Eötvös Loránd Tudományegyetem

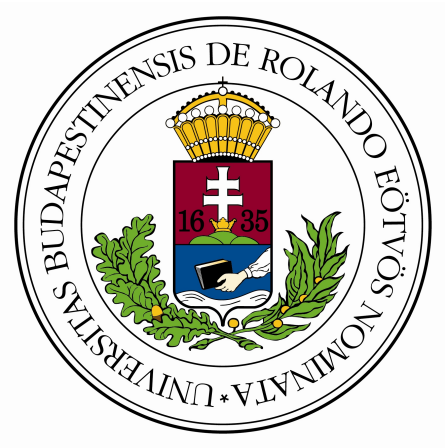
Természettudományi kar

Optimális befektetések

Miklós Áron
Szakdolgozat
Matematika BSc
Alkalmazott matematikus szakirány

Témavezető:

Rásonyi Miklós



Budapest, 2024

Tartalomjegyzék

Bevezetés

Ezen dolgozat célja áttekinteni egy, a pénzügyi matematikában fontos témakör, a portfóliók optimalizálásának néhány problémáját, és bizonyos modellekre megoldani magát az optimalizálási feladatot egy általunk elkészített program segítségével.

A feladat a következő: Keresünk egy portfóliót, azaz azokat a számokat, melyek meghatározzák, hogy az egyes tőzsdei termékekből hányat veszünk. A keresés kritériuma az, hogy valamely funkcionált próbálunk maximalizálni. A piac általános körülményeit úgy jellemezzük, hogy a részvényárakat valószínűségi változókkal írjuk le.

Az első fejezetben megismerkedünk a probléma megértéséhez szükséges alapvető matematikai és közgazdaságtani fogalmakkal, definíciókkal, összefüggésekkel és tételekkel.

A második fejezetben meghatározzuk, hogy n kockázatos és egy nem kockázatos pénzügyi termékből álló portfóliót hogyan kell optimálisan összeállítani. Azaz olyan módon szeretnénk megválasztani a portfóliónkban található részvények mennyiségét, hogy egy kitűzött jövőbeli időpontban a portfólió értéke az elérhető legnagyobb legyen, ahol a feladat első és legfontosabb része ennek az elérhető maximális értéknek a meghatározása. A kockázatos termékeink általában részvények, míg a nem kockázatos termékünk egy kötvény. A dolgozat során mindig egylépéses modellekkel fogunk foglalkozni, akár optimális portfólió értéket, akár optimális stratégiát keresünk. Vagyis a 0 időpontban megválasztjuk azt, hogy az egyes részvényekből hányat vásárolunk. A portfolió értéke az 1. időpontban ezek után a véletlentől függ, hiszen a részvények ára is véletlen.

A probléma megoldásához elengedhetetlen a hasznossági függvény ismerete. Ez a függvény azt írja le, hogy az adott befektetőnek, akinek a portfólióját optimalizálni szeretnénk, milyen a kockázathoz való viszonya, vagyis egy adott x értékű portfólió mennyi értéket hordoz az adott befektető számára. Matematikai értelemben ez egy u valós értékű, folytonosan differenciálható, szigorúan monoton növő és szakaszonként szigorúan konvex vagy konkáv függvény (azaz az adott szakaszokon belül nem változik a konkavitása). Az u függvény egy adott x (valós) portfólió értékhez hozzárendeli azt a valós $u(x)$ számot, amely kifejezi mennyire értékes a befektetőnek az adott portfólió.

Fontos megjegyezni, hogy a második fejezetben annak érdekében, hogy minél szemléletesebb példákat tudjunk mutatni, csak konkáv hasznossági függvényekkel dolgozunk, de a harmadik fejezet során vizsgálunk változó konkavitású hasznossági függvényeket is.

A hasznossági függvény a matematikai leírás során is kulcsszerepet kap. Ugyanis a feladat felírható egy olyan maximalizálási problémaként, amely során az $E[u(X)]$-et, azaz a hasznossági függvény várható értéket kell maximalizálni, ahol az X végigfut a portfólió lehetséges (véletlen) értékein.

Miután meghatároztuk a portfólió optimális értékét, azt is megvizsgáljuk, hogy mi az optimális stratégia, vagyis az adott pénzügyi termékekből mennyit kell vásárolnunk, illetve eladnunk ahhoz, hogy a portfólió elérje az optimális értékét.

Többlépéses modellek is hasonlóan kezelhetők, de a számítások elbonyolódnak, a mi célunk pedig inkább az ötletek, módszerek bemutatása, amit ilyen egyszerű, egylépéses modellekben tudunk a legjobban megtenni.

A harmadik fejezetben azt vizsgáljuk meg, hogy a kockázathoz különféleképpen viszonyuló, azaz más és más u kockázatfüggvényekkel rendelkező befektetőknél hogyan változik a portfólió maximalizáláshoz szükséges optimális stratégia. Emellett összehasonlítjuk a befektető elégedettség abban az esetben ha az x tőkéjével kereskedhet a piacon, azzal az esettel, amikor ezt a tőkét nem

fektetheti be (ez az utóbbi maga az $u(x)$ érték).

Először konkáv hasznossági függvényeket vizsgálunk, melyek a kockázatkerülő befektetőknek felelnek meg. Ezt követően foglalkozunk olyan hasznossági függvényekkel is, melyeknek egy szakasza konkáv, a másik konvex. Ezek olyan befektetőket írnak le, akik egy bizonyos határig (azaz amíg a portfólió egy adott értéket el nem ér) kockázatvállalóbbak, de ezen értékhatár átlépése után az érték növekedésével párhuzamosan egyre kockázatkerülőbbek lesznek, tehát a portfólió értékének egységnyi növekedése már egyre kevesbé értékes nekik.

Ennek során dolgozunk majd egy program segítségével véletlenszerűen generált portfóliókkal, és olyanokkal is, amelyek egy ismert közgazdaságtani, matematikai modellen alapulnak.

1. Matematikai és közgazdaságtani alapok

A probléma matematikai leírásához először be kell vezetnünk néhány gyakran használt, fontos fogalmat.
Legyenek $t_0 < t_1 < ... < t_N$ valós számok, melyek a kereskedési napokat jelölik. t_0 a kiindulási időpont és t_N az az időpont, amikor szeretnénk, hogy a portfóliónk a lehető legtöbbet érje. A modellünk két fő termékcsaládból áll: egy kockázatmentes befektetésből, például kötvényből, ennek n-edik időpillanatbeli értékét B_n-el jelöljük és az (n+1)-edik pillanatbeli értékét, rekurzívan a

$$B_{n+1} = B_n(1 + r_{n+1})$$

képlettel számoljuk, ahol $r_n > -1$ a $[t_{n-1}, t_n]$ periódusban érvényes kockázatmentes kamatláb. Adott emellett k darab kockázatos termék, például részvények. Az i-edik részvény n-edik időpillanatbeli értékét S_n^i-vel jelöljük, rekurzívan az

$$S_n^{i+1} = S_n^i(1 + \mu_n^i)$$

képlet segítségével számoljuk ki, ahol μ_n^i a $[t_{n-1}, t_n]$ egy olyan valószínűségi változó, amely a $[t_{n-1}, t_n]$ periódusban az i. részvényre vonatkozó megtérülési rátát írja le. Itt a **megtérülési ráta** a befektetés nettó nyeresége vagy vesztesége egy meghatározott időszak alatt, a befektetés kezdeti értékének százalékában kifejezve.

A μ_n sorozat következő tulajdonságának megértéséhez be kell vezetnünk a természetes filtráció fogalmát.

1. Definíció. *Legyen $(\Omega, \mathcal{F}, P)$ egy valószínűségi mező, I egy rendezett indexhalmaz és $(\mathcal{S}, \Sigma)$ mérhető tér, és $(\mu_n)_{n\in I} : Ix\Omega \to \mathcal{S}$ valószínűségi változókból álló sorozat. Ekkor $\mathcal{F}$ μ_n-re vett* ***természetes filtrációja*** *egy olyan $(\mathcal{F}_n)_{n\in I}$ sorozat, melyre*

$$\mathcal{F}_i = \sigma\{\mu_j^{-1}(A) | j \in I, j \leq i, A \in \Sigma\}$$

azaz a legkisebb σ-algebra az Ω-n, amely az összes Σ mérhető részhalmaz ősképét tartalmazza a j. "időpillanatig".

Valószínűségi változók egy folyamatának a természetes filtrációja egy olyan σ-algebra sorozat, mely i. eleme tartalmazza a folyamatról az i. időillanatig megtudott információkat. Most tegyük fel, hogy az $(\Omega, \mathcal{F}, P)$ véges valószínűségi mezőn dolgozunk, ahol $\mathcal{F}$ Ω hatványhalmaza, és $P(\omega) > 0$ minden $\omega \in \Omega$-ra. Ekkor a $\mu_n = (\mu_n^1, ..., \mu_n^k)$ a természetes filtrációra nézve adaptált folyamat.

Ezeknek segítségével definiáljuk a portfóliót és annak legfontosabb tulajdonságait.

2. Definíció. *Egy* ***portfólió*** *egy k+1 változós sztochasztikus folyamat, melynek n-edik elemének első k komponense a k részvény-, k+1-edik eleme pedig a kötvény n-edik időpillanatbeli mennyisége, melyeket rendre α_n^i-vel és β_n-el jelölünk. Innen adódik a portfóliónk n-edik pillanatbeli értéke:*

$$V_n^{\alpha,\beta} = \sum_{i=1}^{k} \alpha_n^i S_n^i + \beta_n B_n \tag{1}$$

1. Jelölés. *Jelöljük a kockázatos termékek mennyiségeinek vektorát $\alpha = (\alpha^1, \ldots, \alpha^k)$-val, az áraik vektorát pedig $S = (S^1, \ldots, S^k)$-val, valamint jelölje a kettő $\mathbb{R}^d$ beli skalárszorzatát $\alpha S = \sum_{i=1}^k \alpha^i S^i$. Ennek segítségével az (1) egyenlet*

$$V_n = \alpha_n S_n + \beta_n B_n$$

alakban írható fel.

Mostantól az $\alpha S = \sum_{i=1}^{k} \alpha^i S^i$ *jelölést akalmazzuk.*

3. Definíció. *Az* α *és* β *vektorokból alkotott* (α, β) *párokat* ***stratégiáknak*** *nevezzük.*

4. Definíció. *Legyen az* α_n *egy* ***jósolható*** *folyamat, azaz olyan folyamat, hogy minden* $i = 1, \ldots, k$*-ra* α_n^i $\mathcal{F}_{n-1}$ *mérhető, ahol* $(\mathcal{F}_n)$ *a természetes filtráció.*
Egy portfóliót ***önfinanszírozónak*** *nevezünk, ha minden* $1 \leq n$ *esetén teljesül a*

$$V_{n-1} = \alpha_n S_{n-1} + \beta_n B_{n-1}$$

egyenlőség.

Ez úgy értelmezhető, hogy ha a t_{n-1} pillanatban rendelkezésünkre áll V_{n-1} tőke, akkor a részvények és a kötvény új mennyiségét változatlanul hagyva a portfólió értéke is változatlan marad.

2. Jelölés. *Az olyan stratégiák halmazát amik jósolhatók és önfinanszírozóak is* $\mathcal{A}$*-val jelöljük.*

5. Definíció. *Az* $\tilde{S}_n^i = \frac{S_n^i}{B_n}$ *összefüggéssel kapott* $\tilde{S}$ *értéket* ***diszkontált értéknek*** *nevezzük A diszkontált értékekből álló portfólió a* ***diszkontált portfólió***
A ***diszkontált portfólió értéke****:*

$$\tilde{V_n} = \alpha_n \tilde{S}_n + \beta_n$$

Itt B az úgynevezett ***számláló folyamat*** *azaz az az egység, amihez képest a többi termék értékét vesszük. Ilyen egységnek bármely szigorúan pozitív árú termék választható, azonban mi most ennek az egységnek a kötvényt választjuk.*

A problémáink megoldása során mindig fontos feltennünk, hogy a piac arbitrázsmentes, ezért vezessük be a következő fogalmakat.

6. Definíció. *Egy* $\mathcal{A}$ *beli önfinanszírozó stratégiát* ***arbitrázsnak*** *nevezünk, ha teljesülnek rá az*

1. $V_0^{(\alpha,\beta)} = 0$*;*
2. $V_N^{(\alpha,\beta)} \geq 0$*;*
3. $P(V_N^{(\alpha,\beta)} > 0) > 0$

tulajdonságok. Egy piac akkor ***arbitrázsmentes*** *ha az* $\mathcal{A}$ *stratégiacsalád nem tartalmaz arbitrázst.*

Az arbitrázsmentességet azért fontos feltennünk, mert az arbitrázs létezése azt jelentené, hogy kockázatmentesen tudunk profitra szert tenni.

Az optimalizáláshoz tartozó elengedhetetlen eszköz a martingál mérték.

7. Definíció. *Egy ekvivalens martingál mérték B numerai-al egy Q valószínűségi mérték az* $(\Omega, \mathcal{F}$ *téren, úgy, hogy:*

1. *Q és P ekvivalens mértékek és*
2. $\tilde{S}_{n-1} = E^Q[\tilde{S}_n | \mathcal{F}_{n-1}]$ *minden* $n = 1, \ldots, N$*-re, azaz* $\tilde{S}$ *Q martingál, ahol* E^Q *a Q mérték szerinti várható értéket jelöli.*

Ezen fogalmak segítségével már kimondhatjuk a termékek árazásának első alaptételét.

1. Tétel. *Egy diszkrét idejű piac pontosan akkor arbitrázsmentes, ha létezik legalább egy ekvivalens martingál mérték.*

[2]

Bevezetésünk zárásaként bevezetjük a replikáció fogalmát és megismerjük a replikációs egyenletet, valamint az európai opciót.

8. Definíció. *Ha egy X pénzügyi termékre létezik olyan $(\alpha, \beta) \in \mathcal{A}$ statégia, amire $V_N^{(\alpha,\beta)} = X$, akkor X-et **replikálhatónak**, az (α, β) stratégiát pedig X **replikációs stratégiájának** nevezzük.*

A következőnek ismertetett replikációs tétel a martingál módszer bevezetésénél lesz nagy jelentőségű.

2. Tétel. *Legyen X egy replikálható termék egy arbitrázsmentes piacon. Ekkor minden $(\alpha, \beta) \in \mathcal{A}$ replikációs statégiára és minden Q B egységgel ellátott ekvivalens martingál mértékre teljesül, hogy*

$$E^Q\left[\frac{X}{B_N}|\mathcal{F}_n\right] = \frac{V_N^{(\alpha,\beta)}}{B_n}, n = 0, \ldots, N$$

*A $V_N^{(\alpha,\beta)}$ folyamatot X **arbitrázs árának** nevezzük.*

9. Definíció. *A **mögöttes termékek** azok az egyszerűbb pénzügyi termékek, amelyek árainak változása meghatározza a belőlük felépülő összetettebb pénzügyi termék árváltozását.*

10. Definíció. *A **származékos termékek** olyan összetetteb pénzügyi termékek, melyek árváltozását az általa tartalmazott mögöttes termékek árváltozása határozza meg.*

11. Definíció. *Az **európai származékos termék** az $S = (S^1, \ldots, S^d)$ mögöttes termékekkel egy X valószínűségi változó az $(\Omega, \mathcal{F}, P)$ valószíűségi mezőn, amely mérhető az $\mathcal{F}_N^S := \sigma(\{S_n | n \leq N\})$ σ-algebrára nézve.*
*Ennek speciális esete az **európai opció**, amely megvásárlójának jogot, de nem kötelességet ad arra, hogy a mögöttes termékeket egy előre megbeszélt időpontban és összegért megvegye vagy eladja (**call**, illetve **put** opció).*
*Azt mondjuk, hogy X a pénzügyi termék **kifizetése**.*

2. Optimális portfólió érték keresése

Ebben a fejezetben először megvizsgáljuk az optimális portfólió érték és stratégia megtaláláshoz szükséges matematikai módszereket, elsősorban az úgynevezett martingál módszert. Megismerkedünk néhány új, fontos fogalmommal és a problémánk megoldásához elngedhetetlen legfontosabb tétellel. Ezután részben kézi számítással, részben az elkészített program segítségével megoldunk három szemléletes, gyakorlati példát. Az első példa egy kézzel elkészített példa, amely még nagyrészt kézi számolással, azaz számítógépes segítség nélkül is megoldható rövid idő alatt. A második példánk ennek egy bonyolított változata, több részvényt tartalmazó portfólióval, amely azt hivatott bemutatni, hogy a feladat nagyon gyorsan elbonyolódik, és már csak a programunk segítségével oldható meg. A harmadik példa szintén egy bonyolult, csak számítógéppel megoldható példa, melyet egy neves közgazdaságtani modell, a faktor modell alapján állítottuk össze.

2.1. Martingál módszer

Az optimális értékek és stratégiák megkeresésére használt egyik legfontosabb és leghasznosabb módszer az úgynvezett martingál módszer.

Ennek alapja, hogy szeretnénk olyan stratégiát találni amelyre teljesül a $V_N^{(\alpha)} = X$ egyenlőség. Ez visszavezethető egy martingál reprezentációjának megkeresésére. Rögzítsünk ugyanis egy Q martingál mértéket és vegyük az

$$\tilde{M}_n := E^Q[B_N^{-1} X | \mathcal{F}_n], n = 0, \ldots, N$$

martingált, ahol X egy európai opció kifizetését jelölő valószínűségi változó (az európai opció olyan pénzügyi termék, mely csak a lejárati dátumkor érvényesíthető). Itt megjegyezzük, hogy a replikációs egyenlet alapján

$$v = E^Q[\tilde{V}_N^{(\alpha)}] = E^Q[B_N^{-1} X] \tag{2}$$

Ha meg tudunk határozni egy olyan α startégiát, melyre az

$$\tilde{M}_n = \tilde{V}_n^{(\alpha)} = v + \sum_{k=1}^{n} \alpha_k (\tilde{S}_k - \tilde{S}_{k-1}) \tag{3}$$

formula teljesül, akkor innen $\tilde{V}_N^{(\alpha)} = \tilde{X}$, amiből adódik az általunk célként kitűzött $V_N^{(\alpha)} = X$ egyenlőség. Tehát tényleg elég egy reprezentációját megtalálni a (3) alakú $\tilde{M}$ martingálnak.

Tekintsük a következő maximalizálási problémát:

$$\max_{\alpha} E[u(V_N^{(\alpha)})] \tag{4}$$

ahol u az úgynevezett hasznossági függvény.

12. Definíció. *Az u függvényt akkor nevezzük* ***hasznossági függvénynek****, ha* $u : I \to \mathbb{R}$ *folytonosan differenciálható, szigorúan monoton és szakaszonként szigorúan konkáv vagy konvex, ahol I az* $]a, +\infty[$ *intervallum, rögzített* $a \leq 0$ *mellett (a lehet akár* $-\infty$ *is). Emellett u-ra teljesül, hogy* $a > -\infty$ *esetén* $\lim_{v \to a^+} u'(v) = +\infty$ *és* $a = -\infty$ *esetén u felülről korlátos.*

Ebben a fejezetben csak konkáv hasznossági függvényekkel fogunk foglalkozni, de később elő fognak fordulni nem konkáv példák is. Nevezetes példák az $I = (0, \infty)$ esetben a logaritmikus hasznossági függvény,

$$u(v) = log(v)$$

és a hatvány hasznossági függvény,

$$u(v) = \frac{v^k}{k},$$

ahol $k < 1, k \neq 0$ valós paraméter.

A fenti, (4)-es pont beli maximalizálási problémára nyújt megoldást a martingál módszer, amely három fő lépésből áll:

1. Megkeresi azon végső, azaz N. időpillata beli portfólió értékek halmazát, amelyek elérhetőek egy önfinaszírozó és jósolható stratégiával
2. Meghatározza az optimális elérhető értéket, azaz azt a V_N-et amelyben a maximum az (4)-ben felvétetik
3. Meghatároz egy olyan önfinanszírozó stratégiát, amivel a 2-es pontban kapott optimum elérhető

Ebben a fejezetben az első és második probléma megoldását fogjuk tárgyalni.

2.2. Probléma teljes piacokra

Először is fontos bevezetnünk a problémáink megértéséhez a teljes piac fogalmát.

13. Definíció. *Legyen X egy származékos termék és (α, β) olyan stratégia, hogy $V_N^{(\alpha,\beta)} = X$. Ekkor X-et **replikálhatónak**, (α, β)-t **replikáló stratégiának** nevezzük.*
*Az olyan piacot, amelyen minden származékos termék replikálható, **teljes piacnak** nevezzük.*

Ennek segítségével kimondhatjuk a termékek árazásának második alaptételét:

3. Tétel. *Egy arbitrázsmentes piac pontosan akkor teljes, ha pontosan egy ekvivalens martingál mérték létezik.*

Tehát teljes piacon egy ekvivalens martingál mértékünk van, amely segítségével a második probléma megoldható, a következő tétel segítségével.

4. Tétel. *Legyen a piacunk teljes és arbitrázsmentes. Tekintsük a (4) pontban felírt*

$$\max_{\alpha} E[u(V_N^{(\alpha)})]$$

maximalizálási problémát, $v \in \mathbb{R}_+$ kezdeti portfólió értékkel. Az

$$u'(I) = \mathbb{R}_+$$

feltétel mellett az optimális elérhető értéket a

$$\overline{V}_N = \mathcal{I}(\lambda \tilde{L})$$

egyenlet adja meg, ahol $\mathcal{I}$ a hasznossági függvény deriváltjának, azaz I-nek az inverze, $\tilde{L} = B_N^{-1} L, L = \frac{dQ}{dP}$, Q a martingál mérték és a $\lambda \in \mathbb{R}$ értékét az

$$E^P[\mathcal{I}(\lambda \tilde{L})] = v$$

*úgynevezett **költségvetési egyenlet** határozza meg.*

A tételt itt nem bizonyítjuk, mivel a következő, nem teljes piacokról szóló fejezetben egy ennél általánosabb tételt bizonyítunk, amely nem teljes piacokra is működik.

1. Megjegyzés. *A nem teljes piacokra vonatkozó általánosabb tételben extremális martingál mértékek segítségével számoljuk ki a martingál mértéket. Annak ez a tétel speciális esete, itt nincs szükség extremális martingál mértékek használatára, mert a martingál mérték egyértelmű.*

2.3. Probléma nem teljes piacokra

A teljes piacokkal ugyan könnyebb dolgozni és optimális portfólió értéket, illetve stratégiát keresni, azonban ezek nem írják le elég pontosan a valóságot.
Ezért mi főként nem teljes, arbitrázsmentes piacokkal fogunk foglalkozni. Ezek esetében már a martingál módszer első lépése is nehezebb a teljes piacokhoz képest. Itt ugyanis a v kezdeti vagyonból elérhető végső értékek halmazát a

$$\nu_v = \{V | E^Q[B_N^{-1}V] = v \text{ minden } Q \text{ martingál mértékre}\} \tag{5}$$

adja. A legfontosabb különbség a teljes piachoz képest, hogy itt nincs egyértelmű martingál mértékünk, ezért úgynevezett extremális martingál mértékek segítségével tudunk dolgozni.

A martingál mértékek halmaza $\mathbb{R}^M$ egy affin alterének és a szigorúan pozitív valószínűségi mértékek halmazának metszete, ahol M a lehetséges kimenetelek, azaz az $\omega \in \Omega$ elemek száma, formulával:

$$\mathbb{R}_+^M = \{Q = (Q_1, \ldots, Q_M) | Q_j > 0 \ \ \forall j = 1, \ldots, M\}$$

14. Definíció. *Az **extremális martingál mértékek** azok a $Q^{(1)}, \ldots, Q^{(r)} \in \overline{\mathbb{R}_{\mathbb{M}}^+}$ mértékek, melyekre minden Q martingál mérték előáll a konvex kombinációjukként, azaz $Q = a_1Q^{(1)} + \cdots + a_rQ^{(r)}$ alakban, ahol $\sum_{i=1}^r a_i = 1$.*

Ezekre az extremális martingál mértékekre teljesül a (2)-es azonosság. hogy Most, hogy áttekintettük a fogalmakat következzen a portfólió optimalizálási feladatunk megoldásához szükséges legfontosabb tétel:

5. Tétel. *Legyen a piacunk nem teljes és arbitrázsmentes. Az*

$$u'(I) = \mathbb{R}_+ \tag{6}$$

feltétel mellett a végső pillanatbeli optimális érték

$$\bar{V}_N = \mathcal{I}(\sum_{j=1}^r \lambda_j \tilde{L}^{(j)}) \tag{7}$$

*ahol $\mathcal{I}$ az u hasznossági függvény deriváltjának inverze, r az extremális martingál mértékek száma, $\tilde{L}^{(j)} = B_N^{-1}L^{(j)}$, $L^{(j)} = \frac{dQ^{(j)}}{dP}$ (itt ez $Q^{(j)}P$-re vett Radon-Nikodym deriváltját jelöli) és a $\lambda_1, \ldots, \lambda_r$ valós súlyokat az úgynevezett **költségvetési egyenletből** kapjuk meg:*

$$E^P = \left[\mathcal{I}(\sum_{k=1}^r \lambda_j \tilde{L}^{(k)})\tilde{L}^{(j)}\right] = v \ \ j = 1, \ldots, r \tag{8}$$

[2]

Bizonyítás. A tétel bizonyításához először is szeretnénk visszavezetni a portfólió optimalizálásának problémánkat egy adott függvény maximalizálásának problémájára. Ehhez ágyazzuk be a problémát az euklideszi térbe. Legyen M az Ω alaphalmaz számossága, $\omega_1, \ldots, \omega_M$ jelöljék az elemi eseményeket. Ha Y egy valós valószínűségi változó Ω-n legyen $Y(\omega_j) = Y_j$ minden $j = 1, \ldots, M$-re és jelölje az $Y \in \mathbf{R}^M$ vektort:

$$Y = (Y_1, \ldots, Y_M)$$

Ekkor

$$E^P[Y] = \sum_{j=1}^{M} Y_j P(\omega_j)$$

Innen adódik, hogy a portfólió optimalizálási problémánk ekvivalens az

$$f(V) := \sum_{i=1}^{M} u(V_i) P_i = E^P[u(V)] \tag{9}$$

függvény maximalizálásának problémájával, ahol a V-re és minden V_i nemnegatív kell, hogy legyen, hiszen u az $\mathbb{R}^+$-on van értelmezve. Ebből a megkötésből adódik a költségvetési egyenletrendszerben kiszámolt és a végső optimális értéket megadó egyenletben használt λ_i értékek nemnegativitása. Ez szükséges is, mivel a hasznossági függvény a nemnegatív valós számok halmazán van értelmezve, így az ő derváltjának inverze, $\mathfrak{I}$ is. Itt fontos megkötés, hogy $V \in \nu_v$, ami a (4) azonosság segítségével kifejezhető

$$g^{(j)}(V) := \sum_{j=1}^{M} B_N^{-1} V_i Q_i^{(j)} - v = E^{Q^{(j)}}[B_N^{(-1)} V] - v = 0 \tag{10}$$

alakban.

Tehát szeretnénk egy olyan rendszert kapni, amelyben megjelennek az extremális martingál mértékek és aminek segítségével az f függvény maximuma kiszámítható. Ehhez vezessük be a Lagrange-módszert.

Tekintsük a következő nemlineáris optimalizálási problémát:
Maximalizáljuk $f(x)$-et a következő feltételek mellett

- $g_j(x) = 0$

ahol $x \in X$ az optimalizálni kívánt változó $\mathbb{R}^n$ konvex részhalmazából választva, X pedig az alaphalmaz, ahonnan a változó kikerülhet. f a célfüggvény amit maximalizálunk g_j $(j = 1, \ldots, l)$ pedig az egyenlőségi feltételek függvényei. A probléma megoldásához a következő függvényt írhatjuk fel:

$$\mathcal{L}(x, \lambda) = f(x) + \lambda g(x)$$

Itt $\mathcal{L}$-et Lagrange-függvénynek, λ-t Lagrange-multiplikátornak nevezzük. Ezzel kapcsolatban tesz állítást a Lagrange multiplikátor tétel.

Lemma. *Legyen $f : \mathbb{R}^n \to \mathbb{R}$ a célfüggvény, $g : \mathbb{R}^n \to \mathbb{R}^c$ a feltételeket meghatározó függvény, és legyen mindkettő folytonosan differenciálható.*
Emellett legyen $x_\star$ egy optimális megoldása a következő optimalizálási problémának:

Maximalizáljuk $f(x)$-et a $g(x) = 0$ feltétel mellett

úgy, hogy a parciális deriváltak által alkotott mátrix, $[Dg(x_\star)]_{j,k} = \frac{\delta g_j}{\delta x_k}$ rangja legfeljebb n lehet. Ekkor egyértelműen létezik egy $\lambda_\star \in \mathbb{R}^c$ Lagrange-multiplikátor úgy, hogy $Df(x_\star) = \lambda_\star^T Dg(x_\star)$.

[4]
Ezt a módszert alkalmazzuk a mostani problémánkra ahol a maximalizálandó függvény a (8)-es képletben szereplő $f(V)$, az egyenleteinket pedig a (9)-as pont adja. Ezekre a Lagrange-módszert alkalmazva kapjuk:

$$\mathcal{L}(V,\lambda) = f(V) - \sum_{k=1}^{r} \lambda_k g^{(k)}(V)$$

Ahhoz, hogy az extremális martingál mértékeket megkapjuk, a gradienst válasszuk 0-nak. Fontos megjegyezni, hogy a szélsőérték megtalálása azért lehetséges, mert egy korlátos és kompakt halmazon vagyunk.. Ekkor a szélsőértékek a Weierstraß-féle szélsőértéktétel miatt felvétetnek a halmazon. Ez azt jelenti, hogy létezik olyan V pont ahol az f függvény maximuma felvétetik és eleget tesz az $S_i(V) = 0$ peremfeltételnek. Ekkor az f és g függvény, illetve a V pont kielégítik a Lemma feltételeit. Tehát a Lemma szerint a maximumhely, azaz V ki kell, hogy elégítse a következő két egyenletet:

$$\partial_{V_i}\mathcal{L}(V,\lambda) = u'(V_i)P_i - B_N^{-1}\sum_{k=1}^{r}\lambda_k Q_i^{(k)} = 0, \quad i = 1,\dots,M, \tag{11}$$

$$\partial_{\lambda_j}\mathcal{L}(V,\lambda) = \sum_{i=1}^{M} B_N^{-1} V_i Q_i^{(j)} - v = 0 \quad j = 1,\dots,r \tag{12}$$

A (6) feltételt, vagyis azt felhasználva, hogy a hasznossági függvény az I intervallumon pozitív valós értékeket vesz fel, a (8)-as egyenlet ekvivalens a

$$(\bar{V}_N)_i = \mathcal{I}(B_N^{-1}\sum_{k=1}^{r}\lambda_k \frac{Q_i^{(k)}}{P_i}) \quad i = 1,\dots,M$$

egyenlettel, így a (7)-es egyenlettel is. Ezt a (8)-as egyenletbe behelyettesítve kapjuk a

$$\sum_{i=1}^{M} B_N^{-1}\mathcal{I}(B_N^{-1}\sum k = 1^r \lambda_k \frac{Q_i^{(k)}}{P_i})Q_i^{(j)} = v \quad j = 1,\dots,r$$

egyenletrendszert, ami pont a (7)-es egyenlettel ekvivalens, és így beláttuk a tételt.

□

Ennek a tételnek a segítségével elvégezhető a martingál módszer második lépése, vagyis az optimális portfólió érték kiszámítása. Ezen felül, miután kiszámoltuk az előbb említett V optimális portfólió értéket, ennek segítségével meg tudjuk határozni azt a ϕ stratégiát amivel ez a V érték elérhető. Pontosabban keressük azt a $\phi \in \mathbb{R}^r$ vektort amire teljesül a

$$V(\omega_j) = v + \sum_{k=1}^{r}\phi_k \Delta S^k(\omega_j)$$

egyenlőség minden $\omega_j \in \Omega$, $j = 1,\dots,M$-re. Itt $\Delta S^k(\omega_j) = S^k(\omega_j) - 1$ az k. részvény árának változása az ω_j esemény esetén. Mivel az ω_j események M száma szinte mindig legalább akkora mint a részvények r száma, ezért az egyenletrendszerünk túlhatározott, és elég az első r egyenlet által alkotott egyenletrendszert tekintenünk a ϕ kiszámolásához. A ϕ vektor k. eleme annak felel meg, hogy a k. részvényből mennyit kell a portfóliónkba vásárolni az optimális érték eléréséhez. Fontos, hogy ϕ koordinátái lehetnek negatívak is, ami annak felel meg, hogy a koordinátához tartozó részvényből akkora mennyiséget eladunk.

Fontos megjegyezni, hogy ugyan a példáinkban egylépéses eseteket vizsgálunk, azaz egy lépést teszünk meg előre az időben, de a most bebizonyított tételt felhasználva nincs technikai akadálya a többlépéses esetek vizsgálatának sem, csupán a számítási idő nőne meg.

2.4. Példa nem teljes piacokra két részvény esetén

Kezdetben nézzünk egy egyszerű példát olyan nem teljes piacra, ahol 2 részvényünk van és az Ω alaphalmaz 4 elemű. Fontos, hogy ebben és a következő példákban is mindig egylépéses modellekkel dolgozunk. Mivel a részvények árváltozását az 1-hez képest szeretnénk megmondani, ezért fontos, hogy a csupa 1 pont (ami annyi koordinátából áll, ahány részvényünk van) beleessen a részvények árváltozásainak konvex burkába. Erre azért van szükség, hogy kapott poliéderünknek legyenek szélsőértékei. Ezt szem előtt tartva, bevezető példa lévén, válasszuk úgy a pontokat, hogy egy egyenesre essenek, valamint az $(1,1)$ pont is essen az egyenesre, így garantálva, hogy benne lesz a konvex burok belsejében. Válasszunk egy tetszőleges kezdeti pontot, például a $(0.5, 0.6)$. Ez és az $(1,1)$ pont definiálja a

$$y = 0.8x + 0.2$$

egyenletű egyenest. Ehhez válasszuk az ω-kat a következő módon, úgy hogy a pontok az egyenesre essenek:

$$(S_1(\omega_1), S_2(\omega_1)), (S_1(\omega_2), S_2(\omega_2)), (S_1(\omega_3), S_2(\omega_3)), (S_1(\omega_4), S_2(\omega_4)) =$$
$$= (0.16, 0.33), (0.5, 0.6), (1.77, 1.61), (3.18, 2.74)$$

Ezek segítségével írjuk fel a martingál mértékek poliéderét meghatározó egyenleteket és egyenlőtlenségeket. Ez 2 részvényre és 4 állapotra általánosan:

- $q_1 S_1(\omega_1) + q_2 S_1(\omega_2) + q_3 S_1(\omega_3) + q_4 S_1(\omega_4) = 1 + r$
- $q_1 S_2(\omega_1) + q_2 S_2(\omega_2) + q_3 S_2(\omega_3) + q_4 S_2(\omega_4) = 1 + r$
- $q_1 + q_2 + q_3 + q_4 = 1$
- $q_1, q_2, q_3, q_4 \geq 0$

ahol r a kockázatmentes kamat. Mivel most a nem teljes piacokra vonatkozó legegyszerűbb példát szeretnénk bemutatni legyen $r = 0$.
Ide behelyettesítve a generált pontjaink megfelelő koordinátáit:

- $1.5q_1 + 0.81q_2 + 1.26q_3 + 0.67q_4 = 1$
- $2q_1 + 0.61q_2 + 1.56q_3 + 0.35q_4 = 1$
- $q_1 + q_2 + q_3 + q_4 = 1$
- $q_1, q_2, q_3, q_4 \geq 0$

Erre alkalmazzuk a kódjegyzékben található extremális martingál mértékeket kiszámító kódot. Ezt használva tehát a martingál mértékek poliéderének extremális értékei:

$$Q_1 = (0.17, 0.67, 0.17, 0), Q_2 = (0.34, 0.33, 0, 0.33)$$

Most az egyszerűség kedvéért válasszuk a hasznossági függvénynek a logaritmikus hasznossági függvényt, mert ekkor

$$u(v) = logv \rightarrow u'(v) = \frac{1}{v} \rightarrow \mathcal{I}(\omega) = \frac{1}{\omega}$$

teljesül. Fontos feltétel, hogy u' értékkészlete a pozitív valós számok halmaza, itt a logaritmusra ez teljesül. A B_1-et, tehát a nem kockázatos termékünk, azaz kötvényünk 1. lépés utáni értékét a könnyebb számolás végett válasszuk 1-nek. (Azért az 1. lépés utáni értéket választottuk ki, mert most 1 lépésre vizsgáljuk a portfólió optimalizálási problémát, így a végső $N = 1$ lépés utáni állapot $B_N = B_1$). Ekkor a kapott egyenletrendszer, amiből a költségvetési egyenlethez szükséges λ változókat kiszámolhatjuk:

$$E^P\left[\mathcal{I}(\sum_{k=1}^{n}\lambda_k\tilde{L}^{(k)})\tilde{L}^{(j)}\right] = v, \quad j = 1,\ldots,n$$

ahol n az extremális mértékek száma. Használjuk ki, hogy most $n = 2$ és $\mathcal{I}(y) = \frac{1}{y}$, valamint, hogy $B_1 = 1$ miatt

$$\tilde{L}^{(1)} = \tilde{L}^{(2)} = L^{(1)} = L^{(2)}$$

Emellett válasszuk a v kezdeti portfólió értéket 1-nek (például dollárban, de a számolásunkhoz bármilyen valutát használhatunk, a fontos az, hogy minden érték ugyanabban a valutában legyen megadva). A kódjegyzékben található program segítségével random generáljunk egy P valószínűségi mértéket, ez most $P = (0.07, 0.67, 0.18, 0.08)$. Ezeket felhasználva kapjuk a következő két egyenletet:

- $E^P\left[\frac{1}{\lambda_1 L^{(1)}+\lambda_2 L^{(2)}}L^{(1)}\right] = v$
- $E^P\left[\frac{1}{\lambda_1 L^{(1)}+\lambda_2 L^{(2)}}L^{(2)}\right] = v$

Ehhez $L^{(1)}$-et és $L^{(2)}$ számoljuk ki, ezek a Radon-Nikodym deriváltak:

- $L^{(1)} = (\frac{Q_1^{(1)}}{P^{(1)}}, \frac{Q_1^{(2)}}{P^{(2)}}, \frac{Q_1^{(3)}}{P^{(3)}}, \frac{Q_1^{(4)}}{P^{(4)}}) = (\frac{0.17}{0.07}, \frac{0.67}{0.67}, \frac{0.17}{0.18}, \frac{0}{0.08}) = (2.43, 1, 0.94, 0)$
- $L^{(2)} = (\frac{Q_2^{(1)}}{P^{(1)}}, \frac{Q_2^{(2)}}{P^{(2)}}, \frac{Q_2^{(3)}}{P^{(3)}}, \frac{Q_2^{(4)}}{P^{(4)}}) = (\frac{0.34}{0.07}, \frac{0.33}{0.67}, \frac{0}{0.18}, \frac{0.33}{0.08}) = (4.86, 0.49, 0, 4.13)$

Mivel a részvények árváltozása diszkrét eloszlású, ezért a $L^{(i)}$ várható értékét a

$$E[L^{(i)}] = \sum_{j=1}^{d} P(L^{(i)}(\omega_j))L^{(i)}(\omega_j) = \sum_{j=1}^{d} p_j L^{(i)}(\omega_j)$$

ahol p_j az ω_j esemény bekövetkezésének valószínűsége és d az Ω eseményhalmaz számossága. Most $d = 4$:

- $E^P\left[\frac{1}{\lambda_1 L^{(1)}+\lambda_2 L^{(2)}}L^{(1)}\right] = \sum_{j=1}^{4} p_j\left(\frac{1}{\lambda_1 L^{(1)}(\omega_j)+\lambda_2 L^{(2)}(\omega_j)}L^{(1)}(\omega_j)\right) = 1$
- $E^P\left[\frac{1}{\lambda_1 L^{(1)}+\lambda_2 L^{(2)}}L^{(2)}\right] = \sum_{j=1}^{4} p_j\left(\frac{1}{\lambda_1 L^{(1)}(\omega_j)+\lambda_2 L^{(2)}(\omega_j)}L^{(2)}(\omega_j)\right) = 1$

Ide a fent kiszámolt értékeiket behelyettesítve:

- $0.07 \cdot \frac{1}{2.43\lambda_1+4.86\lambda_2} \cdot 2.43 + 0.67 \cdot \frac{1}{\lambda_1+0.49\lambda_2} + 0.18 \cdot \frac{1}{0.94\lambda_1} \cdot 0.94 = 1$
- $0.07 \cdot \frac{1}{2.43\lambda_1+4.86\lambda_2} \cdot 4.86 + 0.67 \cdot \frac{1}{\lambda_1+0.49\lambda_2} \cdot 0.49 + 0.08 \cdot \frac{1}{4.13\lambda_2} \cdot 4.13 = 1$

A fenti egyenletrendszerre az erre való kódunkat használva kapjuk, hogy
$\lambda_1 = 0.85$, $\lambda_2 = 0.15$. Ha a λ_j együtthatókat meg tudjuk kapni az optimális értéket a

$$\overline{V}_N = \mathcal{I}\left(\sum_{j=1}^{r}\lambda_j\tilde{L}^{(j)}\right)$$

Most $N = 1$, $r = 2$ $B_1 = B_1^{-1} = 1$ miatt $\tilde{L} = L$. Az $L^{(1)}$ és $L^{(2)}$ vektorokat már ismerjük, a λ_1, λ_2 együtthatókat pedig most számoltuk ki a költségvetési egyenletrendszerből. $\mathcal{I}$ szintén ismert, így mindent behelyettesítve:

$$\overline{V}_1 = \frac{1}{\lambda_1 L^{(1)} + \lambda_2 L^{(2)}} = \frac{1}{0.85 \cdot (2.43, 1, 0.94, 0) + 0.15 \cdot (4.86, 0.49, 0, 4.13)} =$$

$$= \frac{1}{(2.07, 0.85, 0.8, 0) + (0.73, 0.07, 0, 0.62)} = \frac{1}{(2.8, 0.92, 0.8, 0.62)} =$$

$$= (0.36, 1.09, 1.25, 1.61)$$

Ez a vektor tehát megadja a portfóliónk optimális értékeit egy lépés után, az i. koordináta azt mutatja, hogy az ω_i esemény bekövetkezése esetén mi a portfóliónk optimális értéke.
Most ennek segítségével számítsuk ki az optimális stratégiát meghatározó ϕ vektort. Mivel 2 részvényünk van, ezért az egyenletrendszerünk most csak 2 elemű:

- $V_1(\omega_1) = v + \phi_1 \Delta S_1^1(\omega_1) + \phi_2 \Delta S_1^2(\omega_1)$
- $V_1(\omega_2) = v + \phi_1 \Delta S_1^1(\omega_2) + \phi_2 \Delta S_1^2(\omega_2)$

Ebbe a kiszámolt V_1 értékeit, illetve a korábban megadott többi paramétert behelyettesítve:

- $0.36 = 1 + \phi_1(0.16 - 1) + \phi_2(0.33 - 1)$
- $1.09 = 1 + \phi_1(0.5 - 1) + \phi_2(0.6 - 1)$

Ezt az egyenletrendszert megoldva kapjuk, hogy $\phi_1 = 316.3$ és $\phi_2 = -395.6$, azaz az optimális stratégiát meghatározó vektor $\phi = (316.3, -395.6)$.

2.5. Példa nem teljes piacokra sok részvény esetén

Az előző példában mivel 2 részvénnyel és 4 elemű eseményhalmazzal dolgoztunk, a portfólió optimális értékének kiszámolása még kézzel is elvégezhető volt, bár igaz, hogy az extremális martingál mértékek megtalálásához már ott is számítógépes eszközöket alkalmaztunk. Azonban sok részvény és lehetséges esemény esetén a számolási idő exponenciálisan nő. Így nem érdemes és kellően rövid idő alatt nem is lehetséges megoldani kézzel a problémát. Ezért a sok részvényből álló portfólió optimális értékének kiszámolását teljes egészében a kódjegyzékben található OPTIMAL_VALUE_CALCULATOR függvénnyel fogjuk végezni, a fontosabb lépéseket, eredményeket kiírva.
Legyen a konkrét példánk egy olyan portfólió, amely 3 részvényből áll és az Ω eseményhalmaz 8 elemű. Először is generáljuk le a részvények árváltozásait a különböző eseményekre a GENERATE_POINTS függvény segítségével. Az így kapott értékeinkből pedig írjuk fel a martingál mértékek poliéderét meghatározó egyenleteket és egyenlőtlenségeket a korábbi mintájára. Válasszuk a kockázatmentes kamatot $r = 0.05$-nek:

- $0.32q_1 + 0.45q_2 + 0.31q_3 + 0.17q_4 + 1.36q_5 + 1.28q_6 + 0.74q_7 + 1.02q_8 = 1.05$
- $0.4q_1 + 0.94q_2 + 1.18q_3 + 0.78q_4 + 1.31q_5 + 1.03q_6 + 1.07q_7 + q_8 = 1.05$
- $0.67q_1 + 0.02q_2 + 1.17q_3 + 0.04q_4 + 1.17q_5 + 1.49q_6 + 1.06q_7 + 1.02q_8 = 1.05$
- $q_1 + q_2 + q_3 + q_4 + q_5 + q_6 + q_7 + q_8 = 1.05$
- $q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8 \geq 0$

Erre a rendszerre alkalmazzuk a *get_extreme* függvényt, hogy megkapjuk az extremális martingál mértékeket, most 9 darabot kaptunk:

- $Q_1 = (0, 0, 0, 0, 0.13, 0.02, 0.06, 0.79)$
- $Q_2 = (0, 0, 0, 0.12, 0.22, 0.25, 0, 0.41)$
- $Q_3 = (0, 0, 0.02, 0, 0.13, 0.02, 0, 0.83)$
- $Q_4 = (0, 0.26, 0.02, 0, 0.14, 0.57, 0, 0)$
- $Q_5 = (0, 0.14, 0, 0.12, 0.22, 0.53, 0, 0)$
- $Q_6 = (0, 0.25, 0, 0, 0.14, 0.55, 0.05, 0)$
- $Q_7 = (0.27, 0, 0.02, 0, 0.66, 0.04, 0, 0)$
- $Q_8 = (0.14, 0, 0, 0.12, 0.49, 0.25, 0, 0)$
- $Q_9 = (0.26, 0, 0, 0, 0.64, 0.04, 0.05, 0)$

Most a GENERATE_PROBABILITY_MEASURE függvénnyel generáljunk P valószínűségi mértéket: $P = (0.25, 0.11, 0.06, 0.21, 0.01, 0.25, 0.01, 0.1)$ Innen már a Radon-Nikodym deriváltakat megkapjuk, ha az extremális mértékek koordinátáit leosztjuk a P valószínűségi mérték azonos indexű koordinátáival. Most az egyszerűség kedvéért legyen a kötvényünk értéke az első időpillanatban $B_1 = 1$, így rögtön az $\tilde{L}$ értékeket kapjuk, valamint legyen a kezdeti érték $v = 1$:

- $\tilde{L}^{(1)} = (0, 0, 0, 0, 13.76, 0.08, 6.37, 8.06)$
- $\tilde{L}^{(2)} = (0, 0, 0, 0.58, 22.06, 1.02, 0, 4.15)$
- $\tilde{L}^{(3)} = (0, 0, 0.37, 0, 13.72, 0.07, 0, 8.45)$
- $\tilde{L}^{(4)} = (0, 2.41, 0.33, 0, 14.68, 2.29, 0, 0)$
- $\tilde{L}^{(5)} = (0, 1.26, 0, 0.54, 22.06, 2.13, 0, 0)$
- $\tilde{L}^{(6)} = (0, 2.31, 0, 0, 14.68, 2.2, 5.66, 0)$
- $\tilde{L}^{(7)} = (1.09, 0, 0.33, 0, 68, 0.18, 0, 0)$
- $\tilde{L}^{(8)} = (0.57, 0, 0, 0.54, 49.9, 1.01, 0, 0)$
- $\tilde{L}^{(9)} = (1.05, 0, 0, 0, 65.79, 0.18, 5.66, 0)$

Válasszuk a hasznossági függvényünket $u(x) = \frac{x^\gamma}{\gamma}$-nak, így deriváltjának inverze $\mathcal{I}(z) = z^{\frac{1}{\gamma - 1}}$, ahol $x \in \mathbb{R}$ és $y \in \mathbb{R}_+$. Ezekből alkossuk meg a

$$E^P \left[\mathcal{I}(\sum_{k=1}^{n} \lambda_k \tilde{L}^{(k)}) \tilde{L}^{(j)} \right] = v, \quad j = 1, \ldots, n$$

egyenletrendszert és a SOLVE_BUDGET_EQUATIONS függvény segítségével számoljuk ki a λ_k változókat:

$\lambda_1 = 0.02, \lambda_2 = 0.005, \lambda_3 = 0.009, \lambda_4 = 0.17, \lambda_5 = 0.29, \lambda_6 = 0.000001, \lambda_7 = 0.01, \lambda_8 = 0.3, \lambda_9 = 0.15$

Ezek összege a kerekítési hibáktól eltekintve valóban 1, ami megfelel annak, hogy a martingál mértéket az extremális martingál mértékek konvex kombinációjaként kapjuk meg. Ezután utolsó lépésként helyettesítsünk be a

$$\overline{V}_N = \mathcal{I} \left(\sum_{j=1}^{r} \lambda_j \tilde{L}^{(j)} \right)$$

egyenletbe, majd alkalmazzuk a GET_OPTIMAL_VALUE függvényt:

$$\overline{V}_1 = (0.58, 0.88, 0.25, 0.57, 5.92, 1.16, 0.98, 0.49)$$

Így tehát megkaptuk, hogy a 8 különböző eseményre mennyi a portfólió optimális értéke egy lépés után.
Az előző példához hasonló módon itt is felírható a ϕ vektorra az egyenletrendszerünk V_1 segítségével, annyi különbséggel, hogy most 3 részvényünk van, így az egyenletrendszer is 3 egyenletből fog állni:

- $V_1(\omega_1) = v + \phi_1 \Delta S_1^1(\omega_1) + \phi_2 \Delta S_1^2(\omega_1) + \phi_3 \Delta S_1^3(\omega_1)$
- $V_1(\omega_2) = v + \phi_1 \Delta S_1^1(\omega_2) + \phi_2 \Delta S_1^2(\omega_2) + \phi_3 \Delta S_1^3(\omega_2)$
- $V_1(\omega_3) = v + \phi_1 \Delta S_1^1(\omega_3) + \phi_2 \Delta S_1^2(\omega_3) + \phi_3 \Delta S_1^3(\omega_3)$

Most V_1 és a többi ismert paraméter helyére behelyettesítve:

- $0.58 = 1 + \phi_1(0.32 - 1) + \phi_2(0.4 - 1) + \phi_3(0.67 - 1)$
- $0.88 = 1 + \phi_1(0.45 - 1) + \phi_2(0.94 - 1) + \phi_3(0.02 - 1)$
- $0.25 = 1 + \phi_1(0.31 - 1) + \phi_2(1.18 - 1) + \phi_3(1.17 - 1)$

Ennek az egyenletrendszernek a megoldásaként kapjuk a $\phi = (0.93, -0.22, -0.39)$ vektort.

2.6. Faktor modell

A mostani példánk legfőbb eltérése az eddigiekhez képest, hogy eddig magát a modellt is véletlenszerűen, a programunk segítségével generáltuk, most viszont egy neves pénzügyi modellt, a faktor modellt vesszük alapul.
Általában a konkrét modelleknek egy meghatározott struktúrája van, amely közdazdaságtani és matematikai módszereken, megfontolásokon alapszik. A faktor modell a lényege, hogy az egyes kockázatos termékekhez (részvényekhez) különböző faktorok tartoznak, illetve vannak közös, minden termékre kiható faktorok. Az egyes termékekre vonatkozó faktorok olyan hatásokat írnak le, amely csak az adott részvény értékére vannak hatással, a többi részvényre nem. Ilyen például egy adott cég részvénye esetében az adott cég termékeit érintő negatív vagy pozitív változások, a szállítás során felmerülő gondok. A közös faktorok olyan hatásokat írnak le, amelyek egy egész szektorra vagy az egész piacra hatással vannak. Ilyen lehet például az infláció, a munkanélküliség, bizonyos nyersanyagok hiánya.

Ahhoz, hogy az optimalizálási problémánkat a faktor modell esetén is meg tudjuk oldani, szeretnénk matematikailag is leírni a működését. Először ehhez azt kell látni, hogy a részvények árváltozásai hogyan jellemezhetők matematikai eszközökkel:

1. $\Delta S_1 = \mu_1 + \beta_1 \epsilon_1$
2. $\Delta S_i = \mu_i + \beta_i \epsilon_1 + \overline{\beta_i} \epsilon_i$

ahol ϵ_1 a közös faktor (most 1 közös faktorunk van), ϵ_i, $i \geq 2$ pedig az egyéni faktorok, μ_i-k a kockázatmentes kamatok, a β_i-k pedig a faktorok súlyai, melyek $\beta_i^2 + \overline{\beta}_i^2$ kombinációi megadják az S_i-k volatilitását. ΔS_1 egy tőzsdeindex árváltozása (például a budapesti tőzsdeindex, a BUX), ami az összes részvényt befolyásolja, ΔS_i, $i = 2, \ldots, n+1$ pedig a részvényeink árváltozásai.

15. Definíció. *Egy pénzügyi termék vagy index* ***volatilitása*** *az index vagy termék hozamánK szórása.*

Most ΔS_1-re rendezve:

$$\Delta S_1 = \overline{\beta_1}(\epsilon_1 - \underline{1})$$

ahol $b_1 = -\frac{\mu_1}{\beta_1}$. Ennek segítségével a ΔS_i-ket kifejezve:

$$\Delta S_i = \beta_i(\epsilon_1 - b_i) + \overline{\beta_i}(\epsilon_i - b_i)$$

ahol

$$b_i = -\frac{\mu_i}{\overline{\beta_i}} + \frac{\mu_1 \beta_i}{\overline{\beta_i}\,\overline{\beta_1}}$$

Most a martingál mérték létezése ekvivalens azzal, hogy teljesül minden i-re, hogy $P(\epsilon_i > b_i) > 0$ és $P(\epsilon_i < b_i) > 0$, azaz, hogy b_i ϵ_i-nél nagyobb és kisebb értéket is nagyobb mint 0 valószínűséggel vesz fel.

Szeretnénk, hogy minden i ($i = 2, \ldots, n+1$, ahol n részvények száma) esetén az ϵ_i várható értéke 0 legyen, $E[\epsilon_i] = 0$. A modellünkben ϵ_i két értéket vehet fel p_i, illetve $1-p_i$ valószínűséggel, azaz most

1. $P(\epsilon_i = A_i) = p_i$
2. $P(\epsilon_i = -B_i) = 1 - p_i$

ahol $A_i, B_i \in \mathbb{R}^+$.
Ahhoz tehát, hogy létezzen martingál mérték teljesülnie kell az

$$A_i > b_i > -B_i$$

és

$$p_i A_i - (1 - p_i)B_i = 0$$

feltételeknek. Itt az egyenletünket B_i-re rendezve, majd ezzel az egyenlőtlenségünk közepébe és jobb oldalába visszahelyettesítve:

$$B_i = \frac{p_i A_i}{1 - p_i} \to -\frac{p_i A_i}{1 - p_i} < b_i \to \frac{p_i A_i}{1 - p_i} > -b_i$$

Itt b_i ismeretében A_i-t válasszuk úgy, hogy b_i-nél nagyobb legyen és a $\frac{p_i A_i}{1-p_i} > -b_i$ egyenlőtlenséget is teljesítse. Ebből már a $B_i = \frac{p_i A_i}{1-p_i}$ összefüggés megadja B_i-t.

2.6.1. Példa faktor modellre

Tekintsük a konkrét példánkat két részvény esetén. A v kezdeti portfólió érték és az r kockázatmentes kamat legyen az előző példánkkal megegyezően 1 illetve, 0.05. Emellett legyen a hasznossági függvényünk a korábban is használt logaritmikus hasznossági függvény. Először is generáljuk le a μ_i, β_i, $\overline{\beta_i}$, p_i értékeket, ezek rendre:

- $\mu_1 = 0.057$, $\mu_2 = 0.0.025$, $\mu_3 = 0.28$
- $\beta_1 = 1.23$, $\beta_2 = 0.29$, $\beta_3 = 0.0.17$
- $\overline{\beta_1} = 1.18$, $\overline{\beta_2} = 1.63$, $\overline{\beta_3} = 0.64$
- $p_1 = 0.94$, $p_2 = 0.84$, $p_3 = 0.17$

Innen megkapjuk a b_i-ket a fenti képleteket használva:

- $b_1 = -0.05$
- $b_2 = -0.007$
- $b_3 = -0.43$

Ezután az A_i-kre vonatkozó két feltételt szem előtt tartva megfelelő ideig növeljük őket, majd a segítségükkel számoljuk ki a B_i-ket:

- $A_1 = 0.05$
- $A_2 = 0.09$
- $A_3 = 2.17$

Innen a B_i-k:

- $B_1 = 0.87$
- $B_2 = 0.49$
- $B_3 = 0.43$

Ezek segítségével vegyük az összes lehetséges eseményt, így kapjuk a már korábbiakhoz is hasonló pontokat, melyek egy-egy eseményhez tartoznak:

$$S_1(\omega_1), S_2(\omega_1)), (S_1(\omega_2), S_2(\omega_2)), (S_1(\omega_3), S_2(\omega_3)), (S_1(\omega_4), S_2(\omega_4)),$$

$$S_1(\omega_5), S_2(\omega_5)), (S_1(\omega_6), S_2(\omega_6)), (S_1(\omega_7), S_2(\omega_7)), (S_1(\omega_8), S_2(\omega_8)) =$$

$$= (1.07, 0.71), (1.07, 1.83), (0.43, 0.71), (0.43, 1.83), (0.83, 0.57), (0.83, 1.69), (0.19, 0.57), (0.19, 1.69)$$

Innen a szokásos. a poliéderünket meghatározó egyenlőségeinket és egyenlőtlenségünket felírva:

- $1.07q_1 + 1.07q_2 + 0.43q_3 + 0.43q_4 + 0.83q_5 + 0.83q_6 + 0.19_{q7} + 0.19q_8 = 1.05$
- $0.71q_1 + 1.83q_2 + 0.71q_3 + 1.83q_4 + 0.57q_5 + 1.69q_6 + 0.57_{q7} + 1.69q_8 = 1.05$
- $q_1 + q_2 + q_3 + q_4 + q_5 + q_6 + q_7 + q_8 = 1$
- $q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8 \geq 0$

Ennek a rendszernek most hat extremális pontja van, tehát most hat extremális martingál mértéket kaptunk:

- $Q_1 = (0.7, 0.29, 0, 0, 0, 0, 0, 0.017)$
- $Q_2 = (0.68, 0.3, 0, 0, 0, 0, 0.017, 0)$
- $Q_3 = (0.69, 0.25, 0, 0, 0, 0.07, 0, 0)$
- $Q_4 = (0.62, 0.31, 0, 0, 0.07, 0, 0, 0)$
- $Q_5 = (0.7, 0.28, 0, 0.02, 0, 0, 0, 0)$
- $Q_6 = (0.67, 0.3, 0.02, 0, 0, 0, 0, 0)$

Most generáljuk le a P valószínűségi mértéket: $P = (0.1, 0.09, 0.43, 0.1, 0.11, 0.05, 0.02, 0.1)$. Innen a Radon-Nikodym deriváltak:

- $\tilde{L}^{(1)} = (6.88, 3.12, 0, 0, 0, 0, 0, 0.18)$
- $\tilde{L}^{(2)} = (6.7, 3.31, 0, 0, 0, 0, 1.12, 0)$

- $\tilde{L}^{(3)} = (6.82, 2.68, 0, 0, 0, 1.25, 0, 0)$
- $\tilde{L}^{(4)} = (6.18, 3.38, 0, 0, 0.58, 0, 0, 0)$
- $\tilde{L}^{(5)} = (6.9, 3.03, 0, 0.24, 0, 0, 0, 0)$
- $\tilde{L}^{(6)} = (6.67, 3.29, 0.05, 0, 0, 0, 0, 0)$

Az előző példával megegyezően legyen a hasznossági függvényünk az $u(x) = \frac{x^\gamma}{\gamma}$, így deriváltjának inverze $\mathcal{I}(z) = z^{\frac{1}{\gamma-1}}$, ahol $x \in \mathbb{R}$ és $y \in \mathbb{R}_+$. Innen alkossuk meg a költségvetési egyenletekből álló egyenletrendszert, majd oldjuk meg a program segítségével:

$$\lambda_1 = 0.02, \lambda_2 = 0.15, \lambda_3 = 0.03, \lambda_4 = 0.32, \lambda_5 = 0.11, \lambda_6 = 0.33$$

Innen a

$$\overline{V}_N = \mathcal{I}\left(\sum_{j=1}^{r} \lambda_j \tilde{L}^{(j)}\right)$$

egyenletbe behelyettesítve, majd a programunkat alkalmazva:

$$\overline{V}_1 = (2.5, 1.77, 0.13, 0.16, 0.43, 0.18, 0.41, 0.05)$$

Így tehát a faktor modellre, 2 részvény és 1 közös faktor esetén megkaptuk, hogy a 8 különböző eseményre mennyi a portfólió optimális értéke egy lépés után, faktor modell esetén, 1 közös faktorral.
Az optimális stratégiát meghatározó ϕ vektor egyenletrendszerének felírása az első, két részvényt használó példával azonos módon történik, az így kapott rendszer:

- $2.5 = 1 + \phi_1(1.07 - 1) + \phi_2(0.71 - 1)$
- $1.77 = 1 + \phi_1(1.07 - 1) + \phi_2(1.83 - 1)$

Innen kapjuk a $\phi = (18.73, -0.65)$ vektort.

3. Optimális stratégiák különböző hasznossági függvények esetén

Mivel az eltérő befektetők más és más féleképpen viszonyulnak a kockázathoz, vannak kockázatkerülőbb és kockázatvállaóbb befektetők, ezért a portfóliójuk értékének maximalizáláshoz is más és más stratégia szükséges.

Ebben a fejezetben azzal fogunk foglalkozni, hogy az egyes befektetők kockázathoz való viszonya hogyan befolyásolja az optimális stratégia megválasztását, vagyis azt, hogy a portfóliónkban található egyes termékekből mennyit érdemes vásárolni, illetve eladni. Ezt a problémát szeretnénk matematikailag leírni és megoldást találni rá.

3.1. A hasznossági függvény szerepe

A befektetők kockázatvállalását, kockázathoz fűződő viszonyukat az u-val jelölt hasznossági függvényekkel fogjuk leírni. Ha a hasznossági függvény konkáv, az azt jelenti, hogy a derivált, vagyis a függvény meredeksége szigorúan monoton csökken. Azaz bármekkora is legyen a portfólió aktuális értéke, a befektető egyre több kockázatot vállal a portfólió értékének egységnyi növelése érdekében. Ebbe a kategóriába tartozik az *exponenciális* és a *exponential-identity*, melyeket a későbbiekben definiálunk majd. Ezzel szemben látni fogjuk, hogy vannak olyan hasznossági függvények is melyek az első szakaszukon (ameddig egy adott határt a portfólió értéke el nem ér, például $x = 0$-ig) konvexek, majd utána konkávak. Ez azt jelenti, hogy a derivált az első szakaszon szigorúan monoton nő, majd a második szakaszon szigorúan monoton csökken. Ez fejezi ki, hogy a befektetőnk egy bizonyos határig kockázatvállalóbb, de miután a portfóliója egy bizonyos értékhatárt átlépett, utána az érték növekedésével párhuzamosan egyre kevesebb kockázatot vállal. Ilyen függvények a *exponential-sqrt* és a *béta-gamma*, melyeket lentebb definiálunk. [3] [1]

3.2. A probléma matematikai értelemben

A hasznossági függvényünk a fent is említett $u : \mathbb{R} \to \mathbb{R}$ függvény. Végig egy részvénnyel fogunk dolgozni. A problémánk matematikai értelemben egy maximalizálási probléma:

$$\phi^\star(x) = \arg\max_{\phi} E\left[u(x + \phi \Delta S_1)\right] \tag{13}$$

ahol u a hasznossági függvény, ΔS_1 pedig a részvény értékének változása a 0. időpillanattól az 1. időpillanatig. Tehát a feladatunk az, hogy megtaláljuk azt vagy azokat a ϕ értékeket, melyekre az $E\left[u(x + \phi \Delta S_1)\right]$ kifejezés maximuma felvétetik.

Ezt egy adott x kezdeti értékre kiszámolva megkapjuk az optimális stratégiát, amelyet az $\overline{u}(x)$ hasznossági függvény ír le:

$$\overline{u}(x) = \sup_{\phi} E\left[u(x + \phi \Delta S_1)\right] = E\left[u(x + \phi^\star \Delta S_1)\right]$$

A részvény értékének változását egy választott eloszlással fogjuk modellezni. Azaz például ha ΔS_1 eloszlása az két pontra (az a és b pontra) koncentrálódik, akkor a várható értéket a $P(\Delta S_1 = +a) = p$, $P(\Delta S_1 = -b) = 1 - p$ valószínűségek segítségével tudjuk számolni.

Fontos megjegyezni, hogy most az előző fejezetben tárgyalt optimális érték megtalálásval szemben mindig csak 1 részvényünk van az egyszerűség kedvéért. Emellett most is egy lépésre

előre határoztuk meg az optimális stratégiát. Ha többlépéses esetet szeretnénk vizsgálni, akkor a korábban kiszámolt $\overline{u}$-t kell maximalizálnunk, majd újabb lépés esetén az így kapott új hasznossági függvényt, és így tovább. Tehát lényegében egy dinamikus programozási módszerrel mindig meghatározunk adott lépésszámra egy optimális hasznossági függvényt, majd a lépésszám növelése esetén ebből kiindulva kezdünk újra optimalizálni. A jelenlegi dolgozatban mi csak az egylépéses esettel dolgozunk, és nem foglalkozunk a többlépéses általánosításokkal.

A maximalizálási problémánkat abban az esetben a legegyszerűbb megoldani, ha az u hasznossági függvény differenciálható. Ekkor a problémánk átalakul a

$$E\left[u'(x+\phi\Delta S_1)\Delta S_1\right]=0 \tag{14}$$

egyenlet megoldásának problémájává.

Abban az esetben, ha a hasznossági függvény nem differenciálható, akkor direkt módszerrel kell maximalizálni. Pontosabban keresünk egy olyan $M(x)$ küszöböt, hogy a $[-M(x), M(x)]$ intervallumon kívül semmiképp ne legyen optimális a ϕ, majd ezután meghatározott pontossággal, numerikus módszerek segítségével meghatározzuk $\phi^\star(x)$-et.

Itt felmerül a kérdés, hogy létezik e mindig, minden x-re $\phi^\star(x)$ optimális érték. Erre ad választ a következő tétel.

6. Tétel. *Legyen $u:\mathbb{R}\to\mathbb{R}$ felülről korlátos, folytonos, monoton növő függvény és tegyük fel, hogy $u(0)=0$, $\lim_{x\to-\infty}u(x)=-\infty$. Emellett tegyük fel, hogy van egy felső korlátja a részvény árváltozásának abszolútértékének, azaz $|\Delta S_1|\leq D$, $D\in\mathbb{R}^+$ Valamint az arbitrázsmentesség miatt tegyük fel, hogy*

$$P(\Delta S_1>0)>0 és P(\Delta S_1<0)>0 \tag{15}$$

Ekkor minden x-re létezik $\phi^\star=\phi^\star(x)$ úgy, hogy

$$E\left[u\left(x+\phi^\star\Delta S_1\right)\right]=sup_{\phi\in\mathbb{R}}E\left[u\left(x+\phi\Delta S_1\right)\right]=\overline{u}(x)$$

[5]

Bizonyítás.

Lemma (1). *$\phi\to E\left[u\left(x+\phi\Delta S_1\right)\right]$ folytonos.*

Lemma bizonyítása. Ha $\phi_n\to\phi$, akkor ϕ_n korlátos, tehát létezik $K\in\mathbb{R}$, hogy $|\phi_n|\leq K$. Továbbá mivel u folytonos, ezért használhatjuk a folytonos függvényekre vonatkozó átviteli elvet, így

$$u\left(x+\phi_n\Delta S_1\right)\to u\left(x+\phi\Delta S_1\right) \tag{16}$$

majdnem minden $\omega\in\Omega$-ra. ΔS_1 és ϕ_n korlátosságából, valamint mivel u monoton növő:

$$\infty>u(\infty)\left(x+\phi_n\Delta S_1\right)\geq u\left(x-K|\Delta S_1|\right)\geq u\left(x-KD\right)$$

Ez alulról korlátolja u-t Tehát $u\left(x+\phi_n\Delta S_1\right)$-t dominnáltuk egy integrálható függvénnyel. Ezt és a fent bebizonyított (16)-os pont beli konvergenciát felhasználva alkalmazhatjuk a dominált konvergencia tételt, így:

$$E\left[u\left(x+\phi_n\Delta S_1\right)\right]\to E\left[u\left(x+\phi\Delta S_1\right)\right]$$

Lemma (2). *Ha $\phi_n\to\pm\infty$ akkor $E\left[u\left(x+\phi_n\Delta S_1\right)\right]\to-\infty$*

Lemma bizonyítása. ϕ_n felírható $\phi_n = |\phi_n| sgn(\phi_n)$ alakba. Tudjuk a (15)-ös pontból, hogy $P(sgn(\phi_n)\Delta S_1 < -\epsilon) \geq \epsilon$ igaz valamely $\epsilon > 0$-ra. Ezen felül válasszuk $|\phi_n|$-t olyan nagynak, hogy $|\phi_n|(-\epsilon) + x < 0$. Ekkor:

$$E\left[u\left(x+\phi_n\Delta S_1\right)\right] \leq E\left[u\left(x+\phi_n\Delta S_1\right)\mathbb{1}_{\{sgn(\phi_n)\Delta S_1<-\epsilon\}}\right]+E\left[u\left(x+\phi_n\Delta S_1\right)\mathbb{1}_{\{sgn(\phi_n)\Delta S_1\geq-\epsilon\}}\right] \leq$$

$$\leq E\left[u\left(x+\phi_n\Delta S_1\right)\mathbb{1}_{\{sgn(\phi_n)\Delta S_1<-\epsilon\}}\right] + C \leq u\left(|\phi_n|(-\epsilon)+x\right)\epsilon + C$$

Ez a kifejezés tart $-\infty$-hez ha $|\phi_n|$ tart ∞-hez, ezzel beláttuk a lemmát. Itt kihasználtuk, hogy az u hasznossági függvény korlátos, így az összeg második tagját C-vel felül tudtuk becsülni. Az utolsó becslés azért igaz, mert az indikátor várható értékét definíció szerint átírtuk az indikált esemény valószínűségére, amely legalább ϵ volt.

Most legyen ϕ_n olyan, hogy

$$E\left[u\left(x+\phi_n\Delta S_1\right)\right] \to \sup_{\phi\in\mathbb{R}} E\left[u\left(x+\phi\Delta S_1\right)\right]$$

ha $n \to \infty$ Két eset lehetséges:

1. ϕ_n nem korlátos. Ekkor létezik n_k részsorozat, hogy $|\phi_{n_k}| \to \infty$. De ekkor a második lemma miatt $E\left[u\left(x+\phi_n\Delta S_1\right)\right] \to -\infty$, ami lehetetlen, mert $\phi = 0$ választással tudjuk, hogy $u(x) \leq sup_{\phi\in\mathbb{R}} E\left[u\left(x+\phi\Delta S_1\right)\right]$. Azonban mivel x valamilyen valós szám (nem $-\infty$), ezért $u(x)$ is valamilyen valós szám, tehát nem $-\infty$ így ellentmondást kaptunk.

2. $|\phi_n|$ korlátos. Ekkor a Bolzano-Weierstraß létezik konvergens részsorozata, válasszuk $\phi^\star$-ot annak, ahova ez konvergál. Ekkor

   $$E\left[u\left(x+\phi_n\Delta S_1\right)\right] \to E\left[u\left(x+\phi^\star\Delta S_1\right)\right]$$

   ha $n \to \infty$ Most felhasználva az első lemmát, kapjuk:

   $$E\left[u\left(x+\phi^\star\Delta S_1\right)\right] = \sup_{\phi\in\mathbb{R}} E\left[u\left(x+\phi\Delta S_1\right)\right]$$

   Ez pont az amit be szerettünk volna látni, így befejeztük a tétel bizonyítását.

□

3.3. Eloszlástípusaink

Minden hasznossági függvényt három különböző eloszlás esetén vizsgálunk. Ebből kettő abszolút folytonos, a harmadik pedig diszkrét.

A legelső eset legyen az amikor a ΔS_1 egyenletes eloszlású a $[-a, b]$ intervallumon, ahol a és b nemnegatív valós számok.

A második eloszlás az (m, σ^2) paraméterű normális eloszlás.

A harmadik olyan p paraméterű diszkrét eloszlás, mely esetén a valószínűségi változó p valószínűséggel 1-et, $1-p$ valószínűséggel -1-t vesz fel. Itt p 0 és 1 között lehet.

Mindhárom esetben mi határozzuk meg az eloszlások paramétereit, azok a csatolt kódban is változtathatók, a korábban említett megszorítások mellett. Ezen felül azt is kiválaszthatjuk, hogy mekkora intervallumon vizsgáljuk az S_1 változását.

Abszolút folytonos eloszlások esetében használhatjuk, hogy ha f a ΔS_1 sűrűségfüggvénye, akkor a várható érték:

$$E[u(x+\phi\Delta S_1)] = \int_{-a}^{b} u(x+\phi S) * f(S) dS$$

vagyis csak az általunk vizsgált intervallumon integráljuk a hasznossági függvény és a sűrűségfüggvény (ami a részvény árváltozásától függ) szorzatát a részvény árváltozása, azaz ΔS_1 szerint. Ez a sűrűségfüggvény az alábbiak szerint írható fel különböző abzolút folytonos eloszlású ΔS_1-ek esetén:

- ΔS_1 egyenletes eloszlású az $(-a, b)$ intervallumon: $f(\Delta S_1) = \frac{1}{a+b}$
- ΔS_1 normális eloszlású az $(-a, b)$ intervallumon: $f(\Delta S_1) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\Delta S_1 - m}{\sigma})^2}$

Mostantól ha ΔS_1 abszolút folytonos eloszlású, akkor a ΔS_1 sűrűségfüggvényét $f(S)$-el fogjuk jelölni. Mivel $f(A)$–től eltekintve az egyenletes és normális eloszlás esetén a várható érték képlete nem különbözik, ezért sokszor ezt a két eloszlást egyben fogjuk kezelni, a sűrűségfüggvényt $f(S)$-el jelölve.

Diszkrét esetben használjuk az

$$E[u(x + \phi\Delta S_1)] = \sum_{i=1}^{n} u(x + \phi x_i) P(\Delta S_1 = x_i$$

összefüggést, ahol n az Ω eseménytér elemszáma, x_i pedig az i. esemény esetén való árváltozás nagysága. A most említett diszkrét esetben ez

$$E[u(x + \phi\Delta S_1)] = u(x + \phi)p + (u(x - \phi)(p - 1) \tag{17}$$

ahol p annak a valószínűsége, hogy a valószínűségi változónk 1-et vesz fel.

3.4. Optimalizálandó függvény *exponenciális* hasznossági függvény esetén

Az első hasznossági függvényünk az úgynevezett kontrollfüggvényünk, amivel tesztelni tudjuk számításaink, programunk példáink helyességét, mivel ennek viselkedését ismerjük. Pontosan képlettel a függvény $u(x) = -e^{-x}$. Ebben az esetben az $u(x + \phi\Delta S_1)$ hasznossági függvény:

$$u(x + \phi\Delta S_1) = -e^{-x-\phi\Delta S_1}$$

Így a várható érték abszolút folytonos esetben:

$$E[u(x + \phi\Delta S_1)] = \int_{-a}^{b} e^{-x-\phi S} f(S) dS$$

Ha ΔS_1 diszkrét, plusz-mínusz 1 eloszlású, akkor ennek az eloszlásnak a fent említett (15)-ös pont beli, azaz

$$E[u(x + \phi\Delta S_1)] = u(x + \phi)p + (u(x - \phi)(p - 1)$$

alakú.

3.5. Optimalizálandó függvény *exponential-identity* hasznossági függvény esetén

Az első összetettebb hasznossági függvényünk a exponential-identity függvény. Ez 0-nál kisebb számokra az identitásfüggvényként, 0-ra, illetve annál nagyobb számokra pedig egy speciális exponenciális függvényként. Ez képlettel:

$$u(x) = \begin{cases} x, & \text{ha } x < 0 \\ 1 - e^{-x}, & \text{ha } x \geq 0 \end{cases}$$

Ebben az esetben az $u(x+\phi\Delta S_1)$ hasznossági függvény:

$$u(x+\phi\Delta S_1)=\begin{cases}x+\phi\Delta S_1, & \text{ha } x+\phi\Delta S_1<0\\ 1-e^{-x-\phi\Delta S_1}, & \text{ha } x+\phi\Delta S_1\geq 0\end{cases}$$

Ha ΔS_1 abszolút folytonos eloszlású, akkor két esetre kell bontanunk mindig a feladatot: ha $\phi>0$, illetve ha $\phi<0$. Előbbi esetben az $x+\phi\Delta S_1<0$ pontosan akkor ha $\Delta S_1<\frac{-x}{\phi}$. Így a *exponential-identity* hasznossági függvény fenti definíciója alapján, a várható érték $\phi>0$ esetén (amit most E_1-val, míg a $\phi<0$ esetbrn E_2-el jelölünk)

$$E_1[u(x+\phi\Delta S_1)]=\int_{-a}^{min(b,\frac{-x}{\phi})}(x+\phi S)f(S)dS+\int_{min(b,\frac{-x}{\phi})}^{b}(1-e^{-x-\phi S})f(S)dS$$

Ha $\phi<0$, akkor annyi változik, hogy a $x+\phi\Delta S_1<0$ egyenlőtlenségben a ϕ-vel való átosztás ϕ negativitása miatt megfordítja a relációs jelet, így ez az egyenlőtlenség pontosan akkor teljesül, ha $S>\frac{-x}{\phi}$. A várható érték a $\phi>0$ esethez hasonlóan két integrál összegeként írható fel, ám most az integrálok tartalma felcserélődik egymással:

$$E_2[u(x+\phi\Delta S_1)]=\int_{-a}^{min(b,\frac{-x}{\phi})}(1-e^{-x-\phi S})f(S)dS+\int_{min(b,\frac{-x}{\phi})}^{b}(x+\phi S)f(S)dS$$

Nekünk tehát azt a ϕ értéket kell megkeresni, amire a $\max(E_1,E_2)$ kifejezés maximális.

Diszkrét, plusz-mínusz 1 eloszlás esetén a *exponenciális* hasznossági függvénnyel megegyező eset áll fent, és a (15)-ös pontban leírt formula írható fel a várható értékre.

3.6. Optimalizálandó függvény *power-sqrt* hasznossági függvény esetén

A harmadik hasznossági függvényünk a *exponential-identity* függvényhez hasonló, ám a lényeges eltérés ahhoz képest, hogy ez már nem konkáv, hanem az első szakasza konkáv és a második szakasza konvex. Ezt úgy definiáljuk, hogy a 0-nál kisebb számokra a szám abszolút értékének gyökének ellentetjét veszi fel, 0-ra, illetve annál nagyobb számokra pedig a *exponential-identity* függvényben használt speciális exponenciális függvénykén viselkedik. Ez képlettel:

$$u(x)=\begin{cases}-\sqrt{|x|}, & \text{ha } x<0\\ 1-e^{-x}, & \text{ha } x\geq 0\end{cases}$$

Ebben az esetben az $u(x+\phi\Delta S_1)$ hasznossági függvény:

$$u(x+\phi\Delta S_1)=\begin{cases}-\sqrt{|x+\phi\Delta S_1|}, & \text{ha } x+\phi\Delta S_1<0\\ 1-e^{-x-\phi\Delta S_1}, & \text{ha } x+\phi\Delta S_1\geq 0\end{cases}$$

Abszolút folytonos eloszlás esetén itt is két eset lehetséges, hogy $\phi>0$ vagy, hogy $\phi<0$. Itt a *exponential-identity* hasznossági függvénynél látott gondolatmenetet követve $\phi>0$ esetén a várható érték

$$E_1[u(x+\phi\Delta S_1)]=\int_{-a}^{min(b,\frac{-x}{\phi})}(-\sqrt{|x+\phi S|})f(S)dS+\int_{min(b,\frac{-x}{\phi})}^{b}(1-e^{-x-\phi S})f(S)dS$$

, míg $\phi < 0$ esetén

$$E_2[u(x+\phi\Delta S_1)] = \int_{-a}^{min(b,\frac{-x}{\phi})} (1-e^{-x-\phi S})f(S)dS + \int_{min(b,\frac{-x}{\phi})}^{b} (-\sqrt{|x+\phi S|})f(S)dS$$

és itt is azt a ϕ értéket keressük, amely maximalizálja a $\max(E_1, E_2)$ kifejezést.

Diszkrét, plusz-mínusz 1 eloszlás esetén az előző hasznossági függvényekkel megegyező (15)-ös pont beli kifejezés írható fel.

3.7. Optimalizálandó függvény *béta-gamma* hasznossági függvény esetén

A következő hasznossági függvényünk, a *béta-gamma* szintén egy nem konkáv hasznossági függvény, az előzőekhez hasonlóan 0-nál kisebb és nagyobb szakaszokra bontva. Ennek képlete:

$$u(x) = \begin{cases} 1-|x|^\gamma, & \text{ha } x < 0 \\ (1+x)^\beta, & \text{ha } x \geq 0 \end{cases}$$

ahol β és γ pozitív valós számok. Ebben az esetben az $u(x+\phi\Delta S_1)$ hasznossági függvény:

$$u(x+\phi\Delta S_1) = \begin{cases} 1-|x+\phi\Delta S_1|^\gamma, & \text{ha } x+\phi\Delta S_1 < 0 \\ (1+x+\phi\Delta S_1)^\beta, & \text{ha } x+\phi\Delta S_1 \geq 0 \end{cases}$$

Itt is a korábbiakban látott gondolatmenetet követve ϕ pozitivitása szerint két eset írható fel a várható értékre, ami most $\phi > 0$ esetén

$$E_1[u(x+\phi\Delta S_1)] = \int_{-a}^{min(b,\frac{-x}{\phi})} (1-|x+\phi S|^\gamma)f(S)dS + \int_{min(b,\frac{-x}{\phi})}^{b} ((1+x+\phi S)^\beta)f(S)dS$$

illetve $\phi < 0$ esetén

$$E_2[u(x+\phi\Delta S_1)] = \int_{-a}^{min(b,\frac{-x}{\phi})} ((1+x+\phi S)^\beta)f(S)dS + \int_{min(b,\frac{-x}{\phi})}^{b} (1-|x+\phi\ S|^\gamma)f(S)dS$$

és itt is a $\max(E_1, E_2)$ kifejezést maximalizáló ϕ-t keressük.

Az előző függvényeknél használt, plusz-mínusz 1 eloszlásra vonatkozó hasznossági függvényekkel megegyező képlet írható itt is fel.

3.8. Megfigyelések az optimális $\phi^\star$-al és várható értékkel kapcsolatban

A most következő példákban, ábrákon 1000 darab x kezdeti portfólió értékre számoltuk ki az $\overline{u}$ illetve $\phi^\star$ függvényeket (kivétel ez alól a *power-sqrt* hasznossági függvény esetén ábrázolt $\phi^\star$ függvény, ahol a szemléletesebb ábra és a numerikus hiba csökkentésének érdekében 100000 x értékre végeztük el a számolást). Az előbbit a $(-5,5)$, utóbbit a $(-10,10)$ intervallumon vettük és ábrázoltuk. Mind az uniform, mind a normális eloszlást a $(-5,10)$ intervallumon értelmeztük, azon kívül az eloszlásfüggvény értékeit 0-nak tekintettük. A normális eloszlás $(1,1)$ paraméterű, azaz a várható értéke és szórásnégyzete is 1. A plusz-mínusz 1 eloszlásnál $p = 0.3$, azaz annak a valószínűsége, hogy az árváltozás 1. A *béta-gamma* hasznossági függvénynél $\beta = 0.5$, $\gamma = 1$. Amikor az optimalizálási problémánkat megoldjuk ϕ alsó korlátja -100, felső korlátja pedig 100.

3.8.1. Exponenciális függvény

Először is vezessük be úgynevezett ellenőrző függvénynek az exponenciális függvényt, pontosabban $u(x) = -e^{-x}$-et. Ezzel azt fogjuk tudni ellenőrzini, hogy jól működik-e a programunk. Ugyanis igaz a következő állítás:

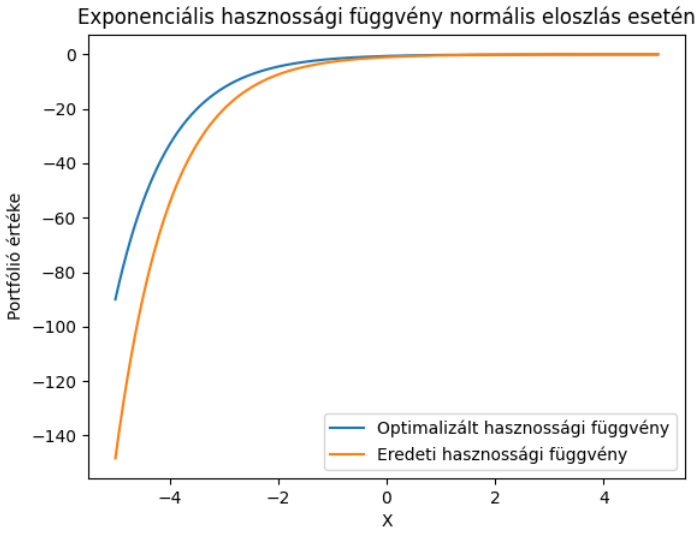
1. Állítás. *Az $u(x) = -e^{-x}$ hasznossági függvény optimális várható értéke önmagának konstanszorosa, azaz minden x pontban $c \cdot (-e)^{-x}$, ahol $c \in \mathbb{R}$.*

Bizonyítás. Teljesül a következő lánc:

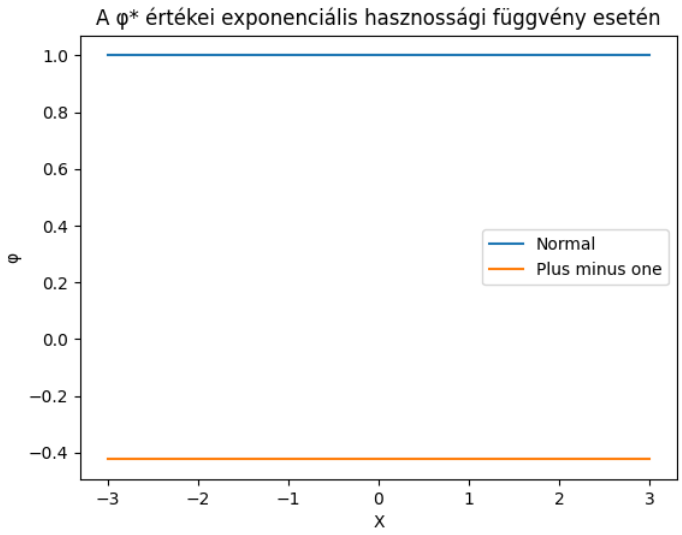
$$\overline{u}(x) = sup_{\phi} E\left[u\left(-e^{-\phi \Delta S_1 - x}\right)\right] = -e^{-x} \cdot sup_{\phi} E\left[-e^{-phi \Delta S_1}\right]$$

Mivel $E\left[-e^{-phi \Delta S_1}\right]$ nem függ x-től, ezért most a $c = E\left[-e^{-phi \Delta S_1}\right]$ megfelelő. □

Azaz az állításunk alapját exponenciális hasznossági függvény esetén $\overline{u}(x)$ megőrzi $u(x)$ alakját. Ezt láthatjuk az alábbi ábrán, normális eloszlás esetében:



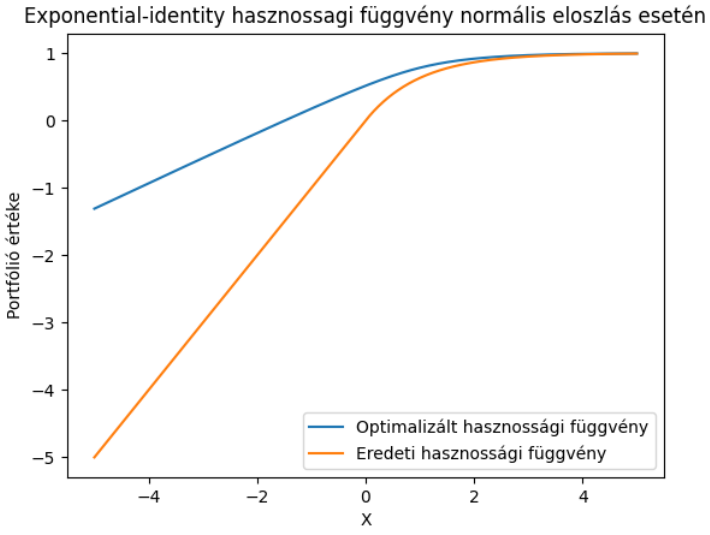
Itt az állításunk alapján azt is megfigyelhetjük, hogy a $\phi^{\star}$ tényleg nincsen hatással az $\overline{u}$-ra, azaz konstans:



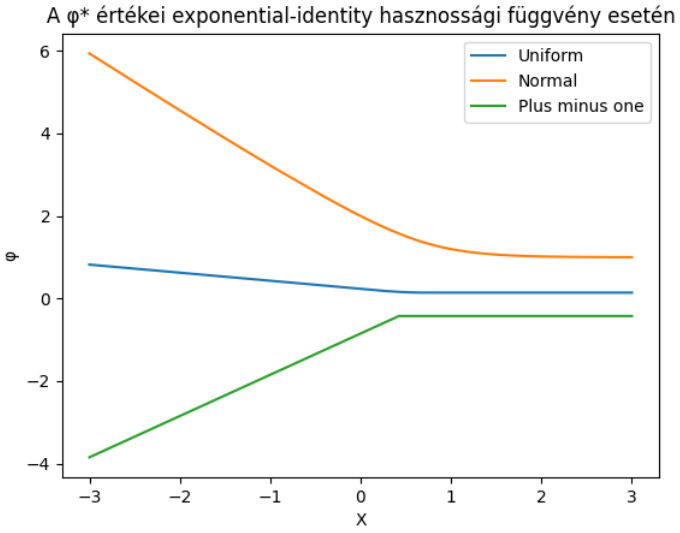
3.8.2. Konkáv hasznossági függvény

Most nézzük a *exponential-identity* hasznossági függvényünket, azaz az

$$u(x) = \begin{cases} x, & \text{ha } x < 0 \\ 1 - e^{-x}, & \text{ha } x \geq 0 \end{cases}$$



Valamint a $\phi^\star$ függvény különböző eloszlások esetén:



Mivel az optimális hasznossági függvényünk 0-nál kisebb helyettesítési értékekre az identitás, azaz $x + \phi^\star \Delta S_1$ értékeket vesz fel (ez a $\phi^\star$ az adott x értékhez tartozó optimális ϕ-t jelöli, nem pedig a $\phi^\star$ függvényt), így az első fele a $\phi^\star$ függvényeknek majdnem lineáris. Ez a következővel magyarázható:

$$E\left[u(k \cdot x + \phi \Delta S_1)\right] = E\left[u(k \cdot (x + \frac{\phi}{k} \Delta S_1)\right] \approx k \cdot E\left[u(x + \frac{\phi}{k} \Delta S_1)\right]$$

ahol k egy tetszőleges valós szám Itt az utolsó egyenlőség azért közelítő, mivel az u függvény alakját ΔS_1 eloszlása befolyásolja, és ezért csak közelítőleg lineáris negatív helyettesítési értékekre. Mivel

$$\arg\max_\phi k \cdot E\left[u(x + \frac{\phi}{k} \Delta S_1)\right] = \arg\max_\phi E\left[u(x + \frac{\phi}{k} \Delta S_1)\right] = k \cdot \arg\max_\phi E\left[u(x + \phi \Delta S_1)\right]$$

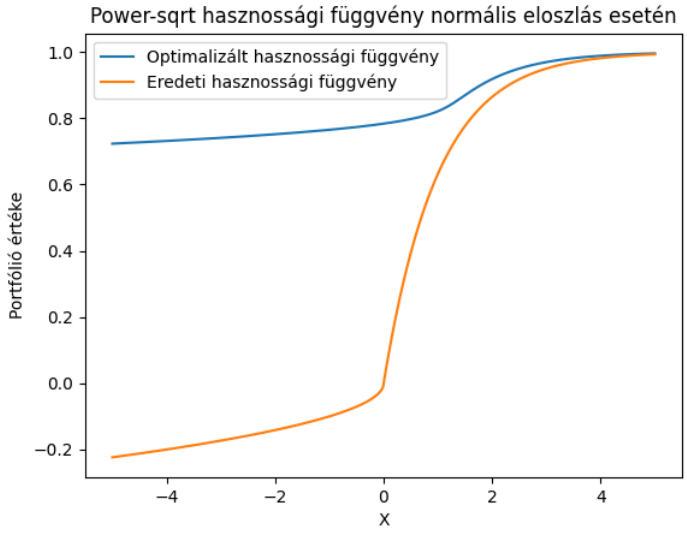
ezért $\phi^\star(k \cdot x) \approx k \cdot \phi^\star(x)$, vagyis tényleg majdnem lineáris a $\phi^\star$ függvény negatív értékekre.

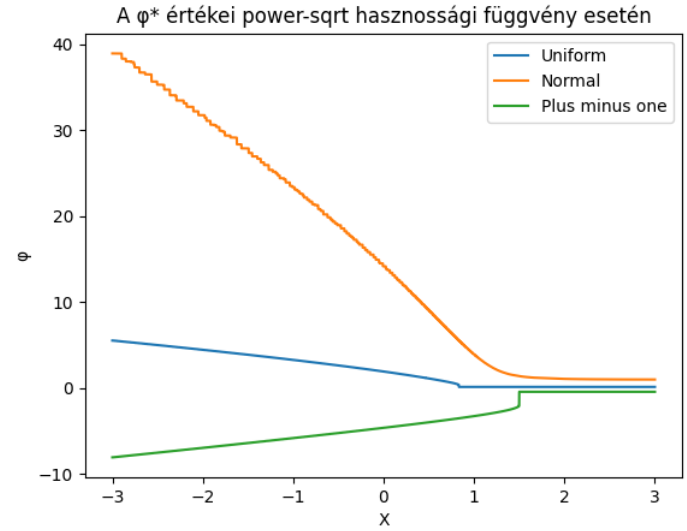
3.8.3. Nem konkáv hasznossági függvények

Az első nem konkáv hasznossági függvényünk a már korábban is említett *power-sqrt*, képlettel:

$$u(x) = \begin{cases} -\sqrt{|x|}, & \text{ha } x < 0 \\ 1 - e^{-x}, & \text{ha } x \geq 0 \end{cases}$$

Az ehhez tartozó $\overline{u}(x)$ illetve $\phi^\star$ függvények:





Itt az előző esethez hasonlóan megfigyelhetjük, hogy a $\phi^\star$ függvény alakja 0-nál kisebb helyettesítési értékekre az u alakjához, azaz $-\sqrt{|x|}$ alakjához hasonló. Ennek magyarázata szintén az előző esethez hasonló:

$$E\left[u(k\cdot x+\phi\Delta S_1)\right]=E\left[u(k\cdot(x+\frac{\phi}{k}\Delta S_1)\right]\approx-\sqrt{|k|}\cdot E\left[u(x+\frac{\phi}{k}\Delta S_1)\right]$$

ahol k szintén tetszőleges valós szám. Itt az egyenlőség azért csak közelítőleg igaz, mert a hasznossági függvényünk negatív helyettesítési értékekre csak közelítőleg $-\sqrt{|x|}$ alakú, hiszen befolyásolja ΔS_1 eloszlása. Mivel

$$\arg\max_\phi -\sqrt{|k|}\cdot E\left[u(x+\frac{\phi}{k}\Delta S_1)\right]=\arg\max_\phi E\left[u(x+\frac{\phi}{k}\Delta S_1)\right]=k\cdot\arg\max_\phi E\left[u(x+\phi\Delta S_1)\right]$$

ezért $\phi^\star(k \cdot x) \approx -\sqrt{|k|} \cdot \phi^\star(x)$, vagyis tényleg majdnem $-\sqrt{|y|}$ alakú a $\phi^\star$ függvény negatív értékekre.

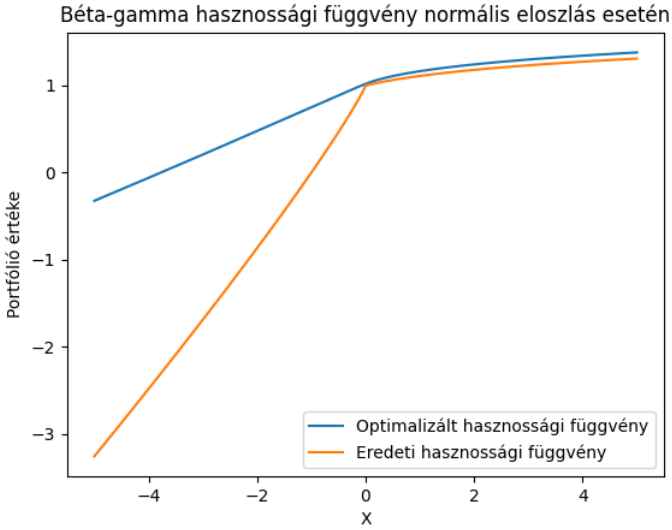
A második nem konkáv hasznossági függvényünk az úgynevezett *gamma-béta*, amely képlettel:

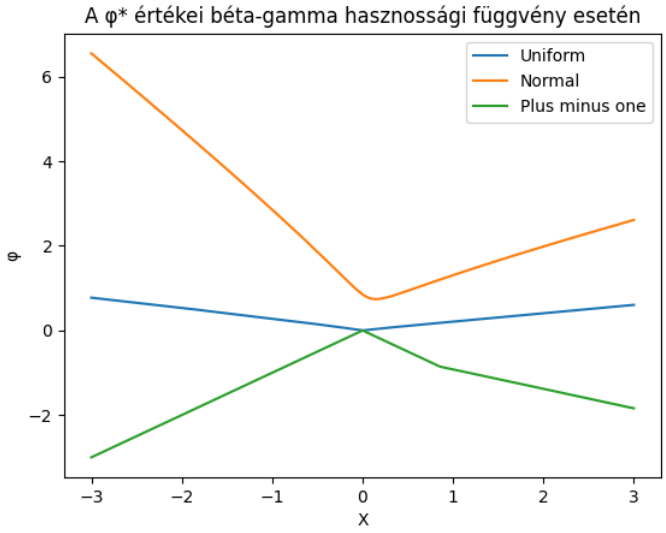
$$u(x) = \begin{cases} 1 - |x|^\gamma, & \text{ha } x < 0 \\ (1+x)^\beta, & \text{ha } x \geq 0 \end{cases}$$

ahol β és γ pozitív valós számok, ezeket mi választjuk ki. Most legyen $\beta = 0.15$ és $\gamma = 0.9$, azaz most a hasznossági függvényünk:

$$u(x) = \begin{cases} 1 - |x|^{0.9}, & \text{ha } x < 0 \\ (1+x)^{0.15}, & \text{ha } x \geq 0 \end{cases}$$

Ekkor az $\overline{u}(x)$ és a $\phi^\star$ függvények:





Itt azt figyelhetjük meg, hogy mind a 0-nál kisebb mind az azoknál nagyobb értékekre a $\phi^\star$ függvény közel lineáris alakú. Ez azzal magyarázható, hogy mind $1 - |x|^{0.9}$, mind $(1 + x)^{0.15}$ közel lineáris alakú, lineáris hasznossági függvény esetében pedig a *power-identity* hasznossági függvénynél láttuk, hogy közelítőleg lineáris alakú lesz a $\phi^\star$.

4. Programkódok leírása

Az alábbiakban a különböző példákban használt programkódok és az azokban használt matematikai módszerek rövid áttekintése következik.

4.1. Optimális értékek kiszámolása martingál módszerrel

4.1.1. Pontgenerálás-generate_points

Ahhoz, hogy a martingál módszert tudjuk alkalmazni a problémánk megoldására, először is szeretnénk példákat generálni, amelyekre az alkalmazás meg fog történni. Két fontos feltételünk van:

- A megfelelő dimenziós csupa 1 pont legyen benne a generált pontjaink konvex burkának lezártjában.
- Minden pontunk minden koordinátája szigorúan pozitív kell, hogy legyen, hiszen ezek a részvények árainak egyes események esetén való változásainak szorzói.

Tehát szeretnénk adott számú $S_1, S_2, \ldots, S_n$ részvényekhez és adott k elemszámú Ω eseménytérre megadni az

$$(S_1(\omega_i, S_2(\omega_i), \ldots, S_n(\omega_i))$$

alakú pontokat, ahol i fut 1-től k-ig. Vagyis az i. pont j. koordinátája azt jelöli, hogy az i ω_i esemény esetén hányszorosára változik a j. S_j részvény értéke.

Ehhez venni fogjuk a megfelelő dimenziós bázisvektorokat, majd ezekre alkalmazzuk a Graham-Schmidt ortogonalizációt. Így egy ortonolmált bázist kapok. Ezután ezeknek az ortonormált bázisvektoroknak veszem a random lineáris kombinációit és így kapom meg a generálandó pontok első felét. Ezek továbbra is lineárisan függetlenek. Ezután azért, hogy a csupa 1 pont a konvex burok belsejében legyen, minden ponthoz vesszük az általa és a csupa 1 pont által definiált egyenest, majd ennek a csupa 1 ponttal átellenes felén random felveszünk egy másik pontot, itt is figyelve arra, hogy minden pont minden koordinátája szigorúan pozitív legyen. Így megkaptuk a kívánt pontjainkat, azaz tudjuk, hogy adott esemény esetén adott részvény értéke hogyan változik.

4.1.2. Pontgenerálás faktor modell esetén-generate_points_factor

Először is generáljunk le annyi μ-t, β-t és $\overline{\beta_i}$-t amennyi a részvények száma plusz 1-et a tőzsdeindexnek. Emellett generáljuk le a p_i-ket is, tehát azokat a valószínűségeket, amik megadják, hogy mennyi valószínűséggel vesznek fel az ϵ_i-k A_i-t (ez már meghatározza, hogy B_i-t $1 - p_i$ valószínűséggel vesznek fel). Ezeket igazából tetszőlegesen választjuk meg, de a mostani kód esetében a μ_i-ket a $(0, 0.5)$, a β-kat és $\overline{\beta}$-kat a $(0, 2)$, a p_i-ket pedig a $(0, 1)$ intervallumból választjuk egyenletes eloszlással, a korábban is használt NP.RANDOM csomag UNIFORM függvényével. A β_i-ket és $\overline{\beta_i}$-ket a fő fejezetben is említett $b_1 = -\frac{\mu_1}{\beta_1}$ illetve $b_i = -\frac{\mu_i}{\beta_i} + \frac{\mu_1 \beta_i}{\beta_i \beta_1}$ képletek segítségével a generált adatokból számoljuk. Ezután minden A_i-t úgy választunk meg, hogy addig növeljük egy adott EPSILON értékkel amíg mind az $A_i > b_i$, mind az $A_i > -\frac{b_i(1-p_i)}{p_i}$ feltételnek meg nem felel. Ezután, hogy ϵ_i várható értéke 0 legyen, B_i-t $A_i - b\H{o}l$, a $B_i = \frac{p_i A}{1-p_i}$ képlet segítségével számoljuk. Ezután vesszük az összes lehetséges eseményt és az ezekhez tartozó pontokat legeneráljuk. Egy esemény egy kombinációja az ϵ_i-k által felvett értékeknek, így mivel most egy közös faktor van,

az Ω eseményterünk 2^{n+1} elemű lesz, ahol n a részvények száma. A kódban ez úgy van megvalósítva, hogy minden ilyen kombinációt megfeleltetünk egy bináris 01 sorozatnak, majd ezeken végigiterálva hozzuk létre az egyes eseményekhez tartozó pontokat.

4.1.3. Extremális mértékek megtalálása-get_extreme

Miután megvannak a pontjaink ezekből létre kell hozni a megfelelő poliédert a következő alakban:

- $\sum_{k=1}^{n} q_k S_j(\omega_k) = 1 + r, \quad j = 1, \ldots, m$
- $\sum_{k=1}^{n} q_k = 1$
- $q_k \geq 0, \quad k = 1, \ldots, n$

ahol n az Ω állapothalmaz számossága, azaz az állapotok száma, m pedig a részvényeink száma. A feladatunk tehát a fenti egyenletekből és egyenlőtlenségekből álló rendszer által definiált poliéder extremális pontjainak megkeresése. Ehhez először is inicializáljuk a feladatunkhoz tartozó egyenletrendszert, mégpedig $Ax \leq b$ alakban. Tehát leírjuk az egyenleteinket ilyen egyenlőtlenségként, majd az ellentetjüket is leírjuk ilyen alakban, hiszen $Ax \leq b$ és $-Ax \leq -b$-ből következik, hogy $Ax = b$, ami pont az amit szeretnénk. A nemnegativitási feltétel betartása miatt vezessük be a $-q_k \leq 0$ egyenlőtlenségeket minden $k = 1, \ldots, n$-re. Így már egy Axb alakú egyenlőtlenség rendszert kaptunk, amire alkalmazhatjuk a Pythonban meglévő PYPOMAN csomagot. Ennek a csomagnak a
COMPUTE_POLYTOPE_VERTICES nevű függvénye pontosan azt csinálja amit szeretnénk: végighalad a politópunk csúcsain, és listába szedi őket.

4.1.4. P valószínűségi mérték generálása-generate_probability_measure

Ez a függvény generál nekünk egy random valószínűségi mértéket, melynek dimenziója annyi, mint az extremális martingál mértékeink dimenziója. A koordinátái random vannak kiválasztva, olyan módon, hogy a $(0, 1)$ intervallumot random felosztjuk dimenziónyi részre.

4.1.5. Radon Nikodym deriváltak kiszámítása-calculate_radon_nikodym

Itt történik a Radon-Nikodym deriváltak, azaz az $L^{(i)}$-k kiszámítása. $L^{(i)}$-t úgy kapjuk meg, hogy a Q_i extremális mérték koordinátáit elosztjuk a P valószínűségi mérték megfelelő koordinátáival. Azaz $L^{(i)}(\omega_j) = \frac{Q_i(\omega_j)}{P(\omega_j)}$, $i = 1, \ldots, r$, $j = 1, \ldots, n$, ahol r az extremális martingál mértékek és így az extremális Radon-Nikodym deriváltak száma, n pedig az Ω halmaz elemszáma, azaz a lehetséges események száma. Ezután még minden Radon-Nikodym derivált minden koordinátáját elosztjuk a B_1-el, azaz a kötvényünk 1. időpillanatbeli értékével az $\tilde{L} = B_N^{-1} L$ összefüggés alapján.

4.1.6. Költségvetési egyenletrendszer felírása-create_budget_equations_fsolve_form

Itt történik a költségvetési egyenletrendszer inicializálása. Először is megnézi a függvényünk, hogy mi a hasznossági függvényünk, hiszen ekkor már $\mathcal{I} = (u')^{-1}$ ismert. Ezután a hasznossági függvénynek megfelelő segédfüggvény hívódik meg, amik felírnak egy-egy egyenletet a Radon-Nikodym deriváltak, a P valószínűségi mérték és az $\mathcal{I}$ függvény ismeretében. Egy segédfüggvényünk egyszeri hívása egy Radon-Nikodym deriválthoz tartozó egyenletet állít elő,

ezért a külső függvényünk mindegyik Radon-Nykodim deriváltra meghívja a megfelelő segédfüggvényt és a kapott egyenletekből elkészíti egy az egyenletrendszert reprezentáló listát. Fontos, hogy itt az egyenleteink függvényekkel vannak reprezentálva mivel a SCIPY.OPTIMIZE.FSOLVE csomag függvények zérushelyeit tudja megkeresni. Itt az egyenletrendszerünk egy függvényéből álló vektor, erre az előbb említett függvényt alkalmazva pont az egyenletrendszerünk megoldásait kapjuk majd

4.1.7. Költségvetési egyenletrendszer megoldása-solve_budget_equations

Az előző függvényünkel kapott egyenletrendszert továbbadjuk a megoldó függvénynek. Ez a SCIPY.OPTIMIZE csomag MINIMIZE függvényét használja. Ezzel a függvénnyel fogjuk minimalizálni az egyenleteink abszolútértékeinek összegét. Ez azért jó, mert az egyenletrendszerünket úgy rendeztük, hogy a jobb oldalon csupa 0 legyen, így ha az aktuális becslésünket behelyettesítjük a jobb oldalak abszolútértékeinek összegébe, akkor pont a becslésünk 0-tól, azaz pontos megoldástól való távolságát kapjuk meg. Ez maga a becslésünk hibája, mi ezt szeretnénk kellően kicsire szorítani. Az algoritmusunk fő lépései:

1. Kezdeti becslés: Megadunk egy kezdeti becslést az egyenlet gyökeire, ez az *initial_guess* változó.
2. Megszorítások: Mivel $_i$-kre nemnegativitási feltétel vonatkozik, ezért meg kell adni a 0-t alsó korlátnak, aminél a megoldás értéke nem lehet kisebb, ez jelenik meg a *bounds* paraméterben.
3. Optimalizálási algoritmus: A függvény a Broyden-Fletcher-Goldfarb-Shanno algoritmus egy változata, amely tud korlátokat kezelni.
   Az algoritmus a Hessian-mátrixot becsüli meg a gradiens becslésének segítségével. A Hessian-mátrix leírja a függvényünk alakját, ami segít eldönteni, hogy milyen irányba kell mennünk, hogy minimalizálhassuk a kifejezést. Ha megvan a keresett irány, akkor meg kell találnunk a megfelelő lépéshosszt. Kiindulunk egy kezdeti lépéshosszból és megnézzük, hogy ha az irányunk mentén lépünk ennyit, akkor megfelelő mértékben csökkent-e a függvény értéke (a megfelelő mérték valamilyen adott küszöb). Ha igen, fogadjuk el a lépéshosszt és becsüljük újra a Hessian-mátrixot és így tovább. Ezt a két lépést ismételjük, amíg az iterációk közötti javítás mértéke egy adott küszöb alá nem megy vagy elértük a lehetséges iterációk számának felső határát.
4. Ha a 4-es pont végén említett kettő feltétel egyike bekövetkezik, leállítjuk az algoritmust és az aktuális becslésünket adjuk vissza.

4.2. Optimális stratégiák kiszámolása

4.2.1. Maximalizálandó függvény létrehozása-create_function

A feladat egyik legnagyobb része, hogy létrehozzuk azt a függvényt, amellyel meg szeretnénk találni azt a ϕ-t amire a maximális értéket felveszi, illetve utána magát a maximális értéket is. Azaz most szeretnénk felírni az

$$E[u(x + \phi \Delta S_1)]$$

függvényt különböző u hasznossági függvények és ΔS_1 különböző eloszlásai esetén. Abszolút folytonos eloszlás esetén, tehát ha az árváltozásnak van sűrűségfüggvénye, akkor alkalmazhatjuk

az

$$E[u(x+\phi\Delta S_1)] = \int_{\mathbb{R}} u(x+\phi y) f(y) dy$$

összefüggést, ahol f a ΔS_1 sűrűségfüggvénye. Ha az $u(x)$ hasznossági függvényünk szakaszos módon van definiálva, azaz x értékétől függően "különböző függvényként viselkedik", akkor a várható értéket a megfelelő szakaszokon vett integrálok összegeként írhatjuk fel. Itt az integrálás vagy az $\mathbb{R}$-en vagy a megadott szűkebb intervallumon történik numerikus módszerrel a SCIPY.INTEGRATE csomag QUAD függvényét használva.

Ez a függvény az úgynevezett adaptív kvadratúra módszert használja. Ennek lényege, hogy egy adott intervallumon vett integrálra úgy adunk közelítést, hogy megbecsüljük az adott intervallumon vett integrált, majd felosztjuk részintervallumokra, azokon is megcsináljuk ezt a becslést, majd a részintervallumokon vett becslések összegét az egész intervallumon vett becsléssel összehasonlítjuk. Ha kellően közel van egymáshoz a két becslés (azaz valamilyen adott tolerancia alatt), akkor továbblépünk. Ha nem, akkor még finomabb felosztását vesszük az intervallumnak és megismételjük a becslést. Az egyes intervallumokon vett integrálok becslésére egy egyszerű kvadratikus becslést, például a trapézszabályt vagy a Simpson-formulát használja.
Diszkrét esetben pedig a várható érték az

$$E[u(x+\phi\Delta S_1] = \sum_{i=1}^{n} u(x+\phi x_1) P(\Delta S_1 = x_i)$$

összefüggéssel adhatjuk meg, ahol n a Ω eseménytér elemszáma, x_i pedig az i. esemény esetén való árváltozás nagysága.
Ha szükséges, akkor az egyes hasznossági függvényeket külön is definiáljuk, hogy más függvényekben meg tudjuk őket hívni, ilyen például a POWER_IDENTITY függvény, amely az $u(x) = \{x|x < 0, -e^{-x}|x \geq 0\}$ hasznossági függvény implementációja.

4.2.2. Optimális ϕ és várható érték kiszámolása- calculate_optimal_strategy

Ebben a függvényben először is teszteljük a bemeneti paraméterek helyességét. Ide tartozik, hogy e megadott hasznossági függvény és eloszlás implementálva van e, illetve például az, hogy megadott valószínűség esetén az érték 0 és 1 között legyen.
Ezután az x-re vonatkozó megadott intervallum és az osztópontok száma alapján, az adott intervallumon generáljuk le a kiértékelési pontokat, vagyis azon x értékeket, amelyekre az optimális ϕ-t és várható értéket keresni fogjuk. Az optimális ϕ, azaz a $\phi^\star$ megkeresése a korábbiakban is használt SCIPY.OPTIMIZE csomag MINIMIZE függvényével fog megvalósulni. A fent leírt CREATE_FUNCTION függvénnyel elkészítjük a várható értéket minden adott x pontra, vagyis azt a függvényt amelyet maximalizálni szeretnénk. Majd a MINIMIZE függvény segítségével megkeressük a maximum helyét, mégpedig úgy, hogy az ellentettjét minimalizáljuk. Ezután a maximum értékét úgy kapjuk, hogy az optimális ϕ-t visszahelyettesítjük a függvényünkbe.
Az így kapott $x-phi^\star$ és $x-E[u(x+\phi^\star\Delta S_1)]$ párokat egy szótárba helyezzük, majd a PLOT_PHI_STAR_OR_EXP_VALUE függvény segítségével grafikonon ábrázoljuk őket.

Összefoglalás

A dolgozat során megismerkedtünk a portfólió optimalizálás legfontosabb matematikai és közgazdaságtani alapjaival, és megvizsgáltuk az ezekkel kapcsolatban felmerülő alapvető problémákat.

A matematikai alapfogalmak és legfontosabb tételek megismerése után, a második fejezetben készítettünk egy programot, mely képes kiszámolni konkáv hasznossági függvény esetén, egylépéses esetben és véges Ω eseményhalmaz esetén a portfólió optimális értékét. Először kézzel, majd ennek a programnak a segítségével megoldottunk több portfólió optimalizálási problémát. Ezek egy része általunk véletlenszerűen generált modellen alapult, míg szerepelt olyan is, melyet egy ismert közdazdaságtani modell alapján építettünk fel.

Ezután a harmadik fejezetben konkáv és nem konkáv hasznossági függvények esetében is megvizsgáltuk az optimális stratégiát, és az ezekhez tartozó optimális portfólió értékeket. Ezeket részletesen elemeztük és összevetettük az optimalizálás nélküli portfólió értékeivel.

A jövőben érdemes lehet megvizsgálni a két problémánkat többlépéses esetben is. Ez az első problémánál, amit a második fejeztben vizsgáltunk, nem eredményez egy matematikaliag jelentősen nehezebb feladatot. Itt nincsen technikai akadálya a többlépéses eset megoldásának, csak a számítási igény és a számítás ideje nőne meg. A harmadik fejezetben tárgyalt probléma esetében azonban, ha többlépéses eseteket vizsgálnánk, akkor minden újabb lépésben az előző lépésben kapott optimalizált hasznossági függvény venné fel az eredeti hasznossági függvény szerepét, egyfajta dinamikus programozási módszerrel lehetne megoldani a problémánkat. Ennek eredményeképp a többlépéses eset nem csak számításigényesebb, de matematikai értelemben is jelentősen bonyolultabb az egylépéses esetnél.

Hivatkozások

[1] Daniel Kahneman Amos Tversky. *Advances in prospect theory: Cumulative representation of uncertainty*, pages 297–323. Journal of Risk and Uncertainty, 1992.

[2] Wolfgang J. Runggaldier Andrea Pascucci. *Financial Mathematics: Theory and Problems for Multi-period Models*, pages 1–93. Springer, 2012.

[3] Amos Tversky Daniel Kahneman. *Prospect Theory: An Analysis of Decision under Risk*, pages 263–291. Econometrica, 1979.

[4] C. M. Shetty Mokhtar S. Bazaraa, Hanif D. Sherali. *Nonlinear Programming: Theory and Algorithms, 3rd Edition*, pages 257–263. Wiley, 2006.

[5] Miklós Rásonyi. *Optimal Investment with Nonconcave Utilities in Discrete-Time Markets*. Society for Industrial and Applied Mathematics, 2015.

Kódjegyzék

Optimális értékek kiszámolása martingál módszerrel

```
import numpy as np
from pypoman import compute_polytope_vertices
import scipy.linalg
import scipy.stats
from scipy.optimize import minimize
from copy import deepcopy
import random
from typing import Union, List, Callable
from itertools import product

CLOSE_TO_ZERO = 1e-6
BIG_NUMBER = 1e6
IMPLEMENTED_UTILITY_FUNCTIONS = ['log', 'power']
MAX_ITERATIONS = 1000
EPSILON = 0.1


def generate_points_factor(number_of_assets: int) -> List[List[float]]:
    mus = [np.random.uniform(0, 0.5) for _ in range(number_of_assets + 1)
        ]
    betas = [np.random.uniform(0, 2) for _ in range(number_of_assets + 1)
        ]
    overline_betas = [np.random.uniform(0, 2) for _ in range(
        number_of_assets + 1)]
    b_list = [-mu / overline_beta + (mus[0] * beta) / (overline_beta *
        overline_betas[0]) for mu, beta, overline_beta in
              zip(mus, betas, overline_betas)]
    b_list[0] = -mus[0] / overline_betas[0]
    p_list = [np.random.uniform(0, 1) for _ in range(number_of_assets +
        1)]
    A_list = list()
    B_list = list()
    for p, b in zip(p_list, b_list):
        A = b + EPSILON
        while A <= (-b * (1 - p) / p):
            A += EPSILON
        while A in A_list:
            random_addition = random.uniform(0, 1)
            A += random_addition
        B = (p * A) / (1 - p)
        A_list.append(A)
        B_list.append(B)

    binary_combinations = list(product([0, 1], repeat=number_of_assets +
        1))
    points = list()
    for combination in binary_combinations:
        if combination[0] == 1:
            epsilon1 = A_list[0]
```

```
44          else:
45              epsilon1 = B_list[0]
46          point = [mus[i] + betas[i] * epsilon1 + overline_betas[i] *
                A_list[i] if combination[i] == 1 else
47                   mus[i] + betas[i] * epsilon1 + overline_betas[i] *
                         B_list[i] for i in range(1, number_of_assets + 1)]
48          points.append(point)
49
50      return points
51
52
53  def generate_points(number_of_assets: int, half_of_events: int) -> List[
        List[float]]:
54      all_one_point = np.ones(number_of_assets)
55      point_list = list()
56      points_as_arrays = list()
57      vertices = np.eye(number_of_assets)
58      Q, R = scipy.linalg.qr(vertices)
59      Q = Q / np.linalg.norm(Q, axis=1, keepdims=True)
60
61      while len(point_list) < half_of_events:
62          coefficients = np.random.uniform(CLOSE_TO_ZERO, 1.5,
                number_of_assets)
63          length_multiplier = np.random.uniform(0.5, 2)
64          new_point = Q.T @ (coefficients * length_multiplier)
65          new_point += CLOSE_TO_ZERO
66          for other_point in point_list:
67              matrix = np.vstack([new_point - all_one_point, other_point -
                    all_one_point])
68              if np.linalg.matrix_rank(matrix) < 2:
69                  new_point = Q.T @ (np.random.uniform(CLOSE_TO_ZERO, 1.5,
                        number_of_assets) * length_multiplier)
70                  new_point += CLOSE_TO_ZERO
71                  break
72          while not np.all(new_point > 0):
73              new_point = Q.T @ (np.random.uniform(CLOSE_TO_ZERO, 1.5,
                    number_of_assets) * length_multiplier)
74              new_point += CLOSE_TO_ZERO
75          point_list.append(new_point.tolist())
76          points_as_arrays.append(new_point)
77
78      final_point_list = deepcopy(point_list)
79      for point in points_as_arrays:
80          mirror_point = 2 * all_one_point - point
81          min_factor = np.max(np.clip((0 - point) / (mirror_point - point),
                0, 1))
82          max_factor = np.min(np.clip((np.inf - point) / (mirror_point -
                point), 0, 1))
83          factor = random.uniform(max(min_factor, 0.5), max_factor)
84          new_point = point + factor * (mirror_point - point)
85          final_point_list.append(new_point.tolist())
86
87      return final_point_list
```

```
88
89
90  def get_extreme(point_list: List[List[float]], riskless_interest_rate:
        Union[int, float]) -> List[List[float]]:
91      positive_stock_matrix = [[point[i] for point in point_list] for i in
            range(len(point_list[0]))]
92      all_one_row = [[1 for _ in range(len(positive_stock_matrix[0]))]]
93      negative_stock_matrix = [[-point[i] for point in point_list] for i in
             range(len(point_list[0]))]
94      all_minus_one_row = [[-1 for _ in range(len(positive_stock_matrix[0])
            )]]
95      non_negativity_constraints = (-np.eye(len(point_list))).tolist()
96      stock_matrix = (positive_stock_matrix + all_one_row +
            negative_stock_matrix +
97                      all_minus_one_row + non_negativity_constraints)
98
99      stock_matrix = np.array(stock_matrix)
100     b = ([1 + riskless_interest_rate for _ in range(len(
            positive_stock_matrix))] + [1] +
101           [-1 - riskless_interest_rate for _ in range(len(
                positive_stock_matrix))] + [-1] +
102           [0 for _ in range(len(point_list))])
103
104     b = np.array(b)
105     vertices = compute_polytope_vertices(stock_matrix, b)
106     vertices = [list(map(float, vertex)) for vertex in vertices]
107     return vertices
108
109
110 def generate_probability_measure(dimension: int) -> List[float]:
111     measure = list()
112     for _ in range(dimension - 1):
113         measure.append(random.random())
114     measure.sort()
115     measure.insert(0, 0)
116     measure.append(1)
117     measure = [measure[i + 1] - measure[i] for i in range(dimension)]
118     random.shuffle(measure)
119     return measure
120
121
122 def calculate_radon_nikodym(extreme_measures: List[List[float]],
        original_measure: List[float],
123                             bond_value: float) -> List[List[float]]:
124     radon_nikodym_list = list()
125     for extreme_measure in extreme_measures:
126         radon_nikodym = [x / y for x, y in zip(extreme_measure,
                original_measure)]
127         radon_nikodym = [r / bond_value for r in radon_nikodym]
128         radon_nikodym_list.append(radon_nikodym)
129
130     return radon_nikodym_list
131
```

```
132
133 def create_budget_equations_log(lambdas: List[float], measure: List[float
    ], radon_nikodym_list: List[List[float]],
134                                 k: int, riskless_interest_rate: float,
                                        starting_value: float, gamma: float)
                                        -> float:
135     return (sum(measure[j] * (1 / sum(lambdas[i] * radon_nikodym_list[i][
        j] for i in range(len(lambdas))))
136                 * radon_nikodym_list[k][j] for j in range(len(measure)))
                    - starting_value - riskless_interest_rate)
137
138
139 def create_budget_equations_power(lambdas: List[float], measure: List[
    float], radon_nikodym_list: List[List[float]],
140                                   k: int, riskless_interest_rate: float,
                                          starting_value: float, gamma: float)
                                           -> float:
141     return (sum(measure[j] * (sum(lambdas[i] * radon_nikodym_list[i][j]
        for i in range(len(lambdas)))) **
142                 (1 / (gamma - 1))
143                 * radon_nikodym_list[k][j] for j in range(len(measure)))
                    - starting_value - riskless_interest_rate)
144
145
146 def create_budget_equations(utility_function: str, radon_nikodym_list:
    List[List[float]],
147                             measure: List[float], riskless_interest_rate:
                                    float,
148                             starting_value: float = 1, gamma: float = -1)
                                    -> Callable:
149     if utility_function == 'log':
150         equation_creator = create_budget_equations_log
151     elif utility_function == 'power':
152         equation_creator = create_budget_equations_power
153     else:
154         raise ValueError(
155             f'Unknown utility function {utility_function}, it has to be
                    one of {IMPLEMENTED_UTILITY_FUNCTIONS}')
156
157     def equations(lambdas, measure, radon_nikodym_list, starting_value):
158         return [equation_creator(lambdas, measure, radon_nikodym_list, k,
                riskless_interest_rate, starting_value, gamma)
159                 for k in range(len(radon_nikodym_list))]
160
161     return equations
162
163
164 def solve_budget_equations(radon_nikodym_list: List[List[float]],
165                            measure: List[float], budget_equations: Callable,
166                            starting_value: float = 1) -> List[float]:
167     initial_guess = np.array([1.0 for _ in range(len(radon_nikodym_list))
        ])
```

```
168     bounds = [(CLOSE_TO_ZERO, None) for _ in range(len(radon_nikodym_list
            ))]
169     solution = minimize(lambda x: np.sum(np.abs(budget_equations(x,
            measure, radon_nikodym_list, starting_value))),
170                         initial_guess, bounds=bounds)
171     return solution.x.tolist()
172
173
174 def get_optimal_value(lambdas: List[float], utility_function: str,
175                       radon_nikodym_list: List[List[float]], gamma: float
                              = -1) -> List[float]:
176     new_radon_nikodym_list = [[radon_nikodym_list[i][j] * lambdas[i] for
            j in range(len(radon_nikodym_list[0]))] for i
177                               in range(len(radon_nikodym_list))]
178     inner_sum = [sum(sublist[i] for sublist in new_radon_nikodym_list)
            for i in range(len(new_radon_nikodym_list[0]))]
179     if utility_function == 'log':
180         return [1 / x for x in inner_sum]
181     elif utility_function == 'power':
182         return [x ** (1 / (1 - gamma)) for x in inner_sum]
183     else:
184         raise ValueError(
185             f'Unknown utility function {utility_function}, it has to be
                    one of {IMPLEMENTED_UTILITY_FUNCTIONS}')
186
187
188 def optimal_value_calculator(number_of_assets: int, half_of_events: int,
189                              riskless_interest_rate: Union[int, float] =
                                     0, utility_function: str = 'log',
190                              bond_value: Union[int, float] = 1, gamma:
                                     float = -1, starting_value: float = 1,
191                              factor_modell: bool = False) -> Union[str,
                                     List[float]]:
192     for _ in range(MAX_ITERATIONS):
193         if factor_modell:
194             stocks_and_movements = generate_points_factor(
                    number_of_assets)
195         else:
196             stocks_and_movements = generate_points(number_of_assets,
                    half_of_events)
197         extreme_measures = get_extreme(stocks_and_movements,
                riskless_interest_rate)
198         if len(extreme_measures) != 0:
199             if factor_modell:
200                 original_measure = generate_probability_measure(2 ** (
                        number_of_assets + 1))
201             else:
202                 original_measure = generate_probability_measure(
                        half_of_events * 2)
203             radon_nikodym_list = calculate_radon_nikodym(extreme_measures
                    , original_measure, bond_value)
204             measure = generate_probability_measure(2 * half_of_events)
```

```
205             budget_equations = create_budget_equations(utility_function,
                    radon_nikodym_list, measure,
206                                                        riskless_interest_rate
                                                            ,
                                                            starting_value,
                                                             gamma)
207             lambdas = solve_budget_equations(radon_nikodym_list, measure,
                     budget_equations, starting_value)
208             optimal_value = get_optimal_value(lambdas, utility_function,
                    radon_nikodym_list, gamma)
209             return optimal_value
210
211     return 'No extreme measures found after reaching the maximum number
            of iterations'
```

Optimális stratégiák kiszámolása

```
1  import math
2  from typing import Tuple, Callable, List
3  import matplotlib.pyplot as plt
4  import numpy as np
5  from scipy.integrate import quad
6  from scipy.optimize import minimize
7
8  IMPLEMENTED_UTILITY_FUNCTIONS = ['exponential_identity', 'power_sqrt', '
       power_square', 'beta_gamma', 'exponential']
9  IMPLEMENTED_DISTRIBUTIONS = ['uniform', 'normal', 'plus_minus_one']
10 EULER_CONSTANT = math.e
11
12
13 # power_identity: u(x) = x, if x<0, u(x) = 1-e^{x}, if x>=0
14 # power_sqrt: u(x) = -sqrt(|x|), if x<0, u(x) = 1-e^{x}, if x>=0
15 # power_square: u(x) = -x^2 if x<0, u(x) = 1-e^{x}, if x>=0
16 # beta_gamma: u(x) = 1-|x|^gamma, if x<0, u(x) = (1+x)^beta, if x>=0
17 # exponential: u(x) = -e^{-x}
18
19 def calculate_optimal_strategy(number_of_points: int, utility_type: str,
       distribution: str,
20                                bound_for_price_change: Tuple[float, float
                                       ] = (-5, 10),
21                                bounds_for_points: Tuple[float, float] =
                                       (-3, 3), bound_for_phi: float = 100,
22                                normal_parameters: Tuple[float, float] =
                                       (1, 1),
23                                prob_plus_minus_one: float = 0.3, beta:
                                       float = 0.15, gamma: float = 0.9) ->
                                       List[dict]:
24     assert utility_type in IMPLEMENTED_UTILITY_FUNCTIONS, \
25         f'Unknown utility function {utility_type}, it has to be one of {
               IMPLEMENTED_UTILITY_FUNCTIONS}'
26
27     assert distribution in IMPLEMENTED_DISTRIBUTIONS, \
```

```
28          f'Unknown stock change distribution {distribution}, it has to be
                one of {IMPLEMENTED_DISTRIBUTIONS}'
29
30      assert 0 <= prob_plus_minus_one <= 1, ('The probability for the
            plus_minus_one distribution'
31                                             ' has to be between 0 and 1')
32
33      lower_point_bound, upper_point_bound = bounds_for_points
34      assessment_points = np.linspace(lower_point_bound, upper_point_bound,
             number_of_points).tolist()
35      assessment_points = [p for p in assessment_points]
36      phi_star_dict = dict()
37      optimized_utility_dict = dict()
38      original_utility_dict = dict()
39      initial_guess = 1.0
40      utility_function = choose_utility_function(utility_type)
41
42      for i, point in enumerate(assessment_points):
43          f = create_function(utility_function, distribution,
                bound_for_price_change, point, normal_parameters,
44                              prob_plus_minus_one, beta, gamma)
45          result = minimize(lambda x: -f(x), initial_guess, bounds=[(-
                bound_for_phi, bound_for_phi)])
46          phi_star = result.x[0]
47          phi_star_dict[point] = phi_star
48          optimized_utility_dict[point] = f(phi_star)
49          initial_guess = phi_star
50          original_utility_dict[point] = utility_function(point, beta,
                gamma)
51      print(f'Result for {distribution} distribution with {utility_type}
            utility function is calculated')
52
53      return [phi_star_dict, optimized_utility_dict, original_utility_dict]
54
55
56  def exponential_identity(y: float, beta: float, gamma: float) -> float:
57      return y if y < 0 else (1 - np.exp(-y))
58
59
60  def power_sqrt(y: float, beta: float, gamma: float) -> float:
61      return -math.sqrt(abs(y)) if y < 0 else (1 - np.exp(-y))
62
63
64  def beta_gamma(y: float, beta: float, gamma: float) -> float:
65      return (1 - abs(y) ** gamma) if y < 0 else ((1 + y) ** (beta))
66
67
68  def exponential(y: float, beta: float, gamma: float) -> float:
69      return -np.exp(-y)
70
71
72  def power_square(y: float, beta: float, gamma: float) -> float:
73      return -y ** 2 if y < 0 else (1 - np.exp(-y))
```

```
74
75
76 def choose_utility_function(utility_type: str) -> Callable:
77     if utility_type == 'exponential_identity':
78         utility_func = exponential_identity
79     elif utility_type == 'power_sqrt':
80         utility_func = power_sqrt
81     elif utility_type == 'beta_gamma':
82         utility_func = beta_gamma
83     elif utility_type == 'exponential':
84         utility_func = exponential
85     elif utility_type == 'power_square':
86         utility_func = power_square
87     else:
88         raise ValueError(
89             f'Unknown utility function {utility_type}, it has to be one
                of {IMPLEMENTED_UTILITY_FUNCTIONS}')
90     return utility_func
91
92
93 def create_function(utility_func: Callable, distribution: str, bounds:
    Tuple[float, float], x: float,
94                     normal_parameters: Tuple[float, float], p: float,
                        beta: float, gamma: float) -> Callable:
95     lower_bound, upper_bound = bounds
96
97     if distribution == 'uniform':
98
99         def integrand(y, phi):
100             term = utility_func(x + phi * y, beta, gamma) / abs(
                    upper_bound - lower_bound)
101             return term
102
103         return lambda phi: quad(integrand, lower_bound, upper_bound, args
                =(phi))[0]
104
105     elif distribution == 'normal':
106         lower_bound = -upper_bound
107         m, sigma = normal_parameters
108
109         def integrand(y, phi):
110             term = utility_func(x + phi * y, beta, gamma) * 1 / (sigma *
                    np.sqrt(2 * np.pi)) * np.exp(
111                 -0.5 * ((y - m) / sigma) ** 2)
112             return term
113
114         return lambda phi: quad(integrand, lower_bound, upper_bound, args
                =(phi))[0]
115
116     elif distribution == 'plus_minus_one':
117         return lambda phi: utility_func(x + phi, beta, gamma) * p +
                utility_func(x - phi, beta, gamma) * (1 - p)
118     else:
```

```
119         raise ValueError(f'Unknown distribution {distribution}, it has to
                be one of {IMPLEMENTED_DISTRIBUTIONS}')
120
121
122 def plot_phi_star_single(phi_star_dict: dict):
123     x_values = list(phi_star_dict.keys())
124     y_values = list(phi_star_dict.values())
125
126     plt.figure()
127     plt.plot(x_values, y_values)
128     plt.xlabel('X')
129     plt.ylabel('Phi')
130     plt.show()
131
132
133 def plot_phi_star_or_exp_value(utility_type: str, number_of_points: int =
        1000, type: str = 'opt_util'):
134     uniform_phi, uniform_opt_util, _ = calculate_optimal_strategy(
            number_of_points, utility_type, 'uniform')
135     normal_phi, normal_opt_util, _ = calculate_optimal_strategy(
            number_of_points, utility_type, 'normal')
136     plus_minus_one_phi, plus_minus_one_opt_util, _ =
            calculate_optimal_strategy(number_of_points, utility_type, '
            plus_minus_one')
137
138     if type == 'phi_star':
139         uniform, normal, plus_minus_one = uniform_phi, normal_phi,
                plus_minus_one_phi
140     else:
141         uniform, normal, plus_minus_one = uniform_opt_util,
                normal_opt_util, plus_minus_one_opt_util
142
143     x_values_uniform = list(uniform.keys())
144     y_values_uniform = list(uniform.values())
145
146     x_values_normal = list(normal.keys())
147     y_values_normal = list(normal.values())
148
149     x_values_plus_minus_one = list(plus_minus_one.keys())
150     y_values_plus_minus_one = list(plus_minus_one.values())
151
152     utility_name, _ = get_plot_names(utility_type)
153     utility_name = utility_name.lower()
154
155     plt.figure()
156     plt.plot(x_values_uniform, y_values_uniform, label='Uniform')
157     plt.plot(x_values_normal, y_values_normal, label='Normal')
158     plt.plot(x_values_plus_minus_one, y_values_plus_minus_one, label='
            Plus minus one')
159     plt.xlabel('X')
160     if type == 'phi_star':
161         plt.ylabel('phi')
```

```
162             plt.title(f'A phi* ertekei {utility_name} hasznossagi fuggveny
                    eseten')
163         else:
164             plt.ylabel('Expected value')
165             plt.title('Expected value for different x values')
166         plt.legend()
167         plt.show()
168 
169 
170 def plot_exp_values_multiple_utility_func(exp_value_exponential_identity:
        dict, exp_value_power_sqrt: dict,
171                                           distribution: str):
172         for exp_value, utility_name in zip([exp_value_power_identity,
                exp_value_power_sqrt], ['Exponential identity', 'Power square root
                '']):
173             x_values = list(exp_value.keys())
174             y_values = list(exp_value.values())
175             plt.plot(x_values, y_values, label=f'{utility_name}')
176 
177         plt.xlabel('X')
178         plt.ylabel('Portfolio erteke')
179         plt.title(f'Portfolio erteke kulonbozo kezdeti ertekekre {
                distribution} eloszlas eseten')
180         plt.legend()
181         plt.show()
182 
183 
184 def plot_original_vs_optimized_utility(bounds_for_points: Tuple[float,
        float], utility_type: str,
185                                        distribution: str, beta: float =
                                               0.5, gamma: float = 1.0,
186                                        number_of_points: int = 1000):
187         _, optimized_utility, original_utility = calculate_optimal_strategy(
                bounds_for_points=bounds_for_points,     utility_type=utility_type
                , beta=beta, gamma=gamma, number_of_points=number_of_points,
                distribution=distribution)
188 
189         x_values_optimized = list(optimized_utility.keys())
190         y_values_optimized = list(optimized_utility.values())
191 
192         x_values_original = list(original_utility.keys())
193         y_values_original = list(original_utility.values())
194 
195         utility_name, distribution_name = get_plot_names(utility_type,
                distribution)
196 
197         plt.figure()
198         plt.plot(x_values_optimized, y_values_optimized, label='Optimalizalt
                hasznossagi fuggveny')
199         plt.plot(x_values_original, y_values_original, label='Eredeti
                hasznossagi fuggveny')
200         plt.xlabel('X')
201         plt.ylabel('Portfolio erteke')
```

```
202     plt.title(f'{utility_name} hasznossagi fuggveny {distribution_name}
            eloszlas eseten')
203     plt.legend()
204     plt.show()
205
206
207 def plot_single_utility_func(bounds_for_points: Tuple[float, float],
        utility_type: str, distribution: str, beta: float = 0.5, gamma: float
        = 1.0,  number_of_points: int = 1000):
208
209     _, optimized_utility, _ = calculate_optimal_strategy(
            bounds_for_points=bounds_for_points, utility_type=utility_type,
            beta=beta, gamma=gamma, number_of_points=number_of_points,
            distribution=distribution)
210
211     x_values_optimized = list(optimized_utility.keys())
212     y_values_optimized = list(optimized_utility.values())
213
214     plt.figure()
215     plt.plot(x_values_optimized, y_values_optimized, label='Optimalizalt
            hasznossagi fuggveny')
216     plt.xlabel('X')
217     plt.ylabel('Portfolio erteke')
218     plt.show()
219
220
221 def get_plot_names(utility: str = 'exponential_identity', distribution:
        str = 'uniform'):
222     if utility == 'exponential_identity':
223         utility_name = 'Exponential-identity'
224     elif utility == 'power_sqrt':
225         utility_name = 'Power-sqrt'
226     elif utility == 'power_square':
227         utility_name = 'Power-square'
228     elif utility == 'beta_gamma':
229         utility_name = 'Beta-gamma'
230     elif utility == 'exponential':
231         utility_name = 'Exponencialis'
232     else:
233         raise ValueError(f'Unknown utility function {utility}, it has to
                be one of {IMPLEMENTED_UTILITY_FUNCTIONS}')
234
235     if distribution == 'uniform':
236         distribution_name = 'egyenletes'
237     elif distribution == 'normal':
238         distribution_name = 'normalis'
239     elif distribution == 'plus_minus_one':
240         distribution_name = 'plusz-minusz egy'
241     else:
242         raise ValueError(f'Unknown distribution {distribution}, it has to
                be one of {IMPLEMENTED_DISTRIBUTIONS}')
243
244     return utility_name, distribution_name
```