|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| SoC | State of Charge |
| DoD | Depth of Discharge |
| DST | Development Support Tool |
| LoL | Loss of Load |
| MSSE | Micro and Small Scale Enterprises |
| PV | Photovoltaic |

# Introduction

A general introduction goes here.

## Background

* Problem Formulation
* Literature Survey
* What remains to be done.

## Objectives

The main objectives of this Master’s project are:

1. To rewrite, scrutinize, and test the DST
2. To develop new features for the DST

## Limitations

## Approach

Listing the main parts of the methodology, stating why each step was important. Check the disposition note in evernote

## Structure of the Report

# Methodology

Part 1, present; part 2 explain how. Results

## Code Review and Improvement

The DST will be an Open Source collaboration. This implies some demands on the code quality of the project.

### Naming Conventions, Commenting and Formatting

Table 1: the Complete List of Variable Name Changes in Rewritten DST

|  |  |
| --- | --- |
| Names | |
| Before Rewrite | After Rewrite |
| x\_llp | SimulationParameters.llpSearchTargets |
| min\_PV | SimulationParameters.pvStartKw |
| max\_PV | SimulationParameters.pvStopKw |
| step\_PV | SimulationParameters.pvStepKw |
| min\_batt | SimulationParameters.battStartKwh |
| max\_batt | SimulationParameters.battStopKwh |
| step\_batt | SimulationParameters.battStepKwh |
| n\_PV | SimulationParameters.nPvSteps |
| n\_batt | SimulationParameters.nBattSteps |
| irr | SimulationInputData.irradiation |
| filename | (deprecated) |
| Load | SimulationInputData.load |
| T\_amb | SimulationInputData.temperature |
| EPV | (deprecated) |
| ELPV | SimulationOutputs.pvPowerAbsorbedUnused |
| LL | SimulationOutputs.lossOfLoad |
| batt\_balance | SimulationOutputs.neededBattOutputKw\*\* |
| num\_batt | EconomicAnalysisOutput.nBattEmployed |
| SoC | SimulationOutputs.stateOfCharge\*\* |
| IC | EconomicAnalysisOutput.investmentCost |
| YC | EconomicAnalysisOutput.operationMaintenanceReplacementCost\*\* |
| BoS | EconomicParameters.balanceOfSystem |
| nomAmbientTemp | PvParameters.nominalAmbientTemperatureC |
| nomCellTemp | PvParameters.nominalCellTemperatureC |
| coeff\_T\_pow | PvParameters.powerDearteDueTemperature |
| irr\_nom | PvParameters.nominalIrradiation |
| SoC\_min | BatteryParameters.minStateOfCharge |
| SoC\_start | BatteryParameters.initialStateOfCharge |
| eff\_char | BatteryParameters.chargingEfficiency |
| eff\_disch | BatteryParameters.dischargingEfficiency |
| max\_y\_repl | BatteryParameters.maxOperationalYears |
| batt\_ratio | BatteryParameters.powerEnergyRatio |
| eff\_inv | InverterParameters.efficiency |
| costPV | EconomicParameters.pvCostKw |
| costINV | EconomicParameters.inverterCostKw |
| costOeM | operationMaintenanceCost |
| costOeM\_spec | EconomicParameters.operationMaintenanceCostKw |
| coeff\_cost\_BoSeI | EconomicParameters.installBalanceOfSystemCost |
| costBatt\_coeff\_a | EconomicParameters.battCostKwh |
| costBatt\_coef\_b | EconomicParameters.battCostFixed |
| LT | EconomicParameters.plantLifetime |
| r\_int | EconomicParameters.interestRate |
| PV\_i | iPv |
| Pvpower\_i | iPvKw |
| T\_cell | pvTemperature |
| eff\_cell | cellEfficiency |
| P\_pv | SimulationOutputs.pvPowerAbsorbed\*\*\*\* |
| batt\_balance | SimulationOutputs.neededBattOutputKw\*\*\* |
| batt\_i | jBatt |
| Pow\_max | battMaxPowerFlow |
| Den\_rainflow | nMaxPartialCycles |
| DoD | depthOfDischarge |
| cycles\_failure | SimulationOutputs.sumPartialCyclesUsed\*\* |
| flow\_from\_batt | SimulationOutputs.battOutputKw\*\* |
| costBatt\_tot | EconomicAnalysisOutputs.battCostTot\*\* |
| peak | loadPeakKw |
| costINV\_tot | EconomicAnalysisOutputs.inverterCostTot |
| costBoSeI\_tot | EconomicAnalysisOutputs.installBalanceOfSystemTotCost |
| years\_to\_go\_batt | battOperationalYears |
| NPC | EconomicAnalysisOutputs.netPresentCost |
| CRF | EconomicAnalysisOutputs.capitalRecoveryFactor |
| total\_loss\_load | SimulationOutputs.lossOfLoadTot\*\* |
| LCoE | EconomicAnalysisOutputs.levelizedCostOfEnergy |
| LLP | SimulationOutputs.lossOfLoadProbability |
| MA\_opt\_norm\_bhut\_jun15\_20\_10(1) | OptimalSolutions.lossOfLoadProbabilities |
| MA\_opt\_norm\_bhut\_jun15\_20\_10(2) | OptimalSolutions..netPresentCosts |
| MA\_opt\_norm\_bhut\_jun15\_20\_10(3) | OptimalSolutions.pvKw |
| MA\_opt\_norm\_bhut\_jun15\_20\_10(4) | OptimalSolutions.battKwh |
| MA\_opt\_norm\_bhut\_jun15\_20\_10(5) | OptimalSolutions.levelizedCostsOfEnergy |
| MA\_opt\_norm\_bhut\_jun15\_20\_10(6) | OptimalSolutions.investmentCosts |
| budget | EconomicParameters.budget |
| 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 | |

### Modularization and Encapsulation

Describe the concept

Describe the implementation

Example the SoC and power balance average implementation

The rewritten DST has key variables stored for later calculations and plots, where the previous DST would only keep these in memory during simulation. Storing the simulation history will enable the user to only run the simulation once, and use these results until the simulation parameters need to change. I.e. when the current solution become unsatisfactory.

### Testing

The new

Years to go batt example, if tests were ready one would be able to spot a jump in prices.

### Plotting

Plots are important in the DST. Users can visualize the hourly levels in order to understand how the system work, but also when and how much the micro grid fail. The previous DST would not allow any single function to run without running entire simulations. In the case of plotting, one would have to simulate the system in order to find what solution (combination of PV and battery) one wished to examine, then one would have to hard-code the DST to plot this solution as it was iterated over. This continuous hard coding and redundant computation was vastly time consuming. The rewritten DST has one function for each plot.

#### State of Charge

The existing SoC plotting was not modified extensively. It was made a module (function) and renamed, and will also access any given simulation. This is a very useful plot, it gives the user an intuitive impression of how the micro grid performs in simulation.

The green plot line is power from the PV that neither serves load nor charges batteries because the batteries are already fully charged. The red plot line indicate loss of load in kW as there is no sufficient output to serve the load.

|  |
| --- |
| Figure 2:1: A full year overview from the State Of Charge plot |

|  |
| --- |
| Figure 2:2: A more detailed part of the State of Charge plot |

#### Power Balance

The idea that one should be able to track.

Figure 2:3: The previous attempt at making a power balance plot in the outdated DST

Figure 2:4: The new power balance plotting implementation.

This function will plot the average day, the two functions are identical except the %averages part.

Figure 2:5: The get\_daily\_average function will return an average for each hour of a day.

This is a generic algorithm that both improves readability and ease of implementation.

|  |
| --- |
| Figure 2:6: The old attempt at making a power balance plot  The ‘Energy from PV’ and ‘Energy flow from battery’ is in reality the potential energy absorbed by the PV, and the needed battery output to meet the load demands respectively. |

|  |
| --- |
| Figure 2:7: The same error that is explained in Figure 2:6 is replicated in the daily average version. |

|  |
| --- |
| Figure 2:8: The new power balance plot.  Here the yellow plot lines represent the power that is actually utilized from the PV. The blue line represent the net power to serve the load from PV and batteries. The red line is the load demand in Kw. The blue line will overlap the red line whenever the load demands are met. |

|  |
| --- |
|  |

### The Rainflow Counting Algorithm

The Rainflow Counting algorithm is an algorithm initially used to account for stress exposure in materials, initially in full cycles, and later in partial cycles. One cycle is one instance of full stress exposure, a partial cycle is an instance of a partial stress exposure. The algorithm has since been developed to account for stress in batteries (You and Rasmussen 2011) and other appliances that go through similar wear.

The expense of battery replacement is high, importance The DST is intended for micro and small scale enterprises (MSSE). These are likely to rely on small economical margins.

#### 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

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.

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

#### Implemented Algorithm:

1. Discover discharge valleys (can currently only occur after 8 consecutive hours of discharging)
2. Count cycles to failure:
3. Accumulate cycles to failure
4. Estimate battery lifespan

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.

#### Improving the Implemented Rainflow Algorithm

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, there was therefore added a year counter to the algorithm as seen when calculating ExpL in the original algorithm.

|  |
| --- |
| Figure 2:9: 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 2:3, 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 , the algorithm triggers a maximum of 3 times per day. The decision was made to keep the initial algorithm that calculates on each occurrence of DoD minimums. The computation time gain of employing a precalculated array is too small, compared to the extensive testing and assumptions that would be required.

The 8-hour consecutiveness condition is similarly implemented in (Downing and Socie 1982), here the condition is 3 points.

### Evaluation of Finished Rewrite

In order to make a quantitative comparison of the rewrite and the original code, the bugs from the original DST was replicated in the rewritten DST. This way the changes in architecture is isolated and will produce the exact same data, if the implementation is correct. The relevant module is only Economic Analysis, the replicated module is named bugged\_economic\_analysis and prints a warning when run.

#### Correctness

Comparison of different outputs was used to prove maintained 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 should be enough data points to state that the new implementation conserve the functionality from the old, with a high degree of certainty. We assume that the probabilities are independent, because we assume that the case where the probabilities are dependent is the correct implementation. The probability of two matrices being identical coincidentally can be described as follows.

We make an assumption that the average is no more than 50%. We wish to be at least 99.999% sure that our implementation is correct. We can see that, and consequently the required number of data points are:

For one time series in the DST we have 8760 data points in hours, which alone is sufficient to ensure correctness. When we evaluate the modules with lesser complexity, the data points decrease, but the chance of a data point being coincidentally wrong is presumed lower, since they are derived from the entirety of the previous data points.

Table 2: Complexity and output source of compared simulation pair outputs

|  |  |
| --- | --- |
| Variable Names | |
| Before Rewrite | After Rewrite | Module | n Data Points |
| SoC | stateOfCharge | sapv\_plant\_simulation |  |
| LL | lossOfLoad | sapv\_plant\_simulation |  |
| NPC | netPresentCost | economic\_analysis |  |
| LCoE | levelizedCostOfEnergy | economic\_analysis |  |
| MA\_opt\_norm\_bhut\_jun15\_20\_10(:,1) | lossOfLoadProbabilities | llp\_constrained\_optimum |  |
| MA\_opt\_norm\_bhut\_jun15\_20\_10(:,3:4) | [pvKw, battKwh] | llp\_constrained\_optimum |  |

In the rewritten DST, the calculations are executed in modules that pass parameters between them. These are placed in three classes that describe the key outputs of each module, and the names have the benefit of a prefix to explain them (ex. SimOutputs.stateOfCharge). The following parameters were set in both the rewritten and original DST.

Given the validity of the coarse calculations above, it can be assumed more than 99.99% certainty for different PV and battery size combinations. In order to thoroughly amend for possible misassumptions, two simulation pairs were executed. The first simulation had PV and battery combinations, and 15 optimal solutions. The second simulation pair output PV and battery combinations.

% 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 2:10: Parameters for simulations that produce comparison outputs for simulation pair 1.

Table 3: 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 2:11: Parameters for simulations that produce comparison outputs for simulation pair 2.

Table 4: Results of comparisons in simulation pair 2

|  |  |  |  |
| --- | --- | --- | --- |
| Output A | Output 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 |

These results prove that the functionality of the DST has been preserved completely in the rewrite process.

#### Computation Speed

The computation speed is a specification in (Mandelli, et al. 2014) and should be maintained in the rewritten DST. There are however some functionality that is worth

#### Bug Fixes

The years\_to\_batt bug

The plot\_power\_balance bug

## Biomass Power Generation

# Summary and Recommendations for Further Work

## Summary and Conclusions

Through renaming every variable to optimize the readability of the DST, the design flaws and inner working of the tool became apparent. It was discovered that certain parts of the hard coded and hard-to-find parameters were responsible for….

## Discussion

## Recommendation for Further Work