VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

ÚSTAV SOUDNÍHO INŽENÝRSTVÍ
INSTITUTE OF FORENSIC ENGINEERING

ODBOR INŽENÝRSTVÍ RIZIK
DEPARTMENT OF RISK ENGINEERING

OPTIMALIZAČNÍ MODELOVÁNÍ RIZIK VE STRATEGICKÝCH APLIKACÍCH
OPTIMIZATION RISK MODELLING IN STRATEGIC APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Marek Kovalčík**
AUTHOR

VEDOUCÍ PRÁCE — **RNDr. Pavel Popela, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
ÚSTAV SOUDNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Student:	**Bc. Marek Kovalčík**
Studijní program:	Řízení rizik technických a ekonomických systémů
Studijní obor:	Řízení rizik technických systémů
Vedoucí práce:	**RNDr. Pavel Popela, Ph.D.**
Akademický rok:	2020/21
Ústav:	Odbor inženýrství rizik

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Optimalizační modelování rizik ve strategických aplikacích

Stručná charakteristika problematiky úkolu:

Student se seznámí s problematikou řešení pokročilých optimalizačních úloh. Zaměří se na modelování problémů správného rozhodování ve vybraných strategických aplikacích. Důraz bude kladen zejména na stochastické aspekty řešených úloh a související modelování rizik. Při modelování a řešení úloh s testovacími i reálnými daty budou využity vhodné softwarové nástroje. Student se z pohledu řešitelnosti uvažovaných úloh bude zabývat modifikací optimalizačních modelů, studiem jejich vlastností a jejich transformacemi. Důraz bude kladen na vhodný výběr a úpravy algoritmů a efektivní softwarové implementace. Předpokládá se zapojení studenta do řešení optimalizačních úloh vybraných výzkumných projektů.

Cíle diplomové práce:

1. Studium problematiky optimalizačních úloh.
2. Studium problematiky vybraných strategických aplikací.
3. Studium vlastností a transformací vybraných modelů matematického programování pro zvolenou třídu aplikačních úloh.
4. Výběr algoritmů pro studované modely.
5. Efektivní softwarová implementace modelů a algoritmů.
6. Testovací výpočty, případná aplikace na reálná data vybraného problému.

Seznam doporučené literatury:

Kall, P., Wallace, S. W.: Stochastic Programming, Wiley, 1994.

Birge,J., Louveaux F.: Introduction to Stochastic Programming. 2nd edition, Springer Verlag, 2011.

Klapka, J. a kol.: Metody operačního výzkumu, Brno 2000.

ABSTRAKT

Hlavním cílem této diplomové práce je navrhnout a efektivním způsobem implementovat framework pro podporu optimalizačního modelování. Důraz je zde kladen na vícestupňové stochastické optimalizační úlohy a provádění výpočtů na velkých datech. Výpočetní jádro používá systém GAMS, a s vužitím jeho aplikačního rozhraní a programovacího jazyku Python, bude uživatel schopen efektivně získávat a zpracovávat vstupní i výstupní data. Oddělením datové a aplikační logiky se potom nabízí široké možnosti testování a experimentování s obecným modelem na dynamicky se měnících vstupních datech. Součástí práce je také zhodnocení komplexnosti užití navrženého frameworku a hodnocení jeho výkonnosti, kdy byl měřen čas nutný k dokončení požadované úlohy pro různé případy použití, na zvyšující se velikosti vzorku vstupních dat.

KLÍČOVÁ SLOVA

Matematické modelování, optimalizace, stochastické programování, řízení rizik, GAMS, Python, API, optimalizační framework

ABSTRACT

The aim of this diploma thesis is to design and effectively implement a framework to support optimization modelling. Emphasis is placed on multistage stochastic optimization problems and performing calculations on large data. The computing core uses the GAMS system and with using its application interface and Python programming language, the user will be able to efficiently acquire and process input and output data. The separation of the data logic and the application logic then offers a wide range of options for testing and experimenting with a general model on dynamically changing input data. The thesis is also focused on an evaluation of the framework complexity. The framework performance was evaluated by measuring the time required to complete the required task for various use cases, on the increasing sample size of input data.

KEYWORDS

Mathematical modeling, optimalization, stochastic programming, risk management, GAMS, Python, API, optimization framework

KOVALČÍK, Marek. *Optimalizační modelování rizik ve strategických aplikacích*. Brno, 2021, 81 s. Diplomová práce. Vysoké učení technické v Brně, Ústav soudního inženýrství, Odbor inženýrství rizik. Vedoucí práce: RNDr. Pavel Popela, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Optimalizační modelování rizik ve strategických aplikacích“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu RNDr. Pavlu Popelovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Děkuji také své rodině za podporu při studiu, bez které bych práci nebyl schopen dokončit.

Obsah

Seznam symbolů, veličin a zkratek

$\mathbf{Z}$	Účelová funkce optimalizačního modelu
Z_{WS}	Účelová funkce optimalizačního modelu wait-and-see
Z_{HN}	Účelová funkce optimalizačního modelu here-and-now
$\mathbf{c}$	Známý vektor koeficientů deterministické části optimalizační úlohy
$\mathbf{x}$	Proměnná optimalizačního modelu
$\mathbf{y}$	Proměnná druhé fáze optimalizačního modelu
$\mathbf{a}$	Index v matici omezení A
$\mathbf{b}$	Vektor levé strany omezení optimalizačního modelu
$\mathbf{A}$	Matice pravé strany omezení optimalizačního modelu
$\mathbf{m}$	Počet řádků matice omezení A
$\mathbf{n}$	Počet sloupců matice omezení A
ξ	Realizace náhodné proměnné
$\mathbf{p}$	Pravděpodobnost realizace scénáře
$\mathbf{s}$	Index scénáře
$\mathbf{S}$	Počet všech scénářů
E_{ξ}	Střední hodnota generovaných vektorů náhdoné veličiny

Úvod

Tato práce se zaobírá technikami použitými v oblasti řešení úloh matematického optimalizačního modelování. V matematickém programování modelujeme a řešíme úlohy definované matematickými konstrukcemi a předpisy, které následně přepisujeme do algoritmických modelů a hledáme požadované výsledky. Principem matematických optimalizačních metod je minimalizovat riziko spojené s rozhodováním o modelovaných problémech. V této práci se zaměříme zejména na lineární a stochastické programování. Budou zde popsány principy obou těchto metod, jejich podobnosti i odlišnosti. V základním principu lineárního programování se jedná o řešení soustavy rovnic či nerovnic, kdy hledáme optimální výsledek. Optimálním výsledkem může být minimální či maximální nalezená hodnota modelované účelové funkce, a to na základě typu dané úlohy. Můžeme hledat například maximální zisk při určitém rozložení pracovníků ve firmě či například minimální časovou náročnost průběhu určitého procesu. Větší důraz zde bude kladen na stochastické programování, které se od klasického lineárního liší zejména zavedením pravděpodobnostních proměnných do matematického modelu. S tímto přístupem můžeme dále uvažovat různé scénářové modely, které budou více popsány v teoretické části diplomové práce. Popsány zde také budou různé modelovací přístupy stochastického programování. S využitím wait-and-see přístupu se budeme snažit rozhodnutí o riziku učinit po realizaci náhodného vektoru a s využitím here-and-now přístupu budeme rozhodnutí uskutečňovat před realizací náhodného vektoru hodnot pro uvažované proměnné modelu. Práce bude probíhat také na dvoustupňových rozhodovacích úlohách, jejichž principy zde budou také popsány.

Následně se přesuneme do praktické části diplomové práce, jejíž hlavním cílem bude implemetace frameworku pro efektivní práci s optimalizačními modely. Bude se zde využívat existující výpočetní jádro optimalizačního softwaru GAMS a programovací jazyk Python. Software GAMS nabízí aplikační rozhraní pro integraci jeho výpočetního jádra do jiných programů. Využití této možnosti a vytvoření efektivní součinnosti s použitím programovacího jazyka Python bude hlavním cílem této práce. Vytvořený framework bude dále hodnocen z pohledu intuitivity jeho použití.

Dále bude hodnocena výkonnost použití, kdy tyto výkonostní testy budou probíhat na zvyšující se velikosti vzorku vstupních dat. Pro možnosti testování bude vytvořena knihovna několika optimalizačních modelů s různými optimalizačními přístupy. Testování bude probíhat na domácím laptopu, čímž budou sledovány možnosti efektivního použití na běžných uživatelských počítačích.

Po vyhodnocení testování implementovaného frameworku se přesuneme do poslední části této práce, která se bude zabývat jeho použitím na reálná data. Zde budeme vycházet z již implementovaného reálného modelu, který se s použitím optimalizačního frameworku budeme snažit zjednodušit a zlepšit jeho možnosti použití pro velká data. Závěreme budou popsány všechny dosažené výsledky a celkové hodnocení implementovaného frameworku pro reálná použití.

Definice problému a vymezení cílů

V této kapitole bude nejprve popsán současný stav a definován problém, který bude následně řešen v následujících částech této diplomové práce. Dále zde pak budou popsány cíle řešení, kterých má být v diplomové práci dosaženo.

Současný stav

Jak bylo naznačeno v úvodu, diplomová práce se zabývá studiem vlastností a transformací vybraných modelů matematického programování. Běžný postup řešení optimalizačních problémů zahrnuje vytvoření matematického modelu a jeho následné otestování. Validace a verifikace probíhá často na malých datech v prostředí běžných tabulkových procesorů, kterým je například MS Excel. Pro komplexnější úlohy a testování na velkých datech se využívá například optimalizační matematický software Gams. Ten nabízí mnohem vhodnější možnosti práce s těmito modely. Jedná se o komerční software, jehož výpočetní jádro je pro řešení úloh matematického programování dobře odladěné.

Kámen úrazu nastává, cheme-li využít software Gams pro velmi náročné výpočetní úlohy s požadavky na komplexní zpracování vstupních dat či export výsledků. K naplnění těchto cílů je třeba zapojit do optimalizačního procesu další prvky. Při použití běžných optimalizačních prostředků je zpracování vstupních dat náročnou až nemožnou operací, stejně jako export výsledků do komplexnějších souborových struktur.

Vytvořit jednoduchý matematický model, jednoúčelový optimalizační model v softwaru Gams a naplnit jej statickými daty není přílíš náročná operace. Pokud bychom chtěli mít možnost zpracovávat vstupní data, ba dokonce využít moderní postupy dolování dat a informací z různých médií a tato data importovat do optimalizačního modelu, celkový proces by se rapidně zkomplikoval a museli bychom využít i další nástroje.

Cíle řešení

Součástí diplomové práce byl průzkum dostupného optimalizačního software a možností jeho použití. Hlavním cílem je efektivní softwarová implementace frameworku, která bude umožňovat komplexní práci s optimalizačními modely a efektivní zpracování vstupních i výstupních dat.

Naplnění cílů této diplomové práce předchází studium problematiky optimalizačních úloh a vybraných strategických aplikací. Následně jsem se zabýval studiem vlastností a transformací vybraných modelů matematického programování, zejména více stupňovými stochastickými úlohami. Právě pro tento typ úloh bylo cílem navrhnout a implementovat řešení, které usnadní uživateli práci s realizací výpočtů.

Součástí implementovaného frameworku musí být možnost získávat vstupní data z různých zdrojů, jako jsou například veřejné portály, textové soubory nebo různé databáze. Dále musí umožňovat přípravu vstupních dat, která zahrnuje jejich čištění a validaci. Je žádoucí aby aplikační jádro pracovalo s vhodnými aplikačními rozhraními odladěných výpočetních programů. Důraz bude také kladen na široké možnosti exportu a vizualizace dosažených výsledků.

Dalším cílem bude důkladné otestování a verifikace získáváných výsledků oproti standardnímu použití optimalizačních metodik a postupů. V této části se očekává také zhodnocení efektivnosti implementace. Ve fázi testování budou vytvořeny testovací scénáře různé velikosti vzorku vstupních dat. Finální částí diplomové práce bude pak aplikace implementovaného frameworku na reálná data a reálné modely s vyhodnocením dosažených výsledků a vyhodnocením efektivity použití frameworku na různých velikostech vzorku vstupních dat.

1 Teoretická část

V této části dokumentace k diplomové práci budou popsány použité metody a nástroje, které jsem při práci využíval. Nejprve zde budou popsány principy matematického programování, konkrétně lineárního programování, deterministického a stochastického programování. Bude zde také popsán pricnip více stupňových optimalizačních úloh. Následovat bude popis použitých nástrojů, optimalizačního softwaru GAMS[1] a programovacího jazyka Python[2].

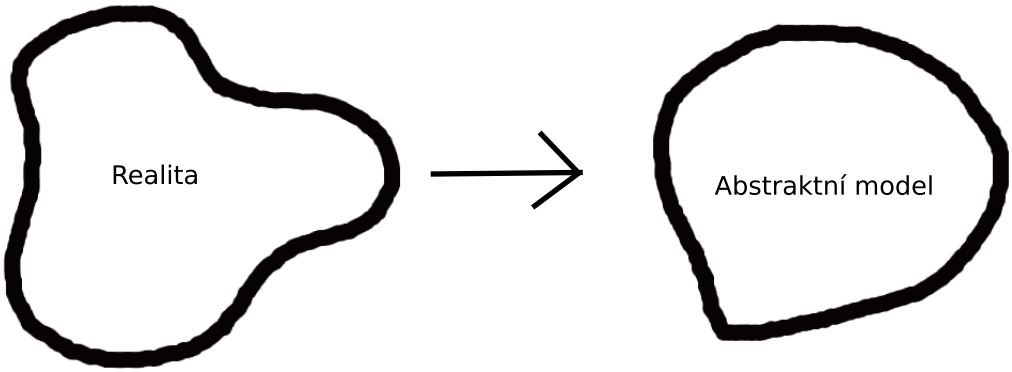
1.1 Matematické modelování a programování

Matematické modelování je proces vytvoření abstraktního modelu reálného problému, zapsaného formou matematického zápisu. Jedná se o zjednodušený obraz reálného jevu, ve kterém jsou zanedbávány některé jeho aspekty. Takový model využívá matematického zápisu k popisu množiny vstupních a výstupních proměnných, parametrů a matematických struktur, které určují stavy a vazby mezi prvky tohoto systému. Na obrázku 1.1 je symbolicky znázorněn proces zanedbávní detailů z reálného jevu při vytváření abstraktního modelu. Správné a detailní vytvoření matematického modelu je důležitým vstupním krokem matematického programování.

Součástí vytváření matematického modelu je převedení reálného problému na matematické formulace. Takový model obsahuje pak nadefinovou účelovou funkci a omezující podmínky, které mají podobu lineárních či nelineárních nerovnic. Velmi důležitou částí matematického programování jsou také kvalitní vstupní data, která vychází z reálných poznatků, experimentů, měření nebo například expertních odhadů. Typickými úlohami matematického programování jsou optimalizace alokalace omezeného množství prostředků tak, abychom například odsáhli minimaliních nákladů či maximálního zisku.

Konkrétním příkladem můžeme být velmi dobře známá farmářova úloha. Představme si farmáře, který se musí rozhodnout kolik které plodiny zasadí na svá pole.

[1] GAMS - The general algebraic modeling systom, https://www.gams.com/
[2] Python - programovací skriptovací jazyk, https://www.python.org/



Obr. 1.1: Proces vytvoření abstraktního modelu

(Zdroj: Vlastní zpracování)

Každá z plodin má různou pořizovací cenu a může jich být zasazen jen určitý počet na určitou plochu. Toto jsou vstupní parametry modelu. Dále pak farmář nemá k dispozici neomezený prostor k sadbě a neomezené množství plodin. Takto můžeme mít definované omezující podmínky modelu. Farmář chce maximalizovat zisk z prodaných vypěstovaných plodin. Součástí účelové funkce je zisk z prodaných vypěstovaných plodin na farmářském trhu. Cílem takové optimalizační úlohy je tedy najít takové rozložení pěstování plodin, aby se maximalizoval farmářův zisk. [1]

Matematické modelování a programování je úzce sjpato s inženýrstvím rizik. Hledáním extrémů nadefinovaného matematického modelu hledáme takové rozložení vstupních veličin, které nám vrátí nejoptimálnější hodnotu účelové funkce, tj. takové rozložení vstupních veličin, kdy jsou například rizika vysokých výdajů či nízkého zisku minimální. [2]

1.1.1 Riziko

Ačkoliv je v posledních letech vynakládáná velká snaha pro sjednocení termínů v oblasti inženýrství rizik, lze stále v různých literaturách najít trochu odlišné definice pojmu riziko, zejména v anglické literatuře, kde je tento termín často nepsrávně zaměňován s pojmem nebezpečí. Historicky tento pojem pochází z italského "risico", což v překladu znamená úskalí. Vznik tohoto slova můžeme přisuzovat italským námořníkům, kteří jej začali používat pro nebezpeční nárazu na skálu. Zanedlouho se tento pojem začal používat i v jiných průmyslových odvětvích, v začátcích zejména v kontextu s ekonomikou a podnikáním. V dnešní době se tento pojem dostal do všech odvětví lidské průmyslu a zvažování rizik je na denní bázi každého člověka.

Velmi obecnou definici termínu riziko lze nalézt v publikaci profesora Janíčka. "Riziko je pravděpodobnost vzniku nestandardního stavu konkrétní entity v daném čase a prostoru."- prof. Janíček. V tomto smyslu uvažujeme entitu jako zdroj nebezpečí. Nestandardním stavem entity v čase a prostoru uvažujeme takový stav, který není v příustných mezích. Tato definice je velmi elegantní. Pokud ji aplikujeme například na nějaký ekonomický problém, máme zde definovány obě strany potenciálních rizik, zisk či ztráta. [3]

$$Riziko = Pravděpodobnost \times Dopad \tag{1.1}$$

V rovnici 1.1 lze vidět nejpoužívanější předpis vyhodnocení rizika. Tato rovnice nám říká, že riziko je definované jako relace mezi pravděpodobností nastání určité situace a jejím možným dopadem. Možný dopad události může mít pozitivní i negativní charakter. Riziko ze své podstaty nemůže být nikdy nulové, jelikož rizika nelze nikdy úplně elimininovat. Lze je pouze minimalizovat na akceptovatelnou úroveň. [4] [5]

1.1.2 Lineární programování

Lineární programování je součástí širšího okruhu disciplín, označovaného jako matematické programování. Jeho úlohou je lineární optimalizace, což je jeden ze základních problémů teorie optimalizace. Hledáme takové řešení účelové funkce, které

spadá do množiny přípustných řešení. Přípustné řešení se nazývá optimální, jestliže hodnota účelové funkce v tomto bodě nabývá svého maxima, respektive minima. Obecně můžou při řešení nastat tři možnosti. Zadaná úloha nemá přípustné řešení, když nejsme schopni splnit všechny omezující podmínky najednou. Druhou možností je, že bylo nalezené řešení, které není optimální. Poslední možností je nalezení optimální řešení.

Účelová funkce Z má tvar součtu součinů známých koeficientů c_i, což může být například zisk z prodané vypěstované plodiny farmářova modelu, a aktuálních hodnot proměnných analyzovaného modelu x_i, což může být například aktuální sledované množství konkrétní plodiny. Předpokládejme, že v modelu máme n proměnných, a hledáme minimální hodnotu účelové funkce. Pak obecný tvat účelové funkce je popsán následující rovnicí.

$$min\ Z = \sum_{i=0}^{n} c_i \times x_j \tag{1.2}$$

Dále je nutné stanovit omezující podmínky. Symbolicky lze tyto podmínky popsat následující soustavou nerovnic. Koeficienty a_{mn} jsou tvoří pravou stranu omezující matice a koeficienty b_m tvoří vektor pravé strany, obsahující minimální požadavkům omezení. Index m nám označuje počet omezení.

$$\begin{array}{cccc} a_{11}x_1 & a_{12}x_2 & \dots & a_{1n}x_n = b_1 \\ \vdots & \vdots & & \vdots \\ a_{m1}x_1 & a_{m2}x_2 & \dots & a_{mn}x_n = b_m \\ x_1 & x_2 & \dots & x_n \geq 0 \end{array}$$

$$\mathbf{A}x = \mathbf{b} \tag{1.3}$$

Pro nalezení optimální řešení je důležité aby platilo, že $m < n$. Z tohoto symbolického zápisu si pak následně vytvoříme matici $\mathbf{A}$ a vektor $\mathbf{b}$, které budou dále využity při procesu optimalizace. [1] [6]

1.1.3 Stochastické programování

V předešlé kapitole bylo rozebírán deterministický přístup k optimalizační úloze. Při řešení jsme disponovali úplnou znalostí všech použitých parametrů. Takový přístup se hodí pro demonstraci základních principů, ale pro řešení reálných problémů vhodný není. V reálném světe hraje velkou roli neurčitost a náhoda při definici parametrů úlohy. Ve stochastickém programování počítáme s pravděpodbností výskystu určitého jevu. Tento přístup tedy lépe reflektuje reálný svět a výsledky z něj plynoucí jsou mnohem relevantnější. Představme si opět příklad s úlohou farmářova problému. Pokud budeme uvažovat, že vliv na jeho zisk má počasí, tj. zda-li bude například celoroční sucho, přemíra deštů nebo naopak ideální podmínky, mluvíme již o pravděpodobnostní úloze, jelikož můžeme odhahovat, že takové počasí nastane s určitou pravděpodobností.

Obecný předpis stochastické úlohy je definován následujícím výrazem. Zde lze vyčíst, že se bude hledat minimum účelové funkce f. Dále ξ je vektor náhodných proměnných, jejich pravděpodobnostní rozdělení je zpravidla nezávislé na vektorové proměnné x.[7]

$$min\, f(x, \xi) | x \in C(\xi) \tag{1.4}$$

$$\xi = (\xi_1, ..., \xi_k) \tag{1.5}$$

Při řešení stochastických úloh je důležité rozlišovat dva základní přístupy. Tyto přístupy jsou známé jako Wait-and-see a Here-and-now. Rozdíl mezi nimi je v rozlišování okamžiku realizace rozhodnutí.

Wait-and-see

Wait-and-see přístup přistupuje k rozhodovací úloze tak, že okamžik rozhodnutí následuje až po realizaci náhodného parametru. Rozhodnutí x je vyvozeno ze výsledku ξ a stává se z něj funkce náhodného vektoru $x(\xi)$. Na základně tohoto pozorování můžeme dále provádět další kroky analýzy modelu a jelikož další rozhodování nastává až po realizaci toho prvnotního, dá se na wait-and-see model nahlížet jako na deterministický model. Dá se říct, že wait-and-see model provádí analýzu za nejistoty a rozhodnutí s jistotou. Tento přístup má svůj význam zejména pro dlouhodobé plánování a matematické statistice (regresní analýzy). [8]

Here-and-now

Při použití Here-and-now přístupu se rozhodnutí uskutečňuje před realizací náhodného vektoru. Pokud se zamyslíme nad rozhodovacími problémy v praxi, je tento model realitě blíže, jelikož většinou neznáme průběh budoucí akce dříve, než-li je třeba vykonat patřičné rozhodnutí. Vektor x, který představuje rozhodnutí, je v tomto případě stejný pro všechny budoucí realizace ξ. [8]

EVPI, EEV a VSS charakteristiky

Definovali jsem si různé deterministické i nedeterministické metody založené na různých přístupech a s různým pojetím náhodných veličin. Charakteristika EVPI určuje, jestli je výhodné zaplatit za předčasnou znalost budoucí realizace pokud je tato informace dostupná. Čím vyšší hodnotu EVPI získáme, tím více se nám predikce budoucí realizace vyplatí. EVPI je definováno podle vztahu níže, kde Z_{HN} je získaná optimální hodnota here-and-now přístupem, Z_{WS} je optimální hodnota získaná wait-and-see přístupem, $p(s)$ značí pravděpodobnost nastání jedné náhodné realizace s. [9]

$$EVPI = Z_{HN} - \sum_{s=0}^{s_{max}} p(s) \times Z_{WS}(s) \tag{1.6}$$

Stochastické procesy jsou výpočetně náročné, a proto se občas náhodný vektor ξ nahrazuje střední hodnotou. Takovému přístupu říkáme expected-value a získanou hodnotu účelové funkce značíme Z_{EV}. Tato hodnota se využívá při získání hodnoty EEV, která je definováno následujícím vztahem, kde $p(s)$ značí pravděpodobnost nastání scénáře. [9]

$$EEV = \sum_{s=0}^{s_{max}} p(s) \times Z_{HN} \tag{1.7}$$

V momentě, kdy máme k dispozici hodnotu EEV, můžeme dopočítat charakteristiku VSS. Ta nám ukazuje přínos zohlednění stochastičnosti v modelu. Pokud je hodnota VSS kladná, zohdlednění náhodnoti v modelu se vyplatí. VSS charakteristika je definována následovně.

$$VSS = EEV - Z_{HN} \tag{1.8}$$

1.1.4 Dvoustupňové úlohy

Představme si situaci, kdy rozhodovatel musí učinit dvě rozhodnutí ve dvou různých okamžicích a to pouze na základě informací dostupných v prvním časovém okamžiku. Typickým příkladem může být opět farmářova úloha kdy prvotním rozhodnutím je výběr plodiny k osázení jeho polí na základě užitné plochy, ceny surovin a tak dále. Druhé rozhodnutí, nebo-li rozhodnutí druhého stupně, nastáva v časovém okamžiku po realizaci náhodné veličiny. V případě farmářova modelu to může rozhodnutí o změně sadby, přidání či odebrání osázeného množství plodiny na základně měnícího se počasí a podmínek pro jeho práci. [1]

Uvažujme úlohu lineárního programování, která je definována následující předpi-

sem, kde x a y jsou neznámé sloupcové vektory o n složkách. Symboly c a q zastupují známé sloupcové vektory o n složkách. Symboly A, T a W jsou matice o n řádcích a m sloupcích.

$$min\{c^T x + q^T y\} \tag{1.9}$$

$$Ax \leq b \tag{1.10}$$

$$Tx + Wy = h \tag{1.11}$$

$$x \geq 0 \tag{1.12}$$

$$y \geq 0 \tag{1.13}$$

V tento moment předpokládejme, že úloha je řešitelná, tzn. že množina $\{x|Ax \leq b, x \geq 0\}$ je ohraničená množina. Nyní se pokusíme tuto úlohy rozdělit na dvě dílčí rozhodnutí, dva dílčí stupně. První stupeň rozhodovací úlohy bude vypadat následovně.

$$min\{c^T x + f(x)\} \tag{1.14}$$

$$Ax \leq b \tag{1.15}$$

$$f(x) = min\{q^T y\} \tag{1.16}$$

$$Tx + Wy = h \tag{1.17}$$

$$y \geq 0 \tag{1.18}$$

Jelikož se jedná o minimalizační úlohu lineárního programování a hledáme minimální hodnotu účelové funkce, můžeme omezit vstupní hodnoty shora. Přidejme tedy k první verzi úlohy podmínky, že $f(x) \leq 0$, která bude vstupovat jako další podmínky pro první stupeň rozhodování. Abychom mohli řešit úlohu 1.14, musíme proměnnou x pevně zafixovat a neuvažovat ji jako proměnnou, ale jako hodnotu, kterou získáme z 1.19. Řešení dvoustupňové úlohy tedy spočívá ve vzájemném poskytování vypočtených hodnot prvního a druhé stupně. [11]

$$min\{c^T x + 0\} \tag{1.19}$$

Dvoustupňové stochastické úlohy

Dvoustupňové stochastické rozhodovací úlohy využívají kombinaci here-and-now a wait-and-see přístupu. V předešlých částech byl popsán rozdíl mezi těmito dvěma přístupy. Uvažujme opět známý farmářův problém. Jelikož časový okamžik prvního rozhodnutí nastává před realizací náhodné veličiny, tj. změna počasí, můžeme říci, že první stupeň využívá here-and-now přístup. Rozhodnutí druhého stupně je prováděno až po časovém okamžiku realizace náhodné veličiny. Z výše zmíněné definice víme, že toto chování popisuje wait-and-see přístup.

Rozhodnutí v první fázi rozhodovacího procesu značíme x a rozhodnutí v druhé fázi značíme $y(\xi)$, kde ξ je vektor náhodné veličiny. Deterministická složka rozhodování je zde zastoupena $c^T x$. V definici účelové funkce nám nyní přibyla očekávaná střední hodnota druhého stupně rozhodnodování E. Hodnoty z druhého stupně rozhodování získáme pomocí $Q(x, \xi)$, což reprezentuje stochastickou složku dvoustupňové úlohy. Takto definovaná účelová funkce je zobrazena v rovnici 1.20. [shapiro]

$$min\{c^T x + E_\xi Q(x, \xi)\} \tag{1.20}$$

Podmínky zůstavají stejné. Promnné x musí obsahovat kladnou hodnotu a omezení jsou dána maticí omezení A a vektorem omezujících hodnot b. Matice A a vektor b společně definují deterministická omezení pro první fázy rozhodování.

$$Ax \leq b \tag{1.21}$$

$$x \geq 0 \tag{1.22}$$

Dále je třeba definovat výpočet v druhé fázi rozhodovacího procesu. Optimální řešení v této fázi, hodnotu $Q(x, \xi)$ získáme pomocí předpisu v rovnici 1.23.

$$Q(x, \xi) = min\{q(\xi)^T \times y(\xi)\} \tag{1.23}$$

Také ve druhé fázi rozhodovacího procesu musí být definovány omezující podmínky. Ty jsou definovány výrazy 1.24 a 1.25. Rovnice 1.24 definuje omezující podmínky, které se vážou k rozhodnutí x i $y(\xi)$ v obou stupních rozhodování.

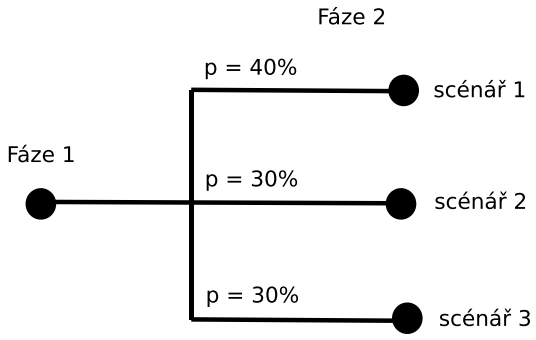
$$B(\xi)x + C(\xi)y(\xi) = d(\xi) \tag{1.24}$$

$$y(\xi) \geq 0 \tag{1.25}$$

Ve druhé fázi pokaždé generujeme vektor náhodné veličiny. Každé této realizaci můžeme přiřadit pravděpodobnost jejího nastání. Potom každé takové možnosti říkáme scénář. Pokud budeme uvažovat, že existuje S scénářů, každý obsahuje vektor náhodné veličiny ξ_s a pravděpodobnost jeho nastání je p_s, můžeme očekávánou střední hodnotu náhodné parametru zapsat jako 1.26. [9]

$$E_\xi[Q(x, \xi)] = \sum_{s=1}^{S} p_s \times Q(x, \xi_s) \tag{1.26}$$

Pro dosažení správných výsledků je důležité aby suma pravděpodobnosti všech scénářů byla rovna jedné. Tato podmínka je popsána rovnicí 1.27. Tento přístup k řešení dvoustupňových stochastických úloh se nazývá scénářový. Je to přístup



Obr. 1.2: Dvoustupňové rozhodování, scénářový model

(Zdroj: Vlastní zpracování)

zvažování alternativní budoucnosti. Pro lepší představu lze vidět grafické znázornění scénářového modelu na obrázku 1.2.

$$\sum_{s=1}^{S} p_s = 1 \tag{1.27}$$

Celá úloha se dá přepsat na 1.28. Podmínky zůstávají téměř stejné, pouze k nim přibylo rozlišení omezení dle scénářů. Omezující podmínky jsou vidět v 1.29, 1.30, 1.31 a 1.32.

$$min\{c^T x + \sum_{s=1}^{S} p_s \times q_s(\xi)^T \times y_s(\xi)\} \tag{1.28}$$

$$Ax = b \tag{1.29}$$

$$B_s(\xi)x + C_s(\xi)y_s(\xi) = d_s(\xi) \tag{1.30}$$

$$x \geq 0, y_s \geq 0 \tag{1.31}$$

$$s \in < 1; S >, S > 0 \tag{1.32}$$

1.2 Použité nástroje

V této kapitole budou stručně popsány použité nástroje při vytváření diplomové práce. Jedná se o optimalizační a modelovací software GAMS a programovací jazyk Python.

1.2.1 GAMS - General Algebraic modeling system

GAMS je vysokoúrovňový modelovací systém pro matematickou optimalizaci. Je navržen pro modelování a řešení lineárních, nelineárních a smíšených optimalizačních problémů a úloh. Systém je přizpůsoben pro použití na složitých rozsáhlých modelovacích aplikacích a umožňuje uživateli vytvářet velké udržovatelné modely, které lze přizpůsobit novým situacím. Systém je k dispozici pro použití na různých počítačových platformách. Modely jsou přenosné z jedné platformy na druhou.

Jazyk systému GAMS byl první jazyk algebraického modelování (AML) a je formálně podobný běžně používaným programovacím jazykům. Obsahuje integrované vývojové prostředí (IDE), které je připojeno ke skupině řešitelů optimalizace třetích stran.

Umožňuje uživatelům implementovat jakýsi hybridní algoritmus kombinující různé řešitele. Modely jsou popsány ve stručných algebraických výrazech čitelných člověkem. Ačkoli je původně navržen pro aplikace související s ekonomikou a vědou o řízení, má komunitu uživatelů z různých oborů inženýrství a vědy. [12] [13]

1.2.2 Programovací jazyk Python

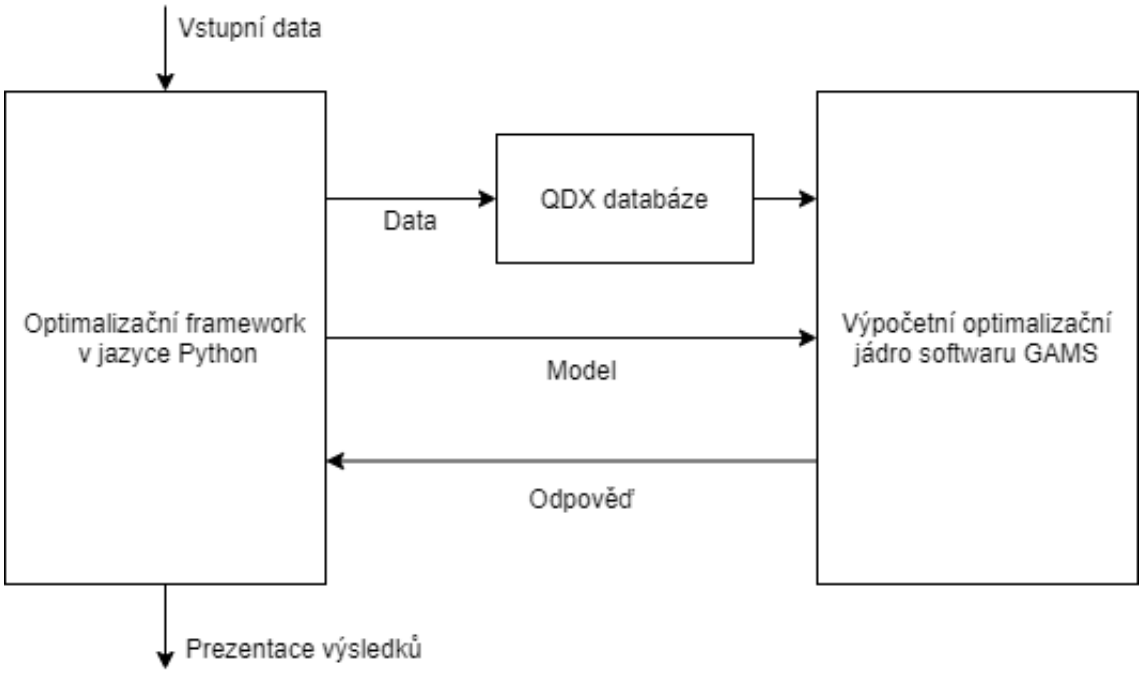
Python univerzální, vysokoúrovňový programovací jazyk. Jeho filozofie klade důraz na čitelnost kódu, také díky jeho pozoruhodnému využití velkých mezer, tabulátorů a řádkování. Jeho jazykové konstrukce a objektově orientovaný přístup mají za cíl pomoci programátorům psát jasný a logický kód pro malé a velké projekty. Jedná se o dynamicky typovaný programovací jazyk. Podporuje několik paradigmat programování, včetně strukturovaného (zejména procedurálního), objektově orientovaného a funkcionálního programování.

Nyní krátký náhled do historie tohoto jazyka. Python byl vytvořen na konci 80. let a poprvé vydán v roce 1991 zakladatelem Guido van Rossum. Python 2.0, vydaný v roce 2000, představil nové funkce, jako je práce se seznamy a systémem ničení objektů s počítáním odkazů. Ve verzi 2.7 byl ukončen v roce 2020. Python ve verzi 3.0, vydaný v roce 2008, byl hlavní revizí jazyka, který není zcela zpětně kompatibilní a mnoho kódu Pythonu 2 nepracuje beze změn na Pythonu 3. S koncem životnosti Pythonu 2, jsou podporovány pouze verze Python 3.6 a novější. Starší verze stále podporují např Windows 7, lze ho však najít také na operačních systémech založených na unixu.

1.2.3 Aplikační rozhraní Python - GAMS

Pod pojmem aplikační rozhraní si lze představit soubor dostupných funkcí, procedur, knihoven či protokolů, které operují na vnějším perimetru aplikace. Jeho hlavním cílem je přenos infromací a dat mezi systémy. Rozlišujeme více typů aplikačních rozhraní a možností jejich použití. Na obrázku 1.3 je zobrazeno schéma použití výpočetní knihovny GAMS a programovacího jazyku Python.

Řídícím modulem celého procesu je skript napsaný v Pythonu. Zde má programátor obrovské množství možností přizpůsobení, která vychází z principů programovacích jazyků. Může se jednat o předzpracovávání vstupních dat či vlastní prezentace dosažených výsledků. Aplikační rozhraní funguje na principu požadavek - odpověď. Požadavek se skládá s definice optimalizačního modelu a konkrétních dat. Data mohou být přímou součástí optimalizačního modelu, ale mohou být uložena separátně v dočasné databázi, QDX databázi. Jakmile výpočetní jádro obdrží požadavek, začne provádět definované výpočetní operace a po jejich ukončení předá odpověď zpátky řídímu programu.



Obr. 1.3: Aplikační rozhraní Python - GAMS

(Zdroj: Vlastní zpracování)

2 Vlastní řešení

V této části diplomové práce budě podrobně rozebrána praktická část, která obsahovala navržení a implementaci frameworku postaveného na využití aplikačního rozhraní mezi GAMS a programovacím jazykem Python, jeho testování, zhodnocení kladů a záporů a nakonec ukázky použití na konkrétních příkladech.

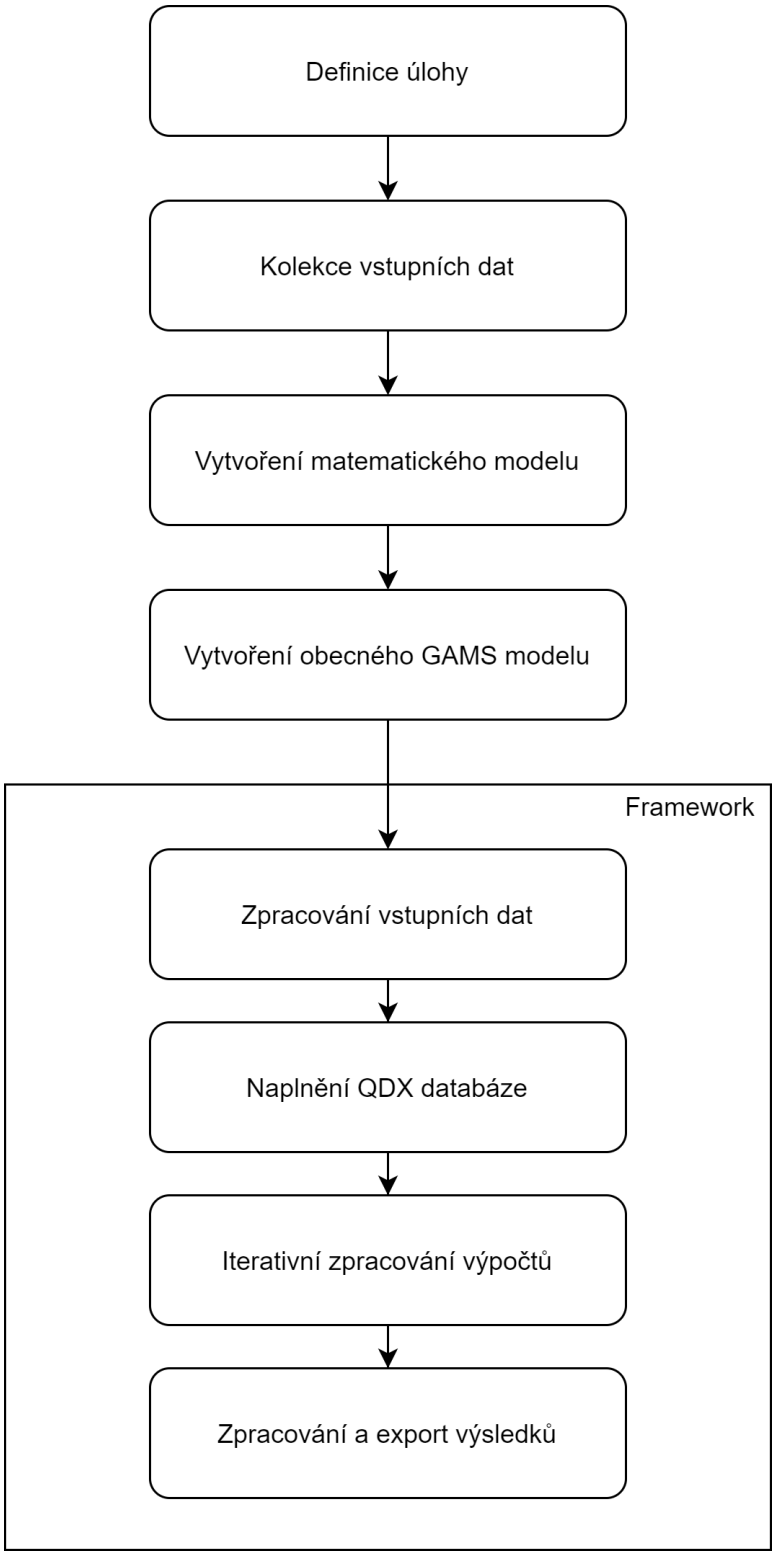
2.1 Framework pro práci s optimalizačními modely

V dalších podkapitolách bude detailně popsán návrh samotného frameworku, bude následovat popis implementace a hodnocení komplexnosti použití na různých typech úloh. Nakonec zde bude provedeno testování výkonnosti jeho použití oproti samotnému použití GAMS studia, kde bude sledován čas nutný k dokončení operace v závilosti na velikosti vzorku vstupních dat.

2.1.1 Návrh frameworku

Nejprve se podíváme na samotný návrh, který je znázorněn postupovým diagramem na obrázku 2.1. Prvním krokem, stejně jako u klasického použití matematického modelování v GAMS studiu je nutné definovat úlohu a získat data, na kterých budou výpočty prováděny. Následně je třeba vytvořit matematický model, na jehož základě vytvoříme model v GAMS. V tomto většinou provádíme testování na malých datech, abychom ověřili správnost navrženého programu.

Až do této chvíle byl postup identický s použitím GAMS studia. Nyní přichází na řadu využití pokročilých možností s využitím frameworku, což je na diagramu 2.1 vyobrazeno obdélníkem obsahujícím čtyři další procesy. Prvním z nich je zpracování vstupních dat. Né vždy dostaneme data ve správné struktuře, která by bylo přímo použitelná. Sesbíraná dala mohou obsahovat chybějící hodnoty, mohou být špatně naformátovaná či obsahovat jiné nedostatky. S pomocí struktur a nástrojů programovacího jazyka Python můžeme tato data vyčistit a předpřipravit pro další efektivní zpracování.



Obr. 2.1: Návrh frameworku

(Zdroj: Vlastní zpracování)

Aplikační rozhraní mezi Python a GAMS využívá GDX databázi, ze které jsou využívány hodnoty při výpočtu. Tato databáze je uložena v samostatném souboru. Jelikož jsou data od výpočetního modelu naprosto oddělena, je možné využívat velký rozsah dat bez nutnosti optimalizace struktury programového kódu GAMS modelu. Dalším krokem je samotné zpracování výpočtů. V případě modelu se scénářemi scénářů jsou výpočty prováděny iterativně vždy po jedné sadě scénářů. O řízení zpracování výpočtu a zaznamenání výsledků se stará jádro frameworku a tudíž odpadá nutnost vytvářet komplikovaný model v GAMS.

Celý framework funguje jako terminálová aplikace se standardním unixovým použitím. Jakmile jsou výpočty dokončeny, jsou výsledky předány uživateli na standardní výstup. Výsledky je možné exportovat do textového souboru i sešitu programu MS Excel[1].

2.1.2 Implementace frameworku

V této části diplomové práce budou popsány vybrané implementační detaily vytvořeného frameworku. Nejprve bude popsán princip fungování výpočetního jádra programu pro řízení zpracování optimalizační aplikace. Následně budou popsány dílčí části, které je nutné nastavit či doplnit pro dosažení správných výsledků.

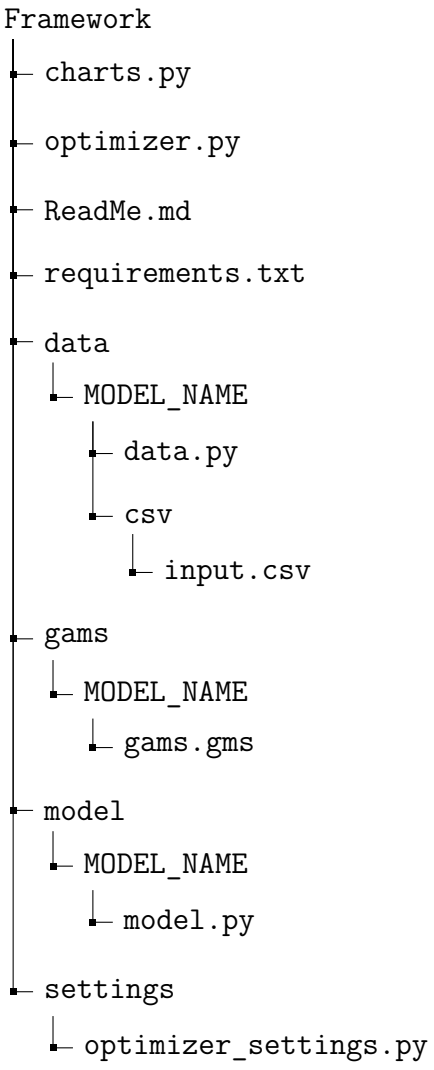
Adresářová struktura

Pro zachování přehlednosti a snadného použití frameworku při více projektech je nutné dodržovat určité konvence. Z těchto důvodu jsem navrhl adresářovou strukturu, která by měla pomoci s organizováním práce při použití tohoto frameworku. Strukturu lze vidět na schématu níže.

V kořenovém adresáři se nachází soubor ReadMe.md, který obsahuje návod k instalaci nutných částí pro správnou funkčnost a možnosti použití optimalizačního frameworku. Dále se zde nachází soubor requirements.txt, který obsahuje seznam knihoven programovacího jazyka Python nutných k instalaci. Pomocí souboru charts.py je možné vizualizovat dosažené výsledky. Posledním souborem, který se zde

[1]MS Excel - Tabulkový procesor společnosti Microsoft

nachází je optimizer.py, ve kterém se nachází implementace jádra optimalizačního frameworku.



Adresářová struktura frameworku

Implementace jádra frameworku

Jádro implementovaného frameworku s nachází v souboru optimizer.py. Implementace vychází z oficiální dokumentace k aplikačnímu rozhraní programovacího jazyka Python a výpočetního systému GAMS. Jádro je navrženo tak, aby do něj uživatel při zpracovávání nejběžnějších úloh nemusel vůbec zasahovat. Při inicializaci programu se do paměti načtou všechna potřebná data, která jsou následně ve strukturované formě předána do GDX databáze a následně do samotného zpracování GAMSem. Framework je přizpůsoben zejména pro scénářové aplikace, poradí si také s modely

využívající scénáře scénářů. Každé zpracování jednoho konkrétního scénáře probíhá v samostatném vlákně, což činí zpracování velmi efektivním. Způsob tohoto zpracování je ukázán na výpisu 2.1. V jádře jsou také implementovány možnosti exportu dosažených výsledků.

Výpis 2.1: Implementace vláknového zpracování

```
# Each scenario runs in its own thread
for scenario_data in scenarios_data:
  t = threading.Thread(target=run_scenario,
            args=(optim, data, scenario_data))
  t.start()

# waiting until all thread finishes execution
t.join()
```

Nastavení a výběr modelu ke zpracování

Při návrhu fungování tohoto frameworku byla uvažována také organizace práce s více modely. Pro každý model je vytvořen adresář označený názvem modelu ve složkách data, Gams a model. Toto schéma je zobrazeno na adresářové struktuře v předešlé kapitole. V souboru optimizer_settings.py je pak definováno, s jakým modelem má program aktuálně pracovat. Takto je docíleno oddělení dat, aplikační logiky, modelu a zároveň uživateli zůstává k dispozici přehledná adresářová struktura. V případě, že máme k dispozici dva modely, kdy první zpracovává farmářův optimalizační problém a druhý zpracovává transportní optimalizační problém, může obsah souboru s nastavením vypadat viz 2.2.

Výpis 2.2: Nastavení aktivního modelu

```
## Model farm
from data.farm.data import *
from model.farm.model import *

## Model transport
#from data.transport.data import *
#from model.transport.model import *
```

Ve výše uvedené části kódu lze vidět, že máme k dispozici dva modely. Použit je model pro farmářův optimalizační problém, což lze poznat z toho, že řádky importující vstupní data a informace o modelu jsou u něj odkomentovány.

Zpracování vstupních dat

Zpracování vstupních dat je velmi důležitou částí každého procesu, který s daty nějakým způsobem pracuje. Vstupní data mohou mít špatnou strukturu, mohou obsahovat nevalidní hodnoty nebo nemusí být kompletní. GAMS jako takový má téměř nulové možnosti validace a zpracování vstupních dat. Python má v tomhle ohledu obrovskou výhodu.

Pro zpracování vstupních data je nutné, aby uživatel měl alespoň základní znalosti v oblasti používání programovacího jazyka Python. Vstupní data mohou být definována třemi způsoby:

- Statická data - konkrétní deterministické hodnoty
- Dynamická data - konkrétní hodnoty jsou generovány s každou iterací spuštění výpočetního modelu
- Vstupní soubory - konkrétní data jsou načítána a parsována ze vstupních souborů

Načtení a zpracování vstupních dat probíhá v souboru data.py. Tento soubor obsahuje vždy minimálně dvě funkce. První z nich se jmenuje *get_data*(), ve které jsou

definovány hodnoty pro entity výpočetního modelu, které se s možnými iteracemi nemění. Příklad je demonstrován ve výpisu 2.3.

Výpis 2.3: Definice deterministických dat

```
def get_data():                                                    1
  data = dict()                                                    2
  data['a'] = 13                                                   3
  data['b'] = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]              4
  data['s'] = sum(1 for line in open('data.csv'))                  5
  return data                                                      6
```

Ve výpisu 2.3 jsou definovány tři parametry, které budou následně načteny a použity ve výpočetním modelu GAMS. Uživatel má v tento moment k dispozici mnoho možností, jak může vstupní data definovat. U parametru *a* je definována pouze jedna celočíselná hodnota, z čehož můžeme usuzovat, že se ve výpočetním modelu bude jednat pravděpodobně o skalár. Parametr *b* je definován jako seznam celočíselných hodnot, který se ve výpočetním modelu může použít například pro definici množiny. Parametr *s* bude obsahovat hodnot, která definuje počet řádků ve vstupním souboru *data.csv*.

Druhá funkce, která v souboru *data.py* musí být vždy je funkce *get_scenarios_data*(). V ní jsou definována vstupní data pro jednotlivé scénáře. Je možné využívat i další funkce, které pomohou s parsováním vstupních souborů. Tento princip je ukázán ve výpisu 2.4, kde je zobrazen kód funkce *get_scenarios_data*().

Výpis 2.4: Definice stochastických dat

```
def get_scenarios_data():                                          1
  scenarios = parse(input_file='scenarios.csv')                    2
  return [[{"index": 1, "xi": scenarios}]]                         3
```

Zde si uživatel může vyhrát s přípravou dat. Data pro jednotlivé scénáře jsou definována v souboru *scenarios.csv*, který je zpracován funkcí *parse*(). Tato funkce obstarává celou logiku přípravy vstupních dat. Může se jednat o nějakou externí funkci přídatné knihovny, vlastní implementaci parsovací funkce či například funkci načítající data z jiného aplikačního rozhraní.

Důležité je, aby v souboru *data.py* byly pro každý výpočetní model přítomny vždy minimálně funkce *get_data*() a *get_scenarios_data*(). Jádro frameworku je volá při inicializaci a jejich absence způsobí nefunkčnost celého frameworku.

Definice modelu a naplění QDX databáze

Pokud máme připravena vstupní data je na čase jimi naplnit databázi. Tento krok se provádí ve funkci *set_model*() v souboru model.py v adresáři používaného modelu. Tento soubor musí vždy obsahovat minimálně tří následující funkce. Funkce $set_model()$ provádí naplnění databáze požadovanými daty. Jejími vstupními parametry jsou hodnoty, které jsme získali zavoláním funkcí *get_data*() a *get_scenarios_data*(). Naplnění databáze se může jevit složité, ale při používání připravených šablon a dodržování správné syntaxe je použití naopak jednoduché.

Ve výpisu 2.5 je možné najít několik možných příkladů, jak naplnit parametr databáze QDX. První ukázkou je definice skalární hodnoty, která je inicializována jako parametr s dimenzí 0. Tomuto parametru je následně přidělena hodnota získaná ze struktury *data*, jejíž získání je popsáno v předešlých kapitolách.

V dalším příkladě je možné vidět inicializaci množiny a její následné naplnění daty. Naplnění probíhá v cyklu přes všechny dostupné hodnoty.

Výpis 2.5: Vytvoření parametrů QDX databáze a naplnění hodnotami

```
def set_model(db, data, scenarios_data):                    1
  # přidání skaláru pojmenovaného 'a'                       2
  a = db.add_parameter("a", 0, "popis␣parametru␣a")         3
  a.add_record().value = data.get("a")                      4
                                                            5
  # pridani množiny pojmenované 's'                         6
```

```
s = db.add_set("s", 1, "popis␣množiny␣s")
for item in range(data.get("s")):
  i.add_record(str(item))

# indexy sloupců
indexes = range(1, data.get("j"))
j = db.add_set("j", 1, "popis␣indexu␣j")
for index in indexes:
  j.add_record(str(index))

# parametr b s hodnotami pro konkrétní index
b = db.add_parameter_dc("b", [j], "popis␣parametru␣b")
for j_item, item in zip(j_indexes, data.get("b")):
  b.add_record(str(j_item)).value = item

# indexy řádků
indexes = range(1, data.get("i"))
j = db.add_set("j", 1, "popis␣indexu␣i")
for index in indexes:
  i.add_record(str(index))

# parametr s daty o jednotlivých scénářích
xi = db.add_parameter_dc("xi", [i], "popis␣parametru␣xi")
for ii, item in zip(i_indexes, scenarios_data.get("xi")):
  xi.add_record(str(ii)).value = item

return db
```

Následuje inicializace a vytvoření parametru indexů sloupců, který je následně využit jako dimenze parametru b. Jinými slovy, parametr b bude n´po inicializaci naplněn hodnotou pro každý jednotlivý index parametru j. Podobným způsobem jsou pak vytvořeny index řádků i, které slouží jako dimenze pro parametr xi zastu-

pující jednotlivé scénáře. Zde si lze povšimnout, že data do toho parametru nejsou načítán ze struktury *data*, nýbrž ze struktury *scenarios_data*. Popis naplnění této struktury vstupními daty lze nalézt také v předešlých kapitolách.

Druhá je funkce nacházející se v tomto souboru je *get_gams_model*(), která definuje vstupní GAMS model. Jejím jediným účelem je definování zdrojového modelového souboru GAMS. Tělo této funkce je ukázáno na výpisu 2.6.

Výpis 2.6: Definice zdrojového GAMS souboru

```
def get_gams_model():                                    1
  return 'gams\\MODEL_NAME\\model.gms'                   2
```

Třetí a poslední funkce v souboru má jméno *get_return_values*() a definuje entity, které chceme vytáhnout z vypočítaných výsledků. Zde je pro správnou interpretaci důležité také uvést jejich návratový typ. Ve výpisu 2.7 je možno vidět tělo této funkce. V tomto konkrétním případě bude vracena hodnota účelové funkce označená 'Z', která je typu proměnná.

Výpis 2.7: Definice návratových hodnot

```
def get_return_values():                                 1
  # návratové typy                                       2
  ## - var = proměnná                                    3
  ## - par = parametr                                    4
  ## - sca = skalár                                      5
  return_values = {'Z': 'var'}                           6
```

Ve všech popsaných funkcích má uživatel možnost provádět s daty různé transformace. V tomto souboru se mohou nacházet i další pomocné funkce, avšak tři výše zmiňované v něm musí být přítomny vždy s korektním názvem.

Upravení GAMS modelu

Vytvoření programu v GAMS, který bude kompatibilní s tímto frameworkem se příliš neliší od standardního vytváření modelů. Jediná změna, kterou je třeba mít na paměti je, že u množin, parametrů a hodnot, které mají být importovány řídícím programem frameworku, je nutné, aby neměly nastavenou počáteční hodnotu a byla pro načtení hodnoty byla použita funkce pro import záznamu z QDX databáze. Tato akce nemusí být provedena u všech množin, parametrů či hodnot ve výpočetním modelu použitých. Pro správnou funkčnost a dosažení správných výsledků je však nezbytné, aby byl tento postup aplikován u všech entit, které se s výpočetním modelem mění.

Výpis 2.8: Příklad použití množiny s obsahující deset hodnot

```
set s /1 * 10/;
```

Ve výpisu 2.8 lze vidět deterministické definování počtu hodnot v množině s, zatímco ve výpisu 2.9 jsou hodnoty v této množině importovány s QDX databáze s každou iterací spuštění výpočetního modelu. Stejným způsobem jsou načítána data pro všechny měnící se entity modelu.

Výpis 2.9: Příklad použití množiny s, počet hodnot je načten z databáze

```
set s;
$gdxin %gdxincname%
$load s
$gdxin
```

2.1.3 Analýza komplexnosti použití frameworku

V této podkapitole bude popsáno srovnání komplexnosti použití navrženého frameworku oproti standardními použití GAMS studia. Hodnocení probíhalo na zá-

Tab. 2.1: Škála hodnocení komplexnosti použití

Náročnost dokončení požadované úlohy	
nízká	
střední	
vysoká	
nerelavatní	

kladě přidělování náročnosti provedení dané operace dle navržené škály. Tuto škálu lze vidět v tabulce 2.1. V případě, že úlohu je jednoduché dokončit, je ohodocena zelenou barvou. V případě střední náročnosti dokončení úlohy je označeno žlutou a v případě vysoké náročnosti dokončení hodnocené úlohy je označena červenou barvou. Šedou barvou jsou označeny úlohy, které z aplikačního pohledu nemá smysl hodnotit.

V tabulce 2.2 jsou zobrazeny hodnocené úlohy a náročnosti jejich dokončení. Mezi analyzované úlohy patřilo zpracování jednoduchého modelu, složitějšího scénářového modelu a velmi složitého modelu pracujícího se scénáři scénářů. Dále se hodnotila náročnost načítání a zpracování vstupních dat a možnosti exportu dosažených výsledků. Posledním analyzovaným kritériem bylo hodnocení náročnosti udržení přehledného kódu i při velmi rozsáhlých modelech.

Pokud budeme uvažovat jednoduché úlohy s jedním scénářem, nebo jen pár scénáři, nebude využití frameworku příliš efektivní. Vhodnější variantou zde bude použití samotného GAMS studia, kdy si ušetříme práci se zpracováváním dat a odpadne také nutnost využít QDX databázi. Z tohoto důvodu můžeme na prvním řádku tabulky 2.2 vidět o něco horší ohodnocení komplexnosti při použití Python s Gamsem.

Jiná situace nastává již na druhém řádku, který znázorňuje hodnocení komplexnosti využití scénářového modelu. Zde jsem dílčí entity ohodnotil stejně střední náročností, ale budeme-li uvažovat velké množství možných scénářů, bude použití

Tab. 2.2: Tabulka hodnocení komplexnosti použití

Hodnocení náročnosti dokončení požadované úlohy			
Typ úlohy	**GAMS Studio**	**Framework GAMS**	**Framework Python**
Jednoduchý model			
Scénářový model			
Načítání vstupních dat			
Zpracování vstupních dat			
Model se scénáři scénářů			
Udržitelnosti zdrojového kódu			
Možnosti exportu			

kombinace GAMS a programovacího jazyka Python mnohem efektivnější, což bude popsáno v následující kapitole.

Přestože je možné určitým způsobem zařídit načítání vstupních dat i v GAMS studiu, při načítání vstupních dat v jazyce Python máme možnost načítat data z téměř libovolného zdroje. Může se jednat o strukturované textové soubory, nejrůznější databáze či jiná aplikační rozhraní. Zde je však nutné podoknout, že bude-li chtít uživatel využívat široké možnosti získávání či dolování vstupních dat, je nutné mít alespoň základní povědomí o programových konstrukcích programovacího jazyka Python či jiného systému pracujícího s daty. Tohle velmi úzce souvisí se zpracováním vstupních dat. Pokud získáváme data z různých zdrojů a souborů, mohou být velmi často nekompletní, mohou se v nich objevovat nevalidní hodnoty nebo jiný šum. Python nám umožňuje velmi efektivně s daty pracovat. V případě, že jsou vstupní data zpracovávána před načtením do výpočetní modelu, v GAMS modelu poté úplně odpadá nutnost validace dat, což u úlohy zpracování vstupních dat v tabulce 2.2 naznačeno šedivou barvou. V samotném GAMS studiu je tato operace

naopak velmi náročná až nemožná, což je v tabulce označeno červeně.

Uvažujme nyní použití modelu pracujícího se scénáři scénářů. Představme si model kdy můžeme uvažujeme deset scénářů nějakého problému, kdy každý scénář se rozděluje na dalších deset scénářů. V takovém případě by byla definice dat v prostředí GAMS studia velmi náročně řešitelná a samotný výpočetní model by byl také velmi komplexní. Tohle také velmi úzce souvisí s další hodnocenou úlohou a to je udržení přehlednosti kódu. V případě použití 10 scénářů z nichž každý obsahuje deset scénářů, by se třebaže ještě dalo pracovat i v prostředí GAMS studia, ale představme si variantu kdy místo deseti budeme používat deset tisíc scénářů. V takovém případě už se nedá ani přemýšlet o nějaké šanci na udržení přehlednosti kódu a efektivní práci. Proto jsou u obou políček těchto úloh pro prostředí GAMS studia červená pole. Naprosto jiná situace nastává, pokud oddělíme data od výpočetní logiky a uložíme je do databáze. V tom případě se nám práce s vytvořením Gamsového kódu rapidně zjednoduší a mnohonásobně se zlepší přehlednost kódu. Další zjednodušení přichází s tím, že v multi scénářových úlohách při použití tohoto frameworku, se bude počítat vždy pouze s jednou sadou scénářů. Při použití správné struktury vstupních dat a vhodnému naplnění databáze může program počítat pouze s jednou sadou scénářů a výsledky zpracovávat vně Gamsového modelu. To přispěje k dalšímu rapidnímu zjednodušení Gamsového modelu. Jádro frameworku navíc každý jeden scénář zpracovává v samostatném výpočetním vláknu, takže i při použití mnoha potenciálních scénářů bude výpočet maximálně efektivní.

Poslední analyzovanou oblastí jsou možnosti exportování a prezentace výsledků. Přestože GAMS nabízí určité možnosti práce s textovými soubory, nedá se hovořit o efektivní a jednoduché úloze, zvláště pak je-li třeba výstupy lehce změnit či optimalizovat. Proto je v hodnocení této úlohy za použití GAMS studia vidět opět červené ohodnocení. V případě použití aplikačního rozhraní pythonu a gamsu je situace odlišná, zejména při použití připraveného frameworku. Protože celou práci obstarává jádro implementovaného frameworku, zpracování ve výpočetním modelu Gams je naprosto bezúčelné a je zde ohodnoceno šedivou barvou. S navrácenými hodnotami z výpočetního modelu do jádra frameworku může uživatel provádět libovolné další operace. Práce se soubory je v tomto prostředí velmi jednoduchá a efektivní. Sa-

motný framework je schopen exportovat získané výsledky do textového formátu a do sešitu programu Microsoft Excel. Dále je v něm implementována možnost vizualizace získaných dat ve formě atraktivních grafů. Tato možnost není zabudována přímo do jádra systému, je ji třeba použít odděleně. Je to z důvodu velké škály možností vzhledu daných grafů a interpretace dat.

2.1.4 Analýza výkonnosti

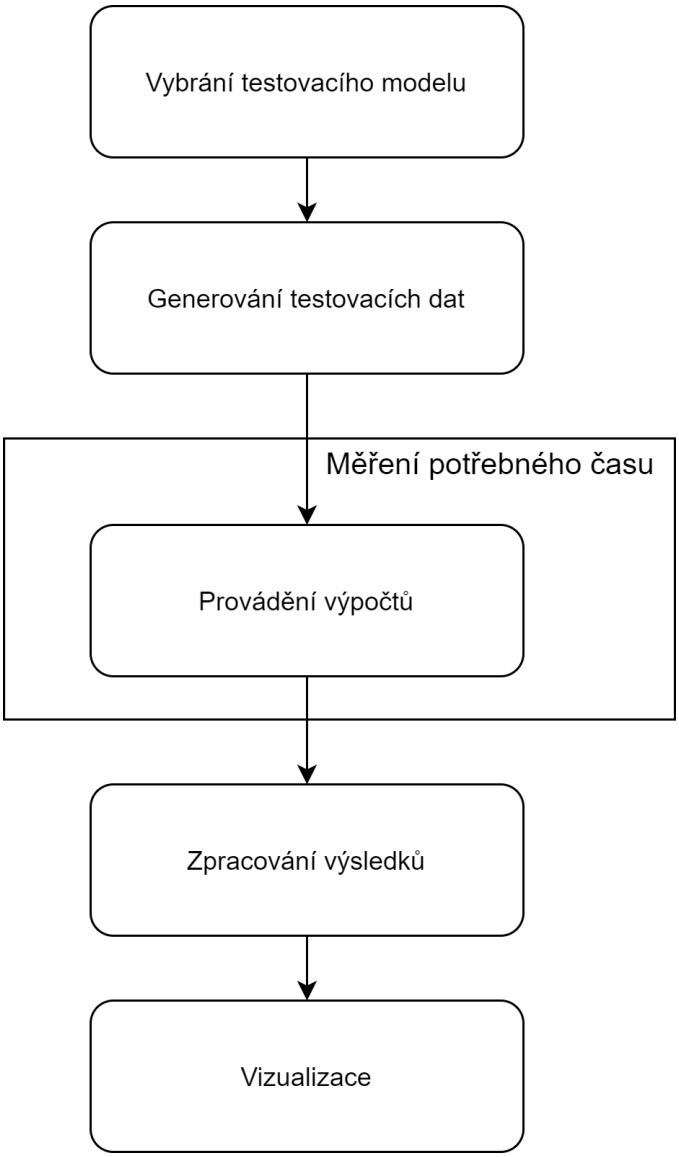
V této části dokumentace bude probíhat analýza výkonnosti použití implementovaného frameworku oproti standardním optimalizačním výpočtům v GAMS studiu. Bude se zde sledovat čas potřebný k dokončení výpočtu v závilosti na počtu analyzovaných scénářů. Vstupní data budou naprosto totožná, stejně tak jako použitý výpočetní model. Zásadní rozdíl bude v oddělení aplikační logiky od dat, oproštění se od grafického rozhraní a využití multi vláknového modelu zpracování.

Testování probíhalo na domácím laptopu s 16 GB operační paměti DDR4, SSD diskem 512 GB a procesorem Intel Core i7 osmé generace. V průběhu testování dosáhlo maximální vytížení procesoru 20 % a celý výpočetní program zabíral maximálně 5 % paměti při nejnáročnějších výpočtech.

Testovací model a data

Pro testování byl použit model, který vyhodnocoval délku kritické cesty projektu předpřipraveného testovacího modelu. Byl využit přístup *here and now*, kdy je nutné učinit rozhodování ještě před budoucí realizací náhodného parametru ξ. Model zpracovávaný v GAMS studiu obsahoval všechna potřebná data. Model připravený pro použití v kombinaci s implementovaným frameworkem naopak neobsahoval žádná konkrétní data. Vstupní data byla načtena z připravených csv souborů. Mimo samotný výpočet bylo také úkolem frameworku zpracovat vstupní data, naplnit QDX databázi, spustit výpočetní model, který zpracovával data vytažená z databáze a následně ošetřit správné dokončení procesu.

Na obrázku 2.2 je zobrazeno schéma procesu testování výkonnosti implemetovaného frameworku. V prvním kroku byl vybrán testovací model a následně byla vygenerována náhodná testovací data. Následně probíhal samotný výpočet. Testování probíhalo na šesti různých datových sadách, kde každá datová sada obsahovala různý počet testových scénářů. Testy probíhaly na vzorku dat čítajícím 1-100 000 testovacích scénářů. Měření potřebného času ke zpracování testových dat bylo prováděno pro 5 různých případů užití.



Obr. 2.2: Schéma procesu testování

Pro testování nebyla použita žádná reálná data. Všechna vstupní data byla uměle vygenerována pseudonáhodným generátorem hodnot pro všechny potřebné elementy jednotlivých scénářů. Testování probíhalo na 1 a ž 100 000 scénářích. Z důvodu omezených technických i časových zdrojů nebylo možné provádět testování na větších datech.

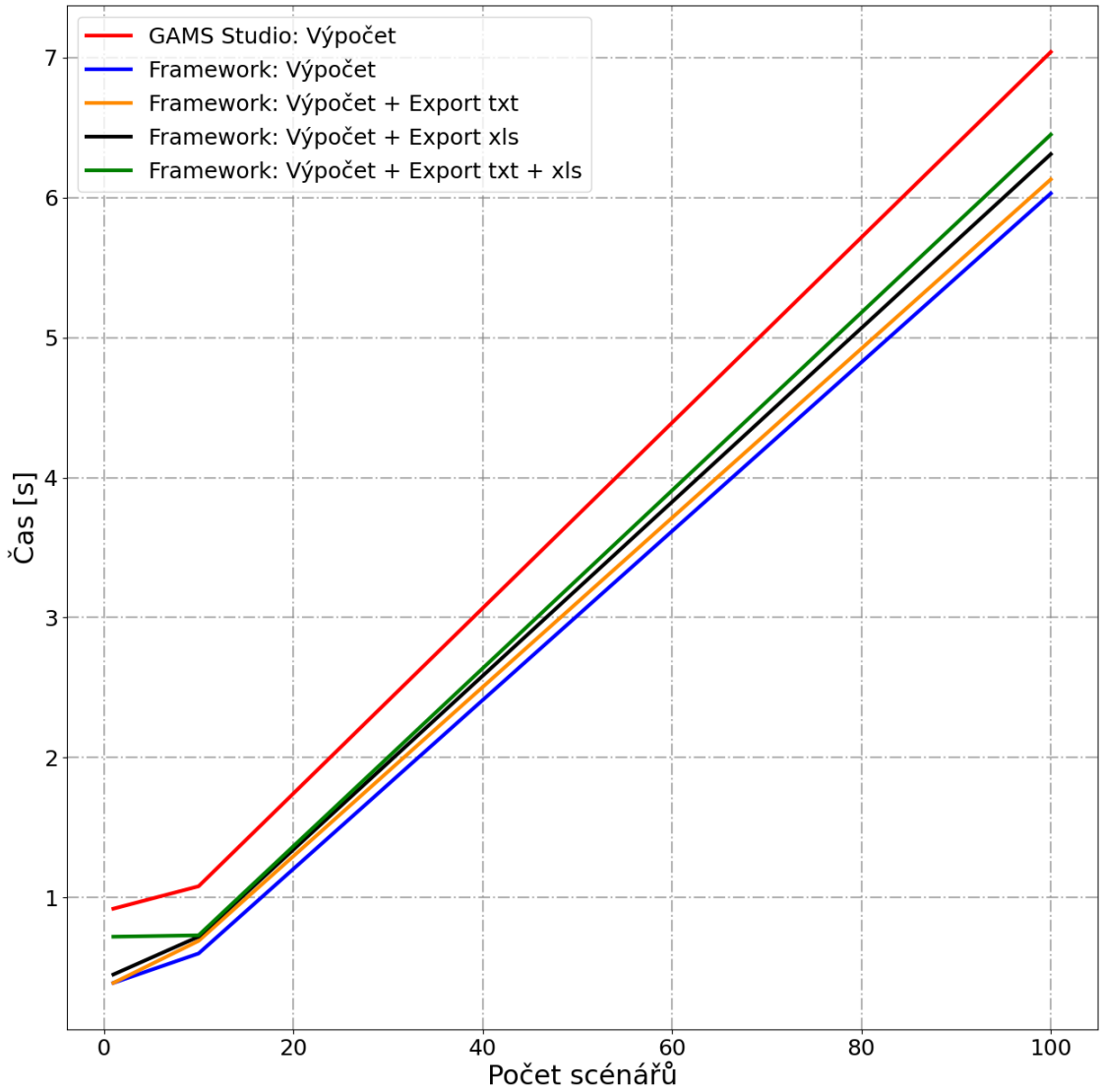
Výsledky experimentů

V této části diplomové práce budou prezentovány výsledky testování výkonnosti implementovaného frameworku oproti standardnímu použití v GAMS studiu. Testováním výkonnosti je zde myšleno měření času nutné k dokončení specifické úlohy. Pro obě možnosti výpočtu byl vždy použit naprosto identický matematický a Gams model a také identická vstupní data. Je třeba zdůraznit, že testování probíhalo na domácím počítači s omezenými zdroji, ale v průběhu testování nebyly spuštěné žádné další programy mimo operační systém, které by mohly způsobit výraznější zkreslení výsledků.

Testování probíhalo pro celkem pět různých případů užití a šest různých velikostí vzorků vstupních dat. První případem použití bylo standardní použití GAMS studia. V tomto případě probíhal pouze samotný výpočet a zobrazení výsledné hodnoty účelové funkce testovaného modelu. Tato hodnota sloužila k verifikaci dosažených výsledků výpočtů. Kromě výpočtu a zobrazené výsledné hodnoty účelové funkce, zde neprobíhalo žádné další zobrazování hodnot uživateli či výstup do souboru.

Další čtyři případy užití souvisely s implemetovaným frameworkem. Nejdříve byl měřen čas nutný k dokončení samotného výpočtu, v další fázi byly dosažené výsledky exportovány do výstupního textového souboru. Následovalo měření času nutného k dokončení úlohy při výpočtu dané aplikace a exportu výsledných dat sešitu s možností zobrazení v aplikaci Microsoft Excel. Poslední sledovaný případ užití byl kombinací předchozích sledovaných, konkrétně sledování času nutného k dokončení úlohy s následným exportem dosažených výsledků do textového souboru a sešitu aplikace Microsoft Excel.

Na obrázku 2.3 lze vidět graf znázorňující čas potřebný k dosažení požadované úlohy v závilosti na zvyšujícím se množství vstupních dat. Čas je zde uveden v sekundách a množství vstupních dat v počtu vygenerovaných testovacích scénářů. Z popisů os grafu lze vyčíst, že jsou zde prezentovány výsledky pro velikost vstupního vzorku dat od jednoho po stovku scénářů. V grafu se nachází celkem pět křivek, které korespondují s analyzovanými případy užití výpočetních modelů popsaných výše v této kapitole.



Obr. 2.3: Výsledky testování výkonosti - 1 až 100 scénářů

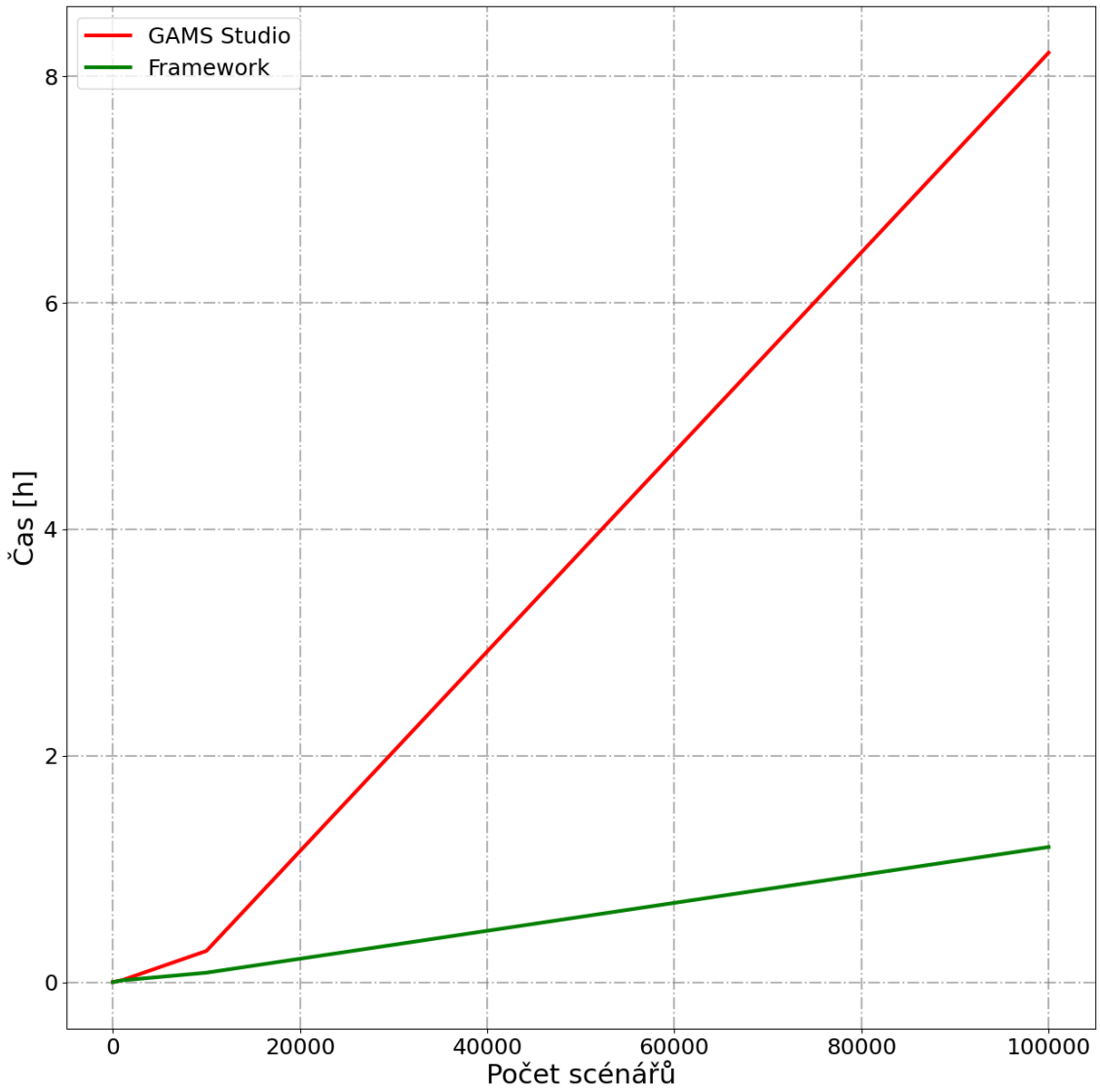
Při pohledu na tento graf je patrné, že všechny křivky v něm mají stejný charakteristický průběh a liší se pouze v hodnotách času nutného k dokončení úlohy. Modrá křivka, která v grafu znázorňuje použití implementovaného frameworku, má nejnižší hodnoty po celé definičním oboru. To znamená, že pro všechny znázorněné velikosti vzorků vstupních dat, zde byl naměřen nejnižší čas nutný k dokončení úlohy. Naopak červená křivka, která zde znázorňuje standardní použití GAMS studia, obsahuje nejvyšší hodnoty pro všechny velikosti vzorků vstupních dat. Mezi červenou a modou křivkou se pak nalézají, postupně od křivky s nejvyššími naměřenými hodnotami po křivku s nejnižšími naměřenými hodnotami, křivka zelená, černá a žlutá. Ty znázor-

ňují případy měření výkonnosti výpočtu modelu s exportem výsledků do textového souboru (žlutá křivka), exportem do sešitu aplikace Microsoft Excel (černá křivka) a exportem do textového souboru i sešitu aplikace Microsoft Excel (zelená křivka). Lze pozorovat, že čas nutný k dokočení úlohy se zde měnil jen nepatrně, což bylo způsobeno testováním na né příliš velkém vzorku vstupních dat.

Přestože lze z tohoto grafu vyčíst, že framework dosáhl lepších výkonostních výsledků i při využití možností exportu, celkový čas pro definici zpracování vstupních dat a jejich načtení do QDX databáze by zabralo podstatně více času, než prosté použití v GAMS studiu. Z tohoto pohledu by se práce s malými daty v implementovaném frameworku mohla zdát kontraproduktivní. Jeho přínos bude zjevnější při analýze na velkých datech, ale teoreticky může být velmi přínosný i pro výpočetní modely pracující s malými daty. V případě, že by bylo nutné vstupní data natahovat z nějakých aplikačních rozhraní či dolovat například z textů, obrazů, zvuků nebo jiných zdrojů, mělo by zde velký potenciál využít možnosti zpracování takovéto úlohy v programovacím jazyce Python a následné přímé napojení na výpočetní model.

Na obrázku 2.4 jsou vizualizovány naměřené výsledky pro 1-100 000 vstupních scénářů. V tomto grafu je opět vidět naměřený čas konkrétních případů užití v závislosti na velikosti vstupního vzorku dat. Jelikož se v tomto případě jedná o vizualizaci malých i velkých vstupních dat zároveň, čas nutný k dokončení úlohy znázorněný na vertikální ose, je přepočítán na hodiny. Dále si lze povšimnout, že v grafu se nyní nacházejí pouze dvě křivky. Červená koresponduje stejně jako v předchozím případě s výpočty realizovanými pouze v prostředí GAMS studia. Zelená křivka zastupuje stejně jako výše v této kapitole případ užití frameworku s následným exportem do textového souboru i sešitu zobrazitelného v aplikaci Microsoft Excel. Další případy užití frameworku zde nejsou uvedeny, jelikož se jejich časy v grafu vzájemě překrývaly se zelenou křivkou. Pro zjednodušení a zlepšení vizualizace byly z tohoto důvodu z grafu vynechány, avšak budou zmíněny a zobrazeny v následující části této kapitoly.

Z grafu lze vyčíst, že i v tomto případě je použití implementovaného frameworku oproti samostanému GAMS studiu efektivnější. Při stále se zvětšující velikosti vstupních dat roste také účinnost použití frameworku. Z grafu lze vyčíst, že zlomový bod pro zaznamenání zřetelných rozdílů nastává u použití 10 000 scénářů.



Obr. 2.4: Výsledky testování výkonosti - 1 až 100 000 scénářů

Při analýze výpočetního modelu a měření času potřebného k dokončení úlohy při práci se 100 000 scénáři bylo použití implementovaného frameworku téměř 8 krát efektivnější. Zatímco při použití takto velkých vstupních dat jsme se při použití samostatného GAMS studia dostali na čas převyšující 8 hodin, u stejného úlohy s

totžnými vstupními daty potřeboval framework k dokončení přibližně 71 minut.

Výše v této kapitole byly prezentovány a vizualizovány průběhy měření výkonnosti frameworku oproti standardnímu použití GAMS studia, respektive měření času potřebného k dokončení testovací úlohy. Toto testování probíhalo na stejném vzorku dat. V tabulce 2.3 jsou vidět konkrétní časy pro všechny analyzované případy užití a velikosti vzorku testovacích dat.

Tab. 2.3: Konkrétní výsledky testování výkonnosti

Časová náročnost dokončení úlohy					
Rozsah úlohy (scénáře)	**GAMS Studio**	**Framework**			
		Výpočet	**Exporty**		
			Text	**Excel**	**Txt+Xlsx**
1	0,92 s	0,39 s	0,39 s	0,45 s	0,72 s
10	1,08 s	0,60 s	0,69 s	0,72 s	0,73 s
100	7,04 s	6,03 s	6,13 s	6,31 s	6,45 s
1 000	1m 4s	54,64 s	54,99 s	57,36 s	57,72 s
10 000	16m 34s	4m 53 s	4m 55s	4m 57s	5m 1s
100 000	8h 12m 36s	1h 11m 9s	1h 11m 15s	1h 11m 18s	1h 11m 37s

Pro zajištění co nejmenšího zkreslení výsledků bylo každé měření opakováno třikrát. Tyto časy byly následně zprůměrovány a zapsány do výsledné tabulky časové náročnosti dokončení úloh jednotlivých případů užití v závilosti na velikosti vstupního vzorku dat.

První řádek tabulky obsahuje časy nutné k dokončení úlohy pro všechny případy užití, kdy byl použit pouze jeden vstupní scénář. Lze vidět, že u všech případů užití se výsledná hodnota dostala pod 1 vteřinu. Přesto lze také vidět, že použití frameworku

oproti použití GAMS studiu bylo výpočtně efektivnější ve všech případech. Dokonce při využití možnosti exportu výsledků do textového souboru nebo sešitu aplikace Microsoft Excel byl nutný čas k dokončení nižší o více než 50 %.

Druhý řádek tabulky ukazuje naměřené hodnoty pro 10-násobnou velikost vstupního vzorku oproti předhozímu případu. I zde byl využití frameworku výpočetně efektivnější oproti standardnímu použití GAMS studia. Časové náročnost dokončená úlohy se pro GAMS studio vyhoupla nad jednu vteřinu, zatímco pro všechny analyzované případy užití implementovaného frameworku zůstala časová náročnost pod 1 vteřinou.

Při analýze výkonnosti při použití 100 vstupních scénářů se opět změny projevovaly jen nepatrně. Čas nutný k dokončení úlohy pro GAMS studio se pohyboval okolo 7 vteřin, zatímco čas nutný k dokončení úlohy při použití implementovaného frameworku byl o vteřinu nižší. Zde lze polemizovat nad tím, že znatelným přínosem by mohlo být iterativní zpracování různých sad vstupních dat se stejnou velikostí jednoho vzorku. Kdybychom používali například model scénáře scénářů a iterativně procházeli všechny scénáře, kde každý by obsahoval 100 scénářů, kromě jednodušší implemetace bychom dosáhli také zefektivnění výpočtu o přibližně 100 vteřin. Stejně bychom mohli uvažovat také při analýze výsledků dosažených při měření výkonnosti, kdy vstupní vzorek dat čítal 1 000 scénářů. Zde byl naměřen čas nutný pro dokončení úlohy při použití implementovaného frameworku nižší přibližně o 10 sekund oproti standardnímu použití GAMS studia.

Pátý řádek tabulky výsledků měření časové náročnosti dokončení úlohy obsahuje časy získané při testování na vzorku dat o velikosti 10 000 scénáčů. Zde si lze povšimnout již velmi znatelného rozdílu dosažených výsledků. Při použití samotného GAMS studia se časy nutné k dokončení této úlohy pohybovaly okolo 16 minut a 34 vteřin. Pokud toto srovnáme s naměrenými hodnotami při použití implementovaného frameworku, zjistime, že potřebný čas k dokočení této úlohy byl více než třetinový. Výsledný čas samotného výpočtu pomocí implementovaného frameworku se pohyboval pod 5 minut a při použití obou možností exportu okolo 5 minut. Došlo tedy k téměř 70% redukci času nutného k dokončení požadavané úlohy.

Poslední řádek tabulky ukazující čas potřebný k dokočení úlohy pro všechny

analyzované případy užití ukazuje hodnoty naměřené při analýze, kdy vstupní vzorek dat čítal sto tisíc testových scénářů. Z prvního sloupce, který ukazuje získané hodnoty z měření výkonnosti při použití GAMS studia lze vyčíst, že nutný čas k dokončení této úlohy přesáhl 8 hodin. Zde bych rád zdůraznil informaci uvednou na začátku této podkapitoly, že všechna měření byla prováděna třikrát a naměřené výsledky byly průměrovány, aby se co nejvíce minimalizovalo zkreslení naměřených časů. V případě použití implementovaného frameworku s identinckým výpočetním modelem a identickými vstupními daty, došlo k rapidnímu snížení potřebného času k dokončení úlohy. I zde byla veškerá měření prováděna třikrát a naměřené výsledky byly průměrovány. Zde se výsledky pohybovaly okolo 71 minut pro všechny případy užití frameworku, včetně využití možností exportů výsledků do textového souboru a sešitu aplikace Microsoft Excel. Pokud tyto výsledky srovnáme, došlo k více něž 85 % zefektivnění prováděné činnosti.

2.1.5 Knihovna ukázkových příkladů

Kromě návrhu a implementace výpočetního jádra je součástí optimalizačního frameworku také knihovna ukázových příkladů. Knihovna obsahuje čtyři různé moduly:

- **Transportní model** - Jedná se o standardní implementaci transportního modelu, kdy hledáme optimální minimální náklady spojené s rozvozem dodávek do definovaných lokalit.

- **Famářův model** - Ukázkové řešení příkladu dvoustupňového stochastického rozhodovacího problému, kdy se farmář rozhoduje, jak rozčlenit své zdroje pro maximalizaci zisku při nejistotě sekundárního rozhodnutí.

- **CPM model** - Tento model demonstruje problém výpočtu kritické cesty v projektech. Součástí je více výpočetních modelů implementující různé opimalizační přístupy (*wait-and-see i here-and-now*). Lze zde najít jednoduché modely lineárního programování i ukázku dvoustupňové úlohy společně s výpočty charakteristik EVPI, EEV i VSS.

- **Model alokace výrobních zdrojů** - Zde se jedná o reálný model převzatý z [14] a přizpůsobený pro použití v optimalizačním frameworku. Tento model zde byl použit pro demonstraci užití frameworku na reálném modelu s reálnými daty a provádění experimentů s velkými daty. Detailům ohledně implemtace a použití se bude detailně věnovat kapitola 3.

Dále tato knihovna obsahuje podpůrné skripty pro testování výkonnosti prováděných experimentů či generování vstupních simulačních dat. Součástí je také skript umožňující vykreslování atraktivních grafů z dosažených výsledků včetně ukázky jeho použití.

2.2 Shrnutí

V předešlých kapitolách této sekce dokumentace k diplomové práci byl nejprve popsán návrh frameworku pro práci s optimalizačními modely. Pricip spočíval ve využití propojení aplikačního rozhraní jádra modelovacího systému GAMS a programovacího jazyka Python. Základními požadavky bylo efektivní načítání vstupních dat a jejich načtení do QDX databáze, se kterou je modelovací systém GAMS schopen spolupracovat. Dále musel framework umožňovat načítat zdrojový kód GAMSového modelu. Po načtení vstupních zdrojů bylo vhodné, aby měl uživatel možnost s daty manipulovat a přizpůsobit si další chování modelu. Tento bod je důležitý při zpracování a čištění získaných dat z externích zdrojů, jiných aplikačních rozhraní či z dolování z velkých dat. K zajištění tohoto požadavku bylo nutné framework navrhnout modulárně, kdy každý přidaný modul znamenal samostný výpočetní model. Tímto způsobem je pak možné s frameworkem využívat více různých modelů bez nutnosi zakládání různých samostatných instancí. Důraz je zde kladen na maximální oddělení vstupních dat, definice výpočetního modelu a aplikační logiky. Framework dále umožňuje prezentaci dosažených výsledků uživateli na standardní výstup stejně jako exportovat výsledky do textového souboru či sešitu aplikace Microsoft Excel.

Následně zde byla popsána struktura samotného frameworku a vybrané pasáže implementace. Byly zde popsány pasáže, se kterými potenciální uživatel bude pracovat nejčastěji. Tyto kódy demonstrovaly použití několika modelů, definici deterministických i stochastických dat, načítání GAMSového zdrojového kódu výpočetního modelu a definici návratových hodnot.

Další část se věnovala hodnocení náročnosti použití implementovaného frameworku oproti standardnímu použití GAMS studia. Hodnoceno zde bylo 7 různých typů úloh zpracování, mezi které patřilo například zpracování jednoduché modelu, komplexního modelu, zpracování vstupních dat, udržení přehlednosti kódu u komplexních úloh a další. Zde jsem došel k závěru, že pro jednoduché úlohy by bylo použití frameworku velmi kontraproduktivní, zatímco u velmi rozsáhlých a komplexních úloh je jeho použiti uživatelsky mnohem přívětivější než použití GAMS studia.

Poslední částí této kapitoly bylo srovnání výkonnosti implementovaného frameworku oproti standardnímu použití GAMS studia. Testování probíhalo za použití

identického výpočetního modelu a identických dat. Vstupní data reprezentovala sada scénářů čítající 1-100 000 testovacích scénářů. Pro verifikaci dosažených výsledků byla kontrolována vypočtená hodnota. Testováno zde bylo 5 různých případů užití. Prvním bylo použití samotného GAMS studia. Další 4 byly variance použití implementovaného frameoworku s možnostmi samotného výpočtu a možností exportů výsledků do textového souboru a sešitu aplikace MS Excel. Čas nutný k dokončení testové úlohy byl nižší u použití frameworku ve všech případech. Na malých datech nebyl tento rozdíl tak znatelný, ale u testování na velkém vzorku dat byl čas potřebný k dokočení úlohy redukován o více než 85 %.

Celkově bych hodnotil tak, že efektivní použití tohoto frameworku nastává při použití na velkých datech z pohledu výkonnosti i uživatelské přívětivosti. Na malých datech je jeho použití spíše kontraproduktivní.

3 Aplikace na reálná data

V této části dokumentace k diplomové práci bude dříve popsaný vytvořený optimalizační framework použit na model s reálnými daty. Budu zde vycházet z již hotového optimalizačního modelu Ing. Tetoura, který se zabýval problémem alokace výrobních zdrojů v energetickém systému s ohledem na technické parametry zdrojů. [14] Cílem bude na tento model aplikovat principy implementovaného optimalizačního frameworku a dosáhnout jeho vylepšení. Dalším cílem bude otestování použití modelu pro větší počet možných scénářů a vyhodnodit efektivnost použití této metodiky. Než-li však přejdu k samotné aplikaci, popíšu zde ještě možnosti spuštění optimalizátoru.

3.1 Možnosti použití frameworku

V sekci 2.1.2 byla popsána adresářová struktura navrženého frameworku. Bylo zde také řečeno, že řídícím skriptem je skript *optimizer.py*. Nyní se podívejme blíže na možnosti jeho spuštění. Jedná se o konzolovou aplikaci a klasickým UNIXovým použitím přepínačů. Hlavní nastavení optimalizátoru se nachází v souborou *optimizer_settings.py*, přepínače slouží k definici zobrazení a ukládání dosažených výsledků. Popis k instalaci nutných částí k použití frameworku je obsahem souboru *ReadMe.md*. Pro vypsání nápovědy ke spuštění stačí spustit skript s následujcí syntaxí: *python optimizer.py* $-h$. Tímto se uživateli na standardní výstup konzole vypíše nápověda s popisem všech dostupných přepínačů s popisem jejich funkcionality. Všechny dostupné přepínače jsou k vidění v tabulce 3.1. Optimalizátor je možno pustit bez jakýchkoliv argumentů, v tom případě budou pouze vypsány výsledky na standardní konzolový výstup.

Ve výchozím nastavení se po dokončení výpočetů vypíšou výsledky na standardní konzolový výstup. Pokud je optimalizátor spuštěn s přepínačem $-r$, tyto výsledky vypisovány nebudou. Přepínače $-s$ a $-m$ plní podobnou funkcionalitu. Ve výchozím nastavení nejsou na standardní výstup vypisována hodnoty proměnných jednotlivých scénářů či konstanty modelu.

Tab. 3.1: Možnosti spuštění optimalizátoru

Přepínač	**Popis funkcionality**
-h/–help	Zobrazení nápovědy ke spuštění optimalizátoru
-r/–results	Nevypisování výsledků na konzolový výstup
-s/–scenarios	Vypisování dat scénářů na konzolový výstup
-m/model_print	Vypisování dat použitého modelu na kon. výstup
-o OUTPUT/–output OUTPUT	Uložení výstupů do textového souboru
-x EXCEL/–excel EXCEL	Uložení výsledků do strukturovaného Excel souboru

Použitím těchto přepínačů mohou být tyto informace vypisovány na konzoli. Přepínače $-o$ a $-x$ se používají při exportu výsledků do souboru. Řetězec za těmito přepínači definuje jméno výstupního souboru.

3.2 Úloha a její orginální řešení

V úvodu této sekce diplomové práce bylo uvedeno, že následující testování na reálných datech bude probíhat na optimalizačním modelu Ing. Tetoura[14]. Jedná se o model, kdy na základě dostupných informací bude probíhat optimalizace alokace výrobních zdrojů s ohledem na poptávané množství. Alokace výrobních zdrojů zde probíhá v energetickém systému, kdy je třeba zajistit požadovanou dodávku energie a zároveň minimalizovat náklady, tudíž se jedná o snahu zvýšit zisky.

Navržený optimalizační model vychází v více stupňového scénářového modelu stochastického programování. Ing. Tetour v originální práci využíval simulovaná data, která byla založena na reálných datech, statisticky stanovených odhadech a lineární regresi. V této části tedy budou prováděny experimenty na identických datech pro validaci a verifikaci prováděných úkonů a dalšími simulovanými daty pro testování na velkých datech.

Cílem prováděných experimentů není validovat již vytvořený model, ale zhodnodit přínosy jeho implementace do navženého frameworku pracujícího s aplikačním

rozhraním Python a GAMS.

3.2.1 Originální optimalizační model

Pokud se podíváme na navržený optimalizační model Ing. Tetoura, zjistíme, že se jedná o jeden samostatný soubor GAMS. Jedná se o 360 řádků dlouhý zdrojový kód, který zároveň obsahuje všechna potřebná statická data. Nyní si tento model probereme pro získání více informací a detailů pro pochopení funkčnosti a následné použití v optimalizačním frameworku.

Hned na začátku souboru vidíme definaci několika množin použitých při následných výpočtech, což lze vidět ve výpisu 3.1 Jejich popis je dostatečně vypovídající, a proto nemá smysl je zde více popisovat. Důležitou věcí k povšimnutí je, že všechny tyto množiny mají přesně definovány svůj obsah. Můžeme zde vidět, že v prováděných výpočtech figurují přesně dva typy kotle na biomasu, jeden kotel na plyn či 4 simulace. Bez nadefinování těchto hodnot není možné optimalizační výpočty provádět, avšak v kapitole 3.3 nutnost definice těchto dat v této části kódu naprosto eliminujeme.

Výpis 3.1: Definice množin pro výpočty [14]

```
set i     typ kotle na biomasu          / Bio1, Bio2 /,          1
  j       typ kotle na plyn             / Plyn /,                2
  t       uvazovane casy                / 1 * 49 /,              3
  s       uvazovane scenare             / 1 * 3 /,               4
  m       mesice                        / 1 * 12 /,              5
  r       hranice pro generatory        / d, h /,                6
  l       pocet simulaci                / 1*4 /;                 7
```

Po této definici vstupních množin následuje definice parametru rozsahu spotřeby vody na osobu čase. Tento parametr je indexován dvojicí čas a hranice pro generátory, které byly definovány v výše. Originální použití parametru a jeho inicializace je vidět ve výpisu 3.2. Lze zde vidět, že definice tohoto vektoru je již zančně rozsáhlá

a pokud bychom uvažovali více časů či více různých hranic pro generátory, je toto použití z hlediska udržitelnosti a editovatelnosti kódu naprosto nevyhovující.

Výpis 3.2: Definice rozsahu spotřeby vody na osobu v čase [14]

```
Parameter v(t,r) rozsah spotreby vody na osobu v case t /
1.d  1.12,   1.h  1.12,   2.d  0.68,   2.h  0.68,
3.d  0.68,   3.h  0.68,   4.d  0.48,   4.h  0.48,
5.d  0.48,   5.h  0.48,   6.d  0.28,   6.h  0.28,
7.d  0.28,   7.h  0.28,   8.d  0.12,   8.h  0.12,
9.d  0.12,   9.h  0.12,  10.d  0.04,  10.h  0.04,
11.d  0.04,  11.h  0.04,  12.d  0.08,  12.h  0.08,
13.d  0.08,  13.h  0.08,  14.d  0.24,  14.h  0.24,
15.d  0.24,  15.h  0.24,  16.d  0.52,  16.h  0.52,
17.d  0.52,  17.h  0.52,  18.d  0.56,  18.h  0.56,
19.d  0.56,  19.h  0.56,  20.d  1.064, 20.h  1.064,
21.d  1.06,  21.h  1.06,  22.d  1.024, 22.h  1.024,
23.d  1.02,  23.h  1.02,  24.d  1.08,  24.h  1.08,
25.d  1.08,  25.h  1.08,  26.d  0.984, 26.h  0.984,
27.d  0.98,  27.h  0.98,  28.d  1.28,  28.h  1.28,
29.d  1.28,  29.h  1.28,  30.d  1.104, 30.h  1.104,
31.d  1.1,   31.h  1.1,   32.d  0.96,  32.h  0.96,
33.d  0.96,  33.h  0.96,  34.d  0.76,  34.h  0.76,
35.d  0.76,  35.h  0.76,  36.d  0.8,   36.h  0.8,
37.d  0.8,   37.h  0.8,   38.d  1,     38.h  1,
39.d  1,     39.h  1,     40.d  1.008, 40.h  1.008,
41.d  1.008, 41.h  1.008, 42.d  1.16,  42.h  1.16,
43.d  1.16,  43.h  1.16,  44.d  1.344, 44.h  1.344,
45.d  1.34,  45.h  1.34,  46.d  1.264, 46.h  1.264,
47.d  1.26,  47.h  1.26,  48.d  1.12,  48.h  1.12,
49.d  1.12,  49.h  1.12/;
```

Podobně jsou vstupní data optimalizačního procesu definována v celém projektu. Ně-

kde se jedná jednodimenziální parametry, které se s průběhem jednotlivých iterací optimalizačních výpočtů nemění, jinde zase o vícedimenzionální parametry, které se mění s každou iterací optimalizačních výpočtů. Výpočty na modelu Ing. Tetoura byly prováděny pouze ve třech variantách, nebo-li scénářích. Lze spekulovat o tom, že testování na větších datech nebylo prováděno právě z důvodu nemožnosti udržet čitelný a vhodně strukturovaný zdrojový kód. Ve výpisu lze vidět, jak autor inicializoval dvoudimenzionální parametr udržující informace o teplotách v jednotlivých čase pro každý scénář. Snaha tímto způsobem udržovat informace o stovkách či tisících scénářů je téměř nereálná.

Výpis 3.3: Průměrné nominální teploty v čase v jednotlivých scénářích [14]

```
Parameter v(t,r) rozsah spotreby vody na osobu v case t /                    1
1.d  1.12,   1.h  1.12,   2.d  0.68,   2.h  0.68,                            2
3.d  0.68,   3.h  0.68,   4.d  0.48,   4.h  0.48,                            3
5.d  0.48,   5.h  0.48,   6.d  0.28,   6.h  0.28,                            4
7.d  0.28,   7.h  0.28,   8.d  0.12,   8.h  0.12,                            5
9.d  0.12,   9.h  0.12,  10.d  0.04,  10.h  0.04,                            6
11.d  0.04,  11.h  0.04,  12.d  0.08,  12.h  0.08,                           7
13.d  0.08,  13.h  0.08,  14.d  0.24,  14.h  0.24,                           8
15.d  0.24,  15.h  0.24,  16.d  0.52,  16.h  0.52,                           9
17.d  0.52,  17.h  0.52,  18.d  0.56,  18.h  0.56,                           10
19.d  0.56,  19.h  0.56,  20.d  1.064, 20.h  1.064,                          11
21.d  1.06,  21.h  1.06,  22.d  1.024, 22.h  1.024,                          12
23.d  1.02,  23.h  1.02,  24.d  1.08,  24.h  1.08,                           13
25.d  1.08,  25.h  1.08,  26.d  0.984, 26.h  0.984,                          14
27.d  0.98,  27.h  0.98,  28.d  1.28,  28.h  1.28,                           15
29.d  1.28,  29.h  1.28,  30.d  1.104, 30.h  1.104,                          16
31.d  1.1,   31.h  1.1,   32.d  0.96,  32.h  0.96,                           17
33.d  0.96,  33.h  0.96,  34.d  0.76,  34.h  0.76,                           18
35.d  0.76,  35.h  0.76,  36.d  0.8,   36.h  0.8,                            19
37.d  0.8,   37.h  0.8,   38.d  1,     38.h  1,                              20
39.d  1,     39.h  1,     40.d  1.008, 40.h  1.008,                          21
```

```
41.d  1.008, 41.h  1.008, 42.d  1.16,  42.h  1.16,                22
43.d  1.16,  43.h  1.16,  44.d  1.344, 44.h  1.344,               23
45.d  1.34,  45.h  1.34,  46.d  1.264, 46.h  1.264,               24
47.d  1.26,  47.h  1.26,  48.d  1.12,  48.h  1.12,                25
49.d  1.12,  49.h  1.12/;                                         26
```

Optimalizace definice matematického modelu a jeho implementace není předmětem těchto experimentů. Zde je primárním zaměřením optimalizace použití definovaného modelu, a proto zde nemá smysl uvádět a popisovat implementovaný matematický model. Značnou částí zdrojového kodu Ing. Tetoura je také zpracování a prezentace výstupních dat. GAMS obsahuje možnost intepretace dosažených výsledků do výstupního textového souboru, avšak implementace je poněkud těžkopádná. Je vidět ve výpisu 3.4.

Výpis 3.4: Zdrojový kód prezentace dosažených výsledků neoptimalizovaného modelu [14]

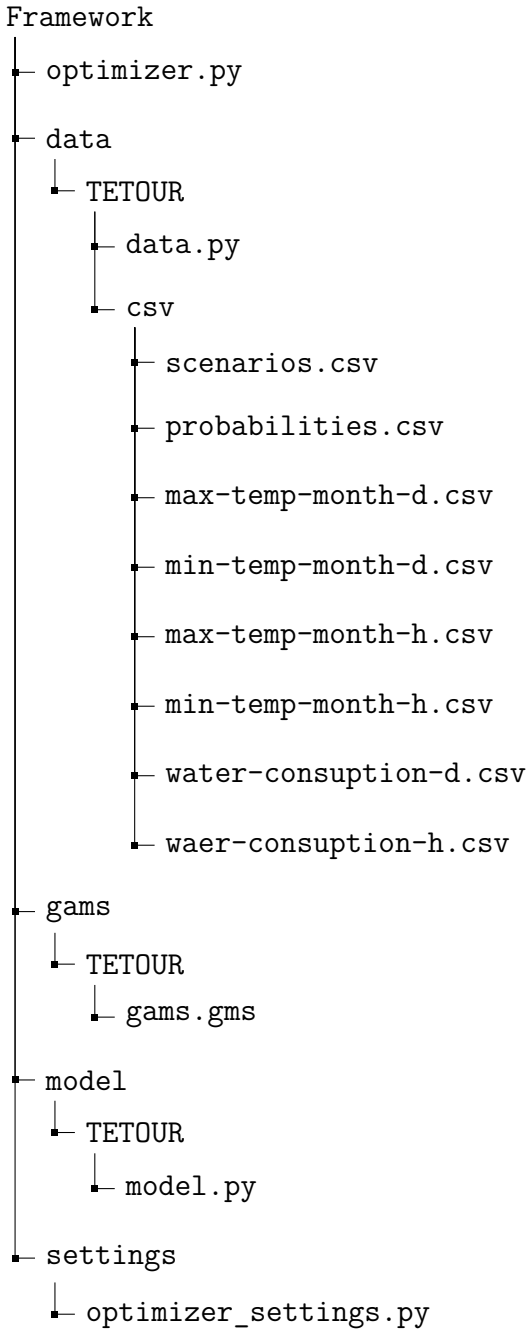
```
put "Vysledky" /;                                                 1
put "========" / /;                                               2
put "Naklady␣=␣", tc.L:10:2 /;                                    3
put "Vynosy␣␣=␣", tr.L:10:2 /;                                    4
put "Zisk␣␣␣␣=␣", z.L:10:2 / /;                                   5
put "Minimalni␣denni␣teplota␣=␣", min_tp:6:2 /;                   6
put "Maximalni␣denni␣teplota␣=␣",max_tp:6:2 / /;                  7
put "t␣␣", "␣␣␣␣symbol", "␣|␣naklad", "␣|␣kapacita", "␣|          8
␣Prepinani␣|␣␣␣␣␣␣";                                              9
loop(s, put s.TL:3,"␣|␣␣␣␣␣␣";); put /;                           10
put "␣␣␣", "␣␣␣␣pst(s)", "␣|␣␣␣␣␣␣", "␣␣|␣␣␣␣␣␣␣␣␣␣|",          11
"␣␣␣␣␣␣␣␣␣";                                                      12
put "␣␣|",loop(s, put pst(s):9:2,"␣|";); put /;                   13
put "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣␣␣|  14
␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣␣␣␣|" /;                15
```

```
put "␣␣␣",  "␣␣␣␣scn(s)", "␣|␣␣␣␣␣␣", "␣␣|␣␣␣␣␣␣␣␣␣␣|",
"␣␣␣␣␣␣␣␣␣␣";
put "␣␣|",loop(s, put snc.L(s):9,"␣|";); put /;
put "␣␣␣",  "␣␣␣␣scr(s)", "␣|␣␣␣␣␣␣", "␣␣|␣␣␣␣␣␣␣␣␣␣|",
"␣␣␣␣␣␣␣␣␣␣";
put "␣␣|",loop(s, put snr.L(s):9,"␣|";); put /;
put "␣␣␣",  "␣␣␣␣␣Pt(s)", "␣|␣␣␣␣␣␣", "␣␣|␣␣␣␣␣␣␣␣␣␣|",
"␣␣␣␣␣␣␣␣␣␣";
put "␣␣|",loop(s, put sum(t,Pt(t,s)):9,"␣|";); put /;
put "␣␣␣",  "␣␣␣␣␣Vr(s)", "␣|␣␣␣␣␣␣", "␣␣|␣␣␣␣␣␣␣␣␣␣|",
"␣␣␣␣␣␣␣␣␣␣";
put "␣␣|",loop(s, put Vr(s):9,"␣|";); put /;
put "=============================================="/;
loop(t,
put t.TL:3, "␣␣␣Ve(t,s)", "␣|␣␣␣␣␣␣", "␣␣|␣␣␣␣␣␣␣␣␣␣|",
 "␣␣␣␣␣␣␣␣␣␣␣␣|";
loop(s, put Ve(t,s):9:2,"␣|";);  put / ;
loop(s, put yZ.L(t,s):9:2,"␣|";); put /;
put "----------------------------------------------------"/;
);
put /;
```

Výše lze vidět zkrácený výpis originální zdrojového kódu pro prezentaci výsledků optimalizačního procesu. Pro nezkušeného uživatele může být tato notace velmi nepřehledná a matoucí. Vytvoření takového algoritmu je časově náročné a pokud by uživatel potřeboval si tento výpis přizpůsobit, mohl by na tom ztratit mnoho cenného času, což by jeho práci činilo kontraproduktivní.

3.3 Řešení s využitím optimalizačního frameworku

V této kapitole bude demonstrováno použití navrženého optimalizačního frameworku pro úlohu popsanou v předešlé kapitole. Prvním krokem na je založení vhodné adresářové struktry, dle doporučené šablony rozvržení struktury frameworku. Pokud nazveme projekt TETOUR, může adresářová struktura vypadat následovně.



Adresářová struktura konkrétní aplikace

Jelikož z je možné z principu používání frameworku mít k dispozici více separátních projektů, je nutné nastavit v souboru *optimizer_settings.py* aktivní model. To se provede přidáním řádků 3.5.

Výpis 3.5: Nastavení aktivního modelu

```
from data.tetour.data import *                                    1
from model.tetour.model import *                                  2
```

Nyní se podívejme na definici a získání vstupních dat pro optimalizační model. V adresářové struktuře založeného projektu se nachází adresář *data*, soubor *dada.py* a podadresář obsahující csv soubory. Tyto soubory obsahují vstupní data ve formátu csv, která budou zpracována a nahrána do QDX databáze pro použití v GAMS modelu. Nejprve se však podíváme na definici nescénářových dat. Jedná se o hodnoty, které se s optimalizačními výpočty jednotlivých scénářů nemění. Zaměříme se zde na použití frameworku pro optimalizovanou definici dat z výpisu 3.1.

V souboru *data.py* se nachází funkce *get_data*(), ve které máme možnost uložit právě takovéto hodnoty. Jak bude nová definice dat vypadat je vidět na výpisu 3.6. Tento výpis koresponduje s originálním zdrojovým kódem, ale lze si povšimnout, že počet scénářů je zde nyní definován dynamicky na základě počtu vstupních scénářů a není třeba hodnotu parametru *s* explicitně inicializovat.

Výpis 3.6: Definice hodnot vstupních dat ve funkci get_data()

```
def get_data():                                                   1
  data = dict()                                                   2
  data['i'] = ['Bio1', 'Bio2']  # typ kotle na biomasu            3
  data['j'] = ['Plyn']   # typ kotle na plyn                      4
  data['t'] = 49         # uvazovane casy                         5
  scenarios_data_file = 'csv/scenarios.csv'                       6
  data['s'] = sum(1 for line in open(scenarios_data_file))        7
  data['m'] = 12         # mesice                                 8
```

```
  data['r'] = ['d', 'h'] # hranice pro generatory
  data['l'] = 4          # pocet simulaci
  return data
```

Dále je nutné zpracovaná vstupní data načíst do QDX databáze, aby byla dostupná v GAMS modelu. To je nutné udělat v souboru *model.py* ve funkci *set_model*(). Ve výpisu 3.7 je ukázáno, jak probíhá vytvoření množiny v QDX databázi načtení zpracovaných hodnot. Pro tento úkon je nutné mít alespoň základní znalosti programování v jazyce Python. Součástí frameworku je také knihovna ukázkových příkladů, na kterých je demonstrováno vytváření záznamů v QDX databázi pro různé gamsové entity. Tyto příklady obsahují komentované ukázky definice množin, jednodimenziálních i více dimenziálních parametrů (tabulek) a skalárních veličin. I nezkušený uživatel by tedy měl být schopen naplnit správně QDX databázi.

Výpis 3.7: Naplění QDX databáze

```
def set_model(db, data, scenarios_data):
  i = db.add_set("i", 1, "typ␣kotle␣na␣biomasu")
  for item in data.get("i"):
    i.add_record(str(item))

  j = db.add_set("j", 1, "typ␣kotle␣na␣plyn")
  for item in data.get("j"):
    j.add_record(str(item))
```

V souboru *model.py* se také nachází další funkce, které slouží k definici zdrojového kódu modelu GAMS 3.8 a návratových hodnot pro export výsledků 3.9 .

Výpis 3.8: Definice GAMS modelu

```
def get_gams_model():                                    1
  return 'gams\\tetour\model.gms'                        2
```

Pro správnou interpretaci dosažených výsledků je třeba u výstupních hodnot definovat jméno této entity a její typ. Rozlišují se tři typy - proměnná, parametr a skalární veličina.

Výpis 3.9: Definice výstupních dat

```
def get_return_values():                                 1
  # return types                                         2
  ## - var = variable                                    3
  ## - par = parameter                                   4
  ## - sca = scalar                                      5
  return_values = {'z': 'var', 'tc': 'var', 'tr': 'var'} 6
  return return_values                                   7
```

Nyní se podívejme, jak se změní zdrojový kód GAMS pro načtení takto definovaných dat. To lze vidět ve výpisu 3.10. Pokud následující zdrojový kód srovnáme s originálním zdrojovým kódem z 3.1, dojdeme ke zjištění, že kromě úspory místa ve zdrojovém souboru jsme docílili naprostého oddělení konkrétních dat od použitého modelu. Tímto způsobem můžeme efektivně pracovat s jedním výpočetním modelem a různými variantami hodnot vstupních dat.

Výpis 3.10: Optimalizovaná definice nescénářových vstupních dat

```
set i, j, t, s, m, r, l;                                    1
$gdxin %gdxincname%                                         2
$load i j t s m r l                                         3
$gdxin                                                      4
```

Optimalizace definice scénářových dat

Nyní se podíváme na část zpracování scénářových dat. Ve výpisu 3.2 byla ukázána originální definice dat jednotlivých scénářů. Analýzou zdrojového kódu jsme se dostali k závěru, že pro větší počty scénářů je tento přístup neudržitelný. S využitím csv notace máme všechna data jednotlivých scénářů vhodně strukturována. Každý řádek souboru *scenarios.csv* obsahuje 49 číselných hodnot oddělených středníkem, což charakterizuje právě jeden scénář. Můžeme zde tedy jednoduše a efektivně uchovávat data daných časů konkrétních scénářů. Soubor může obsahovat velké množství takových scénářů. V souboru *data.py*, který se stará o zpracování vstupních dat je připraveno několik funkcí, včetně těch na zpracování takto formátovaných scénářových dat. Stačí nastavit jméno vstupního souboru s daty scénářů ve funkci *get_scenarios_data()*. Funkce vrací naformátovaný vstup scénářových dat pro jádro optimalizačního frameworku.

Výpis 3.11: Definice množin pro výpočty [14]

```
def get_scenarios_data():                                   1
  filename = 'data/csv/scenarios.csv'                       2
  return parse_scenarios_data(input_file=filename)          3
```

Dalším krokem je vytvoření dvoudimenzionálního parametru v QDX databázi a jeho naplnění scénářovými daty. K tomu je opět využita šablona pro vytvoření a

plnění dvoudimenzionálního parametru z knihovny ukázkových příkladů. Kód této akce je zobrazen ve výpisu 3.12 a je třeba jej umístit opět do funkce *set_model*() v souboru *model.py*.

Výpis 3.12: Definice dvoudimenzionálního parametru QDX databáze

```
k = db.add_parameter_dc("k", [t, s],                                    1
  "prumerne␣nominalni␣teploty␣v␣case␣t␣scenari␣s")                      2
for scenario_data in scenarios_data:                                    3
  xi_s = scenario_data.get("xi_s")                                      4
  for key, val in xi_s.items():                                         5
    index = (key[0], key[1])                                            6
    k.add_record(index).value = float(val)                              7
```

Použití ve zdrojovém GAMS souboru je velmi jednoduché. Využijeme naprosto stejný postup načítání hodnot parametrů jako v předchozí ukázce. Definice hodnot scénářů tedy bude ve zdrojovém kódu GAMS vypadat následovně 3.13.

Výpis 3.13: Optimalizovaná definice hodnot scénářů ve zdrojovém textu GAMS

```
parameter k(t,s) nominalni teploty v case t scenari s;                  1
$gdxin %gdxincname%                                                     2
$load k                                                                 3
$gdxin                                                                  4
```

Obdobným způsobem provedeme zpracování vstupních dat a jejich načtení do QDX databáze pro všechna definovaná konkrétní data v optimalizačním modelu. Nejen že docílíme značné redukce množství zdrojové textu, ale získáme možnost experimentovat s výpočetním modelem za použití dynamicky se měnících dat. Můžeme také použít libovolný počet vstupních scénářů, což výrazně zvyšuje škálovatelnost optimalizačního modelu. Limitování zde jsme pouze výpočetním výkonem počítače, případně licenčními omezeními softwaru GAMS.

Ve výpisu 3.4 byla znázorněn originální zdrojový text prezentace dosažených výsledků. V optimalizovaném modelu za použití implementovaného frameworku naprosto odpadá potřeba existence této části. O export výsledků se stará jádro optimalizačního frameworku, kde jako výstupní hodnoty pro export jsou uvažovány entity definované v souboru *model.py*, ve funkci pro definici návratových parametrů.

3.4 Srovnání implementací

V této části dokumentace k diplomové práci bude popsáno, jakých zlepšení bylo dosaženo aplikací implementovaného frameworku na již existující model. Jak již bylo popisováno v předešlých kapitolách, optimalizace se týkala způsobu experimentování s již existujícím optimalizačním modelem a né optimalizací implementovaného matematického modelu.

Sledovaná optimalizační kritéria jsou shrnuty v tabulce 3.2, kde lze vidět srovnání originální implementace a implementace pomocí navrženého frameworku. Originální model obsahoval 360 řádků zdrojového kódu, kde jsou započítána také vstupní data, komentáře a mezery pro zachování čitelnosti implementace. V případě oddělení dat od aplikační logiky byl počet řádků zdrojového kódu redukován téměř na polovinu. Tohoto výrazného zlepšení bylo dosaženo také díky eliminaci kódu obstarávajícího prezentaci dosažených výsledků.

Originální model nevyžadoval žádné externí podpůrné zdrojové texty či programy. To ovšem neplatí pro případ použití navrženého frameworku. Zde zpracování vstupních dat a naplnění QDX databáze zabralo přibližně 230 dalších řádků zdrojového kódu. Toto použití se může zdát na první pohled jako zbytečné a kontraproduktivní, ale v dalších částech srovnání těchto implementací bude popsáno, jaké výhody tím získáme.

Oddělení datové a aplikační logiky přináší mnoho výhod. První z nich je novupoužitelnost stejného modelu na různá vstupní data, možnost iterativních a přizpůsobujících se výpočtů na základě dosavadních výsledků. V originálním modelu byly výpočty prováděny pouze na 3 scénáře, zatímco s využitím navrženého frameworku bylo toto omezení eliminováno. Počet použitých scénářů je nyní možno dynamicky

upravovat a přidávat další scénáře. Počet scénářů je tak limitován pouze výpočetními možnostnmi počítače a licenčními omezeními GAMS. Je možno samozřejmě měnit libovolná data figurující ve výpočetním modelu, čím je zajištěna možnost efektivního experimentování.

Tab. 3.2: Srovnání optimalizačních implementací

Kritérium	**Originální model**	**Použití frameworku**
Rozsah GAMS kódu	360 řádků	183 řádků
Podpůrné zdroj. kódy	0	230
Data součástí modelu	Ano	Ne
Počet scénářů	3	Mnoho
Možnosti exportu	Txt	Txt, Excel, Grafy, HTML, ...
Přizpůsobitelnost	Náročná	Jednoduchá
Rozšířitelnost	Náročná	Jednoduchá
Intuitivnost modelu	-	Vyšší než originál
Modularita	Ne	Ano
Integrace dalších nástrojů	Ne	Ano
Komplexita implementace	-	Vyšší než originál
Výpočetní čas	0,28 s	0,38 s

S využitím dostupných nástrojů GAMS Studia máme velmi limitované možnosti exportu a interpretace dosažených výsledků. S využitím programovacího jazyka Python je situace naprosto opačná. Navržený frameworku disponuje možností jednoduchého automatizovaného exportu výsledků do textového souboru či sešitu aplikace MS Excel. Součástí je také modul pro vytváření efektních grafů. Uživatel disponující znalostmi v oblasti programování v Pythonu se ovšem nabízí možnosti přizpůsobení

chování optimalizačního jádra dle vlastních potřeb (export do databáze, webu, dalších nástrojů, aj.).

Při využití navženého optimalizačního frameworku uživatel dosáhne také větší intuitivnosti prováděných úkonů. Ve výchozím nastavení se framework řídí svými konvencemi, jako jsou například názvy funkcí a proměných, které uživateli napovídají, o co se daná část stará. S tím souvisí také modulárně strukturované části frameworku, což přispívá přehlednosti a udržitelnosti zdrojových textů. Na druhou stranu se komplikuje implementace, kdy uživatel kromě vytvoření výpočetního modelu GAMS musí provádět definice zpracování dat v programovacím jazyce Python.

Posledním sledovaným kritériem srovnání obou implementací byl výpočetní čas. Originální model za použití GAMS Studia byl zpracován z 0,28 s. Stejný model na identických dat s využitím navrženého frameworku byl zpracován za 0,38 s. Zde lze vidět, že původní implementace byla výpčoetně efektivnější, avšak uvažujeme-li pouze malá data. Pro zpracovávání velkých dat bychom dosahovali výrazně lepších výsledků za použití navrženého frameworku.

3.5 Shrnutí

Tato část dokumentace k diplomové práci se zabývala aplikací navrženého a implementovaného frameworku na reálný případ užití. Nejprve zde byly popsány způsoby a možnosti jeho použití. Následně byla popsána originální úloha. Jednalo se o optimalizační model Ing. Tetoura zabývající se optimalizací alokací výrobních zdrojů s ohledem na poptávané zboží. Aplikace se zabývala energetickou tématikou, kde cílem bylo minimalizovat náklady spojené s dodávkou elektrické energie při zachování naplnění poptávaného množství. Cílem bylo analyzovat tuto aplikaci, navrhnout změny vedoucí k efektivnějšímu zpracování a tyto změny otestovat vůči originálnímu řešení.

Následně byly ukázány a popsány úryvky zdrojového kódu originální řešení, zejména ty, na kterých se následná optimalizace projeví nejvíce. Jednalo se o prvky, které obsahovaly definici vstupních dat či algoritmus prezentace výstupních dat. Následně byla popsána adresářová struktura použita při využití optimalizačního frammeworku. Poté byly detailně popsány a na úryvcích zdrojového kódu demonstrovány principy využití zpracování vstupních dat, naplňování QDX databáze a použití aplikačního rozhraní mezi GAMS a programovacím jazykem Python.

Po verifikaci správnosti přepisu aplikace do optimalizačního frameworku bylo prováděno testování na větších datech a následně vyhodnocování rozdílů jednotlivých implementací. Srovnáváno bylo několik kritérií, mezi které spadaly rozsahy zdrojových textů, možnosti práce s danou aplikací či délka výpočetního času nutného k dokončení výpočtů na identických datech. Přestože došlo ke zlepšení celkové škálovatelnosti modelu a jeho využití, došlo také k navýšení komplexnosti zdrojových kódů podpůrných skriptů. Redukovala se složitost zdrojového kódu GAMS, ale na druhou stranu přibyla nutnost definice dat v programovacím jazyce Python. V 10 z 12 sledovaných kritérií došlo za použití optimalizačního frameworku ke zlepšení. Tato zlepšení se projeví zejména při testování a experimentování s výpočetním modelem na velkých datech.

Závěr

Tato práce se zabývala studiem problematiky optimalizačních úloh a strategických aplikací. Důraz byl kladen zejména na výběr vhodných algoritmů pro studované modely a efektivní softwarová aplikace. Pro optimalizační modelování a provádění výpočtů je vhodným nástrojem modelovací nástroj GAMS. Ten kromě integrovaného vývojového prostředí nabízí také možnost propojení aplikačního rozhraní výpočetního jádra s extermíni nástroji. Pro tuto práci bylo zvoleno prostudování možností integrace GAMS s programovacím jazykem Python. Po úspěšném propojení těchto dvou nástrojů byla dalším krokem efektivní softwarová implementace, která povede ke zlepšení provádění optimalizačních aplikací a experimentování s nimi.

Byly analyzovány požadavky na funkcionality takového optimalizačního frameworku. Obecný optimalizační proces obsahuje následující části definice úlohy, kolekce vstupních dat, vytvoření matematického modelu, vytvoření optimalizačního modelu, zpracování vstupních dat, import do optimalizačního modelu, provedení samotných výpočtů a zpracování dosažených výsledků. Kromě definice úlohy a matematického modelu je třeba, aby frameworku umožňoval efektivní práci se všemi zbylími částmi celého procesu. Na základě analýzy těchto požadavků byl vytvořen návrh a vhodná pracovní adresářová struktura. Následovala samotná implementace. Navržený optimalizační framework má modulární strukturu, která umožňuje existenci více projektů v jedné lokaci bez nutnosti vytváření kopií instancí jádra frameworku.

Při implementaci jádra bylo vycházeno z oficiální dokumentace aplikačního rozhraní softwaru GAMS. Nad těmito kódy byla dále implementována vhodná rozšíření, která obstarávají zpracování vstupních dat z různých zdrojů. Pozornost byla zaměřena zejména na zpracování strukturovaných dat z csv souborů. Byly vytvořeny také pomocné skripty umožňující jednoduché generování vzorku vstupních dat libovolné velikosti. Zároveň byla také vytvořena knihovna ukázkových implementací, které pokrývají většinu možných scénářů použití optimalizačního frameworku. Součástí této knihovny jsou dereministické i stochastické modely, jednostupňové í vícestupňové rozhodovací modely a scénářové modely pokrývající také problematiky modelů se scénářemi scénářů.

Dalším krokem bylo zhodnocení komplexnosti použití a měření výpočetní výkonnosti frameworku oproti standardnímu použití softwaru GAMS. Komplexnost dokončení úlohy byla analyzována pro zpracování jednoduché modelu i scénářového modelu, zpracování vstupních a výstupních dat, udržení čitelné struktury zdrojového kódu a možnosti exportu dosažených výsledků. Pro hodnocení komplexnosti požadované úlohy, nebo-li náročnosti jejího dokončení, byla stanovena hodnotící škála a vytvořena matice ukazující náročnost dokončení požadované operace při standardním použití GAMS oproti použití frameworku. Zatímco některé operace se ukázaly být s využitím pouze GAMS nedokončitelné, s využitím navrženého frameworku bylo jejich provedení naopak velmi efektivní.

Následovalo hodnocení výkonnosti, které bylo realizováno měřením času nutného k dokončení požadované operace. Testování probíhalo na stejném modelu a identických datech. V přípapě sledování výkonnosti navrženého frameworku byl sledován čas nutný k dokončení úlohy pro samotný výpočet i všechny kombinace možností exportu dosažených výsledků. Testování probíhalo na vzorku dat čítající určitý počet scénářů, kdy se s každou další iterací tato velikost zvyšovala. Testy byly prováděny na 1 - 100 000 vstupních scénářů. Na větších vzorcích vstupních dat nebylo testování prováděno z důvodu omezených zdrojů (domácí laptop). Výsledky tohoto testování ukázaly, že čas nutný k dokončení úlohy se při použití na malých datech pro GAMS Studio a navržený framework příliš nelišily. Zlomový bod pro vybraný terstovací model nastal při experiment na 10 000 scénářích. Zde bylo použití frameworku více než třikrát efektivnější. Při experimentech na velkých datech, kdy vstupní vzorek čítal 100 000 scénářů došlo při použití navrženého frameworku k 85% úspoře času.

Poslední částí diplomové práce byla aplikace implementovaného frameworku na reálná data. K tomuto účelu byl vybrán optimalizační model Ing. Tetoura, který se zabýval optimalizací alokace výrobních zdrojů v energetickém průmyslu. Nebylo cílem vylepšit navržený matematický ani výpočetní model, nýbrž aplikovat navržené principy na zlepšení práce s tímto modelem a analyzovat možnosti jeho rozšíření. V kapitole zabývající se touto problematikou byly podrobně popsány způsoby integrace Tetourova modelu do optimalizačního frameworku. Po úspěšném oddělení datové a aplikační logiky byly následně prováděny experimenty na dynamicky se mě-

nících vstupních datech. Závěrem bylo provedeno zhodnocení dosažených výsledků. Po odstranění statických dat ze zdrojového souboru výpočetního modelu byla i při zachování komentářů a struktury GAMS kódu velikost tohoto souboru takřka poloviční. Bylo nutné využít několik podpůrných skriptů v modulech optimalizačního frameworku, ale celkově došlo ke značnému zpřehlednění výpočetního modelu. Další velkou výhodou byly možnosti experimentů na velkých dynamicky se měnících simulovaných datech a možnošti exportu dosažených výsledků nejen do textových souborů.

Literatura

[1] BIRGE, John R. a Francois LOUVEAUX.: *Introduction to Stochastic Programming. 2nd edition,* New York: Springer, 1997. ISBN 0-387-98217-5.

[2] Williams, H.P.: *Model Building for Mathematical Programming, 5th edition,* Wiley and Sons, 2013.

[3] Janíček, P., Marek, J.: *Expertní inženýrství v systémovém pojetí.* Praha: Grada, 2013. Expert (Grada). ISBN 978-80-247-4127-7

[4] SMEJKAL, Vladimír a Karel RAIS.: *Řízení rizik ve firmách a jiných organizacích.* 2., aktualiz. a rozš. vyd. Praha: Grada, 2006, 296 s. Expert (Grada). ISBN 80-247- 1667-4.

[5] TICHÝ, Milík.: *Ovládání rizika: analýza a management.* Praha: C.H. Beck, 2006. Beckova edice ekonomie. ISBN 80-717-9415-5.

[6] NASH, S. et al.: *Linear and nonlinear programming*, McGraw-Hill, 1995.

[7] Kall, P., Wallace, S. W.: *Stochastic Programming,* Wiley, 1994.

[8] POPELA P.: *Stochastic programming*, Brno University of Technology, Institute of Mathematics - Division of Stochasting and Optimization Methods

[9] A. SHAPIRO a A. Philpott.: *A Tutorial on Stochastic Programming,* Atlanta: Georgia Institue of Technology, 2007.

[10] KALVELAGEN E.: *Two stage stochasting linear programming with GAMS* , http://www.amsterdamoptimization.com/pdf/twostage.pdf

[11] VITÁSEK, V.: *Dvoustupňový přístup k úlohám lineárního programování,* Vysoké učení technické v Brně, Fakulta strojního inženýrství - Ústav matematiky. Brno.

[12] McCARL B., A. Meeraus, P. Van der Eijk, M. Bussieck.: *GAMS User Guide*, GAMS Development Corporation, Washington, 2012.

[13] GAMS Development Corporation. *Gams Documentation Center* [online]. Washington, DC, USA., 2021 [cit. 2021-04-28]. Dostupné z: https://www.gams.com

[14] TETOUR, Daniel.: *Optimalizační modely rizik v energetických systémech.* Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Ústav soudního inženýrství, Odbor inženýrství rizik. Vedoucí práce RNDr. Pavel Popela, PhD.

Seznam obrázků

Seznam tabulek

Seznam výpisů

Seznam příloh

A Obsah přiloženého média

Součástí přiloženého média jsou kompletní zdrojové kódy implementovaného frameworku včetně knihovny ukázkových příkladů použití. V kořenovém adresáři se kromě hlavního skriptu nachází také soubor *ReadMe.md* obsahující návod ke zprovoznění frameworku.

```
/ ..... kořenový adresář přiloženého CD
├── data ..... Adresář s moduly zpracovávající vstupní data
│   └── library ..... Knihovna ukázkových příkladů
│       ├── cpm ..... Příprava dat CPM úlohy (různé modely)
│       │   ├── csv ..... Soubory se vstupními daty
│       │   ├── data_ev.py
│       │   ├── data_is.py
│       │   ├── data_mm.py
│       │   ├── data_ts.py
│       │   ├── data_ts_evpi_eev_vss.py
│       │   └── data_ws.py
│       ├── farm ..... Příprava dat farmářovy úlohy
│       │   └── data.py
│       ├── tetour ..... Příprava dat úlohy alokace zdrojů
│       │   ├── datafiles
│       │   ├── scenarios
│       │   │   └── generate_scenarios_data.py ..... Generování simulačních dat
│       │   └── data.py
│       └── transport ..... Příprava dat transportní úlohy
│           └── data.py
├── gams ..... Adresář se zdrojovými soubory GAMS
│   └── library ..... Knihovna ukázkových příkladů
│       ├── cpm ..... CPM úloha - různé GAMS modely
│       │   ├── cpm_ev.gms
│       │   ├── cpm_is.gms
│       │   ├── cpm_mm.gms
│       │   ├── cpm_ts.gms
│       │   ├── cpm_ts_evpi_eev_vss.gms
│       │   └── cpm_ws.gms
│       ├── farm ..... Farmářova úloha
│       │   └── farm.gms
│       ├── tetour ..... Úloha alokace zdrojů
│       │   └── tetour.gms
│       └── transport ..... Transportní úloha
│           └── transport.gms
└── gams ..... Skripty pro inicializaci model
    └── library ..... Knihovna ukázkových příkladů
```

```
        ├── cpm ..... CPM úloha
        │   └── model.py
        ├── farm ..... Farmářova úloha
        │   └── model.py
        ├── tetour ..... Úloha alokace zdrojů
        │   └── model.py
        └── transport ..... Transportní úloha
            └── model.py
├── settings ..... Nastavení aktivního modelu
│   └── optimizer_settings.py
├── measure_time.bat ..... Utilita pro měření výpočetního času
├── optimizer.py ..... Řídící program
├── ReadMe.md ..... Návod ke zprovoznění frameworku
├── requirements.txt ..... Knihovny nutné k instalaci
└── docs.pdf ..... Dokumentace k diplomové práci
```