NTNU
Det skapende universitet

Design av et utviklingsverktøy for microgrids

En fleksibel og effektiv tilnærming

Gard Hillestad

Master i kybernetikk og robotikk
Innlevert: juni 2016
Hovedveileder: Marta Maria Cabrera Molinas, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

NTNU

Institutt for Teknisk Kybernetikk

2016

Designing a Development Support Tool for Microgrids

MASTER THESIS

GARD HILLESTAD

Sammendrag

Teknologi som angår solcellepaneler, biomasse kraftver, batterier og microgrids utvikles stadig mot mer kosteffektive og funksjonelle løsninger. Men planlegging av de nødvendige innkjøp for å kunne ta i bruk denne teknologien er noe kompleks. Komponentdimensjoner avhenger tungt av hvordan de slites under forskjellige forhold, og disse variasjonene har store økonomiske og praktiske konsekvenser. Særlig gjelder det de med magre ressurser, som trenger denne teknologien aller mest.

Dette prosjektet har som formal å utvikle og ferdigstille et utviklingsstøtteverktøy for microgrids der størrelse på solceller, batteri og ekonomiske projeksjoner skal beregnes gjennom simulasjoner. Det skal også være mulig å inkludere biomasse generasjonsalternativer i simuleringene.

Verktøyet skal være tilpasset lansering som open source, ha nøye dokumentasjon tilgjengelig på nett, og skal videre ha et brukergrensesnitt for å forbedre prosesser som testing og generell bruk. All koden er skrevet i matlab og har blitt implementert med generiske tilnærminger til problemstillinger som brukere står ovenfor, på denne måten kan utviklingsverktøyet brukes i de fleste kontekster uavhengig av detaljer i planlagte systemer. Koden er strukturert nøye, basert på publiserte studier av kode kvalitet, for å legge til rette for videre utvikling.

Abstract

Technologies concerning photovoltaic (PV), biomass power generation, batteries and microgrids are constantly evolving to more cost efficient and functional solutions. But planning the required purchases to utilize this technology is somewhat complex. Component dimensions depend heavily on the operational conditions, and the variations in these conditions have large economic and practical implications. Especially for those with meager resources who need this technology the most.

This projects puprose is to develop and finish the design of a ‘Development Support Tool’ for microgrids, where size of Photovoltaic panels, batteries and economic projections are calculated with help from simulations. It is possible to include biomass generation strategies in the simulations.

The tool is designed to be suited for open source publishing with a thorough documentation online and a user interface to improve processes such as testing and general use-cases. All the code is written in Matlab and has been implemented with generic approaches to problems that users are faced with, and in this way making the tool applicable to a wide range of microgrid contexts, independent of system details. The code has been structured meticulously based on published studies on code quality to facilitate continuing development.

Preface

Dedicated to Tord Are Meisterplass for being a warrior and generally awesome person.

In memory of Thomas Tveten.

I extend my deep gratitude for everyone who offered their support and encouragement, especially my dear friends who have inspired me to reach difficult goals.

I must thank NTNU and its staff for all the opportunities I have been provided here. My supervisor Marta Molinas for dedicating her time to projects that can make the world a better place, to Stephano Mandelli and Håkon Duus for the inherited work.

Thanks to my large and small family for their continued love and support.

Abbreviations

LCoE	Levelized Cost of Energy
NPC	Net Present Cost
LLP	Loss Of Load Probability
GUI	Graphical User Interface
UI	User Interface
DoD	Depth of Discharge
DST	Development Support Tool
LoL	Loss of Load
MSE	Micro and Small-scale Enterprises
PV	Photovoltaic
SoC	State of Charge
OeMeR	Operations, maintenance and replacement
OeM	Operation and maintenance

List of Figures

Table of Contents

1: Introduction

1.1 Background

The following sections are purposed to properly include the reader in the scope of this thesis. After the motivation comes sections meant to introduce necessary technologies and concepts. What role the technologies play in the thesis context is discussed in section 1.2 and 1.3.

1.1.1 Motivation

The world's population is growing, at the same time the standard of living is raised in developing countries. Combined, these two factors pose an unparalleled challenge for humanity as this will raise the demand on the global environment substantially, which can again lead to grave humanitarian consequences.

The International Energy Agency states that 1.3 billion people live without electricity, 80%, or more than 1.1 billion people out of these, live in rural areas where families mainly depend on small-scale agriculture to survive (International Energy Agency 2016). Renewable off-grid systems are the best option for rural electrification (Mandelli, et al. 2014), but how available are these solutions?

Intellectual resources can be disproportionately expensive, putting up barriers for some wishing to make use of technological progress. Empowering those without the infrastructure and licensed intellectual property (IP) of western businesses is crucial if we want sustainable technology to be utilized seriously where it is needed (Chon 2006); not only out of humanitarian - but also environmental considerations.

Technologies concerning photovoltaic (PV), biomass power generation, batteries and microgrids are presently deveoloping more cost efficient and functional solutions (Maximillian 2015). But the entry barrier in competence is high due to microgrid complexity.

Micro and small-scale enterprises (MSE), such as farmers or small business owners, would benefit greatly from a development support tool for microgrids that can determine what components to purchases and what finances are necessary to realize microgrid power systems.

This would boost a wider use of microgrid- and sustainable technology, it would facilitate project proposals to acquire financial support where it is needed most and improve the lives of people in the present and future, while minimizing climate impact.

1.1.2 Terminology

1.1.2.1 CRF – Capital recovery factor

Capital recovery factor converts a present cost to a steady stream of partial payments given a number of payments *n*, and an interest between each payment, *r*.

$$(1) \quad CRF = \frac{r(1+r)^n}{(1+r)^n - 1}$$

1.1.2.2 LCoE – Levelized cost of energy

LCoE is what you pay for each kW supplied successfully to the load.

$$(2) \qquad LCoE = \frac{NPC \times CRF}{kWh_{supplied}}$$

CRF is necessarily included since we are making an annual analysis of the system. LCoE is therefore the annuity of the NPC divided by the kWh supplied each year.

1.1.2.3 LLP – Loss of load probability

Loss of load probability is the probability that a kW of power in the load profile will be unmet.

$$(3) \qquad LLP = \frac{\sum Lost\ Load}{\sum Load\ Demand}$$

Understanding what affects the LLP is important. Cloudy days with increased power consumption due to increased indoor activity or heaters may consistently lose power with high contribution to the LLP value. If this only happens a few times every year, the instances might be considered unimportant and will have less impact on the perceived quality of the system than reflected in the LLP increase.

On the other hand, if the supply of power from solar panels just barely is insufficient to serve the load over long periods of time, the ratio off lost load to the load demand is then very small in the simulations, but in reality you still have a power shortage, meaning that the low percentage does not reflect the loss of system utility.

1.1.2.4 SoC – State of charge

State of charge is the ratio of energy charged to the maximum capacity of the battery.

$$(4) \qquad SoC = \frac{Charge\ Level\ kWh}{Maximum\ Capacity\ kWh}$$

1.1.2.5 DoD – Depth of discharge

The compliment of SoC

$$(5) \quad DoD = 1 - SoC$$

1.1.2.6 Irradiance

Irradiance is the gross watt per square meter from the sun. Precise data can usually be found on NASAs webpage. https://eosweb.larc.nasa.gov/sse/

$$(6) \quad irradiance = \frac{w}{m^2}$$

1.1.3 Open Source

Open-source is the practice of open process internet based development projects, where developers can make contributions to projects through reviewing existing code or adding new features. Open source will help keep software alive as it invites users to participate in increasing the quality of the software, this way the development will increase with the degree of distribution. Open source code is free for anyone to download under the creative commons license.

Open-source contributors have proved to work with a high level of expertise and commitment, they are highly motivated and work for free. Linux and Ubuntu are examples of software used by developers in high prestige corporations. Wikipedia has become the largest encyclopedia in history compiling information on nearly every subject of modern knowledge.

1.1.4 Microgrids

A microgrid is subsystems of interconnected loads and distributed energy sources that are locally controlled. The microgrid can either be islanded or connected to the grid. This is associated with network operational benefits such as cost efficiency and voltage control. It will also enable the implementation to be more adapted to the local conditions and new technology (Berkeley Lab 2016).

Microgrids are complex, which often lead to increased investment costs that put up barriers for widespread use, they do however facilitate additional value streams that improve economic viability. (Stadler, et al. 2015). This thesis is concerned with making the complexity of the microgrids, more comprehensive, hence reducing the investment cost.

1.1.5 Biomass Power Generation

In this paper, biomass power generation includes the sustainable combustion of biomass from either agriculture, municipal or household waste, or digestion of either. Sewage or livestock waste can also be digested to produce biogas.

1.1.5.1 Thermal Power Generation

Thermal power generation is the transition of thermal energy in steam to a turbine. The steam is heated by a boiler, which in turn is heated by a furnace or by solar focalization. Heating implies a startup delay depending on the furnace efficiency and the boiler size.

The steam quality measured in dryness has to be high, at least 90%, meaning only 10% water vapor, as any higher amount will damage the turbine. The most efficient steam turbines require superheated steam of dryness 99.5%, which in turn require minimum leakage and high efficiency from the boiler and furnace.

Solids in the water needs to be filtered out through a fine mesh. The process lose little water and so there is no need for a stable water source once sufficient filtered water is acquired.

Steam turbines does not usually require much maintenance and can have a 20 year life span without any extra costs. The boiler and pipes might need minor repairs, but these components are normally easily available and does not require special training to handle.

1.1.5.2 Producer Gas from Biomass Gasifier

Gasifiers are special furnaces that use incomplete combustion of biomass to produce ignitable gas, known as producer gas, for several types of generators. Producer gas consist of carbon monoxide and nitrogen, which is “scrubbed” from tar so that it may be used in either gas turbines, gas or diesel generators, the latter in combination with diesel. In any case there will be a startup delay when starting the combustion process.

The gasifiers are approximately the same size as the thermal power plants but the weight of the specialized components are combined much heavier. In the thermal power plant, only the steam turbine has to be shipped and the remaining parts bought locally, meaning that shipping costs are much higher for a gasifier.

The generator can be replaced with several variations of locally available engines, the only required customization to off-the-shelf generators is to change the air intake to producer gas mixture ratio of 1:1. Producer gas therefore offer increased flexibility compared to the thermal power plant and potentially a longer plant life-span. The plant does require some special training to operate which can be supplied by manufacturer.

In conclusion, the producer gas solution provides more flexibility in generator choice, in exchange for a substantially higher investment cost, despite purchasing costs being quite similar.

1.1.5.3 Biogas from Anaerobic Digester

An anaerobic digester is simple to install and operate, and is available in any scale desired and comes at a fair price. The digester utilize biomass, waste, sewage or either to produce biogas. Biogas is mainly methane and carbon-dioxide, but also contain hydrogen sulfide, water and siloxanes, which are considered contaminants. Methods for producing uncontaminated biogas and removing contaminations post-production are continuously being developed.

The biogas are best utilized in a combined heat and power (CHP) gas generator, where the spill heat is used for the digester. The startup time is equal to that of a gasoline generator and can be less than a minute.

The Intergovernmental Panel on Climate Change (IPCC) suggest that over a 100 year period, CH_4 might contribute to trapping 34 times more heat in the atmosphere than CO_2 (Alexander, et al. 2013), measures to counteract dissipation of methane should be taken seriously. Anaerobic digestion will reduce the CH_4 emissions during storage (S.G 2005) and is then replaced by CO_2 emissions at combustion, a lesser of two evils.

Using biomass from agriculture and waste will diminish the strain on the local-, in addition to the global environment. Biogas can replace wood for cooking, a predominant reason for deforestation in areas with high poverty, this is relevant as 2.6 billion people still rely on traditional biomass for cooking (International Energy Agency 2013). Additionally, digested dung will reduce toxicity and odor which attract flies and rodents, creating a health hazard for rural populations (Sooch 2013).

Fossil fuels or reserve stocks of biogas can easily be stored at hand or purchased to completely replace the plant production, this enables for more operational resilience and the microgrid can be assigned to handle more vital tasks. Generally this resilience also implies a potential for purchasing lower losses of load, in the form of fuel, when desired or needed (Stadler, et al. 2015).

Digester solutions are more often subsidized than not, and purchase costs are usually quite low with high likelihood locally available parts. Economic aspects are hence much better than those of thermal power and producer gas plants.

On the other hand, a biogas digester requires much more infrastructure to run. The gas must be stored in special containers and the digester takes up space and the processed biomass must be removed. Larger operational costs are therefore implied.

Production on biogas depends strongly on the quality of biomass and the digester, estimating the needed size for production has to be discussed with manufacturers, the digester itself is relatively cheap and making a larger digester does not imply a big price difference.

For comparison of the different biomass system purchase costs see Table 1

Table 1: Some Rough Purchase-Price Indicators

post	**Thermal Power Plant**	**Producer Gas Gasifier**	**Biogas Digester**
Full system w/grid tie price (dec 2015)	40 000 $	40 500 $	20 000$
Price Quote Source	Green Turbine	All Power Labs	alibaba.com and alberta.gov, rough estimate
Transportation Cost	low 55-140* + insurances up to 550$	very high	very low
Notes	package weight 25 kg boiler locally available	package weight 693kg+70kg	typically locally available

* (DHL 2016)

1.2 Problem Formulation

While the potential and utility of installing microgrids with different technologies is very large, the investment cost is higher than conventional installations because of the complexity in design. The planners are faced with choices where details have large economic consequences, which cannot be taken lightly, especially in poor rural areas of development countries where such installations are needed the most. We wish to facilitate understanding and information that brings safety to investments made in microgrids, this will have substantial both humanitarian and environmental consequences, given the amount of people in need of electricity. We will achieve this with a development support tool (DST) for mictrogrids.

The technologies we wish to exploit are basically any that are eco-friendly and economically feasible. We wish the tool to be easily built upon with new features and new technology as it becomes available. The DST finished in this thesis utilize that of PV, biomass generators and batteries.

To indicate what a user can expect from performance given an investment, simulations are necessary. The simulations will estimate the performance of a given system, but also how systems are worn over time, which will have large consequences on the operations and maintenance of the simulated plant.

The product will be ready for shipping under the creative commons license, free for anyone to use. The development is intended for users and students to undertake, where both groups are encouraged to add features either from necessity while utilizing the DST or any other motivation.

The program follows a set of programming standards which allows for users and developers to inspect modules without understanding their context extensively, and also the program architecture without understanding how the modules work. The DST has a graphical user interface and detailed online documentation

1.2.1 How does the Development Support Tool (DST) Work?

The user supplies a range of component sizes of which performance and economic consequence is calculated. These calculations are based on simulations of systems with the different components in operation with economic and performance parameters. Data on the context of which the system will operate is also necessary, this always includes a load profile. The system finds the best solutions of certain characteristics, such as loss of load probabilities, and produce graphs and key figures to describe the performance and costs associated with the resulting systems to the user.

1.3 Literature Survey and Previous Work

1.3.1 External Research

There has been several papers on dimensioning microgrids. Some are considered mostly with macroeconomic aspects and highly complex considerations (Hawkes 2009), some with technical considerations concerning control (CHEN, et al. 2013) and others using IP software such as Homer (Hafez og Bhattacharaya 2012). The common denominator is however that the software used in the papers either cost money or are unavailable to the public. In addition, the intended reader of these studies are other scientist in the same field, the utility for the average MSE is therefore nonexistent

1.3.2 Existing tools

There are however tools on the market with capability of partially performing the analysis that we propose, these are however with limitations and licenses. An extensive list can be found in the literature review of (Connoly, Mathiesen og Leahy 2010) dated 2010 the paper does not mention BlueSol, PVSyst and RAPSim. Although somewhat outdated, it supplies us with a curriculum of different type-tools and respective core functionalities.

Studies on the functionality and limitations of contemporary tools has been carried out before at NTNU by Daniele Sakepa (Sakepa 2015) and Anne Lozé (Lozé 2015), the following section will try to briefly summarize to draw the distinction of our DST.

1.3.2.1 HOMER

Hybrid Optimization of Multiple Energy Resources (HOMER) is the most popular choice. The tool performs excellent economic analysis and simulations, and demands similar economic parameters as our DST, and additionally a database of products with parameter-sets automatically defined.

Flexibility is lacking in step sizes of PV and batteries. The step sizes are predefined and does not allow refinement of the solution space to better meet poorly funded users' needs where financial margins count more.

The tool does not support budget constrained-searches, described as Top-Down analysis in Connolly et.al. The development of this feature is discussed in section 3.4.2 as further work. The current DST has this problematic partially solved through the 'Simulations Overview' plot described in section 2.2.3.1.

HOMER supports biomass generation as one of the core functionalities. The tool does not, however allow for detailed operation of the biomass system as inputs to simulations, the user does not have control of operational conditions.

Some enigmatic behavior is described by Lozé, the claims are that battery discharge does not happen despite of sufficient capacity, and that different results is produced despite identical inputs. The occurrences are mentioned without plots or data associated for inspection. Concepts such as power-energy ratio (the discharge capacity given spesific charge levels) and the stochastic features of HOMER can explain these behaviors, but are not mentioned. Considering the existing integrity of HOMER and given the frailty of these posed enigmas, the notion should be ignored.

1.3.2.2 BlueSol

Not extensively tested in the reports due to price. The price is equal to the purchase of 1 kW of PV panels which is unacceptable in a small scale setting. The trial version is has too few features to produce satisfactory results.

Keeps a very large database but is then restricted to these choices. There is no user specific data inputs and if a user cannot find a set of choices that matches their use-case, the tool becomes imprecise and quite useless.

Does not include contribution from secondary sources such as biomass generators.

1.3.2.3 RETScreen

A free-of-charge excel based 'Clean Energy Management' tool, mainly focusing on economics and not including simulations and detailed performance data, this does however reduce computational time significantly, but the simulation feature is important to our users as they need to understand the practical implications of how a system operates.

As proposed by Connolly et.al, RETScreens main function is intended to justify clean energy investments compared to conventional power generation, and not necessarily to prospect installations in detail.

RETScreen does consider parallel generation strategies such as biomass power generation.

1.3.2.4 PVSyst

Mainly a highly detailed PV installation optimization tool. Like RETScreen, PVSyst does not seem to produce operational simulations with hourly load data, which are important in our specifications. Instead the tool makes its own load data through user entering the number of domestic appliances, lamps etc. This makes the tool further unsuitable if the user operates a farm, which have machines and appliances that do not translate to household appliances.

1.3.2.5 RAPSim

A review of this tool is included in the thesis of Håkon Duus (Duus 2015). The tool aims to be an open-source microgrid design tool. It works in optimizing placement of components and other electrical performance choices. It does not work with dimensioning and economic analysis which is paramount to our functionality. This tool could however be very useful to complement the use of our DST. "*The software RAPSim is designed for use in science and classroom with a simple to use graphical interface. It is an easily extendable framework that supports users in implementation of their own gridobject models and grid controlling algorithms"* (Pohacker og Khatib 2014)

1.3.2.6 Summarized, what's Generally Lacking

With exceptions to RETScreen and RAPSim, all the tools mentioned introduce increased costs for the user. The fees might be acceptable for a complete licensed DST purchase, but as our argument dictates, the programs have different short-comings, and fail to justify the investments.

Another quality in our DST is flexibility. As our tool is open-source and modularized, any kind of feature can be developed and used with the existing code, there are no restrictions. A benefit that also comes with this is complete transparency of calculations. If the user has any doubts or special cases previously unconceived in the design, the source-code is easily accessible for inspection.

The tool is also simple in use and more available to users with different technical backgrounds.

1.3.3 Past Work on the DST (logplot.m)

The DST was initially described and implemented by Stephano Mandelli and further developed by Håkon Duus and Jemjin Scheen. The program was inherited in a single Matlab .m file named `logplot.m` which is in the current DST under the 'Outdated' folder. The DST is currently too large to enclose as paper, but the entire program can be downloaded or inspected easily, please follow the instructions in Appendix A

The functionality of the DST in `logplot.m,` is explained here. There are three stages of calculations, namely the simulations, the economic analysis and the search for optimal solutions.

1.3.3.1 Producing Simulations

The tool takes a range of PV and battery sizes, every combination of which are used to produce simulations. Every combination of PV and battery sizes will each have a full timeline simulation of absorbing energy from the sun, serving the load, charging and discharging the battery. The simulations use a load profile, irradiation and temperature time series, and various scalar parameters that influence the performance of the components

This results in a computation complexity of $O(n^3)$, given a $number\ of\ PV\ sizes\ \times number\ of\ battery\ sizes\ \times length\ of\ timeseries$.

The system operates very simply: for every PV size, an array of absorbed power is calculated. If the absorbed power is sufficient to cover the load demand the excess power will charge the battery for each battery size.

If the absorbed power is insufficient, the battery will discharge. There will be a loss of load if there is insufficient power available, or if the load exceed the discharge capacity

For every discharge cycle, the depth of discharge (DoD) is used to calculate how large fraction of the battery lifetime is consumed. This calculation is made with a function called `cycles_to_failure`. This is a fitted function to the battery "cycles to failure" vs. "depth of discharge" characteristics, proposed by the battery manufacturer.

1.3.3.2 Conducting Economic Analysis

Each combination of PV and battery now have a values for ‘Lost Load’ and ‘Yearly Battery Replacement Frequency, the values are used to compute the LCoE, and how many battery replacements will be necessary during the plant lifetime, respectively.

The battery replacement present cost is accounted for in the NPC value associated with each simulation. Replacing batteries are very costly, and therefore the `cycles_to_failure` function has a major impact on the economic results.

1.3.3.3 Finding Optimal Solutions

The last step is finding the PV/battery combinations which are worth considering for purchase. The tool finds matches for a user defined LLP range in the simulation outputs, the combinations with the lowest NPC for a given LLP range are deemed optimal solutions.

The code of the tool is located within one file. This implies that the user will have to run the entire file, regardless of the desired functionality. The lack of modularity is caused by somewhat tangled functionality, but also a practice of overwriting variables during iterations.

1.3.3.4 A Use-Case for Logplot.m

A use case of the inherited tool is explained here. The parts that are confusing will hopefully be more apparent when the new DST is explained later in this thesis.

Parameter Input

The user needs to research the parameters of the components considered. Since both technology and economic conditions change as time goes by, any result produced by the DST will be useless if the parameters are not up-to-date. Below is an overview of the different identifiers relevant to the user, how they are named in the DST is seen in the ‘Methodology’ chapter of the thesis.

- The parameters needed from component manufacturers are:
  `batt_ratio, coeff_T_pow, costBatt_coef_b, costBatt_coeff_a, costINV, costOeM_spec, costPV, eff_BoS, eff_char, eff_disch, eff_inv, irr_nom, LT, T_nom, T_ref, SoC_min, max_y_repl, SoC_start,` and the function `cycles_to_failure`.
- The data series estimates necessary to acquire are:
  `irr, T_amb, Load`
- The parameters already found in research are:
  `coeff_cost_BoSeI, r_int.`

These parameters are hardcoded somewhere in the script. The tool depends on the users understanding of the different parameters based on the names and comments that might accompany them. There are no explanation of whether they should be researched and changed or not. There is no clear distinction between calculation variables and constant parameters.

Simulation Scope Refinement

Next the users need to find a range of PV and battery combinations to simulate, I call the matrix indexed by the PV and battery iterations the 'Simulation Space'

- The parameters that will define the simulation space are:
  `max_batt, max_PV, min_batt, min_PV, n_batt, n_PV, step_batt, step_PV`

There will be `n_batt, n_PV` simulations based on these variables. If the user cannot find sizes combinations with appropriate LLP or NPC, an expansion of this range is necessary. If the user is interested in higher resolution of the solution space, a smaller range and smaller step is necessary.

Finding optimal solutions

The user defines a range of LLP in the variable `x_llp`. These values get optimal solutions associated with them in the `MA_opt_norm_bhut_jun_10_16` output matrix. The quality of solutions is given by LLP, NPC and LCoE.

- Plotting and Inspecting Solutions

  The plotting and inspection of values must with few exceptions be hard-coded into the simulations part in order to access the data, either in the form of polling values, break points or producing figures at certain iterations.

If the solution is unsatisfactory, the parameters can be changed and process repeated.

1.4 The Contribution of this Thesis

Starting from logplot.m, a proto-product has taken form. The code has moved from being a single script with many hard-code features to becoming a collection of interchangeable modules, help functions and GUI functions.

The tool has been rewritten with strict coding standards, and bugs have been fixed. The code is more flexible with respect to its structure and readability, but also with respect to data inputs which can now span several years without having to modify the code. The program operates on any operating system without any modification having to be made.

New modules that support biomass power generation are added. The biomass state machine takes on a generic approach that enables it to simulate the behavior of autonomous, or manually operated systems.

The new DST is accompanied by a powerful GUI that speeds up the use of the system significantly. The GUI additionally facilitates an understanding of the intersection between user and program, where this earlier was unclear. The GUI sketches a proto-program for the shipped DST.

The entire set of parameters necessary to run the program is input to the `dst_platform` GUI where the input sets can be saved as presets. This allows the user to cycle quickly between presets, to save, delete or update them when needed when comparing different solutions or working with several configurations.

The `dst_platform` GUI enables execution of simulations in different run-modes, different stages of calculations can be repeated if modes or parameters are changed. Documentation look-up and solution-space plots are quickly produced by mouse clicks.

The `solution_explorer` GUI enables detailed inspection of solutions produced from simulations, economic analysis and the optimum search calculations. Solutions can be chosen from a list, and outputs such as plots and key values can be produced quickly. The outputs can easily be put aside for comparison with other solutions or new parameters.

The DST has an extensive online reference guide which span more than 30 A4 pages. The online documentation is carefully written to express the meaning of each parameter and output of the DST, it also explains the functionality of each module, help function and GUI which combined amount to 23 Matlab files.

The new DST is not only functional but manageable, and each iteration has been made with the project successors in mind to ensure the vitality of the project.

1.5 Objectives

The main objectives of this master's thesis are:

1. To design and implement a powerful DST for microgrids, flexible to both diverse microgrid contexts and user needs.
    a. Strict code quality standards.
    b. A new biomass generation module for simulations.
    c. Elaborate GUIs that produce a wide variety of outputs.
2. To thoroughly document and ensure accessibility for later contributors.

1.6 Approach

1.6.1 Redesign of inherited Tool

The first step in this thesis was changing all the names in `logplot.m` to fit some quality standards. The changes were based on studies and personal experience while using the code, which had been difficult for both the author and other collaborators. The goal was to use self-documenting identifier names so that a line of code could be read completely out of context, and still make sense.

Then the code was split into modules, one for each analysis step described above, namely: simulations, economic analysis and optimal search. The outputs and inputs were organized into classes, and a large amount of variables previously overwritten during iterations in logplot.m were stored as large matrices. Some plotting functionality and `cycles_to_failure` was also placed in modules.

The reorganized code was tested for performance and correctness and some erroneous calculations were corrected.

1.6.1.1 Immediate Benefits of increased Code Quality

The code can now be tested in parts without waiting for large computations each time, matrices of outputs can be inspected and plots produced instantly. This speeds up interaction with - and development of the code tremendously.

Questions about functionality is more frequently answered without looking in the code for comments that explain the variables concerned. Errors stand out as they break the logical narrative that results from these standards.

1.6.1.2 Long-term Benefits of increased Code Quality

A philosophy of open-source is that given enough eyes, all bugs are shallow. This is to some degree true, but some bugs cannot be shallow if they are part of the specification or macro structure; these are called upstream errors. Upstream errors are 10-100 times more resource demanding to fix than downstream errors (shallow bugs) according to (Bollinger, et al. 1999).

Up-stream errors are not easily detected and fixed unless the code quality is at a certain level. High quality code have modularity and readability that express specifications and communications in the program.

Open-source needs to be “rigorously modular, self-contained, and self-explanatory”.

While good closed-source organizations are of course aware of the benefits of good modularity, only open-source methods provide the kinds of individual incentives though which such practices can easily flourish and evolve over time. They also provide a warning about efforts such as Netscape's Mozilla that attempt to move weakly modularized proprietary code into open source (Bollinger, et al. 1999)

It has been shown empirically that there are significant associations between flawed identifiers and code quality issues though a static analysis tool called FindBugs. (Butler, et al. 2009). The tool tests mainly for run-time errors but also maintainability.

The bottom line is that open source projects are highly unlikely to have any success if there aren't some ground rules, much like that of a buildings foundations.

1.6.2 Creating a user interface

A GUI was created with Matlab. The GUI code files are very large but contains simple functionality. Elements and their properties are defined with Matlab functions, the implementation looks much like that of CSS.

The `dst_platform.m` GUI was made to handle the inputs, simulation and calculation part of the DST, and the `solution_explorer.m` was mainly made to handle the outputs to the user. The GUI was used to test and develop features in this thesis.

1.6.3 Simulating a biomass power system

The SAPV simulation module was modified to support the operation of a biomass generation system. The operation is realized by a state machine where the different states determine how long power generation should run each hour.

The inputs were chosen to be as generic as possible to support any kind of biomass power production that can be implemented whether it runs automatically or is operated by humans, or whether it is powered by a gas generator or a steam turbine

1.6.3.1 Why is the Biomass Feature important?

The daily variation of irradiance and load demand can vary so much that dimensioning a SAPV power plant without large amounts of overproduced power is impossible. Variations can be extreme as a result of negative correlation between load and irradiation, this can for example occur in winter if irradiation decrease and simultaneously load demand increase with use of electric heating. The power plant is dimensioned to handle these cases which demand much more than the normal operation.

Overproduction can be considered a loss of money, the PV size ‘too large’ for the battery size. A reduction of the system LLP becomes much more expensive in terms of LCoE once overproduction starts occurring in a system.

Biomass power generation can solve dimensioning problems because it can be stored and used when it is needed. Biomass can also be bought and sold, and provides a flexibility and operational resilience that is unavailable in a SAPV plant. Operational resilience is important because it makes the microgrid eligible to handle tasks that have harder demands on operational reliability, such as providing clean drinking water and sanitation.

1.6.4 Documentation

Documentation was written in HTML and published online. The documentation is written in very simple code so that editing and adding to the HTML code require a minimal of programming skills. The code was made to be easily searched or browsed and can be accessed from a phone or computer.

This is highly important as any user or developer will have to look for explanations within the code if no online reference is available. This does not only take time, but the code comments risk disappearing during new iterations of the software. The survivability of the DST is completely dependent on good and available documentation.

1.7 Limitations

1.7.1 Microgrid Technical insight

Detailed technical understanding of microgrids is not part of the author’s field of study. I have a background in technical cybernetics and is therefore familiar with simulations, automation and program development. This has not been very problematic but there is the chance that someone with an intimate understanding of microgrids would have some objections.

1.7.2 Program Beta Tests

The decision was made not to send me alone to Bhutan for beta testing the program. Beta testing means to test user experience with the software and improve upon these results, meaning that user friendliness is unevaluated by people outside the university. Design improvements will to be done when feedback has been recorded, as there is probably improvement potential here.

1.7.3 Training Sets of Data

Tests have not extended much past the data provided by the previous collaborators. Training sets are not very important for most of the implementations done in this thesis. There is however the testing of the biomass system. The training set provided from Bhutan is a system which has high utility from a secondary power source, hence the testing was satisfactory without acquiring additional data.

1.7.4 Time

The project involves a large amount of tasks to be done. This means that time becomes a constraint. The functionality of features can be fairly simple, but every iteration of the software must be documented properly, tested and designed with successors in mind.

1.7.5 Synchronized work

Updated modules from earlier collaborators has been hard to acquire, hence some potential synergy was lost. The new standards of implementation should amend this as the code is not interconnected the same way so that features can be developed in parallel.

1.8 Structure of the Report

The rest of this thesis mainly consist of the methodology and corresponding discussion of what has been done. During the methodology part each successful addition to the DST is explained in detail. Each chapter aims to present its general approach and then the details of the implementation. The results of finished implementations follow the implementation details directly.

After all modules are discussed, a use case of the tool is presented in the last chapter with the discussion and conclusion. The use case will try to give an understanding of how the finished product can be used.

The source codes are digitally available because they are too large for the paper format. The explanation on how to get the digital resources are in the Appendixes.

2: Methodology

2.1 Restructuring Inherited Code

The decision was made to rewrite and restructure the existing work because logplot.m was difficult to use and develop. The formatting made the code hard to read, there were no naming convention to help developers to understand neither functionality nor equations, and the script had no modularity to enable agile development, testability and encapsulation.

The restructuring of logplot.m to the new DST follows a union of several code quality standards, but mainly the guidelines in Richard Johnsons ‘Matlab Programming Style Guidelines’ (Johnson 2002) and the paper ‘Relating Identifier Naming Flaws and Code Quality’ (Butler, et al. 2009). Macro structure quality guidelines were also found in the book ‘Clean Code’ (Martin 2009).

2.1.1 Naming Conventions, Commenting and Formatting

Metrics were found in Butler et.al to be correlated with error frequency in in 8 established open source Java applications libraries, this is mentioned in the introduction. The different measures of quality is found in Table 2: Code Quality Metrics.

Table 2: Code Quality Metrics (Butler, et al. 2009)

Name	Description	Example of flawed identifier(s)
Capitalization Anomaly	Identifiers should be appropriately capitalized	`HTMLEditorKit, pagecounter`
Consecutive Underscores	Consecutive underscores should not be used in identifier names	`foo__bar`
Dictionary Words	Identifier names should be composed of words found in the dictionary and abbreviations, and acronyms that are more commonly used than the unabbreviated form.	`strlen`
Excessive Words	Identifier names should be composed of no more than four words or abbreviations	`floatToRawIntBits()`
Enumeration Identifier Declaration Order	Unless there are compelling and obvious reasons otherwise, enumeration constants should be declared in alphabetical order	`enum Card {ACE, EIGHT, FIVE, FOUR, JACK, KING ...}`
External Underscores	Identifiers should not have either leading or trailing underscores.	`_foo_`
Identifier Encoding	Type information should not be encoded in identifier names using Hungarian notation or similar	`int iCount;`
Long Identifier Name	Long identifier names should be avoided where possible	`getPolicyQualifiersRejected`
Naming Convention Anomaly	Identifiers should not consist of non-standard mixes of upper and lower case characters.	`FOO_bar`
Number of Words	Identifiers should be composed of between two and four words.	`ArrayOutOfBoundsException, name`
Numeric Identifier Name	Identifiers should not be composed entirely of numeric words or numbers	`FORTY_TWO`
Short Identifier Name	Identifiers should not consist of fewer than eight characters, with the exception of: c, d, e, g, i, in, inOut, j, k, m, n, o, out, t, x, y, z	`name`

To clarify, the ‘Capitalization Anomaly’ states that regardless of the abbreviation, the capitalization should only be used as a substitute for white-space between words. The ‘Identifier Encoding’ metric state that the type, such as integer, double or string, should not be used as a Hungarian-style prefix, whereas the *kind* can be indicated this way, the detailed reasoning here is explained in (Spolsky 2005).

The quality measures will sometimes conflict, and some trade-offs has to be accepted. Most importantly is the conflict between the two metrics ‘Dictionary Words’ and ‘Naming Convention Anomaly’ opposed the metrics ‘Number of Words’ and ‘Long Identifier Name’.

Dictionary words or common abbreviations does not exist for some of our variables so that shortening the identifier means losing the self-explaining property of the longer name. In these cases, ‘Dictionary Words’ are prioritized because it maintains the quality that every line should be as self-explaining as possible.

This priority is justified as the DST usually make simple calculations. Nnumeric estimates, discretization or recursions make the program-counter move unpredictably and demands more on the code being structured to represent the program flow. Whereas in the DST mostly practical meaning of simple arithmetic needs to be understood. This is best expressed in single line calculations that can be understood alone, not depending on a larger logical structure. This relieves the need of a compactly expressed code, and makes the use of ‘Dictionary Words’ a higher priority.

This choice creates a self-documenting effect, the code has a narrative that explain its functionality. However, few cases of variable names longer than 4 words (excluded class prefix) was necessary.

The complete list of name changes are displayed in Table 3. All variables are associated with a class unless encapsulated because of strictly local relevance. The full class names are prefixing the variables in the table below, in the code these are often abbreviated.

The unit of variables is included in some names, in order to easily detect errors when writing the code. When the unit of the variable is obvious, for example when discussing power or load, the unit is implicit and can be excluded.

The variables `iPv` and `jBatt` are important to notice. During simulation we have for loops for PV `iPv = pvStartKw : pvStepKw : pvStopKw`, for battery `jBatt = battStartKwh : battStepKwh : battStopKwh`, and time `t = 1 : nHours`. When accessing data points later we use the same variable name as a place holder. Meaning that if we have a `pvStartKw = 100` and `battStartKw = 100`, the simulation outputs for LoL can be accessed as `SimOut.lossOfLoad(:,1,1)`, here `iPv = 1`, and `jBatt = 1`. This way of referring to simulations is recurring in throughout the DST.

Table 3: the Complete List of Variable Name Changes in Rewritten DST

Names	
Logplot.m	**After Rewrite**
batt_balance	SimulationOutputs.neededBattOutputKw**
batt_balance	SimulationOutputs.neededBattOutputKw***
batt_i	jBatt
batt_ratio	BatteryParameters.powerEnergyRatio
budget	EconomicParameters.budget
coeff_cost_BoSeI	EconomicParameters.installBalanceOfSystemCost
coeff_T_pow	PvParameters.powerDearteDueTemperature
costBatt_coef_b	EconomicParameters.battCostFixed
costBatt_coeff_a	EconomicParameters.battCostKwh
costBatt_tot	EconomicAnalysisOutputs.battCostTot**
costBoSeI_tot	EconomicAnalysisOutputs.installBalanceOfSystemTotCo st
costINV	EconomicParameters.inverterCostKw
costINV_tot	EconomicAnalysisOutputs.inverterCostTot
costOeM	operationMaintenanceCost
costOeM_spec	EconomicParameters.operationMaintenanceCostKw
costPV	EconomicParameters.pvCostKw
CRF	EconomicAnalysisOutputs.capitalRecoveryFactor
cycles_failure	SimulationOutputs.sumPartialCyclesUsed**
Den_rainflow	nMaxPartialCycles
DoD	depthOfDischarge
eff_BoS	EconomicParameters.balanceOfSystem
eff_cell	cellEfficiency
eff_char	BatteryParameters.chargingEfficiency
eff_disch	BatteryParameters.dischargingEfficiency
eff_inv	InverterParameters.efficiency
ELPV	SimulationOutputs.pvPowerAbsorbedUnused
EPV	(deprecated)
filename	(deprecated)
flow_from_batt	SimulationOutputs.battOutputKw**
IC	EconomicAnalysisOutput.investmentCost
irr	SimulationInputData.irradiation
irr_nom	PvParameters.nominalIrradiation
LCoE	EconomicAnalysisOutputs.levelizedCostOfEnergy
LL	SimulationOutputs.lossOfLoad
LLP	SimulationOutputs.lossOfLoadProbability
Load	SimulationInputData.load
LT	EconomicParameters.plantLifetimeYears
MA_opt_norm_bhut_jun 15_20_10(1)	OptimalSolutions.lossOfLoadProbabilities
MA_opt_norm_bhut_jun 15_20_10(2)	OptimalSolutions..netPresentCosts
MA_opt_norm_bhut_jun 15_20_10(3)	OptimalSolutions.pvKw
MA_opt_norm_bhut_jun 15_20_10(4)	OptimalSolutions.battKwh
MA_opt_norm_bhut_jun 15_20_10(5)	OptimalSolutions.levelizedCostsOfEnergy
MA_opt_norm_bhut_jun 15_20_10(6)	OptimalSolutions.investmentCosts
max_batt	SimulationParameters.battStopKwh
max_PV	SimulationParameters.pvStopKw
max_y_repl	BatteryParameters.maxOperationalYears
min_batt	SimulationParameters.battStartKwh
min_PV	SimulationParameters.pvStartKw

`n_batt`	`SimulationParameters.nBattSteps`
`n_PV`	`SimulationParameters.nPvSteps`
`NPC`	`EconomicAnalysisOutputs.netPresentCost`
`num_batt`	`EconomicAnalysisOutput.nBattEmployed`
`P_pv`	`SimulationOutputs.pvPowerAbsorbed****`
`peak`	`loadPeakKw`
`Pow_max`	`battMaxPowerFlow`
`PV_i`	`iPv`
`Pvpower_i`	`iPvKw`
`r_int`	`EconomicParameters.interestRate`
`SoC`	`SimulationOutputs.stateOfCharge**`
`SoC_min`	`BatteryParameters.minStateOfCharge`
`SoC_start`	`BatteryParameters.initialStateOfCharge`
`step_batt`	`SimulationParameters.battStepKwh`
`step_PV`	`SimulationParameters.pvStepKw`
`T_amb`	`SimulationInputData.temperatureC`
`T_cell`	`pvTemperatureC`
`T_nom`	`PvParameters.nominalCellTemperatureC`
`T_ref`	`PvParameters.nominalAmbientTemperatureC`
`total_loss_load`	`SimulationOutputs.lossOfLoadTot**`
`x_llp`	`SimulationParameters.llpSearchTargets`
`YC`	`EconomicAnalysisOutput.operationMaintenanceReplacementCost**`
`years_to_go_batt`	`battOperationalYears`
Stored for after-simulation inspections in the rewritten DST	
* Hourly data points are now stored in this variable	
** PV and battery combination data points are now stored in this variable	
*** Both time and PV/battery combinations are now stored in this variable.	
**** Time and PV iterations are now stored in this variable	

2.1.1.1 Classes in the rewritten DST

Classes has some great advantages as they allow for generation of variables automatically at initiation, hiding this from the developer.

The class `SimulationInputData` will for example only need the names of the files, and the folder containing them for initiation, creating a class containing the data sets with the respective file names. Classes can also generate warnings, for example when vectors are of different lengths, and needs preprocessing.

The `SimulationInputData` constructor is displayed in Figure 2:1 as an example. The constructor loads the data-sets into the class independent of the operative system currently running. This makes for a much more flexible program.

```
function obj = SimulationInputData(loadProfileFilename,...
                                   irradiationFilename,...
                                   temperatureFilename,...
                                   folderName)
    if nargin > 0
        obj.folderName = folderName;
        obj.databasePath = get_path_to_database_folder(folderName);

        obj.loadProfileFilename = loadProfileFilename;
        obj.irradiationFilename = irradiationFilename;
        obj.temperatureFilename = temperatureFilename;

        obj.load = importdata([obj.databasePath...
                               obj.loadProfileFilename]);

        obj.irradiation = importdata([obj.databasePath...
                                      obj.irradiationFilename]);

        obj.temperature = importdata([obj.databasePath...
                                      obj.temperatureFilename]);

        if (length(obj.load) == length(obj.irradiation))...
        && (length(obj.load) == length(obj.temperature))
            obj.nHours = length(obj.temperature);
        else
            fprintf(['Warning: the data sets\n\t- irradiation data\n\t-',...
                     'ambience temperature data\n\t- load profile data \n',...
                     'are NOT the same size'])
        end

        obj.nYears = obj.nHours / 8760;
    end
end
```

Figure 2:1 The SimulationInputData constructor function

Compared to structs, class implementations will prevent assignment and access of variables that are not predefined for the class. The attempt of accessing `SimData.x` returns: "`The class SimulationInputData has no property or method named 'x'`", even though this class instance was shortened to SimData, creating a self-documenting effect .

Lastly, the class will make the origin of the variables explicit, meaning that there is less doubt of what is meant by `BatteryParameters.chargingEfficiency`. This will assist the comprehension of the code.

An overview of how the classes work between modules can be seen in Figure 2:2.

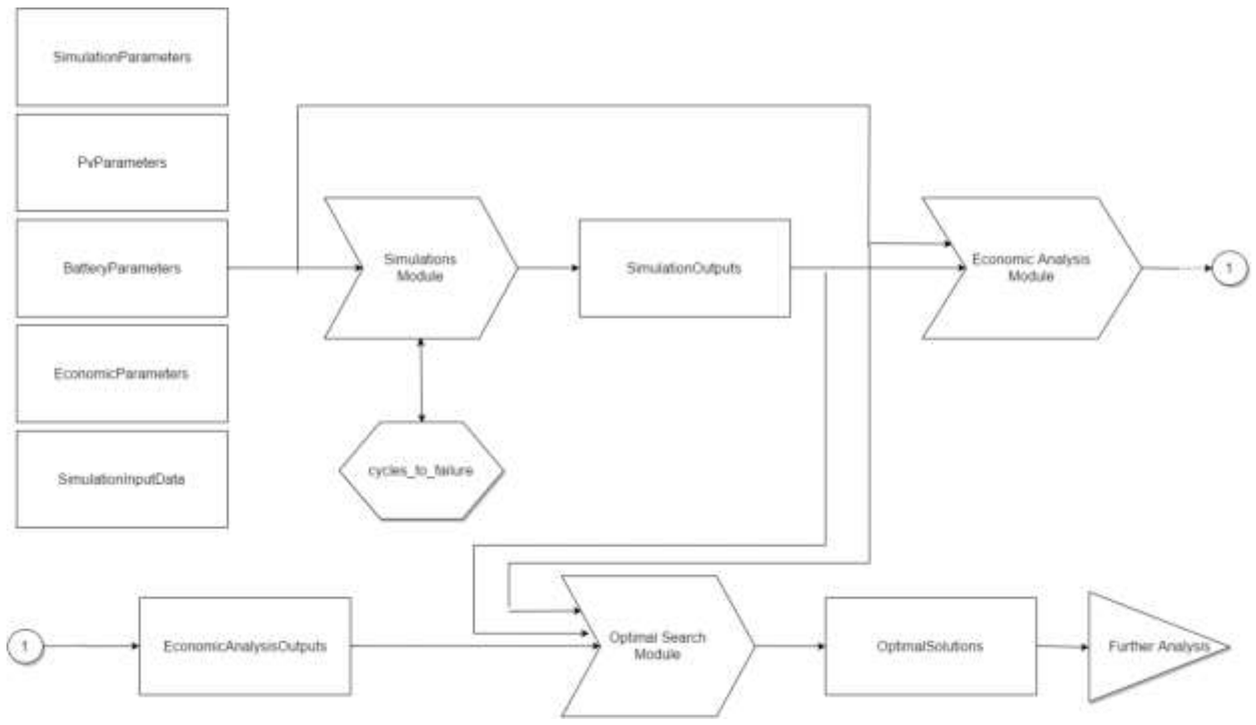


Figure 2:2: Class and Module Diagram.

2.1.2 Modularization and Encapsulation

Encapsulation means that variables are restricted to their scope, they are not available from other functions and they are not stored when function scopes are terminated. This will protect variables from unwanted access or interference, making implementations safer. Encapsulation is mainly used to keep the workspace clean and comprehensive in the DST, all variables that are irrelevant outside the functions are deleted.

In the DST each module has an associated output class, the result is a workspace with only input and output classes (See Figure 2:2: Class and Module Diagram). This defines a clear intersection between modules, helping developers to understand the effect of changes. This also means that modules can be used in any order desired, or even used inside each other. A single simulation can be called by restricting the simulation parameters to one vector, the plotting function can be repeated multiple times with different solutions.

This configuration allows for agile development and testing, as modules can be changed without changing the rest of the code. If you wish to plot a solution, it will be enough to pass a few classes as arguments to the plotting function instead of 20 different variables. If you wish to modify an existing module, it can simply be copied and modified

It has been said about quality code that it “does one thing well” (Martin 2009), this cannot be followed too strictly in the DST Matlab implementation. The reason being that the data-sets are too large to be frequently passed between functions. Function calls in Matlab are ‘call by value’, meaning that a variable passed to a function is copied to the function scope. The potential memory consumption and computation overhead while doing this, could affect the development and testing flow. Hence, further modularization is reserved for a python implementation, where variables are pointers and ‘call by reference’ is default behavior.

2.1.3 Formatting

We discussed earlier that identifiers has become longer, this might lead to visual clutter that makes code harder to read. The length of the variables might in some development environments require that the programmer scrolls horizontally, which can be very disruptive to the process.

As mentioned earlier, the DST performs very straight forward arithmetic operations. The solution is to simply arrange all equation steps in columns, consistently aligned by operators as seen in Figure 2:3. The idea is that the user quickly expects to see the equations in this format, making it predictable and easily read.

This is additionally necessary because of the wide formatting native to Matlab. This format might be unnecessary for readability in later versions, as the Matlab editor use a wide font and is unable to zoom out.

```
% in-flow exceeds the battery power limit
if (abs(neededBattOutputKw(t,iPv))) > battMaxPowerFlow...
&& stateOfCharge(t,iPv,jBatt) < 1

    battOutputKw(t, iPv, jBatt) = battMaxPowerFlow ...
                                * BattParam.chargingEfficiency;

    inputPowerUnusedKw(t,iPv, jBatt) ...
                                = inputPowerUnusedKw(t,iPv, jBatt) ...
                                + (abs(neededBattOutputKw(t,iPv))...
                                - battMaxPowerFlow);
end

stateOfCharge(t+1,iPv,jBatt) = stateOfCharge(t,iPv,jBatt) ...
                             + abs(battOutputKw(t, iPv, jBatt)) ...
                             / jBattKwh;

if stateOfCharge(t+1,iPv,jBatt) > 1

    inputPowerUnusedKw(t,iPv, jBatt) ...
                              = inputPowerUnusedKw(t,iPv, jBatt)...
                              + (stateOfCharge(t+1,iPv,jBatt) - 1) ...
                              * jBattKwh ...
                              / BattParam.chargingEfficiency;

    stateOfCharge(t+1,iPv,jBatt) = 1;

end
```

Figure 2:3Rewritten DST formatting example

2.1.4 Evaluation of Finished Rewrite

In order to compare the DST and logplot.m, some bugs had to be replicated. The bugged module is named `bugged_economic_analysis` and prints a warning when run.

2.1.4.1 Correctness

The function `isequal(A,B)` will compare every element in two matrices, and returns `true` (1) if the matrices are identical, and `false` (0) if there are one or more element with any kind of difference.

Given an input with large span and high resolution, there will be enough data points to safely state that the DST conserve the functionality of logplot.m. We assume that $P_{dataPointIsEqual\_i}(true|coincidence)$, the probability that two data points are equal despite an erroneous implementation. We also assume that these probabilities are independent of each other, and if they should be independent, the implementation would be correct. The probability of two matrices instead of data-points being equal, can then be described as follows.

$$(7) \qquad P_{matricesAreEqual}(true \cap coincidence) = \prod_{i=1}^{nDataPoints} P_{dataPointIsEqual\_i}(true \cap coincidence)$$

One last assumption is that the average probability that two data points are equal but not because the implementation is correct, $\overline{P_{dataPoint\_i}(true \cap coincidence)}$ is not larger 50%. We wish to be at least 99.999% sure that our implementation is correct. We will use this assumption to state that consecutive correct answers despite of a wrong implementation is $P_{matricesAreEqual}(true \cap coincidence) < 0.5^{nDataPoints} \approx 0.00001$, and consequently the minimum required number of data points are:

$$(8) \qquad nDataPoints = \frac{log(0.00001)}{log(0.5)} = 16.6$$

The number of data points produced by different modules is displayed in Table 4

Table 4: Complexity and output source of compared simulation pair outputs

Variable Names			
Before Rewrite	**After Rewrite**	**Module**	**n Data Points**
`SoC`	`stateOfCharge`	`sapv_plant_simulation`	$nPv \times nBatt \times nHours$
`LL`	`lossOfLoad`	`sapv_plant_simulation`	$nPv \times nBatt \times nHours$
`NPC`	`netPresentCost`	`economic_analysis`	$nPv \times nBatt$
`LCoE`	`levelizedCostOfEnergy`	`economic_analysis`	$nPv \times nBatt$
`MA_opt_norm_bhut_jun15_20_10(:,1)`	`lossOfLoadProbabilities`	`llp_constrained_optimum`	$nOptSolutions(\in (nPv \times nBatt) \mid givenLlpRange)$
`MA_opt_norm_bhut_jun15_20_10(:,3:4)`	`[pvKw, battKwh]`	`llp_constrained_optimum`	$nOptSolutions(\in (nPv \times nBatt) \mid givenLlpRange)$

Logplot.m is rewritten to save more variables to workspace so that we may compare the two programs. As a result of the arguments above, it is assumed that $4 \times 4 = 16$ correct calculations are a result of a correctly rewritten DST, with more than 99.99% certainty. Some redundancy is introduced to compensate possible poor assumptions.

Two simulation pairs were executed. The first simulation had $21 \times 21 = 441$ PV and battery combinations, and 15 optimal solutions. The second simulation pair span $25 \times 25 = 625$ PV and battery combinations. The two comparisons are displayed below.

```
% Paramaters for Simulation Solution Space
SimParameters = SimulationParameters;
SimParameters.pvStartKw = 100;
SimParameters.pvStopKw = 200;
SimParameters.pvStepKw = 5;
SimParameters.battStartKwh = 1200;
SimParameters.battStopKwh = 1300;
SimParameters.battStepKwh = 5;
SimParameters.llpSearchAcceptance = 0.005;
SimParameters.llpSearchTargets = 0.01:0.005:0.30;
```

Figure 2:4: Parameters for simulations comparison outputs, simulation pair 1.

Table 5: Results of comparisons in simulation pair 1

Output A	**Output B**	**n Data Points**	**Isequal(A,B)**
SoC	stateOfCharge	3863160	True
LL	lossOfLoad	3863160	True
NPC	netPresentCost	441	True
LCoE	levelizedCostOfEnergy	441	True
MA_opt_norm_bhut_jun15_20_10(:,1)	lossOfLoadProbabilities	15	True
MA_opt_norm_bhut_jun15_20_10(:,3:4)	pvKw, battKwh	15	True

```
% Paramaters for Simulation Solution Space
SimParameters = SimulationParameters;
SimParameters.pvStartKw = 150;
SimParameters.pvStopKw = 170;
SimParameters.pvStepKw = 5;
SimParameters.battStartKwh = 1150;
SimParameters.battStopKwh = 1270;
SimParameters.battStepKwh = 5;
SimParameters.llpSearchAcceptance = 0.005;
SimParameters.llpSearchTargets = 0.10:0.005:0.80;
```

Figure 2:5: Parameters for simulations comparison outputs, simulation pair 2.

Table 6: Results of comparisons in simulation pair 2

Input A	**Input B**	**n Data Points**	**Isequal(A,B)**
SoC	stateOfCharge	183960	True
LL	lossOfLoad	183960	True
NPC	netPresentCost	625	True
LCoE	levelizedCostOfEnergy	625	True
MA_opt_norm_bhut_jun15_20_10(:,1)	lossOfLoadProbabilities	53	True
MA_opt_norm_bhut_jun15_20_10(:,3:4)	pvKw, battKwh	53	True

As seen in Table 5 and Table 6, the output matrices are equal. We hereby accept that the DST is correctly rewritten.

2.1.4.2 Computation Speed

The DST does not have increased complexity, but many new data-structures and several large memory operations are introduced to realize the new structure, it is therefore important to monitor if Matlab function in the new structure is a source of increased computation time.

The ‘Profiler’ timing tool was used to evaluate the rewritten DST, the tool will reveal how much time is spent on calculations, and on waiting for other functions to terminate, this is important because we want the increased computation time to be ‘visible in the code’.

The profiler tool output is displayed in Table 7 and Table 8. Increased self-time means that Matlab does not call many additional help functions. The main function in Table 8 has a large bright blue band, because it calls the other module functions. The important take away is that the increased computation time is in ‘Self Time’, and that native functions does not introduce increased complexity.

Table 7:The Profiler Tool run on the previous DST. Only the 7 top functions are displayed

Function Name	**Calls**	**Total Time**	**Self Time***	**Total Time Plot (dark band = self time)**
logplot	1	4.063 s	3.986 s	▬▬▬▬▬
cycles_to_failure	36500	0.046 s	0.046 s	▏
finfo	3	0.031 s	0.015 s	▏
importdata	3	0.031 s	0.000 s	▏
matfinfo>matfinfosub	3	0.016 s	0.000 s	▏
mat2str	3	0.016 s	0.016 s	▏
matfinfo	3	0.016 s	0.000 s	▏

Table 8: The Profiler Tool Run on the rewritten DST. Only the 7 top functions are displayed

Function Name	**Calls**	**Total Time**	**Self Time***	**Total Time Plot (dark band = self time)**
main	1	5.761 s	0.000 s	▬▬▬▬▬
sapv_plant_simulation	1	5.698 s	5.651 s	▬▬▬▬▬
cycles_to_failure	36500	0.047 s	0.047 s	▏
importdata	3	0.032 s	0.016 s	▏
...mulationInputData.SimulationInputData	1	0.032 s	0.000 s	▏
bugged_economic_analysis	1	0.031 s	0.031 s	▏
fileparts	6	0.016 s	0.016 s	▏

The author believes memory access and allocation overhead is the reason for the increased computation time, this can be reduced by implementing call-by-reference instead of call-by-value modules.

Matlab use both ‘handle classes’ and ‘value classes’ which correspond to call-by-reference and call-by-value respectively. The decision was made to keep the value class implementation because is likely to be easier to debug. Pointer passing tend to produce more cryptic errors, hence complicating development.

The overhead in the rewrite is acceptable because of the added modularity and agile structure of the DST. The DST will only repeat simulations to refine the simulation space. Logplot.m had to repeat simulations for a number of reasons such as plotting and polling.

It is possible to shorten the waiting time by defining frequently called functions inline in the script, rather than in a separate file. This was tested with the `cycles_to_failure` function but the improvement was less than 0.1% so the function was kept external to maintain modularity.

2.2 The Graphical User Interface

The Graphical User Interface (GUI) programing in Matlab involves initiating different uicontrols. These are objects that are constructed by setting their type, appearance, position and call-function.

The call-function is called every time the object is interacted with. An example of a uicontrol and callback function initiation can be seen Figure 2:6 and Figure 2:7 respectively.

```
h.RunOptSolBtn = uicontrol(f, 'Style', 'pushbutton',...
                              'String' , 'Find Optimal Solutions',...
                              'Units', 'pixels',...
                              'Position', [lPresetX,
                              lFindOptimalSolutionBtnY,...
                              lPresetWidth, lRunSingleBtnHeight],...
                              'Callback', @runOptSol_Callback,...
                              'BackgroundColor',[179 204 255]/255);
```

Figure 2:6 The uicontrol initiation for the 'Run Optimal Solutions' button

```
function runOptSol_Callback(~, ~)

    disp('Optimum Solution Finder Running...')

    runbutton_pressed('opt')

end
```

Figure 2:7 The call-function when pressing the 'Run Optimal Solutions' button

2.2.1 dst_platform.m

The `dst_platform` GUI is designed to be the start platform for the user. The main function of the GUI is to initiate the parameter classes, and pass these classes to different modules depending on the run-settings, also set in the GUI. The GUI is displayed in Figure 2:8. The units on biomass cost and biomass delivered are in kW, these have been corrected to kWh in the current version.

The GUI can save presets, meaning that any combination of inputs and settings to the `dst_platform` can be saved given a name, and loaded when it chosen in the dropdown menu. This will allow the user to inspect the differences between different input parameters, and retrace old settings at a later time. Saving is done by entering a name for the current preset and pressing ‘Save’, this will overwrite any preset with the same name. Pressing ‘Delete’ will delete the currently chosen preset in the dropdown menu.

If no preset is found in the presets folder under the GUI folder, the file get_dst_set_default.m will initiate the parameters and create a default file.

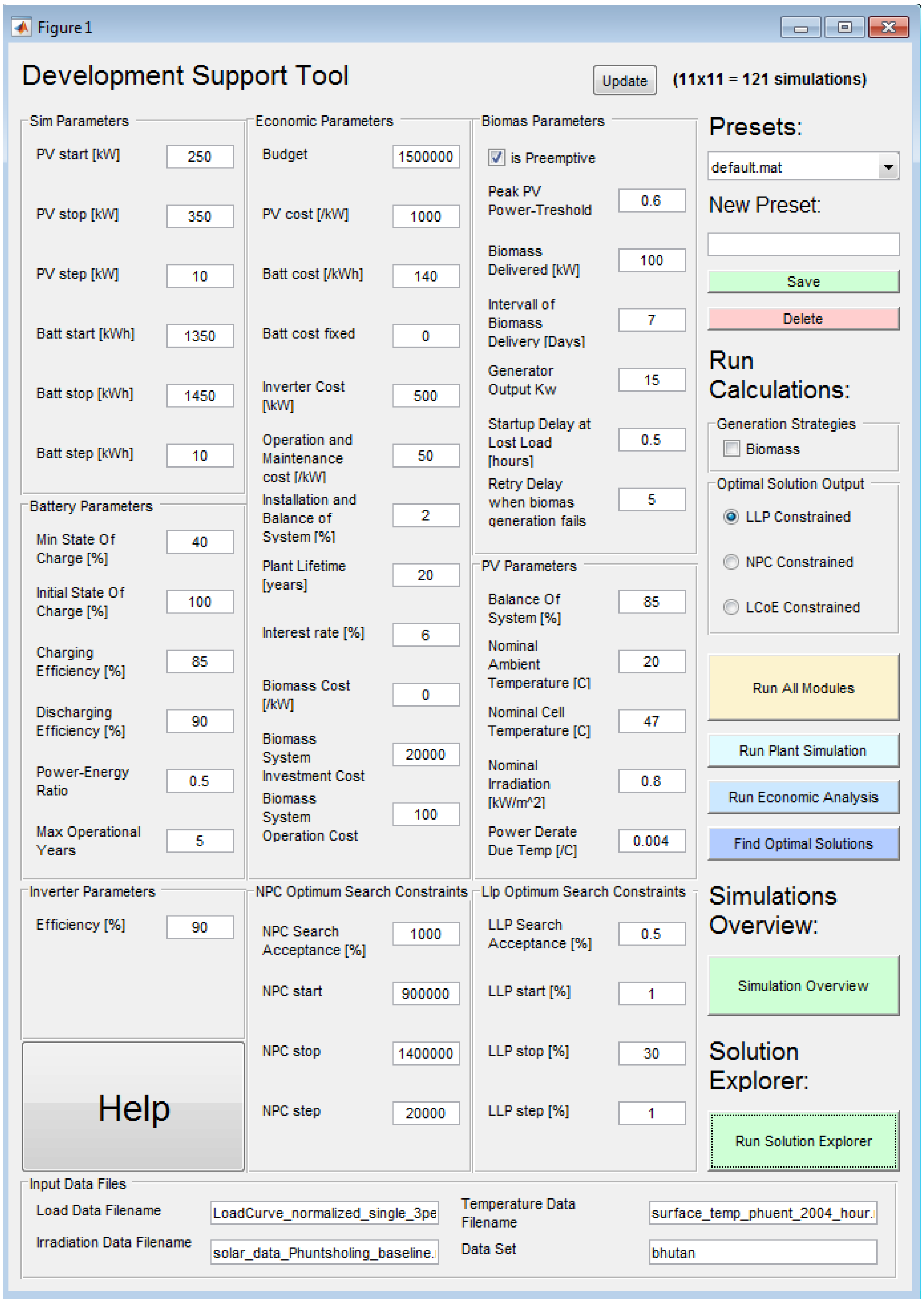


Figure 1

Development Support Tool

Update (11x11 = 121 simulations)

Sim Parameters

PV start [kW]	250
PV stop [kW]	350
PV step [kW]	10
Batt start [kWh]	1350
Batt stop [kWh]	1450
Batt step [kWh]	10

Battery Parameters

Min State Of Charge [%]	40
Initial State Of Charge [%]	100
Charging Efficiency [%]	85
Discharging Efficiency [%]	90
Power-Energy Ratio	0.5
Max Operational Years	5

Inverter Parameters

Efficiency [%]	90

Help

Economic Parameters

Budget	1500000
PV cost [/kW]	1000
Batt cost [/kWh]	140
Batt cost fixed	0
Inverter Cost [\kW]	500
Operation and Maintenance cost [/kW]	50
Installation and Balance of System [%]	2
Plant Lifetime [years]	20
Interest rate [%]	6
Biomass Cost [/kW]	0
Biomass System Investment Cost	20000
Biomass System Operation Cost	100

NPC Optimum Search Constraints

NPC Search Acceptance [%]	1000
NPC start	900000
NPC stop	1400000
NPC step	20000

Biomas Parameters

[✓ is Preemptive]	
Peak PV Power-Treshold	0.6
Biomass Delivered [kW]	100
Intervall of Biomass Delivery [Days]	7
Generator Output Kw	15
Startup Delay at Lost Load [hours]	0.5
Retry Delay when biomas generation fails	5

PV Parameters

Balance Of System [%]	85
Nominal Ambient Temperature [C]	20
Nominal Cell Temperature [C]	47
Nominal Irradiation [kW/m^2]	0.8
Power Derate Due Temp [/C]	0.004

Llp Optimum Search Constraints

LLP Search Acceptance [%]	0.5
LLP start [%]	1
LLP stop [%]	30
LLP step [%]	1

Presets:

default.mat

New Preset:

Save

Delete

Run Calculations:

Generation Strategies

☐ Biomass

Optimal Solution Output

[✓ LLP Constrained]

NPC Constrained

LCoE Constrained

Run All Modules

Run Plant Simulation

Run Economic Analysis

Find Optimal Solutions

Simulations Overview:

Simulation Overview

Solution Explorer:

Run Solution Explorer

Input Data Files

Load Data Filename	LoadCurve_normalized_single_3pe	Temperature Data Filename	surface_temp_phuent_2004_hour.
Irradiation Data Filename	solar_data_Phuntsholing_baseline.	Data Set	bhutan

Figure 2:8: the DST Platform GUI

The `dst_platform` has plotting functionality to get a comprehensive overview of the simulation output space. This plot will appear when pressing the ‘Simulation Overview’ button. A detailed description can be found in section 2.2.3.1. The ‘Help’ button will open a browser window with the online documentation of the DST, this is described in section 2.3

2.2.2 solution_explorer.m

The solution explorer is designed for inspecting and understanding the results from the DST, by allowing the user to compare different solutions. Outputs are categorized as ‘General Info’, ‘Averages’, ‘Worst Case’ and ‘Biomass’, they produce important statistics to understand the implications of a simulation result.

In the panel beneath ‘Pick a Solution to Examine’ in Figure 2:9, the Solution Explorer lists all the solutions stored in the variable ‘`OptSol`’ which is an instance of the Optimal Solutions class in the workspace. The list is described by a solution number, NPC, LCoE and LLP. Each solution has indexes in the simulation-space output matrices, normally referred to as `iPv` and `jBatt` in the code context, in the output windows they are referred to as ‘PV iteration’ and ‘Battery iteration’.

Once a solution is highlighted, the different buttons will produce outputs for examining the current choice. In Figure 2:20 solution 2 with NPC = 1791452 is picked, if the user clicks any of the buttons below the output panels, different outputs will be displayed.

The radio toggle buttons under the Data Analysis Output windows decide which window the output should appear on. If the ‘Output In New Window’ option is chosen, outputs will appear in a new floating windows. This choice will allow for more than two windows to be compared simultaneously. Plots will always appear in new floating windows (plot default behavior).

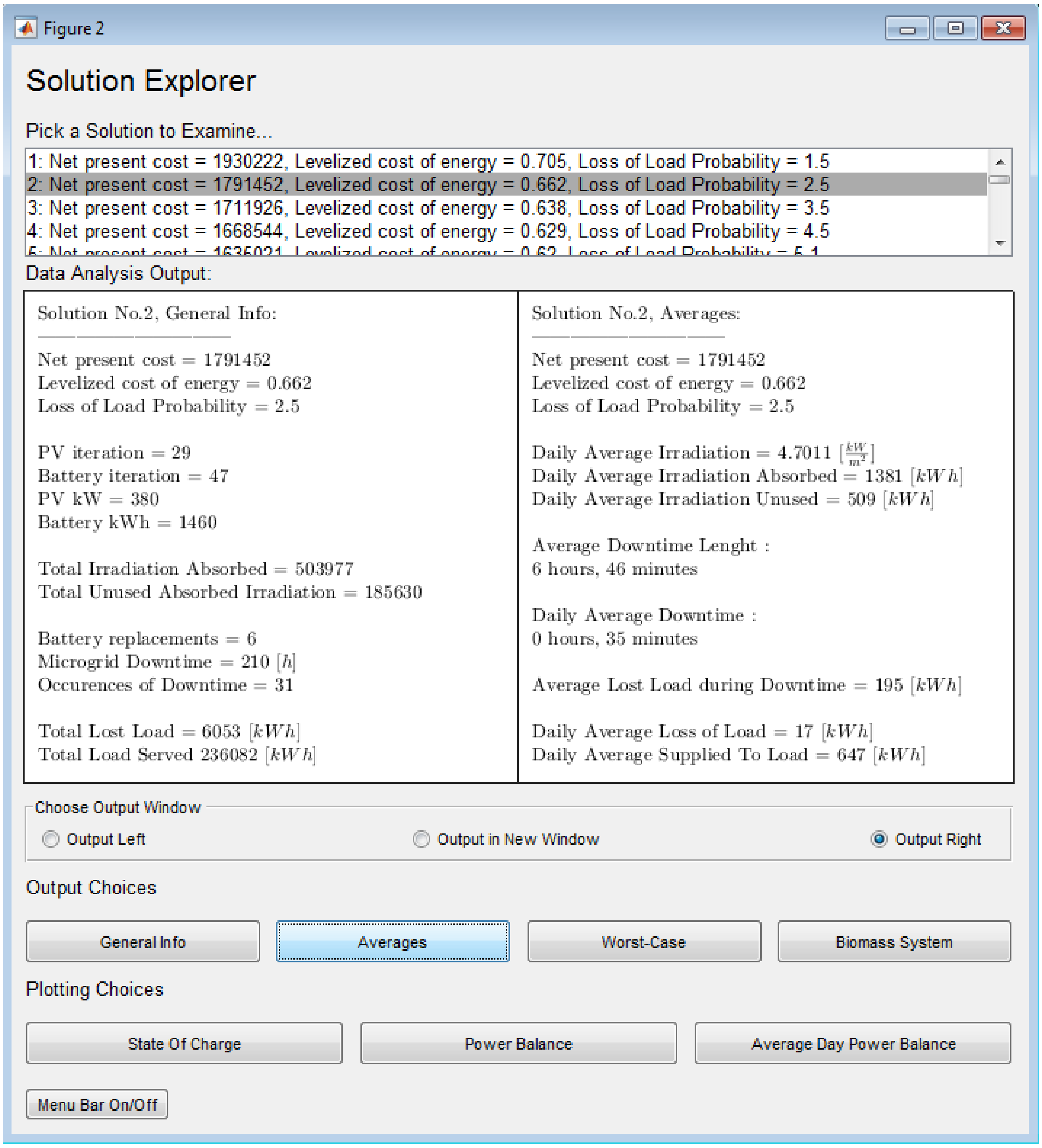


Figure 2

Solution Explorer

Pick a Solution to Examine...

1: Net present cost = 1930222, Levelized cost of energy = 0.705, Loss of Load Probability = 1.5
2: Net present cost = 1791452, Levelized cost of energy = 0.662, Loss of Load Probability = 2.5
3: Net present cost = 1711926, Levelized cost of energy = 0.638, Loss of Load Probability = 3.5
4: Net present cost = 1668544, Levelized cost of energy = 0.629, Loss of Load Probability = 4.5
5: Net present cost = 1636021, Levelized cost of energy = 0.62, Loss of Load Probability = 5.1

Data Analysis Output:

```
Solution No.2, General Info:
————————————
Net present cost = 1791452
Levelized cost of energy = 0.662
Loss of Load Probability = 2.5

PV iteration = 29
Battery iteration = 47
PV kW = 380
Battery kWh = 1460

Total Irradiation Absorbed = 503977
Total Unused Absorbed Irradiation = 185630

Battery replacements = 6
Microgrid Downtime = 210 [h]
Occurences of Downtime = 31

Total Lost Load = 6053 [kWh]
Total Load Served 236082 [kWh]
```

```
Solution No.2, Averages:
————————————
Net present cost = 1791452
Levelized cost of energy = 0.662
Loss of Load Probability = 2.5

Daily Average Irradiation = 4.7011 [kW/m^2]
Daily Average Irradiation Absorbed = 1381 [kWh]
Daily Average Irradiation Unused = 509 [kWh]

Average Downtime Lenght :
6 hours, 46 minutes

Daily Average Downtime :
0 hours, 35 minutes

Average Lost Load during Downtime = 195 [kWh]

Daily Average Loss of Load = 17 [kWh]
Daily Average Supplied To Load = 647 [kWh]
```

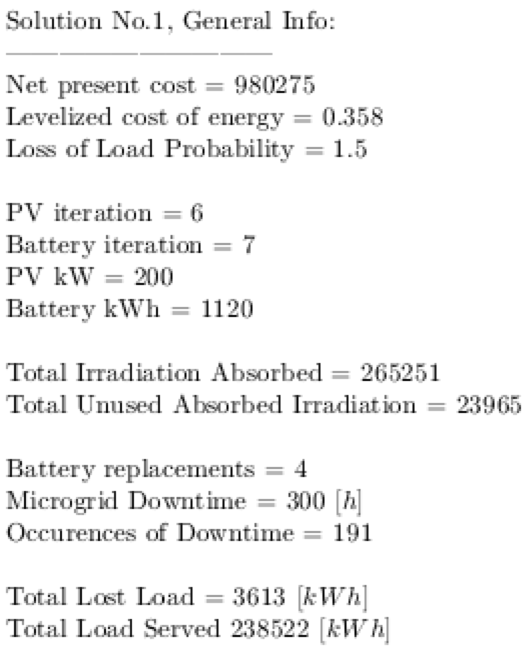
Choose Output Window: Output Left, Output in New Window, [✓ Output Right]

Figure 2:9 The Solution Explorer

2.2.2.1 Available Outputs to the Solution Explorer

The different buttons will output different kinds of information. LLP alone is not always enough to understand how a system operates, especially when seasonal variations are considered. For example, LLP can originate from a few occurrences of lost load, or from a daily regular power-shortage of small magnitude. Frequent power loss is typically considered a worse alternative. A few lost load occurrences of high magnitude might be more acceptable.

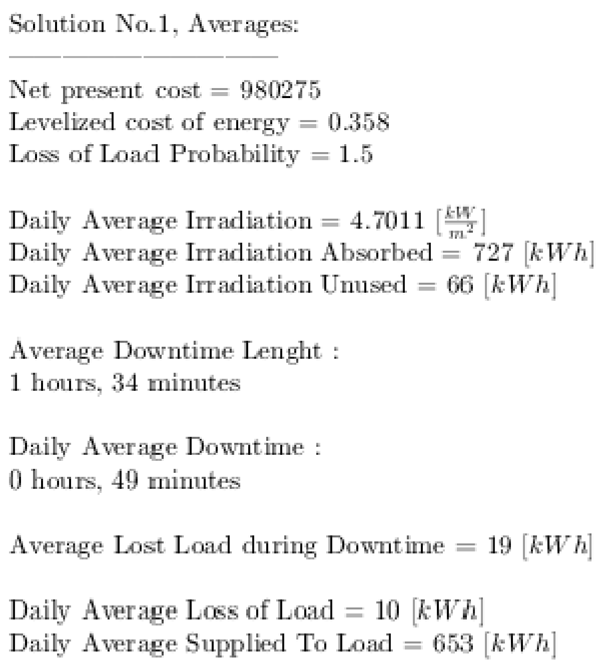
'General Info' serves as an overview of general characteristics of the system. It can be used to recognize which solution you are dealing with, or as a coarse inspection before moving on to details or the next solution.



```
Solution No.1, General Info:
--------------------------------
Net present cost = 980275
Levelized cost of energy = 0.358
Loss of Load Probability = 1.5

PV iteration = 6
Battery iteration = 7
PV kW = 200
Battery kWh = 1120

Total Irradiation Absorbed = 265251
Total Unused Absorbed Irradiation = 23965

Battery replacements = 4
Microgrid Downtime = 300 [h]
Occurences of Downtime = 191

Total Lost Load = 3613 [kWh]
Total Load Served 238522 [kWh]
```

Figure 2:10 The General Info output window

'Averages' are for understanding of the general performance of the system. It will also reveal details of what kind of power shortages are occurring and if an additional generator might solve the problem.



Solution No.1, Averages:
———————————

Net present cost = 980275
Levelized cost of energy = 0.358
Loss of Load Probability = 1.5

Daily Average Irradiation = 4.7011 $[\frac{kW}{m^2}]$
Daily Average Irradiation Absorbed = 727 $[kWh]$
Daily Average Irradiation Unused = 66 $[kWh]$

Average Downtime Lenght :
1 hours, 34 minutes

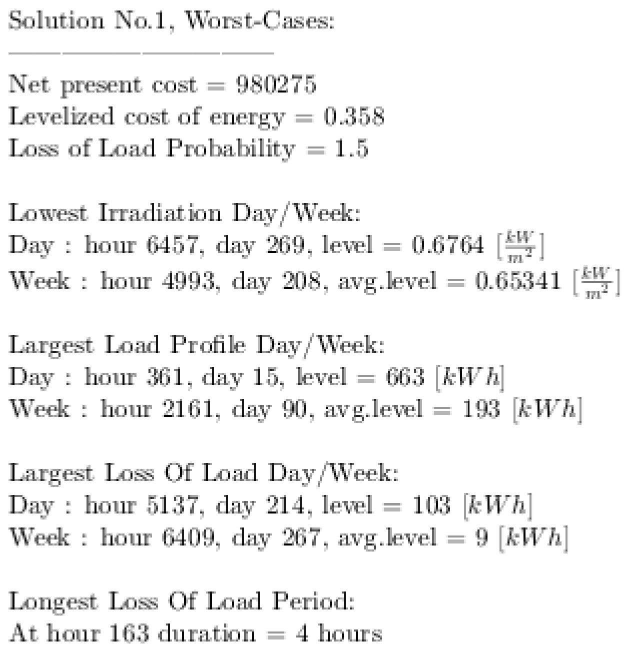
Daily Average Downtime :
0 hours, 49 minutes

Average Lost Load during Downtime = 19 $[kWh]$

Daily Average Loss of Load = 10 $[kWh]$
Daily Average Supplied To Load = 653 $[kWh]$

Figure 2:11 The Averages output window

'Worst-Case' sums up the different worst-case scenarios of a system. Since these are associated with specific instances, the time of the occurrence accompany the data, the worst cases might for example be clustered around the same period. This output is especially useful combined with the Averages output, if the worst case and average are close, you are considering a fairly stable operation, and vice versa.



Solution No.1, Worst-Cases:
———————————

Net present cost = 980275
Levelized cost of energy = 0.358
Loss of Load Probability = 1.5

Lowest Irradiation Day/Week:
Day : hour 6457, day 269, level = 0.6764 $[\frac{kW}{m^2}]$
Week : hour 4993, day 208, avg.level = 0.65341 $[\frac{kW}{m^2}]$

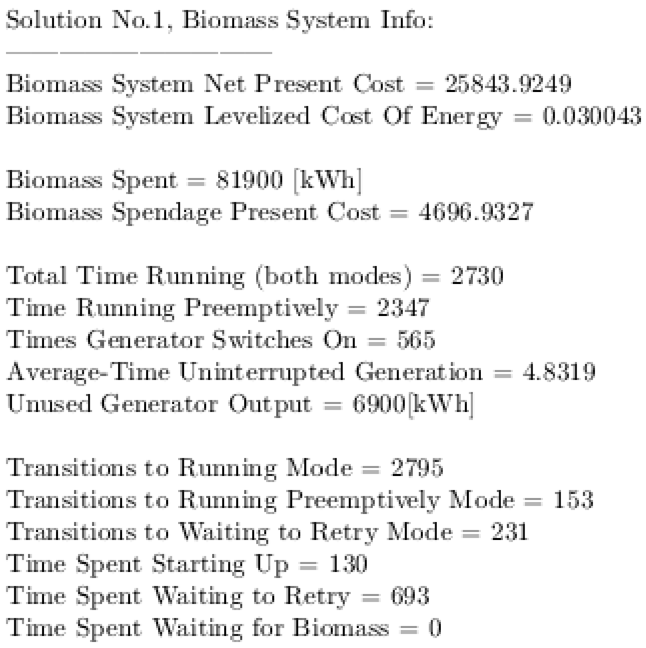
Largest Load Profile Day/Week:
Day : hour 361, day 15, level = 663 $[kWh]$
Week : hour 2161, day 90, avg.level = 193 $[kWh]$

Largest Loss Of Load Day/Week:
Day : hour 5137, day 214, level = 103 $[kWh]$
Week : hour 6409, day 267, avg.level = 9 $[kWh]$

Longest Loss Of Load Period:
At hour 163 duration = 4 hours

Figure 2:12 The Worst-Case output Window

‘Biomass-System’ delivers an operational and economic overview. The LCoE of the biomass system is important to evaluate the benefit from purchasing the system. The running times and transitions communicates how well the system is scaled, much time spent in ‘Waiting To Retry’, or high values of unused generator output, indicates that the generators kW output is too small to handle the LoL magnitudes. The output window also learns about the potential wear and tear and operational effort through the ‘Times Generator Switches On’ and ‘Transitions to Waiting to Retry Mode’, and about the biomass supply, whether it is sufficient.



```
Solution No.1, Biomass System Info:
————————————————

Biomass System Net Present Cost = 25843.9249
Biomass System Levelized Cost Of Energy = 0.030043

Biomass Spent = 81900 [kWh]
Biomass Spendage Present Cost = 4696.9327

Total Time Running (both modes) = 2730
Time Running Preemptively = 2347
Times Generator Switches On = 565
Average-Time Uninterrupted Generation = 4.8319
Unused Generator Output = 6900[kWh]

Transitions to Running Mode = 2795
Transitions to Running Preemptively Mode = 153
Transitions to Waiting to Retry Mode = 231
Time Spent Starting Up = 130
Time Spent Waiting to Retry = 693
Time Spent Waiting for Biomass = 0
```

Figure 2:13 The Biomass output window

2.2.3 Plots

Plots are important in the DST. Users can visualize the hourly levels in order to understand how the system work, but also when and in what fashion the micro grid fail. The rewritten DST has one function for each plot.

2.2.3.1 Simulations Overview

It is vital to understand the simulation space when trying new parameters to the system, or when searching for a feasible solution. The simulations Overview plot is implemented in order to efficiently refine the simulation space, this is the process of changing the `SimulationsParameters` input to find an order of size which the desired optimal solution is contained.

The plots have the same layout as the matrices of which they represent. The matrices displayed in Figure 2:14 are LLP, LCoE, NPC and the number of batteries employed. The units in the plots are assumed understood by the user, the `iPv` and `jBatt` variables are documented well and easily calculated, the units of currency depend on what currency the user choose as parameter standard.

The tool plotter (the beige square in the upper left figure) can pick single solutions to retrieve `iPv`, `jBatt`, the indices of the solution, and the corresponding z-value. The left hand bar display the color legend in the figures. The 3D pan can be used if desired to see depths in the plots, these are by default displayed from above to avoid ambiguities.

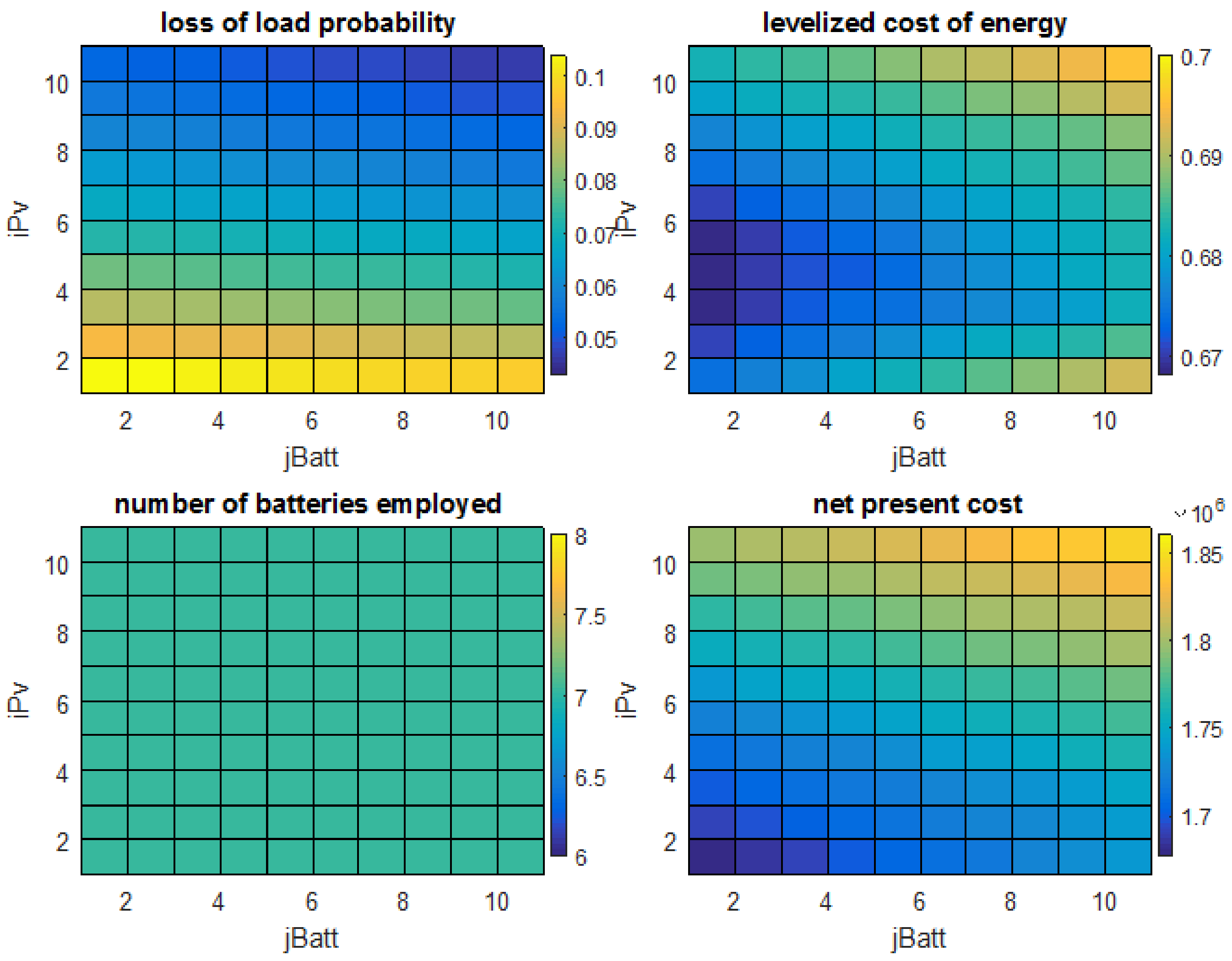


Figure 2:14 A simulation overview of SAPV systems

The source-code of the plotting function is displayed in Figure 2:15, to demonstrate how directly the output matrices are represented in the Simulation Overview plot.

```
function simulation_overview( SimOut, EcoOut )
%SIMULATION_OVERVIEW Plots the simulation-space of important values
% for more details refer to online documentation.

    dim = size(SimOut.lossOfLoadProbability);

    f = figure;
    f.Visible = 'off';

    subplot(2,2,1)
    surf(SimOut.lossOfLoadProbability)
    view(0,90)
    axis([1, dim(2), 1, dim(1)])
    title('loss of load probability')
    xlabel('jBatt')
    ylabel('iPv')
    LlpBar = colorbar('Position',[0.47 0.6024 0.01 0.3143]);

    subplot(2,2,2)
    surf(EcoOut.levelizedCostOfEnergy)
    view(0,90)
    axis([1, dim(2), 1, dim(1)])
    title('levelized cost of energy')
    xlabel('jBatt')
    ylabel('iPv')
    LcoeBar = colorbar('Position', [0.91 0.6024 0.01 0.3143]);

    subplot(2,2,3)
    surf(EcoOut.nBattEmployed)
    view(0,90)
    axis([1, dim(2), 1, dim(1)])
    title('number of batteries employed')
    xlabel('jBatt')
    ylabel('iPv')
    Nbattbar = colorbar('Position',[0.47 0.1286 0.01 0.3143])

    subplot(2,2,4)
    surf(EcoOut.netPresentCost)
    view(0,90)
    axis([1, dim(2), 1, dim(1)])
    title('net present cost')
    xlabel('jBatt')
    ylabel('iPv')
    NpcBar = colorbar('Position',[0.91 0.1286 0.01 0.3143]);


    f.Visible = 'on';

end
```

Figure 2:15 The Simulation Overview Plot module source-code

2.2.3.2 State of Charge

The SoC plot is inherited from `logplot.m`, it is now a single module with some minor layout changes. It will access any simulation given by its indices `iPv` and `jBatt` and plot the full time series of the simulation. This is a very useful plot as it makes a comprehensive interpretation of the `lossOfLoad, batteryOutputKw` and `pvAbsourbedUnusedKw` vectors, and their relationship. All kW values are normalized with respect to the `jBatt` battery capacity.

The plot lines are normalized to the size of the battery pack, but only the blue line characterize battery operation. The green line is theoretical charge to the battery after full capacity is reached and the red line is theoretical discharge after minimum SoC is reached.

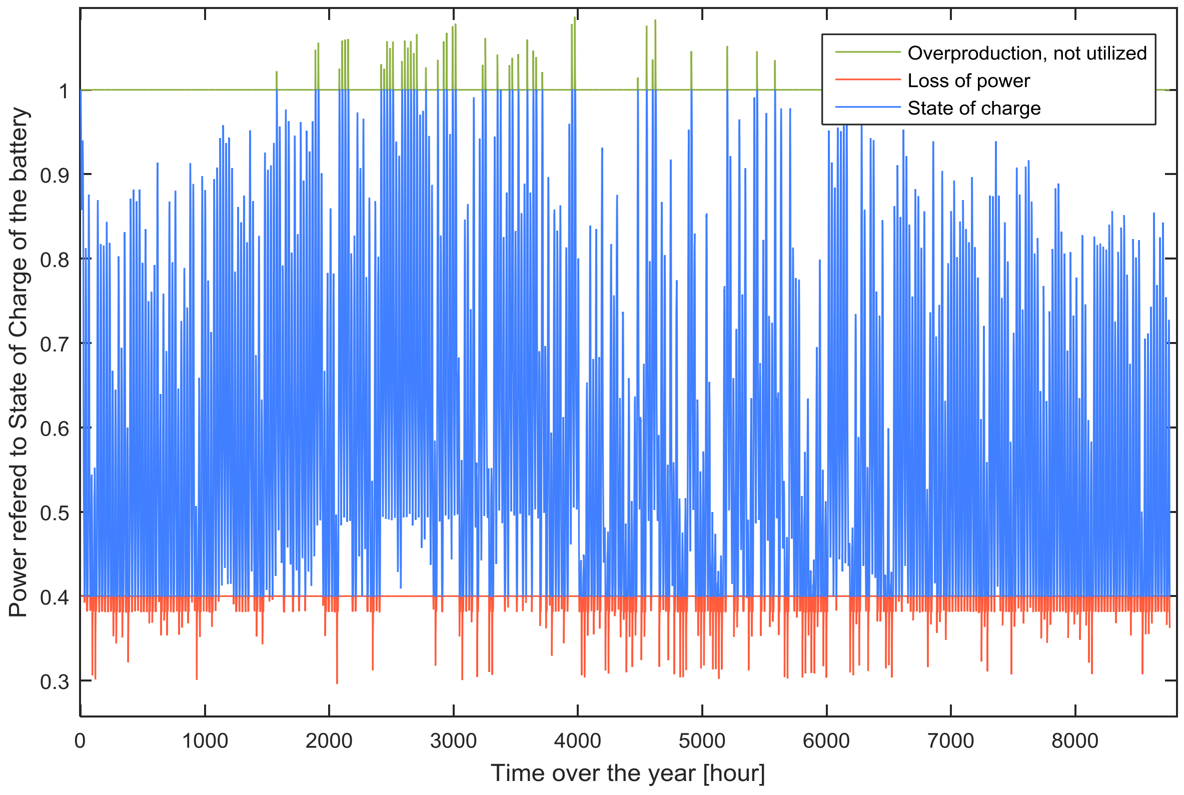


Figure 2:16: A Full Year of the State Of Charge Plot

Plotting the average day SoC removes the original expressive power because the plot lines are normalized and rely on the continuity of the curve to express battery dynamics. When the hourly average of 24 hours is plotted, the continuity disappear as seen in Figure 2:17 An Average-Day State Of Charge Plot.

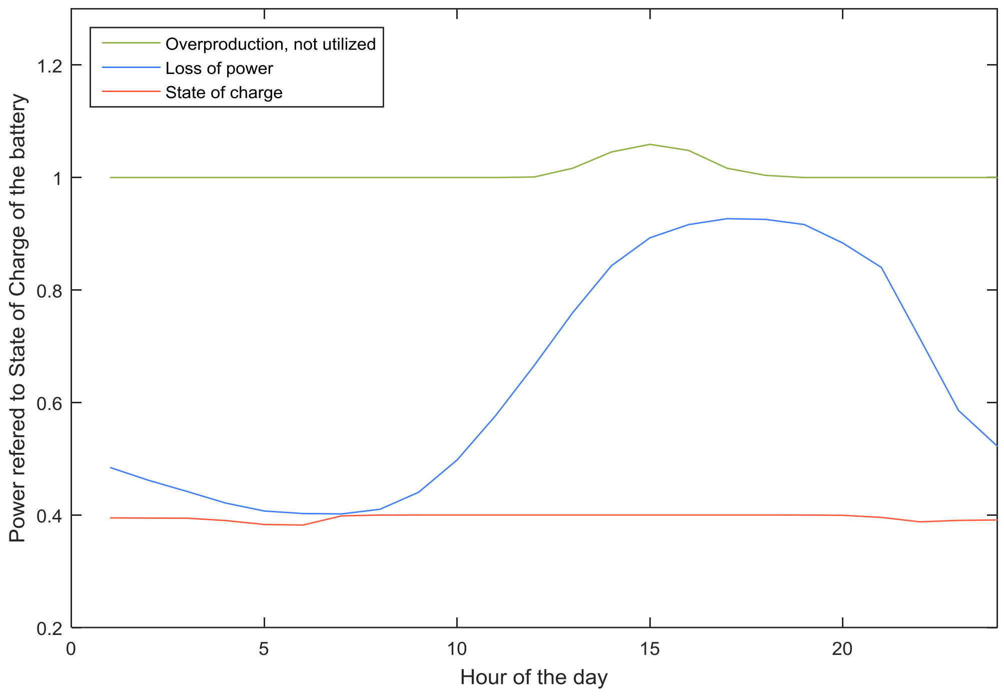


Figure 2:17 An Average-Day State Of Charge Plot is not very usefullr

2.2.3.3 Power Balance

The initial idea of the power balance plot was to enable comparison between irradiation input, battery output and load profile. The idea might also have been to enable an average day plot, by making a plot without normalized values.

`Logplot.m` contains an erroneous implementation where the legend names make sense but the actual plotted values are wrongly described. These mistakes are more likely to arise with poor code quality and testing difficulties. The misleading name-value relation can be summed up as following.

The yellow plot line seen in Figure 2:18 is the value from the variable named `P_pv` in `logplot.m`. The legend says ‘Energy from PV’, but in reality this is the absorbed power, not the utilized power. Consequently, some of this power never serves the load, nor charges the batteries.

The brown plot line is the values from the variable `bat_balance_pos` in `logplot.m`. The legend says ‘Energy flow from battery’, but in reality, this is the needed battery output, demanded by the load. The plot will therefore express a perfectly functioning system every time, the ‘needed power’ is described wrongly as ‘supplied power’.

The difference between the blue load and the brown plot line in the part where only battery supplies the load, is only from efficiency losses which proportionally increase the needed battery input.

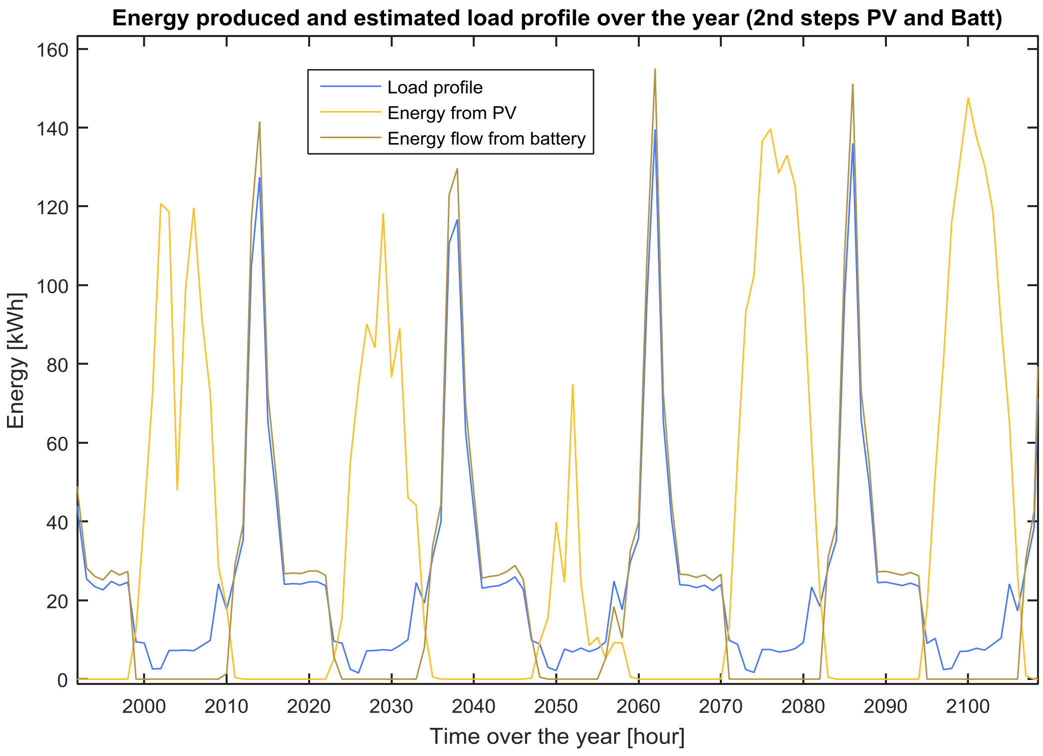


Figure 2:18: The power balance in logplot.m.

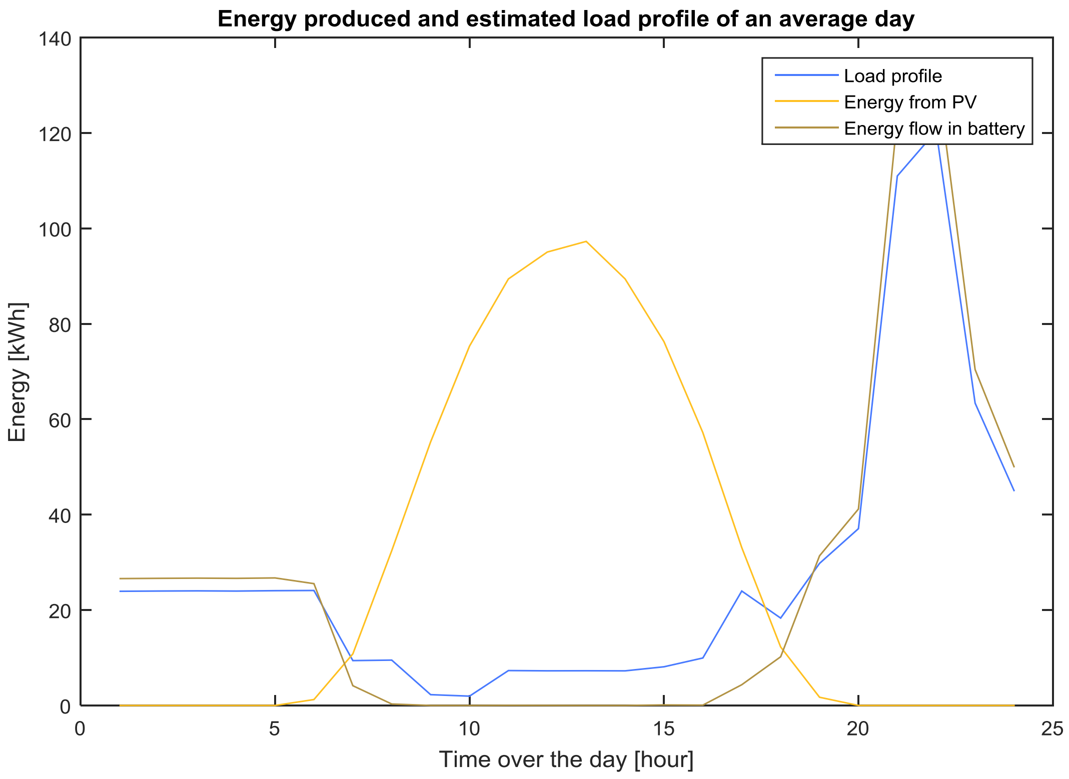


Figure 2:19: The average-day version of the Power Balance plot from logplot.m

The code for plotting the power balance in `logplot.m` is displayed in Figure 2:21 and Figure 2:20. The latter is the same snippet of code with the naming convention from the rewritten DST. It is now easy to see that the legend and names have conflicting meanings.

```
batt_balance_pos = subplus(batt_balance);
figure(1)
plot(Load,'Color',[72 122 255] / 255)
hold on
plot(P_pv,'Color',[255 192 33] / 255)
hold on
plot(batt_balance_pos,'Color',[178 147 68] / 255)
hold off
xlabel('Time over the year [hour]')
ylabel('Energy [kWh]')
title('Energy produced and estimated load profile over the year (2nd steps
PV and Batt)')
legend('Load profile','Energy from PV', 'Energy flow from battery')
```

Figure 2:20: The code for plotting of the Power Balance in logplot.m

```
neededBattOutputKwPositive = subplus(SimOutput.neededBattOutputKw(:,iPv));
figure
plot(SimInputData.load,'Color',[72 122 255] / 255)
hold on
plot(SimOutput.pvPowerAbsorbed(:,iPv),'Color',[255 192 33] / 255)
hold on
plot(neededBattOutputKwPositive,'Color',[178 147 68] / 255)
hold off
xlabel('Time over the year [hour]')
ylabel('Energy [kWh]')
title('Energy produced and estimated load profile over the year (2nd steps
PV and Batt)')
legend('Load profile','Energy from PV', 'Energy flow from battery')
```

Figure 2:21: The same code as in Figure 2:20, with the new naming convention

The Power Balance plot is rewritten as a new module, like the other plots, the module is implemented as a function where the input needed are `iPv` and `jBatt`, the indexes of the desired simulation in the simulation space. This makes it easy to call the function from the workspace, and to plot simulations effortlessly despite lacking a UI, like the `dst_platform`.

There is one function for plotting the average day, and one for plotting the full year of power balance. These are identical except for the part of calculating daily averages. This is to reduce the knowledge needed when using the plotting functions.

The utility of the new Power Balance plot is to compare the different power outputs in scale with each other, and then to clearly express when the system is out of equilibrium (when there is a power loss).

The `batteryNetOutput` is the battery discharge that reach the load without losses. The values of `battOutputKwPositive` represent battery discharge, given by the positive values of battOutputKw found with the subplus function. This vector is scaled down with discharging efficiency and the inverter efficiency to give us the battNetLoadSupply vector.

The pvNetLoadSupply is the power that is directly supplied to the load from the PV. These values are not output as a vector from the simulations module explicitly. Too many output vectors will increase the use of memory and memory operations, which should not be done just to produce a single plot, values were instead calculated from existing outputs.

The pvNetLoadSupply is found in two steps. Either every kW of power from the PV goes to the load and can be found as pvPowerAbsorbed, or the PV power goes to both batteries and the load, in this case the load profile values represent the PV supply directly to the load.

irradiationUtilized is not completely without losses to the load since some of the power absorbed has losses in the batteries. This loss does however not obscure the comparison of irradiation contribution and load supply, which is the purpose of the Power Balance plot.

The code for the plot_average_power_balance function is displayed in Figure 2:23. The calculation of averages is supported by the generic help-function get_average_day.

get_average_day finds the average value of all 24 hour segments a vector. The input vector must have hour resolution. The function is displayed in Figure 2:22.

```
function [ averageDay ] = get_average_day( input )
    %GET_DAILY_AVERAGE Finds the hourly averages of input.
    %   input an array with hourly values and the output will be an array
    %   of 24 hours with the average of every hour of a day. i.e average
    %   value at 8pm etc.

averageDay = zeros(1,24);

for t = 1:length(input)
    dayHour = mod(t,24);
    if dayHour == 0
        dayHour = dayHour + 24;
    end
    averageDay(dayHour) = averageDay(dayHour)...
                        + input(t);
end

averageDay = averageDay ./ (length(input)/ 24);

end
```

Figure 2:22 The get_average_day help-function

```
function plot_average_power_balance( BattParam,InvParam, SimInputData,
SimOutput, iPv, jBatt )
    %PLOT_AVERAGE_POWER_BALANCE an average day of power balance.
    % The power balance is created to track the sources of power serving
    % the load, and compare it in scale to where the power arrives from.

% ---------------------------------------------------------------------------
% Calculations
% ---------------------------------------------------------------------------

% irradiationUtilized is the irradiation that serves load or battery
irradiationUtilized = SimOutput.pvPowerAbsorbed(:,iPv) ...
                    - SimOutput.pvPowerAbsorbedUnused(:,iPv,jBatt);

% discharged is the amount of discharged power from the battery
discharged = subplus(SimOutput.battOutputKw(:,iPv,jBatt))';

% battNetLoadSupply is the net power supply from battery to load without
% accounting for the loss 'on the way' to the load
battNetLoadSupply = discharged...
                  * BattParam.dischargingEfficiency...
                  * InvParam.efficiency;

% pvNetLoadSupply is the net power supply from the pv to load. i.e the
power that actually serves the load directly.
pvNetLoadSupply = zeros(1,SimInputData.nHours);

% when the needed battery is negative it means that the PV serves all the
% load alone successfully, we can use the load data to reflect PV supply.
pvNetLoadSupply(SimOutput.neededBattOutputKw(:,iPv)<0) = ...
              SimInputData.load( SimOutput.neededBattOutputKw(:,iPv)<0 );

% when the needed battery is negative it means that all the power from the
% PV goes to the load and nothing to batteries.
pvNetLoadSupply(SimOutput.neededBattOutputKw(:,iPv)>0) = ...
    SimOutput.pvPowerAbsorbed( SimOutput.neededBattOutputKw(:,iPv) > 0 );

% netLoadSupply shows follows the load as long as the system provides
% enough power.
netLoadSupply = battNetLoadSupply + pvNetLoadSupply;

% average-day values are calculated here:
averageIrradiationUtilized = get_average_day(irradiationUtilized);
averageLoad = get_average_day(SimInputData.load);
averageNetLoadSupply = get_average_day(netLoadSupply);

% ---------------------------------------------------------------------------
% Plotting
% ---------------------------------------------------------------------------
figure; hold on
plot(averageIrradiationUtilized,'Color',[200,200,20]/255)
plot(averageLoad, 'Color', [255,1,1]/255 )
plot(averageNetLoadSupply, 'Color', [1,1,255]/255)
axis([0 24 0 inf])
```

Figure 2:23 The plot_average_power_balance function.

`pvNetLoadSupply` and `battNetLoadSupply` are summed together, this sum covers the load profile plotline perfectly when the system is in equilibrium (load demand is met). When there is a power-shortage, the blue `netLoadSupply` plot-line will sink and reveal the red load line. This will make it clear when the power fails, this can be seen in Figure 2:24.

The origin of power can be traced in the plot, when `irradiationUtilized` is non-zero, the PV power will always first cover the load, if the `irraditationUtilized` is zero, battery will support the load alone. The integral of the difference between `netLoadSupply` and `irradiationUtilized` is the battery output without losses. In the case of 90% efficiency in each transition, the integral difference is 72.9% of the actual battery output.

The new colors aim to appeal to intuition. If the system is functioning well, the red line is hidden by the blue line. The `irradiationUtilized` is yellow to remind of sunlight. The plot will help the user understand the magnitude of the lost load in terms of irradiation and battery dynamics.

The resulting plots can be seen in Figure 2:24 and Figure 2:25. The functions are easily maintained and modified because of modularity.

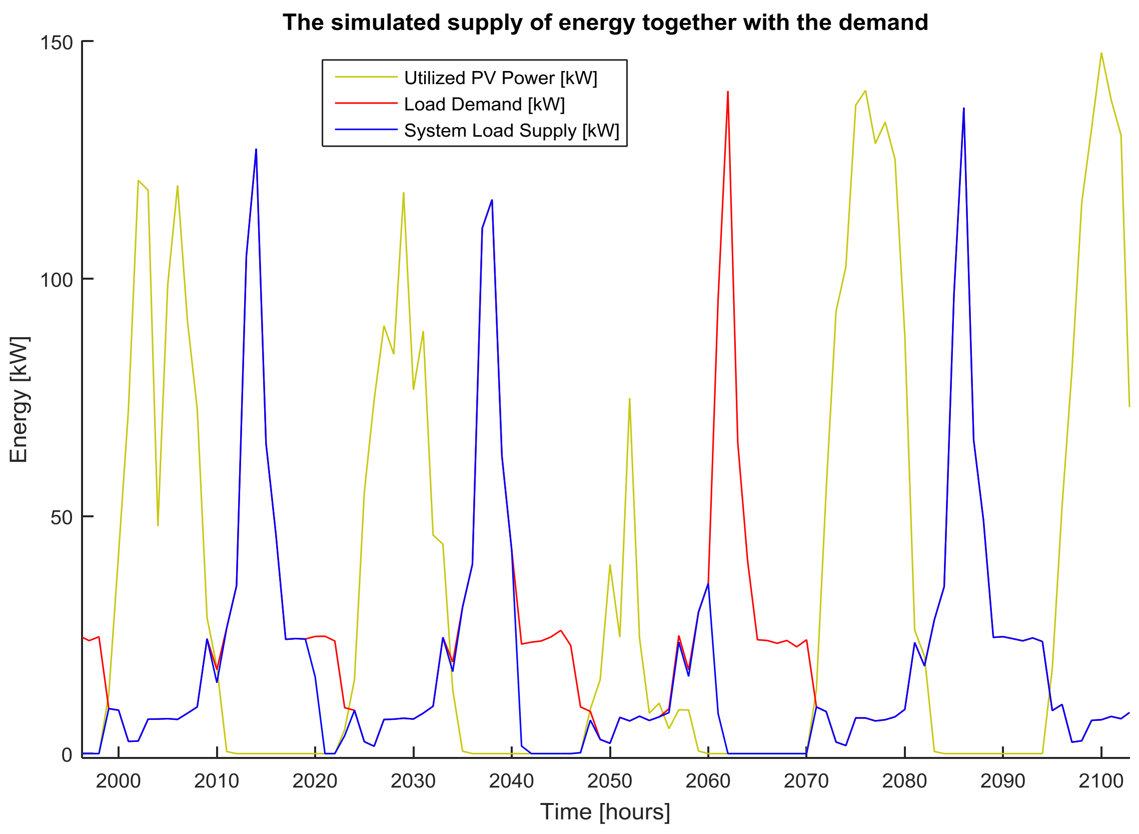


Figure 2:24: The new Power Balance plot, 4 days of the full-year plot.

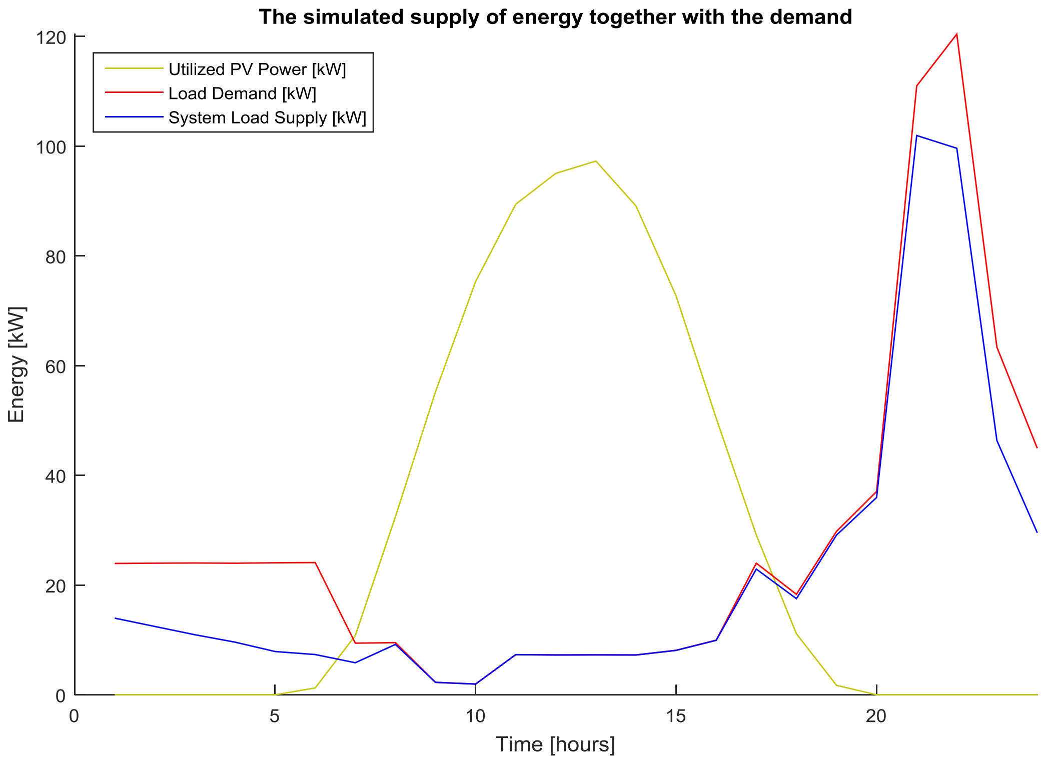


Figure 2:25: The average-day Power Balance plot.

The average day plot enables an overview that would not be produced with the SoC plot because of unit normalization, but it also includes information on the relative size of absorbed power and supplied load.

It is clear from the graph in Figure 2:25 that average performance is unacceptable, seeing that the difference is under 20 kW the user can consider additional generators. This could be a good idea in the considered system, but seeing that the shortage is over longer periods of time it might be cheaper to expand the PV and battery sizes.

2.3 Biomass System

A generic system for simulating the use of a biomass system is implemented in the module `pvbiomass_plant_simulation`. This module is identical to the `sapv_plant_simulation` in the operation of batteries, but also has the contribution from a biomass system for power production. The contribution is added to the `needed_batt_output` variable. This variable is `pvPowerAbsorbedKw - load` when no biomass generator is running. The system is meant to simulate the use of a generator parallel to the stand-alone PV plant to compensate during power shortages.

The system implementation is generic for applicability to any technology utilized, the system will also be able to simulate whether it is operated manually or fully automated.

2.3.1 The Biomass-System State Machine

The implementation of the system biomass simulation was done with a state machine. A state machine has the benefit of only allowing the system to be in one state at any time. This will simplify check of conditions as a state has a limited number of allowed transitions. The state machine implementation will make the code more readable and the program flow comprehensive.

The transition diagram can be seen in Figure 2:26, it is also available in the online documentation..

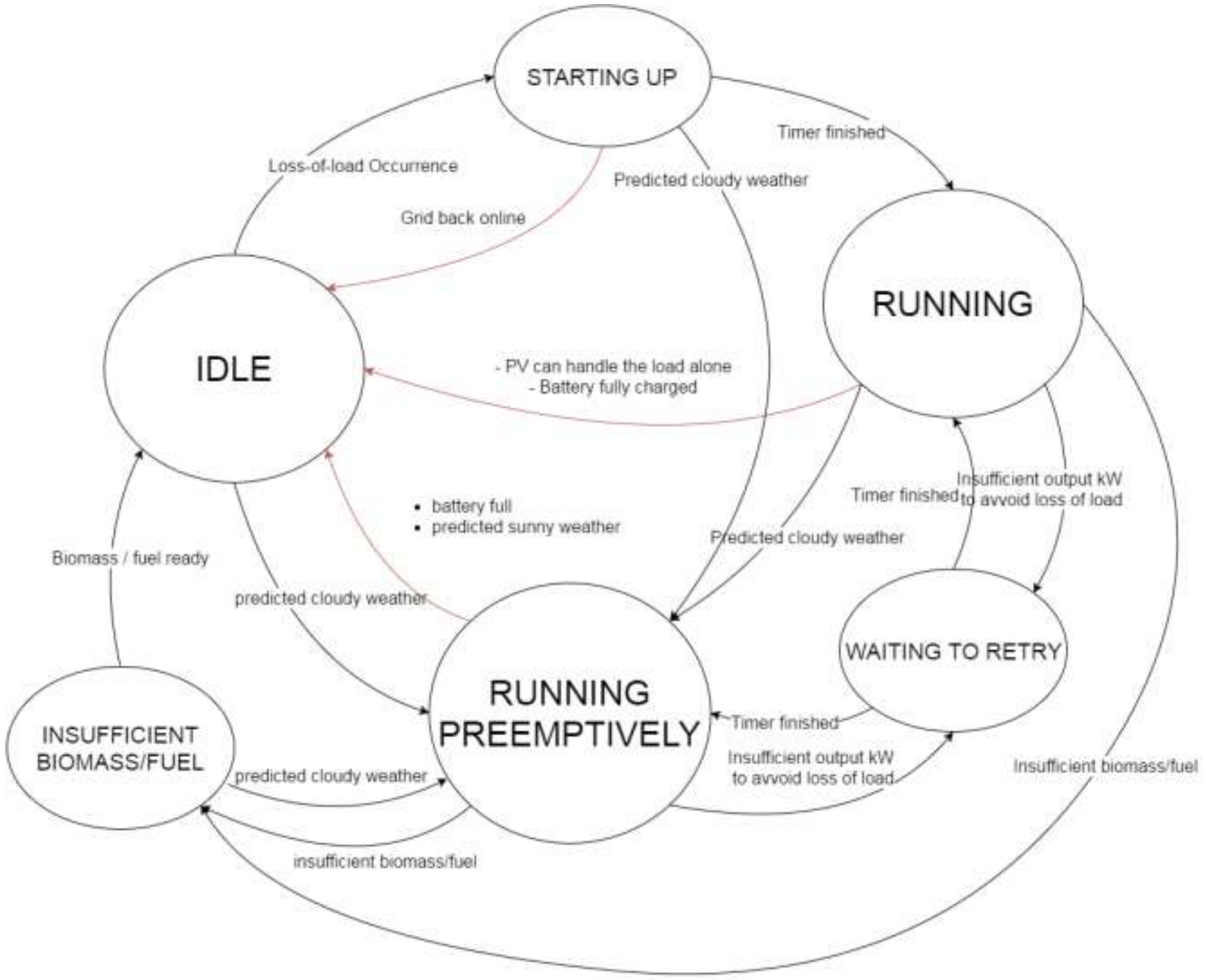


Figure 2:26 Biomass System State-Machine Transition Diagram

2.3.1.1 Preemptive Run-mode

The PV usually can have a very high output capacity in terms of kW. To replace this potential capacity completely can require a very large generator. A large generator is expensive and is more a primary solution rather than a secondary solution, and a biomass power system greater than 30 kW is not typically available off-the-shelf. We have the opportunity to steadily accumulate power in the batteries over time, and so a system with lower output capacity can be sufficient.

The simulation assumes that a day with irradiation levels lower than a certain threshold can be predicted with a weather forecast. The power generation can then be started to accumulate power before the battery is unable to meet load demand. For simplicity, we term the low irradiation days as cloudy days and high irradiation days as sunny days.

To simulate a forecasts, the algorithm finds the peak value of the `pvPowerAbsorbed` vector, the amount absorbed by the PV given irradiation data.

Using the `pvPoweAbsorbedKw` instead of the irradiation directly, will dampen any irregularly high peaks (by nominal operation factor) and give a more predictable peak value. A threshold is given by the user as a fraction named `peakPvPowerAbsorbedTreshold`. This threshold will determine if a day is forecast as cloudy or sunny, depending on its relative size compared to the global peak. This comparison is displayed in Figure 2:27

```
thisPeakPowerAbsorbedKw = max(pvPowerAbsorbedKw(t:t+24));

% check weather the coming day has a relatively low power
% absorbed, and assume that this can be predicted.
if (thisPeakPowerAbsorbedKw/globalPeakPowerAbsorbedKw)...
>  nomPeakPvPowerTreshold

    weatherPredictions{day} = 'sunny';

else

    weatherPredictions{day} = 'cloudy';

end
```

Figure 2:27 The comparison of `peakPvPowerAbsorbedKw` values

At the start of a day, the peak `pvPowerAbsorbedKw` of the following day is compared to the peak `pvPowerAbsorbedKw` value of the entire simulation time series. If the peak of the current day is lower than a fraction of the global maximum, the weather is forecast as cloudy, and vice versa.

When a day is forecast as cloudy, the system will run in preemptive mode until the battery is full, a sunny day is forecast or the biomass/fuel has run out. The entire state machine can be seen in section 2.3.1.

It is important not to run in preemptive mode on days when normal PV operation would have been sufficient as this would mean wasting biomass.

2.3.1.2 Startup Delay

The most important generic simulation choice is the `startup_delay` parameter, this parameter is the time expected to pass between a power loss and when the biomass system starts generating power.

The time needed to start the biomass power generation will vary between solutions. A fast response is a gas generator with automated startup at LoL, and a slow response time is a furnace-powered steam turbine or an incomplete combustion process, initiated when a human discovers the LoL. The very worst case can exceed an hour, the operator might have to physically move to the plant to operate it, and perhaps deal with additional delays.

The alternative to manual input of time delays is preprogrammed time delays for each system choice. The preprogrammed time estimates might not apply because of new innovation or technology, creating unnecessary workarounds for the user.

2.3.1.3 Waiting to Retry

Resources are wasted when running, if the power output of the system is too low to bring the grid online. For these cases the simulation has a parameter if the user wish to turn off the system and wait to retry. If the waiting time is zero, the system will keep running regardless.

The batteries could be detached from load in this period, and so charged while waiting to retry. This might however cause many extra cycles on the batteries, inducing unpredictable wear and tear. A system should not be dimensioned to operate in a potentially destructive way.

2.3.1.4 Biomass kg to kWh Conversion

The biomass supply parameter unit is in kWh, meaning that the user will have to make estimations on conversions from tones to kWh himself. The monitoring of produced kWh can be simpler than weighing the biomass, the biomass change levels of moisture and quality during storage.

The quality of biomass and processing varies a lot between installations. The plant efficiency depends on the kind of system implemented, how it’s operated and the system’s quality. The biomass potential energy depends on the type of biomass, where it is grown and how it is potentially preprocessed and ultimately used to power the system. All these things strongly effect the conversion rate.

The conversion is simple math but with many parameters, these parameters can be cascaded and risk producing large imprecisions. The user should consult the biomass system manufacturer and people with experience from a similar production materials. This way the user maintains control.

2.3.2 pvbiomass_plant_simulation.m

The code can be seen in the file pv_biomass_plant_simulation.m (github link). The implementation flow-chart is displayed in Figure 2:28.

The biomass state machine is included in every hour iteration of the simulation. The system state is kept in the variable `biomassSystemState`, which can be in the following states 'Idle', 'Starting Up', 'Running, 'Running Preemptively', 'Waiting to Retry' and 'Waiting for Biomass'. The states can perform two actions, they handle transitions to new states, and they set the `runBiomassGeneratorHours` variable. This variable decides how long the biomass generator should run the current hour.

The generator runs after the state machine sets `runBiomassGeneratorHours`, if positive, for example 0.5, it means that the generator should run for 30 minutes. When power contribution of running time is processed, the `runBiomassGenerator` variable is set back to zero for the next iteration.

When the biomass system generates power, the contribution is subtracted from the `neededBattOutputKw` variable, the same way power contribution from the PV is subtracted from the load (with losses) to calculate the `neededBattOutputKw` variable. The biomass contribution is not subject to inverter loss, as mechanical generators usually output AC power.

The system can be in two states during one hour iteration if the system changes to the waiting state 'Starting Up'. This is necessary as waiting time can be less than one hour, and require the system to respond before iterating to the next hour. When the timer is less than 1 hour in a waiting state, the remaining time is calculated.

The generator output and biomass consumption is scaled to fit the amount of time of running, found in the variable `runBiomassGeneratorHours`. The `neededBattOutputKw` is also scaled accordingly, when comparing to see whether output is insufficient.

2.3.2.1 Runtime

The biomass state machine is executed every hourly iteration of the simulations. There are no further loops, computational complexity is therefore not increased and runtimes will stay within a predictable scale. The overhead of the state machine is a constant that increased the run time around 10 times during tests. The user is expected to narrow the number of simulations, before speculating in biomass implementations.

The extra run-time is strictly self-time as discussed in section 2.1.4.2 Computation Speed, meaning that there are no external functions slowing down runtime.

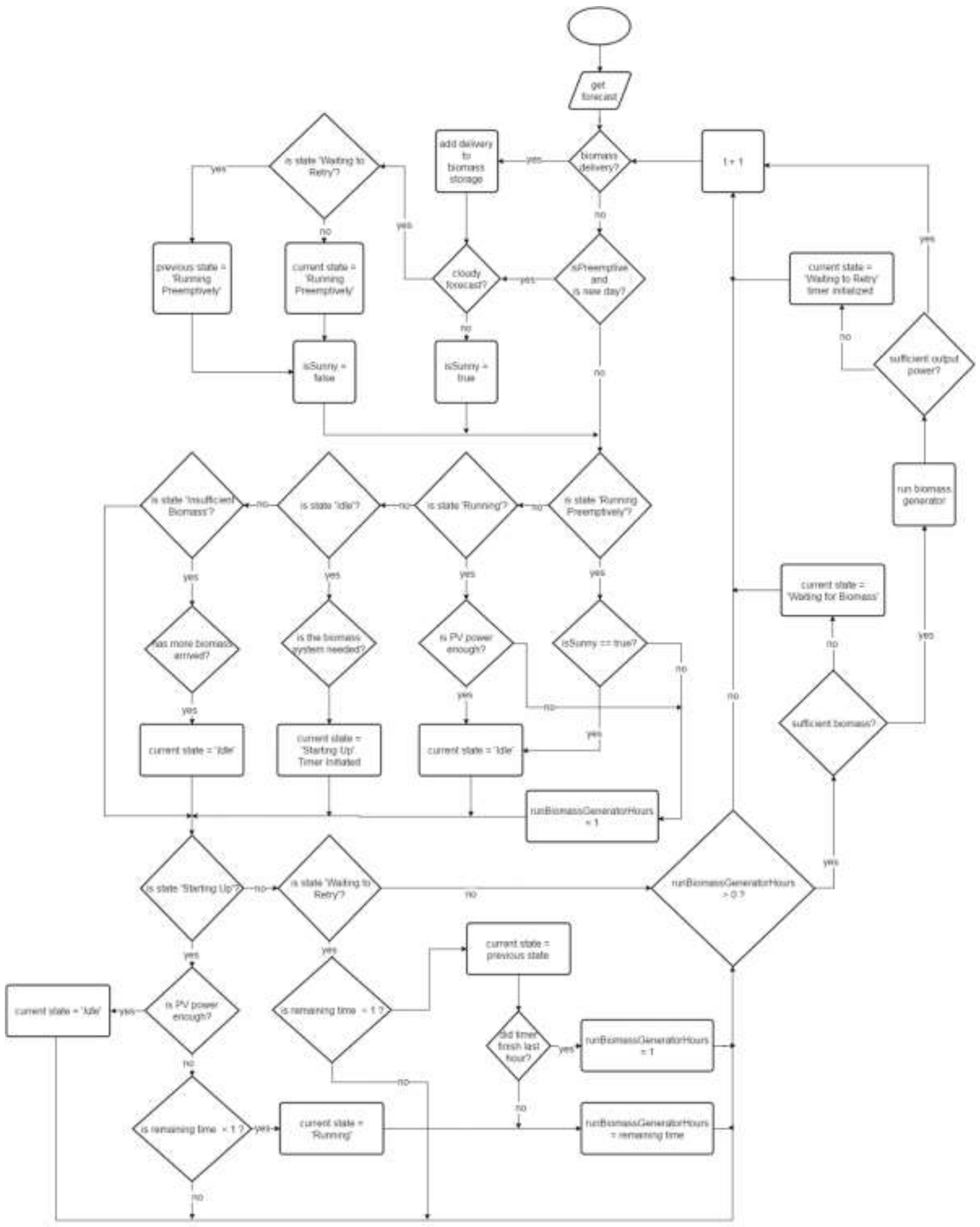

get forecast
biomass delivery?
add delivery to biomass storage
t + 1
is state 'Waiting to Retry'?
previous state = 'Running Preemptively'
current state = 'Running Preemptively'
isSunny = false
cloudy forecast?
isSunny = true
isPreemptive and is new day?
current state = 'Waiting to Retry' timer initialized
sufficient output power?
run biomass generator
is state 'Insufficient Biomass'?
is state 'Idle'?
is state 'Running'?
is state 'Running Preemptively'?
has more biomass arrived?
is the biomass system needed?
is PV power enough?
isSunny == true?
current state = 'Waiting for Biomass'
sufficient biomass?
current state = 'Idle'
current state = 'Starting Up' Timer Initiated
current state = 'Idle'
runBiomassGeneratorHours = 1
is state 'Starting Up'?
is state 'Waiting to Retry'?
runBiomassGeneratorHours > 0 ?
current state = previous state
current state = 'Idle'
is PV power enough?
is remaining time = 1 ?
did timer finish last hour?
runBiomassGeneratorHours = 1
is remaining time < 1 ?
current state = 'Running'
runBiomassGeneratorHours = remaining time
yes
no

Figure 2:28 Flow-chart of pvbiomass_plant_simulation.m

2.3.2.2 Biomass System Outputs

Biomass system output [kW] for each hour is added to the `SimulationOutputs` class. The `BiomassOccurrences` struct is also passed to the output class, this struct counts the occurrences in the state machine in terms of hours and events, to enable performance monitoring.

2.3.3 economic_analysis_biomass.m

economic_analysis_biomass.m is an adaption of economic analysis for the biomass system. The additions to the outputs in EconomicAnalysisOutputs are the variables bioSysNetPresentCost, biomassPresentCost and bioSysLevelizedCostOfEnergy.

These are the NPC of the entire biomass system installation, including operation and maintenance, the money spent on biomass in present cost and the present cost of each kW produced by the system.

It is assumed that the lifetime of a biomass system is approximately the same as the PV-plant and that there will be no significant salvage when the period has ended.

The LCoE is calculated by splitting the NPC into equal annual payments, including interests during the down-payment period. This annual payment is divided by the kWh successfully delivered to the load during simulation, which gives us the present cost of each kW.

The calculations are displayed in Figure 2:29.

```
[…]
investmentCost(iPv,jBatt)  = pvCostTot(iPv, jBatt) ...
                           + battCostTot(iPv, jBatt) ...
                           + inverterCostTot(iPv, jBatt) ...
                           + installBalanceOfSystemTotCost(iPv, jBatt)...
                           + EcoParam.bioSystemInvestmentCost;
[…]
% for year k in plant lifetime
bioSystemOperationCost(iPv, jBatt) ...
                         = bioSystemOperationCost(iPv,jBatt) ...
                         + (EcoParam.bioSystemOperationCostYear...
                         / (1 + EcoParam.interestRate)^k);
[…]
% after iterating over plant lifetime years
bioSystemOperationCost(iPv,jBatt) ...
                           = bioSystemOperationCost(iPv,jBatt)...
                           + biomassPresentCost(iPv,jBatt);
[…]
bioSysNetPresentCost(iPv,jBatt)  = EcoParam.bioSystemInvestmentCost...
                                 + bioSystemOperationCost(iPv,jBatt);
[…]
bioSysLevelizedCostOfEnergy(iPv,jBatt) ...
                    = (bioSysNetPresentCost(iPv,jBatt)...
                    * capitalRecoveryFactor)...
                    / (sum(SimOut.biomassGeneratorOutputKw(:,iPv,jBatt))...
                    - sum(SimOut.biomassGeneratorOutputUnusedKw(:,iPv,jBatt)));
```

Figure 2:29 Additions to the Economic Analysis module

2.3.4 The Results of running the Biomass System

The system was tested both without-, and in parallel with PV/battery. The economic analysis shows that the implementation of biomass system can improve the performance and economic outlook of the plant, this is shown in the use case presented in section 3.1 A Brief Use-Case for the DST.

2.3.4.1 Running without PV

When running with only biomass generators, the parameters of the system was input so that the biomass system would be large enough to support the entire load alone. The result was a functioning plant which had a small LLP for every non-preemptive startup of the generator.

In any real life use case, the generator would be turned on at regular intervals every day and these LLP's would not exist. This module is not made for stand-alone biomass generation and hence any modifications to optimize this functionality is neglected.

The result of one simulation can be seen below, there are examples of both preemptive and non-preemptive behavior in the system. A loss of load triggers normal run-mode and a cloudy forecast triggers preemptive run-mode, as seen in Figure 2:30.

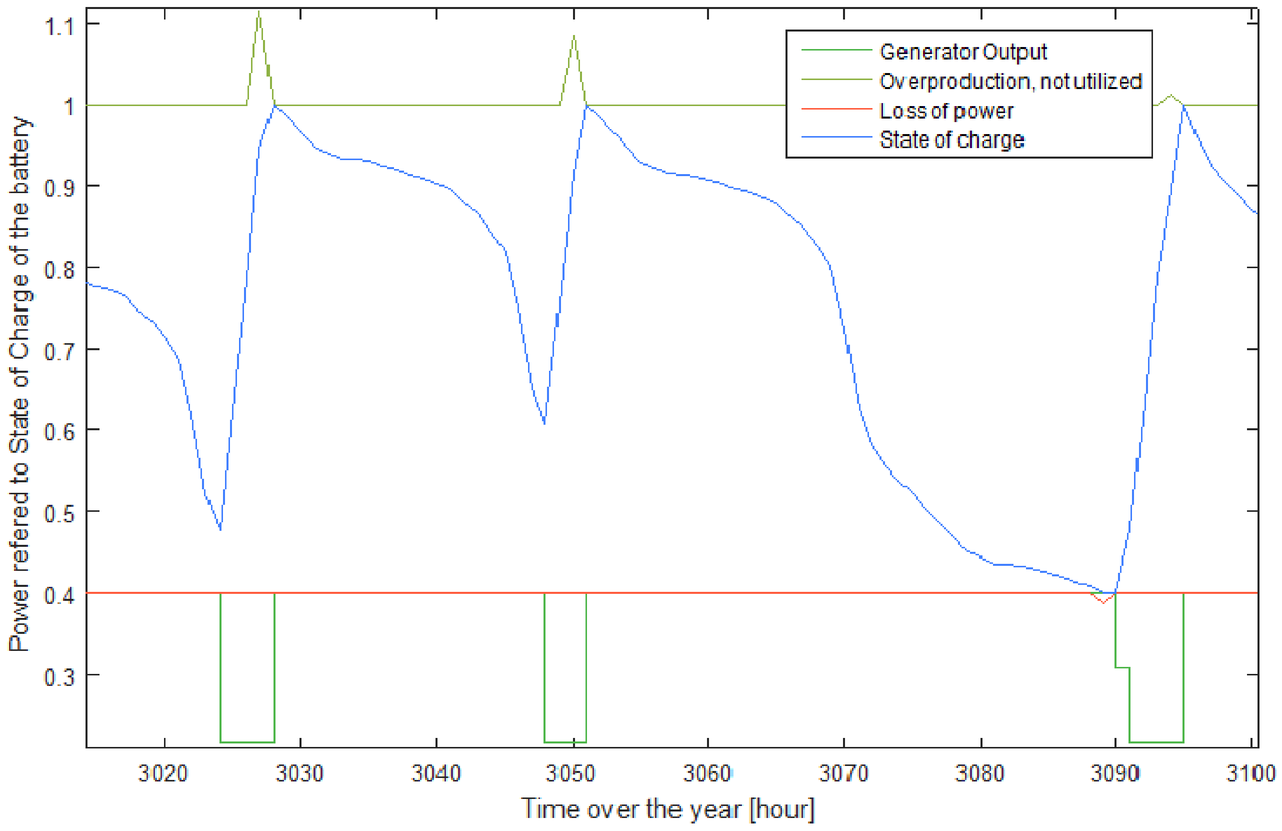


Figure 2:30 SoC with biomass system alone, preemptive / nonpreemptive resoponse

2.3.4.2 Biomass and PV running together

The system responds appropriately and supplies the expected contributions to the system. The system starts up at lost load and in preemptive mode, and biomass system turns off when the PV can handle the load alone or/and when the battery is full depending on run-mode.

The system successfully 'saves' the microgrid from power-shortages even though the generator size is small compared to the battery size. The relative size of the generator is reflected in Figure 2:31.

Preventing longer periods of lost loads will in systems with high LLP often mean an increase of discharge cycles, the batteries are given fewer "breaks" while staying discharged instead of charging. This is however characteristic of a very poorly functioning system, and from my experience can be expected from systems with LLP at least over 10% which is in most cases far too much.

When dealing with systems of low LLP, a normal run mode responding to lost load will not influence the charge cycles, but if the system is run in preemptive mode, the charge cycles are going to be less deep as they are prevented from hitting minimum state of charge as seen in Figure 2:30. Depending on the cycles-to-failure characteristics of the battery, the saved cost can be very high.

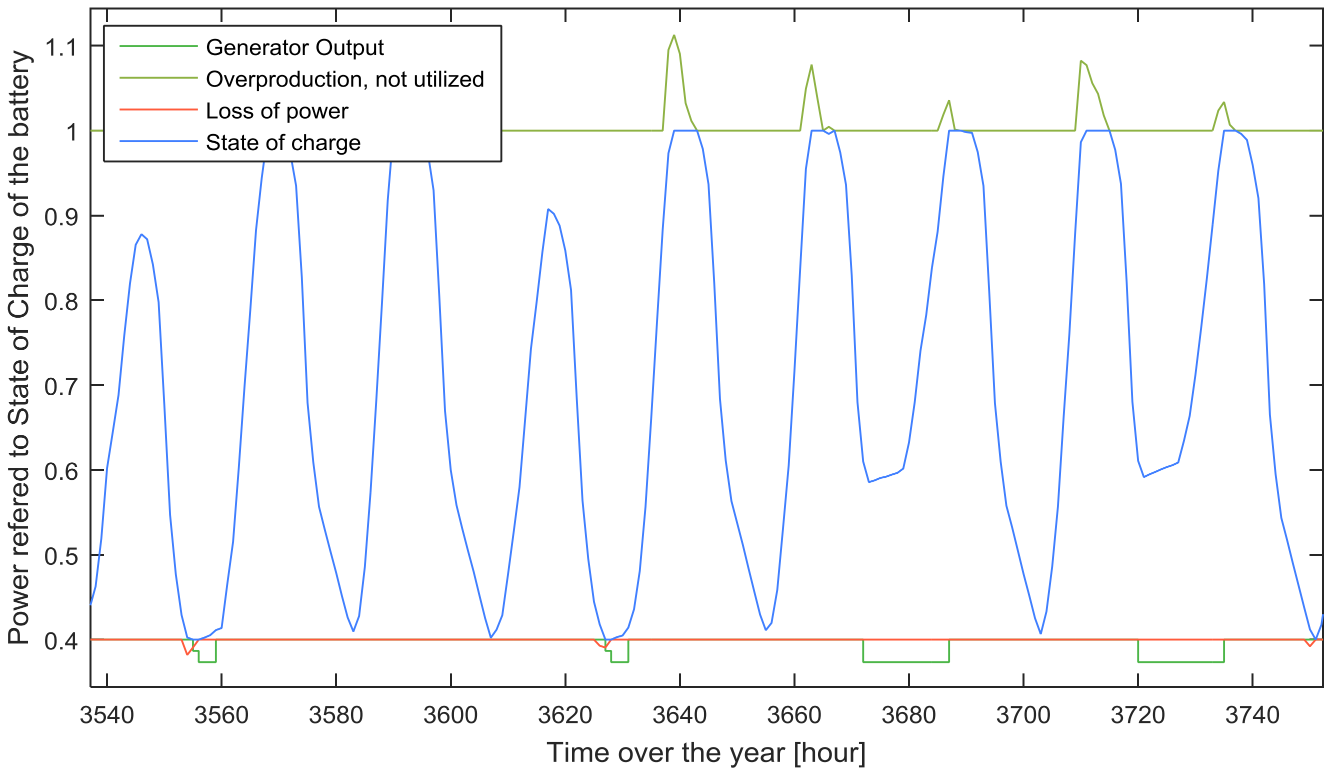


Figure 2:31 SoC with biomass and PV generation, preemptive/non-preemptive response.

2.3.4.3 Effect on Simulation Space

The Simulation Overview plots can with the biomass system in Figure 2:32 and with SAPV simulation and the same component sizes in Figure 2:33. It should be noted that the simulated system has a very high supply of free biomass, the reason is that sporadic shortages of biomass can make the plots harder to interpret.

The simulation was made with pv sizes from 100 kW to 500 kW with a step size of 10 kW and with batteries from 1000 kWh to 1800 kWh with step size of 10 kWh. This should represent a variation of functionalities and costs.

The area with low values of LLP is larger with the biomass system active, this is as expected. The second thing one clearly notice is the change in the number of employed batteries. The batteries are put through fewer deep cycles in preemptive mode, and will not have to go through the same amount of stress as the SAPV batteries on cloudy days.

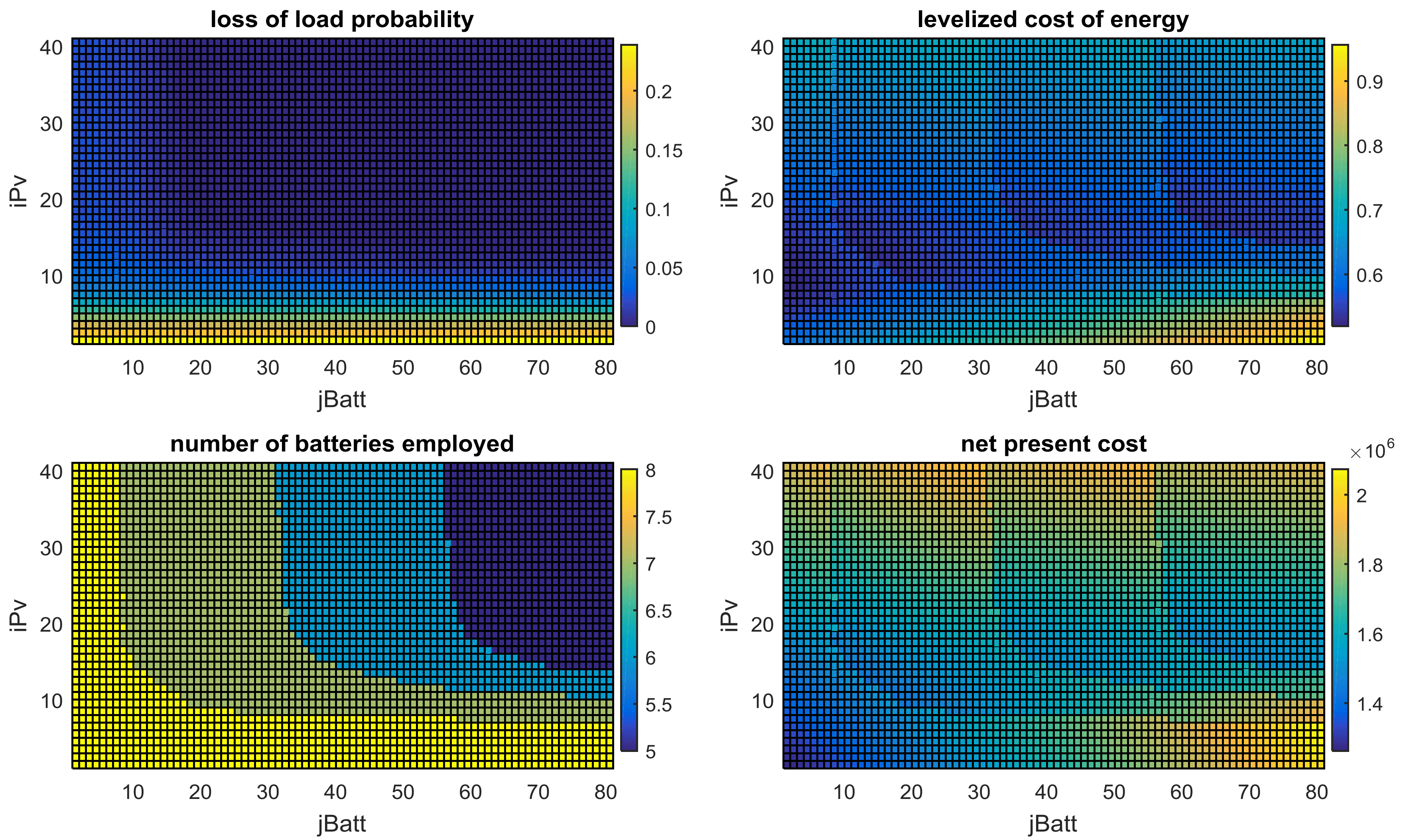


Figure 2:32 Wide Simulation Overview with Biomass System

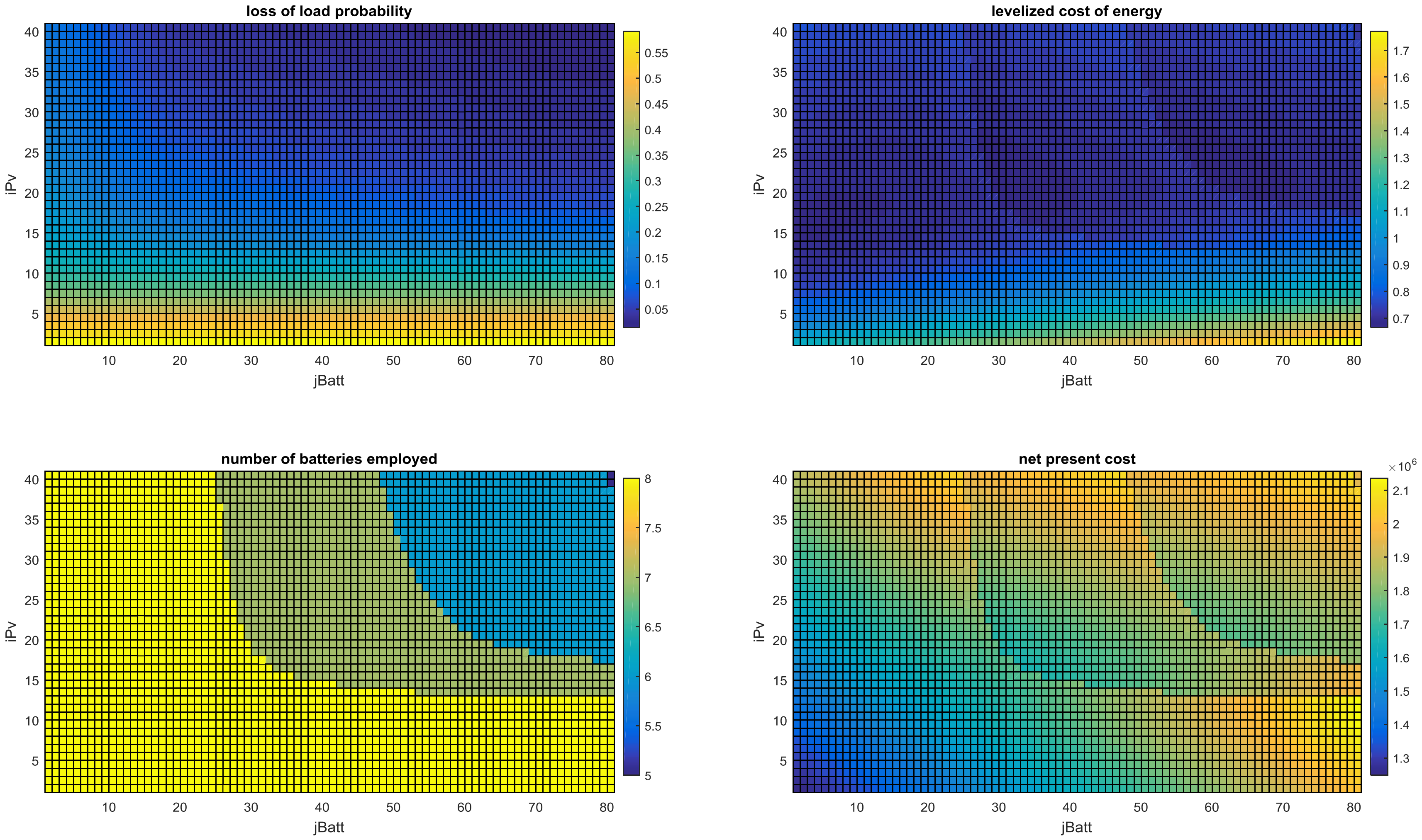


Figure 2:33 Same Simulation Space without Biomass System

2.4 Documentation

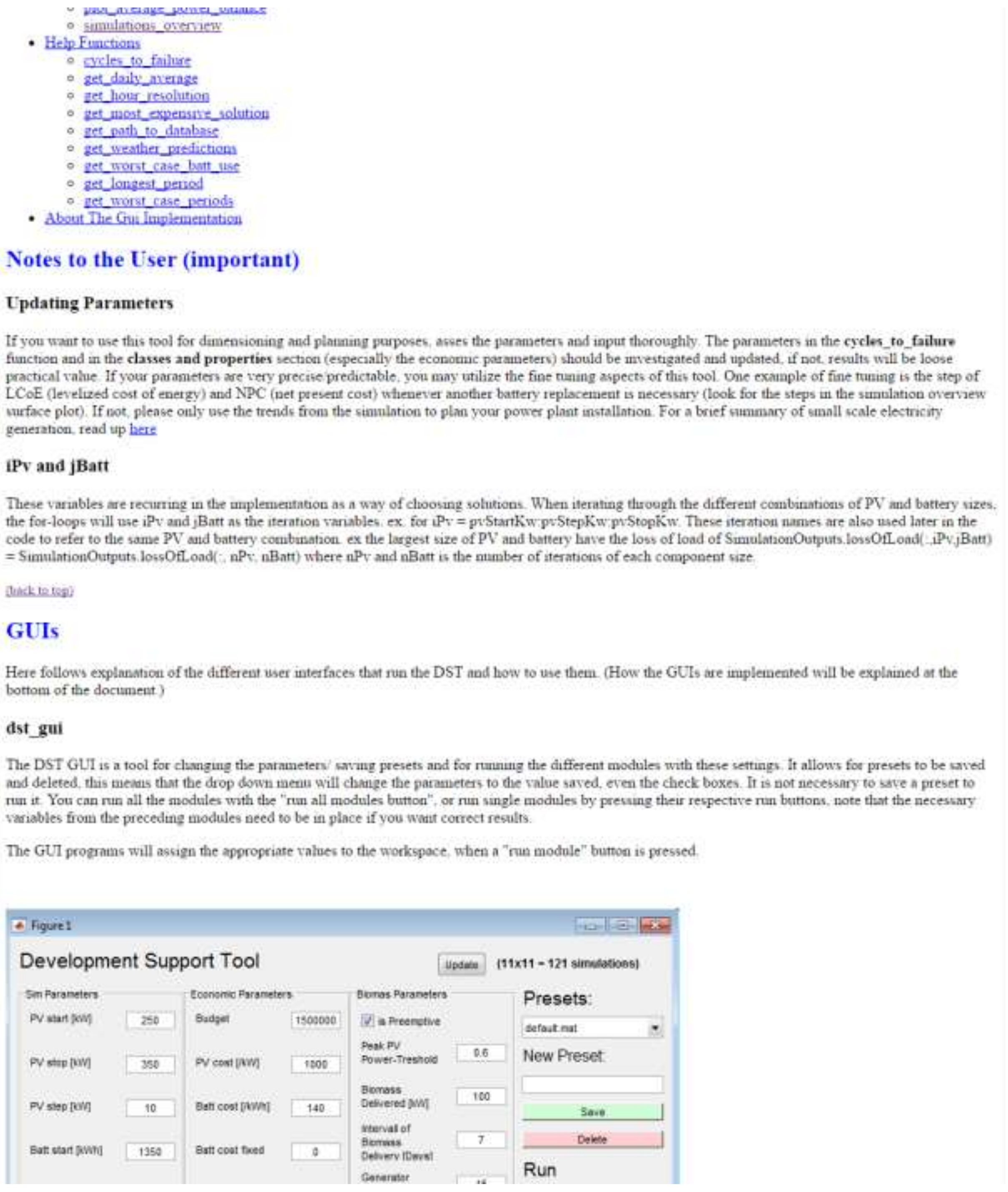
The new DST has online documentation written in HTML. The code is in basic HTML code which can be modified and understood by virtually anyone. HTML can additionally be written and read by practically any device. This enables continuous development of the documentation as the tool grows and is modified, it will also allow documentation to be read from a phone.

The goal is a minimalistic documentation design which is easily navigated by a contents list and a browsers search-functionality. The design is inspired by the reference pages of the Google programing language GO as seen here https://golang.org/ref/spec.

If the user finds a function, parameter or variable that needs explanation, the name of the component can be input with `ctrl+f` and the explanation will appear without having to click any hyperlinks as everything is in one page. Additionally there is a table of contents and a link to the top on the page, this can be handy if one is investigating several objects related to eachother. See Figure 2:34.

The documentation is organized in 6 main parts. ‘Notes to the User’ is reserved for the most vital information needed to work with the DST. ‘Modules’ explain the main modules of the DST, their inputs, outputs and functionality. ‘Classes and Properties’ explain the different classes that is used to pass variables between functions. These contain explanations for every large-scope variable in the DST. ‘Plots’ will explain how to use and interpret the plot functions, and lastly the ‘Help Functions’ is a summary of the globally accessible functions, used different places in the DST.

The documentation is several pages long and can be found at http://folk.ntnu.no/gardhi/dstReferenceManual/referenceManual.html or http://gardhi.github.io/dstReference/referenceManual.html. When someone wants to inherit this code it is sufficient to download the script from the webpage. The Git-hub repository will also be open for collaborators.



- simulations_overview
- Help Functions
    - cycles_to_failure
    - get_daily_average
    - get_hour_resolution
    - get_most_expensive_solution
    - get_path_to_database
    - get_weather_predictions
    - get_worst_case_batt_use
    - get_longest_period
    - get_worst_case_periods
- About The Gui Implementation

Notes to the User (important)

Updating Parameters

If you want to use this tool for dimensioning and planning purposes, asses the parameters and input thoroughly. The parameters in the **cycles_to_failure** function and in the **classes and properties** section (especially the economic parameters) should be investigated and updated, if not, results will be loose practical value. If your parameters are very precise/predictable, you may utilize the fine tuning aspects of this tool. One example of fine tuning is the step of LCoE (levelized cost of energy) and NPC (net present cost) whenever another battery replacement is necessary (look for the steps in the simulation overview surface plot). If not, please only use the trends from the simulation to plan your power plant installation. For a brief summary of small scale electricity generation, read up here

iPv and jBatt

These variables are recurring in the implementation as a way of choosing solutions. When iterating through the different combinations of PV and battery sizes, the for-loops will use iPv and jBatt as the iteration variables. ex. for iPv = pvStartKw:pvStepKw:pvStopKw. These iteration names are also used later in the code to refer to the same PV and battery combination. ex the largest size of PV and battery have the loss of load of SimulationOutputs.lossOfLoad(:,iPv,jBatt) = SimulationOutputs.lossOfLoad(:, nPv, nBatt) where nPv and nBatt is the number of iterations of each component size.

(back to top)

GUIs

Here follows explanation of the different user interfaces that run the DST and how to use them. (How the GUIs are implemented will be explained at the bottom of the document.)

dst_gui

The DST GUI is a tool for changing the parameters/ saving presets and for running the different modules with these settings. It allows for presets to be saved and deleted, this means that the drop down menu will change the parameters to the value saved, even the check boxes. It is not necessary to save a preset to run it. You can run all the modules with the "run all modules button", or run single modules by pressing their respective run buttons, note that the necessary variables from the preceding modules need to be in place if you want correct results.

The GUI programs will assign the appropriate values to the workspace, when a "run module" button is pressed.

Figure 1

Development Support Tool

(11x11 = 121 simulations)

Sim Parameters

PV start [kW]	250
PV stop [kW]	350
PV step [kW]	10
Batt start [kWh]	1350

Economic Parameters

Budget	1500000
PV cost [/kW]	1000
Batt cost [/kWh]	140
Batt cost fixed	0

Biomas Parameters

[✓] is Preemptive	
Peak PV Power-Treshold	0.6
Biomass Delivered [kW]	100
Intervall of Biomass Delivery [Days]	7

Presets: default.mat

Figure 2:34 The DST Online Reference Manual Design

3: Summary and Recommendations for Further Work

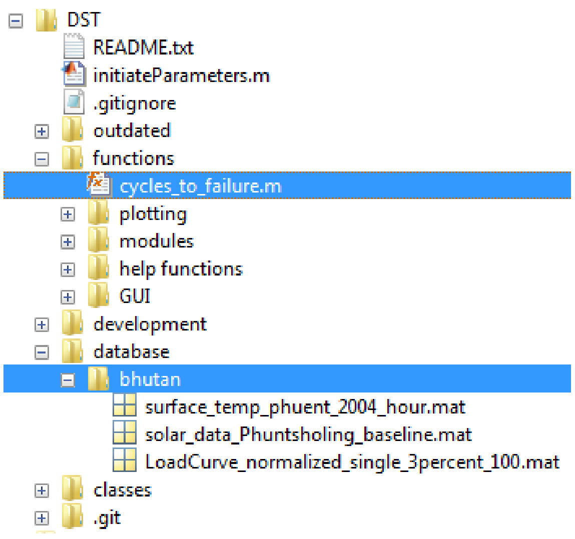
3.1 A Brief Use-Case for the DST

In this chapter the use case of the DST will be presented. The GUI is initialized by running `dst_platform.m` this can be done by writing `dst_platform` in the matlab console

3.1.1 Research and Parameter Input

The first thing one need to consider is what kind of components are going to be purchased. When this is done, the DST parameters need to be updated. This mainly involves finding products and gathering the associated technical specifications from manufacturer or distributor. The user must also make sure that all currencies are the same. To display and input the parameters, open the DST platform GUI by running the `dst_platform.m` function, the open GUI is seen in Figure 3:2.

The `cycles_to_failure` function needs to be updated, this describes how many cycles the battery can perform at a certain level of discharge, before needing replacement. This information is supplied by the battery manufacturer.



```
⊟ DST
    README.txt
    initiateParameters.m
    .gitignore
  ⊞ outdated
  ⊟ functions
      cycles_to_failure.m
    ⊞ plotting
    ⊞ modules
    ⊞ help functions
    ⊞ GUI
  ⊞ development
  ⊟ database
    ⊟ bhutan
        surface_temp_phuent_2004_hour.mat
        solar_data_Phuntsholing_baseline.mat
        LoadCurve_normalized_single_3percent_100.mat
  ⊞ classes
  ⊞ .git
```

Figure 3:1 Location of cycles_to_failure and the data set

If some questions arise, the large ‘Help’ button opens a browser window with the online reference manual.

The files containing input data: irradiance, ambient temperature and load profile must be saved under ‘database’ in a folder named after the current data set. In this case we use data from the ‘bhutan’ set. If the vector sizes are different, the program will warn you about this as the class constructors forbid it. The file names and the data set folder name are input to the GUI under ‘Input Data Files’

Now the preset of all the data input can be saved as a .mat file, here we have named the preset ‘NewSystRefinement1’.

3.1.2 Simulation-Space Refinement

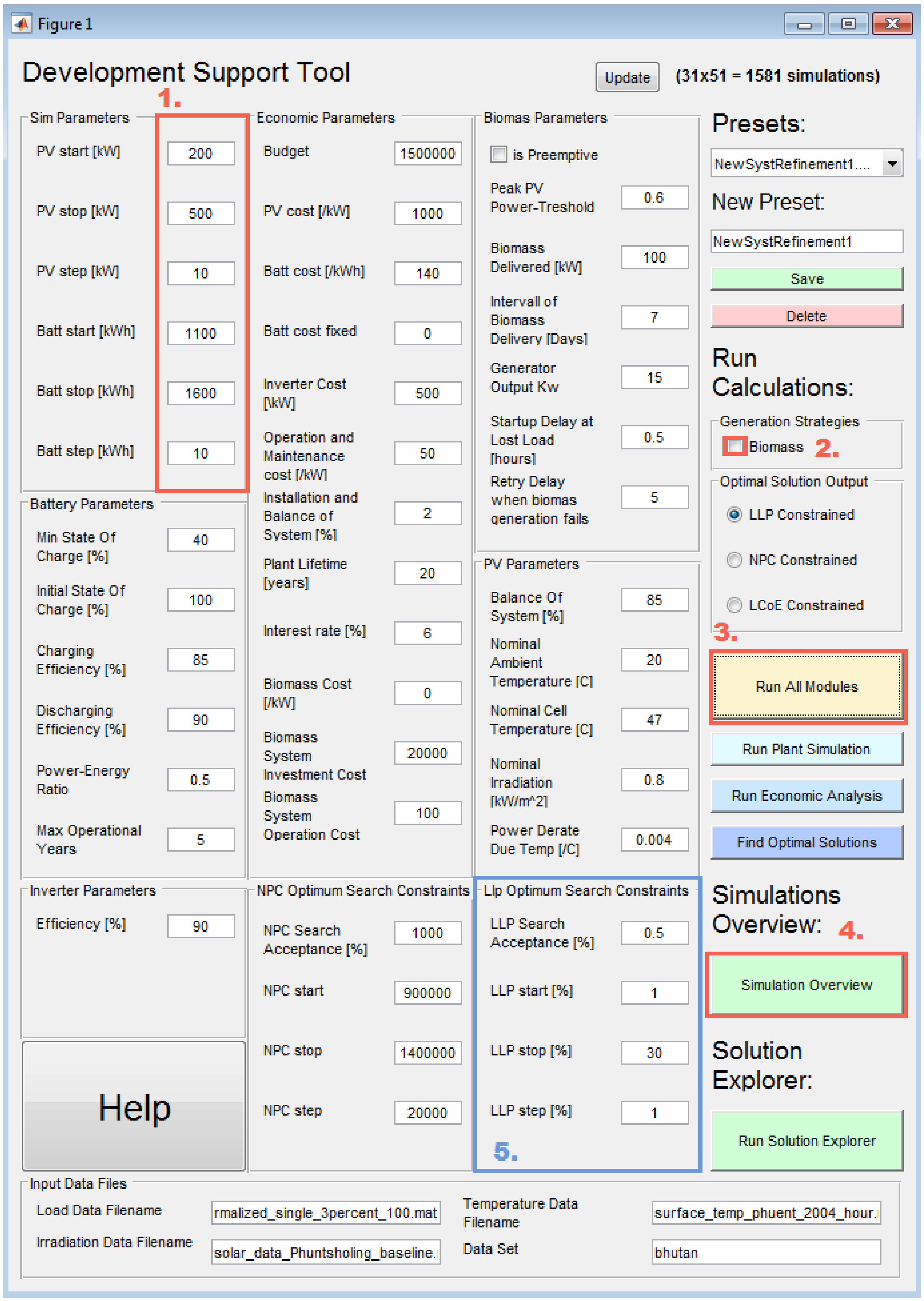


Figure 1

Development Support Tool

Update (31x51 = 1581 simulations)

1.

Sim Parameters

PV start [kW]	200
PV stop [kW]	500
PV step [kW]	10
Batt start [kWh]	1100
Batt stop [kWh]	1600
Batt step [kWh]	10

Battery Parameters

Min State Of Charge [%]	40
Initial State Of Charge [%]	100
Charging Efficiency [%]	85
Discharging Efficiency [%]	90
Power-Energy Ratio	0.5
Max Operational Years	5

Inverter Parameters

Efficiency [%]	90

Help

Economic Parameters

Budget	1500000
PV cost [/kW]	1000
Batt cost [/kWh]	140
Batt cost fixed	0
Inverter Cost [kW]	500
Operation and Maintenance cost [/kW]	50
Installation and Balance of System [%]	2
Plant Lifetime [years]	20
Interest rate [%]	6
Biomass Cost [/kW]	0
Biomass System Investment Cost	20000
Biomass System Operation Cost	100

NPC Optimum Search Constraints

NPC Search Acceptance [%]	1000
NPC start	900000
NPC stop	1400000
NPC step	20000

Biomas Parameters

☐ is Preemptive

Peak PV Power-Treshold	0.6
Biomass Delivered [kW]	100
Intervall of Biomass Delivery [Days]	7
Generator Output Kw	15
Startup Delay at Lost Load [hours]	0.5
Retry Delay when biomas generation fails	5

PV Parameters

Balance Of System [%]	85
Nominal Ambient Temperature [C]	20
Nominal Cell Temperature [C]	47
Nominal Irradiation [kW/m^2]	0.8
Power Derate Due Temp [/C]	0.004

Llp Optimum Search Constraints

LLP Search Acceptance [%]	0.5
LLP start [%]	1
LLP stop [%]	30
LLP step [%]	1

5.

Presets:

NewSystRefinement1....

New Preset:

NewSystRefinement1

Save

Delete

Run Calculations:

Generation Strategies

☐ Biomass 2.

Optimal Solution Output

● LLP Constrained
○ NPC Constrained
○ LCoE Constrained

3.

Run All Modules

Run Plant Simulation

Run Economic Analysis

Find Optimal Solutions

Simulations Overview: 4.

Simulation Overview

Solution Explorer:

Run Solution Explorer

Input Data Files

Load Data Filename	rmalized_single_3percent_100.mat	Temperature Data Filename	surface_temp_phuent_2004_hour.
Irradiation Data Filename	solar_data_Phuntsholing_baseline.	Data Set	bhutan

Figure 3:2 Initial input to the DST Platform

Now the simulation-space can be refined, this means that the simulations that you don't wish to consider should be excluded from the simulation-space while allowing for more detail. This will allow for a faster computation time. The PV and battery size starting points must be guessed if the user don't know what order of sizes are appropriate. The input field is step 1 in Figure 3:2.

The iteration-counter in the upper right corner will track the number of simulations implied, each of these simulations loop 8760 times for each year in the input vector.

The 'Run All Modules' button can be used, or the 'Run Plant Simulation' followed by the 'Run Economic Analysis' buttons which will be queued for execution when each step is finished. The simulation overview button is pressed to explore the simulation-space.

When the 'Simulations Overview' plot appears, you need to determine what qualities that lie within which ranges in the plot. If you happen to be color blind, the 'Rotate 3D' button in the Matlab figure window will allow you to see a contour plot instead of the color scale bar, the Data Cursor will then tell you what values different depths represent.

In Figure 3:3 the user has decided that a LLP of 5% is the maximum accepted LLP, the choice defines the red square, and is reflected in the other plots because they represent the same iterations of PV and battery. The new PV start-value is then chosen precisely as `pvStartOld + iPv*pvStepOld` and the same with battery `battStartOld + jBatt*battStepOld` or chosen roughly. In this case the new simulation range can for example be:

pvStartKw	400
pvStopKw	500
pvStepKw	5
battStartKwh	1350
battStopKwh	1600
battStepKwh	5

The step size has been reduced and the difference between each solution is then smaller. If the resolution is satisfactory and the NPCs acceptable, we can start to explore solutions.

When a battery is replaced, the cost increase. The gradient that defines the costs are also reset around these replacements and the optimal solutions are therefore likely to originate in one of these places. The user will have to consider the quality of his inputs to the DST if chosing a solution close to the jumps. It is probably wise to have some distance to avoid risking that the chosen system in reality has one more battery replacement.

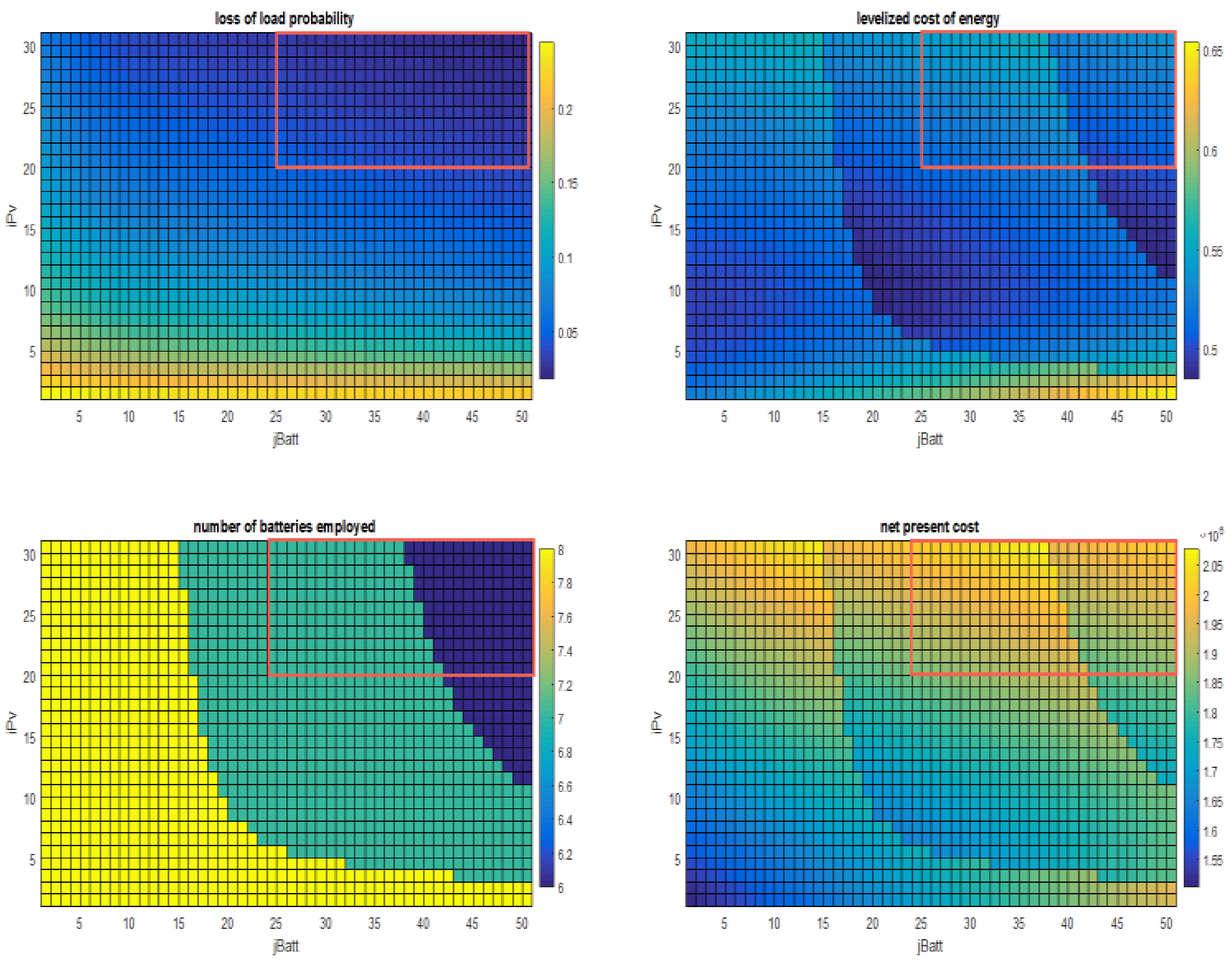


Figure 3:3 Initial Simulation-Space

3.1.3 Solution Exploration

We can now take a better look at the solutions provided within the simulation-space displayed in Figure 3:4.

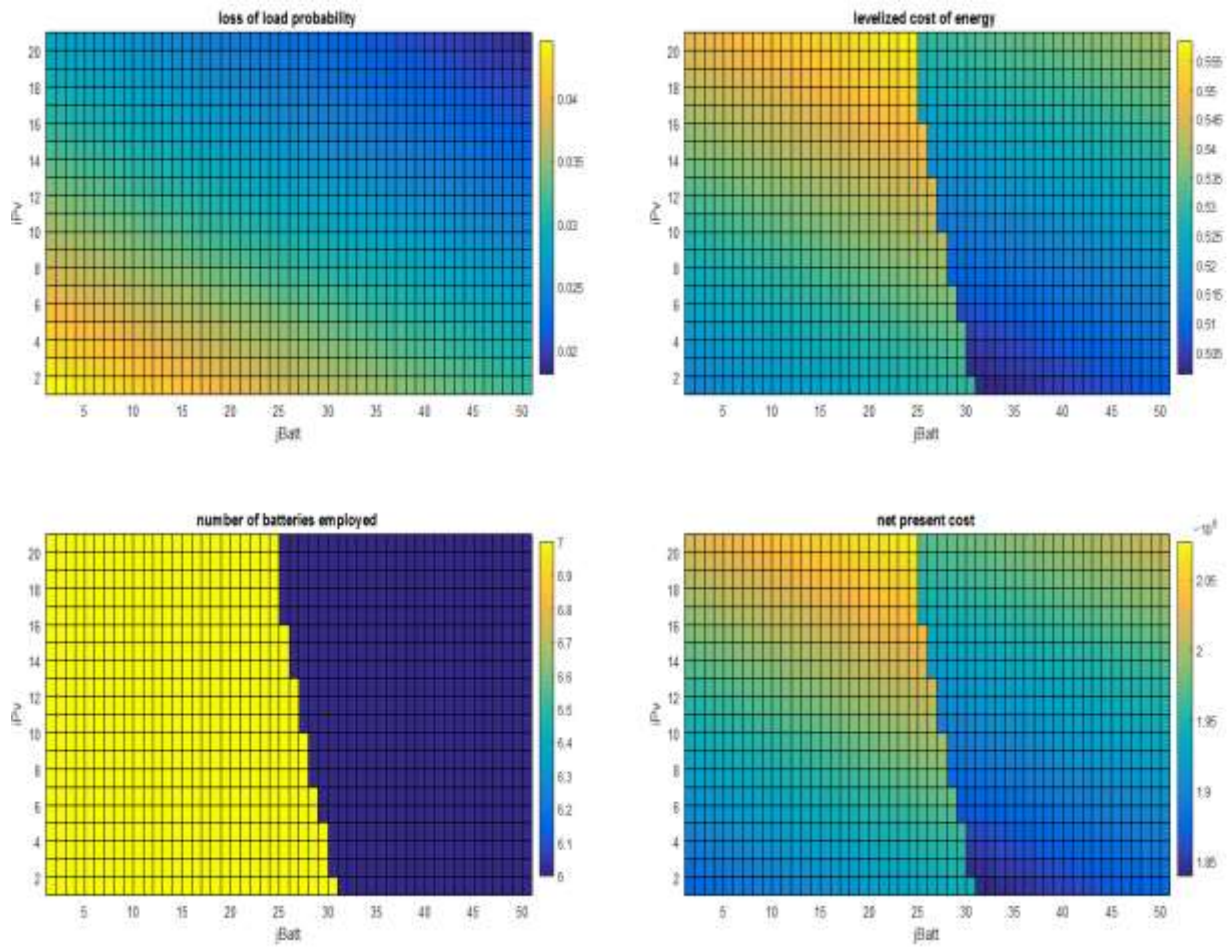


Figure 3:4 Refined Simulation-Space

To open the Solution Explorer, the 'Solution Explorer' button is pressed, or `solution_explorer` typed in the Matlab console. The user can now investigate the qualities of the optimal solutions. By default the 'LLP Constrained' optimums are found, but this can be changed to NPC or LCOE minimum if desired, after changing the optimum search method only the 'Find Optimal Solutions' button should be pressed so that the simulation steps are not unnecessarily repeated.

The 'LLP Optimum Search Constraints' are set to the values in Table 9. The values are set in the field marked 5 in Figure 3:2.

Table 9 LLP Optimum Search Constraints

LLP Search Acceptance	0.1
LLP Start	0
LLP Stop	5
LLP Step	0.5

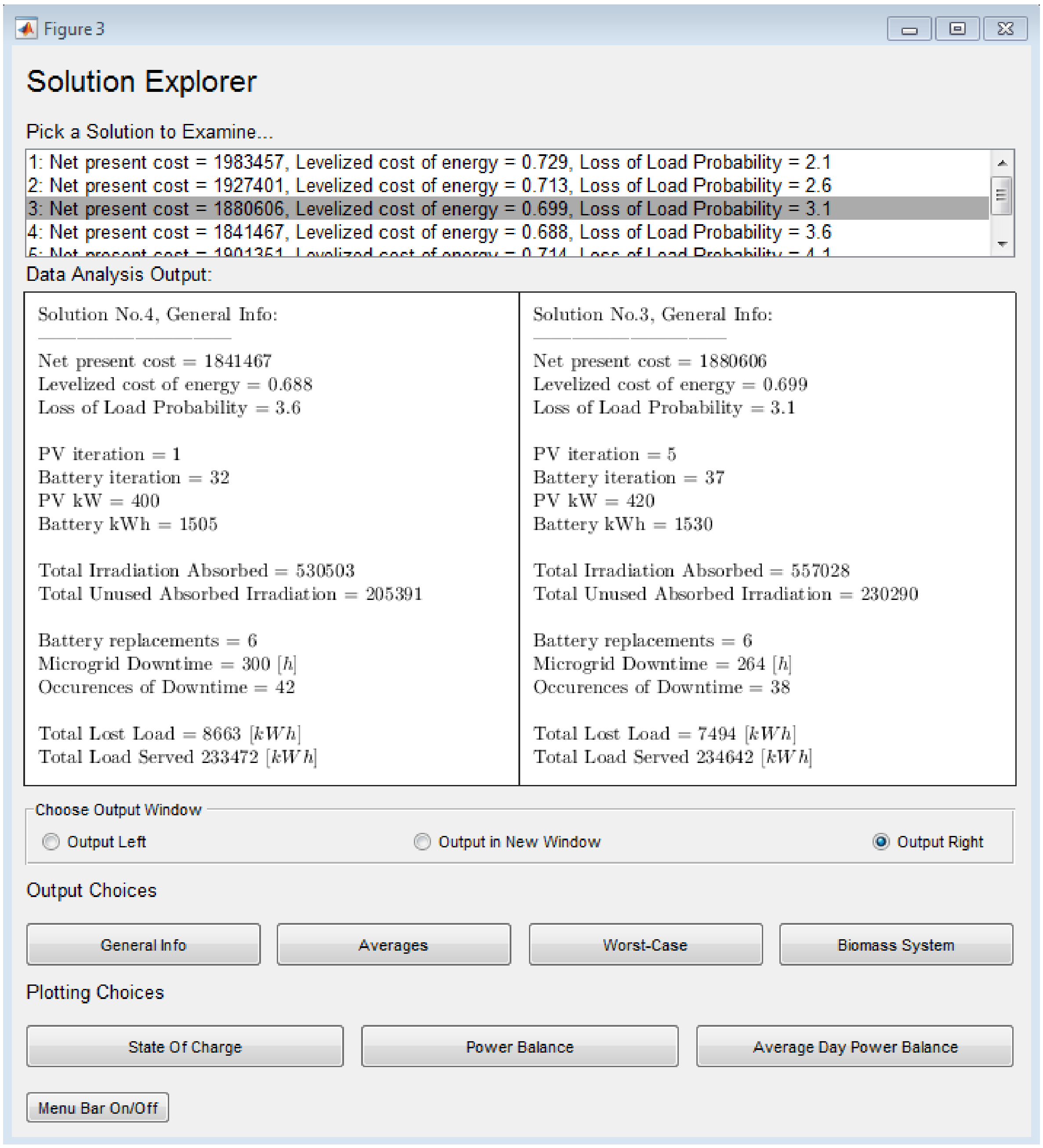


Figure 3

Solution Explorer

Pick a Solution to Examine...

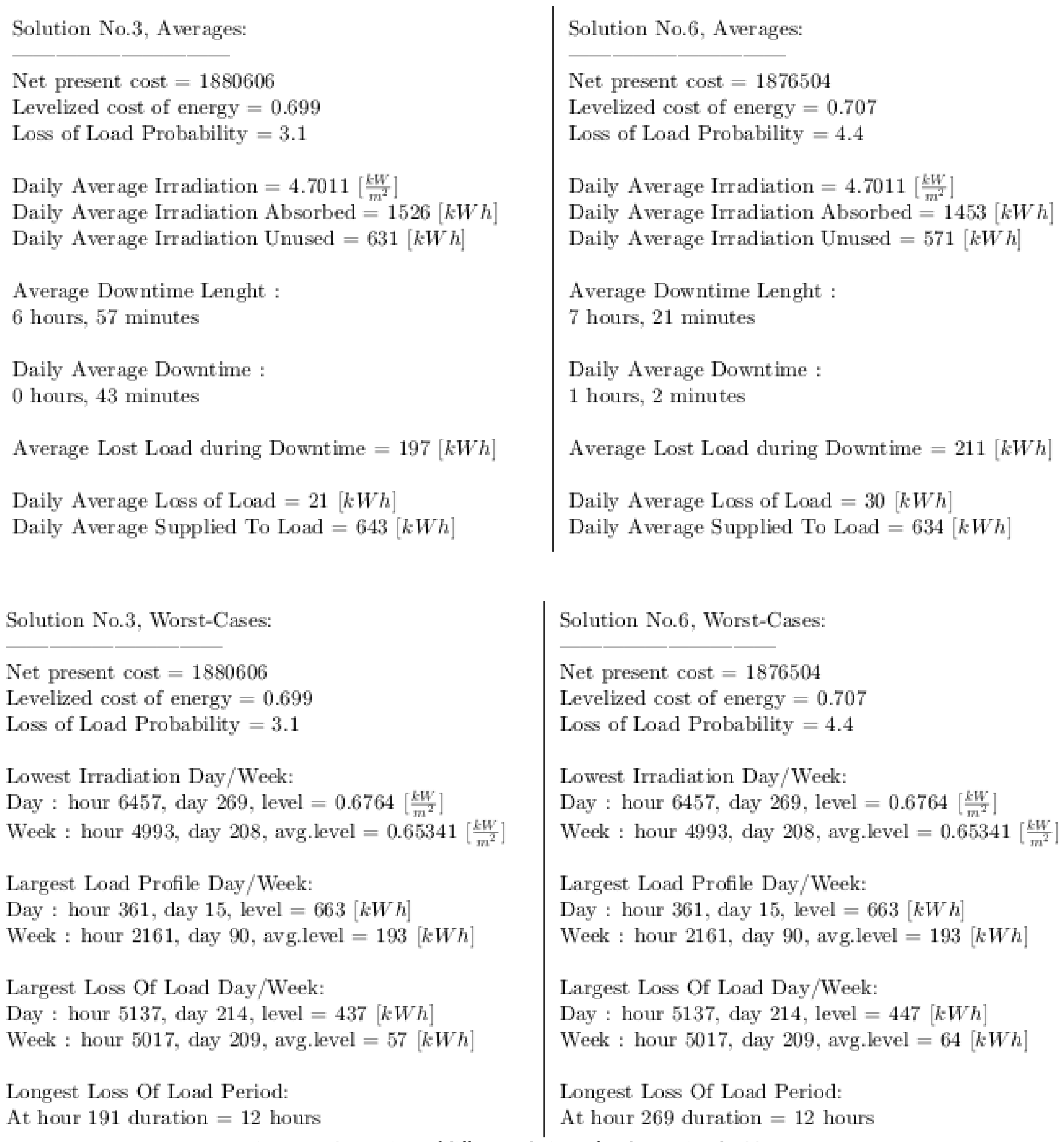
```
1: Net present cost = 1983457, Levelized cost of energy = 0.729, Loss of Load Probability = 2.1
2: Net present cost = 1927401, Levelized cost of energy = 0.713, Loss of Load Probability = 2.6
3: Net present cost = 1880606, Levelized cost of energy = 0.699, Loss of Load Probability = 3.1
4: Net present cost = 1841467, Levelized cost of energy = 0.688, Loss of Load Probability = 3.6
5: Net present cost = 1901351, Levelized cost of energy = 0.714, Loss of Load Probability = 4.1
```

Data Analysis Output:

Solution No.4, General Info:	Solution No.3, General Info:
Net present cost = 1841467	Net present cost = 1880606
Levelized cost of energy = 0.688	Levelized cost of energy = 0.699
Loss of Load Probability = 3.6	Loss of Load Probability = 3.1
PV iteration = 1	PV iteration = 5
Battery iteration = 32	Battery iteration = 37
PV kW = 400	PV kW = 420
Battery kWh = 1505	Battery kWh = 1530
Total Irradiation Absorbed = 530503	Total Irradiation Absorbed = 557028
Total Unused Absorbed Irradiation = 205391	Total Unused Absorbed Irradiation = 230290
Battery replacements = 6	Battery replacements = 6
Microgrid Downtime = 300 [h]	Microgrid Downtime = 264 [h]
Occurences of Downtime = 42	Occurences of Downtime = 38
Total Lost Load = 8663 [kWh]	Total Lost Load = 7494 [kWh]
Total Load Served 233472 [kWh]	Total Load Served 234642 [kWh]

Figure 3:5 Choosing solutions in the Solution Explorer

This step can be difficult because the user must decide what measures of quality should be prioritized. When a solution is chosen in the ‘Pick a Solution to Examine’ window, outputs and plots can be produced quickly by clicking the different buttons. If the user wants to compare more than two windows, the ‘Output in New Window’ radio-button can be toggled on.



Solution No.3, Averages:

Net present cost = 1880606
Levelized cost of energy = 0.699
Loss of Load Probability = 3.1

Daily Average Irradiation = 4.7011 $[\frac{kW}{m^2}]$
Daily Average Irradiation Absorbed = 1526 $[kWh]$
Daily Average Irradiation Unused = 631 $[kWh]$

Average Downtime Lenght :
6 hours, 57 minutes

Daily Average Downtime :
0 hours, 43 minutes

Average Lost Load during Downtime = 197 $[kWh]$

Daily Average Loss of Load = 21 $[kWh]$
Daily Average Supplied To Load = 643 $[kWh]$

Solution No.6, Averages:

Net present cost = 1876504
Levelized cost of energy = 0.707
Loss of Load Probability = 4.4

Daily Average Irradiation = 4.7011 $[\frac{kW}{m^2}]$
Daily Average Irradiation Absorbed = 1453 $[kWh]$
Daily Average Irradiation Unused = 571 $[kWh]$

Average Downtime Lenght :
7 hours, 21 minutes

Daily Average Downtime :
1 hours, 2 minutes

Average Lost Load during Downtime = 211 $[kWh]$

Daily Average Loss of Load = 30 $[kWh]$
Daily Average Supplied To Load = 634 $[kWh]$

Solution No.3, Worst-Cases:

Net present cost = 1880606
Levelized cost of energy = 0.699
Loss of Load Probability = 3.1

Lowest Irradiation Day/Week:
Day : hour 6457, day 269, level = 0.6764 $[\frac{kW}{m^2}]$
Week : hour 4993, day 208, avg.level = 0.65341 $[\frac{kW}{m^2}]$

Largest Load Profile Day/Week:
Day : hour 361, day 15, level = 663 $[kWh]$
Week : hour 2161, day 90, avg.level = 193 $[kWh]$

Largest Loss Of Load Day/Week:
Day : hour 5137, day 214, level = 437 $[kWh]$
Week : hour 5017, day 209, avg.level = 57 $[kWh]$

Longest Loss Of Load Period:
At hour 191 duration = 12 hours

Solution No.6, Worst-Cases:

Net present cost = 1876504
Levelized cost of energy = 0.707
Loss of Load Probability = 4.4

Lowest Irradiation Day/Week:
Day : hour 6457, day 269, level = 0.6764 $[\frac{kW}{m^2}]$
Week : hour 4993, day 208, avg.level = 0.65341 $[\frac{kW}{m^2}]$

Largest Load Profile Day/Week:
Day : hour 361, day 15, level = 663 $[kWh]$
Week : hour 2161, day 90, avg.level = 193 $[kWh]$

Largest Loss Of Load Day/Week:
Day : hour 5137, day 214, level = 447 $[kWh]$
Week : hour 5017, day 209, avg.level = 64 $[kWh]$

Longest Loss Of Load Period:
At hour 269 duration = 12 hours

Figure 3:6 Comparison of different solutions after the running the SAPV system

3.1.4 Problem Solving

How should the system solve its' short comings? The options available here are either that the user decides that some characteristics are more desired than others after inspecting the solutions, that the budget has to cover a smaller load, or that a biomass system should be implemented.

The biomass system takes longer time to compute, depending on the performance of the system one might want to narrow down the simulation space when doing this. The solutions that were produced for the SAPV system can be saved in separate windows to compare the performance and cost of the two technologies.

3.1.5 Decide on a System Purchase

The user can run a biomass simulation with similar ranges as for the previous simulation-space. This round the user might want to see the results with an NPC close to the one found with the SAPV system. The 'Biomass' check box is checked under 'Generation Strategies' and the 'NPC Constrained' radio-button is toggled under 'Optimal Solution Output'.

Utilization of the biomass generation strategy and comparison can be seen in the figures on the next page. This example incorporates a very cheap biomass system as operations cost are very low and biomass is free. Performance is improved substantially for a very small price compared to the same performance increase in a SAPV system. This is due to the flexibility of the generator.

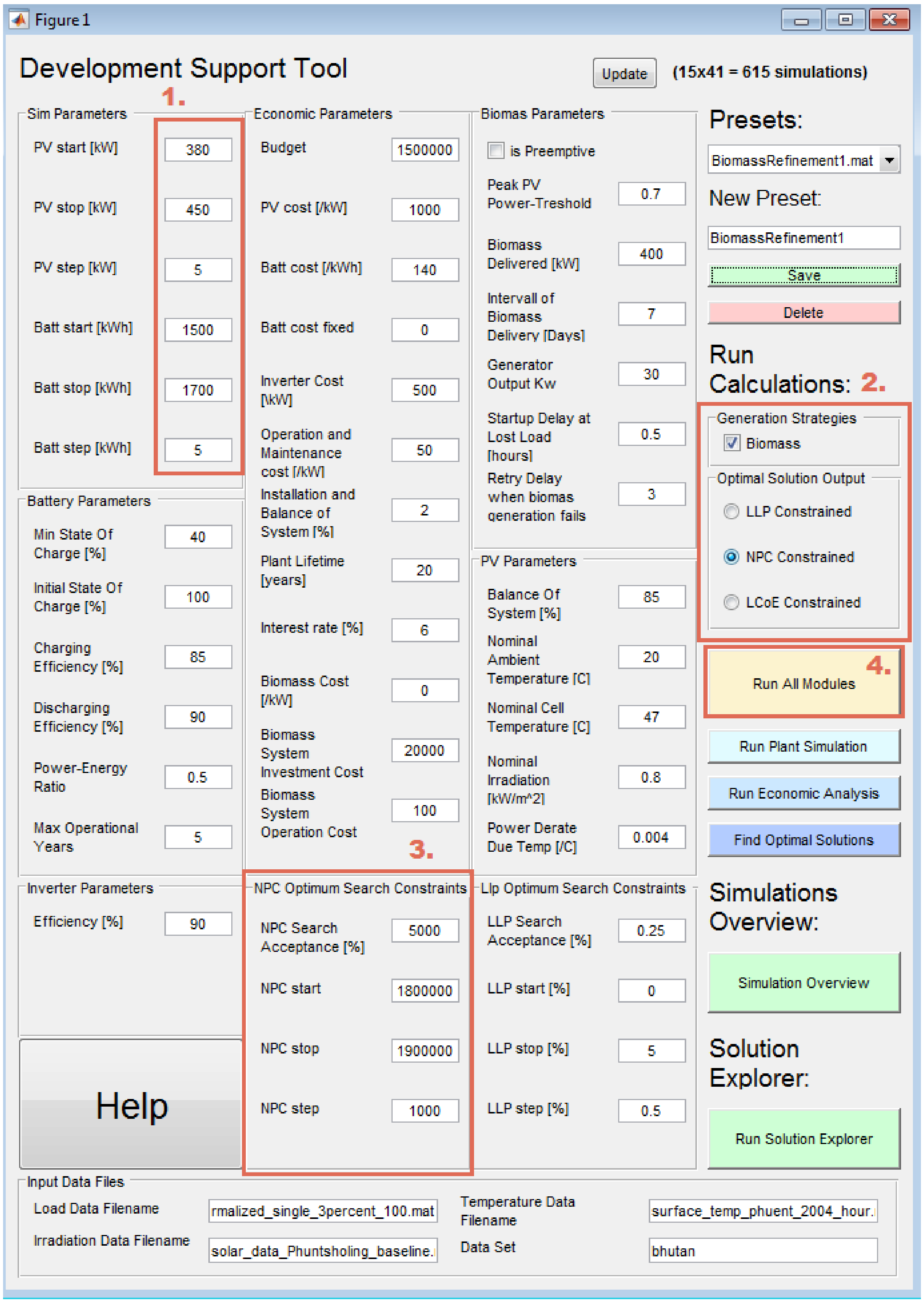


Figure 1

Development Support Tool

Update (15x41 = 615 simulations)

1.

Sim Parameters

PV start [kW]	380
PV stop [kW]	450
PV step [kW]	5
Batt start [kWh]	1500
Batt stop [kWh]	1700
Batt step [kWh]	5

Battery Parameters

Min State Of Charge [%]	40
Initial State Of Charge [%]	100
Charging Efficiency [%]	85
Discharging Efficiency [%]	90
Power-Energy Ratio	0.5
Max Operational Years	5

Inverter Parameters

Efficiency [%]	90

Help

Economic Parameters

Budget	1500000
PV cost [/kW]	1000
Batt cost [/kWh]	140
Batt cost fixed	0
Inverter Cost [kW]	500
Operation and Maintenance cost [/kW]	50
Installation and Balance of System [%]	2
Plant Lifetime [years]	20
Interest rate [%]	6
Biomass Cost [/kW]	0
Biomass System Investment Cost	20000
Biomass System Operation Cost	100

3.

NPC Optimum Search Constraints

NPC Search Acceptance [%]	5000
NPC start	1800000
NPC stop	1900000
NPC step	1000

Biomas Parameters

is Preemptive

Peak PV Power-Treshold	0.7
Biomass Delivered [kW]	400
Intervall of Biomass Delivery [Days]	7
Generator Output Kw	30
Startup Delay at Lost Load [hours]	0.5
Retry Delay when biomas generation fails	3

PV Parameters

Balance Of System [%]	85
Nominal Ambient Temperature [C]	20
Nominal Cell Temperature [C]	47
Nominal Irradiation [kW/m^2]	0.8
Power Derate Due Temp [/C]	0.004

Llp Optimum Search Constraints

LLP Search Acceptance [%]	0.25
LLP start [%]	0
LLP stop [%]	5
LLP step [%]	0.5

Presets: BiomassRefinement1.mat

New Preset: BiomassRefinement1

Save

Delete

Run Calculations: 2.

Generation Strategies

[✓ Biomass]

Optimal Solution Output

LLP Constrained [✓ NPC Constrained] LCoE Constrained

4.

Run All Modules

Run Plant Simulation

Run Economic Analysis

Find Optimal Solutions

Simulations Overview:

Simulation Overview

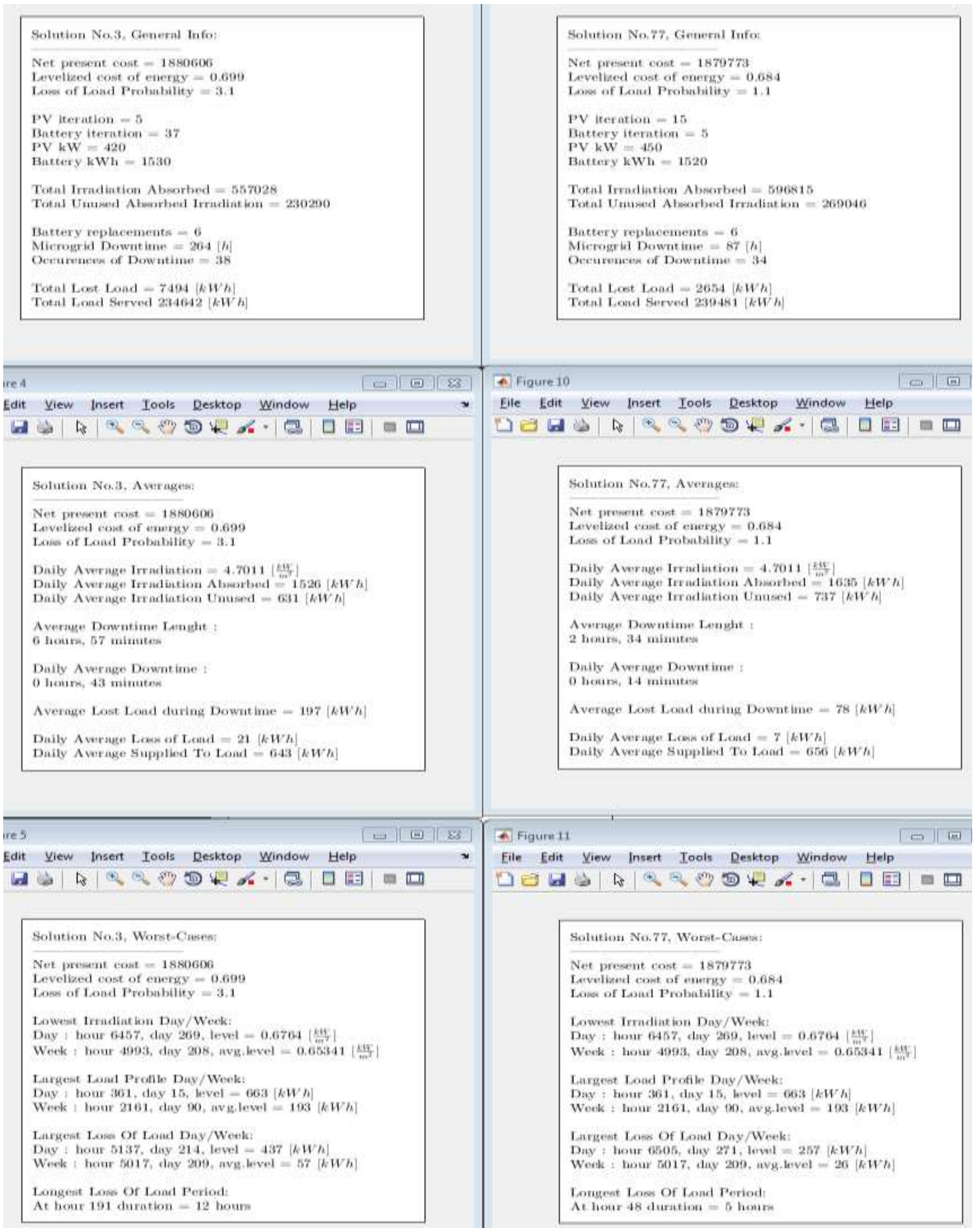
Solution Explorer:

Run Solution Explorer

Input Data Files

Load Data Filename	rmalized_single_3percent_100.mat	Temperature Data Filename	surface_temp_phuent_2004_hour.
Irradiation Data Filename	solar_data_Phuntsholing_baseline.	Data Set	bhutan

Figure 3:7 Including the Biomass Generation Strategy


ure 4
Edit View Insert Tools Desktop Window Help
Figure 10
File Edit View Insert Tools Desktop Window Help
ure 5
Edit View Insert Tools Desktop Window Help
Figure 11
File Edit View Insert Tools Desktop Window Help

```
Solution No.3, General Info:

Net present cost = 1880606
Levelized cost of energy = 0.699
Loss of Load Probability = 3.1

PV iteration = 5
Battery iteration = 37
PV kW = 420
Battery kWh = 1530

Total Irradiation Absorbed = 557028
Total Unused Absorbed Irradiation = 230290

Battery replacements = 6
Microgrid Downtime = 264 [h]
Occurences of Downtime = 38

Total Lost Load = 7494 [kWh]
Total Load Served 234642 [kWh]
```

```
Solution No.77, General Info:

Net present cost = 1879773
Levelized cost of energy = 0.684
Loss of Load Probability = 1.1

PV iteration = 15
Battery iteration = 5
PV kW = 450
Battery kWh = 1520

Total Irradiation Absorbed = 596815
Total Unused Absorbed Irradiation = 269046

Battery replacements = 6
Microgrid Downtime = 87 [h]
Occurences of Downtime = 34

Total Lost Load = 2654 [kWh]
Total Load Served 239481 [kWh]
```

```
Solution No.3, Averages:

Net present cost = 1880606
Levelized cost of energy = 0.699
Loss of Load Probability = 3.1

Daily Average Irradiation = 4.7011 [kW/m²]
Daily Average Irradiation Absorbed = 1526 [kWh]
Daily Average Irradiation Unused = 631 [kWh]

Average Downtime Lenght :
6 hours, 57 minutes

Daily Average Downtime :
0 hours, 43 minutes

Average Lost Load during Downtime = 197 [kWh]

Daily Average Loss of Load = 21 [kWh]
Daily Average Supplied To Load = 643 [kWh]
```

```
Solution No.77, Averages:

Net present cost = 1879773
Levelized cost of energy = 0.684
Loss of Load Probability = 1.1

Daily Average Irradiation = 4.7011 [kW/m²]
Daily Average Irradiation Absorbed = 1635 [kWh]
Daily Average Irradiation Unused = 737 [kWh]

Average Downtime Lenght :
2 hours, 34 minutes

Daily Average Downtime :
0 hours, 14 minutes

Average Lost Load during Downtime = 78 [kWh]

Daily Average Loss of Load = 7 [kWh]
Daily Average Supplied To Load = 656 [kWh]
```

```
Solution No.3, Worst-Cases:

Net present cost = 1880606
Levelized cost of energy = 0.699
Loss of Load Probability = 3.1

Lowest Irradiation Day/Week:
Day : hour 6457, day 269, level = 0.6764 [kW/m²]
Week : hour 4993, day 208, avg.level = 0.65341 [kW/m²]

Largest Load Profile Day/Week:
Day : hour 361, day 15, level = 663 [kWh]
Week : hour 2161, day 90, avg.level = 193 [kWh]

Largest Loss Of Load Day/Week:
Day : hour 5137, day 214, level = 437 [kWh]
Week : hour 5017, day 209, avg.level = 57 [kWh]

Longest Loss Of Load Period:
At hour 191 duration = 12 hours
```

```
Solution No.77, Worst-Cases:

Net present cost = 1879773
Levelized cost of energy = 0.684
Loss of Load Probability = 1.1

Lowest Irradiation Day/Week:
Day : hour 6457, day 269, level = 0.6764 [kW/m²]
Week : hour 4993, day 208, avg.level = 0.65341 [kW/m²]

Largest Load Profile Day/Week:
Day : hour 361, day 15, level = 663 [kWh]
Week : hour 2161, day 90, avg.level = 193 [kWh]

Largest Loss Of Load Day/Week:
Day : hour 6505, day 271, level = 257 [kWh]
Week : hour 5017, day 209, avg.level = 26 [kWh]

Longest Loss Of Load Period:
At hour 48 duration = 5 hours
```

Figure 3:8 Several Windows from the SAPV simulation and Biomass simulation compared

3.2 Summary

In this thesis I have presented the design and development of a DST that has the functionality of a finished product and is ready for beta testing. The changes made in the code structure has allowed for expansion of the tool with both biomass functionality and a graphical user interface.

The graphical user interface has a user friendly design and a wide array of functionality. The simulation platform and the solution explorer have been explained. The outputs provided by the GUIS will help expand the users understanding of the simulated scenarios and microgrid operation, in particular the dynamic between input sources, batteries and load.

The biomass system implementation and functionality has been tested on the data training-set and displays the intended functionality. The result is that the DST will allow users to test the use of a biomass generation system in parallel to their prospected purchases, and what implications this carry.

Finally a use case has been presented to explain the general process of using the DST

3.3 Discussion

3.3.1 Python Implementation

Using Matlab in continuation of the program was a fast way to get started with the program. The certainty of maintained functionality and the comparison of old and new has been central to the reconstruction and further development of the tool.

A python implementation still remains to be done, this is not untypical of a software development process, but there is redundant work implied.

The intention has been all along that a rewrite to Python should take place sooner or later. Due to uncertainties in development strategies and time frame, the work continued in Matlab, this was a mistake because of the extra work implied now that the project is expanded. The reimplementation in Python should have taken place as the first action in this thesis. The validation would have been slightly more problematic but this does not justify the inaccessibility that a Matlab license represent to those with meager financial support.

Matlab is only available to those with a license. This means that students and university staff will temporarily be the only group that are likely to contribute to this work. The tool cannot be shipped efficiently to users before the rewrite has taken place.

3.3.1.1 Technical Implications

The code structure would have been different if a python implementation had been chosen initially. All python operations are call-by-reference since the identifiers are pointers by default. This functionality would have allowed for increased modularity in the program.

A call-by-value function with a large input is unacceptable in the simulations module since there are nPv*nBatt*nHours iterations that would have to copy and erase these values. This implies a large computational overhead which would not have been a problem in python, because of the pointer behavior mentioned above.

This could have allowed for increased modularity, which again increase the code quality and development friendliness.

3.4 Recommendation for Further Work

3.4.1 Python Implementation

The tool will have to be rewritten to python. The reason for this is explained in the 'Discussion' section.

I recommend that this is the next iteration of the DST project, no further actions should be taken until this is done, unless if it's restricted to analysis of existing code.

3.4.1.1 Increased modularity

As the tool is now there is some repeated code especially between the simulation modules for SAPV and biomass, they both have the same battery operation algorithm.

The problem with this is that if changes to one of the modules, has to be performed on both of them. This is unnecessary if we can locate the same functionality to the same function. The requirement is that the function is quick and preferably call by reference. Call by value functions should be avoided considered the high amount of repetitions.

3.4.1.2 New GUI start-platform suggestion

The GUI start platform can be a button panel that calls the different feature GUIs in new popup windows. A 'new window for each feature' would mean that adding a new feature only meant adding a button to the platform window when adding a feature. This can also be expanded to a dynamically generated platform which only has the features that the user needs.

The implementation of this is easily made with the existing design and should be considered as a design choice of the python rewrite.

3.4.2 Finding Borders Analytically

The DST relies initially on producing a very large solution space. The user will usually have to create the simulation space more than once, to pinpoint the desired resolution for analysis. A user might not familiar with the economic proportions of the PV and battery sizes necessary for the load in question. If this is the case, the user has to start with a wide solution space and then iterate to the desired scope and resolution for analysis.

The outputs used from the simulation module for the economic analysis is the `yearsBattOperational` value, stating how long a battery is functional, and the `lossOfLoadTot` for the time series. The years a battery is operational depends on the wear and tear of the system. This value does have a worst case outcome, if every cycle is a full discharge to the `minStateOfCharge` level. The following analytical expression will calculate this worst case life span.

$$(9) \qquad YO_{worst\ case} = \frac{daysInTimeSeries}{cycles\_to\_failure(1 - minStateOfCharge)}$$

YO stands for years operational, the years a battery is operational before being replaced. Using this in the economic analysis we achieve a degree of freedom to find sizes for PV and battery expressed by $B\,kWh$ and $PV\,kW$ in the following equation:

$$(10) \qquad IC = ((battCost_{/kw}\ battKw) + (pvCost_{/kw} pvKw) + inverterCost)(1 + IBoS)$$

$$(11) \qquad OeMeR = B_{\frac{cost}{kWh}} B_{kWh} \left(\sum_{i=1}^{i*YO<PL} \frac{1}{(1+r)^{i*YO}} - \frac{\frac{YR}{YO} B_{\frac{cost}{kWh}} B_{kWh}}{(1+r)^{PL}} \right) + \frac{I_{cost}}{(1+r)^{PL/2}} + \sum_{y=1}^{PL} \frac{OeM_{\frac{cost}{kw}} PV_{kW}}{(1+r)^{y}}$$

$OeMeR$ is the net present cost of operation, maintenance and replacement of batteries. $B\ cost/kWh$ is the cost of batteries per kWh. $I\ cost$ is the inverter cost. PL is the plant lifetime, YR is the years remaining of battery life after plant lifetime has expired, and $OeM\ cost/kW$ is the yearly operation and maintenance cost per kW of PV capacity.

If we assume that IC and OeMeR are known, the equations have 2 unknown variables and the system has a solution. A problem is that the ratio between OeMeR and IC will affect the solution in different ways. This means that the user must know what ratio between the two is to be expected. This is problematic. If we have that $\frac{IC}{OeMeR} \approx constant$ in systems in general, we can exploit that:

$$(12) \qquad NPC = IC + OeMeR$$

This will enable the user to know only the NPC to get a minimum performance size for this budget. A histogram from the test data shows that the ratio is varying more than 10% as seen in Figure 3:9. This might be too much to make any useful estimates.

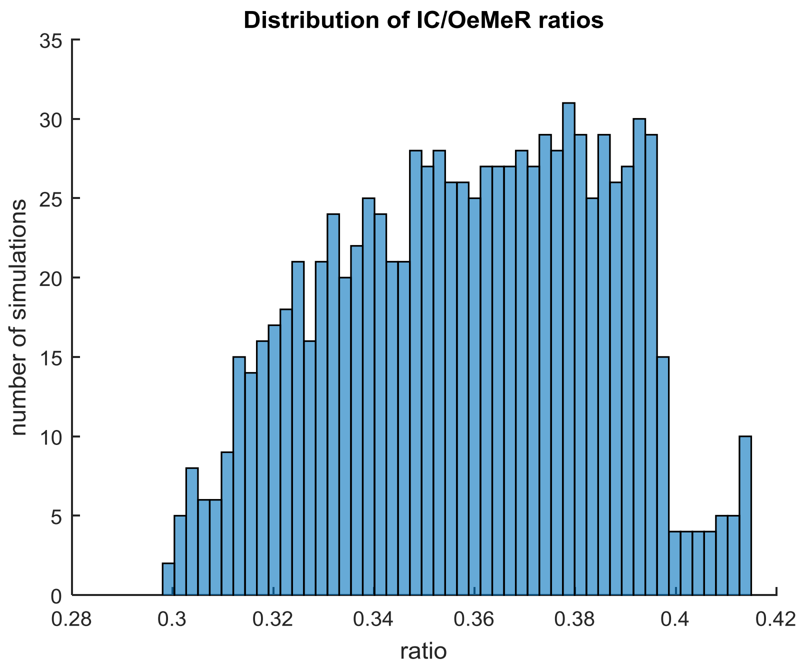


Figure 3:9 Distribution of IC/OeMeR ratios

3.4.2.1 Suggestion for Future Work

A possible connection between the $IC/OeMeR$ ratios and LLP is displayed in Figure 3:10. A general analytical expression may be found from several test-data sets empirically. The results would be useful if precision is consistently higher than 10% deviations. Using these results, a component minimum size can be returned to the user without any exploration of the simulation space.

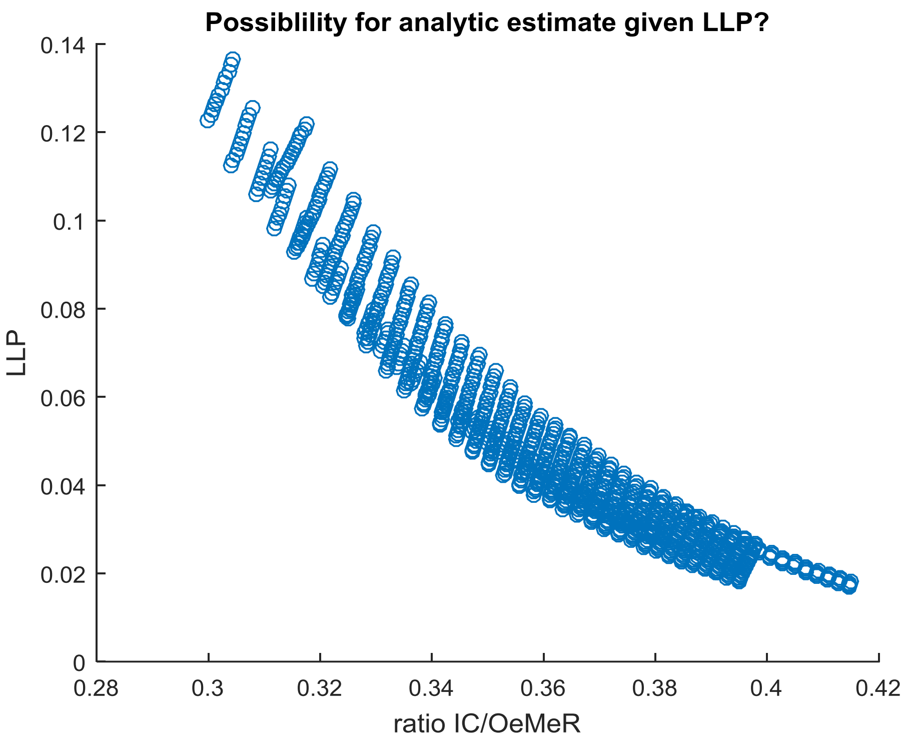


Figure 3:10 Dependency between LLP and IC/OeMeR ratio

4: Appendences

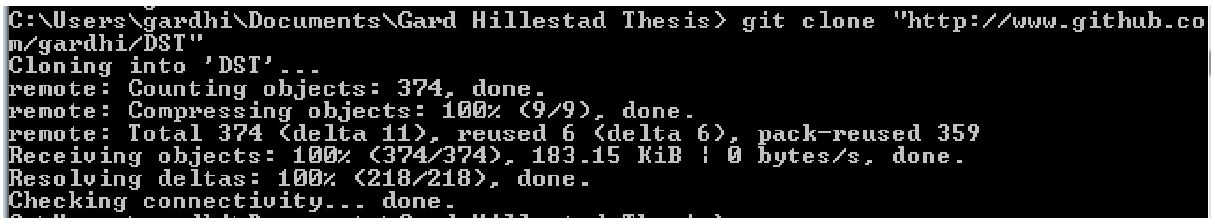
Appendix A The DST Source-Code

The DST source-code is found online on https://github.com/gardhi/DST. To inspect the code one only need to have a browser and internet, to run and test the code the code needs to be downloaded through Git.

Git is a version control program that allows for agile collaboration on projects, the use of Git is established throughout the programming world and widely used by the majority of professionals.

Instructions on Downloading and testing the DST

1. Install a Git client from anywhere.
   For a desktop easy-to-use interface download github here:
   https://desktop.github.com/
2. Open the Git Shell.
   If you are using windows, press the windows-button and write: Git Shell
3. Clone the repository.
   While using the git shell, navigate to the folder where you want the project to be placed. Cloning basically means downloading, this is done in the current folder by writing: `git clone "http://www.github.com/gardhi/DST"`
   It should look similar to this:



```
C:\Users\gardhi\Documents\Gard Hillestad Thesis> git clone "http://www.github.co
m/gardhi/DST"
Cloning into 'DST'...
remote: Counting objects: 374, done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 374 (delta 11), reused 6 (delta 6), pack-reused 359
Receiving objects: 100% (374/374), 183.15 KiB | 0 bytes/s, done.
Resolving deltas: 100% (218/218), done.
Checking connectivity... done.
```

4. Run the code.
   You now have the entire project in the destination folder, open it in Matlab and include the folder DST to path and all subfolders.
   To run the DST, simply write "`dst_platform`" **in the Matlab console window once the folders are on the Matlab path.**
5. Contribute!
   To work on the project, make a branch either in GitHub desktop or in the shell. When your feature is finished it can be added to the project by a merging your branch with the master branch, and then pushing to the origin, or by a practice determined by project administrator. In this case a pull request is required, documentation is found online.

Appendix B The Reference Manual Source-Code

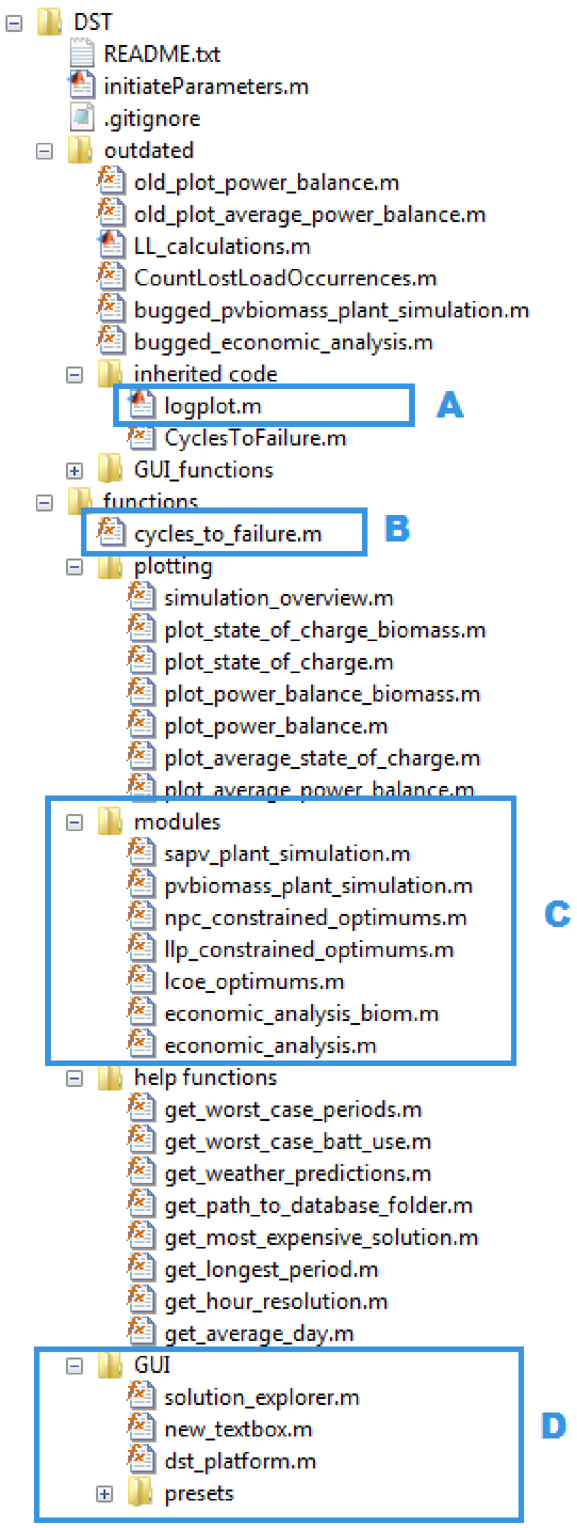
The reference manual has an online repository on github just like the DST. The repository address is https://github.com/gardhi/dstReference and can be cloned just like described in Appendix A.

For a faster way to inspect the source code one can visit the webpage of my NTNU server. http://gardhi.github.io/dstReference/referenceManual.html and right-click, Save As and choose format to be webpage complete.

The ownership of repositories will be transmitted to any successors of the project.

Appendix C Important File Locations

A. The location of logplot.m, the inherited code from the previous collaborators.
B. The cycles to failure function which should be replaced by a function supplied by the battery manufacturer.
C. The folder containing the main modules of the DST
D. The folder containing the GUI functions.



```
DST
    README.txt
    initiateParameters.m
    .gitignore
    outdated
        old_plot_power_balance.m
        old_plot_average_power_balance.m
        LL_calculations.m
        CountLostLoadOccurrences.m
        bugged_pvbiomass_plant_simulation.m
        bugged_economic_analysis.m
        inherited code
            logplot.m
            CyclesToFailure.m
        GUI_functions
    functions
        cycles_to_failure.m
        plotting
            simulation_overview.m
            plot_state_of_charge_biomass.m
            plot_state_of_charge.m
            plot_power_balance_biomass.m
            plot_power_balance.m
            plot_average_state_of_charge.m
            plot_average_power_balance.m
        modules
            sapv_plant_simulation.m
            pvbiomass_plant_simulation.m
            npc_constrained_optimums.m
            llp_constrained_optimums.m
            lcoe_optimums.m
            economic_analysis_biom.m
            economic_analysis.m
        help functions
            get_worst_case_periods.m
            get_worst_case_batt_use.m
            get_weather_predictions.m
            get_path_to_database_folder.m
            get_most_expensive_solution.m
            get_longest_period.m
            get_hour_resolution.m
            get_average_day.m
        GUI
            solution_explorer.m
            new_textbox.m
            dst_platform.m
            presets
```

Appendix D The Rainflow Counting Algorithm

As there were some doubts about this implementation this study of the Rainflow Counting algorithm used in the code was conducted to verify it's correctness.

It is highly important to consider the calculations of the battery replacement interval. Replacing batteries are costly and amounts to a majority of the NPC when dealing with SAPV systems. Additional importance can be considered since the DST is intended for MSEs that are likely to rely on smaller economical margins, and larger personal risks.

Batteries are replaced when they have gone through a certain amount of charging / discharging cycles, the method used in the DST is the Rainflow Counting Algorithm. The Rainflow Counting algorithm was initially to account for stress exposure in materials. One cycle is one instance of full stress exposure, a partial cycle is an instance of a partial stress exposure. The algorithm can be used to account for stress in batteries (You and Rasmussen 2011) and other appliances that go through similar wear.

I. Original Algorithm: (You and Rasmussen 2011)

1. Initiate a vector in encounter of a stress local minimum.
2. Note increase in stress during rainfall of vector
3. Count occurrences of ranges as one cycle
4. Sum the expended partial cycles for every stress level accounted for

$$(13) \qquad LCon = \sum_{DOD=1/n}^{DOD=100\%} \frac{Nc(DOD)}{No(DOD)}$$

Where n represents the number of bins chosen in the study; $Nc(DOD)$ represents the number of consumed partial cycles at a given DoD level, derived by counting in the corresponding period; $No(DOD)$ represents the maximum number of partial cycles that can be performed before battery failure at that DoD level.

$$(14) \qquad ExpL = \frac{1}{LCon} Tp$$

Where $ExpL$ denotes the expected lifetime of the BS, and Tp represents the length of the counting time period.

II. Implemented Algorithm:

1. Discover discharge valleys (can currently only occur after 8 consecutive hours of discharging)
2. Calculate cycles to failure for each occurring DoD:

$$(15) \quad cycles\_to\_failure = 15790e^{-11.96\,DOD} + 2633e^{-1.699DOD}$$

This is the result of fitting a typical lead-acid battery cycles to failure vs. DoD characteristics. This equation has to be replaced or modified by the user.

3. Accumulate cycles to failure during simulation time.

$$(16) \quad rainflowCounter = \sum_{0}^{nValleys} \frac{1}{cyclesToFailure}$$

4. Find replacement interval of batteries:

$$(17) \quad lifespanBatteryThisSystem = \frac{1}{rainflowCounter}$$

A requirement for finding the lifespan in years, is that the spendage of lifetime fractions are summed over one year precicely. This way we get the amount of years that the battery need. The expression from point 2 has to be replaced or modified if the user desire precision in the economic analysis.

III. Considerations

Every input related part of the DST should assume a generic form. This implementation assumes the that the rainflowCounter will keep counting for exactly one year. A more generic calculation is implemented in the rewritten DST as seen below.

$$(18) \quad lifespanBatteryThisSystem = \frac{\frac{nHoursInDataseries}{nHoursInOneYear}}{rainflowCounter}$$

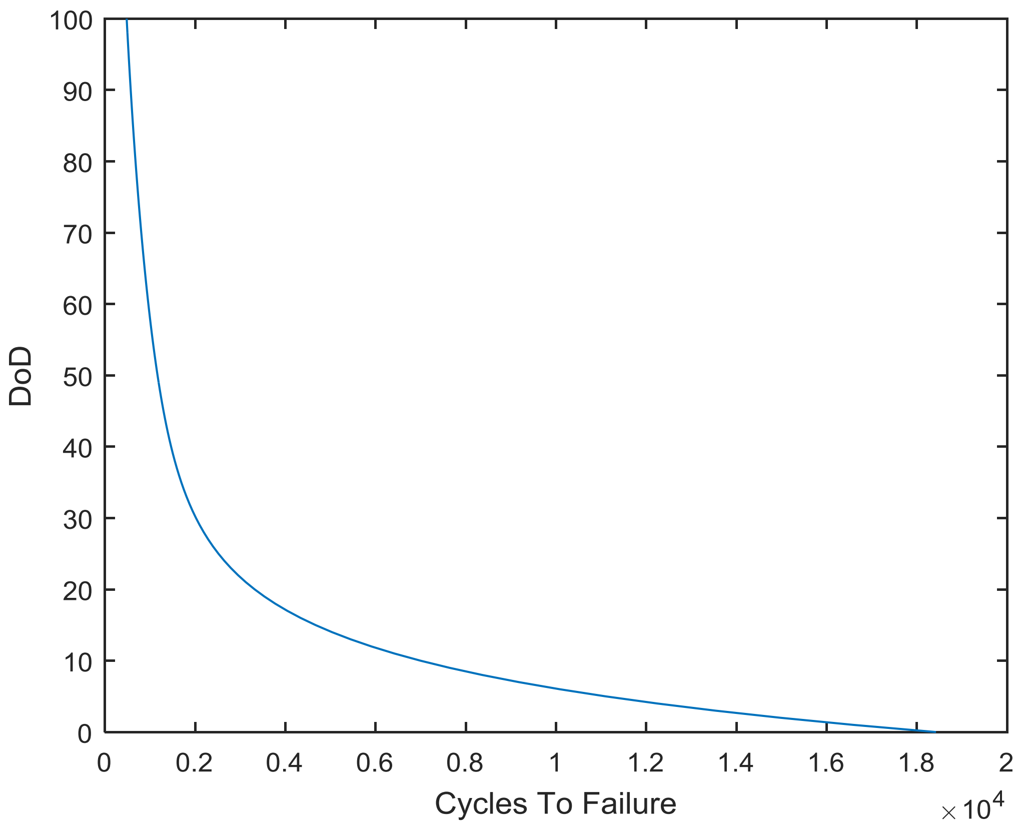


Figure 4:1: Plotting DoD(Cycles To Failure) w/initial parameters.
There are not sufficient points of DoD in the intervall 0-20% to represent the Cycles to Failure accurately.

The cyclesToFailure is calculated for every instance of DoD valley that occurs, which has an complexity of **O(n).** This resolution in cycle values might not be required considered the massive abstraction level of which we are operating. A future extension of the DST will at some point trigger far more occurences of DoD, before a simulation is considered complete. Users might want to input longer time series or change the 8 hour consecutiveness condition. In earlier versions of the algorithm (Downing and Socie 1982), the method proposed is preemptively generating a table of Cycles To Failure for every DoD percentage.

If the rate of change in DoD is not too close to zero, then the points of DoD will sufficiently describe the Cycles To Failure, as there are many values of DoD for each value of Cycles To Failure. By inspecting Figure 4:1, here the points of DoD percentage per cycles varies from 0.0001 to 0.01. It would not be safe generate an array with 1% resolution, the DoD rate is too low until about 20%. We can not guarantee that the DoD valleys wont occur in the 0%-20% range, even though it is less typical of a discharge cycle.

One solution is to increase the resolution of the DoD values to 0,1%, or perhaps smaller, in the precalculated array. The increased resolution will decrease computational gain. The DST overall complexity is $O(n^3)$, and the algorithm triggers a maximum of 3 times per day. Improvement of omputation-time when employing a precalculated array is too small, compared to the reduced precision of the implementation.

The 8-hour consecutiveness condition is similarly implemented in (Downing and Socie 1982), here the condition is 3 points. This can be understood as a lowpass filtering of the input. The physical consequences of potential “flutter” between cycles is neglectable. Additionally, charging cycles have daily periods, as seen in Figure 2:16. The charging cycle follows the irradiation cycle and will not drop below 24 hours. Some discussion is made on this in the Thesis.