Portfolio Team Homework 6

Kyle Chang, Christian Warren, Dev Vadalia

April 13, 2021

Exercise 1

[45]:
```
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([0.22,0.54, 1.44, 2.42, 1.54])/100
mu_cl = np.add(mu_si, [.03]*5)
mu_si = mu_si/252
mu_cl = mu_cl/252

mu_safe = {}
mu_credit = {}

for year_index, year in enumerate(years):
    mu_safe[year[0][:4]] = mu_si[year_index]
    mu_credit[year[0][:4]] = mu_cl[year_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)
```

[46]:
```
# 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())
```

[47]:
```
class Portfolio:
    def __init__(self, assets, name, start_date='2005-12-28',␣
↪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
```

[48]:
```
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')
```

```
[**********100%**********]  3 of 3 completed
[**********100%**********]  6 of 6 completed
[**********100%**********]  9 of 9 completed
```

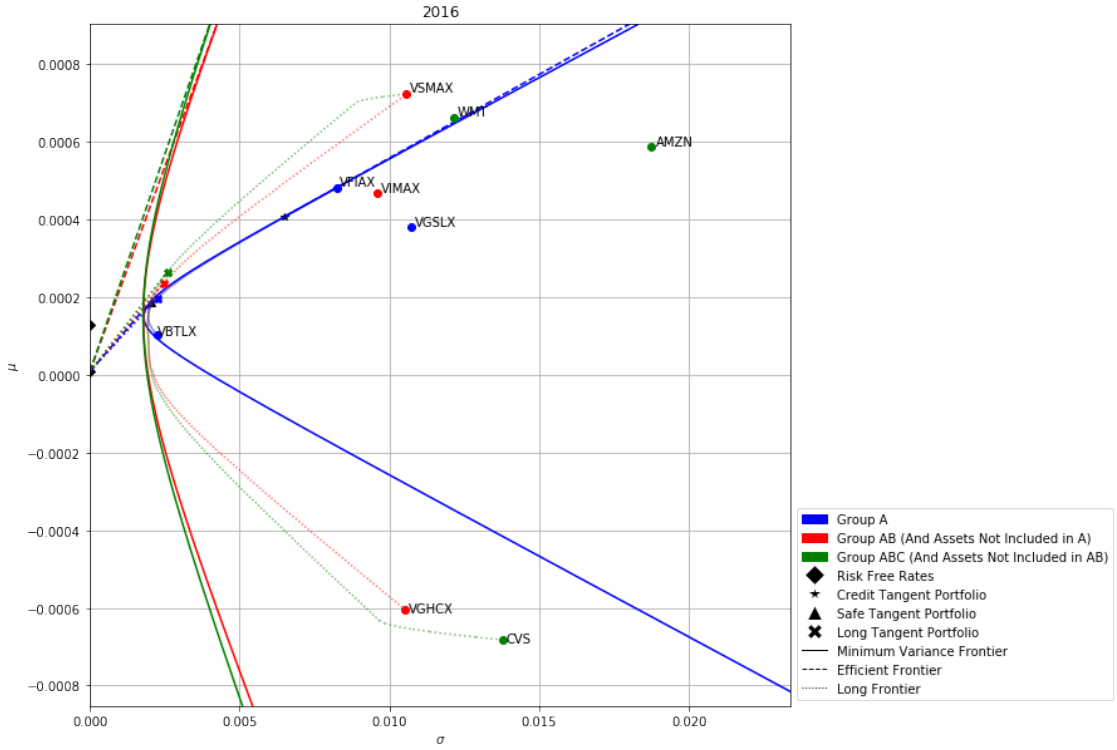
[49]:
```
lgd_groupA = mpatches.Patch(color='blue', label='Group A')
lgd_groupAB = mpatches.Patch(color='red', label='Group AB (And Assets Not␣
 ↪Included in A)')
lgd_groupABC = mpatches.Patch(color='green', label='Group ABC (And Assets Not␣
 ↪Included in AB)')
assets = Line2D([0], [0], marker='o', color='w', 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('-.')
```

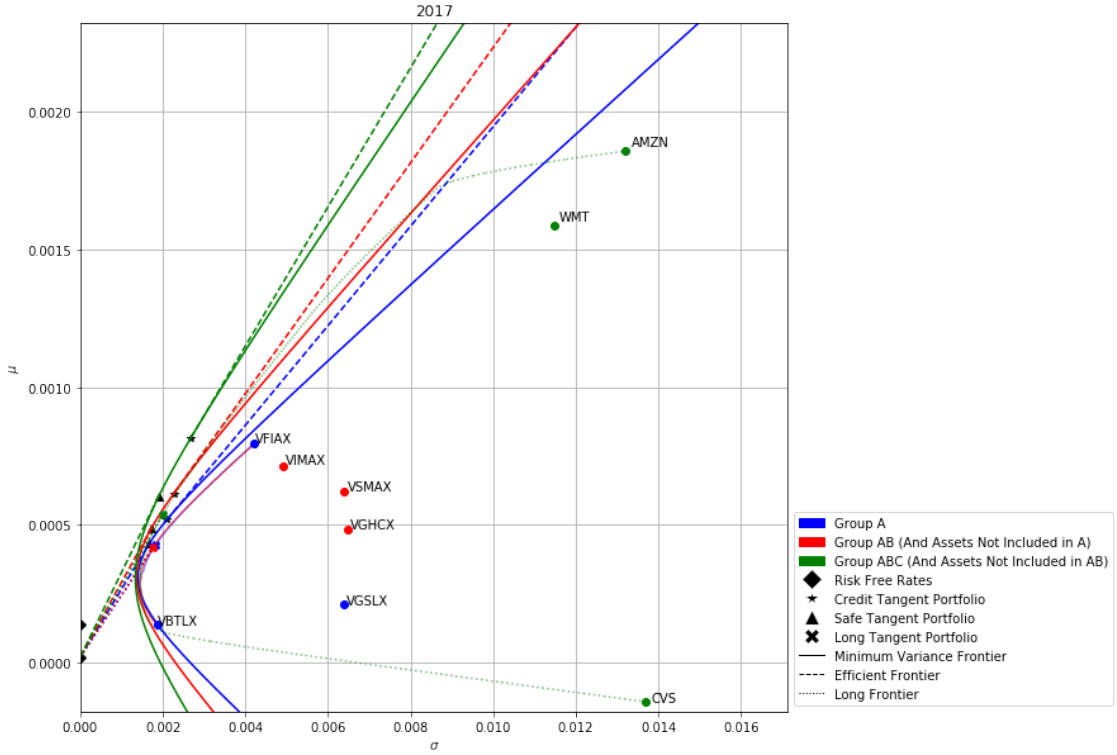
[50]:
```
# Creating efficient frontier graphs
for year_index, (start, end) in enumerate(years):

    period_start = start
    period_end = end

    year = start[:4]
```

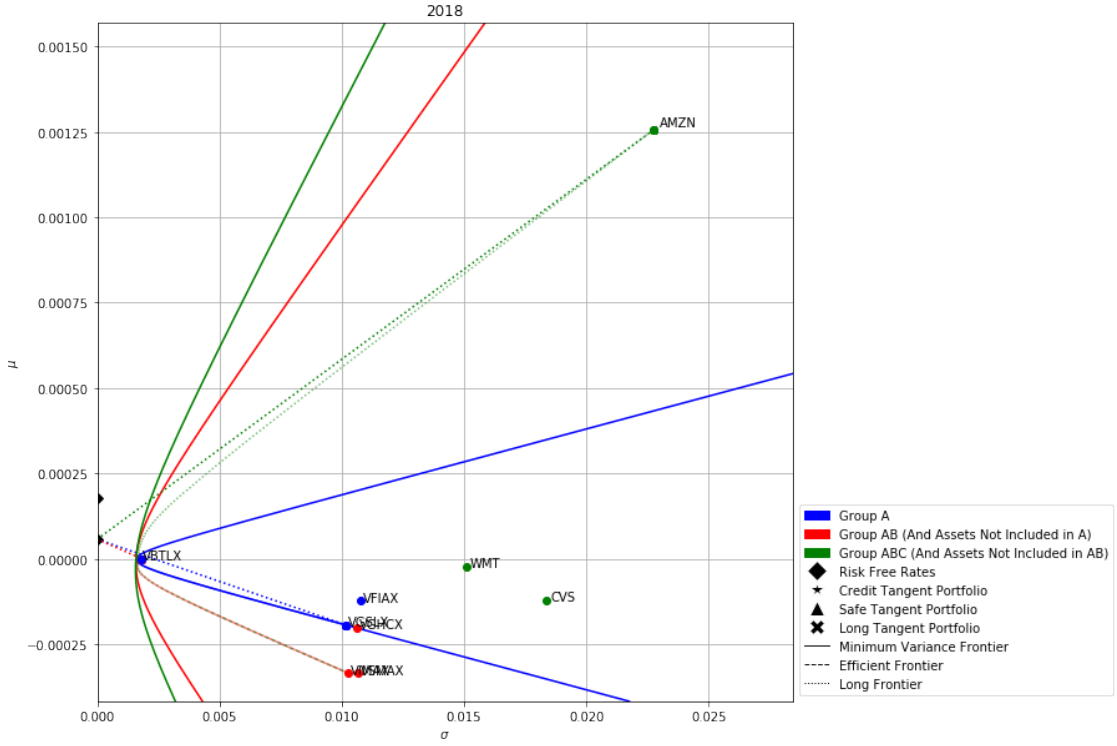
```
    plt.figure(num=year_index, figsize=(10,10))
    colors = ['b', 'r', 'g']

    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
    y_lim = m_all.min()*1.5

    plt.plot(0, mu_si[year], 'kD')
    plt.plot(0, mu_cl[year], 'kD')

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

        m,V = portfolio.get_mean_and_variance(period_start, period_end)

        # Efficient frontier calculations and plotting
        _, sigma_mv, mu_mv, nu_as = portfolio.
↪get_minimum_volatility_portfolio_parameters(period_start, period_end)

        st_vals = portfolio.get_safe_tangent_portfolio_parameters(mu_si[year],␣
↪period_start, period_end)
        ct_vals = portfolio.get_credit_tangent_portfolio_parameters(mu_cl[year],␣
↪period_start, period_end)

        if st_vals is not None:
            _, sigma_st, mu_st, nu_st = st_vals

            plt.plot(sigma_st, mu_st, 'k^')
            plt.plot([0, sigma_st], [mu_si[year], mu_si[year] + nu_st*sigma_st],␣
↪colors[group_number] + '--')

        if ct_vals is not None:
            _, sigma_ct, mu_ct, nu_ct = ct_vals

            plt.plot(sigma_ct, mu_ct, 'k*')
            plt.plot([sigma_ct, max_sigma], [mu_ct, mu_cl[year] +␣
↪nu_ct*max_sigma], colors[group_number] + '--')

        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)))
        plt.plot(frontier, mus, colors[group_number])
```

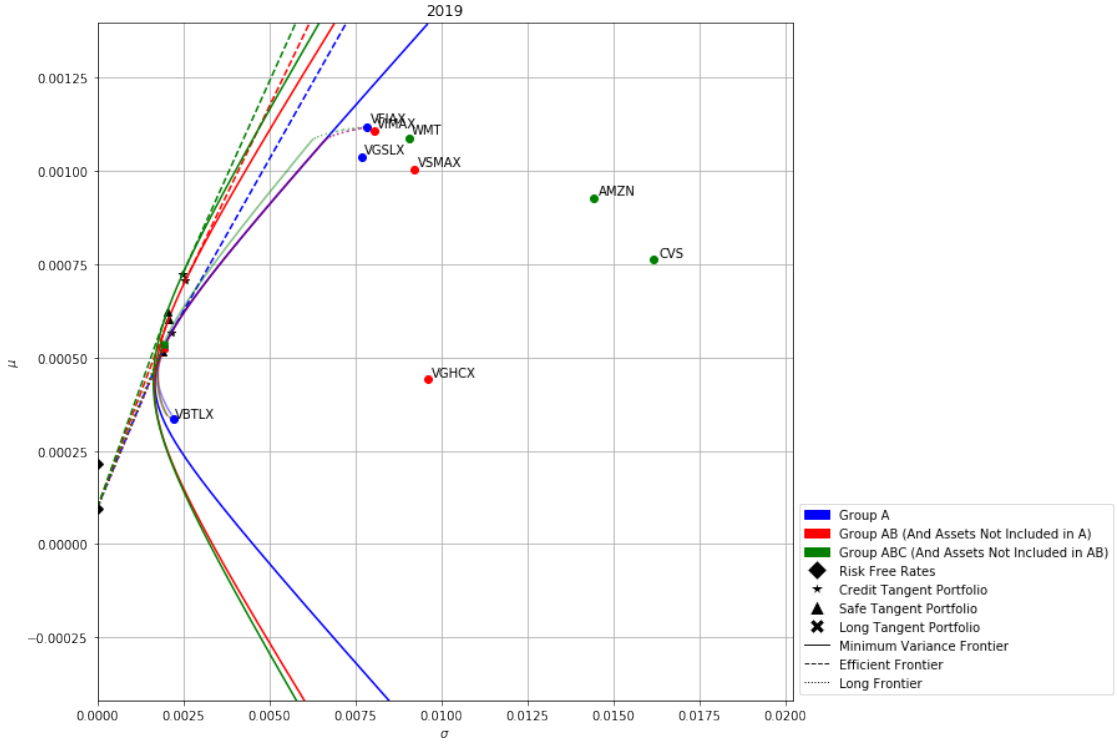
```
        # Efficient long frontier
        f_lt, sigma_lt, mu_lt, nu_lt = portfolio.
↪get_long_tangent_portfolio(mu_si[start[:4]], period_start, period_end)

        plt.plot(sigma_lt, mu_lt, colors[group_number] + 'X')
        plt.plot([0, sigma_lt], [mu_si[start[:4]], mu_lt], colors[group_number]␣
↪+ ':')

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

        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)/20),␣
↪(cur_mu-previous_mu)/20)

            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
```

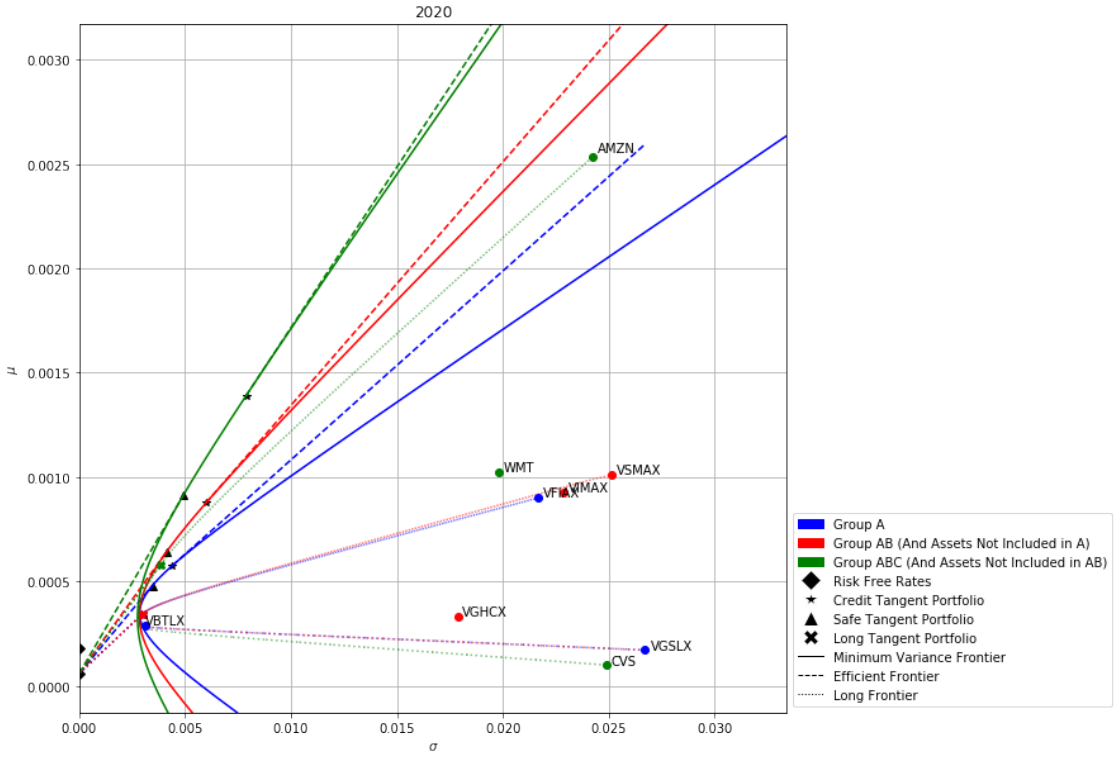
```
    for j in range(len(m_all)):
        plt.plot(V_all.iloc[j,j]**0.5, m_all[j], colors[int(j/3)] + 'o')
        plt.annotate(V_all.index[j], (1.01*V_all.iloc[j,j]**0.5, 1.01*m_all[j]))

    lgd = plt.legend(bbox_to_anchor=(1, 0), loc='lower left', ncol=1,␣
↪handles=[lgd_groupA, lgd_groupAB, lgd_groupABC, rf, ctg, stg, ltg,␣
↪mv_frontier, efficient_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])
    plt.grid()

    # plt.savefig('Graphs/longFrontiers/Year_' + str(year_index+1) + '.png',␣
↪bbox_extra_artists=(lgd,), bbox_inches='tight')
    plt.show()
```


2016
VSMAX
WMT
AMZN
VFIAX
VIMAX
VGSLX
VBTLX
VGHCX
CVS
μ
σ
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Credit Tangent Portfolio
Safe Tangent Portfolio
Long Tangent Portfolio
Minimum Variance Frontier
Efficient Frontier
Long Frontier


2017
AMZN
WMT
VFIAX
VIMAX
VSMAX
VGHCX
VGSLX
VBTLX
CVS
μ
σ
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Credit Tangent Portfolio
Safe Tangent Portfolio
Long Tangent Portfolio
Minimum Variance Frontier
Efficient Frontier
Long Frontier


2018
AMZN
WMT
CVS
VBTLX
VFIAX
VGSLX
VGHCX
VSMAX
μ
σ
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Credit Tangent Portfolio
Safe Tangent Portfolio
Long Tangent Portfolio
Minimum Variance Frontier
Efficient Frontier
Long Frontier

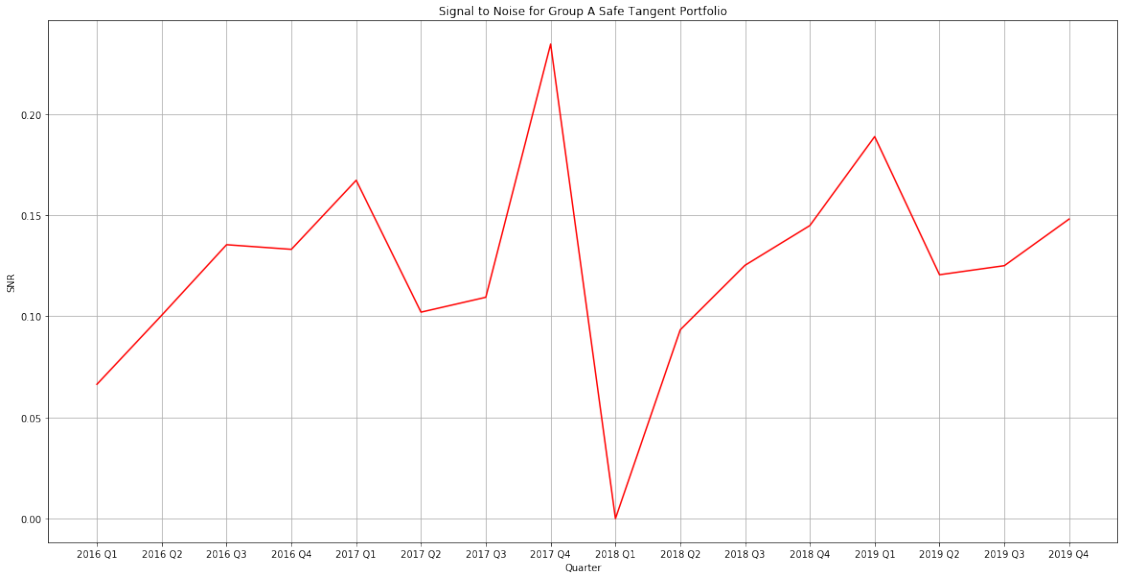

2019
VFIAX
WMT
VGSLX
VSMAX
AMZN
CVS
VGHCX
VBTLX
μ
σ
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Credit Tangent Portfolio
Safe Tangent Portfolio
Long Tangent Portfolio
Minimum Variance Frontier
Efficient Frontier
Long Frontier

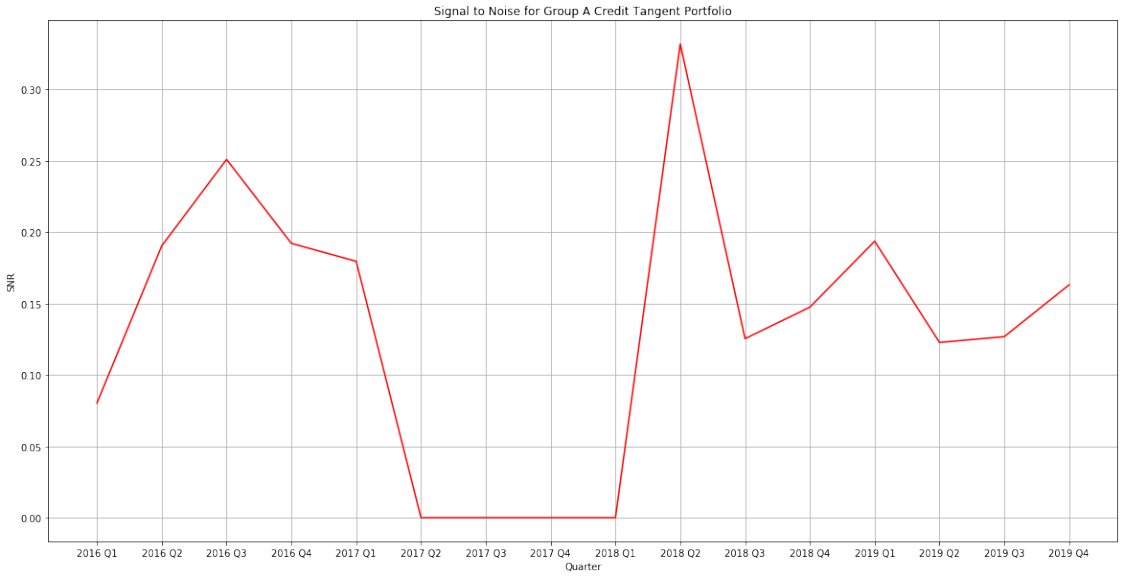

2020
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
μ
0.000
0.005
0.010
0.015
0.020
0.025
0.030
σ
AMZN
WMT
VSMAX
VFIAX
VIMAX
VGHCX
VBTLX
VGSLX
CVS
Group A
Group AB (And Assets Not Included in A)
Group ABC (And Assets Not Included in AB)
Risk Free Rates
Credit Tangent Portfolio
Safe Tangent Portfolio
Long Tangent Portfolio
Minimum Variance Frontier
Efficient Frontier
Long Frontier

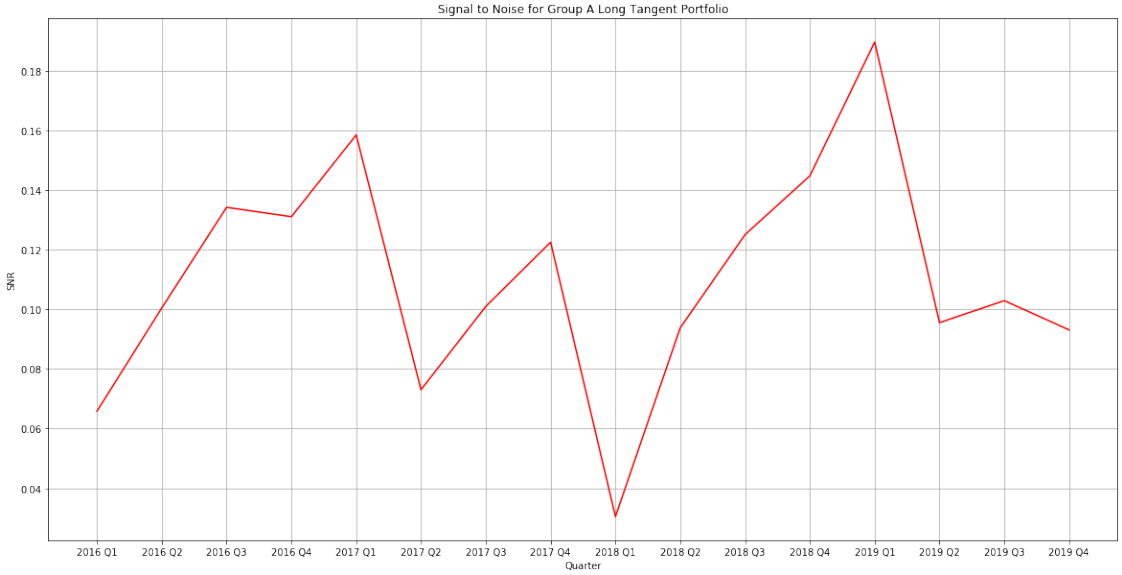
Exercise 2

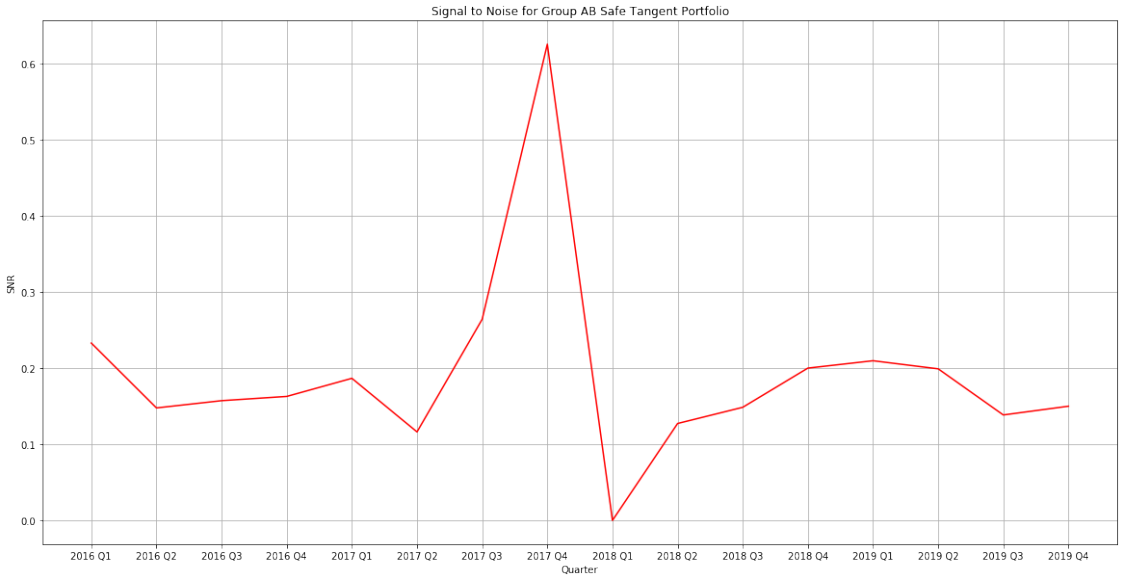
[51]:
```
for group_number, portfolio in enumerate([groupA, groupAB, groupABC]):
    results = [[], [], []]
    x_axis = []
    for year_index, (start, end) in enumerate(years[:-1]):

        cur_year = start[:4]

        for quarter_index, (q_start, q_end) in enumerate(quarters):

            period_start = start[:4]+ q_start
            period_end = str(int(cur_year) + 1) + quarters[quarter_index-1][1]␣
↪if quarter_index != 0 else cur_year + quarters[quarter_index-1][1]

            st_params = portfolio.
↪get_safe_tangent_portfolio_parameters(mu_si[start[:4]],␣
↪period_start=period_start, period_end=period_end)
            ct_params = portfolio.
↪get_credit_tangent_portfolio_parameters(mu_cl[start[:4]],␣
↪period_start=period_start, period_end=period_end)
            lt_params = portfolio.get_long_tangent_portfolio(mu_si[start[:4]],␣
↪period_start=period_start, period_end=period_end)

            for tangent_index, params in enumerate([st_params, ct_params,␣
↪lt_params]):

                if params is not None:
                    f, sigma, mu, _ = params
                    days = len(portfolio.returns[period_start:period_end])
                    uniformWeights = 1/ days
                    wBar = np.sum(np.square(np.ones(days) * uniformWeights))

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

                    signalToNoise = np.absolute(np.array(mean/
↪StdOfExpectedValue)).reshape(-1,1).reshape(-1,)
                    results[tangent_index].append(signalToNoise)

                else:
                    results[tangent_index].append(0)

                if tangent_index == 0:
                    x_axis.append(cur_year + ' Q' + str(quarter_index + 1))
```

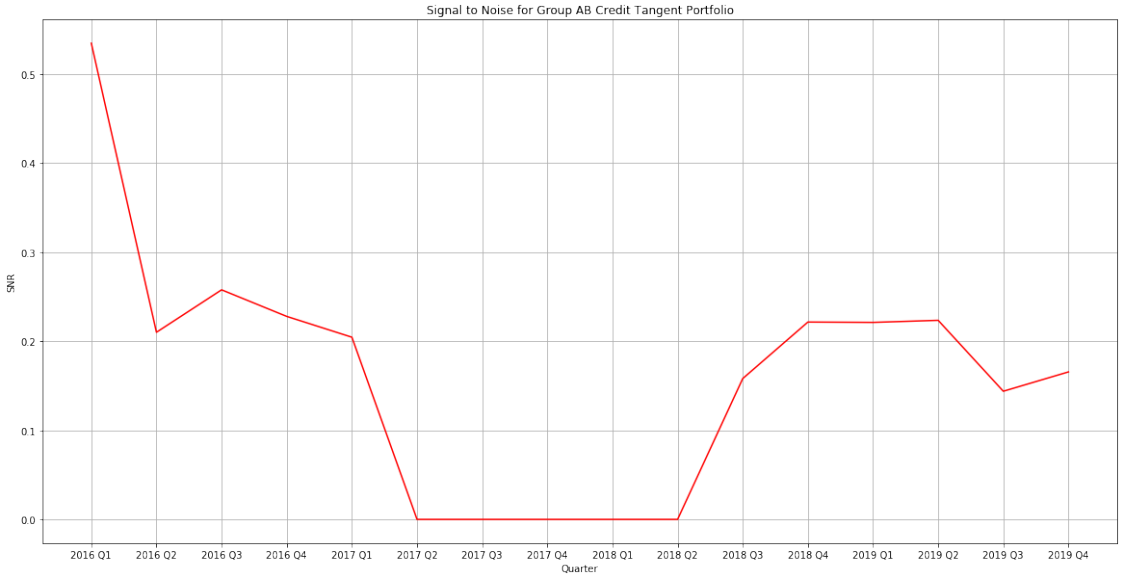
```
    plt.figure(num=group_number*3, figsize=(20,10))
    plt.plot(x_axis, results[0], 'r')
    plt.title('Signal to Noise for ' + portfolio.portfolio_name + ' Safe Tangent␣
↪Portfolio')
    plt.ylabel('SNR')
    plt.xlabel('Quarter')
    plt.grid()
    plt.show()

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

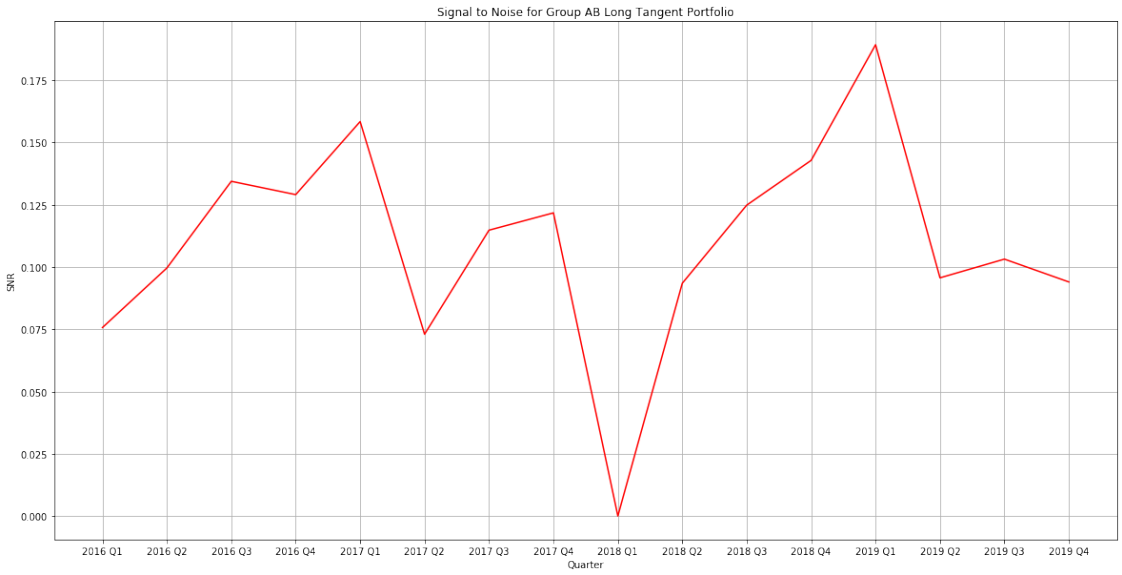

Signal to Noise for Group A Safe Tangent Portfolio
SNR
0.00
0.05
0.10
0.15
0.20
2016 Q1
2016 Q2
2016 Q3
2016 Q4
2017 Q1
2017 Q2
2017 Q3
2017 Q4
2018 Q1
2018 Q2
2018 Q3
2018 Q4
2019 Q1
2019 Q2
2019 Q3
2019 Q4
Quarter

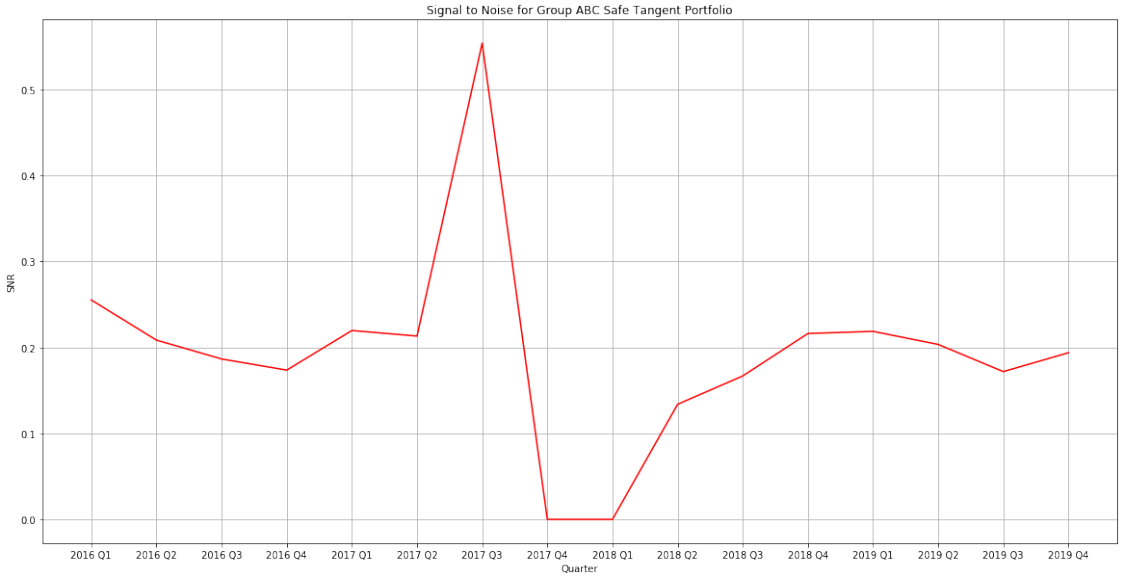

Signal to Noise for Group A Credit Tangent Portfolio
SNR
0.00
0.05
0.10
0.15
0.20
0.25
0.30
2016 Q1
2016 Q2
2016 Q3
2016 Q4
2017 Q1
2017 Q2
2017 Q3
2017 Q4
2018 Q1
2018 Q2
2018 Q3
2018 Q4
2019 Q1
2019 Q2
2019 Q3
2019 Q4
Quarter

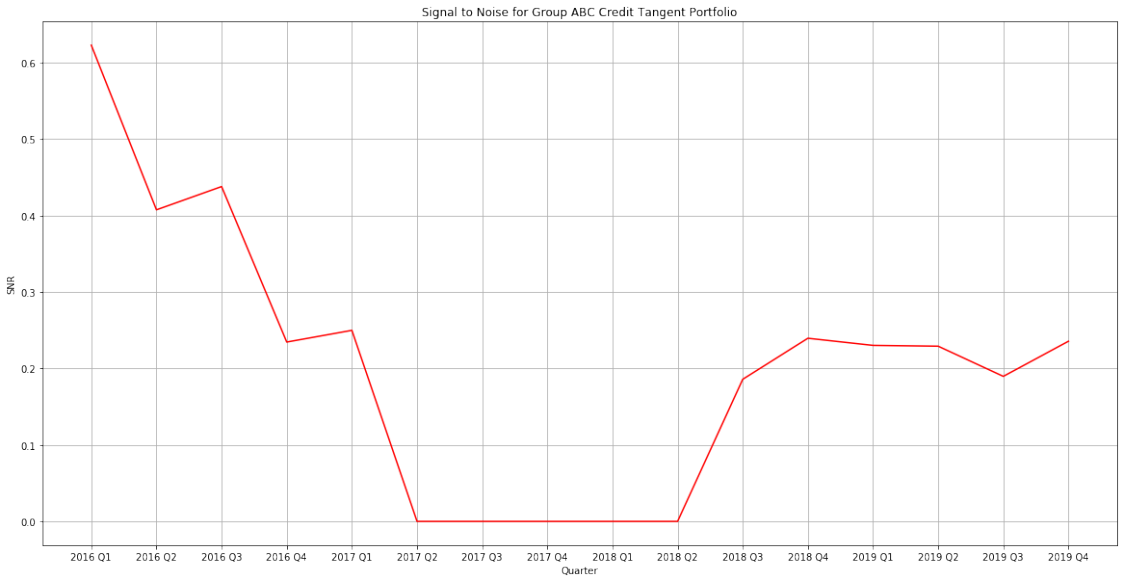

Signal to Noise for Group A Long Tangent Portfolio
SNR
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
2016 Q1
2016 Q2
2016 Q3
2016 Q4
2017 Q1
2017 Q2
2017 Q3
2017 Q4
2018 Q1
2018 Q2
2018 Q3
2018 Q4
2019 Q1
2019 Q2
2019 Q3
2019 Q4
Quarter


Signal to Noise for Group AB Safe Tangent Portfolio
SNR
0.6
0.5
0.4
0.3
0.2
0.1
0.0
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
Quarter


Signal to Noise for Group AB Credit Tangent Portfolio
SNR
0.5
0.4
0.3
0.2
0.1
0.0
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
Quarter


Signal to Noise for Group AB Long Tangent Portfolio
SNR
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
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
Quarter


Signal to Noise for Group ABC Safe Tangent Portfolio
SNR
0.0
0.1
0.2
0.3
0.4
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
Quarter


Signal to Noise for Group ABC Credit Tangent Portfolio
SNR
0.0
0.1
0.2
0.3
0.4
0.5
0.6
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
Quarter

Comparing Results to individual assets:

As a whole the signal to noise ratio for each portfolio is significantly lower than the signal to noise values for individual assets. Also something to note, the ratios of mu to sigma for the portfolios as shown in exercise one are generally lower than the individual assets. Looking at portfolios specifically the credit tangent portfolios specifically seems to be most prone to not existing, with several portfolios not existing around 2017-2018. Interestingly, the long tangent portfolios seem to have a dip in the signal to noise ratio around the start of 2018.

Which portfolio is the most certain:

To see which portfolio has the most certain outcomes, we look at the signal to noise ratio for each of the quarters. Of the 9 graphs, the ABC Credit Tangent portfolio has the largest average and most consistent signal to noise ratio, also with the highest peak. The ABC Safe and AB Credit portfolios are the next highest signal to noise, with AB Safe, ABC Long, with the next highest group both having similar values to each other, and then AB Long, A Credit, A Long, and A Safe all have the lowest signal to noise ratios.

Exercise 3

[52]:
```
print(groupA.get_safe_tangent_portfolio_parameters(mu_si['2017'],␣
↪period_start='2017-01-01', period_end='2017-12-31'))
```

```
VFIAX    0.458007
VBTLX    0.657486
VGSLX   -0.115493
```

[53]:
```
def get_long_tangent_portfolio(mu_si, returns):
    m = returns.mean()
    V = returns.cov()

    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)

    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

def get_safe_tangent_portfolio_parameters(mu_si, returns):

    m = returns.mean()
    V = returns.cov()
```

```
    sigma_mv, mu_mv, nu_as = compute_mv_vars(m, V)
    y = compute_y(V)
    f_mv = pd.Series(data=(sigma_mv**2)*np.array(y))

    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=returns.columns), sigma_st, mu_st, nu_st

def get_credit_tangent_portfolio_parameters(mu_cl, returns):

    m = returns.mean()
    V = returns.cov()

    sigma_mv, mu_mv, nu_as = compute_mv_vars(m, V)
    y = compute_y(V)
    f_mv = pd.Series(data=(sigma_mv**2)*np.array(y))

    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=returns.columns), sigma_ct, mu_ct, nu_ct
```

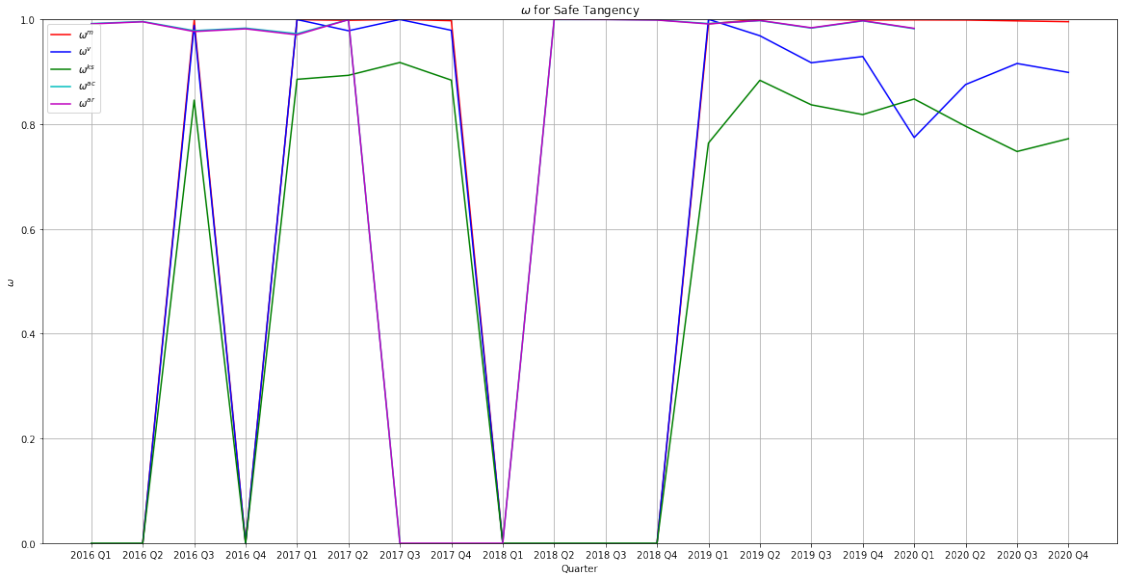
[54]:
```
from scipy import stats

for group_number, portfolio in enumerate([groupA, groupAB, groupABC]):
    omega_m_results = [[],[],[]]
    omega_v_results = [[],[],[]]
```

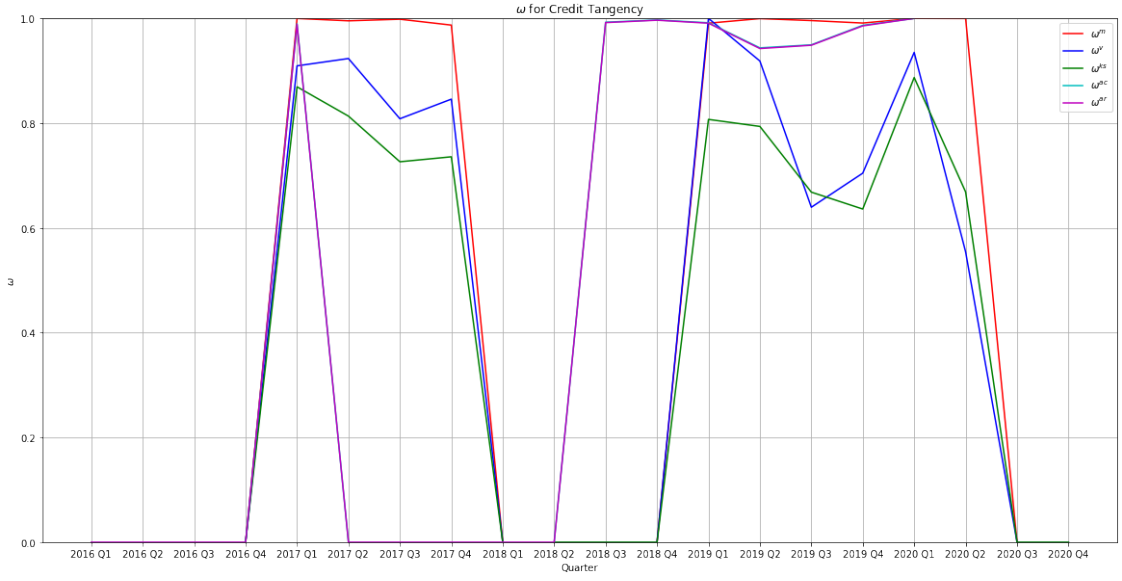
```
    omega_ks_results = [[],[],[]]
    omega_ar_results = [[],[],[]]
    omega_ac_results = [[],[],[]]
    x_axis = []
    ar_ac_labels = []
    for (year_start, year_end) in years:
        cur_year = year_start[:4]
        portfolio_yearly_returns = portfolio.returns[year_start:year_end]

        for quarter_index, (q_start, q_end) in enumerate(quarters):

            x_axis.append(cur_year + ' Q' + str(quarter_index+1))

            cur_quarter_returns =portfolio_yearly_returns[cur_year + q_start:
↪cur_year + q_end]
            other_quarters_returns = portfolio_yearly_returns.
↪drop(cur_quarter_returns.index)

            st_params = (get_safe_tangent_portfolio_parameters(mu_si[start[:4]],␣
↪cur_quarter_returns), get_safe_tangent_portfolio_parameters(mu_si[start[:4]],␣
↪other_quarters_returns))
            ct_params = (get_credit_tangent_portfolio_parameters(mu_cl[start[:
↪4]], cur_quarter_returns),␣
↪get_credit_tangent_portfolio_parameters(mu_cl[start[:4]],␣
↪other_quarters_returns))
            lt_params = (get_long_tangent_portfolio(mu_si[start[:4]],␣
↪cur_quarter_returns), get_long_tangent_portfolio(mu_si[start[:4]],␣
↪other_quarters_returns))


            for tangent_index, (cur_quarter, other_quarters) in␣
↪enumerate([st_params, ct_params, lt_params]):

                if cur_quarter is not None and other_quarters is not None:
                    f1, v1, m1, _ = cur_quarter
                    D1 = len(cur_quarter_returns)

                    f2, v2, m2, _ = other_quarters
                    D2 = len(other_quarters_returns)

                    omega_m = 1/(1 + (m1 - m2)**2/(v1/D1 + v2/D2))
                    omega_m_results[tangent_index].append(omega_m)

                    omega_v = (4*v1*v2)/(v1+v2)**2
                    omega_v_results[tangent_index].append(omega_v)
```

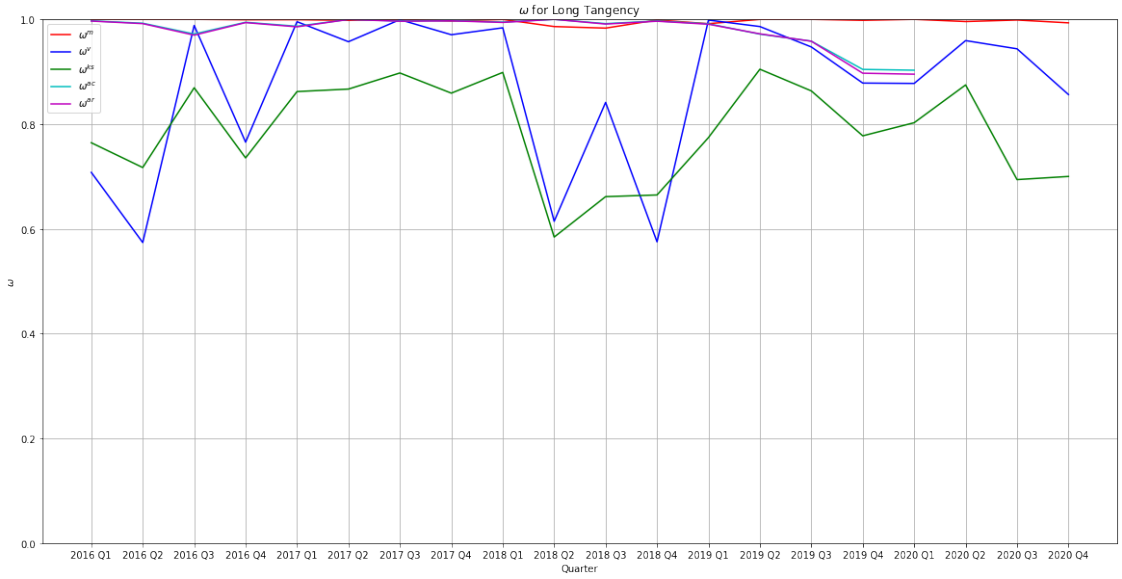
```
                    omega_ks = 1 - stats.ks_2samp((cur_quarter_returns*f1).
↪mean(axis=1), (other_quarters_returns*f2).mean(axis=1))[0]
                    omega_ks_results[tangent_index].append(omega_ks)

                else:
                    omega_m_results[tangent_index].append(0)
                    omega_v_results[tangent_index].append(0)
                    omega_ks_results[tangent_index].append(0)

            if year_start[:4] != '2020' or (year_start[:4] == '2020' and␣
↪quarter_index == 0):

                year_diff = 1

                if quarter_index == 0:
                    year_diff = 0

                period_start = cur_year + q_start
                period_end = str(int(cur_year) + year_diff) +␣
↪quarters[quarter_index-1][1]

                cur_period = portfolio.returns[period_start:period_end]

                st_params = get_safe_tangent_portfolio_parameters(mu_si[start[:
↪4]], cur_period)
                ct_params = get_credit_tangent_portfolio_parameters(mu_cl[start[:
↪4]], cur_period)
                lt_params = get_long_tangent_portfolio(mu_si[start[:4]],␣
↪cur_period)

                for tangent_index, param in enumerate([st_params, ct_params,␣
↪lt_params]):

                    if param is not None:
                        f, _, _, _ = param

                        cur_period = cur_period*f

                        m0 = cur_period.iloc[1:].mean().mean()
                        m1 = cur_period.iloc[:-1].mean().mean()

                        v00 = 0
                        v10 = 0
                        v11 = 0

                        for _, day_return in cur_period.iloc[1:].iterrows():
```

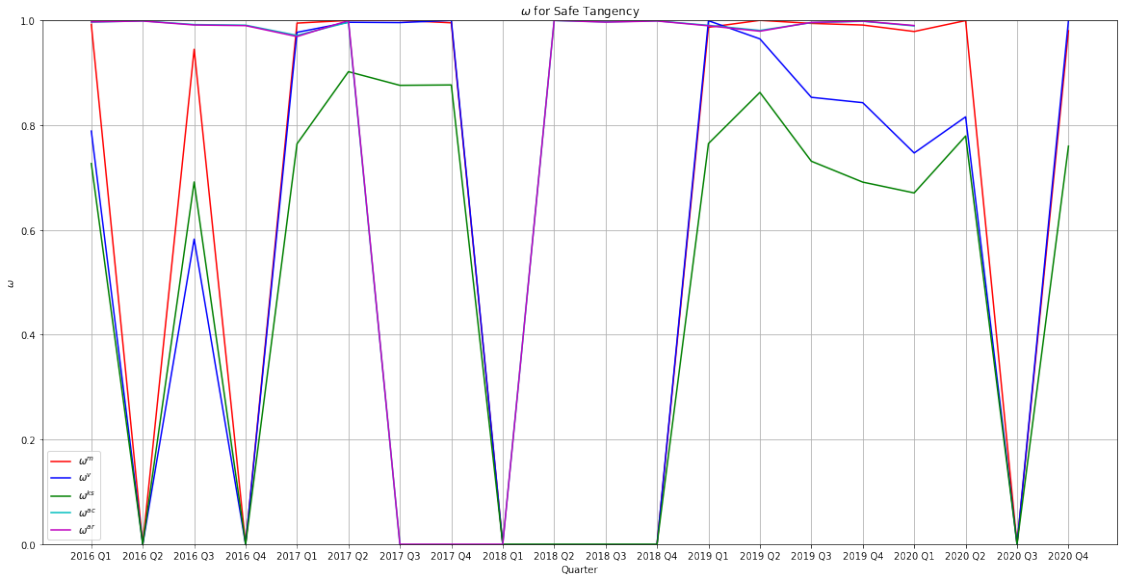
```
                            v00 += (day_return.mean() - m0)**2

                        for _, day_return in cur_period.iloc[:-1].iterrows():
                            v11 += (day_return.mean() - m1)**2

                        for day in range(1, len(cur_period)):
                            v10 += (cur_period.iloc[day-1].mean() -␣
↪m0)*(cur_period.iloc[day].mean() - m1)


                        v11 = v11/len(cur_period)
                        v10 = v10/len(cur_period)
                        v00 = v00/len(cur_period)
                        W_mat = np.matrix([[1 - 1/len(cur_period),␣
↪-(len(cur_period)- 1)/ (len(cur_period) * len(cur_period))],␣
↪[-(len(cur_period)- 1)/ (len(cur_period) * len(cur_period)), 1 - 1/
↪len(cur_period)]])
                        V_mat = np.matrix([[v00, v10], [v10, v11 ]])

                        omega_ar = 1 - (v10**2)/(v00*v11)
                        omega_ac = np.square(np.trace(np.matmul(W_mat, V_mat)))/␣
↪(np.trace(np.matmul(V_mat, V_mat)) * np.trace(np.matmul(W_mat, W_mat)))

                        omega_ar_results[tangent_index].append(omega_ar)
                        omega_ac_results[tangent_index].append(omega_ac)

                    else:
                        omega_ar_results[tangent_index].append(0)
                        omega_ac_results[tangent_index].append(0)

                    if tangent_index == 0:
                        ar_ac_labels.append(cur_year + ' Q' +␣
↪str(quarter_index+1))


   plt.figure(num=group_number*3, figsize=(20,10))
   plt.plot(x_axis, omega_m_results[0], 'r', label='$\omega^m$')
   plt.plot(x_axis, omega_v_results[0], 'b', label='$\omega^v$')
   plt.plot(x_axis, omega_ks_results[0], 'g', label='$\omega^{ks}$')
   plt.plot(ar_ac_labels, omega_ac_results[0], 'c', label='$\omega^{ac}$')
   plt.plot(ar_ac_labels, omega_ar_results[0], 'm', label='$\omega^{ar}$')
   plt.title('$\omega$ for Safe Tangency')
   plt.ylabel('$\omega$')
   plt.xlabel('Quarter')
   plt.ylim((0,1))
   plt.legend()
   plt.grid()
```

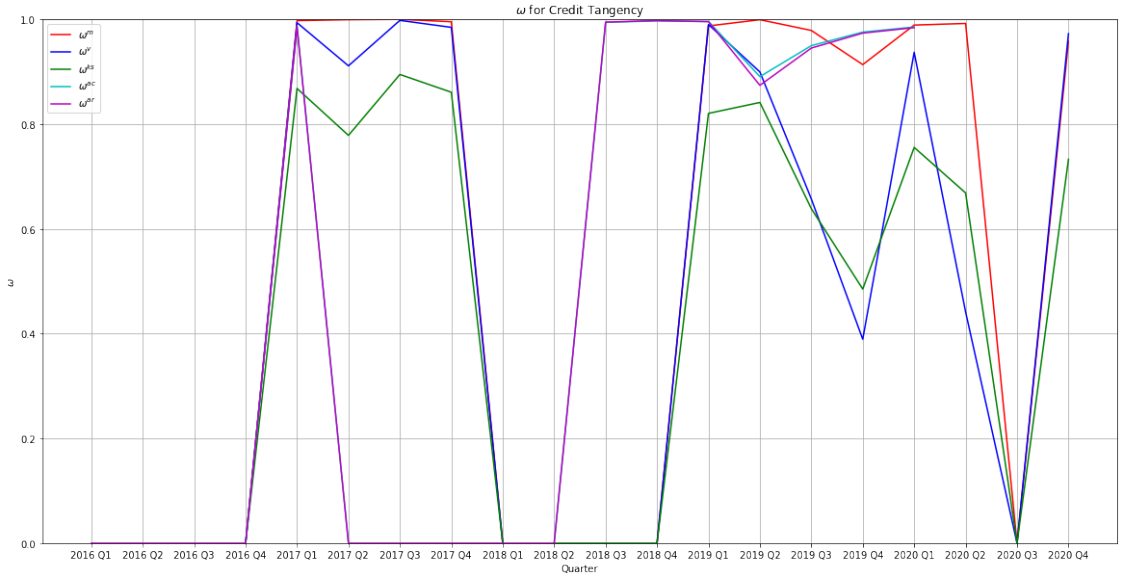
```
    plt.show()

    plt.figure(num=group_number*3 + 1, figsize=(20,10))
    plt.plot(x_axis, omega_m_results[1], 'r', label='$\omega^m$')
    plt.plot(x_axis, omega_v_results[1], 'b', label='$\omega^v$')
    plt.plot(x_axis, omega_ks_results[1], 'g', label='$\omega^{ks}$')
    plt.plot(ar_ac_labels, omega_ac_results[1], 'c', label='$\omega^{ac}$')
    plt.plot(ar_ac_labels, omega_ar_results[1], 'm', label='$\omega^{ar}$')
    plt.title('$\omega$ for Credit Tangency')
    plt.ylabel('$\omega$')
    plt.xlabel('Quarter')
    plt.ylim((0,1))
    plt.legend()
    plt.grid()
    plt.show()

    plt.figure(num=group_number*3 + 2, figsize=(20,10))
    plt.plot(x_axis, omega_m_results[2], 'r', label='$\omega^m$')
    plt.plot(x_axis, omega_v_results[2], 'b', label='$\omega^v$')
    plt.plot(x_axis, omega_ks_results[2], 'g', label='$\omega^{ks}$')
    plt.plot(ar_ac_labels, omega_ac_results[2], 'c', label='$\omega^{ac}$')
    plt.plot(ar_ac_labels, omega_ar_results[2], 'm', label='$\omega^{ar}$')
    plt.title('$\omega$ for Long Tangency')
    plt.ylabel('$\omega$')
    plt.xlabel('Quarter')
    plt.ylim((0,1))
    plt.legend()
    plt.grid()
    plt.show()
```

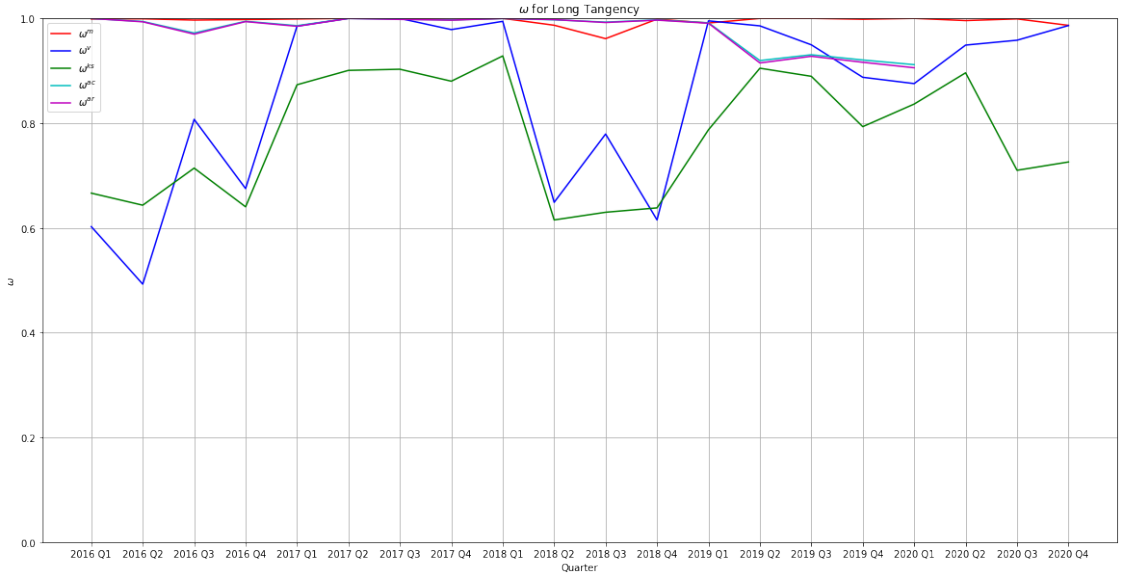

ω for Safe Tangency
ω
Quarter
2016 Q1
2016 Q2
2016 Q3
2016 Q4
2017 Q1
2017 Q2
2017 Q3
2017 Q4
2018 Q1
2018 Q2
2018 Q3
2018 Q4
2019 Q1
2019 Q2
2019 Q3
2019 Q4
2020 Q1
2020 Q2
2020 Q3
2020 Q4
0.0
0.2
0.4
0.6
0.8
1.0

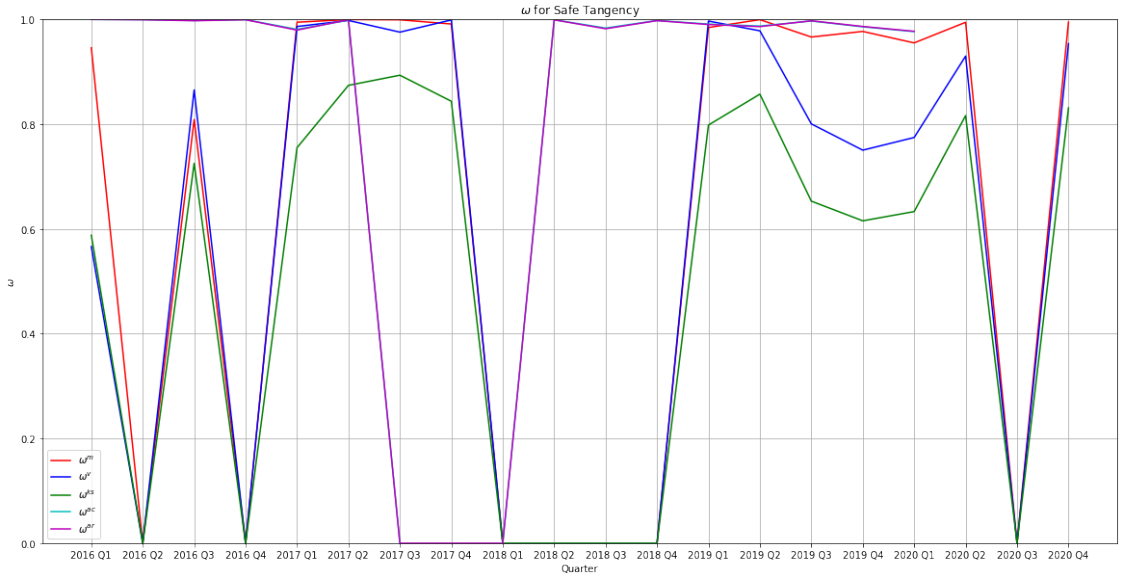

ω for Credit Tangency
ω
Quarter
2016 Q1
2016 Q2
2016 Q3
2016 Q4
2017 Q1
2017 Q2
2017 Q3
2017 Q4
2018 Q1
2018 Q2
2018 Q3
2018 Q4
2019 Q1
2019 Q2
2019 Q3
2019 Q4
2020 Q1
2020 Q2
2020 Q3
2020 Q4
0.0
0.2
0.4
0.6
0.8
1.0

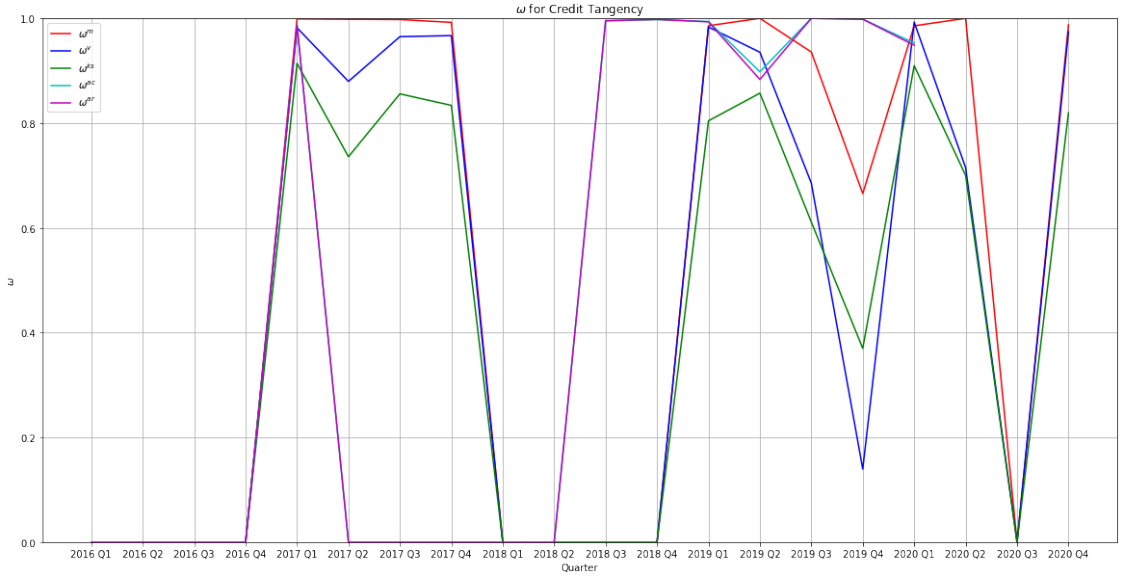

ω for Long Tangency
ω
Quarter
2016 Q1
2016 Q2
2016 Q3
2016 Q4
2017 Q1
2017 Q2
2017 Q3
2017 Q4
2018 Q1
2018 Q2
2018 Q3
2018 Q4
2019 Q1
2019 Q2
2019 Q3
2019 Q4
2020 Q1
2020 Q2
2020 Q3
2020 Q4
0.0
0.2
0.4
0.6
0.8
1.0

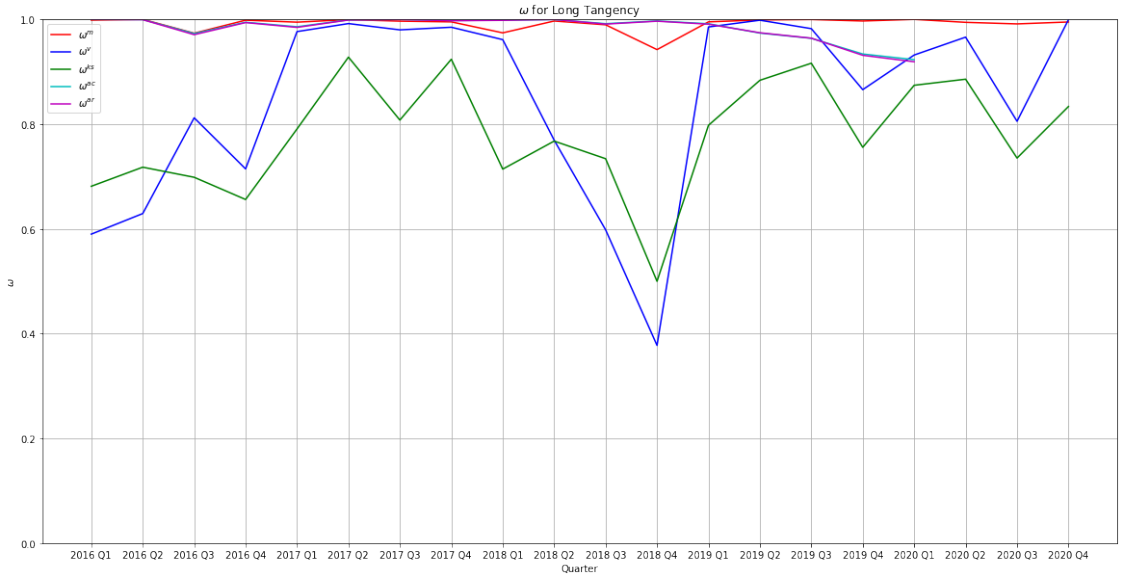

ω for Safe Tangency
ω
Quarter
2016 Q1
2016 Q2
2016 Q3
2016 Q4
2017 Q1
2017 Q2
2017 Q3
2017 Q4
2018 Q1
2018 Q2
2018 Q3
2018 Q4
2019 Q1
2019 Q2
2019 Q3
2019 Q4
2020 Q1
2020 Q2
2020 Q3
2020 Q4
0.0
0.2
0.4
0.6
0.8
1.0


ω for Credit Tangency
ω
Quarter
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
2020 Q2
2020 Q3
2020 Q4
0.0
0.2
0.4
0.6
0.8
1.0


ω for Long Tangency
ω
Quarter
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
2020 Q2
2020 Q3
2020 Q4
0.0
0.2
0.4
0.6
0.8
1.0


ω for Safe Tangency
ω
Quarter
1.0
0.8
0.6
0.4
0.2
0.0
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
2020 Q2
2020 Q3
2020 Q4


ω for Credit Tangency
ω
Quarter
1.0
0.8
0.6
0.4
0.2
0.0
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
2020 Q2
2020 Q3
2020 Q4



Comparing Results to individual assets:

When it comes to ω^{ks}, ω^{ac}, ω^{ar} they seem to have similar results to the individual assets. However with ω^{m} and ω^{v} the portfolio values as a whole seem to be larger than the individual assets, when the given portfolio exists. This difference could be due to the diversification of the portfolio averages out the volatility that could cause the mean and variances to change. Something to note is that the tangent portfolios have gaps where meaningful results are outputted, however these gaps do not exist in the long efficient portfolio.

Which of these portfolios is best described by an IID model?

Comparing the IID models for how good of a model they are, we look for closer to 1 values of our ω's. For each of the three portfolios, the Long Tangent portfolios all have consistent ω values with almost all values over 0.6 for all time periods, while other portfolios have 0 for multiple time periods. After that the Safe Tangency has higher values than the Credit tangent portfolios. This holds for all of the portfolios A AB and ABC.