real_stocks

July 26, 2021

[1]:
```
import sys
import os
import matplotlib.pyplot as plt
import numpy as np
import random

plt.style.use(['science', 'retro', 'grid'])

import sys
import cvxopt as opt
sys.path.insert(0, os.path.abspath('../../src/rl/dir_vec/'))
sys.path.insert(1, os.path.abspath('../../src/rl/environments/'))
sys.path.insert(1, os.path.abspath('../../src/rl/environments/base'))
sys.path.insert(2, os.path.abspath('../../src/mpt/efficient_frontier/'))
sys.path.insert(3, os.path.abspath('../../src/mpt/utils'))
```

[4]:
```
import importlib
importlib.reload(sys.modules['EfficientFrontier'])
importlib.reload(sys.modules['GenerativeMarketEnv'])
importlib.reload(sys.modules['HarnessVec'])
importlib.reload(sys.modules['ParallelRunnerVec'])
importlib.reload(sys.modules['DirichletPolicyVec'])
```

[4]:
```
<module 'DirichletPolicyVec' from '/Users/michael/UCL/Disertation/porfolio-
optimisation/src/rl/dir_vec/DirichletPolicyVec.py'>
```

[5]:
```
from GenerativeMarketEnv import GenerativeMarketEnv
from DirichletPolicyVec import DirichletPolicyVec
from HarnessVec import HarnessVec
from ParallelRunnerVec import ParallelRunnerVec
from EfficientFrontier import EfficientFrontier
from MarketFactory import MarketFactory
```

1 Investigation of the Viability of the Dirichlet Policy on Real Stock

This notebook looks at optimising for real stocks. We calculate the empirical means and covariance matrices of a number of stocks and create a generative environment (multivariate Gaussian). This enables us to plot the efficient frontier for the assets (blue line in plots is EF, yellow crosses are the assets) and the maximum Sharpe ratio (pink cross).

We then learn the policy, first optimising for returns and then the differential Sharpe ratio. Once a policy has been learned we generate a series of returns and plot the empirical mean return and volatility (cyan dots).

[6]:
```
# stocks to consider
stocks = ['AAPL' ,'ADBE' ,'ADI' ,'ADP' ,'ADSK' ,'AEP' ,'ALGN' ,'ALXN' ,'AMAT'␣
 ↪,'AMD' ,'AMGN' ,'AMZN' ,'ANSS' ,'ASML' ,'ATVI'
 ,'AVGO' ,'BIDU' ,'BIIB' ,'BKNG' ,'CDNS' ,'CDW' ,'CERN' ,'CHKP' ,'CHTR'␣
 ↪,'CMCSA' ,'COST' ,'CPRT' ,'CSCO' ,'CSX' ,'CTAS'
 ,'CTSH' ,'DLTR' ,'DOCU' ,'DXCM' ,'EA' ,'EBAY' ,'EXC' ,'FAST' ,'FB' ,'FISV'␣
 ↪,'FOX' ,'FOXA' ,'GILD' ,'GOOG' ,'GOOGL'
 ,'IDXX' ,'ILMN' ,'INCY' ,'INTC' ,'INTU' ,'ISRG' ,'JD' ,'KDP' ,'KHC' ,'KLAC'␣
 ↪,'LRCX' ,'LULU' ,'MAR' ,'MCHP' ,'MDLZ'
 ,'MELI' ,'MNST' ,'MRNA' ,'MRVL' ,'MSFT' ,'MTCH' ,'MU' ,'MXIM' ,'NFLX' ,'NTES'␣
 ↪,'NVDA' ,'NXPI' ,'OKTA' ,'ORLY' ,'PAYX'
 ,'PCAR' ,'PDD' ,'PEP' ,'PTON' ,'PYPL' ,'QCOM' ,'REGN' ,'ROST' ,'SBUX' ,'SGEN'␣
 ↪,'SIRI' ,'SNPS' ,'SPLK' ,'SWKS' ,'TCOM'
 ,'TEAM' ,'TMUS' ,'TSLA' ,'TXN' ,'VRSK' ,'VRSN' ,'VRTX' ,'WBA' ,'WDAY' ,'XEL'␣
 ↪,'XLNX' ,'ZM']
```

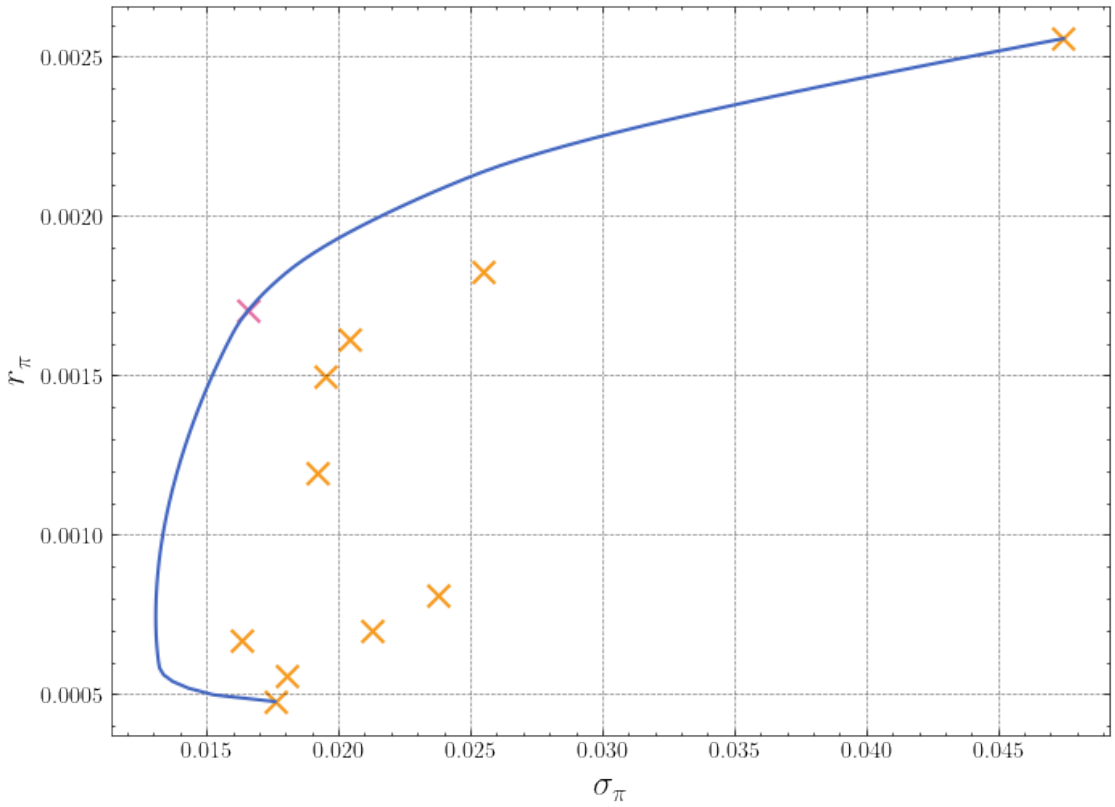
1.1 Maximising Returns of Real Stocks

1.1.1 10 Stocks

[7]:
```
random.shuffle(stocks)
sampled_stocks_10 = stocks[:10]
```

[9]:
```
factory = MarketFactory(sampled_stocks_10, "2017-01-01", "2020-04-30")
market = factory.create_market()
market.plot_efficient_frontier()
```

```
[**********************100%**********************]  10 of 10 completed
```

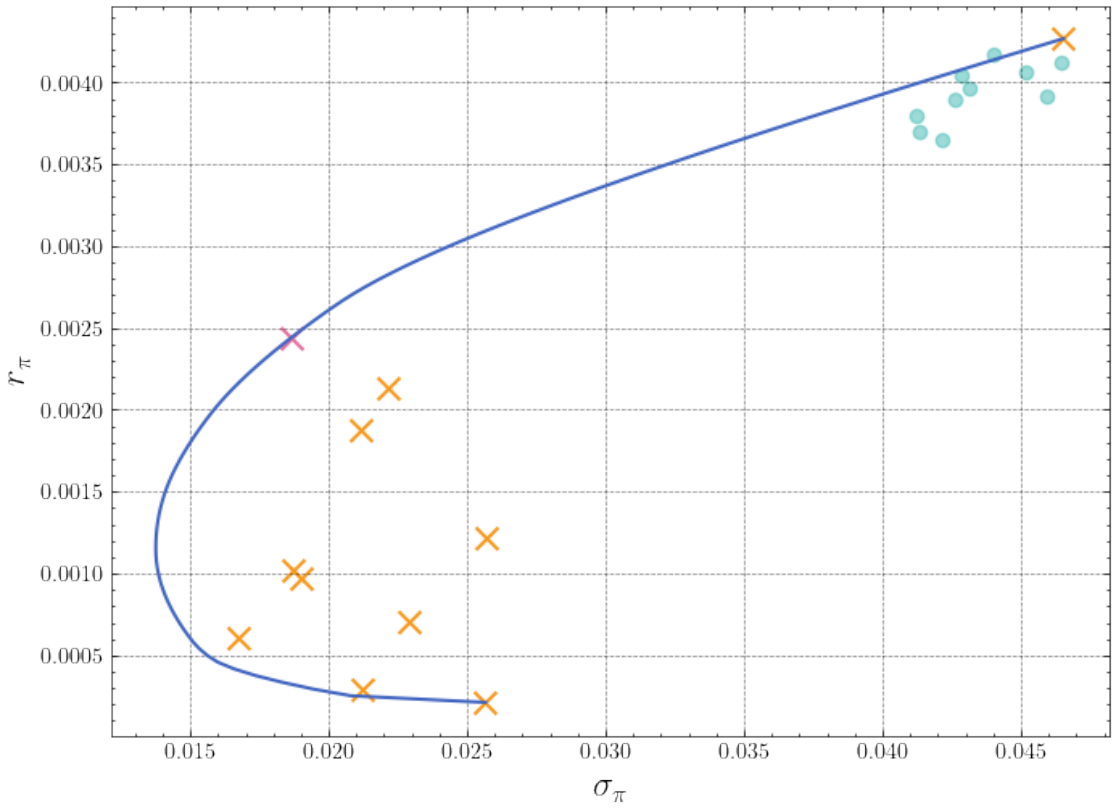


[9]: 0.1027272313852195

[ ]:
```
params = {
    'n_assets': len(sampled_stocks_10),
    ' ': 2,
    ' _end': 9e-3,
    'start_ep': 9999,
    ' _decay_steps': 9999,
    ' ': 0.1,
    'grad_adpt_mode': 'natural_gradient',
    'returns_adpt_mode': None,
    'parameterisation': 'softplus',
    'eps': 10000
}

p_runner = ParallelRunnerVec(market. , market.Σ, 10, [params], 50,␣
 ↪reward_mode='returns')
p_runner.run_test()
means = []
stds = []
for i in range(len(p_runner.results['0'])):
```

```
    means.append(np.mean(p_runner.results['0'][i]))
    stds.append(np.std(p_runner.results['0'][i]))
```

[24]:
```
market.plot_efficient_frontier(stds = stds, returns = means)
```

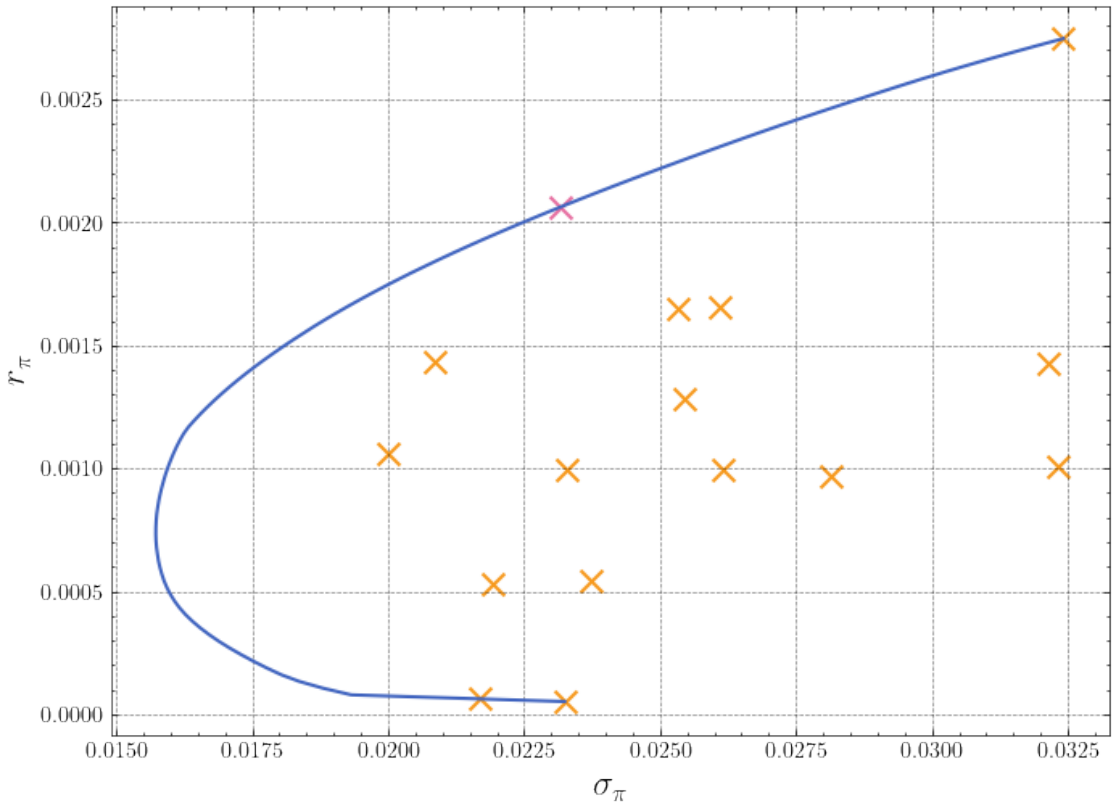


1.1.2 15 stocks

[29]:
```
random.shuffle(stocks)
sampled_stocks_15 = stocks[:15]
```

[30]:
```
factory = MarketFactory(sampled_stocks_15, "2019-04-30", "2021-04-30")
market = factory.create_market()
market.plot_efficient_frontier()
```

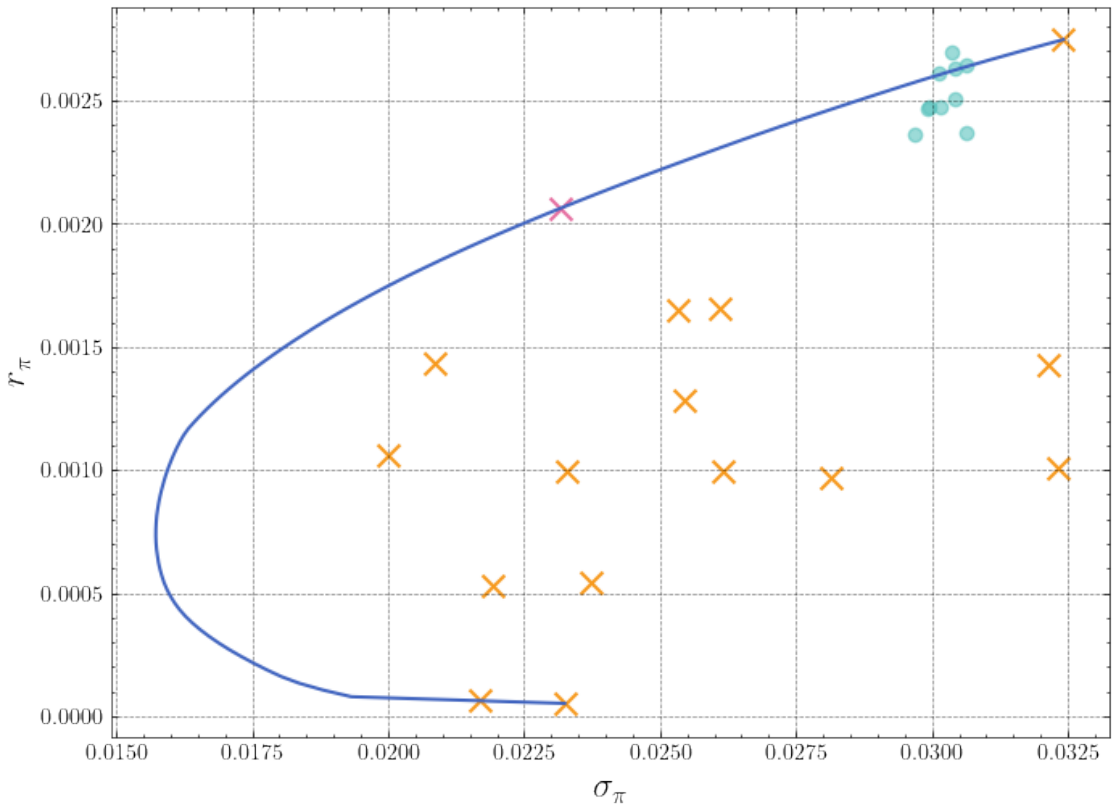
```
[**********************100%**********************]  15 of 15 completed
```



[53]:
```
params = {
    'n_assets': len(sampled_stocks_15),
    ' ': 10,
    ' _end': 1,
    'start_ep': 1000,
    ' _decay_steps': 2000,
    ' ': 0.1,
    'grad_adpt_mode': 'natural_gradient',
    'returns_adpt_mode': None,
    'parameterisation': 'softplus',
    'eps': 10000
}

p_runner = ParallelRunnerVec(market. , market.Σ, 10, [params], 50,␣
 ↪reward_mode='returns')
p_runner.run_test()
means = []
stds = []
for i in range(len(p_runner.results['0'])):
    means.append(np.mean(p_runner.results['0'][i]))
    stds.append(np.std(p_runner.results['0'][i]))
```

```
0it [00:00, ?it/s]
  0%|          | 0/10 [00:00<?, ?it/s]
100%|      | 10/10 [00:00<00:00, 61.27it/s]
1it [03:04, 184.22s/it]
```

[54]:
```
market.plot_efficient_frontier(stds = stds, returns = means)
```

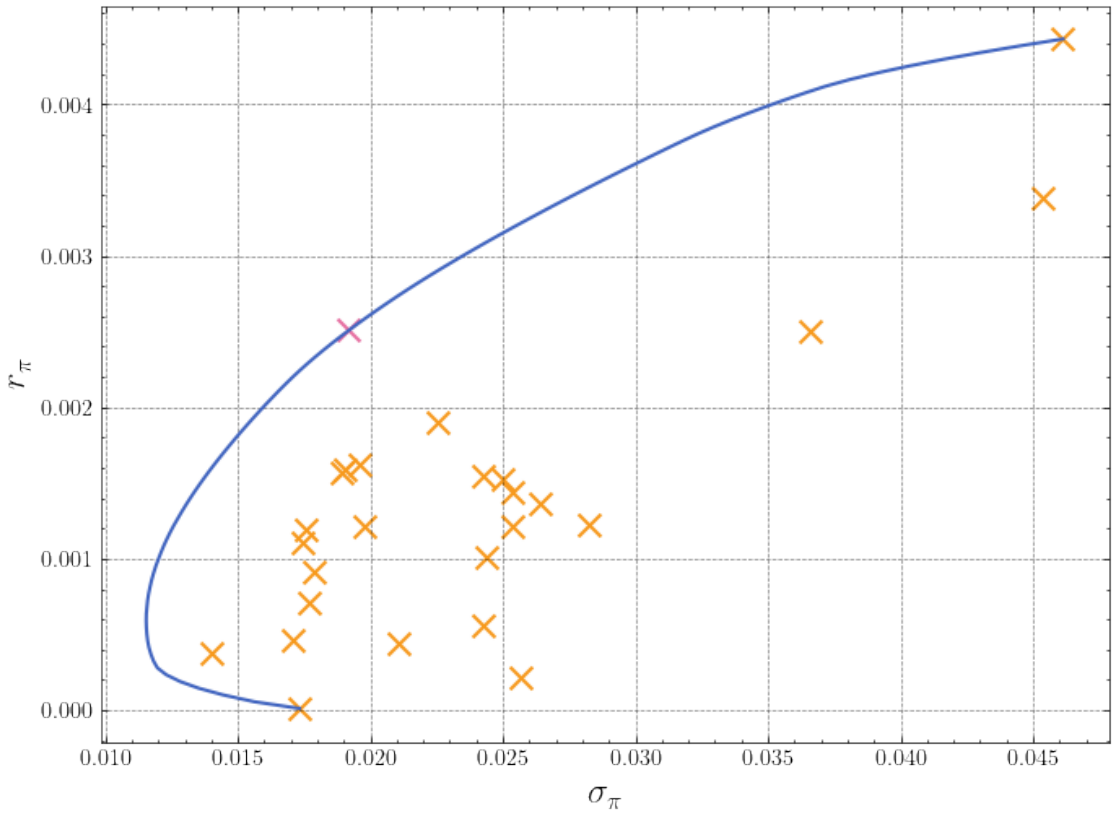


1.1.3 25 Stocks

[57]:
```
random.shuffle(stocks)
sampled_stocks_25 = stocks[:25]
factory = MarketFactory(sampled_stocks_25, "2017-01-01", "2021-04-30")
```

[58]:
```
market = factory.create_market()
market.plot_efficient_frontier()
```

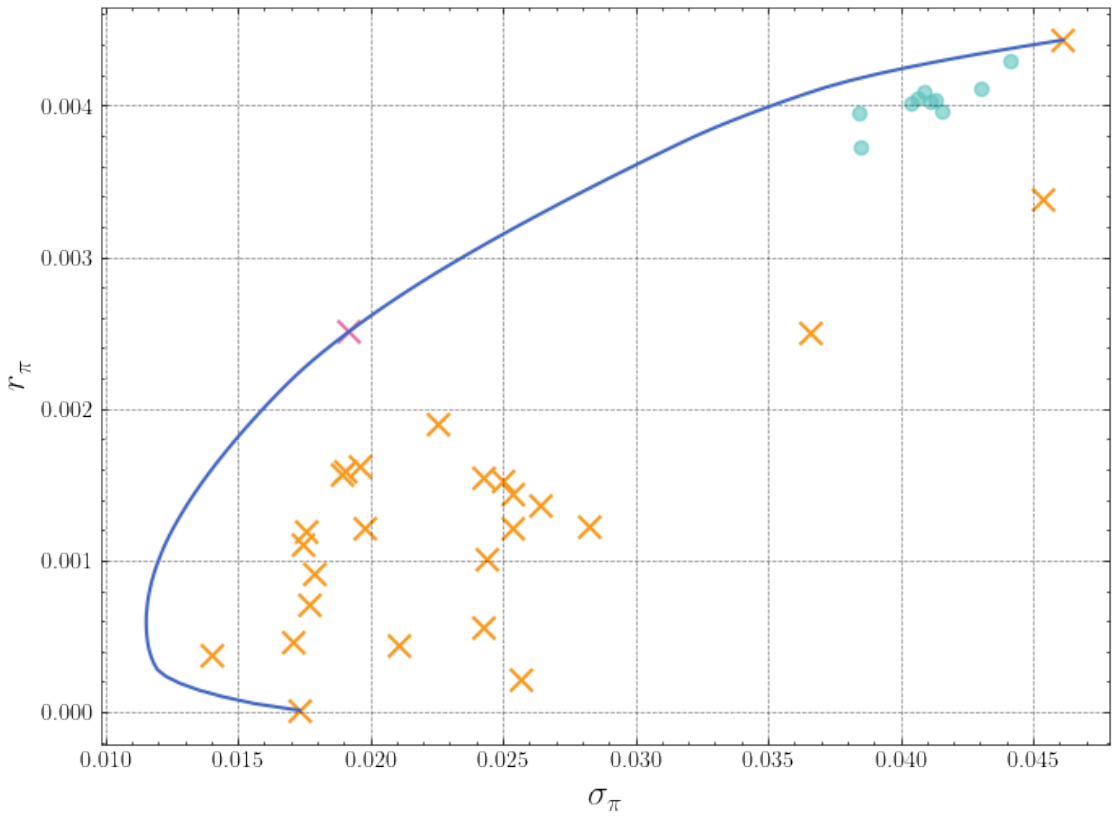
```
[**********************100%**********************]  25 of 25 completed
```



[59]:
```
params = {
    'n_assets': len(sampled_stocks_25),
    ' ': 10,
    ' _end': 1,
    'start_ep': 2000,
    ' _decay_steps': 3000,
    ' ': 0.1,
    'grad_adpt_mode': 'natural_gradient',
    'returns_adpt_mode': None,
    'parameterisation': 'softplus',
    'eps': 10000
}

p_runner = ParallelRunnerVec(market. , market.Σ, 10, [params], 50,␣
 ↪reward_mode='returns')
p_runner.run_test()
means = []
stds = []
for i in range(len(p_runner.results['0'])):
    means.append(np.mean(p_runner.results['0'][i]))
    stds.append(np.std(p_runner.results['0'][i]))
```

```
0it [00:00, ?it/s]
  0%|          | 0/10 [00:00<?, ?it/s]
100%|      | 10/10 [00:00<00:00, 24.56it/s]
1it [03:45, 225.74s/it]
```

[60]:
```
market.plot_efficient_frontier(stds = stds, returns = means)
```

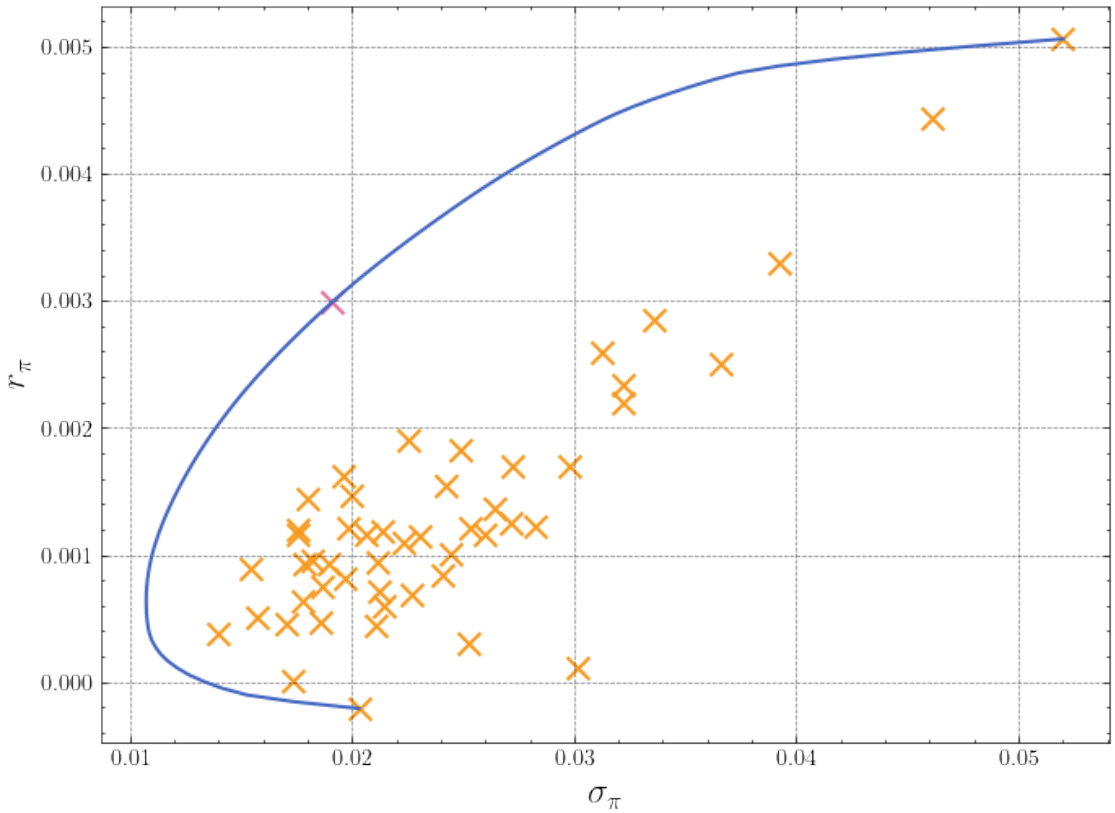


1.1.4 50 Stocks

[61]:
```
random.shuffle(stocks)
sampled_stocks_50 = stocks[:50]
factory = MarketFactory(sampled_stocks_50, "2017-01-01", "2021-04-30")
```

[62]:
```
market = factory.create_market()
market.plot_efficient_frontier()
```

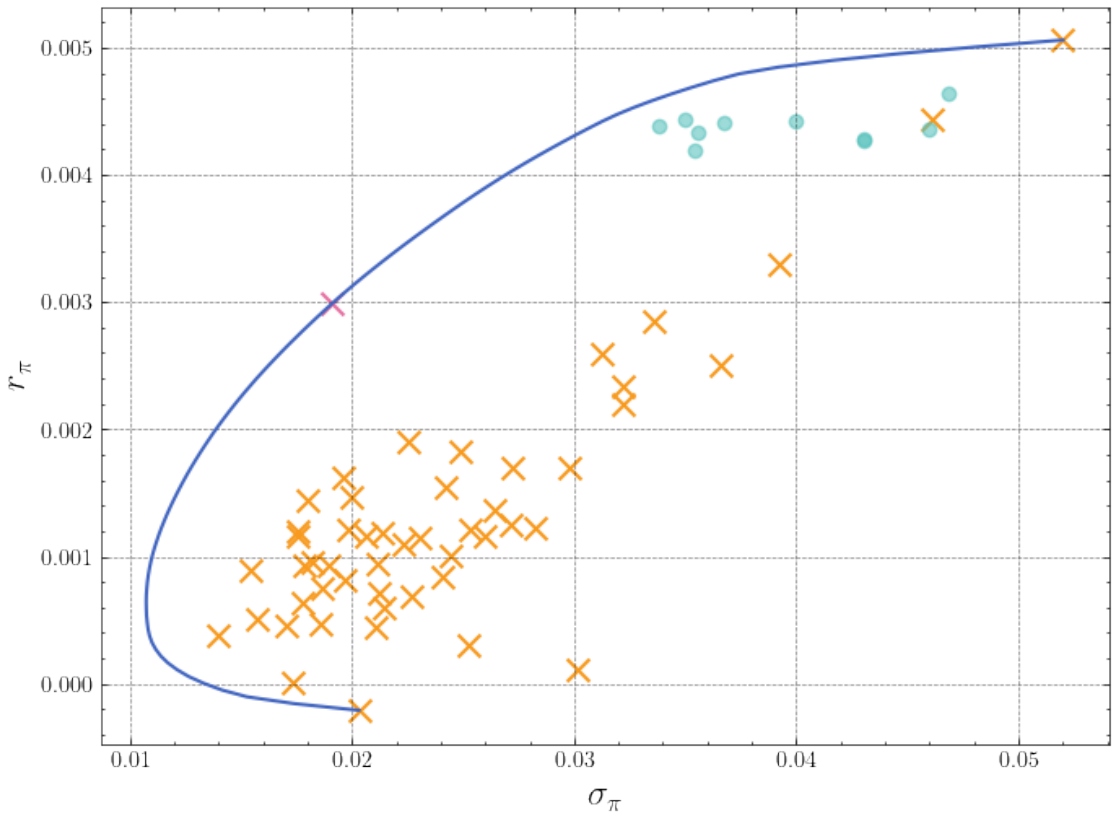
```
[**********100%**********]  50 of 50 completed
```



[69]:
```
params = {
    'n_assets': len(sampled_stocks_50),
    ' ': 10,
    ' _end': 2,
    'start_ep': 4000,
    ' _decay_steps': 6000,
    ' ': 0.1,
    'grad_adpt_mode': 'natural_gradient',
    'returns_adpt_mode': None,
    'parameterisation': 'softplus',
    'eps': 15000
}

p_runner = ParallelRunnerVec(market. , market.Σ, 10, [params], 50,␣
 ↪reward_mode='returns')
p_runner.run_test()
means = []
stds = []
for i in range(len(p_runner.results['0'])):
    means.append(np.mean(p_runner.results['0'][i]))
    stds.append(np.std(p_runner.results['0'][i]))
```

```
0it [00:00, ?it/s]
  0%|          | 0/10 [00:00<?, ?it/s]
100%|      | 10/10 [00:00<00:00, 46.84it/s]
1it [07:11, 431.06s/it]
```

[70]:
```
market.plot_efficient_frontier(stds = stds, returns = means)
```

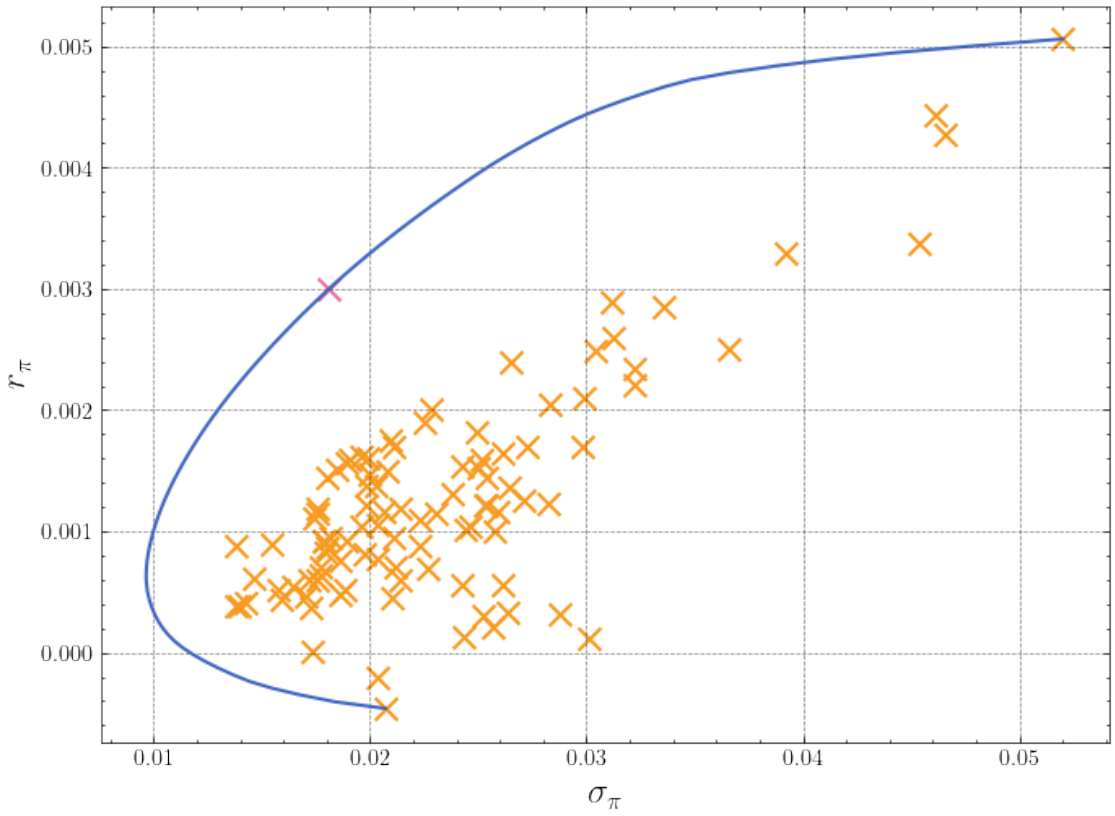


1.1.5 100 Stocks

[71]:
```
random.shuffle(stocks)
sampled_stocks_50 = stocks[:100]
factory = MarketFactory(sampled_stocks_50, "2017-01-01", "2021-04-30")
```

[72]:
```
market = factory.create_market()
market.plot_efficient_frontier()
```

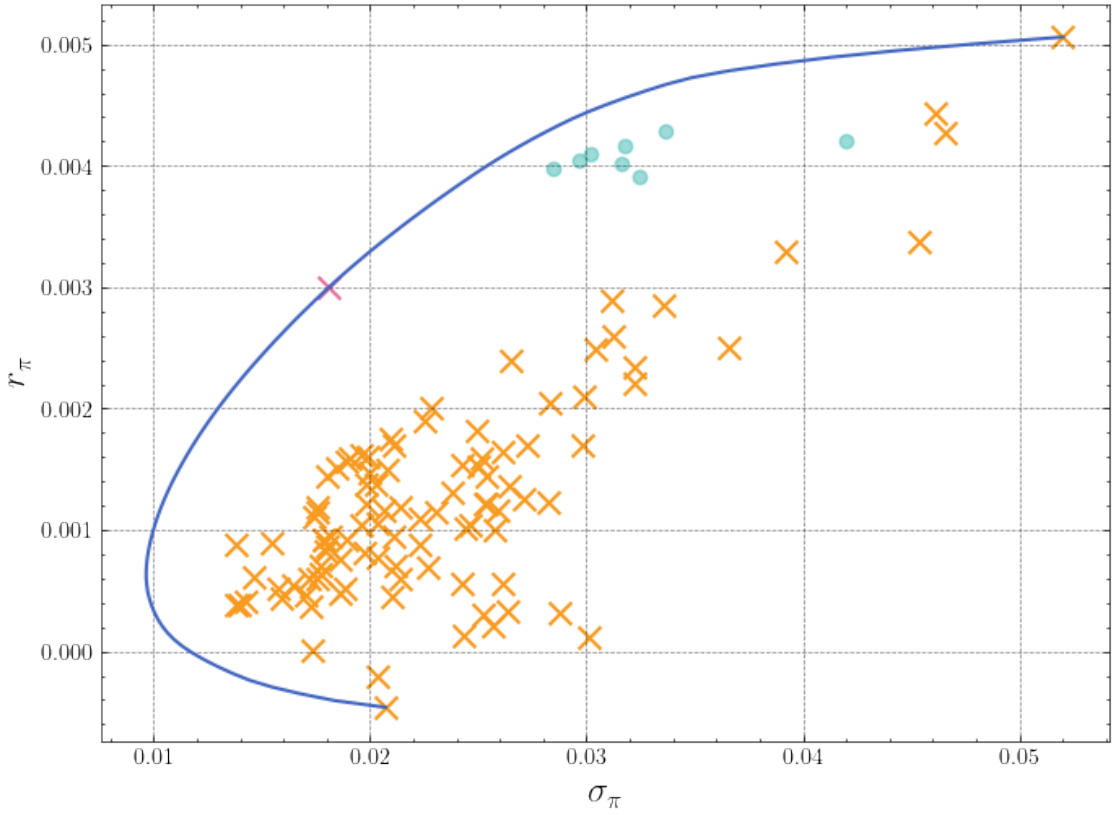
```
[**********************100%**********************]  100 of 100 completed
```



[80]:
```
params = {
    'n_assets': len(sampled_stocks_50),
    ' ': 20,
    ' _end': 2,
    'start_ep': 5000,
    ' _decay_steps': 5000,
    ' ': 0.1,
    'grad_adpt_mode': 'natural_gradient',
    'returns_adpt_mode': None,
    'parameterisation': 'softplus',
    'eps': 20000
}

p_runner = ParallelRunnerVec(market. , market.Σ, 8, [params], 50,␣
 ↪reward_mode='returns')
p_runner.run_test()
means = []
stds = []
for i in range(len(p_runner.results['0'])):
    means.append(np.mean(p_runner.results['0'][i]))
    stds.append(np.std(p_runner.results['0'][i]))
```

```
0it [00:00, ?it/s]
100%|      | 8/8 [00:00<00:00, 1633.38it/s]
1it [20:01, 1201.95s/it]
```

[81]:
```
market.plot_efficient_frontier(stds = stds, returns = means)
```



1.2 Maximising the Differential Sharpe Ratio of Real Stocks

The Sharpe ratio is a measure of risk-adjusted return. It is defined as

$$S_t = \frac{\text{Ave}(R_t)}{\text{Std}(R_t)}$$

Moody et al. 1998 derive an online learning friendly objective called the *differential Sharpe ratio*. It is calculated using exponential moving averages of the returns and standard deviation of returns and then considering a first-order expansion around the decay rate.

$$S_t \approx S_{t-1} + \eta \left.\frac{dS_t}{d\eta}\right|_{\eta=0} + \mathcal{O}(\eta^2)$$

The *differential Sharpe ratio* is then defined as

$$D_t = \frac{dS_t}{d\eta} = \frac{B_{t-1}\Delta A_t - \frac{1}{2}A_{t-1}\Delta B_t}{(B_{t-1} - A_{t-1}^2)^{\frac{3}{2}}}$$

Where A_t and B_t are exponential moving estimates of the first and second moments of the returns at time t or

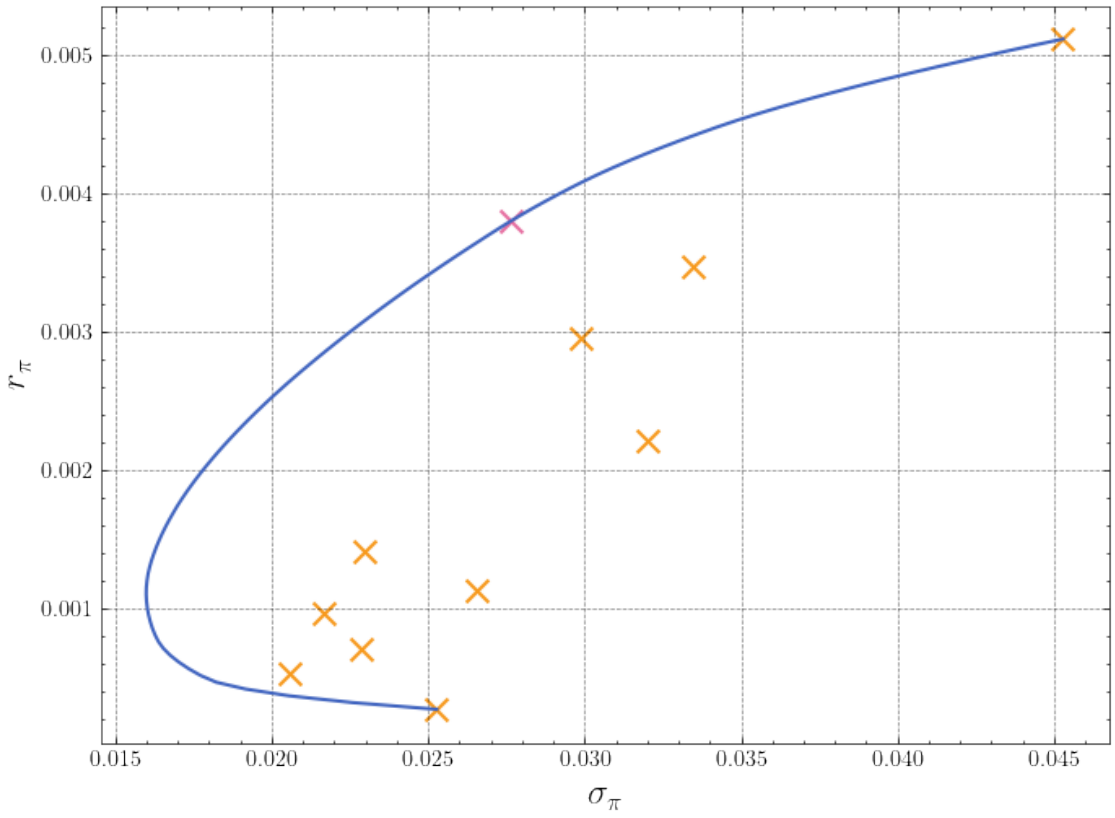
$$A_t = A_{t-1} + \eta(R_t - A_{t-1})$$

$$B_t = B_{t-1} + \eta(R_t^2 - B_{t-1})$$

1.2.1 10 Stocks

[117]:
```
random.shuffle(stocks)
sampled_stocks = stocks[:10]
factory = MarketFactory(sampled_stocks, "2019-01-01", "2021-04-30")
market = factory.create_market()
market.plot_efficient_frontier()
```

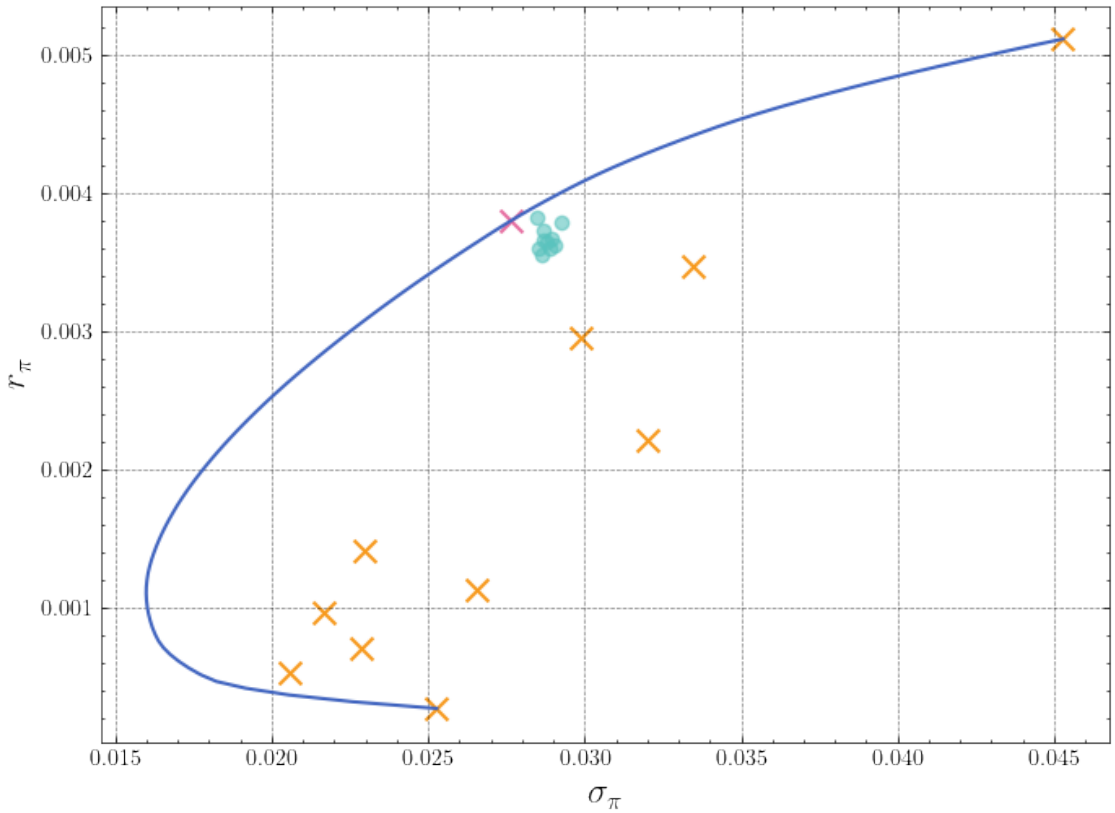
```
[**********************100%**********************]  10 of 10 completed
```



[130]:
```
params_1 = {
    'n_assets': 10,
    ' ': 0.005,
    ' _end': 0.002,
    'start_ep': 1000,
    ' _decay_steps': 4000,
    ' ': 0.1,
    'grad_adpt_mode': 'natural_gradient',
    'returns_adpt_mode': None,
    'parameterisation': 'softplus',
    'eps': 8000
}



legend_labels = {'0': 'DSR'}

params = [params_1]
p_runner = ParallelRunnerVec(market. , market.Σ, 10, params, 100,␣
 ↪legend_labels, reward_mode='dsr',  =0.05)
p_runner.run_test()
```

```
0it [00:00, ?it/s]
100%|      | 10/10 [00:00<00:00, 3034.07it/s]
1it [06:03, 363.26s/it]
```

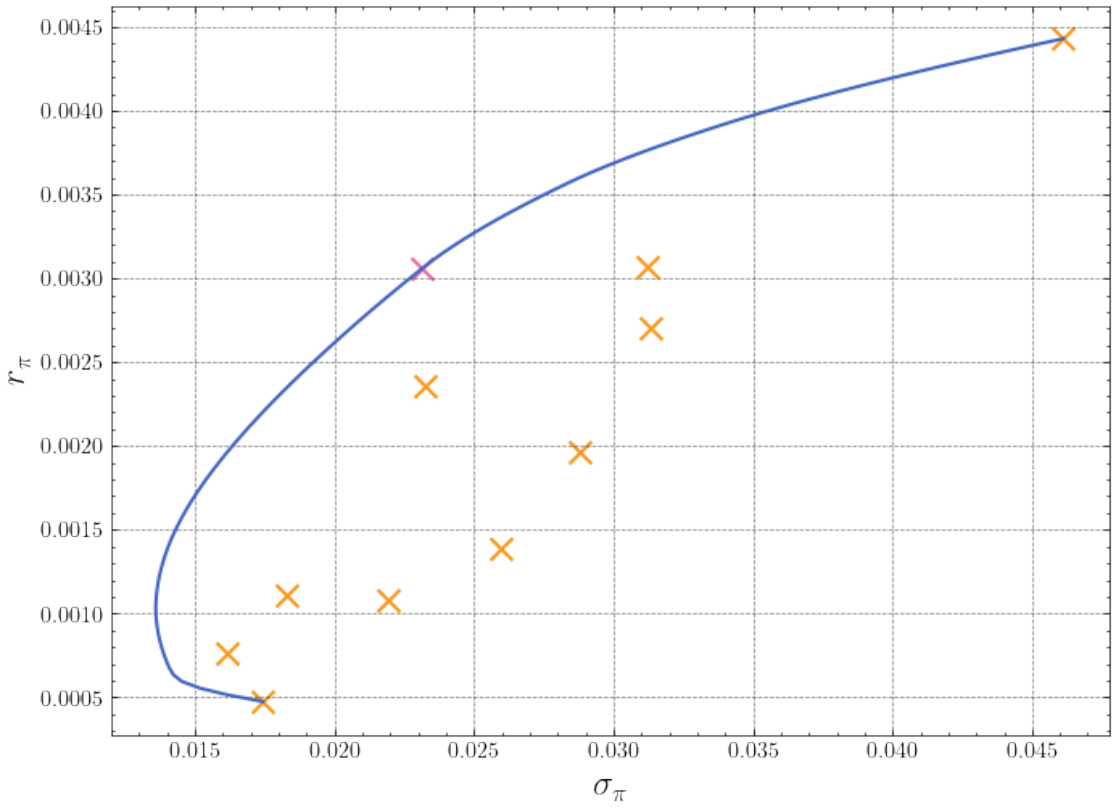
[131]:
```
means = []
stds = []
for i in range(len(p_runner.results['0'])):
    means.append(np.mean(p_runner.results['0'][i]))
    stds.append(np.std(p_runner.results['0'][i]))
market.plot_efficient_frontier(stds=stds, returns=means)
```



[62]:
```
random.shuffle(stocks)
sampled_stocks = stocks[:10]
factory = MarketFactory(sampled_stocks, "2019-01-01", "2021-04-30")
```

[63]:
```
market = factory.create_market()
sr = market.plot_efficient_frontier()
```

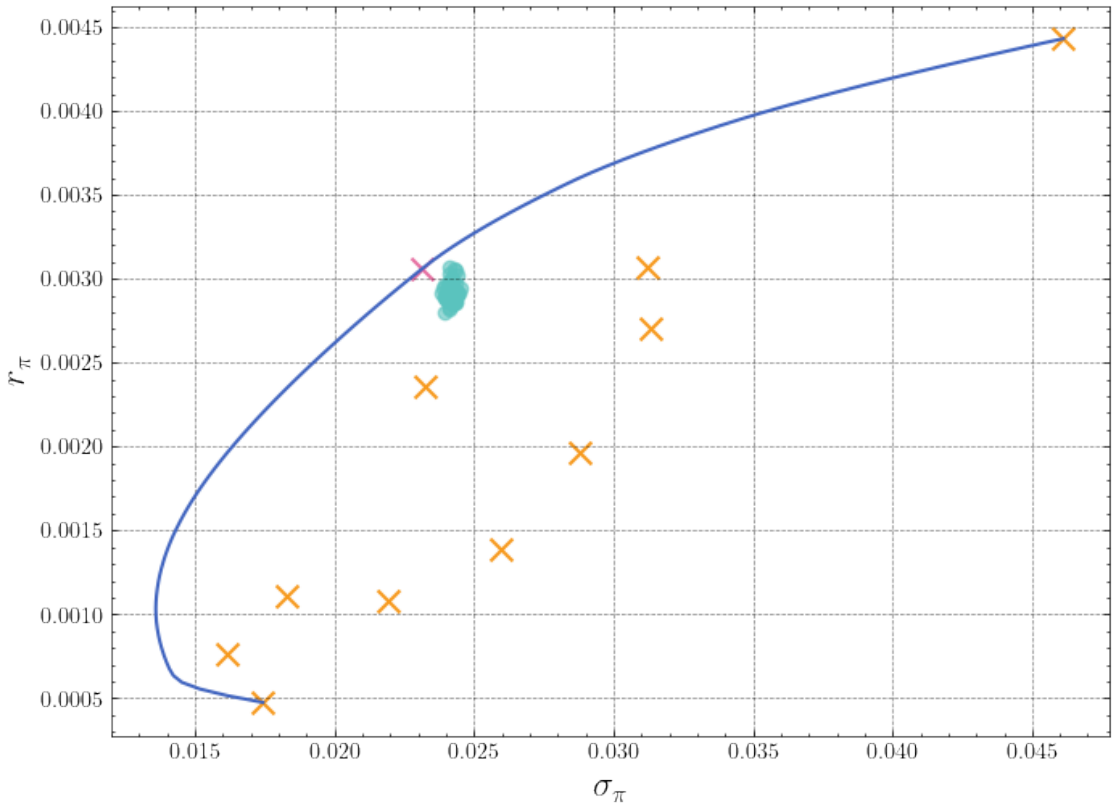
```
[**********************100%**********************]  10 of 10 completed
```



[96]:
```
params_1 = {
    'n_assets': 10,
    ' ': 0.003,
    ' _end': 0.001,
    'start_ep': 1000,
    ' _decay_steps': 3000,
    ' ': 0.1,
    'grad_adpt_mode': 'natural_gradient',
    'returns_adpt_mode': None,
    'parameterisation': 'softplus',
    'eps': 10000
}



legend_labels = {'0': 'DSR'}

params = [params_1]
p_runner = ParallelRunnerVec(market. , market.Σ, 40, params, 100,␣
 ↪legend_labels, reward_mode='dsr',  =0.05)
p_runner.run_test()
```

```
0it [00:00, ?it/s]

  0%|          | 0/40 [00:00<?, ?it/s]

 25%|          | 10/40 [00:00<00:00, 69.80it/s]

 25%|          | 10/40 [00:13<00:00, 69.80it/s]

 50%|         | 20/40 [05:42<03:25, 10.27s/it]

 75%|       | 30/40 [10:49<02:44, 16.42s/it]

100%|      | 40/40 [16:11<00:00, 24.29s/it]
1it [21:43, 1303.91s/it]
```

[97]:
```
means = []
stds = []
for i in range(len(p_runner.results['0'])):
    means.append(np.mean(p_runner.results['0'][i]))
    stds.append(np.std(p_runner.results['0'][i]))
market.plot_efficient_frontier(stds=stds, returns=means)
```



[97]:
```
0.13216786633675448
```

[87]:
```
means_eq = []
stds_eq = []
for i in range(len(p_runner_eq.results['0'])):
    means_eq.append(np.mean(p_runner_eq.results['0'][i]))
    stds_eq.append(np.std(p_runner_eq.results['0'][i]))
```

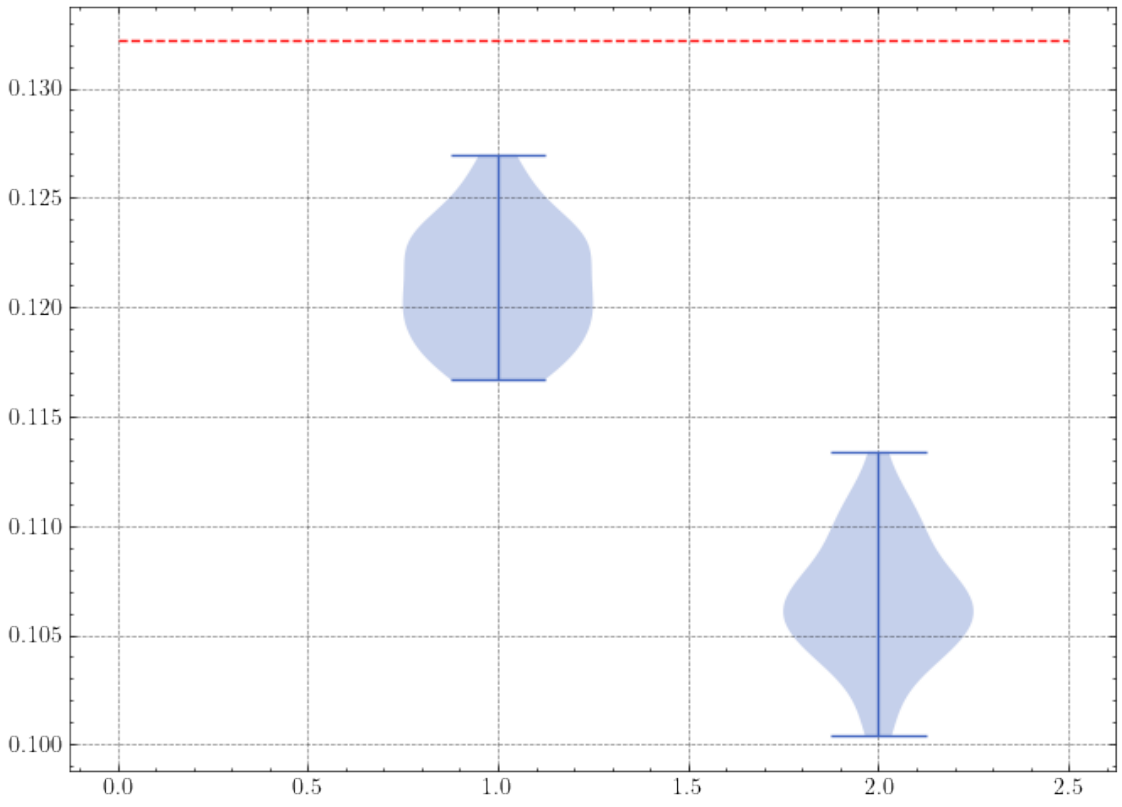
[98]:
```
means = []
stds = []
for i in range(len(p_runner.results['0'])):
    means.append(np.mean(p_runner.results['0'][i]))
    stds.append(np.std(p_runner.results['0'][i]))
```

[99]:
```
sharpe = np.array(means)/np.array(stds)
```

[100]:
```
sharpe_eq = np.array(means_eq)/np.array(stds_eq)
```

[101]:
```
plt.figure(figsize=(8, 6), dpi=100)
plt.violinplot([sharpe, sharpe_eq]);
plt.hlines(y=sr, xmin=0, xmax=2.5, colors='r', linestyles='--')
```

[101]:
```
<matplotlib.collections.LineCollection at 0x12c197490>
```

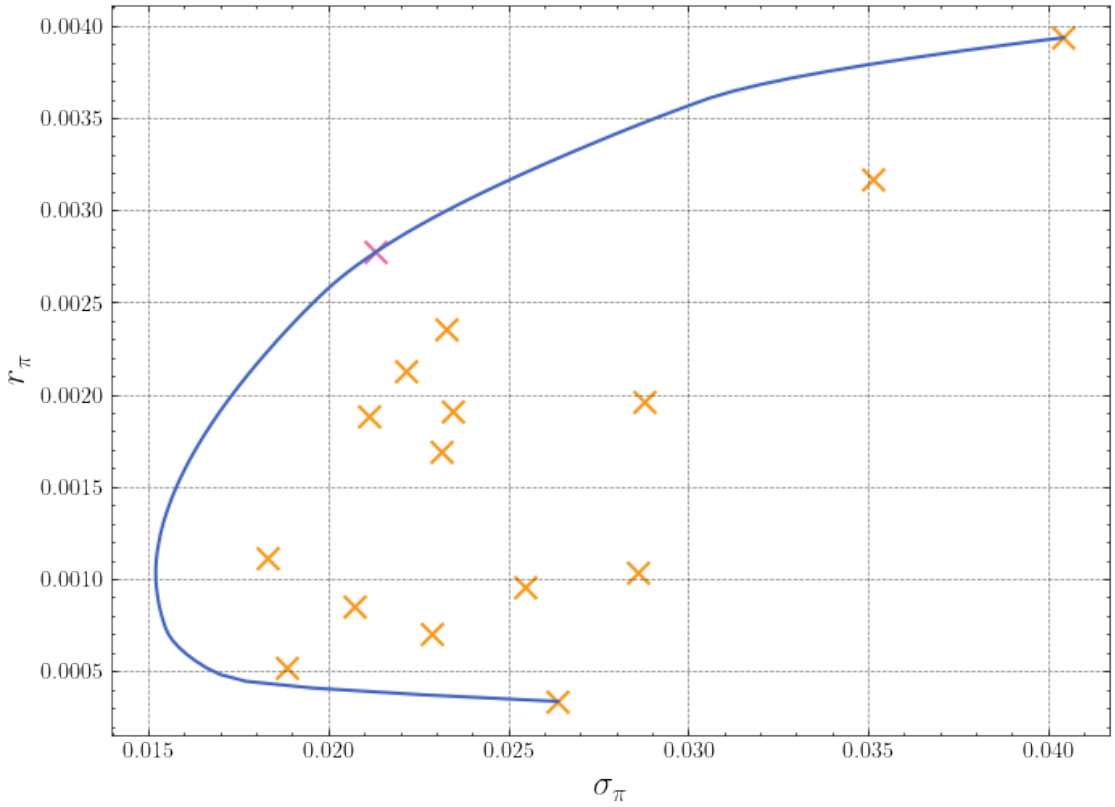


1.2.2 15 Stocks

[140]:
```
random.shuffle(stocks)
sampled_stocks = stocks[:15]
factory = MarketFactory(sampled_stocks, "2019-01-01", "2021-04-30")
```

[141]:
```
market = factory.create_market()
market.plot_efficient_frontier()
```

```
[**********************100%**********************]  15 of 15 completed
```

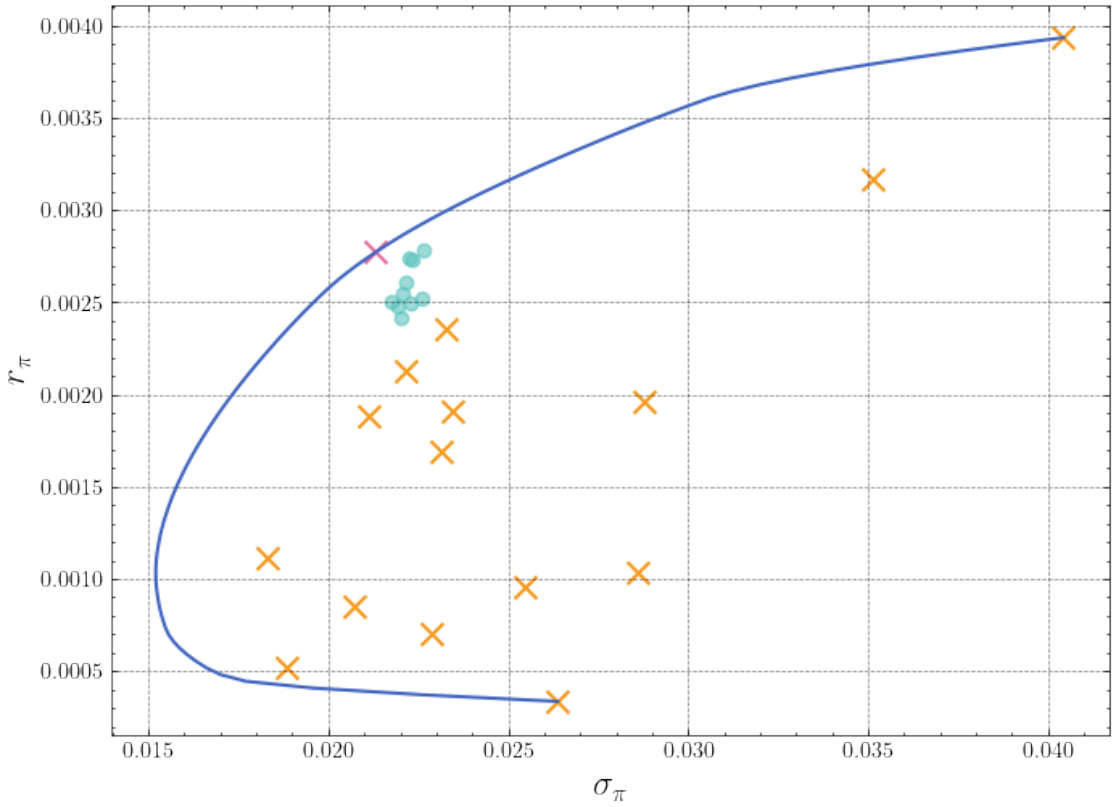


[144]:
```
params_1 = {
    'n_assets': 15,
    ' ': 0.005,
    ' _end': 0.001,
    'start_ep': 1000,
    ' _decay_steps': 5000,
    ' ': 0.1,
    'grad_adpt_mode': 'natural_gradient',
    'returns_adpt_mode': None,
```

```
    'parameterisation': 'softplus',
    'eps': 9000
}



legend_labels = {'0': 'DSR'}

params = [params_1]
p_runner = ParallelRunnerVec(market. , market.Σ, 10, params, 100,␣
 ↪legend_labels, reward_mode='dsr',  =0.05)
p_runner.run_test()
```

```
0it [00:00, ?it/s]
100%|      | 10/10 [00:00<00:00, 3280.90it/s]
1it [07:18, 438.35s/it]
```

[146]:
```
means = []
stds = []
for i in range(len(p_runner.results['0'])):
    means.append(np.mean(p_runner.results['0'][i]))
    stds.append(np.std(p_runner.results['0'][i]))
market.plot_efficient_frontier(stds=stds, returns=means)
```

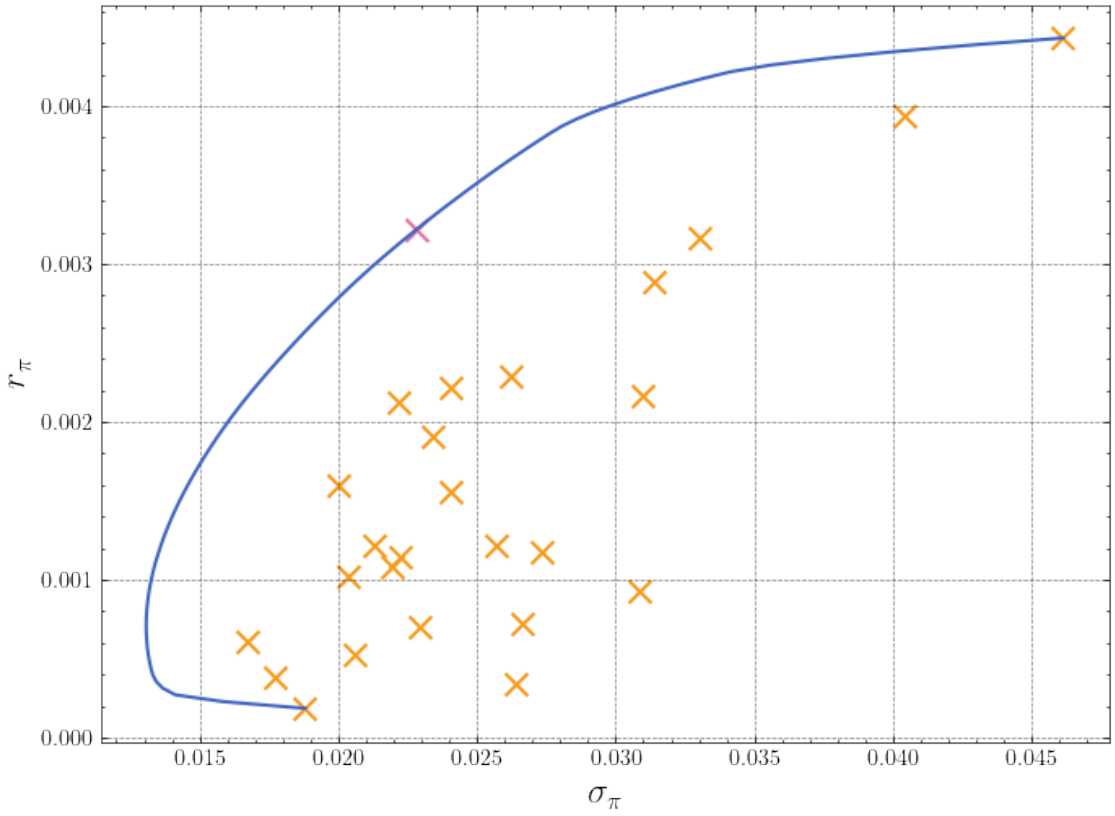


1.2.3 25 Stocks

[147]:
```
random.shuffle(stocks)
sampled_stocks = stocks[:25]
factory = MarketFactory(sampled_stocks, "2019-01-01", "2021-04-30")
```

[149]:
```
market = factory.create_market()
market.plot_efficient_frontier()
```

```
[**********************100%**********************]  25 of 25 completed
```

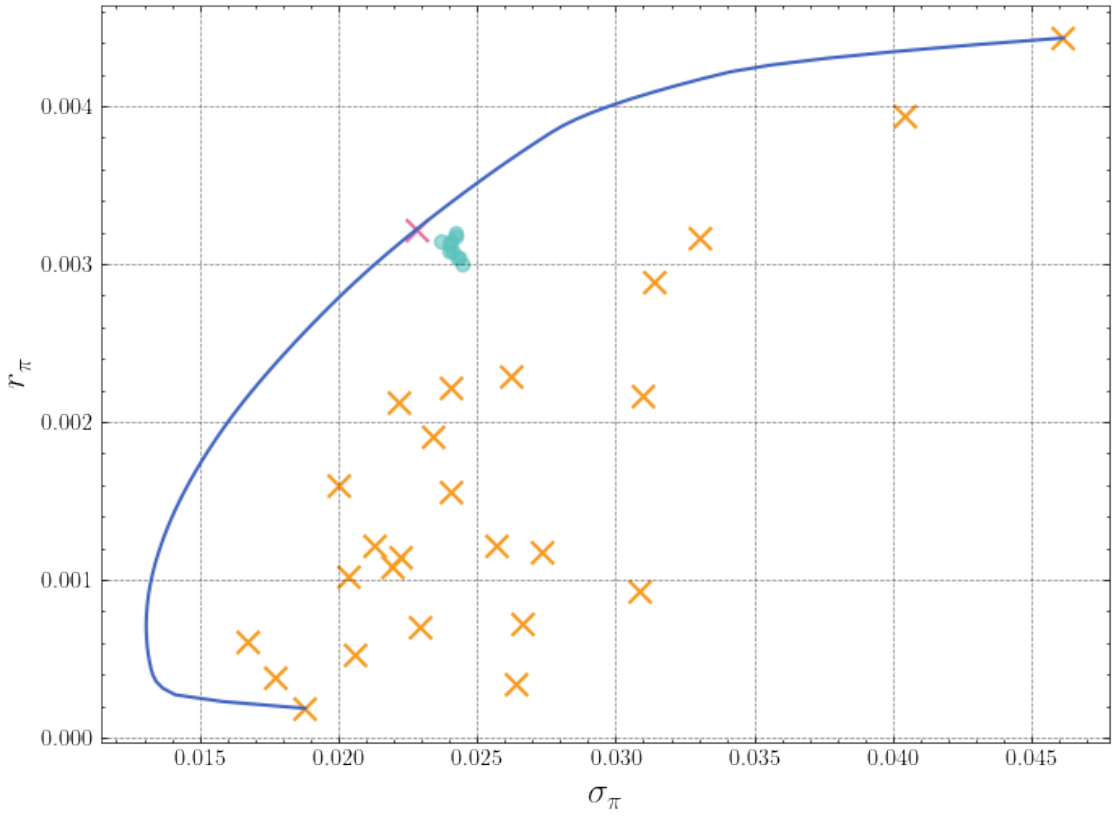


[150]:
```
params_1 = {
    'n_assets': 25,
    ' ': 0.005,
    ' _end': 0.001,
    'start_ep': 9999,
    ' _decay_steps': 9999,
    ' ': 0.1,
    'grad_adpt_mode': 'natural_gradient',
    'returns_adpt_mode': None,
```

```
    'parameterisation': 'softplus',
    'eps': 15000
}



legend_labels = {'0': 'DSR'}

params = [params_1]
p_runner = ParallelRunnerVec(market. , market.Σ, 10, params, 100,␣
 ↪legend_labels, reward_mode='dsr',  =0.05)
p_runner.run_test()
```

```
0it [00:00, ?it/s]
100%|      | 10/10 [00:00<00:00, 1784.81it/s]
1it [13:00, 780.39s/it]
```

[151]:
```
means = []
stds = []
for i in range(len(p_runner.results['0'])):
    means.append(np.mean(p_runner.results['0'][i]))
    stds.append(np.std(p_runner.results['0'][i]))
market.plot_efficient_frontier(stds=stds, returns=means)
```

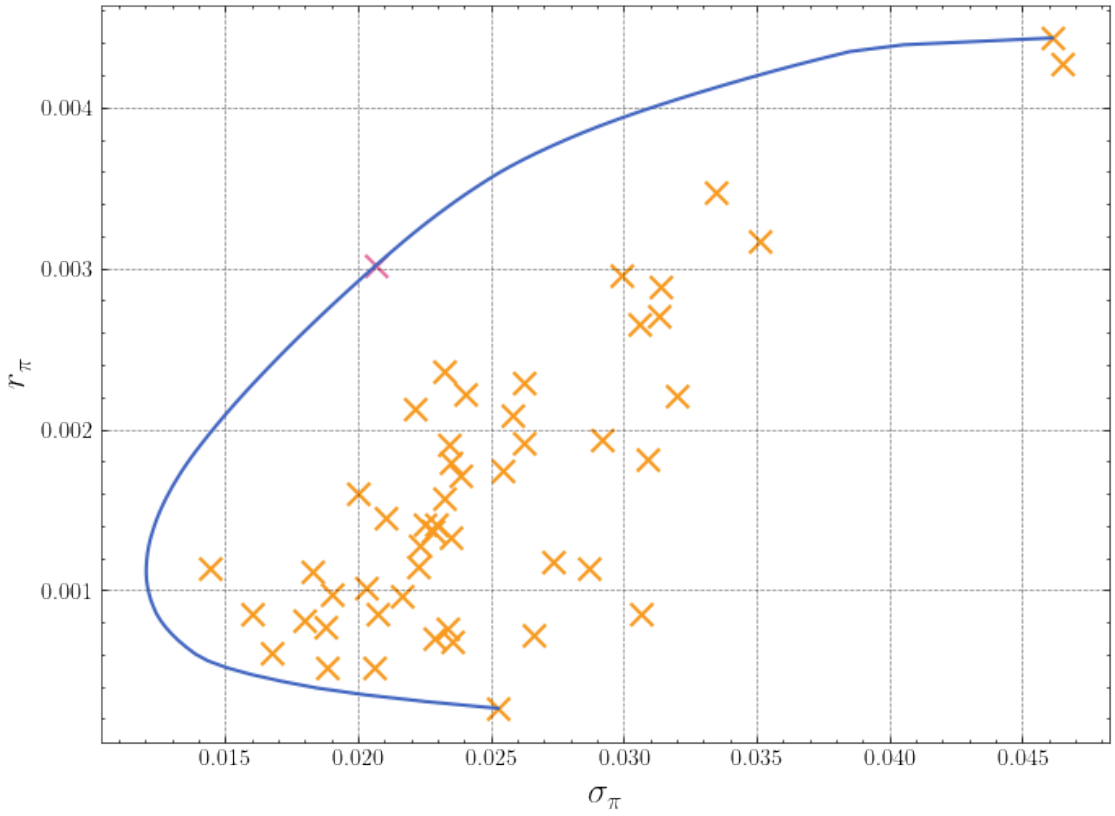


1.2.4 50 Stocks

[152]:
```
random.shuffle(stocks)
sampled_stocks = stocks[:50]
```

[153]:
```
factory = MarketFactory(sampled_stocks, "2019-01-01", "2021-04-30")
market = factory.create_market()
market.plot_efficient_frontier()
```

```
[**********************100%**********************]  50 of 50 completed
```

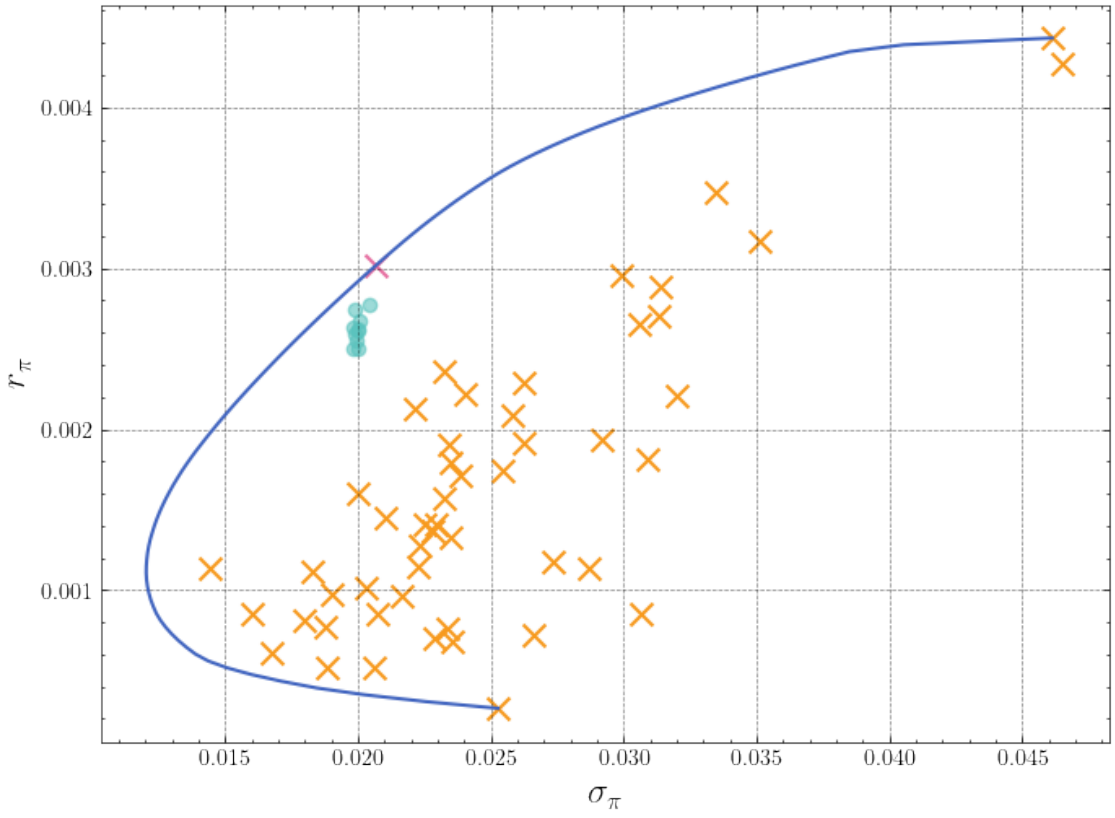


[155]:
```
params_1 = {
    'n_assets': 50,
    ' ': 0.005,
    ' _end': 0.001,
    'start_ep': 9999,
    ' _decay_steps': 9999,
    ' ': 0.1,
    'grad_adpt_mode': 'natural_gradient',
    'returns_adpt_mode': None,
```

```
    'parameterisation': 'softplus',
    'eps': 20000
}



legend_labels = {'0': 'DSR'}

params = [params_1]
p_runner = ParallelRunnerVec(market. , market.Σ, 10, params, 100,␣
 ↪legend_labels, reward_mode='dsr',  =0.05)
p_runner.run_test()
```

```
0it [00:00, ?it/s]
  0%|          | 0/10 [00:00<?, ?it/s]
100%|      | 10/10 [00:00<00:00, 58.32it/s]
1it [26:01, 1561.05s/it]
```

[156]:
```
means = []
stds = []
for i in range(len(p_runner.results['0'])):
    means.append(np.mean(p_runner.results['0'][i]))
    stds.append(np.std(p_runner.results['0'][i]))
market.plot_efficient_frontier(stds=stds, returns=means)
```

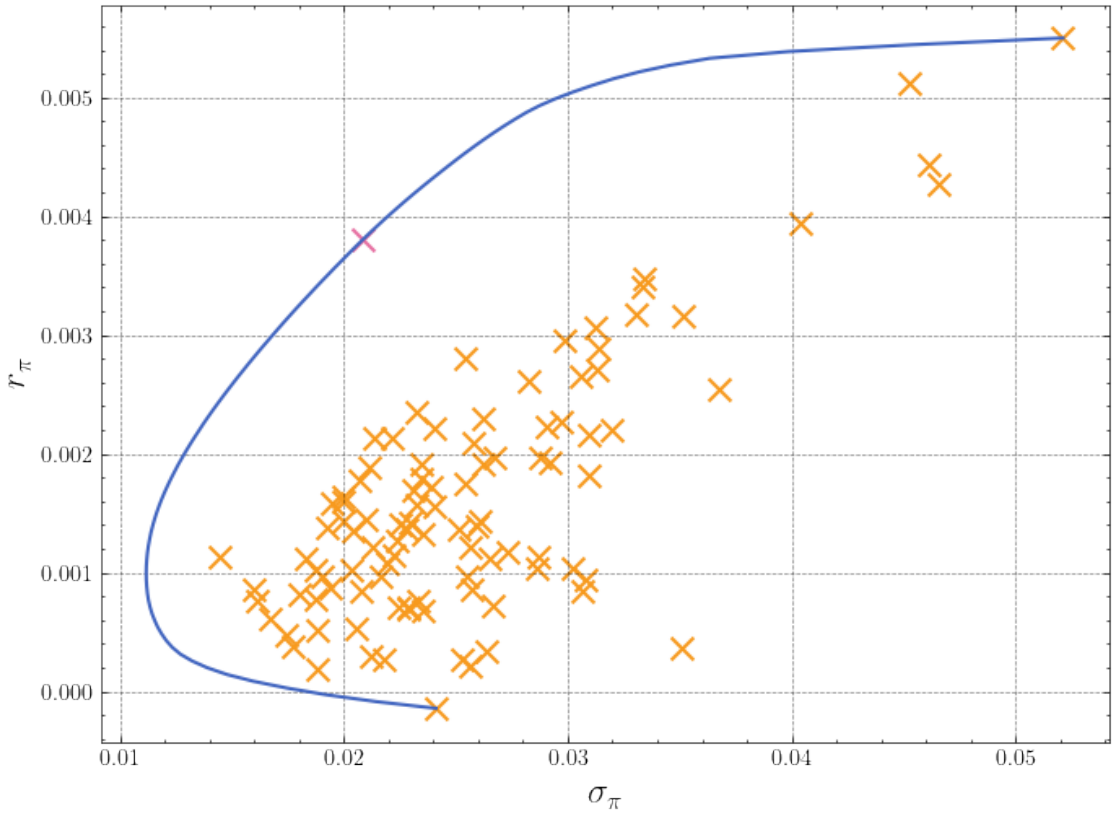


1.2.5 100 Stocks

[157]:
```
random.shuffle(stocks)
sampled_stocks = stocks[:100]
```

[158]:
```
factory = MarketFactory(sampled_stocks, "2019-01-01", "2021-04-30")
market = factory.create_market()
market.plot_efficient_frontier()
```

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



[161]:
```
params_1 = {
    'n_assets': 100,
    ' ': 0.02,
    ' _end': 0.001,
    'start_ep': 6000,
    ' _decay_steps': 6000,
    ' ': 0.1,
    'grad_adpt_mode': 'natural_gradient',
```

```
    'returns_adpt_mode': None,
    'parameterisation': 'softplus',
    'eps': 20000
}



legend_labels = {'0': 'DSR'}

params = [params_1]
p_runner = ParallelRunnerVec(market. , market.Σ, 10, params, 100,␣
 ↪legend_labels, reward_mode='dsr', =0.05)
p_runner.run_test()
```

```
0it [00:00, ?it/s]
100%|      | 10/10 [00:00<00:00, 1746.75it/s]
1it [1:12:43, 4363.90s/it]
```

[162]:
```
means = []
stds = []
for i in range(len(p_runner.results['0'])):
    means.append(np.mean(p_runner.results['0'][i]))
    stds.append(np.std(p_runner.results['0'][i]))
market.plot_efficient_frontier(stds=stds, returns=means)
```