České vysoké učení technické v Praze
Fakulta stavební
Katedra hydrotechniky

DIPLOMOVÁ PRÁCE

Simulace provozu MVE s využitím bateriového úložiště

Hydropower plant simulation with battery storage

Vedoucí diplomové práce: Dr. Ing. Petr Nowak

Leden 2024

Bc. Kraus Jan

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta stavební
Thákurova 7, 166 29 Praha 6

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: Kraus Jméno: Jan Osobní číslo: 477533

Zadávající katedra: K142 - Katedra hydrotechniky

Studijní program: N3607 - Stavební inženýrství

Studijní obor/specializace: 3607T027 - Vodní hospodářství a vodní stavby

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce: Simulace provozu MVE s využitím bateriového úložiště

Název diplomové práce anglicky: Hydropower plant simulation with battery storage

Pokyny pro vypracování:
Sestavení zjednodušeného modelu bateriového úložiště provozovaného v rámci MVE. Model bude umožňovat simulaci provozu se zahrnutím:
- Běžného provozu soustrojí bez akumulace vody v režimu hladinové regulace. Budou dodány časové průběhy reálných provozních dat
- Skutečný průběh hodinových cen v režimu denního trhu – veřejně dostupná data
- Omezení maximálního dodávaného výkonu do sítě podle povolení o připojení k distribuční síti), tj. nebude možné překročit daný výkon
- Výstupem modelu bude výroba a celková výkupní cena za vyrobenou elektřinu

Do modelu bude následně přidáno schematizované bateriové úložiště s omezujícími parametry:
- Maximální výkon nabíječky a střídače
- Maximální kapacita bateriového úložiště
- Zjednodušeně budou zohledněny ztráty aspoň ve formě pevné účinnosti jednotlivých prvků
- Úložiště bude řízeno na základě maximalizace celkového příjmu z provozu

Bude provedena citlivostní analýza na dva základní parametry bateriového úložiště
- Výkon střídače, resp. nabíječky
- Kapacita úložiště
- Výsledky simulací budou interpretovány jako funkce – celkový příjem = fce(výkonu a kapacity)

Seznam doporučené literatury:
https://www.ote-cr.cz; https://www.energy-charts.info; https://pxe.cz; https://www.eex.com; https://www.fraunhofer.de; https://www.xflexhydro.com/; https://spotmarketindex.cz/; https://faktaoklimatu.cz/

Jméno vedoucího diplomové práce: Dr. Ing. Petr Nowak

Datum zadání diplomové práce: 25.9.2023 Termín odevzdání DP v IS KOS: 8.1.2024

Údaj uveďte [redacted] *oku*

[redacted]

Podpis vedoucího práce

III. PŘEVZETÍ ZADÁNÍ

Beru na vědomí, že jsem povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je nutné uvést v diplomové práci a při citování postupovat v souladu s metodickou příručkou ČVUT „Jak psát vysokoškolské závěrečné práce" a metodickým pokynem ČVUT „O dodržování etických principů při přípravě vysokoškolských závěrečných prací".

27.9.2023

Datum převzetí zadání

[redacted]

Podpis studenta(ky)

PROHLÁŠENÍ

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s metodickým pokynem ČVUT 1/2009 „O dodržování etických principů při přípravě vysokoškolských závěrečných prací“.

V Praze dne 8.1.2024 podpis:

PODĚKOVÁNÍ

Rád bych tímto poděkoval Dr. Ing. Petru Nowakovi za jeho čas, odborné i praktické rady, vstřícný přístup, a poskytnutí podkladů. Také bych rád poděkoval Ing. Jiřímu Součkovi za veškeré konzultace a jeho rady. Dále doc. RNDr. Petru Mayerovi, Ph.D., za vstřícné konzultace matematický záležitostí v oblasti lineárního programování a optimalizace.

ABSTRAKT

Cílem této diplomové práce je vytvořit matematický model, který bude schopen simulovat provoz malé vodní elektrárny s bateriovým úložištěm, pomocí softwaru Matlab a Simulink. Díky bateriovému úložišti může vodní elektrárna lépe respektovat nabídku a poptávku po elektrické energii, napomoct tak vyrovnávat výkyvy v síti, maximalizovat zisk nebo prodloužit životnost soustrojí. U vytvořených modelů dochází k optimalizaci prodávané elektřiny a práci baterie, tak aby byl maximalizován zisk pomocí lineárního programováním, konkrétně pro Matlab funkce *linprog*. Tento program, by mohl sloužit k posouzení investice do bateriového úložiště a zjištění její návratnosti. Model vytvořený v Simulinku (nástavbě programu Matlab), je vytvořen pro optimalizaci a řízení, nabíjení a vybíjení bateriového úložiště u malé vodní elektrárny v reálném čase, se zahrnutím ztrát a dalších provozních prvků.

ABSTRACT

The aim of this thesis is to create a mathematical model capable of simulating the operation of a small hydroelectric power plant with a battery storage system, using Matlab and Simulink software. Thanks to the battery storage, the hydroelectric power plant can better respect the supply and demand for electrical energy, thereby helping to balance fluctuations in the network, maximize profit, or extend the lifespan of the **turbine**. The developed models optimize the electricity sold and the operation of the battery to maximize profit using linear programming, specifically the Matlab linprog function. This program could serve to assess the investment in battery storage and determine its return on investment. The model created in Simulink (an extension of the Matlab program) is designed for the optimization and control of charging and discharging of the battery storage at a small hydroelectric power plant in real-time, including losses and other operational elements.

KLÍČOVÁ SLOVA

malá vodní elektrárna, bateriové úložiště, simulace, hybridní elektrárna, výroba elektrické energie, optimalizace energetických systémů, matematický model, Lineární programování, Matlab, Simulink, Stabilizace energetické sítě, Návratnost investice pro bateriové úložiště, Regulační strategie baterií, Obnovitelné zdroje energie

KEYWORDS

Small hydroelectric power plant, Battery storage, Simulation, Hybrid power plant, Electricity generation, Optimization of energy systems, Mathematical model, Linear programming, Matlab, Simulink, Stabilization of the energy network, Return on investment for battery storage, Battery management strategies, Renewable energy sources.

Obsah

1. Úvod

Cílem této diplomové práce je vytvoření matematického modelu, který umožní simulaci provozu malé vodní elektrárny s bateriovým úložištěm. Model bude provádět optimalizaci prodeje elektrické energie do sítě a na základě toho řídit práci baterie, aby došlo k maximalizaci zisku.

Tento revoluční přístup u malých vodních elektráren by mohl vést nejen ke zlepšení ziskovosti, ale také k prodloužení životnosti turbíny, lepší odezvě na výkyvy elektrické sítě a jejich kompenzaci.

Pomocí tohoto modelu bude možné analyzovat ziskovost vybudování bateriového úložiště u existující malé vodní elektrárny a určit návratnost. Tyto data mohou být klíčová pro rozhodování o budoucích projektech a jejich plánování.

Díky softwaru Matlab a Simulink bude možné model využít i pro simulaci jiných malých vodních elektráren se specifickými parametry a poskytnout data pro další rozhodování o projektu.

2. Bateriové úložiště ve spojení s malou vodní elektrárnou

Definice malé vodní elektrárny jsou různé, obecně za ni můžeme považovat elektrárnu s výkonem do 10MW, avšak Evropská unie stanovuje horní hranici na 5MW. Malá vodní elektrárna představuje ekologicky šetrný způsob výroby elektrické energie a v současné době se stávají součástí mnoha vodních staveb. [1]

Bateriové úložiště můžeme s vodní elektrárnou použít na krátkodobou a střednědobou akumulaci elektrické energie. Hlavními výhodami může být zvýšení flexibility, efektivnosti a prodloužení životnosti soustrojí. U malých vodních elektráren s bateriovým úložištěm dojde k ukládání elektrické energie v dobách s nízkou poptávkou a k následnému využití při vyšší poptávce, což může výrazně zvýšit efektivnost jejich využití.

Sektor bateriových úložišť je v dnešní době rychle rostoucím odvětvím, které nabízí flexibilní a modulární řešení prostřednictvím blokových bateriových systémů. Tyto systémy se vyznačují rychlou reakční dobou, což umožňuje téměř okamžitě reagovat na změny v poptávce a nabídce elektrické energie nebo výkyvy elektrické sítě. Díky schopnosti rychle poskytnout

vysoký výkon, mohou bateriová úložiště efektivně vyrovnávat krátkodobé výkyvy v síti bez nutnosti regulovat výkon elektrárny.

Kvůli stále rostoucím výkyvům v elektrické síti a potřebě vyrovnávat tyto nerovnosti, musí vodní elektrárny častěji reagovat na tyto potřeby a měnit výkon mimo optimum. Tento způsob řízení ale značně zatěžuje vodní turbíny a soustrojí, což vede ke značné snížení životnosti. Díky bateriovému úložišti by turbína nemusela tak rychle reagovat a měnit svůj výkon, protože vyrovnávání nerovností by bylo primárně pokryto energií naakumulovanou v bateriích.

Toto je další využití a výhoda bateriového úložiště ve spojení s vodní elektrárnou, která může být hodně podstatná hlavně u velkých kaplanových turbín, které při náhlé regulaci výkonu hodně trpí. Hybridní řešení velkých vodních elektráren by mohlo značně zvýšit jejich životnost, flexibilitu a zároveň i ziskovost. Touto oblastí se zabývá projekt X-Flex Hydro, jehož modely vyvinuté pro provoz hybridního systému zkoumají snížení opotřebení u velkých soustrojí, zlepšení schopnosti poskytovat primární frekvenční odezvu a návratnost hybridizace velkých vodních elektráren.

2.1. Aplikace bateriového úložiště v České republice

V současné době zatím není mnoho elektráren s bateriovým úložištěm, ale tato oblast se stává čím dál tím více populární a zkoumaná. V České republice se s aplikací bateriového úložiště setkáme především ve spojení s elektrárnami na tuhá paliva.

Jedním z příkladů je bateriové úložiště spojené s teplárnou na tuhá paliva v C-Energy Planá, které má výkon 4 MW, kapacitu 2,5 MWh garantovanou po dobu 10 let a účinnost ukládání energie téměř 90 %. Jedná se o jedno z největších zařízením v České republice. Toto úložiště pomáhá vyrovnávat výrobu energie z obnovitelných zdrojů, jako je sluneční a větrná energie. [2] [3]

Ještě větší bateriové úložiště vybudovala na Sokolovsku společnost SUAS Group ve spolupráci s Energetickým investičním fondem (EIF, a.s.), s kapacitou 7,45 MWh. Hlavním účelem tohoto úložiště je udržování stability elektrické sítě a poskytování podpůrných služeb pro provozovatele přenosové soustavy ČEPS. Generálním dodavatelem celého řešení se stala společnost Sokolovská uhelná a dodávku baterií zajistila společnost Alfen z Nizozemska. Bateriové úložiště je součástí plánovaného Energy Hubu, který má propojit různé zdroje energie. [4]

Dále v České republice probíhá projekt výstavby desetimegawattového bateriového úložiště v areálu Energocentra Vítkovice v Ostravě, který měl být dokončen v průběhu letošního léta. Po dokončení se stane největším zařízením svého druhu v České republice. Bateriové úložiště bude mít výkon 10 MW a úložnou kapacitu 9,45 MWh. Dle aktuálních informací by měl být projekt spuštěn do konce roku. Bohužel více aktuálních informací o stavu výstavby, nebo testování není dostupných. [5] [6]

Obrázek 2.1 – Plánované energetické úložiště u Vraňan [7]

Další rekordní baterie je budována na Mělnicku u Vraňan společností E.nest. Toto bateriové úložiště, by mělo být dokončeno v průběhu příštího roku, s instalovaným bateriovým výkonem 20 MW a kapacitou 22 MWh. Dále by zde měly být instalovány spalovací turbíny na zemní plyn, s celkovým výkonem 30 MW. Celé toto nově vybudované energetické centrum umožní částečnou náhradu výkonu uhelných elektráren. [7]

V České republice, se dle dostupných informací, zatím nesetkáme se spojením vodní elektrárny a bateriového úložiště. Zatím se jako energetická úložiště ve spojení s vodními elektrárnami využívají přečerpávací vodní elektrárny, například elektrárna Dlouhé stráně v Jeseníkách, nejvýkonnější z těchto zařízení v Česku. Tyto elektrárny ovšem nenahrazují možné funkce bateriového úložiště, proto si myslím, že by do budoucna neměli být jedinou možností akumulace energie u vodních elektráren na kterou se budeme zaměřovat. Na konferenci akumulace energie, bylo také uvedeno, že ČEZ připravuje úpravu vodní elektrárny Orlík. Dvě ze čtyř turbíny by měli umožňovat využití této vodní nádrže i jako přečerpávací vodní elektrárnu. Dále se také mluví o aplikaci bateriového úložiště a hybridizaci Vltavské kaskády, více informací o tomto projektu není bohužel dostupných. [8]

2.2. Bateriové úložiště ve spojení s vodní elektrárnou ve světě

Na světové scéně se projekty spojení vodní elektrárny a bateriového úložiště začínají postupně objevovat, ale zatím mnohem častěji najdeme kombinaci bateriového úložiště s jiným zdrojem.

Jednu z aplikací bateriového úložiště a elektrárny najdeme v provincii Ontario v Kanadě, kde jsou stovky malých vodních elektráren, které mají omezenou možnost regulace. Propojením s bateriovým úložištěm došlo k vytvoření systému, který umožňuje skladování energie během období s nízkou poptávkou a její využití v obdobích s vysokou poptávkou, což zvyšuje efektivitu provozu těchto malých vodních elektráren a spolehlivost dodávky pro odběratele. [9] [10]

V Bavorsku, v oblasti Pfreimd, byl zaveden kombinovaný systém vodní elektrárny s akumulačním čerpáním a bateriového úložiště. Tento projekt byl realizován společností Engie Deutschland. Jedná se o síť elektráren sestávající se ze tří nádrží, jedné elektrárny na toku a dvou elektráren s akumulačním čerpáním v Horním Falckém lese. V rámci modernizace zde byl vybudován 12,5MW systém bateriového úložiště, které dodala společnost Siemens. Toto bateriové úložiště bylo společností Siemens navrženo a postaveno na základě jejich platformy pro kontejnerové bateriové úložiště Siestorage. Baterie budou poskytovat primární vyrovnávací služby a díky flexibilitě bateriového úložiště napomáhat integraci proměnlivých zdrojů obnovitelné energie. [11]

2.3. Evropský projekt XFLEX Hydro

V Evropě vznikl v souvislosti s vodní elektrárnou a bateriovým úložištěm, již zmiňovaný projekt XFLEX Hydro, který zkoumá, jak může bateriové energetické úložiště (BESS) zvýšit flexibilitu, účinnost a životnost vodních elektráren.

Tento projekt je financovaný Evropskou unií, byl oficiálně zahájen na klimatické konferenci OSN v Madridu v prosinci 2019. V rámci projektu dochází k testování různých technologií v několika vodních elektrárnách po Celém světě, především v EU. Cílem projektu je zlepšit vyrovnávání stability sítě poskytované hydroelektrárnami a najít cestu k efektivnějšímu a šetrnějšímu využití vodních elektráren a spolehlivosti sítě.

Na projektu XFLEX HYDRO se podílí nemálo institucí, včetně energetických společností, výrobců energetických zařízení, univerzit a výzkumných institucí. Celý tento projekt vede švýcarský výzkumný institut a univerzita Ecole Polytechnique Fédérale de Lausanne.

V rámci projektu budou testovány systémy turbín s proměnlivou a pevnou rychlostí, hybridní technologie baterie-turbína, spolu s chytrou řídící technologií nazvanou Smart Power Plant Supervisor. Tato řídící technologie využívá vícedimenzionální systém pro pokročilé řízení, optimální výkon, a prediktivní údržbu soustrojí. Projekt také zkoumá vliv zařazení bateriového úložiště na životnost celého soustrojí.

Instalovaná baterie může totiž sloužit jako hlavní prvek pro vyrovnání nestabilit v elektrické síti. Díky tomu nemusí turbína náhle měnit výkon, což má pozitivní dopad na její životnost, kterou dokáže podle dosavadních informací i zněkolikanásobit.

Hodně náchylné na rychlé změny výkonu jsou rychloběžné Kaplanovy turbíny. V rámci projektu XFLEX HYDRO, je tato souvislost testována na vodní elektrárně Vogelgrun na řece Rýn ve Francii, která má čtyři Kaplanovy turbíny. Pro testování byla turbína modernizována přidáním bateriového úložiště.

Tento celý projekt je významným krokem ve vývoji vodních elektráren v kombinaci s bateriovým úložištěm a měl by přinést cenné nové informace. [12] [13]

3. Optimalizační metody v Programu MATLAB

Optimalizační metody jsou nástroj, který se používá ve spoustě různých odvětví (inženýrství, informatika, ekonomie, …). Jeho cílem je nalezení nejlepšího řešení dané úlohy, na základě kritérií, jakou jsou různé podmínky a omezení.

MATLAB poskytuje velmi výkonné výpočetní prostředí, ve kterém lze řešit různé úlohy, pomocí vlastního programovacího jazyku MATLAB. Tato platforma, nabízí také široké množství možností, včetně speciálních nástrojů a funkcí pro optimalizaci.

3.1. Lineární programování

Jedná se o jednu z nejjednodušších metod optimalizace, a je vhodná pro úlohy, kde jsou všechny funkce (objektivní i omezení) lineární. Velkou výhodou je rychlost a garantované nalezení opravdového globálního optima (pokud existuje). [14]

Pro aplikaci na úlohu vodní elektrárny a bateriového úložiště je tato metoda přijatelná, pokud budeme používat konstantní nebo lineární ztráty a proměnné.

3.1.1. Lineární programování v programu MATLAB s využitím funkce „linprog“

V programu MATLAB, lze řešit úlohu lineárního programování (LP) pomocí funkce *linprog*. Tato funkce hledá minimum objektivní funkce s omezením pomocí lineárních rovnic.

Volání funkce:

x = linprog(f,A,b,Aeq,beq,lb,ub,options)

Kde:

- f = objektivní funkce, kterou chceme minimalizovat
- A , b = definují nerovnicová omezení A*x < b
- Aeq, beq = definují rovnicová omezení Aeq*x = beq
- lb, ub = dolní a horní okrajové podmínky pro proměnné x
- options = specifikuje různé možnosti pro optimalizace

Funkci *linprog* lze také použít pouze s některými parametry, například:

x = linprog(f, A, b)

Ukázkový příklad použití funkce linprog:

Zadání:

Minimalizace objektivní funkce

$-x1 - 2*x2$

+ omezení

$x1 + x2 \leq 2$

$x1 - x2 \leq 1$

a

$x1, x2 \geq 0$

řešení v MATLABU:

```
% Minimalizace objektivní funkce
% -x1 – 2*x2
% + omezení
% x1 + x2 ≤ 2
% x1 - x2 ≤ 1
% a
% x1, x2 ≥ 0
clear all


f = [-1; -2];            % koeficienty objektivní funkce
A = [1 1; 1 -1];         % maticová reprezentace nerovnicových omezení
b = [2; 1];              % pravá strana nerovnicových omezení
lb = [0; 0];             % dolní meze
x = linprog(f, A, b, [], [], lb)
```

```
Optimal solution found.
x = 2×1
     0
     2
x = 2×1
Rows 1:2
     0
     2
```

```
% pro výpočet hodnoty objektivní funkce dosadím do původní rovnice
%tudíž: f(0,2)=0-2*(2)= -4
```

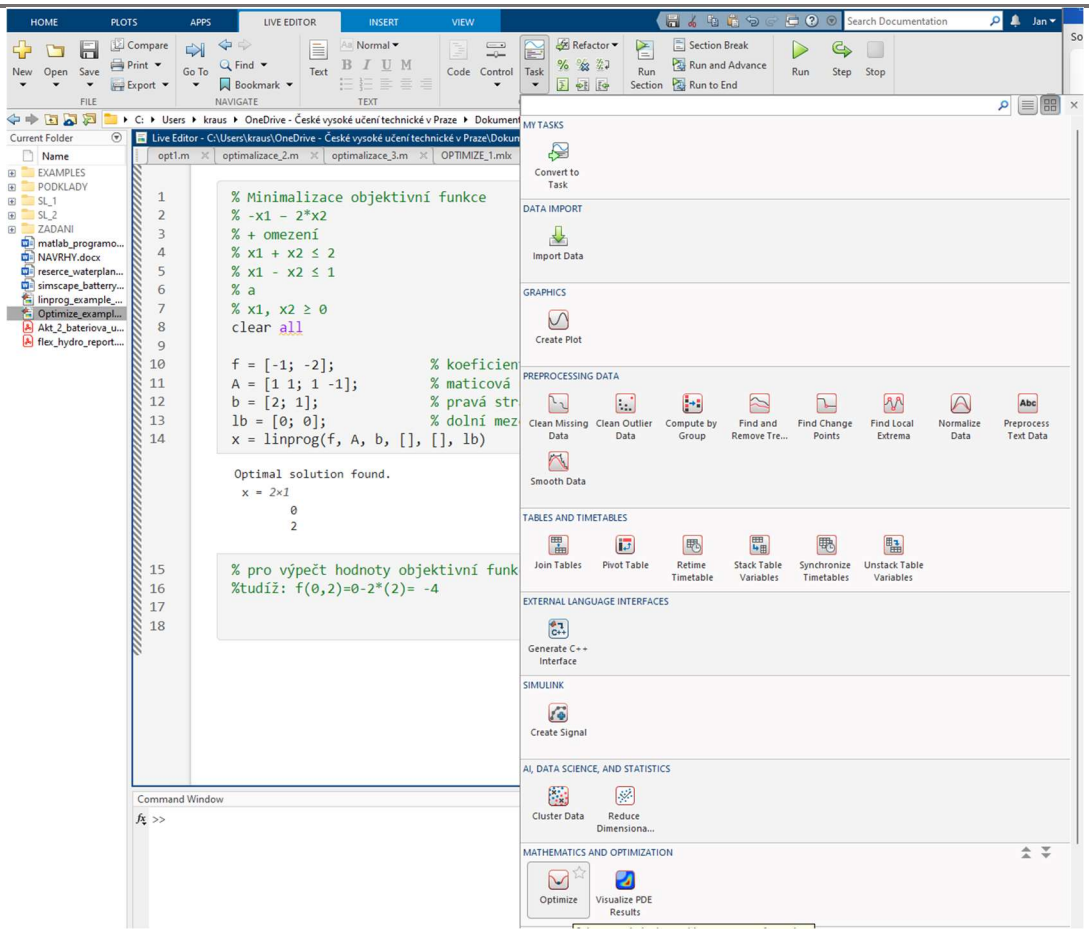
[15]

3.1.2. Lineární programování v programu MATLAB s využitím *„Optimize live editor“*

Jedná se o nástroj v programu MATLAB, který umí také optimalizovat pomocí lineárního programování, často se také nazývá *Optimalization Toolbox*. Veškeré parametry jsou zde ale zadávány pomocí grafického, uživatelsky přívětivějšího prostředí. Program automaticky vytvoří rovnice pro lineární programování a najde řešení (pokud existuje). Tento grafický optimalizační nástroj navíc nabízí další optimalizační možnosti.

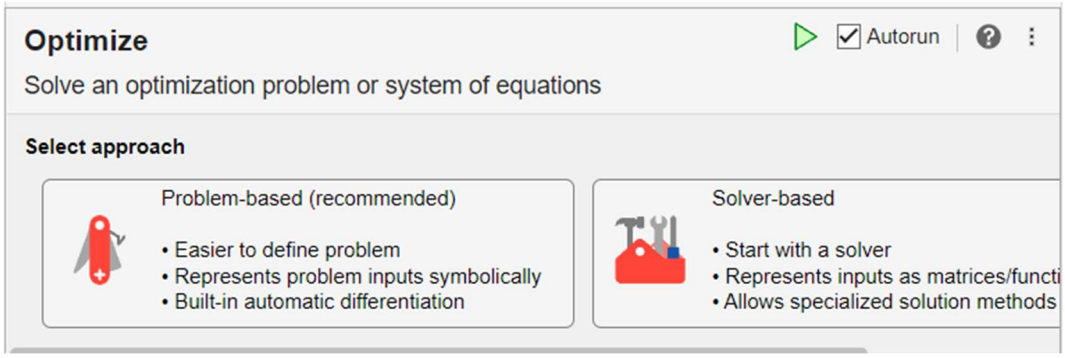
Spuštění a nastavení:

- Otevřeme MATLAB Live Script
- V horním panelu vybereme kartu „LIVE EDITOR“ a najdeme ikonu „Task“ (v sekci „CODE“), následně vybereme možnost „Optimize“ (druhá polovina nabídky)



Obrázek 3.1 - Spuštění Optimalization toolbox

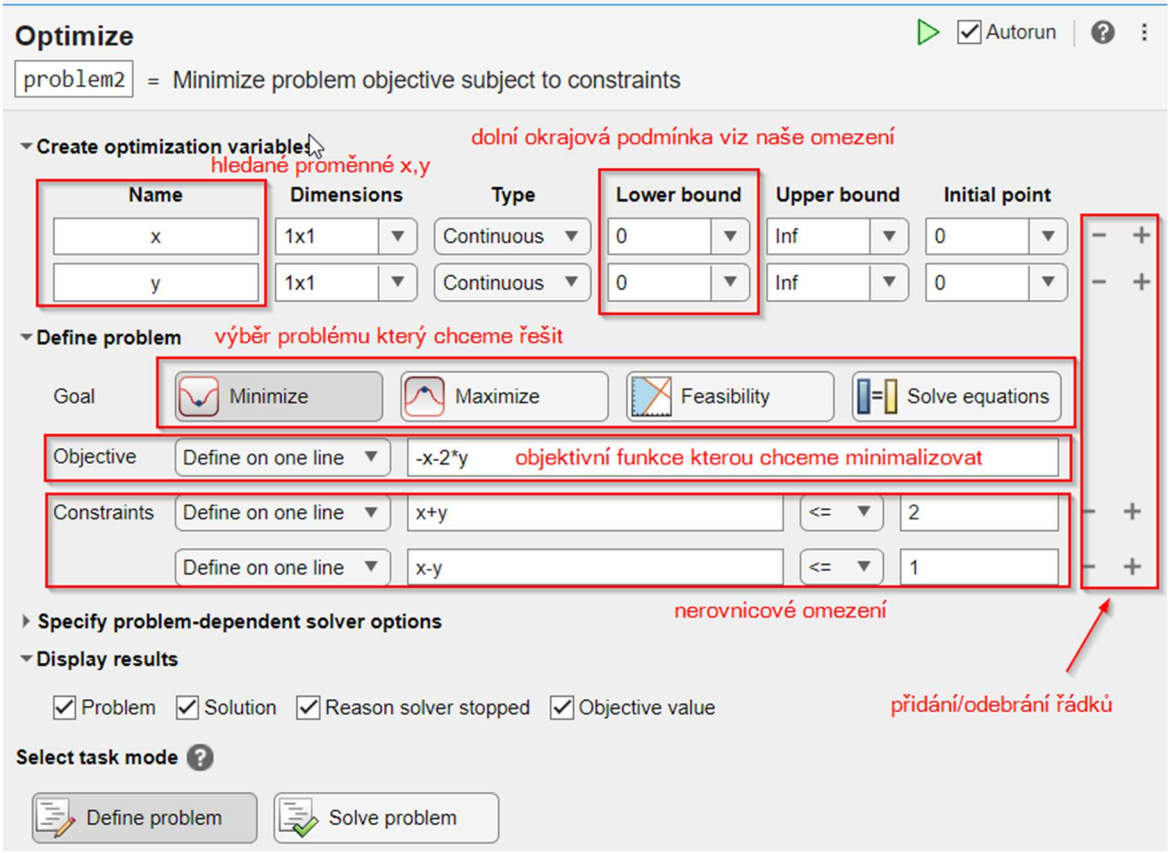
- Tím se nám do MATLAB Skriptu vloží grafický nástroj *Optimize* pro optimalizaci
- Pro relativně jednoduché úlohy se doporučuje zvolit možnost *Problem based*.



Obrázek 3.2 - Optimalization toolbox

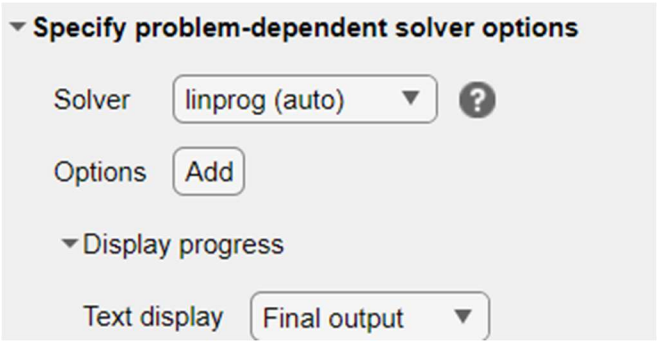
- Následně vyplníme všechny vstupní parametry
- Řešení stejného problému jako s *linprog*
- V grafickém rozhraní zadáme veškeré parametry pro řešení

```
% Minimalizace objektivní funkce
% -x1 – 2*x2
% nutno upravit na odližné variables (např: x,y)
% tudíž x1 = x ; x2 = y ==>  -x-2*y
%
% omezení tedy bude
% x + y ≤ 2
% x - y ≤ 1
% a
% x, y ≥ 0
```



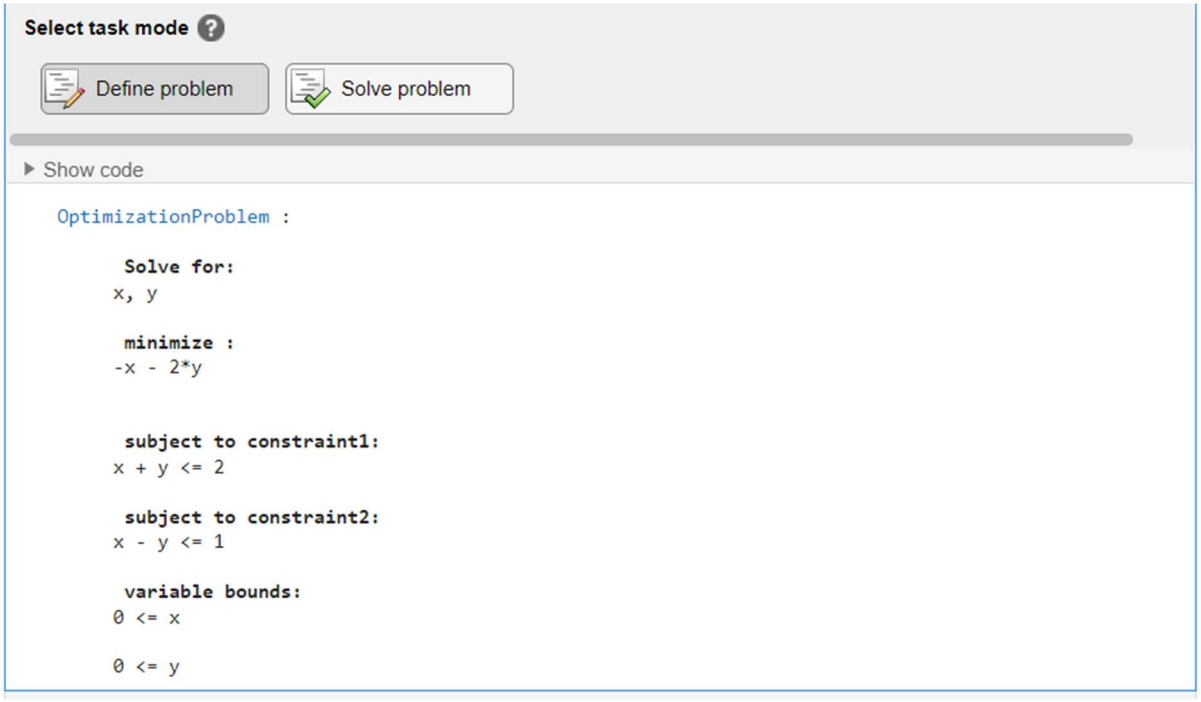
Obrázek 3.3 - Zadání parametrů do optimalization toolbox

- Nastavení řešení necháme na předvolených možnostech (program bude řešit zadaný problém pomocí *linprog*)



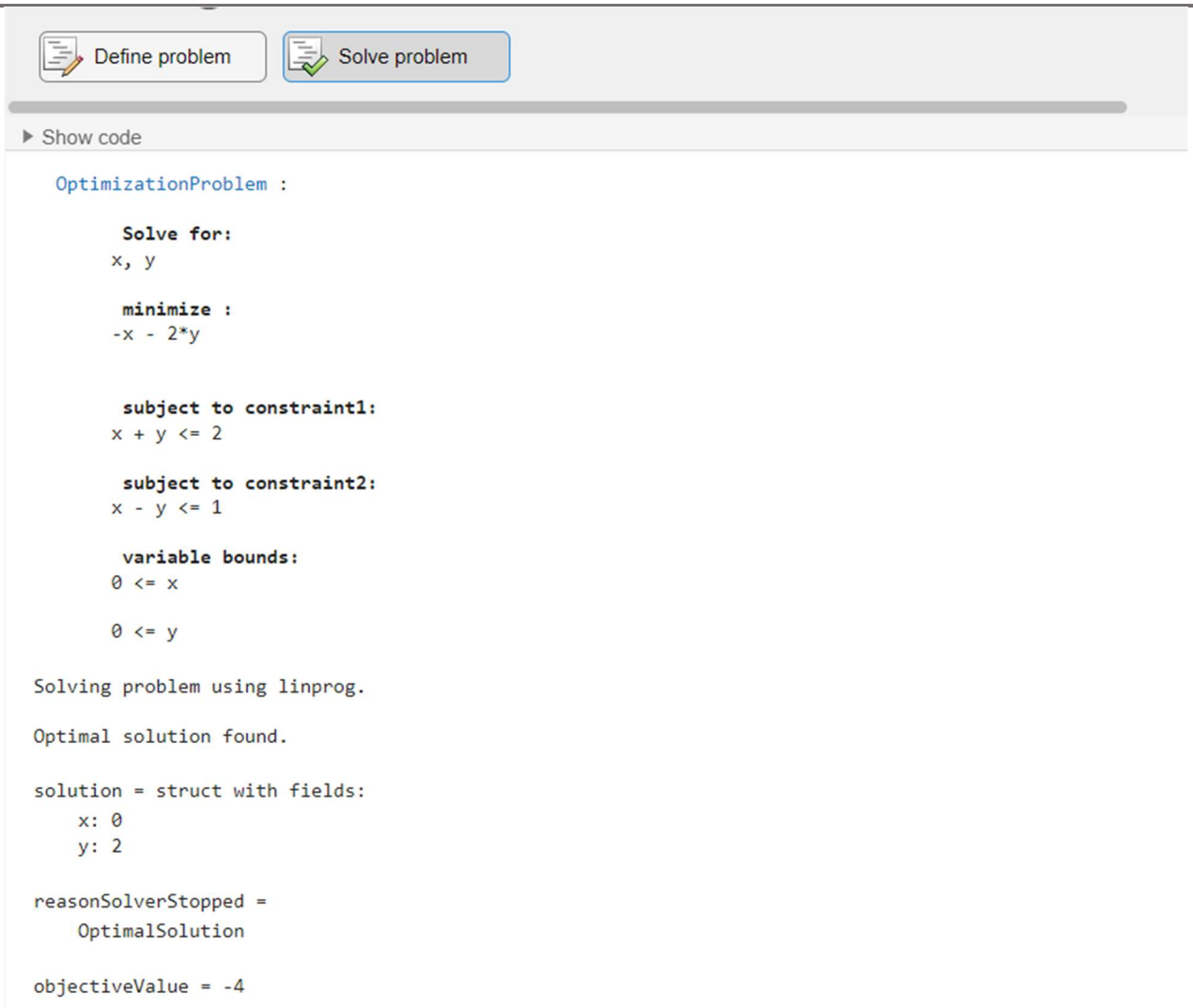
Obrázek 3.4 - Optimalization toolbox – výběr řešiče

- Následně tento nástroj provede automatickou formulaci problému do jazyku MATLAB



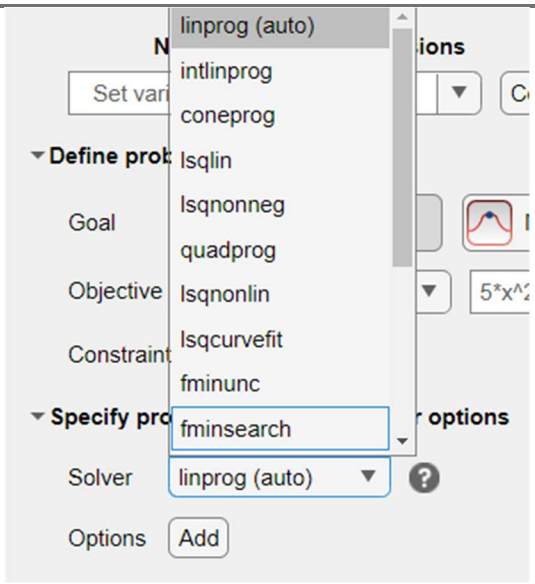
Obrázek 3.5 - Automatická formulace problému v Optimalization toolbox

- Poté můžeme přepnout na *Solve problem*
- Nyní dojde k automatickému vyřešení pomocí *linprog*. Pokud program narazí na chybu v zadání, řešení nebude nalezeno a zobrazí se chybová hláška



Obrázek 3.6 - Řešení a výsledek pomocí Optimalization toolbox

Tento nástroj umí také řešit úlohy nelineárního programování, například pomocí funkce *fminsearch* a další optimalizační problémy, díky možnosti přepnutí způsobu řešení.



Obrázek 3.7 - Další možnosti optimalizace pomocí nástroje Optimalization toolbox

3.2. Nelineární programování

Tento způsob hledání optimálního řešení je vhodný, když má řešená úloha nelineární omezení nebo nelineární objektivní funkci. Největší výzvou pro nelineární programování je možnost existence více lokálních optim, což komplikuje nalezení optima globálního.

Reálné problémy z praxe (inženýrství, ekonomie, …) budou velmi často patřit právě do této kategorie. V kontextu malé vodní elektrárny a bateriového úložiště by to mohlo znamenat, že například ztráty při nabíjení či vybíjení jsou nelineární.

V programu MATLAB můžeme využít několik specifických funkcí:

- *fmincon* – minimalizace nelineární funkce s nelineárními nebo lineárními omezení
- *fminunc* – minimalizace nelineární funkce bez omezení
- *fminbnd* – hledání minimum funkce o jedné proměnné na zadaném intervalu
- *fminsearch* – minimalizace funkce pomocí algoritmu Nelder-Mead, který nevyžaduje derivace

- *Optimalization Toolbox* – nástroj který obsahuje všechny výše zmíněné funkce a další, viz. Předchozí kapitola

3.3. Dynamické programování

Dynamické programování představuje přístup v oblasti optimalizace, založený na principu dekompozice složitých problémů na menší podproblémy. Tato metoda spočívá v rozdělení

hlavního problému na podproblémy, kterou jsou řešeny samostatně a následně je z nich složeno výsledné řešení. Základem dynamického programování je princip optimality Bellmanovy rovnice, který stanovuje, že optimální řešení celkového problému lze sestavit z optimálních řešení jeho dílčích podproblémů.

V praxi dynamické programování umožňuje efektivněji řešit komplexní úlohy tím, že se vyhneme opakovanému řešení stejných podproblémů, což je zásadní pro urychlení celkového procesu řešení. Každý podproblém je řešen pouze jednou, a jeho řešení je uloženo pro budoucí použití, což eliminuje zbytečné opakování výpočtů. Avšak, nalezení efektivní strategie dekompozice (rozložení problému na podproblémy) může být pro některé úlohy opravdu obtížně. Ne všechny problémy se dají efektivně rozložit a vyřešit pomocí dynamického programování, jelikož ne vždy je možné sestavit optimální řešení z dílčích řešení.

V prostředí MATLAB, nenajdeme přímou funkci pro aplikaci dynamického programování, ale můžeme zde díky jeho vysoké výpočetní kapacitě a snadné programovací prostředí, sestavit vlastní algoritmus pro řešení. K tomuto bychom mohli využít hlavně funkce pro cykly, podmínky, datové struktury, atd…

Další možností v MATLABu je využití Simulinku pro modelování a simulaci, kde lze komplexní systémy rozložit na menší interagující komponenty. [18]

3.4. Celo číselné programování

Jedná se o speciální typ, který je vhodný, pokud jsou proměnné omezeny na celočíselné hodnoty, což je vhodné hlavně pro problémy, kde jsou jasné rozhodnutí – ano/ne, zapnuto/vypnuto, když potřebujeme zjistit celočíselný výsledek kolik jednotek použít (dělení je nepřípustné). [22] [23]

V kontextu MVE a bateriového úložiště by tento typ, mohl být využit pro rozhodování kolik baterií s předem danou kapacitou zařadím. Jelikož nemůžu baterie rozpůlit, potřebuji získat celočíselný výsledek, a přesně k tomu je tato metoda vhodná.

Aplikace v MATLAB:

- Funkce *intlinprog,* umožňuje řešit problémy pomocí lineárního programování s celočíselným omezením

3.5. Stochastické metody

Zahrnují použití prvků náhodnosti a pravděpodobnostních přístupů k prozkoumávání prostoru řešení. Jsou vhodné pro úlohy, kde selhávají deterministické přístupy (LP, NP, dynamické programování, …)

Mezi stochastické metody patří:

- Monte Carlo metody: Využívají opakované náhodné vzorkování, pomocí kterých odhaduje vlastnosti daného systému.
- Stochastické neuronové sítě – Využití hlavně pro umělé inteligence pro rychlejší učení a adaptaci
- Stochastická optimalizace – zahrnuju různé algoritmy, které využívají pravděpodobnosti a náhodné procesy pro hledání optimálního řešení.
- Genetické algoritmy: K hledání řešení dochází k vyvíjení jedinců pomocí křížení, mutace a selekce, tak aby určitý jedinec našel optimum dané úlohy.
- Simulované žíhání: Napodobení procesu žíhání v metalurgii, řešení se postupně ochlazuje a hledá globální optimum dané úlohy.

  [26] [27] [28]

Aplikace v programu MATLAB:

Pro některé metody najdeme v programu MATLAB specifické funkce, případně můžeme vytvořit vlastní skript nebo algoritmus. Například pro genetické algoritmy obsahuje MATLAB

Pro metodu Monte Carlo můžeme využít MATLBA funkce *rand* a *randn* pro generování náhodných čísel.

Dále v MATLAB najdeme funkci *simulannealbnd* které je určena pro simulované žíhání a můžeme s ní pracovat i prostřednictvím grafického zobrazení Optimization toolbox.

MATLBA také obsahuje další funkce, které můžeme využít pro stochastickou optimalizaci, stochastické gradientní metody a další.

3.6. Heuristické metody

Tyto optimalizační metody jsou v hodné pro použití na praktická řešení, kde nevyžadujeme vysokou přesnost. Často se používají tam kde je vyhledávání optimálního řešení příliš časově náročné, nebo nemožné. Pro řešení problémů často využívají různé specifické zkratky nebo

naučená pravidla. Bohužel u těchto metod není záruka, že dojde k nalezení globálního optima také zde hrozí riziko, že uvíznou v lokálním optimu.

S jejich aplikací se můžeme setkat například při hledání optimální (nejkratší) cesty mezi jednotlivými body (městy), například pomocí metody Nearest Neighbour (nejbližší soused), opět zde platí, že nalezené řešení nemusí být stoprocentně optimální. Dále se s jejich aplikací můžeme setkat u antivirových a antispam programů, pro hledání potenciálních hrozeb a identifikaci nových virů a malware.

Příklady metod:

- Tabu Search: Využívá seznam tabu kroků, pro zabránění cyklení a prozkoumání nových oblastí.
- Lokální vyhledávání: Pomocí různých technik se postupně provádí malé změny a hledá se lepší řešení v okolí.
- Greedy algoritmy (žravé algoritmy): Při řešení dochází k postupnému volení lokálního optima s nadějí nalezení výsledku, který bude blízko globálnímu optimu.

V programu MATLAB můžeme s těmito metodami pracovat, pomocí vytvoření vlastních skriptů, nebo nástrojů z *Optimization Toolbox*. [29]

3.7. Metaheuristické metody

Metaheuristické metody dokáží řešit různorodé problémy a používají z pravidla pro velmi složité úlohy, kde deterministické metody selhávají. Na rozdíl od heuristických metod jsou více obecné a pro to je jejich aplikace možná na širší spektrum problémů. Většina Metaheuristických metod využívá pravděpodobnostní prvky, aby nedocházelo k uvíznutí v lokálním optimu, a navíc tyto metody pracují iterativně, tak že dochází k postupné úpravě řešení, až je nalezeno vyhovující řešení. Stejně jako u heuristických metod, zde není pokaždé garantováno nalezení absolutně nejlepšího řešení. [30] [31] [32]

Můžeme sem zařadit například tyto metody:

- Genetické algoritmy
- Algoritmy roje částic
- Mravenčí kolonie
- Diferenciální evoluce
- Cuckoo Search a Firefly algoritmus

- Variable neighborhood search
- Greedy Randomized Adaptive Search Procedures (GRASP)

V programu MATLAB je můžeme využít pomocí Optimalization toolbox, který poskytuje určité funkce, které se dají zařadit do Metaheuristických metod např. genetické algoritmy (funkce *ga*). Dále můžeme vytvářet vlastní skripty pomocí dalších vestavěných funkcí.

funkce *ga*

[x,fval] = ga(objFcn, nvars);

Kde:

objFcn – objektivní funkce na které hledáme minimum

nvars – počet proměných

3.8. Simulační optimalizace

Tato metoda integruje optimalizační techniky do modelování a následné analýzy simulace. Pro optimalizaci systému, se využije jeho přímá simulace. Simulace zde reprezentuje komplexní systém, který by byl příliš složitý pro analytické řešení. Obvykle je základní simulační model stochastický, na základě jeho výsledků je prováděna analýza výstupu a iterační metodou hledáno optimum pro rozhodující výstup. Pro parametrickou optimalizaci je vstup proměnných do modelu variabilní s konstantními ostatními parametry, optimalizací se snažíme najít hodnoty proměnných které odpovídají optimálnímu výsledku. Dynamická optimalizace se používá pro „řídící“ rozhodování, nebo pro analýzu časové řady s různými vstupními parametry. Hodnoty proměnných, parametrů i optimální výsledek jsou pro každý krok specifické.

Simulační optimalizace jsou v dnešní době využívány ve spoustě odvětví a pro optimalizaci všemožných systémů. Jak už bylo zmíněno, jejich hlavní výhodou je možnost optimalizovat rozsáhlý a komplexní systém s velkou řadou proměnných. [33] [34]

Pro simulaci v prostředí MATLAB můžeme využít jeho doplňkový modul Simulink, který je přímo určený k modelování a simulaci různých systémů.

3.8.1. Simulink a SimScape

Tento program je podprogramem neboli doplňkovým modulem programu MATLAB, ve kterém je integrovaný. Ke své funkci používá MATLAB jazyk, avšak na rozdíl od klasického MATLAB Skriptu, nabízí grafické interaktivní prostředí pro modelování a simulaci dynamických systémů. Tento modul je pro simulování velmi oblíbený, především v inženýrských a vědeckých oblastech, a to jak pro modelování a simulaci tak i následnou analýzu a optimalizaci systémů.

Simulink nabízí širou knihovnu různých matematických a inženýrských bloků, ze kterých můžeme velmi jednoduše sestavit model. V knihovně můžeme najít také SimScape bloky, které lze při modelování využít.

SimScape prvky umožňují lépe modelovat jednotlivé fyzikální domény (mechanika, hydraulika, elektřina, termodynamika, …). Jsou zde vytvořeny speciální bloky pro každou z těchto domén, tak aby co nejvíce reprezentovali skutečnost, a zároveň uživateli zjednodušili a umožnili modelovat i velmi složité systémy. Většina těchto prvků, také umožňuje detailní zobrazení veškerých výsledků a chování proměnných při simulaci. SimScape modeluje fyzikální systém pomocí fyzikálních sítí na rozdíl od klasické Simulink knihovny, kde bloky pracují se signálovými sítěmi. Pomocí speciálních bloků můžeme také propojovat klasické Simulink bloky s bloky SimScape a díky tomu tvořit komplexní systémy.

Uživatel si může také vytvořit vlastní blok napsáním jeho kódu a z vlastních bloků si následně vytvořit svoji knihovnu.

Simulink nabízí také spoustu nástrojů pro vizualizaci, analýzu a interpretaci výsledků, které lze použít i pro většinu bloků ze SimScape knihovny.

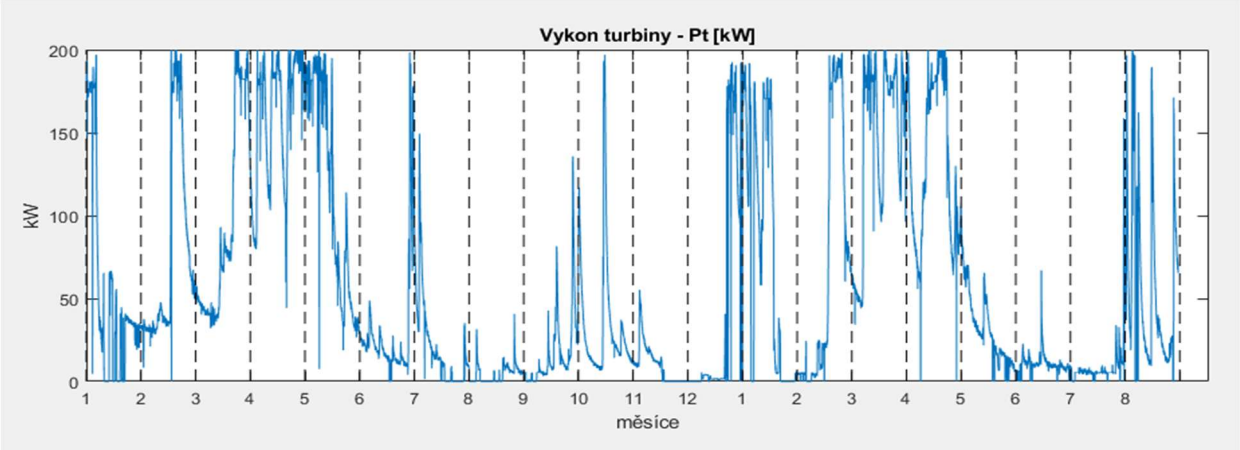
Díky všem těmto vlastnostem představuje Simulink velmi silný nástroj pro modelování, simulaci, analýzu a optimalizaci komplexních sytému, který je zároveň flexibilní, uživatelsky přívětivý, relativně jednoduchý.

4. Základní údaje MVE pro zpracování analýz

V mé práci byly optimalizace a analýzy zpracovávány pro data z malé vodní elektrárny Jánské Lázně II. Tato elektrárna se nachází na Černohorském potoku necelý kilometr pod malou vodní elektrárnou Jánské Lázně I. Elektrárna byla vystavěna v roce 2016, společně s přivaděčem, který začíná vtokovým objektem typu stupeň ve dně. Přivaděč má celkovou

délku 1714 m, a průměr 500 mm. Horní část přivaděče (980 m) je společná s MVE Jánské Lázně I. Přiváděná voda pohání dvě Peltonovy horizontální dvoutryskové turbíny, s asynchronními generátory 90kw. Elektrárna disponuje hrubým spádem 110 m, a celkovým instalovaným výkonem 180 kW. Proudové jištění na hodnotě 315 A, umožňuje maximální výkon lehce přes 200 kW.

Tím že se jedná o menší potok a povodí, odezva na počasí je zde poměrně rychlá. Na ročním grafu výkonů turbíny vidíme, že výkon opravdu velmi rychle kolísá v závislosti na aktuálním počasí.



Obrázek 4.1 - Výkon turbíny 2022 a 2023

5. Optimalizační program maximalizace zisku při použití bateriového úložiště

MVE a bateriové úložiště, si lze představit jako jednoduchý systém, kde máme proměnné, které nedokážeme ovlivnit: řeku (přírodu, počasí) a jednotkové ceny elektřiny (OTE). Dále hardwarové komponenty ve strojovně (ve schéma černou barvou): turbína, střídač a nabíječka, baterie, síť. Veškeré tyto části mají svoje limity a omezení. Práci jednotlivých části ovládá řídící jednotka „Energy balace controller“ (EBC), pro kterou jsem v této práci vytvářel optimalizační algoritmy.

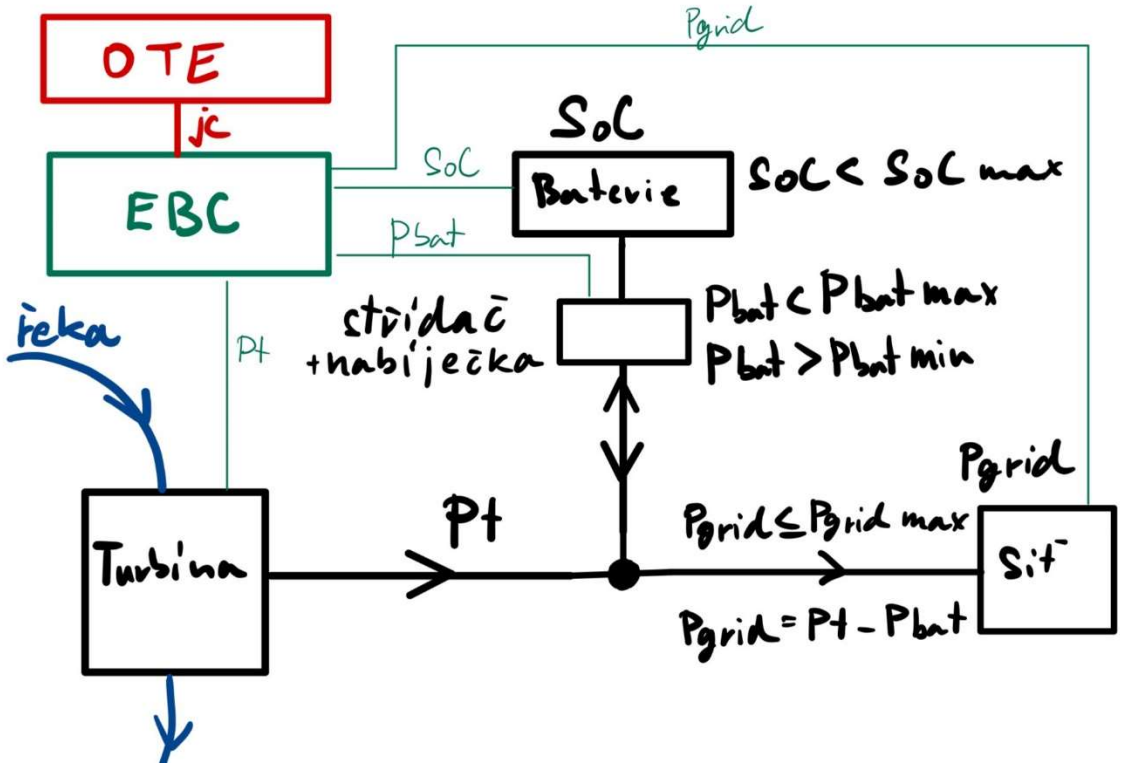


Schéma 1 - MVE a bateriové úložiště

Níže popisované optimalizační programy jsem vytvořil v programu MATLAB, pomocí nástroje Live Script, který umožňuje vytvářet dokumenty, které kombinují MATLAB kód, výstupy, formátovaný text, obrázky a rovnice. Uživatel může psát programový kód v jazyku MATLAB přímo v dokumentu a průběžně aktualizovat výsledky, které mohou být viditelné přímo v kódu. Jednotlivé zavedené proměnné jsou v průběhu psaní kódu aktualizované, což

napomáhá při jeho vytváření. Kód lze také rozdělit do několika sekcí, které uživatel může spouštět samostatně, to opět napomáhá při vytváření a ladění daného programu.

Pro zjednodušenou optimalizaci, řízení nabíjení a vybíjení baterie, jsem zvolil optimalizační metodu lineární programování, která je relativně jednoduchá, ale dokáže najít globální optimum (pokud existuje). Omezení vycházející z volby této metody:

- Objektivní funkce musí být lineární
- Rovnicová i nerovnicová omezení musí být lineární
- Veškeré omezení musí být také lineární

Tato omezení neumožňují zavedení například nelineární změny účinnosti, atd…

V mém skriptu je pro tuto metodu využita funkce *linprog*, kterou MATLAB obsahuje.

5.1. Ideální optimalizace při znalosti dat na celé období

Jako první jsem v programu MATLAB vytvořil skript, „opt_zaklad_kraus_final.mlx“, který provádí optimalizaci s tím že veškerá data o výrobě a ceně zná dopředu, na celé období a neuvažuje žádné ztráty. Toto je naprosto optimální a nereálný stav, v praxi je výkupní cena elektřiny známá na 24 h dopředu a výrobu známe pouze aktuální. Díky znalosti veškerých dat, tento program generuje stoprocentně optimální výsledek, který bude představovat maximum, kterého můžeme s danými parametry dosáhnout.

Tento skript by mohl být využitý pro analýzu na základě historických dat, dimenzování baterie, střídače a určení návratnosti investice na pořízení bateriového systému. Pokud takto optimální simulace ukáže, že se investice nevyplatí, víme že při více reálném řízení by to bylo ještě horší, a tudíž je potřeba hledat jiné řešení nebo lokalitu. Tento program by také mohl sloužit pro srovnání účinnosti programu, který bude prodej do sítě řídit tak jak by tomu bylo ve skutečnosti (data o cenách by znal pouze na 24 h dopředu, a výrobu pouze aktuální). Při srovnání bychom viděli o kolik je „skutečný“ řídící algoritmus horší, než tato optimální simulace a mohli bychom vyhodnotit, jestli je potřeba „skutečný“ řídící algoritmus ještě vylepšovat, jaká je jeho účinnost, případně jaký je tu prostor pro zlepšení.

Tento skript má několik variant, které se liší pouze délkou simulovaného období (vstupními daty). Pro lepší přehlednost výsledků a grafů si tento skript představíme ve variantě simulace jednoho týdne v únoru 2022.

Vstupní data pro tento skript jsou:

- Pt – výkon elektrárny v čase t [kW]
- jc – jednotká cena (unit price) [Kč/kWh]
- SoC_max – maximální kapacita baterie [kWh]
- Pgrid_max – maximální prodej do sítě – maximální výkon jističe [kW]
- Pbat_max – maximální výkon baterie (vybíjení baterie) [kW]
- Pbat_min – maximální příkon do baterie (nabíjení baterie) [kW]
- SoC_start – počáteční nabití baterie [kWh]

Proměnné:

- Pgrid – výkon do sítě v čase t [kW]
- Pbat – výkon baterie v čase t [kW]
- SoC – kapacita baterie v čase t (state of charge) [kWh]

Objektivní funkce:

Cílem skriptu je maximalizovat zisk z prodeje energie s ohledem na omezení bateriového úložiště a cen energie. Zisk je definován jako suma prodané energie a její ceny v každém časovém kroku.

Objektivní funkce je tedy:

Maximalizace zisku=$\sum(Pgrid(t) \cdot jc(t))$

Kde:

Pgrid(t) je množství energie prodané do sítě v čase t

jc(t) je cena energie v čase t.

Stav baterie:

Stav nabití baterie (*SoC*) se v každém kroku aktualizuje na základě příkonu do baterie *Pbat(t)* a odebíraného výkonu z baterie, což je množství energie prodané do sítě *Pgrid(t).*

Aktualizace SoC je dána vztahem:

$SoC(t)=SoC(t-1)+Pbat(t)-Pgrid(t)$

Kde:

SoC(t−1) je stav nabití baterie v předchozím časovém kroku

Pbat(t) je příkon do baterie (výkon elektrárny mínus prodaná energie)

Pgrid(t) je energie prodaná do sítě.

Tato rovnice předpokládá, že baterie může být nabíjena pouze energií vyrobenou turbínou, ne z externího zdroje (ze sítě).

Omezení pro optimalizaci:

1. **Omezení prodeje energie:**

   Prodej energie v čase t nemůže přesáhnout maximální výkon do sítě *Pgrid_max*, což je reprezentováno jako *0≤Pgrid(t)≤Pgrid_max.*

2. **Omezení záporné ceny:**

   Pokud je cena *jc(t)* záporná, prodej *Pgrid(t)* je nastaven na nulu. Nesmí dojít k prodeji za záporné ceny!

   Aby bylo možné tuto úlohu řešit, dojde také k vypnutí turbíny a prodej *Pgrid(t)* je nula.

3. **Omezení kapacity baterie:**

   Energie v baterii v čase t musí být v rozmezí kapacity baterie: *0≤SoC(t)≤SoC_max.*

4. **Omezení prodeje a výroby:**

   Energie prodaná do sítě v čase t nesmí překročit součet energie v baterii v předchozím časovém kroku a výrobu turbíny: *Pgrid(t)≤SoC(t−1)+Pt(t).*

   Prodej do sítě musí být nižší než maximální výkon do sítě: *Pgrid(t)≤Pgrid_max*, kde *Pgrid_max* je omezení maximálního výkonu do sítě.

5. **Omezení baterie:**

   Nabíjení baterie musí být pozitivní a nemůže přesáhnout výrobu elektrárny: *0≤Pbat(t)≤Pt(t*). Baterie se nemůže nabíjet z energie ze sítě.

   Nabíjení baterie je omezeno maximální rychlostí nabíjení:

   Pbat_max: *Pbat(t)≤Pbat_max*

   Vybíjení baterie je omezeno maximální rychlostí vybíjení:

   Pbat_min: −Pbat(t)≤Pbat_min

   Celkový prodej do sítě nemůže překročit *Pgrid_max*, který reprezentuje maximální možný výkon prodaný do sítě.

5.1.1. Popis skriptu „ideální“ optimalizace

V první části skriptu, dochází k načtení, zadání a případné úpravě dat, tak aby byl program schopný řešit svoji úlohu. Proto je potřeba definovat veškeré parametry, které zde jsou uvedeny.

```
clear all;

% načtení vstupních dat
load("unor_cena_kcmwh.mat");    % vstupní hodnoty v [Kc/MWh] !!!
load("unor_vyroba_mwh.mat");    % vstupní hodnoty v [MWh] !!!

% Vstupní parametry
Pt = unor_vyroba_mwh(401:568) * 1000; % výkon elektrárny 1. týden [kWh]
jc = unor_cena_kcMwh(401:568) / 1000; % cena 1. týden [kc/kWh]
SoC_max = 200;      % maximální kapacita baterie [kWh]
Pgrid_max = 200;    % maximální výkon do sítě [kW]
SoC_start = SoC_max; % Start kapacita [kWh]
Pbat_min = 100;    % Maximální rychlost nabíjení baterie [kW]
Pbat_max = 100; % Maximální rychlost vybíjení baterie [kW]

n = length(Pt);
```

Obrázek 5.1 - Ideální optimalizace – vstupní hodnoty

V další části dochází k vytvoření matic, které omezují funkci *linprog* při hledání optimálního řešení. Matice *lb* reprezentuje dolní hranici energie, kterou můžeme prodat do sítě. Tato hodnota je nejprve definována jako 0 pro celou řadu, ale v následujícím kroku je tato matice ještě aktualizována, tak aby se v ní promítnulo omezení maximální rychlosti nabíjení baterie. Matice *ub* představuje horní hranici omezení prodeje do sítě, $Pgrid(t) \leq Pgrid\_max(t)$. V dalším kroku je tato matice opět aktualizována, tak aby respektovala omezení maximální rychlosti vybíjení baterie a výkonu elektrárny (celkovou dostupnou energie na prodej).

```
% Horní a dolní hranice pro Pgrid - omezení se kterými pracuje linprog
lb = zeros(n,1);    % dolní hranice (minimum energie prodané do sítě) == 0
ub = Pgrid_max * ones(n,1); % horní hranice (omezení Pgrid_max) == Pgrid(t)<=Pgrid_max

% Nevybíjíme z baterie, pokud je cena záporná
ub(jc < 0) = 0;

% Aktualizace lb a ub pro Pgrid s ohledem na max rychlost nabíjení/vybíjení
for i = 1:n
    % Nabíjecí omezení, pokud je výroba větší než nabíjecí limit
    if Pt(i) > Pbat_min
        lb(i) = max(lb(i), Pt(i) - Pbat_min);
    end
    % Vybíjecí omezení, pokud je výroba menší než vybíjecí limit a stále respektujeme Pgrid_max
    ub(i) = min(ub(i), max(0, Pt(i) + Pbat_max));

    % Respektování stávajícího omezení Pgrid_max a záporné ceny energie
    ub(i) = min(ub(i), Pgrid_max);
    if jc(i) < 0
        ub(i) = 0;
    end
end
```

Obrázek 5.2 - Ideální optimalizace – omezující podmínky

V následující části skriptu probíhá vytvoření matice *A* a vektoru *b.* Tyto parametry představují nerovnicové omezení pro funkci *linprog,* na základě, kterých tato funkce optimalizuje objektivní funkci a hledá řešení, tak aby respektovalo fyzikální a technické omezení systému.

A , b = nerovnicová omezení

$A*x < b$

Pro tento program jsou tyto omezení odvozeny od kapacity baterie *SoC_max* a dostupné energie.

Matice *A* je dolní trojúhelníková matice, která je vytvořená pomocí funkce *trill(ones(n,n)),* tím dojde k vytvoření matice, u které jsou všechny prvky nad hlavní diagonálou nulové a naopak hlavní diagonála a všechny prvky pod ní jsou nenulové (jsou rovny 1). V tomto programu matice *A* umožňuje díky její struktuře postupnou akumulaci hodnot v čase. Například když tuto matici vynásobíme vektorem *Pt*, dostaneme vektor, který bude mít na každém řádku součet prvků vektoru *Pt* po daný čas, například pro 5. prvek nově vzniklého vektoru, bude hodnota rovna součtu prvních pěti prvků vektoru *Pt.*

Vektor *b* obsahuje v každém kroku maximální možnou hodnotu nabití baterie *SoC*, což je nejprve definováno jako *SoC_max*, následně je vektor upraven, aby zohledňoval aktuální dostupnou energii.

Matice *A_min* a vektor *b_min,* představují dolní omezení pro stav baterie *SoC. A_min* je záporná dolní trojúhelníková matice, díky které je vytvořeno omezení, pro kumulativní výdej elektřiny, tak aby neklesl pod 0. Vektor *b_min*, je vektor samých nul, vytvořený pomocí funkce *zeros* a zajišťuje, že kumulativní množství energie bude větší než 0 (jedná se o pravou stranu nerovnicového omezení).

V dalším kroku jsou matice *A* a *A_min* sloučeny do kombinované matice *A_combined,* se kterou následně pracuje funkce *linprog*, stejnou analogií je vytvořen vektor *b_combined.*

```
% Vytvoření matice A a b pro nerovnostní omezení
A = tril(ones(n,n));
b = SoC_max * ones(n,1);

A_min = -tril(ones(n,n));
b_min = zeros(n,1);

% vytvoření matic b, které omezují SoC, a vybíjení.
for i = 1:n
    b(i) = b(i) + sum(Pt(1:i)) - (SoC_max - SoC_start);
    b_min(i) = -sum(Pt(1:i)) + (SoC_max - SoC_start);
end

% Sjednocení omezení do jednoho nerovnostního omezení
A_combined = [A; A_min];
b_combined = [b; b_min];
```

Obrázek 5.3 - Ideální optimalizace – Vytvoření matic nerovnicových omezení

Dále je v programu volána funkce *linprog,* která provede optimalizaci *Pgrid* (energie prodané do sítě [kW]). Funkcí pro optimalizaci je *jc* (jednotková cena [Kč/kWh]), a protože funkce *linprog* hledá minimum, ale my požadujeme maximalizaci zisku, musí být funkce pro optimalizaci záporná. Jako další vstupy pro *linprog* slouží již předem připravené nerovnicové omezení, horní a dolní okrajové podmínky.

```
% volání linprog
f = -jc; % mínus protože chceme maximum a linprog minimalizuje
options = optimoptions('linprog','Display','none');
Pgrid = linprog(f, A_combined, b_combined, [], [], lb, ub, options);
```

Obrázek 5.4 - Ideální optimalizace – volání funkce linprog pro optimalizaci

Ihned potom je vypočítávám zisk provozu malé vodní elektrárny s bateriovým úložištěm, který je definován jako suma elementárně vynásobených vektorů *Pgrid [kW]* (prodaná elektřina do sítě) a *jc [Kc/kWh]* (jednotková cena).

```
% Výpočet celkového zisku
zisksBat = sum(Pgrid .* jc);
```

Obrázek 5.5 - Ideální optimalizace – výpočet ziskovosti s bateriovým úložištěm

Je zde také vypočítáno chování baterie ve výpočetním období. Nejprve je určen příkon do baterie *Pbat = Pt - Pgrid* (rozdíl výkonu turbíny a výkonu do sítě, neboli kolik energie se vyrobilo a kolik prodalo). A poté je vypočítán stav baterie *SoC* v jednotlivých časových krocích.

SoC(i) = SoC_start(i-1)+ Pbat(i)

Kde:

SoC – stav nabití baterie v daném časovém kroku

SoC_start – stav nabití baterie v předchozím kroku

Pbat – příkon baterie v současném časovém kroku¨

Pro tento výpočet je využit cyklus *for*, který provede vypočtení stavu baterie pro veškeré časové kroky výpočtu. Následně je v dalším cyklu, pomocí této funkce, vypočítán teoretický zisk provozu MVE, bez akumulace energie bateriemi, a to jako maximum možné prodané energie v každém časovém kroku. To je rovno menší z hodnot *Pt* a *Pgrid_max* (vyrobená energie turbínou a maximální množství energie, kterou můžeme prodat do sítě) vynásobené jednotkovou cenou *jc.*

```
% Výpočet celkového zisku
zisksBat = sum(Pgrid .* jc);

% Prikon do baterie
Pbat = Pt - Pgrid; % pokud je přikon + == nabíjení; - == vybíjení

% Stav baterie
SoC = zeros(n,1);
SoC(1) = SoC_start + Pbat(1);
for i = 2:n
    SoC(i) = SoC(i-1) + Pbat(i);
end

% Výnos bez baterie s omezením vypnutí turbíny při záporné ceně
vynosBezBaterie = zeros(1,n);
for t = 1:n
    if jc(t) > 0
        prodej = min(Pt(t), Pgrid_max); % prodáme co nejvíce, ale ne více než Pgrid_max
        vynosBezBaterie(t) = prodej * jc(t);
    end
end

celkovyVynosBezBaterie = sum(vynosBezBaterie);
```

Obrázek 5.6 - Ideální optimalizace – výpočet Pbat, SoC a zisku bez baterie

V předposlední části skriptu jsou definovány podmínky, které ověřují správnost výpočtu:

1. Podmínka dodržení rozsahu baterie $0 \leq SoC \leq SoC\_max$
2. Energie nesmí být prodávána, za záporné ceny. $Pgrid(jc<0) = 0$
3. Rozsah prodeje do sítě: Prodej do sítě nesmí překročit maximální možný prodej daný výkonem jističe *(Pgrid_max),* a nesmí být záporný (to by znamenalo, že došlo k nabíjení baterie ze sítě. $0 \leq Pgrid \leq Pgrid\_max$
4. Kontrola energetické bilance: součet vyrobené elektřiny se musí rovnat součtu prodané energie a rozdílu nabití baterie na začátku a na konci simulace
   $sum(Pt) = sum(Pgrid) + (SoC(end) – SoC(start))$

```
% Kontroly správného fungování modelu - aktualizace s novými proměnnými
% Pokud nějaká veličina nesplní zadané podmínky, program se ukončí s eror hláškou
assert(all(SoC >= 0) & all(SoC <= SoC_max), 'Chyba: Kapacita baterie je mimo očekávaný rozsah.');
assert(all(Pgrid(jc<0) == 0), 'Chyba: Prodej energie za záporné ceny.');
assert(all(Pgrid >= 0) & all(Pgrid <= Pgrid_max), 'Chyba: Prodej mimo očekávaný rozsah.');
assert(abs(sum(Pt) - sum(Pgrid) - (SoC(end) - SoC_start)) < 1e-6, 'Chyba: Energetická bilance neodpovídá.');
```

Obrázek 5.7 - Ideální optimalizace – kontroly výsledků

Poslední část skriptu, slouží pro zobrazení výsledků a vykreslení grafů. Nejprve zde vidíme porovnání zisku z prodeje elektřiny, při provozu bez baterie a s baterií.

```
% Porovnání výnosů s a bez baterie
fprintf('Celkový zisk bez baterie: %f Kč\n', celkovyVynosBezBaterie);
```

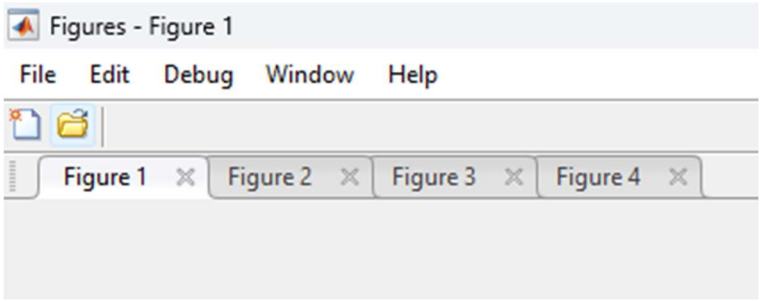
```
Celkový zisk bez baterie: 81202.022510 Kč
```

```
fprintf('Celkový zisk s baterií: %f Kč\n', zisksBat);
```

```
Celkový zisk s baterií: 84514.443270 Kč
```

Obrázek 5.8 - Ideální optimalizace – výsledky 1 týden provozu

Dalšími výstupy tohoto programu jsou grafy, zobrazující různé proměnné po dobu simulace. Grafy jsou vykresleny pomocí funkce *figure* a *plot*. Po úspěšném výpočtu, program automaticky provede vykreslení grafů do dialogového okna, které se automaticky otevře. V horní části můžeme následně přepínat mezi jednotlivými kartami (grafy).



Obrázek 5.9 - Ideální optimalizace – přepínání mezi jednotlivými kartami

V prvním okně jsou vykresleny grafy základních parametrů a proměnných, jako je:

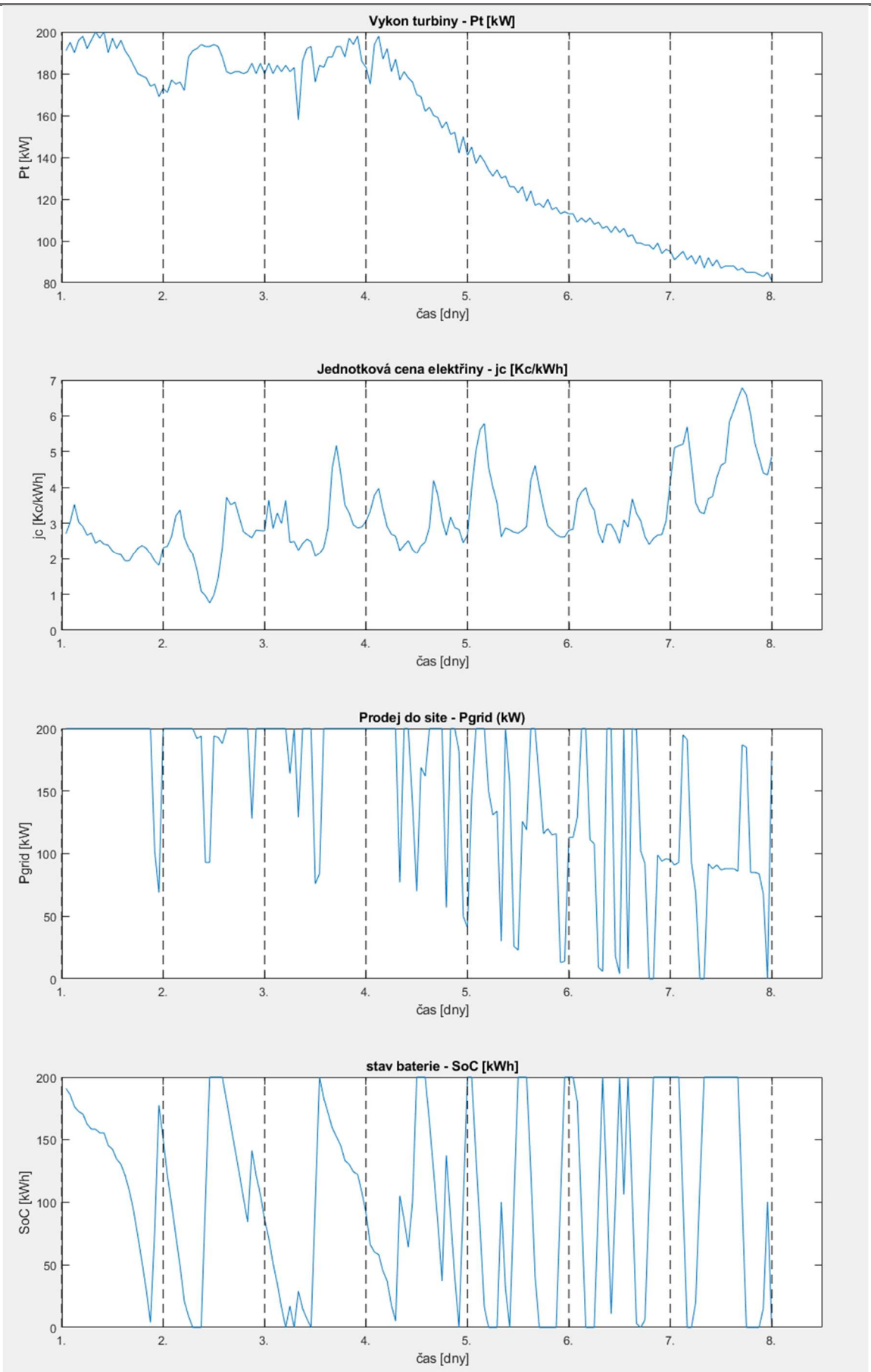
Výkon turbíny – *Pt* [kW]

Jednotková cena elektřiny – *jc* [Kc/kWh]

Prodej elektřiny do sítě (výkon do sítě) – *Pgrid* [kW]

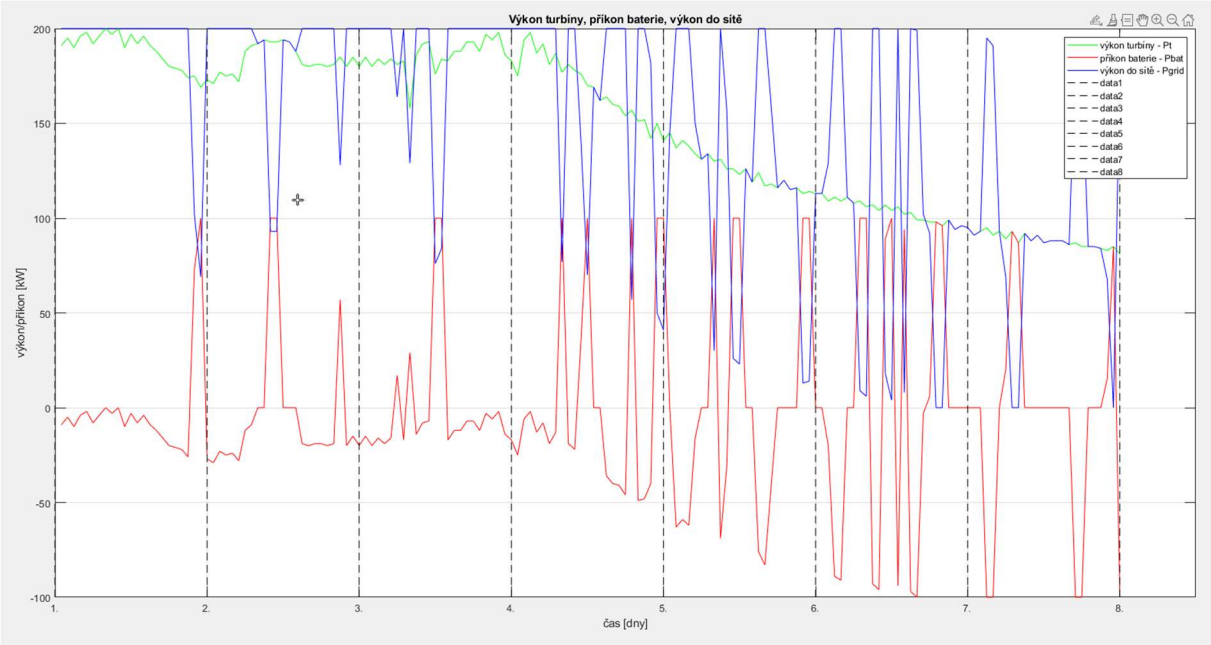
Stav nabití baterie – *SoC* [kWh]

Na tomto základním grafu lze také ověřit správnost chování programu. V hodinách s nižší cenou elektřiny dochází k snížení prodeje do sítě, případně až na úplné zastavení prodeje v úsecích s nízkou cenou a nízkým výkonem turbíny. Naopak v hodinách s vysokou cenou, lze vidět zvýšení prodeje na maximum a vybíjení baterie (baterie je vybíjena tak, aby doplnila výrobu turbíny pro maximální možný prodej).



Graf 5.1 - Ideální optimalizace – Graf se základními proměnnými

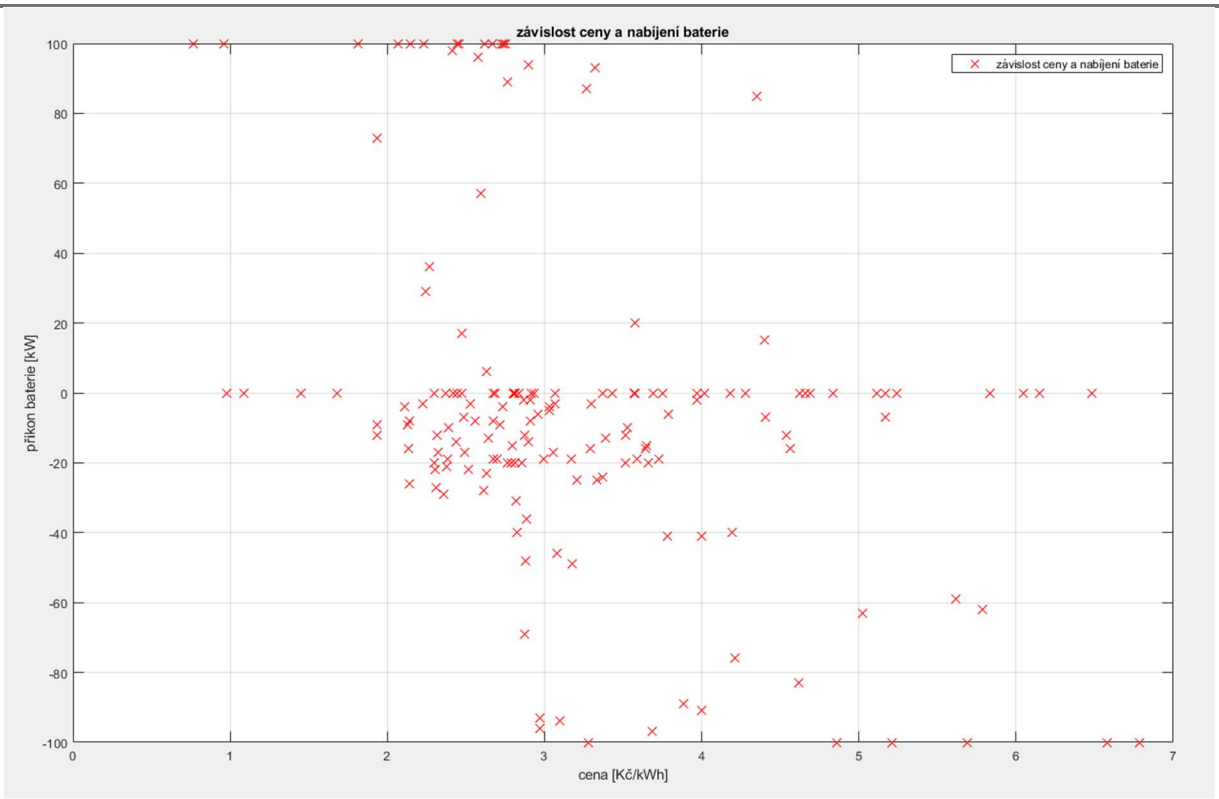
V druhém okně, je vykreslen složený graf, který se skládá z průběhu výkonu turbíny *Pt* (zeleně), příkonu baterie *Pbat* (červeně)a výkonu do sítě (prodeje) *Pgrid* (modře). Na tomto grafu lze pozorovat závislost prodeje elektřiny a příkonu do baterie, jakmile se příkon z baterie zvýší (dochází k nabíjení baterie), výkon do sítě (prodej) se s níží, protože turbínou vyrobenou elektřinu používáme k nabíjení baterie.



Graf 5.2 - Ideální optimalizace – Složený graf Pt, Pbat, Pgrid

V dalším okně, najdeme kontrolní graf, na kterém je vykreslena závislost příkonu baterie a ceny za kterou je elektřina v daném okamžiku prodávána.

Je zde vidět že dochází k nabíjení baterie (kladné hodnoty příkonu), když je cena relativně nízká a vybíjení baterie (prodej do sítě), když je cena vyšší. Tato závislost je lépe vidět na grafu 5.9 v následující kapitole, kde jsou tato data vykreslena pro simulaci období 20 měsíců.

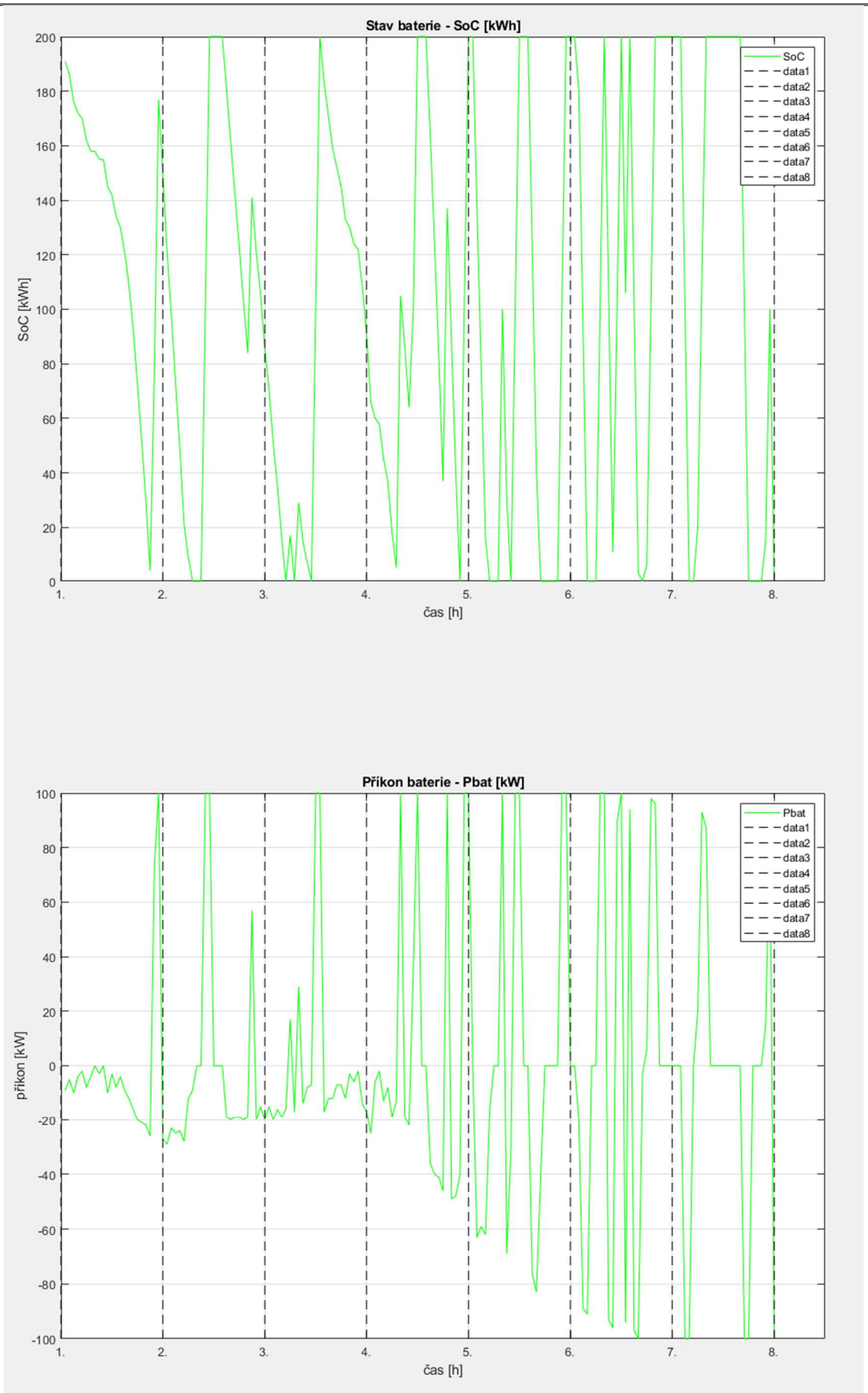


Graf 5.3 - Ideální optimalizace – Závislost jc a Pbat

Poslední okno zobrazuje dva grafy, na horním je vykreslen průběh stavu baterie *SoC* a na dolním průběh příkonu baterie *Pbat*. Na tomto grafu můžeme vidět vzájemnou souvislost těchto dvou proměnných. Jakmile dojde ke kladnému příkonu do baterie (dochází k nabíjení), *SoC* roste a naopak. Tento graf může sloužit jako další kontrola správného fungování algoritmu. Dále zde můžeme kontrolovat, že *SoC* ani *Pbat* nepřesáhly meze, ve kterých by se měli pohybovat. Pro aktuální nastavení:

SoC = 0 – 200 kWh

Pbat = -100 - +100 kW



Graf 5.4 - Ideální optimalizace, grafy SoC a Pbat

Stejně jako ostatní mnou vytvořené programy, které budu prezentovat v této práci, je tento skript plně univerzální a přizpůsobitelný na jakoukoliv jinou elektrárnu, parametry a lokalitu. Stačí pouze upravit parametry simulace a vstupní řady jednotkových cen a výkonů soustrojí. Více k ovládání a přizpůsobení skriptu pro jinou elektrárnu v kapitole 5.1.3.

5.1.2. Výsledky simulace období 1.1.2022 – 1.9.2023

Tyto výsledky byly vypočítány, stejným programem, který jsem představil v předchozí kapitole, pouze vstupem byla 20měsíční data a mírně upravená část, která vytváří grafy, pro jejich lepší přehlednost.

Vstupní parametry pro vypočtené výsledky:

```
% načtení vstupních dat
load('cena_202201_202309.mat');    % vstupní hodnoty v [Kc/MWh] !!!
load('vyroba_202201_202309.mat');    % vstupní hodnoty v [MWh] !!!

% Vstupní parametry
Pt = vyroba_202201_202309 ; % výkon elektrárny 1. týden [kWh]
jc = cena_202201_202309 / 1000; % cena 1. týden [kc/kWh]
SoC_max = 200;      % maximální kapacita baterie [kWh]
Pgrid_max = 200;    % maximální výkon do sítě [kW]
SoC_start = SoC_max; % Start kapacita [kWh]
Pbat_min = 100;    % Maximální rychlost nabíjení baterie [kW]
Pbat_max = 100; % Maximální rychlost vybíjení baterie [kW]
```

Obrázek 5.10 - Ideální optimalizace 1 rok – vstupní parametry

Protože jsem pracoval s hodinovým krokem, program musel, při takto dlouhém období, pracovat s poměrně rozsáhlými daty. Kvůli tomu došlo ke značnému zpomalení řešení *linprog*, program pracuje 5–60 minut (v závislosti na rychlosti zařízení), než se dobere k požadovanému výsledku. Tento čas je také závislý na nastavených parametrech a jak moc „složité“ je pro *linprog* najít globální optimum. Například při nastavení extrémně malé kapacity baterie (SoC_max) a velmi nízkého maximálního výkonu střídače/nabíječky (Pbat_max a Pbat_min), došlo k výraznému zpomalení celého řešení (pro funkci *linprog* bylo mnohem obtížnější najít globální optimum při takovýchto parametrech)

Toto zpomalení je způsobeno tím, jak linprog pracuje a tím, jak funguje simplexová metoda v lineárním programování. Kde je při hledání optima postupně zaplňována (na začátku řídká) matice, a proto čím je řešení složitější, tím program počítá pomaleji.

Při snaze simulovat ještě větší počet dat (celé dva roky) a nepříznivé vstupní hodnoty (Pbat_max, Pbat_min, SoC_max). Funkce *linprog* nebyla vůbec schopná řešení najít, protože překročila maximální paměť (velikost matice), který ji MATLAB dovoluje mít. Řešením by zde mohlo být rozdělení simulace na více částí a následné spojení dohromady. Ještě předtím by bylo vhodné na menším příkladu ověřit, jak velká chyba při takovémto řešení (spojováním) vzniká. Případně opustit program MATLAB a vytvořit si vlastní optimalizační program, který by svým způsobem řešení zvládal zpracovat větší počet dat.

```
Solver stopped prematurely.

Linprog stopped because it exceeded its allocated memory.
```

Obrázek 5.11 - Překročení paměti funkce linprog

Výsledky tohoto období ukázali, že díky bateriovému úložišti (viz. parametry výše) by mohl být profit zvýšený o 320tis Kč což představuje navýšení o 9,7 %. Jak už bylo zmíněno, jedná se o ideální simulace bez ztrát, tudíž je potřeba počítat s tím že v praxi by toto navýšení bylo nižší. Charakter lokality dané vodní elektrárny by také mohl ovlivnit tuto hodnotu, proto je potřeba posudek provést vždy pro konkrétní lokalitu na základě konkrétních dat.

```
% Porovnání výnosů s a bez baterie
fprintf('Celkový zisk bez baterie: %f Kč\n', celkovyVynosBezBaterie);
```

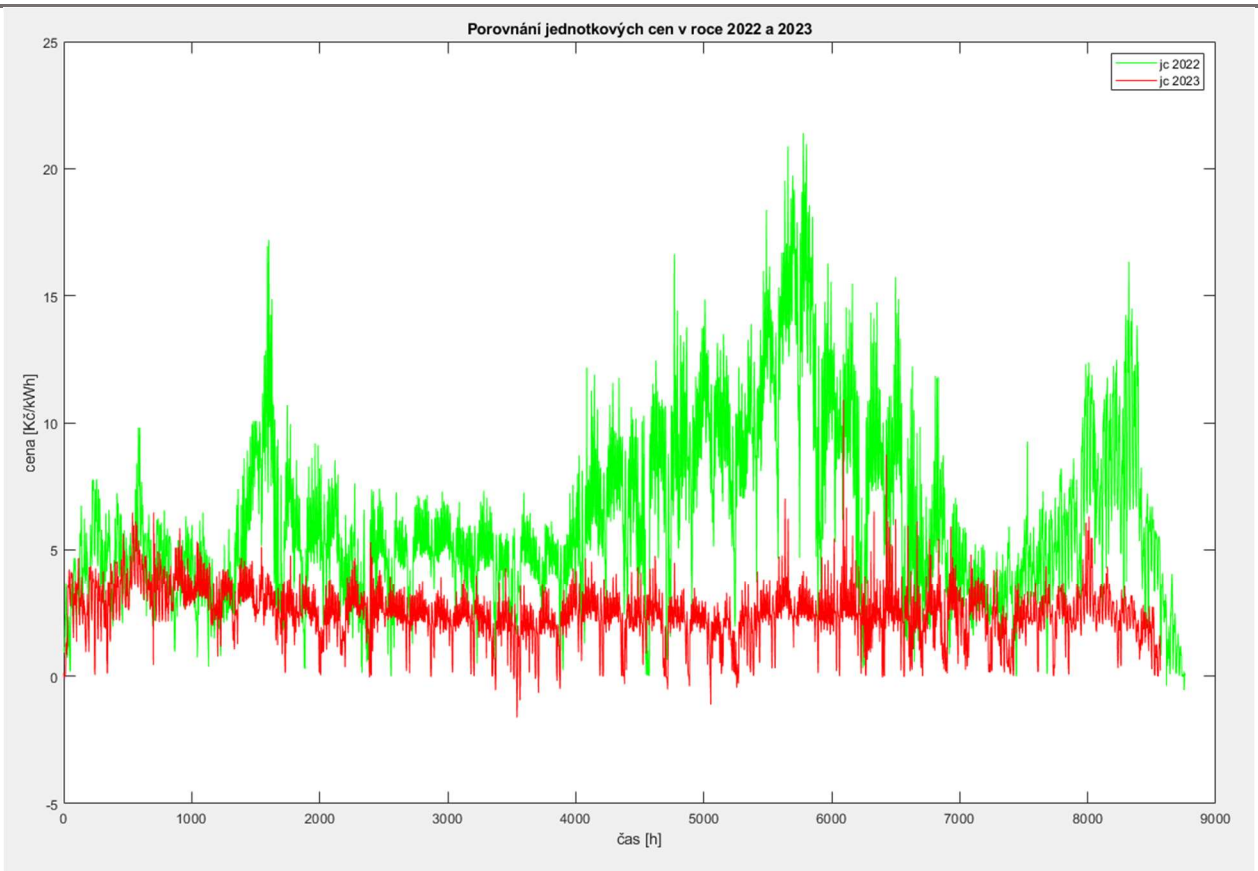
```
Celkový zisk bez baterie: 3189129.919363 Kč
```

```
fprintf('Celkový zisk s baterií: %f Kč\n', zisksBat);
```

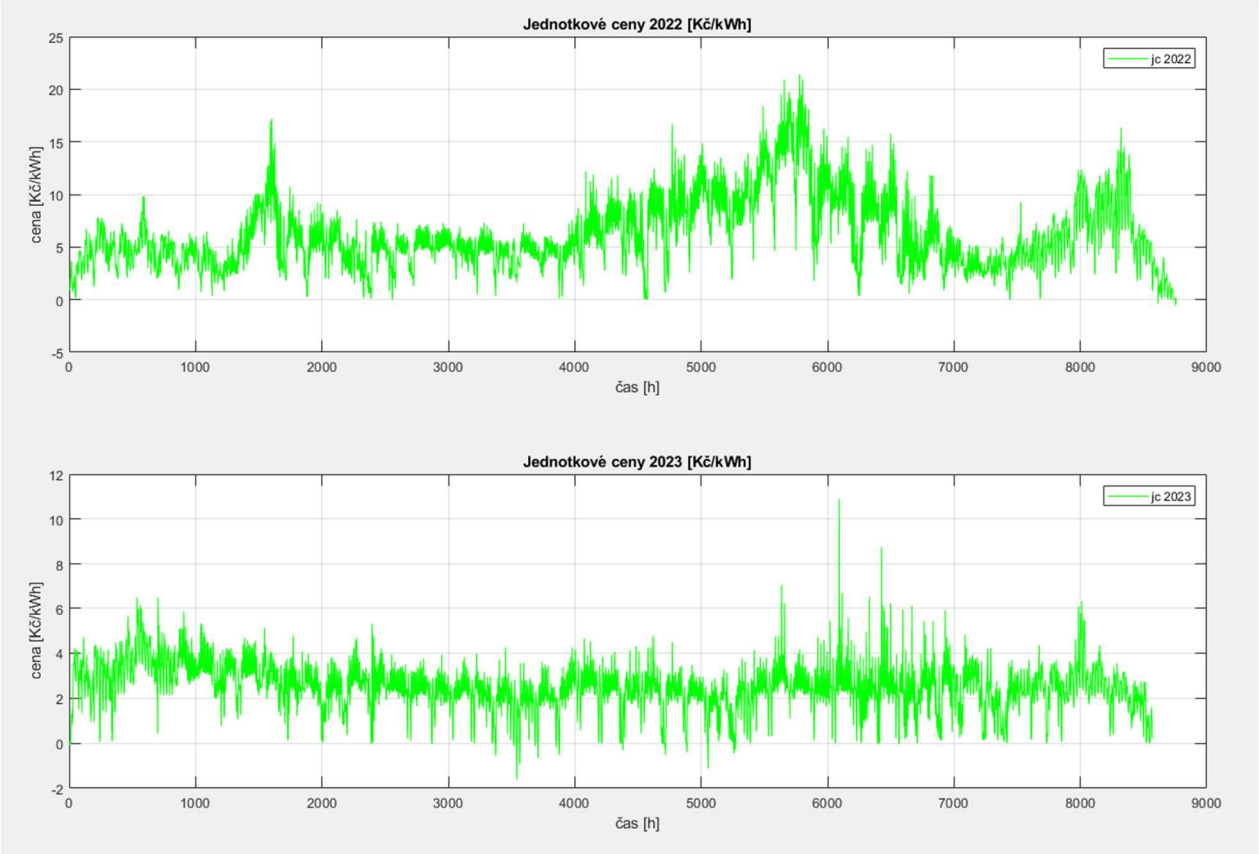
```
Celkový zisk s baterií: 3499797.443587 Kč
```

Obrázek 5.12 - Ideální optimalizace 20 měsíců – porovnání zisku

Navýšení zisku bude také velmi ovlivňovat cena elektřiny, která značně kolísá. Pro srovnání zde uvádím grafy jednotkových cen elektřiny za rok 2022 a 2023 kde je vidět značný rozdíl. I z toho důvodu jsem pro následující analýzu období rozdělil na rok 2022 a 2023, a návratnost investice jsem zjišťoval pro každé období zvlášť.



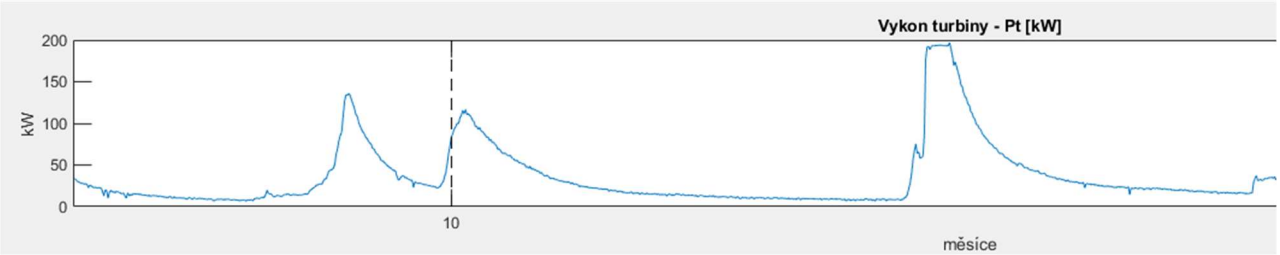
Graf 5.5 - Porovnání jednotkových cen elektřiny v roce 2022 a 2023



Graf 5.6 - Porovnání jednotkových cen elektřiny v roce 2022 a 2023

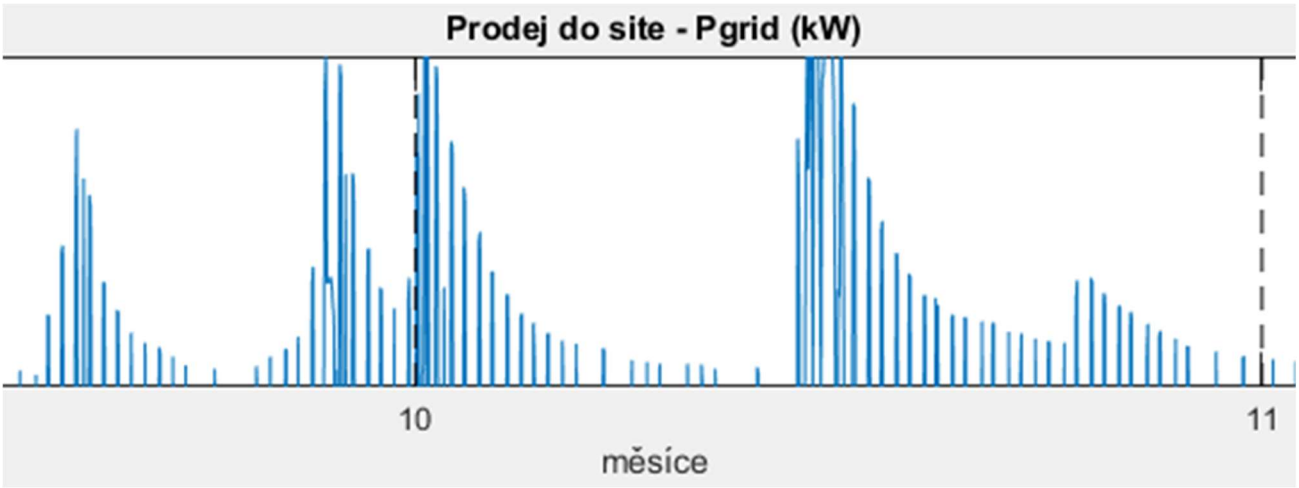
Níže jsou výsledky simulace období 1.1.2022 – 1.9.20203 v podobě grafů. Vzhledem k velkému množství dat, jsou některé grafy značně nepřehledné, pro lepší analýzu a zobrazení je vhodné procházet si je přímo v Matlabu po dokončení výpočtu.

Zajímavý je průběh výkonu turbíny a prodeje do sítě (*Pt* a *Pgrid*), kde je vidět že po rychlém nárůstu (pravděpodobně vlivem deště), klesání není lineární, ale má parabolický průběh.



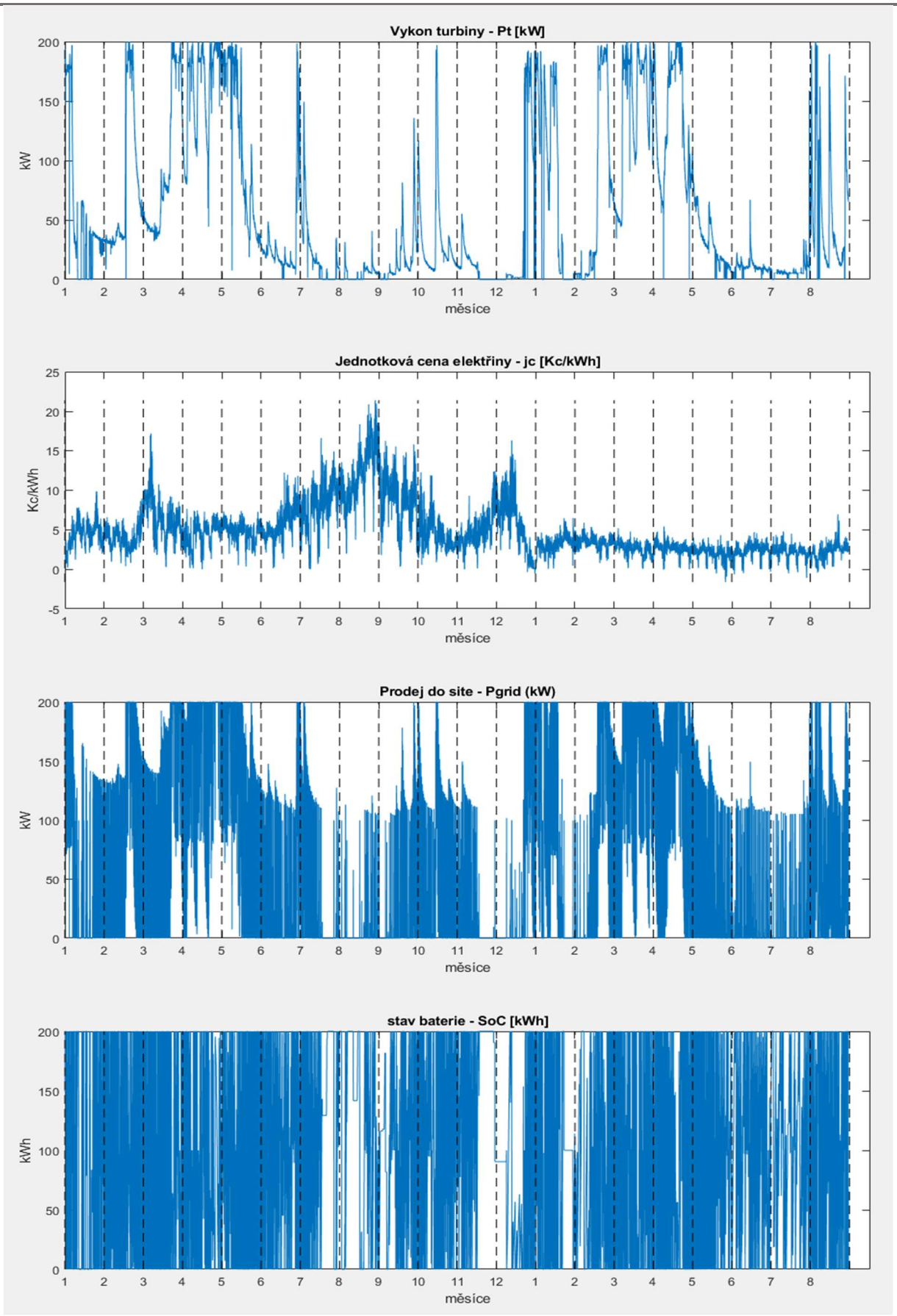
Obrázek 5.13 - Ideální optimalizace 20 měsíců – detail Pt

Při detailním pohledu na *Pgrid*, taktéž vidíme že prodej do sítě není konstantní, tak jak tomu je u výkonu turbíny (a bylo by tomu tak u prodeje elektřiny bez baterie, kde by *Pgrid*, kopírovala *Pt)*, ale téměř v každém dni je díky baterii realizován ve špičkách, tak aby došlo k prodeji, když je poptávka, a tudíž i cena elektřiny vysoká, čímž se maximalizuje zisk.

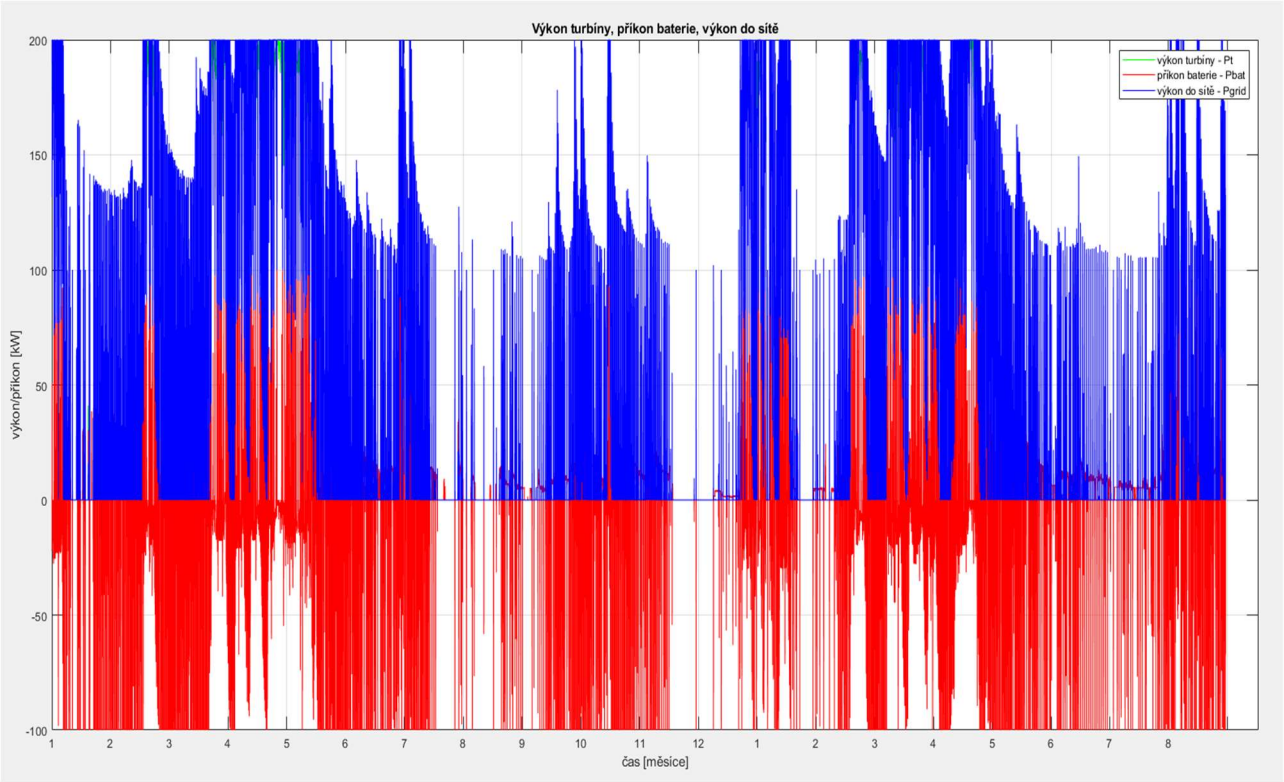


Obrázek 5.14 - Ideální optimalizace 20 měsíců – detail Pgrid

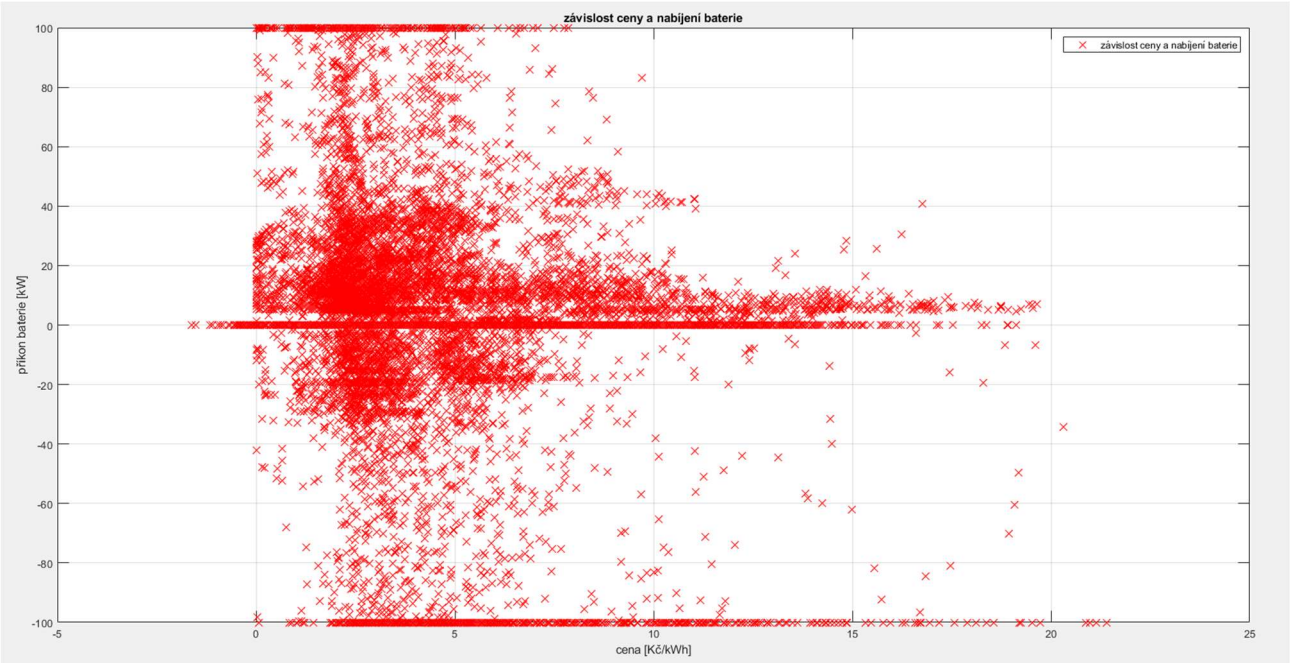
Následující grafy zobrazují totožné grafy, jako při simulaci jednoho týdne, kterou jsem představoval v minulé kapitole, pouze pro výpočet období 20 měsíců.



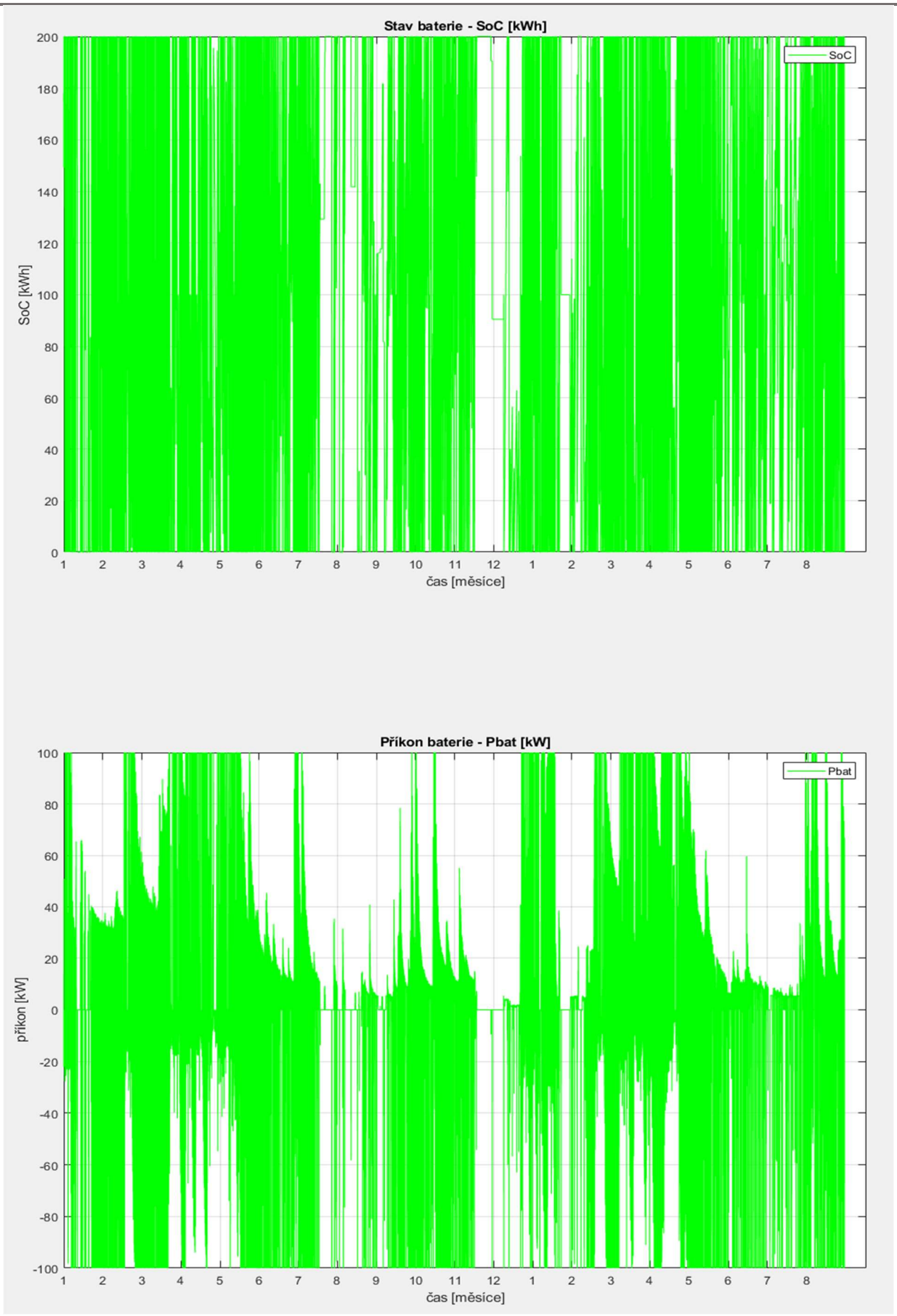
Graf 5.7 - Ideální optimalizace 20 měsíců – Graf se základními proměnnými



Graf 5.8 - Ideální optimalizace 20 měsíců – Složený graf Pt, Pbat, Pgrid



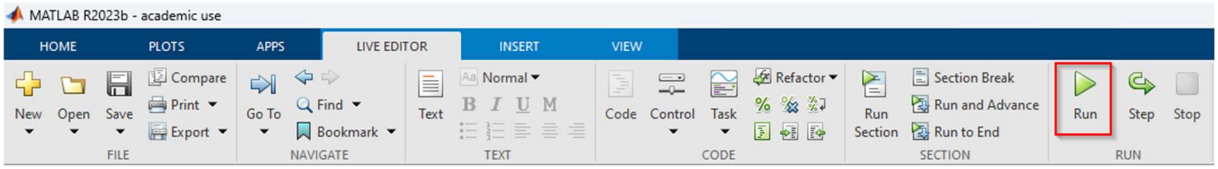
Graf 5.9 - Ideální optimalizace 20 měsíců – Závislost jc a Pbat



Obrázek 5.15 - Ideální optimalizace 20 měsíců, grafy SoC a Pbat

5.1.3. Ovládání skriptu (spuštění, přizpůsobení, zobrazení výsledků)

Ovládání toho skriptu je poměrně jednoduché. Pro spuštění simulace, tak jak byla sestavená mnou, stačí otevřít tento skript v programu MATLAB a kliknou na tlačítko „run“ v horní liště.



Obrázek 5.16 - Ideální optimalizace – spuštění

Následně proběhne výpočet celého skriptu a po úspěšném dokončení výpočtu se otevře okno „figures“ s grafy.

Pokud chce uživatel přizpůsobit simulaci pro jiné vstupní parametry, může přepsat přednastavené hodnoty na začátku skriptu.

```
clear all;

% načtení vstupních dat
load('cena_202201_202309.mat');    % vstupní hodnoty v [Kc/MWh]
load('vyroba_202201_202309.mat');    % vstupní hodnoty v [kWh]

% Vstupní parametry
Pt = vyroba_202201_202309 ; % výkon elektrárny 1. týden [kWh]
jc = cena_202201_202309 / 1000; % cena 1. týden [kc/kWh]
SoC_max = 200;       % maximální kapacita baterie [kWh]
Pgrid_max = 200;     % maximální výkon do sítě [kW]
SoC_start = SoC_max; % Start kapacita [kWh]
Pbat_min = 100;    % Maximální rychlost nabíjení baterie [kW]
Pbat_max = 100; % Maximální rychlost vybíjení baterie [kW]

n = length(Pt);
```

Obrázek 5.17 - Ideální optimalizace – úprava vstupních parametrů

Pokud je potřeba změnit vstupní řady *jc* a *Pt*, je potřeba tyto data nejprve načíst do MATLAB workspace a poté upravit tyto vstupní data ve skriptu.

```
clear all;

% načtení vstupních dat
load('cena_202201_202309.mat');    % vstupní hodnoty v [Kc/MWh]
load('vyroba_202201_202309.mat');    % vstupní hodnoty v [kWh]

% Vstupní parametry
Pt = vyroba_202201_202309 ; % výkon elektrárny 1. týden [kWh]
jc = cena_202201_202309 / 1000; % cena 1. týden [kc/kWh]
SoC_max = 200;       % maximální kapacita baterie [kWh]
Pgrid_max = 200;     % maximální výkon do sítě [kW]
SoC_start = SoC_max; % Start kapacita [kWh]
Pbat_min = 100;    % Maximální rychlost nabíjení baterie [kW]
Pbat_max = 100; % Maximální rychlost vybíjení baterie [kW]
```

Obrázek 5.18 - Ideální optimalizace – změna vstupních dat

Toto jsou veškeré úpravy, které je potřeba udělat, aby proběhla simulace se změněnými parametry. Pokud by chtěl uživatel přizpůsobit grafy, může to provést na konci skriptu kde je jejich vykreslování definováno.

5.2. Analýza vlivu velikosti baterie a maximálního výkonu (SoC_max a Pbat_max)

Na základě předchozího programu, jsem vytvořil program, který provádí analýzu vlivu velikosti baterie a jejího maximálního výkonu (*Pbat_max* a *Pbat_min*) vzhledem k vygenerovanému profitu, ve srovnání se simulací bez baterie. Tento program se skládá ze dvou MATLAB *Function,* které vzájemně spolupracují.

MATLAB *Function,* je podobný formát jako MATLAB Live skript, ale není zde možnost přidávání interaktivních částí, a vykreslování grafů probíhá ve vyskakovacích oknech. Tyto funkce lze spustit pouze voláním přes příkazový řádek, nebo jiným skriptem. Výhoda je jejich jednoduchost a možnost lepšího začlenění do dalších systémů a jejich spolupráce.

První funkce vychází ze skriptu, který jsem představil v předchozí kapitole. Jsou zde pouze drobné úpravy, tak aby tento program mohl fungovat jako MATLAB *function*, a po skončení výpočtu nebyly vykreslovány žádné grafy ani výsledky, to by vedlo ke značnému zpomalení simulace, ohromnému zmatku, a navíc to zde již není podstatné.

Druhá funkce je vytvořená tak, aby volala první funkci (program pro optimalizaci práce baterie) a postupně ho spouštěla s různými parametry a zaznamenávala výsledky. Toho je docíleno vytvoření dvojitého cyklu pomocí MATLAB funkce *for,* která opakuje určitý cyklus dle zadaného počtu opakovaní a v každém cyklu umí změnit jednotlivé parametry.

Parametry, jakými se bude daný program řídit a jak budou vypadat jednotlivé cykly, definuje uživatel před samotným spuštěním analýzy, a to načtením požadovaných parametrů pro analýzu do Matlab workspace, jako řadu hodnot SoC_max (řada kapacit baterie) a řadu Pbat_max (řada maximálních výkonů střídače/nabíječky neboli maximálních výkonů baterie).

5.2.1. Popis programu pro analýzu SoC_max a Pbat_max

Na začátku funkce, v řádku číslo 1 je definována daná funkce svojí rovnicí, která obsahuje název a vstupní parametry (obr. 5.19). Pomocí této rovnice se se funkce následně přes příkazový řádek spouští. Na začátku celého kódu jsou také načteny nebo vytvořeny jednotlivé parametry, které bude tento program následně potřebovat.

```
function [profit, profit_3D] = analyza_SoCmax_Pbatmax_vs_profit_8(SoC_max_total, SoC_step, Pbat_max_total, Pbat_step)
    % spusteni
    % [profit, profit_3D] = analyza_SoCmax_Pbatmax_vs_profit_8(1000, 10, 250, 10);

    % SoC_max_total = maximální kapacita baterie pro kterou bude analýza proveden
    % SoC_step = prirustek kapacity pro kazdy krok - jemnost reseni
    % Pbat_max_total = maximální rychlost nabíjení/vybíjení pro analýzu
    % Pbat_step = pritustek Pbat pro kazdy krok v analyze - jemnost res.

    % Inicializace proměnných
    m = SoC_max_total / SoC_step;
    n = Pbat_max_total / Pbat_step;
    profit_3D = zeros(m, n);

    % rada SoC_max a Pbat_max pro analyzu
    SoC_max = SoC_step:SoC_step:SoC_max_total;
    Pbat_max = Pbat_step:Pbat_step:Pbat_max_total;

    total_iterations = m * n;  % Celkový počet iterací
    iteration_counter = 0;     % Počítadlo pro aktuální iteraci
```

Obrázek 5.19 - Analýza optimálních parametrů – začátek programu

V další části tohoto programu, je dvojitý cyklus, vytvořený pomocí funkce *for,* který postupně spouští druhou m-funkci *optimalizace_f_8,* ta provede výpočet pomocí *linprog*, stejně jako live skript pro ideální optimalizaci, který jsem představoval v přechozí kapitole a do této funkce vrátí výsledek. Výsledek je uložen do matice, do které jsou postupně načítány výsledky pro jednotlivé parametry. Následně dvojitý cyklus, znovu spustí optimalizační m-funkci s novými parametry. Tento cyklus probíhá tak dlouho, dokud nedojde k vypočtení hodnot pro veškeré kombinace vstupní řad SoC_max a Pbat_max.

```
% Dvojitý cyklus pro analýzu Pbat a SoC
for i = 1:m
    for j = 1:n
        profit_3D(i, j) = optimalizace_f_8(SoC_max(i), Pbat_max(j));

        iteration_counter = iteration_counter + 1;
        percentage_done = (iteration_counter / total_iterations) * 100;
        fprintf('Průběh výpočtu: %.2f%%\n', percentage_done);
    end
end
```

Obrázek 5.20 - Analýza optimálních parametrů – dvojitý cyklus for

V tomto cyklu je také zabudováno „počítadlo“, které nám v průběhu výpočtu ukazuje v příkazovém řádku, kolik procent výpočtu bylo provedeno.

```
Command Window
>>
>>
>>
>> [profit_3D] = analyza_SoCmax_Pbatmax_vs_profit_8(SoC_max, Pbat_max)
Průběh výpočtu: 0.33%
Průběh výpočtu: 0.66%
Průběh výpočtu: 0.99%
Průběh výpočtu: 1.32%
```

Obrázek 5.21 - Analýza optimálních parametrů – počítadlo v příkazovém řádku

Poslední část této funkce provádí vykreslení výsledných hodnot tohoto programu do 2D a 3D grafů, pomocí MATLAB funkcí *plot* (vykreslení 2D grafů) a *surf* (vykreslení 3D grafu). Pro další analýzu výsledků (výpočet návratnosti, určení optima, vykreslení více grafů), jsem vytvořil samostatný Matlab Live skript „navratnost_inv.mlx“, který představím níže.

```
% Vykreslení 2D grafů
figure;
subplot(1, 2, 1);
plot(SoC_max, max(profit_3D, [], 2)); % Max profit pro každý SoC_max
title('Závislost velikosti baterie na zisku');
xlabel('Velikost baterie [kWh]');
ylabel('Profit [Kč]');

subplot(1, 2, 2);
plot(Pbat_max, max(profit_3D, [], 1));
title('Závislost výkonu baterie a zisku');
xlabel('Výkon baterie - Pbat [kW]');
ylabel('Profit [Kč]');

% Vykreslení 3D grafu
figure;
[X, Y] = meshgrid(SoC_max, Pbat_max);
surf(X, Y, profit_3D');
title('Závislot kapacity a výkonu baterie na zisku');
xlabel('Kapacita baterie [kWh]');
ylabel('Výkon baterie [kW]');
zlabel('Profit [Kč]');
```

Obrázek 5.22 - Analýza optimálních parametrů – Vykreslení výsledků

5.2.1.1. Spuštění programu pro analýzu SoC_max a Pbat_max

Spouštění m-funkce neprobíhá stejně jako u Matlab Live skrtiptu, kde pro spuštění můžeme použít tlačítko v horním nabídce. M-funkce musí být spuštěna přes příkazový řádek, a to její hlavní rovnicí která je na 1. řádku. Jednotlivé vstupní parametry, které tato rovnice obsahuje je nutno nahradit konkrétními hodnotami, nebo musí být jednotlivé proměnné načteny do Matlab workspace, před jejím spuštěním (příklad příkazu ke spouštění funkce je na obrázku 5.24).

Protože vstupem pro tento program jsou řady SoC a Pbat, musíme tyto řady nejprve načíst do Matlab workspace. Pokud je máme již vytvořené a uložené v některé složce, stačí na ně dvakrát kliknout v prohlížeči souborů. Následně by mělo dojít k jejich „otevření“, načtení do Matlab workspace. Pokud tyto parametry vytvořené nemáme, můžeme pro jejich vytvoření použít přímo Matlab Command Window (Matlab příkazový řádek), Live skript, případně obyčejný skript. Jakmile máme řady načteny ve workspace, můžeme zkopírovat spouštěcí příkaz a spustit výpočet. (pro rychlejší spouštění mám tento příkaz předpřipravený na začátku kódu, na řádku č. 3.

```
1  function [profit_3D] = analyza_SoCmax_Pbatmax_vs_profit_8(SoC_max, Pbat_max)
2      % spusteni
3      % load("Pbat_max.mat");
4      % load('SoC_max.mat');
5      % [profit_3D] = analyza_SoCmax_Pbatmax_vs_profit_8(SoC_max,Pbat_max);
```

Obrázek 5.23 - Analýza optimálních parametrů – Hlavní rovnice pro spuštění

```
Command Window
>> load('Pbat_max.mat')
>> load('SoC_max.mat')
>> [profit_3D] = analyza_SoCmax_Pbatmax_vs_profit_8(SoC_max,Pbat_max)
```

Obrázek 5.24 - Analýza optimálních parametrů – spuštění programu

Toto je celý proces pro spuštění, jakmile dojde k úspěšnému vypočtení, program zobrazí výsledné grafy a uloží vypočtené hodnoty „profit_3D“ do Matlab workspace.

5.2.2. Rychlost (náročnost) výpočtu

Jelikož při analýze musí program provést poměrně hodně cyklů, kompletní výpočet není rychlý. Rychlost zásadně ovlivňuje nastavená jemnost dělení *SoC_max*, *Pbat_max* pro který je analýza prováděna. Další faktor, který velmi ovlivňuje rychlost výpočtu, je délka období, pro které je výpočet prováděný a také složitost jednotlivých parametrů které jsou analyzovány. Pro výpočet s 300 variantami parametrů (15 hodnot pro Pbat_max a 20 hodnot pro SoC_max) a období 1 rok, celý výpočet probíhá 24-72 h v závislosti na „obtížnosti“ dle nastavených parametrů.

5.2.2.1. Multicore výpočet

Matlab umožňuje provádět „multicore“ výpočet. Tento způsob rozdělí výpočet (pokud to je možné) na více procesů, které probíhají současně, každý na jiném jádru. Díky tomu může dojít ke značnému zrychlení výpočtu. Matlab obsahuje v rámci Parallel Computing Tollbox, několik nástrojů pro tento výpočet:

1. parfor – Jedná se o obdobu klasické funkce *for*, s tím rozdílem, že je výpočet rozdělen na více jader a výpočty probíhají současně. Pokud bychom měli smyčku, která bude obsahovat 100 cyklů a máme 4 jádra, Matlab může teoreticky provést 25krát 4 cykly současně.
2. spmd – Tento příkaz umožňuje spustit stejný kód současně, rozdělený na jednotlivé procesy, které budou obsahovat různá data. Toto je užitečné pro práci s velkými datovými řadami, ale je nutné, aby mohl být výpočet proveden rozděleně a poté spojen dohromady, bez ovlivnění výsledku.
3. Distributed Arrays – Tato funkce umožňuje ukládat data rozloženě na různé procesy, což může minimalizovat paměťové nároky.
4. gcp – Tento příkaz vyvolá informace o aktuálních paralelních procesech.

Je potřeba mít na paměti, že použití paralelního výpočtu není vhodné pro každou úlohu. Především není vhodný pro úlohy, kde výpočet nelze rozdělit nezávisle na více částí. Použití tohoto způsobu výpočtu vždy vyžaduje individuální přístup a úpravu daného kódu. Také je potřeba myslet na to, aby docházelo ke správné synchronizaci mezi procesy a správou dat, tak aby byly výsledky korektní a je potřeba mít v Matlabu nainstalován Parallel Computing

Toolbox. Dále je také potřeba myslet na výkon zařízení, a to jestli zvládne provádět více výpočtů vzhledem k jejich složitosti.

Pro samotnou funkci linprog toto není vhodné. Její výpočet nelze rozdělit na více částí a musí být vždy proveden kompletně.

Pro analýzu SoC a Pbat by mohl být využit multicore výpočet. Jednalo by se o zařazení funkce parfor do cyklu, který provádí spouštění druhé m-funkce s různými parametry. Je zde ale potřeba skript upravit, a přizpůsobit pro tento způsob výpočtu.

Další problém s multicore výpočtem pro můj skript je vysoká náročnost funkce *linprog* na operační paměť. Tudíž při spuštění dvou výpočtů *linprog* najednou, bychom mohli narazit na problém s výkonem zařízení, konkrétně kapacitou operační paměti. Pokud se pokusím spustit hodně procesů najednou a MATLAB zaplní operační paměť a nemá kam dál ukládat data, výpočet se zastaví s chybou hláškou.

```
Error using optimalizace_f_8
Arrays have incompatible sizes for this operation.

Error in analyza_SoCmax_Pbatmax_vs_profit_9 (line 31)
    parfor i = 1:m  % Změna for na parfor pro paralelní výpočet

Related documentation
```

Obrázek 5.25 - Analýza optimálních parametrů – Chybová hláška při překročení paměti

Při defaultním nastavení funkce *parfor*, Matlab spustí tolik procesů najednou, kolik umožňuje procesor zařízení, na kterém je program spuštěn (kolik má procesor jader), bez ohledu na výkon dalších hardwarových součástí (například na počítači, který má 8 jádrový procesor dojde ke spuštění 8 paralelních výpočtů).

V závislosti na obtížnosti řešené úlohy, a její požadavky na operační paměť, je dobré pro dané zařízení zvážit snížení počtu procesů, které se najednou spustí. Toho lze v Matlab Skriptu docílit poměrně snadným způsobem. Před spuštěním funkce *parfor* která provádí multicore výpočet, nejprve manuálně vytvoříme „*Parallel Pool*“ (virtuální prostor pro multicore výpočet) s uživatelem stanoveným počtem současných procesů.

Z mého testování a zkoumání jsem došel k závěru že pro můj program a výpočet s ročními daty, je možné na zařízení které disponuje 128Gb RAM pamětí spustit 2-8 procesů najednou, dle náročnosti zvolených parametrů na výpočet (pro velmi malé hodnoty SoC_max a Pbat_max, je pro program velmi obtížné najít dané řešení, takže bude možné spustit 2-3

paralelní výpočty. Pro vysoké hodnoty, je tomu naopak a budeme moc cílit na větší počet procesů, které poběží najednou).

5.2.2.1.1. Multicore analýza SoC_max a Pbat_max

Pro můj program na analýzu SoC_max a Pbat_max jsem vytvořil verzi, pro multicore výpočet s uživatelsky definovatelným počtem procesů, které budou spuštěny pro paralelní výpočet, „analyza_SoCmax_Pbatmax_vs_profit_multicore.mlx“. Tento program je téměř totožný s programem pro analýzu bez multicore výpočtu, jen zde museli být upraveny určité části tak aby fungovali pro paralelní výpočet. Logika a fungování tohoto programu zůstala bez změny. Níže je obrázek upravené části kódu, oproti klasickému výpočtu.

```
% Vytvoření Parallel pool, pokud není již otevřen
if isempty(gcp('nocreate'))
    parpool(2); % Otevře paralelní pool s n workery, dle čísla v závorce
end

% Dvojitý cyklus pro analýzu Pbat a SoC
parfor i = 1:m  % Změna for na parfor pro paralelní výpočet
    for j = 1:n
        profit_3D(i, j) = optimalizace_f_8(SoC_max(i), Pbat_max(j));
        % Tisk průběhu výpočtu může být problémový v paralelním režimu
    end
end
```

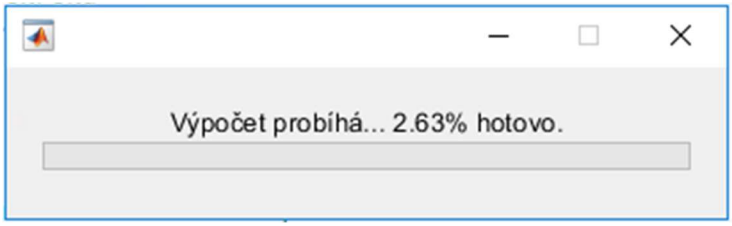
Obrázek 5.26 - Analýza optimálních parametrů – 1. verze upravené části skriptu pro multicore výpočet

Nejprve se manuálně vytvoří parallel pool a následuje upravený dvojitý cyklus pro multicore výpočet. Tento cyklus již probíhá automaticky a po dokončení výpočtu se zobrazí výsledky.

Nevýhoda této úpravy byla, že jsem musel odstranit počítadlo v příkazovém řádku, protože multicore simulace nepodporuje toto zobrazování. Z toho důvodu jsem provedl ještě jednu úpravu skriptu, díky které se při průběhu výpočtu zobrazuje tzv. waitbat (obr. 5.28), neboli ukazatel, kolik procent výpočtu je již hotovo. Malá nevýhoda zde může být mírné zpomalení, protože musí docházet ke komunikaci jednotlivých „workers“ (procesů) a „Parallel Pool Data Queue“, díky které se může zobrazovat tento ukazatel. Silně se domnívám, že pro můj program to nebude problém, jelikož zde neprobíhá hodně cyklů, ale čeká se vždy na řešení linprog. Tato hrozba zpomalení, by měla mnohem větší význam u programů, kde bude probíhat rychle hodně procesů a komunikace by zpomalovala konec a začátek dalšího procesu.

```
% Paralelní pool, pokud není již otevřen, se zadaným počtem workerů
if isempty(gcp('nocreate'))
    parpool(2); % Otevře paralelní pool s n workery (cislo v zavorce)
end

% Vytvoření objektu DataQueue
D = parallel.pool.DataQueue;

% Inicializace ukazatele pokroku
h = waitbar(0, 'Výpočet probíhá...');

% Celkový počet iterací
totalIterations = m * n;

% Callback funkce pro aktualizaci ukazatele pokroku
afterEach(D, @updateProgress);

% Funkce pro aktualizaci pokroku
function updateProgress(~)
    persistent completedIterations
    if isempty(completedIterations)
        completedIterations = 0;
    end
    completedIterations = completedIterations + 1;
    progressPercentage = completedIterations / totalIterations;
    waitbar(progressPercentage, h, sprintf('Výpočet probíhá... %.2f%% hotovo.', progressPercentage * 100));
end

% Paralelní smyčka
parfor i = 1:m
    for j = 1:n
        profit_3D(i, j) = optimalizace_f_8(SoC_max(i), Pbat_max(j));
        send(D, []); % Posíláme prázdná data, používáme pouze pro vyvolání callbacku
    end
end

% Zavření ukazatele pokroku
close(h);
```

Obrázek 5.27- Analýza optimálních parametrů – 2. verze upravené části skriptu pro multicore výpočet a "waitbar"



Obrázek 5.28- Analýza optimálních parametrů – Waitbar

Upraveným skriptem pro multicore výpočet, jsem provedl výpočet stejných dat, jako klasickým způsobem a ověřil jsem, že program pracuje správně. Výsledky, které vygeneroval byly identické ke klasickému, postupnému výpočtu, ale celý výpočet byl několikanásobně rychlejší. Výpočet jsem prováděl na zařízení, které disponovalo kapacitou operační paměti 128Gb, vzhledem k náročnosti řešených parametrů, maximální počet paralelních výpočtů, které bylo možné spustit, tak aby byl výpočet dokončen, byly 3 současné procesy (výpočty).

5.2.3. Výsledky analýzy SoC_max, Pbat_max

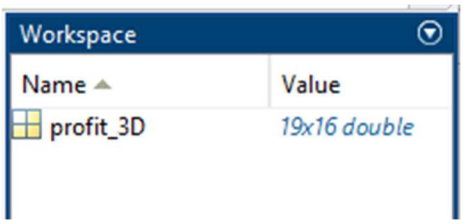
Jelikož jednotková cena elektřiny na trhu (jc) byla pro rok 2022 a 2023 značně odlišná, jak jsem již zmiňoval výše, provedl jsem analýzu pro obě tyto období zvlášť se stejnými parametry:

SoC_max = [10, 15, 20, 30, 40, 50, 75, 100, 150, 200, 250, 300, 400, 500, 600, 700, 800, 900, 1000]

Pbat_max = [5, 10, 15, 20, 25, 33, 40, 50, 60, 75, 100, 125, 150, 175, 200, 225]

Řada SoC obsahuje 19 hodnot a řada Pbat 16 hodnot, program tudíž prováděl 304 cyklů a výpočet trval 51 hodin. U zvolených parametrů jsem neřešil, jak moc jsou reálné vzhledem k výkonu jednotlivých baterií, jejich analýza je teoretická a bylo by potřeba zvolené parametry následně ověřit a zkonzultovat s odborníkem v bateriovém sektoru. Tento program pro analýzu, „analyza_SoCmax_Pbatmax_vs_profit.mlx“, který jsem představoval výše, po svém skončení vykreslí základní grafy a uloží výsledky do matlab workspace.

Hlavním výstupem tohoto programu je matice „profit_3D“, která představuje zisk provozu s baterií oproti provozu bez baterie, pro každou kombinaci analyzovaných parametrů. Tuto matici si lze z workspace uložit do požadované složky, pro další analýzu.



Obrázek 5.29 - Výstup analýzy optimálních parametrů

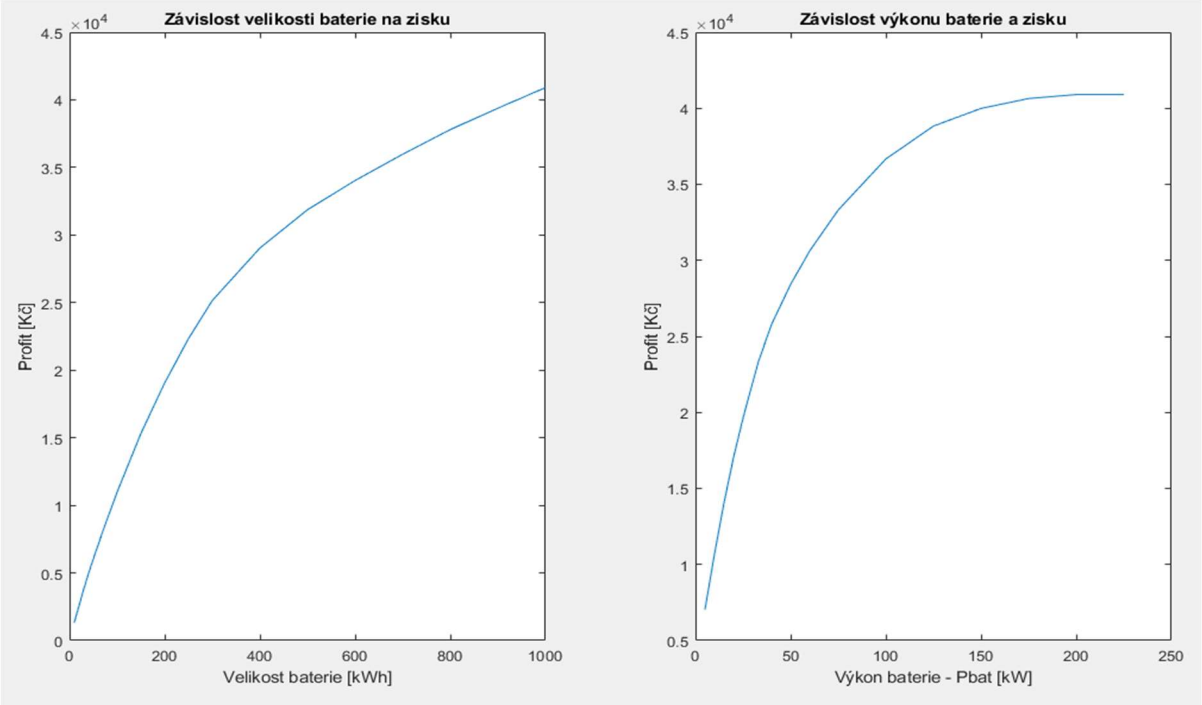
profit_3D

19x16 double

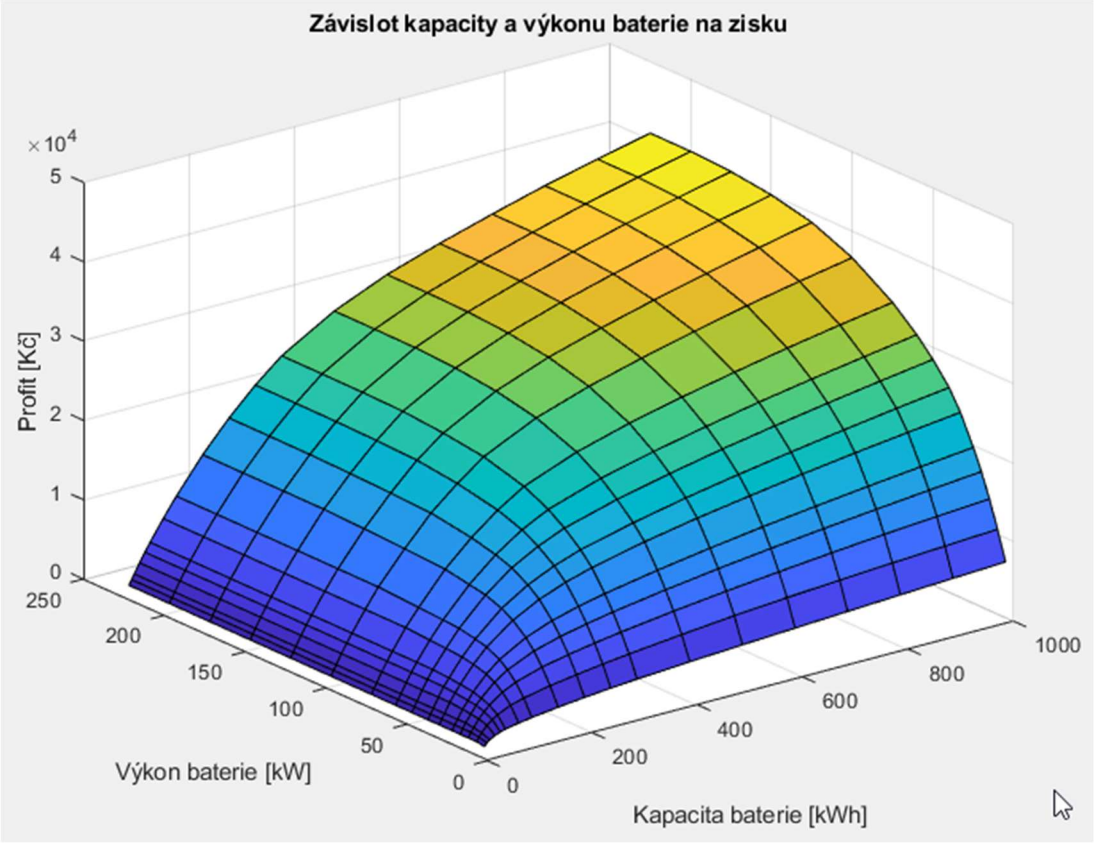
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1.7553e+04	1.9500e+04	1.9500e+04	1.9500e+04	1.9500e+04	1.9500e+04	1.9500e+04	1.9500e+04	1.9500e+04	1.9500e+04	1.9500e+04	1.9500e+04	1.9500e+04	1.9500e+04	1.9500e+04	1.9500e+04
2	2.3641e+04	2.7057e+04	2.8777e+04	2.8777e+04	2.8777e+04	2.8777e+04	2.8777e+04	2.8777e+04	2.8777e+04	2.8777e+04	2.8777e+04	2.8777e+04	2.8777e+04	2.8777e+04	2.8777e+04	2.8777e+04
3	2.8295e+04	3.4418e+04	3.6158e+04	3.7720e+04	3.7720e+04	3.7720e+04	3.7720e+04	3.7720e+04	3.7720e+04	3.7720e+04	3.7720e+04	3.7720e+04	3.7720e+04	3.7720e+04	3.7720e+04	3.7720e+04
4	3.4031e+04	4.6013e+04	5.0437e+04	5.2023e+04	5.3484e+04	5.4860e+04	5.4860e+04	5.4860e+04	5.4860e+04	5.4860e+04	5.4860e+04	5.4860e+04	5.4860e+04	5.4860e+04	5.4860e+04	5.4860e+04
5	3.7153e+04	5.4730e+04	6.1532e+04	6.5591e+04	6.7071e+04	6.9247e+04	7.0959e+04	7.0959e+04	7.0959e+04	7.0959e+04	7.0959e+04	7.0959e+04	7.0959e+04	7.0959e+04	7.0959e+04	7.0959e+04
6	3.9246e+04	6.0833e+04	7.0931e+04	7.6084e+04	7.9839e+04	8.2046e+04	8.3777e+04	8.5963e+04	8.5963e+04	8.5963e+04	8.5963e+04	8.5963e+04	8.5963e+04	8.5963e+04	8.5963e+04	8.5963e+04
7	4.3313e+04	6.9856e+04	8.7230e+04	9.7209e+04	1.0377e+05	1.0941e+05	1.1291e+05	1.1514e+05	1.1717e+05	1.1997e+05	1.1997e+05	1.1997e+05	1.1997e+05	1.1997e+05	1.1997e+05	1.1997e+05
8	4.6202e+04	7.5172e+04	9.6443e+04	1.1102e+05	1.2083e+05	1.3050e+05	1.3515e+05	1.4089e+05	1.4295e+05	1.4577e+05	1.4977e+05	1.4977e+05	1.4977e+05	1.4977e+05	1.4977e+05	1.4977e+05
9	5.0112e+04	8.3165e+04	1.0763e+05	1.2681e+05	1.4173e+05	1.5820e+05	1.6815e+05	1.7792e+05	1.8325e+05	1.9031e+05	1.9440e+05	1.9761e+05	2.0009e+05	2.0009e+05	2.0009e+05	2.0009e+05
10	5.2926e+04	8.8823e+04	1.1558e+05	1.3658e+05	1.5400e+05	1.7576e+05	1.8948e+05	2.0360e+05	2.1242e+05	2.2168e+05	2.3149e+05	2.3475e+05	2.3731e+05	2.3897e+05	2.3946e+05	2.3946e+05
11	5.5120e+04	9.3043e+04	1.2180e+05	1.4403e+05	1.6256e+05	1.8701e+05	2.0360e+05	2.2112e+05	2.3376e+05	2.4639e+05	2.5853e+05	2.6631e+05	2.6896e+05	2.7067e+05	2.7118e+05	2.7118e+05
12	5.6946e+04	9.6508e+04	1.2673e+05	1.5025e+05	1.6962e+05	1.9553e+05	2.1375e+05	2.3390e+05	2.4897e+05	2.6553e+05	2.8131e+05	2.8932e+05	2.9518e+05	2.9692e+05	2.9744e+05	2.9744e+05
13	6.0035e+04	1.0210e+05	1.3465e+05	1.6009e+05	1.8133e+05	2.0907e+05	2.2885e+05	2.5172e+05	2.6999e+05	2.9058e+05	3.1240e+05	3.2411e+05	3.3092e+05	3.3417e+05	3.3499e+05	3.3499e+05
14	6.2853e+04	1.0645e+05	1.4100e+05	1.6803e+05	1.9054e+05	2.2025e+05	2.4096e+05	2.6519e+05	2.8527e+05	3.0908e+05	3.3440e+05	3.4909e+05	3.5712e+05	3.6111e+05	3.6207e+05	3.6207e+05
15	6.5496e+04	1.1002e+05	1.4628e+05	1.7477e+05	1.9842e+05	2.2965e+05	2.5146e+05	2.7674e+05	2.9783e+05	3.2369e+05	3.5225e+05	3.6851e+05	3.7781e+05	3.8216e+05	3.8345e+05	3.8345e+05
16	6.7984e+04	1.1322e+05	1.5062e+05	1.8061e+05	2.0530e+05	2.3779e+05	2.6072e+05	2.8707e+05	3.0895e+05	3.3603e+05	3.6693e+05	3.8496e+05	3.9513e+05	4.0016e+05	4.0158e+05	4.0158e+05
17	7.0321e+04	1.1628e+05	1.5444e+05	1.8556e+05	2.1142e+05	2.4514e+05	2.6881e+05	2.9639e+05	3.1900e+05	3.4698e+05	3.7949e+05	3.9913e+05	4.1008e+05	4.1564e+05	4.1739e+05	4.1739e+05
18	7.2493e+04	1.1919e+05	1.5793e+05	1.8992e+05	2.1694e+05	2.5176e+05	2.7621e+05	3.0484e+05	3.2831e+05	3.5707e+05	3.9101e+05	4.1182e+05	4.2366e+05	4.2969e+05	4.3158e+05	4.3158e+05
19	7.4518e+04	1.2203e+05	1.6122e+05	1.9390e+05	2.2177e+05	2.5776e+05	2.8305e+05	3.1236e+05	3.3683e+05	3.6642e+05	4.0152e+05	4.2335e+05	4.3605e+05	4.4257e+05	4.4464e+05	4.4464e+05

Obrázek 5.30 - Výsledná matice analýzy optimálních parametrů

Ihned po skončení výpočtu, dojde také k vykreslení 2D Grafů, zobrazující závislost velikosti (kapacity) baterie a výkonu střídače na zisku. A 3D graf zobrazující závislost obou těchto parametrů na zisku oproti simulaci bez baterie.



Graf 5.10 - Závislost velikost baterie a výkonu střídače na zisku – 2D grafy (únor 2022)



Graf 5.11 - 3D graf závislost velikost baterie a výkonu střídače na zisku (únor 2022)

5.2.4. Výsledky analýzy pro rok 2023 a 2022 a návratnost investice

Pro detailní zpracování výsledků analýzy, jsem vytvořil Matlab skripty „navratnost_inv_2023.mlx“ a „navratnosti_inv_2022.mlx“, oba tyto skripty fungují stejně a liší se jen vstupními daty, kterými je výstup z analýzy *SoC_max* a *Pbat_max* „profit_3D“. Dalším nutným vstupem pro tento skript jsou řady hodnot nákladů na investici (ceny baterií pro jednotlivé kapacity a střídačů pro jednotlivé výkony). Tyto vstupní parametry jsou definovány hned na začátku skriptu a tím jsou jednoduše přizpůsobitelné pro danou situaci.

Pro analýzu, kterou jsem prováděl jsem vycházel z cen zařízení, které mě poskytnuli společnosti BayWa r.e. a VONSCH spol. s r.o. Pro přesné určení návratnosti dané investice by bylo potřeba provést analýzu na základě cen konkrétních produktů, které by měl v úmyslu investor pořídit.

Ceny zařízení, které jsem uvažoval pro výpočet:

Cena baterie																			
Kapacita [kWh]	10	15	20	30	40	50	75	100	150	200	250	300	400	500	600	700	800	900	1000
Cena [tisíce Kč]	110	160	220	320	430	540	850	1,100	1,650	2,200	2,750	3,300	4,400	5,500	6,600	8,000	9,000	10,000	11,000

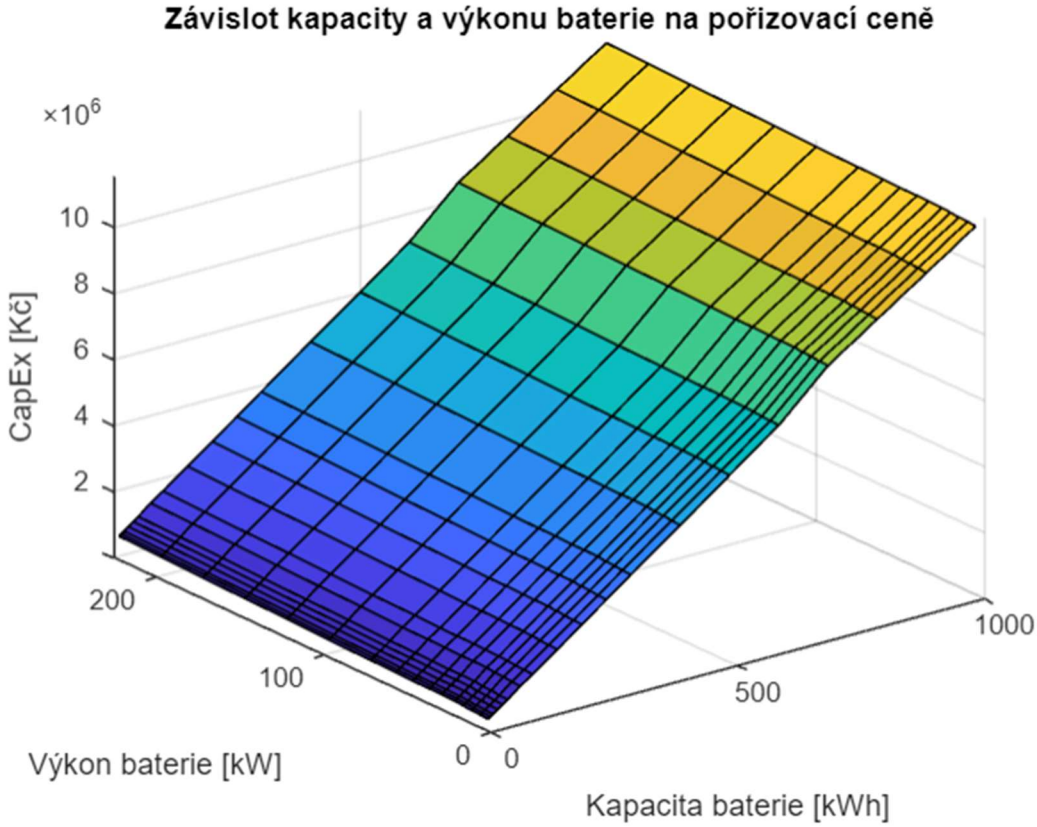
Střídač + nabíječka																
Výkon [kVa]	5	10	15	20	25	33	40	50	60	75	100	125	150	175	200	225
Cena [tisíce Kč]	145	165	185	205	230	240	245	255	265	285	321	360	400	440	480	520

Tabulka 5.1 - Ceny zařízení bateriového úložiště pro analýzu návratnosti

Veškeré ceny, jsou včetně nákladu na realizaci. Tyto vstupní náklady jsou aktuální pro rok 2023 ale uvažoval jsem úplně stejné i pro analýzu roku 2022.

Veškeré grafy a výsledky tohoto programu, je pro lepší přehlednost a možnost přizpůsobit si zobrazení, prohlížet je přímo v programu Matlab po otevření a spuštění toho skritpu. Výsledky analýzy pro rok 2022 a 2023 jdou v dané složce vždy uloženy, takže není potřeba provádět tento náročný výpočet znovu, ale stačí spusti tento skript („navratnosti_inv_2022.mlx“ pro rok 2022, nebo „navratnost_inv_2023.mlx“ pro analýzu dat výpočtu za rok 2023)

Nejprve je ve skriptu vykreslen 3D graf závislosti kapacity baterie a výkonu střídače na výši investice. Z tohoto grafu je jasně vidět, že velikost kapacity baterie, hraje mnohem větší roli na celkovou cenu investice.

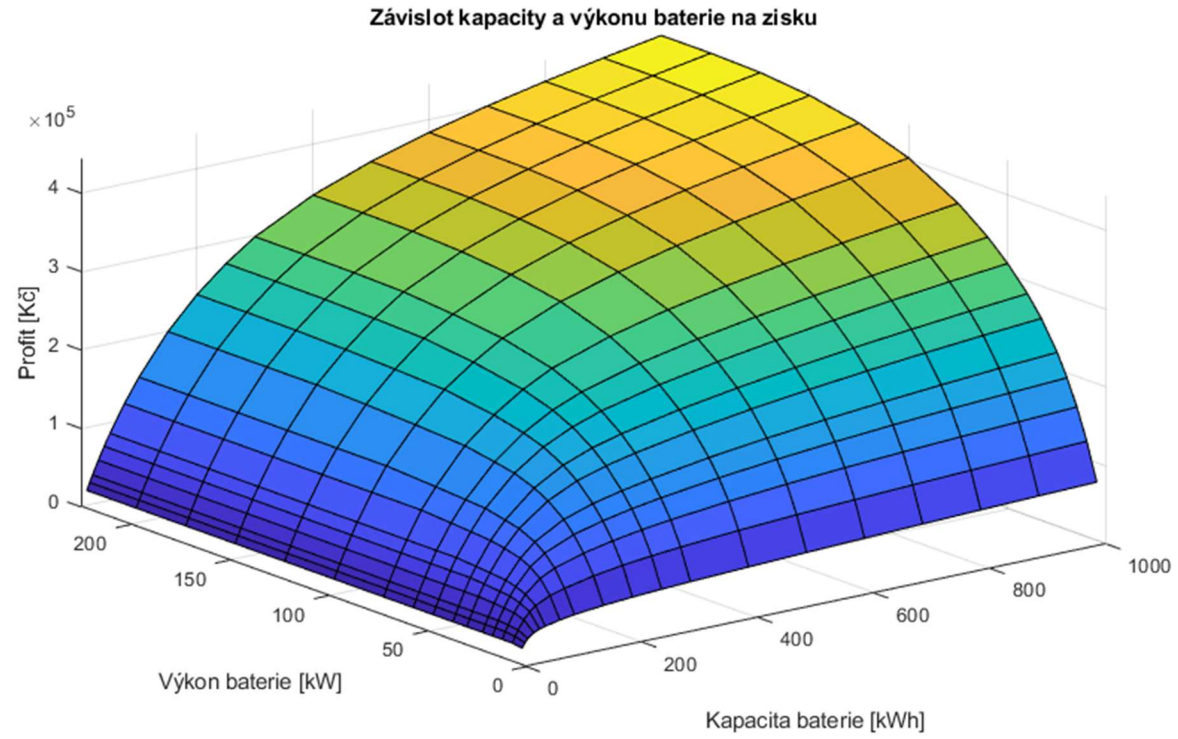


Graf 5.12 - Analýza návratnosti – Závislost investičních nákladů

Dalším grafem vykresleným tímto programem je 3D graf závisloti kapacity baterie a výkonu nabíječky/střídače na zisku. Zisk (profit) zde představuje rozdíl mezi provozem bez baterie a s baterií. Toto je hlavní výstup z programu analýza SoC_max a Pbat_max.

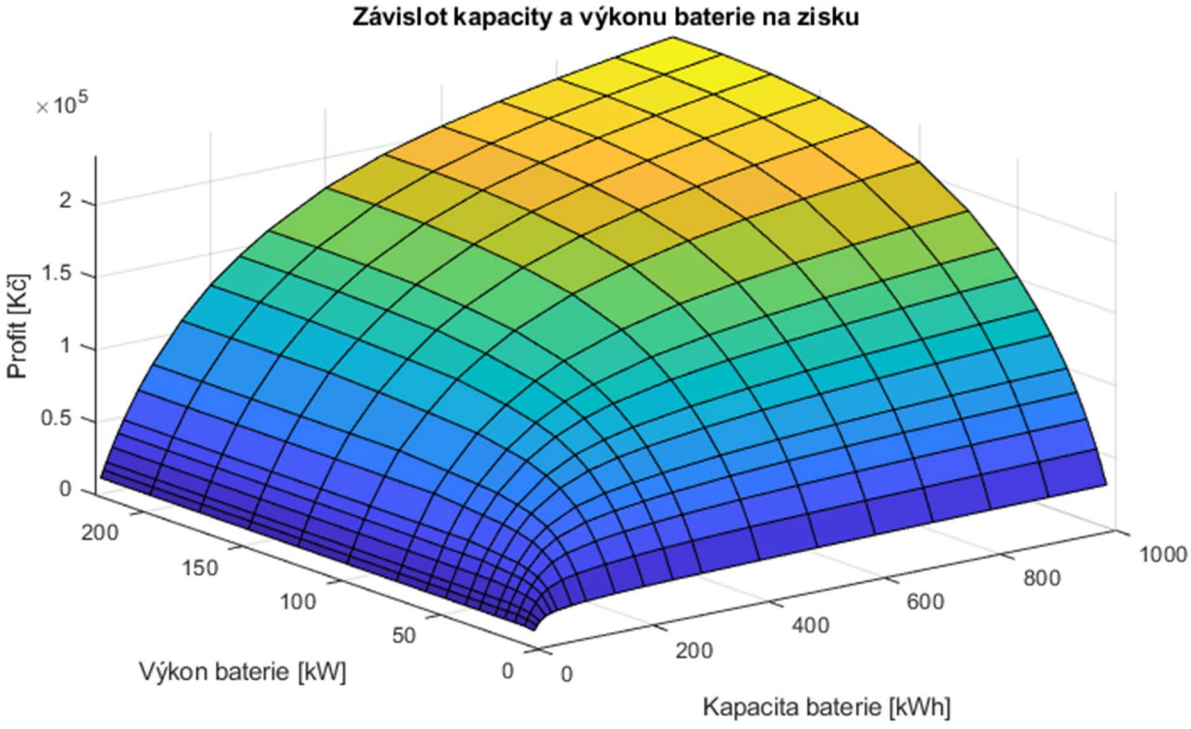
Na tomto grafu vidíme že se tato závislost tvarově pro jednotlivé roky moc neliší ale poměrně zásadně se liší svými hodnotami, což je výsledek toho že jednotková cena elektřiny byla v roce 2022 mnohem výšší než v roce 2023.

Rok 2022



Graf 5.13 - Závislost kapacity a výkonu baterie na zisku 2022

Rok 2023



Graf 5.14 - Závislost kapacity a výkonu baterie na zisku 2023

Následně program vypočítá návratnosti pro jednotlivé body (parametry SoC a Pbat), vyhledá nejnižší návratnost a určí její „souřadnice“, velikost SoC_opt (optimální kapacita baterie pro největší návratnost) a Pbat_opt (optimální velikost střídače pro co největší návratnost). Nejnižší návratnost pro rok 2023 byla významně vyšší, než jaká by byla v roce 2022.

Nejnižší návratnost v roce 2022 = 9,25 let

Nejnižší návratnost v roce 2023 = 16,75 let

Optimální návrhové parametry pro nejvyšší návratnost vyšly pro oba roky totožně:
Optimální velikost baterie = 50 kWh

Optimální výkon střídače = 50kW

Tyto zjištěné parametry budou pravděpodobně pro každou lokalitu individuální a tuto analýzu je potřeba pro každou lokalitu a dané podmínky udělat znovu.

Je také potřeba připomenout a zdůraznit, že se jedná o ideální optimalizaci se zanedbáním veškerých ztrát. Tato ideální optimalizace a její analýza může sloužit hlavně pro prvotní odhad, zdali má smysl tento problém řešit pro danou lokalitu více do hloubky, případně za jakých okolností (zvýšení cen elektřiny, pokles cen zařízení). Pokud se při ideální optimalizaci ukáže že návratnost investice je nadměrně dlouhá, nemá cenu problém zkoumat do větších detailů.

Rok 2022:

```
% Nejnižší doba návratnosti
Payback_period_min = min(Payback_period(:))
```

```
Payback_period_min = 9.2481
```

```
%souřadnice bodu - parametry bat. úložiště
[x_opt , y_opt] = find(Payback_period == Payback_period_min);
SoC_opt = SoC_max(x_opt)
```

```
SoC_opt = 50
```

```
Pbat_opt = Pbat_max(y_opt)
```

```
Pbat_opt = 50
```

Obrázek 5.31 - Nejnižší návratnost investice pro rok 2022

Rok 2023:

```
% Nejnižší doba návratnosti
Payback_period_min = min(Payback_period(:))
```

```
Payback_period_min = 16.7565
```

```
%souřadnice bodu - parametry bat. úložiště
[x_opt, y_opt] = find(Payback_period == Payback_period_min);
SoC_opt = SoC_max(x_opt)
```

```
SoC_opt = 50
```

```
Pbat_opt = Pbat_max(y_opt)
```

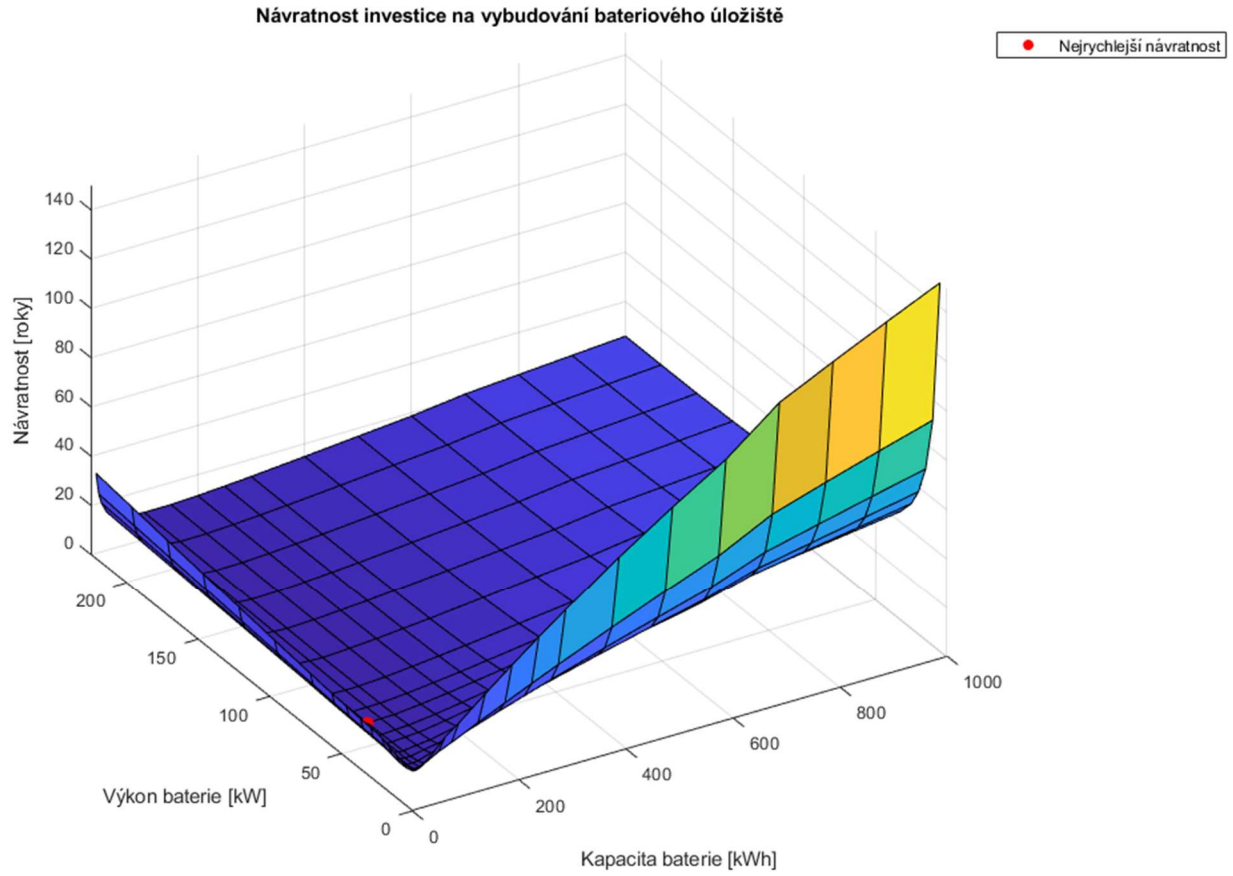
```
Pbat_opt = 50
```

Obrázek 5.32 - Nejnižší návratnost investice pro rok 2023

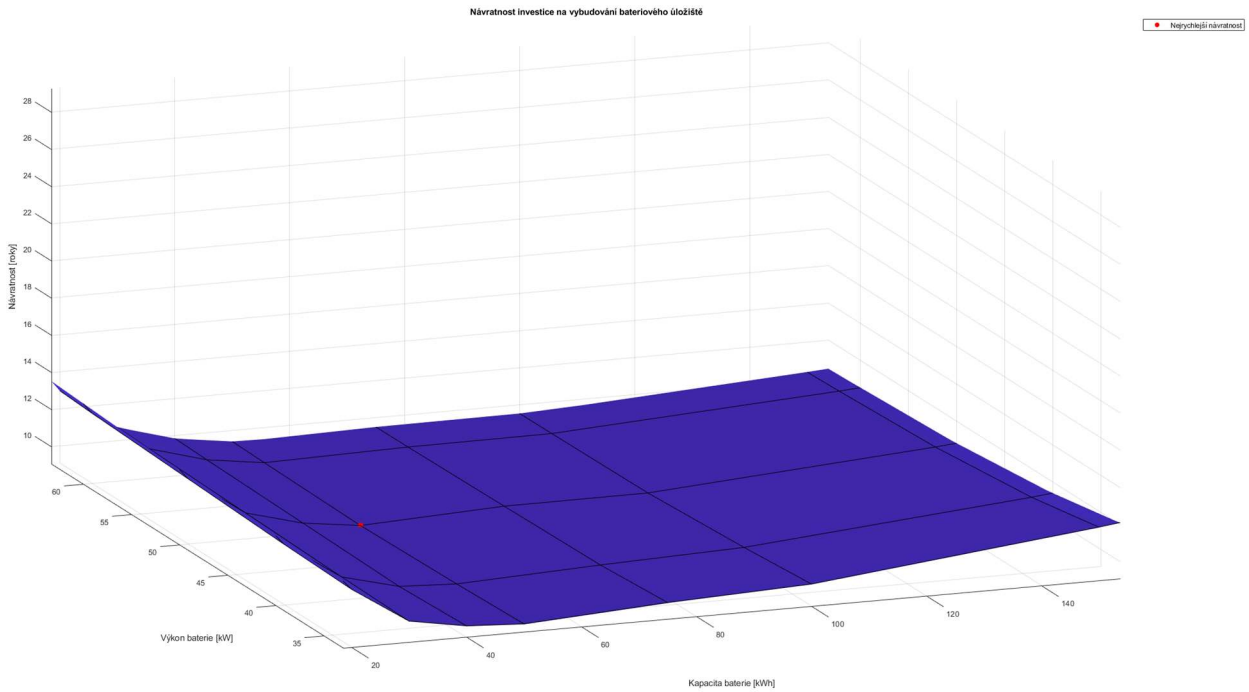
Poté dojde také k vykreslení grafu návratnosti pro jednotlivé kombinace parametrů a vykreslení nalezeného optima.

Na těchto grafech je opět patrné že návratnost investice byla v roce 2022 výrazně nižší pro všechny parametry oproti roku 2023.

Rok 2022

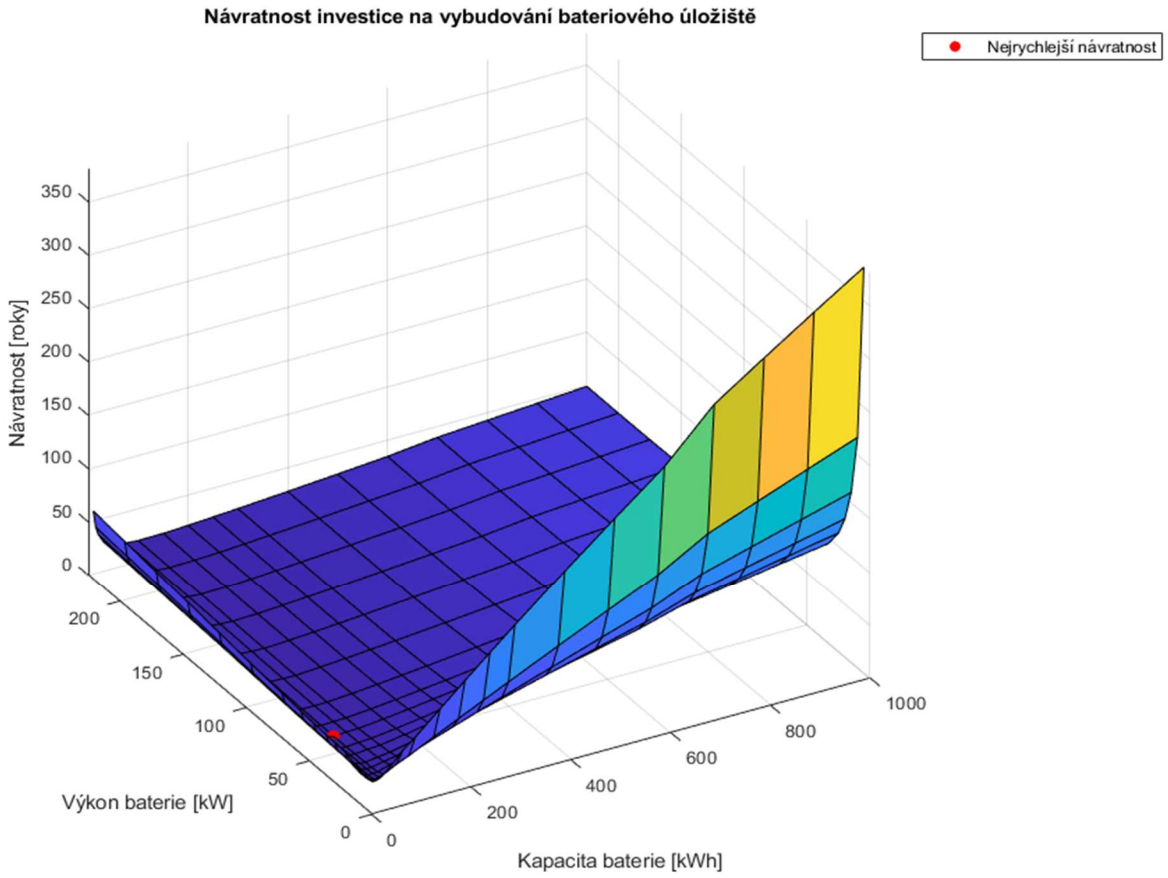


Graf 5.15 - Návratnost investice vzhledem k parametrům úložiště – 2022

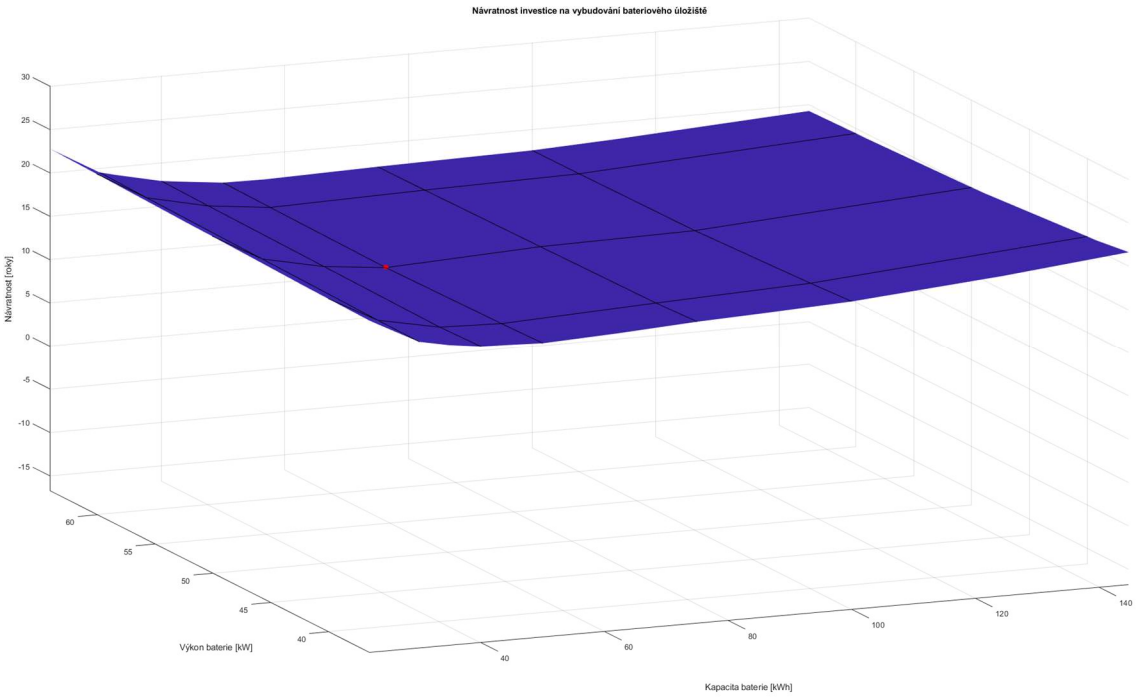


Graf 5.16 - Detail návratnost investice 2022

Rok 2023



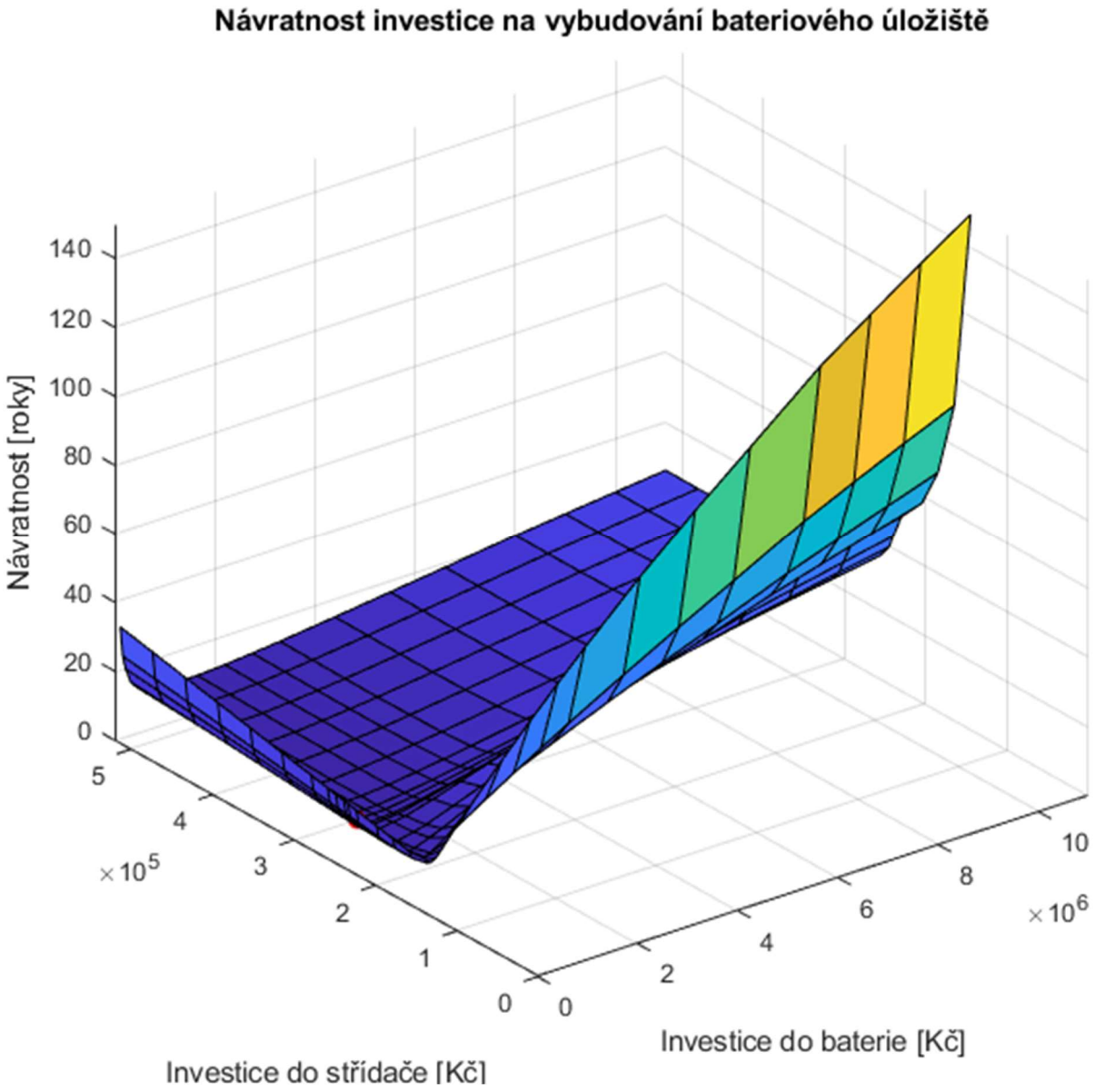
Graf 5.17 - Návratnost investice vzhledem k parametrům úložiště – 2023



Graf 5.18 - Detail návratnosti investice 2023

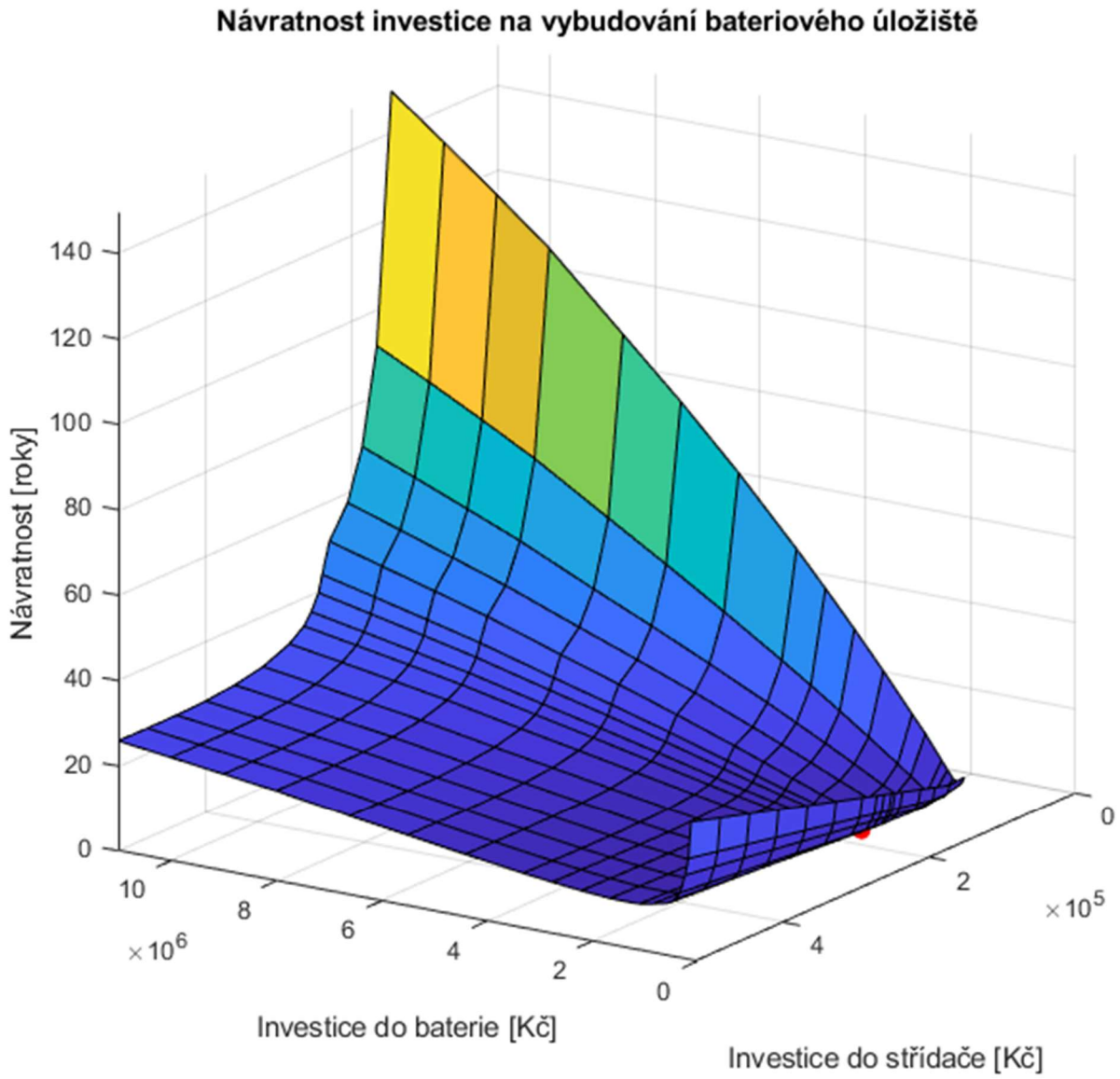
Ze stejných výsledků (vypočtených programem pro analýzu SoC_max a Pbat_max), je vytvořen graf návratnosti investice vzhledem k investičním nákladům na jednotlivá zařízení, včetně vyznačeného optima.

Rok 2022



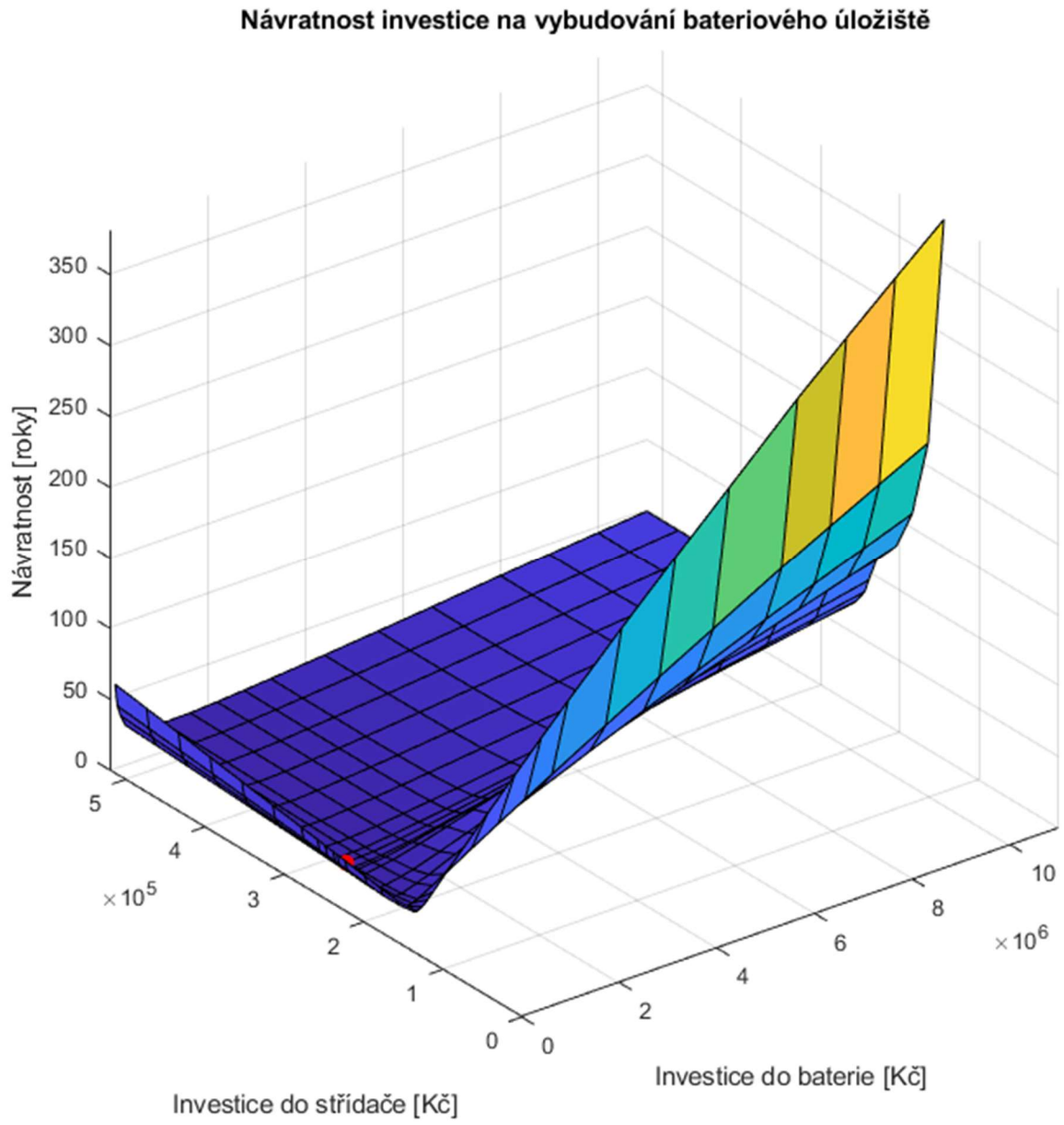
Graf 5.19 - Návratnost vzhledem k investičním nákladům 2022

Rok 2022

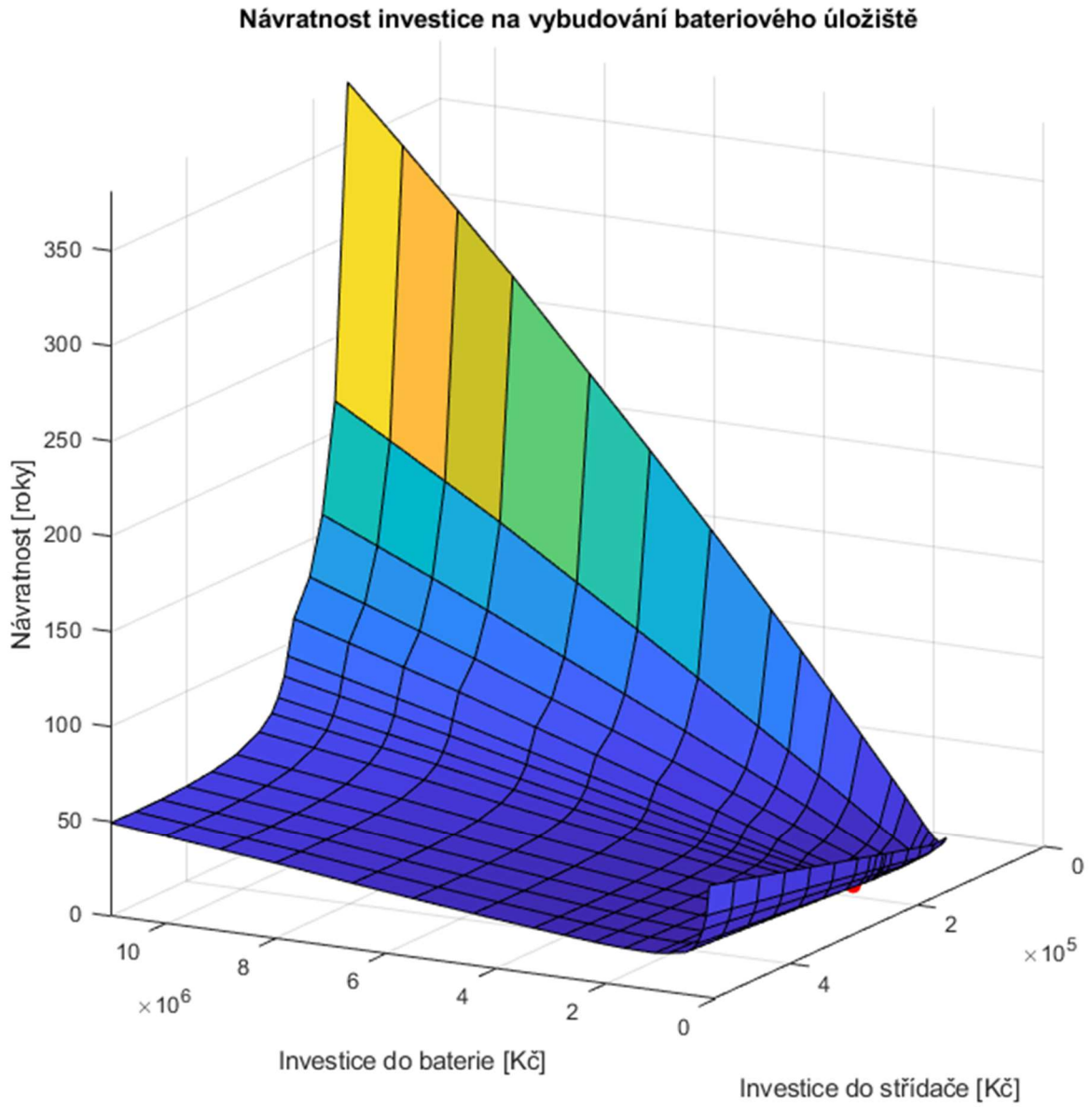


Graf 5.20 - Návratnost vzhledem k investičním nákladům 2022 - druhý pohled

Rok 2023



Graf 5.21 - Návratnost vzhledem k investičním nákladům 2023

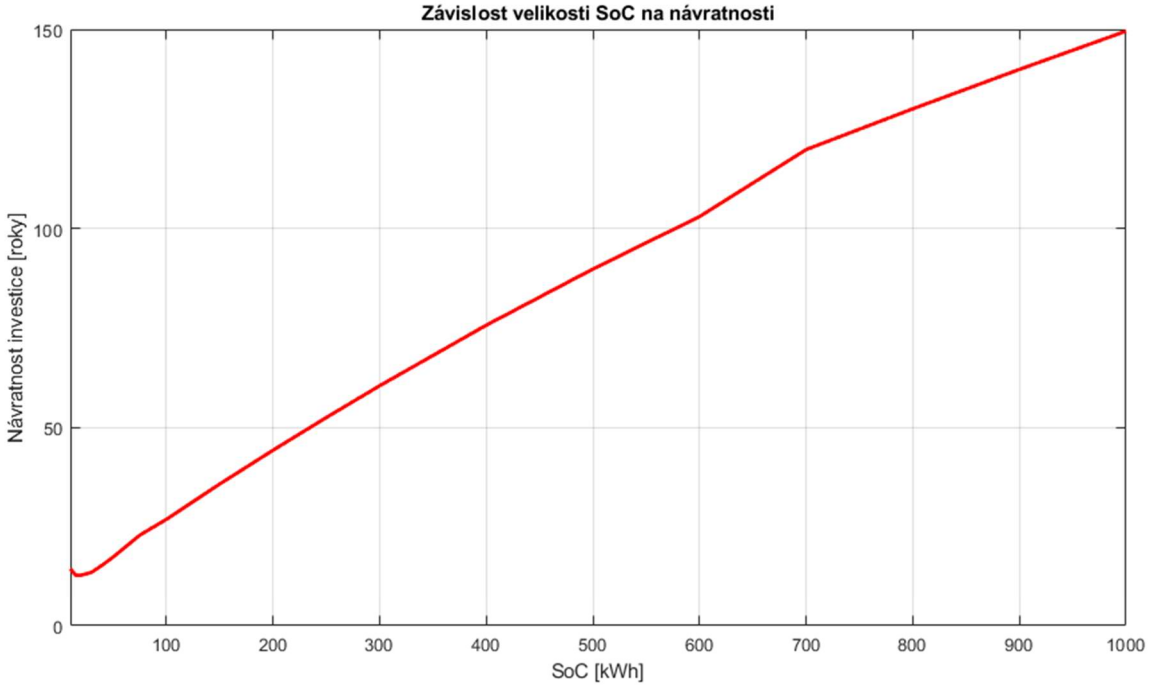


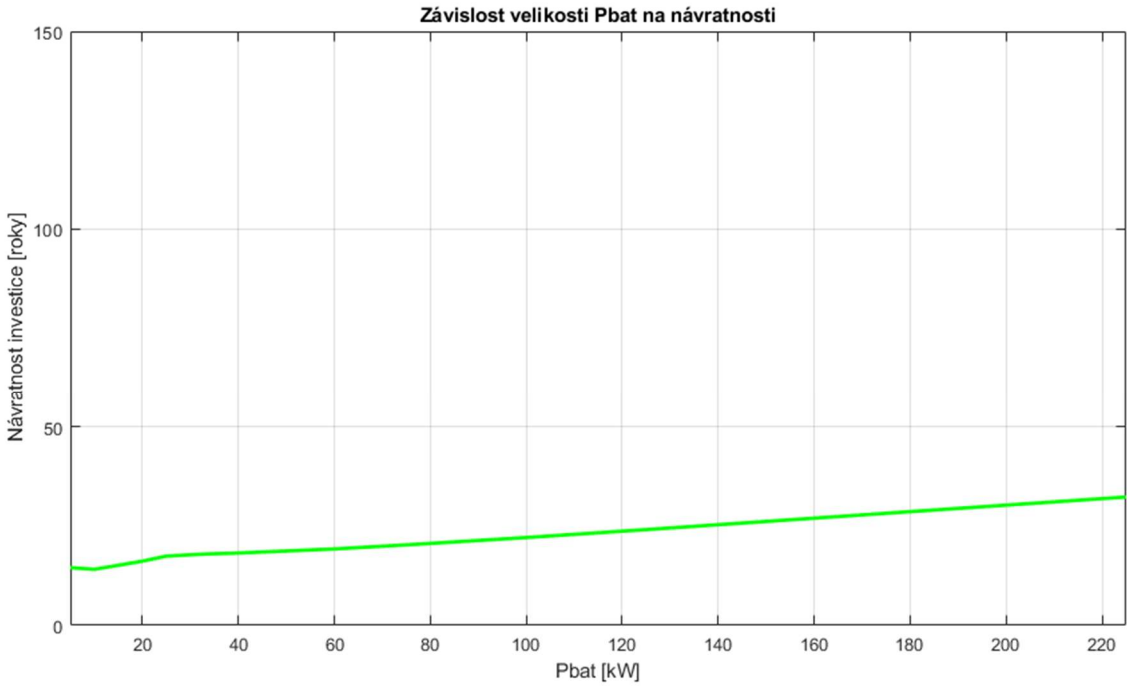
Graf 5.22 - Návratnost vzhledem k investičním nákladům 2023 - druhý pohled

Následující grafy zobrazují závislost jednotlivých parametrů (*SoC_max, Pbat_max*) na návratnosti. Je zde vidět, že zvětšování kapacity baterie *SoC_max*, mnohem více ovlivňuje návratnost investice než zvětšování výkonu střídače/nabíječky *Pbat*. To je způsobeno tím, že navýšení kapacity baterie je mnohem dražší a na zkoumané lokalitě další navyšování nemá takový vliv na zvýšení zisku. Toto může být důležitý ukazatel, když se budeme o investici rozhodovat a budeme řešit konkrétní parametry. Jelikož jsem v mé práci neřešil modelové řady ani konkrétní typy zařízení, může se stát, že při realizaci by se investor musel rozhodnout mezi větším nebo menším zařízením oproti optimu. Poté bychom věděli že nemá smysl investovat do moc větší baterie, ale zainvestování do mírně většího střídače nebude mít tak

dramatický vliv na zhoršení návratnosti. Případné postupné rozšíření baterie v budoucnu budu pravděpodobně jednoduší, a bude stačit přidat baterii do modulu (pokud již nebude plný).

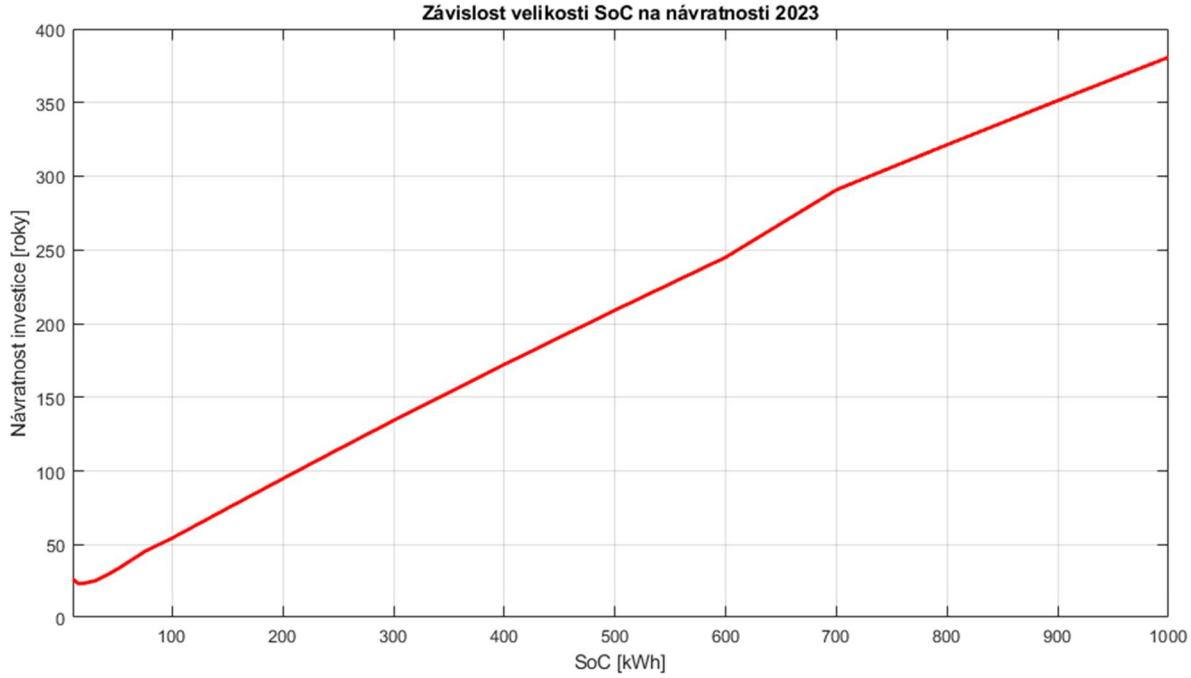
Rok 2022

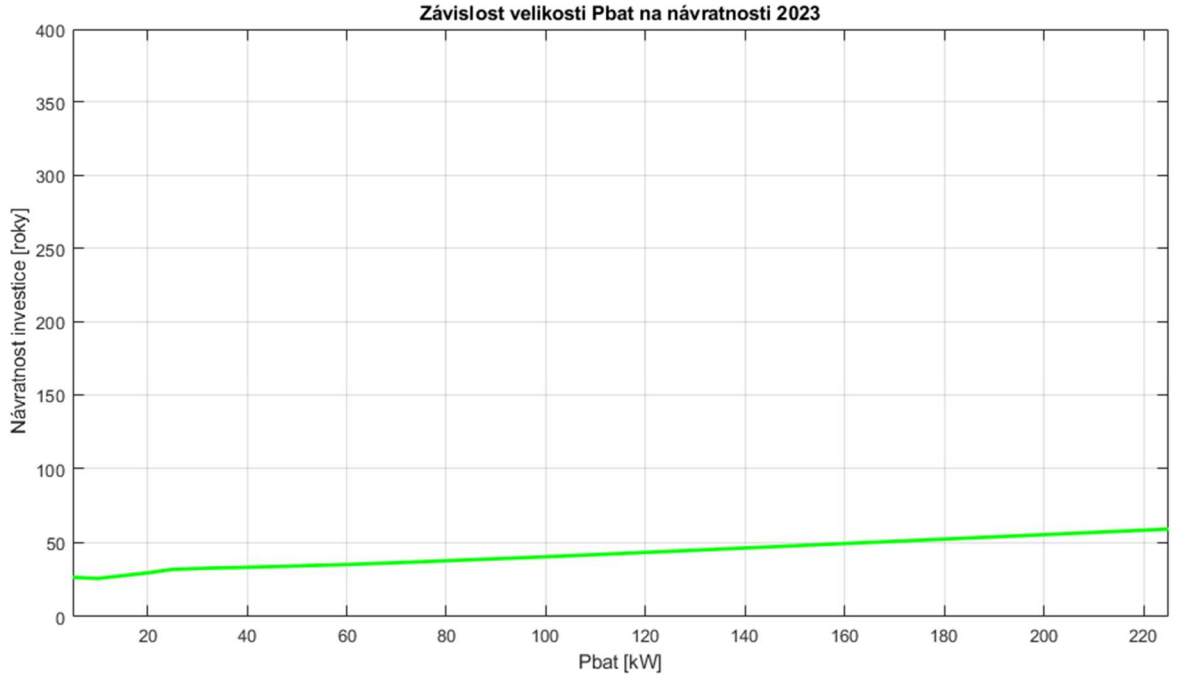




Graf 5.23 - Závislost velikosti SoC a Pbat 2022 - 2D grafy

Rok 2023





Graf 5.24 - Závislost velikosti SoC a Pbat 2023 - 2D grafy

5.3. Optimalizace pro reálnou dostupnost dat („reálné“ řízení MVE)

Další program, který jsem vytvořil, „opt_24h_2023_kraus_final.mlx“, provádí opět optimalizaci práce baterie, tentokrát na základě dostupnosti dat, tak jak by tomu bylo ve skutečnosti.

Program vychází z předchozích programů, opět pracuje na základě lineárního programování a používá funkci *linprog.* Data ohledně cen energie jsou známa vždy ve 14 hodin na 24 hodin dopředu. Data o výrobě elektrické energie turbínou jsou známa pouze pro aktuální krok. Program předpokládá že pro simulovaný interval zůstane výkon turbíny (výroba elektrické energie) *Pt* konstantní. Se znalostí a předpokladem dat provede optimalizaci chování baterie pro daný krok, v dalším kroku je provedena aktualizace na základě skutečných dat výroby. Tímto je provedena korekce skutečného stavu, který program při optimalizaci nabíjení a vybíjení baterie v předchozím kroku neznal, protože pouze předpokládal, že výroba bude stejná pro celý den, pro který je výpočet zrovna prováděn. Takto je výpočet proveden pro celý den, po hodinových krocích (24 výpočtů během jednoho dne), s postupně se snižující řadou dat (každou další hodinu se řada zkrátí o jednu hodnotu). Po výpočtu celého dne, program dostane data o vývoji cen elektrické energie *jc* na dalších 24 h a tento cyklus se opakuje. Tento algoritmus je opět vytvořen v programu Matlab ve formě *Liveskript* a vychází z předchozího výpočtového programu, který prováděl optimalizaci se znalostí dat na celé období.

Při vytváření tohoto skriptu jsem narazil na problém vycházející z toho, jak pracuje *linprog.* Tím, že *linprog* provádí maximalizaci zisku, vždy na konci daného období úplně vybil baterie (maximalizuje tím zisk, což po něm chceme). Protože úplné vybití baterie na konci každého dne není vhodný scénář pro reálné řízení, program jsem upravil tak, aby počítal s tím, že po 24 h nebude konec, ale bude následovat další úplně stejný den, jako byl ten, pro který prováděl výpočet. Jakmile dopočítá 1. den (24 cyklů) program dostane nová data a „fiktivní“ den, kterým měl pokračovat zapomene. Díky tomu nenastane, že by na konci každých 24 h měl program tendenci vybít baterii na 0.

Protože funkce *linprog* provádí optimalizaci maximálně pro 48 h, je výpočet toho programu rychlejší než mého programu „ideální optimalizace“, který jsem představoval v kapitolách výše. Výhodou je, že zde nedojde k problému simulovat dlouhé časové řady, program sice provádí hodně výpočtových cyklů, ale při řešení simplexovou metodou (*linprog*) nedojde

k zaplnění matice, a k následnému ukončení výpočtu s chybovou hláškou. Při porovnání rychlosti výpočtu pro 1letá data, je překvapivě dokonce rychlejší než program „ideální optimalizace“. Další výhodou, je mnohem větší potenciál pro využití multicore simulace.

Tento program by mohl být využit pro reálné řízení nabíjení baterie, simulaci nebo „realtime“ simulaci matematického modelu malé vodní elektrárny a bateriového úložiště.

5.3.1. Popis Live skriptu „opt_24h_2023_kraus_final.mlx“ a jeho ovládání

Skript stejně jako „ideální optimalizace“ vytvořen v Matlab Live skriptu a má velmi podobnou strukturu. Veškerá omezení, řídící rovnice a další podmínky zůstali stejné, co se změnilo je délka řady, s jakou *linprog* pracuje, a jak mu jsou poskytována data.

Na začátku skriptu je potřeba načíst vstupní data (lze pomocí funkce *load,* nebo manuálně do workspace). U řady vstupních dat je potřeba aby začínala ve 14:00, byla dostatečně dlouhá a dělitelná na celé dny (vzhledem k tomu, jak je dopočítáváno kolik cyklů bude program dělat a jak bude s daty postupovat, načtení řady, která nebude vycházet na celé dny, by způsobovalo problémy).

Dále je možné navolit parametry pro výpočet, stejně jako ve skriptu „ideální optimalizace“. Pod tím dojde k výpočtu výpočetních cyklů a připravení matic pro načítání celkových výsledků (pomocí funkce *zeros* jsou vytvořeny matice, o rozměrech délky simulovaného období se samými nulami, toto později usnadní programu načítání celkových výsledků.

```
clear all;

% je potřeba načíst dostatečně dlouhou řadu, která je dělitelná na celé dny
load("cena_2023_day_sim.mat"); % vstupní hodnoty v [Kc/MWh] start ve 14:00
load("vyroba_2023_day_sim.mat"); % vstupní hodnoty v [MWh] start ve 14:00

SoC_max = 200; % maximální kapacita baterie [kWh]
Pgrid_max = 200; % maximální rychlost vybíjení/prodeje do sítě [kW]
SoC_start_celk(1) = SoC_max; % Start. stav baterie
Pbat_min = 100;    % Maximální rychlost nabíjení baterie [kW]
Pbat_max = 100; % Maximální rychlost vybíjení baterie [kW]

dny = length(vyroba_2023_day_sim)/24; % pocet dní které budeme simulovat
n_t = length(vyroba_2023_day_sim); % počet časových úseků

% priprava variable pro nacitani_celk_vysledku
% Pt; jc; Pgrid; SoC; Pbat
Pt_celk = zeros(n_t,1);
jc_celk = zeros(n_t,1);
Pgrid_celk = zeros(n_t,1);
SoC_celk = zeros(1,n_t);
Pbat_celk = zeros(1,n_t);
```

Obrázek 5.33 – Optimalizace pro reálnou dostupnost dat – Načítání vstupních řad a parametrů

Dále je ve skriptu vytvořená smyčka, která provádí výpočet po jednotlivých hodinách pro každý den, na základě předpokladů, které jsem zmínil v úvodu kapitoly o tomto programu. Smyčka je vytvořena pomocí Matlab funkce *for*, v každém kroku jsou znovu definovány vstupní hodnoty pro daný cyklus (*jc* a *Pt*) a musí být dopočítány veškeré vstupy pro *linprog.*

```
% HLAVNÍ DENNÍ CYKLUS
% cyklus pro optimalizaci 24h (od 14h do 14h následujícího dne)
for a = 1:n_t

den = ceil(a/24); % aktuální den pro který provádíme simulaci
hodina = a - (24*(den-1));    %relativní hodina pro kterou probíhá výpočet pro aktuální den


Pt_start = vyroba_2023_day_sim(a); % aktuální výkon turbíny pro daný krok (začátek výpočtu ve 14:00)
jc_24h = cena_2023_day_sim(n_tt(den):(n_tt(den+1)-1)) / 1000; % cena pro výpočtový den [kc/kWh]

jc = [jc_24h;jc_24h];       % cenová řada pro výpočet 2x daný den - aby nedošlo k vyprázdnění baterie na 0
                            % uvažujeme pro výpočet řadu 48h

jc = jc(hodina:48,1);       % pokročení ve výpočtu o další hodinu-řada se zkrátí


Pt = ones(48, 1)*Pt_start;  % Předpoklad že výkon turbíny bude stejný, jako je v akutálním kroku
Pt = Pt(hodina:48,1);       % pokročení ve výpočtu o další hodinu-řada se zkrátí

% počáteční stav nabití pro tento krok
SoC_start = SoC_start_celk(a);
```

Obrázek 5.34 – Optimalizace pro reálnou dostupnost dat – Vstupní data pro každý krok

Omezení, se kterými pracuje *linprog* (nerovnicová omezení *A, b*, horní a dolní podmínky *ub, lb*), jsou totožná se skriptem „ideální optimalizace“. Jediný rozdíl je že musí být znovu vytvořeny pro každý časový krok tudíž jsou také zahrnuty uvnitř cyklu *for*. Vytváření těchto matic není pro program náročné a je velmi rychlé, protože se vytváří pouze pro krátkou časovou řadu, tudíž program to nezpomaluje.

```
% VSTUPY PRO LINPROG
% Omezení prodeje za záporné ceny
% při záporné ceně dojde k vypnutí turbíny
n = length(Pt);
for i = 1:n
    if jc(i)<0
        Pt(i)=0;
    end
end

% Horní a dolní hranice pro Pgrid(t)
lb = zeros(n,1); % dolní hranice (minimum energie prodané do sítě) == 0
% --> zajištění že nebudeme ze sítě baterii nabíjet to by bylo Pgrid(t)<0 což
% tato podmínka omezí == Pgrid(t)>=0

ub = Pgrid_max * ones(n,1); % horní hranice (omezení Pgrid_max) == Pgrid(t)<=Pgrid_max

% Nevybíjíme z baterie, pokud je cena záporná
ub(jc < 0) = 0;

% Aktualizace lb a ub pro Pgrid s ohledem na max rychlost nabíjení/vybíjení
for i = 1:n
    % Nabíjecí omezení, pokud je výroba větší než nabíjecí limit
    if Pt(i) > Pbat_min
        lb(i) = max(lb(i), Pt(i) - Pbat_min);
    end
    % Vybíjecí omezení, pokud je výroba menší než vybíjecí limit a stále respektujeme Pgrid_max
    ub(i) = min(ub(i), max(0, Pt(i) + Pbat_max));

    % Respektování stávajícího omezení Pgrid_max a záporné ceny energie
    ub(i) = min(ub(i), Pgrid_max);
    if jc(i) < 0
        ub(i) = 0;
    end
end

% Vytvoření matice A a b pro nerovnostní omezení
A = tril(ones(n,n));
b = SoC_max * ones(n,1);

A_min = -tril(ones(n,n));
b_min = zeros(n,1);

for i = 1:n
    b(i) = b(i) + sum(Pt(1:i)) - (SoC_max - SoC_start);
    b_min(i) = -sum(Pt(1:i)) + (SoC_max - SoC_start);
end

% Sjednocení omezení do jednoho nerovnostního omezení
A_combined = [A; A_min];
b_combined = [b; b_min];
```

Obrázek 5.35 – Optimalizace pro reálnou dostupnost dat – řídící omezení pro linprog

Dalším krokem je volána funkce *linprog*, se stejným nastavením jako v programu „ideální optimalizace“. Opět zde chceme maximalizovat zisk, a *linprog* hledá optimální výkon do sítě (prodej elektřiny – *Pgrid*).

```
% volání linprog
f = -jc; % mínus protože linprog minimalizuje a my chceme maximalizovat
options = optimoptions('linprog','Display','none');
Pgrid = linprog(f, A_combined, b_combined, [], [], lb, ub, options);
```

Obrázek 5.36 – Optimalizace pro reálnou dostupnost dat – volání linprog

V poslední části cyklu *for*, je vypočítání příkon baterie a stav nabití (*Pbat* a *SoC*) a zapisovány finální výsledky pro danou hodinu. Tím končí celý hlavní cyklus, výpočtu pro jeden časový krok (v mém případě hodinu).

```
% prikon do baterii (+ nabíjíme; - vybíjíme)
for i = 1:n
Pbat(i)=Pt(i)-Pgrid(i);
end

% Stav baterie - SoC
SoC(1) = SoC_start+Pbat(1); % počáteční stav baterie - odběr v první kroku
for i = 2:n
    SoC(i) = SoC(i-1) + Pt(i) - Pgrid(i);
end

SoC_start_celk(a+1) = SoC(1); % načítání stavu baterie pro začátek dalšího cyklu


% nacitani_celk_vysledku
% Pt; jc; Pgrid; SoC; Pbat
    % Výpočet indexů pro plnění
    segment_size = 24;
    start_index = (a - 1) * segment_size + 1;
    end_index = a * segment_size;
% plnění připravených matic
Pt_celk(a,1)=Pt(1);
jc_celk(a,1)=jc(1);
Pgrid_celk(a,1)=Pgrid(1);
SoC_celk(1,a)=SoC(1);
Pbat_celk(1,a)=Pbat(1);


end         % Konec cyklu pro celý den
% KONEC CELÉHO HLAVNÍHO CYKLU
```

Obrázek 5.37 – Optimalizace pro reálnou dostupnost dat – Výpočet Pbat, Soc a načítání výsledků

Po proběhnutí výpočtových cyklů pro každý časový krok, jsou následně automaticky provedeny kontroly logické správnosti výpočtu a vykresleny grafy s výsledky. Tyto kontroly a grafy jsou vytvořeny pomocí totožného kódu jako v programu „ideální optimalizace“, a nachází se na konci tohoto skriptu.

5.3.2. Popis M-funkce „opt_24h_2023_kraus_function.mlx“ a její ovládání

Díky vytvoření Matlab m-funkce, se stejnou strukturou jako Live skript popsaný výše, jsem mohl zařadit ukazatel průběhu výpočtu v příkazovém řádku, což Live skript neumožňuje. To může být velmi užitečné při simulaci dlouhých časových řad, protože při využití Live skriptu, uživatel nemá vůbec žádnou informaci o tom, jak rychle výpočet probíhá a v jaké části se nachází.

Další výhoda m-funkce oproti live skriptu, je lepší možnost využití v dalších funkcích a skriptech, například pro analýzu, nebo pro ladění, kde si uživatel může vytvořit skript, který bude tuto funkci spouštět s různými parametry. Díky jednoduchosti m-funkce, může být výpočet v některých případech rychlejší, zejména pokud obsahuje náročné operace nebo velké smyčky, což si myslím že je případ i mého programu. Celkově by se dalo říct že m-funkce je vhodnější pro výpočty delších časových řad.

Mnou vytvořená m-funkce „opt_24h_2023_kraus_function.mlx“, je téměř identická jako výše popsaný Live skript. Jediné změny v kódu, bylo přidání ukazatele průběhu výpočtu a jiné načtení vstupních dat, které vychází z toho jak se m-funkce spouští.

Pro spuštění této funkce je potřeba provést operace, které jsou uvedeny na jejím začátku.

1. Vyčistit workspace
2. Načíst vstupní data pro simulaci
3. Simulaci spustit pomocí jejího rovnice, přes příkazový řádek

```
Command Window
  Průběh výpočtu: 91.93%
  Průběh výpočtu: 91.95%
  Průběh výpočtu: 91.96%
  Průběh výpočtu: 91.97%
  Průběh výpočtu: 91.98%
  Průběh výpočtu: 91.99%
  Průběh výpočtu: 92.00%
  Průběh výpočtu: 92.02%
  Průběh výpočtu: 92.03%
fx Průběh výpočtu: 92.04%
```

Obrázek 5.38 – M-funkce optimalizace pro reálnou dostupnost dat – ukazatel průběhu výpočtu

```
function [Pgrid_celk, SoC_celk, Pbat_celk] = opt_24h_2023_kraus_function(cena_2023_day_sim, vyroba_2023_day_sim)

% m-funkce vytvořená na základě skriptu "opt_24h_2023_kraus_final.mlx"

% SPUSTENI
% clear all;
% load("cena_2023_day_sim.mat");
% load("vyroba_2023_day_sim.mat");
% [Pgrid_celk, SoC_celk, Pbat_celk] = opt_24h_2023_kraus_function(cena_2023_day_sim, vyroba_2023_day_sim);

SoC_max = 200; % maximální kapacita baterie [kWh]
Pgrid_max = 200; % maximální rychlost vybíjení/prodeje do sítě [kW]
SoC_start_celk(1) = SoC_max; % Start. stav baterie
Pbat_min = 100;    % Maximální rychlost nabíjení baterie [kW]
Pbat_max = 100; % Maximální rychlost vybíjení baterie [kW]
```

Obrázek 5.39 – M-funkce optimalizace pro reálnou dostupnost dat – Rozdíl vkládání vstupních dat

Veškeré operace, které je nutné udělat, jsem vypsal na začátek kódu a pro jejich provedení, stačí jednotlivé příkazy po řádcích zkopírovat, bez „%“ na začátku řádku, vložit do Matlab Command Window (příkazového řádku) a potvrdit enterem.

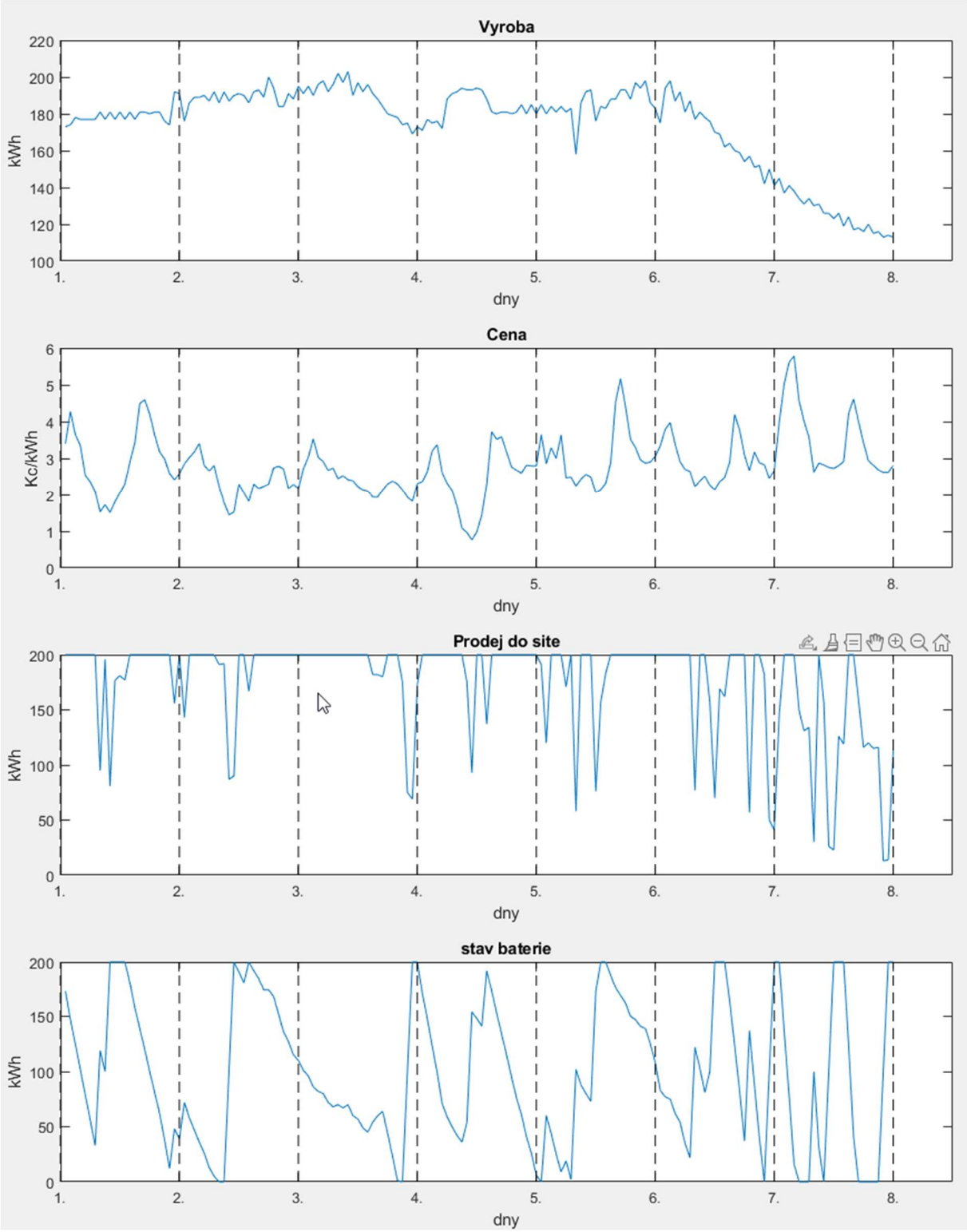
```
Command Window
  >>
  >>
  >>
  >> clear all;
  >> load("cena_2023_day_sim.mat");
  >> load("vyroba_2023_day_sim.mat");
fx >> [Pgrid_celk, SoC_celk, Pbat_celk] = opt_24h_2023_kraus_function(cena_2023_day_sim, vyroba_2023_day_sim);
```

Obrázek 5.40 – M-funkce optimalizace pro reálnou dostupnost dat – Spuštění programu

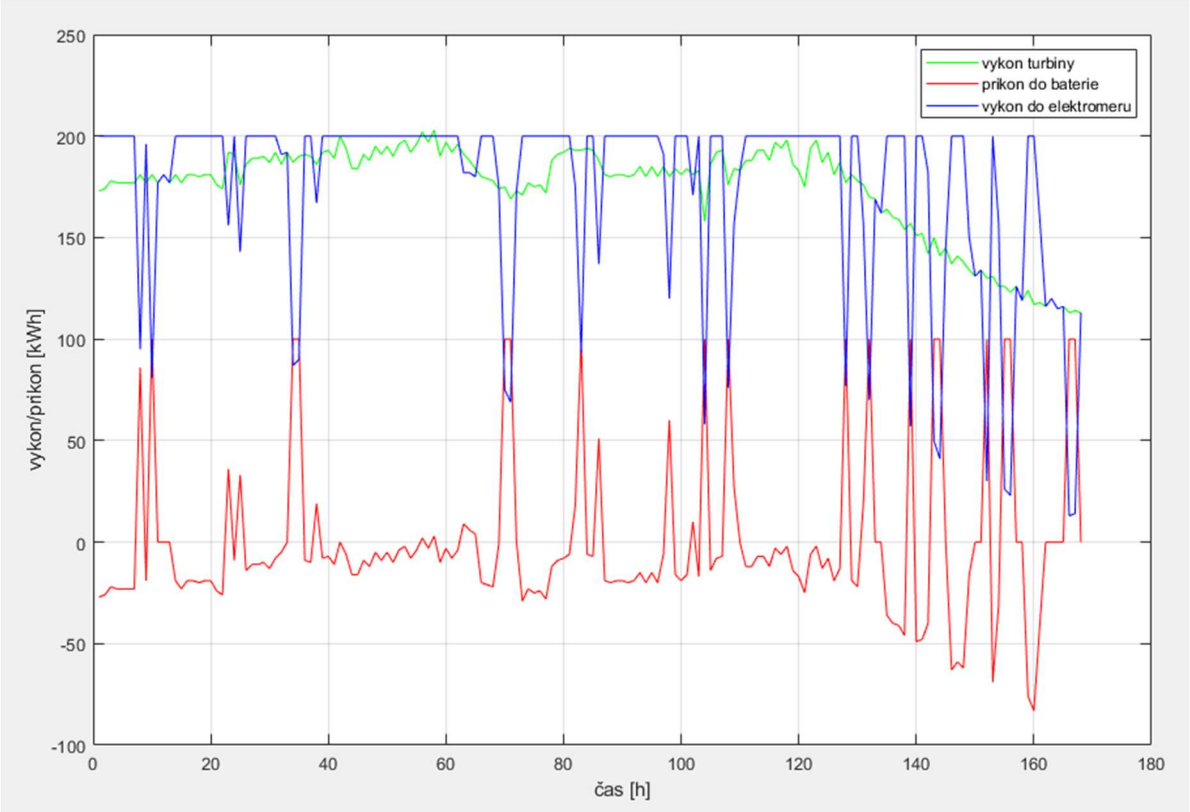
5.3.3. Výsledky řídící optimalizace

Program generuje stejné výstupy (výsledky a grafy), jako skript „ideální optimalizace“. Po dokončení výpočtu se automaticky otevře okno „figures“, ve kterém jsou karty s jednotlivými grafy.

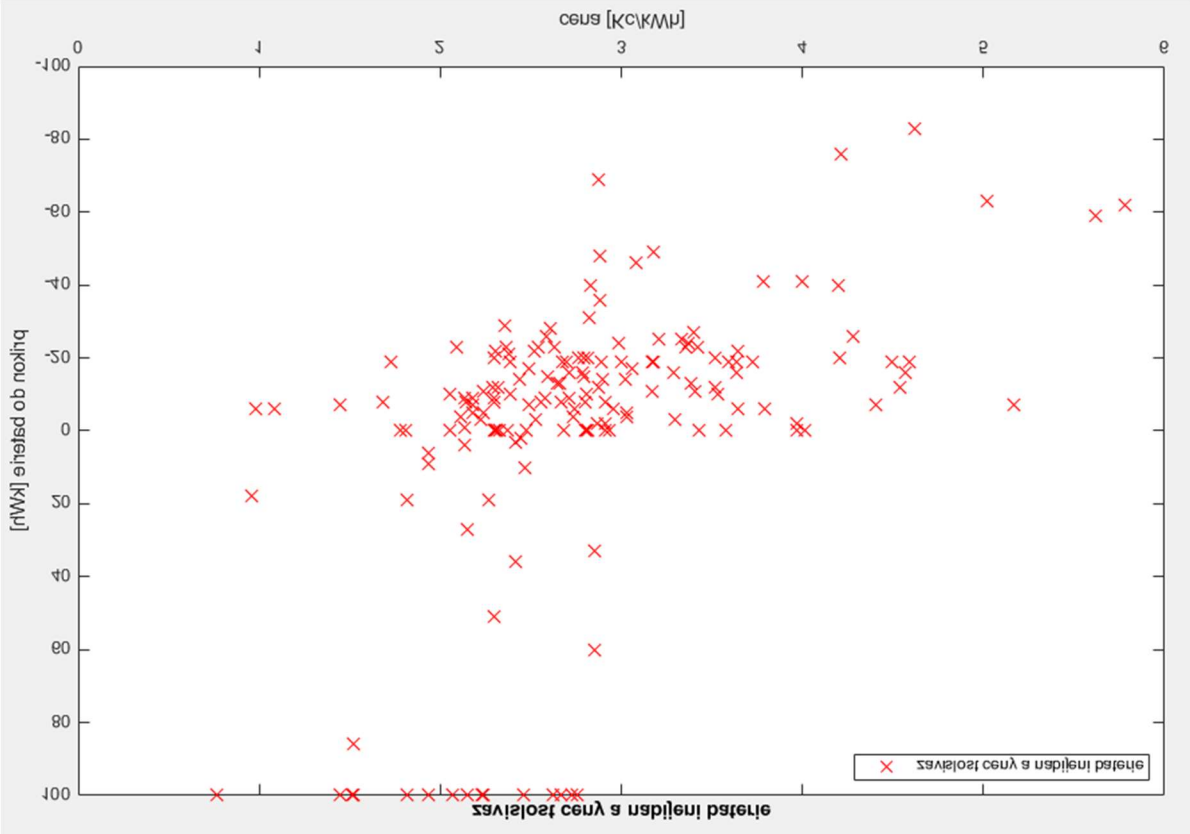
Pro přehlednost zde budu opět prezentovat výsledky výpočtu pro jeden týden v únoru, roční výsledky je vzhledem k vysokému množství dat, lepší prohlížet přímo v Matlabu, po spuštění Live Skriptu.



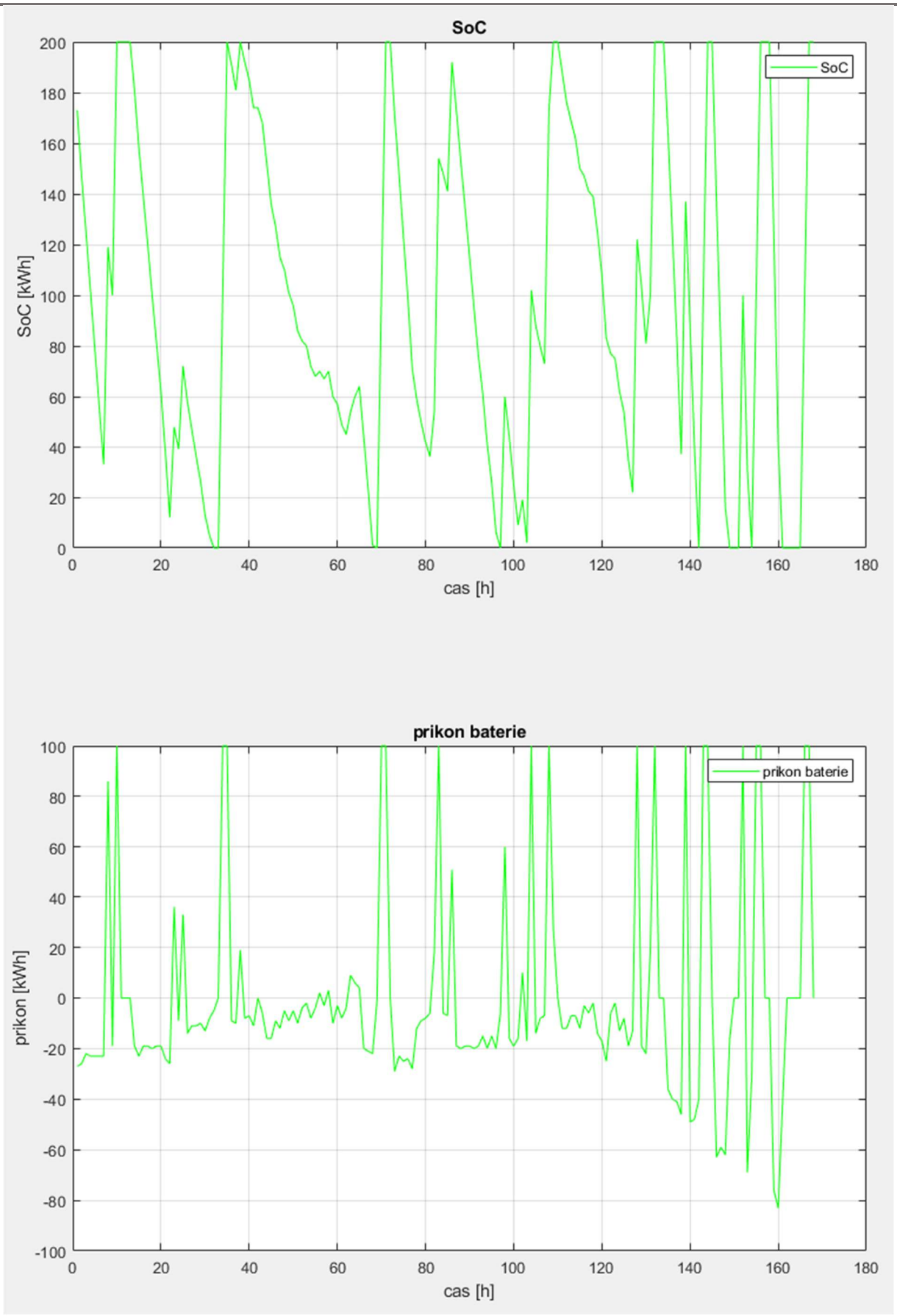
Graf 5.25 - řídicí optimalizace – Graf se základními proměnnými



Graf 5.26 - řídicí optimalizace – Složený graf Pt, Pbat, Pgrid



Graf 5.27 - řídicí optimalizace – Závislost jc a Pbat



Graf 5.28 - řídicí optimalizace - grafy SoC a Pbat

5.3.4. Porovnání „řídící“ a „ideální“ optimalizace

Po vytvoření toho programu jsem provedl analýzu pro zjištění efektivnosti tohoto algoritmu a vlivu „horšího“ řízení na zisk. Vytvořil jsem skript, který porovnává „ideální optimalizaci“ a tuto optimalizaci, která funguje na základě reálné dostupnosti dat (dále „řídící optimalizace“).

Skript pracuje s výslednými daty z výpočtu pro ideální optimalizaci a řídící optimalizaci. Porovnával jsem výsledky simulací pro rok 2023 a to především rozdíl z celkového prodeje elektrické energie do sítě (*sum(Pgrid(i) * jc(i)*).

Při zkoumání výsledků, jsem zjistil že pro různé kapacity *SoC* a maximální výkon baterie *Pbat*, jsou výsledky značně odlišné. Proto jsem pro bližší porovnání zvolil 3 scénáře:

1. *SoC_max* = 50 kWh, *Pbat_max* = 50 kW → optimální velikost dle analýzy návratnosti investice
2. *SoC_max* = 200 kWh, *Pbat_max* = 100 kW
3. *SoC_max* = 1000 kWh, *Pbat_max* = 200 kW → *Pbat_max* == *Pgrid_max*

Z vypočtených výsledků se ukázalo, že čím máme větší kapacitu baterie tím má způsob řízení větší efekt. Velké překvapení pro mě bylo, že při 1. scénáři, tzn. výpočtu dle optimálních parametrů podle mojí analýzy, je vliv řízení velmi malý. Naopak u baterie s kapacitou 1000 kWh a střídačem o výkonu 200 kW, rozdíl činil více než 3 % za jeden rok. Tento rozdíl představuje limit dané regulace a prostor pro její zlepšení.

Možné způsoby pro zlepšení „řídící“ optimalizace by se mohly zaměřovat hlavně na lepší odhad vývoje *Pt,* protože je zřejmé, že *Pt* nebude téměř nikdy konstantní. Jedno z jednodušších řešení by bylo zařazení prosté derivace u odhadu výkonu, program by pro budoucí období, o kterém ještě nemá žádná data, nepředpokládal konstantní výrobu, ale předpokládal by že výroba bude růst nebo klesat dle trendu. Dále by bylo vhodné přemýšlet o více sofistikovaných možnostech předpovědi vývoje výkonu. Vyjít z chování v jednotlivých měsících a v rámci průběhu během dne, dle analýzy historických dat, a provést modelování předpokládaného průběhu *Pt* a *jc* na větší dobu dopředu. Byl by zde také prostor pro aplikaci srážkoodtokového modelu pro předpověď průtoku na základě srážek v daném povodí.

Zlepšování regulace při reálném řízení také otevírá prostor pro aplikaci AI technologie.

```
Porovnání při SoC = 50 ; Pbat = 50
Výnos ideální optimalizace je vyšší o: 2301.473138 Kč
Což představuje: 0.168312 procent
Celkový zisk ideální optimalizace: 1367384.127578 Kč
Celkový zisk řídící optimalizace: 1365082.654440 Kč
```

Obrázek 5.41 - Porovnání ideální a řídící optimalizace – výsledky pro SoC 50 a Pbat 50

```
Porovnání při SoC = 200 ; Pbat = 100
Výnos ideální optimalizace je vyšší o: 9962.344300 Kč
Což představuje: 0.691069 procent
Celkový zisk ideální optimalizace: 1441583.798683 Kč
Celkový zisk řídící optimalizace: 1431621.454383 Kč
```

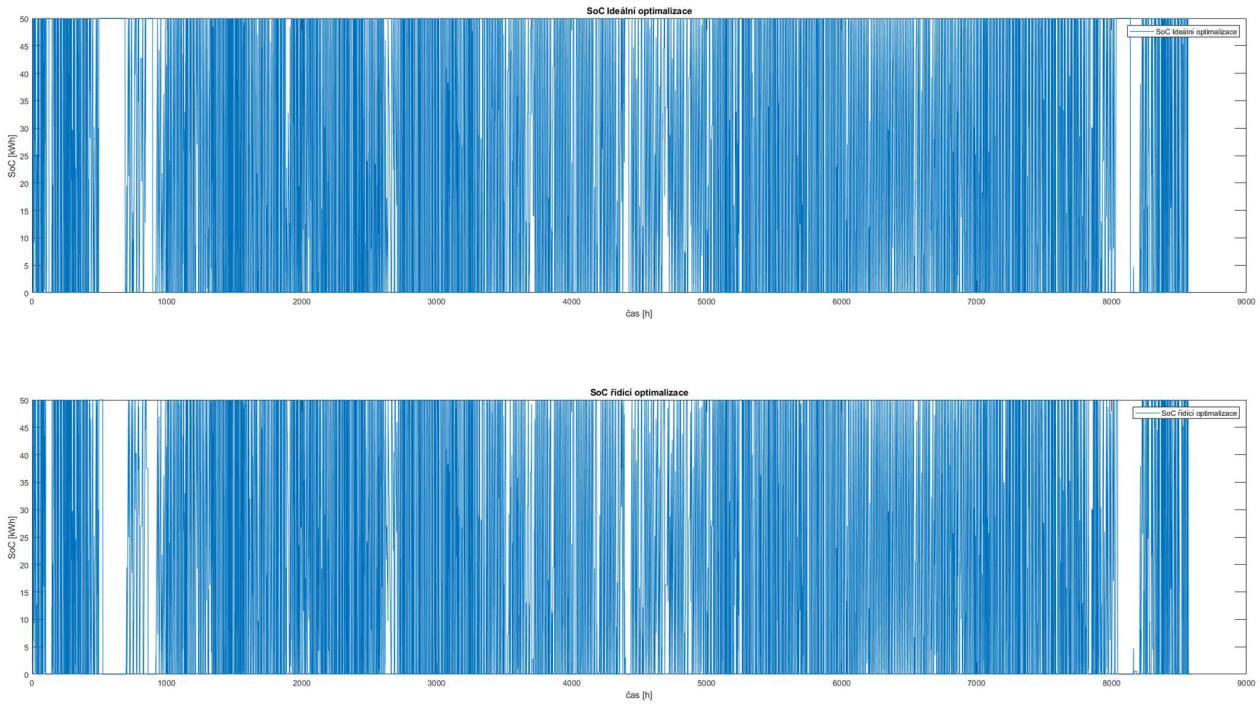
Obrázek 5.42 - Porovnání ideální a řídící optimalizace – výsledky pro SoC 200 a Pbat 100

```
Porovnání při SoC = 1000 ; Pbat = 200
Výnos ideální optimalizace je vyšší o: 49807.031860 Kč
Což představuje: 3.210796 procent
Celkový zisk ideální optimalizace: 1551236.435107 Kč
Celkový zisk řídící optimalizace: 1501429.403248 Kč
```

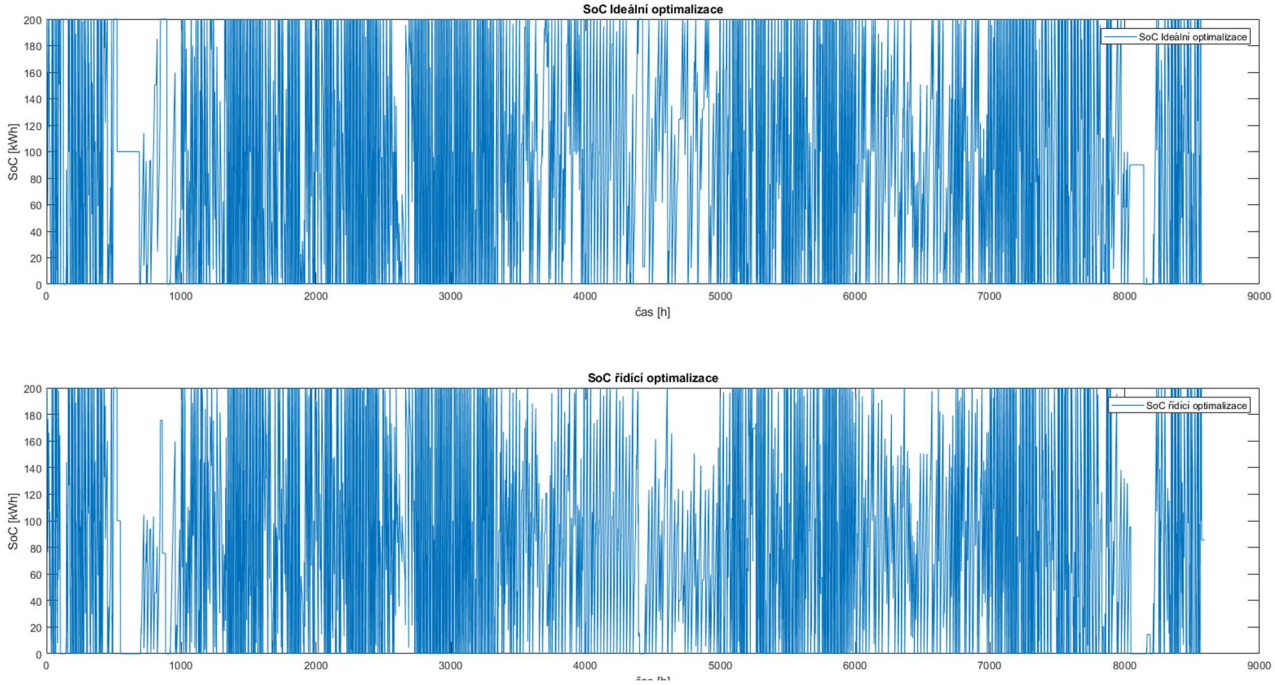
Obrázek 5.43 - Porovnání ideální a řídící optimalizace – výsledky pro SoC 1000 a Pbat 200

V následující části jsou vykresleny jednotlivé grafy, porovnávající průběh *SoC*, pro ideální a řídící optimalizaci, pro všechny výše zmíněné případy, seřazeny vzestupně dle vstupních parametrů. U největší kapacity baterie je jednoznačně vidět, že při řídící optimalizaci, téměř nedojde k využití maximálního potenciálu, zde by program jednoznačně potřeboval vylepšit v oblasti předpovědi průběhu *Pt* a případně i předpověď jednotkové ceny na delší dobu dopředu.

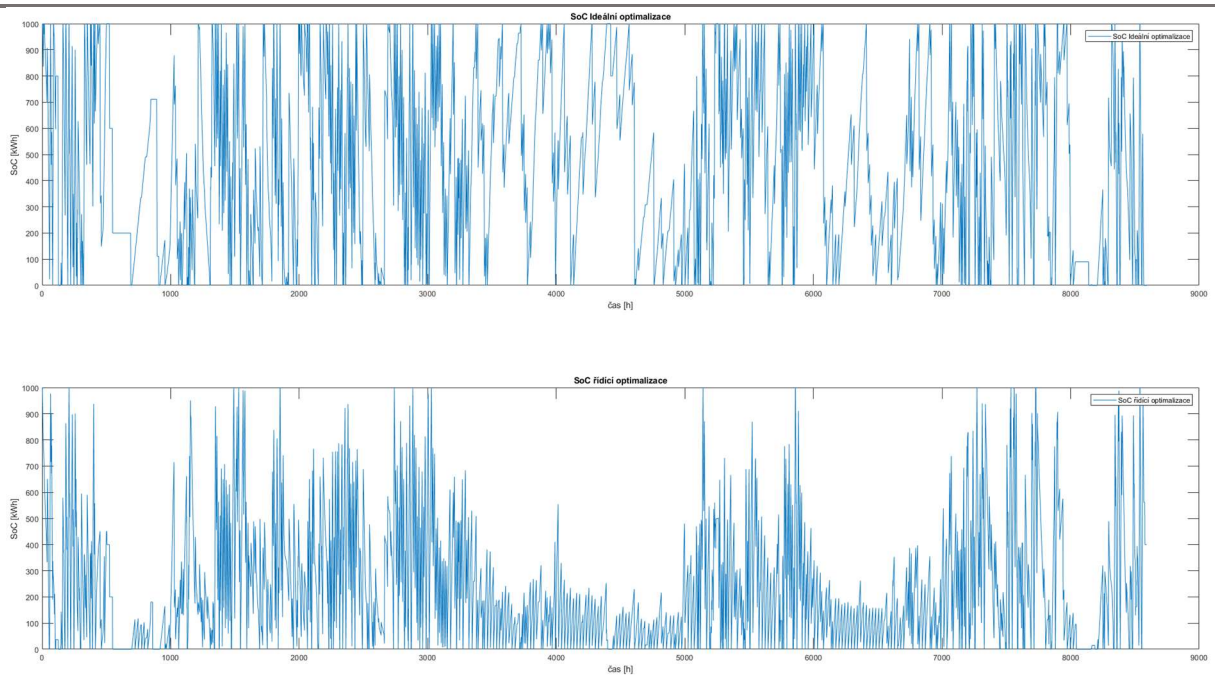
Na základě tohoto docházíme k závěru, že větší baterie znamená nejen vyšší investici do zařízení, ale bylo by také potřeba značně více investovat do řídícího softwaru a předpovědi.



Graf 5.29 - Porovnání ideální a řídící optimalizace – průběh SoC pro SoC 50 a Pbat 50



Graf 5.30 - Porovnání ideální a řídící optimalizace – průběh SoC pro SoC 200 a Pbat 100



Graf 5.31 - Porovnání ideální a řídící optimalizace – průběh SoC pro SoC 1000 a Pbat 200

5.3.5. Verze programu s možností omezení efektivní kapacity baterie

Na základě programu pro optimalizaci s reálnou dostupností dat („řídící“ optimalizace), jsem vytvořil jeho rozšířenou verzi, „opt_24h_1w_kraus_efkt_kap_final.mlx“, která uživateli umožňuje omezit efektivní kapacitu baterie.

To může být využito pro simulaci, chtěného snížení kapacity baterie (nabíjení a vybíjení v určitých mezích), pro zlepšení životnosti baterie a pro simulaci postupné degradace baterie, spouštěním přes externí skript, kde by se každý cyklus snížila kapacita baterie a stanovenou hodnotu.

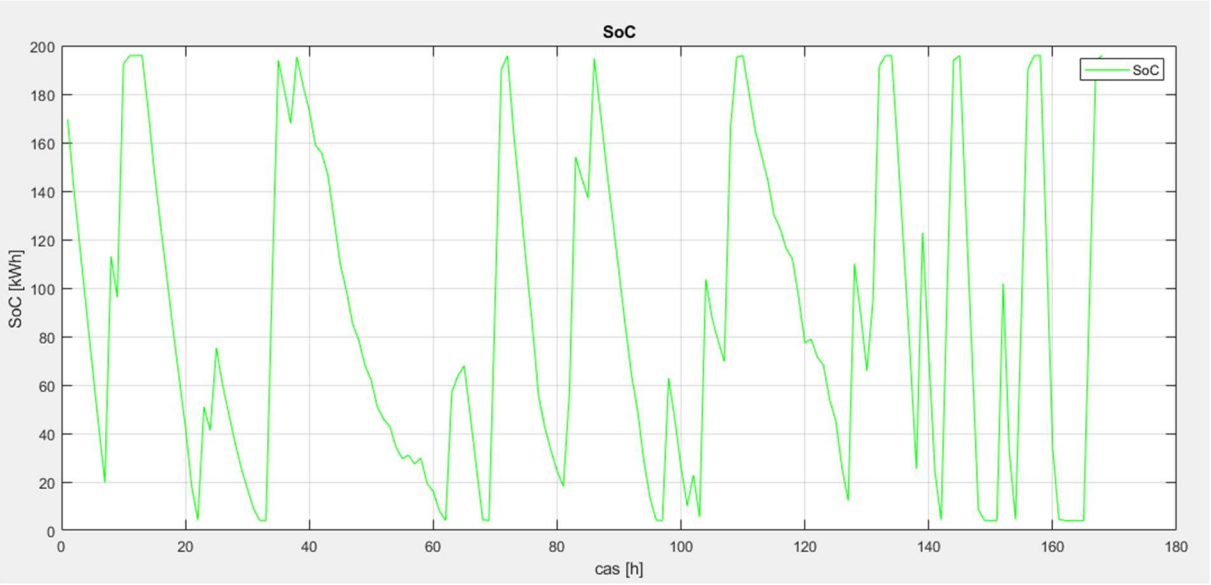
Snížení kapacity baterie je v tomto skriptu vyjádřeno v procentech a lze ho nastavit hnedka na začátku celého kódu. Další části jsou velmi podobné programu pro řídící optimalizaci, pouze jsou upraveny podmínky pro *linprog*, tak aby počítal pouze s omezenou (efektivní) kapacitou baterie.

```
clear all;

load("unor_cena_kcmwh.mat"); % vstupní hodnoty v [Kc/MWh] !!!
load("unor_vyroba_mwh.mat"); % vstupní hodnoty v [MWh] !!!

SoC_max = 200; % maximální kapacita baterie [kWh]
Pgrid_max = 200; % maximální rychlost vybíjení/prodeje do sítě [kW]
SoC_start_celk(1) = SoC_max; % Start. stav baterie
Pbat_min = 100;    % Maximální rychlost nabíjení baterie [kW]
Pbat_max = 100; % Maximální rychlost vybíjení baterie [kW]

SoC_ef_upper_lim = 2; % horní omezení kapacity baterie [%]
SoC_ef_lower_lim = 2; % dolní omezení kapacity baterie [%]
```

Obrázek 5.44 - Optimalizace omezení SoC ef – vstupní parametry

```
% Vytvoření matice A a b pro nerovnostní omezení
A = tril(ones(n,n));
b = SoC_max * ones(n,1);

A_min = -tril(ones(n,n));
b_min = zeros(n,1);

for i = 1:n
    b(i) = b(i) + sum(Pt(1:i)) - (SoC_max*(1+SoC_ef_upper_lim/100) - SoC_start); % max SoC kum
    b_min(i) = -sum(Pt(1:i)) + (SoC_max*SoC_ef_coef_lower_lim - SoC_start); % min SoC kum
end

% Sjednocení omezení do jednoho nerovnostního omezení
A_combined = [A; A_min];
b_combined = [b; b_min];
```

Obrázek 5.45 - Optimalizace omezení SoC ef – úprava nerovnostních omezení pro linprog



Graf 5.32 - Optimalizace omezení SoC ef – průběh SoC s omezenou efektivní kapacitou

6. SIMULINK MODEL

Pro vytvoření modelu, který bude zahrnovat ztráty, a bude zde možné modelovat a zkoumat další provozní faktory, jsem se přesunul do Matlab prostředí Simulink, které umožňuje modelování složitějších systému. S využitím výše zmíněné knihovny SimScape a jejích bloků, jsem sestavil model, který umožňuje simulaci řízení baterie, na základě vstupních parametrů a snaze maximalizovat zisk, s možností zahrnout zde nabíjecí, vybíjecí a další provozní ztráty.

Simulink model, na rozdíl od Matlab Skriptu, umožňuje zkoumání více parametrů, které by bylo potřeba řešit při skutečné provozu, takového systému. Umožňuje nám také zařazení konkrétní baterie, jejíž model by nám mohl poskytnou výrobce a celkově zde můžeme mluvit o jednodušší úpravě a modifikaci modelu pro další použití. Další velkou výhodou Simulinku je jeho grafické zobrazení a názornost, díky které je model více pochopitelný pro většinu lidí, a i proto by mnou vytvořený model mohl sloužit jako výuková pomůcka, pro vysvětlení tohoto problému a všech jeho aspektů.

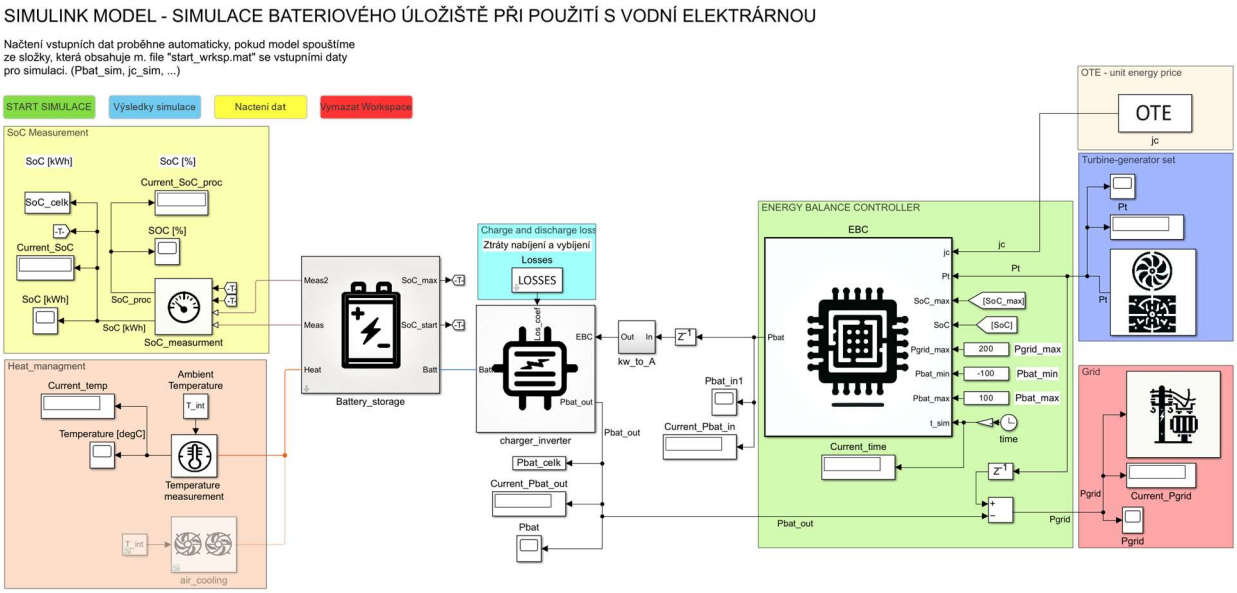
I v tomto modelu, jsem využil řídící algoritmus, který je založený na lineárním programování a využívá Matlab funkci linprog. Tento řídící algoritmus je v modelu aplikován pomocí bloku *Matlab Function*, ve kterém je vytvořen podobně jako moje m-funkce pro optimalizaci na základě reálné dostupnosti dat ze které vychází. Rozdílem je tady vstup proměnných, který probíhá Simulink signály dle aktuálního kroku při simulaci, tak jak by tomu bylo při reálném řízení. Tento řídící algoritmus tvoří jádro tohoto modelu.

6.1. Popis modelu a jeho částí

Celý model je rozdělený do jednotlivých subsystémů, které jsou sestaveny převážně z bloků SimScape knihovny.

Napravo máme vstupní data, jednotkovou cenu a výkon turbíny (*jc* a *Pt*) a pod nimi výstup do sítě *Pgrid*. V zelené oblasti se nachází řídící algoritmus „Energy balance controller“ (EBC), včetně veškerých jeho vstupů nutných pro optimalizaci. Vlevo vedle toho bloku se nachází subsystém, který představuje nabíječku baterie a střídač. Vedle tohoto bloku směrem vlevo je subsystém simulující bateriové úložiště. Vlevo nahoře je umístěn blok pro výpočet kapacity baterie *SoC*. Pod tím jsou umístěny bloky, které souvisí s teplem vzniklým při provozu baterie.

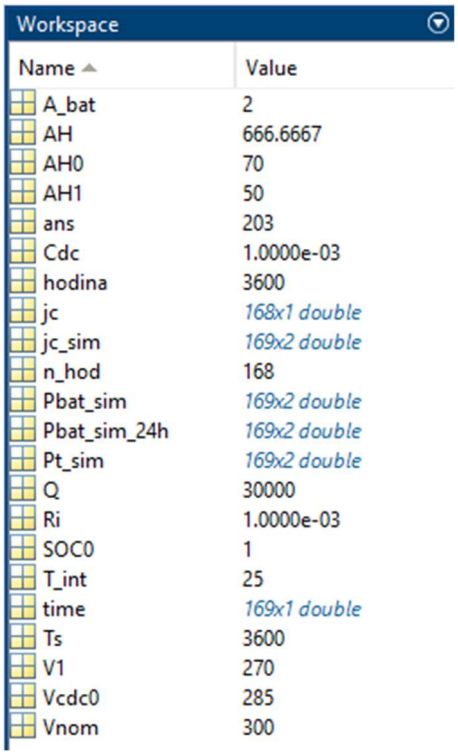
V této sekci od shora nejprve narazíme na subsystém pro měření teploty baterie a pod ním je subsystém pro primitivní chlazení vzduchem, který lze zapnout.



Obrázek 6.1 - Simulink model

6.1.1. Vstupní data

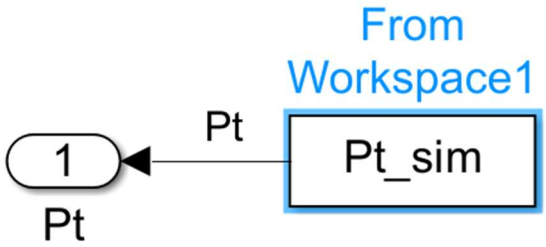
Vstupní data jsou v tomto modelu inicializovány pomocí Simulink bloků, které pracují s daty v Matlab workspace. Proto je potřeba před začátkem simulace mít veškeré vstupní data načtena v Matlab workspace, při otevření modelu to proběhne automaticky a do workspace se načtou vstupní data, které jsem používal pro simulaci. Pro změnu je lze odstranit tlačítkem v modelu a načíst svoje. Pro znovu načtení mých dat pro spuštění simulace je v modelu taktéž udělané tlačítko, které tyto data načte pokud se ovšem nacházejí ve složce, ve které pracujeme.



Obrázek 6.2 - Simulink model – Vstupní hodnoty nutné pro simulaci

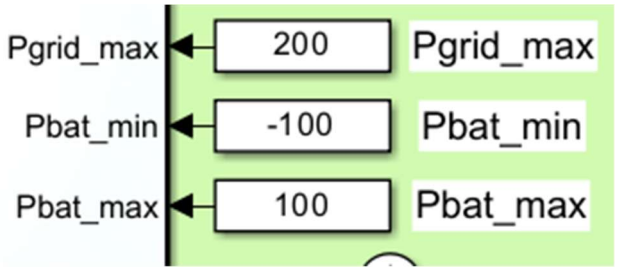
Obrázek 6.3 - Simulink model – Tlačítka pro načtení dat a vyčištění workspace

Některé vstupní parametry, jsou přímo v jednotlivých blocích, kde se zadávají vstupní hodnoty, jiné do bloků vstupují signály pomocí Simulink bloků *Constant* nebo *From Workspace.*



Obrázek 6.4 - Simulink model – Blok From Workspace

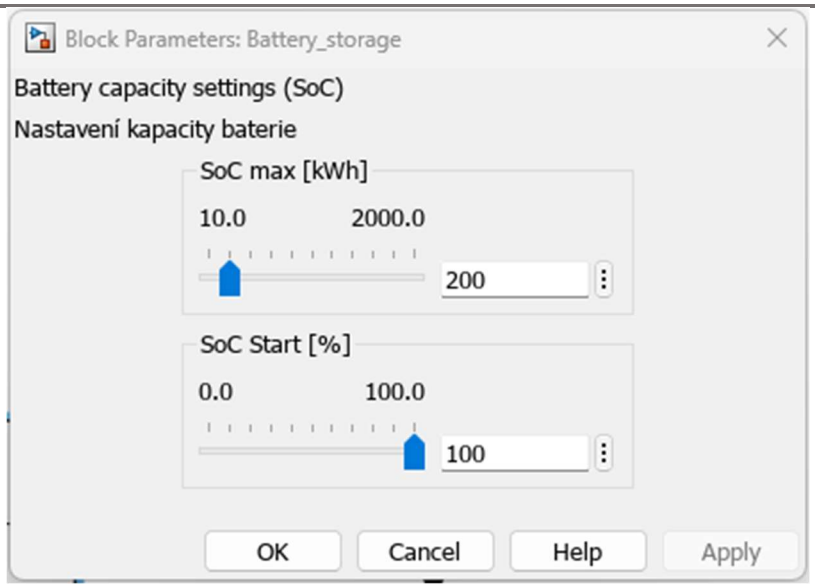
Další vstupní parametry pro simulaci může uživatel nastavit přímo v modelu, přepsáním stávajících hodnot. Například omezení výkonu do sítě a maximální výkon střídače/nabíječky.



Obrázek 6.5 - Simulink model – Nastavení parametrů přímo v modelu

Kapacitu baterie a ztráty je možné zadávat přímo na jejich blocích. Dvojklikem na příslušný blok se otevře okno, kde je možné zadat jednotlivé parametry pro simulaci. Tyto hodnoty se následně automaticky propíšou do modelu.

U baterie můžeme nastavovat její maximální kapacitu a s kolika procentním nabitím bude na začátku simulace.

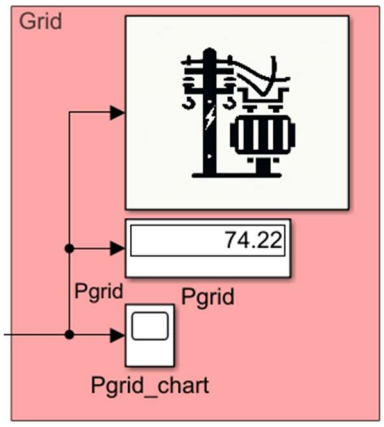


Obrázek 6.6 - Simulink model – Nastavení parametrů baterie

6.1.2. Grid

Tato část představuje výstup elektřiny ze systému do sítě. V modelu provádí zobrazení a ukládání výsledků. Jsou zde použity bloky:

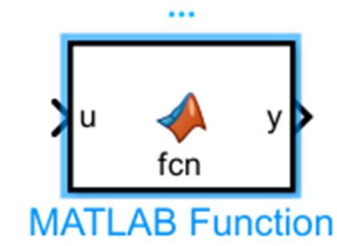
- *Display* – umožňuje zobrazení aktuální hodnoty *Pgrid* při simulaci
- *Scope* – vykresluje graf průběhu *Pgrid*
- *To Workspace* – ukládá výsledná data simulace do Matlab workspace (tento blok najdeme pod grafickou maskou)



Obrázek 6.7 - Simulink model – Blok představující výstup elektřiny ze systému

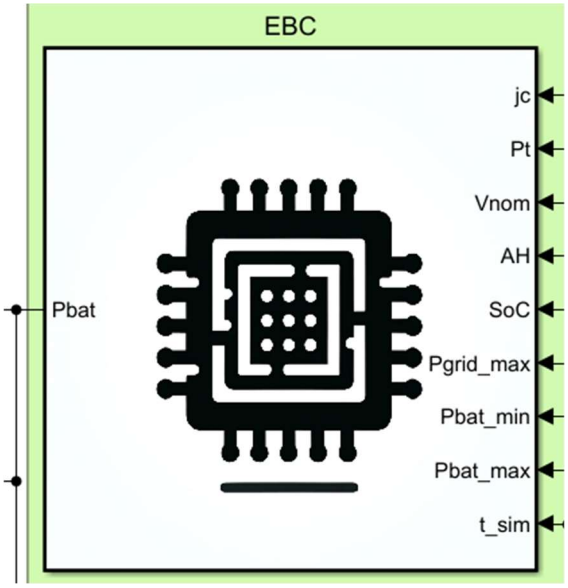
6.1.3. Řídící algoritmus

Řídící algoritmus vychází z mého Matlab programu pro optimalizaci prodeje elektřiny do sítě na základě reálné dostupnosti dat, který jsem představoval výše. Pro zařazení algoritmu do Simulink modelu jsem využil blok *MATLAB Function*.



Obrázek 6.8 - Simulink model – Blok Matlab Function

V mém modelu je tento blok schovaný pod grafickou maskou.



Obrázek 6.9 - Simulink model – Blok řídící funkce

Simulink blok *MATLAB Function,* funguje velmi podobně jako Matlab m-funkce, s tím rozdílem že vstupní hodnoty nejsou zadávány při spouštění přes command window ale jsou definovány jednotlivými vstupy Simulink signálů od proměnných do tohoto bloku. Forma kódu a práce s ním je téměř totožná s Matlab m-funkcí.

Můj řídící algoritmus použitý pro Simulink model vychází s předtím vytvořeného skriptu „řídící“ optimalizace. Tento kód bylo nutno zlehka upravit, tak aby mohl být v tomto modelu využit. Základní struktura je zde zachována, program nejprve definuje jednotlivé proměnné na základě vstupů, následně vypočítá vstupy pro *linprog* (horní a dolní omezení a nerovnicová omezení). Následně pomocí linprog určí optimální výkon do sítě Pgrid, ze kterého je dopočítán

Pbat, což je výstup této funkce pro simulaci modelu. Tento výpočet provádí tato funkce v každém kroku, který se pro mnou použitá data rovná jedné hodině.

Řídící algoritmus:

```
function Pbat = fcn(jc,Pt,Vnom, AH, SoC, Pgrid_max, Pbat_min, Pbat_max, t_sim)

%optimalizace probíhá na 2 dny, aby nedošlo k vybíjení na 0 na konci
%prvního dne, v dalším cyklu je den č. 2 přepsán novými daty a jsou vždy
%výpočet probíhá každou hodinu pro aktuální provozní data
%cenu známe na 24h dopředu


% Deklarace linprog jako extrinsic funkce
coder.extrinsic('linprog'); % linprog není podporovana C kodem

if t_sim == 0
    Pbat = 0;
else

SoC_max = AH*Vnom/1000; % maximální kapacita baterie [kWh]


den = ceil(t_sim/24); % aktuální den pro který provádíme simulaci
hodina = t_sim-((den-1)*24);    %relativní hodina pro kterou probíhá výpočet pro
aktuální den


Pt_start = Pt; % aktuální výkon turbíny pro daný krok
jc_24h = jc(den:(den+23)); % cena pro výpočtový den [kc/kWh]

jc_fikt_den = [jc_24h;jc_24h];       % cenová řada pro výpočet 2x daný den - aby
nedošlo k vyprázdnění baterie na 0
                                % uvažujeme pro výpočet řadu 48h

jc_step = jc_fikt_den(hodina:48,1);       % pokročení ve výpočtu o další hodinu-
řada se zkrátí


Pt_den = ones(48, 1)*Pt_start;  % Předpoklad že výkon turbíny bude stejný, jako je
v akutálním kroku
Pt_step = Pt_den(hodina:48,1);       % pokročení ve výpočtu o další hodinu-řada se
zkrátí


% počáteční stav nabití pro tento krok
SoC_start = SoC;

% Délka datových řad pro daný krok
n_step = length(Pt_step);

% Omezení prodeje za záporné ceny
% při záporné ceně dojde k vypnutí turbíny
for i = 1:n_step
    if jc_step(i)<0
        Pt_step(i)=0;
```

```
        end
    end


    % Horní a dolní hranice pro Pgrid(t)
    lb = zeros(n_step,1); % dolní hranice (minimum energie prodané do sítě) == 0
    % --> zajištění že nebudeme ze sítě baterii nabíjet to by bylo Pgrid(t)<0 což
    % tato podmínka omezí == Pgrid(t)>=0

    ub = Pgrid_max * ones(n_step,1); % horní hranice (omezení Pgrid_max) ==
    Pgrid(t)<=Pgrid_max

    % Nevybíjíme z baterie, pokud je cena záporná
    ub(jc_step < 0) = 0;

    % Aktualizace lb a ub pro Pgrid s ohledem na max rychlost nabíjení/vybíjení
    for i = 1:n_step
        % Nabíjecí omezení, pokud je výroba větší než nabíjecí limit
        if Pt_step(i) > Pbat_min
            lb(i) = max(lb(i), Pt_step(i) + Pbat_min);
        end
        % Vybíjecí omezení, pokud je výroba menší než vybíjecí limit a stále
    respektujeme Pgrid_max
        ub(i) = min(ub(i), max(0, Pt_step(i) + Pbat_max));

        % Respektování stávajícího omezení Pgrid_max a záporné ceny energie
        ub(i) = min(ub(i), Pgrid_max);
        if jc_step(i) < 0
            ub(i) = 0;
        end
    end


    % Vytvoření matice A a b pro nerovnostní omezení
    A = tril(ones(n_step,n_step));
    b = SoC_max * ones(n_step,1);

    A_min = -tril(ones(n_step,n_step));
    b_min = zeros(n_step,1);

    for i = 1:n_step
        b(i) = b(i) + sum(Pt_step(1:i)) - (SoC_max - SoC_start); %
        b_min(i) = -sum(Pt_step(1:i)) + (SoC_max - SoC_start);
    end

    % Sjednocení omezení do jednoho nerovnostního omezení
    A_combined = [A; A_min];
    b_combined = [b; b_min];

    % VYPOCET TEORETICKEHO VYKONU DO SITE
    % volání linprog
    f = -jc_step; % mínus protože linprog minimalizuje a my chceme maximalizovat

    % Inicializace výstupu linprog
    Pgrid = zeros(size(jc_step));

    % Volání linprog
    Pgrid = linprog(f, A_combined, b_combined, [], [], lb, ub);
```

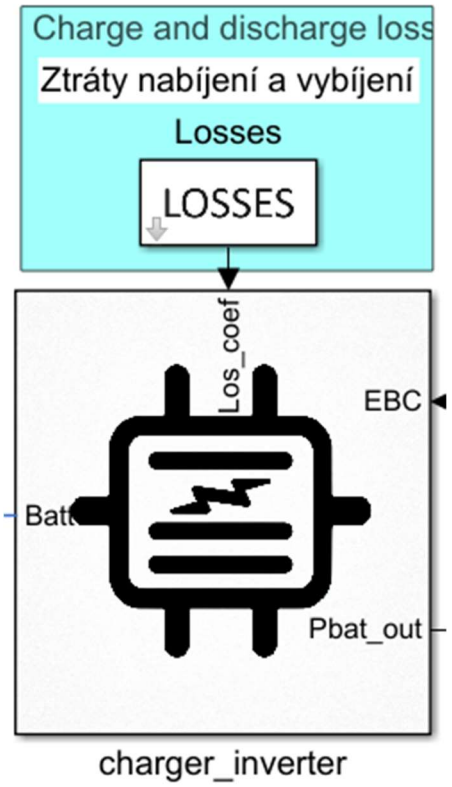
```
% prikon do baterii (+ nabíjíme; - vybíjíme)
Pbat=Pt-Pgrid(1);

end
```

6.1.4. Nabíječka / Střídač + ztráty nabíjení a vybíjení baterie

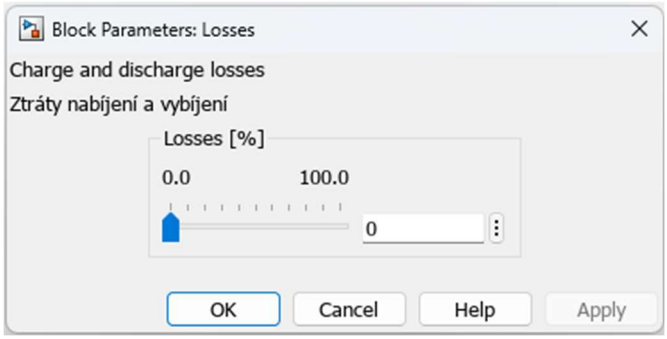
Pod grafickou maskou tohoto bloku najdeme několik SimScape a Simulink bloků, které simulují nabíjení a vybíjení baterie. Vstupy do tohoto systému jsou, hodnoty Pbat z řídicí jednotky a hodnota ztrát.

Velikost ztráty při nabíjení a vybíjení baterie si může uživatel nastavit pomocí bloku „LOSSES“ nad blokem, který představuje nabíječku/střídač. Stačí rozkliknout grafickou masku tohoto bloku a zobrazí se okno pro nastavení ztrát nabíjení a vybíjení baterie, tato informace je dále předávána do bloku „charger_inverter“ (nabíječky/střídače).

Výstupem z bloku, je signál pro chování baterie a hodnota Pbat_out, která udává kolik energie jde skutečně do nebo z baterie po odečtení ztrát. Hlavním blok, který simuluje nabíjení a vybíjení baterie se nachází uvnitř systému a je jím SimScape blok *Controlled Current Source* (zdroj proudu, který lze ovládat vstupním signálem).



Obrázek 6.11 - Simulink model – Blok nabíječka a střídač

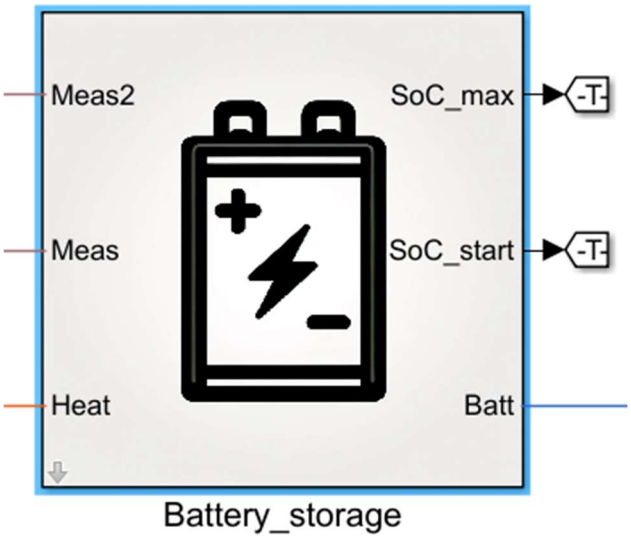


Obrázek 6.10 - Simulink model – nastavení ztrát

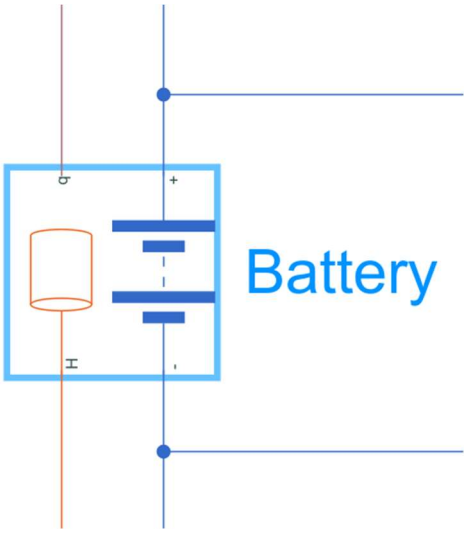
6.1.5. Bateriové úložiště

Jedná se o modifikovatelný blok, který představuje bateriové úložiště. Pro moji simulaci byl použit základní blok baterie ze SimScape knihovny: *Battery.* Tento blok simuluje baterii, a dle jeho nastavení můžeme volit, jestli se bude jednat o ideální baterie (nezahrnující ztráty a stárnutí), nebo zda se bude jednat o více reálnou baterii, která bude zahrnovat ztráty vlivem tepla, stárnutí baterie, dynamiku nabíjení atd.

Pro moji simulaci jsem pracoval s téměř ideální baterií, pro více reálné nastavení by bylo třeba vybrat konkrétní typ baterie, který by se plánoval k elektrárně připojit a s výrobcem prokonzultovat její parametry a podle nich nastavit baterii použitou v modelu. Jak jsem již zmínil, tento SimScape blok umí simulovat i teplotu baterie případně snížení účinnosti na základě teploty. V mém modelu jsem tepelný systém také vytvořil, ale je tu několik zásadních omezení, které by bylo potřeba vyřešit pro přesnější simulaci teplotního vlivu, více o tepelné části v kapitole níže.



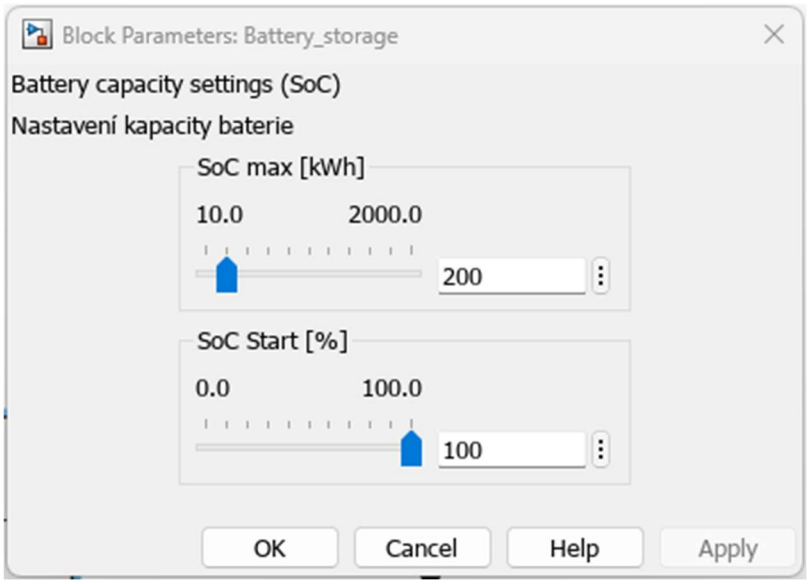
Obrázek 6.13 - Simulink model – Systém bateriového úložiště



Obrázek 6.12 - Simulink model – SimScape blok Battery

Díky vysoké modularitě Simulinku a mého modelu, je možné na místo baterie, kterou jsem použil já, zapojit jakoukoliv jinou baterii, kterou bude chtít uživatel simulovat, případně tuto baterii nastavit tak aby odpovídal reálné baterii. Bylo by zde možné například zapojit konkrétní model baterie od výrobce, kterou plánuje investor pořídit, a otestovat tak zda bude vyhovovat.

Pro nastavení kapacity baterie stačí dvojklikem kliknout na grafickou masku toho sytému a dojde k otevření dialogového okna, kde může uživatel nastavit kapacitu baterie a procentuální kapacitu na začátku simulace.



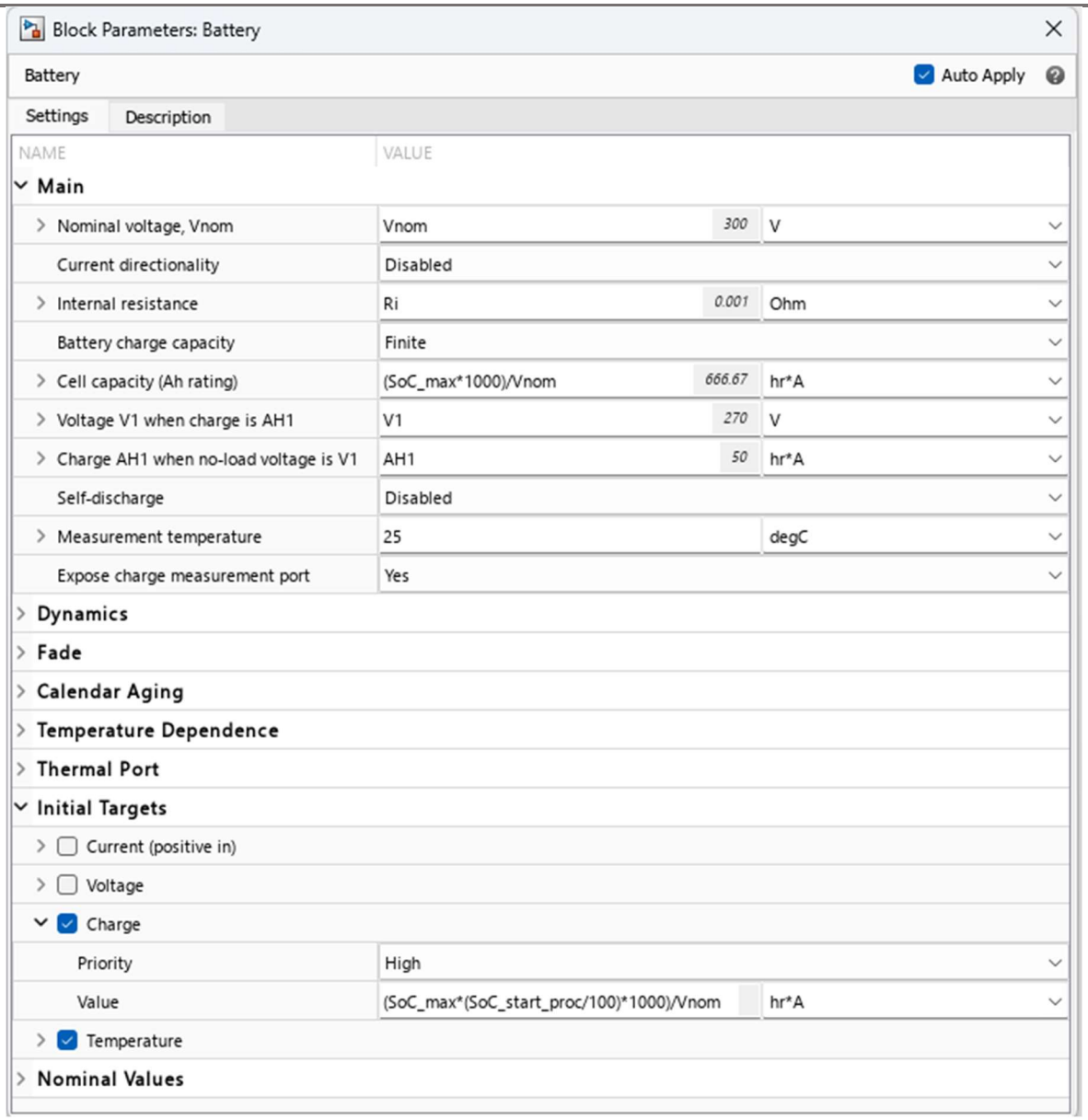
Obrázek 6.14 - Nastavení parametrů baterie

Pro další detailní nastavení je potřeba jít pod masku tohoto systému (pomocí šipky v levém dolním rohu, nebo klávesovou zkratkou ctrl+u), zde otevřít SimScape blok *Battery*, a následně lze nastavovat další detailní parametry.

Pokud by chtěl uživatel nastavit kapacitu baterie mimo rozsah slideru, může tak učinit právě přímo v SimScape bloku *Battery*, stačí přepsat rovnice pro výpočet kapacitu AH a pro AH initial, tak aby tato hodnota odpovídala požadované kapacitě

AH [Ah] = SoC [Wh] / Vnom [V]

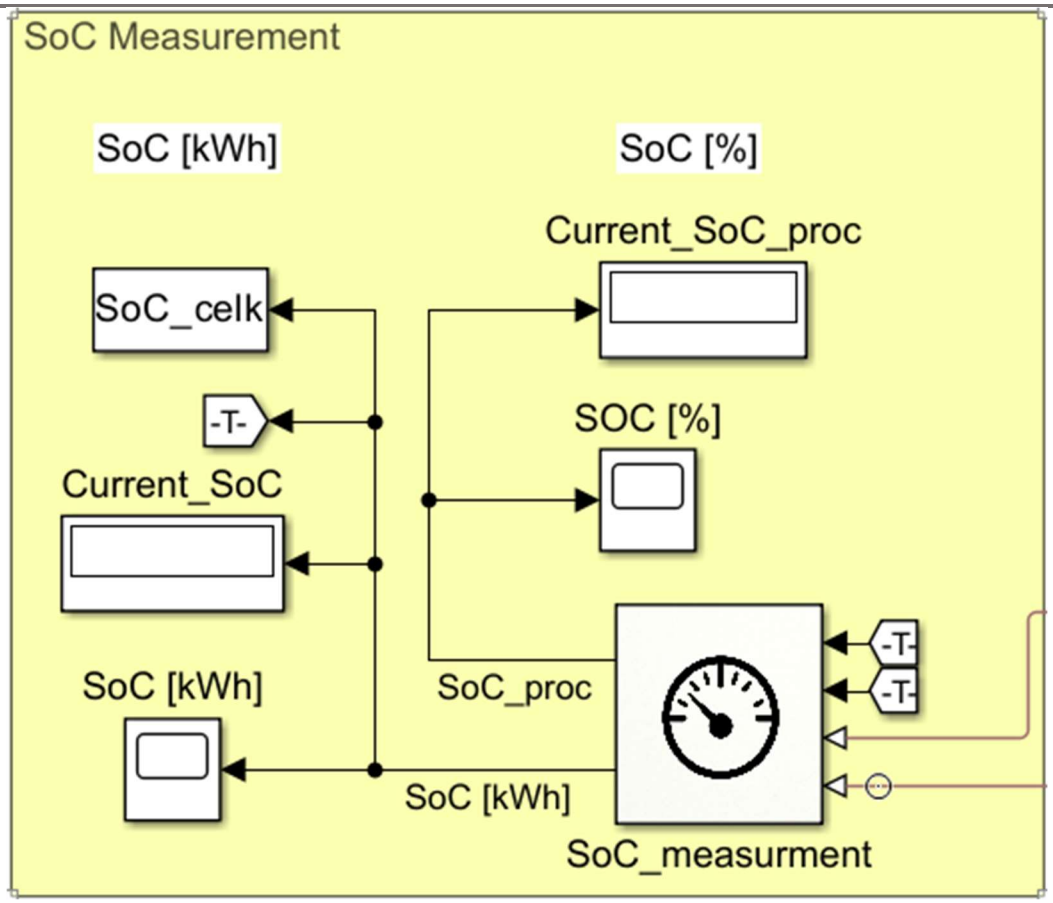
Dále zde lze nastavovat další funkce baterie jako jsou ztráty, stárnutí, chování v závislosti na teplotě, dynamika nabíjení atd.



Obrázek 6.15 - Simulink model – Nastavení bloku Battery

6.1.6. Měření kapacity baterie (SoC)

V mém modelu se nachází prvek, který provádí měření stavu baterie (SoC), vyjádřený jak absolutní hodnotou v kWh, tak relativní hodnotou v procentech. Tato část obsahuje prvky *Display,* na kterých se zobrazují aktuální hodnoty při simulaci, a *Scope* pro vykreslení průběhů hodnot při simulaci. Díky tomu může uživatel sledovat hodnoty při simulaci anebo ihned po dokončení, přímo v modelu, bez nutnosti otevírat další aplikaci. Tato část obsahuje také blok *To Workspace*, kterým jsou ukládány výsledná data do Matlab workspace.

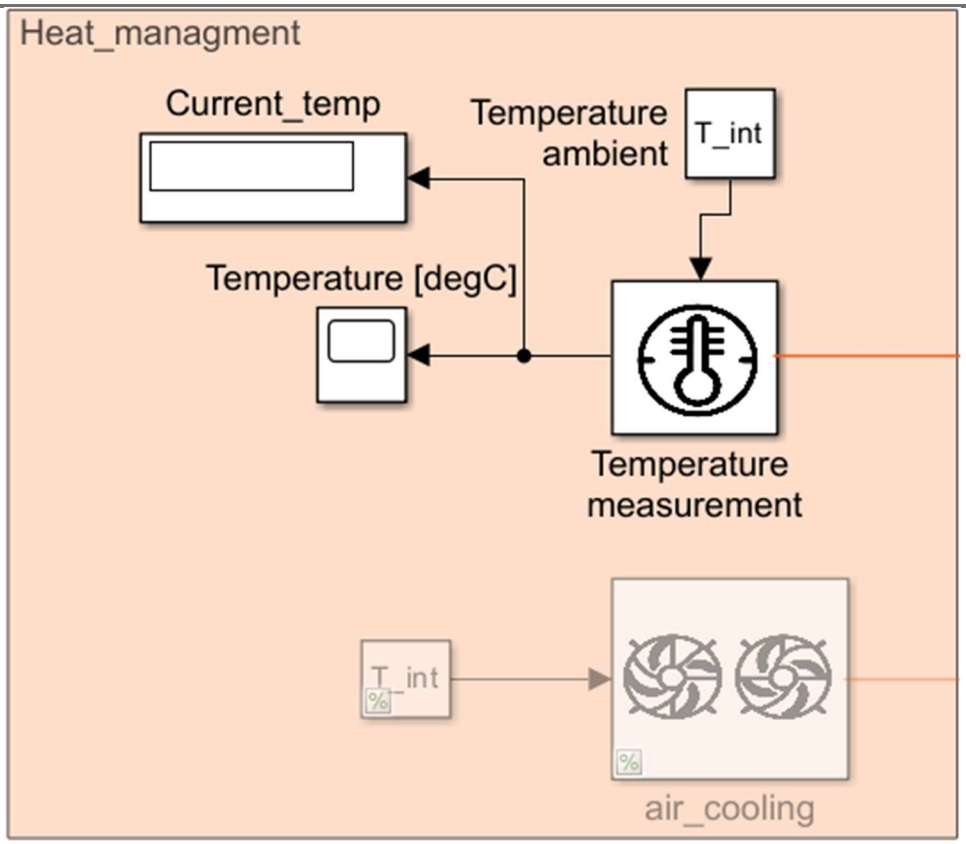


Obrázek 6.16 - Simulink model – Část měřící SoC a zobrazující výsledky

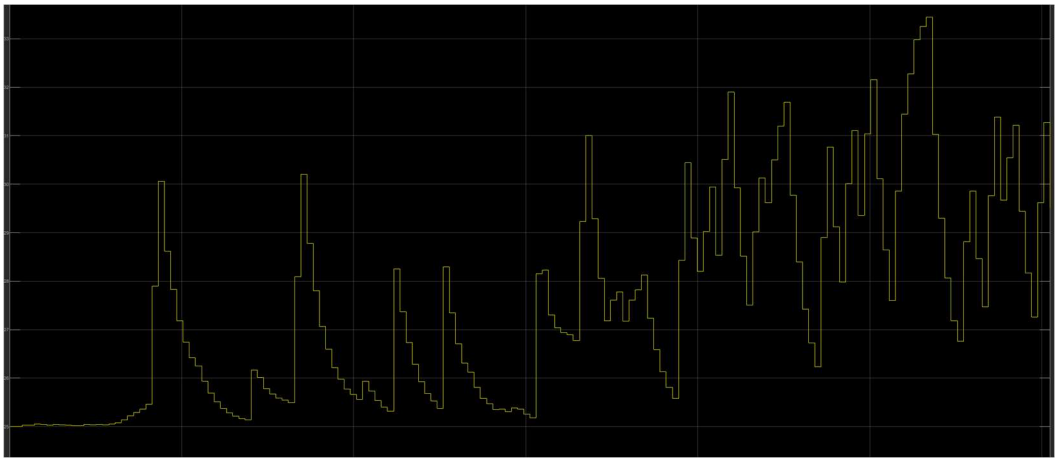
6.1.7. Teplotní systém

Model také disponuje teplotním systémem, který umí měřit a simulovat ohřívání baterie. Tuto část jsem zde vytvořil pro možnost zabývat se i teplotou baterie, pro moji simulaci tento faktor nehrál žádnou roli. Pro věrohodnou simulaci teploty baterie by bylo potřeba upravit její teplotní vlastnosti dle konkrétního typu baterie a vložit řadu vnitřní teploty ve strojovně (nebo v místnosti kde budou baterii uloženy). Tato proměnná hraje velkou roli v tom, jak moc se baterie ohřívá a jak je schopná se chladit „samovolně“. Pro moji simulaci byla uvažována konstantní teplota okolního vzduchu 25 °C, což by pravděpodobně nebylo reálné. Teplota by v reálném provozu určitě kolísala a v létě by byla pravděpodobně mnohem vyšší. Tato teplota by mohla odpovídat průměrné teplotě v zimních měsících.

Měření teploty, stejně jako měření SoC, obsahuje bloky *Display* a *Scope*, pro zobrazení výsledků.



Obrázek 6.17 - Simulink model – Teplotní systém

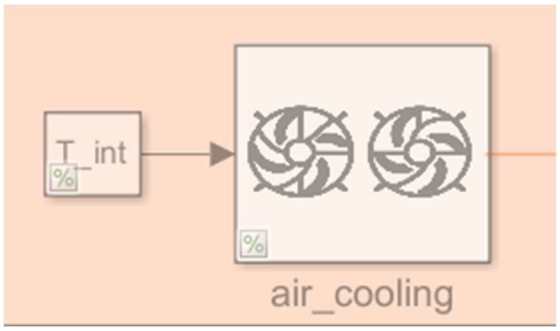


Obrázek 6.18 - Simulink model – Průběh teploty baterií při simulaci

V této části jsem vytvořil i velmi jednoduchou simulaci chlazení baterií vzduchem, ale opět zde platí stejná omezení. Bylo by potřeba přesně specifikovat teplotu vzduchu, zohlednit ohřívání vnitřního vzduchu od baterií, přesně znát povrch baterií, jejich teplotní vlastnosti, … Tato část je zde vytvořená spíše jako příprava pro pokračování specialisty v bateriovém oboru, který by se chtěl zabývat tím, jak se jejich baterie bude chovat při provozu s MVE, a modelovat její chlazení.

Pokud chce uživatel chlazení zapnout může, to provést tak že označí tyto dva bloky, a po kliknutím pravým tlačítkem vybere možnost *„Uncomment“* (v mé verzi Simulinku, je to šestá možnost o shora), případně po označení těchto bloků může použít klávesovou zkratku *ctrl+shift+x*, která má stejnou funkci.

Pokud by chtěl uživatel tento systém dále upravovat, může otevřít blok *air_cooling*, dvojklikem na grafickou masku, a následně ho upravit dle vlastních požadavků.

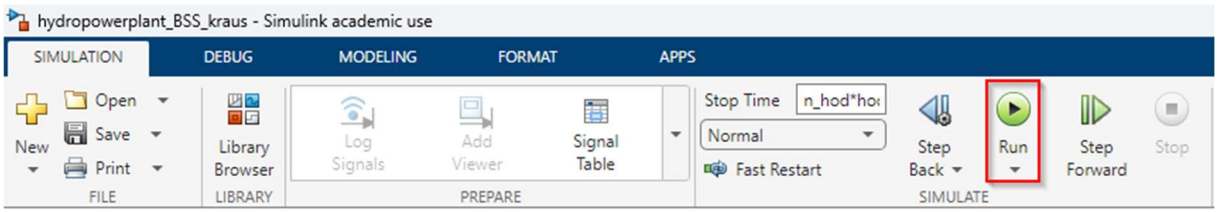


Obrázek 6.19 - Simulink model – Chlazení baterií vzduchem

6.1.8. Spuštění simulace

Pro spuštění je potřeba mít načtená potřebná data ve workspace (při normálním spuštění se tak stane automaticky) a nastavit jednotlivé parametry, viz. kapitola 6.1.1 vstupní data. Následně lze pustit simulaci, to můžeme udělat buď tlačítkem „START SIMULACE“ které je vytvořeno přímo v modelu, kliknutí na tlačítko „*Run*“ v horním pásu karet, nebo simulaci spustit přes příkazový řádkem pokynem *sim(nazev_simulovaneho_modelu.slx).* Následně proběhne simulace, pokud by byla simulace z nějakého důvodu přerušena (v tomto modelu pravděpodobně z důvodu, že *linprog* nedokáže najít řešení pro zvolené parametry a vstupní data), zobrazí se výstražná hláška.

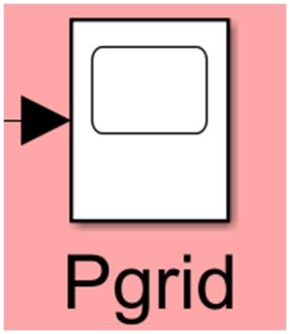
Obrázek 6.20 - Simulink model – Tlačítko "START SIMULACE" pro spuštění



Obrázek 6.21 - Simulink model – Tlačítko "Run" pro spuštění simulace

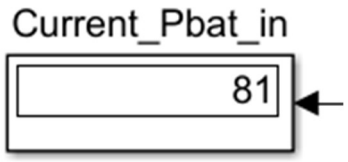
6.2. Výsledky simulace Simulink modelu

Základní výsledky v podobě grafů lze zobrazit přímo v modelu, a to i při ještě probíhající simulaci (výpočtu). Pro toto zobrazení jsou v modelu bloky *Scope,* ve kterých se po otevření zobrazí odpovídající graf.



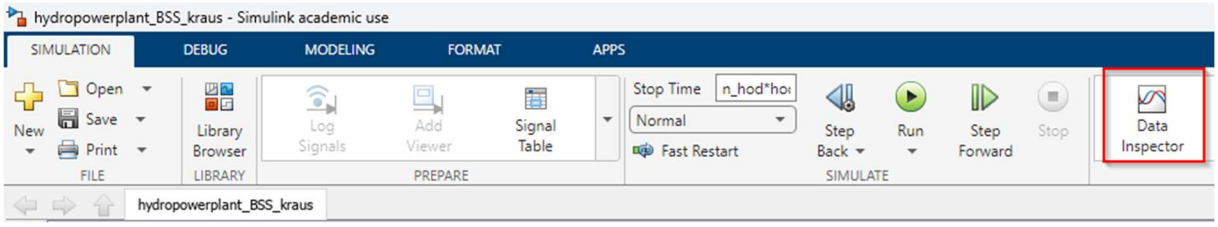
Obrázek 6.22 - Simulink model – Blok Scope

Průběžné výsledky při simulaci (aktuální hodnoty v daném kroku), se zobrazují v modelu v blocích *Display*, které zde jsou umístěny.



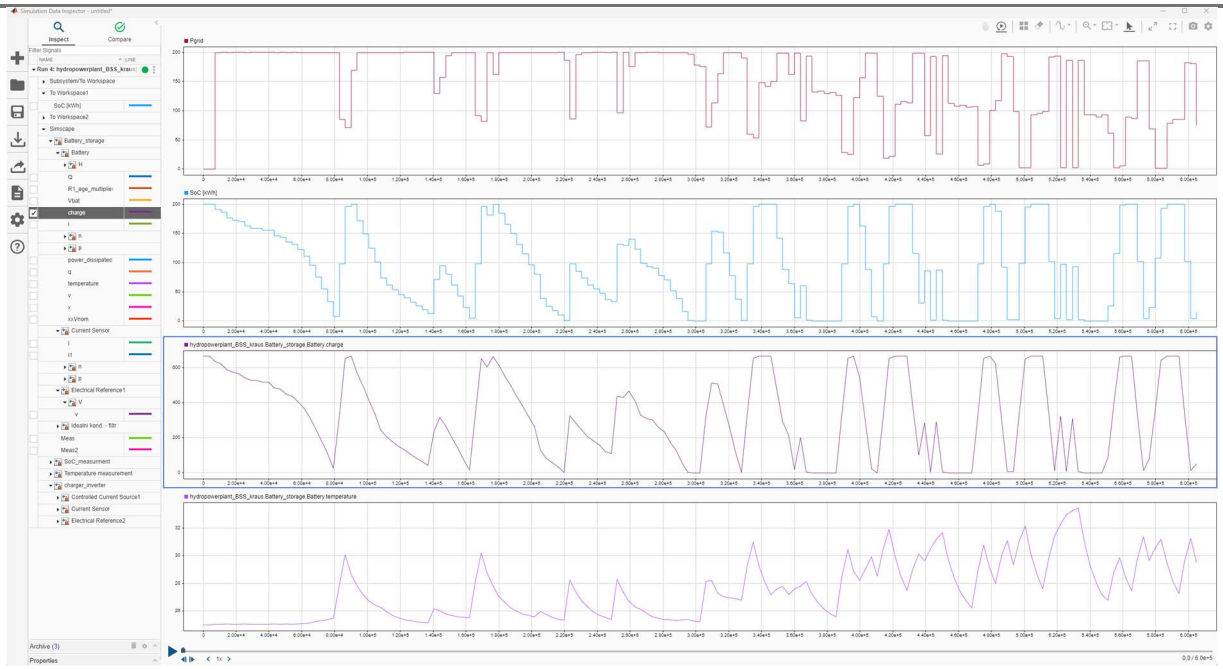
Obrázek 6.23 - Simulink model – Blok Display

Pro detailní zobrazení výsledků můžeme využít Simulink *Data Inspector*, který se nachází v horním pásu karet. Pomocí něho si lze nechat vykreslit průběhy i velmi detailních hodnoty pro veškeré SimScape bloky.



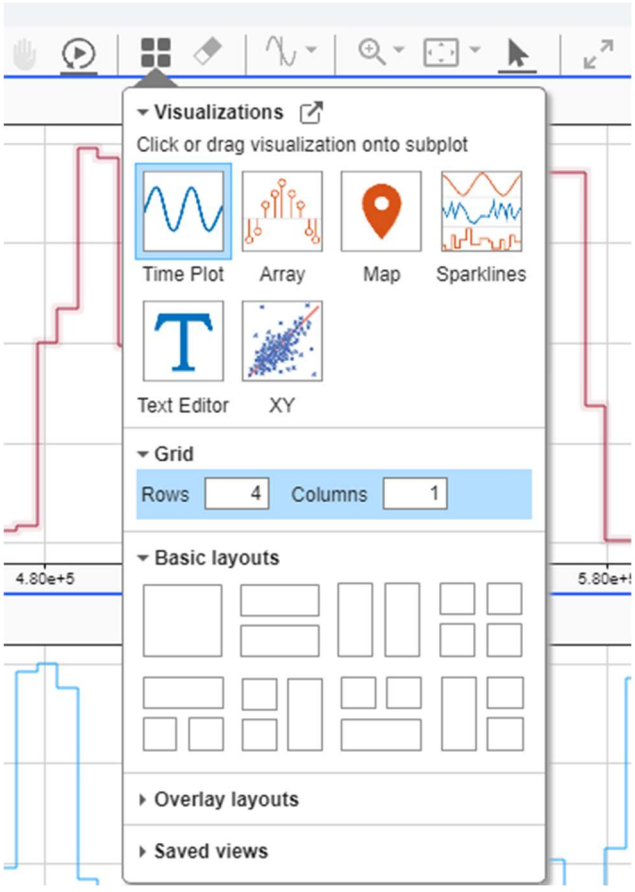
Obrázek 6.24 - Simulink model – Otevření Data Inspector

V pravé části si lze vybírat jaká data se vykreslí do příslušných polí a v dolní části nalezneme i data z předchozích simulací. Díky tomu lze porovnávat dvě simulace po úpravě určitých parametrů.



Obrázek 6.25 - Simulink model – Data Inspector

V záložce vlevo nahoře, si lze nastavit jaké rozložení bude mít plátno Data Inspectoru.



Obrázek 6.26 - Simulink model – Data Inspector, nastavení rozložení

Pro analýzu výsledných dat jsem vytvořil samostatný Matlab Script, „vysledky_simulink_model_kraus.mlx“, který zpracovává výsledná data ze simulace, viz. následující kapitola.

6.2.1. Detailní výsledky modelu

Podrobnější výsledky pro další analýzu, jsou zpracovány Matlab Scriptem, který jsem pro tento účel vytvořil: „vysledky_simulink_model_kraus.mlx“.

Před jeho spuštěním je potřeba provést úspěšnou simulaci Simulink modelu, tak aby se do workspace uložila výsledná data (výsledná data se uloží automaticky, je potřeba jen aby byla simulace úspěšně dokončena).

Tento skript vypočítá výsledný profit z prodeje elektřiny do sítě, porovná ho s výnosem bez baterie a vykreslí grafy do okna figures, s jednotlivými kartami, na kterých jsou vykresleny různé grafy, ve stejném rozložení jako jsou vykreslovány v předchozích Matlab Scriptech, které jsem představoval. V tomto skriptu je navíc přidána jedna karta, na které je vykreslen průběh zisku v jednotlivých časových úsecích.

Pro lepší přehlednost výsledných grafů, zde budu prezentovat výsledky z jednoho týdne simulace, v únoru 2022. Jedná se o stejný týden, na jehož základě jsem prezentoval grafické výsledky skriptu s „ideální“ a „řídicí“ optimalizací. S modelem ale není problém provádět simulaci celého roku nebo i delšího období. Níže prezentované výsledky byly vypočteny simulací s parametry:

SoC_max = 50 kWh (kapcita baterie)

Pbat_max = 50 kW

Pbat_min = - 50 kW

Pgrid_max = 200 kW

Ztráty pro nabíjení a vybíjení = 2 %

Je zde potřeba zmínit že jediné ztráty, které model tímto výpočtem zohlednil, byly ztráty 2 % při nabíjení a vybíjení baterie. Baterie jako taková byla nastavena na „ideální“, nezahrnující žádné ztráty nebo dynamiku nabíjení.

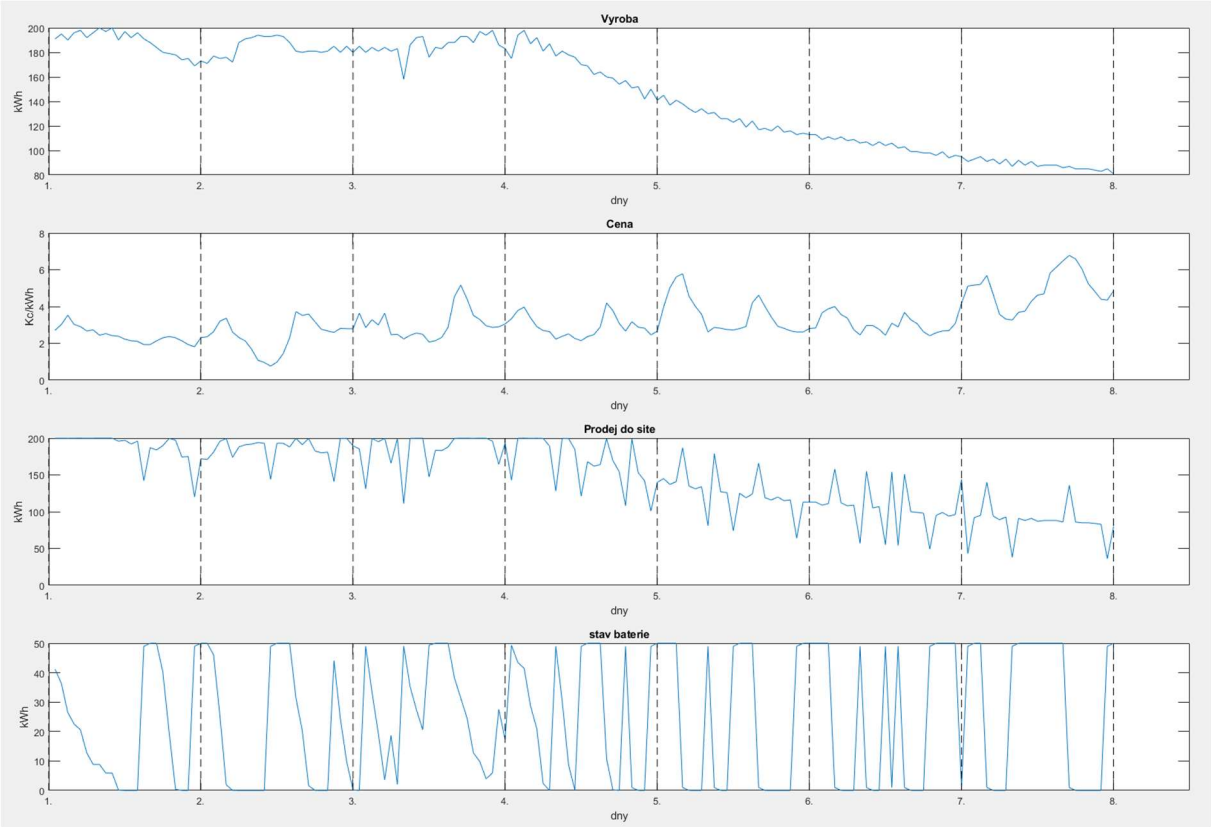
```
% Zisk z prodeje elektřiny
fprintf('Celkový zisk s baterií: %f Kč\n', ziskBat);
```

```
Celkový zisk s baterií: 77060.850406 Kč
```

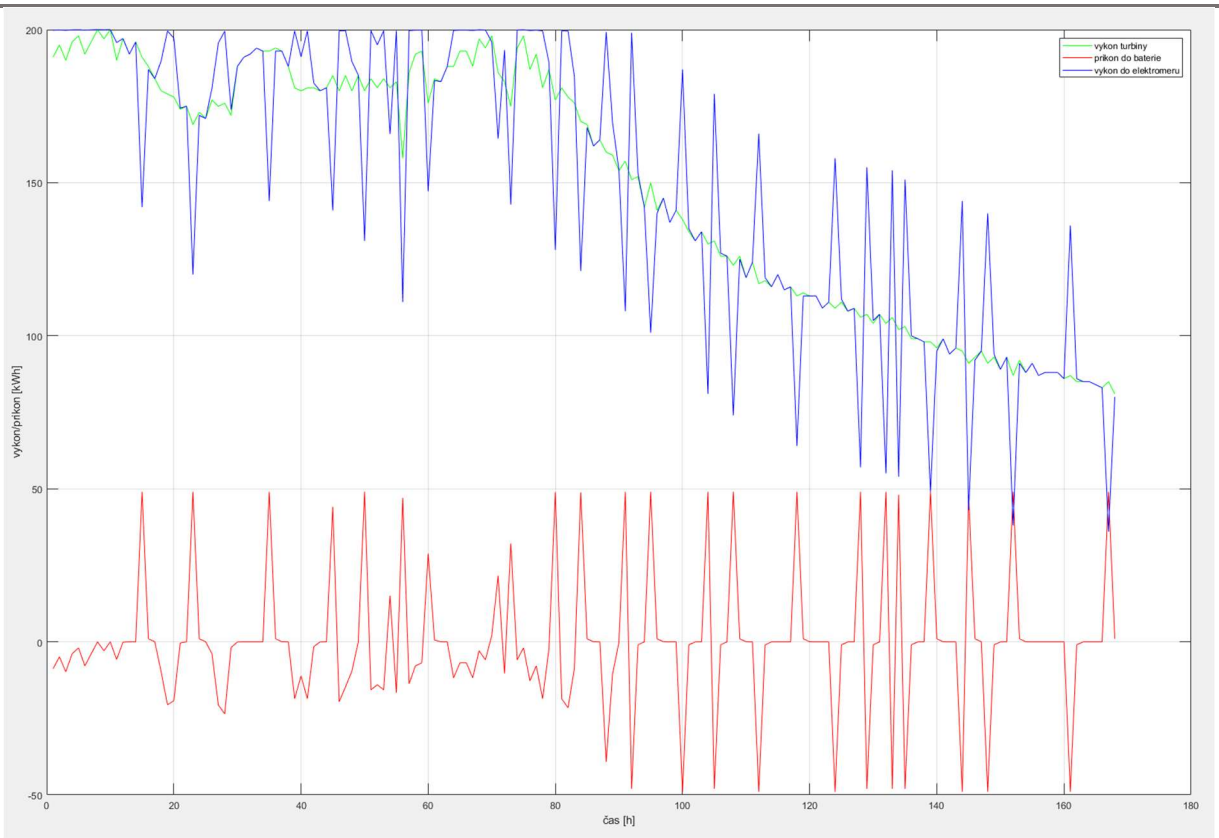
```
fprintf('Celkový teoretický zisk bez baterie: %f Kč\n', zisk_bez_bat);
```

```
Celkový teoretický zisk bez baterie: 75920.481420 Kč
```

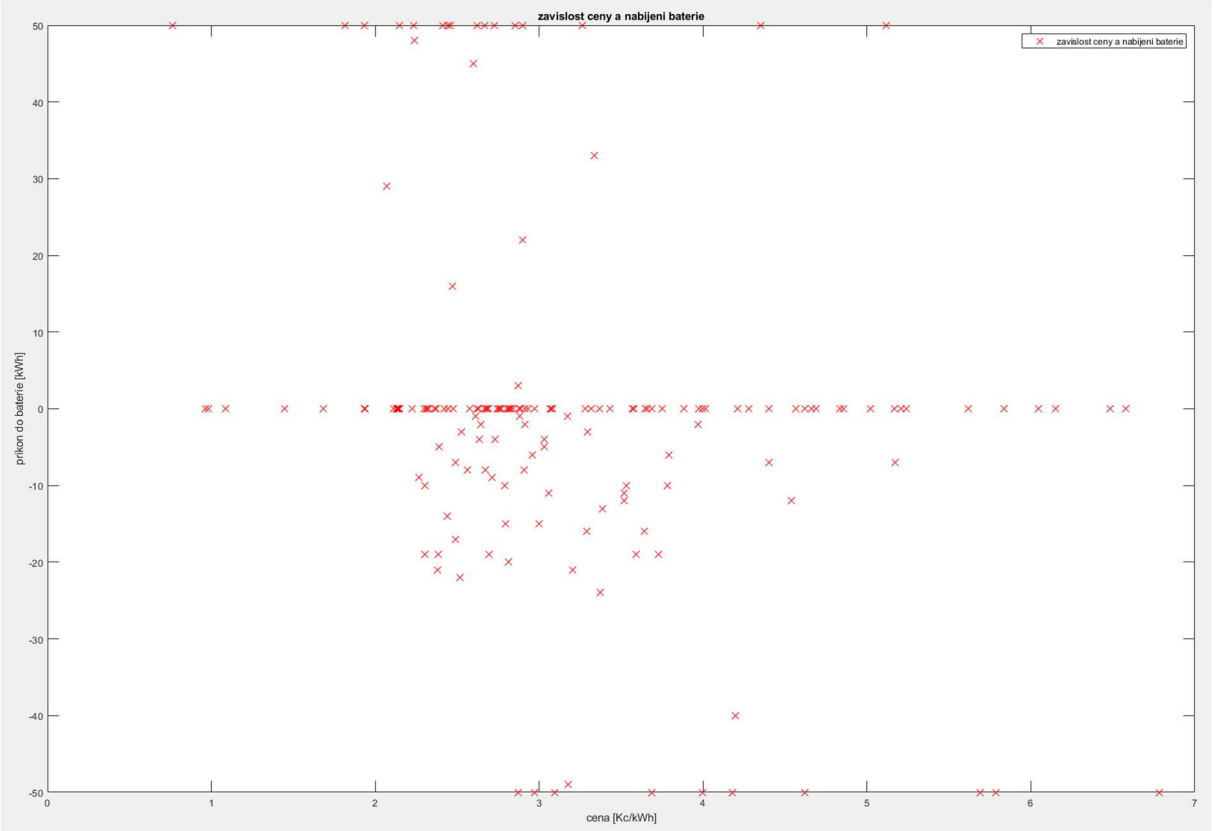
Obrázek 6.27 - Simulink model – zisk ve srovnání s provozem bez baterie



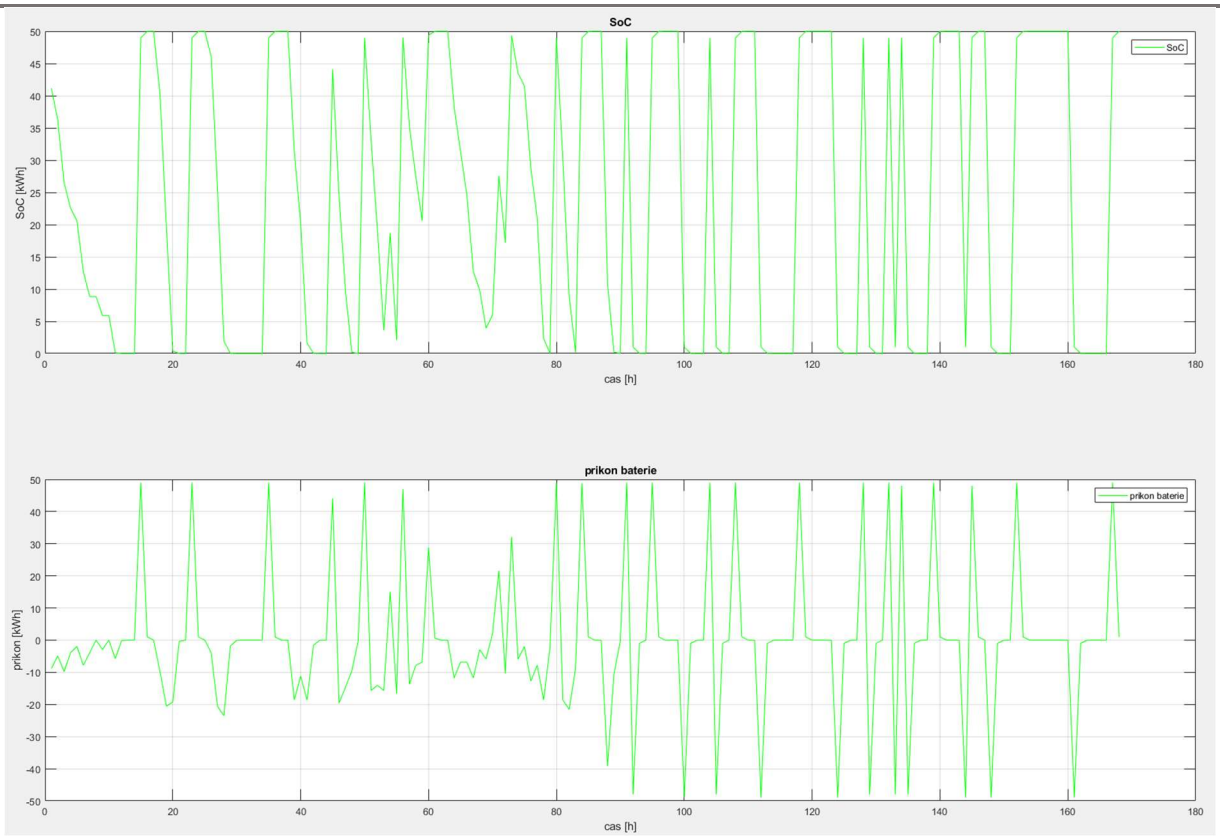
Obrázek 6.28 - Simulink model – Vykreslení průběhu základních parametrů



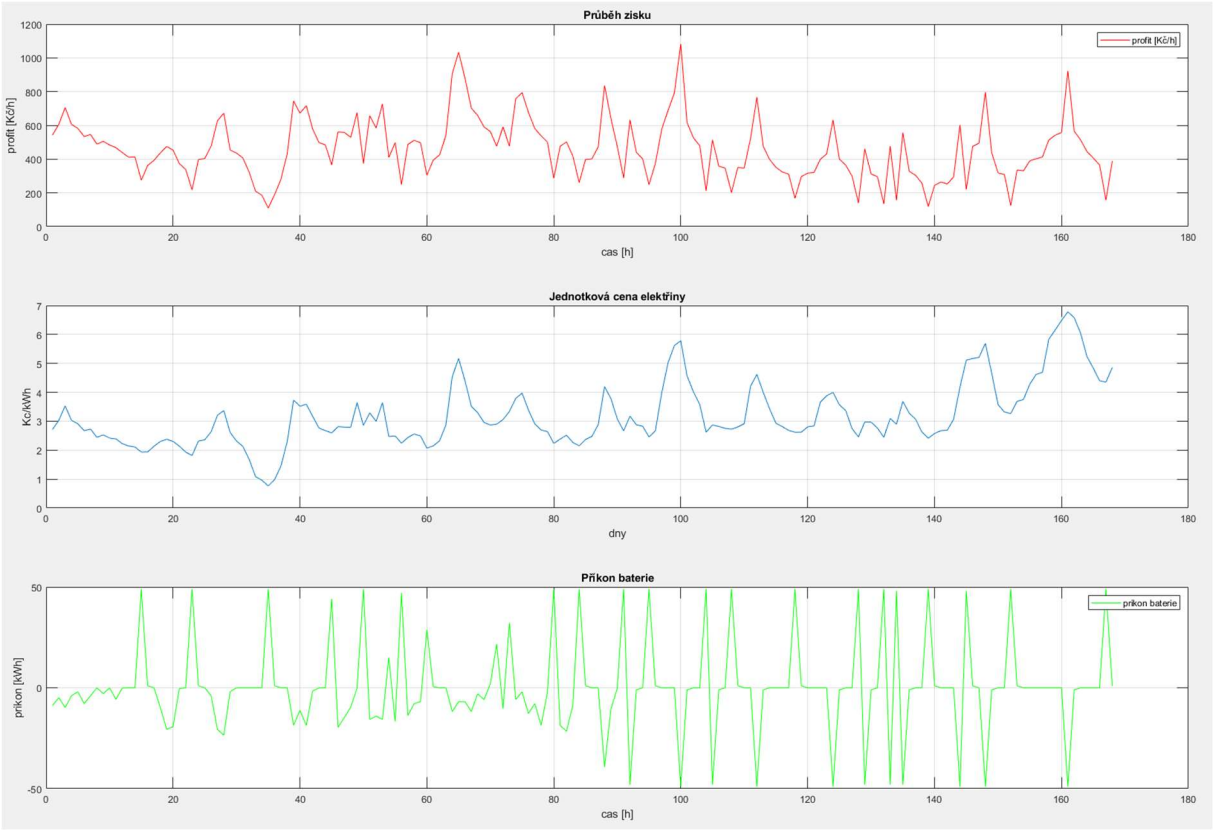
Obrázek 6.29 - Simulink model – Složený graf Pt, Pbat, Pgrid



Obrázek 6.30 - Simulink model – Závislost jc a Pbat



Obrázek 6.31 - Simulink model – grafy SoC a Pbat



Obrázek 6.32 - Simulink model – Vykreslení průběhu zisku

Z výše přiložených grafů lze vidět, že tato simulace má velmi podobný charakter jako výpočet „ideální“ a „řídicí“ optimalizace provedena Matlab skriptem a je zachována logika a návaznost jednotlivých výsledků.

6.2.2. Porovnání Simulink model vs optimalizace vytvořené v Matlab Skriptu

Pro porovnání jsem použil data pro téměř celý rok 2023 a provedl simulaci třech mých programů:

Ideální optimalizace

Řídící optimalizace

Simulink model

Simulink model se ztrátami nabíjení a vybíjení

Se stejnými parametry:

SoC_max = 50 kWh (kapcita baterie)

Pbat_max = 50 kW

Pbat_min = - 50 kW

Pgrid_max = 200 kW

Jedná se o parametry, které vyšly jako optimální vzhledem k návratnosti investice, při analýze, kterou jsem provedl.

Pro porovnání programů jsem vytvořil skript „*porovnani_SL_vs_skripts.mlx*“, který pracuje na základě výsledků z výpočtů jednotlivých programů. Tyto výsledky musí být ve správném formátu uloženy do zdrojové složky, před spuštěním výpočtu tohoto skriptu. Po spuštění, skript vypočítá a porovná rozdíly v zisku jednotlivých programů a vykreslí graf průběhu *Pgrid* (prodeje do sítě).

Při porovnání výsledků, se potvrdilo že řídící program, který jsem vytvořil v Simulink modelu, funguje stejně jako skript „řídící“ optimalizace, jelikož zisk (při simulaci bez ztrát) je totožný. Dále se zde ukazuje již výše zmíněný fakt, že při parametrech *SoC_max* = 50 kWh a *Pbat_max* = 50 kW, není velký rozdíl mezi simulací na základě reálné dostupnosti dat (řídicí simulace nebo Simulink model) a ideální optimalizaci, kde program zná data na celé období dopředu.

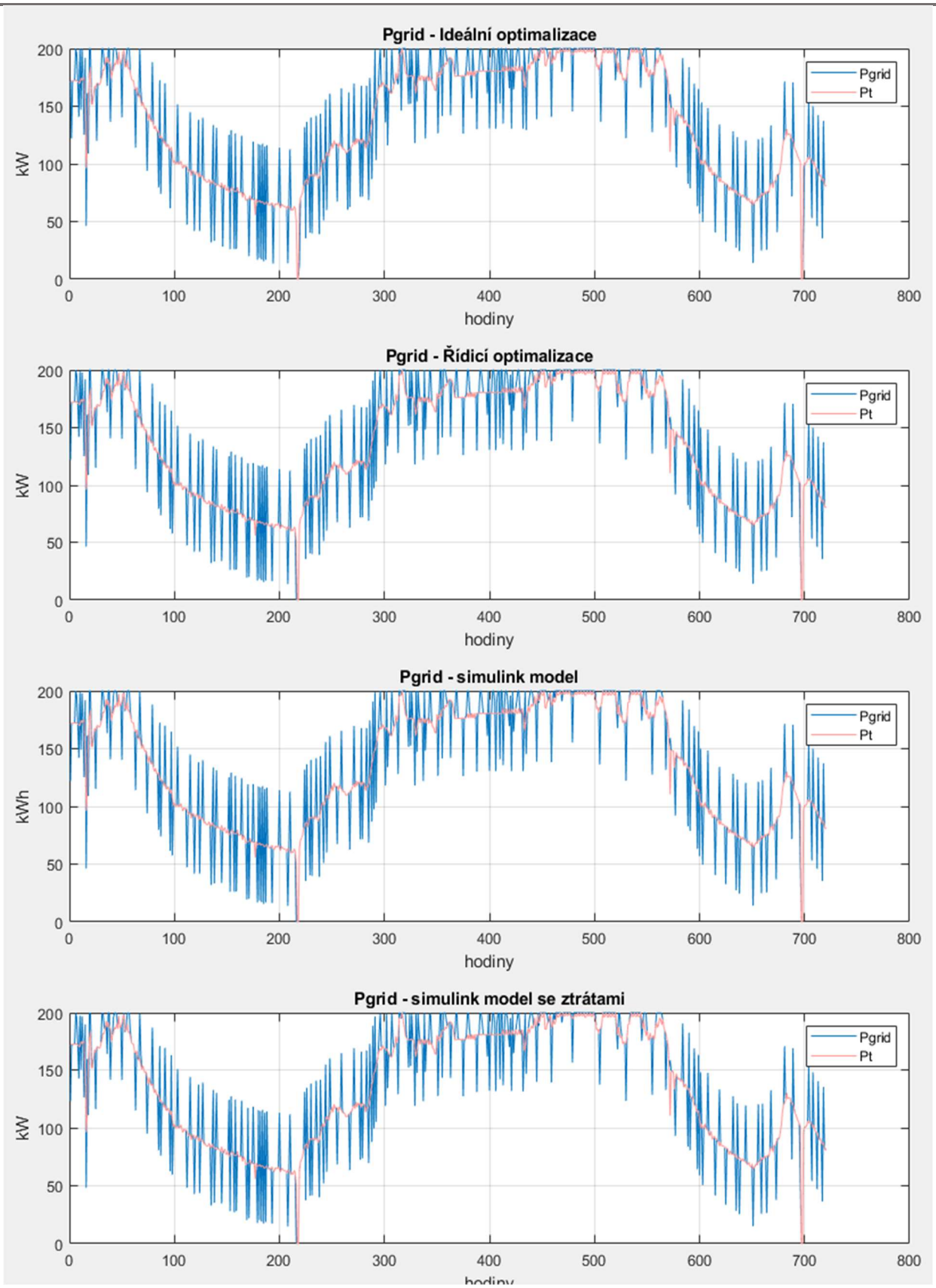
Toto porovnání také ukázalo že vliv přidané ztráty při nabíjení a vybíjení baterie a těchto parametrech není obrovský, představuje pouze 0.9 % z celkového profitu (zisku oproti

provozu bez baterie) a 0.03 % vzhledem k celkovému zisku provozu s baterií. Avšak, je důležité mít na paměti, že ztráty zde byly zohledněny pouze v nabíjení a vybíjení baterie, samotná baterie a zbytek modelu pracovaly v „ideálním“ režimu. Při reálném provozu by byly celkové ztráty ještě vyšší.

```
Celkový zisk ideální optimalizace:   1367012.4 Kč
Celkový zisk řídicí optimalizace:    1364750.4 Kč
Celkový zisk simulink model:         1364750.4 Kč
Celkový zisk simulink model ztráty:  1364333.9 Kč
Celkový teoretický zisk bez baterie: 1320391.0 Kč

Profit (Simulink vs zisk bez baterie):     44359.4 Kč

Rozdíl zisku přidáním ztrát do Simulinku:    416.5 Kč
Procentuální podíl vzhledem k profitu:         0.939 procent
Procentuální podíl vzhledem k celkovému zisku: 0.031 procent

Zisk Simulink se ztrátami vs bez baterie:  43942.9 Kč

Ideální optimalizace vs Simulink bez ztrát: 2262.0 Kč
Ideální optimalizace vs Simulink + ztráty:  2678.5 Kč
```

Obrázek 6.33 - Porovnání zisků jednotlivých optimalizačních programů

Z vykreslených grafů pro průběh *Pgrid* pro jednotlivé programy, lze vidět, že rozdíly zde jsou opravdu velmi malé, což potvrzuje i podobnost výsledků celkových zisků.



Obrázek 6.34 - Porovnání průběhu Pgrid pro jednotlivé optimalizační programy

7. Závěr

Výsledkem této práce je vytvoření několika nových programů v Matlab jazyce a Simulink model umožňující simulaci a optimalizace práci bateriového úložiště ve spolupráci s malou vodní elektrárnou. Na základě Matlab optimalizační programů bylo vytvořeno několik další programů pro analýzy různých veličin.

Program pro analýzu optimálních parametrů SoC_max a Pbat_max (velikosti kapacity baterie a výkonu střídače/nabíječky) pro danou lokalitu vzhledem k návratnosti takovéto investice. Tato analýza ukázala že pro danou malou vodní elektrárnu by nejlépe vycházelo bateriové úložiště o velikosti 50 kWh se střídačem o výkonu 50 kW, jak v roce 2022 tak v roce 2023. Dále potvrdila že nemá cenu investovat do většího výkonu střídače, než je maximální možný výkon elektřiny do sítě. Návratnost této investice vysoce ovlivňuje výše jednotkové ceny elektřiny. Pro rok 2022 by návratnost této investice činila 9.25 let. Zde by mělo smysl dále zjišťovat, jak dlouho by baterie vydržela a o kolik by se prodloužila návratnost při reálném řízení a započtení veškerých ztrát reálného provozu. Pro rok 2023 vyšla návratnost investice mnohem vyšší, jelikož průměrné jednotkové ceny elektřiny byli celý rok mnohem nižší. Konkrétně by návratnost byla více než 16 let. Zde si myslím, že investice při těchto podmínkách nedává smysl. To však neznamená že tomu tak bude i v budoucnu, bateriový sektor je velmi rychle se rozvíjejícím sektorem a bateriová úložiště jsou stále dostupnější. Osobně se domnívám že investice do bateriového úložiště smysl v budoucnosti dávat bude. Výhoda mnou vytvořeného programu pro analýzu je jeho univerzálnost, možnost ho jednoduše přizpůsobit a použití s jinými daty a pro jinou lokalitu. Tudíž pro posouzení, zdali se investice vyplatí může sloužit i v budoucnosti, a i pro jiné projekty. Mimo to, analýza také ukazuje závislosti jednotlivých parametrů na ceně a návratnosti celé investice.

To že se ukázalo, že bateriové úložiště není pro takovýto účel v současné době rentabilní, neznamená že baterie nemají v hydro sektoru svoje využití již v současné době. Zde bych chtěl znovu poukázat na projekt X-Flex Hydro, který zkoumá hybridní provoz velkých kaplanových turbín. Při takovémto využití má baterie hlavní vliv na prodloužení životnosti soustrojí, tím že je baterií pokryta regulace pro vyrovnávání nerovností sítě, turbína nemusí tak často měnit svůj výkon. Tyto změny a regulace mají velký dopad na životnost těchto turbín, pro které je mnohem přívětivější provoz s konstantním výkonem. Jelikož opravy takto velký turbín jsou vysoce nákladné, domnívám se, že zde si svoje uplatnění hybridizace vodních elektráren najde.

Dále jsem také provedl analýzu vlivu řízení na celkovém zisku. Tato analýza proběhla díky mnou vytvořeným programům pro optimalizace prodeje do sítě a baterie, které fungují na odlišných principech (jsou různě vzdáleny od reálného řízení). Ukázalo se že při malé baterii a malém výkonu střídače je rozdíl mezi „ideální“ a „řídící“ optimalizací velmi malý. Naopak čím se baterie a její výkon zvyšuje tím hraje způsob optimalizace a předpovědi větší roli, tudíž při pořízení většího bateriového úložiště se nebudou zvětšovat náklady pouze na toto zařízení, ale také náklady, které bude potřeba vynaložit na lepší řízení a regulaci, tak aby byl využit potenciál většího úložiště.

Vytvořený Simulink model, představuje velmi dobře modifikovatelný nástroj pro simulaci malé vodní elektrárny s aplikací bateriového úložiště, který může být dále rozšiřován a vylepšován v jednotlivých částech, podle potřeb dalšího výzkumu. Tento model také nabízí širokou možnost výstupů, které dokáže uživatel poměrně snadno získat. Zároveň díky grafickému prostředí Matlab Simulink se jedná o velmi názorný systém, který může být využit pro prezentační a výukové účely. Při porovnání s vytvořenými Matlab programy, se potvrdilo, že řídící algoritmus v tomto modelu pracuje stejně.

Ve všech mých programech, jsem se nezabíral modularitou baterií a možností pro jednotlivé velikosti, toto vše je nutné zkonzultovat s odborníkem v daném sektoru a konkrétní projekt.

Veškeré mnou vytvořené programy, byly zkonstruovány jako univerzální a modifikovatelné, aby mohly být přizpůsobeny pro jiná vstupní data a parametry, tím pádem i pro jinou lokalitu a malou vodní elektrárnu.

8. SEZNAMY

8.1. Seznam použitých zkratek a symbolů

P_t	výkon turbíny	[kW]
P_{grid}	výkon do sítě (prodej elektřiny)	[kW]
P_{bat}	příkon baterie	[kW]
jc	jednotková cena elektřiny	[Kč/kWh]
P_{bat_max}	maximální příkon baterie (maximální rychlost nabíjení)	[kW]
P_{bat_min}	minimální příkon baterie (maximální rychlost vybíjení)	[kW]
P_{grid_max}	maximální výkon do sítě	[kW]
SoC	stav nabití baterie	[kWh]
SoC_{max}	maximální stav nabití baterie (kapacita)	[kWh]
SoC_{start}	počáteční stav nabití baterie	[kWh]
V_{nom}	nominální napětí baterie	[V]
T_{int}	teplota interiéru strojovny	[°C]

8.2. Použitá literatura

[1] Jan Kraus. Simulace provozu vodní elektrárny. [Online]. Praha 2022. Bakalářská práce, ČVUT. Vedoucí práce DR. Ing. Petr Nowak. [cit. 2023-12-23]. Dostupné z: https://dspace.cvut.cz/handle/10467/102586

[2] C-Energy. Dlouhodobá udržitelnost výroby energií. [Online]. [cit. 2023-12-23]. Dostupné z: https://www.c-energy.cz/o-nas/technologie.

[3] Hybrid.cz. "V Evropě už jsou stovky bateriových úložišť, v Česku zatím čtyři. Nahradí je v budoucnosti vodík?" [online]. 2023. [cit. 2023-11-11]. Dostupné z: https://www.hybrid.cz/v-evrope-uz-jsou-stovky-bateriovych-ulozist-v-cesku-zatim-ctyri-nahradi-je-v-budoucnosti-vodik

[4] SUASGROUP.CZ. "Na Sokolovsku zprovoznila společnost SUAS BESS největší bateriové úložiště v ČR." [online]. [cit. 2023-11-11]. Dostupné z: https://www.suasgroup.cz/news/na-sokolovsku-zprovoznila-spolecnost-suas-bess-nejvetsi-bateriove-uloziste-v-cr#:~:text=Spole%C4%8Dn%C4%9B%20s%20Energetick%C3%BDm%20investi%C4%8Dn%C3%ADm%20fondem,za%C5%99%C3%ADzen%C3%AD%20sv%C3%A9ho%20druhu%20v%20%C4%8CR.

[5] Energie bez emisí. "Ve Vítkovicích letos začne fungovat největší bateriové úložiště." [online]. [cit. 2023-11-11]. Dostupné z: https://energiebezemisi.cz/novinky-v-oboru/vitkovice-baterie-cez-start/.

[6] David Tramba. "ČEZ chystá další velká bateriová úložiště, vyrostou hlavně v areálu uhelných elektráren." [online]. [cit. 2023-11-11]. Dostupné z: https://ekonomickydenik.cz/cez-chysta-dalsi-velke-bateriova-uloziste-vyrostou-hlavne-v-arealu-uhelnych-elektraren/.

[7] Decci. Česká skupina DECCI zahájila výstavbu unikátního řešení pro vyšší energetickou bezpečnost a flexibilní využití obnovitelných zdrojů. CIIRC je jedním z partnerů. [online]. [cit. 2023-11-11]. Dostupné z: https://www.ciirc.cvut.cz/cs/solution-for-higher-energy-security-and-flexible-use-of-renewable-resources/.

[8] Novinky.cz. ČEZ zvažuje přestavbu Orlíku na přečerpávací elektrárnu. [online]. [cit. 2023-11-11]. Dostupné z: https://www.novinky.cz/clanek/ekonomika-cez-zvazuje-prestavbu-orliku-na-precerpavaci-elektrarnu-186421.

[9] Francesca Ottoni, PE, MBA. Pairing hydropower with battery storage. [online]. [cit. 2023-12-23]. Dostupné z: https://www.hatch.com/About-Us/Publications/Blogs/2021/07/Pairing-hydropower-with-battery-storage#:~:text=In%20addition%20to%20wind%20and,free%2C%20affordable%20power.

[10] Independent Electricity System Operator. Ontario's electricity system moves forward with largest energy storage procurement ever in Canada. [online]. [cit. 2023-12-23]. Dostupné z: https://www.ieso.ca/en/Corporate-IESO/Media/News-Releases/2023/05/Ontarios-electricity-system-moves-forward-with-largest-energy-storage-procurement-ever-in-Canada.

[11] Energy-Storage.News. Pumped hydro and batteries combine at Engie's new Bavaria project. [online]. [cit. 2023-11-11]. Dostupné z: https://www.energy-storage.news/pumped-hydro-and-batteries-combine-at-engies-new-bavaria-project/

[12] XFLEX Hydro. "XFLEX Hydro." [online]. [cit. 2023-11-11].
Dostupné z: www.xflexhydro.com.

[13] NS Energy. "XFLEX HYDRO Project, Portugal, Switzerland, and France." [online]. [cit. 2023-11-11]. Dostupné z: https://www.nsenergybusiness.com/projects/xflex-hydro-project/

[14] Wikipedia. "Lineární programování." [online]. [cit. 2023-11-11]. Dostupné z: https://cs.wikipedia.org/wiki/Lineární_programování.

[15] MathWorks. MATLAB Help - linprog." [online]. [cit. 2023-12-23]. Dostupné z: https://www.mathworks.com/help/optim/ug/linprog.html.

[16] Wikipedia. "Nelineární programování." [online]. [cit. 2023-11-11]. Dostupné z: https://cs.wikipedia.org/wiki/Nelineární_programování

[17] Stanford University. MS&E213 / CS 269O - Introduction to Optimization Theory. [online]. [cit. 2023-11-11]. Dostupné z: https://web.stanford.edu/~sidford/courses/19fa_opt_theory/fa19_opt_theory.html.

[18] Wikipedia. "Dynamické programování." [online]. [cit. 2023-11-11]. Dostupné z: https://cs.wikipedia.org/wiki/Dynamické_programování

[19] Wikipedia. Mathematical optimization. [online]. [cit. 2023-11-11]. Dostupné z: https://en.wikipedia.org/wiki/Mathematical_optimization.

[20] Kirsi Niinimäki, Essi Karell. Optimization Techniques: An Overview. [online]. [cit. 2023-11-11]. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-030-15483-7_2.

[21] T. Agami Reddy, Gregor P. Henze. Optimization Methods. [online]. [cit. 2023-11-11]. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-031-34869-3_7.

[22] Ing. et Ing. Jan Kopřiva, Ph.D.. Srovnání algoritmů při řešení problému obchodního cestujícího. [online]. [cit. 2023-11-11]. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/23246.

[23] Wikipedia. Celočíselné programování. [online]. [cit. 2023-11-11]. Dostupné z: https://cs.wikipedia.org/wiki/Celo%C4%8D%C3%ADseln%C3%A9_programov%C3%A1n%C3%AD.

[24] Gerhard Venter. Review of Optimization Techniques. [online]. [cit. 2023-11-11]. Dostupné z: https://www.researchgate.net/publication/228008187_Review_of_Optimization_Techniques.

[25] Stephen J. Wight. Optimization | Definition, Techniques, & Facts. [online]. [cit. 2023-11-11]. Dostupné z: https://www.britannica.com/science/optimization.

[26] Vinicius Fulber-Garcia. "Deterministic and Stochastic Optimization Methods." [online]. [cit. 2023-11-11]. Dostupné z: https://www.baeldung.com/cs/deterministic-stochastic-optimization

[27] Masárová Mária. Genetické algoritmy. [online]. [cit. 2023-11-11]. Dostupné z: https://www.fit.vut.cz/study/thesis/3454/.en.

[28] Wikipedia. Stochastic. [online]. [cit. 2023-11-11]. Dostupné z: https://en.wikipedia.org/wiki/Stochastic.

[29] Wikipedie. Heuristické algoritmy. [online]. [cit. 2023-11-11]. Dostupné z: https://cs.wikipedia.org/wiki/Heuristick%C3%A9_algoritmy.

[30] Wikipedia. Metaheuristic. [online]. [cit. 2023-11-11]. Dostupné z: https://en.wikipedia.org/wiki/Metaheuristic.

[31] Kumar Chandrakant. Overview of Nature-Inspired Metaheuristic Algorithms. [online]. [cit. 2023-11-11]. Dostupné z: https://www.baeldung.com/cs/nature-inspired-algorithms.

[32] Wikipedia. Genetic algorithm. [online]. [cit. 2023-11-11]. Dostupné z: https://en.wikipedia.org/wiki/Genetic_algorithm

[33] Wikipedia. Simulation-based optimization. [online]. [cit. 2023-12-23]. Dostupné z: https://en.wikipedia.org/wiki/Simulation-based_optimization.

[34] Elham Taghizadeh. Simulation-Based Optimization: Stimulate To Test Potential Scenarios And Optimize For Best Performance. [online]. [cit. 2023-12-23]. Dostupné z: https://www.informs.org/Publications/OR-MS-Tomorrow/Simulation-Based-Optimization-Stimulate-To-Test-Potential-Scenarios-And-Optimize-For-Best-Performance.

[35] Jaroslav Čábelka, Pavel Gabriel. Matematické a fyzikální modelování v hydrotechnice. Praha : Academia, 1987.

[36] Gabriel Pavel, Kučerová Jitka. Navrhování vodních elektráren. První. Praha : České vysoké učení technické, 1995. ISBN 80-01-01304-9

[37] Melichar Jan. Malé vodní trubíny. 2. Přeprac. Praha : Vydavatelství ČVUT, 1996. ISBN 80-01-02164-5.

[38] MathWorks. Simulink User's Guide R2022a. [online]. [cit. 2023-12-25]. Dostupné z: https://www.mathworks.com/help/pdf_doc/simulink/simulink_ug.pdf

[39] MathWorks. Simscape User's Guide R2022a. [online]. [cit. 2023-12-25]. Dostupné z: https://www.mathworks.com/help/pdf_doc/physmod/simscape/simscape_ug.pdf

[40] MathWorks**.** Simscape Language Guide R2022a. [online]. 2022. [cit. 2023-12-25]. Dostupné z: https://www.mathworks.com/help/pdf_doc/physmod/simscape/simscape_lang.pdf

[41] MathWorks. MATLAB Multicore. [online]. [cit. 2023-12-28]. Dostupné z: https://www.mathworks.com/discovery/matlab-multicore.html

[42] MathWorks. Parallel Computing Toolbox. [online]. [cit. 2023-12-28]. Dostupné z: https://www.mathworks.com/products/parallel-computing.html

[43] MathWorks. Interactively Run Loops in Parallel Using parfor. [online]. [cit. 2023-12-28]. Dostupné z: https://www.mathworks.com/help/parallel-computing/interactively-run-a-loop-in-parallel.html

[43] MathWorks. Decide When to Use parfor. [online]. [cit. 2023-12-28]. Dostupné z: https://www.mathworks.com/help/parallel-computing/decide-when-to-use-parfor.html

8.3. Seznam grafů

8.4. Seznam obrázků

8.5. Seznam schémat

8.6. Seznam tabulek