Christian Selmer Petersen 202108784
Valdemar Emil Otte Petersen 202108408
Programmerings projekt – portfolio optimization

Introduction to Programming with Scientific Applications (Spring 2023)

Final project

Study ID	Name	% contributed
202108784	**Christian Selmer Petersen**	**50 %**
202108408	**Valdemar Emil Otte Petersen**	**50 %**

max 3 students

Briefly state the contributions of each of the group members to the project

Valdemar – Erfaren programmør, god troubleshooter hvis koden fejlede, kommentarskriver og kaffehenter

Christian – Benarbejder, formeltjekker, humørløfter, korrekturlæser og skribent

Note on plagiarism

Since the evaluation of the project report and code will be part of the final grade in the course, **plagiarism in your project handin will be considered cheating at the exam**. Whenever adopting code or text from elsewhere you must state this and give a reference/link to your source. It is perfectly fine to search information and adopt partial solutions from the internet – actually, this is encouraged – but always state your source in your handing. Also discussing your problems with your project with other students is perfectly fine, but remember each group should handing their own solution. If you are in doubt if you solution will be very similar to another group because you discussed the details, please put a remark that you have discussed your solution with other groups.

For more Aarhus University information on plagiarism, please visit
http://library.au.dk/en/students/plagiarism/

Christian Selmer Petersen 202108784
Valdemar Emil Otte Petersen 202108408
Programmerings projekt – portfolio optimization

Vi har valgt emne 2, ”Portfolio optimization”. Dette var ikke det store valg for os, da det er relevant for vores studie, matematik-økonomi, og interessere os personligt.

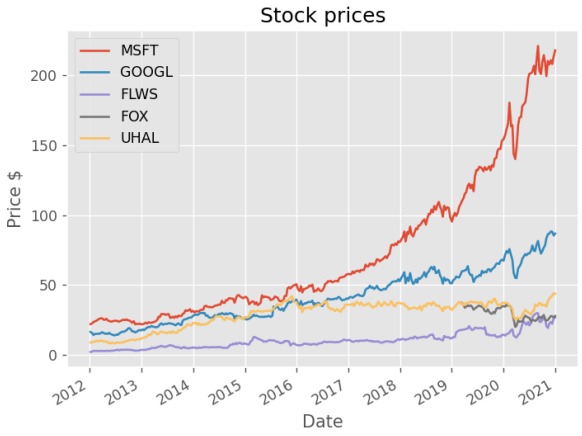
Delopgave 1:

Get_prices:

Til opgavens fik vi downloadet Pandas datareader ned uden de store problemer, hvorefter vi valgte at teste de forskellige hjemmesider den henter sin data nedfra, bl.a. ”Yahoo” og ”Stooq”, hvoraf den sidste var den vi valgte at bruge i hele opgaven. Alt var dog ikke lige let med ”Stooq”, da den jævnfør dokumentation på Github, havde en ødelagt parameter, ”frequence”, hvilket havde været en stor hjælp, da det er lige det vi skal bruge til stepsize. Heldigvis for os har Pandas datareader en anden indbygget funktion, som kan splice, som vi valgte at bruge til stepsize. Lige inden stepsize bliver kaldt, vendes vores dataframe om. Dette gøres da vi senere skal indexer vores data, og dette gør at vi får det rigtige resultat. Vi valgte også at dummy teste denne stepsize og fandt frem til, både logisk og algoritmisk, at det ikke er muligt at få en stepsize der er en fraktion og mindre end 0. Dette valgte vi at skrive ind med en assert statement, så dette vil blive overholdt. Vi havde overvejelser om at der måske ville være nogle problemer med en Pandas dataframe i forhold til en normal Numpy matrice. Dette viste sig dog ikke at blive noget som helst problem.

Plot_stocks:

Vi valgte at bruge matplotlib style ”ggplot”, da vi begge syntes den er pæn. At plotte dette var ikke den store udfordring, da det ikke var første gang vi skulle plotte eller navngive i et plot.



Delopgave 2:

Vi valgte at lave tre forskellige funktioner, til at henholdsvis at regne, $R, \mu \; og \; \Sigma$. Til sidst har vi funktionen ”plot_pdf_norm”, der ligesom navnet angiver vil plotte grafen af pdf.

Calculate_r:

Da dataene stadig er i Pandas dataframe så kan vi bruge Pandas indbyggede funktion, pct change, som giver os den fraktionelle gevinst af aktien til den valgte tid. Derefter rykkes alle rækker en plads op, hvorefter nederste række slettes. Dette gøres for at få dataene i det ønskede format. Tilslut ændrer vi dataene fra en Panda dataframe til Numpy array, derefter returneres funktionen.

Christian Selmer Petersen 202108784
Valdemar Emil Otte Petersen 202108408
Programmerings projekt – portfolio optimization

Calculate_mu:

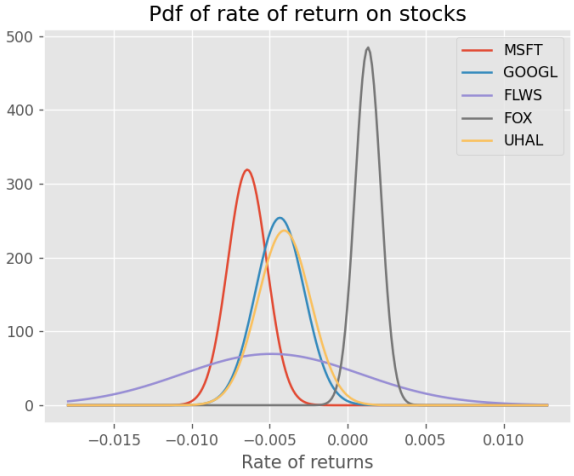
Til udregningen af μ var det blot at følge formlen.

Calculate_sigma:

Til udregningen af Σ var det blot at følge formlen. Dog fik vi ikke den ønskede covarians matrice, da vi i stedet for at dividere med $\frac{1}{n}$ brugte $\frac{1}{antallet\ af\ stocks}$, hvilket vi fik rettet da vi indså det.

Plot_pdf_norm:

Vi skulle bruge numpy's "linspace" til at lave et array med punkter, hvor norm.pdf kan laves deri. Vi valgte at sætte grænserne til at være den mindste μ værdi, fratrukket 2 gange den højeste varians og den højeste μ værdi med 2 gange den højeste varians lagt til, for at vi kan være sikker på at få minimum 95,3% af fordelingerne. Dens default værdi er 50, men da det gav en hakket graf, valgte vi at sætte den op til 250 punkter, hvilket gav fornuftige grafer og den flotte velkendte, klokkeform en normal fordeling har.



Efter lidt læsning på hvilket input norm.pdf skulle have, så var selve plotningen ligetil.

Delopgave 3:

Optimize:

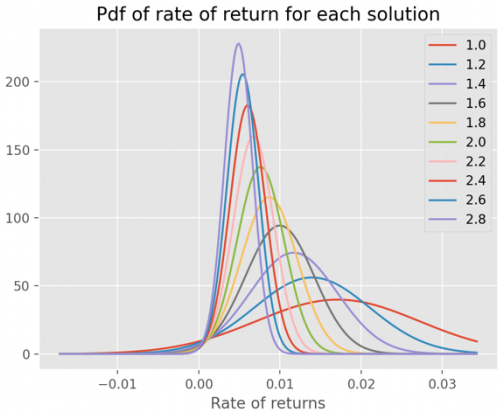
Som vi også har beskrevet i selve koden, så har vi valgt at indsætte objektfunktionen inde i optimeringsfunktionen. Vi bruger "Scipy optimization extension" og bruger dens minimerings funktion. Yderligere bruger vi også en lambdafunktion til at få en ny funktion hver gang. Vi sætter også begrænsningerne til lighed.

Christian Selmer Petersen 202108784
Valdemar Emil Otte Petersen 202108408
Programmerings projekt – portfolio optimization

Plot_gamma_pdf:

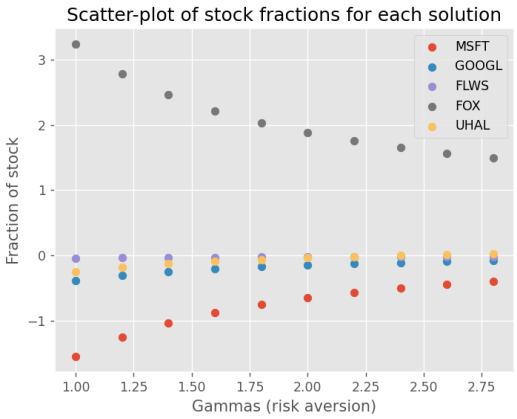
Til at starte med, så var det ikke svært at plotte de rigtige kurver, men farverne skiftede hele tiden. Altså grafen bevæger sig kontinuert, men går fra rød til blå til grøn osv.

Efter at prøve os lidt frem, så indså vi at vi skulle transponere nogle af matricerne i forhold til andre, hvilket så giver os resultat her til højre.



Plot_w_scatter:

Tilsvarende problemer som beskrevet lige herover. Vi valgte at samle alle fractions for en stock, for derefter at samle dem alle sammen og til sidst scatter plot dem.



Afsluttende bemærkninger:

Vi startede med at bruge *numpy np.matmul* til at gange to matricer sammen. Dog, så er dette en ”grim” måde at gøre det på og selve koden kan blive lang, som vi kan se i dette eksempel ”np.matmul(A, np.matmul(B, C)). Heldigvis, så fandt vi den pænere operating, nemlig @. Dermed bliver matrixmultiplikation fra dette herover til ”A @ B @ C”, hvilket gør koden meget mere let læseligt.

Vi læste PEP8 (https://peps.python.org/pep-0008/) igennem og rettede vores fil, således at det passer overens med hvad der er god stil inden for kodning. Vi har også tænkt over hvad navne på vores variabler og funktioner skulle være, da det er mest optimalt, at giv dem et klart og enentydigt navn, såfremt at når man efter et halvt år eller mere stadigvæk hurtigt kan finde ud af hvad de gør.

Christian Selmer Petersen 202108784
Valdemar Emil Otte Petersen 202108408
Programmerings projekt – portfolio optimization

Under alle funktionerne har vi beskrevet, hvordan parametrene ser. Både på skrift men også med ”static typing”. Især med static typing, så gør det at funktionerne forventer en bestemt datatype som input. Hvis denne datatype ikke bliver givet som input, vil programmet giv en fejl. Dette gør f.eks. test og dokumentation nemmere. Det bemærkes også, at vi importerer en række klasser fra diverse models. De bruges nemlig til static typing.

Selve projektet lægger ikke op til den store testning. Vi har dog indført en enkel ”assert” i funktionen ”get_prices”, da programmet ikke vil fejle selvom variablen ”step_size” er negativ eller nul. Altså, vi kontrollere at ”step_size” er et positivt tal.

Ydreligere, så har vi samlet hele programmet i en ”if __name__ == ’__main__’”, da så vil det være et standalone program.

Requirements

python = ">=3.8.1,<3.12"

matplotlib = "^3.7.1"

pandas-datareader = "^0.10.0"

pandas = "^2.0.1"

numpy = "^1.24.3"

scipy = "^1.10.1"