Hwk1

February 12, 2023

1 Math 584 - Homework 1

Diane PERES

1.0.1 Importation

[30]:
```
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('pdf', 'svg')
```

[ ]:
```
import pandas as pd
import numpy as np
import csv
import matplotlib.pyplot as plt
import math
!pip install yfinance
import yfinance as yf
import scipy.optimize
from scipy import stats
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting yfinance
  Downloading yfinance-0.2.11-py2.py3-none-any.whl (59 kB)
                             59.2/59.2 KB
1.8 MB/s eta 0:00:00
Requirement already satisfied: pytz>=2022.5 in
/usr/local/lib/python3.8/dist-packages (from yfinance) (2022.7.1)
Collecting frozendict>=2.3.4
  Downloading
frozendict-2.3.4-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (110
kB)
                            111.0/111.0
KB 4.1 MB/s eta 0:00:00
Collecting cryptography>=3.3.2
  Downloading cryptography-39.0.1-cp36-abi3-manylinux_2_28_x86_64.whl (4.2 MB)
                             4.2/4.2 MB
```

```
16.6 MB/s eta 0:00:00
Collecting beautifulsoup4>=4.11.1
  Downloading beautifulsoup4-4.11.2-py3-none-any.whl (129 kB)
                             129.4/129.4
KB 9.4 MB/s eta 0:00:00
Collecting html5lib>=1.1
  Downloading html5lib-1.1-py2.py3-none-any.whl (112 kB)
                             112.2/112.2
KB 7.1 MB/s eta 0:00:00
Requirement already satisfied: pandas>=1.3.0 in
/usr/local/lib/python3.8/dist-packages (from yfinance) (1.3.5)
Requirement already satisfied: multitasking>=0.0.7 in
/usr/local/lib/python3.8/dist-packages (from yfinance) (0.0.11)
Collecting requests>=2.26
  Downloading requests-2.28.2-py3-none-any.whl (62 kB)
                             62.8/62.8 KB
3.2 MB/s eta 0:00:00
Requirement already satisfied: appdirs>=1.4.4 in
/usr/local/lib/python3.8/dist-packages (from yfinance) (1.4.4)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.8/dist-
packages (from yfinance) (1.21.6)
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.8/dist-
packages (from yfinance) (4.9.2)
Collecting soupsieve>1.2
  Downloading soupsieve-2.3.2.post1-py3-none-any.whl (37 kB)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.8/dist-
packages (from cryptography>=3.3.2->yfinance) (1.15.1)
Requirement already satisfied: webencodings in /usr/local/lib/python3.8/dist-
packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.8/dist-
packages (from html5lib>=1.1->yfinance) (1.15.0)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.8/dist-packages (from pandas>=1.3.0->yfinance) (2.8.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.8/dist-packages (from requests>=2.26->yfinance) (2.1.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.8/dist-
packages (from requests>=2.26->yfinance) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.8/dist-packages (from requests>=2.26->yfinance)
(2022.12.7)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/usr/local/lib/python3.8/dist-packages (from requests>=2.26->yfinance) (1.24.3)
Requirement already satisfied: pycparser in /usr/local/lib/python3.8/dist-
packages (from cffi>=1.12->cryptography>=3.3.2->yfinance) (2.21)
Installing collected packages: soupsieve, requests, html5lib, frozendict,
cryptography, beautifulsoup4, yfinance
```

```
  Attempting uninstall: requests
    Found existing installation: requests 2.25.1
    Uninstalling requests-2.25.1:
      Successfully uninstalled requests-2.25.1
  Attempting uninstall: html5lib
    Found existing installation: html5lib 1.0.1
    Uninstalling html5lib-1.0.1:
      Successfully uninstalled html5lib-1.0.1
  Attempting uninstall: beautifulsoup4
    Found existing installation: beautifulsoup4 4.6.3
    Uninstalling beautifulsoup4-4.6.3:
      Successfully uninstalled beautifulsoup4-4.6.3
Successfully installed beautifulsoup4-4.11.2 cryptography-39.0.1
frozendict-2.3.4 html5lib-1.1 requests-2.28.2 soupsieve-2.3.2.post1
yfinance-0.2.11
```

[ ]:
```
from google.colab import files


uploaded = files.upload()
```

```
<IPython.core.display.HTML object>


Saving TechTickers.csv to TechTickers.csv
```

1.1 1. Construct the efficient frontier

1.1.1 (a) Produce the estimated vector of mean returns (sample mean returns) and the estimated covariance matrix (sample covariance matrix) and print them.

[ ]:
```
#read list of tickers from a csv file
tickers_file = 'TechTickers.csv'
tickers = []
f = open(tickers_file,"r",encoding='utf-8-sig')
for line in csv.reader(f):
    tickers.append(str(line[0]))
f.close
#print(tickers)
tickers_str = tickers[0]
for s in tickers[1:]: tickers_str=tickers_str+", "+s
print(tickers_str)
```

```
AAPL, MSFT, TSM, INTC, CSCO, ORCL, SAP, ADBE, CRM, NVDA, ACN, ASML, AVGO, TXN,
IBM, QCOM, FIS, INTU, MU, VMW, AMAT, AMD, NOW, ADI, LRCX, INFY, ADSK, WDAY,
NXPI, CTSH, TEL, APH, HPQ, CAJ, ERIC, MSI, KLAC, MCHP, FLT, SPLK, STM, VRSN,
GLW, PANW, GIB, ANSS, SNPS, WDC, SWKS, CDNS, FTNT, GRMN, MRVL, CHKP, STX, SSNC,
AKAM, NTAP, BR, IT, LDOS, ZBRA, TDY, EPAM, OTEX, TYL, UI, TER, FICO, JKHY, ASX
```

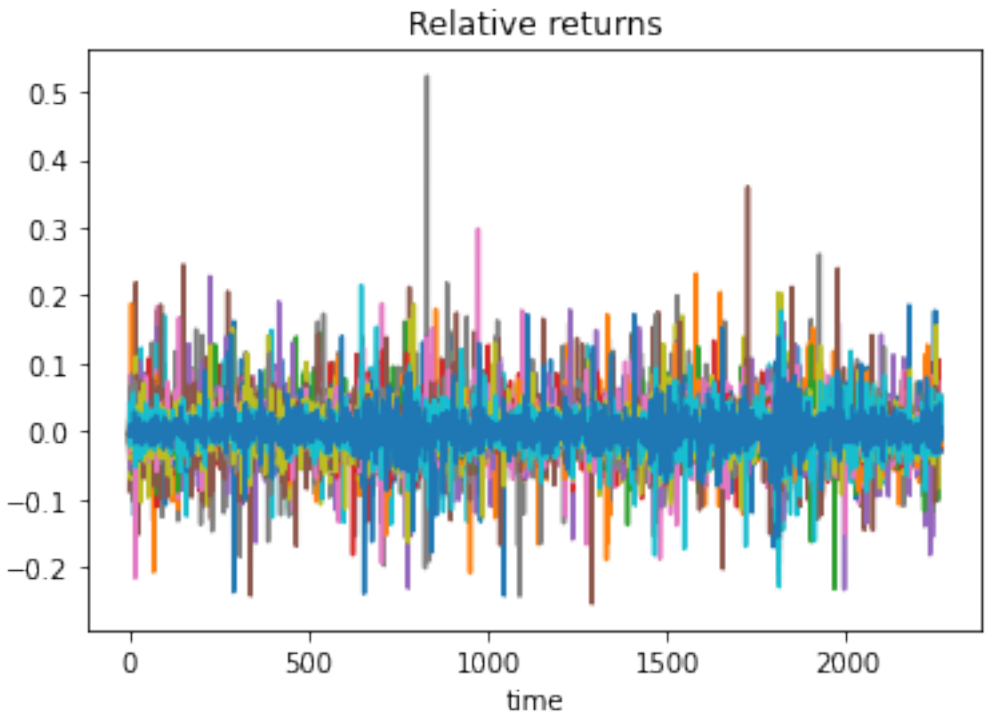
[ ]:
```
#downoad the prices and volumes for the previously read list of tickers for the␣
 ↪first month
#of the earliest year in the proposed time period
start_date = '2013-01-01'
end_date = '2022-01-01'
stock_data = yf.download(tickers_str, start_date, end_date)
```

```
[**********100%**********]  71 of 71 completed
```

[ ]:
```
price = stock_data['Adj Close'].values
#print(price)
days = price.shape[0]
#print(days)
ret = price[1:]/price[:-1]-1 # relative return
plt.plot(ret)
plt.xlabel('time')
plt.ylabel('')
plt.title("Relative returns")

 =np.mean(ret,axis=0)*250 #Annualize 250 days
Σ = np.cov(ret.T)*250 #Annualize 250 days
#print(len( ))
#print(len(Σ))

print(f"The sample mean returns is: \n{ } \n\nThe sample covariance matrix is:␣
 ↪\n{Σ}\n\n")
```

```
The sample mean returns is:
[ 0.2988334   0.24217943  0.34078131  0.21809913  0.28281326  0.1651697
  0.36765375  0.60947123  0.22839646  0.21554279  0.33329209  0.16009747
  0.41838944  0.26907465 -0.02727204  0.32591689  0.12038273  0.25086726
  0.18632112  0.13868407  0.45890831  0.08243921  0.30398148  0.16628163
  0.20221353  0.38081411  0.17605972  0.1829825   0.19943957  0.2738928
  0.0277145   0.23738566  0.17015982  0.30397667  0.25335224  0.18946906
  0.34640359  0.23350236  0.40398077  0.26027304  0.3553763   0.32956413
  0.22121347  0.39374777  0.41520682  0.18489638  0.59375718  0.3117676
  0.14684693  0.18266239  0.33300609  0.19659292  0.10562722  0.29953021
  0.25520573  0.2657734   0.31385925  0.2620913   0.30154406  0.24231486
  0.21144587  0.31461803  0.27832995  0.25618169  0.29961281  0.4674126
  0.12124445  0.23424964  0.25376506  0.15906235  0.36221141]

The sample covariance matrix is:
[[0.07900915 0.03055138 0.04328359 … 0.04197204 0.04422554 0.04195569]
 [0.03055138 0.0514453  0.0369978  … 0.03894735 0.04158941 0.0364286 ]
 [0.04328359 0.0369978  0.0870645  … 0.06482709 0.04618457 0.04359317]
 …
 [0.04197204 0.03894735 0.06482709 … 0.14518421 0.05819681 0.04910807]
```

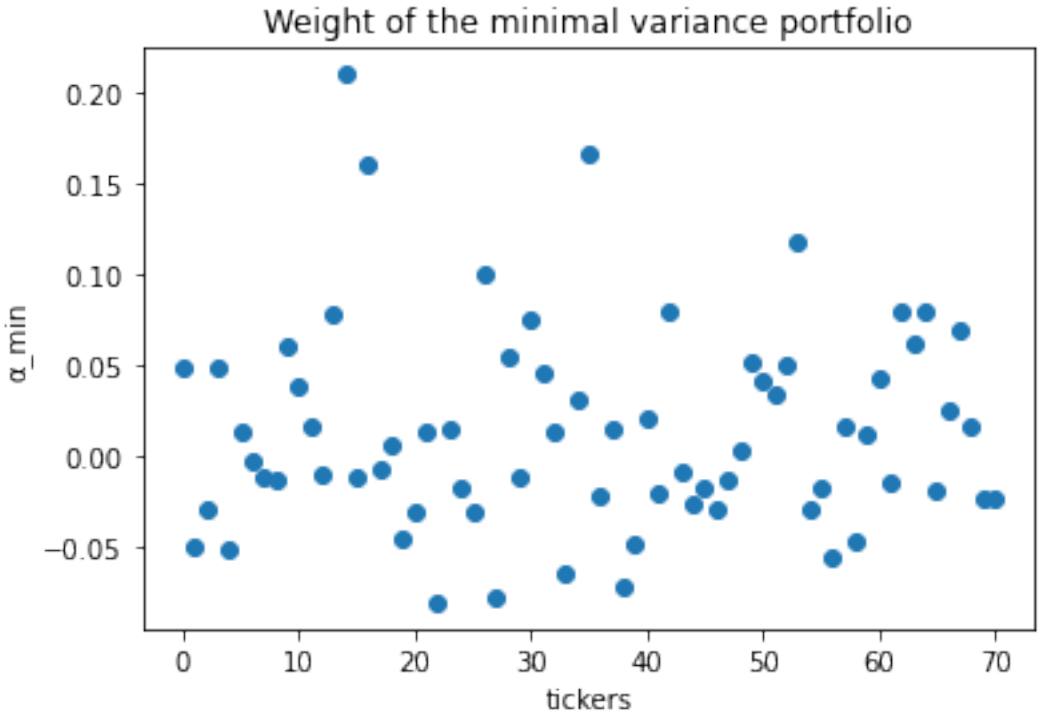
```
[0.04422554 0.04158941 0.04618457 … 0.05819681 0.18074125 0.0627844 ]
[0.04195569 0.0364286  0.04359317 … 0.04910807 0.0627844  0.13278143]]
```



###(b) Compute the **weights of the minimal-variance portfolio** and print them.

[ ]:
```python
d = len( ) #71

#  Weight vector of the minimal-variance portfolio
 _min = np.linalg.solve(Σ,np.ones(d))
 _min =  _min/np.sum( _min)

print(f"The weight of the minimal variance portfolio are: \n{ _min}\n\n")
plt.plot( _min,'o')
plt.title("Weight of the minimal variance portfolio")
plt.xlabel('tickers')
plt.ylabel(' _min')

#Verification [ _1 + ... +  _d] = 1
sum = 0
for weight in  _min:
  sum += weight
#print(sum)
```

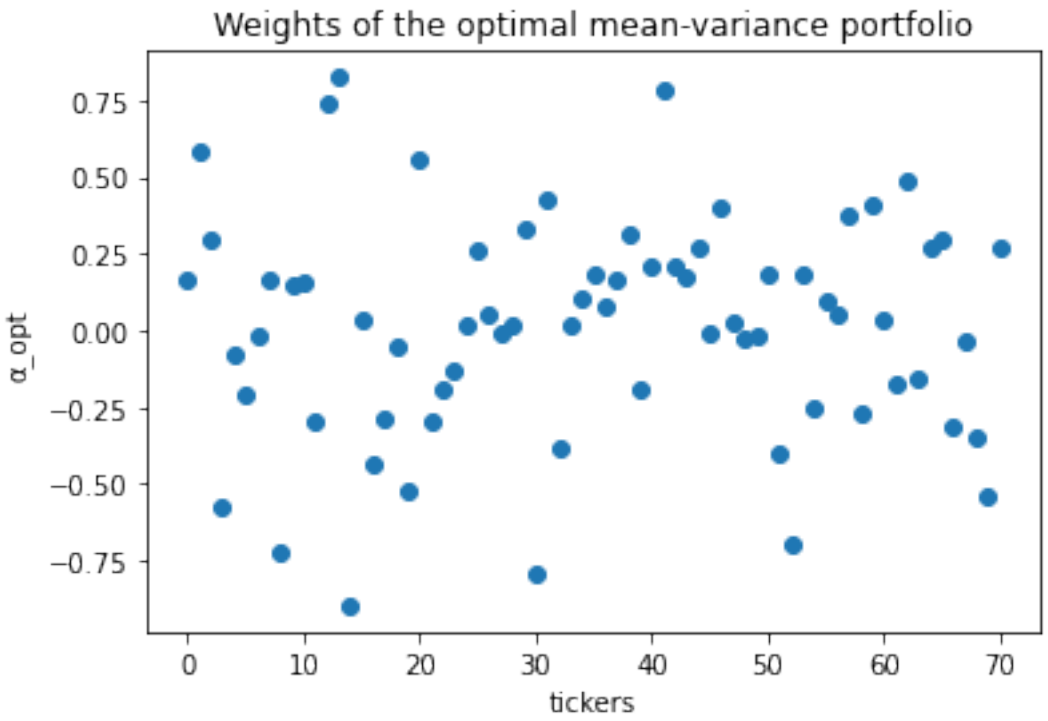
```
The weight of the minimal variance portfolio are:
[ 0.04812238 -0.0498386  -0.02914923  0.04865735 -0.05042467  0.01407984
 -0.00186189 -0.0112087  -0.01284492  0.06110579  0.03811024  0.01726155
 -0.00979846  0.07748428  0.20966522 -0.01071435  0.16090872 -0.00643053
  0.00600005 -0.0446735  -0.02999198  0.01425471 -0.0800125   0.01443178
 -0.01779334 -0.0296215   0.1006083  -0.07721014  0.0545535  -0.01158856
  0.07462415  0.04576906  0.01379218 -0.06377478  0.03118504  0.16546161
 -0.02168277  0.01543671 -0.07125158 -0.0480575   0.0216941  -0.01966085
  0.07997105 -0.00876485 -0.02560725 -0.01685382 -0.02868129 -0.01215976
  0.00344968  0.05127123  0.04131261  0.03412762  0.0507097   0.11709913
 -0.02926702 -0.01749473 -0.05538187  0.01660657 -0.04698065  0.01223119
  0.04361302 -0.01491723  0.07991888  0.06272274  0.07958678 -0.01809032
  0.026078    0.06941279  0.01630673 -0.02321019 -0.02262496]
```



###(c) Compute the **weights of the optimal mean-variance portfolio** (i.e., maximizing a linear combination of mean and variance) with the coefficient of risk aversion = 2. Plot the portfolio weights on a graph. Print the mean and the standard deviation of the resulting portfolio.

[ ]:
```
# Coefficient of risk aversion
 = 2

# The function to minimized
```

```
my_fun = lambda x: -x@  +  *(x@(x@Σ))

# Non-linear constraint
def my_constr(x):
    return (np.sum(x) - 1) # [_1 + ... + _d] - 1 = 0

constr = {'type': 'eq', #type str: Constraint type ‘eq’ for equality - Equality␣
 ↪constraint means that the constraint function result is to be zero
          'fun': my_constr} #fun callable: The function defining the constraint

# Optimal mean-variance portfolio
opt = scipy.optimize.minimize(my_fun,  _min, constraints=constr,␣
 ↪options={'maxiter':1e6})
#print(opt)

#  Weight vector of the optimal mean-variance portfolio
 _opt = opt.x

plt.plot( _opt,'o')
plt.title('Weights of the optimal mean-variance portfolio')
plt.xlabel('tickers')
plt.ylabel(' _opt')
plt.show()

# Mean
 _opt =  _opt@

# Standard deviation
std_opt = np.sqrt( _opt@( _opt@Σ)) #do not use the np.std because it add other␣
 ↪errors

print(f"For the optimal mean-variance portfolio \nThe sample mean returns is:␣
 ↪{ _opt}\nThe standard deviation is: {std_opt}\n")
```




```
For the optimal mean-variance portfolio
The sample mean returns is: 2.0020774916029276
The standard deviation is: 0.7076916624675318
```

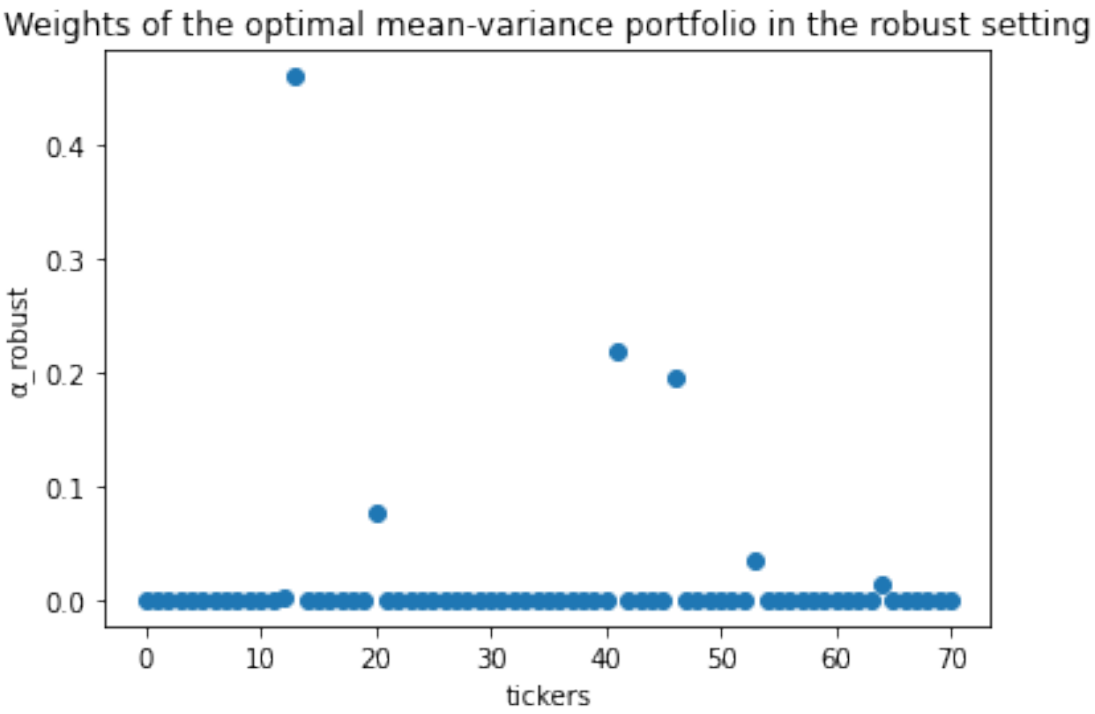
RESULTS of scipy.optimize.minimize

fun, jac: ndarray Values of objective function and its Jacobian
nfev, njev : int Number of evaluations of the objective functions and of its Jacobian
nit: int Number of iterations performed by the optimizer

###(d) Compute the **weights of the optimal mean-variance portfolio in the robust setting**, assuming that the true mean returns of the basic assets are within one **standard deviation** (the standard deviation is estimated from the sample) away from their sample means. Plot the weights of the resulting optimal portfolio and compare this graph to the one produced in part (c). Print the standard deviation of the optimal portfolio return, as well as its worst- and best-case mean return (according to the chosen intervals of possible mean returns of the basic assets). Compare these means and the standard deviation to those produced in part (c) and explain the difference between the two results.

[ ]:
```
# half size of  interval
  = np.sqrt(np.diagonal(Σ))

# The function to minimized
my_fun_robust = lambda x: - @x +  @abs(x) +  *(x@(x@Σ))
```

```
# Optimal mean-variance portfolio in the robust setting
opt_robust = scipy.optimize.minimize(my_fun_robust,  _min, constraints=constr,␣
 ↪options={'maxiter':1e6})
# print(opt_robust)

#  Weight vector of the optimal mean-variance portfolio
 _robust = opt_robust.x
plt.plot( _robust,'o')
plt.title('Weights of the optimal mean-variance portfolio in the robust␣
 ↪setting')
plt.xlabel('tickers')
plt.ylabel(' _robust')
plt.show()

print("The weights of the optimal mean-variance portfolio in the robust setting␣
 ↪is closer to 0 than the same weights without the robust setting.\n")

# Mean
 _robust =  _robust@

# Standard deviation
#std_robust = np.std((ret- _robust)* _robust)
std_robust = np.sqrt( _robust@( _robustΣ))

print(f"For the optimal mean-variance portfolio in a robust setting \nThe␣
 ↪sample mean returns is: { _robust}\nThe standard deviation is:␣
 ↪{std_robust}\n")
```




```
The weights of the optimal mean-variance portfolio in the robust setting is
closer to 0 than the same weights without the robust setting.

For the optimal mean-variance portfolio in a robust setting
The sample mean returns is: 0.3619582807762512
The standard deviation is: 0.21886728259277702
```

###(e) Compute the **efficient frontier** and plot it as a set **{(f( ),)}**, where changes over a grid of 100 equidistant points in [0, 2], and **f( ) is the standard deviation** of the efficient portfolio with **mean return** . On the same plot, show the pairs **(√Σii, i)** corresponding to the standard deviations and the means of the returns of individual basic assets. Comment on where the latter pairs lie relative to the efficient frontier and why.

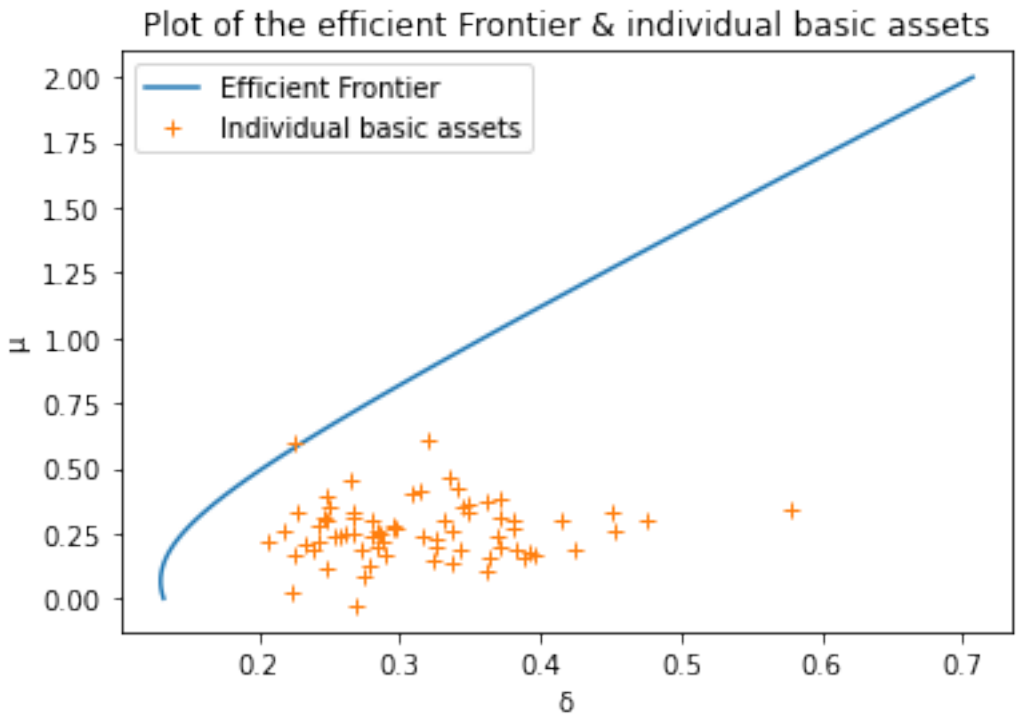
[ ]:
```
# EFFICIENT PORTFOLIO

# Initialize an array of target returns
_target = np.linspace(0,2,100) #there will be 100 portfolios
R = 0.01

_target = [] #f(_target): standard deviation
#Σ_target = np.zeros(d+2)

#Setting up covariance matrix
```

```
Σ_target = np.zeros((73,73))
Σ_target[:71,:71] = 2*Σ
Σ_target[:71,71] = -1*np.ones(71)
Σ_target[:71,72] = -1*
Σ_target[71,:71] = np.ones(71)
Σ_target[72,:71] =

#Setting up b vector
b = np.zeros(73)
b[-2] = 1

for mu in  _target:

  b[-1]= mu

    = np.linalg.solve(Σ_target,b)[:-2]
    =   /np.sum( ) #normalize

  #Find the standard deviation
   _target.append(np.sqrt( .T @ Σ @  )) #f( _target)

print (f'The list of f( ) is: \n{ _target}\n\n')
```

```
The list of f( ) is:
[0.13235849759362184, 0.13124135976301549, 0.1305191306051892,
0.13019838213168514, 0.13028207965705366, 0.13076944660015263,
0.13165600010154552, 0.1329337537108272, 0.13459156623967294,
0.13661560249228627, 0.13898986401927732, 0.1416967468580303, 0.144717587578307,
0.1480331670711192, 0.15162415132824833, 0.15547145813042954,
0.15955654682487008, 0.16386163458978661, 0.1683698466486242,
0.17306531005608955, 0.17793320135214863, 0.18295975802433403,
0.18813226273955827, 0.19343900802313427, 0.19886924769990685,
0.2044131401127734, 0.21006168697572258, 0.2158066707349273, 0.2216405925063927,
0.22755661201942928, 0.23354849050084237, 0.23961053706164034,
0.2457375588732787, 0.2519248152233383, 0.25816797540344344, 0.2644630802906096,
0.2708065074254431, 0.2771949393575223, 0.2836253350128636, 0.29009490383524916,
0.296601082458393, 0.3031415136765059, 0.3097140274946562, 0.31631662405590194,
0.3229474582584135, 0.3296048258920086, 0.33628715113919744, 0.3429929753006884,
0.34972094661916375, 0.35646981108794396, 0.3632384041428295,
0.37002564314607916, 0.376830520581062, 0.3836520978847932, 0.3904894998533212,
0.3973419095618733, 0.4042085637478989, 0.41108874861067063, 0.4179817959860448,
0.42488707985938695, 0.4318040131835731, 0.438732044972471, 0.4456706576433968,
0.4526193645848066, 0.45957770792793606, 0.46654525650328993,
0.4735216039648365, 0.4805063670664794, 0.4874991840769648, 0.49449971332071635,
0.5015076318333752, 0.5085226341218786, 0.5155444310199386, 0.5225727486306342,
0.5296073273486348, 0.5366479209552911, 0.543694295780445, 0.5507462299253952,
0.5578035125419731, 0.5648659431631321, 0.5719333310808795, 0.5790054947677543,
```

```
0.5860822613383869, 0.5931634660479929, 0.60024895182491, 0.6073385688345699,
0.6144321740724743, 0.6215296309840078, 0.6286308091090435, 0.6357355837495251,
0.642843835658314, 0.6499554507477572, 0.6570703198165577, 0.6641883382936139,
0.6713094059976465, 0.6784334269114998, 0.685560308970066, 0.6926899638609383,
0.6998223068368784, 0.7069572565393146]
```

[ ]:
```python
# INDIVIDUAL BASIC ASSETS

# Standard deviation
 _ind = np.sqrt(np.diag(Σ))
#print(Σ_ind)

# Mean
 _ind = []
for i in range(len(tickers)):
  price_ind = stock_data['Adj Close'][tickers[i]].values
  ret_ind = price_ind[1:]/price_ind[:-1]-1 # relative return
   _ind.append(np.mean(ret_ind,axis=0)*250) #Annualize

#print (f'The list of   is: \n{ _ind}\n\nThe list of f( ) is: \n{ _ind}\n\n')

plt.plot( _target, _target, label='Efficient Frontier')
plt.plot( _ind, _ind,'+', label='Individual basic assets')
plt.title('Plot of the efficient Frontier & individual basic assets')
plt.xlabel(' ')
plt.ylabel(' ')
plt.legend()
plt.show()
print('Individual basic assets are below the frontier.')
```



Individual basic assets are below the frontier.

###(f) Add a **riskless asset** to the set of available ones. Compute the **weights** of the market portfolio (i.e., of the optimal mutual fund), as well as the **mean**, **standard deviation** and **Sharpe ratio** of its return, and print them. Compute the efficient frontier for the extended market and plot it in the same coordinates and for the same values of as in part (e). Plot the efficient frontier from part (e) on the same graph and comment on the relationship between the two.

[ ]:
```
# MARKET PORTFOLIO - i.e. OPTIMAL MUTUAL FUND

# Weights such as Σ. _M=b
b =   - R
 _M = np.linalg.solve(Σ,b)

if ( _M.sum()!=0) & (b@ _M > 0):

    # Weights
     _M =  _M/np.sum( _M)
     _M[0] = 1 #initialise with the risky asset
    print(f"Market portfolio exist.\n\n")

    # Mean
     _M= @ _M
    print(f"The mean of the Market portfolio is { _M}\n\n")
```

```
    # Standard deviation
     _M = np.sqrt( _M@( _M@Σ))
    print(f"The standard deviation of the Market portfolio is { _M}\n\n")

    # Sharp ratio
    S = ( _M-R)/ _M
    print(f"The sharp ratio of the Market portfolio is {round(S,3)} ~ 1, which␣
↪means that it is a good stock\n\n")

else:
    print("market portfolio does not exist")
```

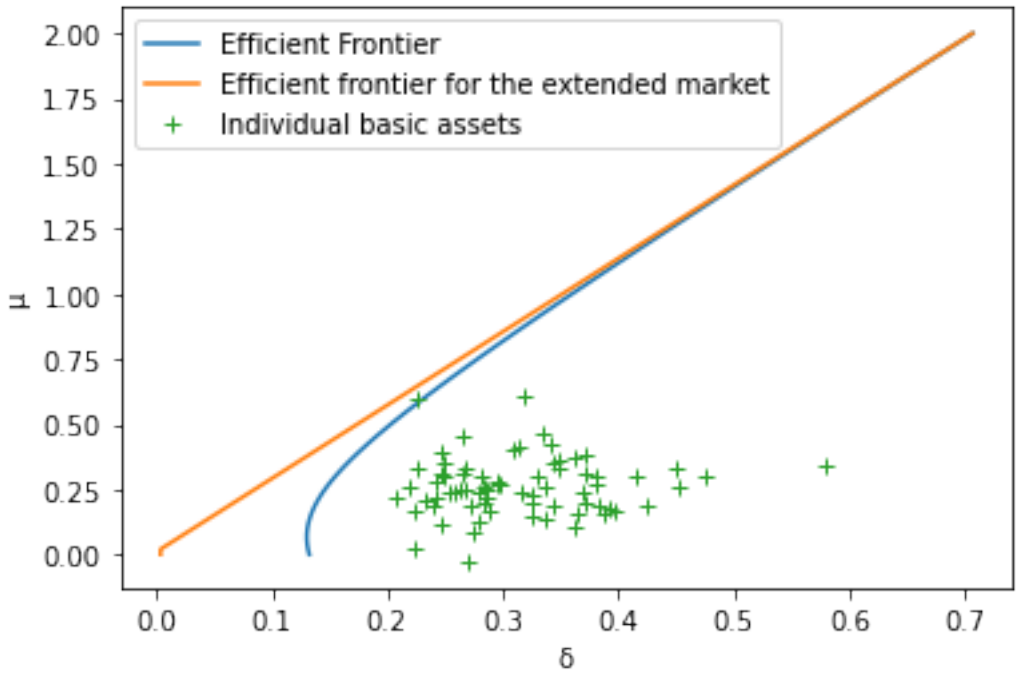
```
Market portfolio exist.


The mean of the Market portfolio is 2.628550526725329


The standard deviation of the Market portfolio is 0.9506720510327086


The sharp ratio of the Market portfolio is 2.754 ~ 1, which means that it is a
good stock
```

[ ]:
```
# EFFICIENT FRONTIER FOR THE EXTENDED MARKET

# Riskless asset
R = 0.01 #risk free
 _rl = 0 #std
 _rl = 1  #weight

#Setting up covariance matrix
Σ_ext_M2 = np.zeros((72,72))
Σ_ext_M2[1:,1:]=Σ

Σ_ext_M = np.zeros((74,74))
Σ_ext_M[:72,:72] = 2*Σ_ext_M2
Σ_ext_M[:72,72] = -1*np.ones(72)
Σ_ext_M[0,73] = - R
Σ_ext_M[1:72,73] = -1*
Σ_ext_M[72,:72] = np.ones(72)
Σ_ext_M[73,0] = R
Σ_ext_M[73,1:72] =
```

```
#Setting up b vector
b = np.zeros(74)
b[-2] = 1

 _ext_M  = [] #f( _target)

for mu in  _target:

  b[-1]= mu
    = np.linalg.solve(Σ_ext_M,b)[:-2] # don't need lambdas
    =   /np.sum( ) #normalize

  #Find the standard deviation
   _ext_M.append(np.sqrt( .T @ Σ_ext_M2 @  )) #f( _target)

print (f'The list of f( ) is: \n{ _ext_M}\n\n')

#fig,ax = plt.subplots(figsize = (15,5))
#plt.title("Efficient Frontier Extended",size=15)
plt.plot( _target, _target, label='Efficient Frontier')
plt.plot( _ext_M , _target, label='Efficient frontier for the extended market')
plt.plot( _ind, _ind,'+', label='Individual basic assets')
plt.xlabel(" ")
plt.ylabel(" ")
plt.legend()
plt.show()
print('The efficient frontier for the extended market is tangent to the initial␣
 ↪one.')
```

```
The list of f( ) is:
[0.003550908860802808, 0.003622644393344279, 0.010796197647491363,
0.017969750901638458, 0.025143304155785546, 0.032316857409932634,
0.03949041066407973, 0.04666396391822681, 0.05383751717237392,
0.06101107042652099, 0.0681846236806681, 0.07535817693481518,
0.08253173018896226, 0.08970528344310938, 0.09687883669725644,
0.10405238995140349, 0.11122594320555057, 0.1183994964596978,
0.1255730497138448, 0.13274660296799193, 0.13992015622213894,
0.14709370947628608, 0.15426726273043312, 0.16144081598458027,
0.1686143692387273, 0.17578792249287453, 0.18296147574702146,
0.19013502900116855, 0.19730858225531567, 0.20448213550946276,
0.21165568876360985, 0.2188292420177569, 0.22600279527190406,
0.23317634852605132, 0.24034990178019822, 0.24752345503434525,
0.25469700828849234, 0.2618705615426394, 0.26904411479678664,
0.2762176680509338, 0.28339122130508076, 0.29056477455922775,
0.2977383278133749, 0.3049118810675221, 0.3120854343216692, 0.31925898757581617,
0.3264325408299635, 0.3336060940841103, 0.3407796473382574, 0.34795320059240453,
0.3551267538465515, 0.36230030710069877, 0.36947386035484564,
```

```
0.3766474136089932, 0.38382096686314005, 0.3909945201172871,
0.39816807337143423, 0.4053416266255815, 0.412515179879728, 0.4196887331338751,
0.42686228638802287, 0.4340358396421696, 0.4412093928963166, 0.448382946150464,
0.45555649940461096, 0.4627300526587579, 0.4699036059129054, 0.4770771591670523,
0.4842507124211988, 0.49142426567534603, 0.4985978189294936, 0.5057713721836407,
0.5129449254377877, 0.520118478691935, 0.5272920319460819, 0.5344655852002284,
0.5416391384543761, 0.5488126917085229, 0.5559862449626701, 0.5631597982168172,
0.5703333514709644, 0.5775069047251112, 0.5846804579792586, 0.5918540112334055,
0.5990275644875525, 0.6062011177416998, 0.6133746709958467, 0.6205482242499936,
0.6277217775041412, 0.6348953307582881, 0.6420688840124354, 0.6492424372665825,
0.6564159905207295, 0.6635895437748764, 0.670763097029024, 0.6779366502831704,
0.6851102035373176, 0.6922837567914649, 0.699457310045612, 0.7066308632997595]
```




```
The efficient frontier for the extended market is tangent to the initial one.
```

##2. The regression interpretation of CAPM.

1.1.2 (a) Compute the for each basic asset, according to the CAPM formula (using the part of the formula that expresses beta through the weights of the market portfolio, which you computed in 1.f), and print the results.

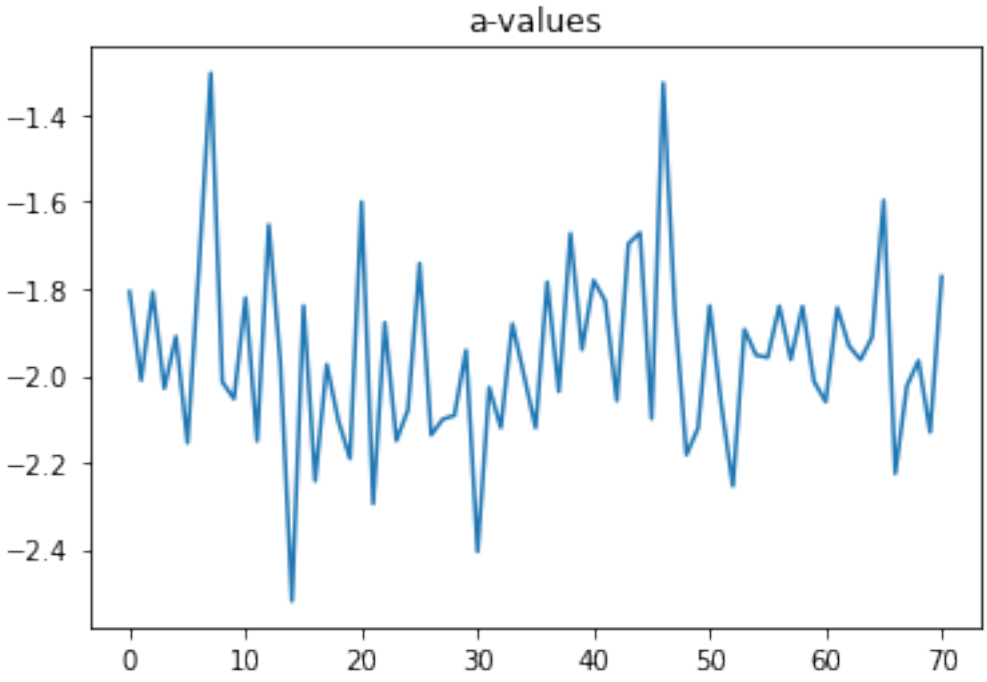
[ ]:
```
#init with the riskless asset as my cov matrix don't have the right length␣
 ↪compare to _M vector
_CAPM=(( _M@Σ)/( _M@( _M@Σ)))
print(f"  for each basic asset (with the riskless asset) is: \n{ _CAPM}\n\n")
```

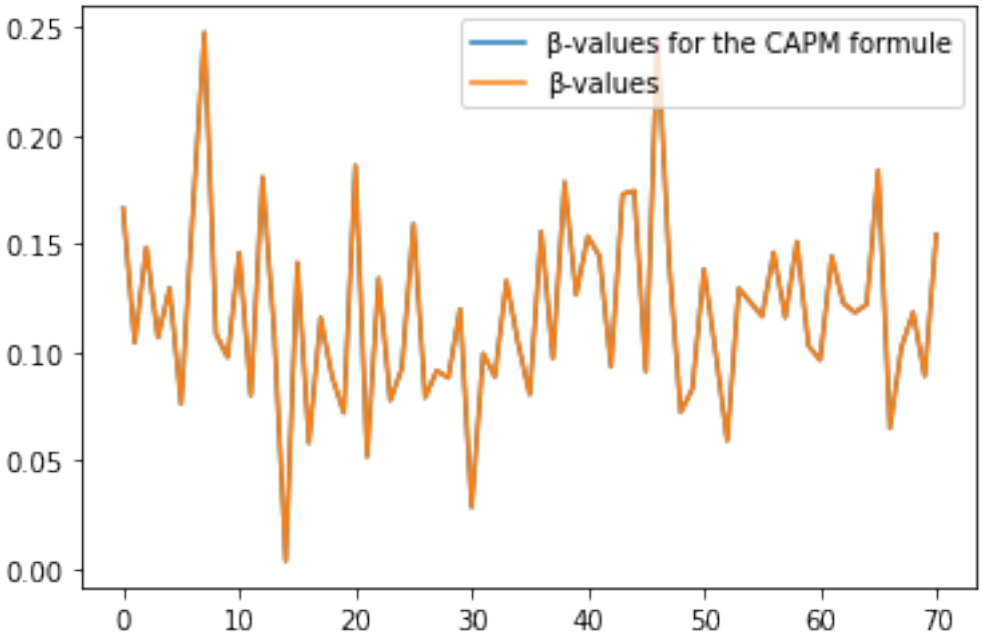
```
  for each basic asset (with the riskless asset) is:
[0.16629251 0.10427099 0.14833657 0.10659206 0.12953983 0.07602502
 0.16464329 0.24753274 0.10831868 0.09752752 0.14588574 0.07990962
 0.18080651 0.10954219 0.0035803  0.14128063 0.05795225 0.11584112
 0.08851156 0.07193807 0.18611095 0.05167717 0.13407493 0.07768836
 0.09243777 0.15921003 0.07901084 0.0914055  0.0880736  0.1198921
 0.02830243 0.09929252 0.08861498 0.1329497  0.10418189 0.08034882
 0.15560505 0.0969585  0.1784838  0.12635805 0.15340701 0.14426132
 0.09330032 0.17286769 0.17448165 0.0910749  0.24455518 0.14186072
 0.07209815 0.08292097 0.13817255 0.09997023 0.05912418 0.12934436
 0.12305952 0.11643494 0.14599352 0.11598496 0.15095094 0.10286953
 0.09634531 0.14417489 0.12251807 0.11822749 0.12190054 0.18380912
 0.06474961 0.10210553 0.11830235 0.0888988  0.15426056]
```

1.1.3 (b) Use the (ordinary least-square) linear regression model, to regress the excess returns of the individual basic assets on the excess return of the market portfolio. Recall that we denote the mean returns of the hedged assets by {ai}_(i=1,...,30) ("hedged" means that we subtract i(R^(M) − R) from the return). Print the resulting {(ai, i)}. Comment on the magnitude of {ai} and compare the resulting { i} to those obtained in part (a).

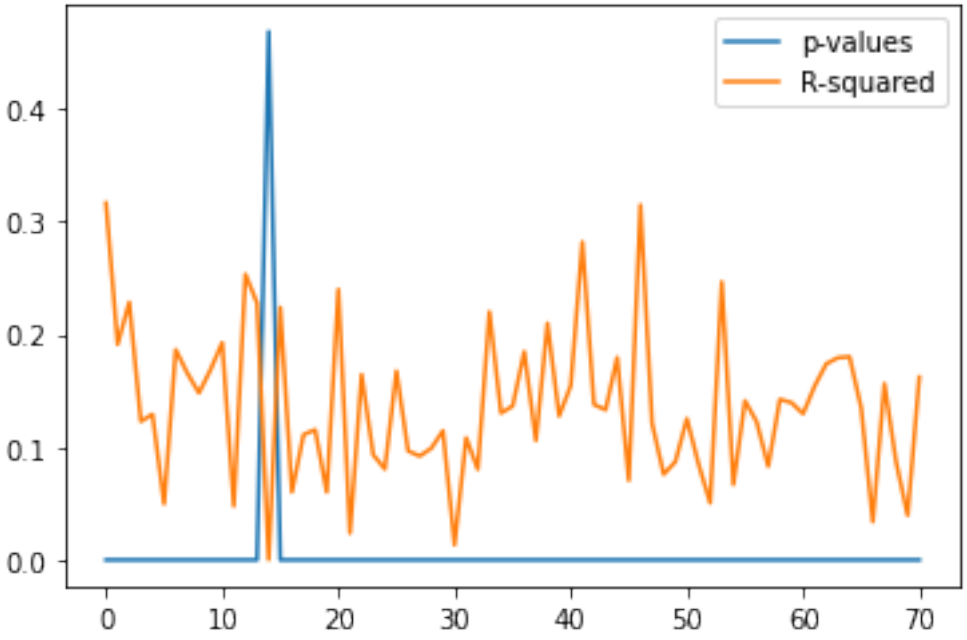
[ ]:
```
 =[]
a=[]
p=[]
r2=[]

# Return for the Market portfolio
ret_M = ret @  _M - R

for i in range(len(ret[0])):

    slope, intercept, r_value, p_value, std_err = stats.linregress(ret_M, ret[:
 ↪,i])
    #   ret: excess returns of the individual basic assets
    # ret_M: excess return of the market portfolio
```

```
     .append(slope)
    a.append((intercept-R+R*slope)*250)
    p.append(p_value)
    r2.append(r_value**2)

plt.plot(a)
plt.title('a-values')
plt.show()


plt.plot( _CAPM,label=' -values for the CAPM formule')
plt.plot( , label=' -values')
plt.legend()
plt.show()
print(' -values are the same \n\n')

plt.plot(p,label='p-values')
plt.plot(r2,label='R-squared')
plt.legend()
plt.show()
print(f'The p-values are low. The measure (1-p) of validity confidence is good.
 ↪\n')
print(f'The R-squared are around 20%. The measure of strenght is good.\n\n')
#print(f'The resulting (a,  ) are {a, }')
```





-values are the same



The p-values are low. The measure (1-p) of validity confidence is good.

The R-squared are around 20%. The measure of strenght is good.

2 Download as PDF

[ ]:
```
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

[39]:
```
!jupyter nbconvert Hwk1.ipynb --stdout
```

```
[NbConvertApp] WARNING | pattern 'Hwk1.ipynb' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
```