Calibration of The Vasicek Model to Historical Data with Python Code

The Vasicek Model

The instantaneous interest rate r_t follows the following stochastic differential equation (SDE):

$$dr_t = \underbrace{a}_{\substack{\text{Speed of reversion} \\ a>0}} \cdot (\underbrace{b}_{\text{Long-term mean}} - r_t) \cdot dt + \underbrace{\sigma}_{\text{Instantaneous volatility}} \cdot dW_t$$

Calibration Methods

We present two methods for calibrating the Vasicek model to historical data:

1) Least Squares
2) Maximum Likelihood Estimation

Least Squares

The Euler-Maruyama discretisation method of the Vasicek model gives:

$$r_{t+\delta t} - r_t = a \cdot (b - r_t) \cdot \delta t + \sigma \cdot \sqrt{\delta t} \cdot \varepsilon \qquad \varepsilon \sim N(0,1)$$

Which can be rewritten as:

$$r_{t+\delta t} = \underbrace{(1 - a \cdot \delta t)}_{\alpha} \cdot r_t + \underbrace{a \cdot b \cdot \delta t}_{\beta} + \underbrace{\sigma \cdot \sqrt{\delta t} \cdot \varepsilon}_{\xi \text{ iid normally distributed}}$$

$$r_{t+\delta t} = \alpha \cdot r_t + \beta + \xi$$

Least Squares

The Euler-Maruyama discretisation method of the Vasicek model gives:

$$r_{t+\delta t} - r_t = a \cdot (b - r_t) \cdot \delta t + \sigma \cdot \sqrt{\delta t} \cdot \varepsilon \qquad \varepsilon \sim N(0,1)$$

Which can be rewritten as:

$$r_{t+\delta t} = \underbrace{(1 - a \cdot \delta t)}_{\alpha} \cdot r_t + \underbrace{a \cdot b \cdot \delta t}_{\beta} + \underbrace{\sigma \cdot \sqrt{\delta t} \cdot \varepsilon}_{\xi}$$

ξ iid normally distributed

$$r_{t+\delta t} = \alpha \cdot r_t + \beta + \xi$$

$\left(\widehat{\alpha}, \widehat{\beta}\right)$ estimated by least squares method (linear regression of $r_{t+\delta t}$ on r_t)

$$\left(\widehat{a}, \hat{b}\right) = \left(\frac{1-\widehat{\alpha}}{\delta t}, \frac{\widehat{\beta}}{1-\widehat{\alpha}}\right) \qquad \widehat{\sigma} = \sqrt{\frac{Var(\xi)}{\delta t}}$$

Maximum Likelihood Estimation

Knowing r_t, the solution of the SDE $r_{t+\delta t}$ is equal to:

$$r_{t+\delta t} = r_t \cdot e^{-a \cdot \delta t} + b \cdot \left(1 - e^{-a \cdot \delta t}\right) + \sigma \cdot e^{-a \cdot \delta t} \cdot \int_t^{t+\delta t} e^{a \cdot s} \cdot dW_s$$

Conditionally to r_t, $r_{t+\delta t}$ follows a normal distribution.

$$r_{t+\delta t} \sim N\left(r_t \cdot e^{-a \cdot \delta t} + \underbrace{b \cdot \left(1 - e^{-a \cdot \delta t}\right)}_{\beta}, \underbrace{\frac{\sigma^2}{2 \cdot a} \cdot \left(1 - e^{-2 \cdot a \cdot \delta t}\right)}_{\Sigma^2}\right)$$

The probability distribution function is:

$$f(r_t, t; r_{t+\delta t}, t + \delta t) = \frac{1}{\sqrt{2 \cdot \pi \cdot \Sigma^2}} \cdot e^{-\frac{1}{2} \cdot \frac{\left(r_{t+\delta t} - r_t \cdot e^{-a \cdot \delta t} - \beta\right)^2}{\Sigma^2}}$$

Maximum Likelihood Estimation

The log-likelihood on partition of time $[t_0, t_1, \ldots, t_i, \ldots, t_n]$ with $t_i - t_{i-1} = \delta t$ is:

$$\ln\left(\prod_{i=0}^{n-1} f(r_{t_i}, t_i; r_{t_{i+1}}, t_{i+1})\right)$$

You can easily check it has the following expression

$$L(a, b, \sigma) = -\frac{n}{2} \cdot \ln\left(\frac{\sigma^2}{2 \cdot a} \cdot \left(1 - e^{-2 \cdot a \cdot \delta t}\right)\right) - \frac{n}{2} \cdot \ln(2 \cdot \pi)$$
$$- \frac{a}{\sigma^2 \cdot \left(1 - e^{-2 \cdot a \cdot \delta t}\right)} \cdot \sum_{i=0}^{n-1} \left(r_{t_{i+1}} - r_{t_i} \cdot e^{-a \cdot \delta t} - b \cdot \left(1 - e^{-a \cdot \delta t}\right)\right)^2$$

Maximum Likelihood Estimation

a, b, σ maximizing the log likelihood have the following expressions:

$$\widehat{a} = \frac{1}{\delta t} \cdot \ln\left(\frac{S_0 - \hat{b}}{S_1 - \hat{b}}\right), \quad \hat{b} = \frac{S_1 \cdot S_{00} - S_0 \cdot S_{01}}{S_0 \cdot S_1 - S_0^2 - S_{01} + S_{00}}, \quad \widehat{\sigma^2} = \frac{1}{n \cdot \beta \cdot \left(1 - \frac{1}{2} \cdot a \cdot \beta\right)} \cdot \sum_{i=1}^{n} \left(r_{t_i} - m_{t_{i-1}}(t_i)\right)^2$$

where:

$$S_0 = \frac{1}{n} \cdot \sum_{i=1}^{n} r_{t_{i-1}}, \quad S_1 = \frac{1}{n} \cdot \sum_{i=1}^{n} r_{t_i}$$

$$S_{00} = \frac{1}{n} \cdot \sum_{i=1}^{n} r_{t_{i-1}} \cdot r_{t_{i-1}}, \quad S_{01} = \frac{1}{n} \cdot \sum_{i=1}^{n} r_{t_{i-1}} \cdot r_{t_i}$$

$$\beta = \frac{1}{a} \cdot \left(1 - e^{-a \cdot \delta t}\right), \quad m_{t_{i-1}}(t_i) = b \cdot a \cdot \beta + r_{t_{i-1}} \cdot (1 - a \cdot \beta))$$

Reference: Fergusson, K., Platen, E. (2015), "Application of maximum likelihood estimation to stochastic short rate models”

Python Code

Import Libraries

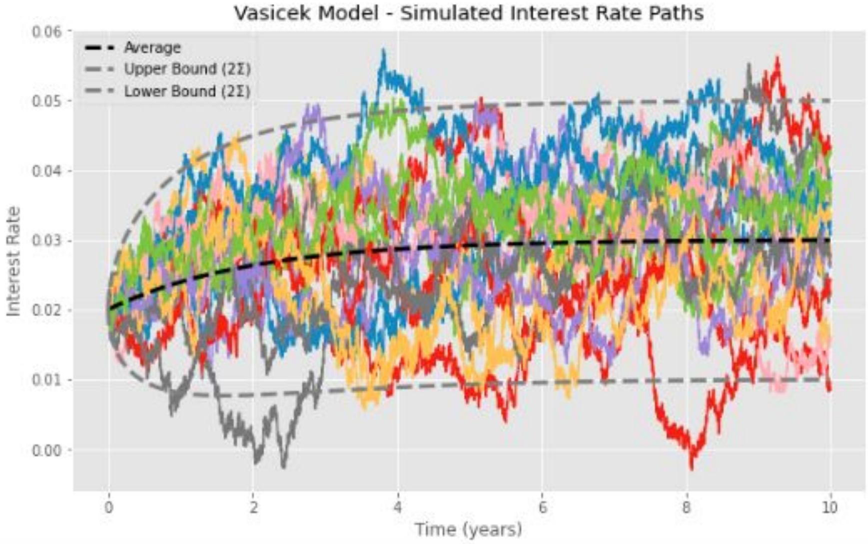
```
import matplotlib.pyplot as plt
plt.style.use('ggplot')
import math
import numpy as np
import pandas as pd
from scipy.stats import norm
%matplotlib inline
from sklearn.linear_model import LinearRegression
```

Simulation Vasicek Process with Euler-Maruyama Discretisation

```
def vasicek(r0, a, b, sigma, T, num_steps, num_paths):
    dt = T / num_steps
    rates = np.zeros((num_steps + 1, num_paths))
    rates[0] = r0

    for t in range(1, num_steps + 1):
        dW = np.random.normal(0, 1, num_paths)
        rates[t] = rates[t - 1] + a * (b - rates[t - 1]) * dt + sigma * np.sqrt(dt) * dW

    return rates
```

Python Code

```python
# Model parameters
r0 = 0.02  # Initial short rate
a = 0.5    # Mean reversion speed
b = 0.03  # Long-term mean
sigma = 0.01    # Volatility
T = 10       # Time horizon
num_steps = 10000  # Number of steps
num_paths = 20   # Number of paths

# Simulate Vasicek model
simulated_rates = vasicek(r0, a, b, sigma, T, num_steps, num_paths)

#average_rates = np.mean(simulated_rates, axis=1)

# Time axis
time_axis = np.linspace(0, T, num_steps + 1)

#average value
average_rates = [r0 * np.exp(-a * t) + b * (1 - np.exp(-a * t)) for t in time_axis]

# standard deviation
std_dev = [(sigma**2 / (2 * a) * (1 - np.exp(-2 * a * t)))**.5 for t in time_axis]

# Calculate upper and lower bounds (±2 sigma)
upper_bound = [average_rates[i] + 2 * std_dev[i] for i in range(len(time_axis))]
lower_bound = [average_rates[i] - 2 * std_dev[i] for i in range(len(time_axis))]

# Plotting multiple paths with time on x-axis
plt.figure(figsize=(10, 6))
plt.title('Vasicek Model - Simulated Interest Rate Paths')
plt.xlabel('Time (years)')
plt.ylabel('Interest Rate')
for i in range(num_paths):
    plt.plot(time_axis, simulated_rates[:, i])

plt.plot(time_axis, average_rates, color='black',linestyle='--', label ='Average', linewidth = 3)
plt.plot(time_axis, upper_bound, color='grey', linestyle='--', label='Upper Bound (2Σ)', linewidth = 3)
plt.plot(time_axis, lower_bound, color='grey', linestyle='--', label='Lower Bound (2Σ)', linewidth = 3)
plt.legend()
plt.show()
```



Python Code

Real parameters, unknown, that we want to estimate:

a = 0.5, b = 0.03, sigma = 0.01

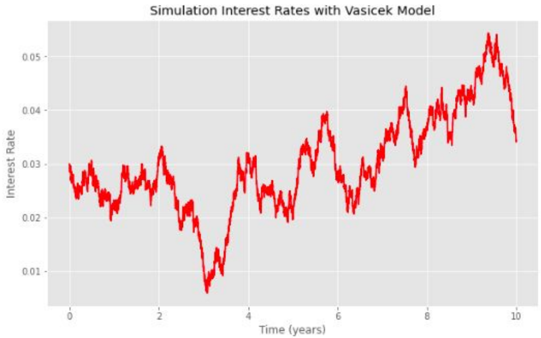
We simulate one path, with these unknown parameters, this is our historical dataset:

Real Parameters

```
# Model parameters we want to estimate
a = .5    # Mean reversion speed
b = 0.03  # Long-term mean
sigma = 0.01    # Volatility
```

Simulation one path

```
r0 = 0.03   # Initial short rate
T = 10      # Time horizon
num_steps = 10000   # Number of steps
num_paths = 1    # Number of paths

# Simulate Vasicek model
simulated_rates = vasicek(r0, a, b, sigma, T, num_steps, num_paths)

# Time axis
time_axis = np.linspace(0, T, num_steps + 1)

# Plotting multiple paths with time on x-axis
plt.figure(figsize=(10, 6))
plt.title('Simulation Interest Rates with Vasicek Model')
plt.xlabel('Time (years)')
plt.ylabel('Interest Rate')
for i in range(num_paths):
    plt.plot(time_axis, simulated_rates[:, i], color='red')

plt.show()
```



Python Code

Real parameters, unknown, that we want to estimate:

a = 0.5, b = 0.03, sigma = 0.01

Least Squares

```
def Vasicek_LS(r, dt):

    #Linear Regression
    r0 = r[:-1,]
    r1 = r[1:, 0]
    reg = LinearRegression().fit(r0, r1)

    #estimation a and b
    a_LS = (1 - reg.coef_) / dt
    b_LS = reg.intercept_ / dt / a_LS

    #estimation sigma
    epsilon = r[1:, 0] - r[:-1,0] * reg.coef_
    sigma_LS = np.std(epsilon) / dt**.5

    return a_LS[0], b_LS[0], sigma_LS
```

```
LS_Estimate = Vasicek_LS(simulated_rates, T / num_steps)
print("a_est: " + str(np.round(LS_Estimate[0],3)))
print("b_est: " + str(np.round(LS_Estimate[1],3)))
print("sigma_est: " + str(np.round(LS_Estimate[2],3)))
```

```
a_est: 0.551
b_est: 0.031
sigma_est: 0.01
```

Maximum Likelihood Estimation

```
def Vasicek_MLE(r, dt):
    r = r[:, 0]
    n = len(r)
    #estimation a and b
    S0 = 0
    S1 = 0
    S00 = 0
    S01 = 0
    for i in range(n-1):
        S0 = S0 + r[i]
        S1 = S1 + r[i + 1]
        S00 = S00 + r[i] * r[i]
        S01 = S01 + r[i] * r[i + 1]
    S0 = S0 / (n-1)
    S1 = S1 / (n-1)
    S00 = S00 / (n-1)
    S01 = S01 / (n-1)
    b_MLE = (S1 * S00 - S0 * S01) / (S0 * S1 - S0**2 - S01 + S00)
    a_MLE = 1 / dt * np.log((S0 - b_MLE) / (S1 - b_MLE))

    #estimation sigma
    beta = 1 / a * (1 - np.exp(-a * dt))
    temp = 0
    for i in range(n-1):
        mi = b * a * beta + r[i] * (1 - a * beta)
        temp = temp + (r[i+1] - mi)**2
    sigma_MLE = (1 / ((n - 1) * beta * (1 - .5 * a * beta)) * temp)**.5
    return a_MLE, b_MLE, sigma_MLE
```

```
MLE_Estimate = Vasicek_MLE(simulated_rates, T / num_steps)
print("a_est: " + str(np.round(MLE_Estimate[0],3)))
print("b_est: " + str(np.round(MLE_Estimate[1],3)))
print("sigma_est: " + str(np.round(MLE_Estimate[2],3)))
```

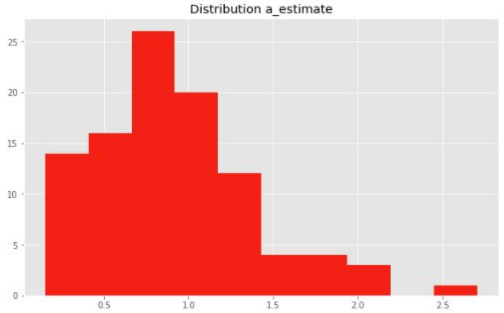
```
a_est: 0.551
b_est: 0.031
sigma_est: 0.01
```

Python Code

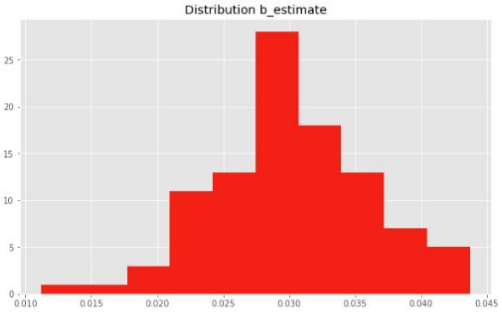
If you simulate several paths, you will remark that b and sigma estimates are quite stable contrary to a, which can then be more difficult to estimate with a strong accuracy in practice.

```python
# Model parameters
r0 = 0.02  # Initial short rate
a = 0.5    # Mean reversion speed
b = 0.03  # Long-term mean
sigma = 0.01    # Volatility
T = 10       # Time horizon
num_steps = 10000  # Number of steps
num_paths = 100   # Number of paths

a_est = []
b_est = []
sigma_est = []
for i in range(num_paths):
    simulated_rates = vasicek(r0, a, b, sigma, T, num_steps, 1)
    LS_Estimate = Vasicek_LS(simulated_rates, T / num_steps)
    a_est.append(LS_Estimate[0])
    b_est.append(LS_Estimate[1])
    sigma_est.append(LS_Estimate[2])
```

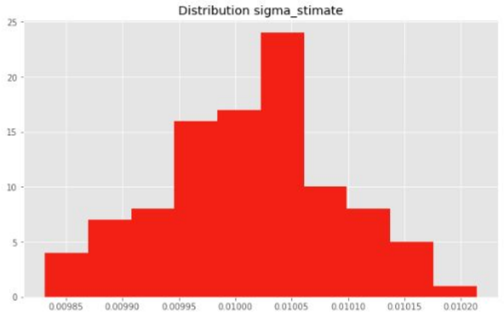
```python
plt.figure(figsize=(10, 6))
plt.title('Distribution a_estimate')
plt.hist(a_est,bins = 10);
```




```python
plt.figure(figsize=(10, 6))
plt.title('Distribution b_estimate')
plt.hist(b_est,bins = 10);
```




```python
plt.figure(figsize=(10, 6))
plt.title('Distribution sigma_stimate')
plt.hist(sigma_est, bins = 10);
```



Contact Us

website: www.quant-next.com

email: contact@quant-next.com

Follow us on LinkedIn

Disclaimer

This document is for educational and information purposes only.

It does not intend to be and does not constitute financial advice, investment advice, trading advice or any other advice, recommendation or promotion of any particular investments.