UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

APLIKÁCIA HEURISTICKÝCH OPTIMALIZAČNÝCH ALGORITMOV NA KALIBRÁCIU MODELOV ÚROKOVÝCH MIER

DIPLOMOVÁ PRÁCA

Bratislava 2021 Bc. Patrícia Dianová

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

APLIKÁCIA HEURISTICKÝCH OPTIMALIZAČNÝCH ALGORITMOV NA KALIBRÁCIU MODELOV ÚROKOVÝCH MIER

DIPLOMOVÁ PRÁCA

Študijný program:	Ekonomická a finančná matematika
Študijný odbor:	1114 Aplikovaná matematika
Školiace pracovisko:	Katedra aplikovanej matematiky a štatistiky
Školiteľ:	doc. RNDr. Mgr. Beáta Stehlíková, PhD.

Bratislava 2021

Bc. Patrícia Dianová



Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky



ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Patrícia Dianová
Študijný program: ekonomicko-finančná matematika a modelovanie (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: matematika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Aplikácia heuristických optimalizačných algoritmov na kalibráciu modelov úrokových mier
Application of heuristic optimization algorithms to calibration of interest rate models

Anotácia: V práci sa použijú heuristické algoritmy z balíka [2] pre R na zvolené optimalizačné úlohy vznikajúce pri kalibrácii modelov úrokových mier. Vlastná implementácia 'grey wolf' algoritmu [1] sa použije na kalibráciu s dvoma kritériami, ktorá berie do úvahy priebeh krátkodobej úrokovej miery aj fitovanie výnosových kriviek. [1] Mirjalili, S., Saremi, S., Mirjalili, S. M., & Coelho,
L. D. S. (2016). Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization. Expert Systems with Applications, 47, 106-119.
[2] https://rdrr.io/cran/metaheuristicOpt/man/

Vedúci: doc. RNDr. Mgr. Beáta Stehlíková, PhD.
Katedra: FMFI.KAMŠ - Katedra aplikovanej matematiky a štatistiky
Vedúci katedry: prof. RNDr. Marek Fila, DrSc.

Dátum zadania: 17.12.2019

Dátum schválenia: 21.12.2019 prof. RNDr. Daniel Ševčovič, DrSc.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Rada by som sa touto cestou poďakovala svojej školiteľke doc. RNDr. Mgr. Beáte Stehlíkovej, PhD. za jej čas, cenné pripomienky a ochotný prístup počas celého vedenia tejto diplomovej práce.

Abstrakt v štátnom jazyku

DIANOVÁ, Patrícia: Aplikácia heuristických optimalizačných algoritmov na kalibráciu modelov úrokových mier [Diplomová práca], Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, Katedra aplikovanej matematiky a štatistiky; školiteľ: doc. RNDr. Mgr. Beáta Stehlíková, PhD., Bratislava, 2021, 63 s.

Témou tejto diplomovej práce je modelovanie úrokových mier pomocou stochastických modelov, konkrétne Vašíčkovým a Cox-Ingersoll-Rossovým modelom. Na optimalizačné úlohy vznikajúce pri kalibrácii modelov úrokových mier sme použili metaheuristické algoritmy inšpirované prírodou. Tieto algoritmy sme najskôr stručne popísali, otestovali na štyroch zvolených testovacích funkciách a vyhodnotili ich úspešnosť. Pri odhadovaní parametrov Vašíčkovým a Cox-Ingersoll-Rossovým modelom nám vyšiel ako najúspešnejší algoritmus diferenciálnej evolúcie DE, ktorému sme venovali detailnejšie aj samostatnú kapitolu. V predposlednej kapitole sa podrobne zaoberáme algoritmom šedých vlkov GWO a jeho rozšíreniu na viackriteriálnu optimalizáciu MOGWO. Záver práce je venovaný kalibrácii MOGWO algoritmu na reálne trhové dáta.

Kľúčové slová: metaheuristické optimalizačné algoritmy, Vašíčkov model, Cox-Ingersoll-Rossov model, metóda maximálnej vierohodnosti, odhad parametrov, viackriteriálna optimalizácia

Abstract

DIANOVÁ, Patrícia: Application of heuristic optimization algorithms to calibration of interest rate models [Diploma Thesis], Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics, Department of Applied Mathematics and Statistics; Supervisor: doc. RNDr. Mgr. Beáta Stehlíková, PhD., Bratislava, 2021, 63p.

The topic of our diploma thesis is the modeling of interest rates using stochastic models, specifically Vasicek model and Cox-Ingersoll-Ross model. We use nature-inspired metaheuristic algorithms for optimalization tasks that have arised during calibration of interest rate models. We first described these algorithms, tested them on four selected test functions and evaluated their success rate. During the estimation of parameters by Vasicek model and Cox-Ingersoll-Ross model, we have discovered that the most successful algorithm is DE algorithm to whichwe have devoted a separate chapter.. In the last but one chapter we deal with the GWO algorithm and its extension to multicriteria MOGWO optimization. The thesis is completed by real-data calibration of the MOGWO algorithm.

Keywords: metaheuristic optimization algorithms, Vasicek model, Cox-Ingersoll-Ross model, maximum likelyhood method, parameter estimation, multi-objective optimalization

Obsah

Úvod 9

Zoznam skratiek optimalizačných algoritmov 11

1. Metaheuristické algoritmy 12

1.1. Balík optimalizačných algoritmov 12

1.2. Testovacie funkcie 15

1.3. Výsledky testovania 18

2. Jednofaktorové modely 23

2.1. Vašíčkov model 24

2.2. Cox-Ingersoll-Rossov model 24

3. Kalibrácia jednofaktorových modelov 25

3.1. Výber dát 25

3.2. Kalibrácia Vašíčkovho modelu 26

3.3. Odhad parametrov Vašíčkovým modelom 27

3.4. Kalibrácia CIR modelu 30

3.5. Odhad parametrov CIR modelom 31

4. DE algoritmus 33

4.1. Etapy DE algoritmu 33

4.2. Funkcia DE v R 37

5. Viackriteriálna optimalizácia 37

5.1. GWO algoritmus 39

5.1.1. Inšpirácia GWO algoritmu 39

5.1.2. Matematický model GWO 40

5.1.3. Funkcia GWO v R 43

5.2. MOGWO algoritmus 43
5.3. Nová funkcia MOGWO v R 45
5.4. Testovanie novej funkcie MOGWO v R 45
6. Aplikácia algoritmu MOGWO na reálne dáta 48
Záver 54
Zoznam použitej literatúry 56
DODATOK A 60

Úvod

Optimalizácia je potrebná takmer všade. Využívame ju pri plánovaní dovolenky, v ekonomike či v inžinierstve. A pretože peniaze, čas a zdroje sú obmedzené, optimálne využitie dostupných zdrojov je mimoriadne dôležité. K známym príkladom patrí problém cestujúceho (angl. Travelling Salesman Problem), ktorý má práve raz navštíviť daný počet miest a to v čo najkratšom čase tak, aby sa minimalizovala celková prejdená vzdialenosť a celkové náklady na túto cestu [36].

Kvôli výpočtovej zložitosti sú na riešenia takýchto problémov potrebné dobré metaheuristické techniky. Cieľom skupiny metaheuristických algoritmov je čo najefektívnejšie prehľadávať priestor a nájsť tak globálne optimálne riešenie. Väčšina metaheuristických algoritmov z balíka [19] je inšpirovaná prírodou alebo biológiou, boli teda vyvinuté na základe úspešných evolučných prírodných systémov.

Existuje síce veľké množstvo optimalizačných algoritmov, žiaľ, neexistuje však žiaden univerzálny s dostatočne dobrými výsledkami pre všetky typy problémov. Inými slovami, univerzálna optimalizačná stratégia neexistuje a jediný spôsob ako môže jedna stratégia prekonať inú je, že sa špecializuje na daný typ problému. Táto myšlienka sa v optimalizácii nazýva „no free lunch“ [36]. Preto je voľba správneho algoritmu dôležitou úlohou v optimalizácii.

V prvej kapitole si najskôr predstavíme balík optimalizačných algoritmov [19]. Následne popíšeme štyri testovacie funkcie, na ktorých daný balík otestujeme, aby sme zistili ako sa algoritmy správajú pri zmenách svojich parametrov. Zo získaných informácií vyhodnotíme úspešnosť jednotlivých algoritmov.

Modely úrokovej miery popisujú vývoj úrokovej miery a ich cieľom je predpovedať vývoj aj do budúcna. Využívajú sa hlavne na oceňovanie finančných derivátov. Druhá kapitola sa zaoberá jednofaktorovými modelmi úrokovej miery, konkrétne Vašíčkovím a Cox-Ingersoll-Rossovým modelom [3], [32] . Neskôr na vzniknuté optimalizačné úlohy pri kalibrácii modelov úrokových mier použijeme balík algoritmov z predchádzajúcej kapitoly.

V tretej kapitole sa budeme venovať spomínanej kalibrácii jednofaktorových modelov. Najskôr si popíšeme jednu z najčastejšie používaných metód na odhad parametrov, a síce metódu maximálnej vierohodnosti [32] Pre metódu maximálnej vierohodnosti je dôležité poznať hustotu rozdelenia, z ktorého dáta pochádzajú a keďže hustota Vašíčkovho modelu sa riadi normálnym rozdelením a hustotu normálneho rozdelenia poznáme, vieme na odhad

parametrov Vašíčkovho modelu odvodiť aj presné vzorce [14] vďaka čomu si budeme môcť overiť úspešnosť získaných výsledkov. Hustota Cox-Ingersoll-Rossovho modelu sa riadi zložitejším necentrálnym χ^2- kvadrát rozdelením [16]. Pri denných dátach bude hustota Cox-Ingersoll-Rossovho modelu podobná hustote z Vašíčka, a preto by na jej optimalizáciu mohol byť vhodný taký algoritmus, ktorý dobre optimalizuje Vašíčka. Následne aplikujeme navrhnuté metódy kalibrácie jednofaktorového Vašíčkovho modelu a Cox-Ingersoll-Rossovho modelu na reálne dáta a odprezentujeme získané výsledky.

Štvrtá časť práce sa zaoberá podrobnejším popisom algoritmu diferenciálnej evolúcie DE [5], ktorý nám z predchádzajúcich odhadov parametrov úrokových mier vyšiel ako najúspešnejší.

Piata kapitola bude venovaná viackriteriálnej optimalizácii, ktorej snahou je nájsť nie jedno riešenie, ale celú množinu Pareto optimálnych riešení. Naformulujeme si teda problém viackriteriálnej optimalizácie a uvedieme si definície o Pareto optimálnych riešeniach [25] . Následne si podrobne predstavíme algoritmus šedých vlkov GWO [24] a potom si pomocou určitých modifikácií ukážeme jeho rozšírenie na viackriteriálnu optimalizáciu, výsledkom ktorej je viacúčelový algoritmus šedých vlkov MOGWO [25] . Pre tento algoritmus MOGWO vytvoríme aj novú funkciu v R, ktorá sa nachádza aj v elektronickej prílohe.

V záverečnej kapitole sa zameriame na vlastnú implementáciu MOGWO algoritmu. Na reálne trhové dáta aplikujeme MOGWO algoritmus viackriteriálnej optimalizácie, ktorá bude brať do úvahy nielen priebeh krátkodobej úrokovej miery, ale aj fitovanie výnosových kriviek [32] a získané výsledky odprezentujeme.

Zoznam skratiek optimalizačných algoritmov

Skratka	Algoritmus	
ABC	Artificial Bee Colony Algorithm	[12]
ALO	Ant Lion Optimizer	[28]
BA	Bat Algorithm	[34]
BHO	Black Hole Optimization Algorithm	[13]
CLONALG	Clonal Selection Algorithm	[2]
CS	Cuckoo Search algorithm	[37]
CSO	Cat Swarm Optimization Algorithm	[15]
DA	Dragonfly Algorithm	[27]
DE	Differential Evolution Algorithm	[5]
FFA	Firefly Algorithm	[35]
GA	Genetic Algorithm	[17]
GBS	Gravitational Based Search Algorithm	[23]
GOA	Grasshopper Optimisation Algorithm	[30]
GWO	Grey Wolf Optimizer	[24]
HS	Harmony Search Algorithm	[10]
KH	Krill-Herd Algorithm	[8]
MOGWO	Multi Objective Grey Wolf Optimizer	[25]
MFO	Moth Flame Optimizer	[21]
PSO	Prticle Swarm Optimization	[20]
SCA	Sine Cosine Algorithm	[22]
SFL	Shuffled Frog Leaping Algorithm	[6]
WOA	Whale Optimization Algorithm	[31]

1. Metaheuristické algoritmy

Metaheuristické algoritmy sa stávajú dôležitou súčasťou modernej optimalizácie. Aby sme mohli preveriť akýkoľvek optimalizačný algoritmus, musíme ho validovať oproti štandardným testovacím funkciám. V tejto časti najskôr stručne predstavíme balík optimalizačných algoritmov [19], následne popíšme štyri zvolené testovacie funkcie, ktorým sme pomocou daného balíka algoritmov hľadali optimálne riešenie a na záver kapitoly zosumarizujeme získané výsledky z testovania.

1.1. Balík optimalizačných algoritmov

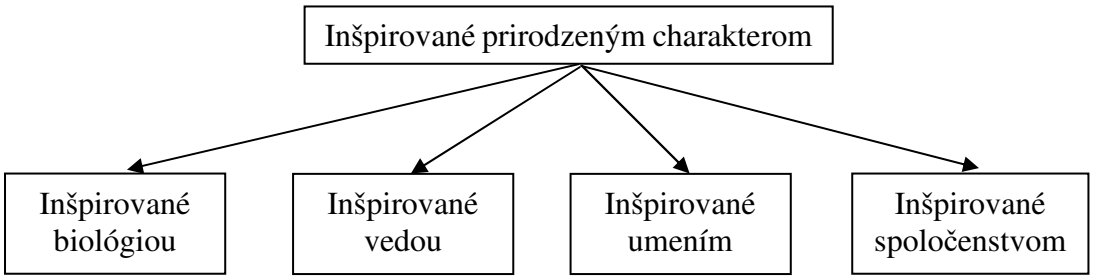
V metaheuristických algoritmoch „meta“ znamená „nad“ alebo „vyššiu úroveň“. Spravidla dosahujú lepšie výsledky ako jednoduchá heuristika. Heuristika znamená „nájsť“ alebo „objaviť pomocou pokusu a omylu“. Napriek popularite metaheuristiky neexistuje v literatúre dohodnutá definícia heuristiky a metaheuristiky a z toho dôvodu ich niektorí vedci zameniteľne používajú. V poslednej dobe sa však stalo akýmsi trendom pomenovávať všetky stochastické algoritmy s náhodnosťou ako metaheuristické. V tejto časti čerpáme najmä z [9].

V našej práci sa budeme zaoberať 21 optimalizačnými algoritmami z balíka [19], ktorý je implementovaný v štatistickom programe R. Predpis jednotlivých algoritmov nájdeme v zdrojových kódoch v dodatku A. Na používanie jednotlivých algoritmov si najskôr musíme načítať knižnicu metaheuristicOpt:

```
> library(metaheuristicOpt).
```

Všetky metaheuristické algoritmy využívajú určitý kompromis lokálneho a globálneho vyhľadávania. Rôznorodosť v riešeniach sa zabezpečuje náhodným výberom. A práve náhodný výber zabezpečuje, že riešenie neuviazne v lokálnych optimách, ale prehľadáva na globálnej úrovni. Preto sú takmer všetky metaheuristické algoritmy vhodné na nelineárne modelovanie a globálnu optimalizáciu.

Popularitu a úspech metaheuristiky možno pripísať mnohým dôvodom. Hlavným z nich je, že tieto algoritmy boli vyvinuté napodobňovaním najúspešnejších biologických, fyzikálnych či chemických procesov v prírode. Podľa zdroja inšpirácie môžeme metaheuristické algoritmy klasifikovať do rôznych kategórií ako je znázornené na Obrázku 1.



Obrázok 1: Zdroj inšpirácie v metaheuristických optimalizačných algoritmoch [9]

Hlavnou kategóriou sú biológiou inšpirované algoritmy, ktoré vo všeobecnosti využívajú kolektívny vývoj alebo kolektívne správanie zvierat. Veda je ďalším zdrojom inšpirácie pre metaheuristiku, tieto algoritmy sú zvyčajne inšpirované fyzikou alebo chémiou. Existujú aj algoritmy inšpirované umením, zväčša napodobňujú umelcov pri vytváraní ich umeleckých diel, napríklad hudobníkov. Ďalej sem patria spoločenstvom inšpirované algoritmy, napríklad vlci majú vo svorke veľmi prísnu sociálno-dominantnú hierarchiu.

Napriek tomu, že sú rôzne zdroje inšpirácie pre metaheuristické optimalizačné algoritmy, všetky sú si podobné vo svojich štruktúrach. Preto ich možno klasifikovať do dvoch hlavných kategórií, a síce evolučné algoritmy a rojové algoritmy.

Evolučné algoritmy používajú vo všeobecnosti na riešenie problémov s optimalizáciou iteračný postup založený na pokroku biologickej evolúcie. Stručne si popíšeme niektoré z nich.

K evolučným algoritmom patrí napríklad genetický algoritmu GA (angl. Genetic Algorithm) [17] , [36], založený na princípoch genetiky a prírodného výberu. Využíva kríženie a rekombináciu, mutáciu a selekciu. V GA môže populácia súčasne prehľadávať priestor vo viacerých smeroch, pretože viac potomkov v populácii funguje ako nezávislí agenti. Genetické algoritmy majú oproti tradičným optimalizačným algoritmom mnoho výhod. Dve najvýraznejšie výhody sú schopnosť čeliť zložitým problémom a paralelnosť. Genetické algoritmy si dokážu poradiť s rôznymi typmi optimalizácie, či už je účelová funkcia stacionárna alebo nestacionárna, lineárna alebo nelineárna, spojitá alebo nespojitá.

Algoritmus hľadania harmónie HS (angl. Harmony Search Algorithm) [10] je inšpirovaný hudobným procesom hľadania „dokonalého stavu“ harmónie. Tak ako klasický optimalizačný algoritmus hľadá globálne optimum vyhodnotením účelovej funkcie, tak napríklad jazzová improvizácia hľadá harmóniu estetickým odhadom. Estetický odhad závisí od optimalizačných operátorov ako sú pamäť harmónie, veľkosť pamäte harmónie,

frekvencia harmonickej pamäte či rýchlosť nastavenia výšky tónu. Kvalita harmónie sa zvyšuje tréningom, rovnako ako sa zvyšuje kvalita riešenia pri klasických optimalizačných algoritmoch počtom iterácií. Úprava výšky tónu je potom podobná operácii mutácie v genetických algoritmoch.

Ďalej sem patrí algoritmus diferenciálnej evolúcie DE (angl. Differential evolution) [5]. Je to stochastický vyhľadávací algoritmus so samo organizujúcou sa tendenciou. Riešenia sú reprezentované v podobe vektorov, pomocou ktorých sa potom uskutočňuje mutácia a kríženie. Podrobnejšie sa tomuto algoritmu venujeme v kapitole 4.

Druhou kategóriou sú algoritmy založené na rojovej inteligencii využívajúce kolektívne správanie sa zvierat ako sú vtáky, hmyz, ryby a podobne. Opäť si stručne niektoré z nich popíšeme.

Algoritmus optimalizácie roja častíc PSO (angl. Particle Swarm Optimalization) [20] je založený na štúdii o kolektívnom správaní sa v samo organizovaných systémoch. V PSO sa do vyhľadávajúceho priestoru nejakého problému, resp. funkcie nasadí náhodne niekoľko častíc (spolu tvoria roj) a každá z nich vyhodnotí účelovú funkciu na svojom aktuálnom mieste. Častica sa potom iteratívne pohybuje po vyhľadávacom priestore a je priťahovaná svojou najlepšou pozíciou (lokálne optimum) a najlepšou pozíciou spomedzi susedov (globálne optimum).

Algoritmus umelých včelstiev ABC (angl. artificial bee colony) [12] je motivovaný inteligentným správaním sa včiel medonosných pri hľadaní potravy. V tomto algoritme sa poloha zdroja potravy považuje za kandidátske riešenie optimalizačného problému a vhodnosť riešenia predstavuje množstvo nektáru nachádzajúceho sa v zdroji potravy. Kolónia včiel v algoritme ABC sa skladá z robotníc, prieskumníčok a vyčkávajúcich včiel. Polovicu kolónie tvoria robotnice hľadajúce zdroje potravy a druhú polovicu tvoria vyčkávajúce včely. Za predpokladu, že množstvo nektáru nového zdroja je väčšie ako množstvo predchádzajúceho, robotnica si zapamätá novú polohu a zabudne na starú. V opačnom prípade si uchová pozíciu predchádzajúceho zdroja v pamäti. Keď všetky robotnice dokončia proces hľadania, zdieľajú nektárové informácie o zdrojoch a informácie o svojej polohe s vyčkávajúcimi včelami, ktoré dovtedy čakajú v úli. Robotnice opúšťajúce úplne využitý zdroj potravy sa zmenia na prieskumníčky. Prieskumníčky náhodne hľadajú nový zdroj potravy v okolí úľa a stanú sa z nich opäť robotnice.

Algoritmus svätojánskych mušiek FFA (angl. Firefly Algorithm) [35] je heuristický algoritmus inšpirovaný jasom svätojánskych mušiek/svetlušiek. Blikajúce svetlo je

produkované procesom bioluminiscencie. Svetlo slúži najmä na prilákanie partnera za účelom párenia sa a pri zháňaní koristi. Blikanie môže navyše slúžiť aj ako obranný mechanizmus. Algoritmus FFA predstavuje populačne založený proces s množstvom agentov (svetlušiek), ktorí riešia optimalizačný problém. FFA má 3 pravidlá: 1. všetky svetlušky majú jedno pohlavie, teda priťahujú sa navzájom bez ohľadu na pohlavie. 2. stupeň atraktivity je priamoúmerný jasu svetlušky. Pri ľubovoľných dvoch svetluškách sa tá menej jasná bude pohybovať smerom k svetlejšej. Navyše so zväčšujúcou sa vzdialenosťou sa zmenšuje jas svetlušky, teda aj jej príťažlivosť. Ak sa v blízkosti nenachádza žiadna žiarivejšia svetluška, pohybuje sa v priestore náhodne. 3. jas a intenzita svetla je určená hodnotou účelovej funkcie daného problému.

Optimalizačný algoritmus vyhľadávania kukučiek CS (angl. Cuckoo Search Algorithm) [37] je inšpirovaný správaním sa kukučiek v prírode. Kukučky sú parazity, ktoré kladú vajíčka do hniezda iných vtákov. Keď hostiteľský vták zistí, že vajcia nie sú jeho, buď ich vyhodí alebo opustí hniezdo a postaví si nové. Vo všeobecnosti platí, že cesta zvieraťa pri hľadaní potravy je v skutočnosti náhodná prechádzka, pretože nasledujúci pohyb je založený na súčasnom mieste a pravdepodobnosti prechodu na ďalšie miesto. Smer implicitne závisí od pravdepodobnosti, ktorú je možné matematicky modelovať. Rôzne štúdie preukázali, že letové správanie mnohých zvierat a hmyzu preukázalo typické vlastnosti Lévyho letu. Lévyho let je typ náhodnej prechádzky. Pre jednoduchosť sa v CS algoritme predpokladajú tri zidealizované pravidlá: 1. každá kukučka znáša po jedenom vajíčku (riešenie), ktoré ukladá do náhodne zvoleného hniezda. 2. hniezdo môže mať viacero vajíčok, pre zjednodušenie sa používa prípad, že každé hniezdo má iba jedno vajce (riešenie) a najlepšie riešenie (podľa hodnoty účelovej funkcie) sa prenesie do ďalšej iterácii. 3. hostiteľ/hniezdo môže nájdené horšie riešenia zahodiť a nahradiť ich novým riešením. Počet hostiteľských hniezd je konštantný a vajíčko znesené kukučkou je objavené hostiteľským vtákom s pravdepodobnosťou $p \in [0,1]$. V takom prípade môže hostiteľský vták buď vyhodiť vajíčko, alebo opustiť hniezdo a postaviť úplne nové.

Toto je len zopár vybraných algoritmov. Zoznam všetkých optimalizačných algoritmov z balíka [19], skratky pod ktorými ich budeme v našej práci používať a odkazy na ich referencie nájdeme v Zozname skratiek optimalizačných algoritmov.

1.2. Testovacie funkcie

Na hľadanie extrémov som si vybrala štyri testovacie funkcie. Tri z nich sú definované v dvojrozmernom priestore a jedna v n-dimenzionálnom priestore.

- **Rastrigin**

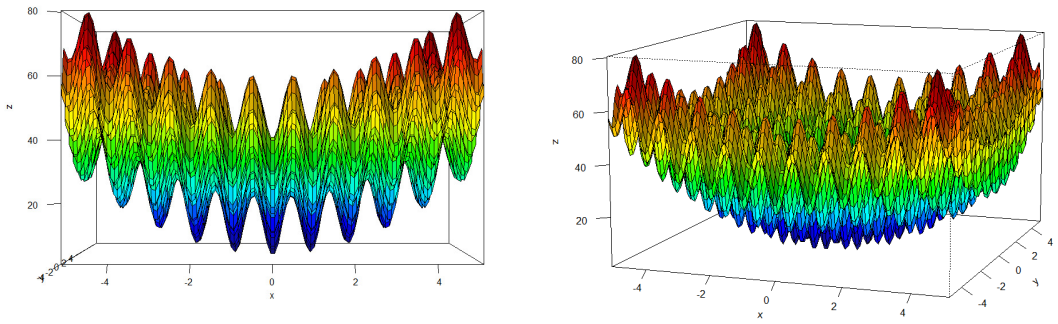
Často používanou testovacou funkciou je Rastriginova funkcia [29] definovaná v n-dimenzionálnom priestore. Jej charakteristickou vlastnosťou je multimodálnosť, čo znamená, že má viacero lokálnych miním, ktoré nadobúda vďaka kosínusovej modulácii. Tieto minimá sú pravidelne distribuované, čo môžeme vidieť aj na Obrázku 2. Rastriginova funkcia je spojitá, konvexná a diferencovateľná. Predpis funkcie je nasledovný:

$$f(x) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i)), \quad i = 1, \dots, n. \tag{1}$$

V našom prípade sme za n určujúce počet dimenzií zvolili 2. Zvolili sme taktiež obvyklý pozorovací priestor tejto funkcie, v našom prípade pre dve premenné to je štvorec

$$-5.12 \leq x_i \leq 5.12, \quad i = 1,2.$$

Globálne minimum má v bode $(x_1, x_2) = (0,0)$ a nadobúda v ňom hodnotu $f(x_1, x_2) = 0$.



Obrázok 2: Rastrigin funkcia

- **Beale**

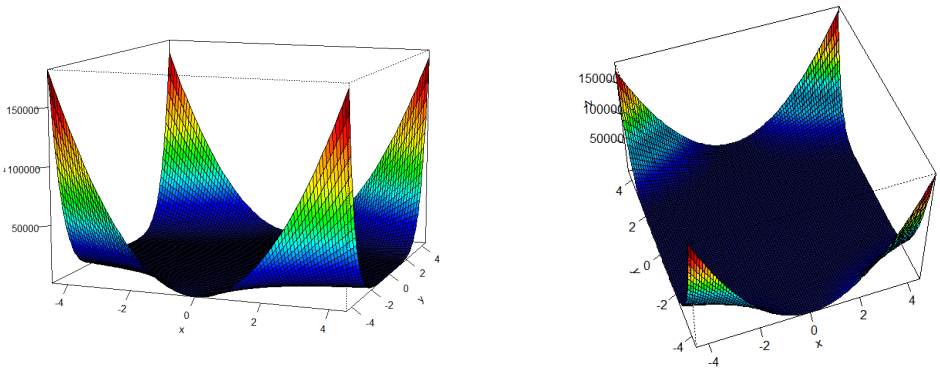
Ďalšou testovacou funkciou je takzvaná Beale funkcia [1], ktorá je spojitá, nekonvexná a definovaná v dvojrozmernom priestore. Je opäť multimodálna s ostrými vrcholmi v rohoch vstupnej oblasti, v ktorých niektoré nájdené riešenia uviazli. Takéto nájdené minimá na hranici sme potom filtrovali. Predpis funkcie je

$$f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2. \tag{2}$$

Je možné ju pozorovať na ľubovoľnej oblasti, no väčšinou sa pozoruje na štvorci

$$-4.5 \leq x, y \leq 4.5.$$

Funkciu sme si vybrali z dôvodu, že má jedno globálne minimum v bodoch $(x, y) = (3, 0.5)$ a nadobúda v ňom hodnotu $f(x, y) = 0$.



Obrázok 3: Beale funkcia

- **Matyas**

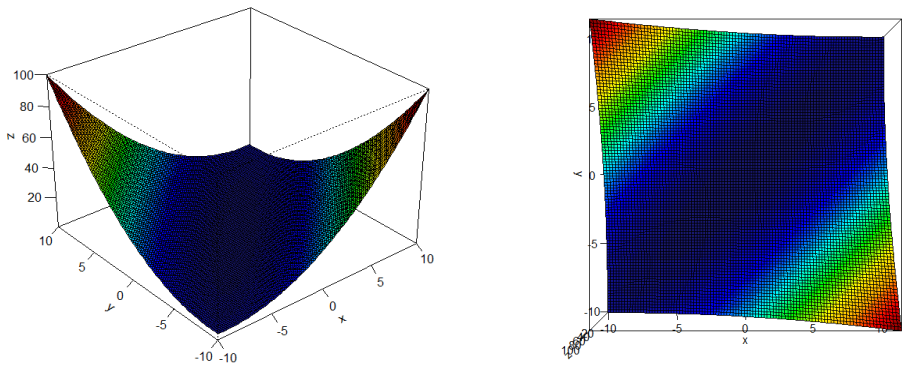
Predpis Matyas funkcie [18] je nasledovný

$$f(x,y) = 0.26(x^2 + y^2) - 0.48xy. \tag{3}$$

Funkcia je spojitá, konvexná, diferencovateľná a unimodálna. Je definovaná v dvojdimenzionálnom priestore. Obvykle je pozorovaná na oblasti

$$-10 \leq x, y \leq 10.$$

Nemá žiadne lokálne minimum. Má však jedno globálne minimum v bodoch $(x,y) = (0,0)$, kde nadobúda funkčnú hodnotu $f(x,y) = 0$.



Obrázok 4: Matyas funkcia

- **Tree-Hump Camel**

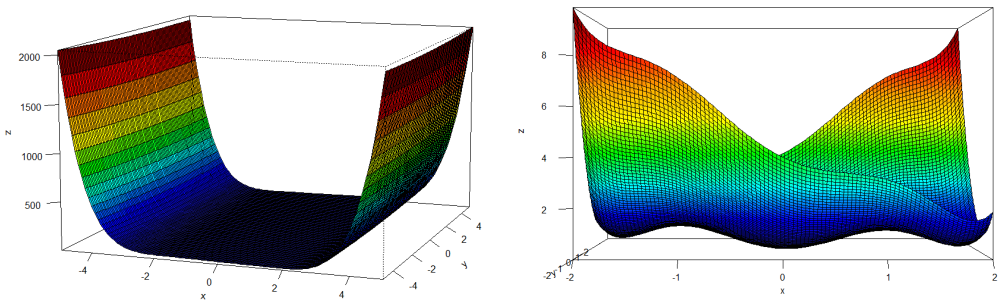
Táto funkcia je spojitá, diferencovateľná, nekonvexná a definovaná v dvojrozmernom priestore. Predpis Three-Hump Camel funkcie [33] je nasledovný

$$f(x,y) = 2x^2 - 1.05x^4 + \frac{x^6}{6} + xy + y^2. \tag{4}$$

Funkcia je definovaná je na ľubovoľnej vstupnej oblasti, no väčšinou sa pozoruje na štvorci

$$-5 \leq x, y \leq 5.$$

Funkcia má tri lokálne minimá, ktoré sú vidieť, len ak funkciu zobrazíme na menšej oblasti a jedno globálne minimum v bodoch $(x, y) = (0, 0)$, v ktorom nadobúda hodnotu $f(x, y) = 0$.



Obrázok 5: Three-Hump Camel funkcia, vľavo zobrazená na väčšej pozorovacej oblasti a vpravo na menšej, kde sú vidieť lokálne minimá

1.3. Výsledky testovania

Cieľom experimentov na balíku algoritmov [19] pomocou štyroch zvolených testovacích funkcií bolo nájsť také parametre, pre ktoré sa výsledok blíži čo najviac ku globálnemu minimu a zároveň takýto výpočet netrvá príliš dlho. Algoritmy sa medzi sebou líšia nielen v inakosti parametrov, ale aj v ich počte. Z tohto dôvodu je náročné porovnať parametre algoritmov medzi sebou. Dva parametre však majú všetky algoritmy spoločné, a síce počet jedincov v populácii *NumPop* a maximálny počet iterácií *MaxIter*, ktoré patria zároveň k rozhodujúcim vstupným parametrom. Ich vplyv sme pozorne sledovali pri všetkých experimentoch. Pre každý algoritmus sme spravili rôzne kombinácie jeho parametrov tak, aby sme zohľadnili čo najviac možných a efektívnych spôsobov nájdenia optimálneho riešenia. Pre každý algoritmus a pre každú kombináciu parametrov sme spustili testovanie 50-krát a ďalej sme pracovali s priemerom týchto hodnôt.

Najskôr sa bližšie pozrieme na desať algoritmov z balíka algoritmov [19], ktoré majú iba dva parametre, a síce *NumPop* a *MaxIter*. Parameter *NumPop* hovorí o počte jedincov v populácii. Menili sme mu hodnoty z množiny {6, 10, 20, 30, 40, 50, 60, 70, 100, 200, 400}, pričom prednastavená hodnota je 40. Parameter *MaxIter* predstavuje maximálny počet iterácií a volili sme ho z množiny {10, 50, 100, 200}. Tu je predvolená hodnota 500, ale takto vysokú hodnotu sme nevolili, a to z dôvodu časovej efektivity testovania.

Pre zjednodušenie sme si Rastriginovu funkciu označili ako F1, Beale funkciu ako F2, Matyas funkciu ako F3 a Three-Hump Camel funkciu ako F4. V Tabuľke 1 sme zobrazili výsledky testovania týchto algoritmov na troch testovacích funkciách, ktoré majú minimum v bode (0,0). V Tabuľke 1 sa v stĺpci riešenie sa nachádza najlepší priemerný výsledok a parametre, pri akých bol nájdený.

Algoritmus	Funkcia	Riešenie		NumPop	MaxIter
ABC	F1	4.32E-11	1.91E-11	100	200
	F3	-7.33E-04	6.03E-04	200	50
	F4	1.42E-08	4.61E-09	400	200
ALO	F1	4.60E-09	-8.83E-10	400	50
	F3	-2.47E-09	-7.24E-09	400	200
	F4	-5.15E-09	6.81E-10	400	50
WOA	F1	-9.06E-11	2.41E-11	400	200
	F3	-3.71E-88	-2.72E-88	400	200
	F4	-1.01E-81	1.30E-81	400	200
SFL	F1	-7.06E-11	6.27E-11	400	100
	F3	7.35E-52	7.13E-52	400	200
	F4	7.60E-133	-2.64E-132	400	200
SCA	F1	-3.02E-13	-9.29E-11	200	100
	F3	-3.27E-22	-2.98E-23	400	200
	F4	8.67E-25	1.26E-24	400	200
MFO	F1	-2.51E-12	8.37E-11	100	200
	F3	-9.70E-25	-1.27E-24	400	200
	F4	1.01E-35	-1.10E-35	400	200
GWO	F1	1.20E-11	-6.41E-12	200	50
	F3	4.34E-153	4.25E-153	400	200
	F4	-8.04E-164	9.06E-164	400	200
DA	F1	3.22E-07	-8.57E-07	200	200
	F3	-1.72E-07	-1.74E-07	200	200
	F4	-3.65E-08	2.86E-08	200	200
GOA	F1	4.24E-06	5.31E-06	200	200
	F3	3.69E-08	6.62E-08	70	200
	F4	-1.63E-08	8.75E-08	100	200
BHO	F1	-3.98E-02	8.68E-07	100	200
	F3	3.69E-08	6.62E-08	70	200
	F4	4.12E-07	-5.34E-07	100	200

Tabuľka 1: Najlepšie nájdené riešenia pre funkcie F1, F3 a F4

V Tabuľke 2 je zobrazených opäť tých istých desať algoritmov, tentokrát iba pre funkciu F2, ktorá má minimum v bode (3,0.5). Zobrazili sme tu navyše aj dva stĺpce s relatívnou odchýlkou najlepšieho priemerného riešenia od presného riešenia.

Alg.	F.	Riešenie		Relatívna odchýlka		Num Pop	Max Iter
ABC	F2	2.99976078465	0.50016834346	-7.97E-05	3.37E-04	400	100
ALO	F2	2.99999999824	0.50000000062	-2.61E-09	-2.92E-09	400	200
WOA	F2	3.00000002399	0.50000000801	8.00E-09	1.60E-08	400	200
SFL	F2	2.99999999999	0.49999999999	-1.01E-14	-1.80E-14	400	200
SCA	F2	2.99991566510	0.49991421473	-2.81E-05	-1.72E-04	50	200
MFO	F2	3.00000000001	0.50000000001	1.53E-13	2.22E-13	400	100
GWO	F2	2.99999540653	0.50000142452	-1.53E-06	2.85E-06	400	100
DA	F2	2.99999322649	0.49997533598	-2.26E-06	-4.93E-05	100	100
GOA	F2	2.99999745797	0.52745655167	-8.47E-07	5.21E-02	100	200
BHO	F2	3.01267403198	0.50274029261	4.21E-03	5.45E-03	200	50

Tabuľka 2: Najlepšie nájdené riešenia pre funkciu F2

Z testovania týchto desiatich algoritmov, ktoré majú iba dva parametre, sme zistili, že pri zvyšovaní hodnoty oboch parametrov súčasne sa zlepšuje aj odhad minima. V niektorých prípadoch sa s narastajúcimi hodnotami parametrov odhad minima ustálil a neviedol viac k presnejším výsledkom. Takéto správanie sme spozorovali napríklad pri algoritme DA. Okrem toho testovanie ukázalo, že algoritmy si lepšie poradili s odhadom minima funkcií F3 a F4 v porovnaní s funkciou F1 a F2. Môže to byť z dôvodu, že funkcia F1 vďaka kosínusovej modulácii má viacero lokálnych miním, s čím mohli mať algoritmy problém. Funkcia F2 má zasa minimum posunuté do bodu (3,0.5). Navyše niektoré riešenia uviazli na hranici, no ako sme spomínali na začiatku pri popise tejto funkcie F2, takéto nájdené riešenia sme filtrovali. Ak porovnáme úspešnosť algoritmov medzi sebou, takmer všetky odhadli minimum pri nejakých parametroch s vysokou presnosťou. K najmenej presným patril algoritmus BHO.

Všetky ostatné algoritmy majú navyše okrem parametrov *numPop* a *maxIter* aj nejaké ďalšie. V prípade algoritmov CLONALG, CS, GA a HS však stále tieto dva parametre zohrávajú najväčšiu rolu pri nájdení minima. Ostatné parametre na riešenie nemajú výraznejší vplyv. Najlepšie nájdené riešenia sú s presnosťou pohybujúcou sa okolo troch či štyroch desatinných miest, takže tieto algoritmy pri testovacích funkciách radíme k tým menej presným.

Algoritmus CSO má štyri parametre, a síce *numPop*, *maxIter*, *mixtureRatio* a *tracingConstant,* pričom jednoznačne najväčší vplyv na nájdenie optimálneho riešenia má parameter *mixtureRatio,* ktorý hovorí o pomere mačiek v režime prenasledovania a vyhľadávania. Nadobúda hodnoty z intervalu $(0,1)$ a platí, čím väčšia hodnota, tým väčší počet mačiek vo vyhľadávacom režime a tým presnejšie je nájdené riešenie.

Podobne je na tom PSO algoritmus, ktorý má taktiež štyri parametre *numPop*, *maxIter, Vmax* a *w*. Okrem týchto parametrov má ešte parametre *ci* a *cg* hovoriace o individuálnej a skupinovej kognitivite, ktorým sme však hodnoty nemenili a nechali sme im prednastavenú hodnotu 1.49445. Najväčší vplyv na presnosť nájdeného riešenia má parameter hmotnosti zotrvačnosti *w*. Jeho defaultná hodnota je 0.729, no testovanie preukázalo, že lepšie výsledky sa dosiahli pri nižších hodnotách tohto parametra. Začínali sme na hodnote 0.1 a čím sa táto hodnota zvyšovala, tým menej presné bolo nájdené riešenie.

Čo sa týka algoritmu DE, nevykazoval logické správanie sa pri zmenách hodnôt jeho parametrov. Nevieme teda určiť, ktorá kombinácia parametrov je tá najsprávnejšia a ktorým smerom sa uberať, ak chceme zlepšiť presnosť nájdeného riešenia.

Algoritmus BA zasa neurčil rovnaké správanie pri rovnako zvolených parametroch pre všetky funkcie F1-F4. BA algoritmus má 5 parametrov, sú to *numPop*, *maxIter, gama, alphaBA, min_Frequency a max_Frequency.* V prípade funkcie F1 sa nastavenie parametrov *min_Frequency/ max_Frequency* na malú hodnotu ukázalo ako prospešné pri zvýšení presnosti nájdeného minima. My sme zvolili najmenšiu hodnotu 0.1. Čo sa týka funkcie F2, tu nenastalo žiadne zlepšenie pri zmenách parametrov, pričom relatívne odchýlky od presného riešenia sa pohybujú okolo štyroch desatinných miest. Na druhú stranu, v prípade funkcií F3, F4 pozorujeme veľmi vysokú presnosť. Pri akejkoľvek voľbe parametrov algoritmus odhadol minimum s presnosťou až na 170 desatinných miest.

Zvyšné algoritmy GBS, FFA a KH sa ukázali ako časovo neefektívne. Pri výbere správneho algoritmu zohráva dôležitú úlohu nielen jeho presnosť, ale aj časová náročnosť. Preto sa nimi nebudeme v našej práci zaoberať.

V Tabuľkách 3 a 4 sú podobne ako v prípade prvých desiatich algoritmoch vypísané najlepšie priemerné riešenia, prípadne ich odchýlky. Nakoľko majú všetky tieto algoritmy iné parametre, tentokrát sme kvôli prehľadnosti nevypisovali pri akých parametroch boli získané.

Algoritmus	Funkcia	Riešenie	
BA	F1	2.35E-43	-8.55E-45
	F3	-2.99E-194	1.61E-194
	F4	1.39E-190	-4.26E-191
CLONALG	F1	6.41E-06	6.70E-06
	F3	5.27E-04	3.45E-04
	F4	6.20E-06	4.04E-06
CS	F1	-3.34E-04	3.32E-03
	F3	-1.49E-03	-6.91E-04
	F4	4.21E-04	-3.90E-03
DE	F1	8.33E-12	-4.41E-11
	F3	-3.11E-163	-6.38E-163
	F4	-1.74E-108	2.75E-108
CSO	F1	2.33E-12	6.87E-11
	F3	-1.09E-17	1.43E-19
	F4	-5.94E-22	7.60E-22
GA	F1	-8.54E-02	6.88E-03
	F3	3.64E-03	-7.19E-04
	F4	1.38E-03	-5.42E-03
HS	F1	-6.98E-06	-9.59E-06
	F3	-8.23E-06	-1.27E-05
	F4	-7.69E-06	-9.66E-07
PSO	F1	9.30E-11	-9.26E-11
	F3	-2.35E-33	-1.93E-33
	F4	2.15E-39	3.84E-40

Tabuľka 3: Najlepšie nájdené riešenia pre funkcie F1, F3 a F4

Alg.	F.	Riešenie		Relatívna	odchýlka
BA	F2	3.0002510974185400	0.5001618043646520	8.37E-05	3.24E-04
CLONALG	F2	3.0006620833603700	0.5001377447973940	2.21E-04	2.75E-04
CS	F2	3.0018316019699000	0.5008837680425500	6.10E-04	1.76E-03
DE	F2	3.0000000000000300	0.5000000000000160	1.01E-14	3.20E-14
CSO	F2	3.0024014753299000	3.0035788823531600	8.00E-04	8.34E-01
GA	F2	2.9869861011600000	0.4653861820200000	-4.36E-03	-7.44E-02
HS	F2	2.9869861011600000	0.4653861820200000	-4.36E-03	-7.44E-02
PSO	F2	3.0001115346332100	0.4998211474476850	3.72E-05	-3.58E-04

Tabuľka 4: Najlepšie nájdené riešenia pre funkcie F2

Týmto sme ukončili testovanie daného balíka algoritmov na štyroch testovacích funkciách. Ukázalo sa, že medzi parametre, ktoré najviac ovplyvňujú presnosť riešenia

patria najmä *numPop*, *maxIter.* V niektorých prípadoch to boli aj iné parametre, napríklad *min_Frequency/ max_Frequency* pri algoritme BA, *mixtureRatio* pri algoritme CSO alebo *w* pri algoritme PSO. Taktiež sme zistili, že dĺžka trvania výpočtu riešenia algoritmov GBS, FFA a KH je časovo neefektívna. Rozhodli sme sa preto nepoužívať tieto algoritmy pri zložitejšom Vašíčkovom a Cox-Ingersoll-Rossovom modeli.

2. Jednofaktorové modely

Ako sme už spomínali, metaheuristické optimalizačné algoritmy majú široké uplatnenie. V našej práci ich použijeme na optimalizačné úlohy vznikajúce pri kalibrácii modelov úrokových mier. V tejto kapitole si predstavíme jednofaktorové modely úrokových mier [32] , v ktorých sa predpokladá, že okamžitá úroková miera r je modelovaná pomocou stochastickej diferenciálnej rovnice, ktorá má vo všeobecnosti tvar

$$dr = \mu(r,t)dt + \sigma(r,t)dw. \tag{5}$$

Deterministický člen $\mu(r,t)$ určuje trend (drift) a veľkosť stochastického členu určuje volatilita $\sigma(r,t)$. Keďže v tomto modeli je prítomný iba jeden stochastický člen, hovoríme o jednofaktorovom modeli úrokovej miery r.

Veľkú triedu týchto modelov tvoria tzv. mean reversion modely. Tento pojem vyjadruje, že vývoj úrokovej miery r je priťahovaný k dlhodobej úrokovej miere. Čiže úroková miera sa približuje k tejto limitnej rovnovážnej hodnote. Toto vieme v modeli dosiahnuť voľbou driftovej funkcie $\mu(r,t)$ v tvare

$$\mu(r,t) = \kappa(\theta - r), \tag{6}$$

kde θ, $\kappa > 0$ sú kladné konštanty. Parameter θ predstavuje limitnú hodnotu úrokovej miery a κ predstavuje rýchlosť návratu k tejto limitnej úrokovej miere. Dokázať sa to dá cez obyčajnú diferenciálnu rovnicu pre strednú hodnotu $E(r_t)$

$$d\mathrm{E}(\mathrm{r_t}) = \mathrm{E}(\mathrm{dr_t}) = \kappa\big(\theta - \mathrm{E}(\mathrm{r_t})\big)\mathrm{dt} + \mathrm{E}(\sigma(r,t)dw) = \kappa\big(\theta - \mathrm{E}(\mathrm{r_t})\big)\mathrm{dt}.$$

Riešením tejto diferenciálnej rovnice je

$$E(r_t) = r_0 e^{-\kappa t} + \theta(1 - e^{-\kappa t}) \tag{7}$$

a limitne pre $t \to \infty$ dostávame vlastnosť mean reversion

$$\lim_{t\to\infty} E(r_t) = \theta. \tag{8}$$

V závislosti od tvaru funkcie volatility $\sigma(r,t)$ potom rozlišujeme jednotlivé modely úrokovej miery.

2.1. Vašíčkov model

Jedným z príkladov modelov typu mean reversion je Vašíčkov model, resp. vo všeobecnosti sa takémuto procesu hovorí Ornstein-Uhlenbeckov proces, pri ktorom uvažujeme volatilitu konštantnú, t.j. $\sigma(r,t) = \sigma$. Navrhol ho český matematik Oldřich Vašíček, podľa ktorého nesie aj pomenovanie [32] .

Vašíčkov model je jeden z prvých modelov okamžitej úrokovej miery, ktorej vývoj je vyjadrený nasledovnou stochastickou diferenciálnou rovnicou

$$dr = \kappa(\theta - r)dt + \sigma\, dw. \tag{8}$$

Výhodou tohto modelu je jednoduchosť, ktorú zabezpečuje konštantná volatilita. Mnoho vlastností vieme vďaka tomu odvodiť, čo si ukážeme neskôr. Má však aj svoje nevýhody. Najväčšou je, že pripúšťa záporné úrokové miery. Intuícia je taká, že aj pre úrokovú mieru r veľmi blízku 0 je volatilita stále konštantná a tak proces môže s nenulovou pravdepodobnosťou dosiahnuť aj záporné hodnoty úrokové miery. Ďalšou nevýhodou je fakt, že volatilita nezávisí od hodnoty krátkodobej úrokovej miery. Ako sme už spomínali, volatilita má rovnakú hodnotu bez ohľadu na to či má úroková miera hodnoty blízke 0 alebo sú nejaké veľké, čo nie je bežné správanie reálnych dát.

2.2. Cox-Ingersoll-Rossov model

Na odstránenie výskytu záporných úrokových mier bolo navrhnutých niekoľko modelov, jedným z nich je Cox-Ingersoll-Ross model (ďalej už iba CIR model) [3]. Drift zostáva rovnaký ako pri Vašíčkovom modeli, čiže sa stále jedná o mean reversion model. Rozdiel nastáva vo volatilite, ktorá už nie je konštantná, ale závisí od okamžitej úrokovej miery

$$dr = \kappa(\theta - r)dt + \sigma\sqrt{r}\, dw. \tag{9}$$

To spôsobuje, že volatilita je pri malých úrokových mierach malá a ak dosiahne nulu, volatilita bude tiež nulová. Ak teda $r = 0$, nulová je aj volatilita a drift je rovný $\kappa\theta$, čo je kladné číslo. To znamená, že tam nie je žiadna náhodnosť a ďalší vývoj je deterministický, určený kladným driftom, čiže úroková miera sa dostane znovu do kladných čísel. Navyše ak platí nerovnosť $2\kappa\theta \geq \sigma^2$, dosiahnutie $r = 0$ je nulovej pravdepodobnosti [32] .

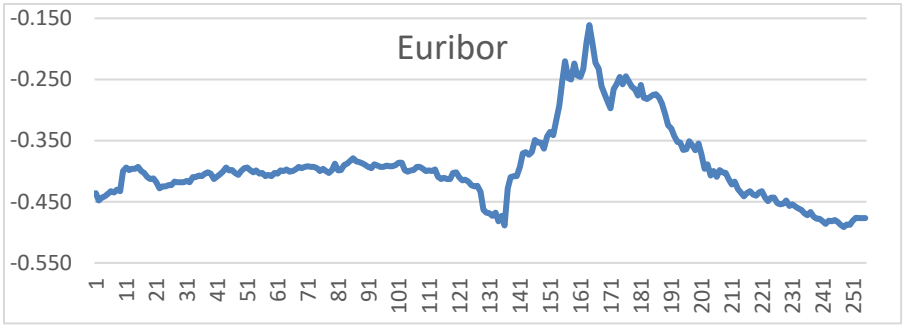
3. Kalibrácia jednofaktorových modelov

Cieľom spomínaných modelov je odhad parametrov tak, aby čo najlepšie a najpresnejšie odrážali pozorované hodnoty. V tejto kapitole si predstavíme dáta, s ktorými budeme pracovať, opíšeme prístup odhadu parametrov pomocou metódy maximálnej vierohodnosti a pokúsime sa ich nakalibrovať. Vychádzame najmä z [32] , [14] , [3], [16].

Pri metóde maximálnej vierohodnosti je dôležité poznať hustotu rozdelenia. Hustota CIR modelu sa riadi necentrálnym χ^2- kvadrát rozdelením, ktoré je zložitejšie v porovnaní s hustotou Vašíčkovho modelu riadiacou sa normálnym rozdelením, pre ktoré sa dajú odvodiť aj presné vzorce. Pri denných dátach bude hustota CIR modelu podobná hustote z Vašíčka. Z toho dôvodu na jej optimalizáciu použijeme taký algoritmus, ktorý dobre optimalizuje Vašíčka, pretože preň poznáme presné vzorce.

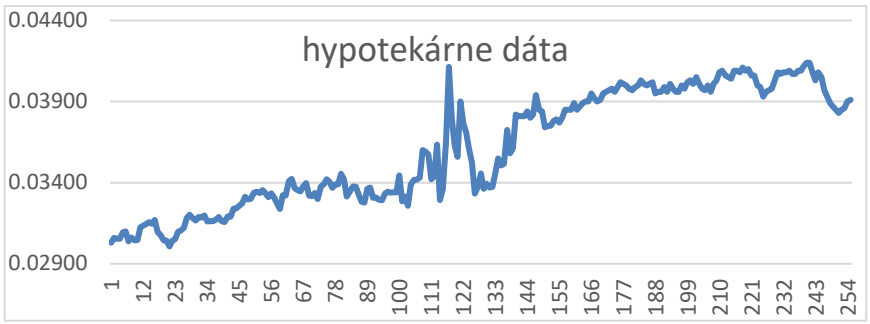
3.1. Výber dát

V tejto diplomovej práci sme sa rozhodli pracovať s dvoma typmi dát. Jedným je sadzba Euribor [7] za obdobie od 2.9.2019 do 31.8.2020 s dobou splatnosti 3 mesiace. Jej vývoj za toto obdobie môžeme vidieť na Obrázku 6. Euribor je úroková sadzba ponúkaná jednou bankou druhej na medzibankovom trhu. Táto sadzba je už dlhodobo záporná, čiže v tomto prípade použijeme na modelovanie iba Vašíčkov model.



Obrázok 6: Vývoj 3M sadzby Euribor za 1 rok

Druhé dáta sa týkajú hypotekárnej sadby istej americkej spoločnosti poskytujúcej financie [4]. Tieto dáta sme si vybrali z dôvodu ich kladných hodnôt. Budeme ich modelovať Vašíčkovým aj CIR modelom. Hypotekárne sadzby sme sledovali za rovnaké obdobie, teda od 2.9.2019 do 31.8.2020 a ich vývoj môžeme vidieť na Obrázku 7.



Obrázok 7: Vývoj hypotekárnej sadzby za 1 rok

3.2. Kalibrácia Vašíčkovho modelu

V nasledujúcej kapitole ukážeme jeden z možných prístupov ku kalibrácii modelov úrokových mier. Jednou z najčastejšie používaných metód na odhad parametrov je metóda maximálnej vierohodnosti. V prípade Vašíčkovho modelu, resp. CIR modelu potrebujeme odhadnúť parametre κ, θ a σ.

V [32] je odvodené podmienené rozdelenie úrokových mier vo Vašíčkovom modeli pomocou Fokker-Planckovej parciálnej diferenciálnej rovnice. Podmienené rozdelenie úrokovej miery o nejaký časový interval $r_{t+\Delta t}$ je normálne, pričom pre strednú hodnotu a variáciu platí

$$E(r_{t+\Delta t}|r_t) = e^{-\kappa\Delta t} r_t + (1 - e^{-\kappa\Delta t})\,\theta, \tag{10}$$

$$D(r_{t+\Delta t}|r_t) = \frac{\sigma^2}{2\kappa}(1 - e^{-2\kappa\Delta t}). \tag{11}$$

Z (10) vidíme, že ak sa t limitne blíži do nekonečna, stredná hodnota bude rovná θ, teda jej dlhodobému priemeru. Označme si $\eta = e^{-\kappa\Delta t}$ a $v^2 = \frac{\sigma^2}{2\kappa}(1 - e^{-2\kappa\Delta t})$. Potom normálne rozdelenie pre $r_{t+\Delta t}$ vyzerá nasledovne

$$r_{t+\Delta t} \sim N[\eta r_t + (1-\eta)\,\theta, v^2].$$

Pre metódu maximálnej vierohodnosti je dôležité poznať hustotu rozdelenia, z ktorého dáta pochádzajú. A keďže hustotu normálneho rozdelenia poznáme

$$f(r_{i+1}|r_i) = \frac{1}{\sqrt{2\pi v^2}}\, e^{-\frac{(r_{i+1}-\eta r_i-(1-\eta)\theta)^2}{2v^2}},$$

hodnotu vierohodnostnej funkcie vieme vyjadriť ako

$$L(\eta, \theta, v^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi v^2}}\, e^{-\frac{(r_{i+1}-\eta r_i-(1-\eta)\theta)^2}{2v^2}}.$$

Vyjadríme si ju ešte v logaritmickom tvare, s ktorým sa lepšie pracuje a dostávame logaritmickú vierohodnostnú funkciu, ktorú chceme maximalizovať

$$l(\eta, \theta, v^2) = \ln L(\eta, \theta, v^2) = -\frac{n}{2}\ln(2\pi v^2) - \frac{1}{2v^2}\sum_{i=1}^{n}(r_{i+1} - \eta r_i - (1-\eta)\theta)^2. \quad (12)$$

Túto funkciu sme nazvali MLE_vasicek a nájdeme ju aj zdrojových kódoch v dodatku A. Pre samotný odhad parametrov potom použijeme balík algoritmov [19]. V prípade Vašíčkovho modelu vieme pre parametre odvodiť aj presný vzorec. V diplomovej práci Simony Chattovej [14] , ktorá vychádzala z [32] sa môžeme dočítať viac o odvodení funkcie vierohodnosti a následne o odvodení jej parametrov η, θ a v^2 cez parciálne diferenciálne rovnice. V našej práci to nebudeme dopodrobna odvádzať. Odhady parametrov η, θ a v^2 teda sú

$$\hat{\eta} = \frac{(\sum_{i=1}^{n} r_i \sum_{i=1}^{n} r_{i+1} - n\sum_{i=1}^{n} r_i r_{i+1})}{[(\sum_{i=1}^{n} r_i)^2 - n \sum_{i=1}^{n} {r_i}^2]}, \quad (13)$$

$$\hat{\theta} = \frac{1}{n(1-\hat{\eta})}\left[\sum_{i=1}^{n} r_{i+1} - \hat{\eta} r_i\right], \quad (14)$$

$$\widehat{v^2} = \frac{1}{n}\sum_{i=1}^{n}\left[r_{i+1} - \hat{\eta} r_i - \hat{\theta}(1-\hat{\eta})\right]^2. \quad (15)$$

A odhady parametrov κ, θ a σ^2 sa dajú potom explicitne vyjadriť ako

$$\hat{\theta} = \frac{1}{n(1-\hat{\eta})}\left[\sum_{i=1}^{n} r_{i+1} - \hat{\eta} r_i\right], \quad (16)$$

$$\hat{\kappa} = -\frac{1}{\Delta t} ln(\hat{\eta}), \quad (17)$$

$$\widehat{\sigma^2} = \frac{2\hat{\kappa}}{1 - e^{-2\hat{\kappa}\Delta t}}\ \widehat{v^2}. \quad (18)$$

3.3. Odhad parametrov Vašíčkovým modelom

Odhad parametrov pomocou vzorcov [16]-[18] pre naše dáta je zobrazený v Tabuľke 5.

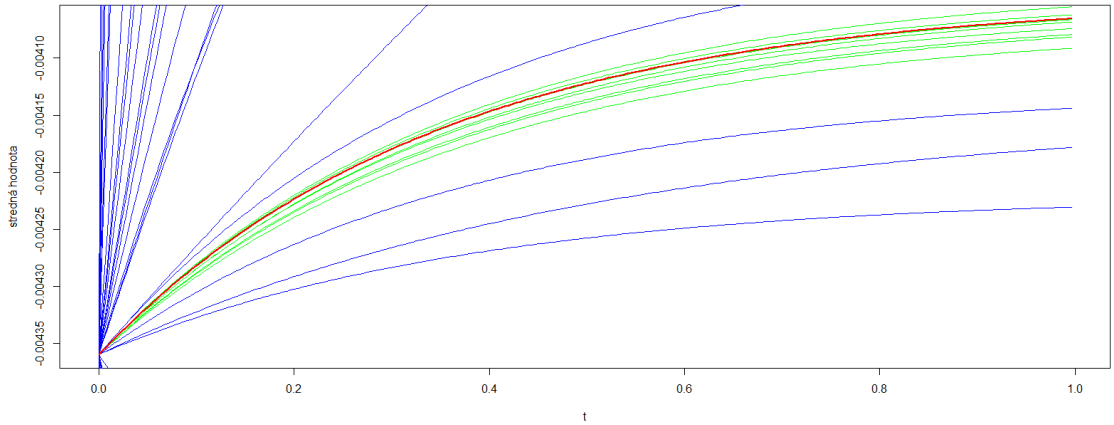
	κ	θ	σ
Euribor	2.86743310	-0.00404742	0.00172486
hypotekárne	6.66364529	0.03757292	0.01179387

Tabuľka 5: Odhad parametrov pomocou presných vzorcov

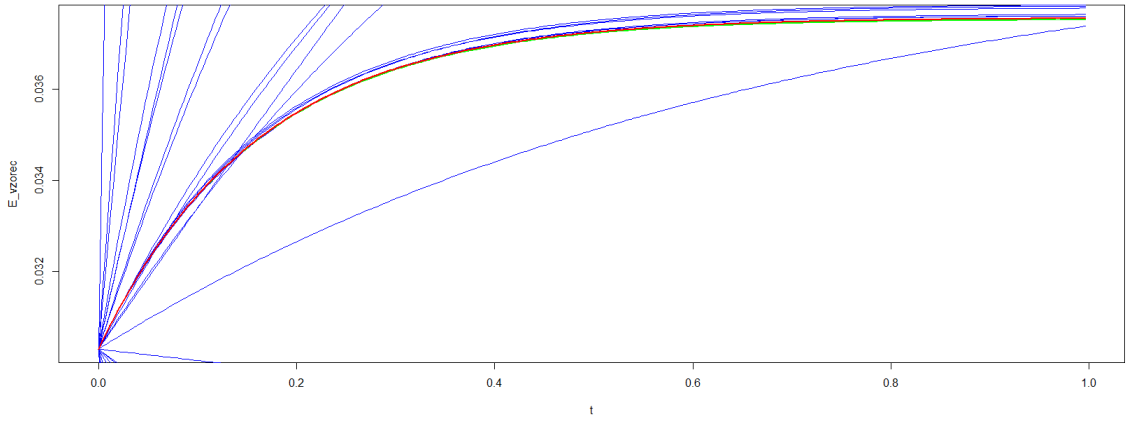
Môžeme si všimnúť, že v prípade sadzieb Euriboru je dlhodobá rovnovážna hodnota -0.41% a v prípade hypotekárnych sadzieb 3.76%. κ a θ sú kladné konštanty.

Pomocou metódy maximálnej vierohodnosti sme odhadli parametre aj daným balíkom algoritmov, ktorý sme v kapitole 1.3. otestovali na testovacích funkciách. Ich úspešnosť sme vyhodnocovali podľa predikcií stredných hodnôt (10) na nasledujúci rok a výsledky sme zobrazili graficky.

Na Obrázku 8 a 9 môžeme vidieť ukážku predikcie strednej hodnoty úrokovej miery algoritmom ABC pre zápornú mieru Euriboru a kladnú hypotekárnu sadzbu na jeden rok. Červenou farbou je znázornená predikcia strednej hodnoty pomocou parametrov z tabuľky 5 vypočítaných pomocou vzorcov [16]-[18] a zelenou sme vyznačili najlepšie predikcie. Môžeme vidieť, že pomerne presných predikcií je hneď viacero.

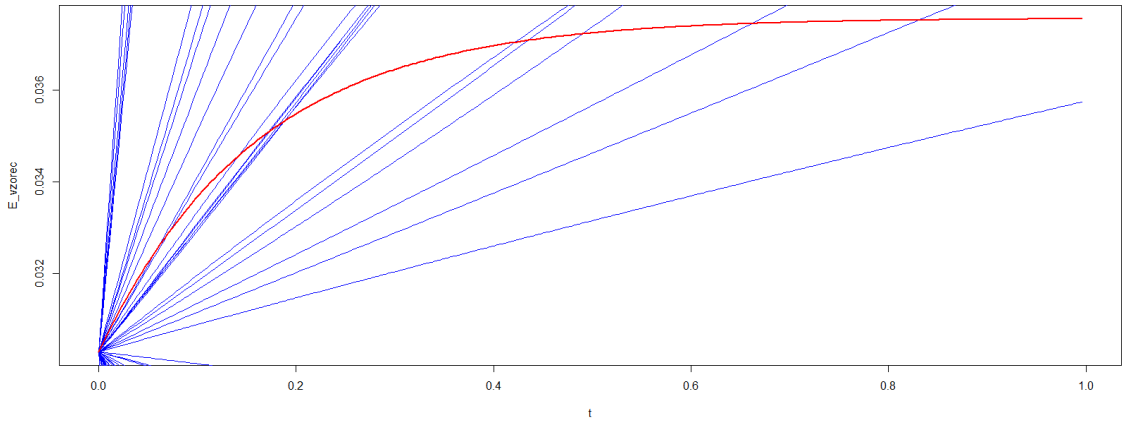


Obrázok 8: Predikcie zápornej sadzby Euribor na 1 rok algoritmom ABC

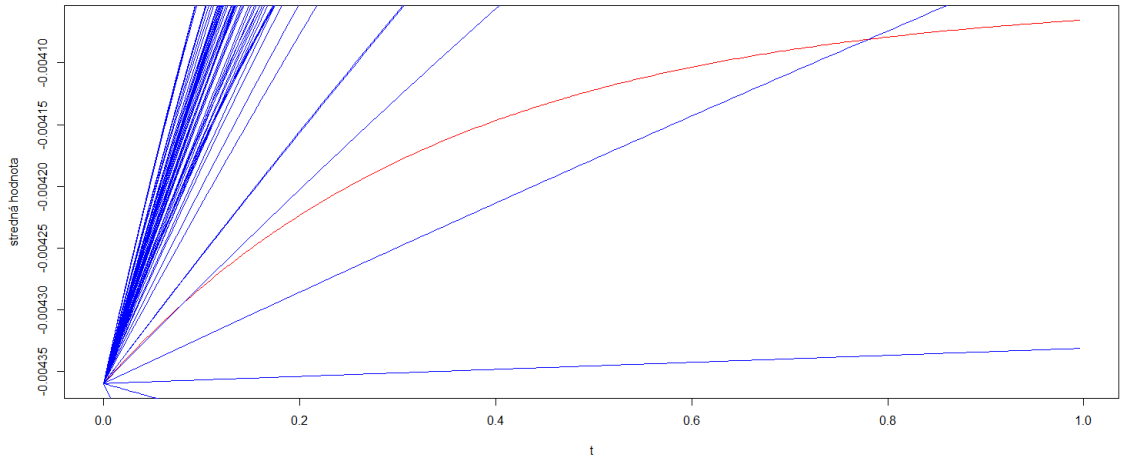


Obrázok 9: Predikcie kladnej hypotekárnej sadzby na 1 rok algoritmom ABC

Na druhú stranu, niektoré algoritmy sa ukázali ako nepresné a žiadna nami vyskúšaná kombinácia parametrov sa predikcii strednej hodnoty z presného vzorca ani nepriblížila. Takým je napríklad algoritmus ALO, ktorého predikcie vývoja strednej hodnoty úrokovej miery sú zobrazené na Obrázkoch 10 a 11.



Obrázok 10: Predikcie zápornej sadzby Euribor na 1 rok algoritmom ALO



Obrázok 11: Predikcie kladnej hypotekárnej sadzby na 1 rok algoritmom ALO

V Tabuľke 6 sme zobrazili prehľad úspešností jednotlivých algoritmov pre obe reálne dáta. Ak aspoň jedna kombinácia parametrov algoritmu predikovala vývoj strednej hodnoty blížiaci sa vývoju strednej hodnoty z presného vzorca, v Tabuľke 6 sa nachádza znak ✓. A naopak, ak nebola nájdená žiadna kombinácia vhodných parametrov, prislúcha tomu krížik X. Ak odhad parametrov trval algoritmu príliš dlho, považujeme ho za časovo neefektívny a v Tabuľke 6 mu prislúcha písmeno T. Pomlčkou označujeme také algoritmy, ktoré pri Vašíčkovom modeli z rôznych dôvodov zlyhali.

	sadzba Euribor	hypotekárna sadzba
ABC	✓	✓
ALO	X	X
WOA	X	X
SFL	X	X
SCA	✓	X
MFO	✓	✓
GWO	X	X
DA	-	-
GOA	T	T
BHO	✓	X
BA	✓	X
CLONALG	✓	✓
CS	✓	X
DE	✓	✓
CSO	-	-
GA	✓	X
GBS	T	T
HS	✓	✓
FFA	T	T
PSO	-	-
KH	T	T

Tabuľka 6: Prehľad úspešnosti algoritmov na reálnych dátach

3.4. Kalibrácia CIR modelu

Pri odhade parametrov CIR modelom potrebujeme opäť poznať hustotu procesu. Podľa [3] vieme, že hustota pravdepodobnosti úrokovej miery r v čase $t + \Delta t$ podmienená jej hodnotou v aktuálnom čase t je daná vzťahom

$$f(r_{t+\Delta t} | r_t) = c\, e^{-u-v} \left(\frac{v}{u}\right)^{\frac{q}{2}} I_q\left(2\sqrt{uv}\right), \tag{19}$$

kde $I_q\left(2\sqrt{uv}\right)$ je modifikovaná Besselova funkcia prvého druhu rádu q a

$$c = \frac{2\kappa}{\sigma^2(1 - e^{-\kappa\Delta t})},$$

$$u = c\, r_t e^{-\kappa\Delta t},$$

$$v = c\, r_{t+\Delta t},$$

$$q = \frac{2\kappa\theta}{\sigma^2} - 1.$$

Niekedy je užitočné pracovať s transformáciou $s_{t+\Delta t} = 2cr_{t+\Delta t}$. Z čoho sa dá ľahko odvodiť hustota pre $s_{t+\Delta t}$

$$f(s_{t+\Delta t}|\, s_t) = f(2cr_{t+\Delta t}|\, 2cr_t) = \frac{1}{2c} p(r_{t+\Delta t}|\, r_t), \tag{20}$$

využívajúca necentrálne χ^2- kvadrát rozdelenie s $2q + 2$ stupňami voľnosti a parametrom necentrality $2u$, ktorý je úmerný aktuálnej úrokovej miere [16].

3.5. Odhad parametrov CIR modelom

V tejto časti sme odhadovali iba hypotekárne dáta kvôli ich kladným hodnotám. Naprogramovali sme si funkciu MLE_CIR, nájdeme ju aj v zdrojových kódoch v dodatku A, ktorá maximalizuje logaritmus vierohodnostnej funkcie pre zvolený algortitmus. Hustotu sme počítali pomocou necentrálneho χ^2- kvadrát rozdelenia podobne ako v [16].

Keďže pri denných dátach je hustota CIR modelu podobná hustote Vašíčkovho modelu, pri odhade parametrov CIR modelu sme použili iba tie algoritmy, ktoré dobre optimalizovali Vašíčkov model. Teda tie, ktoré v Tabuľke 6 sú v stĺpci s hypotekárnymi dátami označené znakom ✓. Sú to algoritmy ABC, MFO, CLONALG, DE a HS. Algoritmus ABC však pri spustení funkcie MLE_CIR pre všetky kombinácie parametrov zlyhával vo výpočte hustoty, preto tento algoritmus v prípade CIR modelu nie je vhodný.

V Tabuľkách 7 až 10 uvádzame jednak parametre, ktoré pre dané algoritmy dobre optimalizovali Vašíčka a k nim príslušné odhady parametrov CIR modelom. Navyše tu uvádzame aj stĺpec s hodnotou účelovej funkcie pre odhadnuté parametre κ, θ, σ^2. Táto hodnota nám je užitočná, keď chceme určiť najlepšie nájdené riešenie pre CIR model. Keďže máme optimalizáciu zameranú na nájdenie minima, najlepšie nájdené riešenie bude mať spomedzi ostatných najmenšiu hodnotu účelovej funkcie. Takýmto najlepším riešením je červeným vyznačeným v Tabuľke 10 algoritmus DE s parametrami $scalingVector = 0.5$, $crossOverRate = 0.1$, $numPop = 100$ *a* $maxIter = 300$ a s hodnotou účelovej funkcie -1483.81039539189.

Môžeme si všimnúť, že všetky odhady parametrov a tým pádom aj účelové funkcie sú veľmi podobné a líšia sa až na určitých desatinných miestach. To značí, že tieto algoritmy s príslušnými kombináciami parametrov sú naozaj dobré.

MFO					
numPop	maxIter	κ	θ	σ^2	účelová funkcia
100	300	6.07313415	0.03761415	0.05985778	-1483.81033071890
200	300	6.07311497	0.03761417	0.05985778	-1483.81033072519
300	200	6.07328824	0.03761414	0.05985777	-1483.81033074851
300	300	6.07333230	0.03761414	0.05985777	-1483.81033071941

Tabuľka 7: Odhady parametrov CIR modelom pre algoritmus MFO

CLONALG							
MF	MR	num Pop	max Iter	κ	θ	σ^2	účelová funkcia
0.5	0.5	300	300	6.06803905	0.0376149	0.05985843	-1483.81038903088
0.9	0.5	300	300	6.06790362	0.0376147	0.05985808	-1483.81039329016
0.9	0.9	300	300	6.07466826	0.0376256	0.05985798	-1483.81036824706
0.5	0.5	300	300	6.06803905	0.0376149	0.05985843	-1483.81038903088
0.9	0.5	300	300	6.06790362	0.0376147	0.05985808	-1483.81039329016

Tabuľka 8: Odhady parametrov CIR modelom pre algoritmus CLONALG

HS							
PAR	HMCR	num Pop	max Iter	κ	θ	σ^2	účelová funkcia
0.1	0.5	300	300	6.06825615	0.03761496	0.05985816	-1483.81039229610
0.5	0.5	300	100	6.06349129	0.03761633	0.05986375	-1483.81038386158
0.5	0.5	300	300	6.07853737	0.03761143	0.05986284	-1483.81039224361
0.9	0.5	100	300	6.06607337	0.03761209	0.05985839	-1483.81038793964
0.9	0.5	300	300	6.06581756	0.03761448	0.05984304	-1483.81037077236
0.1	0.5	300	300	6.06825615	0.03761496	0.05985816	-1483.81039229610

Tabuľka 9: Odhady parametrov CIR modelom pre algoritmus HS

DE							
SV	CR	num Pop	max Iter	κ	θ	σ^2	účelová funkcia
0.3	0.5	100	300	6.07329696	0.03761414	0.05985777	-1483.81033074277
0.3	0.5	300	300	6.07329771	0.03761414	0.05985777	-1483.81033074228
0.5	**0.1**	**100**	**300**	**6.07329738**	**0.03761427**	**0.05985777**	**-1483.81039539189**
0.5	0.1	300	300	6.07363617	0.03761437	0.05985775	-1483.81039537240
0.5	0.5	100	300	6.07329825	0.03761414	0.05985777	-1483.81033074192
0.5	0.5	300	100	6.07327113	0.03761413	0.05985777	-1483.81033076279
0.5	0.5	300	300	6.07329732	0.03761414	0.05985777	-1483.81033074254
0.5	0.9	300	300	6.07329720	0.03761414	0.05985777	-1483.81033074261
0.7	0.1	300	300	6.07416265	0.03761384	0.05985774	-1483.81039531005
0.7	0.5	40	300	6.07329984	0.03761414	0.05985777	-1483.81033074087
0.7	0.5	100	100	6.07303804	0.03761419	0.05985778	-1483.81033076804
0.7	0.5	100	300	6.07329676	0.03761414	0.05985777	-1483.81033074290
0.7	0.5	300	100	6.07341687	0.0376142	0.05985776	-1483.81033076982

0.7	0.5	300	300	6.07330005	0.03761414	0.05985777	-1483.81033074074
0.7	0.5	40	100	6.07312929	0.03761399	0.05985778	-1483.81033076570
0.7	0.9	100	100	6.07329817	0.03761414	0.05985777	-1483.81033074197
0.7	0.9	100	300	6.07329813	0.03761414	0.05985777	-1483.81033074200
0.7	0.9	300	100	6.07329732	0.03761414	0.05985777	-1483.81033074254
0.7	0.9	300	300	6.07329803	0.03761414	0.05985777	-1483.81033074207
0.9	0.1	300	300	6.07234530	0.03761336	0.05985785	-1483.81039509271
0.9	0.5	40	300	6.07326938	0.03761412	0.05985777	-1483.81033076694
0.9	0.5	100	300	6.07331444	0.03761412	0.05985777	-1483.81033073736
0.9	0.5	300	100	6.07308000	0.03761448	0.05985778	-1483.81039533281
0.9	0.5	300	300	6.07330555	0.03761414	0.05985777	-1483.81033073711
0.9	0.9	40	300	6.07329735	0.03761414	0.05985777	-1483.81033074251
0.9	0.9	100	300	6.07329732	0.03761414	0.05985777	-1483.81033074254
0.9	0.9	300	100	6.07321760	0.03761411	0.05985778	-1483.81039537148
0.9	0.9	300	300	6.07329780	0.03761414	0.05985777	-1483.81033074222

Tabuľka 10: Odhady parametrov CIR modelom pre algoritmus DE

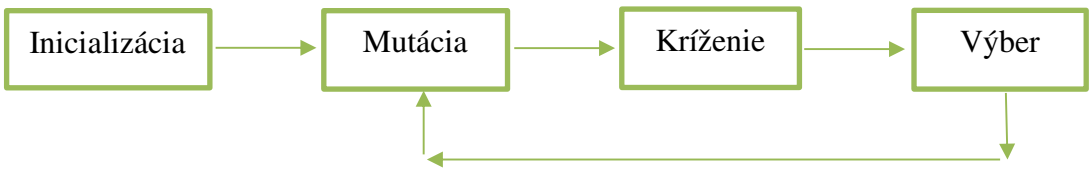
4. DE algoritmus

V tejto kapitole si bližšie popíšeme najúspešnejší algoritmus, ktorý nám vyšiel z predchádzajúcich odhadov parametrov úrokových mier. Najskôr si bližšie predstavíme štyri základné etapy DE algoritmu a potom si ukážeme jeho funkciu v programovacom jazyku R.

Algoritmus diferenciálnej evolúcie DE (angl. Differential Algorithm) patrí k evolučným algoritmom a je pravdepodobne jedným z najsilnejších stochastických optimalizačných algoritmov reálnych parametrov, ktorý sa v súčasnosti používa. V tejto kapitole sme čerpali z ako prvého publikovaného a rozsiahle vypracovaného článku o DE algoritme [5].

4.1. Etapy

DE je algoritmus na optimalizáciu reálnych parametrov funguje na základe jednoduchého cyklu etáp zobrazených na Obrázku 12.



Obrázok 12: Štyri základné cykly etáp DE algoritmu [5]

1. Inicializácia vektorov parametrov

Algoritmus DE hľadá globálny optimálny bod v R^D. Najskôr náhodne inicializuje populáciu veľkosti N. Každý vektor (známy tiež ako chromozóm/gén/jedinec) predstavuje kandidátske riešenie problému. Ďalšie generácie budeme označovať ako $G = 0,1, \dots, G_{max}$. Je zrejmé, že vektory parametrov sa v priebehu rôznych generácií budú meniť, preto na vyjadrenie i-teho vektora populácie súčasnej generácie použijeme označenie

$$\vec{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, \dots, x_{D,i,G}]$$

Pre každý vektor môžu existovať určité obmedzenia. Napríklad fyzické komponenty alebo miery majú prirodzené hranice. Počiatočná populácia ($G = 0$) by mala tento rozsah pokrývať čo najviac rovnomerným náhodným výberom jednotlivcov v rámci vyhľadávacieho priestoru s minimálnymi a maximálnymi hranicami $\vec{X}_{min} = \{x_{1,min}, \dots, x_{D,min}\}$ a $\vec{X}_{max} = \{x_{1,max}, \dots, x_{D,max}\}$. Preto môžeme inicializovať j-tu zložku i-teho vektora ako

$$x_{j,i,0} = x_{j,min} + rand_{i,j}[0,1] * (x_{j,max} - x_{j,min}),$$

kde $rand_{i,j}[0,1]$ je náhodné číslo z rovnomerného rozdelenia $Ro(0,1)$.

2. Mutácia s rozdielnymi vektormi

V našom kontexte sa mutácia vníma ako zmena alebo narušenie chromozómu/génu náhodným prvkom. DE algoritmus rozlišuje nasledovné vektory:

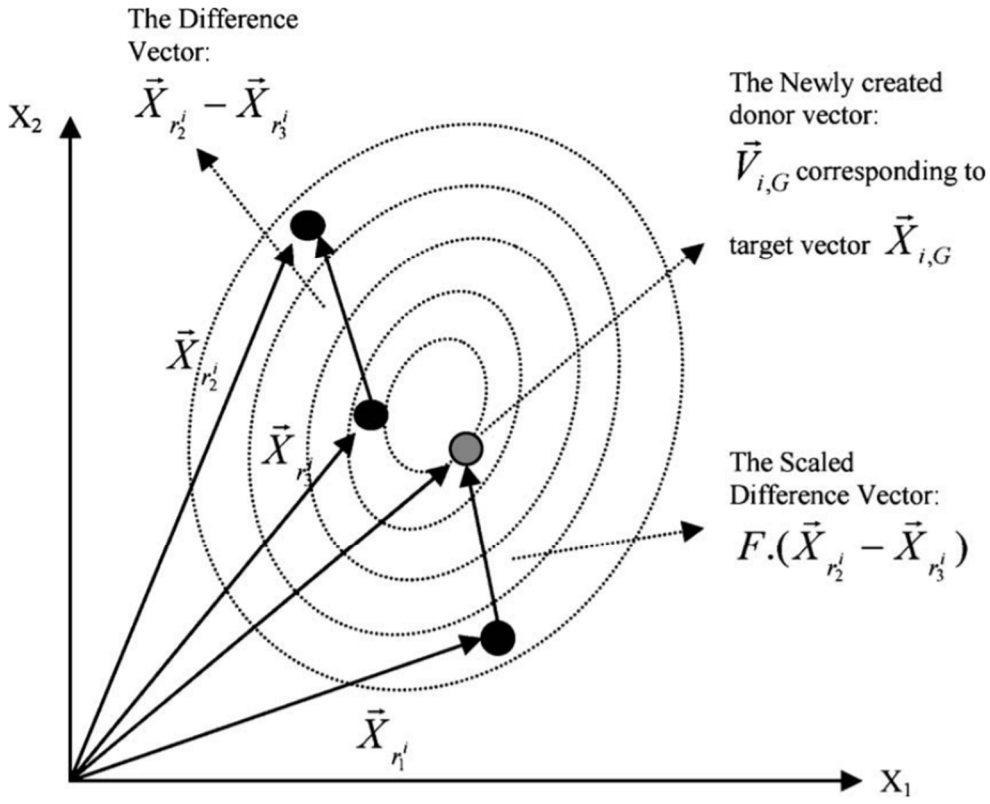
- *target (cieľový vektor)* – je to rodičovský vektor v súčasnej generácii
- *donor (darcovský vektor)* – mutačný vektor získaný operáciou diferenciálnej mutácie
- *trial* – potomok, ktorý vznikol kombináciou *darcovského* a *cieľového vektora*.

Na vytvorenie darcovského vektora pre každý i-ty cieľový vektor sa náhodne vyberú ďalšie tri vektory parametrov zo súčasnej populácie $\vec{X}_{r_1^i}, \vec{X}_{r_2^i}, \vec{X}_{r_3^i}$. Indexy r_1^i, r_2^i, r_3^i sú vzájomne vylučujúce sa náhodné čísla vybrané z rozsahu $[1, \mathrm{N}]$, ktoré sa líšia od základného vektorového indexu i. N ako sme už spomínali predstavuje počet jedincov v populácii. Tieto indexy sa náhodne generujú raz pre každý mutačný vektor. Následne darcovský vektor získame ako rozdiel ktorýchkoľvek dvoch z týchto troch vektorov $\vec{X}_{r_1^i}, \vec{X}_{r_2^i}, \vec{X}_{r_3^i}$ zmenšený

o skalárny faktor F (zvyčajne $F \in [0.4, 1]$) a pripočítame k tretiemu. Matematicky to môžeme vyjadriť nasledovne

$$\vec{V}_{i,G} = \vec{X}_{r_1^i,G} + F * \left(\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}\right)$$

a grafické vyjadrenie vidíme na Obrázku 13.



Obrázok 13: Štyri základné cykly etáp DE algoritmu [5]

3. Kríženie

Aby sa zvýšila potenciálna rozmanitosť populácie, po vytvorení darcovského vektora prichádza na rad kríženie. Darcovský vektor $\vec{V}_{i,G}$ si vymieňa svoje komponenty s cieľovým vektorom $\vec{X}_{i,G}$ a vytvoria potomka $\vec{U}_{i,G} = [u_{1,i,G}, u_{2,i,G}, \dots, u_{D,i,G}]$. Algoritmus DE využíva dve metódy kríženia: exponenciálnu a binomickú.

Pri exponenciálnom krížení sa najskôr náhodne vyberie celé číslo n z intervalu $[1, D]$. Toto číslo slúži ako začiatočný bod v cieľovom vektore, odkiaľ začína kríženie alebo výmena komponentov s darcovským vektorom. Z intervalu $[1, D]$ vyberieme ďalšie celé číslo L predstavujúce počet zložiek, ktoré má darcovský vektor prispieť cieľovému vektoru. Po vybratí n a L získame potomka ako

$$u_{i,j,G} = \begin{cases} v_{i,j,G}, & pre\ j = \langle n \rangle_D \quad \langle n+1 \rangle_D, \dots, \langle n+L-1 \rangle_D \\ x_{i,j,G}, & pre\ všetky\ ostatné\ j \in [1, D] \end{cases}, \tag{21}$$

kde $\langle\ \rangle_D$ označujú funkciu modulo s modulom D. Celé číslo L sa čerpá z nasledujúceho pseudokódu:

$$\begin{gathered} L = 0; \\ DO\ \{L = L + 1\} \\ WHILE\ \big((rand(0,1) \leq Cr)\ AND\ (L \leq D)\big). \end{gathered} \tag{22}$$

Cr sa nazýva miera kríženia a patrí spolu so skalárnym faktorom F k riadiacim parametrom algoritmu DE.

Binomické kríženie sa vykonáva na každej z D premenných, keď je náhodne vygenerované číslo z intervalu $[0, 1] \leq Cr$. V tomto prípade má počet parametrov zdedených od darcovského vektora (takmer) binomické rozdelenie. Schéma potomka by potom mohla byť načrtnutá nasledovne

$$u_{i,j,G} = \begin{cases} v_{i,j,G}, & ak\ rand_{i,j}[0, 1] \leq Cr\ alebo\ j = j_{rand} \\ x_{i,j,G}, & inak \end{cases}, \tag{23}$$

kde $\mathrm{rand_{i,j}}[0, 1]$ je náhodné číslo z rovnomerného rozdelenia $\mathrm{Ro}(0, 1)$, ktoré sa volá nanovo pre každú j-tu zložku i-teho vektora parametrov. $\mathrm{j_{rand}} \in [1, 2, .., \mathrm{D}]$ je náhodne vybraný index, ktorý zaisťuje, že $\vec{\mathrm{U}}_{\mathrm{i,G}}$ získa aspoň jednu zložku z $\vec{\mathrm{V}}_{\mathrm{i,G}}$. Je vytvorený raz pre každý vektor za jednu generáciu.

4. Výber

Na udržanie konštantnej veľkosti populácie počas nasledujúcich generácií ďalší krok algoritmu vyžaduje výber, aby sa určilo či cieľový vektor alebo potomok prežijú do ďalšej generácie, t.j $\mathrm{G} = \mathrm{G} + 1$. Operácia výberu je popísaná ako

$$\vec{X}_{i,G+1} = \begin{cases} \vec{U}_{i,G}, & ak\ f(\vec{U}_{i,G}) \leq\ f(\vec{X}_{i,G}) \\ \vec{X}_{i,G}, & ak\ f(\vec{U}_{i,G}) >\ f(\vec{X}_{i,G}) \end{cases}. \tag{24}$$

Ako sme už spomínali na začiatku, f(X) je účelová (fitness) funkcia, ktorá sa má minimalizovať. Preto na základe vyššie uvedeného sa populácia buď zlepšuje (s ohľadom

na minimalizáciu účelovej funkcie) alebo zostáva rovnaká. Všimnime si, že v (24) je cieľový vektor nahradený potomkom, aj keď sa hodnota účelovej funkcie nezlepšuje. To umožňuje preskúmanie ďalších potencionálnych riešení.

4.2. Funkcia DE v R

Na volanie zabudovanej funkcie DE algoritmu si potrebujeme najskôr načítať knižnicu metaheuristicOpt. Samotná funkcia pre DE algoritmus [19] potom vyzerá nasledovne

```
> DE(FUN, optimType, numVar, numPopulation, maxIter,
      rangeVar, scalingVector, crossOverRate, strategy)
```

- FUN – je účelová funkcia,
- optimType – určuje či budeme účelovú funkciu minimalizovať alebo maximalizovať,
- numVar – počet premenných,
- numPopulation – počet jedincov v populácii, prednastavená hodnota je $N = 40$,
- maxIter – určuje maximálny počet iterácií, slúži pri kritériu zastavenia, prednastavená hodnota je 500,
- randeVar – obmedzenia na rozsah pre jednotlivé premenné,
- scalingVector – skalárny faktor $F \in [0, 1]$, prednastavená hodnota je 0.8,
- crossOverRate – miera kríženia $Cr \in [0, 1]$, prednastavená hodnota je 0.5,
- strategy – určuje typ kríženia.

5. Viackriteriálna optimalizácia

Pojem „optimum“ má iný význam v prípade, keď máme jednu účelovú funkciu a v prípade, keď máme účelových funkcií viac. V prípade viackriteriálnej optimalizácie, ako už samotný názov hovorí, sa hľadajú riešenia, ktoré spĺňajú viacero kritérií, pretože jedno riešenie je len ťažko najlepšie pre všetky kritériá. Často používaný pojem „optimum“ zovšeobecnil ekonóm Vilfredo Pareto, ktorý povedal, že x^* je Pareto optimálne riešenie, teda také riešenie, v ktorom nejde zlepšiť jednu stratégiu bez toho, aby sa iná zhoršila [25] . Cieľom viackriteriálnej optimalizácie nie je teda nájsť jedno globálne riešenie, ale skôr skupinu kompromisných riešení, tzv. Pareto optimálnych riešení. V tejto kapitole čerpáme z [25] , [24] , [16].

Bez straty na všeobecnosti môžeme podľa [25] formulovať problém maximalizácie viackriteriálnej optimalizácie takto:

$$\max F(\vec{x}) = f_1(\vec{x}), f_2(\vec{x}), \dots, f_o(\vec{x}),$$

$$g_i(\vec{x}) \geq 0, \qquad i = 1,2, \dots, m,$$

$$h_i(\vec{x}) = 0, \qquad i = 1,2, \dots, p,$$

$$L_i \leq x_i \leq U_i, \qquad i = 1,2, \dots, n,$$

kde n je počet premenných, o počet účelových funkcií, m počet obmedzení nerovnosti, p počet obmedzení rovnosti a $[L_i, U_i]$ sú hraničné intervaly i-tej premennej.

Pri optimalizácii jednej účelovej funkcii sa dajú získané riešenia ľahko porovnať. V prípade maximalizácie je riešenie X lepšie ako Y práve vtedy, keď $X > Y$. Avšak takto to v prípade, keď máme viac účelových funkcií fungovať nemôže. V takom prípade je jedno riešenie lepšie ako iné (hovoríme, že *dominuje* inému riešeniu) práve vtedy, keď vykazuje lepšiu alebo rovnakú hodnotu účelovej funkcie pre všetky účelové funkcie a poskytuje lepšiu hodnotu aspoň v jednej z nich. Takéto riešenie nemusí existovať iba jedno, môže ich byť aj viac. Na základe [25] si teraz uvedieme štyri definície.

Definícia 1. Pareto dominancia

Uvažujme dva vektory $\vec{x} = (x_1, x_2, \dots, x_k)$ a $\vec{y} = (y_1, y_2, \dots, y_k)$. Vektor $\vec{x}$ dominuje vektoru $\vec{y}$ (označujeme $\vec{x} \succ \vec{y}$), ak

$$\forall\, i \in \{1,2, \dots, k\}, \qquad [f(x_i) \geq f(y_i)] \;\wedge\; [\exists\, \mathrm{i} \in 1,2, \dots, \mathrm{k} : f(x_i)]$$

Definícia 2. Pareto optimálne riešenie

Riešenie $\vec{x} \in X$ sa nazýva Pareto optimálne ak,

$$\nexists\ \vec{y} \in \mathrm{X} \mid f(\vec{y}) \succ f(\vec{x})$$

Riešenie, pre ktoré neexistuje prípustné riešenie, ktoré by mu dominovalo, budeme nazývať nedominované riešenie.

Definícia 3. Pareto optimálna množina

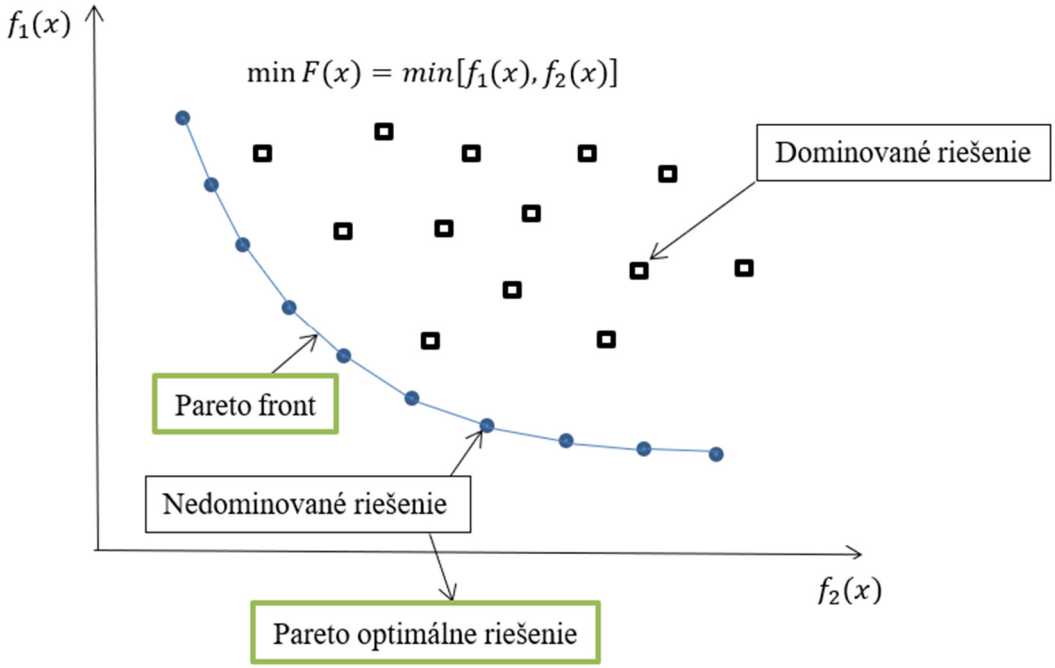
Množina obsahujúca všetky nedominované riešenia problému sa nazýva Pareto optimálna množina.

$$P_s \coloneqq \{x, y \in \mathrm{X} \mid \exists\, \mathrm{F}(y) \succ \mathrm{F}(x)\}$$

Definícia 4. Pareto optimálny front

Množina obsahujúca príslušné hodnoty Pareto optimálnych riešení v Pareto optimálnej množine sa nazýva Pareto optimálny front.

$$P_f \coloneqq \{\, \mathrm{F}(x) \mid \mathrm{x} \in P_s\}$$



Obrázok 14: Pareto optimálne riešenia dvojrozmernej funkcie, ktorú minimalizujeme [26]

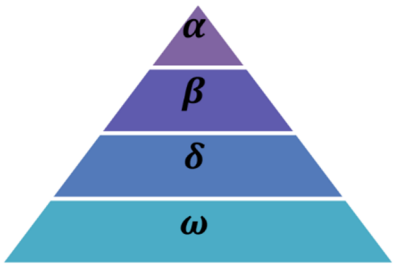
Na Obrázku 14 vidíme modrou farbou krivku predstavujúcu Pareto optimálny front. Jednotlivé body na tejto krivke predstavujú Pareto optimálne riešenia, pričom spolu tvoria množinu Pareto optimálnych riešení.

5.1. GWO algoritmus

V tejto kapitole sa budeme podrobne venovať optimalizačnému algoritmu šedého vlka GWO (angl. Grey Wolf Optimizer) [24] , ktorý bol navrhnutý ako nový algoritmus napodobňujúci vodcovskú hierarchiu a lovecký mechanizmus šedých vlkov v prírode. Najskôr si viac popíšeme, kde algoritmus čerpá inšpiráciu a potom prejdeme na samotný matematický model. V tejto kapitole sme čerpali zo zdrojov [24] a [25] .

5.1.1. Inšpirácia GWO algoritmu

Šedé vlky patria do čeľade psovité a žijú prevažne vo svorkách, ktorých veľkosť sa pohybuje okolo 5-12 členov. Sú považované za vrcholových predátorov, to znamená, že sú na vrchole potravinového reťazca. Vo svojej svorke majú veľmi prísnu sociálno-dominantnú hierarchiu, čo môžeme vidieť znázornené na Obrázku 15.



Obrázok 15: Sociálno-dominantná hierarchia šedého vlka [24]

Lídrami sú alfa samec a samica. Sú to dominantné vlky, ktoré rozhodujú o love, o mieste na prenocovanie či o čase vstávania. Alfy nemusia byť nevyhnutne najsilnejšie vlky, ale sú najlepšími vodcami, pretože udržiavanie disciplíny a organizácia sú oveľa dôležitejšie ako hrubá sila.

Druhou úrovňou v hierarchii sú vlky beta. Sú síce alfám podriadené, ale zároveň im pomáhajú a posilňujú ich príkazy, čím velia ostatným vlkom vo svorke. V prípade, že by niektorý alfa vlk zahynul alebo zostarol, sú najlepšími kandidátmi na jeho nahradenie.

Najnižšie postaveným vlkom v hierarchii je vlk omega, ktorý hrá rolu obetného baránka. Títo vlci sa musia vždy podriadiť všetkým ostatným dominantnejším.

Všetky ostatné vlky sú delta vlci. Patria sem skauti, ktorí sú zodpovední za sledovanie územia a varovanie v prípade nebezpečenstva, strážcovia zasa chránia a zaručujú bezpečnosť svorky, lovci pomáhajú alfám a betám pri love koristi a v neposlednom rade ošetrovatelia, ktorí zabezpečujú starostlivosť o slabých, chorých a zranených vlkov v svorke.

Okrem sociálno-dominantnej hierarchii šedých vlkov je zaujímavý aj ich skupinový lov pozostávajúci z troch fáz [24] :

- stopovanie, prenasledovanie a priblíženie sa ku koristi,
- zahnanie, obkľúčenie a vyčerpanie koristi, až kým sa neprestane hýbať,
- útok na korisť.

5.1.2. Matematický model GWO

Keď už vieme čím je algoritmus GWO inšpirovaný, v tejto kapitole si popíšeme loveckú techniku a sociálnu hierarchiu šedých vlkov matematickým modelom [24] .

Pri GWO algoritme budeme za najlepšie riešenie považovať alfa vlkov (α), druhým najlepším riešením vlkov beta (β) a tretím najlepším riešením vlkov delta (δ). Vlky omega

tvoria kandidátske riešenia (ω). V GWO algoritme sa lov, teda optimalizácia riadi vlkmi α, β, δ a vlci ω ich nasledujú. Samotný lov pozostáva zo 4 krokov [24] .

1. Obkľúčenie koristi

Ako sme už spomínali, šedí vlci najskôr obkľúčia svoju korisť. Matematicky môžeme definovať vzdialenosť medzi vlkmi a korisťou nasledovne

$$\vec{D} = \left|\vec{C}\vec{X}_p(t) - \vec{X}(t)\right|, \tag{25}$$

$$\vec{X}(t+1) = \vec{X}_p(t) - \vec{A}\,\overrightarrow{D}, \tag{26}$$

kde t označuje súčasnú iteráciu, $\vec{X}_p(t)$ je vektor pozície koristi, $\vec{X}(t)$ je vektor pozície šedého vlka a $\vec{A}$ a $\vec{C}$ sú vektory koeficientov, ktoré sa počítajú ako

$$\vec{A} = 2\,\vec{a}\vec{r}_1 - \vec{a}, \tag{27}$$

$$\vec{C} = 2\,\vec{r}_2. \tag{28}$$

Zložka $\vec{a}$ sa lineárne znižuje z 2 na 0 v priebehu iterácií a $\vec{r}_1, \vec{r}_2$ sú náhodné vektory z intervalu [0,1].

2. Lov

V reálnom živote majú šedé vlky schopnosť rozpoznať polohu koristi a obkľúčiť ju. V abstraktnom vyhľadávacom priestore však nemáme predstavu o umiestnení optima a preto predpokladáme, že informáciu o jeho možnom umiestnení by mohli mať najlepšie kandidátske riešenie alfa spolu s betou a deltou. Preto ukladáme prvé tri najlepšie riešenia do α, β, δ a ostatné ω aktualizujú svoju pozíciu podľa nich troch. Na základe tohto môžeme definovať nasledovné vzorce

$$\vec{D}_\alpha = \left|\vec{C}_1\,\vec{X}_\alpha - \vec{X}\right|, \tag{29}$$

$$\vec{D}_\beta = \left|\vec{C}_2\,\vec{X}_\beta - \vec{X}\right|, \tag{30}$$

$$\vec{D}_\delta = \left|\vec{C}_3\,\vec{X}_\delta - \vec{X}\right|, \tag{31}$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1\left(\vec{D}_\alpha\right), \tag{32}$$

$$\vec{X}_2 = \vec{X}_\beta - \vec{A}_2\left(\vec{D}_\beta\right), \tag{33}$$

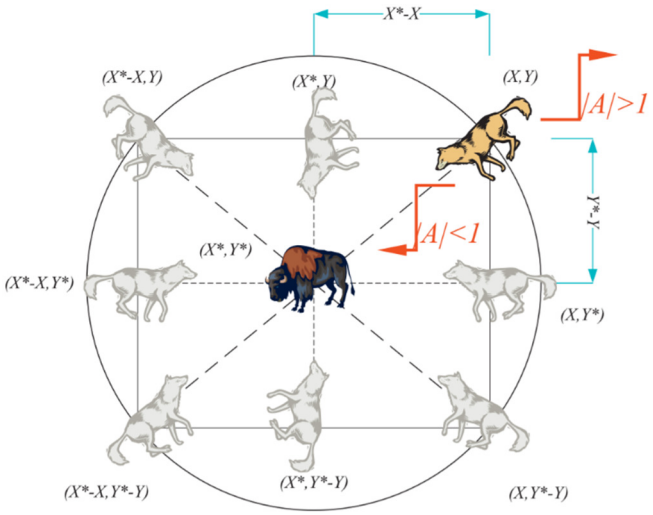
$$\vec{X}_3 = \vec{X}_\delta - \vec{A}_3\left(\vec{D}_\delta\right). \tag{34}$$

A pomocou (29)-(34) môžeme vyjadriť aktualizáciu polohy šedého vlka

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3}. \tag{35}$$

3. Zaútočenie

Ako sme už spomínali, aby sme matematicky modelovali približovanie sa ku koristi, znižujeme hodnotu $\vec{a}$. Všimnime si, že podľa (27) je hodnota $\vec{A}$ tiež znižovaná zložkou $\vec{a}$. $\vec{A}$ je náhodná hodnota z intervalu $[-a, a]$, pričom $\vec{a}$ je v priebehu iterácií zmenšená z 2 na 0. Keď náhodná hodnota $\vec{A}$ bude z intervalu $[-1,1]$, ďalšia poloha šedého vlka bude medzi jeho aktuálnou pozíciou a pozíciou koristi. Na Obrázku 16 vidíme, že $|A| < 1$ núti vlkov k útoku na korisť a naopak $|A| > 1$ núti vlkov, aby sa od nej odklonili.



Obrázok 16: $|A| < 1$ útok na korisť, $|A| > 1$ hľadanie koristi [25]

4. Hľadanie koristi

Doteraz vlky omega ω aktualizovali svoju pozíciu na základe polohy α, β, δ a následne zaútočili na korisť. Takto navrhnutý algoritmus má však tendenciu uviaznuť v lokálnych optimách, a to aj napriek prvému kroku pri obkľúčení koristi. Tam je síce vďaka náhodným vektorom $\vec{r}_1, \vec{r}_2$ do určitej miery prieskum, ale GWO na to potrebuje viac operátorov.

Ďalšou zložkou algoritmu GWO, ktorá podporuje prieskum je $\vec{C}$. V (28) môžeme vidieť, že vďaka náhodnému vektoru $\vec{r}_2$ nadobúda náhodne hodnoty z intervalu [0,2]. Podľa toho či $C < 1$ alebo $C > 1$ sa stochasticky zníži alebo zvýši účinok koristi pri definovaní vzdialenosti v rovnici (25).

Tu treba zdôrazniť, že $\vec{C}$ nie je lineárne znižované v priebehu iterácií ako to bolo v prípade $\vec{A}$, ale neustále poskytuje nové náhodné hodnoty. Tým pádom hľadanie bude

prebiehať počas každej jednej iterácii, čo robí z $\vec{C}$ užitočnú zložku v prípade uviaznutia v lokálnom minime.

5.1.3. Funkcia GWO v R

Opäť na volanie zabudovanej funkcie GWO algoritmu v prostredí R potrebujeme knižnicu metaheuristicOpt. Samotná funkcia pre GWO algoritmus [19] vyzerá nasledovne

```
> GWO(FUN, optimType, numVar, numPopulation, maxIter, rangeVar)
```

- FUN – je účelová funkcia,
- optimType – určuje či budeme účelovú funkciu minimalizovať alebo maximalizovať,
- numVar – počet premenných,
- numPopulation – počet jedincov v populácii, prednastavená hodnota je $N = 40$,
- maxIter – určuje maximálny počet iterácií, slúži pri kritériu zastavenia, prednastavená hodnota je 500,
- randeVar – obmedzenia na rozsah pre jednotlivé premenné.

Môžeme si všimnúť, že všetky z parametrov tejto funkcie obsahovala aj funkcia DE v kapitole 4.2.

5.2. MOGWO algoritmus

Ak okrem úrokovej miery chceme brať do úvahy aj napríklad fitovanie výnosových kriviek, vedie to na úlohu viackriteriálnej optimalizácie. Snahou viackriteriálnej optimalizácie je nájsť nie jedno riešenie, ale celú množinu Pareto optimálnych riešení [25] , ktoré nie sú z pohľadu daných kritérií dominované žiadnym iným riešením. V prípade GWO algoritmu si to vyžaduje určité modifikácie, aby vznikol MOGWO (angl. Multi Objective Grey Wolf Optimizer). Cieľom MOGWO je rozumnou časovou i výpočtovou náročnosťou nájsť, ak sa podarí, čo najviac spomínaných Pareto optimálnych riešení. Tu opäť čerpáme z [25] a [24]

Ako sme už spomínali v kapitole 5.2., algoritmus GWO začína optimalizáciu vygenerovaním množiny náhodných riešení ako prvej populácie. Počas optimalizácie sa ukladajú tri doposiaľ najlepšie získané riešenia a považujú sa za alfa, beta a delta vlkov. Každý vlk omega podľa príslušných vzorcov aktualizuje svoju pozíciu. Medzitým sa parametre a a A v priebehu iterácií lineárne znížia. Na záver sa vráti pozícia alfa vlka ako najlepšie riešenie.

Ak chceme rozšíriť GWO algoritmus na prípad s viacerými kritériami, teda s viacerými účelovými funkciami, integrujeme dva nové komponenty [25] . Prvým z nich je *archív*, do ktorého ukladáme doteraz získané nedominované Pareto optimálne riešenia. Druhým je *stratégia výberu lídra*, ktorá pomáha pri výbere riešení alfa, beta a delta ako lídrov z archívu.

- **Archív**

Archív slúži ako úložisko. Umožňuje ukladať alebo načítať doteraz získané Pareto optimálne riešenia. Kľúčovou zložkou je radič archívu, ktorý je užitočný, keď chce riešenie vstúpiť do archívu alebo keď je archív plný.

V priebehu iterácie sa doteraz najlepšie získané nedominované riešenia porovnávajú s riešeniami v archíve a keďže archív má obmedzenie na maximálny počet členov, môžu nastať tri prípady. Prvý prípad je, keď novému riešeniu dominuje aspoň jedno riešenie z archívu. Vtedy sa nedeje nič. Druhým prípadom je, keď nové riešenie dominuje jednému alebo viacerým riešeniam v archíve, vtedy sú následne takéto riešenia nahradené novým riešením. A posledným prípadom je, keď si ani nové riešenie ani riešenia z archívu navzájom nedominujú. Vtedy je nové riešenie taktiež pridané do archívu.

Ak je archív plný, spustí sa mriežkový mechanizmus, ktorý znova usporiada segmentáciu účelového priestoru a nájde najviac preplnený segment (hyperkocku), z ktorého následne vynechá jedno z riešení. Potom je nové riešenie vložené do najmenej preplneného segmentu, aby sa zlepšila rôznorodosť konečného Pareto optimálneho frontu.

- **Stratégia výberu lídra**

V algoritme GWO sa za vlkov alfa, beta a delta považovali tri doteraz najlepšie nájdené riešenia, ktoré vedú vlkov omega k perspektívnym oblastiam vyhľadávacieho priestoru. Vyhľadávací priestor v prípade viackriteriálnej optimalizácie však nie je možné tak ľahko porovnávať z dôvodu Pareto optimálneho konceptu ako sme si zadefinovali vyššie. Stratégia výberu lídra je navrhnutá na to, aby riešila tento problém. Keďže existuje archív s doteraz najlepšími nedominovanými riešeniami, komponent na výber lídra si vyberie najmenej preplnené segmenty vyhľadávacieho priestoru a ponúkne jedno z jeho nedominovaných riešení alfa, beta alebo delta. Výber sa vykonáva metódou rulety s nasledujúcou pravdepodobnosťou

$$P_i = \frac{c}{N_i},$$

kde $c > 1$ je konštanta a N je počet získaných Pareto optimálnych riešení v i-tom segmente. Z (35) môžeme vidieť, že najmenej preplnené segmenty majú vyššiu pravdepodobnosť, že

navrhnú nových lídrov. A keďže stratégia výberu lídra zvýhodňuje najmenej preplnené segmenty, hľadanie je tým pádom stále orientované na nepreskúmané oblasti prehľadávajúceho priestoru.

Algoritmus MOGWO má všetky vlastnosti ako GWO, čo znamená, že agenti vyhľadávania, teda omega vlci, prehľadávajú priestor rovnakým spôsobom. Hlavný rozdiel je v tom, že MOGWO prehľadáva navyše aj riešenia v archíve, zatiaľ čo GWO ukladá a vylepšuje iba tri najlepšie riešenia.

5.3. Nová funkcia MOGWO v R

V MATLABE existuje naprogramovaná funkcia [11] pre algoritmus MOGWO. A keďže v tejto práci sme prvotné testovanie funkcií i všetky ďalšie výpočty ohľadom odhadu parametrov modelov úrokových mier robili v programe R, rozhodli sme sa, že jedným z našich prínosov bude vytvorenie obdobnej funkcie algoritmu MOGWO ale práve v programe R.

Parametre sme zvolili rovnaké ako použili v kóde v MATLABE [11]:

```
alpha = 0.1
nGrid = 10
beta = 4
gamma = 2
GreyWolves_num = 10
MatIter = 100
Archive_size = 1000
```

Po vytvorení MOGWO v R je dobré si nejako jeho správnosť overiť. Ďalšia kapitola sa bude venovať práve testovaniu nášho kódu.

5.4. Testovanie novej funkcie MOGWO v R

Overiť nami vytvorenú funkciu v R sme sa rozhodli na dvoch testovacích funkciách, ktoré budeme obe minimalizovať. Prvú testovaciu funkciu sme si vymysleli tak, aby sa nám ľahšie overovala správnosť nájdených Pareto optimálnych riešení. Táto testovacia funkcia obsahuje dve účelové funkcie s kvadratickými zložkami, resp. ich súčtom. Predpis funkcií v MATLABE a ich ohraničenia sú nasledovné:

```
function y = F_Pareto1(x)
    y(1,:)  = x(1,:).^2;
    y(2,:)  = (1.0 - x(1,:)).^2 + x(2,:).^2;
end
xrange =
               -10      10
               -10      10
```

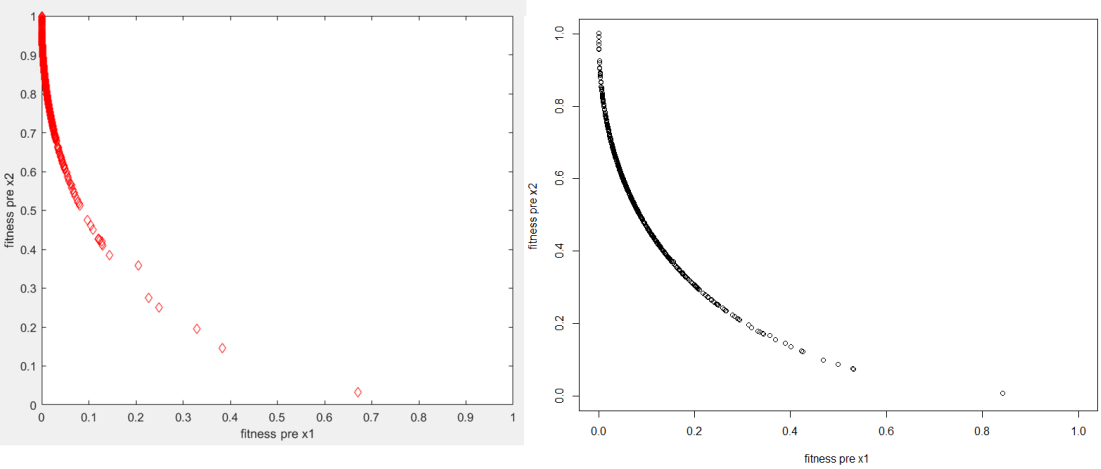
Na Obrázku 17 môžeme vidieť hodnoty účelových funkcií pre Pareto optimálne riešenia MOGWO algoritmu pomocou prvej testovacej funkcie $F_Pareto1$ pre funkciu v MATLABE a pre našu funkciu v R. Vidíme, že nájdené Pareto optimálne riešenia sú dobré. Veľký počet nájdených riešení spôsobil, že máme takmer hladkú krivku, ktorá značí lineárnu závislosť.

V Tabuľke 11 môžeme nájsť príklad počtu nájdených Pareto optimálnych riešení pri daných parametroch pre prvú testovaciu fukciu. Vysoký počet nájdených riešení bude z toho dôvodu, že dvojica účelových funkcií nie je zložitá a nájsť kompromisné riešenia nebolo pre algoritmus MOGWO ťažkou záležitosťou. Pozrieme sa preto aj na zložitejšiu funkciu.

Okrem počtu nájdených Pareto optimálnych riešení môžeme z Tabuľky 11 vyčítať aj čas, za aký boli tieto riešenia nájdené. Vidíme, že výpočty v MATLABE trvali v tomto prípade približne 2 minúty, pričom v R boli vykonané za približne 1 minútu.

	čas trvania výpočtu v sekundách	počet nájdených Pareto optimálnych riešení
MATLAB	121.41	605
R	59.99	729

Tabuľka 11: Porovnanie času trvania výpočtu a počtu nájdených riešení MATLAB kódom a naším R kódom pre prvú testovaciu funkciu



Obrázok 17: Hodnoty účelových funkcií Pareto optimálnych riešení pre testovaciu funkciu F_Pareto1, vľavo výstup z MATLABU a vpravo z R

Ako druhú testovaciu funkciu sme si vybrali jednu z preddefinovaných v [11]. Má opäť dve účelové funkcie tentokrát obsahujúce goniometrické funkcie a rôzne pripočítania. Ich predpis a ohraničenia sú nasledovné:

```
function y = F_Pareto2(x)

    [dim,num] = size(x);
    tmp = zeros(dim,num);
    tmp(2:dim,:)=(x(2:dim,:)-sin(6.0*pi*repmat(x(1,:),
    [dim-1,1])+ pi/dim*repmat((2:dim)',[1,num]))).^2;
    tmp1=sum(tmp(3:2:dim,:));
    tmp2=sum(tmp(2:2:dim,:));
    y(1,:)= x(1,:)+ 2.0*tmp1/size(3:2:dim,2);
    y(2,:)= 1.0 - sqrt(x(1,:)) + 2.0*tmp2/size(2:2:dim,2);

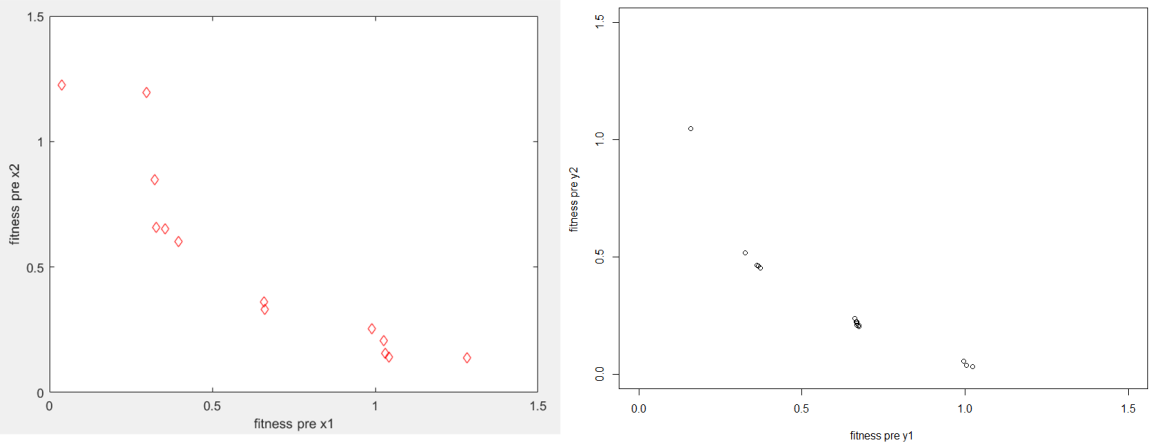
clear tmp;
end

xrange =     0     1
            -1     1
            -1     1
            -1     1
            -1     1
            -1     1
            -1     1
            -1     1
            -1     1
            -1     1
```

Na Obrázku 18 môžeme podobne ako v predchádzajúcom prípade vidieť hodnoty účelových funkcií pre Pareto optimálne riešenia druhej testovacej funkcie $F_Pareto2$ pre funkciu v MATLABE a pre našu funkciu v R. Ak porovnáme tieto dva výstupy, môžeme pozorovať podobné výsledky. Vzhľadom na to, že máme menší počet nájdených Pareto optimálnych riešení (v MATLABE 15 a v R 20), grafom nie je hladká krivka ako tomu bolo v predchádzajúcom prípade. Počet nájdených Pareto optimálnych riešení a čas, za aký boli nájdené nájdeme v Tabuľke 12. Všimnime si, že R dokázalo nájsť 20 riešení už za 1.75 sekundy, pričom MATLAB našiel 15 riešení za 18.05 sekundy.

	čas trvania výpočtu v sekundách	počet nájdených Pareto optimálnych riešení
MATLAB	18.06	15
R	1.75	20

Tabuľka 12: Porovnanie času trvania výpočtu a počtu nájdených riešení MATLAB kódom a naším R kódom pre druhú testovaciu funkciu



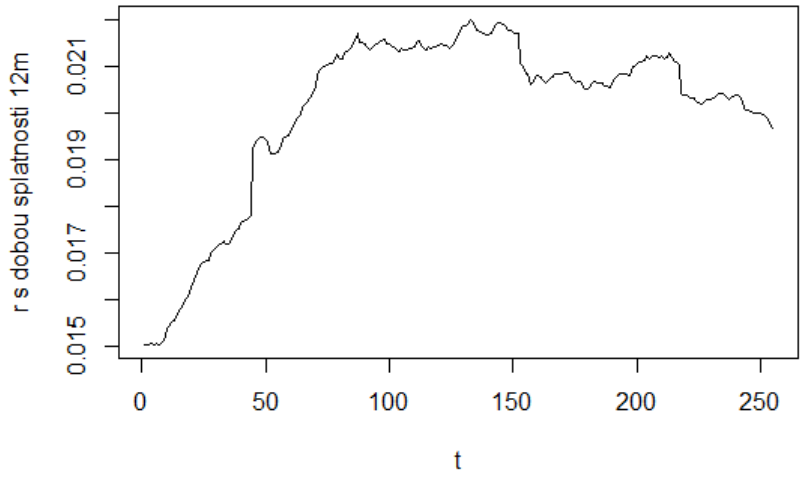
Obrázok 18: Hodnoty účelových funkcií Pareto optimálnych riešení pre testovaciu funkciu F_Pareto2, vľavo výstup z MATLABU a vpravo z R

Na základe informácií získaných z testovania na dvoch testovacích funkciách $F_Pareto1$ a $F_Pareto2$ (obe zložené z dvoch účelových funkcií) prichádzame k záveru, že naša funkcia pre algoritmus MOGWO bola naprogramovaná správne. Navyše pozorujeme, že jej veľkou výhodou oproti funkcii v MATLABE je rýchlosť nájdenia Pareto optimálnych riešení. Z toho dôvodu by naša funkcia mohla byť užitočná nielen pre tých, ktorí radšej pracujú v prostredí R, ale aj pre jej rýchlejšie výpočty v porovnaní v MATLABOM.

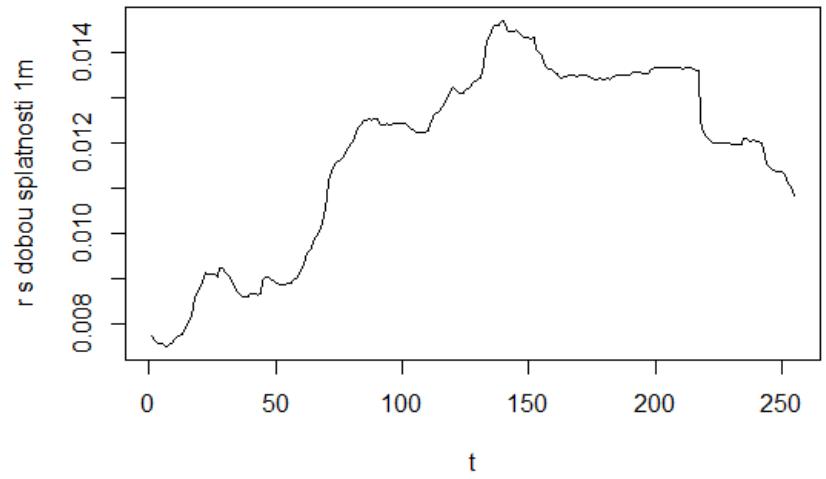
6. Aplikácia algoritmu MOGWO na reálne dáta

Na záver našej práce implementujeme MOGWO algoritmus pre viackriteriálnu optimalizáciu pomocou našej funkcie v R na reálne trhové dáta. Použijeme ju na kalibráciu, ktorá berie do úvahy nielen priebeh krátkodobej úrokovej miery, aj ale fitovanie výnosových kriviek. Pri odvodení druhej účelovej funkcie sme vychádzali z [32].

Za prvú účelovú funkciu si vezmeme už naprogramovanú funkciu MLE_CIR z kapitoly 3.5., ktorá maximalizuje logaritmus vierohodnostnej funkcie. Na kalibráciu použijeme úrokovú mieru EURIBOR s dobou splatnosti jeden mesiac [7]. Túto úrokovú mieru budeme označovať r. Dáta berieme za rok 2011, kedy nenadobúdali ešte záporné hodnoty, čo je podmienkou pre CIR model. Okrem toho budeme používať aj úrokovú mieru EURIBOR s dobou splatnosti 12 mesiacov za ten istý rok 2011, pretože druhou účelovou funkciou budú odchýlky výnosov týchto dvoch splatností. Takúto úrokovú mieru označíme r_12m.



Obrázok 19: Priebeh úrokovej miery EURIBOR s dobou splatnosti 12 mesiacov



Obrázok 20: Priebeh úrokovej miery EURIBOR s dobou splatnosti 1 mesiac

Prevodový vzťah medzi cenou dlhopisu $P(t,T)$ (ceny dlhopisu v čase t, ak jeho maturita je v čase T) a úrokovou mierou $R(t,T)$ (úroková miera s maturitou T v čase t) [32] je nasledovný

$$P(t,T) = e^{-R(t,T)(T-t)} \longrightarrow R(t,T) = -\frac{\ln P(t,T)}{T-t}. \tag{36}$$

Parciálna diferenciálna rovnica pre cenu dlhopisu [32] $P = P(r,t)$ je

$$\frac{\partial \mathrm{P}}{\partial \mathrm{t}} + \big(\mu(r,t) - \lambda(r,t)\,\sigma(r,t)\big)\frac{\partial \mathrm{P}}{\partial \mathrm{r}} + \frac{\sigma^2(r,t)}{2}\frac{\partial^2 P}{\partial \mathrm{r}^2} - rP = 0, \tag{37}$$

pre $r \in (0,\infty)$ a $t \in (0,T)$. V čase splatnosti je hodnota dlhopisu jedna, teda $P(r,T) = 1$. Vystupuje tu navyše funkcia $\lambda = \lambda(r,t)$, ktorá vyjadruje trhovú cenu rizika a je to ďalší parameter, ktorý budeme odhadovať, aby sme boli schopní oceňovať deriváty.

Riešenie ceny dlhopisu hľadáme v špeciálnom tvare

$$P(r,\tau) = A(\tau)\, e^{-B(\tau)\, r}, \tag{38}$$

kde substitúcia $\tau = \mathrm{T} - \mathrm{t}$ vyjadruje čas do splatnosti. Po dosadení ceny dlhopisu (38) do (37) dostaneme systém obyčajných diferenciálnych rovníc pre funkcie $A(\tau)$ a $B(\tau)$, ktorý sa dá explicitne vyriešiť

$$\dot{A} + \kappa\theta AB = 0, \tag{39}$$

$$\dot{B} + (\kappa + \lambda\sigma)B + \frac{\sigma^2}{2}B^2 - 1 = 0, \tag{40}$$

so začiatočnými podmienkami $A(0) = 1$ a $B(0) = 0$.

Použijeme tvar riešenia z [32] pre predpis $A(\tau)$ a $B(\tau)$, s ktorým sa najčastejšie stretneme aj v literatúre

$$B(\tau) = \frac{2(e^{\Phi\tau} - 1)}{(\psi + \Phi)(e^{\Phi\tau} - 1) + 2\Phi}, \tag{41}$$

$$A(\tau) = \left(\frac{2\Phi\left(e^{(\psi+\Phi)}\right)^{\tau/2}}{(\psi + \Phi)(e^{\Phi\tau} - 1) + 2\Phi}\right)^{2\kappa\theta/\sigma^2}, \tag{42}$$

kde $\psi = \kappa + \lambda\sigma$ a $\Phi = \sqrt{\psi^2 + 2\sigma^2}$.

Podľa (38) a (36) teda dostávame finálny výpočet úrokovej miery r s dobou splatnosti 1 mesiac

$$R(t,\tau) = -\frac{\ln(A) - Br}{\tau} = -\frac{\ln(A)}{\tau} + \frac{\mathrm{B}}{\tau}r. \tag{43}$$

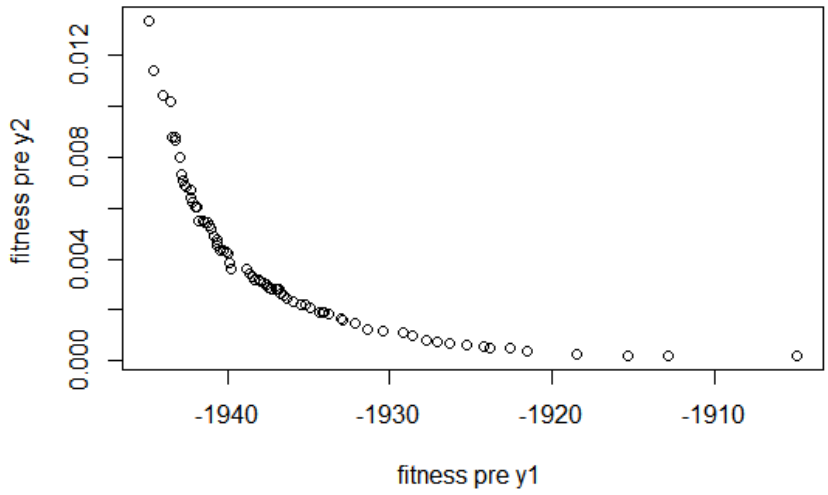
Za druhú účelovú funkciu potom zvolíme výnosovú krivku, teda rozdiel úrokovej miery EURIBOR s dobou splatnosti 12 mesiacov a vypočítanej krátkodobej úrokovej miery s dobou splatnosti 1 mesiac.

$$[r_12m - R(t,\tau)]^2. \tag{44}$$

Dvojicu účelových funkcií sme v programe nazvali fobj a nájsť ju môžeme aj v zdrojových kódoch v dodatku A. Odhadovať budeme neznáme parametre κ, θ, σ, ktoré sú zo stochastickej diferenciálnej rovnice pre okamžitú úrokovú mieru CIR modelu a parameter λ predstavujúci trhovú cenu rizika. Čas do splatnosti je jeden rok, teda za τ zvolíme 1.

Za 4.38 sekundy bolo nájdených 94 Pareto optimálnych riešení. Na Obrázkoch 22 až 25 sme postupne vykreslili histogramy pre parametre κ, θ, σ^2, λ z množiny nájdených Pareto optimálnych riešení a v Tabuľke 13 sme vypísali niekoľko rôznych príkladov takto

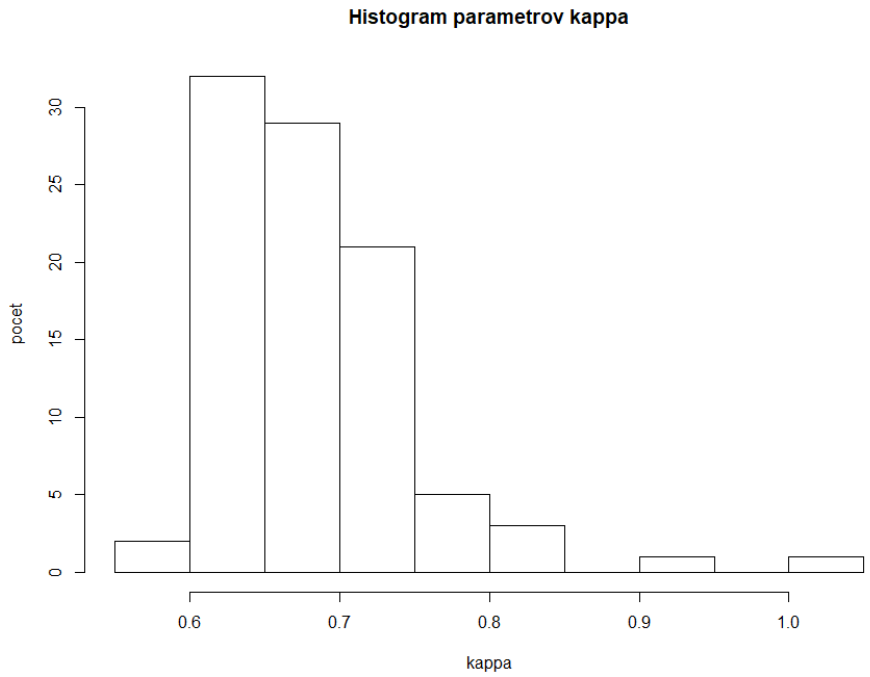
nájdených Pareto optimálnych riešení. Na Obrázku 21 sú znázornené hodnoty účelových funkcií v týchto Pareto optimálnych riešeniach.



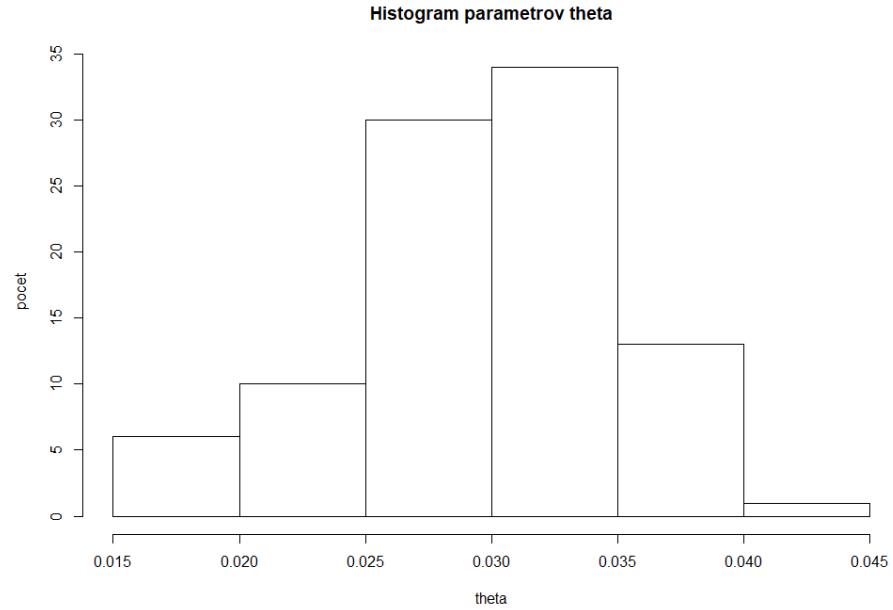
Obrázok 21: Hodnoty účelových funkcií Pareto optimálnych riešení

$\boldsymbol{\kappa}$	$\boldsymbol{\theta}$	$\boldsymbol{\sigma^2}$	$\boldsymbol{\lambda}$
0.918947	0.016799	0.016365	4.425155
1.042236	0.016867	0.016435	1.897746
0.772047	0.021598	0.017399	-0.59802
0.818315	0.019359	0.017435	0.343571
0.797726	0.037432	0.020079	0.230689
0.819603	0.015936	0.016619	0.410538
0.562015	0.025541	0.017472	0.447007
0.704251	0.019244	0.016779	1.061659
0.618318	0.028434	0.017027	0.215207
0.703749	0.023098	0.01712	0.285689
0.623873	0.031832	0.017599	0.279917

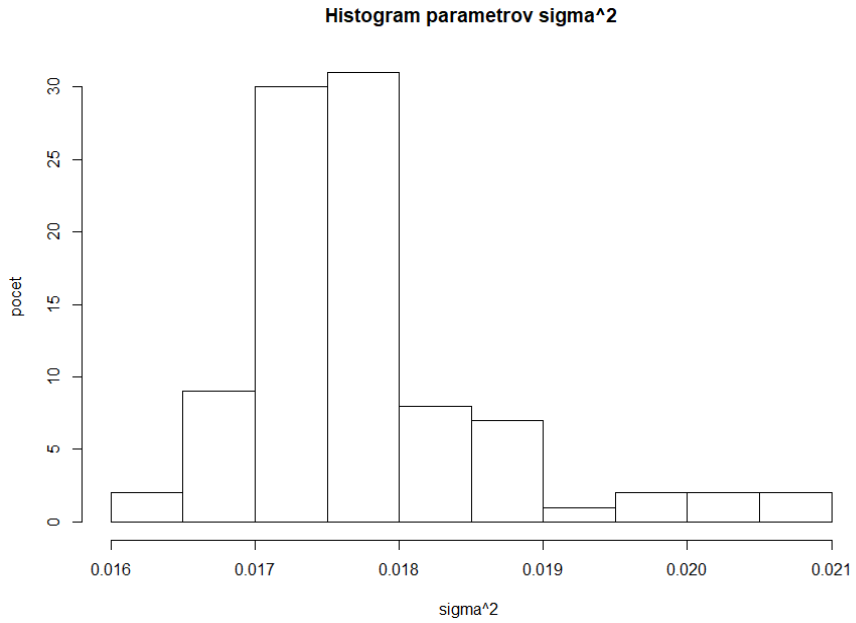
Tabuľka 13: Príklady Pareto optimálnych riešení



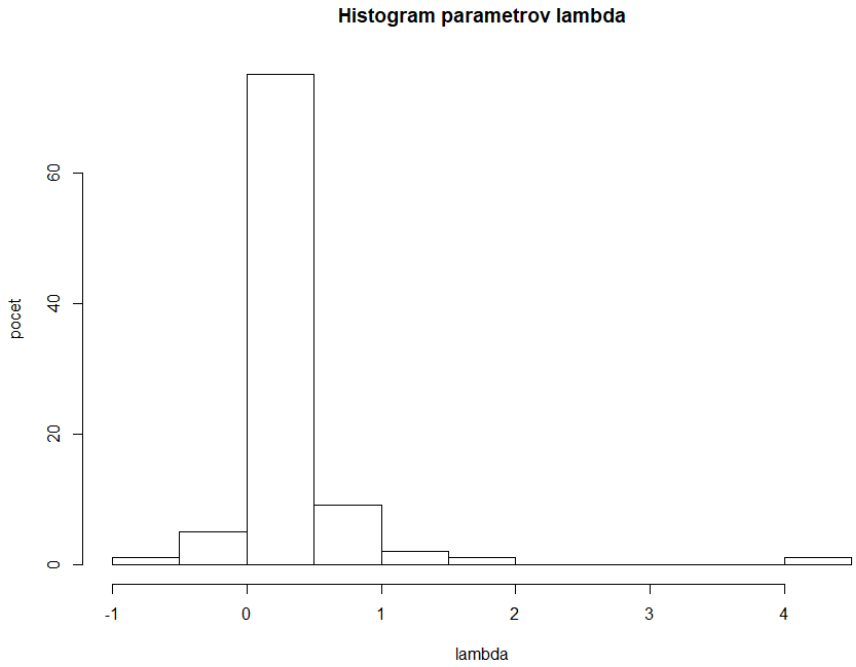
Obrázok 22: Parametre κ z množiny Pareto optimálnych riešení



Obrázok 23: Parametre θ z množiny Pareto optimálnych riešení



Obrázok 24: Parametre σ^2 z množiny Pareto optimálnych riešení



Obrázok 25: Parametre λ z množiny Pareto optimálnych riešení

Záver

Metaheuristické optimalizačné algoritmy za posledné dve desaťročia nadobudli veľkú popularitu a stali sa výkonnými nástrojmi na modelovanie a optimalizáciu.

V úvode tejto práce sme sa snažili poskytnúť prehľad metaheuristických optimalizačných algoritmov inšpirovaných prírodou z balíka [19]. Z triedy evolučných algoritmov sme si stručne predstavili genetický algoritmus GE, algoritmus hľadania harmónie HS a algoritmus diferenciálnej evolúcie DE, tomuto algoritmu vďaka jeho dobrým výsledkom bola neskôr venovaná aj samostatná kapitola. Z rojových algoritmov sme popísali algoritmus optimalizácie roja častíc PSO, algoritmus umelých včelstiev ABC, algoritmus svätojánskych mušiek FFA a algoritmus vyhľadávania kukučiek CS. Prvá kapitola sa ďalej venovala testovaniu tohto balíka algoritmov [19]. Testovanie prebiehalo na štyroch testovacích funkciách, pričom cieľom bolo zistiť ako sa jednotlivé algoritmy správajú pri zmene svojich parametrov. Ukázalo sa, že medzi parametre, ktoré najviac ovplyvňujú presnosť riešenia patria najmä *numPop* a *maxIter.* V niektorých prípadoch to boli aj iné parametre, napríklad *min_Frequency* a *max_Frequency* pri algoritme BA, *mixtureRatio* pri algoritme CSO alebo *w* pri algoritme PSO. Taktiež sme zistili, že algoritmy FFA, HK a GBS nie sú časovo efektívne. A keďže v optimalizácii nám nestačí určiť len správne riešenie, ale taktiež ho chceme určiť v rozumnom čase, označili sme tieto algoritmy za nevyhovujúce.

V druhej časti tejto práce sme sa zoberali dvomi jednofaktorovými modelmi úrokovej miery, a to Vašíčkovím a CIR modelom. Oba patria do triedy mean reversion modelov, pričom sa od seba líšia volatilitou. Zatiaľ čo Vašíčkov model má konštantnú volatilitu σ, volatilita CIR modelu závisí od okamžitej úrokovej miery $\sigma\sqrt{r}$. Vďaka tomu CIR model viac zodpovedá realite a modeluje úrokovú mieru tak, že hodnoty tohto procesu sa nikdy nedostanú do záporných čísel.

V tretej kapitole sme sa venovali kalibrácii týchto modelov na dvoch reálnych dátach, konkrétne na zápornej miere Euriboru a kladnej hypotekárnej sadzbe za referenčné obdobie od 2.9.2019 do 31.8.2020. Na odhad parametrov modelu sme použili metódu maximálnej vierohodnosti, na ktorej optimalizáciu sme použili daný balík optimalizačných algoritmov [19]. Keďže hustota Vašíčkovho modelu sa riadi známym normálnym rozdelením, odhad jeho parametrov sme vedeli vypočítať aj pomocou presných vzorov. Na vyhodnotenie úspešnosti sme potom použili predikcie stredných hodnôt a porovnali to s predikciami pomocou parametrov z presných vzorcov. Úspešných algoritmov bolo hneď viacero, pre

kladné dáta sú to algoritmy ABC, MFO, CLONALG, DE a HS a pre záporné dáta sú to okrem už spomínaných algoritmov aj navyše algoritmy SCA, BHO, BA, CS a GA. Čo sa týka CIR modelu, pri denných dátach je jeho hustota podobná hustote Vašíčkovho modelu. Pri odhade jeho parametrov sme použili preto iba tie algoritmy, ktoré dobre optimalizovali Vašíčkov model. Z nich algoritmus ABC zlyhával pri výpočte hustoty a tak sme ten najúspešnejší vybrali zo zvyšných štyroch algoritmov. Tým najúspešnejším vyšiel algoritmus diferenciálnej evolúcie DE, ktorý sme v kapitole 4. podrobnejšie opísali.

V piatej kapitole sme najskôr predstavili viackriteriálnu optimalizáciu. Potom sme podrobne opísali algoritmus šedých vlkov GWO. Na rozšírenie tohto algoritmu pre prípad viacerých účelových funkcií sme integrovali dva nové komponenty: archív a stratégiu výberu lídra. Vytvorili sme tak viac účelový algoritmus šedých vlkov MOGWO. Pre tento algoritmus MOGWO sme naprogramovali aj novú funkciu v R, ktorú sme uložili v elektronickej prílohe a porovnávali sme jej výsledky s už existujúcou funkciou v MATLABE [11]. Veľkou výhodou našej funkcie v R oproti funkcii v MATLABE je najmä rýchlosť nájdenia Pareto optimálnych riešení.

Záver práce je venovaný implementácii MOGWO algoritmu pre viackriteriálnu optimalizáciu na reálnych trhových dátach. Algoritmus MOGWO sme použili na kalibráciu, ktorá berie do úvahy nielen priebeh krátkodobej úrokovej miery, ale aj fitovanie výnosových kriviek. Za 4.38 sekundy sme našli 94 Pareto optimálnych riešení.

Zoznam použitej literatúry

[1] Beale function, dostupné na internete (1.1.2021): http://benchmarkfcns.xyz/benchmarkfcns/bealefcn.html

[2] Castro, L. N., Von Zuben F. J.: *Learning and Optimization Using the Clonal Selection Principle,* IEEE transactions on evolutionary computation, Vol. 6 (2002), pp. 239-250

[3] Cox, J., Ingersoll, J., Ross, S.: *A Theory of the Term Structure of Interest Rates,* Econometrica, Vol. 53 (1985), pp. 385-408

[4] Current Interest Rates, dostupné na internete (1.1.2021): https://www.nerdwallet.com/blog/mortgages/current-interest-rates/

[5] Das, S., Suganthan, P. N.: *Differential evolution: A survey of the state-of-the-art,* IEEE transactions on evolutionary computation, Vol. 15 (2011), pp. 4-31

[6] Elbeltagi, E., Hegazy, T., Grierson, D.: *A modified shuffled frog-leaping optimization algorithm: applications to project management*, Structure and Infrastructure Engineering, Vol. 3 (2007), 53-60

[7] EURIBOR Rates, dostupné na internete (1.1.2021): https://www.emmi-benchmarks.eu/euribor-org/euribor-rates.html

[8] Gandomi, A. H., Alavi, A. H.: *Krill herd: a new bio-inspired optimization algorithm,* Commun Nonlinear Sci Numer Simulat, Vol. 17 (2012), pp. 4831-4845

[9] Gandomi, A. H., Yang, X.-S., Talatahari, S., Alavi, A. H.: *Metaheuristic Algorithms in Modeling and Optimization,* Metaheuristic Applications in Structures and Infrastructures, (2013), pp 1-24

[10] Geem, Z. W., Joong H. K., Loganathan G. V.: *A new heuristic optimization algorithm: harmony search*, Simulation, Vol. 76 (2001), pp 60-68

[11] Grey Wolf Optimizer, dostupné na internete (13.4.2021): https://seyedalimirjalili.com/gwo

[12] Hadidi, A., Kazemzadeh Azad S., Kazemzadeh Azad S. : *Structural optimization using artificial bee colony algorithm,* Department of Civil Engineering, University of Tabriz, Iran, 2006, dostupné na internete (14.1.2020): http://www.dem.ist.utl.pt/engopt2010/Book_and_CD/Papers_CD_Final_Version/pdf/01/01564-01.pdf

[13] Hatamlou, A.: Black hole: *A new heuristic optimization approach for data clustering,* Information Sciences, Vol. 222 (2013), pp 175-184

[14] Chattová, S.: *Kalibrácia konvergenčného modelu úrokových mier Vašíčkovho typu,* diplomová práca, FMFI UK, 2013, dostupné na internete (1.1.2020): http://www.iam.fmph.uniba.sk/studium/efm/diplomovky/2013/chattova/diplomovka.pdf

[15] Chu, S. Ch., Tsai, P., Pan, J. S.: *Cat Swarm Optimization,* In Pacific Rim international conference on artificial intelligence, 2006, 854-858, Springer, Berlin, Heidelberg

[16] Kladívko, K.: *Maximum Likelihood Estimation of the Cox-Ingersoll-Ross process in Matlab*, Department of Statistics and Probability Calculus and Debt Management Department, Ministry of Finance of the Czech Republic, 2021 dostupné na internete (13.4.2021): https://www.mathworks.com/matlabcentral/fileexchange/37297-maximum-likelihood-estimation-of-the-cox-ingersoll-ross-process-the-matlab-implementation

[17] Konak, A., Coit, D. W., Smith, A. E.: *Multi-objective optimization using genetic algorithms: A tutorial,* Reliability Engineering and System Safety, Vol. 91 (2006), pp. 992–1007

[18] Matyas function, dostupné na internete (1.1.2021): http://benchmarkfcns.xyz/benchmarkfcns/matyasfcn.html

[19] Metaheuristic Optimalization, dostupné na internete (1.1.2021): https://rdrr.io/cran/metaheuristicOpt/man/

[20] Mirjalili, S.: *Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems*, Neural Computing and Applications, Vol. 27 (2016), pp. 1053-1073

[21] Mirjalili, S.: Moth-flame optimization algorithm: *A novel nature-inspired heuristic paradigm*, Knowledge-Based Systems, Vol. 89 (2015), pp. 228-249

[22] Mirjalili, S.: *SCA: A Sine Cosine Algorithm for solving optimization problems,* Knowledge-Based Systems, Vol. 000 (2016), pp. 1-14

[23] Mirjalili, S.: *The Ant Lion Optimizer*, School of Information and Communication Technology, Griffith University, Advances in Engineering Software, Vol. 83 (2015), pp. 80-98

[24] Mirjalili, S., Mirjalili, S. M., Lewis, A.: *Grey Wolf Optimizer,* Advances in Engineering Software, vol. 69 (2014), pp. 46-61

[25] Mirjalili, S., Saremi, S., Mirjalili, S. M., Coelho, L. D. S.: *Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization*, Expert Systems With Applications, Vol. 47 (2016), pp. 106-119

[26] Pareto optimal solutions, dostupné na internete (13.4.2021): https://www.autodesk.com/autodesk-university/ru/node/825

[27] Poli, R., Kennedy, J., Blackwell, T.: *Particle swarm optimization*, Department of Computing and Electronic Systems, University of Essex, Essex, 2007, dostupné na internete (14.1.2020): http://bee22.com/resources/Poli%202007.pdf

[28] Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: *GSA: A Gravitational Search Algorithm,* Information Sciences, Vol. 179 (2009), pp. 2232-2248

[29] Rastrigin function, dostupné na internete (1.1.2021): http://benchmarkfcns.xyz/benchmarkfcns/rastriginfcn.html

[30] Saremi, S., Mirjalili, S., Lewis, A.: *Grasshopper Optimisation Algorithm: Theory and application,* Advances in Engineering Software, Vol. 105 (2017), pp. 30-47

[31] Seyedali, M., Andrew, L.: *The Whale Optimization Algorithm*, Advances in Engineering Software, Vol. 95 (2016), pp. 51-67

[32] Ševčovič D., Stehlíkova B., Mikula K.: Analytické a numerické metódy oceňovania finančných derivátov, Nakladateľstvo STU 2009, Bratislava.

[33] Three-Hump Camel function, dostupné na internete (1.1.2021): http://benchmarkfcns.xyz/benchmarkfcns/threehumpcamelfcn.html

[34] Yang, X.-S.: *A New Metaheuristic Bat-Inspired Algorithm,* Nature Inspired Cooperative Strategies for Optimization, Vol. 284 (2010), pp. 65-74

[35] Yang, X.-S.: *Firefly Algorithms for Multimodal Optimization,* Stochastic Algorithms: Foundations and Applications, Vol. 5792 (2009), pp. 169-178

[36] Yang, X.-S.: *Nature-Inspired Metaheuristic Algorithm Second Edition,* Luniver Press, Frome, BA11 6TT, United Kingdom, 2010

[37] Yang, X.-S., Deb, S.: *Cuckoo Search via L´evy Flights,* Proc. of World Congress on Nature & Biologically Inspired Computing, (2009), pp. 210-214

DODATOK A

Zdrojové kódy

MLE_vasicek

```
## rovnica (12) z kapitoly 3.2.
      MLE_vasicek = function(parametre){
       kappa = parametre[1]
       theta = parametre[2]
       sigma = parametre[3]
       n = length(r)
       r1 = r[1:length(r)-1]
       r2 = r[2:length(r)]

       ni = exp(-kappa*dt)
       v_2 = (sigma^2*(1-exp(-2*kappa*dt)) )/(2*kappa)
       ll = (n-1)/2*log((2*pi*v_2)) + (1/(2*v_2))*sum((r2-ni*r1-theta*(1-ni))^2)
return(ll)
}
```

MLE_CIR

```
## rovnica vytvorená podľa kapitoly 3.4.
      MLE_CIR=function(parametre){
      kappa_odhad = parametre[1]
      theta_odhad = parametre[2]
      sigma_odhad = parametre[3]

      n = length(r)
      r1=r[1:(n-1)]
      r2=r[2:n]

      lambda = (4*kappa_odhad*(r1)*exp(-kappa_odhad*dt))/
```

```
                    (sigma_odhad^2*(1-exp(-kappa_odhad*dt)))
        N = 4*kappa_odhad*theta_odhad/sigma_odhad^2
        c = (2*kappa_odhad)/(sigma_odhad^2*(1-exp(-kappa_odhad*dt)))
        f = 2*c*dchisq(2*c*r2, df=N, ncp = lambda
        ll =  -sum (log(f))
return(ll)
}
```

fobj

```
## dvojica ucelovych funkcii pouzita na aplikaciu
## viackriterialnej optimalizacie pomocou MOGWO
fobj = function(parametre){

        kappa = parametre[1]
        theta = parametre[2]
        sigma = parametre[3]

        ## 1.funkcia - povodny likelyhood pre CIR
        n = length(r_short_rate)
        r1 = r_short_rate[1:(n-1)]
        r2 = r_short_rate[2:n]

        lambda = (4*kappa*(r1)*exp(-kappa*dt))/(sigma^2*(1-exp(-kappa*dt)))
        N = 4*kappa*theta/sigma^2
        c = (2*kappa)/(sigma^2*(1-exp(-kappa*dt)))
        f = 2*c*dchisq(2*c*r2, df=N, ncp = lambda)
        ll =  -sum (log(f))

        ## 2.funkcia - výnosy
        lambda_vynosy = parametre[4]
        r_vynosy = r # 12 month
```

```
        tau = 1

        psi = kappa + lambda_vynosy*theta
        fi = sqrt(psi^2+2*sigma^2)

        B = 2*(exp(fi*tau)-1) / ( (psi+fi)*(exp(fi*tau)-1)+2*fi )
        A = (  2*fi*exp((fi+psi)*tau/2) / ( (psi+fi)*(exp(fi*tau)-1)+2*fi )   )
              ^(2*kappa*theta/(sigma^2))

        ## P = A*exp(-B*r) -> lnP = lnA - B*r -> R = -lnP/tau = -lnA/tau + B/tau * r
        R_diff =  sum((r_vynosy -  (-log(A)/tau + B/tau * r_short_rate) )^2 )

cbind(ll,R_diff)
}
```

balik algoritmov

```
ABC (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar)
ALO (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar)
BA (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar,
      maxFrequency, minFrequency, gama,alphaBA)
BHO (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar)
CLONALG (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar,
              multipicationFactor, hypermutationRate)
CS (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar,
      abandonedFraction)
CSO (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar,
       mixtureRatio, tracingConstant, maximumVelocity, smp = as.integer(20),
       srd = 20, spc = TRUE)
DA (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar)
DE (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar,
      scalingVector, crossOverRate, strategy = "best 1")
```

FFA (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar,
B0, gamma, alphaFFA)

GA (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar, Pm, Pc)

GBS (fun, optimType="MIN", numVar, numPopulation,maxIter, rangeVar,
gravitationalConst = max(rangeVar), kbest)

GOA (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar)

GWO (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar)

HS (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar,
PAR, HMCR, bandwith)

KH (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar,
maxMotionInduced, inertiaWeightOfMotionInduced, psilon,foragingSpeed,
inertiaWeightOfForagingSpeed, maxDifussionSpeed, constantSpace, mu)

MFO (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar)

PSO (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar,
Vmax, ci, cg, w)

SCA (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar)

SFL (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar)

WOA (fun, optimType="MIN", numVar, numPopulation, maxIter, rangeVar)