Project 7

Leslie Dees

Problem 1:

Implement the closed form Greeks for GBSM

I built the function *def greeks()* to calculate the closed form of the Greeks. Methods were all calculated using the formulas in the notes. Call and put values for delta, theta, rho and carry rho were all carried out in separate if-else statements while gamma and vega were uniform between both, therefore were calculated outside. The implied volatility is required to be calculated outside of the Greek method as an input. I normalized my implied volatility calculation through the new method *calculate_implied_volatlity()*. Issues with some convergence though led me to later implement a newton formulation which utilizes vega directly to simulate minimization to converge to an implied volatility.

Implement a finite difference derivative calculation

My finite difference derivative calculation was performed using the finite derivative functional form found in the notes to take a small epsilon on either side of the true value.

Compare the values between the two methods for both a call and a put

Call

Greek	Functional Value	Discrete Value
Delta	1.0642110456591238e-06	1.0642675100808207e-06
Gamma	1.9411776879709066e-06	1.941224565262748e-06
Vega	0.00023855309623545742	0.0008723816583898845
Theta	-0.0003204158333980998	-0.00010464095989856153
Rho	1.4480859300575613e-05	-5.0694704835894944e-08
Carry Rho	1.329946801436884e-05	1.4677310932645728e-05

The Call Delta, Gamma, Theta, and Carry Rho values are almost exactly identical. While Vega and Rho tend to differ more.

Put

Greek	Functional Value	Discrete Value
Delta	-0.9995198724942445	-0.9995198724368493
Gamma	1.9411776879709066e-06	1.9417711882852018e-06
Vega	0.00023855309623545742	0.0008723816577571597
Theta	6.185210465533484	6.18542645032818
Rho	-14.860582557513306	-1.2123716286737363
Carry Rho	-13.64821232609308	-13.648212807628113

For the Put we see near identical values once again for Delta, Gamma, Theta, and Carry Rho. Here we also see a large factor off of Rho as well as a slight difference between discrete and functional Vegas.

Implement the binomial tree valuation for American options with and without discrete dividends

This was implemented as *binomial_tree_option_pricing_american_complete*. Which contains an automatic differentiator between discrete and non-discrete dividend quantities. The code was mirrored from the Julia code provided to us.

Calculate the value of the call and the put, calculate the Greeks of each

Call

Continuous Dividend Binomial Tree:	2.0181023066392756e-06
Discrete Dividend Binomial Tree:	1.4201852429267405e-06

Greek	Functional Value	Discrete Value
Delta	0.9995095329081151	0.9999966333473713
Gamma	-1.6763297380597837e-05	-5.263836711857079e-06
Vega	0.0023047716229026263	0.0007153405376184244
Theta	-6.182068074032886	-6.984530037889826
Rho	14.860441990832683	14.860551520827215
Carry Rho	13.648083227257574	13.575441334136887

Put

Continuous Dividend Binomial Tree:	13.969999999999999
Discrete Dividend Binomial Tree:	14.23500121262204

Greek	Functional Value	Discrete Value
Delta	-1.1403797175100734e-05	-3.3666526286779686e-06
Gamma	-1.6763297380597837e-05	-5.263836711857079e-06
Vega	0.0023047716229026263	0.0007153405376184244
Theta	0.003462807333996986	0.0010763729512366501
Rho	-0.0001550475399247355	-4.551754539248347e-05
Carry Rho	-0.00014239830352150642	-4.158128090222963e-05

What is the sensitivity of the put and call to a change in the dividend amount?

The sensitivity of the put and call to a change in the dividend amount is calculated through the Carry Rho. In this case:

Call:

Carry Rho	13.648083227257574	13.575441334136887

Put:

Carry Rho	-0.00014239830352150642	-4.158128090222963e-05

It appears that the Call is much more sensitive to changes in the dividend amount than the put.

Problem 2.

Using DailyPrices.csv fit a normal distribution to AAPL returns – assume 0 mean return.

I successfully fit a normal distribution to AAPL after centering returns to 0 mean.

Simulate AAPL returns 10 days ahead and apply those returns to the current AAPL price

After centering the returns, I applied an AR(1) model to obtain the 10-day forward predicted returns. From there, I applied them cascading forward from the current Apple price. I ensured to add the dividend amount to each day moving forward from 03/15/2023. In order to do this, I had to make sure that I was only putting down prices for trading days. Moving forward however, I feel that my work became more up in the air.

Calculate Mean, VaR, and ES

From here, I was very confused on how to proceed. In a prior assignment, I was deducted points for not fitting a t-distribution, using a gaussian copula, and simulating values to obtain my VaR and ES. So, in this assignment I made sure to attempt to do all of these things.

Firstly, fitting a t-distribution to the data was infeasible for me. I attempted to fit using returns and prices, both for just the 10-day predictions and the entire prices/returns going back from the 10-day forecast additions. The degrees of freedom, mean, and standard deviation were all impossibly large for the dataset and were providing returns either in the nan range, infinity, or in the several hundred thousand dollars. Therefore, I opted to try to fit a normal instead. This provided some level of improvement, although I believe that my final values were impacted.

I performed 10,000 simulations of choosing Apple returns. From there, I separated out each portfolio into their double option holdings. I calculated the value using a method called *calculate_portfolio_value_american* which uses a binomial tree with American values and discrete dividend payments to obtain the portfolio value. This method either calculates the value of a put or hold, or just takes the difference between a stock price and the current price if it is not an Option. These portfolio values were then multiplied by (1+simulated_returns) in order to get a final value. This was then normalized by the portfolio value. ES and VaR were calculated as normal.

The values I obtained were:

Portfolio	VaR	ES	Mean Value
Straddle	$-6.7850	$-10.1441	$1.0792
SynLong	$0.0000	$0.0000	$0.0000
CallSpread	$0.0000	$0.0000	$0.0000
PutSpread	$0.0000	$0.0000	$0.0000
Stock	$0.4569	$0.7047	$-0.0519
Call	$-1.2128	$-1.4509	$1.1190
Put	$-3.6236	$-5.0721	$0.7049
CoveredCall	$5.3898	$7.9213	$-0.6316
ProtectedPut	$-5.7378	$-6.5119	$-0.3403

I understand that the method of calculation is different than what may have been expected, however I was incapable of producing the results using t-distribution. My results appear to be normal, although SynLong, CallSpread, and PutSpread are oddly all $0.00. I am unsure of why these results are occurring.

Calculate VaR and ES using Delta-Normal

For the VaR and ES using Delta-Normal, I was very confused as to how to approach this, since they are single values based on the stats of the dataframe. Therefore, I decided to calculate these separately as well for the 10-day prediction values. I obtained the following values:
Delta-Normal VaR: $6.07416

Delta-Normal ES: $29.4446

Problem 3.

Use the Fama French 3 factor as well as the Carhart Momentum time series to fit a 4-factor model to the stocks.

In order to solve this problem, I needed to scrap 10 years of returns data for each stock listed. I did this though yahoo finance and stored all of the data in a separate dataframe, which combined the Carhart, Fama, and Stock data into a single merged dataframe.

I took the 10-year average for Mkt-RF, SMB, HML, and MOM. Then, I deducted the risk-free rate from all of the stock returns to make my y value. I constructed an OLS model fitting the Mkt-RF, SMB, HML, and MOM data to the deducted stock returns with an extra constant epsilon included. I then extracted Beta values for each to obtain a formula for the expected return as a percent.

Based on the past 10 years of factor returns, find the expected annual return of each stock.

I obtained the 10-year mean of each factor so that I could include it into my completed OLS model parameter function to obtain annual returns for each stock. They are listed below:

AAPL:	11.058%
MSFT:	10.366%
AMZN:	9.822%
TSLA:	18.9656%
GOOGL:	7.369%
META:	7.801%
NVDA:	21.294%
BRK-B:	5.095%
JPM:	6.698%
JNJ:	4.131%

UNH: 10.398%

HD: 8.3208%

PG: 4.175%

V: 8.181%

BAC: 6.538%

MA: 8.764%

PFE: 4.143%

XOM: 4.206%

DIS: 3.903%

CSCO: 5.646%

Construct an annual covariance matrix for the stocks.

The covariance matrix is too large to paste here, but I obtained this through an exponentially weighted covariance matrix using just the returns of each of the stocks. It can be found as the variable: *covar_matrix.*

Find the super-efficient portfolio.

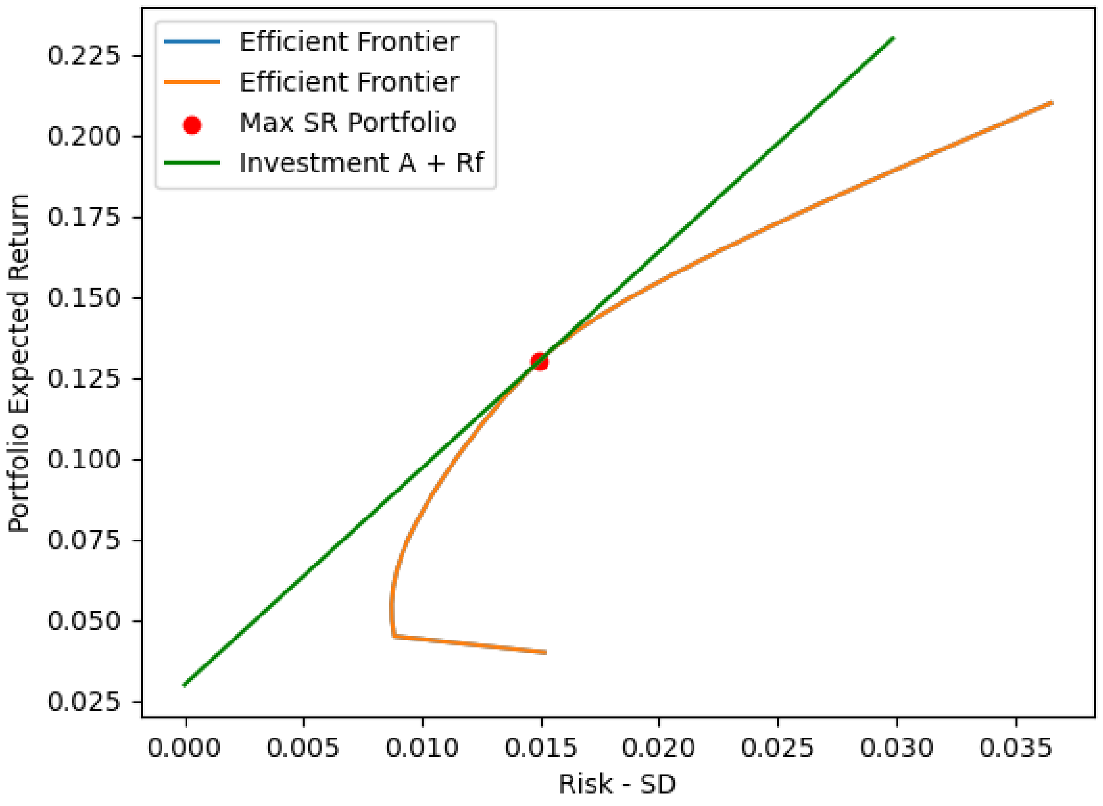
I found the super-efficient portfolio by creating an optimize risk function based on the provided Julia code which minimizes the risk given a covariance matrix that I obtained through my exponentially weighted covariance matrix function. I think plotted the function and obtained the relevant weights.
Portfolio Weights at the Maximum Sharpe Ratio:

[8.75376317e-19, 1.84705557e-19, 1.81893557e-18, 7.94171119e-19, 1.91912755e-18, 3.00271178e-18, 2.39179731e-01, 1.85476583e-18, 1.30642357e-18, 1.53853799e-18, 7.59011285e-01, 1.08139581e-18, 1.61806941e-18, 1.80898462e-03, 1.69937993e-18, 1.42194255e-18, 1.18883571e-18, 1.83404755e-18, 3.90520273e-18, 1.58158327e-18]

These weights are ordered by the stock as listed above.
Portfolio Return: 0.12999999999999995

Portfolio Risk: 0.014930887251642308

Portfolio SR: 6.697525626884668


Efficient Frontier
Efficient Frontier
Max SR Portfolio
Investment A + Rf
Portfolio Expected Return
Risk - SD
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035