# GARCH-Introduction

April 1, 2019

## 1 Brough Lecture Notes: GARCH Models - Introduction

Finance 5330: Financial Econometrics Tyler J. Brough Last Updated: March 28, 2019

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        plt.rcParams['figure.figsize'] = [20, 10]

        import seaborn
        from numpy import size, log, exp, pi, sum, diff, array, zeros, diag, mat, asarray, sqr
        from numpy.linalg import inv
```

### 1.1 Volatility Models

**Q:** Why do we care about volatility?

**1.** Many derivative security pricing models depend explicitly upon volatility.
*Example:* The Black-Scholes-Merton option pricing model for a European call option:

$$c = e^{-qT} S N(d_1) - e^{-rT} N(d_2)$$

where:

- $c =$ current call price
- $S =$ current spot price of the underlying asset
- $q =$ dividend payout rate
- $T =$ time to maturity of the contract
- $K =$ strike price of the contract
- $N(\cdot)$ is the standard normal cumulative distribution function (CDF)

and

$$d_1 = \frac{\ln(S/K) + (r - q + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}$$
$$d_2 = d_1 - \sigma\sqrt{T}$$

In order to correctly price the option we often must first estimate $\sigma$!
**NB:** The BSM model assumes that $\sigma$ is known and constant.
***Remark:*** BSM implied volatility is (empirically) time varying

$$\sigma_t^{implied}: \quad c_t^{obs} - c_t^{BSM}(\sigma_t^{implied}, \ldots) = 0$$

If the BSM assumptions were correct then $\sigma_t^{implied} = \bar{\sigma}$(a constant).

**NB:** $\sigma_t^{implied}$ is an observable time series of volatility estimates based on a model for option prices.

***Remark:*** Solving the BSM model, given observed call prices, for the implied volatility requires numerical optimization techniques. The so-called Newton-Raphson method is one of the most widely used and efficient algorithms.

**2.** Many applications in risk management and hedging require contronting the time-varying nature of volatility in financial time series data.

**3.** Portfolio allocation in a Markowitz mean-variance framework depends explicitly on volatility (also covariance/correlation).

**4.** Modeling the volatility of a time series can improve the efficiency in parameter estimation (e.g. feasible GLS)

### 1.2 Empirical Regularities of Asset Prices

1. Thick tails: excess kurtosis decreases with aggregation
2. Volatility clustering
   - Large changes followed by large changes; small changes followed by small changes
   - FV_RR: macro aggregates are driven by shocks with heteroscedasticity
3. Leverage effects
   - Changes in prices often negatively correlated with changes in volatility
4. Non-trading Periods
   - Volatility is smaller over periods when markets are closed than when open
5. Forecastable events
   - Forecastable releases of information are associated with high ex ante volatility
6. Volatility and serial correlation
   - inverse relationship between volatility and serial correlation of stock indices

In [85]:
```
inFile = "./data/ibm-sp500.csv"
df = pd.read_csv(inFile, parse_dates=True, index_col=0)
df.head()
```

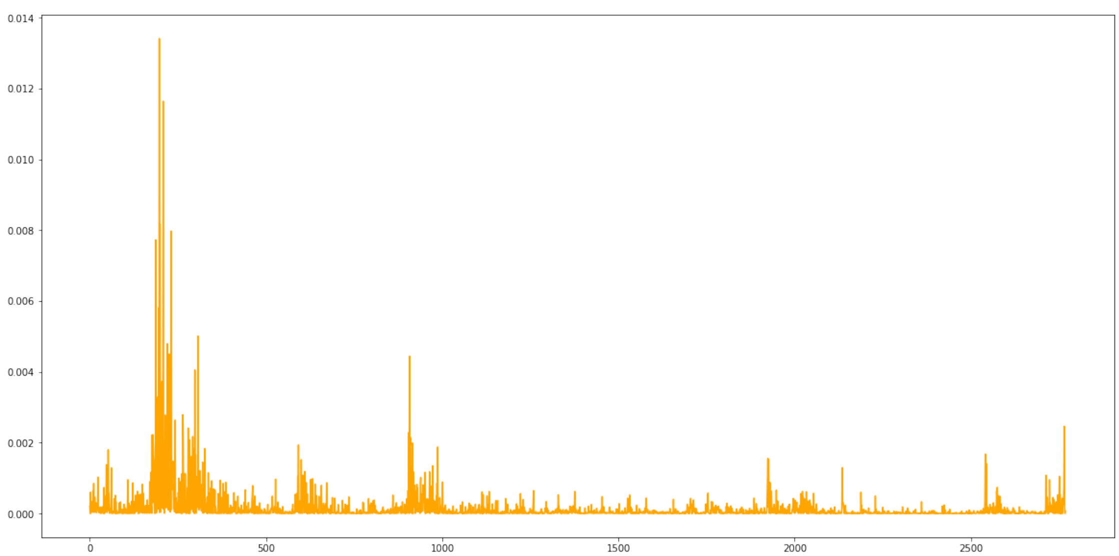
```
Out[85]:                  PRC    sprtrn
         date
         2008-01-02  104.69 -0.014438
         2008-01-03  104.90  0.000000
         2008-01-04  101.13 -0.024552
         2008-01-07  100.05  0.003223
         2008-01-08   97.59 -0.018352
```

```
In [86]: df.tail()
```

```
Out[86]:                  PRC    sprtrn
         date
         2018-12-24  107.57 -0.027112
         2018-12-26  111.39  0.049594
         2018-12-27  113.78  0.008563
         2018-12-28  113.03 -0.001242
         2018-12-31  113.67  0.008492
```

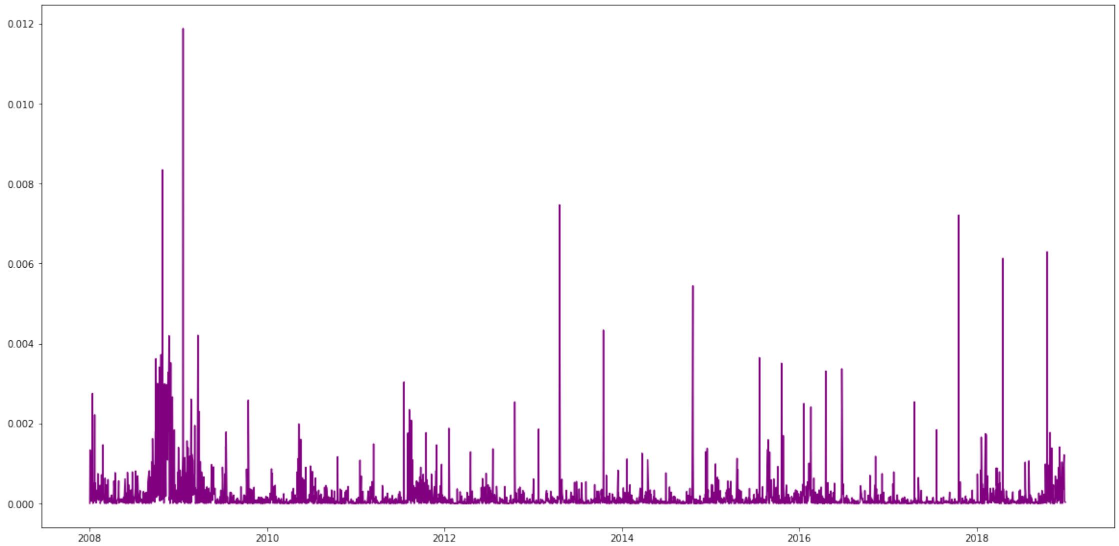
```
In [87]: r2SP = df.sprtrn.values**2
         plt.plot(r2SP, color="orange")
```

```
Out[87]: [<matplotlib.lines.Line2D at 0x7fbc91afd1d0>]
```

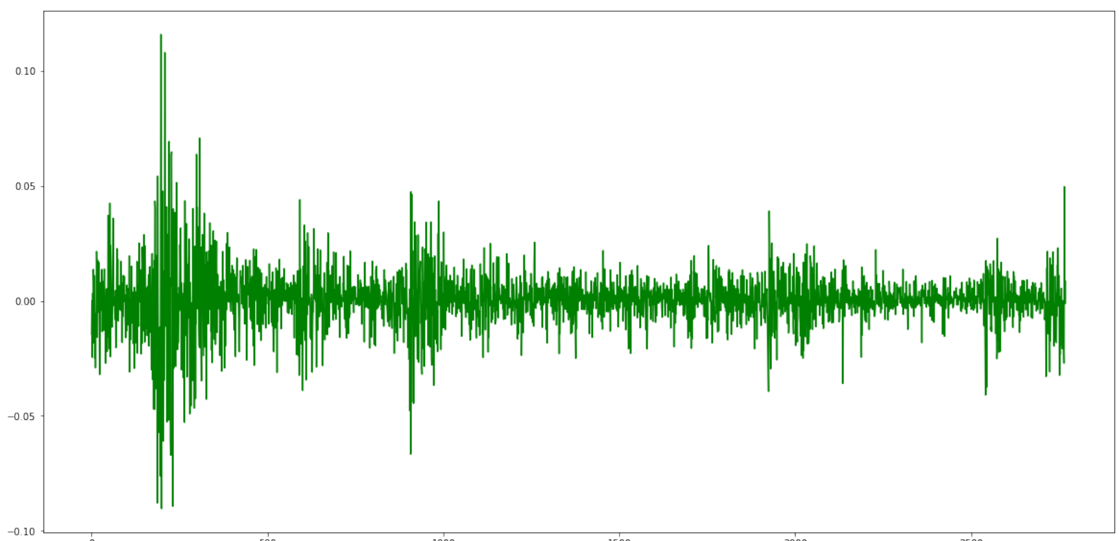



```
In [88]: retIBM = df.PRC.apply(np.abs).apply(np.log).diff()
         r2IBM = retIBM ** 2
         plt.plot(r2IBM, color="purple")
```

```
Out[88]: [<matplotlib.lines.Line2D at 0x7fbc91b5d6a0>]
```




```
In [89]: plt.plot(df.sprtrn.values, color="green")

Out[89]: [<matplotlib.lines.Line2D at 0x7fbc91a11c18>]
```




## 1.3 Engle's ARCH(p) Model

**ARCH:** Autoregressive Conditional Heteroscedasticity

The simplest form is the ARCH(1) model:

$$
\begin{aligned}
y_t &= x_t'\beta + \epsilon_t \\
\epsilon_t &= u_t\sqrt{\alpha_0 + \alpha_1\epsilon_{t-1}^2}
\end{aligned}
$$

with $u_t \sim N(0,1)$
It follows that $E(\epsilon_t|x_t, \epsilon_{t-1}) = 0$, so that $E(\epsilon_t|x_t) = 0$ and $E(y_t|x_t) = x_t'\beta$.
This is a CLRM!
**BUT**,

$$
Var(\epsilon_t|\epsilon_{t-1}) = E(\epsilon_t^2|\epsilon_{t-1}) = E[u_t^2][\alpha_0 + \alpha_1\epsilon_{t-1}^2]
$$

So $\epsilon_t$ is *conditionally heteroscedastic*, not wrt to $x_t$ as before but with respect to $\epsilon_{t-1}$.
The unconditional variance:

$$
\begin{aligned}
Var(\epsilon_t) &= Var[E(\epsilon_t|\epsilon_{t-1})] + E[(Var(\epsilon_t|\epsilon_{t-1})] \\
&= \alpha_0 + \alpha_1 E[\epsilon_{t-1}^2] \\
&= \alpha_0 + \alpha_1 Var(\epsilon_{t-1})
\end{aligned}
$$

If the process generating the disturbances is weakly stationary, then the unconditional variance is not changing over time so

$$
\begin{aligned}
Var[\epsilon_t] &= Var[\epsilon_{t-1}] = \alpha_0 + \alpha_1 Var[\epsilon_{t-1}] \\
&= \frac{\alpha_0}{1-\alpha_1}
\end{aligned}
$$

*Derivation:*

$$
\begin{aligned}
Var[\epsilon_{t-1}] &= \alpha_0 + \alpha_1 Var[\epsilon_{t-1}] \\
Var[\epsilon_{t-1}](1-\alpha_1) &= \alpha_0 \\
Var[\epsilon_{t-1}] &= \frac{\alpha_0}{1-\alpha_1}
\end{aligned}
$$

For this ratio to be finite and positive, $|\alpha_1| < 1$.
Then, unconditionally $\epsilon_t$ is distributed with zero mean and variance $\sigma_2 = \frac{\alpha_0}{1-\alpha_1}$
Therefore, the model obeys the classical assumptions, and OLS is the most efficient linear unbiased estimator of $\beta$.

But, there is a more efficient *nonlinear* estimator. The log-likelihood function for the model is given in Engle (1982). Conditional on starting values $y_0$ and $x_0(\epsilon_0)$, the conditional likelihood for observations is:

$$
lnL = -\frac{T}{2}\ln(2\pi) - \frac{1}{2}\sum_{t=1}^{T} ln(\alpha_0 + \alpha_1\epsilon_{t-1}^2) - \frac{1}{2}\sum_{t=1}^{T}\frac{\epsilon_{t-1}^2}{\alpha_0 + \alpha_1\epsilon_{t-1}^2}
$$

with $\epsilon_t = y_t - \beta' x_t$
Maximization of the $lnL$ can be done with conventional methods (See Appendix E in Greene).
The most common approach is the Newton-Raphson method.
The natural extension to the ARCH(1) model is the ARCH(p) model:

$$
\sigma_t^2 = \alpha_0 + \alpha_1\epsilon_{t-1}^2 + \alpha_2\epsilon_{t-2}^2 + \cdots + \alpha_p\epsilon_{t-p}^2
$$

**NB:** This is an MA(q) process!

**Next:** we will look at testing for ARCH effects and introduce the Genearlized ARCH or ***GARCH*** model.

**NB:** before we move on: Engle specified his ARCH model in terms of the linear regression model. A simple model for heteroscedasticity in return volatility could be the following (in terms of an ARCH(1) model for simplicity):

$$r_t = \epsilon_t \sigma_t \quad \text{with} \quad \epsilon \sim N(0,1)$$
$$\sigma_t^2 = \alpha_0 + \alpha_1 \epsilon_{t-1}^2$$

Or simply as

$$r_t = \epsilon_t \sqrt{\alpha_0 + \alpha_1 \epsilon_{t-1}^2}$$

With this setup, I think one can now see how to apply an IID boostrap scheme to the model by drawing from the residuals:

$$\hat{\epsilon}_t = \frac{r_t}{\hat{\sigma}_t}$$

given the estimated model parameters ($\hat{\alpha_0}$, $\hat{\alpha_1}$), and the initial condition $\epsilon_0$.

One could then use this simple model to simulate a predictive density.

### 1.4 Testing for ARCH Effects

Consider testing the hypotheses

$$H_0: \quad \alpha_1 = \alpha_2 = \cdots = 0 \quad \text{(i.e. No ARCH)}$$
$$H_a: \quad \text{At least one} \quad \alpha_i \neq 0 \quad \text{(ARCH)}$$

Engle derived a simple Lagrange Multiplier (LM) test

- Step 1: Compute squared residuals $\epsilon_t$ from the mean equation regression
- Step 2: Estimate the auxiliary regression

$$\hat{\epsilon}_t^2 = a_0 + a_1 \hat{\epsilon}_{t-1}^2 + \ldots + a_p \hat{\epsilon}_{t-p}^2 + e_t$$

- Step 3: Form the LM statistic

$$LM_{ARCH} = T \cdot R_{Aux}^2$$

- where $T$ = sample size from the auxiliary regression.
- Under $H_0$ (No ARCH) $LM_{ARCH} \overset{a}{\sim} \chi^2(p)$

## 1.5 Weaknesses of the ARCH Models:

- (1) The model assumes that positive and negative shocks have the same effects on volatility because it depends on the square of the previous shocks. This contradicts the well-known leverage effects from the stylized facts.
- (2) ARCH can be rather restrictive. For example, $\alpha_1^2$, of an ARCH(1) model must be in the interval $\left[0, \frac{1}{3}\right]$ if the series has a finite fourth moment. This gets more complicated in higher order ARCH models. It limits the ability of ARCH(p) models to allow for excess kurtosis from the stylized facts.
- (3) The ARCH model does not provide any new insight for understanding the source of variations of time series. It merely provides a mechanical way to describe the behavior of the conditional variance. It gives no indication about what causes such behavior to occur (i.e. we want a structural model)
- (4) ARCH models are likely to overpredict the volatility because they respond slowly to large isolated shocks to the series.

## 1.6 The GARCH Model of Bollerslev (1988)

**GARCH** is *Generalized ARCH*

The model is as follows:

$$y_t = x_t\beta + \epsilon_t$$

is the underlying regression conditioned on an information set at time $t$, $\quad \Psi_t, \quad$ the distribution of the disturbance is assumed to be $\epsilon_t|\Psi_t \sim N(0, \sigma_t^2)$.

with the conditional variance

$$\sigma_t^2 = \omega + \beta_1\sigma_{t-1}^2 + \beta_2\sigma_{t-2}^2 + \cdots + \beta_p\sigma_{t-p}^2 + \alpha_1\epsilon_{t-1}^2 + \cdots + \alpha_p\epsilon_{t-p}^2$$

**NB:** the conditional variance is defined by an ARIMA(p,q) process in the innovations $\epsilon_t^2$

The resulting model is the GARCH(p,q) model

**Remark:** It has been shown that a GARCH(p,q) with small values of $p, q$ performs better than a longer ARCH model

## 1.7 Forecasting from GARCH Models

Consider the basic GARCH(1,1) model

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

from $t = 1, \ldots, T$. The best linear predictor of $\sigma_{T+1}^2$ using information at time $T$ is

$$\begin{aligned} E(\sigma_{T+1}^2|\Psi_T) &= \omega + \alpha E(\epsilon_T^2|\Psi_T) + \beta E(\sigma_T^2|\Psi_T) \\ &= \omega + \alpha\epsilon_T^2 + \beta\sigma_T^2 \end{aligned}$$

Note that $E(\epsilon_{T+1}^2|\Psi_T) = E(\sigma_{T+1}^2|\Psi_T)$ thus

$$\begin{aligned} E(\sigma_{T+2}^2|\Psi_T) &= \omega + \alpha E(\epsilon_{T+1}^2|\Psi_T) + \beta E(\sigma_{T+1}^2|\Psi_T) \\ &= \omega + (\alpha + \beta)E(\sigma_{T+1}^2|\Psi_T) \end{aligned}$$

**Note:** as $k \to \infty$ $E(\sigma^2_{T+k}|\Psi_T) \to E(\sigma^2_T) = \frac{\omega}{1-\alpha-\beta}$

## 1.8 Simulating GARCH Models

We can use GARCH models for Monte Carlo simulation (taking parameters of the model as given). These notes are in part based on the book Practical Financial Econometrics by Carol Alexander.

Let's start by simulating the return process and conditional volatilites. The GARCH returns/volatilities simulation algorithm is as follows:

1. Fix an initial value for $\hat{\sigma}_1$ and set $t = 1$
2. Take a random draw $z_t$ from a standard normal iid process.
3. Form $\epsilon_t = \hat{\sigma}_t z_t$ given the values of $z_t$ and $\hat{\sigma}_t$
4. Find $\hat{\sigma}_{t+1}$ from $\hat{\sigma}_t$ and $\epsilon_t$ using the estimated GARCH model (i.e. given $\hat{\theta} = \{\hat{\omega}, \hat{\alpha}, \hat{\beta}\}$)

The time series $\{\epsilon_t, \epsilon_2, \ldots, \epsilon_T\}$ is a simulated time series with mean zero that exhibits volatility clustering.

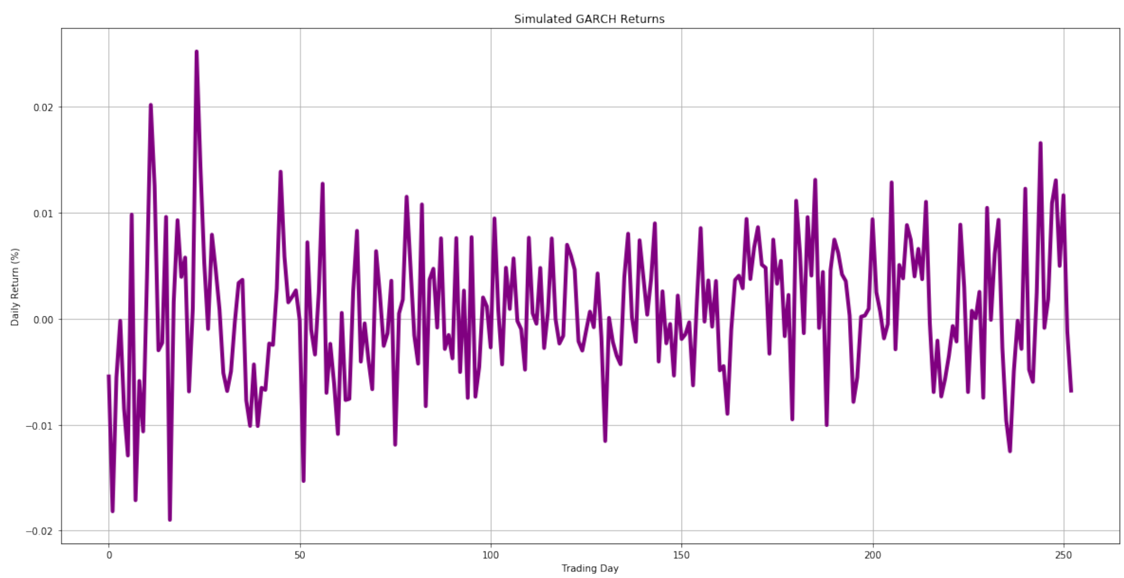
We can implement this in Python as follows:

```
In [90]: ## Set GARCH parameters (these might come from an estimated model)
         w = 10.0**-6
         a = 0.085
         b = 0.905

In [91]: sqrt(w / (1 - a - b) * 252)

Out[91]: 0.15874507866387536

In [92]: def simulate_garch(parameters, numObs):

             ## extract the parameter values
             mu = parameters[0]
             w = parameters[1]
             a = parameters[2]
             b = parameters[3]

             ## initialize arrays for storage
             z = np.random.normal(size=(numObs + 1))
             q = zeros((numObs + 1))
             r = zeros((numObs + 1))

             ## fix initial values
             q[0] = w / (1.0 - a - b)
             r[0] = mu + z[0] * sqrt(q[0])
             e = (r[0] - mu)

             ## run the main simulation loop
             for t in range(1, numObs + 1):
```

```
            q[t] = w + a * (e * e) + b * q[t-1]
            r[t] = mu + z[t] * sqrt(q[t])
            e = (r[t] - mu)

        ## return a tuple with both returns and conditional volatilities
        return (r, q)
```

In [93]:
```
## number of trading days per year
numObs = 252

## daily continuously compounded rate of return correpsonding to 15% annual
mu = log(1.15) / 252

## drift and GARCH(1,1) parameters in an array
params = array([mu, w, a, b])

## run the simulation
r, s = simulate_garch(params, numObs)
```

In [112]:
```
## plot the simulated returns path
fig, ax = plt.subplots()
#ax.plot_date(r, linestyle='--')
ax.grid(True)
plt.title("Simulated GARCH Returns")
plt.ylabel("Daily Return (%)")
plt.xlabel("Trading Day")
plt.plot(r, linewidth=4, color="purple")
plt.show()
```



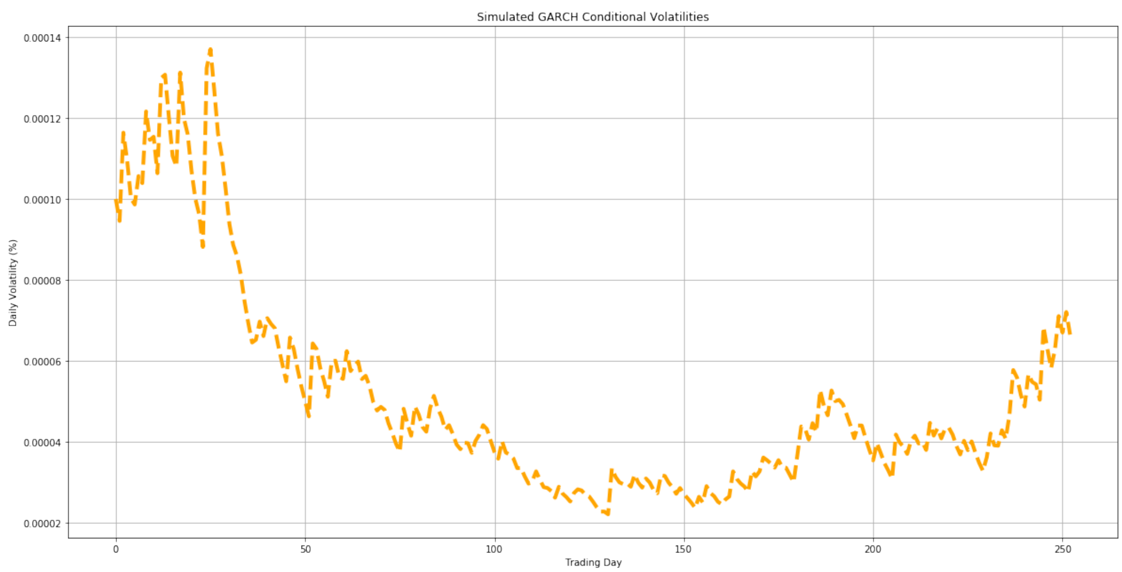
In [95]: 
```
## let's look at the first 10 observations
r[:10]
```

Out[95]: 
```
array([-0.00545627, -0.0181817 , -0.00554792, -0.00020472, -0.00850344,
       -0.0129158 ,  0.00981288, -0.01713787, -0.00588187, -0.01064817])
```

In [96]: 
```
## let's look at the last 10 observations
r[-10:]
```

Out[96]: 
```
array([ 0.00248279,  0.01656368, -0.00087559,  0.00183454,  0.01089587,
        0.01304992,  0.00497261,  0.01163899, -0.00121984, -0.00679405])
```

In [111]: 
```
## plot the simulated conditional volatilities
fig, ax = plt.subplots()
#ax.plot_date(r, linestyle='--')
ax.grid(True)
plt.title("Simulated GARCH Conditional Volatilities")
plt.ylabel("Daily Volatility (%)")
plt.xlabel("Trading Day")
plt.plot(s, linestyle='--', linewidth=4, color="orange")
plt.show()
```




Given a time series of simulated returns, we can now use these to simulate asset prices if we wish.

We do this as follows for the log-price as follows. We express the simulated returns as $\hat{r}_t = \mu + z_t \cdot \hat{\sigma}_t$

$$\ln(S_t) = \ln(S_{t-1}) + \mu + z_{t-1} \cdot \hat{\sigma}_{t-1}$$

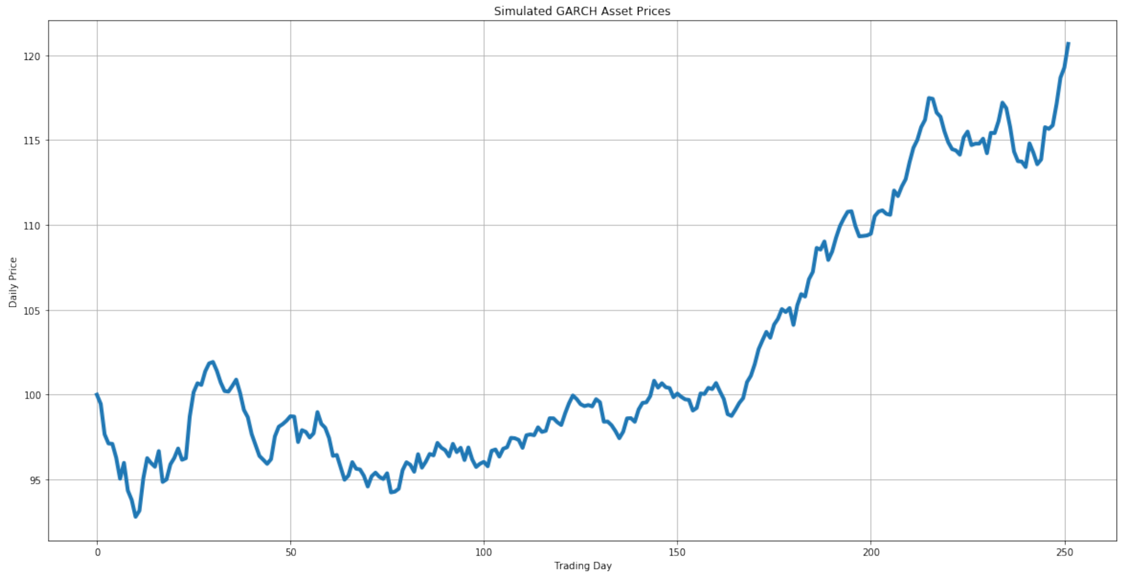
Or for dollar prices:

$$S_t = S_{t-1} \exp\left(\hat{r}_{t-1}\right)$$

We do this in Python with the following:

```
In [2]: ## initialize the spot price array
        spot = zeros(numObs)
        spot[0] = 100.0


        ## run the main simulation loop
        for t in range(1, numObs):
            spot[t] = spot[t-1] * exp(r[t-1])


        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-2-c89676d93107> in <module>
          1 ## initialize the spot price array
    ----> 2 spot = zeros(numObs)
          3 spot[0] = 100.0
          4
          5


        NameError: name 'numObs' is not defined


In [99]: ## let's look at the first 10 prices
         spot[:10]

Out[99]: array([100.        ,  99.45585841,  97.66392147,  97.12358996,
                 97.10370902,  96.28149454,  95.04593849,  95.98320394,
                 94.35227151,  93.79893235])

In [100]: ## let's look at the last 10 prices
          spot[-10:]

Out[100]: array([114.25256477, 113.57284581, 113.85517337, 115.75673952,
                 115.65542832, 115.86779723, 117.13718045, 118.67582928,
                 119.2674279 , 120.66369051])

In [108]: ## plot the simulated price path
          fig, ax = plt.subplots()
          #ax.plot_date(r, linestyle='--')
          ax.grid(True)
          plt.title("Simulated GARCH Asset Prices")
```

```
plt.ylabel("Daily Price")
plt.xlabel("Trading Day")
plt.plot(spot, linestyle='-', linewidth=4)
plt.show()
```




```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```