ENGG 2500
Assignment II
Multi-Dimensional Criteria Analysis

University of Newcastle, Faculty of Engineering and the Built Environment

Abstract

This report presents a multi-dimensional criteria analysis (MCDA) undertaken to assess the various impacts on the Coonabarabran community for the proposed expansion of their energy infrastructure. This expansion consists of the addition of batteries to the current reserve and an installation of wind turbines.

Using HOMER (Hybrid Optimization of Multiple Energy Resources) Pro software in conjunction with gathered data, an optimal configuration and usage of renewable resources can be obtained.

The township of Coonabarabran has requested this MCDA to assess what is the optimal number of wind turbines & batteries to reduce their climate change impact. The optimal numbers found are justified by assess-able criteria ranging from categories of economic, environmental and social impacts of said venture.

Contents

1 Introduction

To assess the performance criteria, HOMER Pro was used to calculate outcomes for a range of possible combinations of infrastructure architecture over 25 years. HOMER Optimiser was used to analyse the search space of turbine & battery combinations and determine performance over a range of criteria. The exported simulation data was used to create continuous approximation functions (Equation 1.1) for important criteria with an implementation of Clough-Tocher interpolation from the SciPy library[1]. Further analysis and generation of graphs was performed using a custom Python script (Appendix 7.1).

$$f_{\text{criteria}}(t, b) = \text{interpolated value} \tag{1.1}$$

HOMERPro simulation data comes with the following caveats;

- The simulations occurred near the origin therefore observations from these approximations can only identify broad data trends.
- Matlab related plots have an artifact due to HOMERPro causing an empty regions in the top left.
- It's assumed that 6000 batteries are already included therefore the initial cost of these batteries is not included, however replacement and upkeep remain.

2 Dispatch Algorithms

2.1 MATLAB Link (ML)

The MATLAB algorithm used (Appendix 7.2) is derived from the supplied algorithm from week 7 workshop. It focuses on the usage of renewables & batteries to meet requirements, excess renewable power charges batteries and the generator is only utilised when the battery is full discharged[2].

2.2 Cycle Charging (CC)

Cycle charging utilises generators to their maximum capacity when it's required, with excess power charging the batteries. Optimal use-case is infrastructures with minimal renewables and can increase the amount of batteries charge cycles used per year[3].

2.3 Load Following (LF)

Load following utilises generators to their minimum capacity to meet their required production. Optimal in infrastructures with excess renewable power that exceeds the required production[4].

3 Performance Criteria

The following economic-based criteria were chosen as they are the most applicable & assessable criteria for the community. Their primary concerns being environmental, with emphasis on climate change impact reduction.

3.1 Economic Impact

Economical concerns were highlighted due to the budget for Warrumbungle Council, that Coonabrabran resides in, is currently in a deficit of $4.113M before Capital Grants and Contributions as of 2021[5].

3.1.1 Net Present Cost

The minimum NPC of each algorithm (Figure 3.1) is largely comparable. LF and ML are less expensive for almost any infrastructure architecture, although CC is disproportionately competitive in certain circumstances. The LF and ML plots present a mostly linear increase as distance from the origin increases, however the CC plot is more chaotic, with a vertical smear centered around 275 turbines.

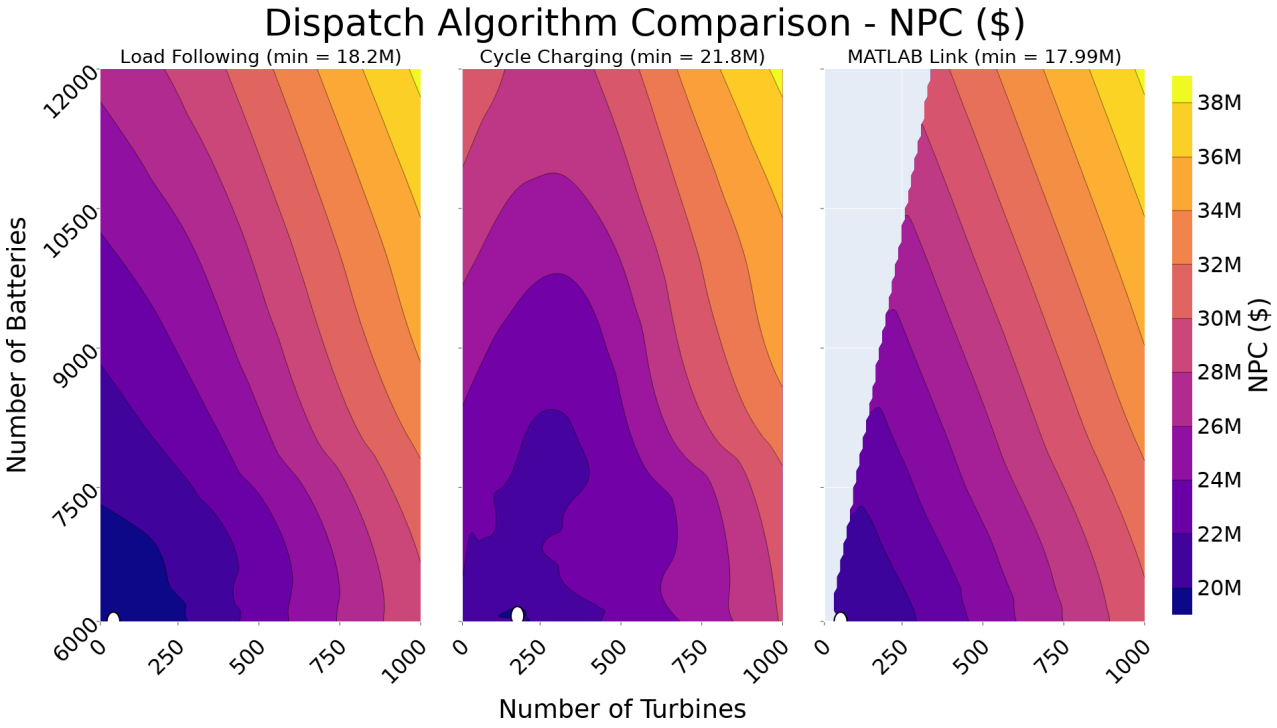

Figure 3.1: Net present cost contour plot

3.1.2 Operating Cost

Minimum operating costs differ by almost $100,000 per annum (Figure 3.2) between algorithms. Note the region in which each of the minimum values fall - LF and ML require little or no additional batteries, whilst the lowest upkeep, CC, requires investment in both batteries and turbines.

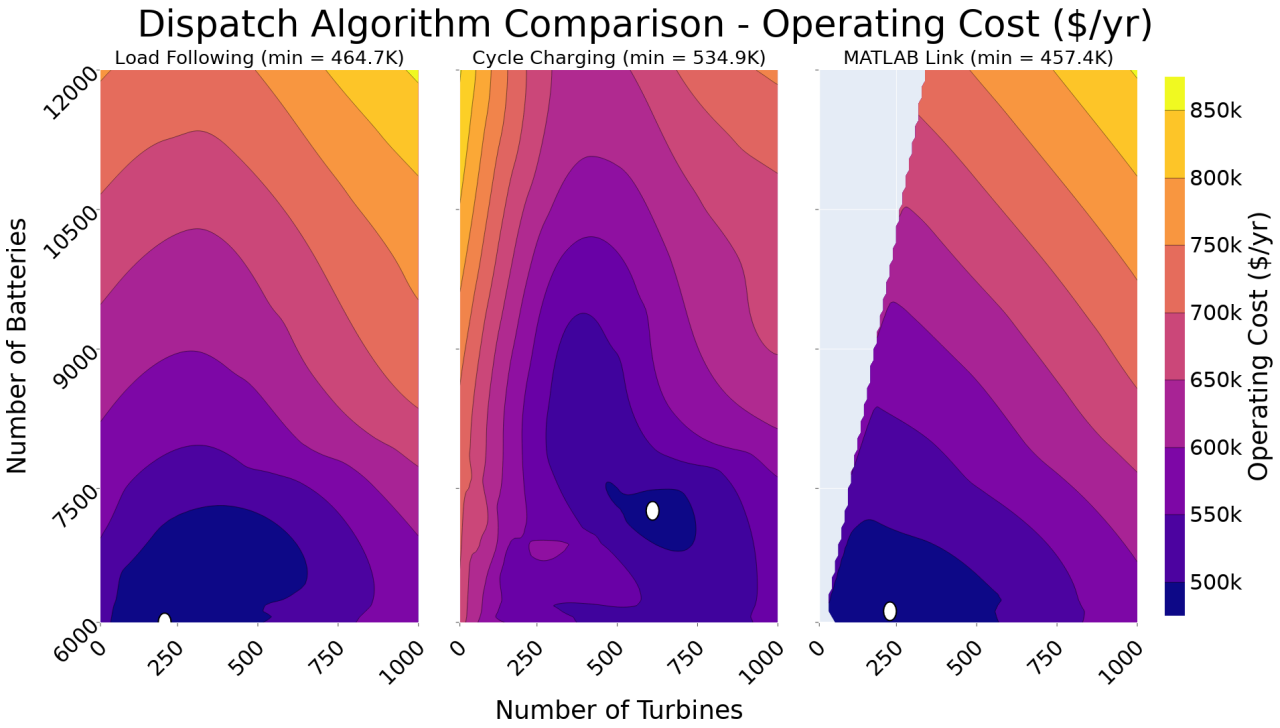

Figure 3.2: Yearly operating cost contour plot

3.1.3 Energy Sold

Energy sold (Figure 3.3) measures the total energy sold back to the grid. This is approximately a linear function of the number of turbines. LF and CC have some small and large distortions respectively. These are potentially artifacts due to the limited resolution of the simulation data. The overall trend is quite clear, increasing turbines increases energy sold.

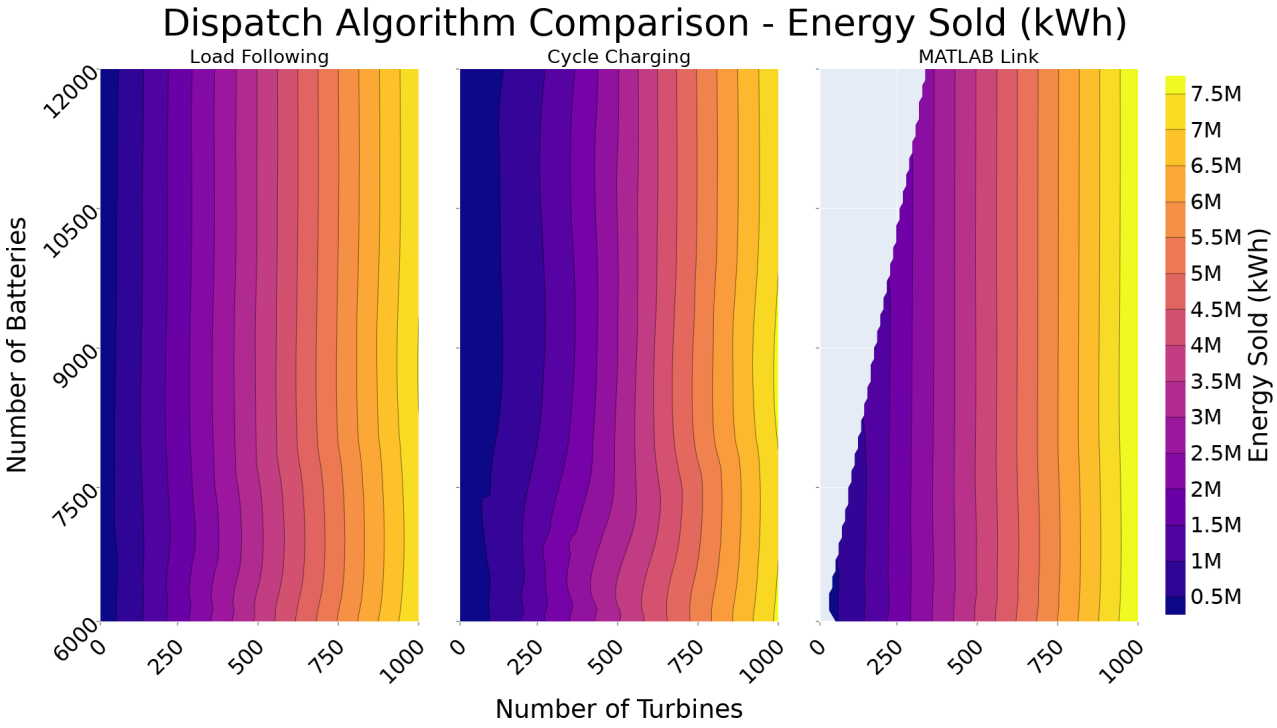

Figure 3.3: Energy sold contour plot

3.2 Environmental Impact

Reducing climate change impact is the reason for the Coonabarabran energy infrastructure expansion. Carbon emissions are a greenhouse gas that is a leading contributor to climate change. Therefore the total carbon released by meeting energy production requirements (Figure 3.4) is a highly effective indicator of an architecture's climate change impact.

3.2.1 Emission Sources

Emissions were simulated and accounted for all sources. The grid, consisting of 80% coal, 10% solar, and 10% hydro, was found to have a weighted total of 0.7275kg CO_2/kWh (Table 3.1).

Emissions	Coal(g/Kwh)	Solar(g/Kwh)	Hydro(g/Kwh)	Weighting Total
Carbon dioxide	900	50	15	727.5
Particulate matter	76g	-	-	-
Sulfur dioxide	56	-	-	-
Nitrogen oxide	23.5	-	-	-

Table 3.1: Emissions in grams per kilowatt hour

Coal

Coal produces a variety of emissions consisting of carbon & sulfur dioxide, nitrogen oxide and various particulate matter[6] (See Table 3.1).

Solar

The production and operation of a solar panel produces 50g CO_2/kW over its 25 year lifespan[7].

Hydro

Hydro power generation produces 15g CO_2/kW over its 100 year life expectancy, which encompasses what's emitted during the construction of the dam and facilities[8].

Batteries

Batteries production produces 3016kg CO_2 over their 10 year lifespan, or 301.6kg CO_2/Yr[9]. The expected lifespan of lithium batteries is between 300 & 500 complete charge/discharge cycles[10]. The can be increased by reducing the charging/discharging currents and number of cycles, LF is optimal for this.

Turbines

Wind turbine production produced 2175Kg of CO_2, or 108.75kg CO_2/Yr over a 20 year lifespan[11].

3.2.2 Emission Results

A linear ratio of turbines to batteries is optimal for LF and ML dispatches, with diminishing returns at around 500 turbines & batteries. LF achieves the lowest total carbon emissions when paired with a small battery expansion and > 500 turbines.

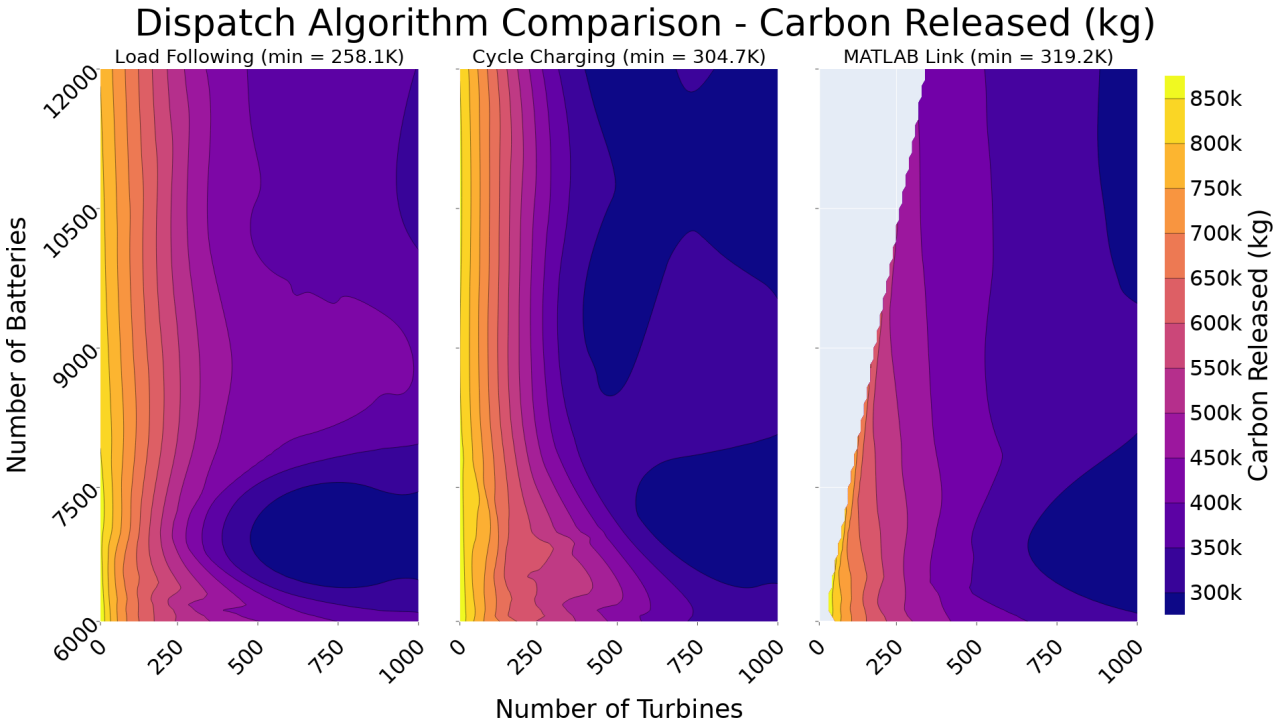

Figure 3.4: Total carbon released contour plot

3.3 Social Impact

3.3.1 Fraction Of Renewables Used

The fraction of renewables used is the fraction of renewables that are responsible for meeting required energy production. It is a community oriented statistic as it signifies the achievement of the level of their total renewable energy infrastructure. For all algorithms, the fraction of renewable energy is primarily a function of the number of turbines (Figure 3.5), with a huge benefit from a small increase in batteries.

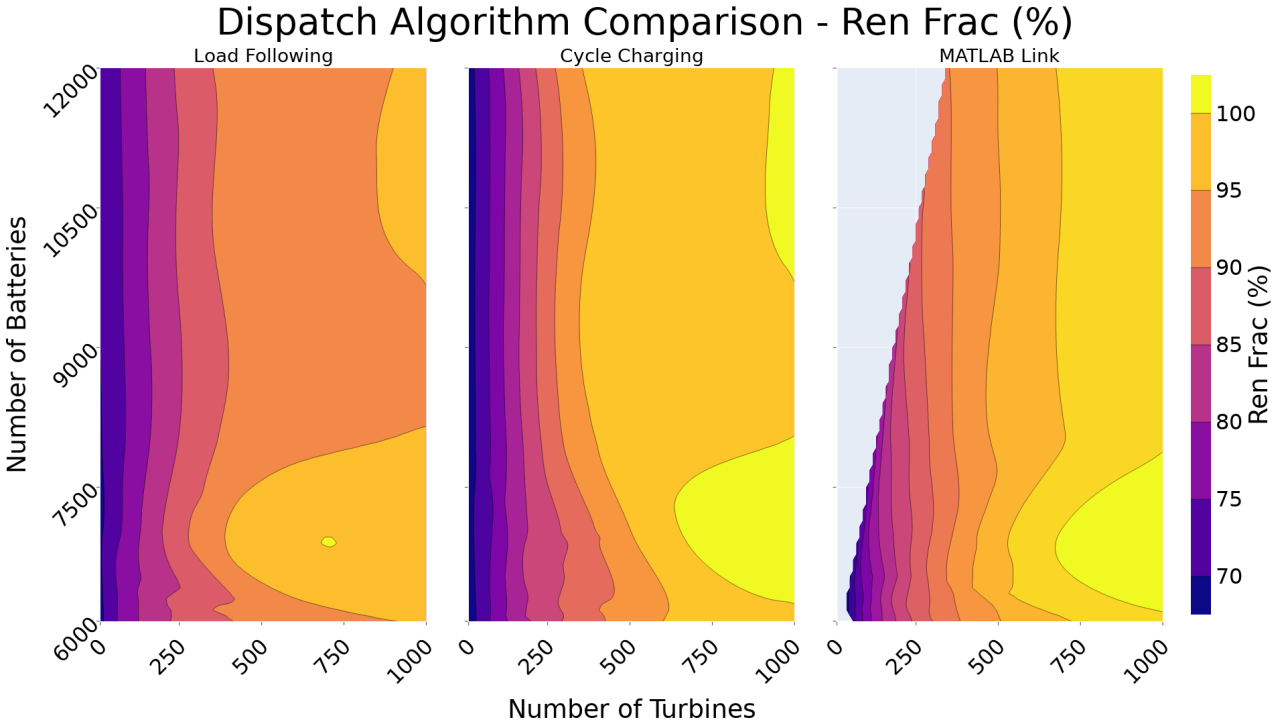

Figure 3.5: Renewable fraction contour plot

3.4 Turbine Land Usage

Directly used land and surrounding areas used for turbines can have potentially negative affects due to the following factors.

3.4.1 Biodiversity Impact

Wind turbines are a hazard for local wildlife such as birds and bats, they can potentially be kill or injured by turbine blades and this may affect the survival of endangered species in the area. But this hazard will potentially be reduced by further technological advancement[11].

3.4.2 Visual Impact

The construction of wind turbines usually erected on hills, for power generation's efficiency, affects the visual aesthetic of the natural surrounding area which is frowned upon by the general public. Another outcome of turbines is shadow flicker, which is the shadows cast by rotating turbine blades. These shadows can be cast across considerable distances when the sun is low on the horizon and can create a strobe-like effect. This is a health concern for people with epilepsy and the recommended average exposure is 30 Hours/Yr of shadow flicker[11].

3.4.3 Noise Impact

Low frequency sound produced by turbines can affect the quality of sleep which in-turn can causes psychological stress in the community[11]. To reduce noise exposure to residents in NSW, the noise limit is < 35dB & < 5dB above background noise for rural & urban residential areas, respectively. This requires a lower numbers of turbines placed 3/5km away from populated areas to ensure noise levels don't exceed these limits. Due to the widespread of the blades, a very low noise is generated in the 1-3Hz range at a volume of 100dB, although due to the low frequency it possesses no discernible health risk but is yet to be researched thoroughly.

4 MCDA

4.1 Criteria and Constraints

The community's economic and environmental concerns discussed in the performance criteria section dictate the following set of constraints (Table 4.1).

- Turbines were constrained to 500 due to a combination of emissions produced and land usage based factors.
- Batteries were constrained to 9000 due to emissions and unsustainable nature of lithium production[12].
- Economic factors of cost were constrained based on budgetary limitations of the Warrumbungle Council.
- Renewable fraction was constrained to ≥ 90 due to the importance of both the social & environmental impact of renewable resources.
- Carbon emissions were not constrained as the renewable fraction criteria was a close proxy.
- Energy sold was not constrained as it was considered to be only a secondary, indirect goal of the project.

Criteria	Constraint
Turbines	≤ 500
Batteries	≤ 6000
Operating Cost ($/Yr)	≤ 700000
Net Present Cost ($)	$\leq 25\,000\,000$
Renewable Frac (%)	≥ 90
Carbon Emissions (kg)	-
Energy Sold (kWh)	-

Table 4.1: Project operating constraints

4.2 Utility Function and Weightings

A weighted exponential utility function is used to rank infrastructure expansion options across a range of factors. Equation 4.1 describes a utility function, $F_{criteria}$, in terms of the function $f_{criteria}$ (Equation 1.1) and the level of risk aversion R.

$$F_{criteria}(t, b) = 1 - e^{\frac{-f(t,b)}{R}} \tag{4.1}$$

- All functions are first normalized using min-max scaling[13] over the criteria score.
- Lower values of R scale scores non-linearly and provide higher scores for low performing criteria that nonetheless meet constraints. Functions in which the lowest value represent the highest utility are considered in reverse.
- Each criteria is associated with a scalar weighting $w_{criteria}$.

Thus, the final weighted score for a given infrastructure architecture is (Equation 4.2):

$$S(\text{t, b}) = \sum F_{criteria}(t, b) \times w_{criteria} \tag{4.2}$$

Weightings and R values (Table 4.2) were determined through analysis of the performance criteria and consideration of the interests community. A 25 point system was developed with local environmental and economic concerns governing the bulk of the score allocation.

- The percentage of renewable energy used and carbon emissions released were each considered of upmost importance and weighted highly, with a lower risk tolerance for renewable percentage.
- NPC was weighted slightly lower and allowed slightly more risk.
- Operating cost and energy sold were considered supplementary indicators of success, and as such were given the lowest rating.

Criteria	w	R
Renewable Frac (%)	7	0.1
Carbon Emissions (kg)	7	0.5
Net Present Cost ($)	5	0.5
Operating Cost ($/Yr)	3	0.2
Energy Sold (kWh)	3	1

Table 4.2: Criteria Weighting And Risk

4.3 Results

The score of each infrastructure architecture is highly sensitive to changes in the number of turbines, as seen by the density of contour lines along the x axis, and the optimal number of turbines across all three dispatch methods lies within 450 turbines. The number of batteries contributes less to the overall score, and is notably less sensitive to change.

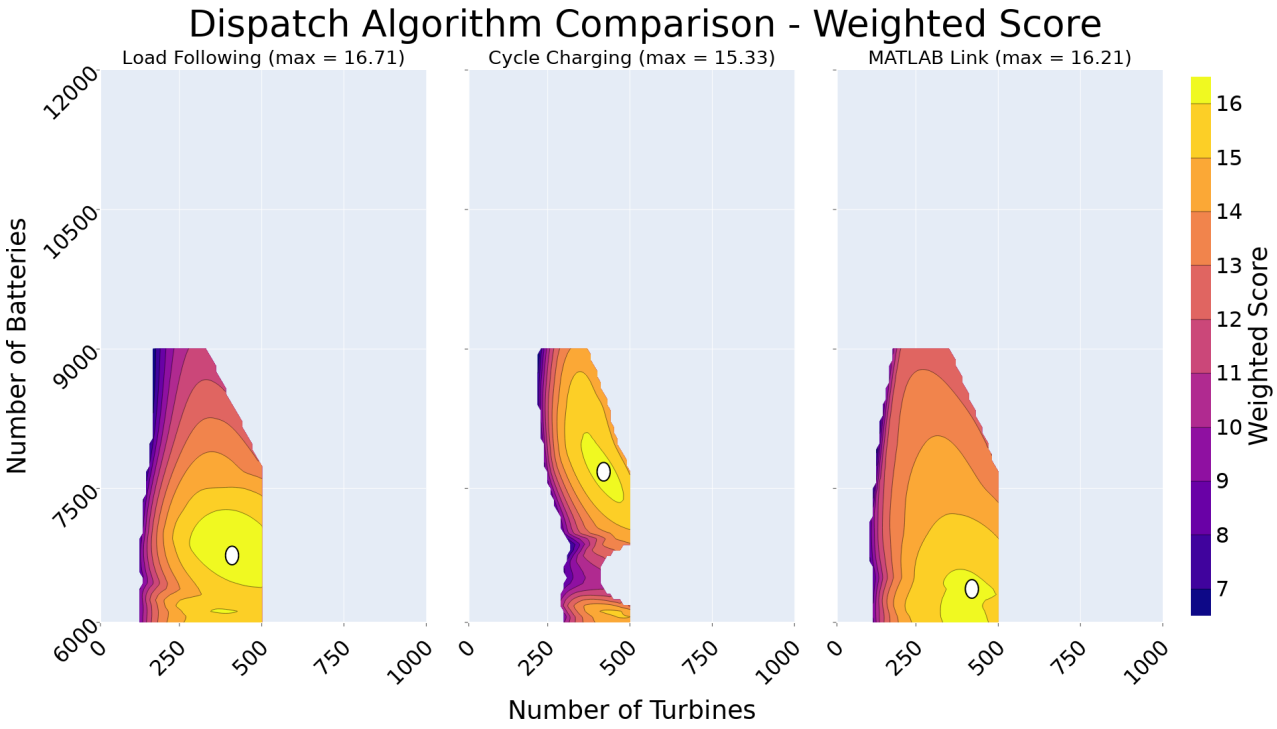

Figure 4.1: Constrained weighted score contour plot

The optimal infrastructure architecture was determined by LF and consists of 450 10kW turbines & an additional 1000 1kW batteries. With a score of 16.71/25 after scaling for risk and weighting, it's benefits include;

- NPC of \$22 000 000
- Cost of Energy at \$0.21/kWh compared to the current \$0.30/kWh
- Operating cost of \$652 755/Yr
- Generates 95% of its energy through its own sustainable sources
- Releases 400 000Kg CO_2/Yr compared to the current 997 153Kg CO_2/Yr

4.4 Sensitivity Analysis Of Energy Pricing

It is expect that the projected energy cost over the next few years will change and this will vary the NPC somewhat. The current price is $\$0.10 - 0.55/kWh$ (time dependent) and the current sellback is $\$0.05/kWh$, with an expected change to price of $\$0.30 - 0.60/kWh$ and a sellback of $\$0.05 - 0.10/kWh$[14]. Under these projected conditions, even in the most expensive scenarios the NPC for the new infrastructure is \$26 615 970 (Figure 4.2) compared to the current infrastructures NPC of \$29 608 820 (Figure 4.3).

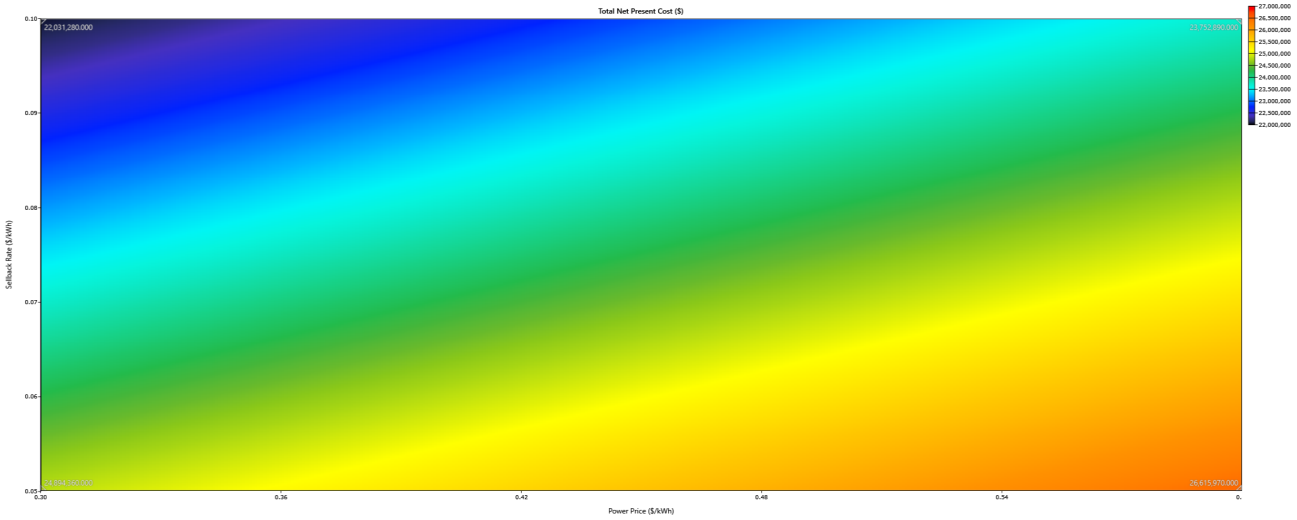

Figure 4.2: Surface Plot Of Energy Pricing New Infrastructure

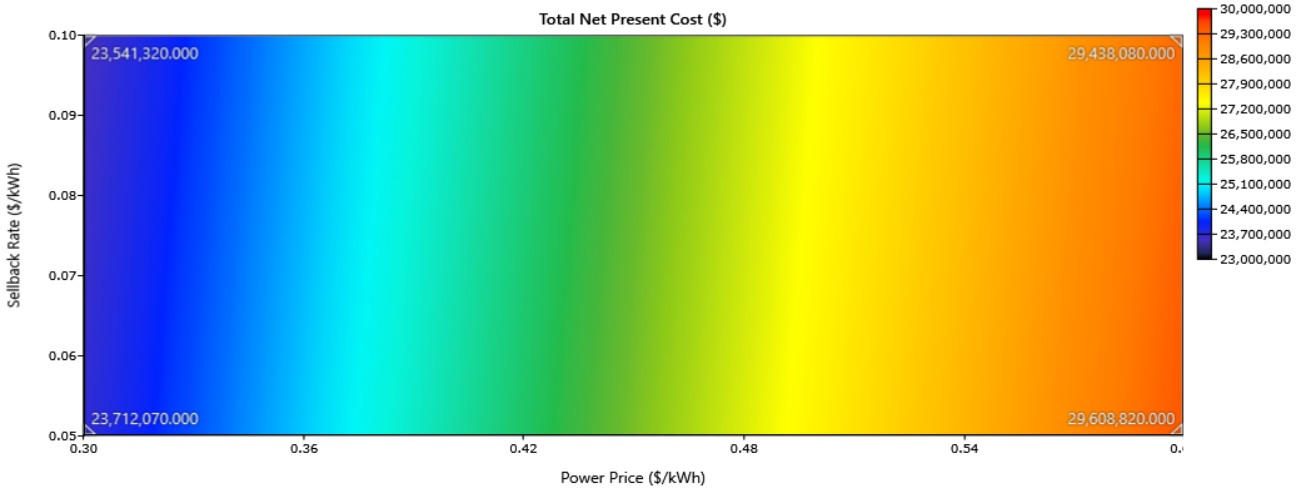

Figure 4.3: Surface Plot Of Energy Pricing Current Infrastructure

5 Conclusion

The optimal recommendation for the Coonabarabran energy infrastructure expansion is the installation of 450 10kW turbines to supplement the existing solar generation and an additional 1000 1kW batteries to expand the existing storage. From a range of dispatch options, load following was found to consistently outperform the alternatives for many individual criteria, and also the final weighted score. Further research into MATLAB Link methods of more complex design may yield further performance benefits.

6 Bibliography

[1] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[2] *Engg2500 dispatch algorithm*, Course Notes, 2021.

[3] (2021). "Cycle charging," [Online]. Available: https://www.homerenergy.com/products/pro/docs/latest/cycle_charging.html.

[4] (2021). "Load following," [Online]. Available: https://www.homerenergy.com/products/pro/docs/latest/load_following.html.

[5] K. Parker, *Council's 2021/22 draft budget and fees and charges*, 2021. [Online]. Available: https://www.warrumbungle.nsw.gov.au/news/articles/22-draft-budget-and-fees-and-charges (visited on 09/15/2021).

[6] M. Garg, J. D. Silver, R. Schofield, and R. G. Ryan. (2021). "Hourly emission inventories for air toxic emissions for eastern australian electricity generators derived from energy distribution data," [Online]. Available: https://link-springer-com.ezproxy.newcastle.edu.au/article/10.1007/s13762-021-03429-5/tables/7.

[7] K. Burkart. (2020). "How much co2 does one solar panel create," [Online]. Available: https://www.treehugger.com/how-much-co-does-one-solar-panel-create-4868753.

[8] L. Gagnon and J. F. de Vate. (2020). "Greenhouse gas emissions from hydropower: The state of research in 1996," [Online]. Available: https://www-sciencedirect-com.ezproxy.newcastle.edu.au/science/article/pii/S0301421596001255.

[9] H. Hao, Z. Mu, S. Jiang, Z. Liu, and F. Zhao. (2017). "Emissions from the production of lithium-ion batteries for electric vehicles," [Online]. Available: https://pdfs.semanticscholar.org/47cc/2bd1590bd79f316e4979a907d3063e83bd58.pdf.

[10] (2017). "What is the typical lifespan of lithium-ion battery vehicles in china," [Online]. Available: https://www.choosesolar.com.au/typical-lifespan-lithium-ion-battery/#:~:text=The%5C%20typical%5C20lifespan%5C20of%5C20lithium-ion%5C20battery%5C20is%5C20around,on%5C20the%5C202-3%5C20year%5C20lifespan%5C20of%5C20the%5C20battery.

[11] J. Moss, A. Coram, and G. Blashki. (2014). "Wind energy, climate and health evidence for the impacts of wind generated energy in australia," [Online]. Available: https://australiainstitute.org.au/wp-content/uploads/2020/12/WEB-Wind-energy-climate-and-health.pdf.

[12] I. for energy research. (2020). "The environmental impact of lithium batteries," [Online]. Available: https://www.instituteforenergyresearch.org/renewable/the-environmental-impact-of-lithium-batteries/.

[13] S. G. K. Patro and K. K. Sahu, "Normalization: A preprocessing stage," *CoRR*, vol. abs/1503.06462, 2015. arXiv: 1503.06462. [Online]. Available: http://arxiv.org/abs/1503.06462.

[14] *Engg2500 optimisation & multi-criteria decision analysis*, Course Notes, 2021.

7 Appendices

7.1 Python Code

7.1.1 Graph Generation Tools (main.py)

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import numpy as np
import pandas as pd
from scipy.interpolate import griddata
#import matplotlib.pyplot as plt

SCALE_FACTOR = 1

indicator_types = {'Cost/NPC (($)': 'Net Present Cost ($)',
                   'G10/Production (kWh/yr)': 'Turbine Generation (
                      kWh/yr)'
                   }

dispatch_types = {'CC': 'Cycle Charging',
                  'LF': 'Load Following',
                  'ML': 'MATLAB Link'}

X_AXIS = 'Architecture/G10'
Y_AXIS = 'Architecture/1kWh LI'


def load_data(file_path):
    #load data
    sim_data = pd.read_csv(file_path)

    #Create carbon released column
    sim_data['System/Carbon Released (kg)'] = (sim_data['Grid/Energy
       Purchased (kWh)'] * 0.7275) + (sim_data['Gen/Fuel (L)'] *
       2.4)

    #Subtract cost of existing 6000 batteries. Divide by two as
       batteries are assumed to be halfway through their life
    sim_data['Cost/NPC (($)'] = sim_data['Cost/NPC (($)'] - ((600 / 2)
        * 6000)

    #Ensure Ren Frac (%) maxes at 100
    sim_data[sim_data['System/Ren Frac (%)'] > 100] = 100
```

```
    #Replace nan value with zero
    g10_n = sim_data['Architecture/G10'].copy()
    g10_n[np.isnan(g10_n)] = 0
    sim_data['Architecture/G10'] = g10_n

    return sim_data


def human_format(num):
    num = float('{:.4g}'.format(num))
    magnitude = 0
    while abs(num) >= 1000:
        magnitude += 1
        num /= 1000.0
    return '{}{}'.format('{:f}'.format(num).rstrip('0').rstrip('.'),
        ['', 'K', 'M', 'B', 'T'][magnitude])


def create_interpolated_data(x, y, z, constraints=None, size=(100,
   100)):
    x = x.astype(np.float32)
    y = y.astype(np.float32)
    xi = np.linspace(1, max(x) / SCALE_FACTOR, size[0])
    yi = np.linspace(max(y) / 2 / SCALE_FACTOR, max(y) /
       SCALE_FACTOR, size[1])

    zi = griddata((x, y), z, (xi[None, :], yi[:, None]), method='
       cubic')

    zi[zi < 0] = 0

    if not isinstance(constraints, type(None)):
        for constraint in constraints:
            condition_string = constraint.format('zi')
            zi[np.invert(eval(condition_string))] = np.nan

    return zi


def create_2d_of_indicator(data, dispatch, indicator, constraints
   ={}, size=(100, 100), normalise=False):
    data = data[data['Architecture/Dispatch'] == dispatch]

    d = data[[indicator, X_AXIS, Y_AXIS]]

    turbines = d[X_AXIS]
    batteries = d[Y_AXIS]
```

```
    value = d[indicator]

    interpolated_values = create_interpolated_data(turbines,
       batteries, value, constraints=constraints, size=size)

    if normalise:
        if np.isnan(interpolated_values).all():
            return interpolated_values
        else:
            normalised_values = (interpolated_values - np.nanmin(
               interpolated_values)) / (np.nanmax(
               interpolated_values) - np.nanmin(interpolated_values)
               )

        return normalised_values

    return interpolated_values


def create_contour_plot(data, dispatch, indicator,
   extreme_value_type='max', constraints=None, data_override=False,
   data_weight=1):
    if isinstance(data_override, bool):
        zi = create_2d_of_indicator(data, dispatch, indicator,
           constraints=constraints)
    else:
        zi = data_override

    plot_object = go.Contour(z=zi, coloraxis='coloraxis1')

    return plot_object


def dispatch_comparison_plot(sim_data, indicator,
   highlight_extreme_point=False, explict_extreme_value=False,
   extreme_value_type='max', override_plot_data=False, save_file=
   False, constraints=None):
    xmin, xmax = 0, 1000
    ymin, ymax = 6000, 12000

    if isinstance(constraints, type(None)):
        constraints = {indicator : None}

    #Dispatch Options

    dispatch_options = ['LF', 'CC', 'ML']
```

```
dispatch_options_full = ['Load Following', 'Cycle Charging', '
    MATLAB Link']

#create master plot with 3 subplots
master_figure = make_subplots(1, 3, subplot_titles=
    dispatch_options_full, shared_yaxes=True, horizontal_spacing
    =0.04)

for n, dispatch_name_short in enumerate(dispatch_options):
    if override_plot_data:
        relevant_data = override_plot_data[n]
        sub_plot = create_contour_plot(sim_data, 'NA', 'NA',
            data_override=relevant_data)
        master_figure.add_trace(sub_plot, row=1, col=n + 1)
    else:
        relevant_data = create_2d_of_indicator(sim_data,
            dispatch_name_short, indicator, constraints=
            constraints[indicator])
        sub_plot = create_contour_plot(sim_data, 'NA', 'NA',
            data_override=relevant_data)
        master_figure.add_trace(sub_plot, row=1, col=n + 1)

    if np.isnan(relevant_data).all():
        extreme_value = np.nan
    elif extreme_value_type == 'min':
        extreme_value = np.nanmin(relevant_data)
    elif extreme_value_type == 'max':
        extreme_value = np.nanmax(relevant_data)

    if extreme_value is np.nan:
        extreme_point = [-100, -100]
        extreme_value = np.nan
    else:
        matching_points = [x[::-1] for x in zip(*np.where(
            relevant_data == extreme_value))]
        extreme_point = matching_points[0]

    extreme_value_string = human_format(extreme_value)

    master_figure.update_xaxes(ticks='outside', range=(0, 99),
        tickmode='array', tickvals=[int(x) for x in np.linspace
        (0, 99, 5)], ticktext=[int(x) for x in np.linspace(xmin,
        xmax / SCALE_FACTOR, 5)], tickangle=-45)
    master_figure.update_yaxes(ticks='outside', range=(0, 99),
        tickmode='array', tickvals=[int(x) for x in np.linspace
        (0, 99, 5)], ticktext=[int(x) for x in np.linspace(ymin,
        ymax / SCALE_FACTOR, 5)], tickangle=-45)
```

```
        if explict_extreme_value:
            master_figure.layout.annotations[n].update(text='{} ({}
                = {})'.format(dispatch_types[dispatch_name_short],
                extreme_value_type, extreme_value_string), font=dict(
                size=28))
        else:
            master_figure.layout.annotations[n].update(text='{}'.
                format(dispatch_types[dispatch_name_short]), font=
                dict(size=28))

        if highlight_extreme_point:
            master_figure.add_shape(type='circle', xref="x{}".format
                (n+1), yref="y{}".format(n+1),x0=extreme_point[0]-2,
                y0=extreme_point[1]-10/6, x1=extreme_point[0] + 2, y1
                =extreme_point[1] + 10/6, line_color='black',
                fillcolor='white')

    master_figure.update_layout(title_x=0.5, title_text='Dispatch
        Algorithm Comparison - {}'.format(indicator.split('/', 1)[1])
        .replace('cost', 'Cost'), font=dict(family="Gravitas One",
        size=32, color="Black"))
    master_figure.update_layout(title=dict(font=dict(size=55)))

    master_figure.layout.coloraxis.colorbar.title = indicator.split(
        '/', 1)[1].replace('cost', 'Cost')
    master_figure.layout.coloraxis.colorbar.titleside = 'right'
    master_figure.update_layout(xaxis2=dict(title='Number of
        Turbines',), yaxis=dict(title="Number of Batteries",))
    master_figure.update_layout(autosize=False, width=1920, height
        =1080,)

    if not save_file:
        master_figure.show()
    else:
        master_figure.write_image('images/' + save_file)


def create_weighted_constraint_plot(sim_data, indicators_to_consider
    , constraints, indicator_weights, size=(100, 100), save_file=
    False, limit_xy=False):
    indicators_to_reverse = []
    formatted_indicators = []
    for indicator in indicators_to_consider:
        if indicator[0] == '-':
            indicators_to_reverse.append(indicator[1:])
        formatted_indicators.append(indicator[1:])
```

```
    data_per_dispatch = [np.zeros((100, 100)) for x in range(len(
        dispatch_types.keys()))]

    for n, dispatch in enumerate(['LF', 'CC', 'ML']):
        for indicator in formatted_indicators:
            data = create_2d_of_indicator(sim_data, dispatch,
                indicator, constraints=constraints[indicator], size=
                size, normalise=True)
            if indicator in indicators_to_reverse:
                data = 1 - data

            R_values = {
                'Cost/NPC (%)': 0.5,
                'System/Ren Frac (%)': 0.2,
                'Cost/Operating Cost ($/yr)': 0.5,
                'Grid/Energy Sold (kWh)': 1,
                'System/Carbon Released': 0.5
            }

            if indicator in R_values.keys():
                R = R_values[indicator]
            else:
                R = 1

            data_per_dispatch[n] += (utility_function(data, R) *
                indicator_weights[indicator])

        if limit_xy:
            data = data_per_dispatch[n][:50, :50]
            empty = np.zeros(size)
            empty[:, :] = np.nan
            empty[:50, :50] = data

            data_per_dispatch[n] = empty

    dispatch_comparison_plot(sim_data=None, indicator='Other/
        Weighted Score', override_plot_data=data_per_dispatch,
        highlight_extreme_point=True, extreme_value_type='max',
        explict_extreme_value=True, save_file=save_file)


def utility_function(normalized_value, R):
    return 1 - np.exp(-normalized_value/R)


if __name__ == '__main__':
    sim_data = load_data('simulation_results.csv')
```

```
indicator_weights = {
                    'Cost/NPC ($)': 5,
                    'System/Ren Frac (%)': 7,
                    'System/Carbon Released (kg)': 7,
                    'Cost/Operating cost ($/yr)': 3,
                    'Grid/Energy Sold (kWh)': 3
}

global_constraints = {
                    'Cost/NPC ($)': ['{}<=25000000'],
                    'System/Ren Frac (%)' : ['{}>=80'],
                    'Cost/Operating cost ($/yr)' : ['{}<=700000'
                       ],
                    'Grid/Energy Sold (kWh)': None,
                    'System/Carbon Released (kg)' : None
                   }

dispatch_comparison_plot(sim_data, 'System/Carbon Released (kg)'
    , highlight_extreme_point=False, explict_extreme_value=True,
    extreme_value_type='min', save_file='CARBON_Comparison.png')
dispatch_comparison_plot(sim_data, 'Grid/Energy Sold (kWh)',
    highlight_extreme_point=False, explict_extreme_value=False,
    extreme_value_type='max', save_file='ENGSOLD_Comparison.png')
dispatch_comparison_plot(sim_data, 'Cost/NPC ($)',
    highlight_extreme_point=True, explict_extreme_value=True,
    extreme_value_type='min', save_file='NPC_Comparison.png')
dispatch_comparison_plot(sim_data, 'Cost/Operating cost ($/yr)',
     highlight_extreme_point=True, explict_extreme_value=True,
    extreme_value_type='min', save_file='OP_COST_Comparison.png')
dispatch_comparison_plot(sim_data, 'System/Ren Frac (%)',
    highlight_extreme_point=False, explict_extreme_value=False,
    extreme_value_type='min', save_file='REN_FRAC_Comparison.png'
    )

indicators_to_consider = ['-Cost/NPC ($)', '+System/Ren Frac (%)
    ', '-System/Carbon Released (kg)', '-Cost/Operating cost ($/
    yr)', '+Grid/Energy Sold (kWh)']
create_weighted_constraint_plot(sim_data, indicators_to_consider
    , global_constraints, indicator_weights, save_file='
    WSCORE_Comparison.png', limit_xy=True)
```

7.2 MATLAB Code

7.2.1 Start Simulation (MatlabStartSimulation.m)

```
function [myErr, matlab_simulation_variables] =
    MatlabStartSimulation(simulation_parameters)
myErr.error_description = '';
myErr.severity_code = '';

%Initialize user-defined simulation variables. We can use these
    throughout the simulation
%for dispatch decisions or to generate errors at the end of the
    simulation.
matlab_simulation_variables.total_energy_test = 0;
matlab_simulation_variables.gen1_CO = 0;
end
```

7.2.2 Dispatch (MatlabDispatch.m)

```
function [simulation_state, matlab_simulation_variables] =
    MatlabDispatch(simulation_parameters, simulation_state,
    matlab_simulation_variables)
buy_from_or_sell_to_grid_if_advantageous = 0;
    if (buy_from_or_sell_to_grid_if_advantageous == 1)
        if(simulation_parameters.has_pv && simulation_parameters.
            has_generator && simulation_parameters.has_battery &&
            simulation_parameters.has_converter)
            simulation_state.pvs(1).power_setpoint =
                simulation_state.pvs(1).power_available;
            % If there is excess renewable energy
            if (simulation_state.pvs(1).power_setpoint >
                simulation_state.ac_bus.load_requested)
                % If the batteries aren't fully charged then charge
                    them
                if (simulation_state.batteries(1).
                    state_of_charge_kwh < simulation_parameters.
                    batteries(1).battery_bank_maximum_absolute_soc)
                    simulation_state.batteries(1).power_setpoint = (
                        simulation_state.pvs(1).power_setpoint -
                        simulation_state.ac_bus.load_requested);
                % If the batteries are fully charged then return
                    power back to the grid
                else
                    simulation_state.grids(1).grid_purchases = 0;
                    simulation_state.grids(1).grid_sales = (
                        simulation_state.pvs(1).power_setpoint -
                        simulation_state.ac_bus.load_requested);
```

```
    end
% If the generated renewable power is insufficient to
    supply the load
elseif (simulation_state.pvs(1).power_setpoint <
    simulation_state.ac_bus.load_requested)
    % If the batteries aren't fully discharged then use
        them to supply the difference
    if (simulation_state.batteries(1).
        state_of_charge_kwh > 0.2*simulation_parameters.
        batteries(1).battery_bank_maximum_absolute_soc)
        simulation_state.batteries(1).power_setpoint = (
            simulation_state.pvs(1).power_setpoint -
            simulation_state.ac_bus.load_requested);
    % If batteries are fully discharged
    else
        % If the price we need to pay for grid power is
            'high' then use the diesel generator to
            supply difference
        if (simulation_state.grids(1).power_price >
            0.30)
            simulation_state.generators(1).
                power_setpoint = simulation_state.ac_bus.
                load_requested - simulation_state.pvs(1).
                power_available;
            simulation_state.grids(1).grid_purchases =
                0;
            simulation_state.grids(1).grid_sales = 0;
        % If the price we need to pay for grid power is
            'low' then use the grid to supply difference
        else
            simulation_state.grids(1).grid_purchases =
                simulation_state.ac_bus.load_requested -
                simulation_state.pvs(1).power_available;
            simulation_state.generators(1).
                power_setpoint = 0;
            simulation_state.grids(1).grid_sales = 0;
        end
    end
%  If renewable power generated and load in perfect
    balance
else
    simulation_state.generators(1).power_setpoint = 0;
    simulation_state.batteries(1).power_setpoint = 0;
    simulation_state.grids(1).grid_purchases = 0;
    simulation_state.grids(1).grid_sales = 0;
end
```

```
        % serve the load
        simulation_state.ac_bus.load_served = simulation_state.
            ac_bus.load_requested;
        simulation_state.primary_loads(1).load_served =
            simulation_state.ac_bus.load_requested;


        % serve operating capacity
        simulation_state.ac_bus.operating_capacity_served =
            simulation_state.ac_bus.load_served;
        simulation_state.dc_bus.operating_capacity_served = 0;

        % set capacity shortage
        simulation_state.ac_bus.capacity_shortage=
            simulation_state.ac_bus.operating_capacity_requested
            - simulation_state.ac_bus.operating_capacity_served;
        simulation_state.dc_bus.capacity_shortage=0;

        % set the unmet load
        simulation_state.dc_bus.unmet_load = 0;
        simulation_state.ac_bus.unmet_load = 0;

        % set excess electricity
        simulation_state.ac_bus.excess_electricity =
            simulation_state.ac_bus.load_served -
            simulation_state.ac_bus.load_requested;
        simulation_state.dc_bus.excess_electricity = 0;

        if (simulation_state.current_timestep == 1)
            save('sim_state.mat');
        end

    else
        simulation_state.ac_bus.load_served = 0;
        simulation_state.primary_loads(1).load_served = 0;
        simulation_state.ac_bus.unmet_load = simulation_state.
            ac_bus.load_requested;
        simulation_state.ac_bus.excess_electricity =
            simulation_state.ac_bus.load_served -
            simulation_state.ac_bus.load_requested;
        simulation_state.ac_bus.operating_capacity_served = 0;
        simulation_state.ac_bus.capacity_shortage =
            simulation_state.ac_bus.operating_capacity_requested
            - simulation_state.ac_bus.operating_capacity_served;

        simulation_state.dc_bus.load_served = simulation_state.
            dc_bus.load_requested;
```

```
            simulation_state.dc_bus.unmet_load = 0;
            simulation_state.dc_bus.excess_electricity = 0;
            simulation_state.dc_bus.operating_capacity_served = 0;
            simulation_state.dc_bus.capacity_shortage = 0;
        end
    else
        if(simulation_parameters.has_pv && simulation_parameters.
            has_generator && simulation_parameters.has_battery &&
            simulation_parameters.has_converter)
            simulation_state.pvs(1).power_setpoint =
                simulation_state.pvs(1).power_available;
            % If there is excess renewable energy
            if (simulation_state.pvs(1).power_setpoint >
                simulation_state.ac_bus.load_requested)
                % If the batteries aren't fully charged then charge
                    them
                if (simulation_state.batteries(1).
                    state_of_charge_kwh < simulation_parameters.
                    batteries(1).battery_bank_maximum_absolute_soc)
                    simulation_state.batteries(1).power_setpoint = (
                        simulation_state.pvs(1).power_setpoint -
                        simulation_state.ac_bus.load_requested);
                % If the batteries are fully charged then waste
                    excess energy
                else
                    simulation_state.grids(1).grid_purchases = 0;
                    simulation_state.grids(1).grid_sales = 0;
                end
            % If the generated renewable power is insufficient to
                supply the load
            elseif (simulation_state.pvs(1).power_setpoint <
                simulation_state.ac_bus.load_requested)
                % If the batteries aren't fully discharged then use
                    them to supply the difference
                if (simulation_state.batteries(1).
                    state_of_charge_kwh > 0.2*simulation_parameters.
                    batteries(1).battery_bank_maximum_absolute_soc)
                    simulation_state.batteries(1).power_setpoint = (
                        simulation_state.pvs(1).power_setpoint -
                        simulation_state.ac_bus.load_requested);
                % If batteries are fully discharged
                else
                    % Use the diesel generator to supply difference
                    simulation_state.generators(1).power_setpoint =
                        simulation_state.ac_bus.load_requested -
                        simulation_state.pvs(1).power_available;
                    simulation_state.grids(1).grid_purchases = 0;
```

```
            simulation_state.grids(1).grid_sales = 0;
        end
    %  If renewable power generated and load in perfect
       balance
    else
        simulation_state.generators(1).power_setpoint = 0;
        simulation_state.batteries(1).power_setpoint = 0;
        simulation_state.grids(1).grid_purchases = 0;
        simulation_state.grids(1).grid_sales = 0;
    end

    % serve the load
    simulation_state.ac_bus.load_served = simulation_state.
       ac_bus.load_requested;
    simulation_state.primary_loads(1).load_served =
       simulation_state.ac_bus.load_requested;


    % serve operating capacity
    simulation_state.ac_bus.operating_capacity_served =
       simulation_state.ac_bus.load_served;
    simulation_state.dc_bus.operating_capacity_served = 0;

    % set capacity shortage
    simulation_state.ac_bus.capacity_shortage=
       simulation_state.ac_bus.operating_capacity_requested
      - simulation_state.ac_bus.operating_capacity_served;
    simulation_state.dc_bus.capacity_shortage=0;

    % set the unmet load
    simulation_state.dc_bus.unmet_load = 0;
    simulation_state.ac_bus.unmet_load = 0;

    % set excess electricity
    simulation_state.ac_bus.excess_electricity =
       simulation_state.ac_bus.load_served -
       simulation_state.ac_bus.load_requested;
    simulation_state.dc_bus.excess_electricity = 0;

    if (simulation_state.current_timestep == 1)
        save('sim_state.mat');
    end

else
    simulation_state.ac_bus.load_served = 0;
    simulation_state.primary_loads(1).load_served = 0;
    simulation_state.ac_bus.unmet_load = simulation_state.
```

```
            ac_bus.load_requested;
        simulation_state.ac_bus.excess_electricity =
            simulation_state.ac_bus.load_served -
            simulation_state.ac_bus.load_requested;
        simulation_state.ac_bus.operating_capacity_served = 0;
        simulation_state.ac_bus.capacity_shortage =
            simulation_state.ac_bus.operating_capacity_requested
            - simulation_state.ac_bus.operating_capacity_served;

        simulation_state.dc_bus.load_served = simulation_state.
            dc_bus.load_requested;
        simulation_state.dc_bus.unmet_load = 0;
        simulation_state.dc_bus.excess_electricity = 0;
        simulation_state.dc_bus.operating_capacity_served = 0;
        simulation_state.dc_bus.capacity_shortage = 0;
    end
  end
end
```

7.2.3 End Simulation (MatlabEndSimulation.m)

```
function myErrs = MatlabEndSimulation(simulation_parameters,
    matlab_simulation_variables)
myErrs.simulation_errors = {};
myErrs.simulation_warnings = {'This is a MATLAB test warning.'};

% if matlab_simulation_variables.total_energy_test < 1e4
%     myErrs.simulation_warnings = [myErrs.simulation_warnings {'Not
    very much energy.'}];
% end

% if simulation_parameters.emissions.use_max_emissions_CO &&
    matlab_simulation_variables.gen1_CO > simulation_parameters.
    emissions.max_emissions_CO
%     myErrs.simulation_errors = [myErrs.simulation_errors {'Too
    much carbon monoxide from generator 1.'}];
% end
end
```