Statistical Estimation

Consider the constant return model (CER):

$$r_t \overset{i.i.d.}{\sim} N(\mu, \sigma^2), t = 1, \ldots, T.$$

We have

Method of moments estimators

$$\begin{aligned} E[r_t] &= \mu \\ Var(r_t) = E[(r_t - \mu)^2] &= \sigma^2 \end{aligned}$$

A natural estimator of μ is the sample mean

$$\hat{\mu} = \frac{1}{T}\sum_{t=1}^{T} r_t.$$

An estimator of σ^2 is

$$\hat{\sigma}^2 = \frac{1}{T-1}\sum_{t=1}^{T}(r_t - \hat{\mu})^2.$$

Maximum likelihood estimators

$\hat{\mu}_{mle}$ and $\hat{\sigma}_{mle}$ are solutions to the log-likelihood estimation problem:

$$\max_{\mu,\sigma} \sum_{t=1}^{T} \log f(r_t; \mu, \sigma),$$

where $f(r_t; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(r_t-\mu)^2}{2\sigma^2}\right)$. We have

$$\begin{aligned} \hat{\mu}_{mle} &= \frac{1}{T}\sum_{t=1}^{T} r_t \\ \hat{\sigma}^2_{mle} &= \frac{1}{T}\sum_{t=1}^{T}(r_t - \hat{\mu})^2. \end{aligned}$$

In [1]:

```
import numpy as np
# pip install pandas_datareader
import pandas_datareader as web

# import data
df = web.get_data_yahoo("MSFT", start = "2002-01-01", end = "2020-12-31", interv
```

```
df.reset_index(inplace = True) # convert index into a column

df['cc'] = np.log(df['Adj Close']/df['Adj Close'].shift(1))
df
```

Out[1]:

	Date	High	Low	Open	Close	Volume	Adj Close	
0	2002-01-01	35.310001	30.665001	33.325001	31.855000	1.360116e+09	20.110750	N
1	2002-02-01	32.250000	28.575001	32.075001	29.170000	1.131159e+09	18.415644	-0.088
2	2002-03-01	32.500000	29.155001	29.525000	30.155001	1.073244e+09	19.037500	0.033
3	2002-04-01	30.200001	25.719999	29.915001	26.129999	1.417479e+09	16.496431	-0.143
4	2002-05-01	28.219999	24.174999	26.080000	25.455000	1.420266e+09	16.070288	-0.026
...	...	...	...	...	...	...	...	
224	2020-09-01	232.860001	196.250000	225.509995	210.330002	7.681763e+08	208.036957	-0.067
225	2020-10-01	225.210007	199.619995	213.490005	202.470001	6.316180e+08	200.262650	-0.038
226	2020-11-01	228.119995	200.119995	204.289993	214.070007	5.734430e+08	211.736176	0.055
227	2020-12-01	227.179993	209.110001	214.509995	222.419998	5.947617e+08	220.571106	0.040
228	2021-01-01	242.639999	211.940002	222.529999	231.960007	6.480764e+08	230.031799	0.041

229 rows × 8 columns

In [2]:

```
cc = df['cc'].iloc[1:]

print('The MM estimate of mu is ', np.mean(cc))
print('The MM estimate of sigma is ', np.std(cc, ddof = 1))
```

```
The MM estimate of mu is  0.010688434427337648
The MM estimate of sigma is  0.06626640985344462
```

In [3]:

```
from scipy.stats import norm
import scipy.optimize as optimize

# method 1
print(norm.fit(cc)) # the maximum likelihood estimates of mu and sigma
print(np.std(cc, ddof = 0)) # divided by T instead of (T-1)
print('=====================================================')
# method 2
def log_likelihood(params, data):
    mu, sigma = params
    # If the standard deviation prameter is negative, return a large value:
    if sigma < 0:
        return(1e8)
```

```
    likelihood = norm.pdf(data, loc = mu, scale = sigma)
    return -np.sum(np.log(likelihood[likelihood > 0]))

res = optimize.minimize(fun = log_likelihood, x0 = [0.1, 0.5],  # initial guess
                            args = cc)
print(res)
print('The maximum likelihood estimates are: ', res.x)
```

```
(0.010688434427337648, 0.0661209290855236)
0.06612092908552358
====================================================
      fun: -295.79156408333745
 hess_inv: array([[1.91465206e-05, 2.60456943e-08],
       [2.60456943e-08, 9.55087404e-06]])
      jac: array([-7.62939453e-06,  0.00000000e+00])
  message: 'Optimization terminated successfully.'
     nfev: 60
      nit: 8
     njev: 20
   status: 0
  success: True
        x: array([0.01068843, 0.06612092])
The maximum likelihood estimates are:  [0.01068843 0.06612092]
```

In what follows, we focus on MM estimators.

Properties of estimators

Unbiasedness

The bias of an estimator $\hat{\theta}$ is $bias(\hat{\theta}, \theta) = E[\hat{\theta}] - \theta$. $\hat{\theta}$ is unbiased if the bias is 0.

Note that

$$E[\hat{\mu}] = E\left[\frac{1}{T}\sum_{t=1}^{T} r_t\right] = \frac{1}{T}\sum_{t=1}^{T} E\left[r_t\right] = \frac{1}{T}\sum_{t=1}^{T}\mu = \mu.$$

Besides,

$$\begin{aligned}
E[(r_t - \hat{\mu})^2] &= E\{[(r_t - \mu) - (\hat{\mu} - \mu)]^2\} \\
&= E[(r_t - \mu)^2] - 2E\left[(r_t - \mu)\frac{\sum_{s=1}^{T}(r_s - \mu)}{T}\right] + E[(\hat{\mu} - \mu)^2] \\
&= Var(r_t) - \frac{2}{T}Var(r_t) + Var(\hat{\mu}) \\
&= \sigma^2 - \frac{2}{T}\sigma^2 + \frac{1}{T}\sigma^2 \\
&= \frac{T-1}{T}\sigma^2.
\end{aligned}$$

and

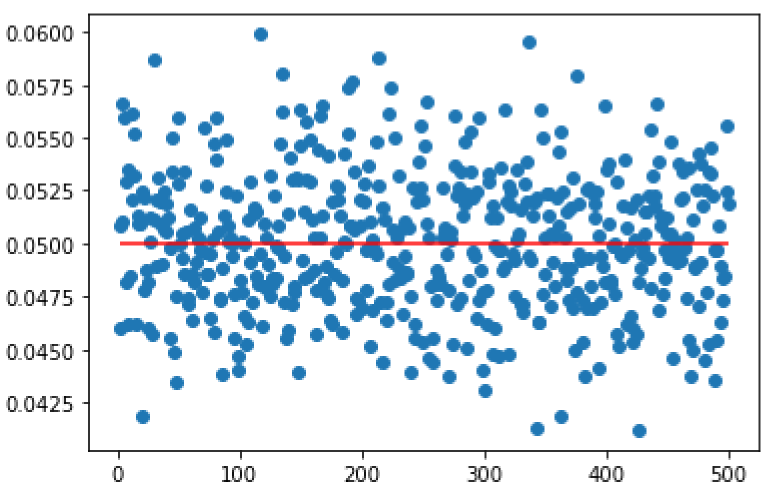
$$E[\hat{\sigma}^2] = \frac{1}{T-1}\sum_{t=1}^{T} E[(r_t - \hat{\mu})^2] = \frac{T}{T-1} \cdot \frac{T-1}{T}\sigma^2 = \sigma^2.$$

In [4]:

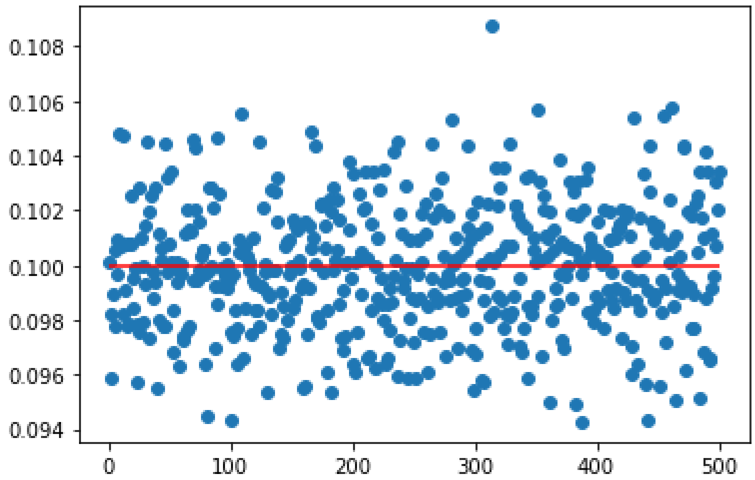
```
# An illustration
from scipy.stats import norm
import matplotlib.pyplot as plt

def gen_sample(n, mu = 0.05, sigma = 0.1):
    sample = norm.rvs(loc = mu, scale = sigma, size = n)
    return np.mean(sample), np.std(sample, ddof = 1)

# generate 500 samples, each of which has sample size 1000
np.random.seed(123)
nsim = 500
n = 1000
mu_hat = np.zeros(nsim)
sigma_hat = np.zeros(nsim)

for i in range(nsim):
    mu_hat[i], sigma_hat[i] = gen_sample(n)
```

In [5]:

```
plt.scatter(np.linspace(1, nsim, nsim), mu_hat)
plt.hlines(y = 0.05, xmin = 1, xmax = nsim, color = 'red')
plt.show()

print(np.mean(mu_hat))
# the mean of simulated mu_hat can be approximately viewed as expected mu_hat (M
# close to the true mu 0.05, which illustrates the unbiasedness of mu hat

plt.scatter(np.linspace(1, nsim, nsim), sigma_hat)
plt.hlines(y = 0.1, xmin = 1, xmax = nsim, color = 'red')
plt.show()
print(np.mean(sigma_hat))
# the mean of simulated sigma_hat can be approximately viewed as expected sigma_
# close to the true sigma 0.1, which illustrates the unbiasedness of sigma hat
```




```
0.05010098782870862
```




```
0.0999329821685706
```

Consistency

An estimator $\hat{\theta}$ is consistent for θ if for any $\epsilon > 0$,

$$\lim_{T \to \infty} P(|\hat{\theta} - \theta| > \epsilon) = 0$$

In [6]:

```
import pandas as pd

df = pd.DataFrame(columns=['Size', 'mu_hat', 'sigma_hat']) # record simulation r

df['Size'] = np.repeat(np.linspace(500, 5000, 10), nsim) # repeat each element n
df['Size'] = df['Size'].astype(int)
for i in range(df.shape[0]):
    df.iloc[i,1:3] = gen_sample(df['Size'].iloc[i])
df
```

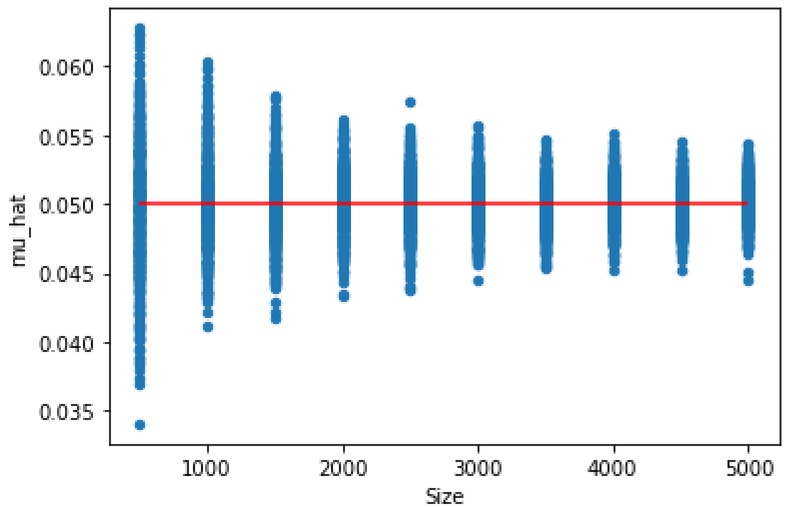
Out[6]:

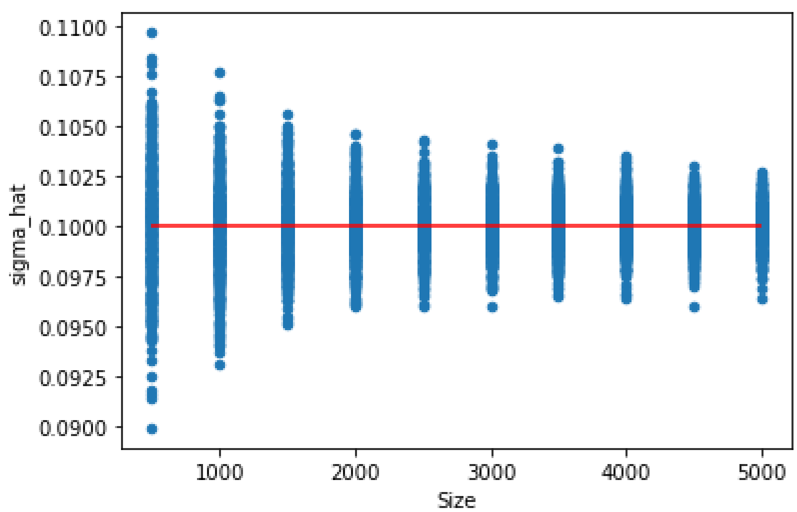
	Size	mu_hat	sigma_hat
0	500	0.047696	0.101427
1	500	0.040671	0.100474
2	500	0.052619	0.105311
3	500	0.047387	0.095438
4	500	0.062366	0.099572
...	...	...	...
4995	5000	0.052206	0.099992
4996	5000	0.049375	0.100458
4997	5000	0.050028	0.099948
4998	5000	0.050189	0.100037
4999	5000	0.049952	0.099341

5000 rows × 3 columns

In [7]:

```
df.plot.scatter(x="Size",y="mu_hat")
plt.hlines(y = 0.05, xmin = 500, xmax = 5000, color = 'red')
plt.show()

df.plot.scatter(x="Size",y="sigma_hat")
plt.hlines(y = 0.1, xmin = 500, xmax = 5000, color = 'red')
plt.show()
```





Asymptotic normality

By the CLT,

$$\frac{\hat{\theta} - \theta}{SE(\hat{\theta})} \to_d N(0,1).$$

Note that

$$SE(\hat{\mu}) = \frac{\hat{\sigma}}{\sqrt{T}}$$
$$SE(\hat{\sigma}^2) \approx \frac{\sqrt{2}\hat{\sigma}^2}{\sqrt{T}}.$$

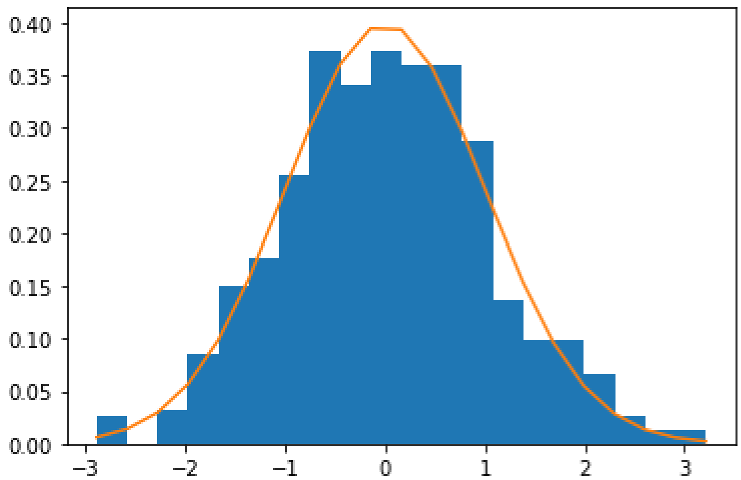
Loosely speaking,

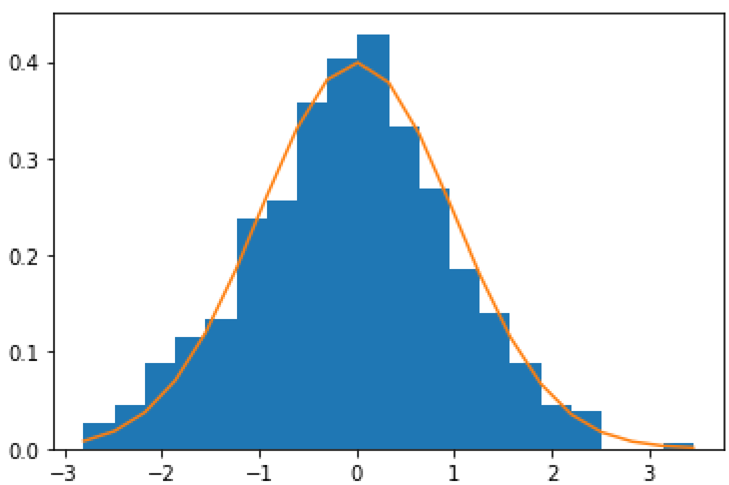
$$\frac{\hat{\mu} - \mu}{\hat{\sigma}/\sqrt{T}} = \frac{1}{\sqrt{T}} \sum_{t=1}^{T} (r_t - \mu)/\hat{\sigma}$$

is approximately $N(0, 1)$ for large T.

In [8]:

```
n, bins, patches = plt.hist(np.sqrt(1000)*(mu_hat - 0.05)/ sigma_hat, bins = 20,
plt.plot(bins, norm.pdf(bins))
plt.show()

n, bins, patches = plt.hist(np.sqrt(1000/2)*(sigma_hat ** 2 - 0.01)/ (sigma_hat
plt.plot(bins, norm.pdf(bins))
plt.show()
```





Mean Squared Error

The MSE of an estimator $\hat{\theta}$ is bias squared plus variance of the estimator.

$$
\begin{aligned}
MSE(\hat{\theta}) &= E[(\hat{\theta}-\theta)^2] \\
&= E[(\hat{\theta}-E[\hat{\theta}]+E[\hat{\theta}]-\theta)^2] \\
&= E[(\hat{\theta}-E[\hat{\theta}])^2]+(E[\hat{\theta}]-\theta)^2+2E[(\hat{\theta}-E[\hat{\theta}])(E[\hat{\theta}]-\theta)] \\
&= Var(\hat{\theta})+bias(\hat{\theta})^2,
\end{aligned}
$$

where $E[(\hat{\theta}-E[\hat{\theta}])=E[\hat{\theta}]-E[\hat{\theta}]=0.$

For two estimators of θ, we prefer the one with samller MSE.

Statistical Inference

Confidence interval

Take $\hat{\sigma}^2=\frac{1}{T-1}\sum_{t=1}^{T}(r_t-\hat{\mu})^2$ as an example. By the central limit theorem,

$$
\frac{\hat{\sigma}^2-\sigma^2}{SE(\hat{\sigma}^2)} \rightarrow_d N(0,1),
$$

where the asymptotic standard error $SE(\hat{\sigma}^2)=\frac{\sqrt{2}\hat{\sigma}^2}{\sqrt{T}}$. Then, the **90\%** asymptotic confidence interval is

$$
[\hat{\sigma}^2-q_{0.95}^{Z}SE(\hat{\sigma}^2),\hat{\sigma}^2+q_{0.95}^{Z}SE(\hat{\sigma}^2)],
$$

where $q_{0.95}^{Z}$ is the 95\%-quantile of the standard normal distribution, i.e., `norm.ppf(0.95)` .

Hypothesis testing

Given a rample sample, consider the two-sided test:

$$
H_0:\sigma^2=\sigma_0^2 \text{ vs } H_1:\sigma^2\neq\sigma_0^2.
$$

We once again rely on the asymptotic distribution

$$
\frac{\hat{\sigma}^2-\sigma^2}{SE(\hat{\sigma}^2)} \rightarrow_d N(0,1),
$$

Under H_0, it becomes

$$
\frac{\hat{\sigma}^2-\sigma_0^2}{SE(\hat{\sigma}^2)} \rightarrow_d N(0,1),
$$

Given the significance level α, if the test-statistic is greater than $q_{1-\frac{\alpha}{2}}^{Z}$ in absolute value, we accept H_1; otherwise, we cannot reject H_0.

Testing for normal distribution

$$H_0 : r_t \overset{i.i.d.}{\sim} N(\mu, \sigma^2) \text{ vs } H_1 : r_t \sim \text{ not normal}$$

Test statistic (Jarque-Bera statistic)

$$JB = \frac{T}{6}\left(\widehat{\text{skew}}^2 + \frac{(\widehat{\text{kurt}} - 3)^2}{4}\right),$$

where

$$\widehat{\text{skew}} = \frac{\frac{1}{T}\sum_{t=1}^{T}(r_t - \hat{\mu})^3}{(\hat{\sigma}^2)^{3/2}}$$
$$\widehat{\text{kurt}} = \frac{\frac{1}{T}\sum_{t=1}^{T}(r_t - \hat{\mu})^4}{(\hat{\sigma}^2)^2}$$

Under H_0,

$$JB \sim \chi^2(2),$$

a chi-square distribution with 2 degrees of freedom.

In [9]:

```
from scipy.stats import chi2, jarque_bera

mu_hat = np.mean(cc) # mu_mle
sigma_hat = np.std(cc, ddof = 0) # sigma_mle
skew_hat = np.mean( (cc - mu_hat) ** 3 ) / sigma_hat ** 3
kurt_hat = np.mean( (cc - mu_hat) ** 4 ) / sigma_hat ** 4

print('The estimates are {:.4f}, {:.4f}, {:.4f}, and {:.4f}'.format(mu_hat, sigm
JB = len(cc)/6 * (skew_hat ** 2 + (kurt_hat - 3) ** 2/4)

print('The test statistic is %.4f' % JB)
print('The p-value is %.4f' % (1-chi2.cdf(JB, df = 2)))
```

```
The estimates are 0.0107, 0.0661, -0.1551, and 3.6091
The test statistic is 4.4386
The p-value is 0.1087
```

In [10]:

```
jarque_bera(cc)
```

Out[10]:

```
Jarque_beraResult(statistic=4.438576660932907, pvalue=0.10868643012911916)
```