Fintech545 Project Week7

Kaiye Chen

November 2022

1 Basic Functions

1.1 Greeks

```
def BS_Greeks(S,K,T,r,q,sigma,option_dir):
    d1 = (np.log(S/K)+(r-q+sigma**2/2)*T)/(sigma*np.sqrt(T))
    d2 = d1-sigma*np.sqrt(T)
    def delta():
        factor = np.exp((-q * T))
        if option_dir == "Call":
            return factor * norm.cdf(d1)
        else:
            return factor * (norm.cdf(d1) - 1)

    def gamma():
        return np.exp(-q * T) * norm.pdf(d1) / (S * sigma * np.sqrt(T))

    def vega():
        return S * np.exp(-q * T) * norm.pdf(d1) * np.sqrt(T) / 100
```

```
        opt = -1
    df = np.exp(-r * T)
    dfq = np.exp(-q * T)
    tmptheta = -0.5 * S * dfq * norm.pdf(d1) * sigma / np.sqrt(T) + opt * q * S * dfq * nor
    return tmptheta / 365

def rho():
    if option_dir == "Call":
        sign = 1
    else:
        sign = -1
    df = np.exp(-r * T)
    return sign * T * K * df * norm.cdf(sign * d2) / 100

def phi():
    if option_dir == "Call":
        sign = 1
    else:
        sign = -1
    dfq = np.exp(-q * T)
    return -sign * T * S * dfq * norm.cdf(sign * d1) / 100

greek_frame = pd.DataFrame(np.array([delta(), gamma(), vega(), theta(), rho(), phi()]),
                           index=['delta', 'gamma', 'vega', 'theta', 'rho', 'phi'], columns=['GBSM'])
```

```
def FD_Greeks(S,K,T,r,q,sigma,option_dir,difference):
    dif_1 = difference
    dif_2 = difference/10
    def delta(ds = dif_1):
        u = black_scholes(S + ds, K, T, r, q, sigma, option_dir)
        d = black_scholes(S - ds, K, T, r, q, sigma, option_dir)
        return 0.5 * (u - d) / ds

    def gamma(ds = dif_1):
        u = black_scholes(S + ds, K, T, r, q, sigma, option_dir)
        m = black_scholes(S, K, T, r, q, sigma, option_dir)
        d = black_scholes(S - ds, K, T, r, q, sigma, option_dir)
        return (u - 2 * m + d) / ds / ds

    def vega(dsig = dif_2):
        u = black_scholes(S, K, T, r, q, sigma + dsig, option_dir)
        d = black_scholes(S, K, T, r, q, sigma - dsig, option_dir)
        return 0.5 * (u - d) / dsig / 100

    def theta(dt = dif_2):
```

```
def theta(dt = dif_2):
    u = black_scholes(S, K, T - dt, r, q, sigma, option_dir)
    d = black_scholes(S, K, T + dt, r, q, sigma, option_dir)
    return 0.5 * (u - d) / dt / 365

def rho(dr = dif_2):
    u = black_scholes(S, K, T, r + dr, q, sigma, option_dir)
    d = black_scholes(S, K, T, r - dr, q, sigma, option_dir)
    return 0.5 * (u - d) / dr / 100

def phi(db = dif_2):
    u = black_scholes(S, K, T, r, q + db, sigma, option_dir)
    d = black_scholes(S, K, T, r, q - db, sigma, option_dir)
    return 0.5 * (u - d) / db / 100

greek_frame = pd.DataFrame(np.array([delta(), gamma(), vega(), theta(), rho(), phi()]),
                           index=['delta', 'gamma', 'vega', 'theta', 'rho', 'phi'], columns=['FDM'])
return greek_frame
```

In this functions, I used to method to calculate greek, one straight-forward and one using difference.

1.2 American Binary Tree

```
def american_binary_tree(underlying,strike,ttm,rf,ivol,N,option_dir,div_amount,div_time):
    dt = ttm/N
    u = np.exp(ivol*np.sqrt(dt))
    d = 1/u
    pu = (np.exp(rf*dt)-d)/(u-d)
    pd = 1.0-pu
    df = np.exp(-rf*dt)
    if option_dir == "Call":
        z = 1
    else:
        z = -1

    if not div_amount or not div_time or div_time[0] > N:
        return binary_tree(underlying,strike,ttm,rf,0,ivol,N,option_dir)

    def nodes_fuc(n):
        nodes_count = (n+1)*(n+2)/2
        return int(nodes_count)

    def index_fuc(i,j):
        nodes_count = nodes_fuc(j-1)
        index = nodes_count + i - 1
```

```
def index_fuc(i,j):
    nodes_count = nodes_fuc(j-1)
    index = nodes_count + i - 1
    return index

total_nodes = nodes_fuc(div_time[0])
option_values = np.zeros(total_nodes)

for j in range(div_time[0],-1,-1):
    for i in range(j,-1,-1):
        idx = index_fuc(i,j)
        price = underlying*u**i*d**(j-i)
        if j < (div_time[0]-1):
            option_values[idx] = max(0, z*(price-strike))
            option_values[idx] = max(option_values[idx], df*(pu*option_values[index_fuc(i+1,j+1)] + pd*option_values[index_fuc
        else:
            valNoExercise = american_binary_tree(price-div_amount[0], strike, ttm-div_time[0]*dt, rf, ivol, N-div_time[0],
            valExercise = max(0,z*(price-strike))
            option_values[idx] = max(valNoExercise,valExercise)
return option_values[0]
```

In this function, I replicate the function in Julia to calculate American option value with binary tree.

1.3 Efficient Frontier

```
def effFrontier(mu, cov, scope, rf, shortsell):
    def portfolio_stats(weights):
        weights = np.array(weights)[:, np.newaxis]
        port_rets = weights.T @ mu
        port_vols = np.sqrt(weights.T @ cov @ weights)

        return np.array([port_rets, port_vols, (port_rets - rf) / port_vols]).flatten()

    # Maximizing sharpe ratio/ method to find tangency portfolio
    def min_sharpe_ratio(weights):
        return -portfolio_stats(weights)[2]

    # Minimize the volatility
    def min_volatility(weights):
        return portfolio_stats(weights)[1]

    numOfAssets = len(mu)
    cons = ({'type': 'eq', 'fun': lambda x: sum(x) - 1})
    stocks = mu.index
    mu = np.array(mu)
```

```
cov = np.array(cov)
# print(mu)
initial_wts = np.array(numOfAssets * [1. / numOfAssets])

# find efficient frontier
targetrets = np.linspace(scope[0], scope[1], scope[2])
tvols = []

for tr in targetrets:

        # for short sale forbid constraints
        bnds = tuple((0, 1) for x in range(numOfAssets))
        ef_cons = ({'type': 'eq', 'fun': lambda x: portfolio_stats(x)[0] - tr},
                           {'type': 'eq', 'fun': lambda x: sum(x) - 1})

        if shortsell == True:
                opt_ef = sco.minimize(min_volatility, initial_wts, method='SLSQP', constraints=ef_cons)
        else:
                opt_ef = sco.minimize(min_volatility, initial_wts, method='SLSQP', bounds=bnds, constraints=ef_cons)

        tvols.append(opt_ef['fun'])
targetvols = np.array(tvols)
# Dataframe for EF
efPort = pd.DataFrame({
```

```
        tvols.append(opt_ef['fun'])
targetvols = np.array(tvols)
# Dataframe for EF
efPort = pd.DataFrame({
        'targetrets': np.around(targetrets, 4),
        'targetvols': np.around(targetvols, 4),
        'targetsharpe': np.around((targetrets - rf) / targetvols, 4)
})
bnds = tuple((0, 1) for x in range(numOfAssets))
# Tangency Portfolio
if shortsell == True:
        opt_sharpe = sco.minimize(min_sharpe_ratio, initial_wts, method='SLSQP', constraints=cons)
else:
        opt_sharpe = sco.minimize(min_sharpe_ratio, initial_wts, method='SLSQP', bounds=bnds, constraints=cons)

w = opt_sharpe.x
stat = portfolio_stats(w)
Tangencyrets, Tangencyvols, Tangencysharpe = stat[0], stat[1], stat[2]
tangencyPort = pd.Series([Tangencyrets, Tangencyvols, Tangencysharpe], index=['ret', 'vol', 'sharpe'])
assetWeight = pd.DataFrame(w, index=stocks, columns=['Weight'])
return efPort, tangencyPort, assetWeight
```

In this function, I calculate efficient frontier with shortsell allowed and not allowed. First minimize the volatility with return set and maximize sharpe ratio to find the optimize portfolio.

2 Question 1

2.1 Greeks Result for European Option

	GBSM	GBSM	FDM	FDM
delta	0.510071	0.510071	0.510071	0.510071
gamma	0.040173	0.040173	0.040173	0.040173
vega	0.197766	0.197766	0.197766	0.197766
theta	-0.059256	-0.059256	-0.059257	-0.059257
rho	0.072533	0.072533	0.072533	0.072533
phi	-0.076091	-0.076091	-0.076091	-0.076091

In this part, I used the function defined above to calculate the greeks under two different method. The results under BSM and finite difference are very closed, which means this two method are almost the same in the context of results.

2.2 Result for American Option

```
]   American Call    American Put
    --------------  -------------
          3.18013         4.75618
```

```
BTcall  =  American_BT_Greeks(165, 165, T, r, sigma, N, "Call", 0.01, div_amoun
BTput  =  American_BT_Greeks(165, 165, T, r, sigma, N, "Put", 0.01, div_amount,
allgreek  =  pd.concat([BTcall, BTput], axis=1)
allgreek
```

	Call	Put
delta	0.4254278837831471	-0.5701060357740761
gamma	3.552713678800501e-11	2.6645352591003757e-11
vega	0.17685542410467156	0.211594087936966
theta	0.053571813065296875	0.06408815568438354
rho	-0.002846432852467995	-0.004257110634595662
phi	No phi	No phi

In this part, I used the binary tree function to calculate the American option value. The question didn't give a set N value for the periods in tree so I tried different N value and find out that as N increases, the put-call difference decreases. The greeks for american option don't have very special characteristics.

3 Question 2

3.1 Portfolio Implied Volatility and Delta

	Portfolio	Type	Underlying	Holding	OptionType	ExpirationDate	Strike	CurrentPrice	Implied vol	Delta
0	Straddle	Option	AAPL	1	Call	3/18/2022	165.0	4.50	0.232402	0.4270297606522311
1	Straddle	Option	AAPL	1	Put	3/18/2022	165.0	4.40	0.217642	-0.5688603871564446
2	SynLong	Option	AAPL	1	Call	3/18/2022	165.0	4.50	0.232402	0.4270297606522311
3	SynLong	Option	AAPL	-1	Put	3/18/2022	165.0	4.40	0.217642	-0.5688603871564446
4	CallSpread	Option	AAPL	1	Call	3/18/2022	165.0	4.50	0.232402	0.4270297606522311
5	CallSpread	Option	AAPL	-1	Call	3/18/2022	175.0	0.72	0.188846	0.118476653784233
6	PutSpread	Option	AAPL	1	Put	3/18/2022	165.0	4.40	0.217642	-0.5688603871564446
7	PutSpread	Option	AAPL	-1	Put	3/18/2022	155.0	1.60	0.264137	-0.24712270990578933
8	Stock	Stock	AAPL	1	NaN	NaN	NaN	164.85	NaN	1
9	Call	Option	AAPL	1	Call	3/18/2022	165.0	4.50	0.232402	0.4270297606522311
10	Put	Option	AAPL	1	Put	3/18/2022	165.0	4.40	0.217642	-0.5688603871564446
11	CoveredCall	Stock	AAPL	1	NaN	NaN	NaN	164.85	NaN	1
12	CoveredCall	Option	AAPL	-1	Call	3/18/2022	165.0	4.50	0.232402	0.4270297606522311
13	ProtectedPut	Stock	AAPL	1	NaN	NaN	NaN	164.85	NaN	1

In this part, I calculate the implied volatility of each portfolio first, then based on that volatility calculate each delta.

3.2 Mean, VaR and ES results

	Mean	VaR	ES
Straddle	-0.008531	1.956320	2.461632
SynLong	0.007863	13.501817	17.027653
CallSpread	-0.017573	4.194568	5.214515
PutSpread	0.004957	4.408845	5.552085
Stock	-0.045021	14.040167	17.263396
Call	0.030325	5.798407	7.182334
Put	-0.001679	7.625133	9.666796
CoveredCall	-0.018850	7.985433	10.031100
ProtectedPut	-0.019480	5.962380	7.444021

	Mean	VaR	ES
Straddle	0.324467	2.442145	2.452667
SynLong	0.127025	13.682455	16.666597
CallSpread	-0.147681	3.636945	3.711153
PutSpread	0.292053	2.679557	2.740726
Stock	0.250977	13.461481	16.427392
Call	0.225746	4.356602	4.431042
Put	0.098721	4.261091	4.331759
CoveredCall	0.025230	9.104879	11.996350
ProtectedPut	0.349698	4.135627	4.191836

In this part, I calculate the mean, VaR and ES after delta hedge based on the delta produced by binary trees. Comparing with last week's results, we can see that for the mean, this week's result get more closed to zero, that's because the hedge effect of delta hedge. However, they won't be exactly zero since there is higher degree of Taylor's expansion of the price sensitivity. For the VaR and ES, results are closed to last week.

4 Question 3

4.1 Expected return and covariance

```
AAPL     0.143414
FB       0.219205
UNH      0.137663
MA       0.261014
MSFT     0.172935
NVDA     0.325676
HD       0.100648
PFE     -0.150353
AMZN     0.186311
BRK-B    0.112803
PG       0.077017
XOM      0.179250
TSLA     0.184864
JPM      0.132777
V        0.190974
DIS      0.150065
GOOGL    0.194495
JNJ      0.075037
BAC      0.172139
CSCO     0.137953
dtype: float64
```

	AAPL	FB	UNH	MA	MSFT	NVDA	HD
AAPL	0.065441	0.031073	0.020744	0.010865	0.039993	0.081306	0.020419
FB	0.031073	0.104613	0.008503	0.040435	0.037940	0.071235	0.007342
UNH	0.020744	0.008503	0.044678	0.025495	0.022884	0.037212	0.016155
MA	0.010865	0.040435	0.025495	0.129913	0.008331	0.032840	0.013766
MSFT	0.039993	0.037940	0.022884	0.008331	0.065237	0.089283	0.022864
NVDA	0.081306	0.071235	0.037212	0.032840	0.089283	0.354876	0.052871
HD	0.020419	0.007342	0.016155	0.013766	0.022864	0.052871	0.058241
PFE	-0.021341	-0.034076	-0.006501	-0.025942	-0.018821	-0.048652	-0.022702
AMZN	0.041714	0.039219	0.018861	0.018757	0.033346	0.100484	0.014528
BRK-B	0.000136	0.009335	0.002269	0.020274	-0.001568	-0.003756	0.000238

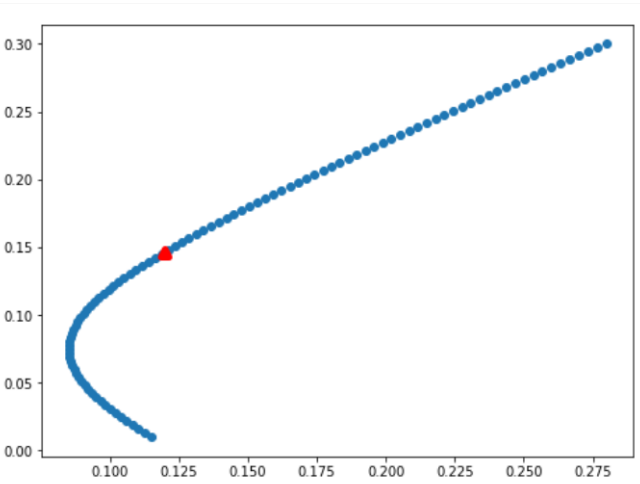
Here is the expected return based on F-F 4 factors model with momentum. I used OLS to get the coefficient of each factors and based on them to calculate each stock's expected return and covariance.

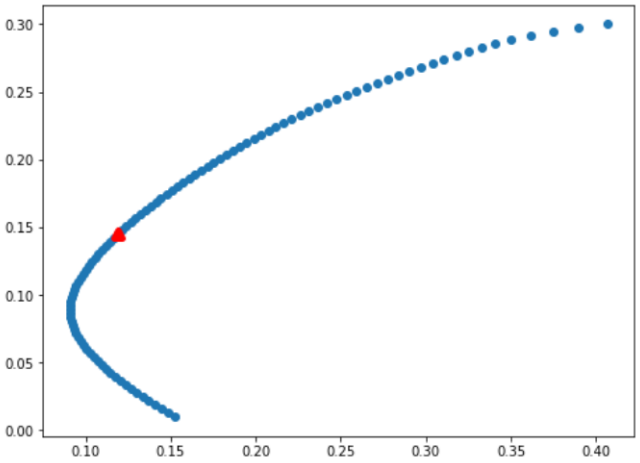
4.2 Efficient Frontier

```
/usr/local/lib/python3.7/dist-packages/i
  app.launch_new_instance()
ret       0.146135
vol       0.119725
sharpe    1.199701
dtype: float64
```

```
efport_noshort,  tangency_noshort,  asse
tangency_noshort
```

```
/usr/local/lib/python3.7/dist-packages/i
  app.launch_new_instance()
ret       0.144614
vol       0.118647
sharpe    1.197782
dtype: float64
```






```
17] round(assetWeight.T,4)
```

	AAPL	FB	UNH	MA	MSFT	NVDA	HD	PFE	AMZN	BRK-B	PG	XOM	TSLA	JPM	V	
Weight	-0.0348	0.0234	0.0468	0.0664	0.0895	-0.0045	0.0289	-0.0035	0.1858	0.2703	0.0224	0.0141	0.0	0.0279	-0.0128	0.0

```
round(assetWeight_noshort.T,4)
```

	AAPL	FB	UNH	MA	MSFT	NVDA	HD	PFE	AMZN	BRK-B	PG	XOM	TSLA	JPM	V	DIS	GOOGL	
Weight	0.0	0.0217	0.0388	0.0591	0.066	0.0	0.0234	0.0	0.1648	0.2628	0.0334	0.0136	0.0	0.0202	0.0	0.0096	0.06	0.1

In this part, I used the efficient frontier function above to calculate the efficient frontier and optimize portfolio. The portfolio with shortsell and without

shortsell have very close sharpe ratio. But in the efficient frontier, we can see that the portfolio without shortsell have a more sparse tail when return is high. And for the portfolio weights, the without shortsell portfolio have all their weights larger than 0.