Portfolio Team Homework 8

Kyle Chang, Christian Warren, Dev Vadalia

May 4, 2021

Exercise 1

[1]:
```
import yfinance as yf
import pandas as pd
import numpy as np
import matlab.engine
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.lines import Line2D
import matplotlib.pyplot as plt
from cvxopt import matrix, solvers
import math
solvers.options['show_progress'] = False

eng = matlab.engine.start_matlab()
quarters = [('-01-01', '-03-31'), ('-04-01', '-06-30'), ('-07-01', '-09-30'),␣
 ↪('-10-01', '-12-31')]
years = [
    ('2016-01-01','2016-12-31'),
    ('2017-01-01','2017-12-31'),
    ('2018-01-01','2018-12-31'),
    ('2019-01-01','2019-12-31'),
    ('2020-01-01','2020-12-31')
]



mu_si = np.array([0.22, 0.33, 0.30, 0.33, 0.54, 0.63, 0.98, 1.02, 1.44, 1.63, 1.
 ↪91, 2.13, 2.42, 2.44, 2.35, 1.98, 1.54, 1.12, 0.14, 0.12])/100
mu_cl = np.add(mu_si, [.03]*20)
mu_si = mu_si/252
mu_cl = mu_cl/252

mu_safe = {}
mu_credit = {}
rolling_quarter_yearly = []
```

```
for year_index, year in enumerate(years):
    cur_year = year[0][:4]
    for quarter_index in range(4):

        if cur_year != '2020' or (cur_year == '2020' and quarter_index == 0):
            period_start = cur_year + quarters[quarter_index][0]
            period_end = str(int(cur_year) + 1) + quarters[quarter_index-1][1]␣
 ↪if quarter_index != 0 else cur_year + quarters[quarter_index-1][1]
            rolling_quarter_yearly.append((period_start, period_end))

        mu_safe[cur_year + ' Q' + str(quarter_index + 1)] = mu_si[year_index*4 +␣
 ↪quarter_index]
        mu_credit[cur_year + ' Q' + str(quarter_index + 1)] = mu_cl[year_index*4␣
 ↪+ quarter_index]

mu_si = mu_safe
mu_cl = mu_credit

# assets = ['VFIAX', 'VBTLX', 'VGSLX', 'VIMAX', 'VSMAX', 'VGHCX', 'AMZN', 'WMT',␣
 ↪'CVS']
# df = yf.download(assets, '2015-12-31', '2021-01-05')['Adj Close'].
 ↪reindex(columns=assets)
# returns = (df - df.shift(1))/df.shift(1)
```

[2]:
```
# Tested and returns same as matlab
def compute_y(var):
    ones = eng.transpose(matlab.double([1]*len(var)))
    return eng.mldivide(matlab.double(var.values.tolist()), matlab.double(ones)).
 ↪_data.tolist()

# Tested and returns same as matlab
def compute_z(mean, variance):
    return eng.mldivide(matlab.double(variance.values.tolist()), eng.
 ↪transpose(matlab.double(mean.values.tolist())))._data.tolist()

# np.dot values return the same as matlab for test data
def compute_abc(mean, variance):
    y = compute_y(variance)
    z = compute_z(mean, variance)
    ones = [1]*len(mean)
    # a = sum(eng.times(ones, matlab.double(y))._data)
    # b = sum(eng.times(ones, matlab.double(z))._data)
    # c = sum(eng.times(ones, matlab.double(mean.values.tolist()))._data)
    return np.sum(np.array(ones).conj()*y, axis=0), np.sum(np.array(ones).
 ↪conj()*z, axis=0), np.sum(np.array(mean.values.tolist()).conj()*z, axis=0)
    # return a,b,c
```

```
# returns same values as matlab
def compute_mv_vars(mean, variance):
    a,b,c = compute_abc(mean, variance)
    sigma_mv = 1/(a**0.5)
    mu_mv = b/a
    nu_as = ((a*c - b**2)/a)**0.5
    return sigma_mv, mu_mv, nu_as


def is_solvent(portfolio, returns):
    return not any(y <= 0 for y in [1 + np.dot(portfolio, returns.iloc[day]) for␣
 ↪day in range(len(returns))])

def is_long(portfolio):
    return not any(y < 0 for y in portfolio.tolist())
```

[3]:
```
class Portfolio:
    def __init__(self, assets, name, start_date='2015-12-28',␣
 ↪end_date='2021-01-05'):
        self.data = yf.download(assets, start_date, end_date)['Adj Close']
        self.data = self.data.reindex(columns=assets)
        self.returns = (self.data - self.data.shift(1))/self.data.shift(1)
        self.start_date = start_date
        self.end_date = end_date
        self.portfolio_name = name

    # Returns mean and variance of portfolio for period
    def get_mean_and_variance(self, period_start = None, period_end = None):

        # Can add checks to make sure dates are valid with data that is contained
        if period_start is not None and period_end is not None:
            return self.returns[period_start:period_end].mean(), self.
 ↪returns[period_start:period_end].cov()

        return self.returns.mean(), self.returns.cov()

    # Returns minimum volatility portfolio allocation and parameters
    def get_minimum_volatility_portfolio_parameters(self, period_start = None,␣
 ↪period_end = None):
        if period_start is None and period_end is None:
            period_start = self.start_date
            period_end = self.end_date

        m, V = Portfolio.get_mean_and_variance(self, period_start, period_end)

        sigma_mv, mu_mv, nu_as = compute_mv_vars(m, V)
```

```
        y = compute_y(V)

        f_mv = (sigma_mv**2)*np.array(y)

        return pd.Series(data=f_mv, index=self.data.columns), sigma_mv, mu_mv,␣
↪nu_as

    # Returns safe tangent portfolio allocation if it exists and relevant␣
↪parameters
    def get_safe_tangent_portfolio_parameters(self, mu_si, period_start = None,␣
↪period_end = None):
        if period_start is None and period_end is None:
            period_start = self.start_date
            period_end = self.end_date

        m, V = Portfolio.get_mean_and_variance(self, period_start, period_end)

        f_mv, sigma_mv, mu_mv, nu_as = Portfolio.
↪get_minimum_volatility_portfolio_parameters(self, period_start, period_end)

        if mu_mv < mu_si:
            return None

        sigma_st = sigma_mv*(1 + ((nu_as*sigma_mv)/(mu_mv-mu_si))**2)**0.5
        mu_st = mu_mv + (nu_as**2)*(sigma_mv**2)/(mu_mv-mu_si)
        nu_st = nu_as*(1 + ((mu_mv-mu_si)/(nu_as*sigma_mv))**2)**0.5

        y = compute_y(V)
        z = compute_z(m, V)

        f_st = (sigma_mv**2/(mu_mv-mu_si))*(z - mu_si*np.array(y)).
↪reshape((len(m)))

        return pd.Series(data=f_st, index=self.data.columns), sigma_st, mu_st,␣
↪nu_st

    def get_credit_tangent_portfolio_parameters(self, mu_cl, period_start =␣
↪None, period_end = None):
        if period_start is None and period_end is None:
            period_start = self.start_date
            period_end = self.end_date

        m, V = Portfolio.get_mean_and_variance(self, period_start, period_end)

        f_mv, sigma_mv, mu_mv, nu_as = Portfolio.
↪get_minimum_volatility_portfolio_parameters(self, period_start, period_end)
```

```
        if mu_mv < mu_cl:
            return None

        sigma_ct = sigma_mv*(1 + ((nu_as*sigma_mv)/(mu_mv-mu_cl))**2)**0.5
        mu_ct = mu_mv + (nu_as**2)*(sigma_mv**2)/(mu_mv-mu_cl)
        nu_ct = nu_as*(1 + ((mu_mv-mu_cl)/(nu_as*sigma_mv))**2)**0.5

        y = compute_y(V)
        z = compute_z(m, V)

        f_ct = (sigma_mv**2/(mu_mv-mu_cl))*(z - mu_cl*np.array(y)).
↪reshape((len(m)))

        return pd.Series(data=f_ct, index=self.data.columns), sigma_ct, mu_ct,␣
↪nu_ct

    def get_long_tangent_portfolio(self, mu_si, period_start=None,␣
↪period_end=None):
        if period_start is None and period_end is None:
            period_start = self.start_date
            period_end = self.end_date

        m, V = Portfolio.get_mean_and_variance(self, period_start, period_end)

        mu_start = min(m)
        mu_end = max(m)
        step_size = (mu_end - mu_start)/300

        long_mus = np.arange(mu_start, mu_end + step_size, step_size)

        # Long Frontier
        Q = matrix(V.values, tc='d')
        z = matrix(np.zeros((len(V))).tolist(), tc='d')
        I = matrix((-1*np.identity(len(V))), tc='d')
        A = matrix(np.array([np.ones((len(V))).tolist(), m.values.tolist()]),␣
↪tc='d')

        nu_ca = (None, None, None, -100)

        # Will need to adjust for time periods longer than a year
        year = period_start[:4]

        for cur_mu in long_mus:
            deq = matrix(np.array([1, cur_mu]), tc='d')
            sol = solvers.qp(Q, z, I, z, A, deq)
            current_long_allocation = np.reshape(np.array(sol['x']), (len(m)))
```

```
            sigma_lf = (np.matmul(np.matmul(current_long_allocation, V.values),␣
↪np.reshape(current_long_allocation, (len(current_long_allocation),1)))[0])**0.5

            # Finding capital allocation line with greatest slope
            curNu_ca = (cur_mu - mu_si)/sigma_lf

            if curNu_ca > nu_ca[-1]:
                nu_ca = (current_long_allocation, sigma_lf, cur_mu, curNu_ca)

        return nu_ca
```

[4]:
```
def calculateSignalToNoiseForEstimators(expectedValue, stdDev, gamma):
    return np.abs(expectedValue - gamma)/ stdDev

def getVarianceEstimators(f_vector, mean, Var, expectedValue, stdDev):

    estimators = pd.DataFrame(columns = ['estimator name','estimator value',␣
↪'signal to Noise'])
    gamma_q = np.transpose(mean) @ f_vector - (np.transpose(f_vector)@ (mean @␣
↪np.transpose(mean) + Var) @ f_vector) * (1/2)
    gamma_p = np.transpose(mean) @ f_vector - (np.transpose(f_vector)@ (Var) @␣
↪f_vector) * (1/2)
    gamma_t = np.log(1 + np.transpose(mean) @ f_vector) - (np.
↪transpose(f_vector)@ (Var) @ f_vector) * (1/(2* np.square(1 + np.
↪transpose(mean) @ f_vector)))
    gamma_r = np.log(1 + np.transpose(mean) @ f_vector) - (np.
↪transpose(f_vector)@ (Var) @ f_vector) * (1/2)
    gamma_s = np.log(1 + np.transpose(mean) @ f_vector) - (np.
↪transpose(f_vector)@ (Var) @ f_vector) * (1/(2* (1 + 2 * np.transpose(mean) @␣
↪f_vector)))

    estimators.loc[len(estimators.index)] = ['gamma q', gamma_q,␣
↪calculateSignalToNoiseForEstimators(expectedValue, stdDev, gamma_q)]
    estimators.loc[len(estimators.index)] = ['gamma p', gamma_p,␣
↪calculateSignalToNoiseForEstimators(expectedValue, stdDev, gamma_p)]
    estimators.loc[len(estimators.index)] = ['gamma t', gamma_t,␣
↪calculateSignalToNoiseForEstimators(expectedValue, stdDev, gamma_t)]
    estimators.loc[len(estimators.index)] = ['gamma r', gamma_r,␣
↪calculateSignalToNoiseForEstimators(expectedValue, stdDev, gamma_r)]
    estimators.loc[len(estimators.index)] = ['gamma s', gamma_s,␣
↪calculateSignalToNoiseForEstimators(expectedValue, stdDev, gamma_s)]
    return estimators
```

[5]:
```
lgd_groupA = mpatches.Patch(color='blue', label='Group A')
```

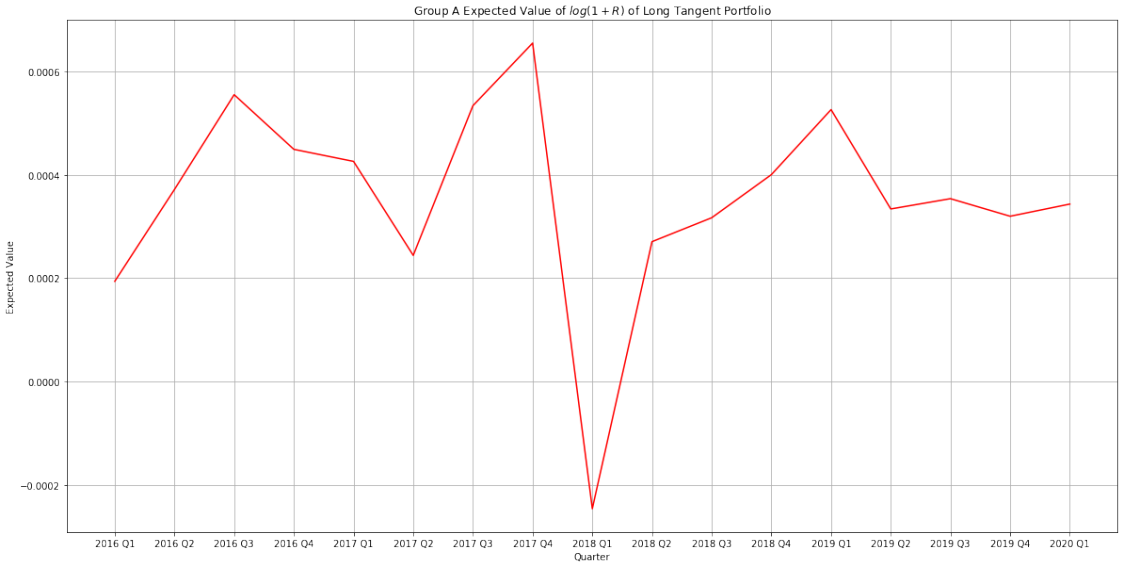
```
lgd_groupAB = mpatches.Patch(color='red', label='Group AB (And Assets Not␣
 ↪Included in A)')
lgd_groupABC = mpatches.Patch(color='green', label='Group ABC (And Assets Not␣
 ↪Included in AB)')
assets = Line2D([0], [0], marker='o', color='w', markerfacecolor='k',␣
 ↪label='Assets', markersize=12)
rf = Line2D([0], [0], marker='D', color='w', markerfacecolor='k', label='Risk␣
 ↪Free Rates', markersize=12)
ctg = Line2D([0], [0], marker='*', color='w', markerfacecolor='k', label='Credit␣
 ↪Tangent Portfolio', markersize=12)
stg = Line2D([0], [0], marker='^', color='w', markerfacecolor='k', label='Safe␣
 ↪Tangent Portfolio', markersize=12)
ltg = Line2D([0], [0], marker='X', color='w', markerfacecolor='k', label='Long␣
 ↪Tangent Portfolio', markersize=12)
equi = Line2D([0], [0], marker='P', color='w', markerfacecolor='k',␣
 ↪label='Equidistributed Portfolio', markersize=12)
mv_portfolio = Line2D([0], [0], marker='s', color='w', markerfacecolor='k',␣
 ↪label='Minimum Variance Portfolio', markersize=12)
mv_frontier = Line2D([0], [0], linewidth=1., color='k', label='Minimum Variance␣
 ↪Frontier', markersize=12)
efficient_frontier = Line2D([0], [0], linestyle='--', linewidth=1., color='k',␣
 ↪label='Efficient Frontier', markersize=12)
efficient_frontier.set_linestyle('--')
long_frontier = Line2D([0], [0], linestyle=':', linewidth=1., color='k',␣
 ↪label='Long Frontier', markersize=12)
long_frontier.set_linestyle(':')
lmtd_frontier_1 = Line2D([0], [0], linestyle='-.', linewidth=1., color='m',␣
 ↪label='Limited Leverage Frontier (l = 1)', markersize=12)
lmtd_frontier_1.set_linestyle('-.')
lmtd_frontier_5 = Line2D([0], [0], linestyle='-.', linewidth=1., color='c',␣
 ↪label='Limited Leverage Frontier (l = 5)', markersize=12)
lmtd_frontier_5.set_linestyle('-.')
```

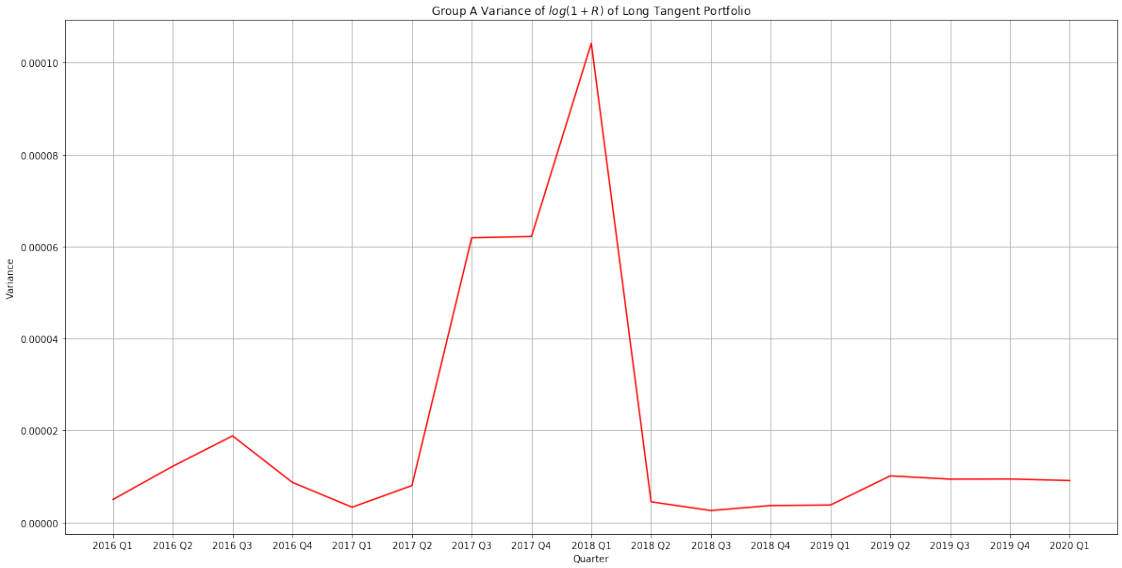
[6]:
```
groupA = Portfolio(['VFIAX', 'VBTLX', 'VGSLX'], 'Group A')
groupAB = Portfolio(['VFIAX', 'VBTLX', 'VGSLX', 'VIMAX', 'VSMAX', 'VGHCX'],␣
 ↪'Group AB')
groupABC = Portfolio(['VFIAX', 'VBTLX', 'VGSLX', 'VIMAX', 'VSMAX', 'VGHCX',␣
 ↪'AMZN', 'WMT', 'CVS'], 'Group ABC')
```

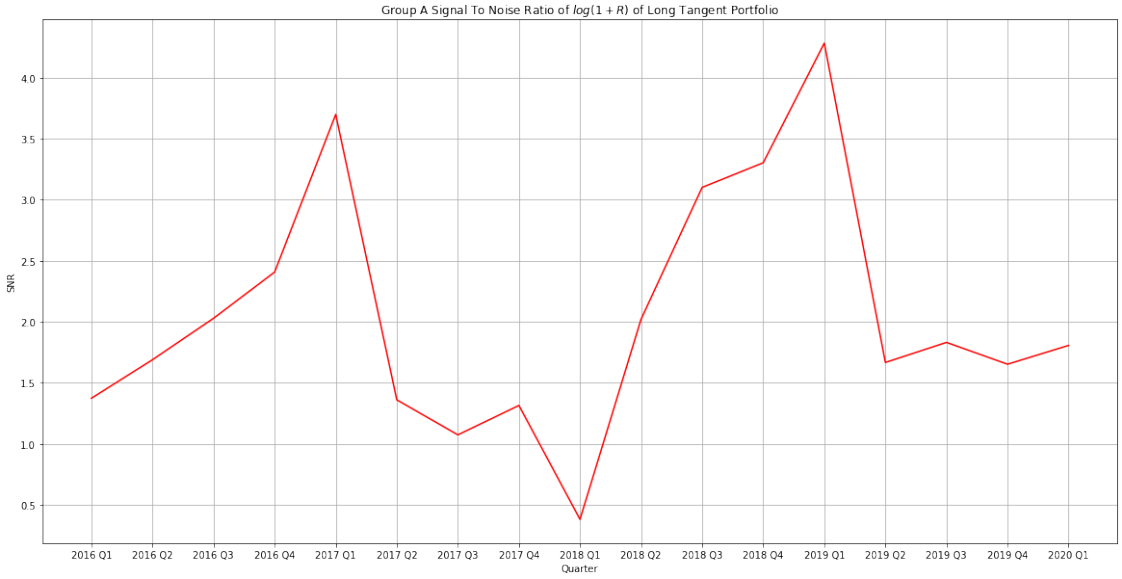
[7]:
```
x_axis = [period_start[0:4] + ' Q' + str((i % 4) + 1) for i, (period_start,␣
 ↪period_end) in enumerate(rolling_quarter_yearly)]
for group_number, portfolio in enumerate([groupA, groupAB, groupABC]):
    results = [[], [], []]

    for rolling_index, (period_start, period_end) in␣
 ↪enumerate(rolling_quarter_yearly):
```

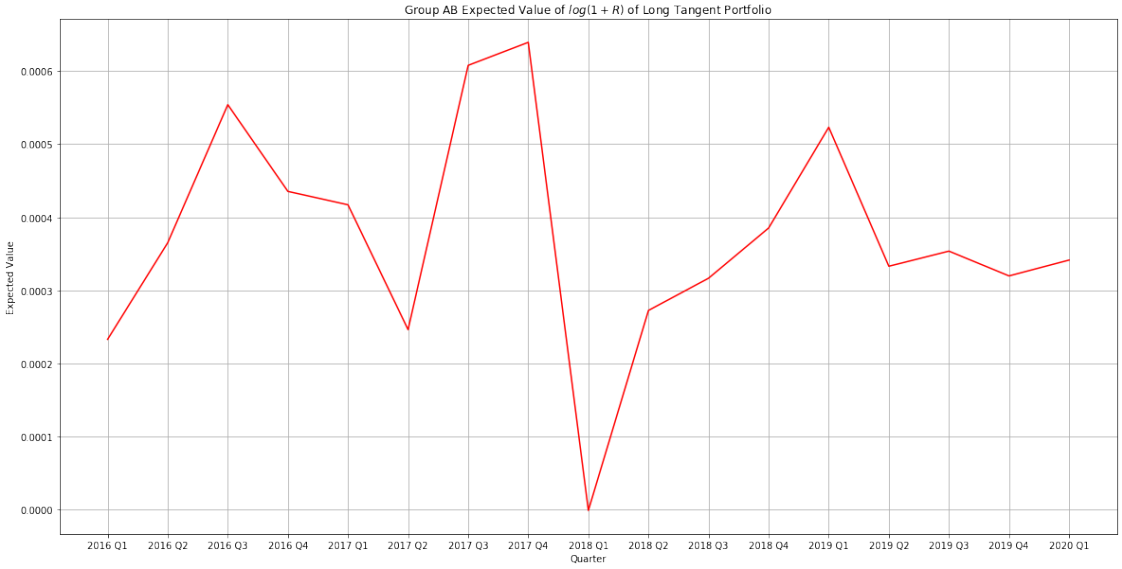
```
        m, V = portfolio.get_mean_and_variance(period_start, period_end)
        period_returns = portfolio.returns[period_start:period_end]

        mu_min = min(m)
        mu_max = max(m)

        _, sigma_mv, mu_mv, nu_as = portfolio.
↪get_minimum_volatility_portfolio_parameters(period_start, period_end)

        mus = np.arange(mu_min, mu_max, 0.000005)
        frontier = np.reshape([(sigma_mv**2 + ((mu - mu_mv)/nu_as)**2)**0.5 for␣
↪mu in mus], (len(mus)))

        f_lt, sigma_lt, mu_lt, nu_lt = portfolio.
↪get_long_tangent_portfolio(mu_si[period_start[0:4] + ' Q' + str((rolling_index␣
↪% 4) + 1)], period_start, period_end)

        log_returns = (period_returns@f_lt).apply(math.log1p)

        uniformWeights = 1/log_returns.shape[0]

        wBar = np.sum(np.square(np.ones(log_returns.shape[0]) * uniformWeights))

        mean = np.sum(log_returns, axis=0) * uniformWeights
        difference = log_returns.subtract(mean)
        variance =  np.array(1/(1 - wBar) * np.sum(uniformWeights * np.
↪square(difference), axis = 0)).reshape(-1,1)

        StdOfExpectedValue = np.array(np.sqrt(wBar) * np.sqrt(variance)).
↪reshape(-1,1)
        signalToNoise = np.absolute(np.array(mean/StdOfExpectedValue)).
↪reshape(-1,1)[0][0]

        results[0].append(mean)
        results[1].append(variance[0][0])
        results[2].append(signalToNoise)

   plt.figure(num=group_number*3, figsize=(20,10))
   plt.plot(x_axis, results[0], 'r')
   plt.title(portfolio.portfolio_name + ' Expected Value of $log(1+R)$ of Long␣
↪Tangent Portfolio')
   plt.ylabel('Expected Value')
   plt.xlabel('Quarter')
   plt.grid()
   plt.show()
```

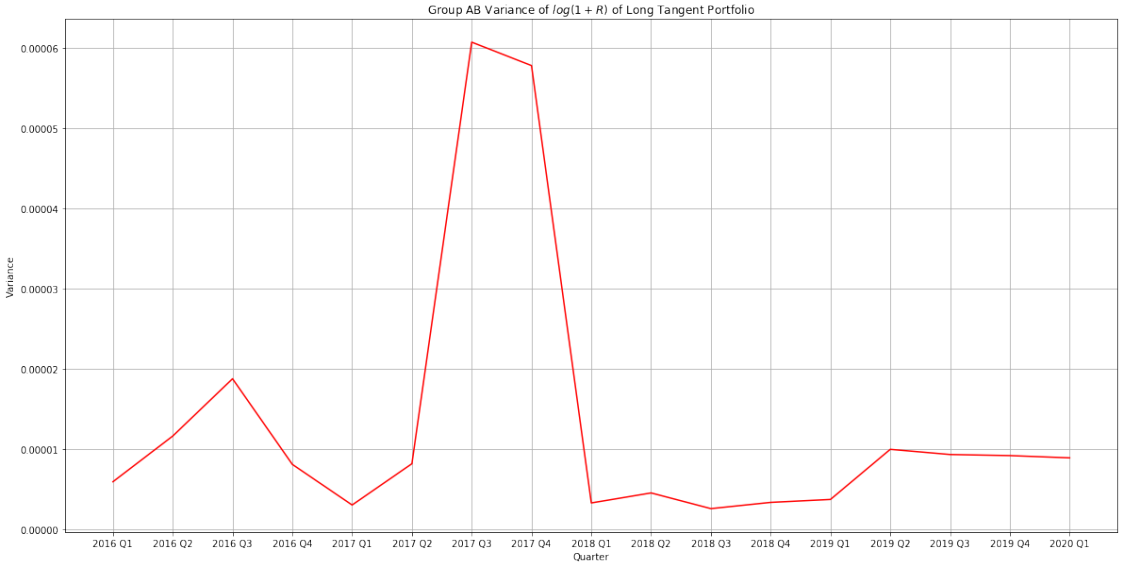
```
   plt.figure(num=group_number*3 + 1, figsize=(20,10))
   plt.plot(x_axis, results[1], 'r')
   plt.title(portfolio.portfolio_name + ' Variance of $log(1+R)$ of Long␣
↪Tangent Portfolio')
   plt.ylabel('Variance')
   plt.xlabel('Quarter')
   plt.grid()
   plt.show()

   plt.figure(num=group_number*3 + 2, figsize=(20,10))
   plt.plot(x_axis, results[2], 'r')
   plt.title(portfolio.portfolio_name + ' Signal To Noise Ratio of $log(1+R)$␣
↪of Long Tangent Portfolio')
   plt.ylabel('SNR')
   plt.xlabel('Quarter')
   plt.grid()
   plt.show()
```



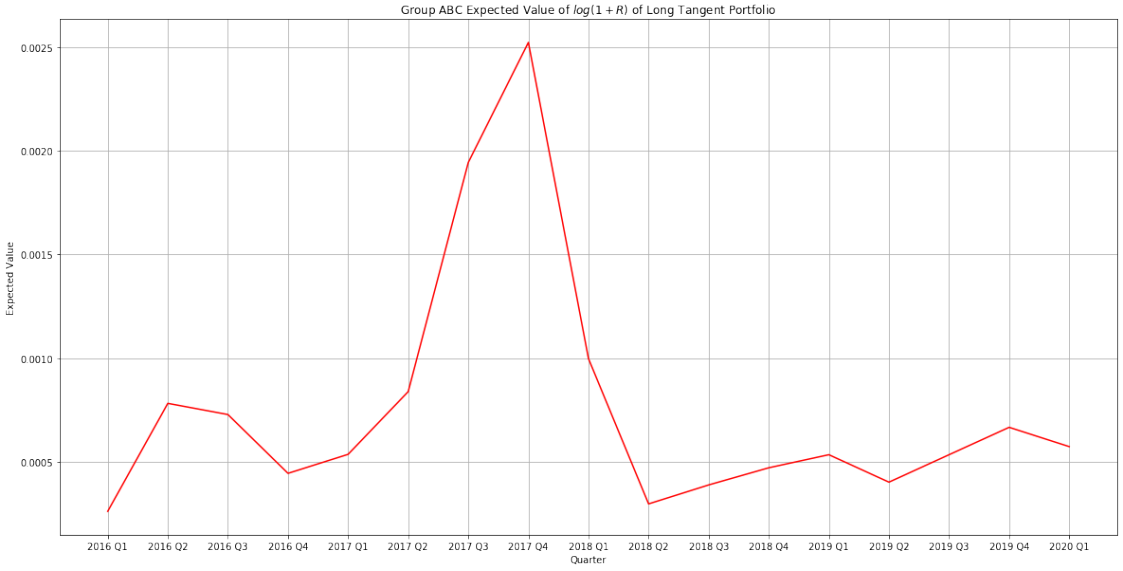

Group A Variance of $log(1+R)$ of Long Tangent Portfolio
Variance
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
2016 Q1
2016 Q2
2016 Q3
2016 Q4
2017 Q1
2017 Q2
2017 Q3
2017 Q4
2018 Q1
2018 Q2
2018 Q3
2018 Q4
2019 Q1
2019 Q2
2019 Q3
2019 Q4
2020 Q1
Quarter

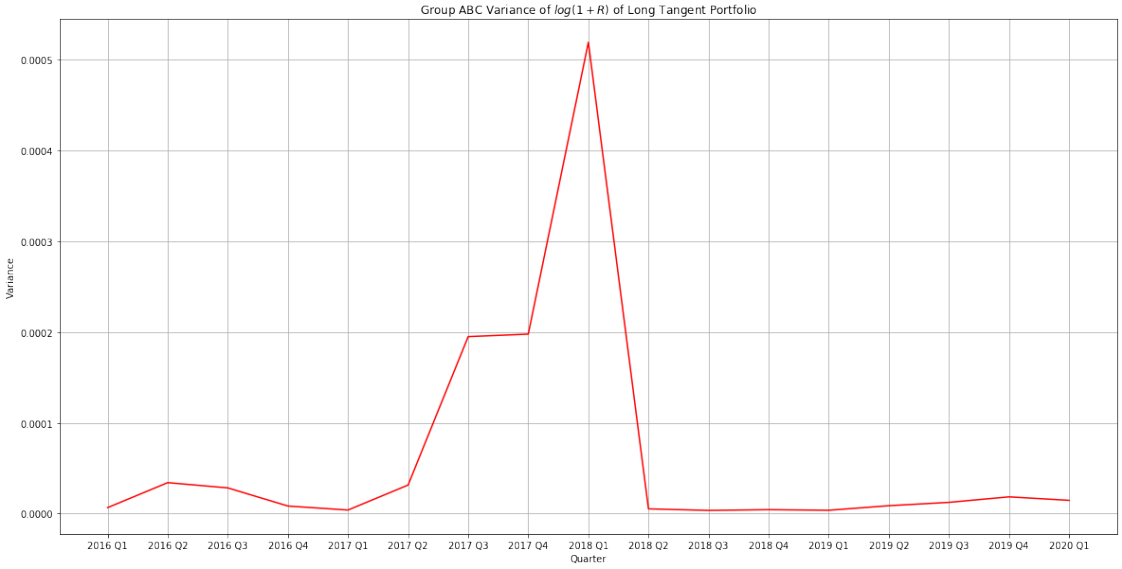

Group A Signal To Noise Ratio of $log(1+R)$ of Long Tangent Portfolio
SNR
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
2016 Q1
2016 Q2
2016 Q3
2016 Q4
2017 Q1
2017 Q2
2017 Q3
2017 Q4
2018 Q1
2018 Q2
2018 Q3
2018 Q4
2019 Q1
2019 Q2
2019 Q3
2019 Q4
2020 Q1
Quarter

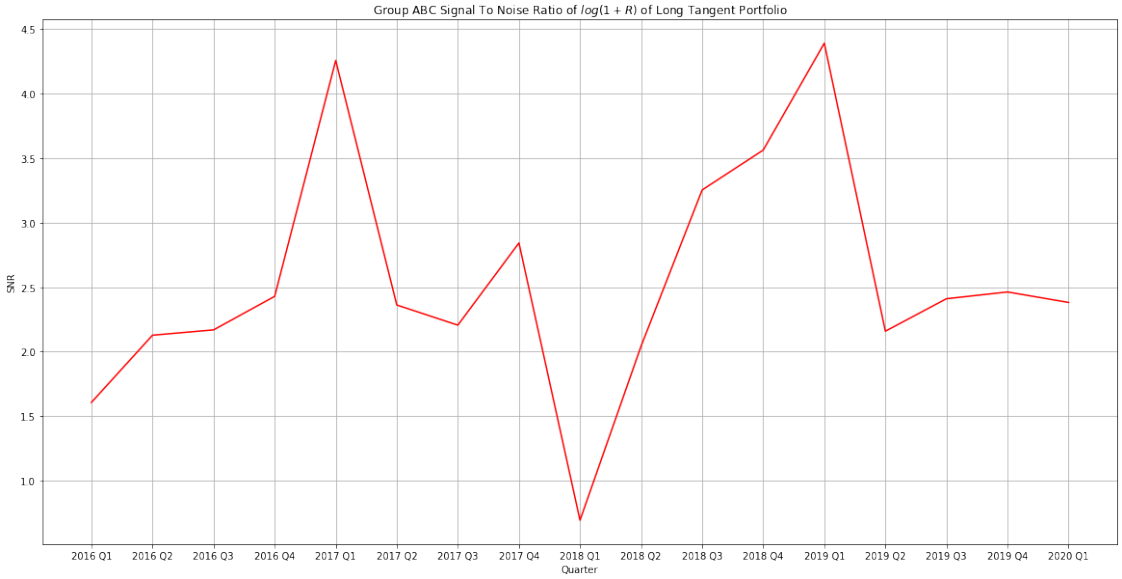

Group AB Expected Value of $log(1+R)$ of Long Tangent Portfolio
Expected Value
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
2016 Q1
2016 Q2
2016 Q3
2016 Q4
2017 Q1
2017 Q2
2017 Q3
2017 Q4
2018 Q1
2018 Q2
2018 Q3
2018 Q4
2019 Q1
2019 Q2
2019 Q3
2019 Q4
2020 Q1
Quarter


Group AB Variance of $log(1+R)$ of Long Tangent Portfolio
Variance
0.00006
0.00005
0.00004
0.00003
0.00002
0.00001
0.00000
2016 Q1
2016 Q2
2016 Q3
2016 Q4
2017 Q1
2017 Q2
2017 Q3
2017 Q4
2018 Q1
2018 Q2
2018 Q3
2018 Q4
2019 Q1
2019 Q2
2019 Q3
2019 Q4
2020 Q1
Quarter


Group AB Signal To Noise Ratio of $log(1+R)$ of Long Tangent Portfolio
SNR
0
1
2
3
4
2016 Q1
2016 Q2
2016 Q3
2016 Q4
2017 Q1
2017 Q2
2017 Q3
2017 Q4
2018 Q1
2018 Q2
2018 Q3
2018 Q4
2019 Q1
2019 Q2
2019 Q3
2019 Q4
2020 Q1
Quarter


Group ABC Expected Value of $log(1+R)$ of Long Tangent Portfolio
Expected Value
0.0005
0.0010
0.0015
0.0020
0.0025
2016 Q1
2016 Q2
2016 Q3
2016 Q4
2017 Q1
2017 Q2
2017 Q3
2017 Q4
2018 Q1
2018 Q2
2018 Q3
2018 Q4
2019 Q1
2019 Q2
2019 Q3
2019 Q4
2020 Q1
Quarter





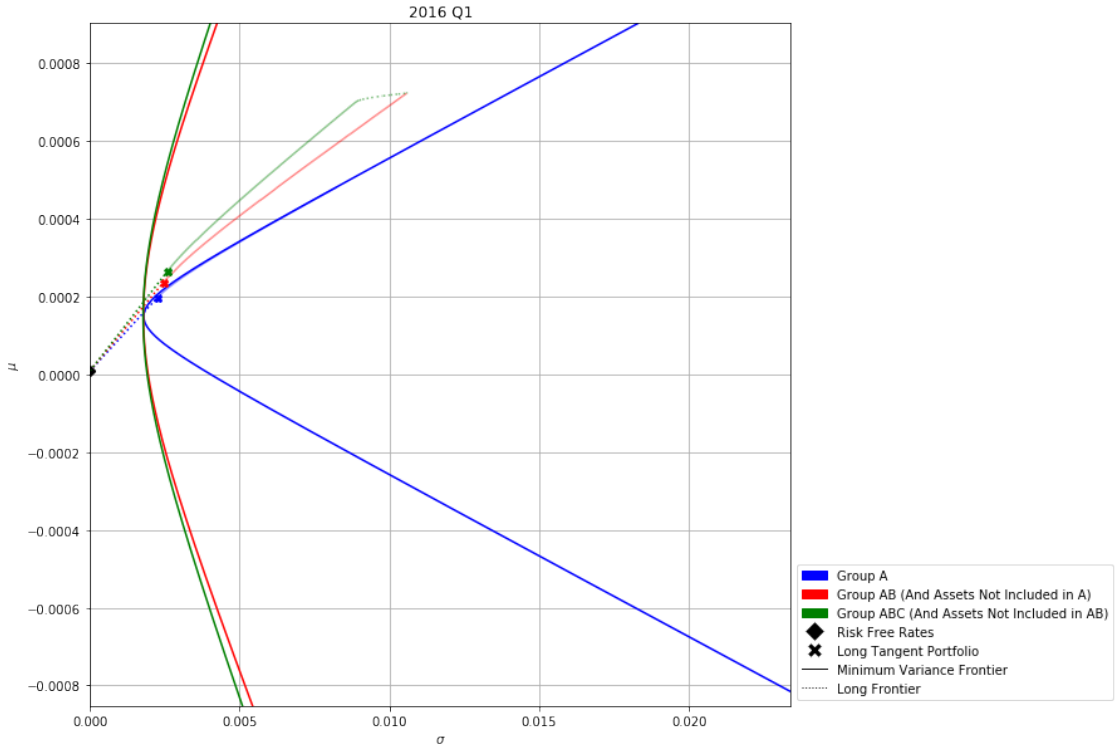
From the signal to noise ratios of each of the portfolios, the long tangent portfolios for group ABC has values higher than the AB and A portfolios Signal to noise ratio, for most if not all of the time periods on the chart. From this, the ABC portfolio has the most certain results.

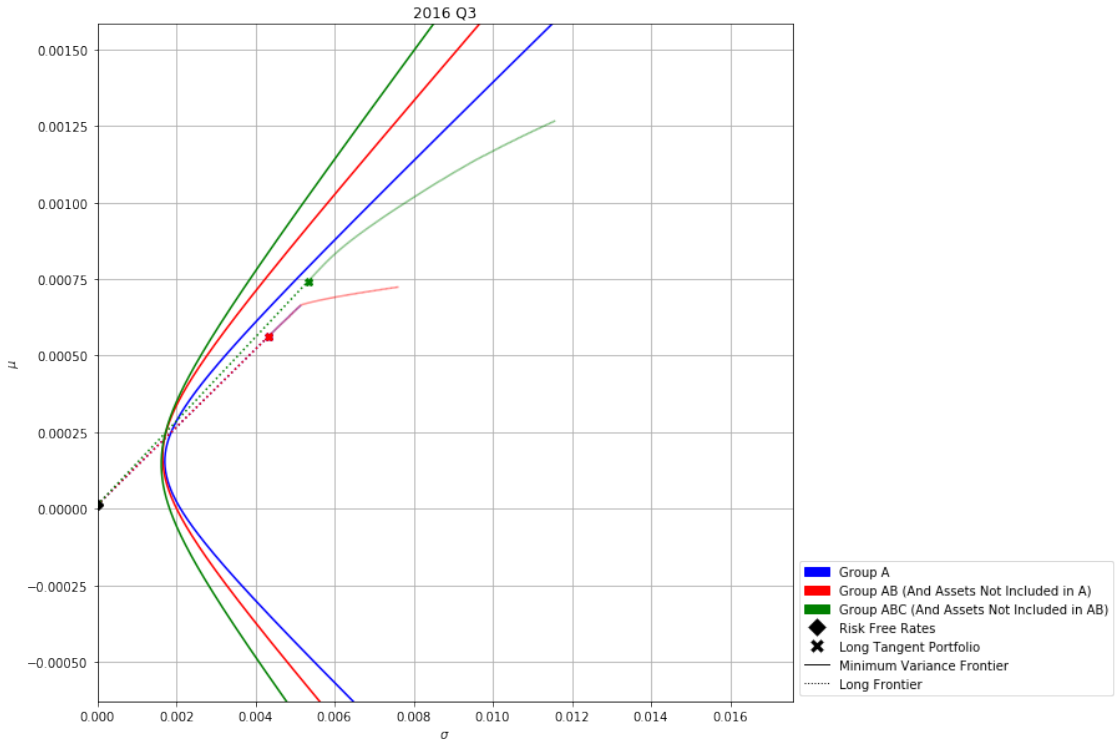
Exercise 2

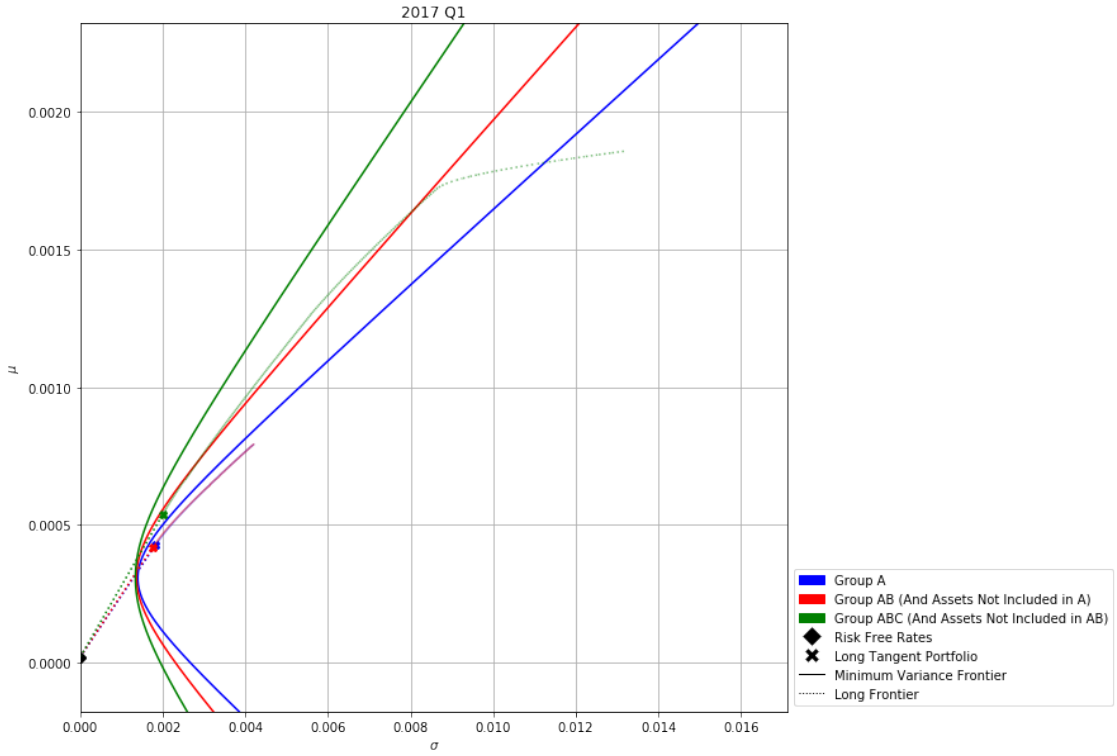
[8]:
```
x_axis = []
colors = ['b', 'r', 'g']
for rolling_index, (period_start, period_end) in␣
↪enumerate(rolling_quarter_yearly[::2]):

    plt.figure(rolling_index, figsize=(10,10))

    m_all, V_all = groupABC.get_mean_and_variance(period_start, period_end)

    mu_end = abs(m_all.max())*1.25
    mu_start = -abs(m_all.min())*1.25

    max_sigma = V_all.values.max()**0.5

    for group_number, portfolio in enumerate([groupA, groupAB, groupABC]):

        m, V = portfolio.get_mean_and_variance(period_start, period_end)
        period_returns = portfolio.returns[period_start:period_end]

        _, sigma_mv, mu_mv, nu_as = portfolio.
↪get_minimum_volatility_portfolio_parameters(period_start, period_end)

        mus = np.arange(mu_start, mu_end, 0.000005)
        frontier = np.reshape([(sigma_mv**2 + ((mu - mu_mv)/nu_as)**2)**0.5 for␣
↪mu in mus], (len(mus)))

        f_lt, sigma_lt, mu_lt, nu_lt = portfolio.
↪get_long_tangent_portfolio(mu_si[period_start[0:4] + ' Q' + str((rolling_index␣
↪% 2)*2 + 1)], period_start, period_end)

        min_mu = mu_lt
        max_mu = max(m)
        step_size = (max_mu-min_mu)/200

        if min_mu < max_mu:

            long_mus = np.arange(min_mu, max_mu + step_size, step_size)

            previous_mu = long_mus[0]

            Q = matrix(V.values, tc='d')
            z = matrix(np.zeros((len(V))).tolist(), tc='d')
            I = matrix((-1*np.identity(len(V))), tc='d')
            A = matrix(np.array([np.ones((len(V))).tolist(), m.values.
↪tolist()]), tc='d')
```

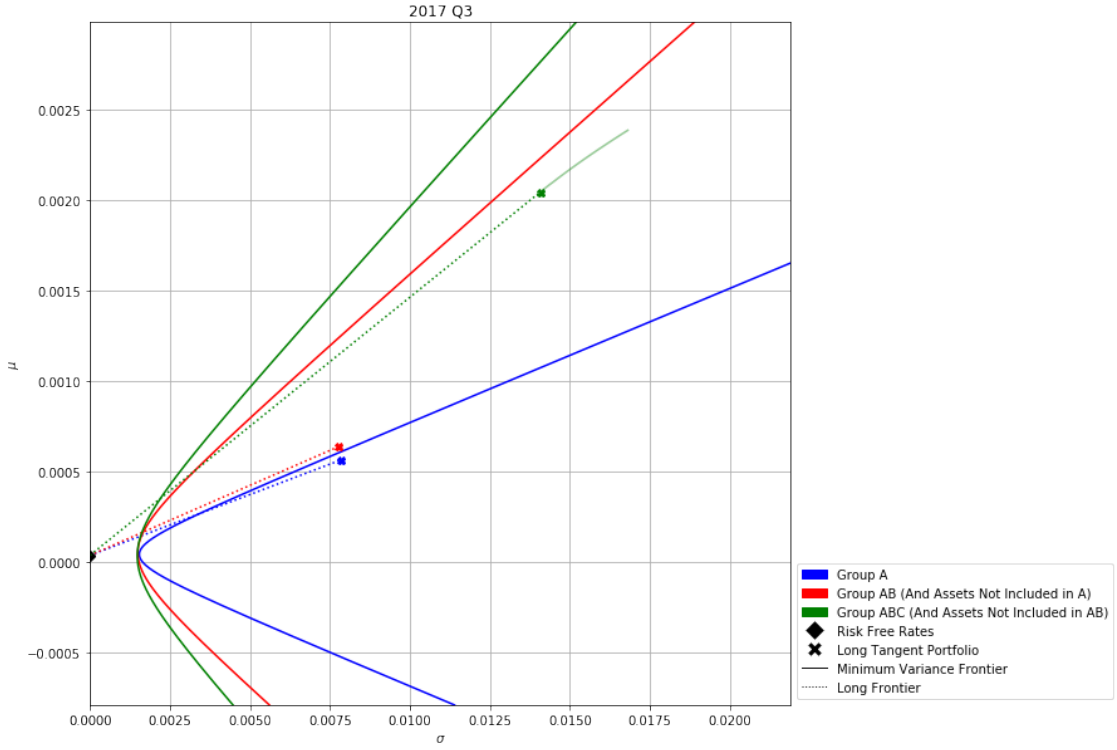
```
            deq = matrix(np.array([1, previous_mu]), tc='d')
            sol = solvers.qp(Q, z, I, z, A, deq)

            prev_long_allocation = np.reshape(np.array(sol['x']), (len(m)))


            for cur_mu in long_mus[1:]:


                mu_range = np.arange(previous_mu, cur_mu, (cur_mu-previous_mu)/
↪10)

                deq = matrix(np.array([1, cur_mu]), tc='d')
                sol = solvers.qp(Q, z, I, z, A, deq)
                current_long_allocation = np.reshape(np.array(sol['x']),␣
↪(len(m)))

                # Linear interpolation between allocations for 2 mus (Long␣
↪frontier)
                f_lf = [(cur_mu - mu)/(cur_mu -␣
↪previous_mu)*prev_long_allocation + (mu - previous_mu)/(cur_mu -␣
↪previous_mu)*current_long_allocation for mu in mu_range]
                sigma_lf = [(np.matmul(np.matmul(f, V.values), np.reshape(f,␣
↪(len(f),1)))[0])**0.5 for f in f_lf]

                # Plot long frontier linear interpolation
                plt.plot(sigma_lf, mu_range, colors[group_number]+':', alpha=0.5)

                previous_mu = cur_mu
                prev_long_allocation = current_long_allocation

        plt.plot(0, mu_si[period_start[0:4] + ' Q' + str((rolling_index % 2)*2 +␣
↪1)], 'kD')
        plt.plot(frontier, mus, colors[group_number])
        plt.plot(sigma_lt, mu_lt, colors[group_number] + 'X')
        plt.plot([0, sigma_lt], [mu_si[period_start[:4] + ' Q' +␣
↪str((rolling_index % 2)*2 + 1)], mu_lt], colors[group_number] + ':')

    lgd = plt.legend(bbox_to_anchor=(1, 0), loc='lower left', ncol=1,␣
↪handles=[lgd_groupA, lgd_groupAB, lgd_groupABC, rf, ltg, mv_frontier,␣
↪long_frontier])
    plt.xlabel('$\sigma$')
    plt.ylabel('$\mu$')
    plt.ylim(mu_start, mu_end)
    plt.xlim(0, max_sigma*1.25)
    plt.title(period_start[:4] + ' Q' + str((rolling_index % 2)*2 + 1))
```

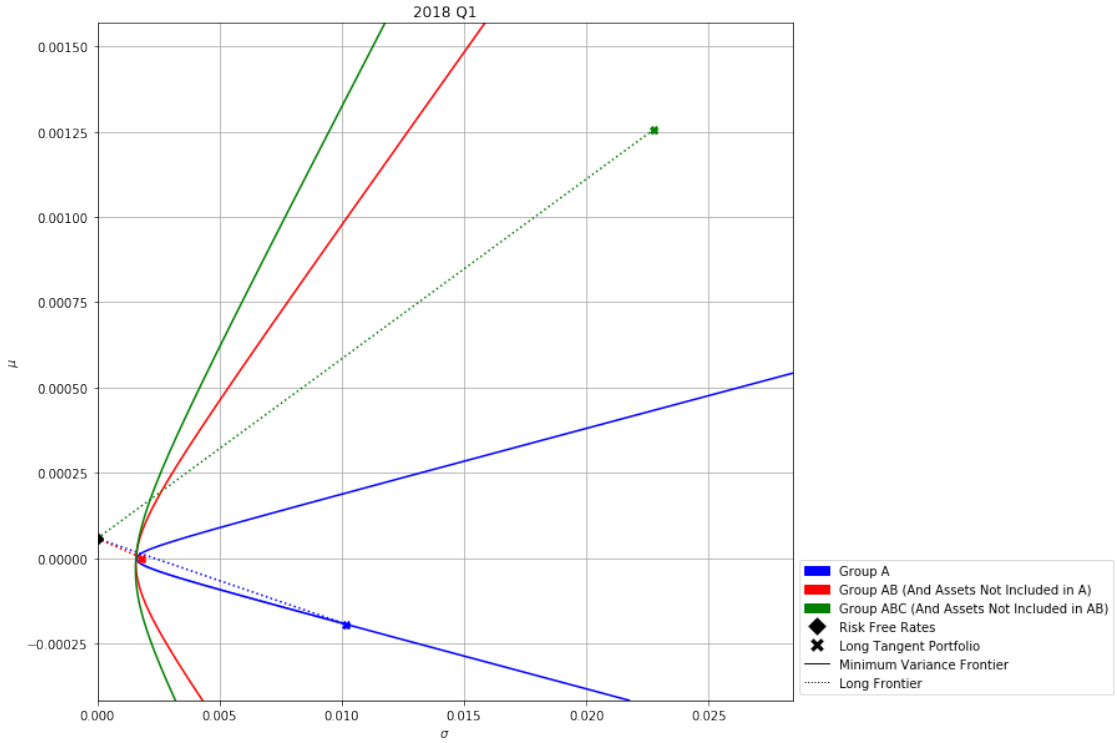
```
plt.grid()
```

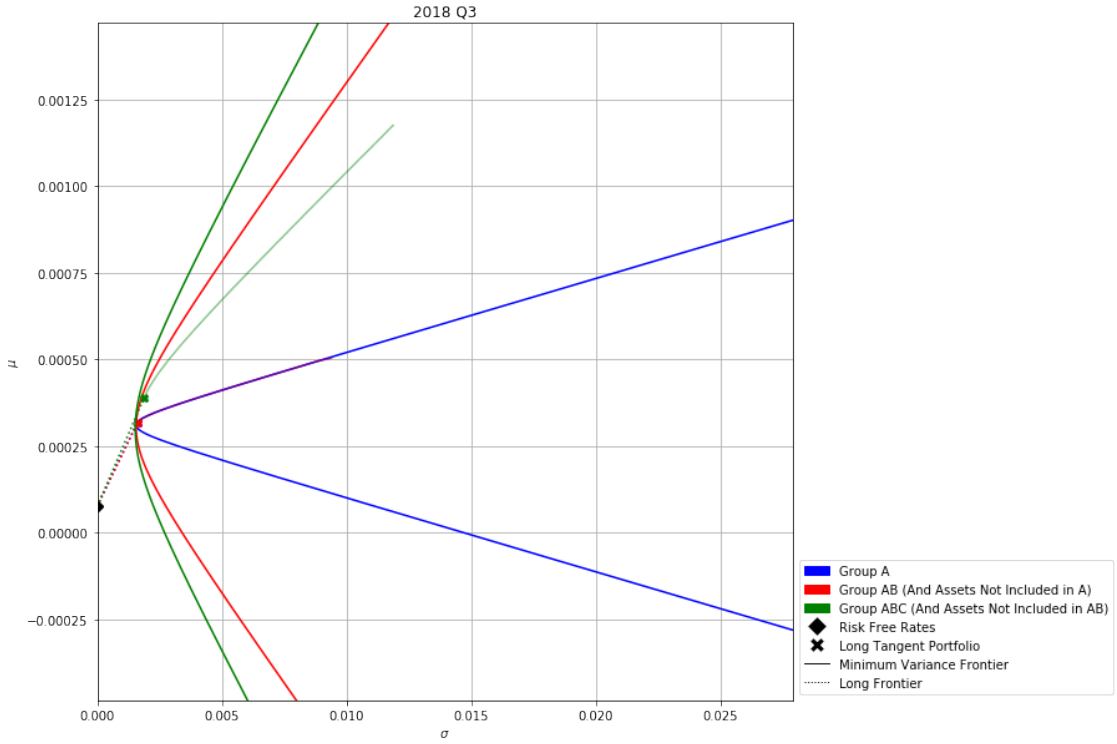

2016 Q1
0.0008
0.0006
0.0004
0.0002
0.0000
−0.0002
−0.0004
−0.0006
−0.0008
μ
0.000
0.005
0.010
0.015
0.020
σ
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Long Tangent Portfolio
Minimum Variance Frontier
Long Frontier

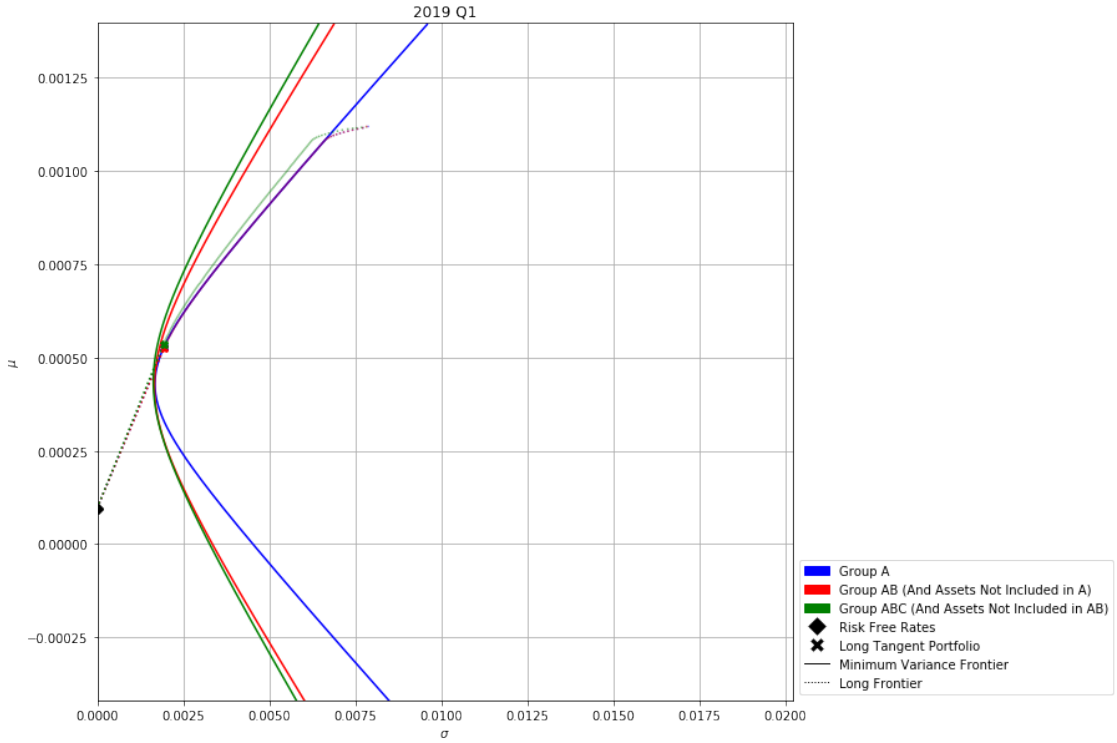

2016 Q3
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
−0.00025
−0.00050
μ
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
0.016
σ
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Long Tangent Portfolio
Minimum Variance Frontier
Long Frontier

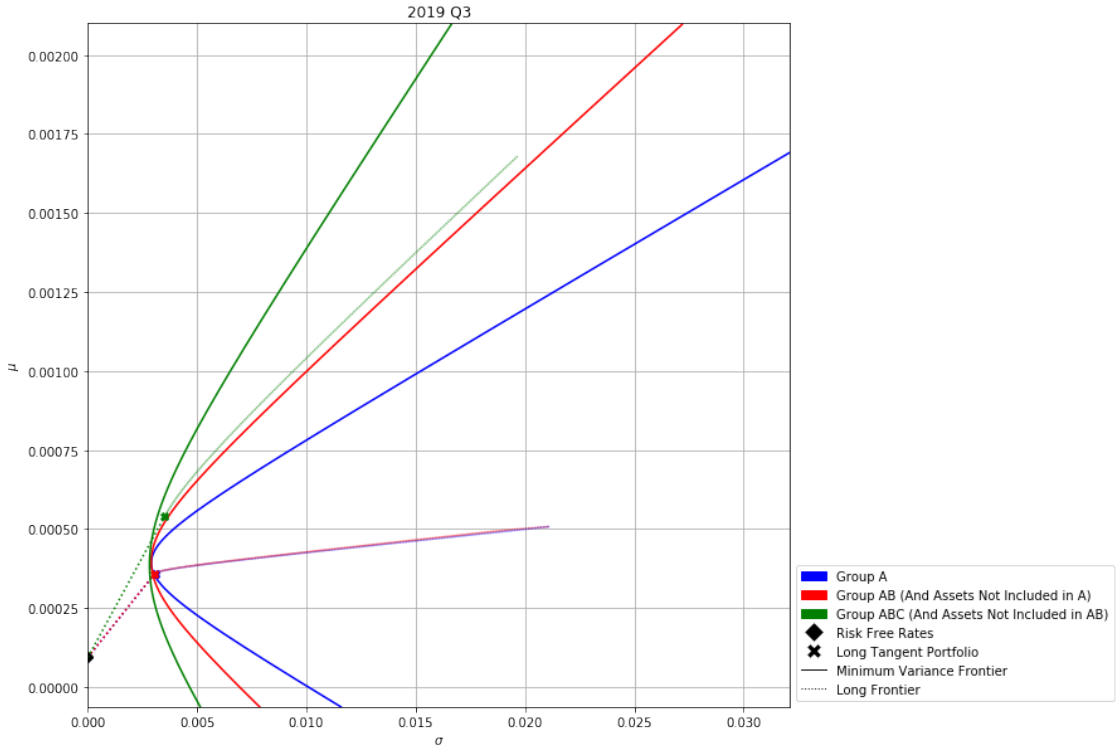

2017 Q1
0.0020
0.0015
0.0010
0.0005
0.0000
μ
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
0.016
σ
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Long Tangent Portfolio
Minimum Variance Frontier
Long Frontier

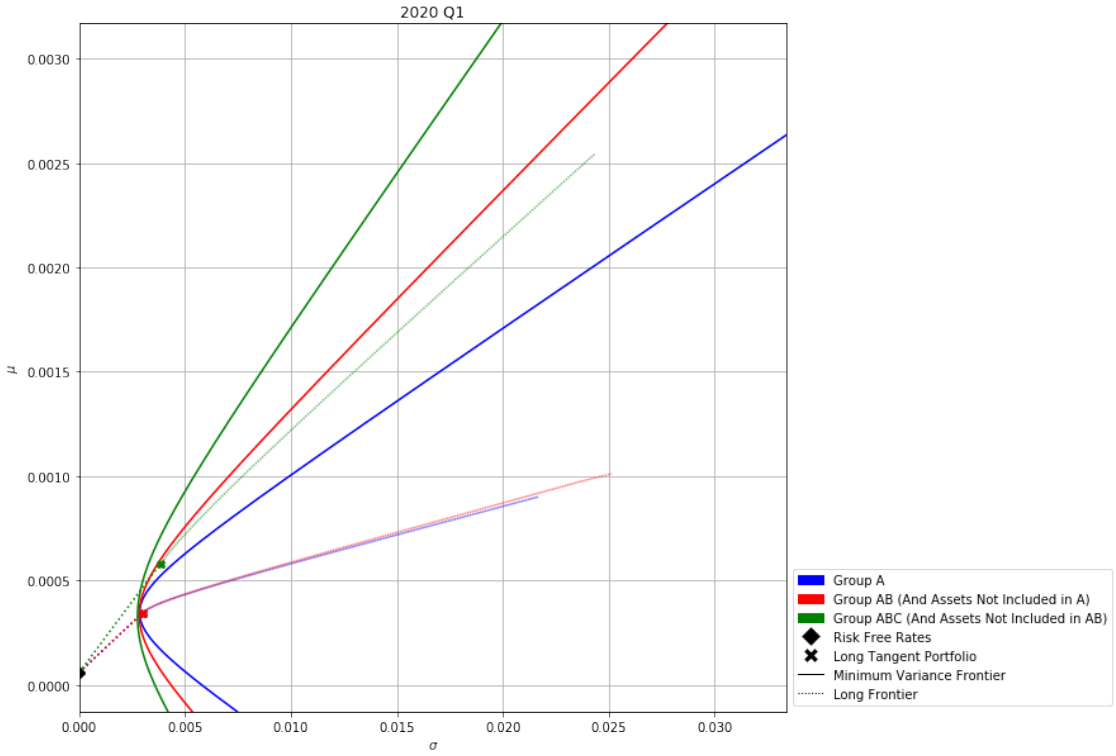

2017 Q3
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
−0.0005
μ
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
σ
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Long Tangent Portfolio
Minimum Variance Frontier
Long Frontier


2018 Q1
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
−0.00025
μ
0.000
0.005
0.010
0.015
0.020
0.025
σ
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Long Tangent Portfolio
Minimum Variance Frontier
Long Frontier


2018 Q3
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
−0.00025
μ
0.000
0.005
0.010
0.015
0.020
0.025
σ
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Long Tangent Portfolio
Minimum Variance Frontier
Long Frontier


2019 Q1
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
−0.00025
μ
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
σ
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Long Tangent Portfolio
Minimum Variance Frontier
Long Frontier


2019 Q3
0.00200
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
μ
0.000
0.005
0.010
0.015
0.020
0.025
0.030
σ
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Long Tangent Portfolio
Minimum Variance Frontier
Long Frontier


2020 Q1
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
μ
0.000
0.005
0.010
0.015
0.020
0.025
0.030
σ
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Long Tangent Portfolio
Minimum Variance Frontier
Long Frontier

Exercise 3

[9]:
```
for group_number, portfolio in enumerate([groupA, groupAB, groupABC]):
    data = pd.DataFrame(columns=['Q', 'T'])
    for rolling_index, (period_start, period_end) in␣
↪enumerate(rolling_quarter_yearly):

        m, V = portfolio.get_mean_and_variance(period_start, period_end)
        period_returns = portfolio.returns[period_start:period_end]

        _, sigma_mv, mu_mv, nu_as = portfolio.
↪get_minimum_volatility_portfolio_parameters(period_start, period_end)

        f_lt, sigma_lt, mu_lt, nu_lt = portfolio.
↪get_long_tangent_portfolio(mu_si[period_start[0:4] + ' Q' + str((rolling_index␣
↪% 2)*2 + 1)], period_start, period_end)

        theta_q = (np.transpose(f_lt)@V)@f_lt
        theta_t = (theta_q)/(1+np.transpose(m)@f_lt)**2

        log_returns = (period_returns@f_lt).apply(math.log1p)

        uniformWeights = 1/log_returns.shape[0]

        wBar = np.sum(np.square(np.ones(log_returns.shape[0]) * uniformWeights))

        mean = np.sum(log_returns, axis=0) * uniformWeights
        difference = log_returns.subtract(mean)
        variance =  np.array(1/(1 - wBar) * np.sum(uniformWeights * np.
↪square(difference), axis = 0)).reshape(-1,1)

        StdOfExpectedValue = np.array(np.sqrt(wBar) * np.sqrt(variance)).
↪reshape(-1,1)

        curYearAndQuarter = period_start[0:4] + ' Q' + str((rolling_index % 4) +␣
↪1)

        r1 = ((variance[0][0]**0.5 - theta_q**0.5)/StdOfExpectedValue)[0][0]
        r2 = ((variance[0][0]**0.5 - theta_t**0.5)/StdOfExpectedValue)[0][0]

        data.loc[curYearAndQuarter] = [r1, r2]

    display(portfolio.portfolio_name, data.style)
```

Group A

	Q	T
2016 Q1	-0.003008	0.000106
2016 Q2	0.011794	0.017759
2016 Q3	0.001497	0.010222
2016 Q4	-0.005998	0.001147
2017 Q1	-0.001751	0.005017
2017 Q2	0.032627	0.037055
2017 Q3	0.072186	0.081078
2017 Q4	0.068940	0.079745
2018 Q1	0.042347	0.039279
2018 Q2	-0.000063	0.004444
2018 Q3	-0.006574	-0.001546
2018 Q4	-0.001725	0.004612
2019 Q1	-0.006824	0.001552
2019 Q2	0.023978	0.029357
2019 Q3	0.037537	0.043220
2019 Q4	0.042272	0.047414
2020 Q1	0.041263	0.046778

Group AB

	Q	T
2016 Q1	-0.004507	-0.000764
2016 Q2	0.011596	0.017446
2016 Q3	0.001494	0.010232
2016 Q4	-0.005704	0.001222
2017 Q1	-0.001885	0.004745
2017 Q2	0.032211	0.036602
2017 Q3	0.051692	0.061764
2017 Q4	0.065730	0.076267
2018 Q1	-0.001678	-0.001674
2018 Q2	-0.000160	0.004350
2018 Q3	-0.006440	-0.001414
2018 Q4	-0.002374	0.003756
2019 Q1	-0.007117	0.001218
2019 Q2	0.024555	0.029920
2019 Q3	0.036660	0.042344
2019 Q4	0.042001	0.047194
2020 Q1	0.040985	0.046470

Group ABC

	Q	T
2016 Q1	-0.003586	0.000603
2016 Q2	-0.016307	-0.003741
2016 Q3	-0.002107	0.009567
2016 Q4	-0.006008	0.001043
2017 Q1	-0.014834	-0.006315
2017 Q2	-0.071649	-0.055590
2017 Q3	-0.132688	-0.100280
2017 Q4	-0.131320	-0.089507
2018 Q1	0.022629	0.042467
2018 Q2	-0.002238	0.002569
2018 Q3	-0.009098	-0.003019
2018 Q4	-0.006883	0.000607
2019 Q1	-0.008739	-0.000237
2019 Q2	0.031025	0.037461
2019 Q3	0.013877	0.022452
2019 Q4	-0.001632	0.009394
2020 Q1	0.009490	0.018705

The Quadratic estimator for groups ABC and AB are better than the Taylor estimator for most of the values, but for the A group they both do about the same. Compared to the previous homework, the quadratic estimator had a divisor term that was missing. From this difference in equations, we can possibly find the errors.