NTNU
Kunnskap for en bedre verden

Department of Economics and Management

MET430 - Applied Mathematics

A Mathematical Approach to Financially Evaluate a Underwater Data Warehouse

Authors:
Candidate: 10027
Candidate: 10011

Executive summary

This project is an exam deliverable in the NTNU course Applied Mathematics for Business and Economics, MET430. The topic of this project was underwater data warehouses, and its corresponding problem was to perform a financial analysis of this possible investment.

Underwater data warehouses is a relatively new concept. Storing data underwater enables several possibilities to counter many of the problems that existing data warehouses faces. There might be substantial benefits using this concept seen from a technical perspective. However, if such a investment is not financial viable, there are few incentives to push it forward. This project has focus on the financial side of the investment derived from mathematical calculations.

To financially analyse this investment one needs both cost and revenues. Three cost drivers have been defined and estimated, these are production, deployment and operational costs. The costs are based on research on data found online. The revenues have also been derived in approximately the same way as the costs. Lastly the revenues and costs takes part in a discounted cash flow model where the analysis will generate financial key insights of the investment.

This project will heavily emphasise mathematics, and this has been used to estimating costs and generate financial insights. The mathematics used for the production costs have been relying on economies of scale, polynomial equations and differentiation. The deployment costs have been derived using several aspects of optimisation techniques. The operational costs is based on trigonometry functions. The discounted cash flow model have mainly been calculated using techniques from matrix algebra.

Based on the results of this project one can conclude that investment is financial viable due to a positive net present value and other metrics which gives positive indications. The results is of value for a potential shareholder, because it gives important financial insights about the investment. However, there are several uncertainties and risks that are impacting the usability of this project. These uncertainties and risks are so grand, that it is not recommended to use the result for real-life applications.

Table of Contents

List of Figures

List of Tables

1 Introduction

The intention of this project is to financially evaluate an investment in a underwater data warehouse located in the southern part of Vestland county.

1.1 Topic and problem

Underwater data warehouses are a relatively new concept, only having a few test projects performed by Microsoft, and some planned projects from a few other companies(Mary Zhang 2022). The Natick project by Microsoft is aimed at investigating the performance, costs and usability of underwater data centres, hoping to get data confirming the assumed benefits these warehouses have. Placing data centres underwater can be beneficial in several ways, most notably is cooling. Data storage facility's produce large amounts of heat from it's many internal components. This heat is counterproductive, as the internal servers work optimally in moderate temperatures (Wylie Wong 2023). With large amounts of unwanted heat, the data centres need to have complex cooling systems. However running these are costly, it is estimated that 40% of energy consumption in a data centre is spent on cooling (Zhang et al. 2017). However, by placing the warehouse underwater, you can leverage the cooler water temperatures as a built in cooling system. Thus, relying less on electrical power and by extension saving money and lowering power consumption. In essence this seems like a beneficial idea both for companies as they could save costs in addition to the climate, as these centres would use far less power than their land based counterparts.

Additionally the concept of underwater data warehouses satisfies or at least is in line width four of the UN's sustainable development goals (United Nations 2015). Those are goal 8, decent work and economic growth. Goal 9, industry, innovation and infrastructure. Goal 11, sustainable cities and communities, and goal 13, climate action. This served as additional motivation as to why this was a topic worth researching and exploring.

Underwater data warehouses might be a good technical solution, that counters many of the current problems related to traditional land based data warehouses. However, if the concept is not economically viable, it will not receive attention and would likely end as just a concept. The financial viability plays a large role in surfacing such concepts and therefore we have chosen to approach this topic from a business angle. This project is aimed at creating a discounted cash flow model (DFC) for the underwater data warehouse to see whether such a project is financially viable. The DCF will consists of revenues and costs. The three main cost drivers are related to production-, deployment- and operational costs of such a facility. The three main revenues for the project is an assumed grant from Innovasjon Norge, annual revenues and residual value of the warehouse. The financial analysis will produce an output advising whether the investment is financially viable or not.

1.2 Background for choice of topic

The inspiration behind the topic choice arose from a lecture in a different course, Applied Data Science TDT4259. Where Richard Hall, a guest lecturer and Equinor employee, presented the topic as a possible angle to the examination project in said course. Mr. Hall's angle was somewhat different phrasing the problem thesis as: creating a good definition of where to place underwater data warehouses or finding a specific location to place them. Reframing this topic we wanted to approach it from a business perspective in order to more adequately conform to this course' objectives and themes.

1.3 Roles and work distribution

This project is a collaboration between two students. Both have participated in all of the phases of the project. Everything from brainstorming, to coding and writing have been performed at least

with a collective understanding of what the goal is. The table below is a rough estimate of the main contributor in the given section. Person X is referring to candidate 10027 and person Y to candidate 10011.

Task	Code	Writing
Introduction		Y
Methods and data	Y	Y
Production costs	X&Y	Y
Optimisation	X&Y	X
Deployment costs	X	X
Operational costs	X	X
Discounted cash flow analysis	Y	Y
Conclusion		X&Y

Table 1: Roles and work distributions

1.4 Exam delivery note

As previously mentioned this topic was borrowed from the course TDT4259[1], which we both have participated in this semester. In this course we have written a similar exam project, but with another approach and intention. Using the same topic but the angels suggested by mr. Hall in his guest lecture. Our candidate numbers on the other exam are 10011 and 10451.

In addition, our deliverable is longer than the suggested length. The deliverable is approximately 10000 words, excluding code, which is longer than the suggested length. This is due wanting this to be a complete project, containing necessary explanation of this complex topic. Both to meet the course requirements in addition to our personal desire of creating a exhaustive analysis.

[1] https://www.ntnu.edu/studies/courses/TDT4259/#tab=omEmnet

2 Methods and data

The input used in the DCF comes from different mathematical methods used on data. All of these calculations are derived using computational methods in the programming language Python as well as examples using numerical approaches.

2.1 Choice of methods

Many of the mathematical methods used in this projects have been learned in the NTNU course MET430[2]. In our calculations we apply many of these methods to varying degrees compared to what have been taught in the course. Our project consists of five phases: production, optimisation, deployment, operation and financial analysis.

The first phase concerns the production costs for the warehouse. In this phase we will apply mathematics and methods such as polynomial functions and differentiation, where we find costs- and marginal costs function.

The second phase concerns deployment optimisation of a underwater data warehouse. For this process we have extracted a map section over the southern part of Vestland county, along with various other important geographical factors. We have duplicated and simplified this map such that it is easier to perform mathematical optimisation procedures. This section utilises several variables with corresponding weights to define the optimal site location for the warehouse in regard to costs. This phase is based on linear optimisation, although the functions will be quite different from the standard linear programming used in the NTNU course. We picked this method based on the fact that they could provide interesting insights to the project topic.

Based on the optimisation one can define cost function for both deployment- and operational costs. The deployment costs are based on the optimal position for the warehouse. This phase is mathematically grounded in the topic of arithmetic operations.

Over to the fourth phase we shall define the operational costs for the warehouse. We assume that the operational cost of the warehouse only consists of energy consumption and maintenance costs. Furthermore we assume that the energy consumption is entirely based on ocean temperature. We have created a simulated function that represent the ocean temperature. This phase touches the mathematical conceptions of recursive equations and integration.

Lastly the results from the four previous phases shall be gathered and used in the financial analysis phase. The first step in this phase is simulating revenues and then adding the costs. The financial analysis method used is a discounted cash flow model. This model will be calculated using matrix algebra and various mathematical financial concepts.

In total, the project will use the following mathematical methods, concepts and topics: differentiation, integration, polynomial equations, trigonometry, linear programming, recursive equations, matrix algebra and finance.

2.2 Choice of data

This projects analyses are comprised of both actual and simulated data. Most of the project is based of the simulated data, based on reasonable estimates. For example the production costs of our model are based on estimates found from various sources, combining these with some randomness and logical assumptions results in the costs used in the analysis. The only section that uses actual data is while plotting the map showcasing the research area. This map serves as a base layer for the optimisation / deployment part. This data is geographical data retrieved from various websites, mostly governmental sites dedicated to keeping such data.

[2] https://jmaurit.github.io/math2/

Feature	Enginereed	Source
Municipalities	True	(GeoNorge 2023b)
Harbours	False	(Kystinfo 2023)
Powerline land	False	(Norges vassdrags- og energidirektorat 2023)
Powerline sea	False	(Norges vassdrags- og energidirektorat 2023)
Fiber cable	True	(GeoNorge 2023a)
Hydro powerplant	False	(Norges vassdrags- og energidirektorat 2023)
Wind powerplant	False	(Norges vassdrags- og energidirektorat 2023)

Table 2: The geographical data and their sources

2.3 Imports and downloads

This subsection displays the libraries used in the programming, and the importing of the data files used for plotting the actual map. The libraries and data are crucial for enabling effective programming, generating valuable visualisations and adding a factor of reality to the analysis.

```
# Importing nessecary packages
import numpy as np
from math import exp
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.pyplot import subplot2grid
from matplotlib.patches import Rectangle, Circle
import matplotlib.patches as mpatches
import scipy
import numpy_financial as npf
import geopandas as gpd
```

```
# Importing datasets with geographical data to plot map
vestland = gpd.read_file('__vestlandet.geojson')

pow_line_land = gpd.read_file('__kraftlinje.geojson')
pow_line_sea = gpd.read_file('__sjokraftlinje.geojson')

windpower = gpd.read_file('__vindkraft.geojson')
hydropower = gpd.read_file('__vannkraft.geojson')

harbor = gpd.read_file('__havner.geojson').set_crs(epsg=25833,
                                                   allow_override=True)
municipalities = gpd.read_file('tryit_32.geojson')
fiber_actual = [[287000,290000],[6604000,6616000]]
```

3 Production costs

The first phase of the DCF model is estimating production costs. Production of the warehouse has been divided into two separate components. The first component is the warehouse itself, the external part. The second component is the content of the warehouse, the internal part. The numbers used in the calculations are based on a mixture of research and arbitrary values from various websites. The cost function itself is based on a conceptual estimation of the costs nature.

3.1 Production costs

Starting with the internal part, which mainly consists of data servers. It is assumed that the cost of adding servers is linear and that there are no fixed costs related to this part. However, that as the warehouse size increases the cost increases linearly. The cost function for the internal part is:

$$IC(x) = internals * cubic \tag{1}$$

Over to the external part. When producing the external casing there is a fixed height and width meaning it can only increase length wise. Therefore the endpoints of the warehouse can be considered a fixed cost. As no matter the length their cost remains unchanged. We estimate that as the length increases the unit costs of length decreases. We assume that this kind of cost has many similarities with the sigmoid function, therefore we chose to use such a function to generate the cost of the external part (GeoNorge 2021).

$$sigmoid = \frac{1}{1 + e^{-}x} \tag{2}$$

The upside of using a sigmoid function is that we are able to capture a lot of the natural movements of the costs. However, the sigmoid function ends up stabilising at the numerator value, which is not inline with reality. This downside has to be taken into consideration when evaluating the cost function. In addition, a d term have been included to elongate the range of warehouse volumes. The cost function for the external part is:

$$EC(x) = \frac{externals}{1 + e^{d*-cubic}} \tag{3}$$

When adding the cost function for the internal- and external part, one ends up with this equation:

$$PC(x) = internals * cubic + \frac{externals}{(1 + e^{d*-cubic})} \tag{4}$$

Below we implement the cost function in Python.

This next section is somewhat technical as we aim to calculate the internal and external costs related to the warehouse production. First we need to calculate the number of servers we can fit inside the warehouse in order to get both cost- and income data. We used the price of a Dell PowerEdge R630 which retails for around 2000$ and has a storage capacity of 300GB (Bryn Burns 2021). The dimensions of this server is height(4,3cm), Width(48,2cm) and depth(75,5cm) (DellEMC 2016). These servers are stackable and when rounding of the dimensions one get H(5cm), W(50cm) and D(100cm). However, this spacing is to be considered unlikely.

Using numbers from Mircosoft's Natick 2[3] project where they had 864 servers on $340m^3$ (Mary Zhang 2022), this is far less than the projected 40 servers per m^3 in this investment. Based of this increasing the space needed per server seems reasonable, this would allow more airflow around the servers and accessibility for maintenance crews. The new dimensions are therefore H(10cm), W(100cm) and D(100cm), meaning it allows for 10 servers per cubic meter. This allows for

[3]The second iteration of Microsoft's prototype warehouse (John Roach 2020)

easier calculations as well as some wiggle-room for a rack and cable connections, for simplification purposes both cables and the racks are complimentary to purchasing the appropriate amount of servers. As each server is about 2 000$ it costs 20 000$ to fill each cubic meter and it yields 3TB of data in return.

It is assumed that when the business increase it's investment of servers, it will experienced economies of scale[4] in form of discounts. The discounts will occur at every 26.6 cubic meter, or with the purchase of 266 additional servers. The discount comes in the form of a percentage discount on the cumulative amount of purchased servers, it has a cap of 15% discount. Meaning the largest possible discount achievable is 15% off.

Finding the external costs are also quite a intricate problem. As this course does not entail, nor require, the understanding of how to create a data warehouse with waterproof seals and a complex cooling system on par with the Natick 2. We decided simple relationships and assumptions was the best course of action. Therefore an estimation is made, most likely a quite poor one at that, but we digress. A ton of steel costs around 1 000$ (Andreas Janisch 2023). However, the cost of labour needed to shape this to the needed specificity is likely high, so we estimate that a cubic block of steel would cost around 4 000$. If we use this as a benchmark for the outer casing and add additional costs for a outer support structure, complexity and technological structures like cooling etc. We can add an additional 10 000$ per cubic meters based of those factors. This puts the total external costs estimation up to 14 000$. There is is no specific reason for the large increase in costs other than our preference of overestimating our costs, rather than underestimate them. In addition to our assumptions that we likely overlooked something.

Due to production constraints the warehouse needs to have a height and width of 4 meters, while only the length can be adjusted. Increases in the warehouse length can only occur one meter at a times, because any smaller increase will not allow enough space for more servers and the increase, from a financial viewpoint, becomes redundant. The warehouse can have a maximum length of 16 meters, which allows for a total maximum volume of 320.

$$4 * 4 * 16 = 320m^3 \tag{5}$$

.

Implementing the cost function with the economies of scale in python code it looks like this.

```
    def production_cost_function(cubic):
        discount_idx = np.where(cubic >= steps_discount)[0]
        rel_discount = discount_number[len(discount_idx)]
        C = internal * rel_discount * cubic + external / (1 + np.exp(steep *
        ↪  cubic))
        # Returns the production costs based of the warehouse size in cubic
        ↪  meters
        return C
```

```
    # Defining the cost function
    internal = 20000  # Fixed costs for the internal servers per cubic meter
    external = 14000  # Fixed costs for the outer casing per cubic meter
    steep = 0.05  # A variable determining the steepness of the logistic term.

    n_discount_steps = 12 # Number of discounts
    max_discount_percent = 15 # Max discount %

    steps_discount = np.linspace(16, 321, n_discount_steps)
    discount_number = 1 -((max_discount_percent/n_discount_steps)*
                          np.linspace(1, n_discount_steps, n_discount_steps))/100
```

[4]Economies of scale refers to large scale benefits.

```
12
13      # Creating a list of possible volumes
14      volumes_x = np.arange(16, 321, 16)
15
16      # List of costs by volumes
17      costs = []
18      for cubic in volumes_x:
19          cost = production_cost_function(cubic)
20          costs.append(cost)
21
22      # Plotting costs and volumes
23      fig, ax = plt.subplots()
24      plt.plot(volumes_x, costs, c="indianred")
25      ax.set_xlabel("Wearhouse Volume")
26      ax.set_ylabel("Production costs")
27      ax.set_title("Production costs v Volume")
28      plt.show()
```

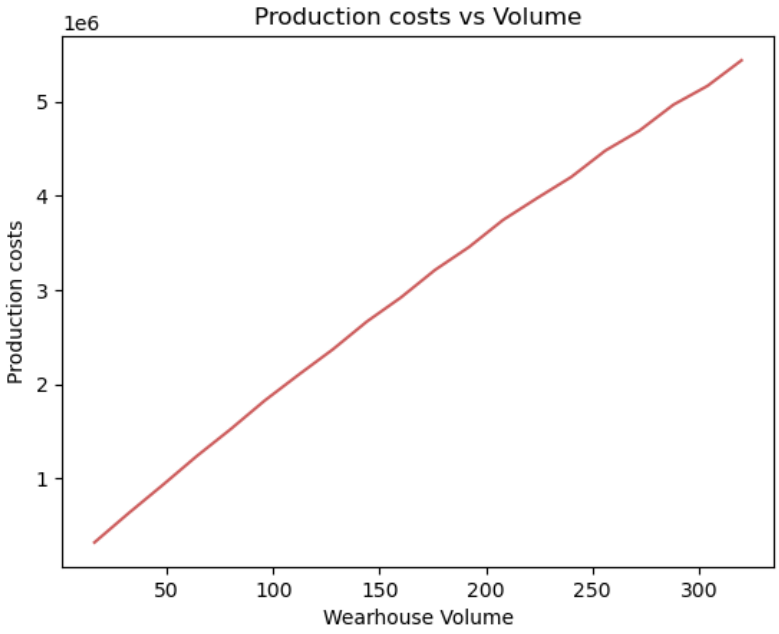

Figure 1: The line represents the development of production costs (y) as the warehouse volume (x) increases.

Based of the plot one can observe that the function has a polynomial curve. At a glance the function almost looks linear, however it has a slight logarithmic curve and it's slope is trailing of slightly as the x values increase. Additionally looking closely the discount steps are visible as waves on the tail-end of the function.

3.2 Marginal production costs

Differentiating the cost function is quite difficult due to the large scale benefits. For simplicity the function is differentiated without the economies of scale. We start with the cost function

$$PC(x) = internals * cubic + \frac{externals}{(1 + e^{d*-cubic})} \tag{6}$$

Then we split the term in two separate equations:

$$\frac{dcubic}{dy} = (f + g)' = f' + g' \tag{7}$$

The f term is easily solved with basic derivation.

$$f = internals * cubic f' = internals \tag{8}$$

Moving over to the more complex g term.

$$g = \frac{externals}{(1 + e^{d*-cubic})} \tag{9}$$

Using the formula for differentiating a division part:

$$g' = \frac{(u' * v) - (u * v'}{v^2} \tag{10}$$

Separately solving the u and v term.

$$u = externals u' = 0 v = 1 + e^{d*-cubic} v' = e^{d*-cubic} * -d \tag{11}$$

Furthermore the u and v terms are inserted back into the fractions formula.

$$g' = \frac{(0 * e^{d*-cubic} - externals * e^{d*-cubic} * -d)}{(e^{d*-cubic})^2} \tag{12}$$

Lastly the derived terms for f and g are placed back into the new derived cost function

$$TC' = internals + \frac{(0 * e^{d*-cubic} - externals * e^{d*-cubic} * -d)}{(e^{d*-cubic})^2} \tag{13}$$

$$TC' = internals + \frac{externals * e^{d*-cubic} * d}{(1 + e^{d*-cubic})^2} \tag{14}$$

```
    # Visualising the marginal costs, with the standard cost function
    # Writing the derivated cost function
    def d_cost(cubic):
        d_C = internal + (external*steep*np.exp(steep*-cubic))/
                (1 + np.exp(steep*-cubic))**2
        # Returns the derviative costs
        # based of the given warehouse size in cubic meters
        return d_C

    # Plotting the derived costs, without any discounts
    fig, ax = plt.subplots()
    ax.plot(volumes_x, d_cost(volumes_x), c="indianred")
    ax.set_xlabel("Wearhouse Volume")
    ax.set_ylabel("Marginal costs")
    ax.set_title("Derivate: Costs vs Volume (without large-scale benefits)")
    plt.show()
```

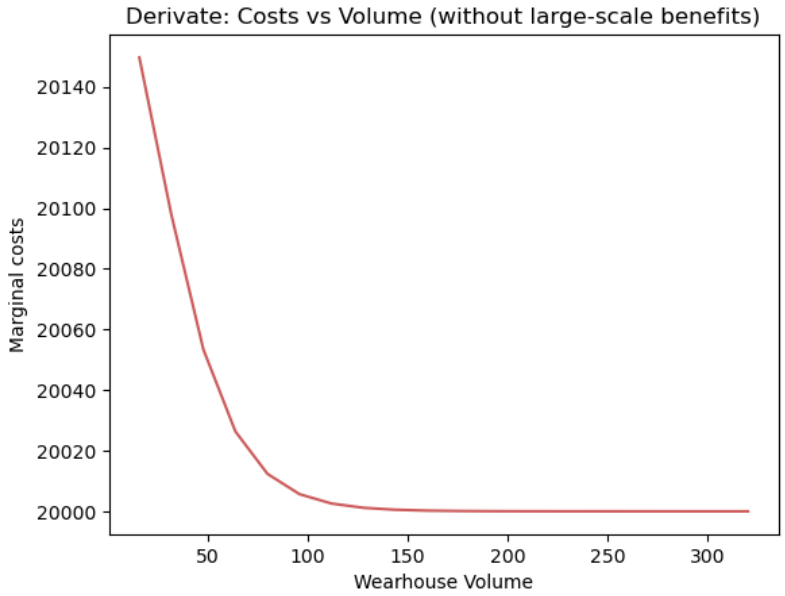

Figure 2: The marginal production costs (y) for different volumes (x), using the marginal production cost function without large scale benefits.

The marginal costs for the warehouse without large scale benefits looks like this. From the plot one can observe that the curve flattens when the volume exceeds 150 cubic. It is credible to think that the flattening out is a consequence of the sigmoid function. However, from the business perspective we are interested in the cost with economies of scale, or rather the actual differentiated curve.

```
    # Finding the cost function's actual derivative
    derivate_costs = np.diff(costs)
    derivate_volumes = volumes_x[:-1]

    # Plotting the derivatives against volume
    fig, ax = plt.subplots()
    ax.plot(derivate_volumes, derivate_costs, c="indianred")
    ax.set_xlabel("Wearhouse Volume")
    ax.set_ylabel("Marginal costs")
    ax.set_title("Derivate: Costs vs Volume")
    plt.show()
```

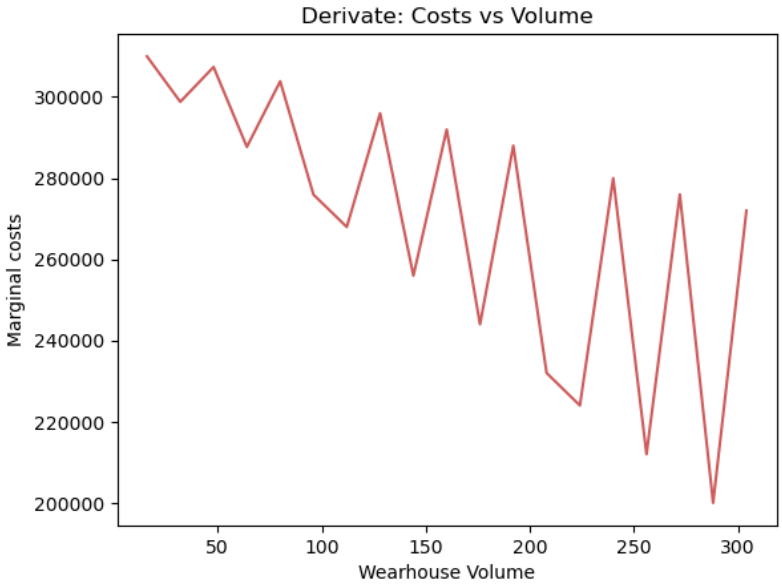

Figure 3: The actual marginal production costs (y) for different volumes (x), using the marginal production cost function with large scale benefits.

Based of this plot we can observe the marginal costs across different volume sizes. The slight curves we saw in plot 1 are now enhanced and displayed as sharp changes in the costs given changes in volume. Interestingly in one occurrence the marginal costs decline twice. Likely this is due to the large scale benefits having an impact that reduces

From this plot one can observe different marginal cost of new investments. An interesting observation is that once the volumes have a sinking marginal cost twice in a row. This means that the discount effect from the economies of scale sometimes have an impact for two consecutive volume increases. The important business insight from this plot is to only invest in valleys on this graph, not the peaks. This will be elaborated in section 7.1.3.

3.3 Production cost conclusion

The production costs varies with amount of cubic meters. The costs varies in the range of 0 to approximately 5 million USD. There are certain investment point in regard to cubic meters that are economically beneficial. Precisely which will be defined later when the total cost is established and we can compare the costs relative to the revenue.

4 Optimisation

4.1 Map optimisation

This nonlinear optimisation problem is aimed at finding the optimal position for the underwater data warehouse. With a specified area of interest, by minimising the costs the output should be the x, y coordinates with the least costly position. The resulting map will consist of simple shapes rather than a complex coastlines as this allows for approximately accurate results using relatively simple mathematics rather than an extremely complex model.

The map is based of an actual cutout of the Norwegian coastline, more specifically an area of approximately 110 square kilometres north of Haugesund. Starting close to the Vestlandet county border with Langevåg being the biggest town encompassed within the artificial borders. The cutout is scaled to 110×100 in x and y coordinates making the interpretation of actual distances simpler. These numbers are however, only approximations and the map is an extreme simplification of reality. With this in mind the results generated are a lot more theoretical than practical. However, as long as we are aware of this the results can be interpreted accordingly.

We have decided to focus on a simple collection of variables, well aware that this compromises the practical use and realism of our model. However, in order to keep the project from becoming too large and the complexity within the scope of this course. The specific area of interest was chosen randomly with the ulterior motive of minimising the geographical changes we would have to make. Variables such as water depth, seabed topography and conflict zones such as fishing activity, wreaks, bridges and other marine infrastructure where all disregarded in the analysis, despite their relevance to the topic. Prior to the optimisation analysis we included a plot of the actual area of interest as to give a reference to what is being discussed and analysed.

```
    land =  mpatches.Patch(color='yellowgreen', alpha=1, label='Land')
    pow_line =  mpatches.Patch(color='maroon', alpha=.6, label='Powerlines')
    pow_pla =  mpatches.Patch(color='black', alpha=1,
                              label='Power plants (wind and hydro)')
    har_bours =  mpatches.Patch(color='peru', label='Harbors')
    fib_er =  mpatches.Patch(color='darkslategrey', label='Fiber cable',
    ↪  alpha=1)
    se_a = mpatches.Patch(color='paleturquoise', label='Sea', alpha=0.6)

    handles = [land, pow_line, pow_pla, har_bours, fib_er]
```

```
    def plot_actual_map(ax):
        municipalities.plot(ax=ax, color='yellowgreen', alpha=1)

        pow_line_land.plot(ax=ax, color='maroon', linestyle='--',alpha=.6)
        pow_line_sea.plot(ax=ax, color='maroon', linestyle='--', alpha=.6)

        windpower.plot(ax=ax, color='black', markersize=100, marker='*',
        ↪  alpha=1)
        hydropower.plot(ax=ax, color='black', markersize=100, alpha=1)

        harbor.plot(ax=ax, marker='s', markersize=100,color='peru')
        plt.plot(fiber_actual[0], fiber_actual[1], color='darkslategrey')

        ax.set_xlim(277500,292500)
        ax.set_ylim(6604500,6616000)
        ax.set_xticks([],[])
        ax.set_yticks([],[])
```

```
    ax = vestland['geometry'].plot(color='paleturquoise', alpha=0.6,
                                   figsize=(10,10), aspect=1)
    plot_actual_map(ax)
    plt.title('Vestland county, geospaital plot')
    plt.legend(handles=handles)
    plt.xlabel('Longitude, UTM zone 32')
    plt.ylabel('Latitude, UTM zone 32')
    plt.show()
```

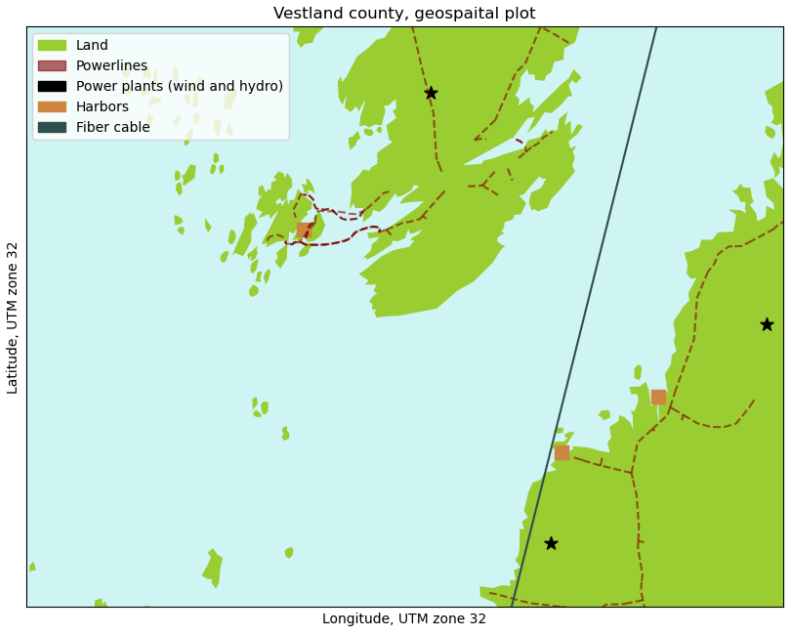

Figure 4: Area of interest: A map extraction based of real geographical data, including infrastructure like power plants, power cables, harbours and an estimated fibre cable.

This plot contains all the variables that were used in the optimisation process like harbours, power cables, fibre cables, power plants and the main land and islands. And is mostly included as a point of reference for the operations done below.

The nonlinear programming is not done by the use of package, but rather a type of brute force optimisation. Utilising functions that contain constraints, the objective function and other operations. It iterates through all combinations of x, y coordinates and discards coordinates that fall outside of the area of possibility i.e. on land. Preserving the cost values for each coordinate and returning the coordinate combination that minimises the costs needed to place the warehouse in that position. These weights are based of cost estimates for warehouse deployment, considering an increase in distance to, harbours, power cables, fibre network and surplus power sources like windmills or hydroelectric plants.

4.2 The objective function

Defining key variables for use in the objective function. In the development phase of this project, some inspiration stems from ChatGPT. The chatbot has assisted in the formulation and creation of the code (OpenAI 2023).

The chatbot was asked the following two prompts: "Create a python function that takes any given point on a x,y map and calculates the distance to the closest line"

"Can you create the an integral expression a sinus function in python expressing water temperature"

```
    # Defining som key variables for objective function
    # Variables for points of interest
    X_coordinates = np.linspace(0, 110, 111)
    Y_coordinates = np.linspace(0, 100, 101)

    docks = [[11, 55], [58, 20], [80, 30]] # [x1, y1]
    power_plants =  [[37, 90], [102, 45]]
    point_lists = [docks, power_plants]
    weights_points = [5000, 250] # Docks, Powerplants,
    # cost per extra coordinate from position

    # Variables for lines of interest
    power_cables = [(11, 55, 55, 55), (37, 55, 37, 100), (37, 55, 51, 100),
                    (77, 0, 77, 20), (59, 20, 81, 20), (81, 20, 81, 60),
                    (81, 60, 110, 60)] # (x_start, y_start, x_end, y_end)
    fiber = [(66, 0, 84, 100)] # (x_start, y_start, x_end, y_end)
    line_lists = [fiber, power_cables]
    weights_lines = [4000, 7000] # Fiber, Power_cables,
    # cost per coordinate from position
```

The weights assigned to each type of variables are estimations of the cost an additional increase in distance would generate. The power and fibre cable proximity have more or less direct costs tied to them. When deploying a warehouse it needs access to these infrastructures and there are direct costs in drawing both fibre and power cables to a given location. Based of some research and estimation the cost related to the power cables were 7000$ (Finn Halvorsen 2003), per one hundred meters, and for fibre cables an estimated 4000$ (GeoNorge 2023a) for the same distance. Considering the map grid is not symmetrical, and the weights are relative to entire degree movements, there are some differences regarding the cost related to moving east/west as opposed to north/south. This means that in reality the costs should be 10% higher when moving north or south, this impact is disregarded.

The proximity to harbours and especially power plants have a more indirect cost relation. Proximity to harbour will impact the deployment costs, as the time needed in order to successfully deploy the warehouse should increase with the distance to the closest harbour. The costs related to this are likely quite large. Despite this, allowing for a more balanced weight to avoid overpowering the analysis opens up for the other variables to have impact. The power plant variable is far more complex as it has no direct bearing on the deployment or implementation costs. However, there are benefits of placing the warehouse in relatively close proximity to a renewable energy source. Mostly this is found in the long term sustainability of the project.

4.2.1 Closest proximity, and distance from

```
    # Function for finding closest the closest dock and power plant
    def min_point_distance(x, y, point_type):
        min_point_distance = float('inf')
        for i in range(len(point_type)):
            # Calculating the squared distance from points (x, y) to the points
            distance_squared = np.sqrt((x - point_type[i][0])**2 +
                                       (y - point_type[i][1])**2)
            if distance_squared < min_point_distance:
            min_point_distance = distance_squared
        # Returns the distance to the closets point in degrees
```

```
        # for a given x, y coordinate
        return min_point_distance
```

In the function min point distance we are effectively calculating the euclidean distance between the two points. Using Pythagoras theorem where:

$$H^2 = K^2 + K^2 \tag{15}$$

$$Distance = \sqrt{xDist + yDist} \tag{16}$$

With this formula you can calculate the distance between two points. Using an arbitrary example we can calculate this by hand. We want to find the euclidean distance between the dock (11, 55) and the warehouse in point(19, 60).

$$Distance = \sqrt{(19-11)^2 + (60-55)^2} \tag{17}$$

$$Distance = \sqrt{(8)^2 + (5)^2} \tag{18}$$

$$Distance = \sqrt{64 + 25} \tag{19}$$

$$Distance = \sqrt{89} = 9.434 \tag{20}$$

The distance calculated is in degrees on our map. This is not by any means a perfect measurement as our map is 11×10 kilometres. However, as long as the costs tied to an increase in one degree, or roughly 100-110 meters, is representative, the calculations should be adequate.

```
    def min_line_distance(x, y, line_type):
        min_distance = float('inf')

        for line in line_type:
            start_x, start_y, end_x, end_y = line

            # Calculate the normalized direction vector of the line segment
            direction_vector = np.array([end_x - start_x, end_y - start_y])
            normalized_direction = direction_vector /
            ↪  np.linalg.norm(direction_vector)

            # Vector from line start to the given point
            point_vector = np.array([x - start_x, y - start_y])

            # Project the point vector onto the direction vector
            projection = np.dot(point_vector, normalized_direction)

            # If the projection is outside the line segment, use the closest
            ↪  endpoint
            if projection <= 0:
                distance = np.sqrt((x - start_x)**2 + (y - start_y)**2)
            elif projection >= np.linalg.norm(direction_vector):
                distance = np.sqrt((x - end_x)**2 + (y - end_y)**2)
            else:
                # Use the perpendicular distance to the line
                distance = abs((x - start_x) * (end_y - start_y) - (y - start_y)
                ↪  *
                               (end_x - start_x)) /
                               ↪  np.linalg.norm(direction_vector)

            # Update the minimum distance if needed
            if distance < min_distance:
                min_distance = distance
```

```
        # Returns the distance to the closest line in degrees
        # for a given x, y coordinate
        return min_distance
```

This function calculates the distance from the coordinate point to the closest point on a given line. This is done by either calculating the perpendicular[5] line distance between the two points. If the coordinate point is not in front of the given line the distance is calculated from either one of the lines endpoints. In all cases the distance measured is euclidean. The direction vector above is representing the vector for a given line, using the fibre cable as an example we can go through how this function finds the closest line and and returns the distance from the given point. We start with the the fibre cable's start and end points (66, 0, 84, 100).

$$direction\_vector = \begin{bmatrix} 84-66 & 100-0 \end{bmatrix} = \begin{bmatrix} 18 & 100 \end{bmatrix} \tag{21}$$

$$normalized\_direction = \frac{\begin{bmatrix} 18 & 100 \end{bmatrix}}{|| \begin{bmatrix} 18 & 100 \end{bmatrix} ||} \tag{22}$$

$$normalized\_direction = \frac{\begin{bmatrix} 18 & 100 \end{bmatrix}}{\sqrt{\begin{bmatrix} 18^2 & 100^2 \end{bmatrix}}} = \frac{\begin{bmatrix} 18 & 100 \end{bmatrix}}{101.6} \tag{23}$$

$$normalized\_direction = \begin{bmatrix} 0.177 & 0.984 \end{bmatrix} \tag{24}$$

Using the point (19, 60) as an example. When calculating the point_vector which is a vector going from the start of the line to the given point.

$$point\_vector = \begin{bmatrix} 19-66 & 60-0 \end{bmatrix} = \begin{bmatrix} -47 & 60 \end{bmatrix} \tag{25}$$

$$projection = \begin{bmatrix} -47 & 60 \end{bmatrix} * \begin{bmatrix} 0.177 \\ 0.984 \end{bmatrix} = 50.724 \tag{26}$$

Checking if the projection is outside of the line i.e. less than zero or larger than the vectors norm length. This releases knowledge about whether the distance should be measured as a perpendicular line from the vector to the point or if the distance should be measured from either of the vectors end points. In this case $0 < projection < 101.6$ is true therefore the distance is calculated as a perpendicular line to the vector.

$$distance = \frac{|(19-66)*(100-0)-(60-0)*(84-66)|}{101.6} = \frac{|-47*100-60*18|}{101.6} = 56.88 \tag{27}$$

```
    # Checking with the same values as the example to check answer.
    print(" The distance to the fiber cable in point (19, 60) is: ",
            min_line_distance(19, 60, fiber))
```

```
The distance to the fiber cable in point (19, 60) is:  56.88579612567959
```

[5] A perpendicular line refers to a line that is at a 90°angle two a given line

4.2.2 Water temperature

The reason underwater data warehouses are a supposedly a viable option for data storage is the benefit of "cost effective" cooling systems, saving resources on heat-management and increasing the longevity of a data centre (Mary Zhang 2022). Since the water temperature is such a large part of this concept, including an element indicating the possible cost impact of temperature, is not so far fetched. The water temperature is expressed as a a recursive equation based of the month as a time variable, being dependent on it's past temperatures. Including a sinus element meant the temperatures could be measured in cycles, in accordance with real seasonal changes. The main difficulty with this function was estimating a realistic cost relative to it's impact. These considerations resulted in this expression:

$$x_{t+1} = (x_t + a\sin(\frac{2\pi t}{12}) + b)c \tag{28}$$

The variables in this function are explained as followed:

x = Water temperature

t = The current time, measured in month

a = amplitude of seasonal changes

b = base water temperature

c = climate change coefficient

The term $a * sin(2\pi t/12)$ is circular, and every 12^{th} month the temperature has moved in a cycle from cold to hot to cold again. The amplitude (a) controls the impact this has, a larger a means larger seasonal fluctuations. The b term functions as a baseline. The aim with these two terms was being able to control the positioning of the warehouse. As the coordinates moved further east the seasonal fluctuations would decrease as the position is further from the coast, similar to how the actual water temperature moves. Positioning the warehouse further south would increase the b term making the water slightly warmer. The opposite reaction was initiated when the warehouse moved west or north. These effects are supposed to simulate the actual changes in water temperature across latitude and longitude movements(NOAA 2018). Additionally, as the project is expected to last 40 years we wanted to add a coefficient that expressed the impacts of climate change over time, hence the c term impacting the entire expression, exponentially increasing the temperature. Although the warehouse depth would likely decrease both the seasonal and baseline impacts in a realistic scenario, the interesting insights that this function can provide would be lost with to small coefficients. Given that the area of interest is rather small there are no extreme impacts, given the weights used to change a and b. However, if this function were to be used across the entire Norwegian coast the function weights would need to be significantly altered.

```
# Defining water_temp function variables
t0 = 5  # The average water temperature in haugesund in december
a = 3 # amplitude - How volitoale is the temp
b = 0 # base temprature
c = 1.0075 # Climate change coef
degree_cost = 10 # The weight of temprature changes

# variables needed for integration
start = 1
stop = 481
rectangles = 481
width = (stop - start) / (rectangles - 1)
```

The base definition of these variables are for the coordinates (55, 50) as they are the centre of the map. The numbers are mostly arbitrary, a and b where defined based on the impact we wanted them to have in (55, 50) to represent Haugesund's normal temperatures (sea temprature.net 2023). The degree_cost is set to ten as an arbitrary number rather representing the weight of the temperature fluctuations more so than an actual cost. Looking further into the mathematics of this expression we can calculate the temperature of a given example where t = 1.

$$x_{t+1} = (x_t + a\sin(\frac{2\pi t}{12}) + b)c \tag{29}$$

$$x_1 = (x_0 + 3\sin(\frac{2\pi 1}{12}) + 0)1.0075 \tag{30}$$

$$x_1 = (5 + 3\sin(0.523))1.0075 \tag{31}$$

$$x_1 = (5 + 0.0274)1.0075 = 5.065 \tag{32}$$

The actual function is more complex having additional elements where the a and b term change as the coordinates stray from the map centre. Again using arbitrary numbers to weight this impact, in such a way to not overpower the optimal position but still leave an impact on the results. The function also uses integration as this cost is for the entire life cycle of the warehouse this should be accounted for. Integrating the function allows us to get the area below the curve or in this case the cumulative temperature given the x and y coordinates.

```
    def water_temp(x, y):
        # Defining variables localy
        a = 3
        b = 0
        c = 1.0075
        temp = [5]

        # Checking coordinate value to alter volatility and base temprature
        if x == 55:
            base_a = 0
        else:
            base_a = (x - 55)*0.001
        if y == 50:
            base_b = 0
        else:
            base_b = -(y - 50)*0.001
        # Adding changes in base temprature and volatility based of coordinates
        a += base_a
        b += base_b

        # Creating a list of all months and tempratures those months
        months = np.linspace(1, 480, 480)
        for t in range(len(months)-1):
            temprature = (temp[-1] + a * np.sin(2*np.pi*t/12) + b)*c
            temp.append(temprature)
        temp.pop(0) # Removing initial month 0

        # Integrating these numbers to get area below curve
        # or total water     temprature for project lifetime
        L_rieman = width * sum(temp[:-1])
        R_rieman = width * sum(temp[1:])
        total_water_temp = (L_rieman + R_rieman) / 2

        # Returns the cumulative total water temprature
        # for a given x, y coordinate
        return total_water_temp
```

The water temperature function above uses the Riemann integral for calculating the area below the curve. The Riemann integral uses rectangles to estimate the area below the curve, increasing the number of rectangles will make the calculation more precise. The width was set to = 1, meaning all observations are summed up. It was decided that finding the average of the left and right Riemann integral would be a sufficient way to estimate the area below this curve. The expression above is somewhat complex in that it has several conditions changing it's values based on the input given, it also finds the integral of the expression to calculate the area below the curve, expressing the total temperature expected in the coordinates. Previously we have explained how the temperature function finds a specific temperature for x_1. However in order to calculate the integral for a given point it is slightly more convoluted. In general the expression can be formatted to equal any temperature x_t in coordinates 55, 50:

$$x_t = (x_0 + \sum_{0}^{t} (a \sin(\frac{2\pi t}{12}) + b))c^t \tag{33}$$

$$x_t = (5 + \sum_{0}^{t} (3 \sin(\frac{2\pi t}{12}) + 0))1.0075^t \tag{34}$$

This general expression gives us the temperature for any given time (month) for the coordinates 55, 50. Now in order to calculate the Riemann integral we need a sum of all the rectangles. Since we made the width of these = 1 we in reality we are finding the sum of all temperature observations from 0 to t.

$$I_t = \sum_{0}^{t} ((5 + \sum_{0}^{t} (3 \sin(\frac{2\pi t}{12}) + 0)) * 1.0075^t) \tag{35}$$

or

$$I_t = \sum_{0}^{t} x_t \tag{36}$$

Now this expression is quite complex but breaking it down and using low numbers of iteration will allow us to explain how the function works on a basic level. If we want to find the area below the curve where $t = 2$ it would look like this.

$$I_2 = \sum_{0}^{2} x_t = x_2 + x_1 \tag{37}$$

So in order to calculate the area below the curve from $t = 0$ to $t = 2$, we calculate the individual x_1 and x_2 values.

$$x_1 = (5 + \sum_{0}^{1} (3 \sin(\frac{2\pi t}{12}) + 0))1.0075^t \tag{38}$$

$$x_1 = (5 + 3 \sin(\frac{\pi 1}{6}))1.0075^1 \tag{39}$$

$$x_1 = (5 + (1.5))1.0075 = 6.54875 \tag{40}$$

$$x_2 = (5 + \sum_{0}^{2} (3 \sin(\frac{2\pi t}{12}) + 0))1.0075^t \tag{41}$$

$$x_2 = (5 + (3 \sin(\frac{\pi 2}{6}) + 0) + (3 \sin(\frac{\pi 1}{6}) + 0))1.0075^2 \tag{42}$$

$$x_2 = (5 + (2.6) + (1.5))1.015 = 9.2365 \tag{43}$$

$$I_2 = x_2 + x_1 \tag{44}$$

$$I_2 = 9.2365 + 6.54875 = 15.78525 \tag{45}$$

```
# Checking the actucal values to confirm correct answer
print(" The integral of I_2: ", sum(temp[:2]))
```

```
The integral of I_2:  15.614653424761832
```

Additionally it was desirable to plot the temperature function with it's integrated area. This allows for a visualisation of how the seasonal changes impact the temperature as well as the impact climate change is expected to have on our temperature estimates.

```
# Plotting the temprature curve for coordinates 55, 50
months = np.linspace(1, 480, 480)
temp = []
temp.append(t0)
for t in range(1, len(months)+1):
    temprature = (temp[-1] + a * np.sin((2*np.pi*t)/12) + b)*1.00075
    temp.append(temprature)
temp.pop(0) # Removing initial month 0

L_rieman = width * sum(temp[:-1])
R_rieman = width * sum(temp[1:])
avg_total_water_temp = (L_rieman + R_rieman) / 2
print("The total water temperature in (55, 50) is: ", avg_total_water_temp)

# Plotting the water temprature
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(12, 4))

for i in range(0, 2):
    # Plot on the left subplot
    axs[i].plot(months, temp, c="indianred", linewidth=1,
                label="Water Temprature")
    if i == 1:
        axs[i].bar(months, temp, width=width, label="Area below graph",
                color="skyblue")
    axs[i].set_xlabel("Months from project start")
    axs[i].set_ylabel("Water temprature in (55,50)")
    axs[i].set_title("Water temperature accross project lifespan")

plt.tight_layout()
plt.legend()
plt.show()
```

```
The total water temperature in (55, 50) is:  6119.506416519455
```

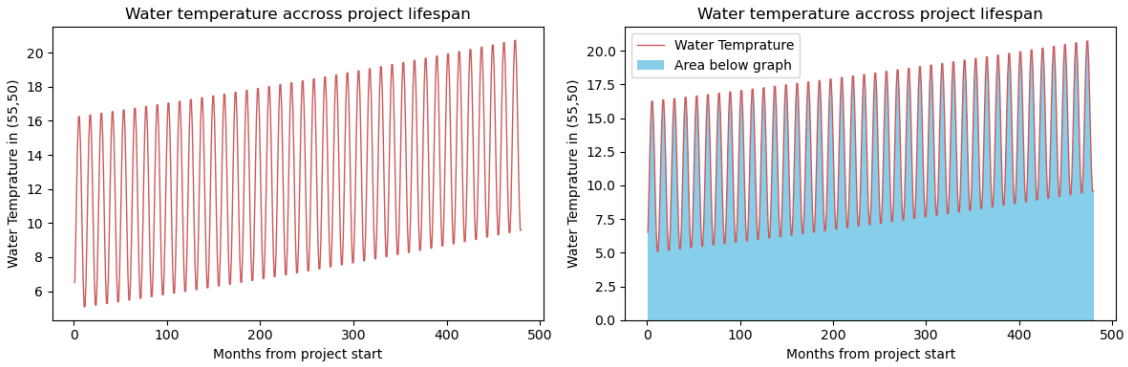

Figure 5: The left figure displays the expected water temperature across the the entire project lifespan in coordinates 55, 50. The right figure is the same curve only with the additional area below visualising what the Riemann integral outputs from the given coordinates 55, 50.

4.2.3 Objective function

The mathematics behind the objective function are rather simple as we are only calling back the results of the other functions explained previously and multiplying the results with their respective weight. Then compiling these costs into a single variable.

```
    def objective_function(x, y):
        cost = 0
        # Running function to find the closest dock and powerplant
        # then adding their costs
        for i, point_type in enumerate(point_lists):
            point_distance = min_point_distance(x, y, point_type)
            cost_of_points.append([x, y, weights_points[i] * point_distance])
            cost += weights_points[i] * point_distance

        # Running function to find the closest fiber cable and power cable
        # then adding their costs
        for j, line_type in enumerate(line_lists):
            line_distance = min_line_distance(x, y, line_type)
            cost_of_lines.append([x, y, weights_lines[j] * line_distance])
            cost += weights_lines[j] * line_distance

        # Running function to find the total water tempratures
        # and adding their costs
        total_temp = water_temp(x,y)
        cost_of_temprature.append([x, y, degree_cost * total_temp])
        cost += degree_cost * (total_temp / avg_total_water_temp)

        return cost, cost_of_points, cost_of_lines, cost_of_temprature
```

However, there are some additional elements that should be explained. Firstly there are several outputs besides just the cost, these are used in visualisations later and have no real bearing on the results. There are some changes to the temperature cost as it was of interest to lower it's effect we divide by the average total temperature, keeping the sign but lowering the overall cost and by extension it's impact on the general results. By dividing with the average, the cost added from this function represents a deviation rather than a large total sum.

4.3 The optimisation function

The optimisation function below utilises brute force optimisation through *if* statements, these statements are constraints that restricts the optimal warehouse position to be on land by confining rectangular and circular areas and checking if the x, y coordinate combination is within those bounds. Before running the function and looking at the results, there are some mathematical functions that should be explained.

```
    # Defining variables for optimization function
    # Variables for avoiding land areas // taken from the sketch map
    min_cost = float("inf")
    Cx, Cy, Cradius = 11, 60, 5
    S1xmin, S1xmax, S1ymin, S1ymax = 22, 55, 50, 100
    S2xmin, S2xmax, S2ymin, S2ymax = 58, 80, 0, 20
    S3xmin, S3xmax, S3ymin, S3ymax = 80, 110, 0, 60

    # Saving lists to plot costs later
    cost_by_coordinates = []
    cost_of_points = []
    cost_of_lines = []
    cost_of_temprature = []
```

Above all of the variables needed for these constraints are defined. These variables are important in restricting the warehouse position to water. The rectangular constraints are mathematically simple. By defining min and max values for both x and y coordinates you can confine an entire rectangle, as long as not both x and y are within those ranges, the statement remains false and the combination plausible. The circular constraint works similar. Firstly we find the distance the x, y coordinates have from the centre of the circle which is represented by the Cx, Cy variables. This distance is squared in order to find the euclidean distance, similarly to the calculations for straight line between a location and a harbour. Then this distance is compared to the circle's radius, if it is less or equal the coordinates are discarded as they are not plausible. Running through a quick example with the point (19, 60) the calculations look like this:

$$(x - Cx)^2 + (y - Cy)^2 <= Cradius^2 \tag{46}$$

$$(x - 11)^2 + (y - 60)^2 <= 5^2 \tag{47}$$

$$(19 - 11)^2 + (60 - 60)^2 <= 25 \tag{48}$$

$$(8)^2 <= 25 \; so \; 64 <= 25 \tag{49}$$

The statement is false therefore the coordinate combination is possible, at least for the circular area constraint.

```
    # Function running the actual optimization, itterating through all x, y
    ↪ combinations
    # if the values passes the land constraints they run through the objective
    ↪ function to find their respective cost
    # Ultimatly the function passes out the min_cost combination of x,y
    def optimization():
        min_cost = float("inf")

        # These nested for loops run through all itterations of x,y
        ↪ combinations
        # If they meet any of the if/elif conditions the function skips to the
        ↪ next coordinate combination
        # This allows only the combinations outside of these areas to run
        ↪ through the objective function
```

```
        for x in X_coordinates:
            for y in Y_coordinates:
                # Constraining off the island land area
                if (x - Cx)**2 + (y - Cy)**2 <= Cradius**2:
                    continue
                # Constraining off all the square areas
                elif (S1xmin <= x <= S1xmax and S1ymin <= y <= S1ymax) or \
                        (S2xmin <= x <= S2xmax and S2ymin <= y <= S2ymax) or \
                        (S3xmin <= x <= S3xmax and S3ymin <= y <= S3ymax):
                    continue
                else:
                    # Evaluate the objective function
                    # for points outside the circle and rectangle
                    cost = objective_function(x, y)
                    total_cost = cost[0]
                    cost_by_coordinates.append([x, y, total_cost])
                    #print("For coordinates (",x, y,"), cost is: ",
                    ↪  round(cost))

                    # Saving or overwriting the minimal cost x,y combination
                    if total_cost < min_cost:
                        min_cost = total_cost
                        best_x, best_y = x, y

        print("\nLowest cost combination found at coordinates
                ", best_x, best_y,") with cost: ", round(min_cost))
        return cost_by_coordinates, best_x, best_y, min_cost
    optimized_position = optimization()
    cost_by_coordinates, best_x, best_y, min_cost = optimized_position[0],
    ↪  optimized_position[1], optimized_position[2], optimized_position[3]
```

```
Lowest cost combination found at coordinates ( 79.0 30.0 ) with cost:  55883
```

4.4 The optimal warehouse position

Below is a large code block dedicated to plotting different maps from the area of interest. It was decided to make one large plot containing all the information needed to see how the analysis worked and the results they provided.

```
    # A function adding all general map infrastructure
    # like landmasses, and points/lines of interest
    def map_infrastructure(ax):
        # Adding shapes to plot
        ax.add_patch(Rectangle((22, 50), 33, 50, color= "yellowgreen"))
        ax.add_patch(Rectangle((58, 0), 22, 20, color= "yellowgreen"))
        ax.add_patch(Rectangle((80, 0), 30, 60, color= "yellowgreen"))
        ax.add_patch(Circle((11, 60), radius=5, color= "yellowgreen"))

        # Adding lines to plot
        for list_index, line in enumerate(line_lists):
            for i in range(len(line)):
                ax.plot((line[i][0], line[i][2]), (line[i][1], line[i][3]),
                        c = line_elements[list_index],
                        label = line_elements[2+list_index],
```

```
                        linestyle = line_elements[4+list_index])

        # Adding points to plot
        for list_index, point in enumerate(point_lists):
            for j in range(len(point)):
                ax.scatter(point[j][0], point[j][1], c =
                ↪  point_elements[list_index],
                            label = point_elements[2+list_index],
                                marker = point_elements[4+list_index], s=70)

        # Plotting the optimal position
        ax.scatter(best_x, best_y, c="gold", marker="x", s=80, label = min_cost)

        ax.set_xlim(0, 110)
        ax.set_ylim(0, 100)


    def split_coordinate_list(coordinate_list):
        # Function splitting the cost coordinate lists into three arrays
        x = [point[0] for point in coordinate_list]
        y = [point[1] for point in coordinate_list]
        costs = [point[2] for point in coordinate_list]
        return x, y, costs
```

```
    line_elements = ["darkslategrey", "maroon", "Fiber", "Power_cables",
                        "solid", "dashdot"]
    point_elements = ["peru", "black", "Docks", "Powerplants", "s", "*"]

    fig = plt.figure(figsize=(10,10))

    # Sektch map
    ax0 = plt.subplot2grid((7,7),(0,0), colspan=3, rowspan=2)
    ax0.add_patch(Rectangle((0, 0), 110, 100, color= "paleturquoise",
    ↪  alpha=0.6))
    map_infrastructure(ax0)

    # Actual map
    ax1 = plt.subplot2grid((7,7),(0,3), colspan=3, rowspan=2)
    vestland["geometry"].plot(ax=ax1, color="paleturquoise", alpha=0.6)
    plot_actual_map(ax1)

    # Point distance heatmap
    split_list = split_coordinate_list(cost_of_points)
    x, y, costs = split_list[0], split_list[1], split_list[2]

    ax2 = plt.subplot2grid((7,7),(2,0), colspan=2, rowspan=2)
    ax2.scatter(x, y, c=costs, cmap="Blues", s=100, edgecolors="none",
    ↪  linewidth=1)
    map_infrastructure(ax2)

    # Line distance heatmap
    split_list = split_coordinate_list(cost_of_lines)
    x, y, costs = split_list[0], split_list[1], split_list[2]

    ax3 = plt.subplot2grid((7,7),(2,2), colspan=2, rowspan=2)
    ax3.scatter(x, y, c=costs, cmap="Blues", s=100, edgecolors="none",
    ↪  linewidth=1)
```

```
    map_infrastructure(ax3)

    # Water temprature heatmap
    split_list = split_coordinate_list(cost_of_temprature)
    x, y, costs = split_list[0], split_list[1], split_list[2]

    ax4 = plt.subplot2grid((7,7),(2,4), colspan=2, rowspan=2)
    ax4.scatter(x, y, c=costs, cmap="Blues", s=100, edgecolors="none",
    ↪  linewidth=1)
    map_infrastructure(ax4)

    # Total cost heatmap
    split_list = split_coordinate_list(cost_by_coordinates)
    x, y, costs = split_list[0], split_list[1], split_list[2]

    ax5 = plt.subplot2grid((7,7),(4,0), colspan=6, rowspan=3)
    ax5.scatter(x, y, c=costs, cmap="Blues", s=100, edgecolors="none",
    ↪  linewidth=1)
    map_infrastructure(ax5)

    ax6 = plt.subplot2grid((7,7),(0,6), colspan=1, rowspan=7)
    norm = mpl.colors.Normalize(vmin=min(costs), vmax=max(costs))
    mpl.colorbar.ColorbarBase(ax6, norm=norm, cmap='Blues',
    ↪  orientation='vertical')
    ax6.set_ylabel("Costs")

    axes = [ax0,ax1,ax2,ax3,ax4,ax5]
    axes_names = ["Sketch Map","Actual Map","Cost of point distance",
                  "Cost of line distance", "Cost of water temprature",
                  "Total cost by coordinates"]
    for i in range(len(axes)):
        axes[i].set_title(axes_names[i])
        axes[i].set_xlabel("X_coordinates")
        axes[i].set_ylabel("Y_coordinates")
        axes[i].set_xticks([],[])
        axes[i].set_yticks([],[])

    # fig.sub_title('sd')
    fig.suptitle("Site selection optimisation", fontsize=20)

    fig.tight_layout()
    plt.savefig("site_selection.png")
    ax5.legend(handles=handles, loc=3)
    plt.show()
```

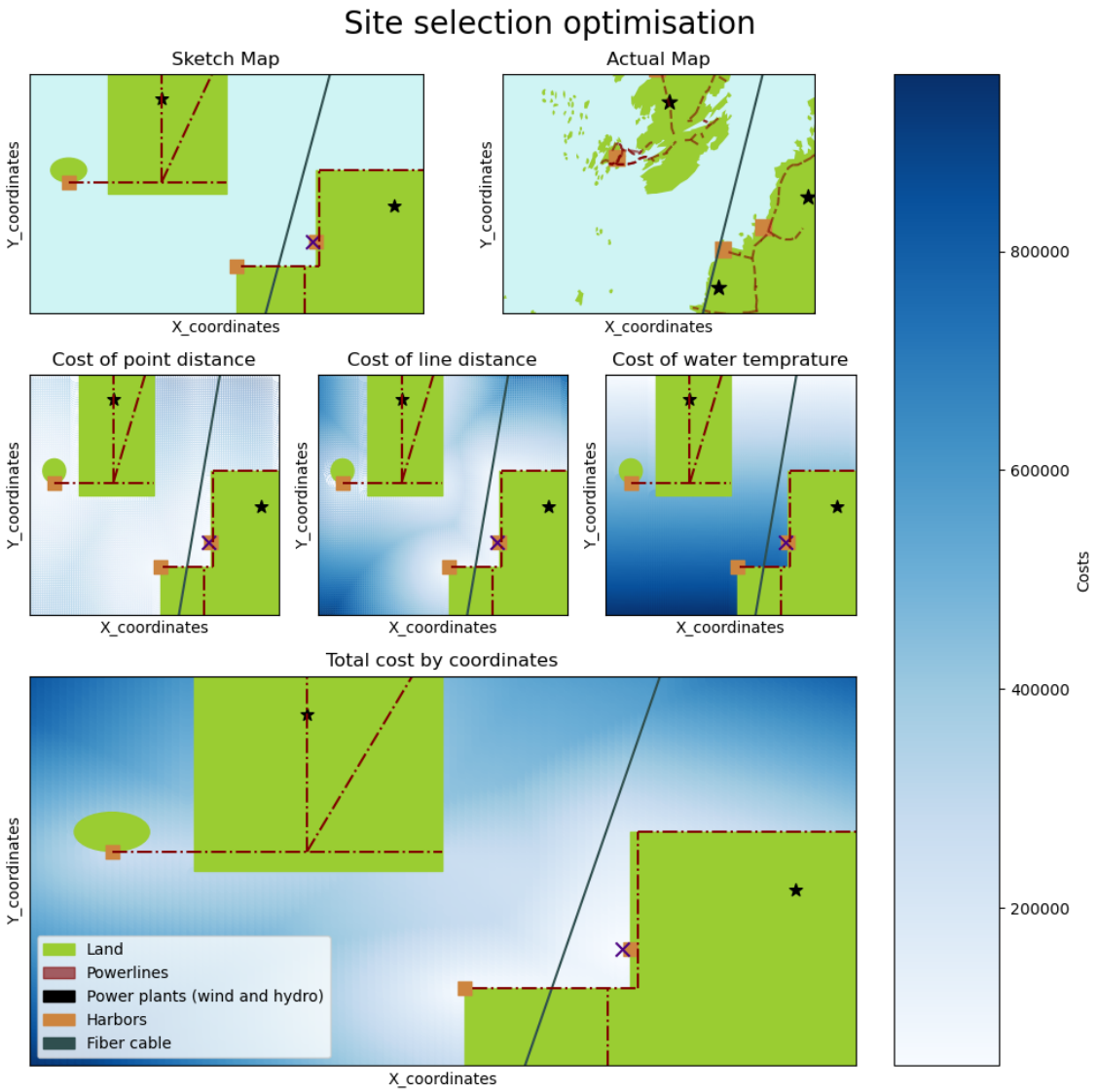

Figure 6: A compilation of all stages of the analysis, including the original sketch map, made from the estimated positions of various infrastructure (top left figure), the actual map (top right figure), the individual variable cost impact displayed with a water colour heat map (middle row figures) and the final results with a similar heat map, compiling all the previous variables impact.

On the first row to the right of this plot the actual area of interest is plotted at the start of the optimisation section, containing real geographical data. To it's left is the simplified sketch map that was devised as a framework to base the decisions off. Using simple shapes rather than complex coastlines allowed for easily constraining off the mainland and islands. On the second row is a heat map where the ocean colour represent the cost that position entails. We decided to create one for each of the variable functions, as they represent a layer of the final product. The bottom heat map shows the final cost distribution, confirming that the optimal position is in a low cost area, close to all the important infrastructure.

In general the cost ranges from seemingly 55 000 to 1 000 000 dollars. The lighter colours are always a sign of a lower cost but the strength of their colours are only accurate in their relative map. This allows us to see which variables have the biggest impact on the results. In this case the line distance seems to have the biggest impact, this is in line with how the model and weights were created. As both line elements have a high cost tied to them, the point types only have any significant costs tied to harbour distance. Water temperature was created with the intention of having a muted impact as the area of interest is relatively small and the real impact would likely

be less than the included impact in this analysis. The results where the point (79, 30) marked with a purple cross. It is in a position closest to both harbour and power cables, which are the two variables with the highest weights. While also being close to the fibre cable and relatively neutral to both the temperature and power plant variable. These results are as expected, proving that the model is functioning as designed, and is basing it's decisions of the given weights.

5 Deployment cost

This section is dedicated to creating a deployment cost function. The previous section used several elements attempting to determine the optimal position of the warehouse, with the costs being more akin to relative weights rather than actual costs. This function removes elements not tied directly to the deployment phase, like water temperature, this makes it possible to estimate the projects deployment costs.

```
    # Creating a function only for the costs tied to deployment.
    weights_points = [350, 0] # Redefining the actual costs for deployment
    def deployment_cost_function(x, y, cubic):
        deployment_cost = 0
        # Running function to find the closest harbour and powerplant
        #then adding their costs
        for i, point_type in enumerate(point_lists):
            point_distance = min_point_distance(x, y, point_type)
            deployment_cost += weights_points[i] * point_distance *
                                    cubic + 100000

        # Running function to find the closest fiber cable and power cable
        #then adding their costs
        for j, line_type in enumerate(line_lists):
            line_distance = min_line_distance(x, y, line_type)
            deployment_cost += weights_lines[j] * line_distance
        # Returns the deployment cost from the given x, y coordinates
        return deployment_cost
```

This function is an expression for the deployment costs given the best x and y coordinates. Additionally it takes the argument of cubic as we expect the deployment costs to increase with the size of the warehouse. This relation is expressed linearly with the addition of a constant element, due to a majority of the costs being tied to the process of deployment rather than the size of what is being deployed. Not adding this element would create a flaw where small warehouses would seem extremely cheap to deploy compared to their bigger counterparts. Apart from this the relationship is estimated to be linear allowing us to also control for the warehouse size. The weight tied to power plants are removed as they pose no cost to deploying the warehouse. Weights for power cables and fibre cables remain the same as their weights were cost estimates.

6 Operational costs

After the position of the warehouse is decided it moves through both production and deployment phases, leaving the operational costs. After the warehouse is deployed it needs power, and is compiling a electricity bill. Additionally we expect it to need regular maintenance, Microsoft has deemed underwater data centres as more reliable than their land based counterpart. However, we still expect failures as the internal technology is highly fragile. Based of this we have estimated the project lifetime to be 40 years, and regular maintenance checks conducted every five years. By reusing the water temperature function the costs can be estimated more precisely, only using more accurate costs values in addition to the costs being relative to the warehouse volume. We estimated that a maintenance check would cost upwards of 200 000$, this number multiplied by the total amount of maintenance checks results in $40/5 = 8$ -> we get $8 * 200000 = 1600000$. The maintenance costs included with the costs related to temperature leaves this function for operational costs.

```
    power_cost = 45.2
    def operational_cost_function(x, y, cubic):
        operational_cost = 0
        # Running function to find the total water tempratures
        # and adding their costs
        total_temp = water_temp(x,y)
        operational_cost += power_cost * (total_temp/avg_total_water_temp) *
        ↪  cubic

        # Cost for maintenance depends on the distance to closest harbour
        point_distance = min_point_distance(x, y, docks)
        operational_cost += 1600000 * point_distance

        # Returns the operational costs from the given x, y coordinates
        #and the size of the warehouse in cubic meters
        return operational_cost
```

In order to estimate the power cost we borrowed some numbers from the only past project of this kind, Microsoft's Natick 2, this warehouse was around $340m^3$ using 240 kw at max output (Mary Zhang 2022). The Natick 2 had 864 servers, per server the max output would therefore be 0.277 kw or 0.277 kWh for each hour of max output power per server per cubic meter. Multiplying this by ten [6] results in 2.77 kWh of power needed for each cubic meter per hour. Multiplying this with 24 and 30 for hours and days produces an estimate of how many kWh a cubic meter in the warehouse is expected to use over the course of one month. This number is 2 000 kWh, multiplying it with the kWh price of 0.226 dollar this equals 452 dollars per month per cubic meter. Based of these number we can calculate the per degree cost of Natick 2's average temperature which is about 10, so the dollar cost per degree is 45.2 dollars.

The average temperature in the Orkney islands (sea temprature.org 2023) are similar to Haugesund (sea temprature.net 2023) with a slightly tighter range. Now these calculations are quite far fetched from actual scientific standards and it is not believed that temperature behave like illustrated in the calculations above. However, as this project is an exam deliverable for an applied mathematics course we concluded it would be more beneficial to focus on how to utilise mathematics to work towards a goal rather than getting consumed by the details of science and physics that would be relevant if this should be of any use in a practical application.

[6] There are 10 servers per cubic meter 3.1.

7 Discounted cash flow analysis

The calculations and functions for generating costs have been completed. Now the second half needed for the financial analysis will be estimated; the revenues. When the volume, location, revenues and cost of the project have been decided, you can start analysing the project from a financial perspective. In this project the financial analysis is a discounted cash flow model (DCF). The method aims at evaluating the project based on future cash flows across it's entire period(Investopedia 2023b). The DCF model consists of the following parts: production-, deployment-, operational costs, grant from Innovasjon Norge, revenues, depreciation's and taxes.

The output of the model are two cash flows, a non-adjusted- and a discounted cash flow. These cash flows are used to produce key financial insights about the warehouse project, such as net present value, payback time, and internal rate of return. These financial figures serve as a foundation for decision making and give an indication of the consequences of investing in the warehouse from a financial point of view.

7.1 Retrieving cost and revenues

This section retrieves the original costs and revenues of the project. The more traditional way of doing financial modelling is by means of spreadsheets and tabular setup. However, for this assignment the procedures will be done by means of matrix algebra, as this is more inline with the course curriculum. Therefore, when retrieving the revenues and costs the output is in formats of one dimensional matrices, with a range of 41 representing the the project lifespan of years 0 to 40. Using costs in this format allows for matrix algebra, and can satisfy outer and inner conditions[7].

7.1.1 Costs

The cost function consists of three part: production-, deployment- and operational costs. The following function returns each of these costs. For the production- and deployment costs the cost occurs at the first index of each matrix, the rest of the matrix is filled with zeros. The operational costs have been aggregated and then divided into the last 40 indexes of the matrix.

One dimensional matrix for production cost:

$$P = \begin{bmatrix} -p_cost & 0 & ... & 0 \end{bmatrix} \tag{50}$$

One dimensional matrix for deployment cost:

$$DC = \begin{bmatrix} -D_cost & 0 & ... & 0 \end{bmatrix} \tag{51}$$

One dimensional matrix for operational costs:

$$OC = \begin{bmatrix} 0 & O_cost & ... & O_cost \end{bmatrix} \tag{52}$$

```
    def total_cost(x, y, size):
        #Negative production costs
        PC = np.append(-production_cost_function(size), np.zeros(40))
        #Negative deployment costs
        DC = np.append(-deployment_cost_function(x,y,size), np.zeros(40))
        #Negative operational total cost divided by 40 years and size.
        OC = np.append(np.zeros(1),
                np.repeat(-1*operational_cost_function(x, y, size)/40,40))

        #Returns costs for production, deployment and operation.
```

[7] In matrix algebra outer and inner conditions refer to the process of confirming eligibility of multiplication (inner), and apprehending the size of the resulting matrix (outer) (Johannes Mauritzen 2023)

```
        #In a 41*1 array format.
        return [PC,DC,OC]
```

7.1.2 Revenues

The original revenues consists of two part, general revenue streams from the sale of storage capacity and the grant for related to the underwater data centre concept. The third part is the residual value of the investment, which is discussed later in the project.

The first part is an assumed grant from Innovasjon Norge (Innovasjon Norge 2023). The size of this grant is assumed to have a direct linear relation to the costs. If the project only costs 100'000 the project can achieve a maximum of 50% grant of the total costs. If the project costs 10'000'000, the grant percentage is only 25%. The support function is able to calculate this automatically. Calculating the grant for a project with a cost of 5'000'000 looks like this:

$$Grant\% => 37\% = 0.5 + ((5'000'000 - 100'000)/9'900'000) * -0.25 \tag{53}$$

The grant revenue looks like this in a matrix format:

$$DC = \begin{bmatrix} Grant & 0 & ... & 0 \end{bmatrix} \tag{54}$$

The next part are the annual revenues generated from storing data. It's assumed that the price per GB of storage is 0.026 USD and that each cubic holds up to 3000 GBs (Bryn Burns 2021). Therefore the annual revenue per cubic of storage is.

$$Annual revenue => 936 = 0.026USD * 3000GB * 12months. \tag{55}$$

The annual revenue looks like this in a matrix format:

$$AR = \begin{bmatrix} 0 & 936 * volume & ... & 936 * volume \end{bmatrix} \tag{56}$$

```
    def support(cost):
        #Defining min and max for support
        min_supp = 100000
        max_supp = 10000000
        #Defining min and max percentage support
        min_supp_pct = 0.25
        max_supp_pct = 0.5
        #Calculating the  percentage of support the project shall recieve
        support = max_supp_pct + ((cost-min_supp)/(max_supp-min_supp)) *
        ↪  -min_supp_pct
        #Returning the support recieved
        return support * cost


    def total_revenue(size, cost):
        #Calculating the yearly revenue for each cubic
        price_per_giga = 0.026
        giga_per_cubic = 3000
        yearly_rev_pr_cubic = (giga_per_cubic * price_per_giga) * 12

        #Generating cashflows for revenues
        #Revenues
        R =  np.append(np.zeros(1),np.repeat(yearly_rev_pr_cubic,40)*size)
```

```
        #SUport revenue
        SR = np.append(support(cost), np.zeros(40))

        #Returning the revenues
        return [R, SR]
```

The total revenue function returns both the grant and the annual revenues. The residual value of the warehouse is also treated as a revenue. However, this revenue is first estimated during the DCF calculations.

7.1.3 Deciding optimal volume

The functions for revenues and costs have been built. These functions requires a input volume to generate costs and revenues. Therefore, the next function will produce costs and revenues for different values of volume. This gives insights concerning the optimal size of the warehouse.

```
    def rev_cost():
        #Range of volumes
        cubics = np.arange(0,321,16)[1:]
        #Cost of volume per range
        volume_costs = np.array([sum(sum(total_cost(best_x,
                                    best_y,cubics[i]))) for i in range(20)])
        #Revenue of volume per range
        volume_revenues = np.array([sum(sum(total_revenue(cubics[i],
                    -1*volume_costs[i])[:2])) for i in range(len(cubics))])

        return [cubics, volume_revenues, volume_costs]
```

The next step is determining marginal cost and revenues for the project. Based on microeconomic theory and producer theory it is considered optimal to produce the the amount of volumes that lower the marginal costs (Investopedia 2023e). The producer theory combined with consumer theory results in the optimal volume size to achieve the highest profit (Investopedia 2023d). The optimal production point is where marginal costs intercept with marginal revenues. However, from this projects point of view, there are no intercepts with marginal costs and -revenues. Since there are no intercepts for these graphs, the optimisation of profit was altered into optimisation of costs. Due to the economies of scale the marginal costs vary substantially. It is of interest to invest in an object when the marginal costs is at their lowest, and therefore the optimal investment points are in the valleys, and not on the peaks of the graph.

```
    def volume_decision(data):
        vol_range, vol_rev, vol_cost = data[0], data[1], data[2]

        #Defining marginal revenues and costs
        MC = np.diff(vol_cost)
        MR = np.diff(vol_rev)
        #Finds peaks in MRs
        peaksMC =  scipy.signal.find_peaks(np.diff(vol_cost))

        invest_diffs = []
        invest_xs = []
        for i in range(len(vol_cost)):
            if i in peaksMC[0]:
                invest_diffs.append(MC[i]*-1)
                invest_xs.append(32+i*16)
```

```
        return [MR, MC, [invest_xs,invest_diffs]]
```

```
revcost =  rev_cost()
vol_dec = volume_decision(revcost)

fig, ax = plt.subplots(1,2, figsize=(10,5))
ax[0].plot(revcost[0], revcost[1], color='green', label='Revenues')
ax[0].plot(revcost[0], revcost[2] *-1,color='red', label='Costs')
ax[0].set_title('Revenues and costs')
ax[0].legend()

ax[1].plot(revcost[0][1:], vol_dec[0], color='green',
                marker='o', label='MR')
ax[1].plot(revcost[0][1:], vol_dec[1]*-1, color='red', label='MC')
ax[1].scatter(vol_dec[2][0], vol_dec[2][1], color='black',
                      label='Investment points')
for i in range(len(vol_dec[2][0])):
    ax[1].text(vol_dec[2][0][i],vol_dec[2][1][i], str(vol_dec[2][0][i]))
ax[1].set_title('Marginal  revenues and costs')
ax[1].legend()
fig.suptitle('Investment based on volumes')
plt.show()
```

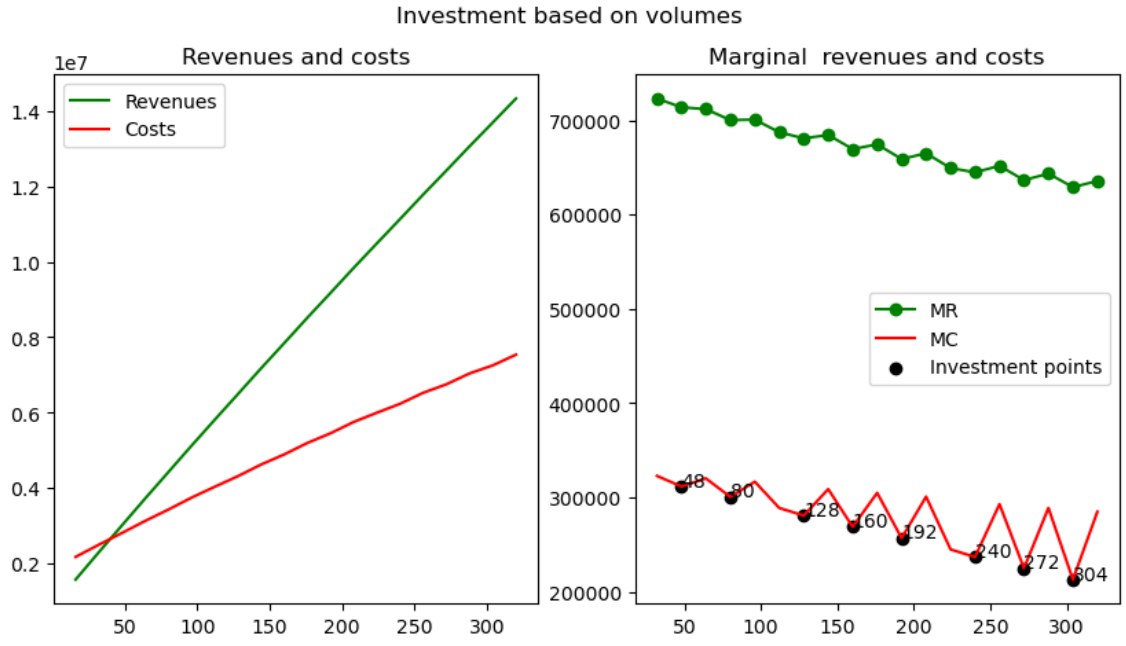

Figure 7: The left plot visualise the total revenues and -costs. The right plot shows the marginal revenues and -costs.

Observing the visualisation to the left above one can see that the the project generates profit when the volume of the warehouse exceeds approximately 40 cubic meters. This means that the project begins to generate profit for every increased volume after 40 cubic meters. The right plot has 20 peaks and valleys on the MR curve. This is due to the fact that there are only 20 possible configurations of the warehouse, because increases or decreases of the warehouse can only happen at 1 meter in length or 16 cubic meters.

There are only 8 valleys in the MR graph, and these are the data points where one can minimise the marginal costs. From a business perspective it is desirable to get as much profit as possible unless it results in negative externalities (Investopedia 2022). This is assumed to not be the case and therefore, the optimal volume of the warehouse is 304 cubic meters.

7.2 Discounted cash flow model

The model takes into consideration the revenues and costs mentioned earlier as well as depreciation, taxes and residual value on the warehouse and the discount rate.

7.2.1 Calculations

Starting of by defining various input. The volume is defined as 304 cubic meters. The general tax rate for the project is 22% (Regjeringen.no 2023). The depreciation rate is set at 5% in regard to the Norwegian standard for depreciating fixed cost, the warehouse is assumed belong to "depreciation group g" (Skatteetaten 2023). The discount rate has been set at 5%. No calculations have been performed on the WACC [8], since it is not out of scope for this project. These inputs in addition to the optimal position of the warehouse, costs and revenues will be the data used in the financial analysis.

```
1    decided_volume  = 304
2    tax_rate = 0.22
3    depreciation_rate = 0.05
4    discount_rate = 0.05
```

When all functions for retrieving revenues and costs, and all input for the DCF model has been defined, the analysis itself can take place. The function starts by creating diagonal matrices for depreciation-, tax and discount rates. This was done by using a built in function in NumPy, but it can also be calculated by multiplying the one dimensional matrices with it's identity matrix.

Calculation for diagonal depreciation matrix:

$$DiagonalDepreciation = Depreciation * I \tag{57}$$

Calculation for diagonal tax matrix:

$$DiagnoalTax = Tax * I \tag{58}$$

Calculation for diagonal discount matrix:

$$DiagnoalDiscount = Discount * I \tag{59}$$

The next step is to generate the original cash flow. This cash flow consists of revenues, grant, production-, deployment- and operational costs, code named cf_part1. Calculations below:

$$cf\_part1 = Originalrevenues - Originalcosts \tag{60}$$

After determining the production cost the next step is beginning depreciating. Depreciating is the process of reducing the value of a fixed asset over time (Investopedia 2023a). To do this a simulated array is created, where the value of investment cost is repeated 41 times. To calculate the individual depreciation you need to multiply the repeated investment cost matrix with the diagonal depreciation matrix. Then differentiate the array. This process is handled using matrix algebra:

[8] WACC stands for weighted average cost of capital, and is synonym to discount rate (Investopedia 2023h).

$$depreciations = diff(INV * DiagnoalDepreciation) \tag{61}$$

Based on the depreciation array one can take the sum of this array and subtract it from the investment cost. This value is the residual value of the warehouse or in other words, what you can sell the warehouse for.

$$RV = -InvCost + sum(depreciations) \tag{62}$$

The RV matrix looks like this:

$$RV = \begin{bmatrix} 0 & 0 & ... & RV \end{bmatrix} \tag{63}$$

A new cash flow can now be generated to calculate the cost of taxes. Taking cf_part1 and adding the residual value, and lastly subtracting the depreciation's results in a new matrix which will be the subject for calculating taxes.

Calculations for the new cash flow:

$$cf\_part2 = cf\_part1 + depreciations + residualvalue \tag{64}$$

This cash flow can now be multiplied with the diagonal matrix for tax rates:

$$taxes = cf\_part2 * tax_rates \tag{65}$$

Second to last, one can calculate the final non discounted cash flow:

$$CF = cf\_part2 - taxes + depreciations \tag{66}$$

The final cash flow will now be calculated with the diagonal discount matrix. The output is the discounted cash flow, and it is calculated as following:

$$DCF = CF * DiagnoalDiscount \tag{67}$$

Running the DCF_model function returns both the normal cash flow and the discounted one. These two cash flows will be analysed using financial parameters and plots to see the projects financial performance.

```
    def dcf_model(decided_volume, tax_rate, depreciation_rate,  discount_rate):

        #Deciding the different diagonal rate matrices for the DCF model
        dep_rate = depreciation_rate
        dep_rates =np.array([(1-dep_rate)**t for t in range(41)])
        dep_rates = np.diag(dep_rates)

        tax_rate = np.diag(np.repeat(tax_rate,40))

        disc_rate = discount_rate
        disc_rates = np.array([1/(1+disc_rate)**t for t in range(41)])
        disc_rates = np.diag(disc_rates)

        #Retrieving the relevant costs and -revenues, and creating the CF
        rel_cost = total_cost(best_x,best_y,decided_volume)
        rel_rev = total_revenue(decided_volume, -sum(sum(rel_cost)))
        cf_part1 = rel_rev[0]+ rel_rev[1] + rel_cost[0] +
                                    rel_cost[1] + rel_cost[2]

```

```
        #Creating depreciations and residual value on the project
        repeat_inv = np.repeat(rel_cost[0][0],41)
        individual_depreciations =
                        - np.append(0,np.diff(np.dot(repeat_inv,dep_rates)))
        residual_value_org = -1*rel_cost[0][0] + sum(individual_depreciations)
        residual_value = np.append(np.zeros(40), residual_value_org)
        cf_part2 = cf_part1 + individual_depreciations + residual_value

        #Generating taxes and the final cf and dcf
        taxes = np.dot(tax_rate,cf_part2[1:])
        final_cf = cf_part2 +  -1*np.append(0,taxes) +
        ↪  -1*individual_depreciations
        final_dcf = final_cf@disc_rates

        #Returning the final cf and dcf
        return [final_cf, final_dcf]
```

7.2.2 Key insights: numbers

A simple and useful extensions of the output for the dcf_model is creating arrays with their corresponding aggregated sums. These arrays can help detecting where the project begins earning money and where it exceeds NPV = 0, which is the point where the profits occur.

Accumulating the cash flow produces the profit of the project during it's lifespan, and it's estimated to be approximately 5.8 million USD. An isolated evaluation of the profit results in a positive review of the project from a financial point of view.

Accumulating the discounted cash flow produces the net present value for the project during it's lifespan, is approximately 660'000 USD. This value indicates the project's value (Investopedia 2023f). The NPV is one of the most common evaluation metrics for financial analysis. Considering the NPV is positive, the project, given it's current considerations and values, is financial viable.

By using a built in function in the NumPy financial library one can quickly estimate the internal rate of return (IRR) for the project. The IRR for this project is 6.47%. This indicates the maximum possible discount rate before the net present value turns into zero. (Investopedia 2023c). Meaning this project could have raised the discount rate by 1.47% before the project would have had a negative NPV, and become unprofitable.

```
    cf_results = dcf_model(decided_volume, tax_rate, depreciation_rate,
                                discount_rate)
```

```
    cf =  cf_results[0]
    cscf = np.cumsum(cf_results[0])
    dcf = cf_results[1]
    csdcf = np.cumsum(dcf)

    print('At 5% discount rate you get:')
    print('Profit', sum(cf))
    print('NPV', sum(dcf))
    print('IRR', round(npf.irr(cf)*100,2),'%')
```

```
At 5% discount rate you get:
```

```
Profit 5830394.221194913
NPV 657290.6708122161
IRR 6.47 %
```

7.2.3 Key insights: visualisation

By visualising the cash flow and the discounted cash flow you can quickly get a overview of the projects financial evaluation. The two financial metrics observable from the visualisation are: payback method and NPV. The payback method is used to evaluate when a project becomes profitable, in this case the income exceeds the costs at year 14. Therefore, based on the payback method if the project lasts longer than 14 years it will be profitable, (Investopedia 2023g). The project yields a positive NPV from year 26 and beyond. Additionally the discounted cash flow is concave, this means that the increases are getting smaller for each increase in years. This seems to match up with the fact that the difference in IRR and discount rate is low despite the gap of years between the NPV being larger than 0 and the end of project is 14 years.

```
zeros =np.repeat(0,41)
zero_profit = np.trim_zeros(np.where(cscf-zeros >= 0, cscf,0))
added_value = np.trim_zeros(np.where(csdcf-zeros >= 0, csdcf,0))

zero_profit_x =  41 - len(zero_profit)
added_value_x = 41 - len(added_value)
```

```
fig, ax = plt.subplots(figsize=(10,7))

ax.plot(cscf, color='grey', label='Cashflow')
ax.plot(csdcf, color='green', label='Discounted cashflow')

ax.axhline(0, color='red', label='Zero')
ax.axvline(zero_profit_x, color='grey')
ax.axvline(added_value_x,color='black')

ax.scatter(x=zero_profit_x, y=zero_profit[0], color='grey', s=70,
                label='Profit > 0 || year ='+str(zero_profit_x))
ax.scatter(x=added_value_x, y=added_value[0], color='black', s=70,
                    label='NPV > 0 || year='+str(added_value_x))

plt.title('Financial Analysis, cubic ='+str(decided_volume))
plt.xlabel('Years')
plt.ylabel('USD in millions')
plt.legend()
plt.show()
```

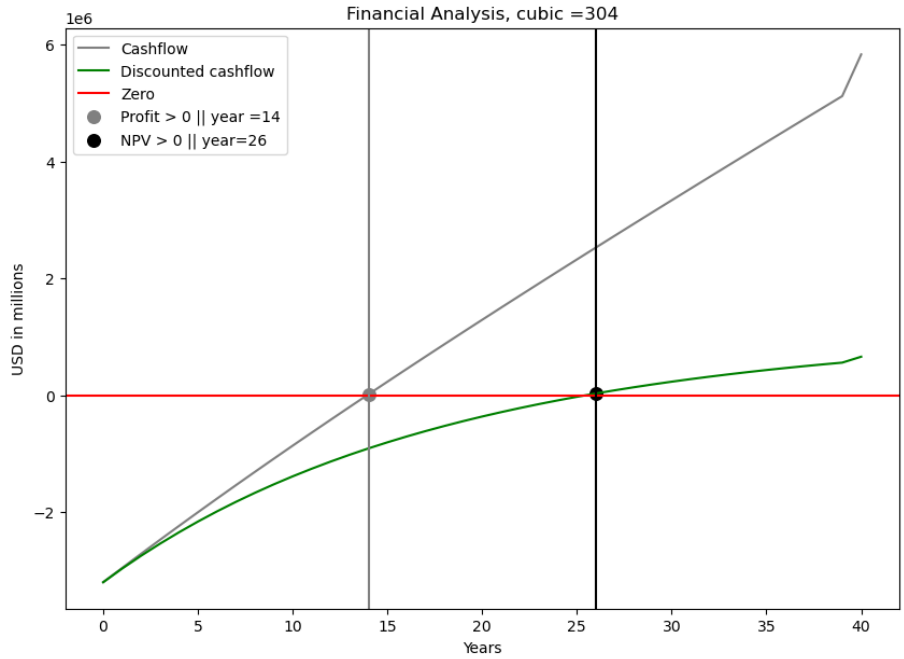

Figure 8: Financial analysis

7.3 Financial conclusion

The results of the financial analysis returns a positive indication about the financial prospect of the project. The project profits after 14 years, it has a positive NPV with some margin and has a IRR almost 1.5% above the discount rate. From a financial perspective the project, as is, seems to be viable and there are no clear financial reasons to not invest in the project. To conclude: it is profitable to invest in the project.

8 Conclusion

This project has had the intention of analysing an investment of an underwater data warehouse from a financial point of view. This has been done through estimating costs and revenues and putting them into a discounted cash flow model.

8.1 Results and conclusions

Having decided the best x and y coordinate for our warehouse and it's volume we estimated the total costs for the project and it's corresponding estimated revenues. Running these numbers through the financial analysis gave us indicators of the projects potential financial consequences. Investing in the project would result in profit and a positive NPV, concluding that the project is financial viable. It is beneficial to invest in the project given that there are no other mutually exclusive and more profitable investment alternatives.

Looking at the project from other aspect besides a financial point of view shows that there are many positive sides by investing in the warehouse. The project is in accordance with several of the UN sustainable development goals, including goals 8, 9, 11 and 13. The project is to be considered a good investment both from financial- and corporate social responsibility perspectives.

8.2 Learning outcomes

Working on this project have given us a lot of knowledge concerning implementation of advanced mathematical methods for solving business problems. During this project we have mostly been focusing on the topics of business. However, as mentioned in subsection 1.4 Important Note, we have also written a exam deliverable with an approach focusing on geography, and the site selection to a much higher degree. Combining these projects has resulted in a greater understanding of how one can turn a geospatial data science projects into a mathematical analysis for investment decisions.

Regarding the applied mathematics course at NTNU, we have learned how to perform and solve advanced mathematical problems. In addition we have learned how to interpret these problems as computational problems solvable by programming.

8.3 Shortcomings, faults and mistakes

As previously mentioned this analysis has a magnitude of flaws, most of them are related to the practical application of the results and the project. We went into this attempting to create an analysis highlighting a new technology in underwater data warehouses and presenting their presence through a business perspective. We soon realised that this was a task far to convoluted to undertake in a single mathematics course. As the premise of this course is to apply maths in a practical scenario and show an understanding for how the subject can be utilised in business cases, we decided the numbers where not important.

8.3.1 Data shortcomings

A financial analysis is built on the preciseness of it's numbers and values, without trustworthy numbers the results are hollow and meaningless. Yet, with only two previous test projects from Microsoft as a research base, there was little data to work with. The research needed to estimate accurate or actual costs surpasses the time we could pour into this project, despite our wish to make it work. Therefore, almost all noteworthy numbers are estimations at best or at worst simply arbitrary in order for the functions or mathematics to make logical sense. In order to salvage the project we angled it towards creating mathematical functions that we could apply to our problem

and that seemed fitting of the financial analysis we where creating. This is of course not how one should approach any noteworthy scientific or economic project, and we are well aware of that. However, given the nature of the course, especially it's emphasis on mathematical application, we made an exception. So to conclude the biggest shortcoming of this project are it's results, as they provide little to no insights in any realistic scenario.

In addition there are a many variables equally or perhaps more important to a project about underwater data warehouses that we excluded. Mostly these are related to the sea's many variables. Elements like, water depth, topography, seabed sediments amongst others where excluded despite their importance, especially when optimising a warehouse position, but also because their existence could entail additional costs. Either for researching an areas applicability or because they force changes to the warehouse construction and position.

8.3.2 Methodological shortcomings

The mathematical methods utilised in this project are in varying degree recognised. The financial analysis used is a globally recognised evaluation method, where there there are few uncertainties about it's connection to reality .

However, there are more uncertainties concerning the methods used for generating costs. The internal part of the production costs are based on a sigmoid function. This means that the total costs will stabilise at a given point. This is not a reasonable assumption, but it is assumed that the costs for this projects interval is somewhat realistic. Furthermore the deployment phase of this project is based on a nonlinear optimisation method. Since the optimisation consists of several optimisation problems, with their own sub problems, the most convenient way of solving the problem was a brute force optimisation algorithm. Brute force optimisation requires a lot of computational power and may lack mathematical optimisation foundation. However, this method was chosen because it was appropriate for solving the problem and it is easy and interpretable.

Lastly the operational cost are based on a trigonometric function. This function is an extreme simplification of the ocean temperatures, and is probably only partial representative for reality. This function is based on several varying factors such as the changes in latitude and longitude, and increasing ocean temperatures factors. These factors are applicable in the chosen geographical area. However, implementing this function with the same input for the variable factors in other places such as Svalbard or increasing the scale to Norway would probably produce completely obscure temperatures. Additionally, there might be other factors concerning the methods and mathematical applications that have been used in this project that might have alternatives which could have produced a better representation of reality.

8.4 Improvements

There are several ways of improving this project, and many of them share the same characteristics as what have been mentioned in section 8.3 Shortcomings, faults and mistakes. These characteristics are mainly focused on the preciseness of the numbers used in our calculations. Given the lack of research and prior projects it is difficult to get good estimations or precise values for the costs and revenues for the project. The second improvement concerns the data used for the optimisation. Our map does not include data on for example nature reserves, coastal traffic, marine infrastructure and sediments. This means that our optimal location might not be suitable if it is conflicting with some of the data mentioned.

Bibliography

Andreas Janisch (2023). *The cost of structural steel per kg in 2023*. URL: https://jactio.com/en/the-cost-of-structural-steel-per-kg/ (visited on 20/11/2023).

Bryn Burns (2021). *Building a Data Warehouse: Storage*. URL: https://chartio.com/learn/data-warehouses/building-data-warehouse-storage/ (visited on 10/11/2023).

DellEMC (2016). *PowerEdge R630 Spec Sheet*. URL: https://i.dell.com/sites/doccontent/shared-content/data-sheets/en/documents/dell-poweredge-r630-spec-sheet.pdf (visited on 10/11/2023).

Finn Halvorsen (2003). *Billigere sjøkabel*. URL: https://www.tu.no/artikler/billigere-sjokabel/240454?fbclid=IwAR14iAEAyKQvjN04DqzRo9EmR33wX2LGnhmMSR-fdbgvqakpRLKKc5QQepE (visited on 17/11/2023).

GeoNorge (2021). *Sigmoid Function - an overview*. URL: https://www.sciencedirect.com/topics/computer-science/sigmoid-function.

— (2023a). *Administrative enheter kommuner [Dataset]*. URL: https://www.datasenterindustrien.no/connectivity (visited on 20/11/2023).

— (2023b). *Norske fylker og kommuner illustrasjonsdata 2020 (klippet etter kyst) [Dataset]*. URL: https://kartkatalog.geonorge.no/metadata/norske-fylker-og-kommuner-illustrasjonsdata-2020-klippet-etter-kyst/7408853f-eb7d-48dd-bb6c-80c7e80f7392 (visited on 04/10/2023).

Innovasjon Norge (2023). *Tilskudd til innovasjonskontrakter*. URL: https://www.innovasjonnorge.no/tjeneste/tilskudd-til-innovasjonskontrakter (visited on 20/11/2023).

Investopedia (2022). *Externality: What It Means in Economics, With Positive and Negative Examples*. URL: https://www.investopedia.com/terms/e/externality.asp (visited on 13/12/2022).

— (2023a). *Depreciation: Definition and Types, With Calculation Examples*. URL: https://www.investopedia.com/terms/d/depreciation.asp (visited on 20/11/2023).

— (2023b). *Discounted Cash Flow (DCF) Explained With Formula and Examples*. URL: https://www.investopedia.com/terms/d/dcf.asp (visited on 20/11/2023).

— (2023c). *Internal Rate of Return (IRR) Rule: Definition and Example*. URL: https://www.investopedia.com/terms/i/irr.asp (visited on 20/11/2023).

— (2023d). *Introduction to Supply and Demand*. URL: https://www.investopedia.com/articles/economics/11/intro-supply-demand.asp (visited on 20/11/2023).

— (2023e). *Marginal Cost Meaning, Formula, and Examples*. URL: https://www.investopedia.com/terms/m/marginalcostofproduction.asp (visited on 20/11/2023).

— (2023f). *Net Present Value (NPV): What It Means and Steps to Calculate It*. URL: https://www.investopedia.com/terms/n/npv.asp (visited on 20/11/2023).

— (2023g). *Payback Period Explained, With the Formula and How to Calculate It*. URL: https://www.investopedia.com/terms/p/paybackperiod.asp (visited on 20/11/2023).

— (2023h). *Weighted Average Cost of Capital (WACC): Definition and Formula*. URL: https://www.investopedia.com/terms/w/wacc.asp (visited on 20/11/2023).

Johannes Mauritzen (2023). *Basics of Matrix and Vector Algebra*. URL: https://jmaurit.github.io/math2/matrixAlgebra1.html (visited on 20/11/2023).

John Roach (2020). *Microsoft finds underwater datacenters are reliable, practical and use energy sustainably*. URL: news.microsoft.com/source/features/sustainability/project-natick-underwater-datacenter/ (visited on 26/11/2023).

Kystinfo (2023). *Kystinfo [Dataset]*. URL: https://a3.kystverket.no/kystinfo (visited on 04/10/2023).

Mary Zhang (2022). *Underwater Data Centers: Servers Beneath the Surface*. URL: https://dgtlinfra.com/underwater-data-centers-servers/ (visited on 10/11/2023).

NOAA (2018). *How does the temperature of ocean water vary?* URL: https://oceanexplorer.noaa.gov/facts/temp-vary.htm (visited on 18/11/2023).

Norges vassdrags- og energidirektorat (2023). *Nedlastning av fagdata fra NVE [Dataset]*. URL: http://nedlasting.nve.no/gis/ (visited on 04/10/2023).

OpenAI (2023). *[LLM]*. Version ChatGPT 3.5 version. URL: https://chat.openai.com/ (visited on 28/10/2023).

Regjeringen.no (2023). *Skattesatser 2023*. URL: https://www.regjeringen.no/no/tema/okonomi-og-budsjett/skatter-og-avgifter/skattesatser-2023/id2929581/ (visited on 20/11/2023).

sea temprature.net (2023). *Haugesund water temprature*. URL: https://no.seatemperature.net/current/norway/haugesund-rogaland-norway (visited on 18/11/2023).

sea temprature.org (2023). *Stromness Sea Temperature*. URL: https://www.seatemperature.org/europe/united-kingdom/stromness.htm (visited on 18/11/2023).

Skatteetaten (2023). *Avskrivningssatser*. URL: https://www.skatteetaten.no/satser/avskrivningssatser/ (visited on 20/11/2023).

United Nations (2015). *THE 17 GOALS*. URL: https://sdgs.un.org/goals (visited on 20/11/2023).

Wylie Wong (2023). *Hot in Here: Is Raising Temperatures in Data Centers Good for Hardware?* URL: https://www.datacenterknowledge.com/equinix/hot-here-raising-temperatures-data-centers-good-hardware (visited on 26/11/2023).

Zhang, X. et al. (2017). 'Cooling Energy Consumption Investigation of Data Center IT Room with Vertical Placed Server'. In: *Energy Procedia* 105. 8th International Conference on Applied Energy, ICAE2016, 8-11 October 2016, Beijing, China, pp. 2047–2052. ISSN: 1876-6102. DOI: https://doi.org/10.1016/j.egypro.2017.03.581. URL: https://www.sciencedirect.com/science/article/pii/S1876610217306331.