Time Series Analysis (14)

GMM (Generalized Method of Moments) with GARCH

> 前面我们介绍了GARCH方法和实践，当时我们在估算参数的时候采用的最大似然估计(MLE)。本篇我们要介绍另外一个可能更适合金融时序分析的参数估算方法：GMM(Generalized Method of Moments)。GMM在统计模型中的参数估算问题上起着非常重要的作用，并有广泛应用。

理论部分

- 在经济学和统计学中，GMM是用来估算统计模型参数的一种通用方法。GMM通常比较适合于准参数统计模型(semiparametric models),尤其是在所需参数是有限维度的，且数据的分布是未知的情况下，而这种情况MLE并不十分适用。 这种方法需要为模型定义一定的矩条件(moment conditions)，而这些矩条件是模型参数和数据的函数。GMM方法目标是最小化矩条件的样本平均值的范数。 一般来说，GMM方法的估算结果具有一致性、近正态分布性和高效性等优点。

- 假如我们有观察值$\{Y_t; t = 1, 2, . . . T\}$，其中每一个变量$Y_t$是一个n维随机变量。我们认为数据是来自某个特定的统计模型，该模型的参数为θ。我们的目标是估算出这个参数的真实值θ_0，或者是一个比较靠近真实值的结果。

- GMM一般假设数据是由一个弱平稳各态历经随机过程产生的。(独立同分布变量是满足这个条件的)

- 我们定义矩条件：
$$m(\theta_0) = E[g(Y_t, \theta_0)] = 0$$
当$\theta \neq \theta_0$时，m也不相等。$g(Y, \theta)$是一个我们定义的接受向量为参数值的矩函数。
最简单的方式是采用样本平均值求期望：
$$\hat{m}(\theta) = \frac{1}{T}\sum_{t=1}^{T} g(Y_t, \theta)$$
根据大数法则，我们可以通过最小化其范数来求解θ，二阶范数定义为:
$$\|\hat{m}(\theta)\|^2_W = \hat{m}(\theta)^T W \hat{m}(\theta)$$
其中，W是正定权重矩阵，实践中多是从数据中计算出来的，记作$\hat{W}$。
所以，我们就是要求解：
$$\hat{\theta} = argmin_\theta(\frac{1}{T}\sum_{t=1}^{T} g(Y_t, \theta))^T \hat{W}(\frac{1}{T}\sum_{t=1}^{T} g(Y_t, \theta))$$

- $g(Y_t, \theta)$可以选择任何连续函数
例如最简单的一种方式就是线性方程，即一阶矩。$E[g(Y_t, \theta)] = \mu$，这里μ就是观测值的平均值。

实践部分

In [1]:

```
import cvxopt
from functools import partial
import math
import numpy as np
import scipy
from scipy import stats
import statsmodels.api as sm
from statsmodels.stats.stattools import jarque_bera

import matplotlib.pyplot as plt
```

我们按照以下模型模拟GARCH过程

$$\begin{aligned}\sigma_1 &= \sqrt{\frac{a_0}{1 - a_1 - b_1}} \\ \sigma_t &= a_0 + a_1 x_{t-1}^2 + b_1 \sigma_{t-1}^2 \\ x_t &= \sigma_t \epsilon_t \\ \epsilon &\sim \mathcal{N}(0, 1)\end{aligned}$$

参数为:

$$a_0 = 1, a_1 = 0.1, b_1 = 0.8$$

采用蒙特卡洛采样，去除前10%的模拟数据。

In [32]:

```
# Define parameters
a0 = 1.0
a1 = 0.1
b1 = 0.8
sigma1 = math.sqrt(a0 / (1 - a1 - b1))
```

In [33]:

```
def simulate_GARCH(T, a0, a1, b1, sigma1):

    # Initialize our values
    X = np.ndarray(T)
    sigma = np.ndarray(T)
    sigma[0] = sigma1

    for t in range(1, T):
        # Draw the next x_t
        X[t - 1] = sigma[t - 1] * np.random.normal(0, 1)
        # Draw the next sigma_t
        sigma[t] = math.sqrt(a0 + b1 * sigma[t - 1]**2 + a1 * X[t - 1]**2)

    X[T - 1] = sigma[T - 1] * np.random.normal(0, 1)

    return X, sigma
```

我们将比较GARCH过程与高斯过程的尾部，理论上说GARCH过程应该具有厚尾特性。

In [35]:

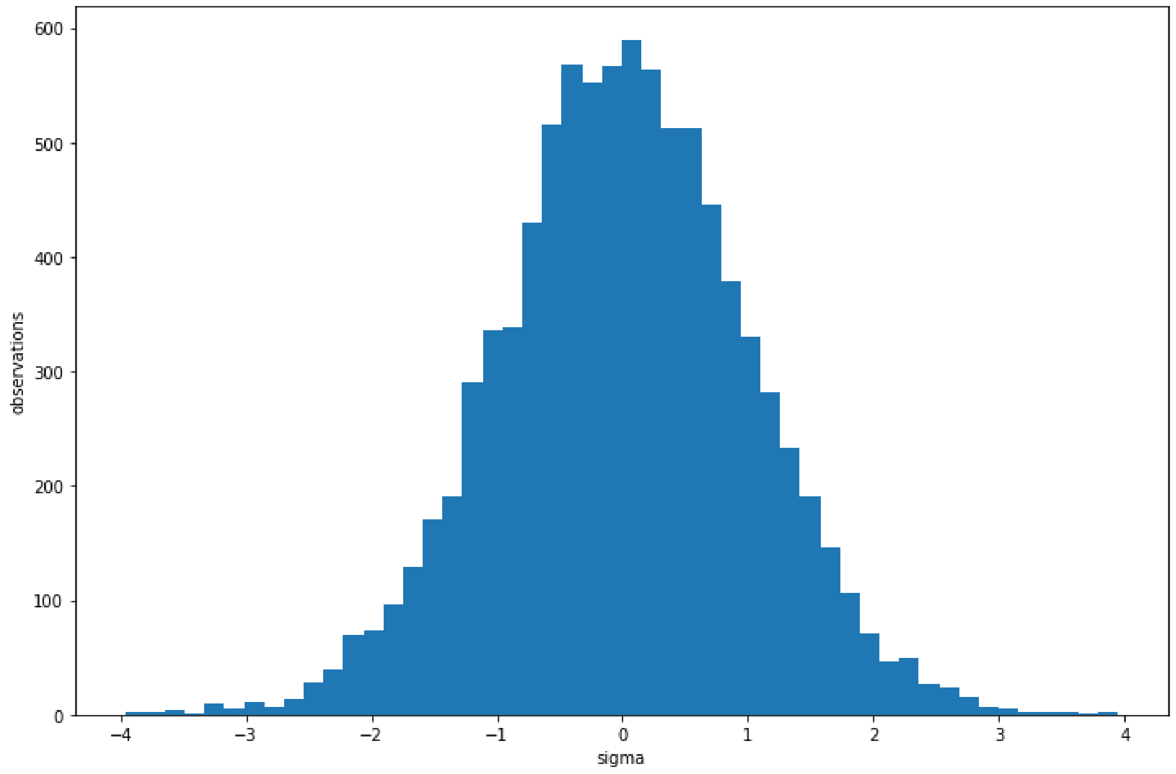
```
X, _ = simulate_GARCH(10000, a0, a1, b1, sigma1)
X = X[1000:] # Drop burn in
X = X / np.std(X) # Normalize X

def compare_tails_to_normal(X):
    # Define matrix to store comparisons
    A = np.zeros((2,4))
    for k in range(4):
        A[0, k] = len(X[X > (k + 1)]) / float(len(X)) # Estimate tails of X
        A[1, k] = 1 - stats.norm.cdf(k + 1) # Compare to Gaussian distribution
    return A

compare_tails_to_normal(X)
```

Out[35]:

```
array([[1.58555556e-01, 2.27777778e-02, 2.00000000e-03, 0.00000000e+00],
       [1.58655254e-01, 2.27501319e-02, 1.34989803e-03, 3.16712418e-05]])
```

In [36]:

```
plt.figure(figsize=(12,8))
plt.hist(X, bins=50)
plt.xlabel('sigma')
plt.ylabel('observations');
```

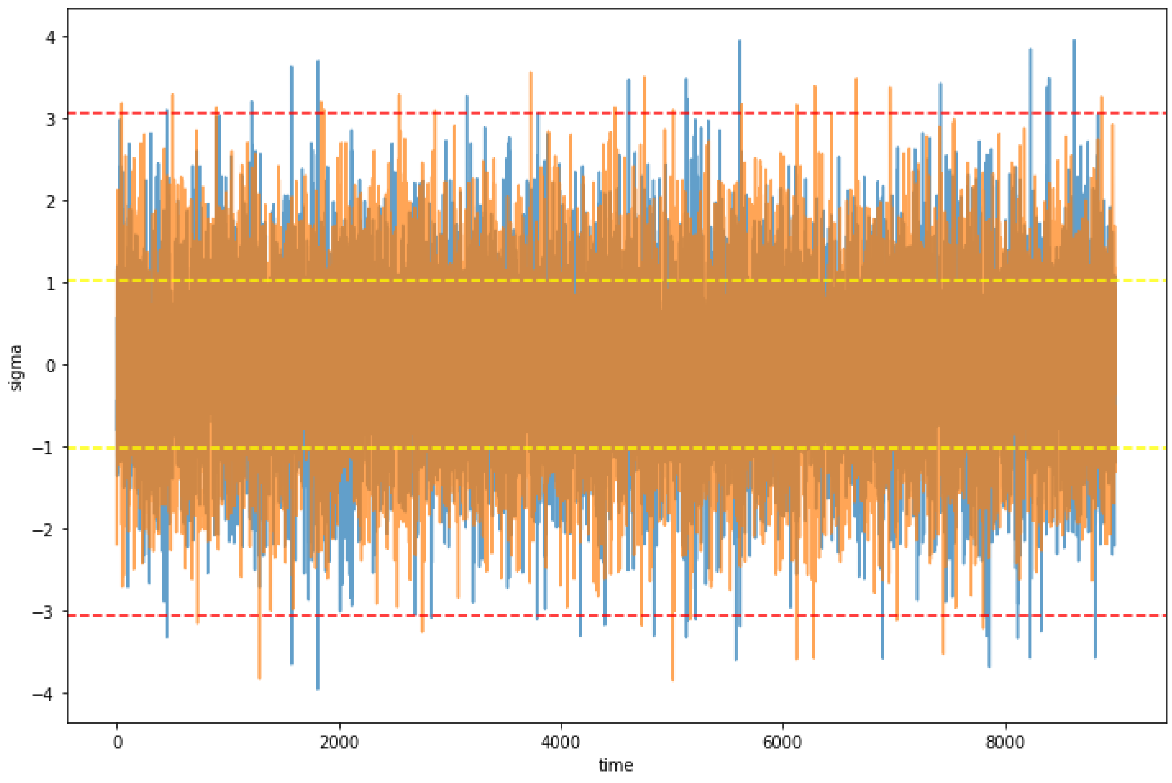


In [37]:

```
# Sample values from a normal distribution
X2 = np.random.normal(0, 1, 9000)
both = np.matrix([X, X2])
```

In [38]:

```
# Plot both the GARCH and normal values
plt.figure(figsize=(12,8))
plt.plot(both.T, alpha=.7);
plt.axhline(X2.std(), color='yellow', linestyle='--')
plt.axhline(-X2.std(), color='yellow', linestyle='--')
plt.axhline(3*X2.std(), color='red', linestyle='--')
plt.axhline(-3*X2.std(), color='red', linestyle='--')
plt.xlabel('time')
plt.ylabel('sigma');
```



上图中，蓝色代表GARCH过程，黄线是两个标准差范围，红线是三个标准差范围。可以明显看出，GARCH模型的数据超过红线的次数和幅度高于正态过程。

第一步

我们检查序列的ARCH特性，检验如下模型的显著性：

$$x_t^2 = a_0 + a_1 x_{t-1}^2 + \ldots + x_{t-p}^2$$

采用OLS回归来估算参数$\hat{\theta} = (a_0, a_1, \ldots, a_p)$和协方差矩阵$\hat{\Omega}$ 计算统计量：

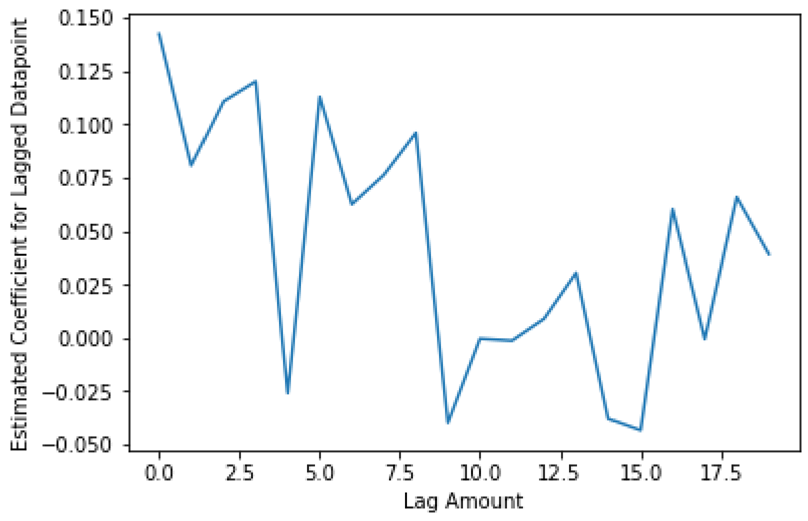
$$F = \hat{\theta}\hat{\Omega}^{-1}\hat{\theta}'$$

我们设$p = 20$

In [39]:

```
X, _ = simulate_GARCH(1100, a0, a1, b1, sigma1)
X = X[100:] # Drop burn in

p = 20

# Drop the first 20 so we have a lag of p's
Y2 = (X**2)[p:]
X2 = np.ndarray((980, p))
for i in range(p, 1000):
    X2[i - p, :] = np.asarray((X**2)[i-p:i])[::-1]

model = sm.OLS(Y2, X2)
model = model.fit()
theta = np.matrix(model.params)
omega = np.matrix(model.cov_HC0)
F = np.asscalar(theta * np.linalg.inv(omega) * theta.T)

print(np.asarray(theta.T).shape)

plt.plot(range(20), np.asarray(theta.T))
plt.xlabel('Lag Amount')
plt.ylabel('Estimated Coefficient for Lagged Datapoint')

print('F = ' + str(F))

chi2dist = scipy.stats.chi2(p)
pvalue = 1-chi2dist.cdf(F)
print('p-value = ' + str(pvalue))

# Finally let's look at the significance of each a_p as measured by the standard deviations away fro
print(theta/np.diag(omega))
```

```
(20, 1)
F = 314.5695016883417
p-value = 0.0
[[ 69.69590505   28.55370456   38.34863481   67.24243561  -23.76256854
   32.81417587   28.72670244   21.09944792   65.72231451  -29.04143138
   -0.19125359   -0.81082864   13.17250074   27.72390135  -27.42987519
  -40.64382106   54.07913911   -0.64594302   59.47254611   49.84119476]]
```



MLE

检验其GARCH特性显著性之后，我们首先采用MLE来估算GARCH参数。

In [40]:

```
X, _ = simulate_GARCH(10000, a0, a1, b1, sigma1)
X = X[1000:] # Drop burn in
```

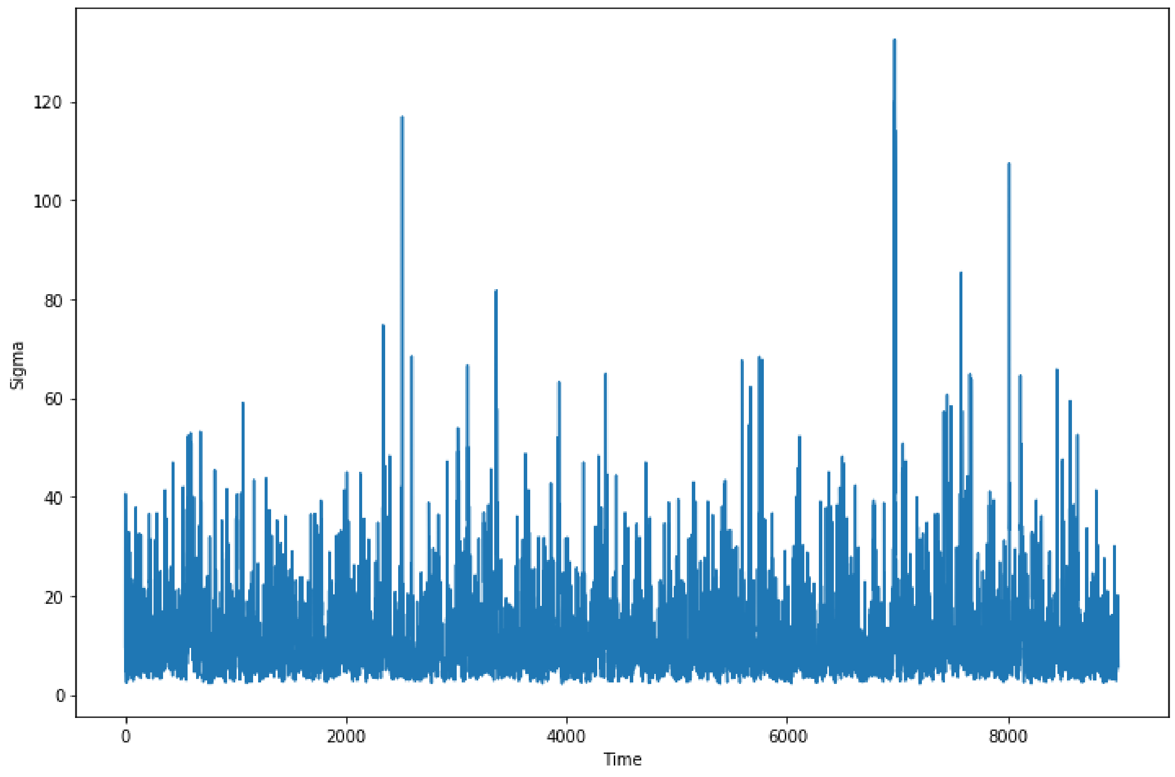
In [41]:

```
# Here's our function to compute the sigmas given the initial guess
def compute_squared_sigmas(X, initial_sigma, theta):

    a0 = theta[0]
    a1 = theta[1]
    b1 = theta[2]

    T = len(X)
    sigma2 = np.ndarray(T)

    sigma2[0] = initial_sigma ** 2

    for t in range(1, T):
        # Here's where we apply the equation
        sigma2[t] = a0 + a1 * X[t-1]**2 + b1 * sigma2[t-1]

    return sigma2
```

看一下我们所模拟的数据的情况

In [42]:

```
plt.figure(figsize=(12,8))
plt.plot(range(len(X)), compute_squared_sigmas(X, np.sqrt(np.mean(X**2)), (1, 0.5, 0.5)))
plt.xlabel('Time')
plt.ylabel('Sigma');
```



我们最大化其对数似然函数，也就是最小化负的对数似然函数。
约束条件为：$a_1 \geq 0, b_1 \geq 0, a_1 + b_1 < 1$

In [43]:

```
def negative_log_likelihood(X, theta):

    T = len(X)

    # Estimate initial sigma squared
    initial_sigma = np.sqrt(np.mean(X ** 2))

    # Generate the squared sigma values
    sigma2 = compute_squared_sigmas(X, initial_sigma, theta)

    # Now actually compute
    return -sum(
        [-np.log(np.sqrt(2.0 * np.pi)) -
        (X[t] ** 2) / (2.0 * sigma2[t]) -
        0.5 * np.log(sigma2[t]) for
         t in range(T)]
    )
```

In [44]:

```
# Make our objective function by plugging X into our log likelihood function
objective = partial(negative_log_likelihood, X)

# Define the constraints for our minimizer
def constraint1(theta):
    return np.array([1 - (theta[1] + theta[2])])

def constraint2(theta):
    return np.array([theta[1]])

def constraint3(theta):
    return np.array([theta[2]])

cons = ({'type': 'ineq', 'fun': constraint1},
        {'type': 'ineq', 'fun': constraint2},
        {'type': 'ineq', 'fun': constraint3})

# Actually do the minimization
result = scipy.optimize.minimize(objective, (1, 0.5, 0.5),
                        method='SLSQP',
                        constraints = cons)
theta_mle = result.x
print('theta MLE: ' + str(theta_mle))
```

```
d:\Anaconda3\lib\site-packages\ipykernel_launcher.py:16: RuntimeWarning: invalid val
ue encountered in log
  app.launch_new_instance()

theta MLE: [1.06202894 0.08797051 0.80050204]
```

最后还需要检查两点:

1. 残差的尾部的厚度
2. 残差的Jarque-Bera正态检验

In [45]:

```
def check_theta_estimate(X, theta_estimate):
    initial_sigma = np.sqrt(np.mean(X ** 2))
    sigma = np.sqrt(compute_squared_sigmas(X, initial_sigma, theta_estimate))
    epsilon = X / sigma
    print('Tails table')
    print(compare_tails_to_normal(epsilon / np.std(epsilon)))
    print('')

    _, pvalue, _, _ = jarque_bera(epsilon)
    print('Jarque-Bera probability normal: ' + str(pvalue))

check_theta_estimate(X, theta_mle)
```

```
Tails table
[[1.59444444e-01 2.16666667e-02 1.66666667e-03 0.00000000e+00]
 [1.58655254e-01 2.27501319e-02 1.34989803e-03 3.16712418e-05]]

Jarque-Bera probability normal: 0.6247564240277035
```

GMM

采用GMM方法来估算GARCH(1,1)模型参数，采用moments如下:

1. 残差$\hat{\epsilon_t} = x_t/\hat{\sigma_t}$
2. 残差的方差$\hat{\epsilon_t}^2$
3. 偏度$\nu_3/\hat{\sigma^3}_t = (\hat{\epsilon_t} - E[\hat{\epsilon_t}])^3/\hat{\epsilon_t}^3$
4. 峰度$\nu_4/\hat{\sigma^4}_t = (\hat{\epsilon_t} - E[\hat{\epsilon_t}])^4/\hat{\epsilon_t}^4$

In [46]:

```python
# The n-th standardized moment
# skewness is 3, kurtosis is 4
def standardized_moment(x, mu, sigma, n):
    return ((x - mu) ** n) / (sigma ** n)
```

GMM的估算分为三步，
设置W为初始单位矩阵

1. 估算$\hat{\theta}_1$，最小化下式:
$$\hat{\theta} = argmin_{\theta}(\frac{1}{T}\sum_{t=1}^{T} g(Y_t, \theta))^T \hat{W}(\frac{1}{T}\sum_{t=1}^{T} g(Y_t, \theta))$$
2. 根据上一步计算的$\hat{\theta}_1$重新计算W
$$(\sum_{t=1}^{T} g(Y_t, \theta))^T g(Y_t, \theta)))^{-1}$$
3. 重复直到收敛，或者两次估算的值之间的差非常小。

In [47]:

```
def gmm_objective(X, W, theta):
    # Compute the residuals for X and theta
    initial_sigma = np.sqrt(np.mean(X ** 2))
    sigma = np.sqrt(compute_squared_sigmas(X, initial_sigma, theta))
    e = X / sigma

    # Compute the mean moments
    m1 = np.mean(e)
    m2 = np.mean(e ** 2) - 1
    m3 = np.mean(standardized_moment(e, np.mean(e), np.std(e), 3))
    m4 = np.mean(standardized_moment(e, np.mean(e), np.std(e), 4) - 3)

    G = np.matrix([m1, m2, m3, m4]).T

    return np.asscalar(G.T * W * G)

def gmm_variance(X, theta):
    # Compute the residuals for X and theta
    initial_sigma = np.sqrt(np.mean(X ** 2))
    sigma = np.sqrt(compute_squared_sigmas(X, initial_sigma, theta))
    e = X / sigma

    # Compute the squared moments
    m1 = e ** 2
    m2 = (e ** 2 - 1) ** 2
    m3 = standardized_moment(e, np.mean(e), np.std(e), 3) ** 2
    m4 = (standardized_moment(e, np.mean(e), np.std(e), 4) - 3) ** 2

    # Compute the covariance matrix g * g'
    T = len(X)
    s = np.ndarray((4, 1))
    for t in range(T):
        G = np.matrix([m1[t], m2[t], m3[t], m4[t]]).T
        s = s + G * G.T

    return s / T
```

In [48]:

```
# Initialize GMM parameters
W = np.identity(4)
gmm_iterations = 10

# First guess
theta_gmm_estimate = theta_mle

# Perform iterated GMM
for i in range(gmm_iterations):
    # Estimate new theta
    objective = partial(gmm_objective, X, W)
    result = scipy.optimize.minimize(objective, theta_gmm_estimate, constraints=cons)
    theta_gmm_estimate = result.x
    print('Iteration ' + str(i) + ' theta: ' + str(theta_gmm_estimate))

    # Recompute W
    W = np.linalg.inv(gmm_variance(X, theta_gmm_estimate))


check_theta_estimate(X, theta_gmm_estimate)
```

```
Iteration 0 theta: [1.05570865 0.14208397 0.75473263]
Iteration 1 theta: [1.05575988 0.14243671 0.75510334]
Iteration 2 theta: [1.05575988 0.14243671 0.75510334]
Iteration 3 theta: [1.05575988 0.14243671 0.75510334]
Iteration 4 theta: [1.05575988 0.14243671 0.75510334]
Iteration 5 theta: [1.05575988 0.14243671 0.75510334]
Iteration 6 theta: [1.05575988 0.14243671 0.75510334]
Iteration 7 theta: [1.05575988 0.14243671 0.75510334]
Iteration 8 theta: [1.05575988 0.14243671 0.75510334]
Iteration 9 theta: [1.05575988 0.14243671 0.75510334]
Tails table
[[1.58666667e-01 2.03333333e-02 1.77777778e-03 0.00000000e+00]
 [1.58655254e-01 2.27501319e-02 1.34989803e-03 3.16712418e-05]]

Jarque-Bera probability normal: 0.7672679639553297
```

用GARCH来预测

初始化σ

In [49]:

```
sigma_hats = np.sqrt(compute_squared_sigmas(X, np.sqrt(np.mean(X**2)), theta_mle))
initial_sigma = sigma_hats[-1]
initial_sigma
```

Out[49]:

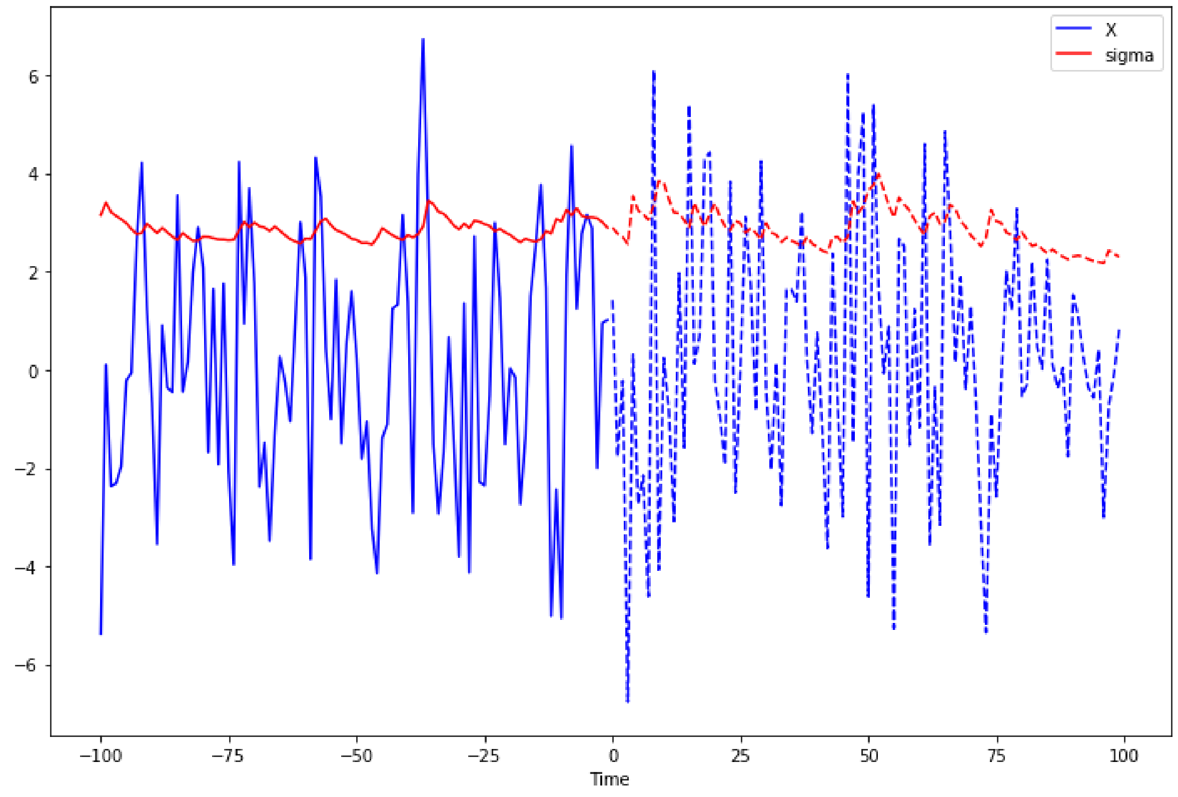
```
2.8936229656984263
```

In [50]:

```
a0_estimate = theta_gmm_estimate[0]
a1_estimate = theta_gmm_estimate[1]
b1_estimate = theta_gmm_estimate[2]

X_forecast, sigma_forecast = simulate_GARCH(100, a0_estimate, a1_estimate, b1_estimate, initial_sigm
```

In [51]:

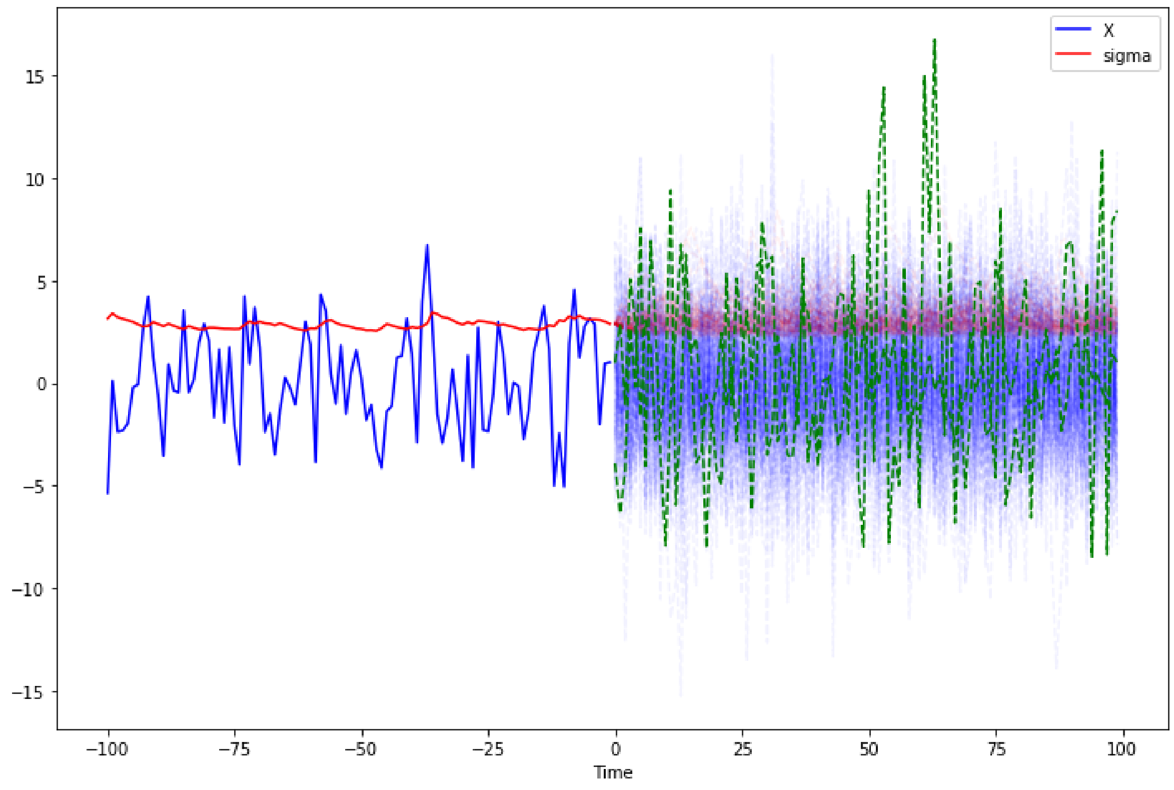
```
plt.figure(figsize=(12,8))
plt.plot(range(-100, 0), X[-100:], 'b-')
plt.plot(range(-100, 0), sigma_hats[-100:], 'r-')
plt.plot(range(0, 100), X_forecast, 'b--')
plt.plot(range(0, 100), sigma_forecast, 'r--')
plt.xlabel('Time')
plt.legend(['X', 'sigma']);
```



蒙特卡洛采样

In [52]:

```
plt.figure(figsize=(12,8))
plt.plot(range(-100, 0), X[-100:], 'b-')
plt.plot(range(-100, 0), sigma_hats[-100:], 'r-')
plt.xlabel('Time')
plt.legend(['X', 'sigma'])



max_X = [-np.inf]
min_X = [np.inf]
for i in range(100):
    X_forecast, sigma_forecast = simulate_GARCH(100, a0_estimate, a1_estimate, b1_estimate, initial_
    if max(X_forecast) > max(max_X):
        max_X = X_forecast
    elif min(X_forecast) < min(max_X):
        min_X = X_forecast
    plt.plot(range(0, 100), X_forecast, 'b--', alpha=0.05)
    plt.plot(range(0, 100), sigma_forecast, 'r--', alpha=0.05)

# Draw the most extreme X values specially
plt.plot(range(0, 100), max_X, 'g--', alpha=1.0)
plt.plot(range(0, 100), min_X, 'g--', alpha=1.0);
```



总结

本片文章从理论和实践上介绍了GMM模型用于GARCH模型的参数估算，实际上对于时序预测技术还有很多其他的优秀方案，例如Bayesian Cones等。本系列也会在后续文章中进行介绍。