In [374]:

```
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
```

Problem 1

First we generate paths of 10000 stocks over 1000 periods

In [375]:

```
# define num of periods and num of stocks
periods = 1000
num_stocks = 10000

# a function to generate one stock path
def gen_one_path():
    p = [1]
    for i in range(1,periods):
        p.append((1+0.007*np.random.normal(0,1))*p[i-1])
    return np.array(p)
```

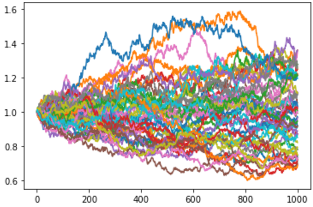
In [376]:

```
# store all paths into a list
paths = []
for i in range(num_stocks):
    paths.append(gen_one_path())
```

Here we can plot 50 sample paths:

In [377]:

```
for i in range(50):
    plt.plot(paths[i])
```



From the previous figure, generally our stock paths follow the random walk as specified in the homework instruction.

Then we apply our selecting rule to generate Russell 1000 index

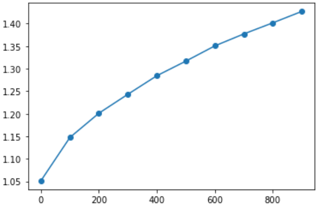
In [378]:

```
def update_index(start,end):
    """return the index from start time to end time

    start: the update time
    end: the next update time
    """
    index = np.repeat(0.0,end-start)
    path = []
    for i in paths:
        path.append(i[start])
    tops = np.argsort(path)[-1000:]
    for i in tops:
        index+=paths[i][start:end]
    return index/1000
```

In [379]:
```
# select stocks into Russell 1000 index
update_time = np.arange(1,1000,50)
update_time = np.append(update_time,1000)
index = np.array([])
index_mean = np.array([])
for i in range(1,len(update_time)):
    new_index = update_index(update_time[i-1],update_time[i])
    index=np.append(index, new_index)
index = np.append(1,index)

windows = np.arange(0,1001,100)
for i in range(1,len(windows)):
    mean = index[windows[i-1]:windows[i]].mean()
    index_mean = np.append(index_mean, mean)
```

Q1: Plot simulated Russell 1000 Index

In [380]:
```
plt.scatter(np.arange(0,1000,100),index_mean)
plt.plot(np.arange(0,1000,100),index_mean)
plt.show()
```



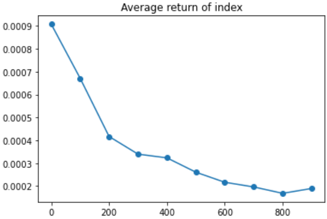
It seems reasonable because we select 1000 stocks with laregest market cap every 50 periods. In each window(every 100 periods), there's one re-seleciton, so the average Russell 1000 index keeps going up.

After we have the path of Russell 1000 index, we can claculate the average return of index in each window

In [381]:
```
windows[-1]=999
re1 = np.array([])
for i in range(1,len(windows)):
    returns = (index[windows[i]] - index[windows[i-1]])/index[windows[i-1]]
    re1 = np.append(re1,returns/100)
```

Q2: Plot the average return of Russell 1000 Index

In [382]:
```
plt.title("Average return of index")
plt.scatter(np.arange(0,1000,100),re1)
plt.plot(np.arange(0,1000,100),re1)
plt.show()
```



From the figure we find that the average return of index in each windows keeps going down, it's because the number of stocks performing better than the Russell 1000 index becomes smaller as selecting the stocks into the index again and again.

Then we calculate the average return of 10000 stocks in each window

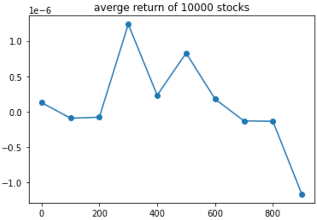
In [383]:

```
re2 = np.array([])
for i in range(1,len(windows)):
    returns = 0
    for j in paths:
        returns += (j[i]-j[i-1])/j[i]
    re2 = np.append(re2,returns/100/len(paths))
```

Q3: Plot the average return of the 10,000 stocks

In [384]:

```
plt.title("averge return of 10000 stocks")
plt.scatter(np.arange(0,1000,100),re2)
plt.plot(np.arange(0,1000,100),re2)
plt.show()
```



From the figure we find that the average return of 1000 index fluctuates around 0, it's proper because our 10000 stocks' paths are random walks.

Q4:

- Obviously, even from the shape of figures of **Q2** and **Q3** we can conclude that Russell 1000 index cannot represent the performance of the whole market.

- The average return of Russell 1000 index is larger than the average return of 10000 stocks. Because in every window, we will always re-select the stocks that have the largest market cap, which drives the return of Russell 1000 index dramatically up.

Problem 2

Download data (Ajusted price of 11 ETFs and SPY)

In [409]:

```
import yfinance as yf
symbols = ['VOX','VCR', 'VDC', 'VDE', 'VFH', 'VHT', 'VIS', 'VGT', 'VAW', 'VNQ', 'VPU', 'SPY']
secs = ['COMM', 'CONSUMER DISC', 'CONSUMER ST', 'ENERGY', 'FINANCIALS', 'HEALTH', 'INDUSTRIALS',
        'TECHNOLOGY', 'MATIREALS', 'REAL ESTATE', 'UTILITIES']
start = '2010-1-1'
end = '2020-12-31'
data = yf.download(symbols,start,end)['Adj Close']
data.head()
```

```
[*********************100%***********************]  12 of 12 completed
```

Out[409]:

	SPY	VAW	VCR	VDC	VDE	VFH	VGT	VHT	VIS	VNQ	VOX	VP
Date												
2010-01-04	90.205002	56.640873	40.879322	49.588455	63.985836	23.370375	49.731972	46.846146	43.845589	28.118284	43.386730	44.06461
2010-01-05	90.443810	56.981335	41.139488	49.633282	64.640831	23.631197	49.678482	46.556236	44.037510	28.086729	43.679874	43.65404

2010-01-06	90.507477	57.783897	41.182846	49.625816	65.385086	23.670712	49.330898	46.820560	44.146000	28.036236	42.988350	43.84922
2010-01-07	90.889534	57.532593	41.538399	49.625816	65.228798	24.129108	49.143738	46.982574	44.688446	28.339193	42.830479	43.65404
2010-01-08	91.192009	58.156784	41.573078	49.356804	65.720024	24.034260	49.526974	47.153107	45.280952	28.130911	42.635052	43.60693

Calculate log return

In [412]:

```
returns = data.pct_change()
returns = returns.dropna()
returns.head()
```

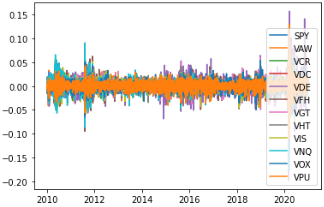
Out[412]:

	SPY	VAW	VCR	VDC	VDE	VFH	VGT	VHT	VIS	VNQ	VOX	VPU
Date												
2010-01-05	0.002647	0.006011	0.006364	0.000904	0.010237	0.011160	-0.001076	-0.006189	0.004377	-0.001122	0.006757	-0.009317
2010-01-06	0.000704	0.014085	0.001054	-0.000150	0.011514	0.001672	-0.006997	0.005678	0.002464	-0.001798	-0.015832	0.004471
2010-01-07	0.004221	-0.004349	0.008634	0.000000	-0.002390	0.019366	-0.003794	0.003460	0.012288	0.010806	-0.003672	-0.004451
2010-01-08	0.003328	0.010849	0.000835	-0.005421	0.007531	-0.003931	0.007798	0.003630	0.013259	-0.007350	-0.004563	-0.001079
2010-01-11	0.001396	-0.003345	-0.001251	0.003482	-0.000793	-0.000329	-0.004319	0.005063	0.009399	0.005833	0.000882	0.007717

Q1: Plot the return processes of the 11 selected sector ETFs and the S&P 500 index.

In [413]:

```
for col in returns.columns:
    plt.plot(returns[col], label = col)
plt.legend()
plt.show()
```



Then we create a dataframe consisting of the return of 11 etfs so as to do factor analysis

In [414]:

```
df = returns.loc[:,['VOX','VCR', 'VDC', 'VDE', 'VFH', 'VHT', 'VIS', 'VGT', 'VAW', 'VNQ', 'VPU']]
df.head()
```

Out[414]:

	VOX	VCR	VDC	VDE	VFH	VHT	VIS	VGT	VAW	VNQ	VPU
Date											
2010-01-05	0.006757	0.006364	0.000904	0.010237	0.011160	-0.006189	0.004377	-0.001076	0.006011	-0.001122	-0.009317
2010-01-06	-0.015832	0.001054	-0.000150	0.011514	0.001672	0.005678	0.002464	-0.006997	0.014085	-0.001798	0.004471
2010-01-07	-0.003672	0.008634	0.000000	-0.002390	0.019366	0.003460	0.012288	-0.003794	-0.004349	0.010806	-0.004451
2010-01-08	-0.004563	0.000835	-0.005421	0.007531	-0.003931	0.003630	0.013259	0.007798	0.010849	-0.007350	-0.001079
2010-01-11	0.000882	-0.001251	0.003482	-0.000793	-0.000329	0.005063	0.009399	-0.004319	-0.003345	0.005833	0.007717

In [415]:

```
from factor_analyzer import FactorAnalyzer
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity
from factor_analyzer.factor_analyzer import calculate_kmo
```

Bartlett’s test of sphericity checks whether or not the observed variables intercorrelate at all

```
In [416]: chi_square_value,p_value=calculate_bartlett_sphericity(df)
          chi_square_value,p_value
```

```
Out[416]: (37073.57984030266, 0.0)
```

The p-value is 0. The test was statistically significant, indicating that the observed correlation matrix is not an identity matrix.

Kaiser-Meyer-Olkin (KMO) Test measures the suitability of data for factor analysis.

```
In [417]: kmo_all,kmo_model=calculate_kmo(df)
          kmo_model
```

```
/opt/anaconda3/lib/python3.8/site-packages/factor_analyzer/utils.py:248: UserWarning: The inverse of
the variance-covariance matrix was calculated using the Moore-Penrose generalized matrix inversion, d
ue to its determinant being at or very close to zero.
  warnings.warn('The inverse of the variance-covariance matrix '
```

```
Out[417]: 0.9471425864905267
```

The overall KMO for our data is 0.94, which is excellent.

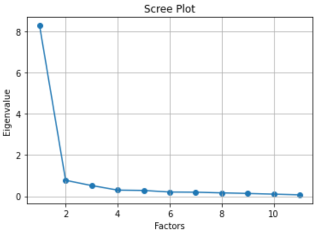
Create factor analysis object and perform factor analysis

```
In [418]: fa = FactorAnalyzer(n_factors=25, rotation=None)
          fa.fit(df)
          ev, v = fa.get_eigenvalues()
          ev # eigenvalues
```

```
Out[418]: array([8.28271548, 0.77087021, 0.51717587, 0.29609546, 0.27969889,
                 0.2016662 , 0.19410051, 0.16009417, 0.13257676, 0.09917465,
                 0.0658318 ])
```

Scree plot

```
In [419]: # Create scree plot using matplotlib
          plt.scatter(range(1,df.shape[1]+1),ev)
          plt.plot(range(1,df.shape[1]+1),ev)
          plt.title('Scree Plot')
          plt.xlabel('Factors')
          plt.ylabel('Eigenvalue')
          plt.grid()
          plt.show()
```



Q2: use the varimax method to find a final rotated factor solution, which is the loadings

- From the scree plot, we may choose 2 or 3 or 4 factors in our model, we first choose a 4 factor model

In [420]:

```
fa = FactorAnalyzer(n_factors=4, rotation="varimax")
fa.fit(df)
loadings = pd.DataFrame(fa.loadings_)
loadings.index = df.columns
loadings.columns = ['factor1','factor2','factor3','factor4']

# highlight the factors which are larger than .5
def style_highlight(v, props=''):
    return props if v > 0.5 else None
loadings_hightlight = loadings.style.applymap(style_highlight, props='color:red;')
loadings_hightlight
```

Out[420]:

	factor1	factor2	factor3	factor4
VOX	0.410466	0.609959	0.376963	0.193972
VCR	0.489970	0.728050	0.273568	0.264277
VDC	0.321250	0.472676	0.719524	0.117477
VDE	0.734203	0.312238	0.249344	0.155897
VFH	0.674355	0.455130	0.332321	0.281255
VHT	0.411484	0.623389	0.426334	0.145048
VIS	0.703501	0.523452	0.340472	0.219489
VGT	0.413711	0.772021	0.283756	0.168444
VAW	0.728799	0.470427	0.319832	0.191336
VNQ	0.365393	0.307730	0.527396	0.699770
VPU	0.256725	0.217474	0.743435	0.273102

We can find that factor4 only have one high factor loading, so we consider a model that has only 3 factors

In [421]:

```
fa = FactorAnalyzer(n_factors=3, rotation="varimax")
fa.fit(df)
loadings = pd.DataFrame(fa.loadings_)
loadings.index = df.columns
loadings.columns = ['factor1','factor2','factor3']

# highlight the factors which are larger than .5
def style_highlight(v, props=''):
    return props if v > 0.5 else None
loadings_hightlight = loadings.style.applymap(style_highlight, props='color:red;')
loadings_hightlight
```

Out[421]:

	factor1	factor2	factor3
VOX	0.429673	0.630152	0.376165
VCR	0.534355	0.720488	0.306040
VDC	0.335488	0.523337	0.622755
VDE	0.732237	0.320621	0.263133
VFH	0.712490	0.459573	0.370938
VHT	0.419507	0.652411	0.396982
VIS	0.726879	0.530229	0.353638
VGT	0.435826	0.777554	0.277961
VAW	0.741745	0.479135	0.327709
VNQ	0.463355	0.377836	0.612159
VPU	0.239578	0.222257	0.890986

Now we look at the model with 2 factors

In [422]:

```
fa = FactorAnalyzer(n_factors=2, rotation="varimax")
fa.fit(df)
loadings = pd.DataFrame(fa.loadings_)
loadings.index = df.columns
loadings.columns = ['factor1','factor2']

# highlight the factors which are larger than .5
def style_highlight(v, props=''):
    return props if v > 0.5 else None
loadings_hightlight = loadings.style.applymap(style_highlight, props='color:red;')
loadings_hightlight
```

Out[422]:

	factor1	factor2
VOX	0.716570	0.431418
VCR	0.857187	0.372094
VDC	0.564539	0.661346
VDE	0.728019	0.297289
VFH	0.813571	0.406731
VHT	0.722191	0.453260
VIS	0.875133	0.395684
VGT	0.809353	0.359870
VAW	0.848784	0.368388
VNQ	0.562749	0.638738
VPU	0.271676	0.908936

Now every factor has 3 or more high factor loadings, so I decide to choose a 2 factor model in this problem

Regression and prediction

The model is

$$r_t = \beta f_t + \epsilon_t$$

return of SPY

In [423]:

```
spy = data['SPY']
spy_return = spy.pct_change()
spy_return = spy_return.dropna()
```

Transformation

In [424]:

```
new_v = fa.transform(df)
new_v = pd.DataFrame(new_v)
new_v.index = spy_return.index
new_v.columns = ['factor1', 'factor2']
```

In [425]:

```
new_data = new_v.join(spy_return)
new_data.columns = ['spy', 'factor1', 'factor2']
new_data
```

Out[425]:

	spy	factor1	factor2
Date			
2010-01-05	0.821316	-0.979308	0.002647
2010-01-06	0.032371	0.163148	0.000704
2010-01-07	0.724313	-0.448892	0.004221
2010-01-08	0.711067	-0.638953	0.003328
2010-01-11	-0.158254	0.651585	0.001396
...	...	...	...
2020-12-23	0.452836	-0.274390	0.000899
2020-12-24	-0.106490	0.558913	0.003890
2020-12-28	-0.082484	0.624228	0.008591

2020-12-29	-0.354990	-0.159572	-0.001908
2020-12-30	0.359135	0.249017	0.001427

2767 rows × 3 columns

In [431]:

```
from statsmodels.api import OLS, add_constant
```

In [432]:

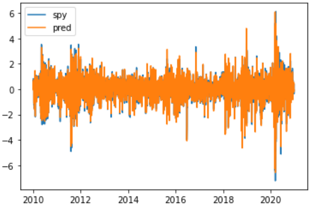
```
factors = OLS(new_data.spy, add_constant(new_data[['factor1', 'factor2']])).fit()
factors = factors.params
```

In [433]:

```
new_data['pred'] = new_data.factor1 * factors.factor1 + new_data.factor2*factors.factor2 + factors.const
```

In [434]:

```
plt.plot(new_data.spy,label='spy')
plt.plot(new_data.pred,label='pred')
plt.legend()
plt.show()
```



In [435]:

```
new_data[['spy','pred']]
```

Out[435]:

Date	**spy**	**pred**
2010-01-05	0.821316	0.719573
2010-01-06	0.032371	-0.071193
2010-01-07	0.724313	0.602356
2010-01-08	0.711067	0.611114
2010-01-11	-0.158254	-0.255521
...	...	...
2020-12-23	0.452836	0.176182
2020-12-24	-0.106490	0.044425
2020-12-28	-0.082484	0.484840
2020-12-29	-0.354990	-0.166827
2020-12-30	0.359135	-0.042919

2767 rows × 2 columns

From the figure we can find that our factor model can mimic the return process of S&P 500 index

Q3: Apply the factor sorting method

We now choose the facotr1 which is the most important factor and then we sort them:

In [436]:

```
factor_sort = loadings.loc[:,'factor1']
factor_sort.index = secs
```

In [437]:

```
factor_sort = factor_sort.sort_values()
factor_sort
```

Out[437]:

```
UTILITIES          0.271676
REAL ESTATE        0.562749
```

```
CONSUMER ST       0.564539
COMM              0.716570
HEALTH            0.722191
ENERGY            0.728019
TECHNOLOGY        0.809353
FINANCIALS        0.813571
MATIREALS         0.848784
CONSUMER DISC     0.857187
INDUSTRIALS       0.875133
Name: factor1, dtype: float64
```

From the table above, we short the **UTILITIES, CONSUMER ST, REAL ESTATE** sectors and long **INDUSTRIALS, MATIREALS, CONSUMER DISC** sectors