Rodrigo Prodohl

Wello Wave App

Mapping out a databse of the world wave energy resources

Helsinki Metropolia University of Applied Sciences

Degree: Bachelors Degree

Degree Programme: Environmental Engineering

Thesis

Date

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Rodrigo Prodohl Wello Wave App 31 pages + 27 appendices 22 March 2018
Degree	Environmental Engineering
Degree Programme	Environmental Engineering
Specialisation option	Water, waste and Environmental Engineering
Instructor(s)	Veli-Matti Taavitsainen

In this thesis, the creation of a Python based interactive map of the world's wave energy resources is explained and presented. The application will be used by Wello Oy to generate values needed in wave energy production in various locations across the world, and to calculate the theoretical power of Wello's Penguin wave energy converter.

The application is planned to be used as an easy-to-use tool to quickly and effectively measure the power output of the Penguin wave energy converter at any given site in the world. The application allows tailored results that are calculated by using values obtained from each potential wave energy site. The results show the end user the wave energy at the site, and the optimal dimensions of the wave energy converter for maximal power production and maximal return of investment.

The fundamentals of wave energy and wave energy converters are discussed, as well as the methodology for calculating the theoretical power output of waves and calculating the levelized cost of energy at each of these sites. All this is done in hope of making the process of finding potential wave energy projects more efficient and effective, and thus providing a clean, renewable and continuous energy source to the world.

Keywords	wave, energy, converter, power, wello, penguin, renewable, ocean

Contents

Symbols and units

g	acceleration due to gravity, 9.81 (m/s²)
ρ	density: Freshwater = 1000 kg/m³
	Seawater = 1027 kg/m³
α	energy scale parameter for Pierson–Moskowitz spectra, 0.0081
	peak enhancement factor for JONSWAP spectra, mean = 3.3, range [1:7]
λ	Froude scaling parameter, length ratio of model to full size prototype
θm	mean direction of wave propagation
S(f)	spectral variance density (m2 /Hz)
Δf	frequency division/step
m_n	spectral moment, $m_n = \int S(f)\, f^n \Delta f$
a	amplitude of sinusoidal wave
T	period of a wave, time taken for a particle to return to its original position
f	frequency, inverse of period, T (Hz)
f_p	the frequency at which the variance of the spectrum is at its maximum
H_s	significant wave height
$H_{\frac{1}{3}}$	average of the highest third of waves in a time series
H_{m0}	significant wave height derived from spectral moments, $4\sqrt{m_0}$
T_p	peak period, inverse of the frequency of the maximum of $S(f)$, f_p
T_e	energy period, (m_{-1}/m_0)
P	energy flux per metre wave crest (kW/m)
h	water depth
k	wave number
kW	1,000 watts, measurement of power
MW	1,000,000 Watts

1 Introduction

According to a UN study, 40% of the world's population live less than 100 km from the coast, and this value is estimated to double in size by 2025 (Percentage of total population living in coastal areas, 2007). Greater population density and economic activates in these coastal regions put added strain on the already dwindling energy supplies. Ocean energy provides an excellent emergent solution to this issue; clean, continuous and localized energy at local deployment sites offering direct energy to those who live near the coastlines and further afield.

According to the International energy agency, it is estimated that there is the potential to develop 20,000 – 80,000 terawatt-hours (TWh) of electricity in the form of ocean energy per year. These ocean energies come in the forms of tidal energy, marine current power, osmotic power, ocean thermal energy and wave energy. Out of these sources, wave energy poses the largest theoretical energy production, with an estimated 8,000-80,000 TWh/year of production. Wave energy converters (WEC) are located along three different ocean environments: onshore, nearshore and offshore. They are briefly discussed below:

- Onshore devices are integrated into the coastline, along natural rock faces or man-made structures. They are close to the utility network and easy to maintain. The tradeoff for this that captured energy is minimized due to the act of friction with the seabed.
- Near-Shore devices are typically located in water shallow enough for them to be fixed to the seabed, providing a stable stationary base which the oscillating device can produce. Disadvantages are akin to Onshore devices.
- Offshore devices are located further from the coastline in deep water, fixed to the sea bed using moorings mass. They have the highest potential energy resource.

This thesis focuses on offshore wave energy converters, specifically in relation to a rotating mass converter. The thesis is being conducted in conjunction with Finnish company clean-tech Wello Oy and is focusing on their Penguin wave energy converter (henceforth called 'The Penguin').

Presently, finding the potential for a WEC at a specific site is a long and taxing task. Wave resource data such as wave heights, wave periods need to be collected to gauge energy in the waves at a certain site along the world's coastlines. Once the data has been procured it is processed, filtered and put through models to give the site-specific wave resources as well as theoretical power output of the Penguin. This process is repeated every time a new site is looked for and currently is only done by very few employees within Wello Oy.

The aim of this thesis was to create a computer-based application which maps out the world's wave energy resources. The application will be based around the concept of an interactive map of the world, allowing the user to select a point along the world's coastlines and instantly receive the wave resources, energy production capabilities and the theoretical power output of The Penguin.

2 Theory

2.1 Penguin

This thesis focuses on Wello Oy's Penguin wave energy converter (WEC). The Penguin is categorized as a rotating mass WEC. A large rotating mass is fixed on the inside of the Penguin and rotates around a central shaft (Figure 1). Power is led from the rotator to a 600-kW generator using the same shaft eliminating conversion losses. The Penguin 'rolls' in the water converting the movement of the waves to gyration, which is turn amplified. The asymmetrical shape of the hull promotes this rolling motion of the device across the waves, pushing the rotating mass and in turn producing clean energy.

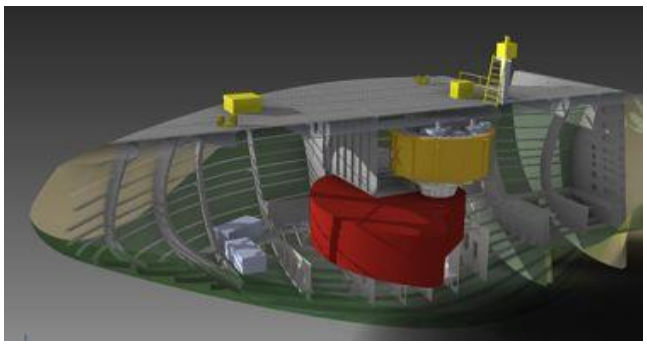


Figure 1: Cross section of the Penguin wave energy converter

The Penguin utilizes the same components that are already used in wind turbines, meaning that the Penguin is cost competitive compared to offshore wind energy. The Penguin brings opportunities for local and regional employment, as manufacturing of the Penguin can be done in most shipyards. Deployment of device is cost effective and can be achieved with small vessels available locally. The operations and maintenance is easy; it is similar to maintaining any floating device. Remote connection allows for continuous monitoring and adjustment of the device. Wello has deployed their first Penguin device in 2012 in Orkney, Scotland, figures of the device can be found in Table 1 below.

Table 1: Penguin in numbers

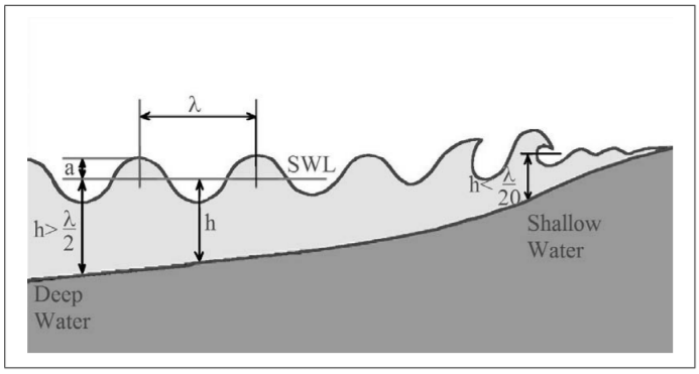
Parameter	Penguin
Mass	2100 Tons
Length	43 m
Width	22 m
Draft	6.82
Generator Size	600 kW

Figure 2: Penguin in Orkney

2.2 Wave theory

To understand and access the potential wave energy resources for different sites, a knowledge of wave theory is required. This section covers the basic principles of monochromatic waves and the types of ocean spectra used to gauge the power output for these sites.

Monochromatic waves or regular single frequency waves provide an indication of theoretical power of a WEC and how it operates. Regular waves are defined by their wave periods (wavelength), amplitude (height) and surface profile. The shape of an ocean wave is often depicted as sinusoidal waves (waves or curves which represent periodic oscillations of constant amplitude following the sine function). Figure 3 shows this relationship.



SWL = Still water line

h = water depth

λ = wavelength

a = amplitude of a sinusoidal wave

Figure 3: Linear wave theory definitions (EMEC, 2009)

Table 2 provides some physical definitions for sinusoidal waves based on linear theory.

	Water Depth		
Parameter	**Deep water:** $h > \frac{\lambda}{2}$	**Intermediate:** $\frac{\lambda}{2} > h < \frac{\lambda}{20}$	**Shallow:** $h < \frac{\lambda}{20}$
Wavelength, λ	$\frac{gT^2}{2\pi}$	$\frac{gT^2}{2\pi}\tanh(kh)$	$T\sqrt{gh}$
Wave celerity (speed), c	$\frac{gT}{2\pi}$	$\frac{gT}{2\pi}\tanh(kh)$	$\sqrt{gh}$
Wave velocity, c_g	$\frac{c}{2}$	$\frac{c}{2}\left(1+\frac{2kh}{sinh2kh}\right)$	c

Wave energy/unit area	Wave power/unit length
$E = \frac{1}{2}\rho g a^2 \left(\frac{J}{m^2}\right)$	$P = Ec_g\ (kW/m)$

Table 2: Linear theory definitions (EMEC, 2009)

2.2.1 Ocean wave spectra

Ocean waves are formed as a result of wind acting over the surface of the water, the strength, length and frequency of the wind will all affect the type of waves generated. These waves have irregular wave heights and periods, caused by the irregular nature of the wind. This makes it difficult to describe the sea surface, with a purely deterministic approach. However, properties like average wave height, wave periods and directions vary slowly in time and space compared to these parameters at a specific time. Using spectral density analysis, one can find the statistical average of a wave over a specific time series. The spectral density of a wave time series is the distribution of power into the frequency components composing the signal. We apply the knowledge that the surface elevation of waves, at any given time, can be seen as the sum of a large number of harmonic waves (i.e. waves with a frequency that is the multiple of the frequency of the original waves), which have been generated by wind in different places and times. We decompose these harmonic waves into a spectrum of frequencies over a range, i.e. a signal. This allows for the statistical average the signal in terms of its frequency content to be analyzed, which is its spectrum. Typical wave spectrums are produced from recording the waves over a period of 30 minutes, albeit other time series are also used.

There are a few different spectral equations which describe the frequency composition of an irregular water surface elevation time history. The spectra in relation to the ocean surface depict the relationship between the frequencies of an ocean spectra and specific wave conditions, such as: peak period, peak energy period, significant wave height etc.

There are two principle spectral forms which are commonly used to describe waves: Pierson-Moskowitz spectrum (PM) and Joint North Sea Wave Project spectra (JONSWAP). Both the spectrums are related as the JONSWAP spectra is an evolved version of the Pierson-Moskowitz spectrum and a is can be used to describe both. Figure 4 demonstrated the typical shape of both these spectra.

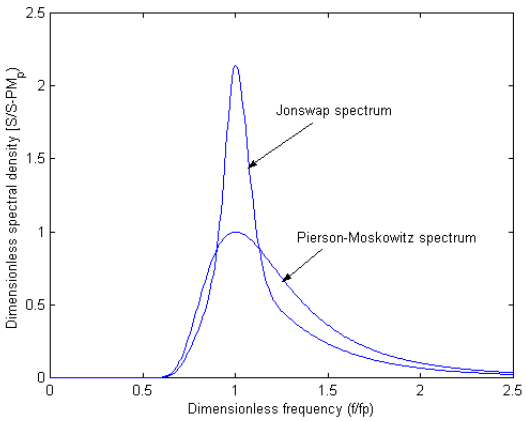


Figure 4: Comparison of spectral shapes (NTNU, 2008)

The Pierson-Moskowitz spectrum is an equation defining the one-dimensional spectral shape over a frequency range, based on the assumption that the sea is fully developed, i.e. that the waves have come into equilibrium with the sea. The formula for this is:

$$S(f)_{PM} = \frac{\alpha g^2}{(2\pi)^4}\frac{1}{f^5} exp\left[-\frac{5}{4}\left(\frac{f_p}{f}\right)^4\right], \quad (1)$$

In the equation 1 f is the frequency, f_p is the frequency period and $S(f)_{PM}$ is the Pierson-Moskowitz spectrum (EMEC,2009).

The limitation of using this spectrum is that it only considers fully developed conditions, waves which have the maximum size theoretically possible for wind of a specific strength and duration. The equation does not facilitate fetch limited conditions, i.e. when wave height is limited by the area where the waves are forming. However, Hasselmann et al. (1973), after analyzing the code during the JONSWAP project, inferred that the wave spectrum is never fully developed. Rather, it continues to develop through non-linear wave-to-wave interactions over long distances and times. An extra peak enhancement factor (γ) was added to the Pierson-Moskowitz spectrum to improve the fit for various sea states.

$$S(f)_J = \left(\frac{\alpha g^2}{(2\pi)^4}\frac{1}{f^5} exp\left[-\frac{5}{4}\left(\frac{f_p}{f}\right)^4\right]\right)\left(\gamma^{exp\left[-\left(\frac{(f-f_p)^2}{2\sigma^2 f_p^2}\right)\right]}\right) \quad (2)$$

One could then infer that the JONSWAP could be used as the base formula to accommodate all the different sea states. Therefore, JONSWAP is the spectrum which was chosen as this could be used to describe varying sea states along the world's coastlines.

2.2.2 Significant wave height, peak period and Power

The most common type of wave data lies in the form of significant wave height and peak period, both of which are either computed by wave models or outputted by buoys relaying the wave data. This proves to be the most accurate measurement to generate models for wave power for a specific site. The data used for this study follows the same conventions.

The significant wave height (Hs) of a wave is defined as the mean height of the highest third of the waves in a given time series. It can also be defined as four times the square root of the zeroth-order moment of the wave spectrum (JONSWAP Equation 2).

The peak period (Tp), is the wave period with the highest energy. The analysis of the distribution of the wave energy as a function of wave frequency (period-1) for a time-series of individual waves is referred to as a spectral analysis. These wave periods (frequencies) follow the JONSWAP spectrum (1), described above.

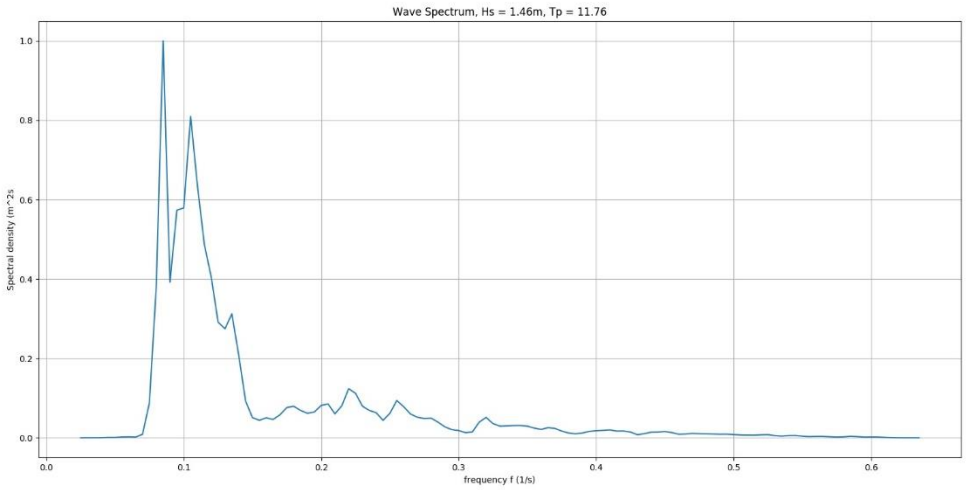


Figure 5: Wave spectra for 3 min bin time series analysis

Figure 5 depicts a typical wave spectrum for a given time series. This type of graph is the typical output for wave buoys, giving the spectral density of a wave for a given time series over frequency. One can apply the principles of the JONSWAP spectrum to find relevant information concerning the wave during a certain period. The peak period is calculated from this spectrum by taking the inverse of the frequency at the maximum spectral density:

$$T_p = \frac{1}{\max(S(f))} \tag{3}$$

For the case in *Figure 2* the formula to calculate T_p is:

$$\frac{1}{\max(S(f))} = \frac{1}{0.085} = 11.6\ s$$

The power for monochromatic waves is given as the following (EMEC, 2009), H being average wave height and T being average wave period:

$$Power = \frac{\rho g^2 H^2 T}{32\pi}\ (W/m) \tag{4}$$

A similar approach and formula that is used to define regular waves can be implemented to obtain the wave power in real seas. From the wave spectrum one is able to derive a single value for peak period and significant wave height, enabling wave power to be calculated for this spectrum. The wave power/unit length (in deep water) is defined as follows (EMEC,2009):

$$Power = \frac{\rho g^2 H_s^2 T_e}{64\pi}\ (W/m) \tag{5}$$

In this formula, time is given as T_e or rather as the wave energy period and height is given as significant wave height H_s, rather than average wave height as denoted in Equation 4.

$$T_e = \frac{m_{-1}}{m_0} \tag{6}$$

T_e as given by EMEC (2009) is the variance-weighted mean period of the density spectrum, m_n are the spectral moments of the frequency spectrum. The spectral moments are defined as the area under the spectral curve, the positive moments give weight to higher frequency values. The negative moments give the opposite, giving weight to the lower frequencies.

As stated previously, each wave spectrum describes the waves over a set period; in the case of Figure 2, the 3-minute spectrum shows a significant wave height of 1.46 meters and peak period as 11.76 seconds. However, to calculate power, we need to convert T_p to T_e for each spectrum. In order to do this we need to know the peak enhancement factor, γ (JONSWAP eq.1). The peak enhancement factor is related to each specific stationary bin, allowing for power values across multiple sea states to be calculated. DNV-GL (2008) noted that the peak enhancement factor may be calculated by the following equations:

$$\gamma = 5 \; for \; T_p/\sqrt{H_s} \leq 3.6 \tag{7.1}$$

$$\gamma = \exp(5.75 - 1.15T_p/\sqrt{H_s}) \;\; for \; 3.6 \; < T_p/\sqrt{H_s} < 5 \tag{7.2}$$

$$\gamma = 1 \; for \; 5 \leq T_p/\sqrt{H_s} \tag{7.3}$$

The relation between T_p and T_e is given as follows:

$$\frac{T_e}{T_p} = \frac{4.2 + \gamma}{5 + \gamma}$$

For WEC's the amount of energy flux available to the device can be quantified in several ways. The most appropriate approach for gauging the power of The Penguin, is the relationship between the capture width of the device and wave power. Capture width is defined as the power available to drive the wave energy converter derived from the power of the device and the wave energy flux (EMEC, 2009).

$$C_w = \frac{Device \; power \; (W)}{Wave \; power/unit \; length \; (W/m))} (m) \tag{8}$$

Through wave tank testing of the WEC, a function has been created to calculate the capture width of the device given a certain sea state. If the significant wave height and peak period of a site is known one can calculate wave power and a result can be gathered for the Device power.

As is the case for most WECs, the device scale is dependent on the site. Thus, it is necessary that the calculations for power of the device considers the optimized scale for each site. This is dependent on the power in the site, the type of wave resources in the area and the cost or energy for that site. WECs scale in relation to Froude's scaling. The scaling parameters are given in Table 3 where F_n is the scale in question.

Table 3: Froude's scaling Factors (EMEC,2009)

Characteristic	**Froude scaling**
Length	F_n
Area	F_n^2
Volume	F_n^3
Mass	F_n^3
Force	F_n^3
Power	$F_n^{3.5}$
Time	$\sqrt{F_n}$
Capture width	F_n

The scaling factors are used in relation to the wave tank testing data of the current Penguin prototype. During testing the WEC is subjected to very specific wave conditions. These tests give us specific figures on how the WEC performs at given conditions and how this affects the device. Specifically, how it moves along the x and z axis i.e. pitch and roll of the device. The values are scaled using time and height scaling factors as given above. A capture width model is made from the results of the tank testing. This model is omitted from the thesis as it is confidential. From these relationships, given spectral data of peak period and significant wave heights, T_e the power of a site can be calculated. Once the power of the site is known, we can rearrange the capture width model to solve for power of the device.

2.3 Levelized cost of energy

There are various ways of generating electricity and these incur different costs. These costs can come from the initial investment into the energy source, the operation, its fuel and its maintenance.

Levelized cost of energy (LCOE) is a measure of a power source which attempts to give a comparable value for different methods of electricity generation of unequal life spans, project size, different capital costs etc. LCOE is an economic assessment of the average total cost produce and maintain a generating device/method during its life span divided by the total energy output of the device/method over its lifetime. The formula for LCOE is as follows, where I_t is the investment during the year t, M_t is the maintenance in the year t, F_t is the fuel expenditures in the year t, E_t is the electrical energy generated in the year t, r is the discount rate and n is the expected lifetime of the device:

$$LCOE = \frac{\sum_{t=1}^{n} \frac{I_t + M_t + F_t}{(1+r)^t}}{\sum_{t=1}^{n} \frac{E_t}{(1+r)^t}} \quad (9)$$

In the case of the Penguin, F_t was omitted at the device does not use any fuel to run. The energy required to run the device is utilised for the generation of electricity from the waves.

3 Methodology

3.1 Data Collection

The data used for this thesis has been provided by the BMT Argoss group, specifically from their waveclimate.com website. The data is given as a 2D scatter table (*Figure 6*) which shows peak period and significant wave height for a given site as a percentage of their joint occurrence throughout the period of collected data (typically 30 years). The size of the site which can be selected range from grids of 50x50km, 100x100km, 200x200km and 500x500km, for this thesis, a grid size of 100x100 km was chosen (*Figure 3)*. This is due to time constraints on the project itself, a 50x50km grid would have proven more accurate when mapping out the wave resources but would have significantly increased work time.

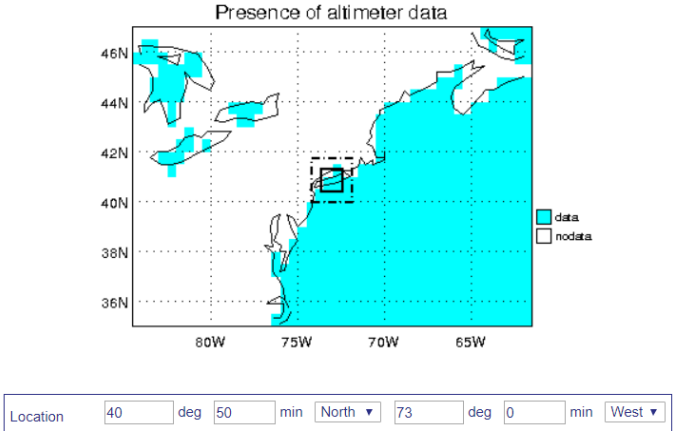


Figure 6: Grid view of waveclimate.com

coordinate points 40.5 N, -73 E

The data had to be hand collected due to the format of the waveclimate.com site and thus was by far the lengthiest process of the project. It should also be mentioned that the data for Antarctica was omitted as the cold/harsh conditions would not favor WECs. An example of a data set provided by BMT argoss is shown below. The data is from the east coast of New York, USA:

			Tp (s)																
	lower	uppe	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
	lower	uppe		3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	total
Hs (m)	0	0.2	0.29	0.115	0.011	0.034	0.108	0.12	0.208	0.334	0.5	0.19	0.06	0.02	0.01	0	0	0.01	0		2.005
	0.2	0.4	1.19	1.105	0.163	0.386	0.91	0.94	1.792	1.932	1.34	0.41	0.19	0.09	0.02	0	0.01	0	0		10.55
	0.4	0.6	1.18	2.879	0.713	1.894	2.636	1.87	2.565	2.32	1.33	0.47	0.21	0.11	0.04	0	0.02	0	0		18.23
	0.6	0.8	0.24	4.217	1.562	2.109	3.788	1.8	1.837	1.677	1.02	0.36	0.17	0.06	0.04	0	0	0	0		18.88
	0.8	1	0.01	2.814	2.575	1.6	3.148	1.48	1.354	1.185	0.6	0.24	0.1	0.03	0.03	0	0.01	0	0		15.17
	1	1.2	0	0.754	3.158	1.388	1.901	1.18	1.07	0.847	0.5	0.16	0.09	0.05	0.02	0	0.01	0	0		11.13
	1.2	1.4	0	0.094	2.556	1.289	1.205	0.91	0.676	0.552	0.26	0.13	0.07	0.06	0.02	0	0.01	0	0		7.82
	1.4	1.6	0	0.016	1.184	1.373	0.861	0.57	0.554	0.374	0.23	0.08	0.06	0.03	0.02	0	0.01	0	0		5.366
	1.6	1.8	0	0	0.318	1.237	0.68	0.37	0.412	0.296	0.14	0.04	0.05	0.04	0.03	0	0.01	0	0		3.612
	1.8	2	0	0	0.057	0.775	0.486	0.25	0.275	0.222	0.11	0.03	0.04	0.05	0.02	0	0.01	0	0		2.322
	2	2.2	0	0	0.005	0.431	0.423	0.16	0.209	0.175	0.09	0.03	0.01	0.03	0.02	0	0	0	0		1.582
	2.2	2.4	0	0	0	0.198	0.315	0.14	0.115	0.104	0.04	0.02	0.01	0.01	0.01	0	0	0	0		0.962
	2.4	2.6	0	0	0	0.092	0.204	0.11	0.093	0.107	0.05	0.02	0.01	0.01	0	0	0	0	0		0.697
	2.6	2.8	0	0	0	0.034	0.156	0.07	0.066	0.064	0.04	0.03	0.01	0	0	0	0	0	0		0.478
	2.8	3	0	0	0	0.003	0.083	0.08	0.062	0.037	0.04	0.02	0	0	0	0	0	0	0		0.323
	3	3.2	0	0	0	0.001	0.064	0.05	0.042	0.041	0.03	0.03	0	0	0	0	0	0	0		0.266
	3.2	3.4	0	0	0	0	0.038	0.03	0.045	0.038	0.03	0.02	0	0	0	0	0	0	0		0.197
	3.4	3.6	0	0	0	0	0.015	0.03	0.034	0.023	0.02	0.01	0	0	0	0	0	0	0		0.133
	3.6	3.8	0	0	0	0	0.005	0.02	0.021	0.014	0.02	0	0	0	0	0	0	0	0		0.088
	3.8	4	0	0	0	0	0.003	0	0.007	0.008	0.01	0	0	0	0	0	0	0	0		0.038
	4	4.2	0	0	0	0	0.003	0.01	0.012	0.007	0.01	0.01	0	0	0	0	0	0	0		0.048
	4.2	4.4	0	0	0	0	0.001	0.01	0.01	0.01	0.01	0.01	0	0	0	0	0	0	0		0.045
	4.4	4.6	0	0	0	0	0	0	0.007	0.004	0.01	0	0	0	0	0	0	0	0		0.025
	4.6	4.8	0	0	0	0	0	0	0.001	0.003	0	0	0	0	0	0	0	0	0		0.007
	4.8	5	0	0	0	0	0	0	0.001	0	0	0	0	0	0	0	0	0	0		0.007
	5	5.2	0	0	0	0	0	0	0	0.001	0	0	0	0	0	0	0	0	0		0.005
	5.2	5.4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0.001
	5.4	5.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0.003
	5.6	5.8	0	0	0	0	0	0	0.001	0	0	0	0	0	0	0	0	0	0		0.003
	5.8	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0
	6	6.2	0	0	0	0	0	0	0	0.001	0	0	0	0	0	0	0	0	0		0.003
	6.2	6.4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0
	6.4	6.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0
	6.6	6.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0.001
	6.8	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0.003
	7	7.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0.001
	7.2	7.4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0.001
	7.4	7.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0
	7.6	7.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0
	7.8	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0.001
	8	8.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0
		total	2.91	12	12.3	12.85	17.04	10.2	11.47	10.44	6.42	2.31	1.11	0.61	0.27	0	0.09	0.01	0		100

Figure 7: Wave data, coordinate points 40.5 N, -73 E

The data gives Hs and Tp as a percentage of occurrences over a 30-year period. The original data consists of more than 70000 data points taken at 3-hourly intervals. The data was exported to a text file (.txt) and saved in a folder containing the rest of the wave data. The reasoning behind this is that it was the quickest way to save the data and the fastest method for the program to read the data.

3.2 Python:

The Thesis utilizes Python programming language for all the calculations and backend of the GUI. There are other powerful programming languages and mathematical tools which this application could have been built on such as MATLAB, R and octave. MATLAB could have been used to form the core basis of this project, it's also an extremely powerful mathematical programming language and focuses on readability. However, the commercial software license for MATLAB is

relatively expensive and costs of just the platform would have exceeded 6000 euros to include all of the necessary packages to create a GUI. Octave and R are other open source software's which could have formed the backend of this project. However, Python was chosen due to the level of fluency the author had with the language.

Python is a widely used programming language, created by Guido van Rossum and released in 1991. Python emphasizes code readability, using whitespace indentation to delimit code blocks. It also uses syntax to express concepts in fewer lines of code in comparison to other languages such as C++.

Python is heavily supported by multiple actors, supporting object-orientated, imperative and functional programming. A large and comprehensive library with a wide range of methods make python ideal for an array of computing scenarios. This thesis utilized a number of different libraries or packages, the most important of which are mentioned below. All the packages have been used under the correct licensing agreements.

3.3 Python packages

3.3.1 NumPy

NumPy is a strong package for scientific computing within Python. It contains a powerful N-dimensional array manipulating package, ideal for use in this thesis as the main core of the program deals with many 2 and 3 dimensional arrays (NumPy Developers, 2005).

3.3.2 Matplotlib

Matplotlib is a 2D and 3D plotting library which production quality figures in a plethora of formats and interactive environments across multiple platforms. One can generate plots, histograms, power spectra and scatterplots with a few lines of code. It also supports a multitude of different styles allowing for clear data presentation, for example ggplot (a popular plotting package for R). (Hunter et al., 2007)

3.3.3 Basemap

The matplotlib basemap toolkit is a library for plotting 2D data on maps in Python. It is similar in functionality to the matlab mapping toolbox, the IDL mapping facilities, GrADS, or the Generic Mapping Tools. PyNGL and CDAT are other libraries that provide similar capabilities in Python. (Jeffrey Whitaker.,2016)

3.3.4 Tkinter

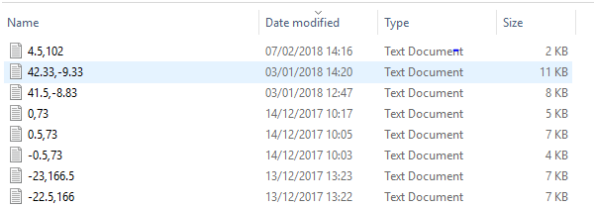
Tkinter is one of Python's GUI (Graphical user Interface) packages. One of the reasons for choosing this package over other similar packages is the solid integration it has with matplotlib packages. Coupled with the fact that it can be used on multiple platforms makes it ideal for this project.

3.4 Wave App Creation

The main objective of the thesis was to create a user-friendly applications (henceforth called wave app) for calling wave data from a map and in turn a database. This section of the thesis will focus on functions and methods to obtain the wave data, the data manipulation and the creation of the user interface. Before the user interface of the wave app was created the backend for manipulating data and conducting the necessary calculations was built. This would allow for easier implementation to the GUI instead of attempting to create it once the backend of the GUI was ready. A more detailed explanation of the backend of the wave app and the full code can be found in the Appendix.

3.4.1 Data storage method

The wave data for each site was stored in a directory containing all the data sets for each site with the convention of the file name being the site's coordinate points, for example, the data set in Figure 6 was saved as '40.5, -73.txt' (figure 7). The program reads the directory and stores the directory contents as an array of latitudes and longitudes with the name coord_list (See Appendix 1).



Name	Date modified	Type	Size
4.5,102	07/02/2018 14:16	Text Document	2 KB
42.33,-9.33	03/01/2018 14:20	Text Document	11 KB
41.5,-8.83	03/01/2018 12:47	Text Document	8 KB
0,73	14/12/2017 10:17	Text Document	5 KB
0.5,73	14/12/2017 10:05	Text Document	7 KB
-0.5,73	14/12/2017 10:03	Text Document	4 KB
-23,166.5	13/12/2017 13:23	Text Document	7 KB
-22.5,166	13/12/2017 13:22	Text Document	7 KB

Figure 8: Example of wave data folder

3.4.2 Data retrieval

To extract the useful information from each sites data set, the function getdata(lat,lon) was created. Once the user has selected their desired data, it is passed through this function. The function takes two arguments, the latitude (lat) and the longitude (lon) of the users selected data. The users latitude and longitude input data is compared to the coordinate list array with the function spatial.KDTree(), this function finds the closest matching coordinate points in the coordinate list array to the users defined data. For example, for the case above if the user gave the coordinate values of 40.456, -72.97 it would open the data in 40.5, -73.txt as these are the closest matching points to the user's data. After the getdata() function has selected the correct data set it reads the content of the text file and converts it to an array. Once this is achieved the data is manipulated and sliced to extract the Tp, Hs and percentage of occurrence matrix from the file. Knowing both the Tp and Hs, Te can be calculated by iterating over all Hs and Tp bins (equation 7.1 – 7.3). Finally, the function returns Tp, Hs, Te, the occurrence matrix and the file handle of the chosen data set.

3.4.3 Scaling factors and Penguin model

A simple function was then created to equate to Froude's scaling parameters (scale(mu)), mu in this case is the user defined scale. The function scales mu accordingly and returns the scaled parameters (see scaling table).

The code contains a function P3_8_CWfit(Tp,Hm,s), this is the model created the Penguin: model 3.8 and returns the capture width of the device once Tp, Hs and s (scale) are passed through it. This section of the code remains confidential but is not necessary to explain for this thesis.

3.4.4 Power Calculations

The main function to calculate wave power, theoretical power of the device and other variables is in the function: PowerGeneration(Tpi,Hsi,Tei,TpHs_array). The function allows for the global variable SCL which is the chosen scale of the device. In python a global variable can be accessed from any function or class within the script meaning it does not have to be passed in the original function. This seemed the simplest method to later allow the user to select the scale of the device at a defined site. The site data is passed through the model for the Penguin giving the variable CW_Data (CW_Data = P3_8_CWfit(Tpi,Hsi,SCL)), which is the capture width of the device. The power stored in the waves is then calculated using the same equation as shown above (Equation 5), the equation in the script appears as: rho*(g**2)*(Tei)*(Hsi**2))/(64*pi), with rho being the specific density of water and g the coefficient for gravity. The formula calculates the power in the waves over all the Hs and Tp bins. The TpHs_array (percentage of occurrence of each wave condition matrix) is converted to hours per year and multiplied by the power in the waves, giving energy in the waves at each bin in kilowatt hours per year (kWh/year). The average power in all the waves at the specified site is calculated by summing the energy in the waves at each bin and dividing by the total hours per year to give a value of 'kW/m'.

Using the value for capture width of the device and the power in the waves we can generate the power in the device by manipulating the formula for capture width (equation 8), giving the desired amount of energy produced at the site per year at a given scale of the device. The PowerGeneration() function also finds out the device power over every scale and it does this at the end of the code section by looping over every scale from 1 to 100.

3.4.5 Plotting LCOE

As aforementioned above a calculation of great interest for energy producing technologies is their LCOE (levelized cost of energy). The wave app should output the correct LCOE for each site, this is achieved in the function LCOE(lat,lon). This function takes in the users latitude and longitude data, which is used to generate results from the getdata() function and in turn PowerGeneration function(). The function returns a plot showing the LCOE over multiple scales, allowing the end user to select the most appropriate scale for the site. The lower the LCOE, the better; Figure 9 gives an example data set which depicts the optimum scale of device as 13, as this gives the lowest LCOE.

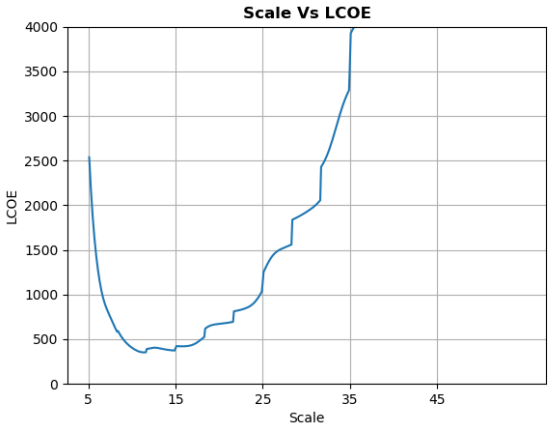


Figure 9: Output of LCOE function. Plot showing scale vs LCOE

3.4.6 Distribution plots

The final functions of the calculation section of the code are two functions which take the values calculated by the PowerGeneration() function and plots the results. Two sets of graphs are calculated, one showing the hours of wave conditions per year at the site, the power in the waves, power of the Penguin at the user defined scale and the capture width of the device (Figure 10). These plots are created as surface plots as they best show the relationship between 3 sets of data. The second set of graphs shows the device power, mass and operational hours of the device all as a function of the Penguin as different scales (Figure 11).

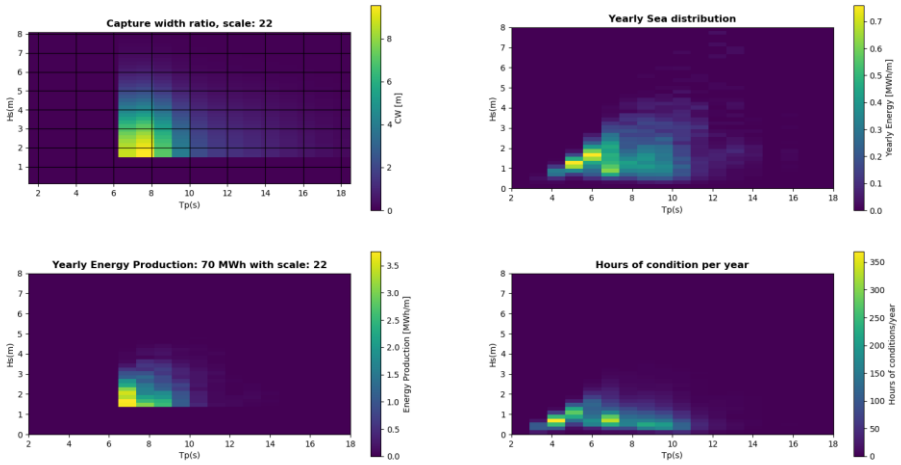


Figure 10 - Top Left Power chart for site. Top right yearly energy distribution of site. Bottom left Yearly Energy Production. Bottom right yearly sea distribution.

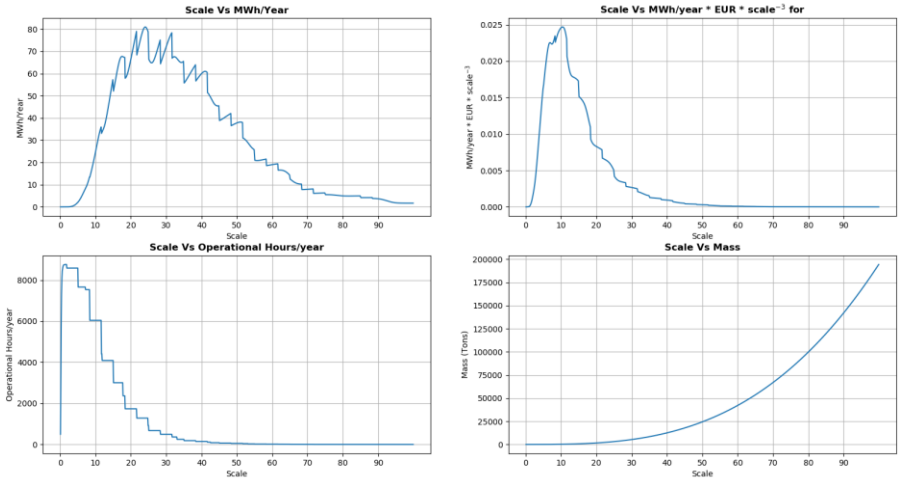


*Figure 11 - Scale plots: (a) scale vs power production, (b) MWh/year*Euro*scale^-3 vs scale, (c) operational hours vs scale, and (d) right scale vs mass..*

3.4.7 Basemap window implementation

The first stage of creating the graphical interface of the wave application was to obtain a method of importing an interactive map into a Tkinter window which would allow the user to select coordinate data by clicking on points on the map. Many different libraries were tested to achieve this, as well as testing application program interfaces (API). Most of the initial creation of the wave app was spent in researching and testing the most suitable platform for the wave app. After extensive research it was deemed that the Basemap library was the most applicable, whilst maintaining relatively easy integration into a Tkinter window. In the wave app Basemap is initialized in its own window by a method which controls what type of map is produced, the resolution of the map, the map color, the boundaries shown on the map and a few other ascetic features. A drawback of the Basemap libraries is that they do not contain the country names on the map; to counter this, a function was added to the map to include some of the country names. Country names are read from a separate text file which contains the country names as well as the latitude and longitude of the country. In a similar method to online maps, the initial zoom setting (whole world view) only some of the country names are given. This was to reduce clutter from printing all the names on one map. However, this solution is less than ideal and will be revisited on a later version of the wave app. Figure 12 shows an image of the finished map image

after all the methods have been added to it. The method for the map is given as m and initialized as m = Basemap(projection='mill', resolution = 'h') (See appendix for full details).

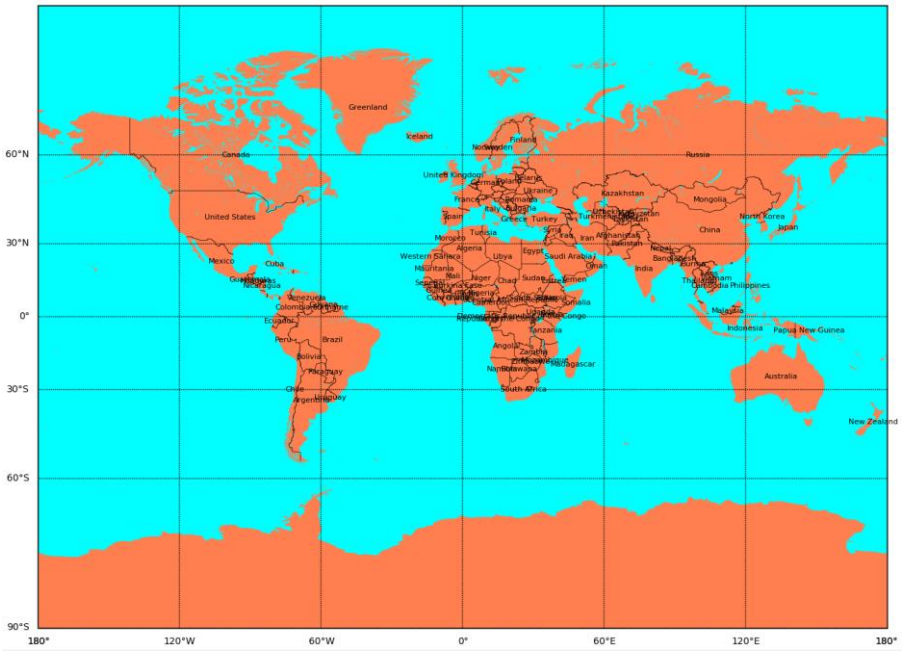


Figure 12: Appearance of the map after all the methods are added to it.

3.4.8 GUI Framework

After the map was established, a class was created forming the core backend of the GUI (class GlobalWaveDataApp(tk.Tk); see Appendix 1). This class initializes the GUI window and creates the platform on which the GUI is built upon. In this class the formatting and layout of the GUI is implementing, adding such things as menu items which appear on every page, a title for the wave app's window and a set configuration of where the objects within the window will be housed on all windows within the app. The class also initializes any new pages and gives them a uniform configuration to allow for easy expansion of the wave app if more features are to be added to it in the future. The rest of the GUI's backend was created to support the implementations of any new pages, menu items and other such functionality which may appear in a GUI.

3.4.9 GUI windows

A ‘home’ page was created for informing the user what the wave app is and what to do if any bugs are encountered in the app. The home page was to clickable buttons which serve their respective functions:

- Agree: Takes you to the main page of the app, where the user can select data, manipulate the map and generate results.
- Disagree: Closes the app

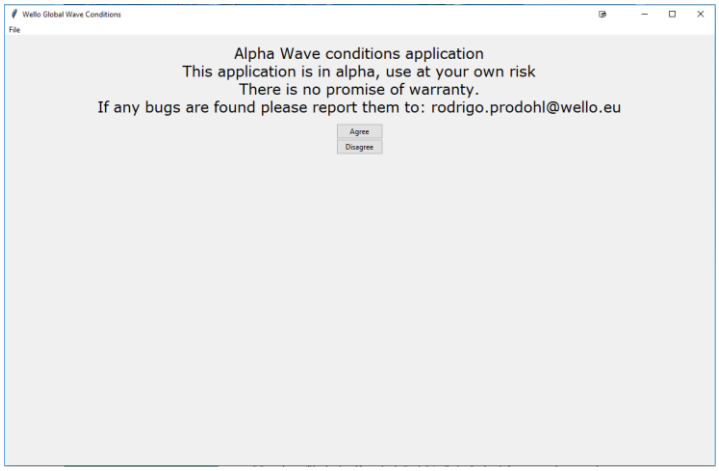


Figure 13: Home page of the wave app

Once the user selects the ‘Agree’ button, they are brought to the main application, which is built upon the Basemap map shown in Figure 12 but with added functionality.

The next stage of the code was to ensure that the map would be able to handle the user clicking on the map to select data points in the form of an ‘entry’ (a user input). As Basemap is based off Matplotlib, the mpl.connect() method was utilized, this function reads the users inputs and saves the x and y axis coordinate data of where the user clicked on the plot or image. The function is also able to differentiate between different types of inputs, such as left or right mouse clicks, double mouse clicks, scroll-wheel up/down, etc. The function was used so that in the event of a user double clicking on anywhere on the map the function onclick(event) would be called and pass the event data. In case of a double click the onclick(event) function stores the points of where the user clicked on as x and y coordinates. These are converted to their corresponding latitude and longitude values and then passed through the getdata() function, which in turn is called by the other aforementioned functions to produce all the desired outputs. The onclick(event) function also allows the user to right click on the map to resets the map to

its original view, the user can zoom in with ease using the toolbar below the map, but zooming out proves more difficult, so this was added to increase functionality.

The second way which the user may select data is by a search option. An entry box was allocated to the top of the window of the map page which takes in the user defined latitude and longitude. The data is queried in the same way as the entry method. Finally, on the page another entry box was created to enable the user to define a scale for the Penguin at the desired site. This entry saves the user defined scale and overwrites the current global variable for scale. The final map page can be seen on Figure 14.

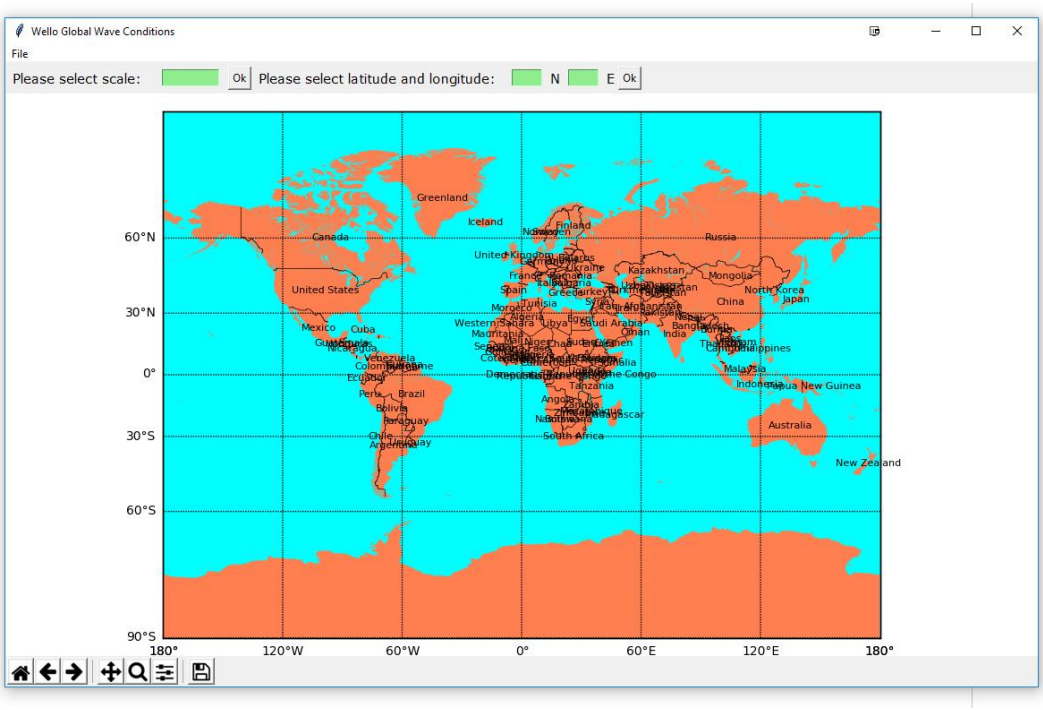


Figure 14: Main wave app page

3.4.10 Data popup window

The final stage of the GUI was to give the user a method of viewing the data once the entry function had been passed. A function for the program to open a new window, or rather popup box was implemented into the code which would display the relevant user inputted data. The popup displays the coordinate points where the data is retrieved from, the average wave power in the site, the chosen scale of the device and the total power of the device per year given as

megawatt hours per year (Figure 15). The popup message also contains clickable buttons which call the functions to generate the different plots which were implemented in the calculation section of the code and finally the button labelled ‘Okay’ which closes the text box and allows the user to continue selecting data.

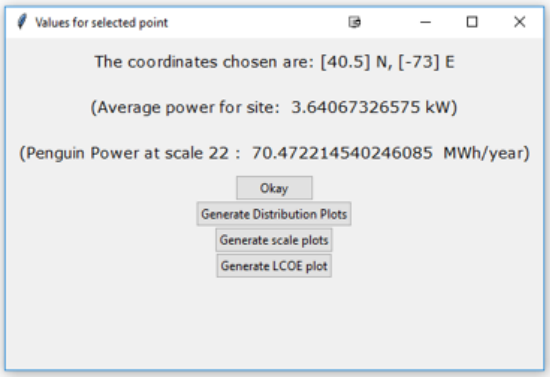


Figure 15: Popup message generated once user selects data points.

4 Results

The outcome of this thesis was to create an application which would allow users to generate wave data for the world’s coastlines via an interactive map. This was achieved as outlined above (Methodology section). The wave app is not freely distributed as the data itself was granted by BMT Argoss and commercial access to the data is granted on a case by case basis and incurs heavy costs. Furthermore, some of the core functions for calculating the power in the Penguin wave energy converter are kept as the property of Wello Oy. As such this section will go through the wave app from the perspective of the user and will outline the steps for using the wave app.

4.1 Navigating the Wave App

Upon opening the wave app, the user is greeted to the home screen (Fgure 15). The home screen acts as a warning to the user as the app is still in Alpha versions and there may be bugs with ever revision of the app.

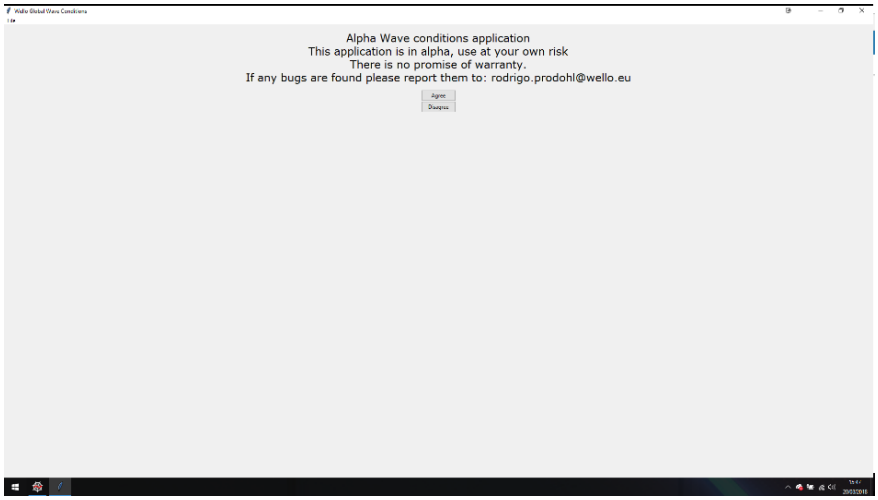


Figure 15: Home page of the wave app

The home screen contains two clickable buttons: Disagree and Accept. If the user selects the 'Disagree' button the application is terminated and all windows are closed. If the 'Accept' button is selected the main page of the wave app opens. The main page is where the user can select data and interact with the map (Figure 16).

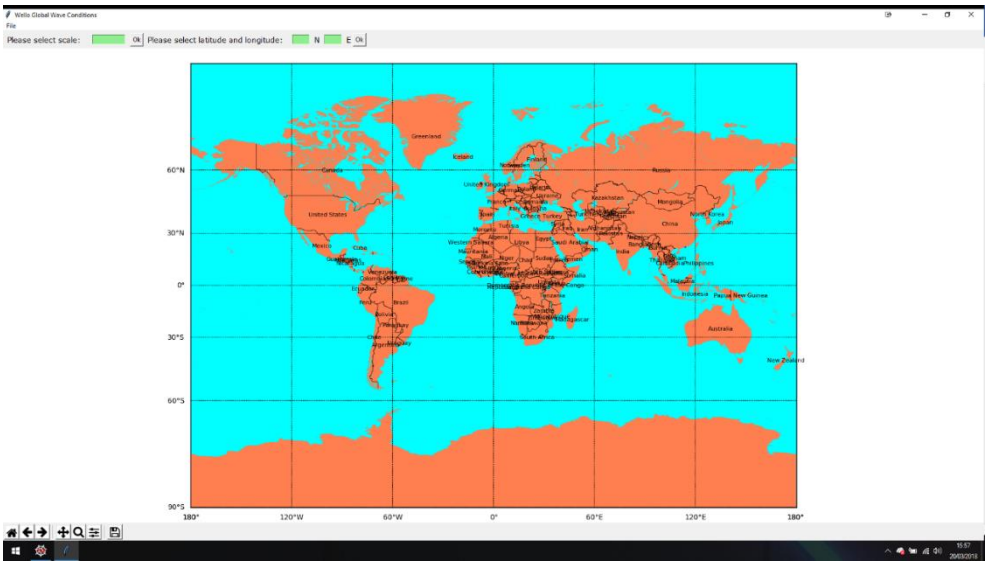


Figure 16: Main page of the wave app showing the map

4.2 Selecting data

4.2.1 Inputting data manually

The first method of selecting is for the user to input their desired longitude and latitude points manually in the boxes labelled ‘Please select latitude and longitude’. To illustrate how the user might input coordinate data, the points for New York, USA, were used in this example (40.65, -73.41). The user would write in the desired data points in the correct boxes and select the ‘ok’ button, prompting a popup window as shown in Figure 17. It should also be noted that in the window there is an option to change the scale of the Penguin, this is due to varying wave conditions found around the world, and so the optimal scale of the Penguin also varies. The user is given the option to change the scale by writing in their desired scale and selecting the ‘ok’ button. The program defaults the scale of the device to 22 unless the user changes the scale; for this example, the scale of the Penguin was changed to 15.

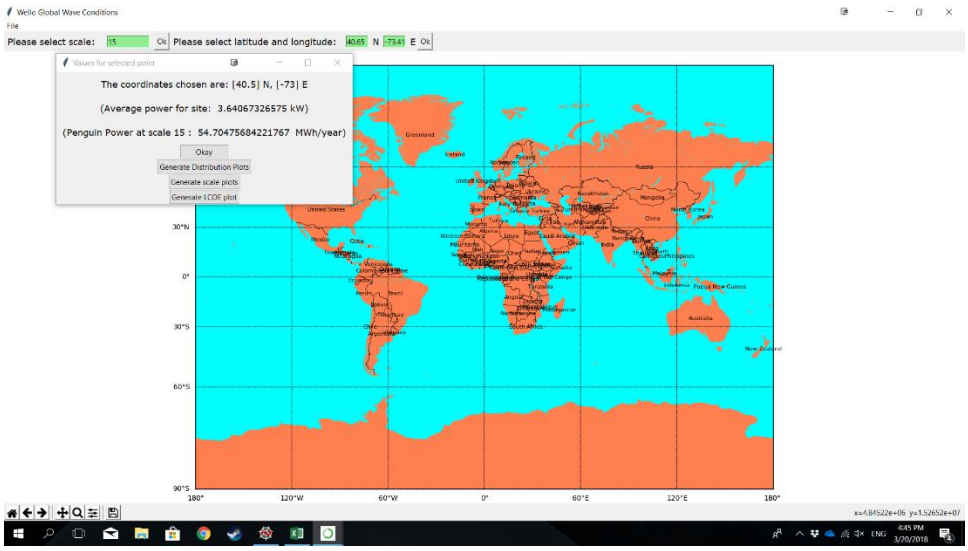


Figure 17: Output from user manually selecting coordinate data

4.2.2 Selecting points on the map

The second method of gaining access to the data is by double clicking on the map itself to generate data from the desired data points. The initial viewing setting of the map page shows the whole view of the world, making it somewhat cumbersome to access specific locations. To get around this a toolbar (Figure 7) was added at the bottom of the page which allows the user to zoom in on a selected area by clicking on the zoom icon and dragging it over the desired area. If, however, the user wants to view the original whole world view they can right click on the map

which will reset the image. In this case, the user selected a point off the west coast of the United States. Once the user double clicks on the desired area, the popup menu is instantly generated. A scale of 25 was also selected for this site as the user wanted a larger device.

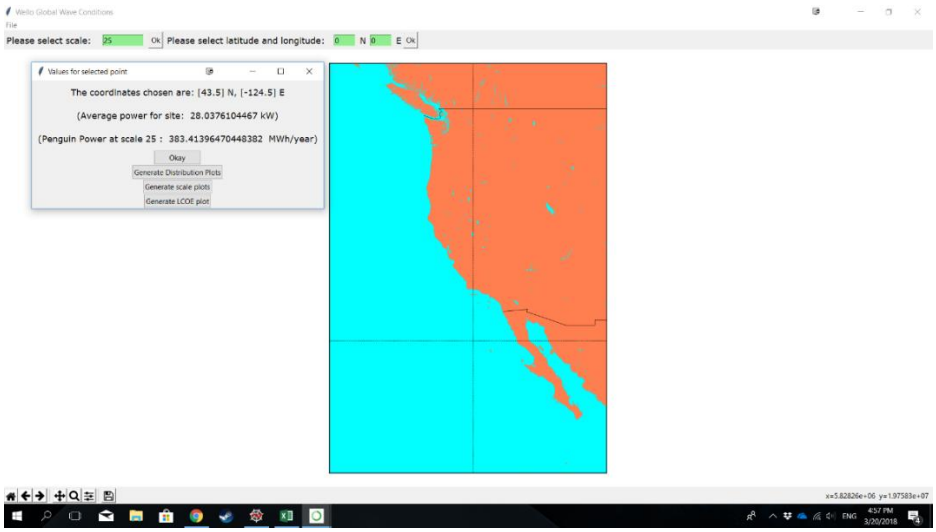


figure 18: Zoomed in view of west coast of USA

4.3 Data presentation

The popup window which is generated whenever the user has selected their data points using either selection method allows the user to view the site-specific data which has been calculated. The popup window in Figure 17 gives the results for the coordinate points, -40.5 N, -73 E. The window displays the location of where the data has been taken from, the average power of the waves in the chosen site and the power per year of the Penguin at the user defined scale (15 in this case.) The popup window also contains buttons which gives the user the option to create plots based on the selected grid points. The 'Okay' button closes the popup window and allows the user to continue selecting data. The outputs of pressing the plotting buttons are presented in Figure 18 and Figure 19.

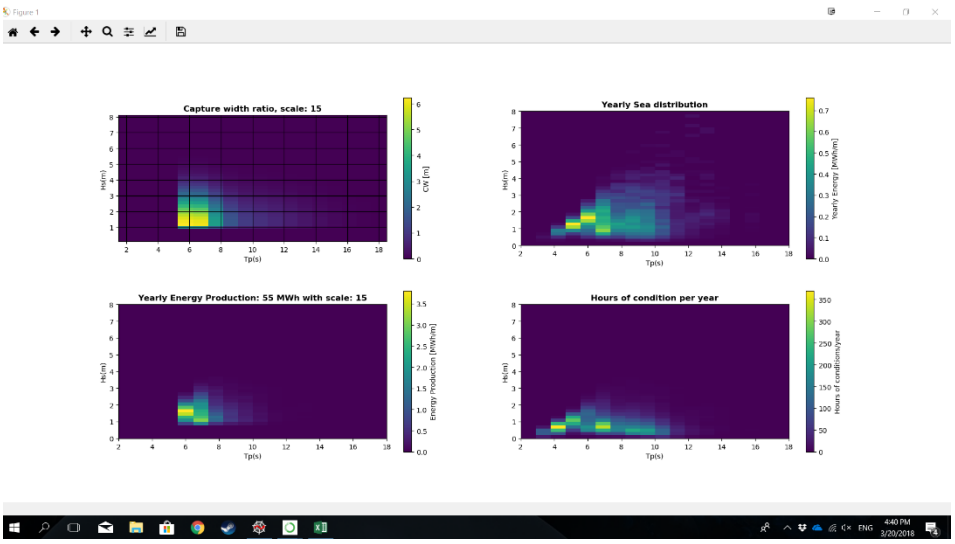


Figure 18: Output from user selecting 'Generate Distribution Plots' Button

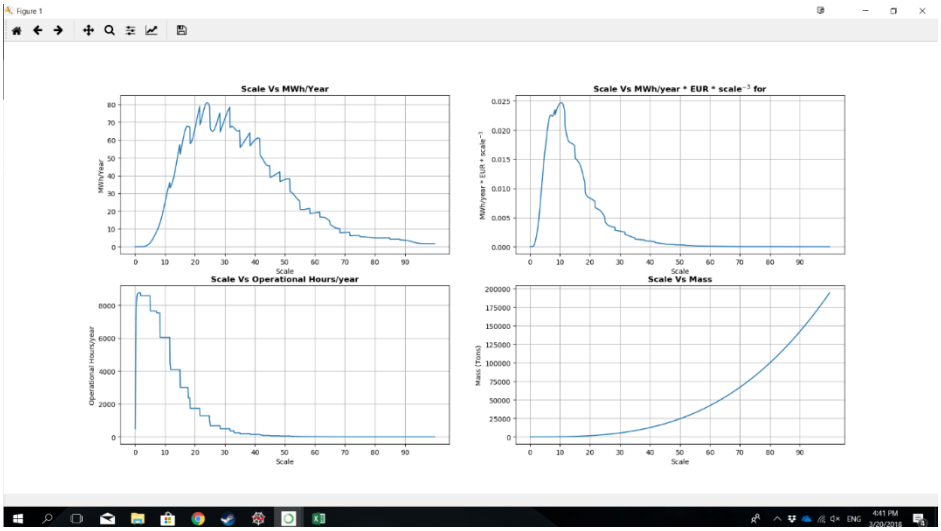


Figure 19: Output from user selecting 'Generate Scale Plots' Button

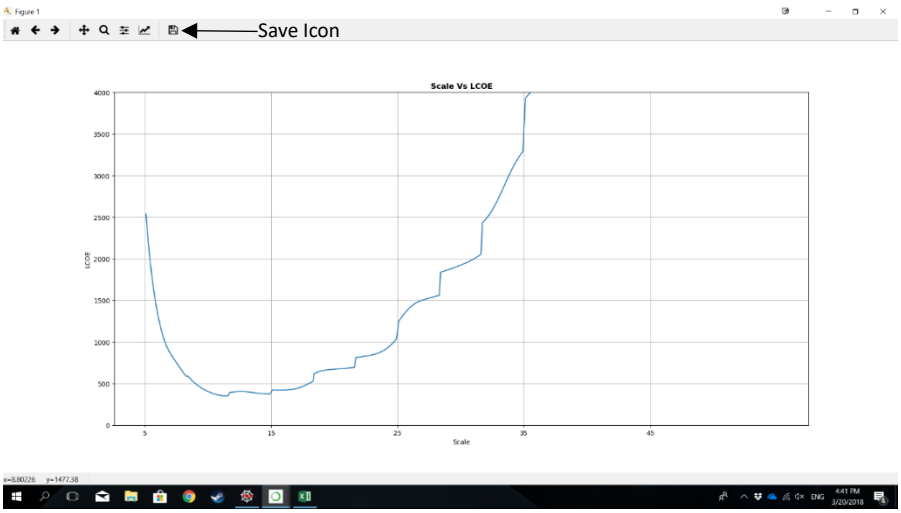


Figure 20: Output from user selecting 'Generate LCOE Plot' Button

It should be noted that if that is the user has the need; each of the plots can be saved and stored to be viewed outside of the program by selecting the save icon (Figure 20). The user should then proceed to save the image as they see fit and follow the standard convention for saving files.

4.4 Exiting the Wave app

They are then free to create plots as explained in the method above. If the user is finished using the program, they can select the 'x' icon on the top right of the window to terminate the window (as is standard with most software packages).

5 Conclusion

Previously in Wello Oy the process for gauging the power output of the Penguin wave energy converter for different sites has been a long and repetitive process. With data procurements, data manipulation and then finally presenting graphically in a neat manner being an arduous process. With the alpha version of the Wello Wave App which has been created this is no longer the case.

The Wello Wave App allows enquires about the Penguin from third parties to be processed at a far faster rate than previous methods, allowing Wello to speed up the rate at which a proper dialogue is formed with these parties and move forward with any perspective projects. Furthermore, the application has been used for market research purposes to quickly gauge the most energetic areas of a coastline allowing for the marketing team to quickly and effectively determine a potential target group.

The thesis was a success in what it aimed to achieve and has built a solid framework for further iterations of the app to be developed and refined. The next steps for the beta version of the application will be to create a more attractive user interface than currently in place. Research into different mapping methods has already been studied to form a more pleasing user experience. Open street maps is one such option which would give a map interface similar to Google maps. More functionality will be added to the map such as a city and country search options as well as the option to email the graph and results directly to any desired email address.

With the help of the Wave App, hopefully more important parties will release the power that the Penguin wave energy converter, and wave energy as a concept, can provide for them. And hopefully the globe can get passed its over-reliance on traditional fossil fuels and invest more in alternate energy sources.

References

Center for International Earth Science Information Network at Columbia University. 2007. *Percentage of total population living in coastal areas.* [ONLINE]. Available at: http://www.un.org/esa/sustdev/natlinfo/indicators/methodology_sheets/oceans_seas_coasts/pop_coastal_areas.pdf

International Energy Agency (IEA). 2018. *Ocean energy potential*. [ONLINE] Available at: https://web.archive.org/web/20150522054948/http://www.iea.org/techinitiatives/renewableenergy/ocean/

European Marine Energy Centre Ltd. 2009. *Tank testing of renewable energy systems.* [ONLINE] Available at: http://user.it.uu.se/~ps/SAS-new.pdf

NTNU University. 2008. *Sea state parameters and engineering wave spectra*. [ONLINE] Available at: http://folk.ntnu.no/oivarn/hercules_ntnu/LWTcourse/partB/3seastate/3%20SEA%20STATE%20PARAMETERS%20AND%20ENGINEERING%20WAVE%20SPECTRA.htm

DNV-GL. 2008. *Certification of Tidal and Wave Energy Converters*. [ONLINE] Available at: https://rules.dnvgl.com/docs/pdf/DNV/codes/docs/2012-04/Oss-312.pdf

Mark Lutz, O'Reilly. 2003. *Python3 Tutorial: History and Philosophy of Python*. [ONLINE] Available at: https://www.python-course.eu/python3_history_and_philosophy.php

NumPy Developers. 2005-2017. *NumPy*. [ONLINE] Available at: http://www.numpy.org/

Hunter et al. 2018. *Matplotlib: Python plotting — Matplotlib 2.1.1 documentation.* [ONLINE] Available at: https://matplotlib.org/

Jeffrey Whitaker. 2011. *Matplotlib Toolkit 1.1.0 documentation. — Basemap Matplotlib Toolkit 1.1.0 documentation.* [ONLINE] Available at: https://matplotlib.org/basemap/

Appendix 1. Wello Wave App source code

```
# -*- coding: utf-8 -*-
"""
Last edited  Mar 20 16:06:26 2018

@author: rprod
"""

import matplotlib, os
matplotlib.use("TkAgg")
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2TkAgg
import matplotlib.animation as animation
import matplotlib.pyplot as plt
from matplotlib import style
import numpy as np
import tkinter as tk
from tkinter import ttk
from scipy import spatial
from mpl_toolkits.basemap import Basemap
from numpy import genfromtxt
from math import exp, pi, sqrt
from scipy.special import erf


'''Font variables to allow for easy fpnt selection in GUI'''
LARGEST_FONT = ('Verdana',24)
LARGE_FONT = ('Verdana',18)
NORM_FONT = ('Verdana',11)
SMALL_FONT = ('Verdana',8)

'''Initialising the scale of the device'''
SCL = 22

Map_type = 'Coastline'
programName ='C_Map'

'''Sets directory to where the global wave data is stored'''
os.chdir(r"WaveData")

'''Creates an array of values for the latitudes and longitudes of the data points.
The files are in the format LAT,LON.txt. The array acts as a lookup table to query
the wave data at each specific point. The final variable coord_list, acts as a database
for querying the wave data.
```

```
    Parameters
    ----------
    listdirec : list
        Provides a list of the files in the working directory

    coord_list : NumpPy array
        Creats an array of the directory names, column 0 = latitude
        column 1 = longitude
    ----------
'''

listdirec = os.listdir()

coord_list = []
for i in listdirec:
    x = i[:-4]
    z = x.split(',')
    coord_list.append(z)

coord_list = np.array(coord_list,dtype=float)


def getdata(lat,lon):
    '''Fucntion which finds the wave data from the given latitudes and longitudes
which are fed into the program by the means of an entry. The function takes in two
values latitude and longitude

    Parameters
    ----------
    lat : float
        Provides the Latitude values for the user defined coordinate point
    lon : float
        Provides the Longitude values for the user defined coordinate point

    Returns
    -------
    Tpi : NumpPy array
        Array of peak period created from coordinates

    Hsi : NumpPy array
        Array of Significant wave height created from coordinates
    Tei : NumpPy array
        Array of energy period derived from significant wave height and
        peak period
```

```
TpHs_array :NumpPy array
   meshgrid of peak period and significant wave height
filehandle: string
   Filename of chosen data source'''

print('Data function')
global coord_list
pt = [lat,lon]

#K-d tree finds coordinate points which most closely matches the user defined points
distance,index = spatial.KDTree(coord_list).query(pt)
filehandle = listdirec[index]
print(filehandle)

#genfromtxt opens the text file where the coordinate data is stored and creates a data array
pt_data = genfromtxt(filehandle,delimiter=',')
pt_array = np.nan_to_num(np.array(pt_data))

#pt_array.shape finds the shape of the array
y,x = pt_array.shape
y,x = y-1,x-1 # Finds the max length of x and y, -1 to account for NAN which comes after

Tp = pt_array[0:2,2:x]
Hs = pt_array[2:y:,0:2]

# Array based off the inputs from the data file
TpHs_array = pt_array[2:y,2:x]

Hs_low = Hs[:,0]
Hs_high = Hs[:,1]

#Hs_bin contains mean values of significatn wave height
Hs_bin = (Hs_low+Hs_high)/2


#Tp_bin contains mean values of peak period
Tp_bin = ((Tp[0])+(Tp[1]))/2


#Creates a new Tp and Hs array for use in the function
[Tpi,Hsi] = np.meshgrid(Tp_bin,Hs_bin)

inter = (Tpi/(np.sqrt(Hsi)))
inter = np.array(inter)
```

```
    #Store matrix shape
    x,y = inter.shape
    inter = inter.ravel()

    gamma = []
    for i in inter:
        if i <= 3.6:
            gamma.append(5)
        elif i<5 and i>3.6:
            gamma.append(exp(5.75-1.15*i))
        elif i>=5:
            gamma.append(1)

    #Gamma is made into a matrix to mimic the shape of the Hs, Tp array
    gamma = np.array(gamma)
    gamma = gamma.reshape(x,y)


    # Te calculated to use in power calculation'''
    Tei = Tpi*((4.2+gamma)/(5+gamma))

    return(Tpi,Hsi,Tei,TpHs_array,filehandle)



def scale(mu):
    '''Function for the scaling factors.

        Parameters
    ----------
    mu : float
        The chosen scaling factor for the device

    Returns
    -------
    ts : float
        time scaled
    hs : float
        length scaling factor
    ps : float
        power sclaed
    cws : float
        capture width scaled'''
```

```
    ts=mu**(.5)
    hs=mu
    ps=mu**3.5
    cws=mu
    return(ts,hs,ps,cws)



def P3_7_CWfit(Tp,Hm,s):
    '''Function based off tank testing results of The Penguin: model 3.7
the final function of this script is to give a capture width model for
each scale. We can then relate than model to a given wave distribution to generate
estimated power values for the device at each scale.

”’Section on code omitted”’

    return(cw)


def P3_8_CWfit(Tp,Hm,s):
    '''Function based off tank testing results of The Penguin: model 3.8
    gives same inputs and outputs as above'''
    ts,hs,ps,cws = scale(s)

”’Section on code omitted”’

    return(cw)


''' Main code to generate estimated power at each site'''
def PowerGeneration(Tpi,Hsi,Tei,TpHs_array):
    '''Function for calculating Power values based off the available
    wave resources from the select data point

        Parameters
    ----------

    Tpi: array
        Peak period array pulled from data sourse
    Hsi: array
        Significant have height array pulled from data source
    Tei: array
        energy period array based off calculations between peak period
        and significant wave height
    TpHs_array:
```

```
Returns
-------
P_ave : float
   average power in waves kW/m
EnergyperyearTot: float
   total energy per year in waves MW/m
Energy_Peng: array
   How many MW/h of energy is produced by the device at each wave condition
CW_Data: array
   Capture width array of device for selected wave resources
SCL: float
   Scale
Pricecurve: list
   vector of price curve of the device versus scaling
TotalPowperyear: float
   Total MWh/year is produced by the device per year
Hours: array
   Hours of each condition per year
POW: array
   power of the device at selected point
Op_toti: list
   operational hours per year at each scale
Total_EPi: list
   MWh/year of device at each scale
   '''
global SCL

print('The scale is',SCL)

# Axilary factor for the device (power draw)
Aux = 0.9

#Number of hours of condition per year
Hours = (np.dot(TpHs_array,8760)/100)

#Capture width model for device @ certain scale
CW_Data = P3_8_CWfit(Tpi,Hsi,SCL)

#Rho = specific density of water
rho = 1.03E3
#g = gravity
g  = 9.81
```

```
#Creates a power matrix for the power of the waves at a given site
Jirr = (rho*(g**2)*(Tei)*(Hsi**2))/(64*pi)

#array of energy per year per meter of wave resource
Energyperyear = Jirr*Hours*1e-6
#Total energy per year per meter of wave resource
EnergyperyearTot = np.sum(Energyperyear)


print(EnergyperyearTot,' MWh/m')



sumHours = np.sum(Hours)
P_ave = (EnergyperyearTot/sumHours)*1e3
print(P_ave,'kW/m')

#Power is derived from capture width x energy in waves. It takes into
#Account the power required from the auxilary
POW =CW_Data*Jirr*1e-3
POW = POW*Aux
Energy_Peng = POW*Hours*1e-3


TotalPowperyear = np.sum(Energy_Peng)
print(TotalPowperyear)

#This is for calculating the price
Price = 1

#dictonaries to allow data to be saved across every iteration of the loop
Total_EPi = []
Op_toti = []
Pricecurve = []

'''Iterates over a range of different scales(from 1 to 100) to best determine
what scale should be used at each site and how the changing
of the size of the device affects the performance. Naming conventions for this
are the same a their single scale counterparts but with the added -i at
the end of the variable name'''
for i in range(1,1000):

    CW_scalei = P3_8_CWfit(Tpi,Hsi,(i/10))

    Powi = (CW_scalei*Jirr*1e-3)*Aux
```

```
        Powi[Powi<0]=0

        Energy_Pengi = Powi*Hours*1e-3

        Total_EP=np.sum(Energy_Pengi)
        Total_EPi.append(Total_EP)


        Energyperyeari= Energyperyear*CW_scalei

        #Creates a matrix of ones with the same shape as the array of energy
        onesi = Energyperyeari
        onesi[Energyperyeari>0]=1

        #Gives operational hours per year
        Op_Hi = onesi*Hours
        Op_toti.append(np.sum(Op_Hi))
        Pricecurve.append(Total_EP*Price*(i/10)**(-3))


return(P_ave,EnergyperyearTot,Total_EPi,Op_toti,Pricecurve,CW_Data,SCL,TotalPowperyear,Energyperyear,POW,Energy_Peng,Hours)


def LCOE(lat,lon):
    '''Function to give LCOE of the device at different scale. LCOE gives
        best comparison for other energy types

        Parameters
    ----------
    lat : float
        Provides the Latitude values for the user defined coordinate point
    lon : float
        Provides the Longitude values for the user defined coordinate point

    Returns
    -------
    Plot : Plot
        Returns plot of estimated LCOE results over a series of scales
        '''

    Tpi,Hsi,Tei,TpHs_array,filehandle = getdata(lat,lon)
```

```
P_ave,EnergyperyearTot,Total_EPi,Op_toti,Pricecurve,CW_Data,SCL,TotalPowperyear,Energyperyear,POW,Energy_Peng,Hours =
PowerGeneration(Tpi,Hsi,Tei,TpHs_array)


scaling = []
for i in range(1,1000):
    scaling.append(i/10)


#Mass of the scale 1 device in tons
mass = 0.2015
mass_scl = []
for i in range(0,len(scaling)):
    mass_scl.append(mass*scaling[i]**3)

#Set price for penguin at 22 scale (as we know the price)
P_Price = 900000
#Life Time
LT = 30
#Loan rate
Loan_rate = 0.015
#Insurance rate/year
In_rate = 0.015

#Price per ton, this is based off of value for price @ 22 scale.'''
Ton_P = P_Price/2.076360000000000e+03

#Price at each scale in relation to mass
P_scl =  np.asarray(mass_scl)*Ton_P

#loan per year
Loan_y = ((np.asarray(P_scl)*Loan_rate)/(1-((1+(Loan_rate))**(-LT))))

#Insurance per year at each scale of the device
In_y = P_scl*In_rate
Loan_tot = Loan_y + In_y

#Utilisation factor is set at a constant now. However, there is
#a correct formula as shown below which should be used when agreed upon'''
# UF = (P_scl*Op_toti*Mt)/H_y
#Eur_OpH = np.asarray(UF)/Op_toti

UF = 5000
Tot_costs = UF + Loan_tot
```

```
    #E_pro_p is the actual LCOE at the specific scale
    E_pro_p = Tot_costs/Total_EPi


    y = E_pro_p[50:550:1]
    x = scaling[50:550:1]

    #Plots the LCOE plot, which is generated via entry
    plt.plot(x,y)
    plt.grid()
    plt.ylim(0,4000)
    plt.xticks(np.arange(5,55,10))
    plt.xlabel('Scale')
    plt.ylabel('LCOE')
    plt.title('Scale Vs LCOE',weight='bold')
    plt.show()



def DistributionPlots(lat,lon):
    '''This function takes lat and lon data and plots graphs from the power
    values, masses etc of the chosen coordinate site. The function plots
    distribution graphs at the user defined scale

     Parameters
    ----------
    lat : float
        Provides the Latitude values for the user defined coordinate point
    lon : float
        Provides the Longitude values for the user defined coordinate point

    Returns
    -------
    Plot : Plot
        Returns 4 plots on a single figure. Showing wave resources at site,
        Energy of device per year, Capture width of the device, and yearly
        distribution of waves
        '''

    plt.close()

    Tpi,Hsi,Tei,TpHs_array,filehandle = getdata(lat,lon)
    P_ave,EnergyperyearTot,Total_EPi,Op_toti,Pricecurve,CW_Data,SCL,TotalPowperyear,Energyperyear,POW,Energy_Peng,Hours =
PowerGeneration(Tpi,Hsi,Tei,TpHs_array)
```

```
Tp_vec = Tpi[1]
Hs_vec = Hsi[:,1]


#Capture Width Model
CW_Data = np.matrix(CW_Data)
plt.subplot(2,2,1)
#imshow plots a graph showing the correlation between Tp and Hs in relation
#to Capture width. Visually informative way of viewing the array data. This is
#similarly repeated below
plt.imshow(CW_Data,interpolation='nearest',origin='lower',
        extent=[min(Tp_vec),max(Tp_vec),
                min(Hs_vec),max(Hs_vec)])

#Capture Width Model graph
plt.grid(color='#000000')
plt.xlabel('Tp(s)')
plt.ylabel('Hs(m)')
clb=plt.colorbar()
clb.set_label('CW [m]')
plt.title('Capture width ratio, scale: %i' %SCL,weight='bold')
plt.grid(which='minor',alpha=0.5)


#Yearly Sea distribtuion
plt.subplot(2,2,2)
plt.imshow(Energyperyear,interpolation='nearest',origin='lower',
        extent=[int(round(min(Tp_vec))),int(round(max(Tp_vec))),
                int(round(min(Hs_vec))),int(round(max(Hs_vec)))])
plt.xlabel('Tp(s)')
plt.ylabel('Hs(m)')
clb=plt.colorbar()
clb.set_label('Yearly Energy [MWh/m]')
plt.title('Yearly Sea distribution',weight='bold')


#Yearly Energy Production
plt.subplot(2,2,3)
plt.imshow(Energy_Peng,interpolation='nearest',origin='lower',
        extent=[int(round(min(Tp_vec))),int(round(max(Tp_vec))),
                int(round(min(Hs_vec))),int(round(max(Hs_vec)))])
plt.xlabel('Tp(s)')
plt.ylabel('Hs(m)')
clb=plt.colorbar()
clb.set_label('Energy Production [MWh/m]')
```

```
    plt.title('Yearly Energy Production: %i MWh with scale: %i' %(round(TotalPowperyear),SCL),weight='bold')


    #Hours of conditions per year
    plt.subplot(2,2,4)
    plt.imshow(Hours,interpolation='nearest',origin='lower',
            extent=[int(round(min(Tp_vec))),int(round(max(Tp_vec))),
                    int(round(min(Hs_vec))),int(round(max(Hs_vec)))])
    plt.xlabel('Tp(s)')
    plt.ylabel('Hs(m)')
    clb=plt.colorbar()
    clb.set_label('Hours of conditions/year')
    plt.title('Hours of condition per year',weight='bold')
    plt.show()


def PowerChart(lat,lon):
    '''Plot a power chart showing the maxtrix of power values for the
   user defined scale

    Parameters
    ----------
    lat : float
        Provides the Latitude values for the user defined coordinate point
    lon : float
        Provides the Longitude values for the user defined coordinate point

    Returns
    -------
    Plot : Plot
        Returns plot a power chart showing the maxtrix of power values for the
   user defined scale
    '''
    plt.close()
    Tpi,Hsi,Tei,TpHs_array,filehandle = getdata(lat,lon)
    P_ave,EnergyperyearTot,Total_EPi,Op_toti,Pricecurve,CW_Data,SCL,TotalPowperyear,Energyperyear,POW,Energy_Peng,Hours =
PowerGeneration(Tpi,Hsi,Tei,TpHs_array)

    Tp_vec = Tpi[1]
    Hs_vec = Hsi[:,1]


    plt.imshow(POW,interpolation='nearest',origin='lower',
        extent=[int(round(min(Tp_vec))),int(round(max(Tp_vec))),
            int(round(min(Hs_vec))),int(round(max(Hs_vec)))])
```

```
    plt.xlabel('Tp(s)')
    plt.ylabel('Hs(m)')
    clb=plt.colorbar()
    clb.set_label('Power kW')
    plt.title('Power Chart',weight='bold')
    plt.show()



def ScalePlots(lat,lon):
    '''Plots the results of PowerGeneration function over each scale from 1 to
       100

    Parameters
    ----------
    lat : float
        Provides the Latitude values for the user defined coordinate point
    lon : float
        Provides the Longitude values for the user defined coordinate point

    Returns
    -------
    Plot : Plot
        Returns 4 plots on a single figure. Showing Scale vs energy per year,
        scale vs mass of device, Scale vs price, Scale vs operaitional hours
        distribution of waves
    '''

    plt.close()
    Tpi,Hsi,Tei,TpHs_array,filehandle = getdata(lat,lon)
    P_ave,EnergyperyearTot,Total_EPi,Op_toti,Pricecurve,CW_Data,SCL,TotalPowperyear,Energyperyear,POW,Energy_Peng,Hours =
PowerGeneration(Tpi,Hsi,Tei,TpHs_array)


    scaling = []
    for i in range(1,1000):
        scaling.append(i/10)

    #Scale vs MWh
    plt.subplot(2,2,1)
    plt.plot(scaling,Total_EPi)
    plt.grid()
    plt.xticks(np.arange(0,100,10))
    plt.xlabel('Scale')
```

```
    plt.ylabel('MWh/Year')
    plt.title('Scale Vs MWh/Year',weight='bold')

    #Scale vs Price
    plt.subplot(2,2,2)
    plt.plot(scaling,Pricecurve)
    plt.grid()
    plt.xticks(np.arange(0,100,10))
    plt.xlabel('Scale')
    plt.ylabel('MWh/year * EUR * scale$^{-3}$')
    plt.title('Scale Vs MWh/year * EUR * scale$^{-3}$ for',weight='bold')

    #Scale vs Operational Hours
    plt.subplot(2,2,3)
    plt.plot(scaling,Op_toti)
    plt.grid()
    plt.xticks(np.arange(0,100,10))
    plt.xlabel('Scale')
    plt.ylabel('Operational Hours/year')
    plt.title('Scale Vs Operational Hours/year',weight='bold')


    #Scale vs Mass
    mass = 0.195
    mass_scl = []
    for i in range(0,len(scaling)):
        mass_scl.append(mass*scaling[i]**3)

    plt.subplot(2,2,4)
    plt.plot(scaling,mass_scl)
    plt.grid()
    plt.xticks(np.arange(0,100,10))
    plt.xlabel('Scale')
    plt.ylabel('Mass (Tons)')
    plt.title('Scale Vs Mass',weight='bold')


#def City_Coords(cityname):
#    '''Function to hopefully allow for searchable cities'''
#    cityfile = open(r'C:\Users\rprod\.spyder-py3\WaveData\City_Coords.txt')
#    content = cityfile.readlines()
#
#    citycontent = []
#
#    for i in content:
```

```
#       citycontent.append(i.split(','))



def Country_Names():
    '''Function which reads CountryCentre.txt which contains: countrycode,
    latitude, longitude and population

    Paramaters
    ----------
    countrycodes: array
        array of data from CountryCentre.txt

    Returns
    -------
    lat : float
        Provides the Latitude values for the country
    lon : float
        Provides the Longitude values for the country
    countrynames: string
        Country name
    countrypop: float
        Country population

    '''
    countryfile = open(r'C:\Users\rprod\Documents\Python Scripts\Thesis\CountryCentre.txt','r')
    content = countryfile.readlines()

    #iterates over CountryCentre.txt and splits the data by commas ','
    countrycodes = []
    for i in content:
        countrycodes.append(i.split(','))

    #fromats the data to an array
    countrycodes = np.array(countrycodes)



    # lat and lon of the countries
    lat_lon = countrycodes[:,1:3]
    lat = lat_lon[:,0]
```

```
    lon = lat_lon[:,1]

    #converts the data from strings to float
    lat = lat.astype(np.float)
    lon = lon.astype(np.float)

    #converts data to a list
    lat = lat.tolist()
    lon = lon.tolist()


    countrynames = countrycodes[:,3]
    countrypop = countrycodes[:,4]

    countrypop = countrypop.astype(np.float)
    countrylist = countrynames.tolist()

    countrynames = []
    for i in countrylist:
        countrynames.append(i.strip())


    return(lat,lon,countrynames,countrypop)




def popupmsg(msg1,msg2,msg3,lat,lon):
    '''Creates popup box based off user's inputs. The Data is printed in the
    popup box. Also gives bottons to call functions which creates plots for
    a user defined site.

        Parameters
    ----------
    msg1: string
        Prints the coordinates chosen by the user
    msg2: string
        Prints the average wave power in the site (kW/m)
    msg3: string
        Prints the total amount of energy in MWh/year of the Penguin at the
        users chosen site and scale (default scale is 22)
    lat : float
        Provides the Latitude values for the user defined coordinate point
    lon : float
```

```
    Provides the Longitude values for the user defined coordinate point

'''

#tk.Tk initialises a Tkinter window
popup = tk.Tk()
#tkraise the popup window to the front'''
popup.tkraise()
popup.lift()

#wm_title sets site for popup
popup.wm_title("Values for selected point")

#tk.Label creates a text label for the Tkinter popup window. .pack
#places the
label_1 = tk.Label(popup,text=msg1,font=NORM_FONT)
label_1.pack(pady=10, padx=10)

label_2 = tk.Label(popup,text=msg2,font=NORM_FONT)
label_2.pack(pady=10, padx=10)

label_3 = tk.Label(popup,text=msg3,font=NORM_FONT)
label_3.pack(pady=10, padx=10)

B1 = ttk.Button(popup, text="Okay", command = popup.destroy)
B1.pack()


#Functions have to be initialised to call outer functions. This
#allows for function calling from an entry within the popup menu
def GenerateDistPlots():
    DistributionPlots(lat,lon)

def GenerateScalePlots():
    ScalePlots(lat,lon)

def GenerateLCOE():
    LCOE(lat,lon)

def GeneratePowerChart():
    LCOE(lat,lon)

#ttk.Button creates a button which the user can 'press' to generate some
#response. In this case it generates the plotting functions based on which
#button the user selects
```

```
    B2 = ttk.Button(popup, text="Generate Distribution Plots", command = GenerateDistPlots)
    B2.pack()

    B3 = ttk.Button(popup, text="Generate scale plots", command = GenerateScalePlots)
    B3.pack()

    B4 = ttk.Button(popup, text="Generate LCOE plot", command = GenerateLCOE)
    B4.pack()

    B5 = ttk.Button(popup, text="Generate Power Chart", command = GeneratePowerChart)
    B5.pack()

    popup.mainloop()



def datatotext(P_ave,TotalPowperyear,filehandle,SCL):
    '''Function which converts results from the user chosen data to strings,
    to allow for easier data handling in other function.

    Parameters
    ----------
    P_ave : float
        average power in waves kW/m
    TotalPowperyear: float
        Total MWh/year is produced by the device per year
    filehandle: string
        Returns the filehandle of the data source of the user defined data
    SCL:
        User defined scale of the device) (default is 22)

    Returns
    -------
    msg1: string
        Prints the coordinates chosen by the user
    msg2: string
        Prints the average wave power in the site (kW/m)
    msg3: string
        Prints the total amount of energy in MWh/year of the Penguin at the
        users chosen site and scale (default scale is 22)
    '''

    filehandle = filehandle[:-4]
    NE = filehandle.split(',')
```

```
    N = NE[:1]
    E = NE[1:2]


    msg1 = 'The coordinates chosen are: '+str(N)+' N, '+str(E)+' E'
    msg1 = str(msg1)
    msg1 = msg1.replace("'",'')

    msg2 = 'Average power for site: ',str(P_ave) + ' kW'
    msg2 = str(msg2)
    msg2 = msg2.replace("'",'')
    msg2 = msg2.replace(",",'')


    msg3 = 'Penguin Power at scale', SCL,': ',TotalPowperyear,' MWh/year'
    msg3 = str(msg3)
    msg3 = msg3.replace("'",'')
    msg3 = msg3.replace(",",'')

    return(msg1,msg2,msg3)


def Search_latlon(lat,lon):
    '''Generates a popup window based off typed user defined coordinate data

    Parameters
    ----------
    lat : float
        Provides the Latitude values for the user defined coordinate point
    lon : float
        Provides the Longitude values for the user defined coordinate point

    Returns
    ----------
    msg1: string
        Prints the coordinates chosen by the user
    msg2: string
        Prints the average wave power in the site (kW/m)
    msg3: string
        Prints the total amount of energy in MWh/year of the Penguin at the
        users chosen site and scale (default scale is 22)
    lat : float
    lon : float
    '''
```

```
    Tpi,Hsi,Tei,TpHs_array,filehandle = getdata(lat,lon)

    P_ave,EnergyperyearTot,Total_EPi,Op_toti,Pricecurve,CW_Data,SCL,TotalPowperyear,Energyperyear,POW,Energy_Peng,Hours =
PowerGeneration(Tpi,Hsi,Tei,TpHs_array)

    msg1,msg2,msg3 = datatotext(P_ave,TotalPowperyear,filehandle,SCL)

    popupmsg(msg1,msg2,msg3,lat,lon)



def onclick(event,lat,lon):
    '''Event handling of clicking on the map to generate data. When the user
    double clicks on the map it finds the closest data set to the users 'clicked'
    coordinate values. It generates a popup window of the results.

    Parameters
    ----------
    event : event
        user created 'event' i.e. click on image

    Returns
    ----------
    msg1: string
        Prints the coordinates chosen by the user
    msg2: string
        Prints the average wave power in the site (kW/m)
    msg3: string
        Prints the total amount of energy in MWh/year of the Penguin at the
        users chosen site and scale (default scale is 22)
    latpt : float
        Provides the Latitude values for the user defined coordinate point
    lonpt : float
        Provides the Longitude values for the user defined coordinate point

    '''


    #event.dblclick handles the doubleclick event from the user
    if event.dblclick:
        ''' Function for reading the files will have to go somewhere here'''
```

```
        #Basemap creates a map of the world. The projection of the map is millers
        #projection (mil) and generates it in 'medium' resolution
#       m = Basemap(projection='mill',
#                   resolution = 'i')

        #Takes the x and y data of the user 'clicked' point
#       xdata = event.xdata
#       ydata = event.ydata

        #Converts the x and y data to coordinate values
#       lonpt,latpt =xdata,ydata


#       lonpt,latpt =m(xdata,ydata,inverse=True)

        Tpi,Hsi,Tei,TpHs_array,filehandle = getdata(lon,lat)


P_ave,EnergyperyearTot,Total_EPi,Op_toti,Pricecurve,CW_Data,SCL,TotalPowperyear,Energyperyear,POW,Energy_Peng,Hours    =
PowerGeneration(Tpi,Hsi,Tei,TpHs_array)
        msg1,msg2,msg3 = datatotext(P_ave,TotalPowperyear,filehandle,SCL)

        popupmsg(msg1,msg2,msg3,lon,lat)

    #event button 3 is right click and resets the image if the user has zoomed
    #and wishes to return to the original world view
    elif event.button == 3:
        print('resseting the image')
        plt.draw() # force re-draw
        ax.set_xlim(0.0, 40030154.742485225)
        ax.set_ylim(0.0, 29350068.807226714)
        plt.draw()



fig = plt.figure()
ax = fig.add_subplot(111)


class GlobalWaveDataApp(tk.Tk):
    '''Tkinter object which forms the core backend of the GUI'''



    def __init__(self, *args, **kwargs):
```

```
'''Function which initializes the GUI window and creates the platform
on which the GUI is built upon'''


tk.Tk.__init__(self, *args, **kwargs)
tk.Tk.wm_title(self,"Wello Global Wave Conditions")

#tk.Frame creates a frame for text as well as a scrollbar
container = tk.Frame(self)
#container.pack configures where the ojects within the window will be housed
container.pack(side='top',fill='both',expand=True)
#implement stretchability along rows and column
container.grid_rowconfigure(0,weight=1)
container.grid_columnconfigure(0,weight=1)

#add menubar to window
menubar = tk.Menu(container)


filemenu = tk.Menu(menubar,tearoff=0)

#filemenu.add_command adds menubar items
filemenu.add_separator()
filemenu.add_command(label='Exit',command=quit)
menubar.add_cascade(label='File',menu=filemenu)


#Configure the menu items onto the actual menu
tk.Tk.config(self, menu = menubar)



self.frames={}

#Initialise all of the pages and all create backend for
#easy page addition in the future
for F in (StartPage, PageOne):

    frame = F(container,self)

    self.frames[F] = frame

    frame.grid(row=0,column=0,sticky='nsew')

self.show_frame(StartPage)
```

```
    #Shows the frame and brings it to the top of the GUI
    def show_frame(self,cont):

        frame =self.frames[cont]
        frame.tkraise()


class StartPage(tk.Frame):
    '''Start page of the Application. First thing the user
    views when running the script'''


    def __init__(self,parent,controller):

        tk.Frame.__init__(self,parent)
        label = tk.Label(self,text="""Alpha Wave conditions application
This application is in alpha, use at your own risk
There is no promise of warranty.
If any bugs are found please report them to: rodrigo.prodohl@wello.eu""",font=LARGE_FONT)
        label.pack(pady=10,padx=10)

        button_1 = ttk.Button(self,text="Agree",
                            command=lambda:controller.show_frame(PageOne))
        button_1.pack()

        button_2 = ttk.Button(self,text="Disagree",
                            command=quit)
        button_2.pack()




class PageOne(tk.Frame):
    '''Main page of the application. Where the user can view the map
    and either input data points or select a point on a map'''

    def __init__(self,parent,controller):
        tk.Frame.__init__(self,parent)
        global programName



        #Basemap () takes two arguments, projection: map type(millers projection)
```

```
# h(high)and resolution:
m = Basemap(projection='mill',
      resolution = 'h')

#.drawcountries draws coutry boarders
m.drawcountries()

#include a graticule grid, a reference network of labelled latitude and longitude lines.
m.drawparallels(np.arange(-90,90,30),labels=[1,0,0,0])
m.drawmeridians(np.arange(m.lonmin,m.lonmax+30,60),labels=[0,0,0,1])

#fills in color of the country
m.fillcontinents(color='coral',lake_color='aqua')
m.drawmapboundary(fill_color='aqua')

#calls Country_Names() function
lat,lon,countrynames,countrypop = Country_Names()

#converts the lat and lon values to Basemap coordinate format
xpt,ypt = m(lon,lat)

new_list = []
new_xpt = []
new_ypt = []

#Removes country names with a population of less than 1e6. Done to
#eliminate clutter from the map
for i in range(len(countrypop)):
   if countrypop[i] >= 100000:
      new_list.append(countrynames[i])
      new_xpt.append(xpt[i])
      new_ypt.append(ypt[i])

#'Writes' the remaining names to the Basmeap image
for i in range(len(new_list)):
   plt.text(new_xpt[i],new_ypt[i],new_list[i],fontsize=8,va='center',ha='center',color='black')

fig.tight_layout()

#FigureCanvasTkAgg() creates a figure/canvas in Tkinter window to allow
#for addition of graphics
canvas = FigureCanvasTkAgg(fig,self)

def clicking(event):
   xdata = event.xdata
```

```
    ydata = event.ydata
    #Converts the x and y data to coordinate values

    lonpt,latpt =m(xdata,ydata,inverse=True)
    onclick(event,lonpt,latpt)



#.mpl_connect connects with a function
canvas.mpl_connect('button_press_event', clicking)

#NavigationToolbar2TkAgg creates toolbar for image image manipulation
toolbar= NavigationToolbar2TkAgg(canvas,self)
toolbar.update()


canvas.show()
#.get_tk_widget() fits to canvas onto the window
canvas.get_tk_widget().pack(side=tk.BOTTOM,fill=tk.BOTH,expand=True,anchor='s')
scale = tk.IntVar()


label = tk.Label(self,text='Please select scale: ',font=NORM_FONT)
label.pack(side='left',pady=5,padx=5)

#tk.Entry() is a entry widget which allows displaying simple text.
entry_box = tk.Entry(self,textvariable=scale,width=10,bg='lightgreen')
entry_box.pack(side='left',padx=10)


def get_entry():
    '''Calls scale.get() function to read user inputted coordinate
    data'''
    global SCL
    SCL = scale.get()

#tk.Button creates a button widget
select_button = tk.Button(self,text='Ok',command=get_entry)
select_button.pack(side='left')


#tk.DoubleVar() a value holder for float variables
lat = tk.DoubleVar()
lon = tk.DoubleVar()
cityname = tk.StringVar()
```

```
def get_latlonentry():
    latse = lat.get()
    lonse = lon.get()
    Search_latlon(latse,lonse)

def getcityname():
    x = cityname.get()
    City_Coords(x)

label_2 = tk.Label(self,text='Please select latitude and longitude: ',font=NORM_FONT)
label_2.pack(side='left',pady=5,padx=5

entry_box_2 = tk.Entry(self,textvariable=lat,width=5,bg='lightgreen')
entry_box_2.pack(side='left',padx=5)

label_3 = tk.Label(self,text='N',font=NORM_FONT)
label_3.pack(side='left',pady=5,padx=1)

entry_box_3 = tk.Entry(self,textvariable=lon,width=5,bg='lightgreen')
entry_box_3.pack(side='left',padx=5)

label_4 = tk.Label(self,text='E',font=NORM_FONT)
label_4.pack(side='left',pady=5,padx=1)

select_button_3 = tk.Button(self,text='Ok',command=get_latlonentry)
select_button_3.pack(side='left')


#     label_5 = tk.Label(self,text='              Select desired city',font=NORM_FONT)
#     label_5.pack(side='left',pady=5,padx=1)
#
#     entry_box_4 = tk.Entry(self,textvariable=cityname,width=5,bg='lightgreen')
#     entry_box_4.pack(side='left',padx=5)
#
#     select_button_2 = tk.Button(self,text='Ok',command=getcityname)
#     select_button_2.pack(side='left')
#



#calls Search_latlon() to search for coordinate data


#Button which once pressed calls the function to search for the
```

```
        #most relevant coordinate points from the wavedata database.
        #opens a popup of the user's data

app = GlobalWaveDataApp()

app.geometry('1920x1080')
app.mainloop()
```