Financial Signal Processing Coursework

Student: Xinyuan Xu

CID: 01183830

Feburary 2020

1 Regression Methods

Question 1.1.1

In [1]:

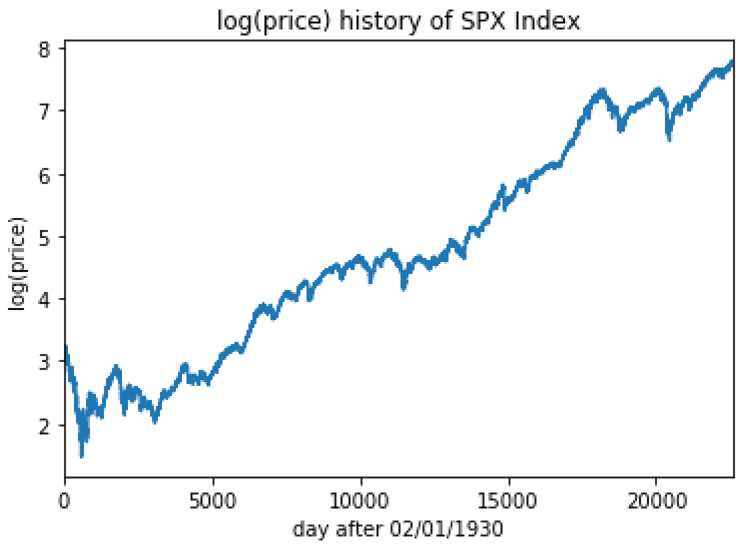
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

px = pd.read_csv("priceData.csv")
# the origial hear of the price column is "SPX Index"
# the space will affect calling of that column, so is replaced by "_" by directly editing t
# print(px.SPX_Index)
date = px.date
price = px.SPX_Index
price = pd.Series(price)
logpx = np.log(px.SPX_Index)
logpx.plot()

plt.xlabel('day after 02/01/1930')
plt.ylabel('log(price)')
plt.title('log(price) history of SPX Index')
```

Out[1]:

```
Text(0.5, 1.0, 'log(price) history of SPX Index')
```



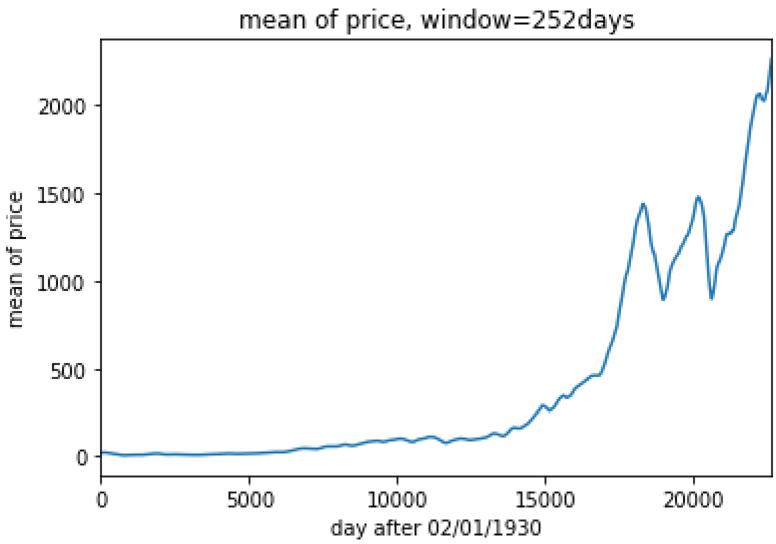
Question 1.1.2

In [2]:

```
plt.figure()
mean_price = price.rolling(252,min_periods=1).mean()
mean_price.plot()
plt.xlabel('day after 02/01/1930')
plt.ylabel('mean of price')
plt.title('mean of price, window=252days')
```

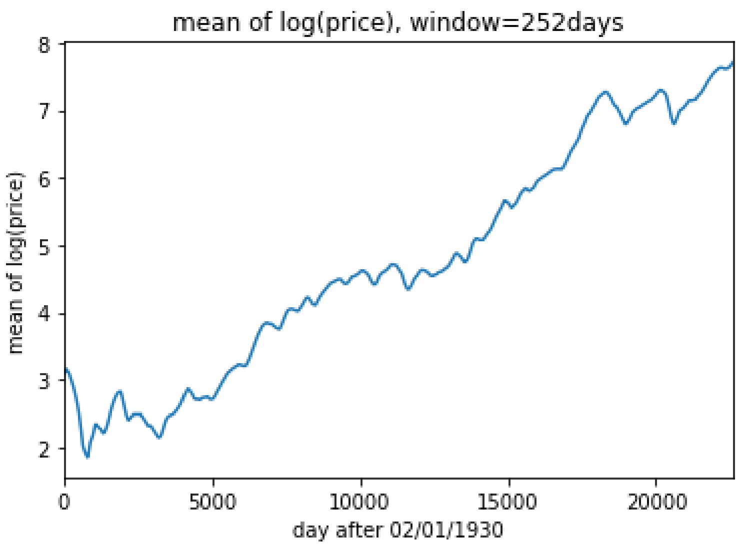
Out[2]:

```
Text(0.5, 1.0, 'mean of price, window=252days')
```



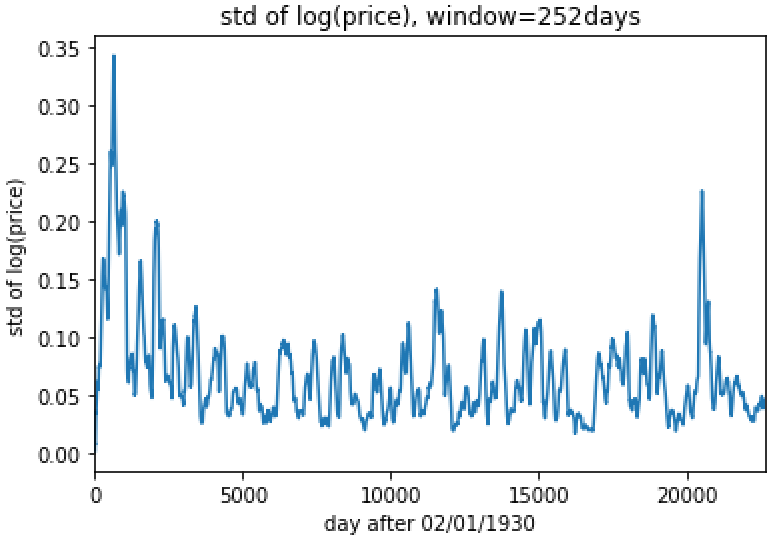
In [3]:

```
plt.figure()
mean_log_price=logpx.rolling(252, min_periods=1).mean()
mean_log_price.plot()
plt.xlabel('day after 02/01/1930')
plt.ylabel('mean of log(price)')
plt.title('mean of log(price), window=252days')
plt.show()
```



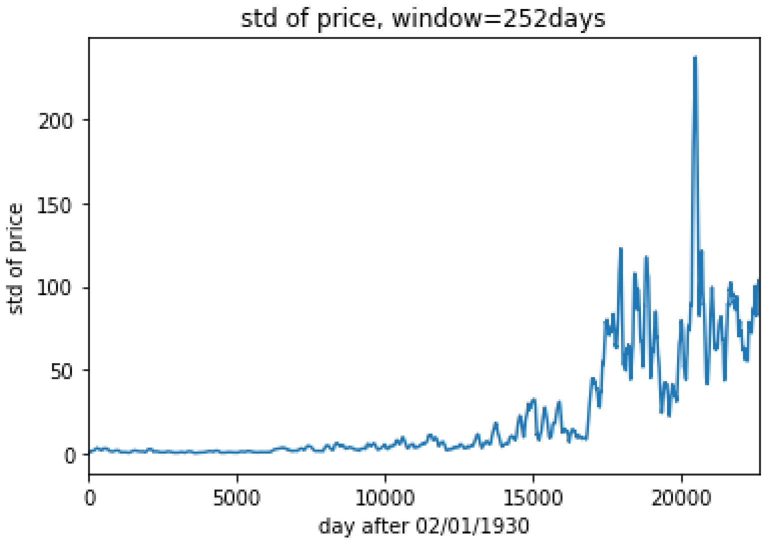
In [4]:

```
plt.figure()
std_log_price = logpx.rolling(252,min_periods=1).std()
std_log_price.plot()
plt.xlabel('day after 02/01/1930')
plt.ylabel('std of log(price)')
plt.title('std of log(price), window=252days')
plt.show()
```



In [5]:

```
plt.figure()
std_price = price.rolling(252,min_periods=1).std()
std_price.plot()
plt.xlabel('day after 02/01/1930')
plt.ylabel('std of price')
plt.title('std of price, window=252days')
plt.show()
```



The price time-series is not stationary at all, because the mean and std of both price and log price are always changing with time.

Question1.1.3

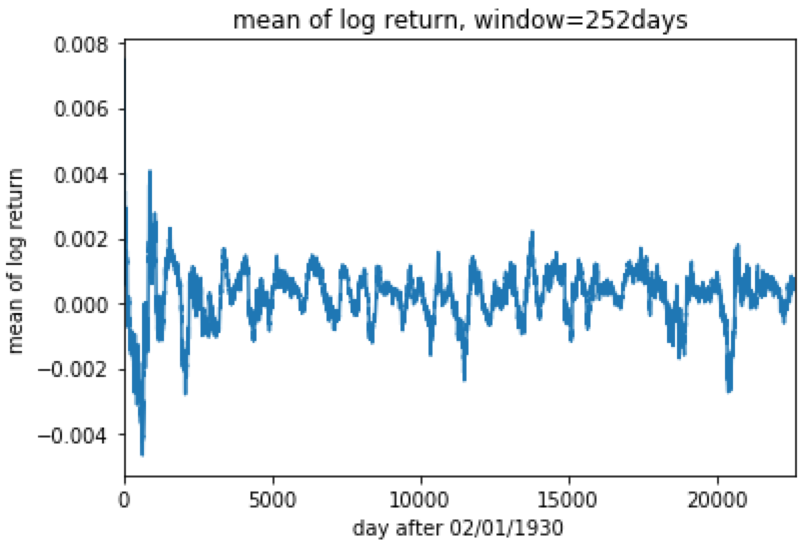
In [6]:

```
logret = logpx.diff()
simpret = price.pct_change()

plt.figure()
logret.rolling(252,min_periods=1).mean().plot()
plt.xlabel('day after 02/01/1930')
plt.ylabel('mean of log return')
plt.title('mean of log return, window=252days')
```

Out[6]:

```
Text(0.5, 1.0, 'mean of log return, window=252days')
```

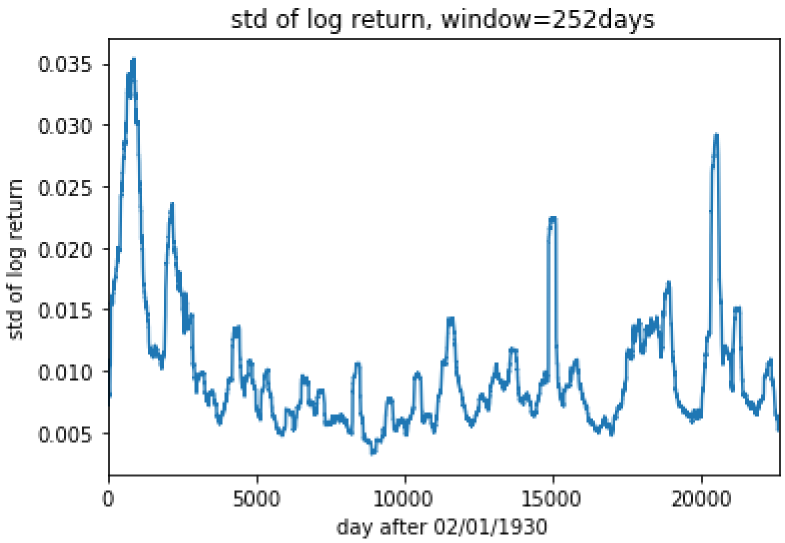


In [7]:

```
plt.figure()
logret.rolling(252,min_periods=1).std().plot()
plt.xlabel('day after 02/01/1930')
plt.ylabel('std of log return')
plt.title('std of log return, window=252days')
```

Out[7]:

```
Text(0.5, 1.0, 'std of log return, window=252days')
```



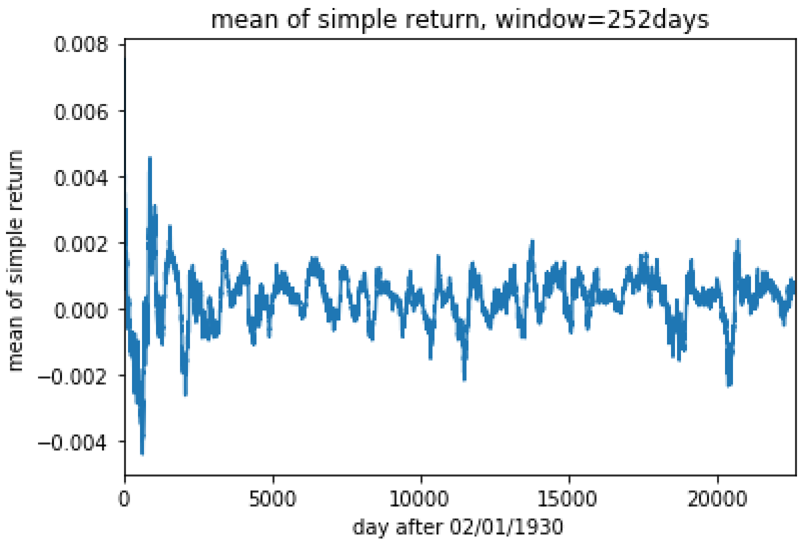
Log return is not stationary. Mean of log return is relatively stable around 0, which is more stationary than mean of mean of price or log price. Similar to std of price or log price, standard deviation of log return could vary between 0.035 and 0.005 during this time peroid, so is not very stationary.

In [8]:

```
plt.figure()
simpret.rolling(252,min_periods=1).mean().plot()
plt.xlabel('day after 02/01/1930')
plt.ylabel('mean of simple return')
plt.title('mean of simple return, window=252days')
```

Out[8]:

```
Text(0.5, 1.0, 'mean of simple return, window=252days')
```

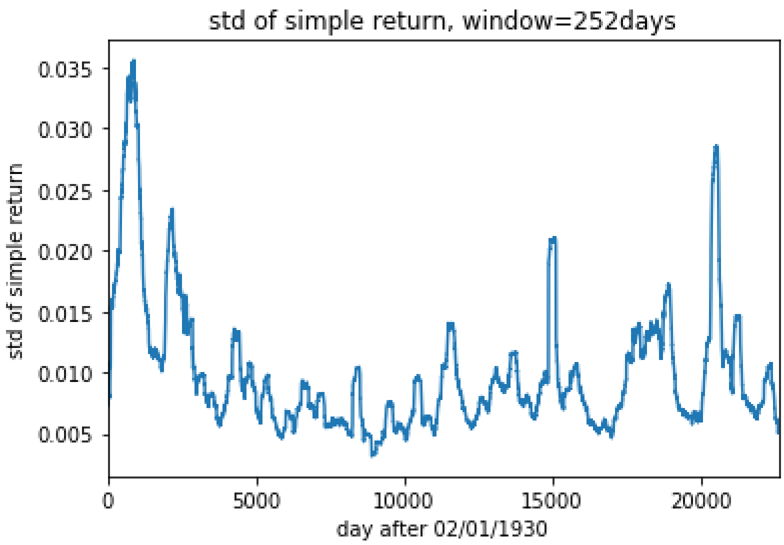


In [9]:

```
plt.figure()
simpret.rolling(252,min_periods=1).std().plot()
plt.xlabel('day after 02/01/1930')
plt.ylabel('std of simple return')
plt.title('std of simple return, window=252days')
```

Out[9]:

```
Text(0.5, 1.0, 'std of simple return, window=252days')
```



The mean and std of simple return has similar shape with that of log return, because at small return, log return is approximately equal to simple return . As std of simple return also varies a lot, simple return is not stationary as well.

In [10]:

```
print(logret[100])
print(simpret[100])
```

```
0.008288485080056862
0.00832292967124415
```

Question 1.1.4

Suitability of Log return over simple return:

1) behavior of log return is symmetric: a 10% increase followed by a 10% decrease in terms of log return means price goes back to the original number; (this is also time daditivity)

2) log function is a monotonic function which preserves "relative ordering"

3) log function compresses the range of data and also conditions the original data probability distribution function

4) if price follows log-normal distribution, then log return is conveniently normal distributed.

5) other advantages include mathematical tractability, numerical stability (safer to add instead of multiply small numbersf) and homomorphic processing.

In [11]:

```
from scipy import stats
# type(logret) pandas series
logret = logret.dropna()
stats.jarque_bera(logret)
```

Out[11]:

```
(311807.2704898989, 0.0)
```

In [12]:

```
simpret = simpret.dropna()
stats.jarque_bera(simpret)
```

Out[12]:

```
(284160.67730312835, 0.0)
```

In [13]:

```
stats.jarque_bera(logret[:30])
```

Out[13]:

```
(1.463519177132134, 0.4810617741357005)
```

In [14]:

```
stats.jarque_bera(simpret[:30])
```

Out[14]:

```
(1.4592296470606705, 0.48209464584652406)
```

In [15]:

```
stats.jarque_bera(logret[:300])
```

Out[15]:

```
(35.931594681478074, 1.5759896165867815e-08)
```

In [16]:

```
stats.jarque_bera(simpret[:300])
```

Out[16]:

```
(27.202029022873496, 1.239237221728473e-06)
```

Over a long time scale (the whole history of data set), the statistics of both log return and simple return are large numbers, meaning they do not follow a normal distribution. Over a period of 30 days, both simple return and log return follows normal distribution. So the assuption of normality only applies in the short term.

Question 1.1.5

| Day 0 | Day 1 | Day 2

Price: 1 | 2 | 1

Simple return: / | 100% | -50%

Log return: / | log(2) | log(0.5) = -log(2)

This is an example that demonstrates the dame additivity or symmetricity of log return.

Question 1.16:

First, Log return should not be used over a long time scale, because it is unreasonable to assume log-mormality over a long period of time. Due to "periodic" financial crisis, financial data is negatively skewed, while log-normal distributions are positively skewed.

Second, log return could not be used in portforlio, as log return numbers could not be linearly added across asset. For example, log return of Apple stock and Microsoft stock of one day could not be added together to get their total return. Whereas simple returns are linearly additive.

1.2 ARMA vs ARIMA models for financial applications

Question 1.2.1

In [17]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.ar_model import AR
import copy
```

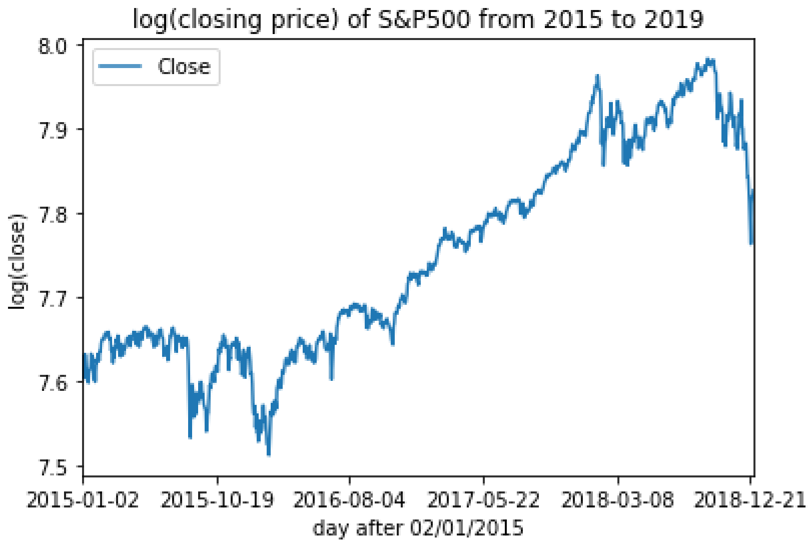
In [18]:

```
snp = pd.read_csv('snp_500_2015_2019.csv')
snp.set_index(['Date'],inplace=True)
snp_close = snp['Close'].to_frame().apply(np.log)
plt.figure()
snp_close.plot()
plt.xlabel('day after 02/01/2015')
plt.ylabel('log(close)')
plt.title('log(closing price) of S&P500 from 2015 to 2019')
```

Out[18]:

```
Text(0.5, 1.0, 'log(closing price) of S&P500 from 2015 to 2019')
```

```
<Figure size 432x288 with 0 Axes>
```



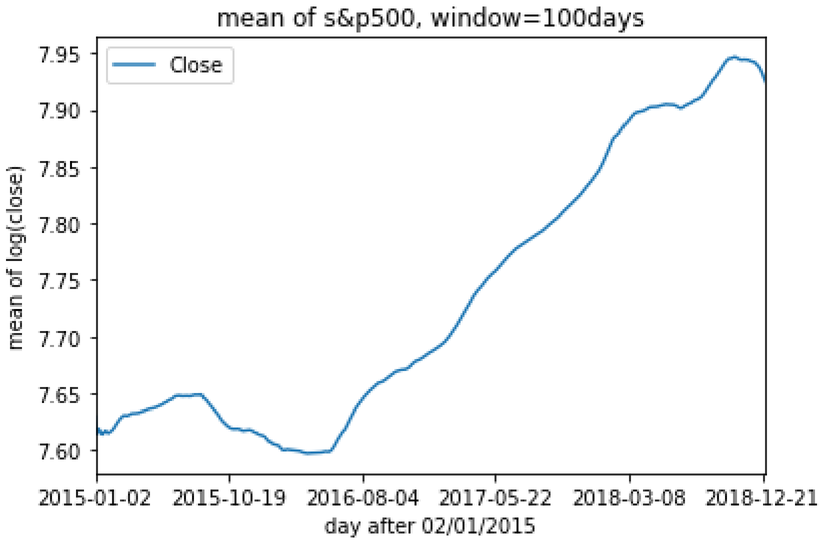
In [19]:

```
plt.figure()
mean_snp = snp_close.rolling(100,min_periods=1).mean()
mean_snp.plot()
plt.xlabel('day after 02/01/2015')
plt.ylabel('mean of log(close)')
plt.title('mean of s&p500, window=100days')
```

Out[19]:

```
Text(0.5, 1.0, 'mean of s&p500, window=100days')
```

```
<Figure size 432x288 with 0 Axes>
```



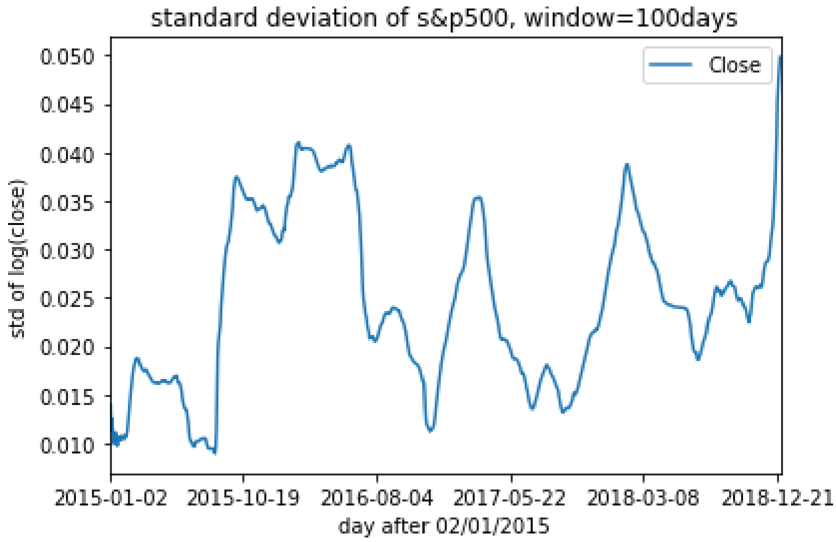
In [20]:

```
plt.figure()
mean_snp = snp_close.rolling(100,min_periods=1).std()
mean_snp.plot()
plt.xlabel('day after 02/01/2015')
plt.ylabel('std of log(close)')
plt.title('standard deviation of s&p500, window=100days')
```

Out[20]:

```
Text(0.5, 1.0, 'standard deviation of s&p500, window=100days')
```

```
<Figure size 432x288 with 0 Axes>
```



The two figures above clearly shows that S&P500 index over this 4 years is non stationary, thus ARIMA model should be used instead of ARMA model.

Question 1.2.2

In [21]:

```
snp_arma = copy.deepcopy(snp_close)
snp_arma.columns = ['True']
ARIMA(snp_arma, order=(1,0,0)).fit().params
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:219: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

Out[21]:

```
const          7.739998
ar.L1.True     0.997359
dtype: float64
```

The parameters suggest that

1) the log closing price centres around 7.74, which could be confirmed by direction observation of snp_close data

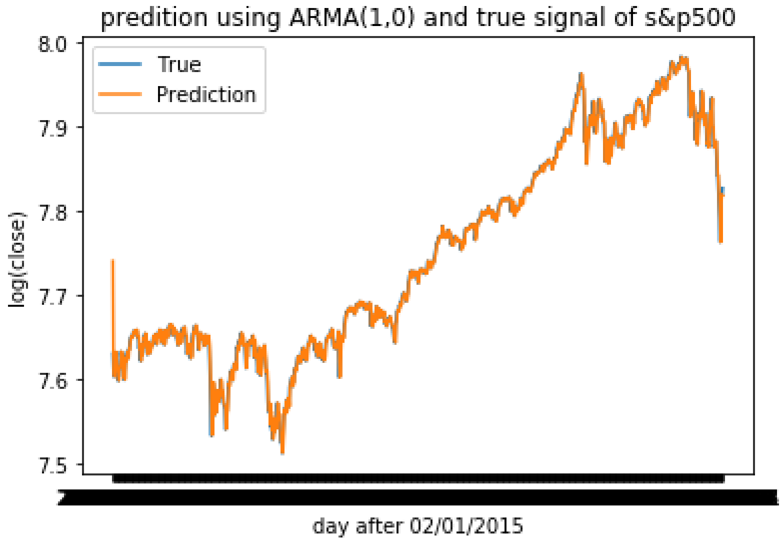
2) the closing price of one day is closly correlated to the price of the day before. This of course makes sense, as price rarely moves in large magnitude arbitrarily. By observing the plot below, the AR(1) model prediction follows the true signal quite closly.

In [22]:

```
snp_arma = copy.deepcopy(snp_close)
snp_arma.columns = ['True']
snp_arma['Res']=ARIMA(snp_arma, order=(1,0,0)).fit().resid
snp_arma['Prediction']=snp_arma['True']-snp_arma['Res']

plt.figure()
line1, = plt.plot(snp_arma['True'],label='True')
line2, = plt.plot(snp_arma['Prediction'],label='Prediction')
plt.legend(handles=[line1, line2])
plt.xlabel('day after 02/01/2015')
plt.ylabel('log(close)')
plt.title('predition using ARMA(1,0) and true signal of s&p500')
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:219: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```



In [23]:

```
MSE = snp_arma['Res'].pow(2).mean()
print(MSE)
# mean square error of ARMA(1,0) model
```

```
8.627195459688114e-05
```

Question 1.2.3

In [24]:

```
snp_arima = copy.deepcopy(snp_close)
snp_arima.columns = ['True']
ARIMA(snp_arima, order=(1,1,0)).fit().params
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:219: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:219: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

Out[24]:

```
const           0.000196
ar.L1.D.True   -0.008752
dtype: float64
```

In [25]:

```
snp_arima = copy.deepcopy(snp_close)
snp_arima.columns = ['True']
snp_arima['Res']=ARIMA(snp_arima, order=(1,1,0)).fit().resid
snp_arima['Prediction']=snp_arima['True']-snp_arima['Res']

plt.figure()
line1, = plt.plot(snp_arima['True'],label='True')
line2, = plt.plot(snp_arima['Prediction'],label='Prediction')
plt.legend(handles=[line1, line2])
plt.xlabel('day after 02/01/2015')
plt.ylabel('log(close)')
plt.title('predition using ARIMA(1,1,0) and true signal of s&p500')
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:219: ValueWarning: A date index has been provided, but it has no associ
ated frequency information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

In [26]:

```
MSE = snp_arima['Res'].pow(2).mean()
print(MSE)
# mean square error of ARIMA(1,1,0) model
```

7.428310856507248e-05

The two models are similarly meaningful since both predict the true signal quite well. ARMA model works better when the signal has been properly detrended. ARIMA model has has an integration term thus does require detrending before modeling.

Question 1.1.4

log function compresses the range of data and also conditions the original data probability distribution function. The prices originally have changing variance (see figure before), so log function would help suppressing this. As we can see from previous figure, the log(price) of S&P500 shows a linear growing trend in a previous graph. So the origial price has an exponential growing trend, which has been transformed to linear by log function, thus enables ARIMA model to follow this linear in log price trend.

1.3 Vector Autoregressive Models

Question 1.3.1

$\mathbf{Y} = \mathbf{BZ} + \mathbf{U}$

$\mathbf{Y} \in R^{K \times T}$

$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1 & \mathbf{y}_2 & \dots & \mathbf{y}_T \end{bmatrix}$

Matrix Y is made up of T columns of $\mathbf{y}_t$ representing the output of AR process at time t. Each $\mathbf{y}_t$ has K elements in the vector, thus Y has K rows.

$\mathbf{B} = \begin{bmatrix} \mathbf{c} & \mathbf{A}_1 & \mathbf{A}_2 & \dots & \mathbf{A}_p \end{bmatrix}$

$\mathbf{c}$ is a vector of K constants, invariant of time.

$$\mathbf{A}_i = \begin{bmatrix} a^i_{1,1} & a^i_{1,2} & \dots & a^i_{1,k} \\ a^i_{2,1} & a^i_{2,2} & \dots & a^i_{2,k} \\ \dots & \dots & \dots & \dots \\ a^i_{k,1} & a^i_{k,2} & \dots & a^i_{k,k} \end{bmatrix}$$

$i = 1, 2, \dots, k$

$\mathbf{A}_i$ represents the AR coefficients and is a $K \times K$ matrix.

It is clear that each element in B has K rows. $\mathbf{c}$ is a column vector of 1 column. Each $\mathbf{A}_i$ has K columns and there are P of them, so $\mathbf{B} \in R^{K \times (KP+1)}$.

For each column of Y, it can be represented as: $\mathbf{y}_t = \mathbf{c} + \mathbf{A}_1\mathbf{y}_{t-1} + \mathbf{A}_2\mathbf{y}_{t-2} + \dots + \mathbf{A}_p\mathbf{y}_{t-p} + \mathbf{e}_t$

$$\mathbf{y}_t = \mathbf{B}\begin{bmatrix} 1 \\ \mathbf{y}_{t-1} \\ \mathbf{y}_{t-2} \\ \dots \\ \mathbf{y}_{t-p} \end{bmatrix} + \mathbf{e}_t = \mathbf{B}\mathbf{z}_t + \mathbf{e}_t$$

z_t is the past process output vector stacking up under a scalar 1. So it has 1+KxP rows.

$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_T \end{bmatrix}$. Therefore $\mathbf{Z} \in R^{(KP+1)\times T}$.

$\mathbf{U} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \dots & \mathbf{e}_T \end{bmatrix}$

e_t is the process model error at time t and is a K-element vector. Thus $\mathbf{U} \in R^{K\times T}$.

Question 1.3.2

$\varepsilon = \mathbf{Y} - \mathbf{BZ}$

For $\mathbf{B}_{opt}$, error must be othorgonal to $\mathbf{Z}$: $\mathbf{Z}^T\varepsilon = \mathbf{0}$.

$\mathbf{Z}^T(\mathbf{Y} - \mathbf{BZ}) = \mathbf{0}$

$\mathbf{Z}^T\mathbf{Y} - \mathbf{Z}^T\mathbf{BZ} = \mathbf{0}$ or $\mathbf{YZ}^T = \mathbf{BZZ}^T$

Therefore $\mathbf{B}_{opt} = \mathbf{YZ}^T(\mathbf{ZZ}^T)^{-1}$

Question 1.3.3

$\mathbf{y}_t = \mathbf{A}\mathbf{y}_{t-1} + \mathbf{e}_t$

$\mathbf{y}_t = \mathbf{A}(\mathbf{A}\mathbf{y}_{t-2} + \mathbf{e}_{t-1}) + \mathbf{e}_t$

$\mathbf{y}_t = \mathbf{A}^2\mathbf{A}\mathbf{y}_{t-2} + \mathbf{A}\mathbf{e}_{t-1} + \mathbf{e}_t$

$||\mathbf{y}_t||_2 \approx ||\mathbf{A}^2||_2 \times ||\mathbf{y}_{t-2}||_2 + ||\varepsilon||_2$

The p2 norm, $||\mathbf{A}^2||_2 = \max_i^k(\lambda_i^2)$, where λ_i is the one of K eigenvalues of matrix $\mathbf{A}$ (A is a sqaure and symmetric matrix).

In order to keep $\mathbf{y}_t$ stable, at least $||\mathbf{A}^2||_2 \leq 1$. Therefore $|\lambda_i| \leq 1$ is necessary, for all eigenvalues of matrix A.

Question 1.3.4

In [27]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from statsmodels.tsa.api import VAR

df = pd.read_csv(r'snp_allstocks_2015_2019.csv')
df = df.set_index('Date')

info = pd.read_csv(r'snp_info.csv')
info.drop(columns=info.columns[0], inplace=True)
```

In [28]:

```
tickers = ['CAG','MAR','LIN','HCP','MAT']
stocks = df[tickers]
stocks_ma = stocks.rolling(window=66,min_periods=1).mean()
stocks_detrended = stocks.sub(stocks_ma).dropna()
```

In [29]:

```
model = VAR(stocks_detrended)
results = model.fit(1)
A = results.params[1:].values
eigA, _ = np.linalg.eig(A)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:219: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

In [30]:

```
print(stocks)
```

```
                  CAG         MAR         LIN        HCP        MAT
Date
2015-01-02  27.875486   77.750000         NaN  40.846996  30.469999
2015-01-05  27.571985   75.620003         NaN  41.256832  30.459999
2015-01-06  27.299610   74.669998         NaN  42.040073  29.809999
2015-01-07  27.945526   76.379997         NaN  42.932606  29.910000
2015-01-08  28.552528   78.910004         NaN  42.358833  29.980000
...               ...         ...         ...        ...        ...
2018-12-24  20.959999  100.989998  148.130005  26.770000   9.260000
2018-12-26  21.430000  105.580002  152.740005  27.940001   9.940000
2018-12-27  21.219999  106.629997  156.410004  27.809999   9.980000
2018-12-28  21.180000  107.239998  156.309998  27.879999   9.940000
2018-12-31  21.360001  108.559998  156.039993  27.930000   9.990000

[1006 rows x 5 columns]
```

In [31]:

```
print(A)
```

```
[[ 0.97022759  0.03311629 -0.08698923  0.00376341  0.01327549]
 [-0.00526079  0.91271878 -0.02948557 -0.01099732  0.00675337]
 [ 0.01326282 -0.0188172   0.82107702 -0.00267192 -0.01369525]
 [-0.08026766 -0.012754   -0.22984883  0.90870322 -0.02529306]
 [ 0.08521642  0.00978876  0.39225905  0.03673742  0.98377705]]
```

noice that all off diagonal values are approximately 0. This could suggest that each stock is quite independent from each other.

In [32]:

```
print(eigA)
```

```
[0.99825478+0.j         0.85124731+0.j         0.90337502+0.05525527j
 0.90337502-0.05525527j 0.94025153+0.j        ]
```

The largest eigenvalue belons to CAG. The smallest eigenvalue belongs to MAR. LIN and HCP has almost the same eigenvalue.

It could make sense to construct a portfolio using these stocks. VAR(1) model fits the 5 stocks quite well so the trend of them are quite easy to track. Long and short decisions could be made depending on whether the stock price has a increasing trend or decreasing trend.

Question 1.3.5

In [33]:

```
for sector in info['GICS Sector'].unique():
    tickers= info.loc[info['GICS Sector']==sector]['Symbol'].tolist()
    stocks = df[tickers]
```

In [34]:

```
stocks_ma = stocks.rolling(window=66,min_periods=1).mean()
stocks_detrended = stocks.sub(stocks_ma).dropna()
model = VAR(stocks_detrended)
results = model.fit(1)
A = results.params[1:].values
eigA, _ = np.linalg.eig(A)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:219: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

In [35]:

```
print(stocks)
```

```
                  APC        APA       BHGE        COG         CVX  \
Date
2015-01-02  82.290001  63.830002  56.169998  30.100000  112.580002
2015-01-05  75.800003  59.910000  55.160000  28.450001  108.080002
2015-01-06  75.239998  59.259998  54.990002  28.049999  108.029999
2015-01-07  76.440002  58.599998  56.470001  28.290001  107.940002
2015-01-08  78.910004  60.869999  57.029999  28.860001  110.410004
...               ...        ...        ...        ...         ...
2018-12-24  40.570000  25.400000  20.480000  22.160000  100.989998
2018-12-26  43.849998  27.040001  21.520000  23.590000  107.389999
2018-12-27  43.970001  26.740000  21.480000  23.770000  109.320000
2018-12-28  43.200001  26.340000  21.440001  22.950001  108.650002
2018-12-31  43.840000  26.250000  21.500000  22.350000  108.790001

                   XEC         CXO        COP        DVN       FANG  ...  \
Date                                                                 ...
2015-01-02  106.879997  101.010002  68.919998  60.959999  59.930000  ...
2015-01-05  100.690002   96.110001  65.639999  58.700001  57.279999  ...
2015-01-06   98.919998   94.180000  62.930000  57.500000  57.369999  ...
2015-01-07   98.080002   91.989998  63.349998  57.770000  58.250000  ...
2015-01-08  100.529999   96.400002  64.930000  59.820000  61.090000  ...
...                ...         ...        ...        ...        ...  ...
2018-12-24   57.320000   94.900002  57.009998  20.980000  85.669998  ...
2018-12-26   61.990002  102.570000  61.500000  22.709999  91.989998  ...
2018-12-27   61.980000  102.849998  62.230000  22.900000  92.879997  ...
2018-12-28   61.250000  101.809998  61.669998  22.459999  91.889999  ...
2018-12-31   61.650002  102.790001  62.349998  22.540001  92.699997  ...

                  NFX        NBL        OXY        OKE        PSX         PX
D  \
Date
2015-01-02  26.600000  46.880001  80.513130  49.919998  72.290001  149.88999
9
2015-01-05  24.900000  42.389999  77.528198  47.060001  67.919998  139.83000
2
2015-01-06  24.600000  41.900002  76.869324  45.099998  66.019997  139.46000
7
2015-01-07  23.610001  41.740002  76.879303  44.770000  66.309998  137.92999
3
2015-01-08  23.980000  43.180000  77.558151  45.259998  68.889999  141.41000
4
...               ...        ...        ...        ...        ...
...
2018-12-24  12.920000  17.360001  57.110001  50.790001  78.500000  120.51000
2
2018-12-26  14.540000  19.080000  60.599998  53.700001  84.199997  130.02999
9
2018-12-27  14.500000  18.900000  60.759998  53.320000  85.820000  130.08000
2
2018-12-28  14.330000  18.430000  60.470001  52.980000  84.959999  130.50999
5
2018-12-31  14.660000  18.760000  61.380001  53.950001  86.150002  131.52000
4

                  SLB        FTI        VLO        WMB
Date
2015-01-02  85.669998  46.520000  50.340000  45.220001
```

```
2015-01-05  83.349998  44.700001  48.270000  43.430000
2015-01-06  81.720001  43.770000  47.680000  42.840000
2015-01-07  81.709999  42.279999  47.310001  42.610001
2015-01-08  82.699997  43.430000  50.020000  43.020000
...               ...        ...        ...        ...
2018-12-24  35.189999  18.540001  68.940002  20.580000
2018-12-26  36.610001  19.639999  73.309998  21.670000
2018-12-27  36.330002  19.590000  74.629997  21.459999
2018-12-28  36.599998  19.670000  73.480003  21.480000
2018-12-31  36.080002  19.580000  74.970001  22.049999

[1006 rows x 30 columns]
```

In [36]:

```
print(A)
```

```
[[ 9.05539244e-01  1.10516148e-02  1.27430556e-03 -1.20511181e-02
  -7.87534968e-02 -4.76046271e-02 -9.78439284e-03 -2.12329623e-02
  -1.23444670e-02 -5.95136687e-03 -1.97885838e-02 -5.04180573e-02
  -3.84435523e-03  1.23797862e-02 -2.08999088e-02 -8.07039464e-03
  -7.16407597e-03 -1.51281105e-02 -3.42644852e-02 -2.22057455e-03
  -1.61109880e-02 -8.09110683e-03 -2.78022753e-02 -3.40108735e-02
  -3.64777023e-02 -5.51594729e-02 -6.68970760e-03  2.36798450e-03
  -4.66258902e-02 -7.61670677e-03]
 [ 4.41804917e-02  9.41548897e-01 -7.25709471e-03 -5.65322981e-03
   4.28108365e-02  2.55284087e-02  3.52205349e-02  2.64033904e-02
   9.88150785e-04 -8.34098749e-03  4.24742512e-02  2.77731855e-02
   8.76634514e-03  3.71085729e-02  3.71433971e-02 -1.27906222e-02
   6.37802406e-03  1.78884473e-03 -5.14742628e-03  2.24218282e-03
   3.78791315e-03  2.86166457e-02  7.99215006e-03  3.83015409e-02
   4.40077584e-03  1.09899292e-03  1.21167323e-02  1.25298039e-02
  -1.38854216e-02  1.56868654e-02]
 [ 1.49450021e-02  1.40307914e-02  9.52556652e-01  2.21492984e-03
   9.05812103e-03  7.00209368e-03 -1.54547513e-02  9.62977380e-03
   1.86737606e-03 -2.72062548e-02  2.35660733e-02  2.14306444e-02
```

In [37]:

```
print(eigA)
```

```
[0.85128312+0.06747729j 0.85128312-0.06747729j 0.83991778+0.j
 0.86996137+0.05056108j 0.86996137-0.05056108j 0.86937011+0.j
 0.89197014+0.02678096j 0.89197014-0.02678096j 0.90282092+0.04039893j
 0.90282092-0.04039893j 0.91777393+0.04897028j 0.91777393-0.04897028j
 0.91735332+0.j         0.97264723+0.0464787j  0.97264723-0.0464787j
 0.95433701+0.04237971j 0.95433701-0.04237971j 0.93785911+0.j
 0.96252182+0.03027034j 0.96252182-0.03027034j 0.97758519+0.03130794j
 0.97758519-0.03130794j 0.9792577 +0.02342929j 0.9792577 -0.02342929j
 0.98597666+0.j         0.97756501+0.01113046j 0.97756501-0.01113046j
 0.95776805+0.00994676j 0.95776805-0.00994676j 0.97380626+0.j        ]
```

It is generally not advisable to build a portfolio with stocks all in one sector. Stocks in one sector has generally similar trend in performance, thus if the whole sector is not doing well, the portfolio is hard to do well. Diversity of stocks could help the portfolio construction to be healthy. However, if there is big confidence that one sector would do well, then it makes sense to invest in that sector.