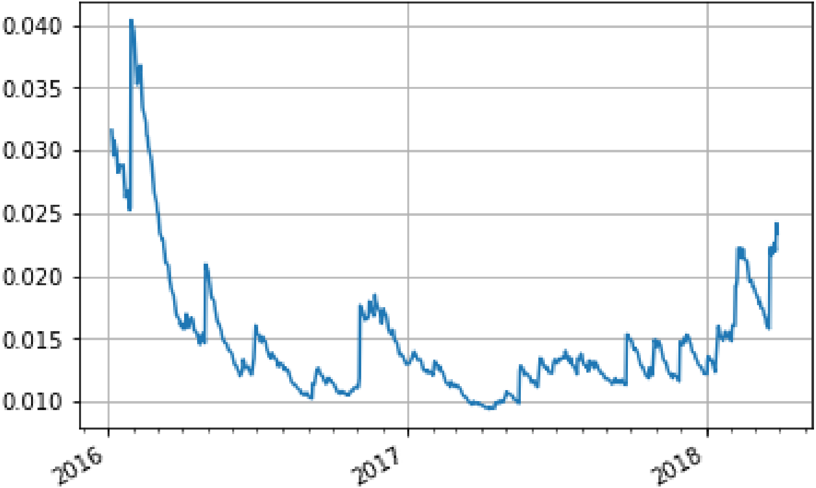
Then we can estimate the model parameters. The default estimation methodology is Maximum Likelihood.

```
res = am.fit()
res.summary()
```

Then using the forecast method we can generate log-returns:

```
fcast = res.forecast(method='simulation',
    simulations=N, horizon=days)
```

Conditional Volatility of Facebook



Effect of "GARCH" on Facebook

Comparing Facebook VaR and ES with and without GARCH(1,1)

	GARCH	Normal
VaR_{95}	0.0387	0.0248
ES_{95}	0.0474	0.0313
VaR_{99}	0.0514	0.0354
ES_{99}	0.0605	0.0407

Outline

Bootstrap Method

1. There is no assumption on the return distribution, so no parameter estimation is required. It will use the historical distribution.
2. Random sampling on the historical returns. NOTE: Keep the Correlation structure!
3. Generate return path
4. Calculate Risk Measures

Random sampling

The fastest way to generate random sample of historical returns, we can assign an integer for each observation time, and using a random sampling algorithm we can generate a the set of simulated returns.

```
df2 = df.reset_index()
nr_of_rets = len(df2)
return_indexes =
    npr.choice(range(nr_of_rets), N)
retruns = df2.loc[return_indexes]
```

Backtesting VaR

The Basel Committee introduced a VaR Backtest Methodology in the Basel regulation, which is called the BIS Traffic Light System. Their approach is based on how many breach would happen in the 1 day 99% VaR value today, if the last 250 days return scenario would repeat.

BIS Traffic Light System

Zone	Breach Value	Cummulative Probability
Green	0	8.11%
Green	1	28.58%
Green	2	54.32%
Green	3	75.81%
Green	4	89.22%
Yellow	5	95.88%
Yellow	6	98.63%
Yellow	7	99.60%
Yellow	8	99.89%
Yellow	9	99.97%
Red	10+	99.99%

Thank you for you attention!