In [1]:
```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import minimize_scalar
from functools import partial
```

In [2]:
```
spy_index = pd.read_excel("spy500.xlsx")
```

In [3]:
```
spy_list = [string.replace('.', '-') for string in list(spy_index['Symbol'])]
```

In [4]:
```
stock_data = yf.download('SPY', start='2008-01-01', end='2023-11-30')
monthly_spy = stock_data.resample('M').last()
```

```
[*********************100%***********************]  1 of 1 completed
```

In [5]:
```
m_price = pd.DataFrame(index = monthly_spy.index,columns = spy_list)
for stock_code in m_price.columns:
    stock_data = yf.download(stock_code, start='2008-01-01', end='2023-11-30')
    monthly_data = stock_data.resample('M').last()
    m_price[stock_code] = monthly_data['Adj Close']
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

In [6]:
```
m_price = m_price.iloc[:-11]
```

In [36]:
```
m_price.to_excel("All SP500.xlsx")
```

In [7]:
```
monthly_spy = monthly_spy.iloc[:-11]
```

In [8]:
```
m_price
```

Out[8]:

Date	MMM	AOS	ABT	ABBV	ACN	ADM	ADBE	ADP	A
2008-01-31	48.898937	4.546392	18.715803	NaN	25.789371	29.657907	34.930000	23.896479	13.6233
2008-02-29	48.434818	4.733497	17.839647	NaN	26.258673	30.503391	33.650002	23.525488	12.8312
2008-03-31	48.898144	4.270931	18.372671	NaN	26.199074	27.838570	35.590000	25.144592	11.8963
2008-04-30	47.508129	4.043722	17.691273	NaN	27.972004	29.799994	37.290001	26.218239	12.3888
2008-05-31	48.226898	4.724647	18.898642	NaN	30.407913	26.931993	44.060001	25.536093	13.9017
...	...	...	...	...	...	...	...	...	...
2022-08-31	115.903831	55.108898	100.164154	127.937202	282.691589	85.537041	373.440002	236.802841	24.4962
2022-09-30	102.994560	47.425869	94.416801	127.699333	252.154648	78.296211	275.200012	220.096909	21.7530
2022-10-31	117.246025	53.789490	96.999535	140.715378	279.460022	94.383690	318.500000	235.189087	25.3308
2022-11-30	118.788635	59.641720	105.470078	154.921448	296.223663	95.294067	344.929993	257.024628	28.0033
2022-12-31	113.083527	56.205013	107.636734	155.334778	262.666779	90.749268	336.529999	233.551819	27.8484

180 rows × 503 columns

In [9]:
```
def ER(p,start_index,end_index):
    port_return = 1
    ew_return = 1
    if end_index == len(m_price):
        end_index -= 1
    for i in range(start_index,end_index):
        price_series = m_price.iloc[i].dropna()   # Price for existing companies
        port_size = len(price_series)
        miu_series = price_series/sum(price_series)
        miu_p_series = miu_series**p
        port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
        ew_w = np.full(port_size,1/port_size)   # weights for EW
        now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].i
        port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for p
        ew_return_now = np.dot(np.array(ew_w),np.array(now_return).T) # return for EW
        port_return *= (1 + port_return_now)
        ew_return *= (1 + ew_return_now)
    return (port_return - ew_return)
```

In [10]:
```
def DWP_cal(p,start_index,end_index):
    port_return = 1
    if end_index == len(m_price):
        end_index -= 1
    for i in range(start_index,end_index):
        price_series = m_price.iloc[i].dropna()   # Price for existing companies
        port_size = len(price_series)
        miu_series = price_series/sum(price_series)
        miu_p_series = miu_series**p
        port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
        now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].i
        port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for p
        port_return *= (1 + port_return_now)
    return port_return
```

In [11]:
```
def EW_cal(start_index,end_index):
    ew_return = 1
    if end_index == len(m_price):
        end_index -= 1
    for i in range(start_index,end_index):
        price_series = m_price.iloc[i].dropna()   # Price for existing companies
        port_size = len(price_series)
        ew_w = np.full(port_size,1/port_size)   # weights for EW
        now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].i
        ew_return_now = np.dot(np.array(ew_w),np.array(now_return).T) # return for EW
        ew_return *= (1 + ew_return_now)
    return ew_return
```

In [12]:
```
def DWP_sp(p,start_index,end_index):
    return_list = []
    if end_index == len(m_price):
        end_index -= 1
    for i in range(start_index,end_index):
        price_series = m_price.iloc[i].dropna()   # Price for existing companies
        port_size = len(price_series)
        miu_series = price_series/sum(price_series)
        miu_p_series = miu_series**p
        port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
        now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].i
        port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for p
        return_list.append(port_return_now)
    return np.sqrt(12)*np.mean(return_list)/np.std(return_list)
```

In [13]:
```python
def EW_sp(start_index,end_index):
    return_list = []
    if end_index == len(m_price):
        end_index -= 1
    for i in range(start_index,end_index):
        price_series = m_price.iloc[i].dropna()   # Price for existing companies
        port_size = len(price_series)
        ew_w = np.full(port_size,1/port_size)   # weights for EW
        now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].i
        ew_return_now = np.dot(np.array(ew_w),np.array(now_return).T) # return for EW
        return_list.append(ew_return_now)
    return np.sqrt(12)*np.mean(return_list)/np.std(return_list)
```

In [14]:
```python
def target_pdf(x):
    return np.exp(-2 * (x-7)**2) / (np.sqrt(2 * np.pi)*0.5)
def positive_mh(start_index,end_index,times):
    samples = [0.1]
    total_num = 0
    while total_num < times:
        current_sample = samples[-1]
        proposal = np.random.normal(current_sample, 0.5)
        if 0 < proposal <= 10:
            now_er = ER(proposal,start_index,end_index)  # new ER
            past_er = ER(current_sample,start_index,end_index) # last ER
            acceptance_prob = min(1, target_pdf(now_er) / target_pdf(past_er))
        else:
            acceptance_prob = 0
        if np.random.rand() < acceptance_prob:
            samples.append(proposal)
        else:
            samples.append(current_sample)
        total_num += 1
    use = int(times/2)
    return sum(samples[use:])/len(samples[use:])
```

In [15]:
```python
def negative_mh(start_index,end_index,times):
    samples = [-0.1]
    total_num = 0
    while total_num < times:
        current_sample = samples[-1]
        proposal = np.random.normal(current_sample, 0.5)
        if -10 <= proposal < 0:
            now_er = ER(proposal,start_index,end_index)  # new ER
            past_er = ER(current_sample,start_index,end_index) # last ER
            acceptance_prob = min(1, target_pdf(now_er) / target_pdf(past_er))
        else:
            acceptance_prob = 0
        if np.random.rand() < acceptance_prob:
            samples.append(proposal)
        else:
            samples.append(current_sample)
        total_num += 1
    use = int(times/2)
    return sum(samples[use:])/len(samples[use:])
```

In [165]:
```python
ewp_result = pd.DataFrame(index = ['2008-2013-2017','2009-2014-2018','2010-2015-2019','2
                          columns = ['In sample return','Out of sample return','In sample
```

In [172]:
```python
for i in range(6):
    ewp_result['In sample return'].iloc[i] = (EW_cal(i*12,i*12+60)**(1/5))-1
    ewp_result['Out of sample return'].iloc[i] = (EW_cal(i*12+60,i*12+120)**(1/5))-1
    ewp_result['In sample Sharpe ratio'].iloc[i] = EW_sp(i*12,i*12+60)
    ewp_result['Out of sample Sharpe ratio'].iloc[i] = EW_sp(i*12+60,i*12+120)
```

In [173]:
```python
ewp_result
```

Out[173]:

	In sample return	Out of sample return	In sample Sharpe ratio	Out of sample Sharpe ratio
2008-2013-2017	0.128880	0.206449	0.668387	1.984674
2009-2014-2018	0.294831	0.147824	1.585950	1.236313
2010-2015-2019	0.226901	0.154104	1.560587	1.240552
2011-2016-2020	0.155977	0.197047	1.202287	1.158742
2022-2017-2021	0.195389	0.192977	1.767384	1.112454
2013-2018-2022	0.206449	0.121392	1.984674	0.683283

In [175]:
```python
dwp_positive_result = pd.DataFrame(index = ['2008-2013-2017','2009-2014-2018','2010-2015
                          columns = ['In sample return','Out of sample return','In sample
```

In [180]:
```python
for i in range(6):
    f_fixed = partial(ER, start_index = i*12, end_index = i*12+60)
    result = minimize_scalar(lambda x: -f_fixed(x), bounds=(0,10), method='bounded')
    p_op = result.x  # The optimal p calculated
    dwp_positive_result['p'].iloc[i] = p_op
    dwp_positive_result['In sample return'].iloc[i] = (DWP_cal(p_op,i*12,i*12+60)**(1/5)
    dwp_positive_result['Out of sample return'].iloc[i] = (DWP_cal(p_op,i*12+60,i*12+120
    dwp_positive_result['In sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12,i*12+60)
    dwp_positive_result['Out of sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12+60,i*12
```

In [181]:
```python
dwp_positive_result
```

Out[181]:

	In sample return	Out of sample return	In sample Sharpe ratio	Out of sample Sharpe ratio	p
2008-2013-2017	0.034941	0.199054	0.263008	1.094159	9.999993
2009-2014-2018	0.245718	0.117571	1.287796	0.875638	2.059512
2010-2015-2019	0.226901	0.154104	1.560586	1.240551	0.000004
2011-2016-2020	0.144734	0.172419	1.0207	0.929848	2.167683
2022-2017-2021	0.178457	0.185201	1.141053	0.834859	3.1102
2013-2018-2022	0.212113	0.062519	1.295071	0.374085	3.40947

In [182]:
```
dwp_negative_result = pd.DataFrame(index = ['2008-2013-2017','2009-2014-2018','2010-2015
                                 columns = ['In sample return','Out of sample return','In sample
```

In [183]:
```
for i in range(6):
    f_fixed = partial(ER, start_index = i*12, end_index = i*12+60)
    result = minimize_scalar(lambda x: -f_fixed(x), bounds=(-10,0), method='bounded')
    p_op = result.x  # The optimal p calculated
    dwp_negative_result['p'].iloc[i] = p_op
    dwp_negative_result['In sample return'].iloc[i] = (DWP_cal(p_op,i*12,i*12+60)**(1/5)
    dwp_negative_result['Out of sample return'].iloc[i] = (DWP_cal(p_op,i*12+60,i*12+120
    dwp_negative_result['In sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12,i*12+60)
    dwp_negative_result['Out of sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12+60,i*12
```

In [184]:
```
dwp_negative_result
```

Out[184]:

	In sample return	Out of sample return	In sample Sharpe ratio	Out of sample Sharpe ratio	p
2008-2013-2017	0.709799	0.324198	1.260885	0.676629	-3.994956
2009-2014-2018	0.853127	0.415565	1.777244	0.753048	-5.790833
2010-2015-2019	0.425495	0.565043	1.288168	0.880516	-4.726369
2011-2016-2020	0.412751	0.882758	0.809628	1.093726	-9.999995
2022-2017-2021	0.358023	0.637785	1.044382	1.156099	-2.32685
2013-2018-2022	0.465132	0.394466	1.000173	1.129521	-2.169052

In [185]:
```
dwp_mcmc_positive_result = pd.DataFrame(index = ['2008-2013-2017','2009-2014-2018','2010
                                 columns = ['In sample return','Out of sample return','In sample
```

In [186]:
```
for i in range(6):
    p_op = positive_mh(i*12,i*12+60,10000)  # The optimal p calculated
    dwp_mcmc_positive_result['p'].iloc[i] = p_op
    dwp_mcmc_positive_result['In sample return'].iloc[i] = (DWP_cal(p_op,i*12,i*12+60)**
    dwp_mcmc_positive_result['Out of sample return'].iloc[i] = (DWP_cal(p_op,i*12+60,i*1
    dwp_mcmc_positive_result['In sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12,i*12+6
    dwp_mcmc_positive_result['Out of sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12+60
```

In [190]:
```
dwp_mcmc_positive_result
```

Out[190]:

	In sample return	Out of sample return	In sample Sharpe ratio	Out of sample Sharpe ratio	p
2008-2013-2017	0.124145	0.204595	0.651217	1.97552	0.066688
2009-2014-2018	0.292322	0.146929	1.580352	1.231204	0.036686
2010-2015-2019	0.223552	0.15213	1.548176	1.233339	0.112518
2011-2016-2020	0.145077	0.185554	1.157055	1.170171	0.680614
2022-2017-2021	0.177104	0.180825	1.65503	1.098746	0.719661
2013-2018-2022	0.211047	0.063985	1.243715	0.376436	4.116748

In [188]:
```
dwp_mcmc_negative_result = pd.DataFrame(index = ['2008-2013-2017','2009-2014-2018','2010
                                        columns = ['In sample return','Out of sample return','In sample
```

In [191]:
```
for i in range(6):
    p_op = negative_mh(i*12,i*12+60,10000)  # The optimal p calculated
    dwp_mcmc_negative_result['p'].iloc[i] = p_op
    dwp_mcmc_negative_result['In sample return'].iloc[i] = (DWP_cal(p_op,i*12,i*12+60)**
    dwp_mcmc_negative_result['Out of sample return'].iloc[i] = (DWP_cal(p_op,i*12+60,i*1
    dwp_mcmc_negative_result['In sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12,i*12+6
    dwp_mcmc_negative_result['Out of sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12+60
```

In [192]:
```
dwp_mcmc_negative_result
```

Out[192]:

	In sample return	Out of sample return	In sample Sharpe ratio	Out of sample Sharpe ratio	p
2008-2013-2017	0.545999	0.451453	1.378603	0.888869	-2.47227
2009-2014-2018	0.603569	0.479669	2.001137	0.989865	-2.201698
2010-2015-2019	0.425108	0.567071	1.250468	0.879548	-5.043002
2011-2016-2020	0.409159	0.877568	0.80913	1.089986	-9.133334
2022-2017-2021	0.357996	0.642508	1.035215	1.154934	-2.345971
2013-2018-2022	0.465128	0.395604	0.99814	1.130416	-2.173547

In [193]:
```
market_result = pd.DataFrame(index = ['2008-2013-2017','2009-2014-2018','2010-2015-2019'
                             columns = ['In sample return','Out of sample return','In sample
```

```
In [196]: for i in range(6):
              in_return = []
              out_return = []
              for j in range(i*12,i*12+60):
                  in_return.append(monthly_spy['Adj Close'].iloc[j+1]/monthly_spy['Adj Close'].ilo
              if i!=5:
                  market_result['Out of sample return'].iloc[i] = ((monthly_spy['Adj Close'].iloc[
                  for k in range(i*12+60,i*12+120):
                      out_return.append(monthly_spy['Adj Close'].iloc[k+1]/monthly_spy['Adj Close'
              else:
                  market_result['Out of sample return'].iloc[i] = ((monthly_spy['Adj Close'].iloc[
                  for k in range(i*12+60,i*12+119):
                      out_return.append(monthly_spy['Adj Close'].iloc[k+1]/monthly_spy['Adj Close'
              market_result['In sample return'].iloc[i] = ((monthly_spy['Adj Close'].iloc[i*12+60]
              market_result['In sample Sharpe ratio'].iloc[i] = np.mean(in_return)/np.std(in_retur
              market_result['Out of sample Sharpe ratio'].iloc[i] = np.mean(out_return)/np.std(out
```

```
In [197]: market_result
```

Out[197]:

	In sample return	Out of sample return	In sample Sharpe ratio	Out of sample Sharpe ratio
2008-2013-2017	0.039721	0.157935	0.087764	0.462775
2009-2014-2018	0.190298	0.108372	0.354377	0.284419
2010-2015-2019	0.155017	0.122244	0.344500	0.302337
2011-2016-2020	0.107727	0.160408	0.265928	0.313492
2022-2017-2021	0.139419	0.166570	0.386198	0.311661
2013-2018-2022	0.157935	0.081210	0.462775	0.151122

```
In [65]: wealth_process = pd.DataFrame(index = m_price.index[72:133],columns = ['EWP','DWP_positi
                                                                          'MCMC_positive','MC
```

```
In [66]: wealth_process.iloc[0] = 1
```

```
In [67]: start_index = 72
         end_index = 132
         for i in range(start_index,end_index):
             price_series = m_price.iloc[i].dropna()   # Price for existing companies
             port_size = len(price_series)
             ew_w = np.full(port_size,1/port_size)   # weights for EW
             now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].iloc[
             ew_return_now = np.dot(np.array(ew_w),np.array(now_return).T) # return for EW
             wealth_process['EWP'].iloc[i-71] = wealth_process['EWP'].iloc[i-72]*(1 + ew_return_n
```

In [68]:
```
start_index = 72
end_index = 132
p = 2.059512
for i in range(start_index,end_index):
    price_series = m_price.iloc[i].dropna()   # Price for existing companies
    port_size = len(price_series)
    miu_series = price_series/sum(price_series)
    miu_p_series = miu_series**p
    port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
    now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].iloc[
    port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for portf
    wealth_process['DWP_positive'].iloc[i-71] = wealth_process['DWP_positive'].iloc[i-72
```

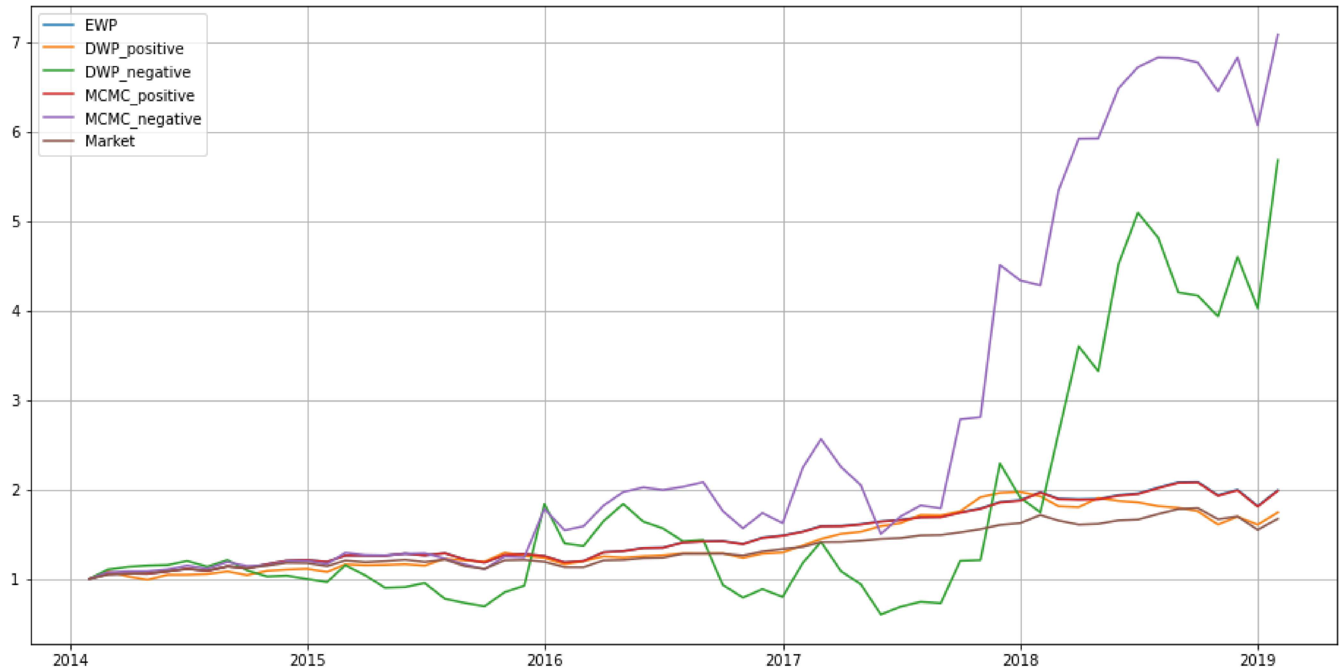
In [69]:
```
start_index = 72
end_index = 132
p = -5.790833
for i in range(start_index,end_index):
    price_series = m_price.iloc[i].dropna()   # Price for existing companies
    port_size = len(price_series)
    miu_series = price_series/sum(price_series)
    miu_p_series = miu_series**p
    port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
    now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].iloc[
    port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for portf
    wealth_process['DWP_negative'].iloc[i-71] = wealth_process['DWP_negative'].iloc[i-72
```

In [70]:
```
start_index = 72
end_index = 132
p = 0.036686
for i in range(start_index,end_index):
    price_series = m_price.iloc[i].dropna()   # Price for existing companies
    port_size = len(price_series)
    miu_series = price_series/sum(price_series)
    miu_p_series = miu_series**p
    port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
    now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].iloc[
    port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for portf
    wealth_process['MCMC_positive'].iloc[i-71] = wealth_process['MCMC_positive'].iloc[i-
```

In [71]:
```
start_index = 72
end_index = 132
p = -2.201698
for i in range(start_index,end_index):
    price_series = m_price.iloc[i].dropna()   # Price for existing companies
    port_size = len(price_series)
    miu_series = price_series/sum(price_series)
    miu_p_series = miu_series**p
    port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
    now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].iloc[
    port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for portf
    wealth_process['MCMC_negative'].iloc[i-71] = wealth_process['MCMC_negative'].iloc[i-
```

In [74]:
```
for i in range(1,len(wealth_process)):
    wealth_process['Market'].iloc[i] = wealth_process['Market'].iloc[i-1]*monthly_spy['A
```

In [76]:
```
plt.figure(figsize = (16,8))
plt.plot(wealth_process,label = wealth_process.columns)
plt.legend()
plt.grid()
```



In [75]:
```
wealth_process
```

Out[75]:

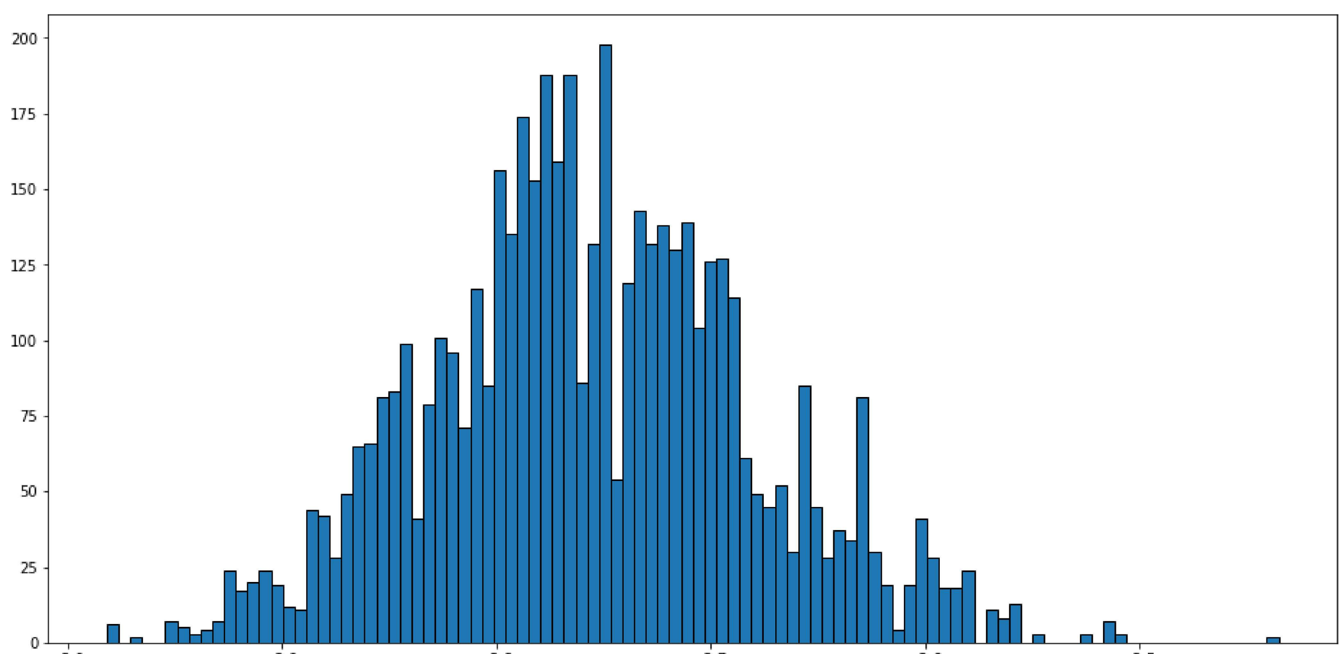
Date	EWP	DWP_positive	DWP_negative	MCMC_positive	MCMC_negative	Market
2014-01-31	1	1	1	1	1	1
2014-02-28	1.053516	1.083561	1.104321	1.053486	1.072681	1.045515
2014-03-31	1.057414	1.026214	1.134327	1.057174	1.083873	1.054188
2014-04-30	1.055021	0.989733	1.14848	1.054656	1.08527	1.061517
2014-05-31	1.083096	1.044341	1.154399	1.082829	1.10613	1.086151
...	...	...	...	...	...	...
2018-09-30	2.084444	1.756266	4.168382	2.077252	6.77684	1.791115
2018-10-31	1.936487	1.607426	3.935585	1.92886	6.454705	1.667342
2018-11-30	1.997767	1.695909	4.603026	1.990119	6.833905	1.69827
2018-12-31	1.814436	1.605865	4.027833	1.807924	6.071789	1.54874
2019-01-31	1.9924	1.743342	5.685332	1.984646	7.088731	1.672741

61 rows × 6 columns

In [44]:
```
start_index = 120
end_index = 180
times = 10000
samples = [-0.1]
total_num = 0
while total_num < times:
    current_sample = samples[-1]
    proposal = np.random.normal(current_sample, 0.5)
    if -10 <= proposal < 0:
        now_er = ER(proposal,start_index,end_index)  # new ER
        past_er = ER(current_sample,start_index,end_index) # last ER
        acceptance_prob = min(1, target_pdf(now_er) / target_pdf(past_er))
    else:
        acceptance_prob = 0
    if np.random.rand() < acceptance_prob:
        samples.append(proposal)
    else:
        samples.append(current_sample)
    total_num += 1
```

```
In [53]: plt.figure(figsize = (16,8))
         plt.hist(samples[5000:],bins = 100,edgecolor = 'black')
```

```
Out[53]: (array([  6.,   0.,   2.,   0.,   0.,   7.,   5.,   3.,   4.,   7.,  24.,
          17.,  20.,  24.,  19.,  12.,  11.,  44.,  42.,  28.,  49.,  65.,
          66.,  81.,  83.,  99.,  41.,  79., 101.,  96.,  71., 117.,  85.,
         156., 135., 174., 153., 188., 159., 188.,  86., 132., 198.,  54.,
         119., 143., 132., 138., 130., 139., 104., 126., 127., 114.,  61.,
          49.,  45.,  52.,  30.,  85.,  45.,  28.,  37.,  34.,  81.,  30.,
          19.,   4.,  19.,  41.,  28.,  18.,  18.,  24.,   0.,  11.,   8.,
          13.,   0.,   3.,   0.,   0.,   0.,   3.,   0.,   7.,   3.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           2.]),
  array([-2.9818129 , -2.97634696, -2.97088101, -2.96541507, -2.95994912,
         -2.95448317, -2.94901723, -2.94355128, -2.93808534, -2.93261939,
         -2.92715344, -2.9216875 , -2.91622155, -2.91075561, -2.90528966,
         -2.89982372, -2.89435777, -2.88889182, -2.88342588, -2.87795993,
         -2.87249399, -2.86702804, -2.86156209, -2.85609615, -2.8506302 ,
         -2.84516426, -2.83969831, -2.83423236, -2.82876642, -2.82330047,
         -2.81783453, -2.81236858, -2.80690263, -2.80143669, -2.79597074,
         -2.7905048 , -2.78503885, -2.77957291, -2.77410696, -2.76864101,
         -2.76317507, -2.75770912, -2.75224318, -2.74677723, -2.74131128,
         -2.73584534, -2.73037939, -2.72491345, -2.7194475 , -2.71398155,
         -2.70851561, -2.70304966, -2.69758372, -2.69211777, -2.68665182,
         -2.68118588, -2.67571993, -2.67025399, -2.66478804, -2.65932209,
         -2.65385615, -2.6483902 , -2.64292426, -2.63745831, -2.63199237,
         -2.62652642, -2.62106047, -2.61559453, -2.61012858, -2.60466264,
         -2.59919669, -2.59373074, -2.5882648 , -2.58279885, -2.57733291,
         -2.57186696, -2.56640101, -2.56093507, -2.55546912, -2.55000318,
         -2.54453723, -2.53907128, -2.53360534, -2.52813939, -2.52267345,
         -2.5172075 , -2.51174155, -2.50627561, -2.50080966, -2.49534372,
         -2.48987777, -2.48441183, -2.47894588, -2.47347993, -2.46801399,
         -2.46254804, -2.4570821 , -2.45161615, -2.4461502 , -2.44068426,
         -2.43521831]),
  <BarContainer object of 100 artists>)
```



In [ ]:

```

```