

# POLITECNICO DI TORINO

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATIONS

**Master of Science in Mechatronic Engineering**

*2021/2022*

Final thesis

## ***A collaborative learning strategy for model-free control of an array of wave energy converters***



**Supervisors:**

prof. Giovanni Bracco

Edoardo Pasta

**Author:**

Taylor Veale

## *Abstract*

Ocean energy is an abundant but relatively unexploited renewable energy source which has the potential to become one of the key players in the upscaling of global renewable energy production for the near future.

Despite this huge potential, ocean energy technology and especially wave energy technology is still considered to be immature compared to other renewable energy technologies.

One of the main goals to be achieved is to reduce the levelized cost of electricity (LCOE) coming from wave energy converter devices in order to make them economically competitive with respect to other more established renewable energy sources. To achieve this, one of the main areas of focus in recent years has been to develop and optimise control strategies to improve the efficiency of the energy conversion process.

The main challenges that wave energy converters (WECs) face, stem from the irregular reciprocating nature of the energy source, making the design of the control strategy, the WEC itself, and any modelling of the WEC-wave interaction extremely challenging.

A large portion of the most popular control strategies adopted on wave energy converters rely on a model-based control strategy to determine the optimal control action to be taken. The control action is usually optimised over a predefined range of wave conditions which can be grouped within a single statistical description of the current sea state by using parameters such as the significant wave height  $H_s$  and the wave energy period  $T_e$ . Although these control strategies may give good results, they are inevitably affected by modelling errors and uncertainties, together with a control which is not optimised on a wave-by-wave basis.

In this thesis, a model free control strategy for an array of heaving point absorbers is explored. A model free control approach was chosen since it allows to neglect the device model and wave interaction modelling, which in turn allows to avoid modelling errors and to directly develop the control strategy using data obtained while at sea.

The proposed strategy involves an initial online optimisation of the control parameters of a reactive control law using genetic algorithms to map the point absorber array to a population of individuals within a metaheuristic optimization framework. This allows the single point absorbers to collaborate and learn from one another to reach the common goal of finding the optimal control parameters for each discrete sea state encountered.

After the initial sea-state-based model free collaborative optimisation has reached satisfactory results, a secondary mechanism based on machine learning through neural networks is used to try and learn interdependencies between the discrete sea states and the relative optimal control parameters to achieve a continuous control, no longer dependant on a statistical description of the sea state but based on direct force measurements on the heaving point absorbers. In this framework,

the data collected during the initial optimization using genetic algorithms is then used to train the neural networks so to output a continuous control command.

Preliminary simulation results show that an array of point absorbers using a genetic algorithm based collaborative optimisation is able to achieve control parameters close to the theoretical optimal ones within only a few days from deployment, while the neural networks show comparable performance, indicating that with further tweaking of the learning procedure, superior results may be obtainable.

## Contents

<i>Abstract</i> .....	2
1 – Wave energy and wave energy converters, an introduction.....	12
1.1 Introduction.....	12
1.2 Wave energy converter technologies and classification .....	14
2 – Wave energy .....	21
2.1 The global wave energy resource.....	21
2.2 Wave physical description .....	22
2.3 Spectral characterization of sea states.....	24
2.3.1 Idealized wave spectra .....	25
2.4 Sea state characterizing parameters and wave power density .....	28
2.5 Characterizing Ocean sites.....	30
2.5.1 Temporal characteristics .....	30
2.5.2 Directional characteristics.....	31
2.5.3 Spectral characteristics.....	32
2.5.4 The scatter diagram .....	32
2.6 Generating Sea states in MATLAB .....	34
3 – Point absorbers.....	35
3.1 Point absorber technology.....	35
3.2 Power take off.....	36
3.2.1 Hydraulic PTOs.....	37
3.2.2 Mechanical PTOs.....	38
3.2.3 Direct drive PTOs .....	38
3.3 Point absorber model .....	39
3.4 Control of wave energy converters .....	40
3.4.1 Introduction.....	40
3.4.2 Discrete (slow) vs Continuous (fast) control.....	41
3.4.3 Optimal control of WECs.....	43
3.4.4 Resistive/bang-bang control strategies.....	45
3.4.5 Reactive control strategies .....	48

3.5 Challenges.....	50
3.6 Proposed control solution.....	50
4 – The genetic algorithm .....	53
4.1 Considerations to be made when designing a genetic algorithm .....	53
4.1.2 System stability for C and K values .....	56
4.2 Genetic algorithm design .....	58
4.2.1 Representation.....	58
4.2.2 Evaluation function (Fitness Function) .....	59
4.2.3 Population .....	60
4.2.4 Initialization procedure .....	61
4.2.5 Parent selection .....	61
4.2.6 Crossover (Recombination) .....	63
4.2.7 Mutation .....	64
4.2.8 Survivor Selection.....	65
4.3 Tuning the genetic algorithm .....	66
4.3.1 Introduction.....	66
4.3.2 Tuning notions .....	67
4.3.3 Tuning vs control .....	68
4.3.4 Tuning procedure layout .....	68
4.3.5 Algorithm performance and utility measure definition.....	70
4.3.6 Tuning results.....	73
4.3.7 Tuning procedure for physical application.....	75
5 – Optimization through the genetic algorithm .....	76
5.1 Sea state generation.....	76
5.2 Simulation results.....	77
5.3 Expected Annual Energy Production .....	84
6 – The neural network .....	86
6.1 Introduction.....	86
6.2 The Feed-forward neural network.....	88
6.2.1 The network structure .....	88

6.2.2 Training the feed forward network .....	90
6.2.3 Testing the feed forward network .....	93
6.3 The Long Short Term Memory Neural Network .....	100
6.3.1 The network structure .....	100
6.3.2 Training the LSTM network .....	102
6.3.3 Testing the LSTM network .....	102
7 – Conclusions and future work .....	113
Appendix A: Evolutionary algorithms .....	117
Appendix B: Neural Networks.....	140
Bibliography.....	167

## Table of Figures

Figure 1. 1 - Active and projected tidal stream and wave energy capacity beyond 2020 [3].	13
Figure 1. 2 - Wave energy converter technologies [1].	14
Figure 1. 3 - Wave energy converter location classification and relative wave power.	16
Figure 1. 4 - Oscillating water column devices.	16
Figure 1. 5 - Overtopping devices.	17
Figure 1. 6 - Attenuator devices.	17
Figure 1. 7 - Point Absorbers.	18
Figure 1. 8 - Oscillating water surge devices.	18
Figure 1. 9 - Submerged pressure differential devices.	19
Figure 1. 10 - Bulge wave devices.	19
Figure 1. 11 - Rotating mass devices.	20
Figure 2. 1 - Annual mean omni-directional power [9].	21
Figure 2. 2 - Sinusoidal wave parameters [10].	22
Figure 2. 3 - Ranges of applicability of wave theories according to Le Méhauté [11].	23
Figure 2. 4 - Irregular wave from sum of sinusoidal regular waves [12].	24
Figure 2. 5 - Typical spectrum graph.	25
Figure 2. 6 - Spectrum parameters [15].	26
Figure 2. 7 - Swell and wind wave classifications [12].	27
Figure 2. 8 - Wave significant height and energy period yearly variations from the FINO1b dataset.	30
Figure 2. 9 - Wave rose for significant wave height [15].	31
Figure 2. 10 - Average directional wave spectrum from SBF7-1A GPS wave buoy [16].	32
Figure 2. 11 - Scatter table produced from occurrences off the coast of Pantelleria [17].	33
Figure 3. 1 - Popularity of developed wave energy converter devices.	35
Figure 3. 2 - Example of one body point absorber model [22].	36
Figure 3. 3 - Block diagram of four different PTO configurations [23].	37
Figure 3. 4 - Example of a hydraulic PTO with hydraulic rectifier [7].	37
Figure 3. 5 - Example of a direct drive PTO [7].	39
Figure 3. 6 - Graphical scheme of a point absorber [26].	39
Figure 3. 7 - System variables under Latching control [29].	46

Figure 3. 8 - System variables under Clutching control [29] .....	47
Figure 4. 1 - Plot of the largest eigenvalue of A matrix with varying stiffness and damping parameters .....	57
Figure 4. 2 - Qualitative representation of deterministic tournament selection .....	63
Figure 4. 3 - View of Evolutionary algorithm performance in the 1980s [58]. .....	66
Figure 4. 4 - View of evolutionary algorithms with added problem specific knowledge [58]. .....	67
Figure 4. 5 - Control flow (left) and information flow (right) .....	69
Figure 4. 6 - Tuning results .....	74
Figure 5. 1 - Lookup table from Pantelleria site .....	76
Figure 5. 2 - Point absorber power plot.....	78
Figure 5. 3 - Examples of evolution of cost of best solution in two different sea states.....	78
Figure 5. 4 - First generation population.....	79
Figure 5. 5 - Fifth generation population .....	79
Figure 5. 6 - Fully converged population after 12 generations .....	80
Figure 5. 7 - Evolution of damping values.....	80
Figure 5. 8 - Evolution of stiffness values .....	81
Figure 5. 9 - Best constant damping values from genetic algorithm optimization.....	82
Figure 5. 10 - Best stiffness values form genetic algorithm optimization .....	82
Figure 5. 11 - Wave energy periods of the 14 generated sea states .....	83
Figure 5. 12 - Wave significant height of the 14 generated sea states .....	83
Figure 5. 13 - Expected Annual Energy Production over the 90-day simulation .....	84
Figure 6. 1 - Example of a simple neural network [67]. .....	86
Figure 6. 2 - Qualitative representation of the feed-forward neural network.....	91
Figure 6. 3 - Simulink model for point absorber and neural network simulation .....	93
Figure 6. 4 - Percentage mean power difference between using constant control parameters or neural network parameters .....	95
Figure 6. 5 – Max and min peak power difference when using constant control parameters or neural network parameters .....	96
Figure 6. 6 - Control signal of damping control parameter 'C' generated for sea state number 5 ....	97
Figure 6. 7 - Control signal of stiffness control parameter 'K' generated for sea state number 5 ....	97
Figure 6. 8 - Normalized damping, stiffness and heave force signal for sea state number 5 .....	97



Figure 6. 9 - Control signal of damping control parameter 'C' generated for sea state number 4....	98
Figure 6. 10 - Control signal of stiffness control parameter 'K' generated for sea state number 4 ..	98
Figure 6. 11 - Normalized damping, stiffness and heave force signal for sea state number 4 .....	99
Figure 6. 12 - An unrolled LSTM network layer [75]. .....	100
Figure 6. 13 - Basic LSTM unit .....	101
Figure 6. 14 - Simulink model for point absorber and neural network simulation .....	103
Figure 6. 15 - Percentage mean power difference between using constant control parameters or LSTM network parameters .....	105
Figure 6. 16 - Max and min peak power difference when using constant control parameters or neural network parameters .....	105
Figure 6. 17 - Control signal of damping control parameter 'C' generated for sea state number 1	106
Figure 6. 18 - Control signal of stiffness control parameter 'K' generated for sea state number 1	106
Figure 6. 19 – Close-up of the control signal of the damping control parameter 'C' generated for sea state number 1 .....	107
Figure 6. 20 – Close-up of the control signal of the stiffness control parameter 'K' generated for sea state number 1 .....	107
Figure 6. 21 - Normalized damping and heave force signal for sea state number 1 .....	108
Figure 6. 22 – Damping control signals generated by LSTM neural network for all 14 sea states	109
Figure 6. 23 – Stiffness control signals generated by LSTM neural network for all 14 sea states	109
Figure 6. 24 - Percentage power difference when using shifted and rescaled LSTM network damping and constant stiffness .....	111
Figure 6. 25 - Max and min power difference when using shifted and rescaled LSTM network damping and constant stiffness .....	112
Figure A. 1 - Pseudo code for a generic EA [58]. .....	119
Figure A. 2 - One point crossover [58] .....	128
Figure A. 3 - Two-point crossover [58] .....	128
Figure A. 4 - Uniform crossover [58] .....	129
Figure A. 5 - Simple arithmetic crossover [58]. .....	131
Figure A. 6 - Single arithmetic crossover [58]. .....	131
Figure A. 7 - Whole arithmetic crossover [58] .....	132
Figure A. 8 - Bitwise mutation. ....	133
Figure A. 9 - Non uniform mutation with gaussian distribution of step size .....	134
Figure A. 10 - Comparison of different stopping times [58]. .....	138

Figure B. 1 - Basic structure of a feed-forward neural network.....	140
Figure B. 2 - Working principle of an ANN neuron.....	141
Figure B. 3 - ReLU activation function .....	143
Figure B. 4 - Leaky ReLU activation function.....	143
Figure B. 5 - Sigmoid activation function.....	144
Figure B. 6 - Hyperbolic tangent.....	145
Figure B. 7 - Linear activation function.....	146
Figure B. 8 - Mean absolute error .....	148
Figure B. 9 - Mean squared error .....	149
Figure B. 10 - Root mean squared error.....	150
Figure B. 11 - Trajectory for steepest descent with small $\alpha$ .....	154
Figure B. 12 - Trajectory for steepest descent with larger $\alpha$ .....	154
Figure B. 13 - Trajectory for steepest descent with unstable $\alpha$ .....	154
Figure B. 14 - Examples of trajectories taken from SGD and simple GD.....	157
Figure B. 15 - Examples of trajectories taken from SGD and simple Mini-Batch GD.....	157
Figure B. 16 – Basic idea behind recurring neural networks and LSTM networks [78]. .....	161
Figure B. 17 - An unrolled LSTM network [78].....	161
Figure B. 18 - Repeating module or cell for a RNN [78].....	162
Figure B. 19 - Repeating module of a LSTM neural network [78].....	162
Figure B. 20 - List of symbols for LSTM module [78].....	163
Figure B. 21 – Forget gate and operation on old cell state [78]. .....	163
Figure B. 22 - Ignore gate operation [78]. .....	164
Figure B. 23 - Updating the cell state [78].....	164
Figure B. 24 - Creating the output vector $h_t$ [78]. .....	165



# 1 – Wave energy and wave energy converters, an introduction

## 1.1 Introduction

As energy demands grow, due mainly to an increasing population and an increasing demand for electrical energy to power an ever-increasing number of devices, new energy resources must be exploited if such energy demands need to be met.

Predictions suggest that most of the population growth will be in non-OECD countries, *i.e.*, presently under-developed and with opportunities to build new energy infrastructures. This is thus a great opportunity to create new energy infrastructures which heavily rely on renewable energy sources to supply the required energy.

With an increasing pressure by worldwide governments to reduce greenhouse gas emissions and to cut back on fossil fuels, the energy sector, which is responsible for two thirds of the global greenhouse gas emissions [1], needs to transition to renewable energy sources to help combat climate change.

Focusing on the energy sector may be one of the easiest ways to enforce the required changes. Other sectors such as transport and household heating will be harder to tackle since the pollution sources are more distributed and, because of an obvious human factor involved in the equation, change will take a considerable amount of time.

According to the renewable capacity statistics report produced by IRENA in 2021 [2], global renewable capacity as of 2020 is approximately 2800 GW, about a two-fold increase over the past 10 years. Whilst the major contribution to the total capacity still comes from hydropower, more than 80 per cent of all new electricity capacity added last year was from other renewable sources, with solar and wind accounting for 91 per cent of new renewables.

Whilst the rate of growth of ocean energy has been slower than expected (with the exception of the commissioning of the Sihwa Lake Tidal Power Plant in South Korea in 2011), predictions suggest that ocean energy may experience similar rates of rapid growth between 2030 and 2050 as offshore wind and solar experienced in the last 20 years.

Currently, a total capacity of 12.91 megawatts (MW) of tidal stream and wave energy is operational of which 2.31 MW from wave and 10.6 MW from tidal stream [3]. In both fields a significant number of new devices are being developed, with some units being able to reach 1 MW or higher. More capacity additions can be expected in the upcoming years. As of 2020, wave and tidal stream projects with total capacity of 2,83 GW were in the pipeline.

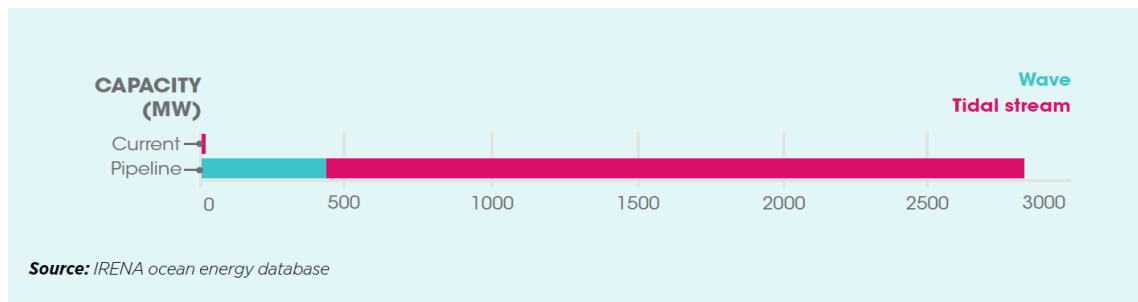


Figure 1. 1 - Active and projected tidal stream and wave energy capacity beyond 2020 [3].

Ocean energy is an abundant, yet relatively unexploited renewable energy source, which can be tapped using offshore technologies including offshore wind, wave energy, tidal stream and tidal barrage. Globally, 40% of the population live within 100 kilometres from the coast [4]. An obvious choice as a main energy source used to provide for such communities would be to use some sort of offshore renewable. Ocean energy is an abundant and greatly untapped resource which holds enough potential to meet the current and projected global electricity demand well into the future. Based on an analysis performed by IRENA in 2020 [3], the global potential ranges from 45.000 TWh to above 130.000 TWh annually.

At least theoretically, ocean energy alone has the potential to satisfy more than twice the current global electricity demand, but the benefits that offshore renewables can provide are not limited to the energy sector alone. Energy harnessed from the ocean has the ability to spark and drive a sustainable global blue economy while providing a reliable and local energy source to small island developing states (SIDS) and least developed countries (LDCs) bringing remarkable socio-economic benefits through job creations and promotion of energy independence.

Despite all the benefits listed above, offshore renewables are generally still emerging technologies, with most still in development stages. With the exception of offshore wind, offshore renewables like wave energy and tidal stream technologies are still in the research and development phase with a generally high degree of immaturity. The reduced pace at which the maturity of the ocean energy sector has grown could probably mainly be attributed to the presence of other renewable energy sources who outshined ocean energy due to larger funding, structured governmental policies and simply because they have been looked in to for a longer period of time. A concrete example provided by IRENA [5] shows how global funding for offshore wind amounted to 27,30 billion USD in 2006, while global funding for marine renewables only capped at a mere 0,02 billion USD in the same year. This clearly shows that although marine technologies hold great potential, additional funding and policy support is needed to enable greater performance, reliability and especially cost reduction to allow for the commissioning of larger commercial plants.

That said, an increasing number of companies, universities and investors outside the previously mentioned countries are sponsoring the development of ocean technologies, consequently substantial growth in the installed capacity is expected in the upcoming years.

## 1.2 Wave energy converter technologies and classification

Wave energy converters (WECs) are devices which harness the energy coming from ocean waves to generate electricity. The main categories of wave energy converters produce electricity by either exploiting the kinetic energy of the waves or the potential energy of the waves. The first kind usually use moving bodies which thanks to the imparted energy, create electricity. The second type uses the height of the wave itself to propel turbines thanks to overtopping or water column mechanisms.

As can be noticed, wave energy technologies have not yet converged towards a single type of technological design. Instead, numerous different WEC technologies are being pursued, and unlike tidal energy that aims at large-scale arrays, wave energy converters are currently following two parallel paths: one leading to the deployment of largescale devices above 1 MW and possibly towards deploying such devices in arrays, and the other aimed at smaller devices for specific offshore applications such as water culture.

Although ocean energy is globally available, European costal countries, together with Australia, Canada the United States and China, have been at the forefront of the ocean energy technological development, with the largest number of projects deployed and the most device manufacturers.

An overview of the different wave energy converter technologies is presented in the following table:

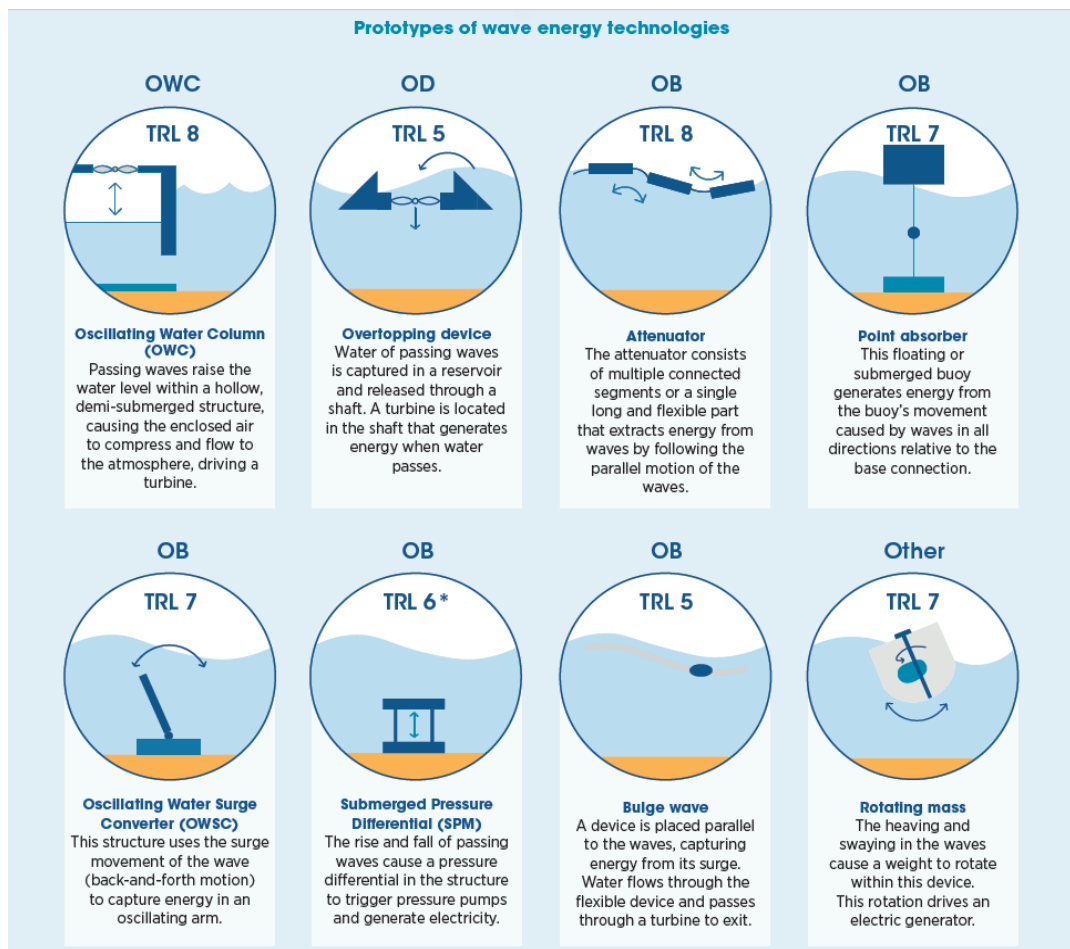


Figure 1. 2 - Wave energy converter technologies [1].

In figure 1.2, TRL stands for Technology Readiness Level, which indicates the maturity achieved by the given technology. Such scale goes from 1 to 9 where 1-3 represents a device in its research phase, 4-5 a device in its development phase, 6 the demonstration phase and finally 7-9 the deployment phase where the bottom range, i.e.7, represents a prototype stage while 9 represents a fully deployed stage.

Based on the classification in figure 1.2, a more in-depth description of different kind of devices and some examples of physically deployed devices will follow.

A first classification of wave energy converters can be based on their location of dispatchment [6] [7] [8]

- **Onshore** devices, as the name suggests, are devices which are rigidly connected to land. These converters can be placed in shallow water, integrated in a dam or fixed to a manmade structure such as a water breaker. The main advantages of such devices include their ease of maintenance and the lack of long power lines to connect them to the mainland grid. The main drawbacks for such devices instead are the lack of wave energy near the shoreline because of interaction with the seabed and the limited number of sites in which such devices can be installed. Typical examples of technologies adopted for such devices are oscillating water column and overtopping devices.
- **Nearshore** devices are devices which are installed in waters of moderate depth, usually a few hundred meters from the shore. Such structures can be either anchored on the sea bed or floating. One of the advantages such devices have compared to onshore devices is their placement in waters which naturally have higher power density. The main drawback might be their negative impact on a social level. Because of their closeness to the shore and because of their size, which does not go unnoticed, they might be seen as detrimental to the costal landscape.
- **Offshore** devices are installed in deep waters far from the coast. These devices are most often floating devices equipped with a mooring system. The advantage of such positioning is the availability of waters with a high power density, allowing to potentially extract a large amount of energy. The disadvantages are quite obviously due to the distance from the shore, resulting in a long electrical connection line needed to connect to the mainland grid and in difficult deployment, maintenance, and monitoring. For such devices, reliability and sturdiness is of paramount importance in order to avoid excessive maintenance and to survive the high loads they might be subject to.

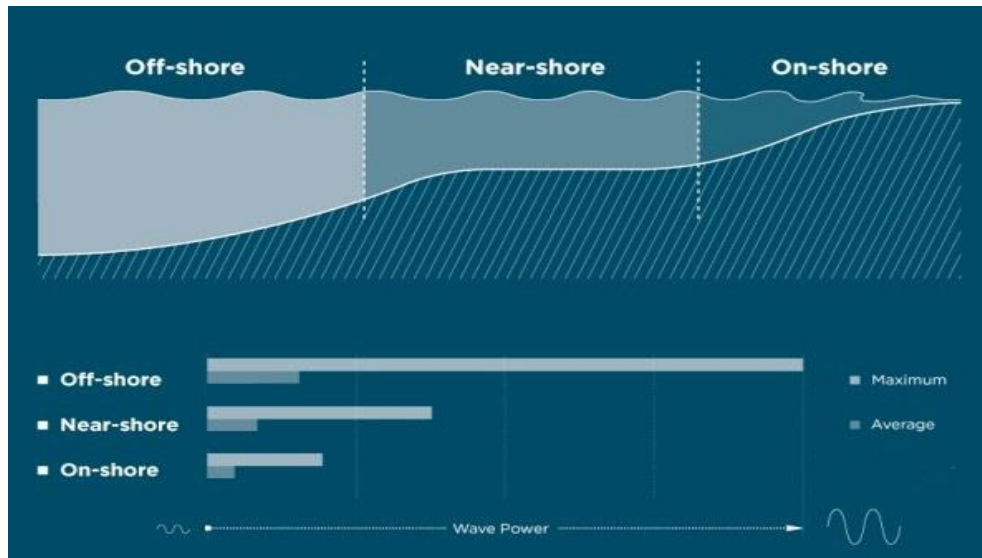


Figure 1. 3 - Wave energy converter location classification and relative wave power

Another type of classification which can be made is based on the working principle of the device. This type of classification allows to better understand how the device works and which physical phenomenon it exploits to harvest energy from the wave motion.

- **Oscillating Water Column devices** are devices composed by a partially submerged hollow structure. Such structure is open to the sea below the water line, creating a chamber in which water can enter from the bottom. Wave motion cause water to enter from such opening and flow into the main chamber, creating a water column which rises and falls according to the wave motion. This rising and falling of the water column within the device causes to air above the water column to move across a turbine placed at the top of the device whose other end is exposed to atmospheric pressure. Usually the turbine is bidirectional, meaning it will rotate both when the air is being expelled from the chamber (water column rising) and when air is being sucked into the chamber (water column falling). These devices are usually onshore devices, but nearshore and offshore devices have also been conceived.



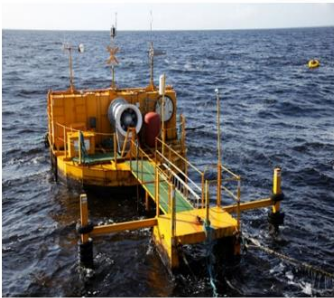
Location		
Onshore	Nearshore	Offshore
		
Limpet	Ocean Linx	OE Buoy

Figure 1. 4 - Oscillating water column devices



- **Overtopping devices** are devices which exploit a difference in potential energy which is artificially created by raising a volume of water above the ocean's surface. This can be achieved thanks to structures which mimic the wave action you might find naturally on a beach. As waves approach and hit the artificial beach they run up a ramp and into a storage reservoir which is at a higher level than the average surround sea level. From this reservoir the water then flows through a turbine back into the sea simply thanks to gravity. Examples of these devices have been developed for onshore, nearshore, and offshore applications.

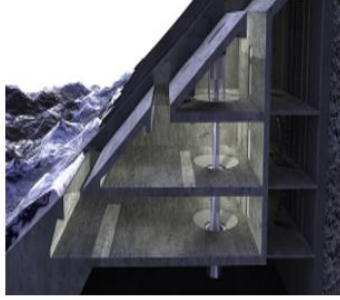


Location		
Onshore	Nearshore	Offshore
		
SSG	Wave Plane	Wave Dragon

Figure 1. 5 - Overtopping devices

- **Attenuator devices** are devices whose most significant dimension is oriented parallel to the direction of wave travel. Their working principle relies on wave motion to flex the joints connecting the main bodies to generate power. Often these devices are modular, allowing for multiple floaters to be attached in sequence. Some devices are also able to capture energy utilizing multiple degrees of freedom of motion between the attached bodies, relying on surge, heave and sway to capture energy. These devices are most commonly only found in nearshore and offshore applications because of their working nature.



Location	
Nearshore / Offshore	Nearshore / Offshore
	
Wave star	Pelamis

Figure 1. 6 - Attenuator devices

- **Point absorbers** are floating structures who usually, but not always, possess a small horizontal dimension with respect to their vertical dimension. In most point absorber designs, one end of the device is anchored to the seabed while the other end can freely float and move vertically as the wave crests and troughs move the device up and down. This reciprocal vertical motion between the two ends of the device is what is used to provide usable power. As can be noticed, the main motion used to generate power is thus the heaving motion. These devices are most prominently used in offshore applications since they can better exploit the more powerful waves to extract more energy, but nearshore applications do exist.



Location	
Nearshore	Offshore
	
Searaser	OPT Power Buoy

Figure 1. 7 - Point Absorbers

- **Oscillating water surge converter devices** use the surge motion of waves to swing back and forth. Most designs rely on a structure, hinged at its base which is able to pivot as the wave motion acts on it, causing the whole pivoted structure to swing back and forth. This motion resembles that of a large lever whose base is then linked to either a generator or a pump to move fluid or to drive a hydraulic motor. These devices are usually completely submerged and directly anchored to the sea floor. Such devices are usually deployed in nearshore applications or even in breakwater areas.

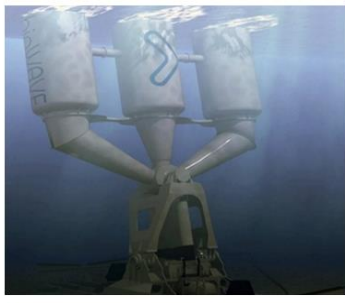

Location	
Nearshore	Nearshore
	
bioWAVE	Resolute marine energy

Figure 1. 8 - Oscillating water surge devices

- **Submerged pressure differential devices** work by exploiting the pressure differential that passing waves create. By placing the device on the seabed, the motion of the waves causes the sea level to rise and fall above the device, creating a pressure differential above the device which can be used to compress a pliable material such as an air bladder, thus moving a fluid which can be used to drive a turbine or some other power take off unit. These devices are typically located nearshore.

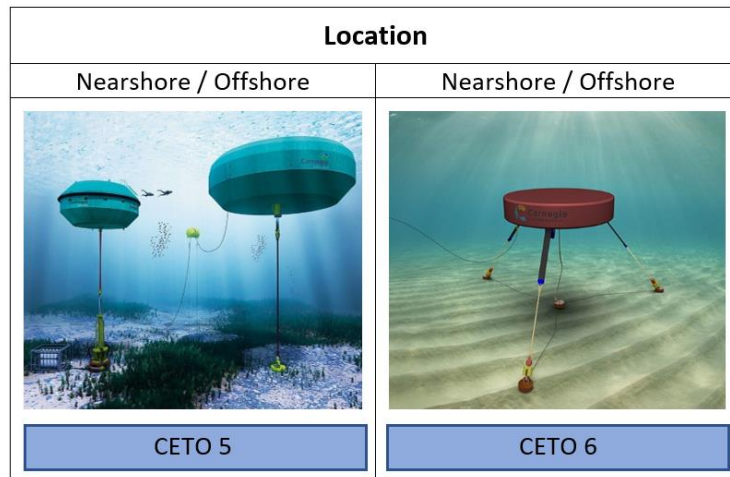


Figure 1. 9 - Submerged pressure differential devices

- **Bulge wave devices** use wave forces to push a fluid along a flexible channel and through a turbine. Such devices are aligned perpendicular to the wave and their shape can resemble that of a sea snake. Such devices operate very close to the water's surface and their design must be carefully tuned based on the expected sea conditions.

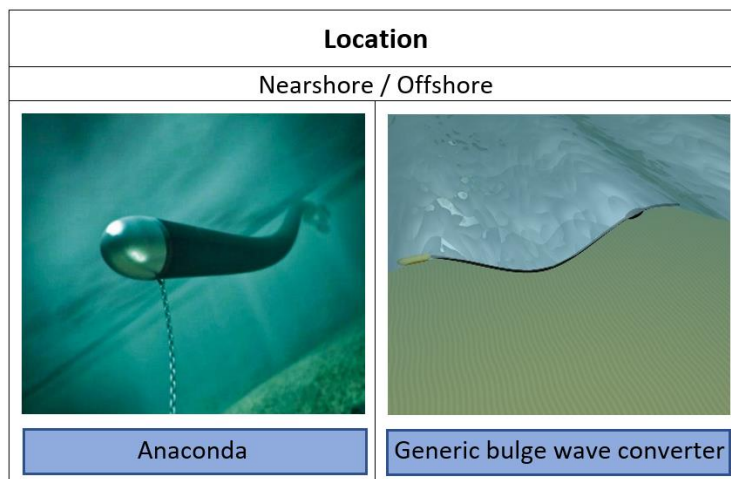




Figure 1. 10 - Bulge wave devices

- **Rotating mass devices** can be of two generic kinds. The first design includes a closed hull, which floats on the water's surface like a vessel, enclosing multiple eccentric masses which are able to rotate on different axis. With this design, if the device is excited in any direction from the incoming waves, one or more of the eccentric masses will rotate. This rotation is then used to generate energy.

The second kind of design uses a gyroscope instead of the eccentric masses. These devices take advantage of the gyroscopic precession effect which takes place if the hull of the vessel moves because of the interaction with the waves. This precession motion can then be used to extract energy from different PTO technologies.

These devices can be placed nearshore or offshore but are most prominently used offshore because of the more energetic waves.

Location	
Offshore	Nearshore / Offshore
	
ISWEC	Wello penguin

*Figure 1. 11 - Rotating mass devices*

- **Other:** It must be noted that although the above classification of wave energy converters based on their working principle does encapsulate most of the current designs being used, other designs exist which do not fit in any of the above categories.

The above classification considers the most common working principles but also some of the designs which are currently in the research phase but which show promising results and in which funding is currently being invested.



## 2 – Wave energy

Understanding the medium with which a wave energy converter has to interact with is a key requirement in designing efficient and reliable WECs. The study of sea behavior and of wave characterization is not only related to the wave energy industry and has been a research area for other industries for many years. Although wave-body interactions have also been a topic of study for any floating device, such as ships, the peculiarity about the wave energy industry is the need to study how this interaction might affect the power transfer from the sea to the wave energy converter device in order to extract the most amount of energy for each condition encountered.

In the following chapters an overview of the wave energy resource will be given.

### 2.1 The global wave energy resource

Waves on the ocean surface are primarily created by wind passing across the surface of the ocean. Initially, waves start as small ripples, but then grow in size thanks to the continuous energy provided by the wind. If the wind blows for long enough, the waves will eventually reach a limit past which they cannot grow any longer for that particular weather scenario because of internal energy losses. This stage is known as a fully developed sea.

The average wave power density is very diverse from location to location depending on seasonal average temperatures, winds, and many other local factors.

Figure 2.1 shows the global annual mean omni-directional wave power density.

As can be noticed, the most energetic regions occur in bands located in the north and southern hemispheres.

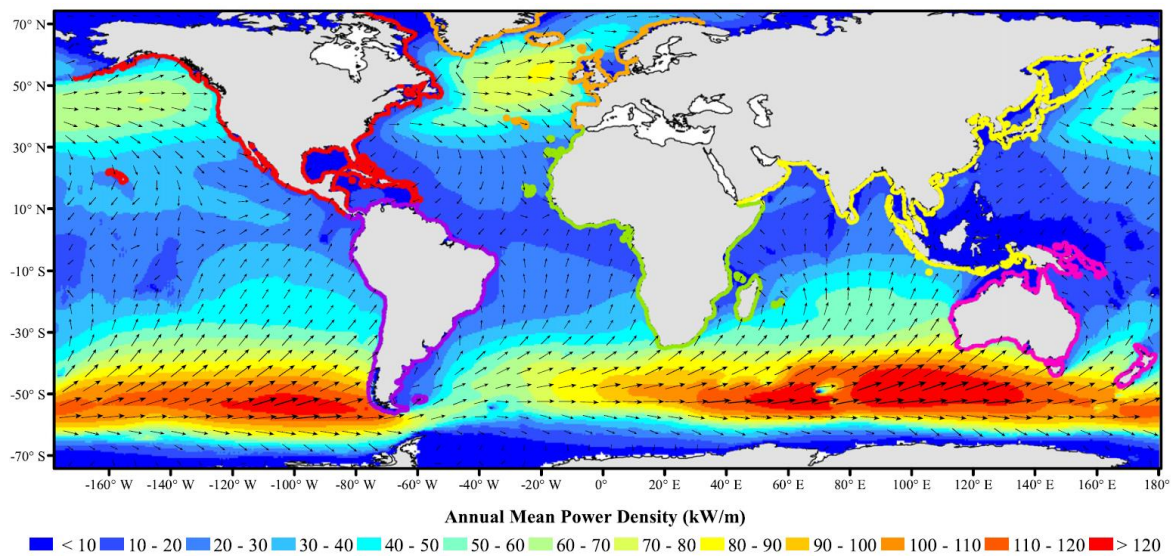


Figure 2. 1 - Annual mean omni-directional power [9].

These regions of high power-density are located for the most part far from coastal regions. This poses the additional problem of choosing the most optimal position for a WEC.

Although more energetic regions would favor a greater energy extraction, they would also need exceptionally long underwater cables to carry the produced electricity ashore. Furthermore, because of their distance from the coast, any repair operation would be extremely costly.

In comparison, one might argue that deploying an offshore WEC in a region closer to the coast, at the expense of a lower average wave power density, might be a better solution in the long run.

It must be noted that although average wave power density does give an insight on how much energy might be extracted by a WEC in any given location, because of the complexity and different working principles that each wave energy converter possesses, it is not possible to arbitrarily choose a single factor describing wave energy that can singlehandedly define which location is most suitable for energy extraction, no matter what WEC might be used.

## 2.2 Wave physical description

The most simplistic definition of a sea wave is a sinusoidal wave at the water surface with a characteristic wave height or amplitude and a wavelength associated with a corresponding wave period. This kind of wave is known as a regular linear wave.

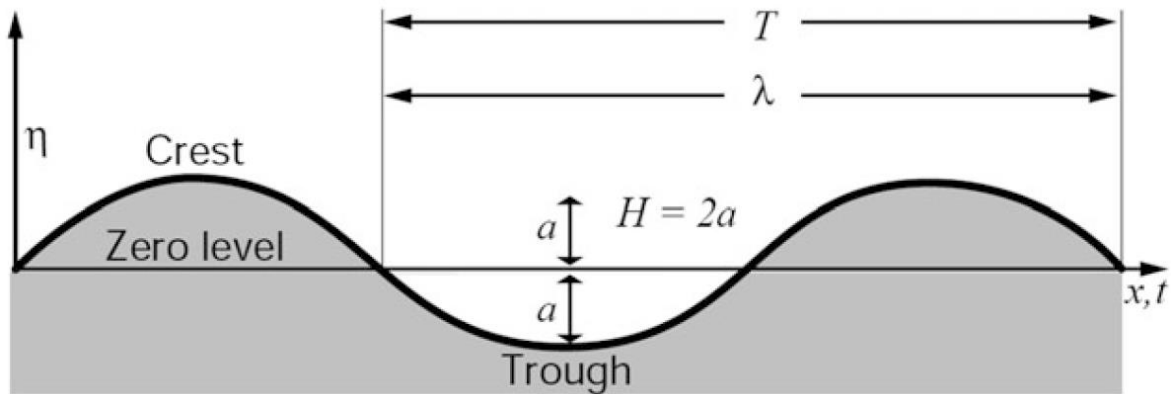


Figure 2. 2 - Sinusoidal wave parameters [10]

Other wave parameters can be produced by using the fundamental parameters described above.

$$s = H/\lambda, \text{ Wave steepness}$$

$$k = 2\pi/\lambda, \text{ Wave number}$$

$$\omega = 2\pi/T, \text{ Wave frequency}$$

In particular, the wave steepness can be used as a reference to distinguish between linear and non-linear waves and between which wave theory better describes the particular wave scenario.

Typically, if the wave steepness is smaller than 0.01, then linear wave theory can be used, but as the wave steepness increases, linear theory becomes less valid and other wave model theories should

theoretically be used. Figure 2.3 shows the validity of different wave theories based on the wave steepness (pictured on the vertical axis) and the relative water depth (on the horizontal axis) where  $h$  is the mean water depth and  $\tau$  the wave period.

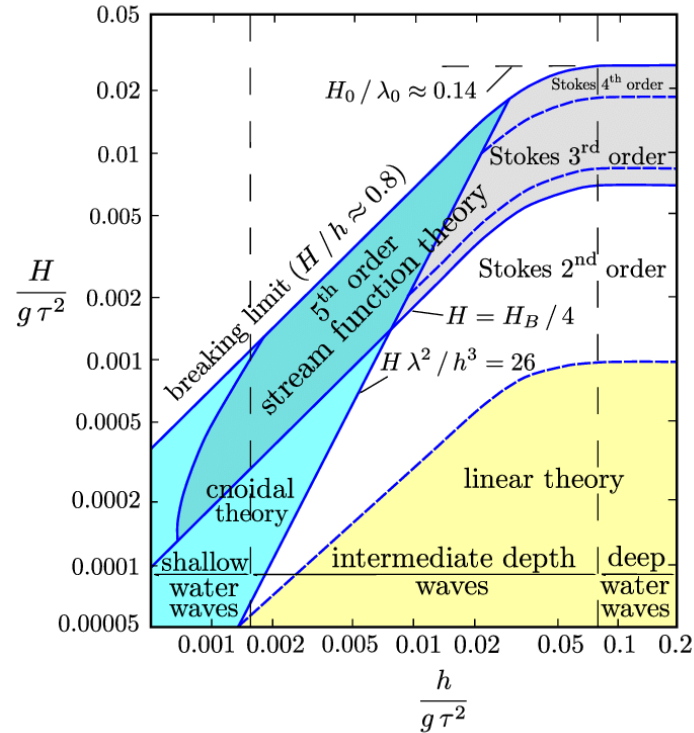


Figure 2. 3 - Ranges of applicability of wave theories according to Le Méhauté [11].

It must be noted that although these guidelines exist, it is extremely complex to apply any theory other than linear wave theory to irregular waves. It is thus common practice to use linear wave theory even beyond the bounds shown in figure 2.3.

Figure 2.4 shows qualitative examples of different regular waves based on different wave theories.

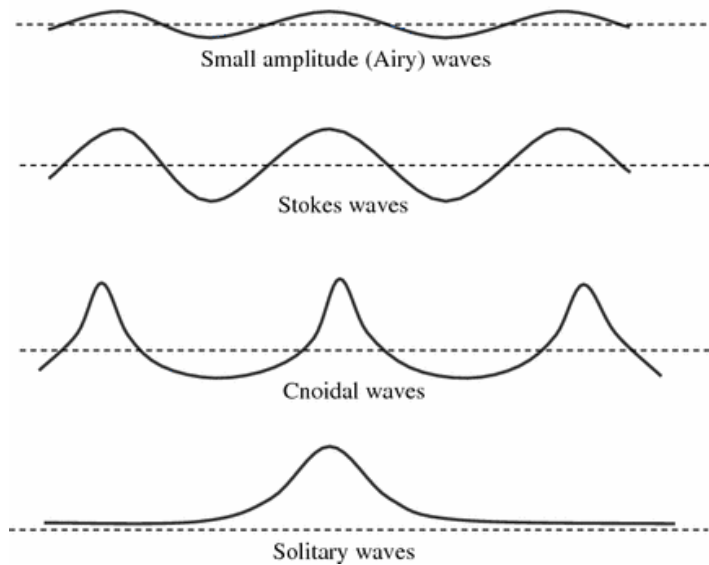


Figure 2.4 - Qualitative wave profiles according to different wave theories

## 2.3 Spectral characterization of sea states

The natural behavior of sea waves is irregular, often also referred to as random sea. The sea rarely shows a regular sinusoidal wave pattern, instead, a mixture of different contributions forms the final waves that we observe.

One of the most used characterizations of a given sea condition is the definition of the sea using a spectrum. This representation considers the variation in the elevation at the water surface (the waves) as a linear superposition of sinusoidal waves of different frequencies, amplitudes, and phases. This assumption is accompanied by considering the superimposed waves to have a random phase between each other.

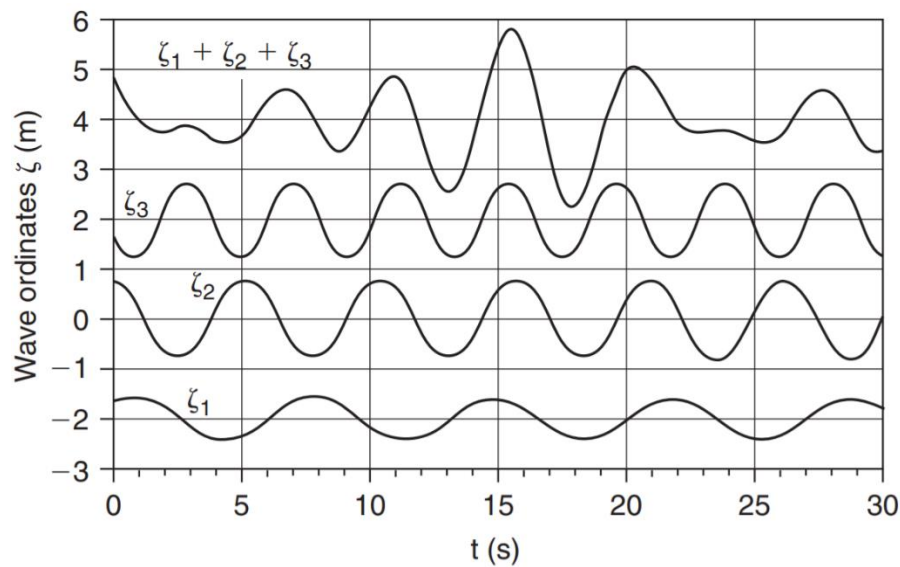


Figure 2. 4 - Irregular wave from sum of sinusoidal regular waves [12].

A generic irregular wave pattern  $\zeta$  can be decomposed into the partial regular waves that form it using Fourier analysis. If given a set of partial waves instead, an irregular wave can be created as the sum of the partial waves according to [12]:

$$\zeta(x, t) = \sum_{i=1}^n c_i \cos(k_i x - \omega_i t + \varepsilon_i)$$

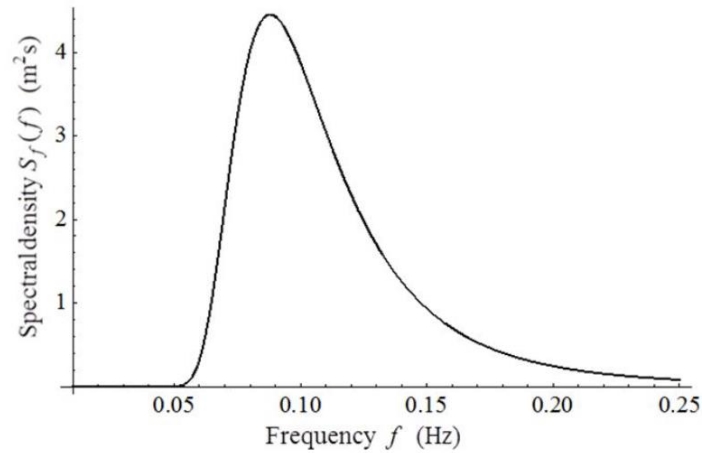
Where:

- $\zeta$  Wave ordinate expressing surface elevation
- $i$  Number of wave component
- $c_i$  Amplitude of the  $i^{\text{th}}$  partial wave
- $\omega_i$  Frequency of partial wave
- $x, t$  Direction of progress, time
- $k_i$  Wave number
- $\varepsilon_i$  Phase angle of partial wave

Where the phase  $\varepsilon_i$  is randomly distributed.



Given an irregular sea state, the variation of the wave energy with frequency, given the components that for such irregular sea state is called wave spectrum.



*Figure 2. 5 - Typical spectrum graph*

The wave energy spectrum gives an understanding of the distribution of energy across the different frequencies composing the irregular sea condition. This can give an insight of what are the predominant frequencies in the current sea condition and how narrow or large banded is a given sea state. This is especially important during the design phase of a WEC in order to match the design characteristics with the desired deployment location.

### 2.3.1 Idealized wave spectra

In an attempt to characterize sea conditions given certain external meteorological conditions and predefined assumptions, different standardized wave spectrum equations have been developed.

These equations are used to create wave spectrums under very specific assumptions, and thus do not perfectly represent all possible sea conditions.

One of the most commonly used spectrums is the Pierson-Moscowitz spectrum [13]. This spectrum was developed under the assumptions that the wind has been blowing across a sufficiently large and deep body of water for enough time to create a fully developed sea. Under these assumptions, the proposed spectrum now only depends on the wind speed at 19.5m above the water surface.

This idea was furtherly refined by Hasselman et al. after analyzing data collected during the Joint North Sea Wave Observation Project (JONSWAP) [14].

The produced spectrum was an extension and refinement on the Pierson-Moscowitz spectrum for conditions in which the sea is not fully developed and for limited fetch length.

$$S(\omega) = \frac{\alpha g^2}{\omega^5} \exp \left[ -\beta \left( \frac{\omega_p}{\omega} \right)^4 \right]$$

PM spectrum

$$S(\omega) = \frac{\alpha g^2}{\omega^5} \exp \left[ -\frac{5}{4} \left( \frac{\omega_p}{\omega} \right)^4 \right] \gamma^{\exp \left[ -\frac{(\omega - \omega_p)^2}{2\sigma^2 \omega_p^2} \right]}$$

JONSWAP spectrum

PM spectrum	JONSWAP spectrum
$\alpha = 0.0081$	$\alpha = 0.076 \left( \frac{U_{10}^2}{Fg} \right)^{0.22}$
$\beta = 0.74$	$\omega_p = 22 \left( \frac{g^2}{U_{10} F} \right)^{1/3}$
$\omega_p = \frac{g}{U_{19.5}}$	$\gamma = 3.3$
	$\sigma = \begin{cases} 0.07\omega \leq \omega_p \\ 0.09\omega > \omega_p \end{cases}$

Figure 2. 6 - Spectrum parameters [15].

Where:

- $S(\omega)$  Spectral variance density
- $\omega_p$  Peak frequency
- $\omega$  Wave component frequency
- $g$  Gravitational acceleration
- $U_{19.5}$  Wind speed at 19.5 meters above water level
- $U_{10}$  Wind speed at 10 meters above water level
- $F$  Fetch length
- $\gamma$  Peak enhancement factor

The above discussion, and the spectrum in figure 2.6 were only referring to sea states that have been generated by wind blowing in only one direction. However, there might be situations in which multiple sources of wind act on the same location, or where swell and local wind conditions are relatively uncorrelated. In these cases, spectra containing multiple peaks can occur.

Examples of how such sea states can be classified are given in figure 2.7.

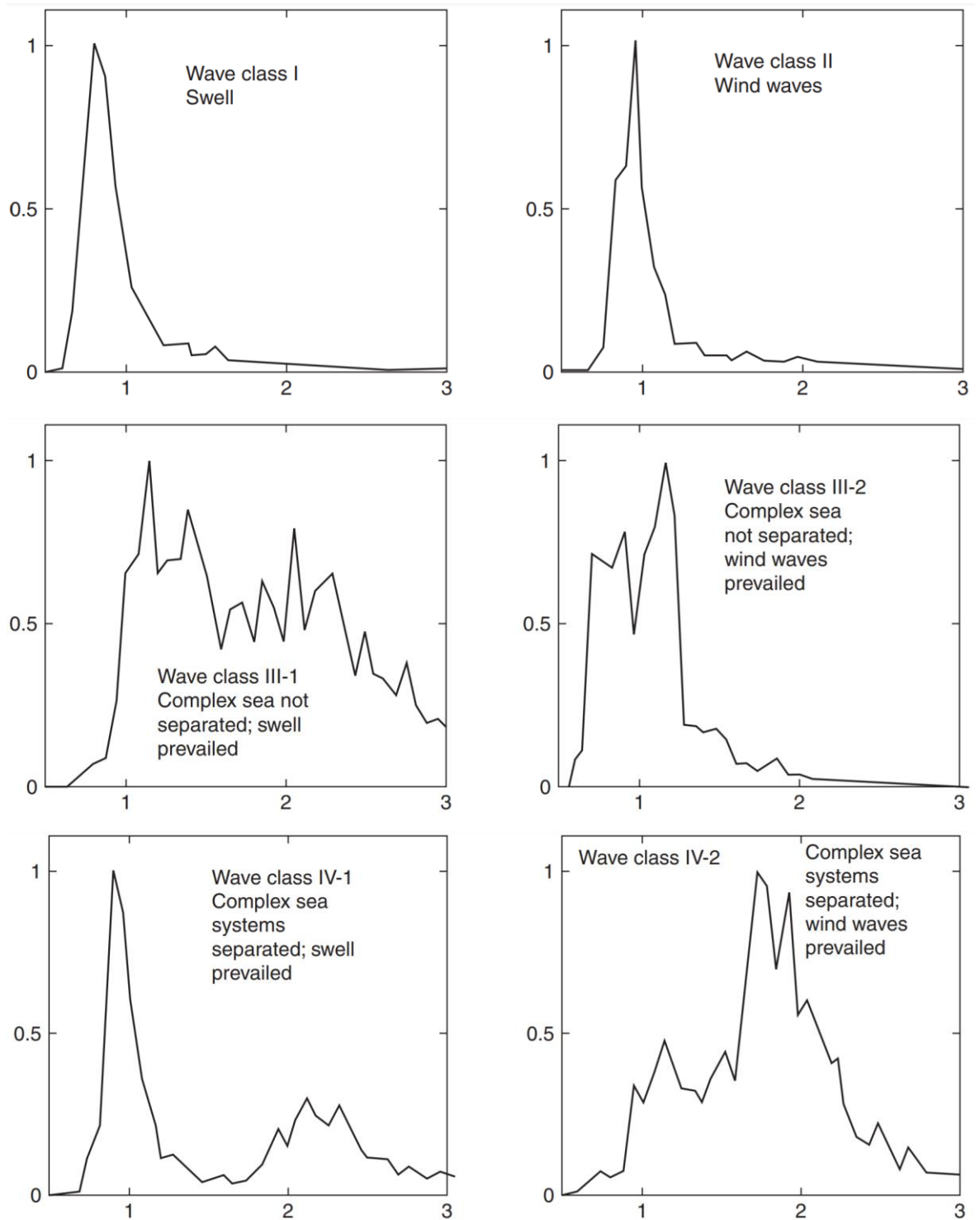


Figure 2. 7 - Swell and wind wave classifications [12].

The variety of wave spectral shapes presents a big challenge when designing and controlling a WEC since both the design and the control strategy could be much easier to develop if a narrow-banded sea state was always present, allowing for optimal design for a small range of frequencies.

## 2.4 Sea state characterizing parameters and wave power density

Sea states can be generically described by using two important parameters, which in some sense represent the wave height and the wave period.

Before measuring devices allowed for a precise local and statistical measurement of the sea state, the wave height was historically based solely on the direct observation of an experienced observer.

This height was named Significant Wave Height, symbolized by  $H_s$ .

When technology allowed for a precise measurement of the water surface elevation, this method was replaced by a more reliable and unbiased methodology. Because of the existing records of the significant wave height, the new method had to be consistent with the historical reports, thus producing a measurement equivalent to the  $H_s$ .

A good analogy to the old method was found to be given by considering the height of the third highest waves considering a time series record of the measured surface elevation. To distinguish this method from the previous one, the significant height was marked as  $H_{1/3}$ .

$$H_{1/3} = \frac{1}{\frac{1}{3}N} \sum_{m=1}^{\frac{1}{3}N} H_m$$

Where  $N$  is the total number of measurements of the wave height and  $H_m$  is the individual height measurement.

The third, and most commonly used measurement of the significant wave height is represented by a measurement based on the wave spectrum, denoted with  $H_{m0}$ .

$$H_{m0} = 4 \sqrt{\int_0^{\infty} S(\omega) d\omega}$$

Where  $S(\omega)$ , is the spectral variance density previously used when defining the sea state spectrum.

This method of measuring the significant wave height is useful since it's directly related to the average wave power density.

The power of a given sea state can be initially described by considering a single wave component at a time. Using linear superposition, it is then possible to find the total average wave power density.

Considering a single wave component, the wave power is:

$$J(\omega) = \rho g S(\omega) \cdot C_g(\omega)$$

Where  $C_g$ , is the velocity at which the energy is propagating known as the group velocity.

$$C_g(\omega) = \frac{1}{2} \frac{\omega}{k(\omega)} \left( 1 + \frac{2k(\omega)h}{\sinh 2k(\omega)h} \right)$$

Using linear superposition, the average wave power density can be defined as

$$J = \int_0^\infty \rho g S(\omega) \cdot \frac{1}{2} \frac{\omega}{k(\omega)} \left( 1 + \frac{2k(\omega)h}{\sinh 2k(\omega)h} \right) d\omega$$

The other important parameter mentioned which is used to describe a sea state is the wave energy period which can be defined as

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

Where  $m_n$ , are the moments of the spectrum defined as

$$m_n = \int_0^\infty S(\omega) \omega^n d\omega$$

Using the definitions of wave energy period and significant wave height, the omnidirectional wave power in deep water  $J$  can be defined as

$$J = \frac{\rho g^2}{64\pi} H_{m0}^2 T_e$$

Additionally, a final additional consideration may be made regarding the directionality characteristics of the sea state. The directionally resolved wave power density  $J(\vartheta)$  defines the wave power in a particular direction and is an important parameter for directional WECs and for wave farm deployments.

A directional wave spectrum  $S(\omega, \varphi)$  can be used to calculate  $J(\vartheta)$  as

$$J(\vartheta) = \rho g \int_{-\pi}^{+\pi} \int_0^\infty S(\omega, \varphi) C_g(\omega) \cos(\vartheta - \varphi) \delta \cdot d\omega \cdot d\varphi$$

$$\begin{cases} \delta = 1, & \cos(\vartheta - \varphi) \geq 0 \\ \delta = 0, & \cos(\vartheta - \varphi) < 0 \end{cases}$$

The above parameters used to describe a given sea condition and the power that accompanies such state are just some of the many possible parameters that exist to characterize a given sea state.

For the purpose of this thesis, the above description of a sea state using the significant height  $H_{m0}$  and the energy period  $T_e$ , in conjunction with the spectral characterization is enough.

It must be noted that for simplicity, the wave significant height will be referred to as  $H_s$ .

## 2.5 Characterizing Ocean sites

When considering deployment locations for a WEC, it is important to understand how the local sea conditions may affect the performance of the wave energy converter. Some of the main aspects to take into considerations can be considered to be the temporal, directional, and spectral characteristics of the local ocean sea states and how these characteristics may affect the average power generation.

### 2.5.1 Temporal characteristics

Temporal characteristics show how the local sea state changes throughout a given period of time. These changes are usually measured by considering the changes in significant wave height  $H_s$  and wave energy period  $T_e$ .

In general, it can be stated that, for a given average wave power, a wave climate which is more consistent in time is desirable because there will be less need to overengineer the WEC components to withstand extreme sea conditions. Furthermore, the working conditions in which the WEC will be working in most of the time will be the conditions for which it was designed to have greatest efficiency.

The temporal characteristics can be related to different time windows such as daily, monthly or seasonal variations in the sea conditions. Each time span considered will have its own impact on the WEC energy generation, with longer time span variations usually being more predictable than short ones, allowing for possible preemptive measures to be taken.

An example of temporal variations in the wave significant height and energy period is given in figure 2.8.

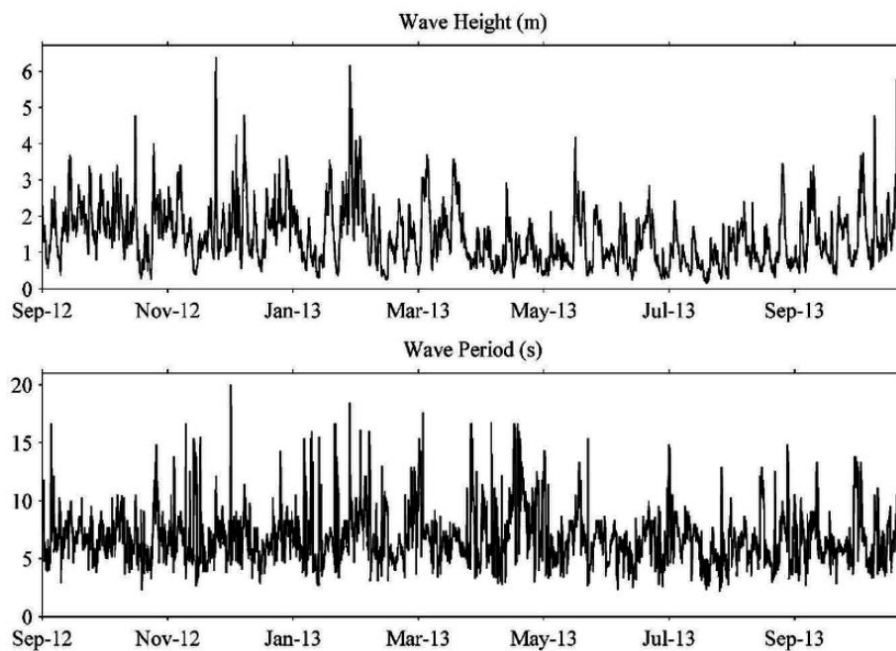


Figure 2. 8 - Wave significant heigh and energy period yearly variations from the FINO1b dataset.

### 2.5.2 Directional characteristics

Directional characteristics of the local sea state are extremely important when considering a deployment location for a directional WEC or when designing a WEC farm. The only situation in which directionality may not be of importance is for a single omni-directional WEC such as a heaving point absorber.

In general, an increase in the variation of the wave directionality of a given site will lead to a decrease in performance since it will be more likely that the directional dependent WEC or the wave farm will be often oriented in a less optimal direction.

Directional measurements can be of many different kinds. One example of a commonly used indicator is the wave rose shown in figure 2.9. This kind of graph can be used to represent average wave power or significant wave height measurements divided in different directional sectors. The length of each coloured sector radiating out in a given direction gives the percentage average occurrence of a given power or significant wave height in that specific direction.

An example of a wave rose is given in figure 2.10.

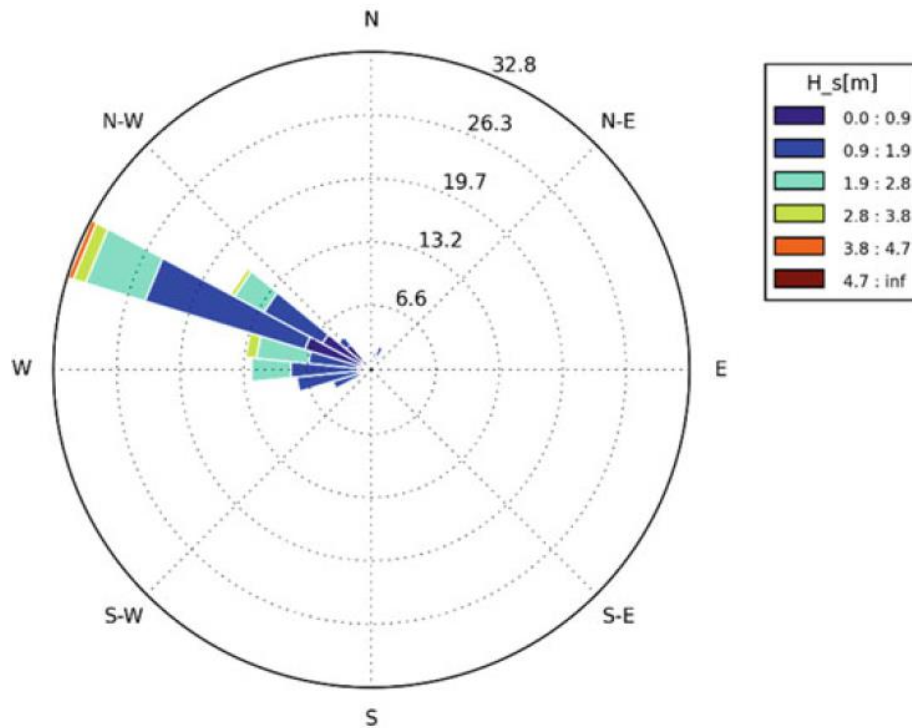


Figure 2. 9 - Wave rose for significant wave height [15].

### 2.5.3 Spectral characteristics

Spectral characteristics of wave climate add a frequency component to the measurements taken. Measurements based on the frequency at which such conditions occur may include wave directionality, wave significant height and other wave measurements varying across a frequency range. Spectral characteristics can be associated to spectral variations of a single sea state or spectral variations of a given site, considering all possible sea states together.

Spectral characteristics of wave climate are particularly important both when designing a WEC and also when analyzing a possible deployment site because the efficiency of most WECs is frequency dependent. Thus, for example, two sites having the same average power density may produce significantly different power outputs because of the different frequency ranges associated with each site.

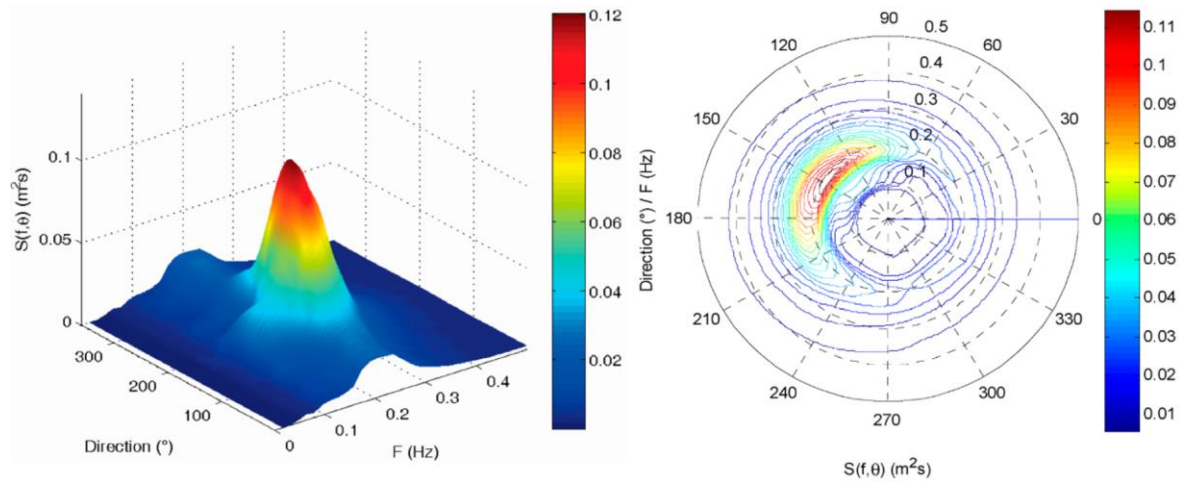


Figure 2. 10 - Average directional wave spectrum from SBF7-1A GPS wave buoy [16].

### 2.5.4 The scatter diagram

One of the most used diagrams to characterize a location is the scatter diagram. The scatter diagram consists of a grid of occurrences of sea states characterized by the significant wave height and wave energy period.

This kind of diagram allow to understand which sea conditions occur most often for the location under analysis but they are also prone to some potential issues.

Firstly, because of their discrete nature, sea states may vary significantly from one cell to the next, especially if the resolution is poor and the significant wave height is small, it is thus usually desirable to produce a scatter table with a good resolution, this although will take a larger number of samples and more precise measurements.



Another issue with the scatter table is that it does not give any indication on the directional distribution or spectral shape of the sea state contained within each cell which as discussed above are important factors when evaluating a location.

Although it does present some possible flaws, the scatter diagram is one of the most used tools to conduct a preliminary examination of a given site. Other techniques which take in to account frequency, temporal and directional dependencies are then used to get a better understanding of the sea behavior for the selected site.

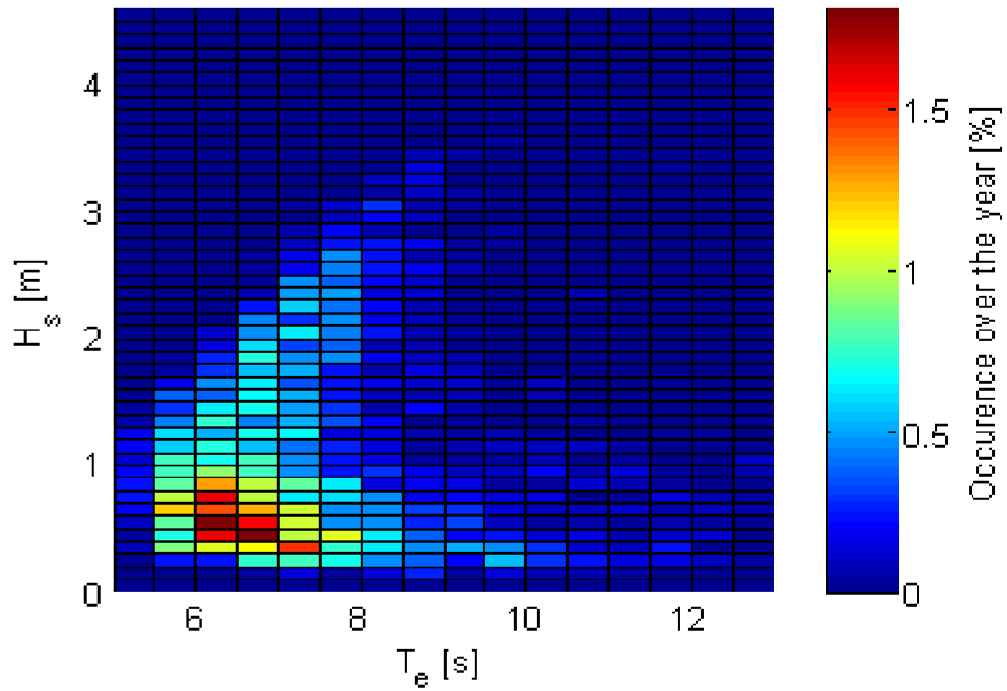


Figure 2.11 - Scatter table produced from occurrences off the coast of Pantelleria [17].

Summarizing, it is important to note that although wave climate characterizations as the ones presented above do give a good insight on the wave climate at a given location and on how well a given WEC might perform at such location, they cannot give a complete picture of how the WEC will behave and cannot be used to properly estimate the power generation of a WEC. However, these diagrams do provide an overview of the potential a site might have. This is especially true if these characterizations are coupled with a strong knowledge of how the given WEC works, allowing to choose the most appropriate information to analyze a give location.

When possible, it is always recommended to use the full time series of directional wave spectra to estimate the power generation of a WEC for a given site and time span [18].

## 2.6 Generating Sea states in MATLAB

In most sea conditions, ocean waves can be modelled as a stationary, zero-mean Gaussian process. This is true as long as the water is deep enough and the waves are not too extreme, which are the conditions in which most offshore WECs will find themselves in most of the time during their deployment. For this reason it is thus desirable to have a wave generation method which is able to reproduce the statistical properties of a Gaussian Sea condition.

For the purpose of this work, the sea states and related sea state forces used to simulate the behavior of the point absorber at sea were modelled following the *Random Amplitude Scheme* presented by M  rigaud and Ringwood in their work in [19].

The Random Amplitude Scheme (RAS) for generating finite length numerical simulations of Gaussian seas belongs to the class of wave superposition methods to produce free-surface time-series. This method consists in adding up harmonic sinusoidal components with random phases and component amplitudes which are chosen randomly with a variance which depends on the sea spectrum.

The discrete sequence generated when using RAS can be written as follows:

$$\eta_{ti} = \sum_{k=1}^{M/2} a_k \cos(2\pi f_k t_i) + b_k \sin(2\pi f_k t_i)$$

Where  $a_k$  and  $b_k$  are independent normally distributed random variables with zero mean and with variance  $S(f_k)\Delta f$ .

The sequence can also be equivalently written as:

$$\eta_{ti} = \sum_{k=1}^{M/2} A_k \cos(2\pi f_k t_i + \phi_k)$$

Where  $\phi_k$  is chosen randomly according to a uniform distribution in  $[0; 2\pi]$  while  $A_k$  follows a Rayleigh distribution with variance  $2S(f_k)\Delta f$ .

It is possible to show that RAS is able to reproduce the true statistical properties of a Gaussian Sea and is able to reflect how short-term WEC performance varies with respect to its long-term average allowing to realistically assess how the WEC power output may vary when measured over a finite duration. Because the basis of this work relies on finite measurements of the average WEC power output which will then be directly used as a driver to select the appropriate control strategy for each sea state, RAS provides the right tool for a probabilistic analysis of a finite length time domain simulation.

### 3 – Point absorbers

As mentioned in chapter 1, point absorbers are wave energy converter devices that have the ability to extract energy from the heaving motion of the incoming waves from all directions.

Currently, most attention worldwide is being focused on the point absorber design as it's one of the most promising designs thanks to its reduced complexity and ability to harvest energy from different wave directions. Additionally, this design is efficient, reliable, and also one of the first wave energy converter designs to be conceived.

One of the earliest recorded patents for a point absorber dates back to 1885 by Leavitt [20]. His design exploited wave forces through a heaving buoy connected to a gear system used to pump water. Not only is the point absorber design one of the most popular as suggested by the chart below [21], but some research suggests that it is currently one of the stronger candidates to be the standout WEC to harvest energy from highly energetic locations [22].

A more in depth overview of point absorber technology will follow in the next chapters.

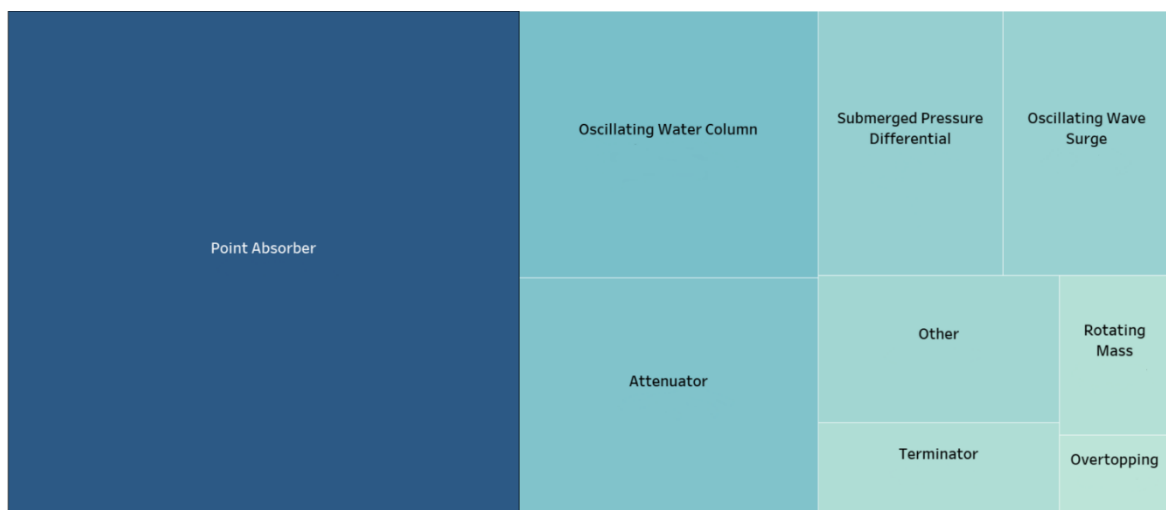


Figure 3. 1 - Popularity of developed wave energy converter devices

#### 3.1 Point absorber technology

Point absorbers are constituted by a spherical or cylindrical buoy which oscillates predominantly in the vertical direction with respect to a fixed reference which can either be the seabed, or another submerged body with much higher inertia with respect to the floater. In general, point absorbers present fewer moving joints than other WECs, thus reducing the complexity of the device. Additionally, if linear wave theory is assumed, point absorbers can be modelled with fewer degrees of freedom, which simplifies the computations and the problem layout in general.

The reciprocating motion between the buoy and the fixed reference is then used to harvest energy through a PTO placed between the buoy and the fixed reference as can be seen in the figure below.

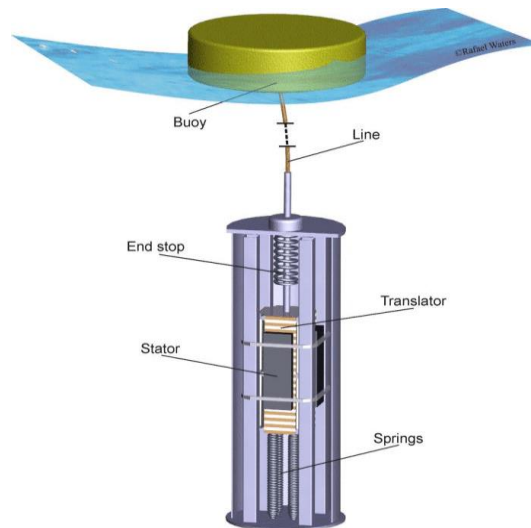


Figure 3. 2 - Example of one body point absorber model [23].

### 3.2 Power take off

The purpose of the PTO system is to transform the reciprocal motion created by the moving floater, into electrical energy suitable for being distributed on the main electrical grid. Because the electricity entering the main grid needs to meet certain standards, especially regarding the wave form, a significant challenge that the complete PTO system must overcome is to transform the irregular power input from the incoming waves into a smooth sinusoidal electrical power output.

This requirement is thus one of the main requirements to consider when designing a PTO for a wave energy converter.

PTO systems are also most often required to convert a slow motion accompanied high forces, coming from the interaction with the waves, into a fast rotational motion to drive an electric motor.

Throughout the years, efforts have been made to meet such requirements by including storage systems (mechanical, hydraulic, or electric) and rectifiers in the PTO design.

The main challenges a PTO has to face are mainly due to the intrinsic properties of the energy source. Ocean energy presents a high variability both in the short and in the long term which in turn means that the displacements, accelerations and forces induced may vary greatly over time. These large possible working ranges will induce different dynamic loadings on the structure, and it is thus mandatory for the PTO design to be robust and reliable, but at the same time, be able to be as efficient as possible in all sea conditions.

It is generally possible to recognize four types of PTO technologies as represented in the figure below [24]: hydraulic with hydraulic rectifier, hydraulic with electrical rectifier, mechanical with mechanical rectifier and direct drive.

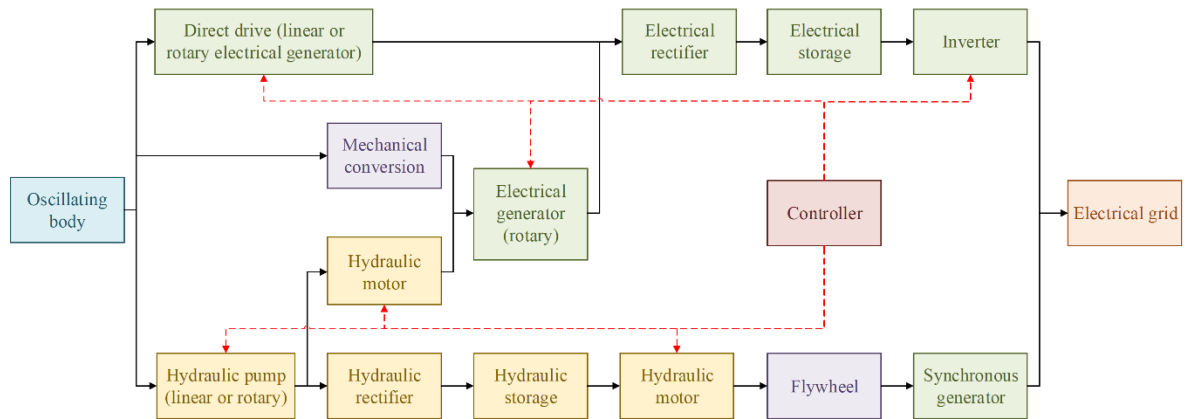


Figure 3. 3 - Block diagram of four different PTO configurations [24].

### 3.2.1 Hydraulic PTOs

Hydraulic converters are a popular solution to interface the external forces acting on the wave energy converter with the electrical generator since they are well suited to absorb energy coming from large forces with low frequencies and can provide good energy storage and rectification capacity.

The movement of the prime mover, the buoy in the case of a point absorber, is used to drive fluid through a hydraulic circuit with two parallel branches. Valves are used to ensure that the fluid moves always in the same direction within the circuit, no matter the direction of motion of the floater. High and low pressure accumulators are used to ensure a smooth hydraulic flow before the fluid reaches the hydraulic motor which then translates the energy into a fast rotation that can be used by an electrical rotary generator. A radial piston hydraulic motor is often used as it is capable of withstanding high loads for low velocity applications.

A depiction of a PTO hydraulic system follows below.

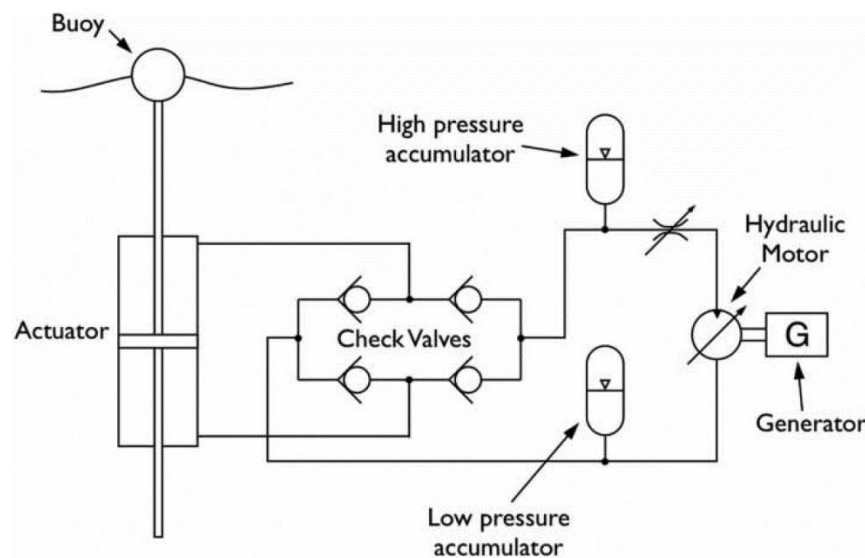


Figure 3. 4 - Example of a hydraulic PTO with hydraulic rectifier [7].

Hydraulic PTOs with electrical rectifiers use the hydraulic part of the system to transform the low velocities of the prime mover into higher velocities which can then be used to move a rotary electrical generator while the storage elements are instead represented by batteries

Particular considerations must be made regarding the possible issues when choosing a hydraulic PTO for a wave energy converter. Some of the main topics on which to reflect are:

- Fluid containment in the hydraulic system and possible environmental issues.
- Maintenance. Hydraulic systems are composed of numerous moving parts, where many are equipped with seals which will wear over time, producing the need for maintenance.
- The hydraulic circuit must be designed to work even in extreme conditions characterized by high forces and fluid pressures.

### 3.2.2 Mechanical PTOs

In mechanical PTOs the increase in speed which in hydraulic PTOs was achieved by a hydraulic motor is instead achieved through a mechanical conversion system such as a gearbox. This gearbox is then linked to a rotary electrical generator to produce electrical energy. A flywheel can be used as a mechanical accumulator to smoothen out power variations.

An advantage of this kind of system is its high efficiency, but because of the high number of cycles and forces that the mechanical conversion system has to bear, the reliability and robustness of such system is crucial.

### 3.2.3 Direct drive PTOs

Direct drive PTOs are able to directly convert the kinetic energy of the reciprocating motion of the prime mover straight into electrical energy through a linear generator with permanent magnets [25]. Because of the large forces and low velocities involved, the design and manufacturing of these devices is usually performed ad-hoc for the specific application, but advancements in the fields of power electronics and permanent magnets have made this solution attractive in the most recent years. Since the alternating wave motion is directly converted in to electricity, rectification must be carried out by a power electronics system before conversion to a sinusoidal waveform with fixed voltage and frequency can be performed.

An example of this technology is given in the following figure coming from [7].

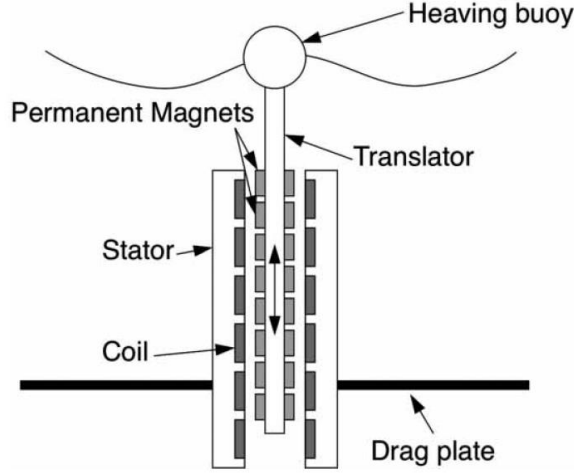


Figure 3. 5 - Example of a direct drive PTO [7].

### 3.3 Point absorber model

For this work, the considered point absorbers consist of simple heaving point absorbers constituted of 2[m] radius spheres with an internal mechanical PTO directly anchored to the seabed. This kind of point absorber was chosen since it represents a simple but realistic model on which to test the devised control strategy. A graphical representation of the point absorber model is shown in figure 3.6.

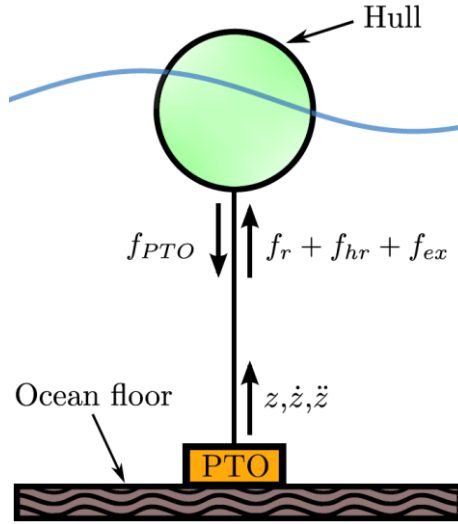


Figure 3. 6 - Graphical scheme of a point absorber [26].

The considered device is limited to extract power in the vertical direction only, meaning that we are considering a 1 Degree of Freedom (DoF) device. For this kind of vertical heaving point absorber the system dynamics can be modelled as [27]:

$$m\ddot{z} = f_r + f_{hr} + f_{ex} - f_{PTO}$$

Where  $z(t)$  is the device vertical displacement,  $f_r(t)$  is the radiation force,  $f_{hr}(t)$  is the hydrostatic restoring force,  $f_{ex}(t)$  is the external heave force, and  $f_{PTO}(t)$  is the controllable force exerted by the power take off.

The hydrostatic force can be written as:

$$f_{hr}(t) = -k_h z(t)$$

Where  $k_h$  is the hydrostatic stiffness constant of the device.

The radiation force instead can be expressed following the Cummins' equation as:

$$f_r(t) = -m_\infty \ddot{z}(t) - \int h_r(\tau) \dot{z}(t - \tau) d\tau$$

Where  $h_r$  is the impulse radiation response and  $m_\infty$  is the added mass at infinite frequency. The convolution term can be approximated by using an LTI system using proper identification techniques [28] which in turn lead to the following input-output state space representation:

$$\begin{cases} \dot{x}_r = A_r x_r + B_r \dot{z} \\ f_r = C_r x_r + D_r \dot{z} \end{cases}$$

Where  $x_r$  are the additional radiation states and the matrices  $A_r, B_r, C_r, D_r$  are the state-space matrices used to approximate the convolution term, identified using the FOAMM toolbox.

As stated in chapter 2.6 instead, the  $f_{ex}$  term has been modelled using the Random Amplitude Scheme (RAS) allowing a force representation which realistically depicts a Gaussian Sea state.

Finally, in a realistic scenario, WEC interactions might need to be considered for a compact array of devices. For this work the devices are considered far enough (at least 160[m] apart) so to not affect each other's dynamics.

## 3.4 Control of wave energy converters

### 3.4.1 Introduction

Wave motion is characterized by a broad frequency band that changes over both relatively short time spans (hours) and across larger time spans (seasons). This is an important consideration since wave energy converters are most efficient at absorbing energy when their natural frequency is close to the dominant frequency of the incident wave [29] [30].

In order to increase the power output of a wave energy converter, in response to the continuously changing wave spectrum, the behavior of the wave energy converter needs to be changed so that its frequency will be in resonance with the incoming waves. This can be done by using an appropriate control strategy.



The physical properties of the device, like mass and shape are difficult if not impossible to vary according to the incident wave. Instead, the behavior of the wave energy converter can be tuned by acting on the stiffness and damping coefficients of the system, which can be accessed through the power take-off.

The main purpose of control is to produce a wave energy converter which is as efficient as possible at capturing energy from the incoming waves, but control can also become useful in the event of extreme sea conditions.

In case of such an event, the device could switch to a safe mode in order to prevent any damage to the structure, ensuring it's survivability.

The design of a control strategy to be implemented on a WEC is usually performed using a model of the system. This is obviously true for model-based control strategies, but it can also be true for model free control strategies since, to both design and to validate the viability and performance of the implemented control strategies, a numerical simulation is needed which in turn involves a model of the WEC. This model will usually not perfectly reflect the dynamic behavior of the device because of approximations made when constructing the model (e.g., reducing the DOFs of the model) and because of unmodelled unknown dynamics.

The design of the control strategy can then be defined as the task of using the model to design a control function, together with its parameters, to satisfy and optimize some desired performance objective, usually energy capture maximization.

Different control strategies have been proposed, and in the following chapters some of the most important strategies will be described.

### 3.4.2 Discrete (slow) vs Continuous (fast) control

As stated in the previous chapter, optimal damping and stiffness coefficients with the goal of maximizing energy absorption depend on the incoming wave frequency [31].

Thus, it is desirable for the control parameters to adapt to the current wave conditions.

This is usually approached from two perspectives: a discrete, also known as “slow” control or a continuous, also known as “fast” control.

#### 3.4.2.1 Discrete control

Discrete control is currently one of the most used classes of control because of its simplicity.

It involves identifying discrete sea states determined by statistical measures of the wave amplitude and wave period. In particular, the significant wave height ( $H_s$ ) and the wave energy period ( $T_e$ ) are most commonly used as the identifying parameters of the sea state.

Once sea conditions have been adequately gridded, numerical simulations are run offline to determine the optimal control coefficients in each sea state by using a numerical model of the wave energy converter. These discrete control coefficients for each sea state in the grid are then stored in a lookup table.

With the wave energy converter deployed, the current sea state is determined by wave buoys placed in the vicinity of the WEC and the optimal control coefficients can simply be selected by using the lookup table.

Although this approach is rather simple, it has been shown to be effective compared to fixed damping control which has made it rather popular in the past because of its relatively good performance and inherent simplicity.

The main drawbacks of using discrete model-based strategies are caused by model accuracy and the discretization strategy.

Because optimal energy capture of a WEC can be achieved only if the WEC changes its control parameters on a wave-by-wave basis, the discretization strategy is clearly non optimal. That said, a discrete control strategy may vary widely in performance based on the information used to tune the device which in turn depends on how fine of a gridding was used to define the sea states. Additionally, it is also important to correctly measure and define which sea state the device is currently in so to correctly select the control parameters.

Finally, for sea conditions in which linear wave theory starts to break down, the highly nonlinear interactions will lead to suboptimal control parameter choices. This can be mitigated by either using a more robust or accurate model or by using model-free techniques

#### *3.4.2.1 Continuous control*

Differently from a discrete control strategy, continuous control aims at developing a control strategy which can adapt the control parameters in real time based on the current incident wave acting on the device. These strategies try to either directly measure or to estimate the current and future wave parameters characterizing the incident waves on the device. In this manner a precise and fine control can be applied based on the specific wave which is currently affecting the wave energy converter.

The measurements or estimations used usually regard either parameters which directly characterize the wave, such as period and amplitude, or which describe the forces acting on the device, such as heave, surge and sway forces.

Although this might lead improvements in the amount of energy harvested, this kind of control is usually accompanied by difficulties in measuring or predicting the exact wave profile which is currently affecting the exact position of the WEC.

### 3.4.3 Optimal control of WECs

In the case of an unconstrained point absorber in a sinusoidal wave, two conditions are necessary in order to achieve optimum energy absorption [32]:

- The velocity of the oscillator must be in phase with the dynamic pressure of the incident wave.
- The amplitude of motion of the oscillator needs to be tuned in order that the amplitude of the radiated wave from the oscillator is half that of the incident wave.

The first condition is related to the adjustment of the phase of the velocity of the oscillator to match that of the incoming wave and is thus often referred to as phase control.

The second condition instead can be met by adjusting the damping factor of the device in order to achieve maximum energy efficiency.

If the damping were to be set too high, then the motions are limited, and a low efficiency would be the result. If instead the damping were to be set too low, then little power would be absorbed and again this would result in low efficiency.

The above conditions can be summed up in the frequency domain as:

$$Z_{PTO}(\omega) = Z_i^*(\omega)$$

Where:

- $Z_{PTO}(\omega)$  is the frequency dependent PTO impedance
- $Z_i^*(\omega)$  is the complex conjugate of the WEC's frequency dependent intrinsic impedance

This is equivalent to saying that the maximum absorbed energy for an oscillating body in one mode is obtained by imposing that the intrinsic WEC reactance  $X_i(\omega)$  is cancelled out by the PTO reactance  $X_{PTO}(\omega)$  while the resistance  $R_{PTO}(\omega)$  must match the intrinsic resistance  $R_i(\omega)$ .

This is what is known as *impedance matching* [33].

This result gives rise to a number of important considerations to be made [34]:

- The above result is frequency dependent, meaning that there is an optimal impedance value for each frequency, which raises a problem on how to specify the PTO resistance for irregular waves which, by definition, are formed by a mixture of different frequencies.
- In some cases, the PTO system may need to *supply* power for some parts of the cycle (an analogy to reactive power). This adds additional design constraints on the PTO which needs to be able to facilitate bidirectional power flow and must also be designed to handle peak reactive power surges which may be greater than peak active power values. This condition puts the optimal control condition of the above equation in the category of *reactive control*.

- The optimal impedance matching control does not take in to account the physical limitations imposed by the WEC and PTO constructions. These limitations include force limitations on the PTO, displacement limitations and even electrical limitations.

To achieve impedance matching, different strategies have been proposed [30]:

- Complex conjugate control
- Phase and amplitude control

Complex conjugate control involves choosing  $Z_{PTO}(\omega)$  as equal to the intrinsic WEC impedance  $Z_i^*$ .

By considering the force to velocity model of a WEC as:

$$\frac{V(\omega)}{F_{ex}(\omega) + F_u(\omega)} = \frac{1}{Z_i(\omega)}$$

Where:

- $V(\omega)$  = Fourier transform of the velocity  $v(t)$
- $F_{ex}(\omega)$  = Fourier transform of the excitation force  $f_{ex}(t)$
- $F_u(\omega)$  = Fourier transform of the control force  $f_{PTO}(t)$

It is easy to see that a feedback of the measured buoy velocity is needed.

This represents a problem since the transfer function from velocity to force becomes *anticausal*.

This implies that the optimal force to be applied depends on future values of the buoy velocity which renders this approach impossible to implement in practice since the velocity itself depends on the applied force. Because of this, many control techniques are based on a realizable but suboptimal approximation of complex conjugate control.

The condition expressed in equation 3.1 can also be equivalently expressed as:

$$V^{opt}(\omega) = F_{ex}(\omega)/(2R_i(\omega))$$

This creates the foundation for *phase and amplitude* control.

The optimal velocity  $V^{opt}(\omega)$  is calculated thanks to a feedforward of the excitation force yielding an optimal velocity reference signal which can then be tracked by a controller with the restriction of having  $v^{opt}(t)$  in phase with  $f_{ex}(t)$ .

Again though, the excitation force transfer function  $H_v(s) = 1/(2R_i(s))$  becomes non causal. This problem can be approximately solved if the future excitation force can be predicted or by approximating the transfer function  $H_v(s)$  with a causal counterpart.

In the next chapters, some examples of commonly applied control strategies will be presented.

The presented strategies will be divided in two groups, *reactive control*, and *resistive/bang-bang control*.

### 3.4.4 Resistive/bang-bang control strategies

This first generic category of control strategies includes control strategies which do not require reactive power flow. Although this inherently causes the control strategy to deviate from what would be the optimal control model, they allow to design simpler PTOs which do not need to handle reversible power flows or high reactive power peaks.

#### 3.4.4.1 Resistive control

Resistive control consists of tuning the PTO force based on the value of the PTO velocity. The control variable in this technique is thus represented by the proportionality constant between the PTO velocity and the PTO force (the damping coefficient). The main drawback of using this technique is that only the amplitude and not the phase of the velocity can be controlled.

This technique is the simplest control technique which can be devised and can be seen as an approximation of impedance matching in which only the resistive part of the PTO impedance is used while the reactance is set to zero. This technique thus avoids the need for the PTO to be able to supply power but results in suboptimal control.

The PTO or machinery force can be written as:

$$F_{PTO}(t) = -R_{PTO} \cdot v(t)$$

Where,  $v(t)$  is the velocity measurement and  $R_{PTO}$  is the proportionality constant to be controlled.

The optimal frequency dependent passive resistive control law objective can be written as:

$$R_{PTO}(\omega) = |Z_i(\omega)|$$

For irregular waves, the value of  $R_{PTO}(\omega)$  is usually set to a fixed value for each sea state, creating a discrete control strategy.

#### 3.4.4.2 Latching Control

This control strategy was first introduced by Budal and Falnes in their work concerning heaving buoys [35]. The basis of latching control is to try and satisfy the phase condition described previously when talking about phase and amplitude control.

In latching control, the motion of the device is blocked at specific points during a cycle, usually at the two extremum points of the displacement when the device velocity is null. This blocking of the device is performed in order to impose the device velocity to be in phase with the excitation force.

Although this causes the velocity to be zero in parts of the cycle, it still allows for a greater energy capture because of the phase alignment of velocity and excitation force.

For regular waves, the frequency for latching and unlatching will be fixed, but for irregular waves, the time interval between force maxima varies in time, not allowing a fixed latching and unlatching scheduling. Additionally, this technique requires the knowledge or at least a prediction of the future

wave excitation force since the unlatching must obviously occur before the peak force in order to allow the velocity profile to build up so to obtain matching peaks in time (same phase).

An example of the evolution of system variables under latching control subject to a regular wave is presented in the following image where:

- Red line = Excitation force
- Blue line = Velocity
- Black line = Heave position

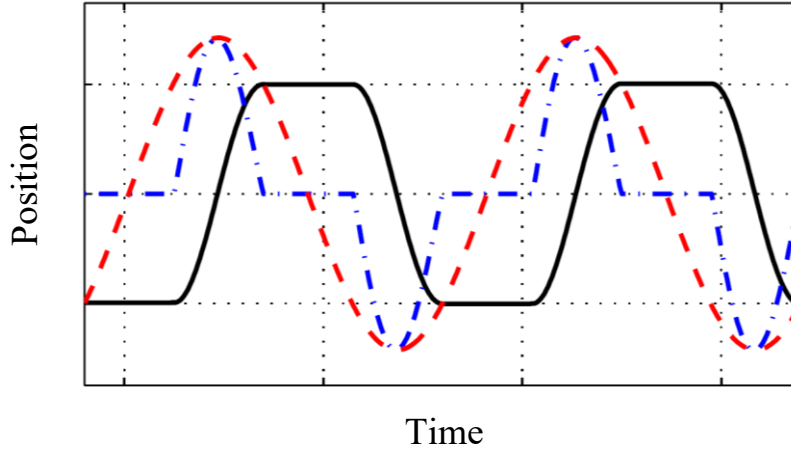


Figure 3. 7 - System variables under Latching control [33].

It can be easily understood that latching control works best when the excitation period is larger than the intrinsic resonant period of the device.

The latching technique described above is also known as *peak-matching latching control* since it tries to match the peaks of the velocity and force signals [36] [37].

Another variation of latching control was suggested in the works of Falcão [38] and Lopes et al. [39]. Instead of aligning the peaks, which requires a prediction of the future excitation force, an estimation of the instantaneous force is used to trigger latching and unlatching. The proposed principle simply involves releasing the buoy once a certain force threshold has been passed. This technique is also known as *threshold unlatching control*.

Whatever the latching implementation may be, during the unlatched period of the motion, the PTO resistance  $R_{PTO}$  is usually kept constant at a value which is optimized based on the current sea state which resembles resistive control.

This is obviously an improvement over simple resistive control since although the resistance value may still be calculated and optimized in the same manner, the devices motion is tried to be kept in phase with the excitation force which is one of the conditions for optimal control, allowing for greater energy capture.

#### 3.4.4.3 Clutching Control

Clutching control, also sometimes known as declutching control, has the same goal as latching control, to achieve phase matching between the excitation force and the device velocity. The difference between latching and clutching is how this is achieved. In latching, the PTO resistance is switched between a constant value and a value ideally equal to infinity which causes the device to halt. In Clutching instead, the device is not clamped, but instead the PTO is *clutched* or decoupled in specific point of the cycle causing the PTO resistance to switch from a constant value to a value equal to zero.

An example of the evolution of system variables under clutching control subject to a regular wave is presented in the following image:

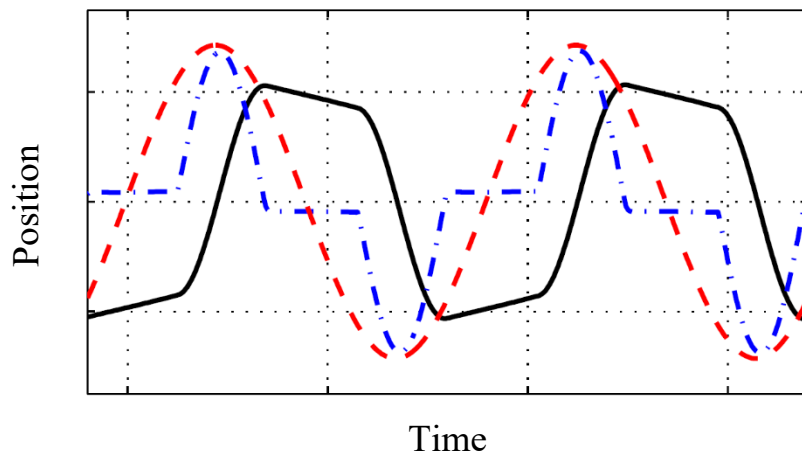


Figure 3. 8 - System variables under Clutching control [33]

Clutching control is most effective when the WECs intrinsic resonant period is larger than the period of the excitation force. This is the opposite working condition as for latching control.

Again, as in latching control, when the device is not clutched, the control law is resistive.

For both latching and clutching, the control goal is to apply the correct switching sequence to maximize power absorption subject to the constraints imposed by the latching or clutching mechanisms and the WECs dynamics.

#### 3.4.4.4 Joint Latching and Clutching Control

As stated earlier, latching control is most suitable when the device intrinsic resonant period is smaller than that of the excitation force, while for clutching the opposite is true. Because of these conditions, both latching and clutching are most suitable only in a well-defined excitation force frequency band based on the intrinsic resonant period of the WEC. A substantial increase in energy absorption can be achieved if both techniques are applied on the same system depending on whether the current excitation force has a larger or smaller period with respect to the device intrinsic period.

This technique has been successfully implemented in [40].

### 3.4.5 Reactive control strategies

As the name suggests, reactive control strategies are those control strategies which involve reactive power flow.

#### 3.4.5.1 Reactive loading control

By adding a stiffness control term to resistive control, it's possible to obtain what is known as reactive loading control.

$$F_{PTO}(t) = -R_{PTO} \cdot v(t) - K_{PTO} \cdot x(t)$$

Where,  $K_{PTO}$  is the additional stiffness coefficient.

In this work the damping coefficient ( $R_{PTO}$ ) will be referred to as C while the stiffness coefficient ( $K_{PTO}$ ) will be referred to as K.

Reactive loading control, in principle, allows for perfect phase and amplitude control. As stated by Slater et.al [31], when incoming waves have a constant frequency equal to the resonant frequency of the WEC, the behavior only depends on the damping factor of the device, which, if set correctly, allows for maximum efficiency.

Thus potentially, reactive loading control has the ability to provide optimal control, but this is an easy task only if the ideal condition of sea states composed of linear single frequency sinusoidal waves. As stated in chapter 3.4.3, optimal WEC control leads to a non-causal problem which in turn requires the prediction of the future excitation force values.

Another difference between ideal impedance matching and what can be achieved using reactive loading is that ideal impedance matching may require optimal damping and stiffness coefficients which may not be feasible because of inherent constraints of the WEC which in turn leads to saturation phenomena of the achievable forces and displacements. Additionally, for point absorbers the optimal stiffness coefficient is likely to be negative [38]. Although this can be achieved with the use of power electronics, the resulting system may become unstable if the control stiffness exceeds the hydrostatic and mooring stiffness of the wave energy converter. For these reasons, limitations on the control parameters must be imposed, which in turn may lead to sub-optimal control in some scenarios.

With all of the above considerations, reactive control is usually not used in the continuous time form to try to achieve optimal energy extraction through time varying damping and stiffness coefficients, instead it is most often used in a tabular manner involving an offline optimization procedure to calculate the most effective damping and stiffness coefficients for a given range of sea states defined by statistical measures of the wave period and amplitude. Hence, sub-optimal values for the control parameters are used. Clearly, a finer gridding of the scatter table and an accurate modelling procedure will produce a higher overall efficiency, still, because of the discrete nature of the gridding strategy



and because of inherent differences between the model and the physical device, this technique is prone to modelling errors.

A solution to the modelling errors is to optimize the control parameters online or over a specified time horizon through a model free strategy. An example of such kind of implementation is given in this thesis. Omitting the use of a model and using an online model-free strategy avoids modelling errors, allowing for a more accurate parameter optimization. Additionally, most model free techniques are able to continuously adapt to changing external variables affecting the relation between sea state, control parameters and power absorption such as phenomena caused by the aging of the device.

An additional further improvement could be achieved if the WECs response was adapted on a wave-by-wave basis. Machine learning techniques such as those used in [41] can be used to predict future wave height and period, which, together with a model free approach and an online optimization of the parameters could potentially allow to firstly learn the optimal parameters for a given wave condition and to then dynamically change the control parameters based on the predicted incoming wave, potentially resulting in a much larger energy capture than what can be obtained with discrete reactive control. The topic of wave forecasting has been extensively studied by Fusco and Ringwood [42].

#### *3.4.5.2 Model predictive control (MPC)*

In recent years, one of the most promising control methods to be used on WECs has been model predictive control. Model predictive control uses a dynamic model of the system together with a feedback of the controlled variables to select an optimal control action at each sample time. This is achieved through a quadratic optimization of a cost function based on the predicted model behavior over a specified time horizon with a moving window, which in turn allows a certain degree of prediction. Additionally, MPC has the inherent ability to handle constraints in the values of the control variables, which as stated before is fundamental when dealing with control of WECs.

Model predictive control applied to WECs was first proposed Gieske in 2007 [43].

Dozens of other studies have been performed since, with many showing promising results [44], [45]. For all its good attributes, model predictive control also has its flaws. Under energetic wave conditions, the accuracy of linear wave theory decreases, and non-linear interactions between the WEC and the impacting waves become significant. Under such conditions the accuracy of the model used by model predictive control will most probably drop significantly, causing the control strategy to become less efficient. Furthermore, Tona et al. have shown that MPC can suffer from measurement noise in the wave elevation measurement and in its forecast [46].

### 3.5 Challenges

The main challenges with WEC control can be usually traced to different root causes, with the most evident being the non-causal nature of the control problem itself, the highly non-linear interactions for rough and uneven sea states, the inherent modelling inaccuracies within model-based strategies. As stated in the previous chapter, optimal WEC control is frequency dependent, which requires the need to continuously vary the control parameters on a wave-by-wave basis in order to obtain the highest possible energy absorption. A solution not the problem of wave forecasting with the aim to then be able to apply a wave-by-wave (fast) control has been extensively studied through techniques such as machine learning, neural networks or autoregressive models [42] [47].

A possible solution to the problem of the highly non-linear interactions which then inevitably lead to modelling errors and consequently suboptimal control actions, is to use a model-free approach.

The advantage of these approaches is that no model is used as a basis of the calculation of the control parameters to be used, thus avoiding modelling errors and also having the possibility to adapt to any changes in the behavior of the WEC. A parallel to this strategy is to define the model of the WEC with the use of machine learning techniques as proposed by Valério et al. [48].

### 3.6 Proposed control solution

The proposed control solution aims at resolving some of the issues presented in chapter 3.5 by using a model free approach. This approach will be focused in finding the optimal control parameters of a reactive control law. A reactive control law was chosen because it has two clear control parameters which can be tuned based on the sea condition, which makes it a perfect candidate for the work which needed to be performed, and also because reactive control laws have proven to be good candidates as WEC control laws [33].

The proposed solution starts by considering each point absorber in the deployed array of point absorbers as an individual of the population of a genetic algorithm [49]. In this framework, each individual carries as genetic information the control parameters (stiffness and damping) currently in use.

Regarding the fitness measure of each individual, the control system in a WEC aims at maximizing the energy absorbed by the device over a certain period of time  $T$ . Since the instantaneous gross power for a point absorber is given by the product between the wave energy converter heave velocity  $\dot{z}(t)$  and the force applied by the PTO  $f_{PTO}(t)$ , the final control aim can be expressed as:

$$J(f_{PTO}) = \frac{1}{T} \int_T f_{PTO}(T) \dot{z}(T) dT$$

To this end, the fitness of each individual in the genetic algorithm was defined as the average absorbed power over a predefined and fixed time span.

With each individual carrying as genetic information the control parameters used to obtain a certain fitness, i.e., a certain average power output, the genetic algorithm framework could be used as an optimization algorithm to find the optimal control parameters in a collaborative manner, with the population of the algorithm evolving over time to reach increasingly better control parameters.

To achieve this, an optimization procedure corresponding to an independent genetic algorithm was performed for each discrete sea state encountered, considering for example, the use of a wave buoy to provide the information of what the current sea condition affecting the array is.

This procedure had to be performed on discrete sea states instead of on a global perspective of all the encountered sea conditions since the latter procedure would lead to a single couple of control parameters which would be optimal in a global sense but would be sub optimal in whatever sea condition they encountered. Furthermore, the optimum of such global landscape would be continuously changing based on the encountered sea states and thus on the encountered information, leading to a continuously morphing optimization landscape.

The solution proposed until now has the goal of finding the optimal control parameters of a reactive control law for each discrete sea condition considering the average power absorbed over a predefined timespan. As can be noticed this can be categorized as “discrete control”, where the control parameters are not chosen on a wave-by-wave basis.

To try and improve the control strategy, a continuous control was then implemented with the use of feed forward and long short term memory (LSTM) neural networks.

Once the optimization in each sea state has reached satisfactory results and the algorithms have converged, the data acquired during the optimization procedure can then be used to train a neural network to produce the control parameters in a continuous and real time manner, independently of the current sea state.

This can be achieved by training the network using as inputs to the net the wave force data recorded during the optimization procedures, and as outputs, the optimal control parameters for that specific force sequence.

This will allow the network to learn dependencies which are not directly related to the specific sea state, allowing for a continuous control instead of a discrete control.

The whole purpose of using a genetic algorithm as an online optimization strategy is to not rely on a model to optimize the control parameters, but to rely on physical data.

Although the whole approach is thus based around a model free ideology, for the purposes of this thesis, a model of the point absorber was used instead of a real physical point absorber to simulate the optimization procedure.

It must be noted that this does not in any manner impact the structure of the problem itself, and the developed strategy may be directly applicable to a real-world scenario.

With the above framework in mind, the next chapters will be dedicated at giving an in-depth explanation of how the genetic algorithm and neural networks were developed and lastly, how they were used together to control in a collaborative and model free manner an array of point absorbers.

## 4 – The genetic algorithm

### 4.1 Considerations to be made when designing a genetic algorithm

Genetic algorithms are optimization algorithms belonging to the family of metaheuristics, specifically in the subfamily of population based, nature inspired algorithms.

This family of algorithms employ a population of individuals each carrying a set of genes representing a candidate solution to the optimization problem. These individuals interact with each other and with the environment with the aim of increasing the average quality of the solutions that the population carries, i.e., the fitness of each individual. To achieve this, mechanisms which try to mimic natural evolution and selection are used. Some of the main processes used are parent selection, reproduction, mutation and survivor selection.

The above-mentioned mechanisms all need to contribute to the final goal, evolving the population such that one or more of the individuals reach an optimal or quasi optimal solution in the optimization landscape.

Because genetic algorithms belong to the family of metaheuristics the algorithms are, by nature, stochastic.

Because of this natural stochasticity introduced in these algorithms, two driving forces to success can be distinguished, **exploration** and **exploitation**.

Generally speaking, exploration can be seen as the tendency of an algorithm to let its individuals explore the space of solutions, without using any knowledge of the optimization space. Exploration is an important facet of the optimization algorithm since it allows individuals to explore the solution space and to possibly find new solutions with higher quality and to then share this information to other individuals. Without exploration, an optimization algorithm would only focus on the current knowledge of the optimization space and would most probably converge very quickly to a local optimum or would stall, a phenomenon called *premature convergence*.

Exploitation instead is the opposing force to exploration. Exploitation, as the word suggests, is used to exploit current knowledge of the optimization landscape to drive further improvement of the population fitness. If no exploitation was introduced into the algorithm, the result would be a completely stochastic algorithm driven by random sampling and would probably fail to converge to a solution for the problem at hand.

It is thus important for these kind optimization strategies to always keep both exploration and exploitation of the solution space in use. The degree of exploration vs exploitation depends on the optimization problem and may even be changed during the run of a single problem as the landscape evolves in time or the population reaches a certain position in the parametric space.

For a simpler, unimodal optimization space, the algorithm may be tailored to focus more on exploitation instead of exploration since there would be less danger of trapping in local minima. For highly multimodal problems instead, a careful balance between exploration and exploitation may be needed to prevent both a quasi-random sampling of the space and to prevent trapping in local minima, known as premature convergence. The issue of exploration and exploitation has been reviewed in depth in [50].

In a genetic algorithm, the main drivers of exploration and exploitation will be both the qualitative parameters chosen, like the kind of crossover operator or the kind of parent selection operator, and the quantitative parameters used by such operators, such as tournament size or mutation probability. Most importantly though, the quantitative parameters will mostly affect the explorative and exploitative behavior of the algorithm.

Operators used within a genetic algorithm can be divided in two broad categories depending on what they promote, exploration or exploitation.

- Variation operators, such as crossover and mutation, add an exploratory behavior and create the necessary diversity within a population to prevent stagnation or premature convergence.
- Selection instead is used to exploit the current knowledge and to drive the quality of the solutions carried by the population.

The quantitative parameters behind the variation operators which drive exploration are specific to the kind of crossover and mutation implementation, but generally the most effective way of increasing exploration in a genetic algorithm is to increase the mutation rate.

A higher mutation rate increases the probability that newly produced offspring get mutated, randomly changing their genome thereby exploring new random solutions.

Regarding selection operators, depending on the specific selection operator chosen for both parent selection and for survivor selection, there usually exists a quantitative parameter defining the selection process which can be used to vary what is known as the selection pressure.

An example of such parameter is the tournament size for tournament selection.

Selection pressure can be seen as the pressure driving better solutions to be picked for both reproductive purposes and for survival. As selection pressure increases, solutions with a higher fitness will be more likely to reproduce and survive and less-fit solutions are correspondingly more likely to be discarded.

Selection pressure can be measured in many different ways [51] but as stated above, the driver behind selection pressure can always be related to the setup of the selection mechanisms

According to how much selection pressure is acting on the GA, the search mechanism may present two extremes.

One extreme occurs when selection pressure is null. In this situation the search is completely stochastic and may resemble a Monte Carlo method [52], randomly sampling the solution space.

On the other extreme, when the selection pressure is very high, the stochasticity in the problem becomes almost insignificant and the algorithm will closely resemble a local hill-climbing search method.

In the case of the former extreme, the search will obviously become more inefficient since the knowledge of the fitness of each individual will simply be wasted, leading to a very long execution time of the algorithm.

In the latter case instead, the algorithm will blindly follow the fitness information of the most fit individual of each generation, and with a reduction of the introduced stochasticity this will lead to confinement to a local optimum point, also known as “*premature convergence*”.

In both cases the fundamental workings and benefits of how a genetic algorithm, and in general, how an evolutionary algorithm works, will be lost.

It is thus important to always have a balance between selection pressure and the stochasticity introduced into the problem.

Historically, it was common practice to try and produce an algorithm which would exhibit low selection pressure at the beginning of the run, and a strong selection pressure at the end of the run.

The idea behind this method was that this would allow the algorithm to initially explore more of the solution landscape, while at the end of the run, when close to the optimum, it would allow to fully exploit the current knowledge to reach such optimum.

Although this might seem like a sound reasoning, a few problems with this approach might arise:

- Unless dynamic feedback is given on how close the run is from completion, or a priori knowledge is used to determine approximately how long run is going to take, there is no way of precisely implementing predetermined rise in selection pressure over a complete run.
- Evolutionary learning, especially if the optimization landscape is highly multimodal, is a more dynamic process than what the previously described technique would seem to propose. In some stages of the run, high selection pressure may be beneficial because the population may find itself on a mostly smooth optimization surface to be climbed, and a strong selection pressure may be beneficial to exploit the current situation.

In other stages of the run, the population may risk trapping in a local minimum, and a low selection pressure is beneficial to explore solutions outside the local minimum valley so that premature convergence does not occur.

Considering these possible scenarios, it is obvious that there is no certainty about at what point during the run they may occur. Additionally, the order in which they occur will most certainly be mixed in most cases.

As can be noticed, tuning the selection pressure of an algorithm is still a difficult task and remains an important factor to be studied in the field.

Setting up the algorithms to have a well-balanced selection pressure is not the only way to ensure that the algorithm will converge without risking either a premature convergence or no convergence at all.

As stated before, the selection operators are only part of the complete process needed to evolve a population of a genetic algorithm.

The other equally important parameters that need to be set correctly are those of the variation operators which are mainly responsible for the exploratory behavior of the individuals within the population. These parameters include the right choice of crossover operator and the quantitative parameters defining it, the right choice of mutation operator and the quantitative parameters defining it, and in general a correct setup of the algorithm structure.

All of the above considerations are needed in order to produce an algorithm which is neither too explorative, which would lead to a very sparse and diverse population and a failure to converge to a solution, nor too exploitative which would lead to a population whose individuals become very similar very quickly, causing a loss in diversity in the population which then consequently leads to the population converging to a suboptimal region of the search space.

#### 4.1.2 System stability for C and K values

Because a genetic algorithm does not inherently have any boundaries within which it should generate candidate solutions, particular attention had to be placed on whether the generated solutions of the parameters for the reactive control law led to a stable or unstable system.

To this end, the 1 DOF model of the point absorber was used to test for which couples of C and K would the system become unstable.

This information could then be used to properly design the genetic algorithm so that it would not generate solutions outside the stability bounds.

The aim of the test was to evaluate for which couples of the control constants would the eigenvalues of the “A” matrix of the state space representation of the point absorber have a real part larger than zero.

An initial range of the parameters to perform the test was chosen as:

Parameter	Min	Max
<b>C</b>	-100 [Ns/m]	$3 * 10^6$ [Ns/m]
<b>K</b>	-2*kh	3*kh

Where  $kh$  is the hydrostatic stiffness parameter whose value is  $1,9743 * 10^5$  N/m for the considered point absorber model.

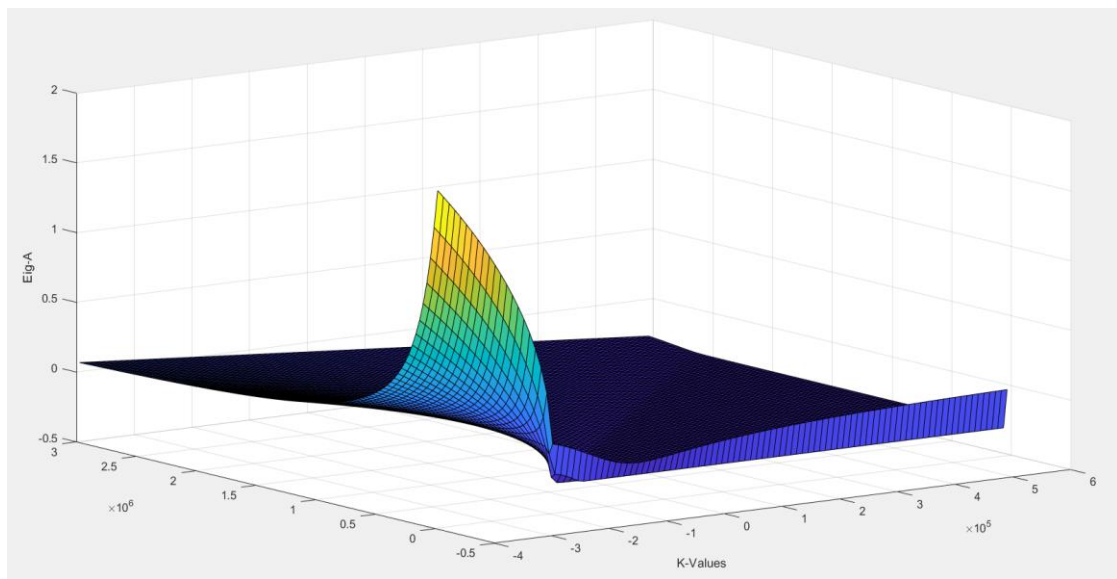


Using the above ranges, two vectors of C and K values to be fed to the pint absorber model, together with a reference sea condition, were created.

These two vectors were then used to create a gridding of the two C and K parameters allowing to see for which ranges was the system stable (real part of eigenvalues of A matrix  $< 0$ ) or unstable (real part of eigenvalues of A matrix  $> 0$ ).

Because only one positive eigenvalue is needed to render the system unstable, and because the A matrix is a square matrix of dimension eight, at each iteration with a couple of control parameters, only the eigenvalue with largest real part was plotted. This procedure saved time and was more accurate than plotting the real parts of all eight eigenvalues.

The resulting plot is the following:



*Figure 4. 1 - Plot of the largest eigenvalue of A matrix with varying stiffness and damping parameters*

Results showed that the whole range of C parameters did not cause instability, while for a K value lower than -kh, the system became unstable.

These results allowed to limit any individual within the genetic algorithm to have Genene within the predefined stable bounds.

For what would concern a possible real application, this test could be done on a meta model of the real device. The final results could then be used as guidelines to design bounds with a given degree of conservativeness in order to prevent possible unstable working conditions.

## 4.2 Genetic algorithm design

Although the underlying idea and goal is usually common between all forms of a genetic algorithm, the mechanisms and strategies which form the algorithm itself vary based on the kind of problem that needs to be solved and on the variables that the algorithm needs to handle.

In the following sections, a description will be given concerning which operators were chosen to set up the genetic algorithms for the problem of optimizing the control parameters of the reactive control law, namely the damping coefficient  $C$  and the stiffness  $K$ , for each sea condition.

### 4.2.1 Representation

The first step to take when designing a genetic algorithm is to link the real-world problem with the genetic algorithm. This procedure entails translating or “representing” the variables of interest for the given optimization problem, into variables which can be manipulated by the genetic algorithm.

Solutions in the variable space of the original problem are called *phenotypes* while solutions encoded in the optimization space of the genetic algorithm are called *genotypes*. The purpose of the representation strategy is to find a mapping between the phenotypic solutions and their genotypic counterparts which will then form the genes belonging to the individuals of the population of the genetic algorithm.

When choosing or designing a representation strategy, a few important considerations need to be made:

- **Each solution of the phenotypic space needs to be explorable:** It’s important to have a mapping between phenotypes and genotypes that does not inherently limit the exploration of the phenotypic space unless specified. This would quite obviously be a limitation in such that possible good solutions may never be explored simply because they were not accessible. In some applications although, the phenotype space is naturally discrete, or may be limited by design. In such cases, a representation may perfectly fit such limitations without causing a loss of potential solutions.
- **The mapping of elements must be an injective mapping:** When a solution of the optimization problem (a genotype) needs to be translated back into a solution of the real physical problem (a phenotype) it’s important for this mapping to be injective, meaning that the genotypic solution must correspond to one and only one phenotypic solution.
- **The mapping must be meaningful:** Lastly it is important to note that when choosing a mapping, it is important to make sure that each possible solution generated by the genetic algorithm can be translated back into a meaningful solution to the original real-world problem.

Following the above guidelines, the chosen representation was a *real representation* strategy.

Real representation in our case entailed directly using the genotypic variables as phenotypic variables. Since the variables in the phenotype space were the two control variables for the reactive control law, namely C and K, these two variables were directly used as the genes which each individual of the genetic algorithm would carry.

This approach allowed to easily follow all three guidelines proposed above, but it also allows the designer to have a good tangible feel of all the following design steps he must take since the variables with the genetic algorithm will manipulate are the same variables used in the real physical problem.

#### 4.2.2 Evaluation function (Fitness Function)

The purpose of the fitness function is to take the gene of an individual and to evaluate how well it fulfills the criteria for which the algorithm is optimizing for. So simply put, it takes as input the gene of a candidate solution previously decoded in to its phenotypic equivalent and gives as output how fit the solution to the problem at hand is when using that particular phenotypic solution.

This function is of particular importance since it's the driver for improvements in the population and it's the basis of one of the main mechanisms within the genetic algorithm structure, selection.

To give an example, if the problem were to maximize the function  $x^2$ , and for example we were considering a binary representation, an individual with gene 1001, which would correspond to a phenotypic solution equal to 9, would have a fitness of:  $9^2 = 81$ .

For the specific problem of maximizing the absorbed energy over a predefined period of time, a simple fitness function that may be used is the average power output over such period of time.

In order to correctly choose a time period over which base the fitness function, a spectrum analysis of the wave conditions under consideration had to be performed.

##### 4.2.2.1 Spectrum analysis of wave conditions

An analysis of the wave conditions considered and the noise that affects them was needed in order to properly choose the time over which evaluate the fitness function.

This parameter would not only affect the fitness function itself but would also affect the simulation running time of the algorithm, or in a physical application, the optimization time needed by the algorithm to reach a solution. This is because the timespan to be chosen coincides with the time each generation will be tested for to retrieve the fitness of each individual. Since the fitness of each individual must be evaluated at each generation, the time over which the average power must be evaluated is obviously of great importance to assure both a good solution to the problem, but also to converge in a timely manner to such solution and to take full advantage of a possibly short sea state.

The main considerations which needed to be considered were

- Too short of a time span would lead to a fast algorithm, with each generation elapsing quickly and with the algorithm being able to fit many generations in a short time window, allowing for fast convergence. The downside would be that the average power reading over a very short time window would firstly not consider the full dynamics of a given sea state and secondly, it would strongly be affected by the random noise components affecting the sea state. This would most probably guide the genetic algorithm to a suboptimal solution.
- A very long timespan instead would do the very opposite. A long time to evaluate the average absorbed power would allow to fully appreciate the dynamics of the given sea state and would make the random noise affecting the reading insignificant, leading to a robust fitness evaluation of the individual. The downside to having a very long time span is that it would take a very long time for the algorithm to converge to a solution. This would be particularly detrimental for sea conditions which are rarely observed.

The spectral analysis showed that a 20-minute time window would be a good trade-off between a fast algorithm, and an algorithm where the random wave noise components wouldn't greatly affect the average power measurement.

This consideration is also supported by previous work from the research team of the MOREnergy Lab which can be read in [53].

#### 4.2.3 Population

The population of a genetic algorithm is composed of individuals, each carrying a set of genes representing a single solution in the genotype space. As generations elapse, the population will evolve to reach the optimum of the fitness function.

An important parameter to be chosen is the population size, which refers to the number of individuals which are present in each generation.

For the specific problem at hand, a fixed population size had to be chosen since each individual in the population represents an actual physical point absorber of the array.

Regarding the population size itself, an array of 16 point absorbers was considered. This choice comes from both an implementation and a computational point of view. Sixteen point absorbers is a reasonable compromise which strikes a balance between a realizable project, with a reasonably small number of devices, and having enough devices to be able to solve the optimization problem using a genetic algorithm.

#### 4.2.4 Initialization procedure

With the representation and fitness function defined, the initialization procedure for the first generation of individuals had to be chosen.

To create the first generation of a genetic algorithm, an initialization procedure must be defined since no previous generation exists from which the first generation can evolve from.

Different initialization techniques exist in the literature, with some techniques taking advantage of partial knowledge of the optimization landscape to initialize part of the population in strategic points of the landscape.

When considering evolutionary algorithms (EAs) in general it has been shown that good initial guesses can facilitate EAs to locate the optimum [54] [55].

When dealing with black box optimization, there exists no a priori knowledge about the search landscape of the given problem, therefore it is not possible to label an initial population as good or bad. For these classes of problems, the most common initialization procedures employ pseudo-random number generators (PRNGs) to create the initial population.

For this work, in order to generalize the process as much as possible, the initial candidate solutions were generated by picking the gene values from the continuous uniform distributions of the variables C and K using the known stable ranges previously computed by testing the values of the eigenvalues of the A matrix belonging to the state space representation of the 1DOF model of the pint absorber. Specifically, the utilized MATLAB function was the “*unifrnd*” function.

#### 4.2.5 Parent selection

Parent selection is the stage at which parents are selected from the individuals of the current generation so that they can then be used to produce new individuals, the offspring.

The parent selection strategy has to be carefully chosen since it's one of the main operators of the genetic algorithm and it has been shown to be one of the main influences on the performance of the algorithm [56] [57].

Different selection strategies were tested for this work, including fitness proportionate selection [58], deterministic tournament selection [59] and stud selection [60].

The final choice fell on tournament selection for a number of reasons:

- Tournament selection allows to easily choose or tune the selection pressure it applies by choosing the size of the tournament.

A large tournament size will have a high probability of including the fittest member of the population or equivalently, a member with relative high fitness.

This means that by using a deterministic tournament strategy, such individual will be chosen to be one of the parents that will produce offspring. As can be clearly understood, having a

large tournament will promote parents with high fitness, resulting in a large selection pressure.

Taking the concept to the extreme, if the tournament size was equal to the population size, all parents would be equal to the best individual in the generation. This is obviously not desirable since only mutation would be capable of producing new genetic material, but it's simply a boundary case scenario.

Conversely, a small tournament size will have a smaller chance of having high fitness individuals within it, allowing for individuals with lower fitness to have a better chance to being chosen as parents. This strategy would then lead to a lower selection pressure.

Again, considering a boundary case scenario, if the tournament size was equal to one, no selection pressure would be present, and all individuals would have equal chance of being selected as parents.

- Tournament selection allows to maintain the selection pressure constant even as the run progresses and solutions become more and more similar.

Although as explained in a previous section this might not be the perfect trend for selection pressure, and a dynamically changing selection pressure may deliver better performance, a constant selection pressure is definitely an improvement over a selection strategy which suffers from decreasing selection pressure such as fitness proportionate selection.

- It has been shown that tournament selection is a robust selection scheme for noisy environments [61].
- Tournament selection has a good propensity to be tuned and to thus adapt to the optimization problem at hand. It was shown by Volker et al. [56] that tuning an algorithm equipped with tournament selection as a parent selection strategy returned remarkable result in terms of solution quality, algorithm speed and tuning effort.

Once the selection strategy had been chosen, the number of parents to be picked and how many offspring they will produce needs to be decided.

For the current implementation, I decided to pick sixteen parents for each generation and to have each couple of parents produce two offspring, so that the final number of offspring matched the fixed population size.

It was obviously important to have the offspring population size to be equal to the predetermined population size since each offspring was directly linked to one of the physical point absorbers in the array. This would then allow to measure the fitness of each offspring with the usual reading of the average power absorbed over a twenty-minute time span.

It must be noted that although the total number of parents to be picked equaled the number of individuals from which the parents were picked, it does not mean that each member of the population got to become a parent, otherwise the parent selection strategy would be useless.

Instead, each parent was picked by a single independent deterministic tournament selection strategy, allowing for each member of the population to be picked multiple times, or possibly none, in a given generation.

To implement this strategy, a total of sixteen tournaments were run for each generation to pick the sixteen parents. Each time a couple of parents was selected, a corresponding couple of offspring would be produced. This process would repeat until all sixteen offspring were generated.

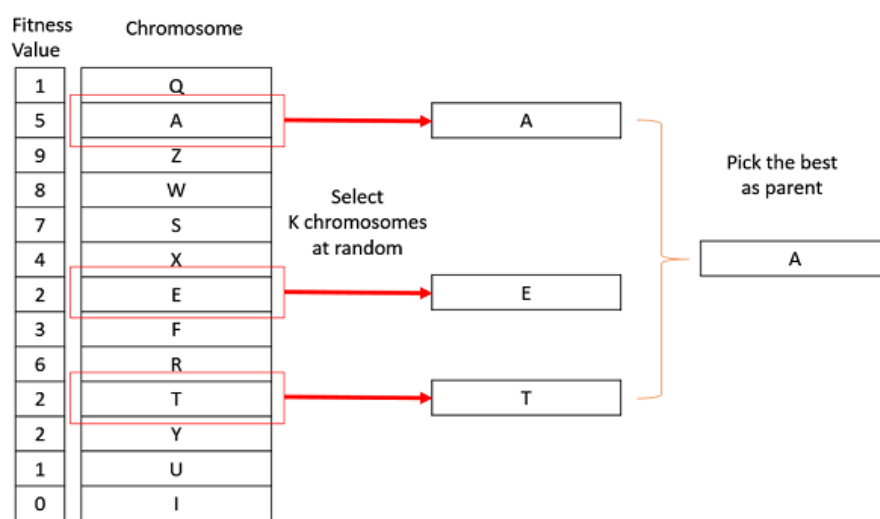


Figure 4. 2 - Qualitative representation of deterministic tournament selection

#### 4.2.6 Crossover (Recombination)

After each couple of parents had been selected using a tournament selection scheme, crossover was deterministically applied to produce two offspring from each parent couple.

Crossover is used to merge the information contained in the parent's genes to produce new genetic material, the offspring.

Crossover is linked to the parent selection stage since, the purpose of the two operators combined is not to blindly produce offspring, but to produce offspring who most likely will have a higher fitness with respect to the parents which gave birth to them. This can be achieved by providing a healthy selection pressure throughout the algorithm, and by using an appropriate crossover mechanism to properly blend the genetic information of the two parents.

A binary crossover strategy was used since, although crossover operators using more than two parents exist, the binary option is the most popularly used and additionally it provides a direct link to most biological reproduction mechanisms.

In particular, a *whole arithmetic recombination* strategy was used as crossover.

Whole Arithmetic Recombination is a fully **arithmetic recombination** strategy. It simply involves taking the weighted sum of each gene from the two parents. This weighted sum is controlled by the parameter  $\alpha$  which tells which parent will influence the weighted sum more greatly [62].

Two offspring can be produced by using:

$$\text{Child 1} = \alpha \cdot \bar{x} + (1 - \alpha) \cdot \bar{y}, \quad \text{Child 2} = \alpha \cdot \bar{y} + (1 - \alpha) \cdot \bar{x}$$

Where  $\bar{x}$  and  $\bar{y}$  are the gene sequences of the two parents.

The parameter  $\alpha$  is used to tune the weight each parent has when forming the two offspring.

It must be noted that when  $\alpha = 1/2$  the two offspring will be exactly identical, thus it is usually preferred to maintain  $\alpha \neq 1/2$ .

#### 4.2.7 Mutation

After the offspring had been produced, a mutation procedure was used to produce the final mutated offspring.

Mutation takes the original gene sequence from an offspring and slightly modifies it to produce the mutated version of the original offspring. This operator is used to introduce some additional stochasticity in the algorithm which helps in delivering a more effective search procedure in the optimization landscape.

In general, it is desired to design a mutation operator such that the change it causes is random, unbiased and most often small rather than large.

To meet the requirements listed above, the chosen mutation operator was a *Nonuniform Mutation*.

The nonuniform mutation operator adds a value drawn from a Gaussian distribution with zero mean and with a predefined standard deviation. This kind of mutation assures that the additional value added to the gene is most likely small, and the probability of having a large mutation decreases as a function of the standard deviation chosen.

The standard deviation of the Gaussian distribution will dictate how large will the mutation be on average and is thus often referred to as *mutation step size*.

For this work, mutation was not applied deterministically to each offspring, instead a mutation probability was accounted for in order to only apply mutation with a certain probability distribution. This is generally the most common way to use mutation since it allows to benefit from mutation by exploring new solutions in the search space with an unbiased and randomic pattern, but, because it is not applied deterministically on all individuals, it also allows to prevent a disruption of the



evolution mechanisms of the genetic algorithm, which would then behave similar to a random search procedure.

After mutation has occurred, the offspring should be theoretically ready to be evaluated, and their fitness function defined. Because the problem at hand needs to work with finite and well defined ranges of the optimization variables, before this evaluation could be performed, a truncation of the offspring gene values to the bounds of the optimization variables had to be performed.

#### 4.2.8 Survivor Selection

Survivor selection is used to select the individuals which will “survive” and will go and form the next generation, from which parents can be picked, which will then produce offspring and the cycle repeats itself until a termination condition is met.

Because the population size is fixed for this work, selection is needed in order to prevent the population size from continuously growing and it is also used to try and keep the better individuals while discarding the worst ones.

The process of discarding the worst individuals although might need to be carefully regulated since in some scenarios, simply discarding all the lesser fit individuals and only allowing the fittest to survival may cause premature convergence to a local optimum.

The main driving factors that are used for survival selection are either age or fitness. Age can be used to discard the “n” oldest individuals each time selection has to be performed while fitness-based selection usually discards the “n” less fit individuals. Stochasticity can be added if needed to make the processes nondeterministic.

Different kind of survivor selection methods are available but generally, the offspring can either:

- Directly form the next generation (if the number of offspring is equal to the population size).
- Compete against each other only (if the number of offspring is larger than the population size).
- Compete against each other and the previous generation.
- No fitness-based competition occurs, the age of the solutions is used to determine who continues to the next generation (youngest) and who is left out (oldest).

Since for the current implementation age is not seen as a driver to a good solution, competition based on fitness has been used in order to drive survivor selection. In particular, a competition between the previous generation and the relative offspring generation was used.

As a mechanism to implement survivor selection, tournament selection was used. Differently from when used for parent selection, the individuals which participate in the tournament are now picked from the joint parent and offspring populations. In this way, parents and offspring are pit against each

other for survival. At each tournament, the winner is allowed to be one of the members of the next generation of individuals. Tournaments are run until the next generation is completed.

Tournament selection was chosen because of all the good characteristics it possesses which were previously listed in section 3.2.5 related to parent selection.

Additionally, it has been shown that, just as tournament selection shows good tuning characteristics for parent selection, it also shows a good ability to be tuned for survivor selection [56].

## 4.3 Tuning the genetic algorithm

### 4.3.1 Introduction

With the structure of the genetic algorithm defined, the next step to be taken was to define the various hyperparameters to be used within each step and operator of the genetic algorithm.

Historically, hyperparameters such as mutation rate, mutation step size and tournament size were chosen without any specific empirical data showing that the chosen parameters would allow the algorithm to perform well on the specific problem.

This practice was common since evolutionary algorithms in general were considered vary robust with respect to variations on the hyperparameters characterizing their operators, allowing practitioners to be confident that choosing hyperparameters that were in the general neighborhood of the optimal hyperparameters was good enough to achieve good performance from the algorithm.

The following image shows how evolutionary algorithms were seen in the 1980s after Goldberg [63].

The image shows how evolutionary algorithms were seen as good search algorithms over a wide range of problems, and thus did not need additional tailoring to the specific problem to be solved.

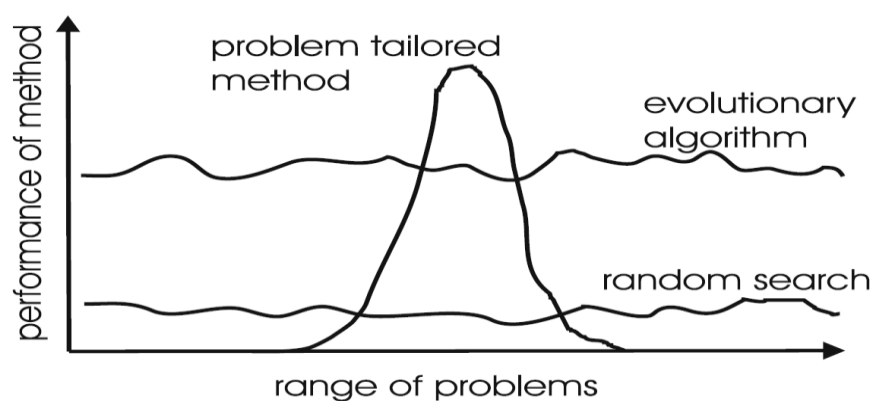


Figure 4. 3 - View of Evolutionary algorithm performance in the 1980s [62].

Although evolutionary algorithms are still considered quite robust with respect to variations in their hyperparameters, it was soon understood that by tailoring the evolutionary algorithm to the problem at hand, adding problem specific knowledge to the design of the algorithm would allow for a much

more performing optimization procedure. In this vie it is possible to transform the above graph by adding problem specific knowledge to a basic evolutionary algorithm in order to achieve better performance for a specific range of problems.

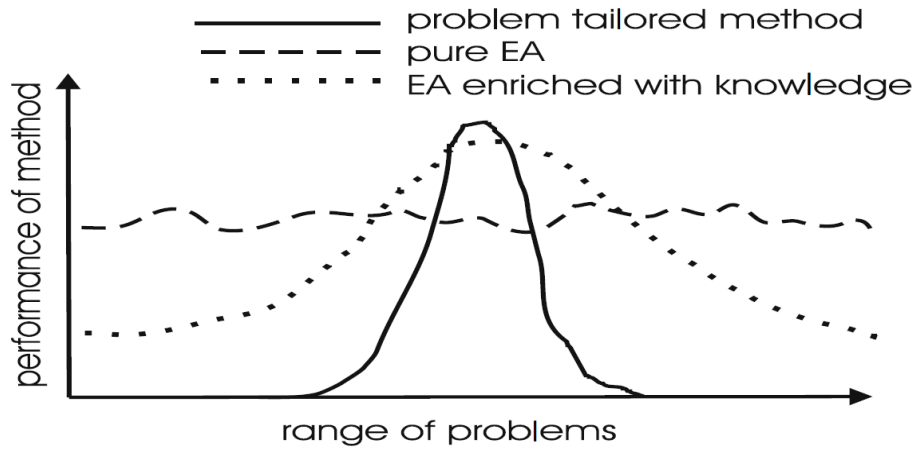


Figure 4. 4 - View of evolutionary algorithms with added problem specific knowledge [62].

In order to achieve the above result, the hyperparameters (also called quantitative parameters) a tuning procedure was applied.

In the following section the basic notions on tuning and the tuning method utilized in the genetic algorithm for this work will be presented.

#### 4.3.2 Tuning notions

After the framework for the genetic algorithm has been decided, the parameters governing this framework and its operators must be selected. In these terms, designing a genetic algorithm is about selecting possibly good values for the parameters involved. It must be noted that the choice of such parameters is fundamentally different from the choice of the operators that made up the structure of the genetic algorithm. In other words, choosing between different parent selection strategies such as tournament vs stochastic universal sampling is different than choosing a crossover rate  $p_c \in [0, 1]$ .

This fundamental difference comes from the fact that the main operators belong to a finite domain with no ordering or distance metric, e.g.,  $crossoveroperator \in \{one\ point, averaging, uniform\}$ , while the hyperparameters used by such operators belong to a subset of real numbers, thus having a natural structure with ordering and distance metrics. This notion is fundamental for searchability. For parameters that belong to an ordered structure, and which have a distance metric, heuristic search and optimization methods can be used in order to find optimal parameter values given a performance function to maximize or a cost function to minimize. For parameters that do not belong to this category, such as crossover operators or selection strategies, the only option to find the optimal operator is sampling.

As suggested by Eiben et.al [64] it is thus important to differentiate the two kinds of parameters, and the naming convention used from here on will be *qualitative parameters* for operators such as tournament selection, roulette wheel selection etc. while *quantitative parameters* will be used for the parameters which belong to a numerical structure such as the mutation step size.

#### 4.3.3 Tuning vs control

Regarding the choice of qualitative parameters, as stated previously, this procedure was often performed choosing the values to be used simply through experience, or by using values which were thought to allow good performance over a wide variety of problems, meaning that the chosen parameters were never specifically tailored to the problem to be solved.

An opposite view on the problem of choosing qualitative parameters is to use information about the problem in order to make an informed decision on the parameters to be used.

In the field of evolutionary computing, this procedure can usually be distinguished in two approaches:

- Parameter tuning: Where the qualitative parameters are established before the run of the EA by optimizing such parameters through the use of a performance function for the EA. Additionally, the parameters do not change for the entire run.
- Parameter control: Where the qualitative parameters are established during the run thanks to feedback from the algorithm state. In this case, parameters are given an initial value, and are then allowed to evolve as the algorithm is running.

Both techniques have been extensively studied in recent years and are both a definite improvement over using “standard” settings.

For this work, parameter tuning was chosen over parameter optimization since parameter tuning was deemed to provide good results with relatively small effort compared to parameter control, which would have also added an extra layer of complexity to be managed online during the algorithm run.

Finally, the choice of using parameter tuning is also justified by the fact that the additional complexity of parameter control is usually justified for performance landscapes which dynamically change during the run [65], while for the purpose of this work, the performance landscapes are static.

#### 4.3.4 Tuning procedure layout

The tuning procedure involves tuning the genetic algorithm quantitative parameters while allowing the underlying genetic algorithm to use such parameters to solve a target problem in order to get feedback on how the used quantitative parameters allowed the algorithm to perform on the optimization problem. This feedback is then used to optimize the quantitative parameters through a secondary optimization procedure. This process is represented in the flow chart below [64].

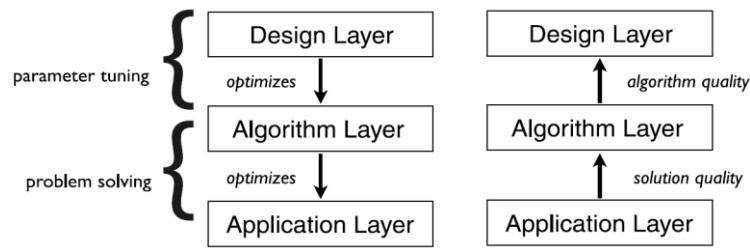


Figure 4. 5 - Control flow (left) and information flow (right)

Using the nomenclature in the image above, it is possible to distinguish:

- The Design Layer, responsible for the optimization of the quantitative parameters of the algorithm layer below. As information, the design layer receives a value from the algorithm layer representing the quality of the parameter vector being optimized (in this case the quality of the parameter vector fed to the genetic algorithm). The Design layer then uses this information in order to optimize the vector of quantitative parameters.
- The Algorithm Layer, which corresponds in this case to the genetic algorithm. As information the genetic algorithm receives feedback from the Application Layer about the solution quality (or fitness) each individual currently possesses during the run and uses this information to try and solve (optimize) the problem at hand.
- The Application Layer, which contains the original problem which the genetic algorithm is being designed to solve. This layer is obviously responsible for giving information to the genetic algorithm about the current fitness of each individual in the optimization landscape.

To avoid confusion regarding the quality of the individuals of the genetic algorithm and the quality of the parameter vector used by the genetic algorithm we denote as *fitness* the quality of a given individual in the original problem optimization landscape, while we denote as *utility* the quality of a given parameter vector being used by the genetic algorithm.

A fundamental difference between the fitness of the individuals of the genetic algorithm and the utility of the parameter vectors is that the fitness of each individual in the genetic algorithm is deterministic, meaning that if two separate calculations of the fitness of an individual carrying a given set of genes is performed using the underlying function to be optimized, the fitness value will not change in time since the fitness landscape for the problem at hand does not dynamically change in time. On the other hand, the utility value of a given vector of quantitative parameters will be necessarily stochastic because of the stochasticity of the genetic algorithm using such vector of parameters. If two runs of a genetic algorithm are performed using the same vector of parameters, the resulting utility measure will differ between the two runs, this is due to the fact that because genetic algorithms are stochastic in nature, two runs with identical setup will evolve differently, giving different utility measure values.

Because of this, the utility measure needs to be defined in some statistical sense.

Furthermore, the utility measure must be defined in order to reflect the target performance of the genetic algorithm. Based on the problem at hand and on the user's preference, one might define a utility measure more focused on algorithm speed, or for example more focused on algorithm performance in terms of quality of the final solution.

In the next section, algorithm quality and performance functions will be addressed, which will form the basis of the definition of the utility measure.

#### 4.3.5 Algorithm performance and utility measure definition

When defining the performance of a given algorithm using a vector of quantitative parameters there are two main factors influencing performance: algorithm speed and solution quality.

Most often the performance metric and thus utility measure will take into account both of these factors when evaluating an algorithm implementation.

Solution quality can be easily related to the fitness function of the genetic algorithm, while algorithm speed is usually related to either the running time or the number of elapsed evaluations (generations). These two factors can then be used to define different performance measures used to evaluate the performance of an algorithm on a single run:

- Given a predefined maximum running time or number of elapsed evaluations, the algorithm performance is defined as the best fitness value in the population at termination
- Given a predefined fitness level, algorithm performance is defined as the time needed to reach such fitness level
- Given a predefined maximum running time and a target fitness level, the algorithm performance is defined through the notion of success. If the algorithm reaches the fitness target within the predefined time window, the run is marked as successful.

Because of the stochastic nature of genetic algorithms, the performance measure needed to be evaluated not on a single run, but on multiple runs in order to have a statistically relevant measure.

Considering multiple runs of the genetic algorithm and by taking the average performance over all runs, the performance measures defined above for a single run take the names of:

- Mean Best Fitness (MBF)
- Average Number of Evaluations to Solution (AES)
- Success Rate (SR)

For this work, the performance measure, which corresponded to the utility measure of the parameter vector used for the run, was chosen as the mean best fitness.

When using mean best fitness, two important considerations need to be addressed:

- The time measure to be used
- The maximum running time limit
- The number of evaluations over which the MBS is calculated

Regarding the time measure to be used, many options are available and have been used in the literature. Some examples include the number of fitness evaluations, the CPU time or the wall-clock time. For this work, the number of fitness evaluations was chosen as a time measure. This choice was made in order to eliminate the effects coming from the particular hardware and software implementations that were used for the optimization procedure [66], but more importantly because in a real world implementation of the proposed algorithm, what really defines the time needed to reach a good solution is the number of fitness evaluations, since each fitness evaluation may last anywhere from 15 to 40 minutes, depending on the implementation strategy proposed by the used. In this work, each fitness evaluation is considered to take 20 minutes, thus the driving factor behind algorithm speed will be the reduction of the number of fitness evaluations needed.

The maximum running time limit in order to define the performance function through MBS had to be carefully chosen since it would define the stopping time at which each optimization would be evaluated. If a large number of maximum evaluations were to be chosen, algorithm instances which reached a good solution, without much regard for speed would be favored, on the other hand, if a really small number of maximum evaluations were to be chosen, algorithms which quickly improved the fitness of individuals would be favored, but without any guarantee that such algorithm implementations would guarantee steady improvement over a longer time period.

A balance had to be struck between favoring algorithm instances which reached good solutions quickly, and algorithm instances which reached good solutions, given a very long computational time window.

A basic idea of how many generations would a genetic algorithm need to reach a good solution had already been acquired through different tests with various GA implementations before the tuning procedure was used, furthermore, another driving consideration to be taken in to account was the duration of each sea state, and the occurrence of each sea state in a yearly scatter table.

With all the above considerations, a maximum number of fitness evaluations of 15 was chosen as it allowed a good balance between favoring swift algorithms, but also considering that within 15 generations a good solution was expected to be found.

Finally, the number of statistical evaluations in order to calculate the MBF had to be decided. This number corresponds to the number of times a genetic algorithm with a given quantitative parameter vector was allowed to run to completion (15 generations), before calculating the MBF, which would then translate in the utility of the used parameter vector which in turn could then be used by the top-level optimizer (in the design layer) in order to optimize the parameter vector values.

Ideally, the larger the number of evaluations, the more statistically robust will the utility of the parameter vector be, and more reliable the results, but on the other hand, each additional evaluation meant that the computation time would grow considerably.

A number of statistical evaluations equal to 5 was chosen for this work.

Another consideration which needed to be made was the application layer function that the genetic algorithm would be tested on.

The choice of the function used for this layer will greatly impact the outcome of the result of the tuning procedure of the qualitative parameters since the tuned parameters will be optimized to produce good results, in terms of speed of convergence and quality, on that specific function.

A possibility which was considered was to produce a test suite with all the sea states considered in this work and to optimize the parameter vector over all sea states. Although this would guarantee a performing algorithm over all sea states, it would also mean that the algorithm would be optimized considering feedback from each sea state to have equal importance, unless some sort of weighting procedure was added. This is obviously undesirable since it would be more useful to reach excellent optimization results in the sea states which are most frequent instead of good results for all sea states. Furthermore, the shape of the power curves under the influence of each of the sea states considered had the basic same shape. With the above considerations in mind, the choice was made to use the power curve of the most frequently encountered sea state to optimize the quantitative parameter vector. This would allow to focus performance on the most frequent sea state while still guaranteeing good performance on other sea state power curves because of the shared similarities between curves.

The final piece of the puzzle is the optimizer used to optimize the quantitative parameter vectors being fed to the genetic algorithm below. For this work two options were considered, either use another genetic algorithm to optimize the parameter vectors [67] [68] [69] or use a surrogate optimization strategy.

Both techniques were tested, but the choice fell on the surrogate optimization strategy because of its capability of handling time consuming objective functions. For reference, the *surrogateopt* Matlab function was used.

To summarize the procedure, a surrogate optimization was used to optimize the quantitative parameters characterizing the genetic algorithm in order to obtain a configuration of the GA which would perform well in its task of finding the optimal control parameters for a given average power optimization landscape in each sea condition. The surrogate optimization considered 5 independent runs of the GA with a given vector of parameters before it could evaluate the utility value of the used parameter vector, which was directly associated to the MBF over the 5 runs, with a predefined



computational effort of 15 generations to be elapsed in each GA iteration. The utility measure was then the driving factor behind the surrogate optimization.

In the following table, a summary of the parameters used for the optimization of the GA parameters is presented.

Design Layer	Algorithm Layer	Application layer	Performance function	Time limit algorithm layer	Number of statistical evaluations for MBF
Surrogate optimization	Genetic algorithm	Power curve of most frequent sea state	MBF	15 generations	5

#### 4.3.6 Tuning results

With the described tuning setup, it was now possible to tune the quantitative parameters belonging to the genetic algorithm structure described in section 4.2.

The parameters to be tuned were:

- Parent selection tournament size;  $\eta_{par,tourn}$
- Mutation step size;  $\sigma_{mut}$
- Whole arithmetic crossover weight;  $\alpha$
- Mutation probability;  $P_{mut}$
- Survivor selection tournament size;  $\eta_{surv,tourn}$

The setup of the surrogate includes lower and upper bound options for its optimization variables and an option to limit some variables to only integer solutions.

These options are very handy for the variables described above in order to limit the search space and to prevent unfeasible solutions such as a non-integer tournament size.

The setup included as bounds for the variables:

- $2 < \eta_{par,tourn} < 15$
- $500 < \sigma_{mut} < 2500$
- $0.01 < \alpha < 1$
- $0.05 < P_{mut} < 1$
- $2 < \eta_{surv,tourn} < 30$

While the tournament sizes were both limited to be integer numbers.

Additionally, a maximum number of function evaluations equal to 60 was used as a stopping criterion of the optimization procedure.

The results of the tuning procedure are summarized in the following table

<i>Symbol</i>	<i>Quantity</i>	<i>Value</i>
$n_{\text{par,tourn}}$	Parent selection tournament size	3
$\sigma_{\text{mut}}$	Mutation step size	1838.1
$\alpha$	Whole arithmetic crossover weight	0.5324
$P_{\text{mut}}$	Mutation probability	0.4963
$n_{\text{par,tourn}}$	Survival selection tournament size	18

Figure 4. 6 - Tuning results

Some of the above results fell in a range of expected values, while others did not fall within what could generally be considered a canonical range for genetic algorithm setup parameters.

Starting from the parent selection tournament size and the survival selection tournament size, which together create the selection pressure which drive progressive evolution, having a small tournament size for the parent selection and a large tournament size for survival selection creates a balanced selection pressure. This result was somewhat expected since it is generally recommended that the two selection pressures coming from the parent selection scheme and from the survival selection should balance each other out.

With the above settings, a small parent selection pressure allows for most individuals to have a good chance of being picked as parents, thus allowing even sub optimal genotypes to reproduce, while a high selection pressure in the survivor selection section is useful to strongly favor the fittest individuals as candidates for the next generation.

The mutation step size is about in the middle of the chosen range. This indicates that the chosen range was adequate for the optimization procedure. A step size close to one of the imposed bounds might indicate that the chosen range should be extended to find the optimal value for such parameter.

The arithmetic crossover weight factor is the weighting factor used to select how much of each of the parent's genes is used to create the offspring. A value of exactly 0.5 would create two identical offspring each having a gene value halfway between the two parent gene values.

An optimal value of 0.5324 indicates that the created offspring are not equal but are quite similar. Additionally, both of the genetic materials from each parent will approximately have the same weight, and thus the same importance, resulting in a good mix of the genetic material from both parents.

Finally, the mutation probability was the most surprising factor. In the literature, it is most common to find mutation probability values close to 10 times less than the optimal value found during the optimization procedure.

A large value of mutation probability indicates that the genetic algorithm will have a large explorative proportion to its behavior, creating offspring which mutate quite often to better explore the environment. This behavior will then be mitigated by a strong selection pressure in the survivor selection section, only allowing stronger candidates to form the next generation.

#### 4.3.7 Tuning procedure for physical application

The whole reason for using genetic algorithms to find optimal control parameters in each sea state for the point absorber array is to disregard the modelling of the point absorber and use a model free approach. But as stated above, a model of the point absorber was used for hyperparameter tuning of the genetic algorithm. This was the only option for this work because of obvious restrictions both in the accessibility to an actual device and also because of time constraints.

In a physical implementation of the device and control algorithm, different possibilities exist regarding the tuning procedure of the parameters of the genetic algorithm.

- A first possible solution is to not tune the genetic algorithm at all, and to use a “standard” genetic algorithm setting. This solution will most probably save setup time, but the main drawback is that the solution found might be suboptimal, leading to suboptimal power extraction.
- A second solution is to use a model as close as possible to the real point absorber in order to perform the tuning phase offline. Having a model which is close to the actual physical point absorber model will allow to tune the genetic algorithm parameters on a fitness function (power curve in this case) very similar to the one from the actual point absorber. This will definitely allow to get into the ballpark of good hyperparameter settings even for an application with a real point absorber. This is a good solution for practitioners who might not have a great knowledge of the relation between the hyperparameter settings and the behaviour of the genetic algorithm on the specific function given by the point absorber used. In any case this method should still give better performance than just manually setting hyperparameters.
- Finally, the third solution is to optimise the genetic algorithm online. This involves an initial deployment phase where the only goal is to tune the parameters of the genetic algorithm. In this scenario, the quantitative parameters of the GA will be changed every so often, and the optimisation results in terms of optimisation speed and result quality can be compared with different implementations of the GA. These results can then be used to determine the quality of the parameter vector used by a given GA implementation, which in turn can then be used to optimise the parameters of the GA itself.

Although this solution is feasible, the time needed to optimise the parameters online would probably render this solution unfeasible in a real application.

## 5 – Optimization through the genetic algorithm

With the genetic algorithm structure chosen and the quantitative parameters tuned, it was possible to deploy the genetic algorithm to optimize the two control parameters (C and K) of the reactive control law for the control of the resistive force that the PTO of the point absorber should apply in each separate sea state.

The test setup consisted of the same genetic algorithm structure, previously tuned, to be used to optimize the control parameters in all sea states while still maintaining all optimization procedures separate. This was achieved by using a memory management strategy, allowing each specific population belonging to a given sea state to evolve separately and only when the corresponding sea state was encountered. This setup thus consisted of one isolated population of individuals for each sea state evolving side by side as different sea states presented themselves and interacted with the point absorber array. As the sea state changes, the memory management system calls on the corresponding population which can carry on the evolution process from where it left off when the given sea state was last seen.

### 5.1 Sea state generation

To test the optimization procedure, a 90-day simulation window was considered in which the sea state occurrence was regulated by the following lookup table.

Wave ID	$T_e$ [s]	$H_s$ [m]	Occurrence percentage [%]
1	4.17	0.1	5.05
2	6.13	0.1	23.56
3	8.08	0.1	1.51
4	10.04	0.1	1.14
5	4.17	1.08	7.49
6	6.13	1.08	28.62
7	8.08	1.08	5.98
8	10.04	1.08	0.90
9	4.17	2.06	0.19
10	6.13	2.06	11.92
11	8.08	2.06	4.67
12	6.13	3.04	1.41
13	8.08	3.04	7.18
14	8.08	4.02	0.38

Figure 5. 1 - Lookup table from Pantelleria site

The values in the above lookup table come from physical data acquired from wave buoys off the coast of Pantelleria in the Mediterranean Sea.

Each sea state was considered to last for one hour and the succession of sea states was considered to be random.

With each sea state lasting one hour and with each genetic algorithm generation taking twenty minutes to evaluate the average power absorbed to be used as a fitness value, each time a sea state

presented itself, three generations of individuals could elapse in the genetic algorithm dedicated to the current sea state. In other words, every three generations elapsed correlates to one hour of elapsed corresponding sea state.

Using the occurrence table, a vector of random sea states was created, with a total duration of 90 days. The vector was simply a vector of couples of  $T_e$  and  $H_s$  values representing a given sea state. The vector was then fed to the memory management system which used the  $T_e$  and  $H_s$  values to trigger the correct genetic algorithm optimization process related to the given sea state and also to generate the simulated sea state with which the point absorber model had to interact with in order to calculate the average absorbed power for each individual.

It must be noted that no stopping condition was imposed since the sea state vector is limited to a 90-day vector. In a real scenario, a stopping condition should be imposed to decide on when the optimization procedure should stop and pure exploitation with the optimized control parameters should begin. The stopping condition could impose a limit on the total days of optimization, or it could impose a limit on the optimization time on each single sea state, may that be through a bound on the generations elapsed or on the number of times a certain sea state is encountered.

The first option would guarantee a well-known stopping time for all optimization procedures, but it would not guarantee satisfactory results for all sea states, especially for the less frequent ones. The second option instead guarantees that a certain number of generations has elapsed in each of the separate genetic algorithms before the optimization procedure stops, thus guaranteeing a certain level of performance but with an unknown time horizon in which all optimization procedures will end.

## 5.2 Simulation results

The most important values to be evaluated from the results of the optimization test were the average number of generations needed for the genetic algorithms to converge and the optimality of the best solution obtained at the end of the simulation in terms of average absorbed power.

The aim of the optimization was thus to reach peak performance in terms of average absorbed power in the least time possible.

The following figure shows an example of one of the power plots for a given sea state with different values of  $C$  and  $K$  control parameters. The  $Z$  axis represents the average absorbed power in a 20-minute time window.

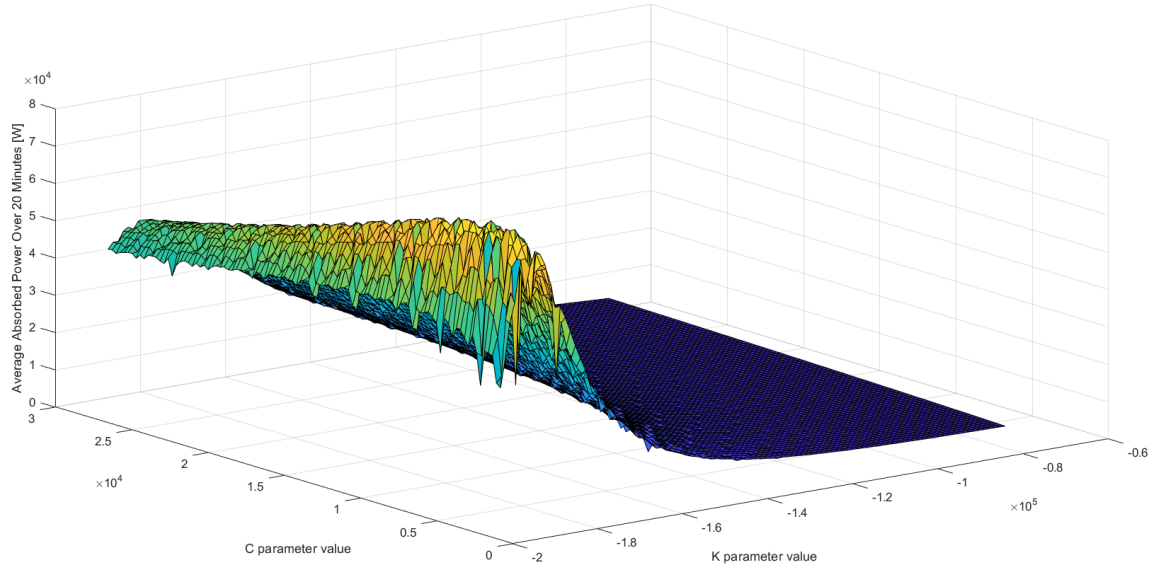


Figure 5. 2 - Point absorber power plot

One of the tools which was used to visually evaluate performance was the power absorption of the best individual in the population as the generations elapsed for a given sea state.

Because of how the optimization in the genetic algorithm was setup, the inverse of the absorbed power was used as the cost function to be minimized, thus the plots which were analyzed show a decreasing trend, meaning an increase of power as generations elapse.

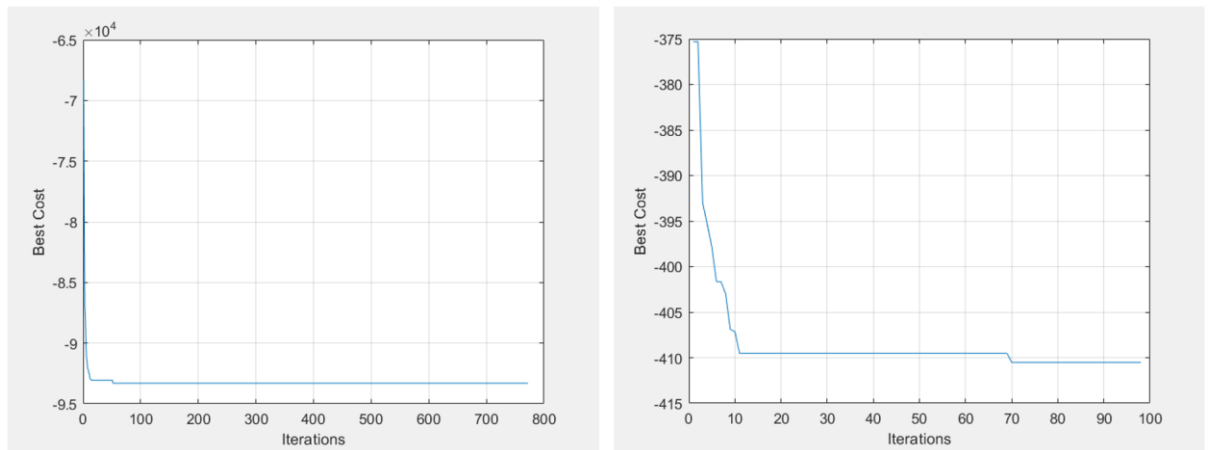


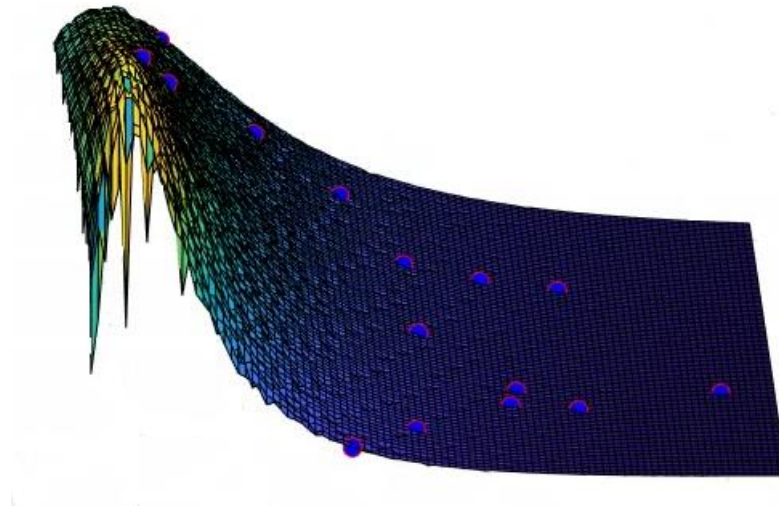
Figure 5. 3 - Examples of evolution of cost of best solution in two different sea states

As can be seen, the above plots show a steep descent at the beginning of the evolution process, meaning a fast initial progress towards optimal regions of the search space.

After only a few generations/iterations, the cost function has practically converged, meaning that the best member of the population is currently using stiffness (K) and damping (C) parameters close to the optimal ones for the given sea state.

Considering that each generation is equivalent to 20 minutes of the corresponding sea state, and that in just over 10 generations the algorithms converged, it would only take approximately 4 hours in a given sea state to find the optimal stiffness and damping factors for that particular sea state.

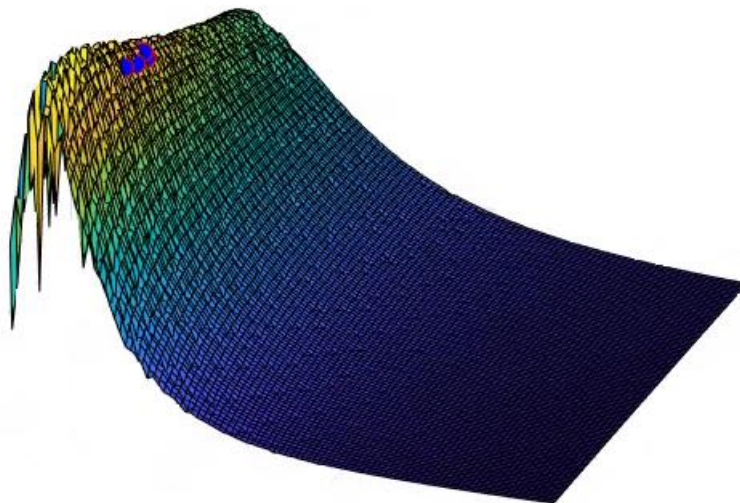
The convergence graphs shown above do show a fast convergence, but no information is shared on how the whole population evolves. To better understand how the population evolves it was useful to visualize the single individuals evolve over the optimization landscape. An example of three instances during a whole evolution process are proposed.



*Figure 5. 4 - First generation population*

The first image shows the randomly generated initial population over the power plot of a given sea state. Each dot represents an individual with its own specific couple of stiffness (K) and damping (C) parameters.

The following images show different stages of the evolution process of the same population on the same power plot, but from a slightly different perspective in order to better visualize the location of the individuals.



*Figure 5. 5 - Fifth generation population*



After 5 generations the population has already started to converge towards the top of the power curve while still maintaining some diversity.

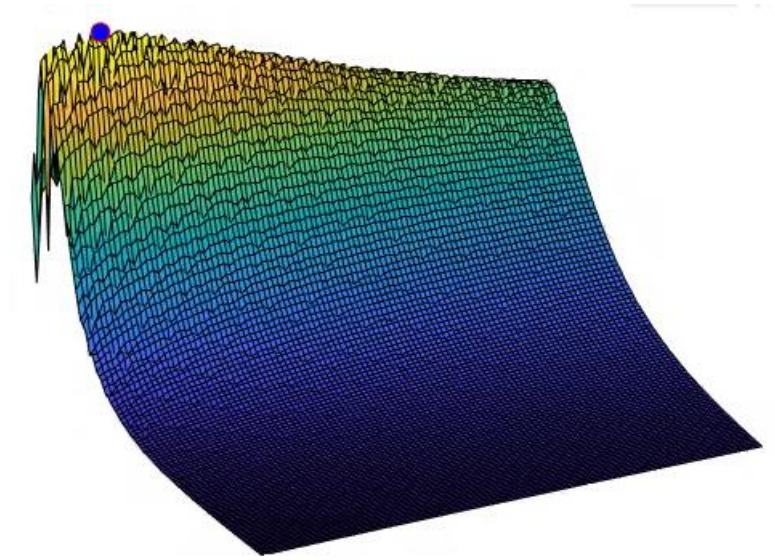


Figure 5. 6 - Fully converged population after 12 generations

Finally, after 12 generations the population has converged to the top of the power curve with most individuals sharing the same genes (K and C). In the above image this translates into what seems to be a single individual, whilst in reality it's multiple individuals sharing nearly identical genes.

These results alone show how well this method works and how after only a few generations the individuals manage to reach optimal locations in the search space.

Other graphical data used to evaluate the evolution progress of the individuals were the convergence plots of the stiffness and damping factors of each individual in the population for a given sea state as generations elapse. These plots show how well the individuals coalesce towards a common set of control parameters for a give sea state as generations pass.

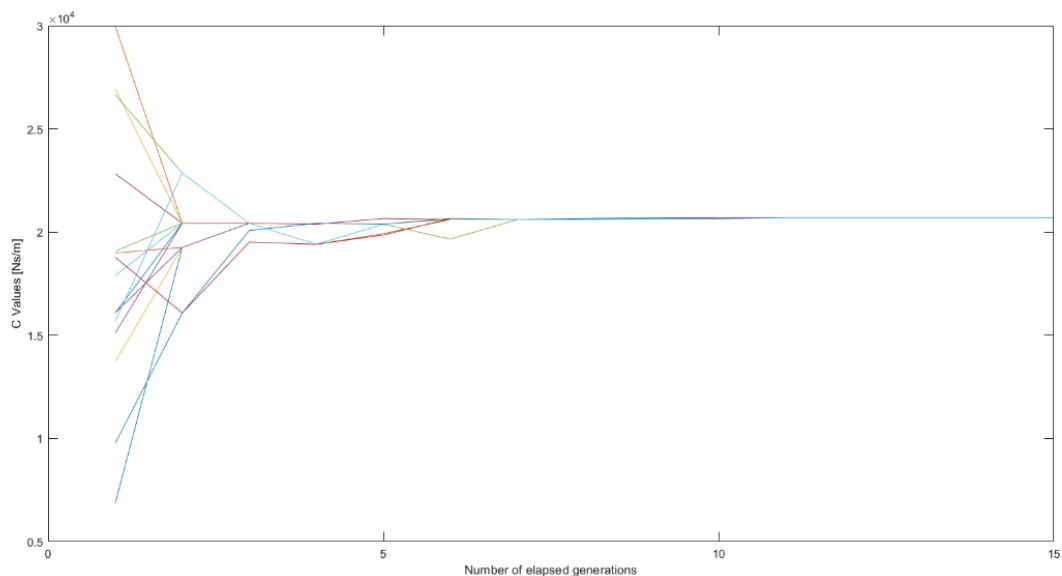


Figure 5. 7 - Evolution of damping values



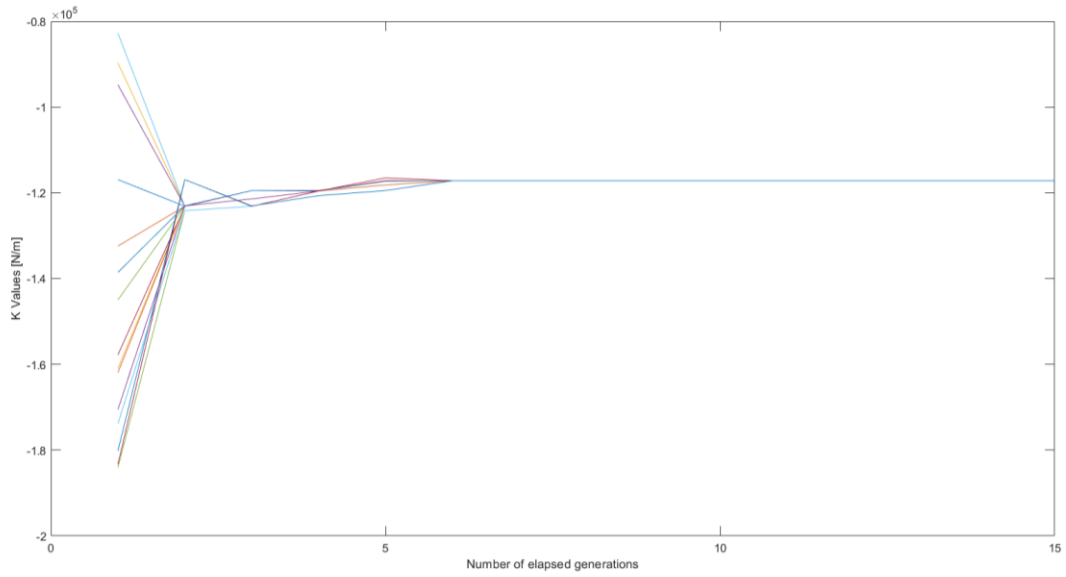


Figure 5. 8 - Evolution of stiffness values

Each line represents the evolution of the control parameter for a single individual in the population. Both graphs show how quickly can the population adapt to the newly encountered sea state and correctly converge to optimal control parameters.

Finally, another important result that was analyzed was the overall best fixed control parameters obtained for each sea state.

These results were picked as the best performing individual in terms of absorbed power for each sea state throughout the whole 90-day simulation.

The following table and graph report the findings for each of the 14 sea states.

Sea state number	C value [Ns/m]	K value [N/m]
1	$2,0567 \cdot 10^4$	$-1,1788 \cdot 10^5$
2	$1,3886 \cdot 10^4$	$-1,5681 \cdot 10^5$
3	$9,0838 \cdot 10^3$	$-1,7266 \cdot 10^5$
4	$7,2034 \cdot 10^3$	$-1,8010 \cdot 10^5$
5	$2,1256 \cdot 10^4$	$-1,1059 \cdot 10^5$
6	$1,3617 \cdot 10^4$	$-1,5666 \cdot 10^5$
7	$7,8073 \cdot 10^3$	$-1,7337 \cdot 10^5$
8	$6,2768 \cdot 10^3$	$-1,8046 \cdot 10^5$
9	$1,9589 \cdot 10^4$	$-1,0172 \cdot 10^5$
10	$1,3376 \cdot 10^4$	$-1,5624 \cdot 10^5$
11	$8,5141 \cdot 10^3$	$-1,7255 \cdot 10^5$
12	$1,2176 \cdot 10^4$	$-1,4903 \cdot 10^5$
13	$8,7092 \cdot 10^3$	$-1,7320 \cdot 10^5$
14	$8,3835 \cdot 10^3$	$-1,7076 \cdot 10^5$

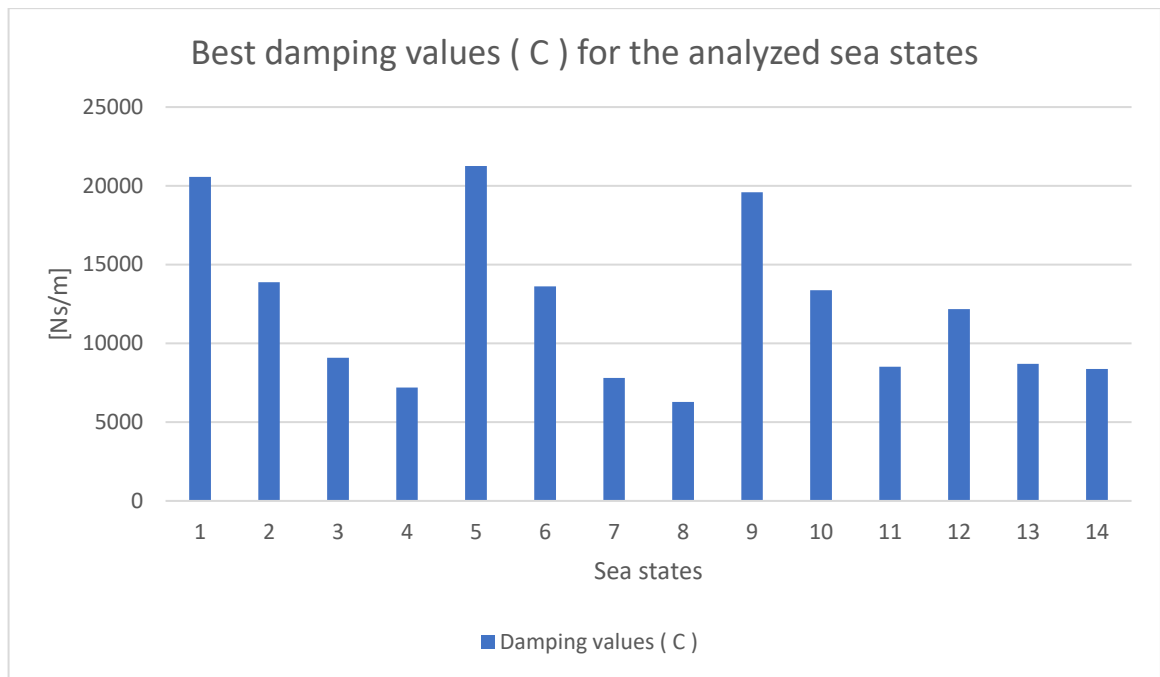


Figure 5. 9 - Best constant damping values from genetic algorithm optimization

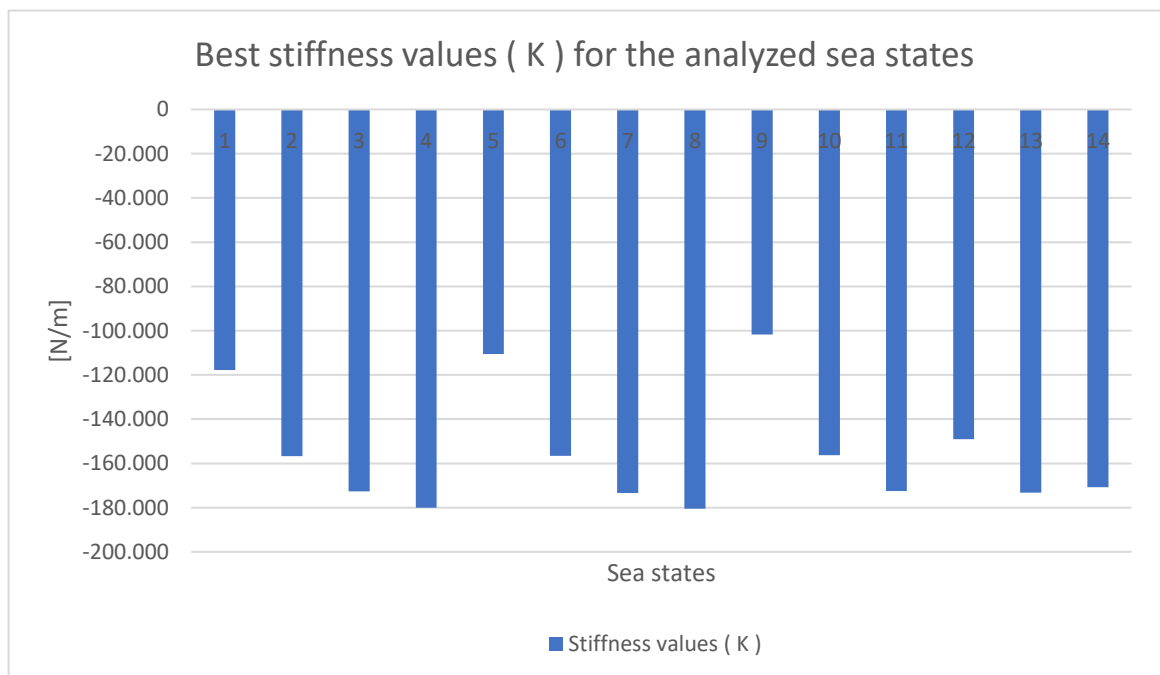


Figure 5. 10 - Best stiffness values form genetic algorithm optimization

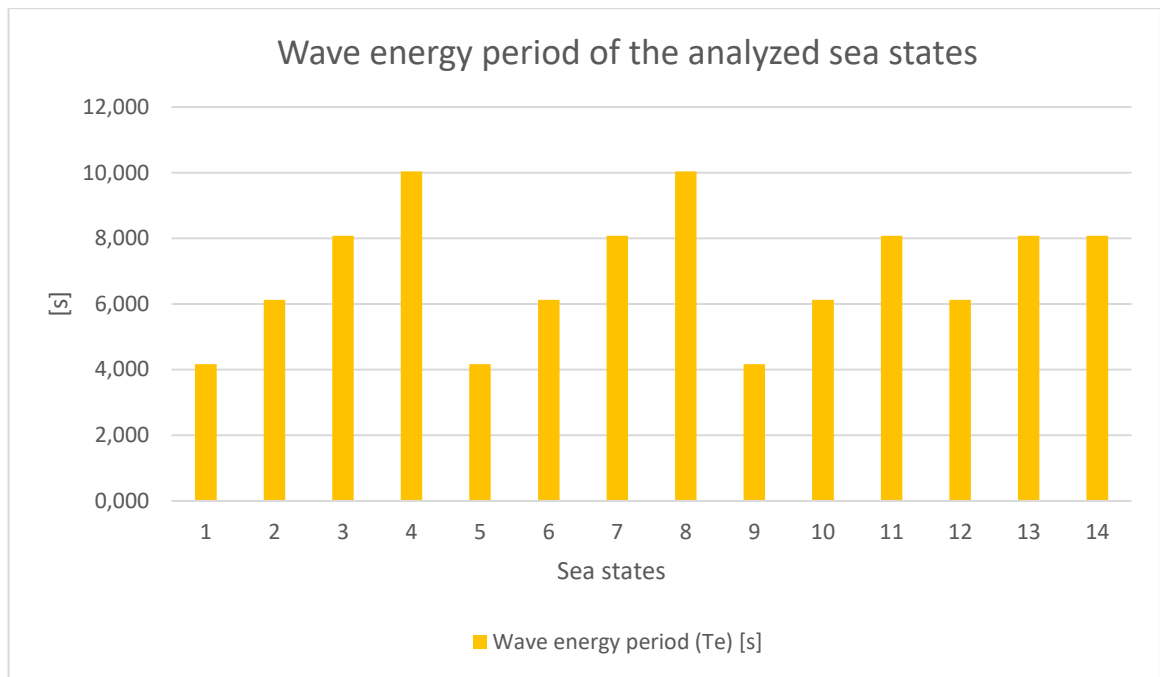


Figure 5. 11 - Wave energy periods of the 14 generated sea states

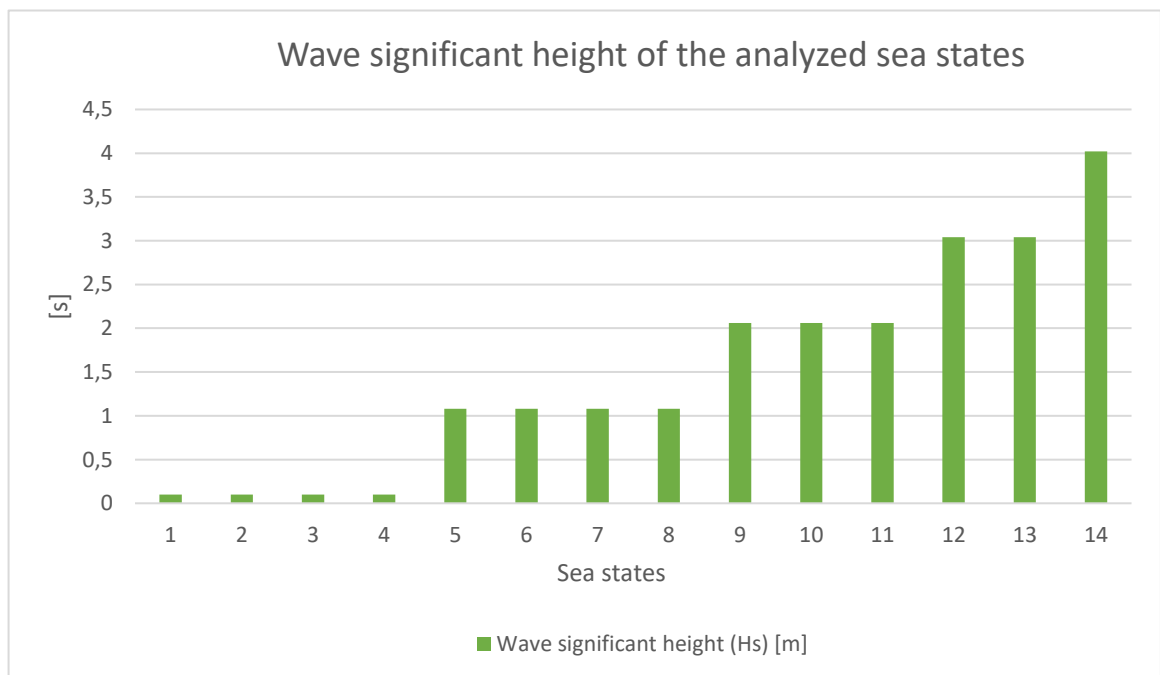


Figure 5. 12 - Wave significant height of the 14 generated sea states

By comparing the results of the optimization process with the graphs related to the wave energy period and significant wave height of each sea state it's clear how the wave period is the driving force behind the most significant changes in both the stiffness and damping control parameters.

### 5.3 Expected Annual Energy Production

To understand what impact the above results would have on energy production in the long term the *Expected Annual Energy Production (EAEP)* was computed.

$$EAEP_{ts} = \frac{3600 \cdot 24 \cdot 365}{100} \sum_{\omega=1}^{N_{\omega}} Occ_{\% \omega} P_{abs_{\omega}}(C_{\omega, t_s}, K_{\omega, t_s})$$

Where  $EAEP_{ts}$  is the Expected Annual Energy Production computed at time  $t_s$  and where:

- $\omega$  : Sea state indicator
- $Occ_{\% \omega}$  : Occurrence percentage of sea state  $\omega$
- $N_{\omega}$  : Total number of sea states
- $P_{abs_{\omega}}(C_{\omega, t_s}, K_{\omega, t_s})$  : Power absorbed in sea state  $\omega$  with the best C and K parameters obtained until time  $t_s$ .

This parameter represents the annual productivity that would be obtained if for each sea state characterizing the deployment site the best control action seen until time  $t_s$  was fixed and used over an entire year. In other words, it's as if the optimization procedure was stopped at a given time, and whatever best results had been found up until such time were then used each time a sea state was encountered, over a whole year of deployment with mixed and random sea states.

The resulting productivity is reported in the following graph.

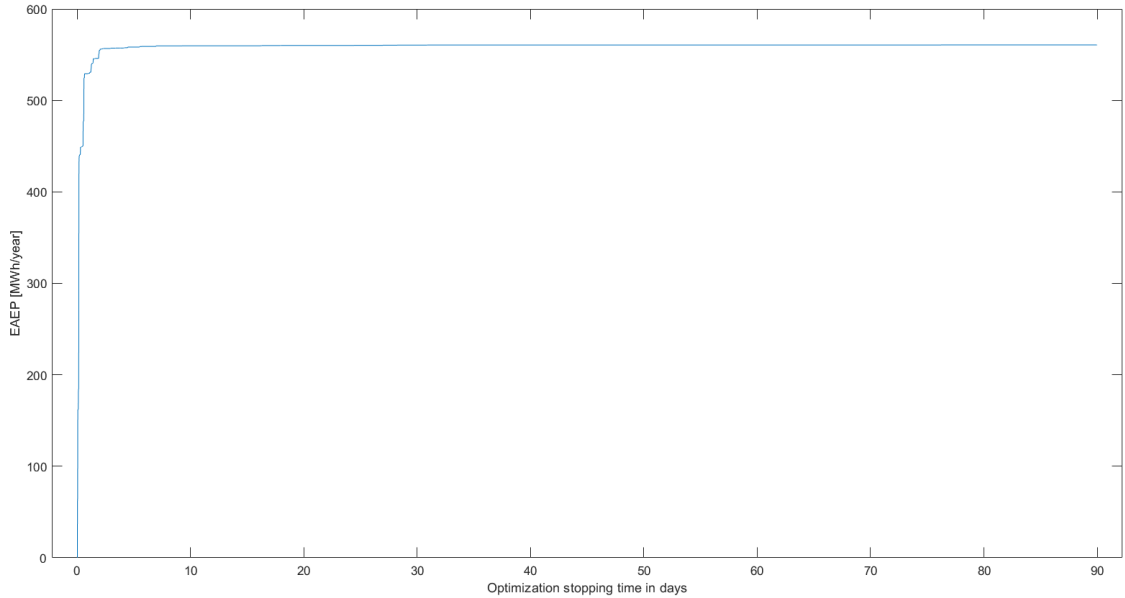


Figure 5.13 - Expected Annual Energy Production over the 90-day simulation

As can be seen, after only a few days at sea, the Expected Annual Energy Production has nearly peaked. This is due to both the ability of the genetic algorithm to quickly hone in to the optimal control parameters for each sea state, but also due to the structure of the problem itself. Most of the

absorbed power for the considered deployment location comes from the most frequently encountered sea states, even though they might not necessarily be the most energetic. Thus, although after a short period of time the most energetic sea states might not have been encountered often or at all, the parameters related to the most frequent sea states have already been optimized, which in turn means that most of the power production over a whole year is at near optimal performance.

## 6 – The neural network

### 6.1 Introduction

Having achieved excellent performance with a discrete model free control strategy through the use of genetic algorithms to optimize the control parameters of a reactive control law, the focus was now shifted on developing a continuous control strategy based on the same reactive law.

The idea is to train a neural network to produce the optimal time-varying sequence of control parameters, no longer based on a description of the current sea condition through discrete sea states as used by the genetic algorithms but based on the current forces acting on the point absorber.

This approach could ideally create a continuous control over the control parameters on a wave-by-wave basis, potentially allowing for much greater energy extraction thanks to a tailored and specific control compared to an averaged control obtained when describing the current sea conditions using statistical variables such as significant height and energy period.

The modern concept of artificial neural networks (see appendix B) was first introduced in the 1940s with the work of McCulloch and Pitts [70] who showed that artificial neural networks possess, at least in principle, the ability to compute any logical function.

Neural networks are mathematical computing systems inspired by the biological neural networks present in our brains. Similarly, to a biological neural network, ANNs are composed of multiple interconnected nodes called *artificial neurons* or simply *neurons* which are capable of receiving and processing information which can then be passed through to other neurons in the network.

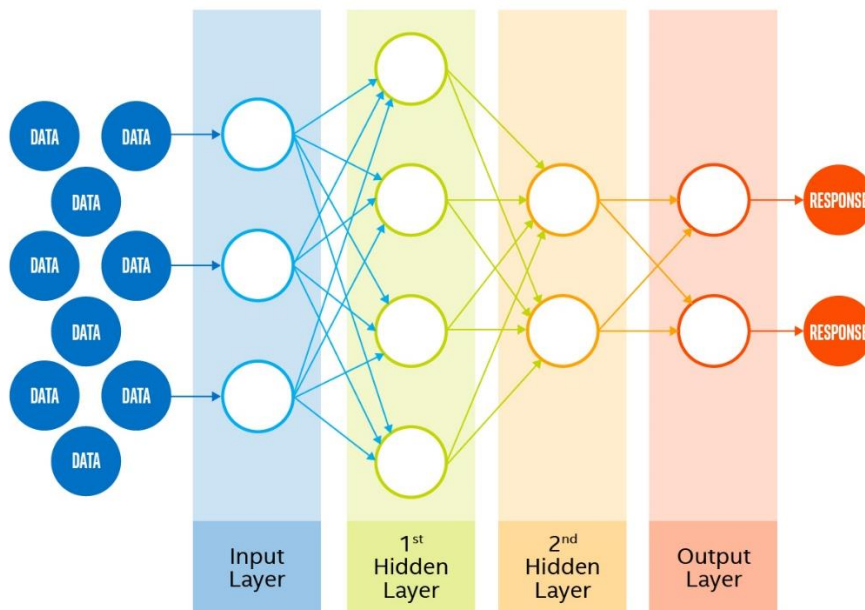


Figure 6. 1 - Example of a simple neural network [71].

Given a certain input, the network is able to process the information to produce a corresponding output. To achieve the desired network behavior, may it be image classification, pattern recognition or any other of the many applications in which ANNs can be used, the network must first be trained. Training involves using a set of known inputs and desired corresponding outputs to teach or train the network to be able to correctly match the training set inputs to the correct desired training outputs. This training process can be achieved by feeding the network with the training input samples and by comparing the current output with the ideal desired output. The error between these two values is then used in what is known as a performance or loss function to determine how well the network is behaving on the assigned task. The magnitude of the loss function indicates if the network is performing well or badly with the given training set. With the loss function known, the gradient information of the loss function is then used to update the weights and biases of all the neurons in the network with the goal of decreasing the loss function at each iteration, in turn producing a neural network which performs well in the desired task. A more detailed explanation of the internal workings of neural networks and how they are trained can be found in appendix B.

Crucially then, the most important aspects of using a neural network for a given task lie within two main categories. The first consists in choosing the correct neural network structure, including layer size and number, layout, performance function metric, optimization algorithm, learning rate and many other hyperparameters affecting the network performance. The second important aspect to consider is the training set used to train the network. A correct training set is fundamental since it's the only external information that will be used to train the network, where a badly chosen training set may cause the network to underperform during its deployment on the field.

The first step towards a functioning neural network is to decide what structure and type of network is needed for the problem at hand.

Several types of networks exist, each with its own specific structure and working principle based on the problem to be solved.

The problem faced in this work can be considered a sequence-to-sequence regression problem where for a given time series sequence of inputs, a corresponding sequence of outputs must be produced.

For this kind of problem, two network architectures were used and tested: a deep feed-forward neural network and a Long-Short Term Memory (LSTM) neural network.

In the next chapters the chosen neural network architecture will be presented together with the how training set was conceived and used.

## 6.2 The Feed-forward neural network

A deep feed-forward neural network is one of the simplest forms of neural network. It consists of multiple successive network layers where all the layers are fully connected (i.e., each neuron in a given layer is connected to all the neurons in the successive layer) and where information can only flow forwards, so from the input towards the output.

The number of layers and the number of neurons in each layer is a design specification, as are the activation functions used in each layer, the number of neurons in the input and output layer, and many other parameters which will be listed and described later on.

The choice of using a feed-forward network comes from the fact that this kind of network is one of the most versatile but simple networks, and also one of the most diffused.

### 6.2.1 The network structure

Multiple preliminary tests were performed to gage the ability of these networks to map a time varying input to a desired output. These tests were performed using a time varying sequence of forces measured on a WEC device and using these forces as an input to a network which was tasked to map such forces to the corresponding time varying vertical displacement of the WEC.

These preliminary tests suggested that using a deeper network, usually with a number of layers larger than 5, allowed to learn more complex dependencies between the inputs and the corresponding outputs, resulting in a smaller final learning error.

All of the tests were performed using MATLAB's Experiment Manager app, which allows to design and run experiments to train and compare deep learning networks. During the tests, both the parameters regarding the structure of the net and the training parameters such as solver type, input normalization and activation functions were evaluated and compared to find a network layout with good performance for a sequence-to-sequence regression problem.

The final network structure was composed of:

- a sequence input layer
- 7 fully connected hidden layers each with 256 neurons
- a fully connected output layer with two outputs
- a scaling layer to scale the outputs so to match the bounds on the stiffness and damping control parameters presented in chapter 4.1.2
- a regression layer.

The whole neural network structure was built in MATLAB using the “layers” function to define the needed layers and their hyperparameters. For more information, please refer to the official MathWorks web page at: <https://www.mathworks.com/help/deeplearning/ug/list-of-deep-learning-layers.html>.



The sequence input layer, as its name suggests, is a specific layer used to feed a sequence as an input to the neural network. The main parameters to be set for this layer are the input size, which refers to how many inputs are fed to the network simultaneously at any given time step, and the normalization type. The normalization chosen by analyzing the preliminary tests was a ‘zscore’ normalization where each input has the mean subtracted before being divided by the standard deviation value.

Both the mean and standard deviation of the whole training set are calculated automatically at training time. The value of the chosen input size will be covered in the chapter related to how the network was trained.

Immediately after the sequence input layer, 7 fully connected layers were placed as the hidden layers of the network. These layers act as basic neural network layers which multiply the input by a weight matrix and then add a bias vector. All the hyperparameters related to the fully connected layers, such as weight and bias initializer, weight and bias learn rate factor and weight and bias L2 factors were left in their default mode.

<b>Weight Initializer</b>	<b>Bias Initializer</b>	<b>Weight learn rate factor</b>	<b>Bias learn rate factor</b>	<b>Weight L2 factor</b>	<b>Bias L2 factor</b>
glorot	zeros	1	1	1	0

After each fully connected layer, an activation function layer was placed acting as the non-linearity function in the layer. For all layers, a reluLayer was used as activation function.

After the seven hidden layers, the output section was composed by a fully connected layer with two outputs (C and K of the reactive control law) accompanied by a tanhLayer as a squishing activation function. After the fully connected layer, a scaling layer was used to scale the bounded output from the tanhLayer from values ranging between -1 and 1 to values ranging from the minimum to the maximum admissible values for C and K calculated in chapter 4.1.2.

As a last layer, a regression layer was used so to compute the half-mean-squared-error loss for the regression task.

### 6.2.2 Training the feed forward network

The training of a neural network is the process which allows a neural network to learn how to perform a given task by examining examples of how such task should be executed and by modifying its internal parameters to try and perform the desired function as best as possible.

Training is achieved by feeding a network with a training set of input data which has a known corresponding desired set of outputs.

For a sequence-to-sequence regression problem, an input sequence is fed to the network, and the output produced by the network is then compared to the known desired output sequence.

The error between the desired output and the actual output is then used as information on how well the network is currently performing. An optimization procedure which uses the gradient information of the error function is then used to update the weights and biases of the network to try and minimize the error. This process of feeding inputs, comparing the produced output to the desired output, and updating the weights and biases is performed iteratively until a predefined stopping condition is met, eventually leading to a more performing network. For a more detailed description of how the training process occurs, please refer to appendix B.

When training a network, the most important considerations to be made are those regarding the training input/output sequence and the hyperparameters to be used for the training process.

The training input, and the corresponding desired output, must be carefully chosen since they will be the driving force behind the training of the network and will be the only information seen by the network before deployment, so it's crucial that the chosen training set input allows the network to be able to learn a correct mapping to the desired output.

With the above considerations in mind, an analysis of the problem at hand is needed to understand the choice of the training set used.

The goal for the neural network is to map the varying force input to the corresponding optimal control parameters for the given sea state and to possibly learn interdependencies between the sea state and corresponding control parameters so to be able to handle any force input which was not directly shown in the training examples, or which does not belong to any of the sea states seen before.

To achieve this, it is not sufficient to feed the network with a single input sequence representing the force time series and mapping such force to the control values because this would mean mapping a single force input value to a single corresponding couple of control parameters at each time step. This is obviously not the goal since past force inputs influence the current WEC dynamics, thus the current force acting on the WEC is not sufficient to determine what the correct control values should be. A solution to this problem was to feed at each time step, not a single input representing the current force acting on the point absorber, but multiple inputs representing present and past values of force acting on the point absorber.

This process creates a sort of memory of the past force values which is then used in conjunction with the present force to determine the correct output values for the two control parameters.

What needed to be determined was how many inputs were needed, which translates in how much past information was needed at each time step to train the network.

Preliminary tests were performed with MATLAB's Experiment Manager which indicated that taking multiple periods of the exciting force yielded better output mapping, hence, considering a sampling frequency of 0.1s and the wave heave forces generated in the sea states presented in chapter 5.1, 600 inputs were used at each time step in order to guarantee that even for the wave excitation forces with lower frequency, at least 4 whole periods of the varying force would be used as inputs to the net.

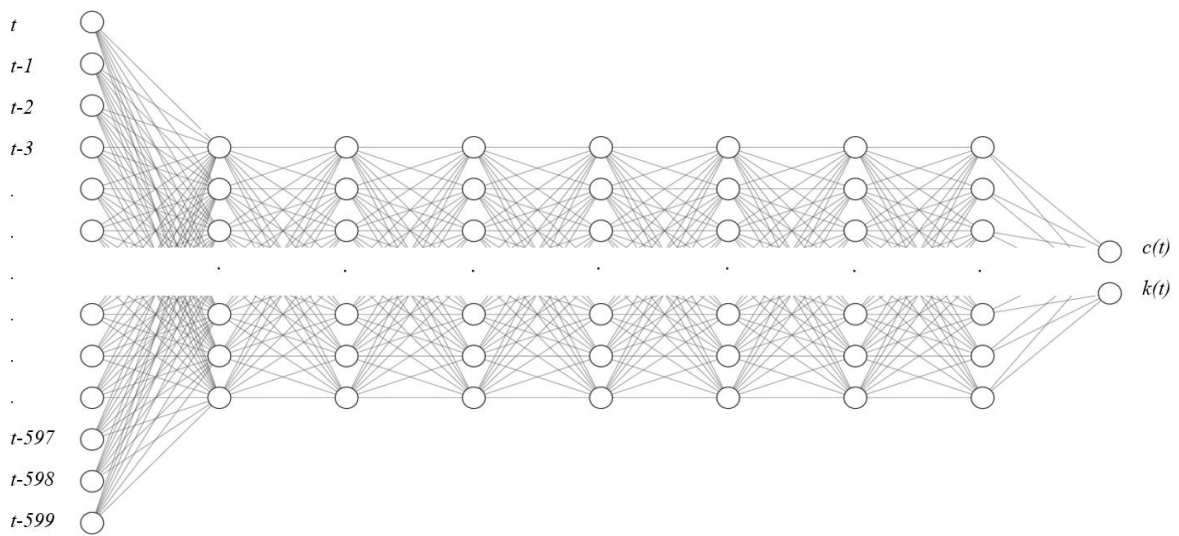


Figure 6. 2 - Qualitative representation of the feed-forward neural network

With the network structure and training input strategy defined, the necessary data for training had to be defined and collected.

The devised strategy consisted in training the network every seven days with the information gathered by the top three performing point absorbers for each sea state, right from the beginning of the launch of the point absorbers when using the genetic algorithm to optimize the control parameters in each sea state.

As each week passes, the information on how each tested control strategy for each sea state encountered is gathered, and for each discrete sea state encountered in the given week, the top three performing control strategies, together with the relative measured heave force values are registered and stored. This information is then used to create a training set composed of an input vector of all the successive measured forces and two corresponding output vectors of the most effective control values related to each of the input force vectors.

The input force vector is then rearranged in a matrix form in order to fit a network paradigm which includes 600 inputs.

The resulting training set is thus composed of a matrix of input forces which were recorded for each sea state encountered during the week from the top three performing point absorbers in a given sea state over the duration of a single generation of the genetic algorithm, which in turn is related to a single sea state evolving for twenty minutes. Each of the recorded force vectors lasting twenty minutes is then associated to two corresponding vectors of the same length which represent the fixed control parameters used by the selected top performing point absorbers during the twenty-minute time period. The training procedure thus associates the input force vector, to the corresponding control parameters which for the given sea state and corresponding force were the best performing parameters found in the given week.

The above considerations result in a vector of recorded forces for each given sea state encountered in the week which has 36003 entries, which corresponds to three stacked vectors, one for each of the top three performing point absorbers in the given sea state each with 12001 entries, which correspond to a twenty-minute force reading, with a sampling frequency of 0.1s.

As stated before, all the vectors for all the sea states encountered are then stacked and placed in matrix form to allow for a 600-input net structure.

The goal of this training setup is to create a network which can map input heave forces in to instantaneous and continuous control parameters by learning possible interdependencies between all the forces seen and the corresponding optimal control parameters found by the populations in the genetic algorithm.

With the network structure, training set and training strategy defined, only the hyperparameters used for training had to be defined before training could commence.

As for the network structure, different network training hyperparameter values were evaluated using the Experiment Manager tool available on MATLAB in the Machine Learning and Deep Learning toolbox. The main hyperparameters tested were:

- Initial learning rate
- Solver type
- Mini-Batch size
- Input normalization type

The final hyperparameters chosen for training were:

<b>Initial learning rate</b>	<b>Solver type</b>	<b>Mini-Batch Size</b>	<b>Input normalization</b>	<b>Shuffle</b>	<b>Max Epochs</b>
0.001	Adam	128	zscore	Never	150

Where all the other many hyperparameters for training a neural network were kept standard according to the standard MATLAB ‘trainingOptions’ function when training a neural network.

The training process was performed using the ‘trainNetwork’ function and simply involved the definition of the input training input sequence, output sequence, layer types and training options defined in this chapter. The training process was performed on a workstation belonging to the MOREnergy research lab because of the computation weight was too much for a standard PC.

### 6.2.3 Testing the feed forward network

The trained network could finally be used to verify its capabilities in mapping an input heave force reading in to control parameters which would potentially allow for a greater energy absorption than what could be obtained with the genetic algorithms.

To test the neural network, a Simulink model was built which would simulate the behavior of the point absorber model in conjunction with the neural network. A wave scenario was generated in MATLAB and the resulting heave force was fed to the neural network as an input, thus generating a corresponding vector of control parameters. This vector was then used in Simulink together with the original wave heave force vector to simulate how the point absorber would behave under such a wave scenario if it adopted as control parameters, the parameters created by the neural network.

The Simulink environment setup was as follows.

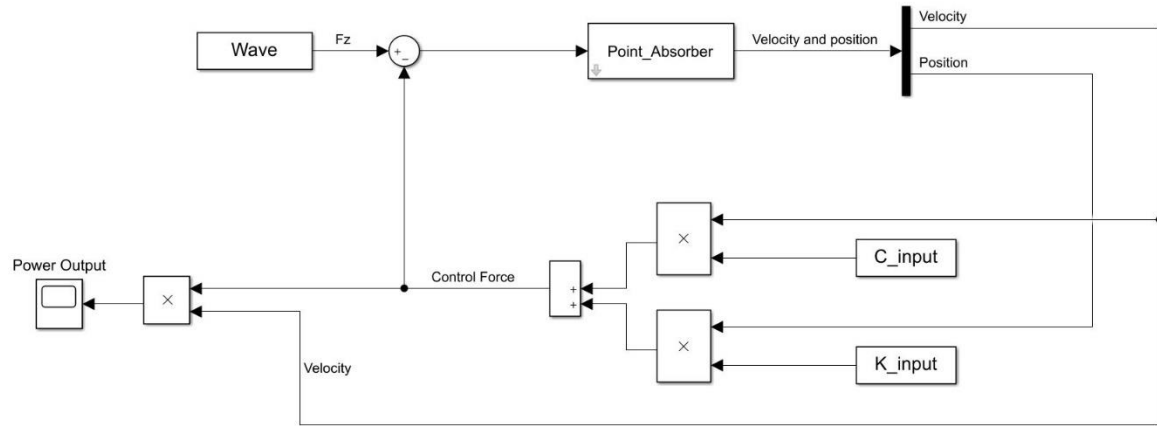


Figure 6. 3 - Simulink model for point absorber and neural network simulation

Where C\_input and K\_input are the control parameter vectors created by the neural network when fed the wave heave force Fz. Both vectors were created through a MATLAB script separately from the Simulink model.

Tests were performed for each of the fourteen sea states to compare the behavior of the model when using the control vectors generated by the neural network and when using the constant control parameters obtained during the optimization phase involving the genetic algorithms.

The tests were conducted one sea state at a time, and for each sea state, a force vector  $F_z$  was generated. This force vector was then used both directly in the Simulink environment and also as an input to the neural network so to obtain the corresponding control parameter vectors to be used in the Simulink simulation. A comparison was drawn between either using the generated control parameter vectors as control inputs in the model or using the best constant control parameters for the sea state under test found during the previous optimization using genetic algorithms.

The generated force vector corresponded to a 20-minute representation of the sea state, which is the same time span used by each generation of the genetic algorithm to evaluate a population.

The analysis of the results will mainly focus on the comparison of the mean power output between the two methods, while also taking into account features like maximum and minimum power peaks.

Sea state number	Te & Hs values [s]; [m]	Mean power constant C & K [W]	Mean power with N.N control [W]	Max power constant C & K [W]	Max power with N.N control [W]	Min power constant C & K [W]	Min power with N.N control [W]
1	4,17; 0,1	$7,953 \cdot 10^1$	$7,932 \cdot 10^1$	$2,013 \cdot 10^3$	$2,284 \cdot 10^3$	$-1,354 \cdot 10^3$	$-1,703 \cdot 10^3$
2	6,13; 0,1	$2,166 \cdot 10^2$	$2,105 \cdot 10^2$	$1,904 \cdot 10^4$	$2,063 \cdot 10^4$	$-1,599 \cdot 10^4$	$-1,820 \cdot 10^4$
3	8,08; 0,1	$3,981 \cdot 10^2$	$3,937 \cdot 10^2$	$6,916 \cdot 10^4$	$7,744 \cdot 10^4$	$-6,768 \cdot 10^4$	$-7,063 \cdot 10^4$
4	10,04; 0,1	$5,860 \cdot 10^2$	$5,483 \cdot 10^2$	$1,376 \cdot 10^5$	$1,181 \cdot 10^5$	$-1,366 \cdot 10^5$	$-1,249 \cdot 10^5$
5	4,17; 1,08	$9,480 \cdot 10^3$	$9,451 \cdot 10^3$	$2,400 \cdot 10^5$	$2,697 \cdot 10^5$	$-1,482 \cdot 10^5$	$-1,829 \cdot 10^5$
6	6,13; 1,08	$2,527 \cdot 10^4$	$2,472 \cdot 10^4$	$2,260 \cdot 10^6$	$1,755 \cdot 10^6$	$-1,900 \cdot 10^6$	$-1,376 \cdot 10^6$
7	8,08; 1,08	$4,609 \cdot 10^4$	$3,964 \cdot 10^4$	$9,794 \cdot 10^6$	$8,718 \cdot 10^6$	$-9,684 \cdot 10^6$	$-7,295 \cdot 10^6$
8	10,04; 1,08	$6,872 \cdot 10^4$	$5,483 \cdot 10^4$	$1,893 \cdot 10^7$	$1,966 \cdot 10^7$	$-1,884 \cdot 10^7$	$-1,664 \cdot 10^7$
9	4,17; 2,06	$3,440 \cdot 10^4$	$3,408 \cdot 10^4$	$8,210 \cdot 10^5$	$7,982 \cdot 10^5$	$-4,898 \cdot 10^5$	$-4,881 \cdot 10^5$
10	6,13; 2,06	$9,220 \cdot 10^4$	$9,256 \cdot 10^4$	$8,358 \cdot 10^6$	$7,166 \cdot 10^6$	$-7,010 \cdot 10^6$	$-5,926 \cdot 10^6$
11	8,08; 2,06	$1,689 \cdot 10^5$	$1,630 \cdot 10^5$	$3,047 \cdot 10^7$	$5,954 \cdot 10^7$	$-3,056 \cdot 10^7$	$-5,781 \cdot 10^7$
12	6,13; 3,04	$2,223 \cdot 10^5$	$2,062 \cdot 10^5$	$2,431 \cdot 10^7$	$1,948 \cdot 10^7$	$-2,167 \cdot 10^7$	$-1,788 \cdot 10^7$
13	8,08; 3,04	$3,667 \cdot 10^5$	$2,835 \cdot 10^5$	$6,979 \cdot 10^7$	$1,113 \cdot 10^8$	$-6,755 \cdot 10^7$	$-1,285 \cdot 10^8$
14	8,08; 4,02	$6,908 \cdot 10^5$	$3,222 \cdot 10^5$	$1,914 \cdot 10^8$	$1,582 \cdot 10^8$	$-1,833 \cdot 10^8$	$-1,560 \cdot 10^8$

The above results show that in most cases, the constant control parameters outperform the continuous control vector produced by the neural network. A summary of the percentage difference in mean absorbed power, peak positive power and peak negative (reactive) power between the constant control strategy versus the continuous control produced by the neural network is presented in the following table where a positive percentage indicates a higher power reading for the neural network implementation.

Sea state number	Percentage mean power difference	Percentage max positive power difference	Percentage min reactive power difference
<b>1</b>	-0,2641	13,4625	25,7755
<b>2</b>	-2,8163	8,3508	13,8211
<b>3</b>	-1,1052	11,9722	4,3587
<b>4</b>	-6,4334	-14,1715	-8,5652
<b>5</b>	-0,3059	12,3750	23,4143
<b>6</b>	-2,1765	-22,3451	-27,5789
<b>7</b>	-13,9944	-10,9863	-24,6696
<b>8</b>	-20,2125	3,8563	-11,6773
<b>9</b>	-0,9302	-2,7771	-0,3471
<b>10</b>	+0,3905	-14,2618	-15,4636
<b>11</b>	-3,4932	95,4053	89,1688
<b>12</b>	-7,2425	-19,8684	-17,4896
<b>13</b>	-22,6888	59,4784	90,2295
<b>14</b>	-53,3584	-17,3459	-14,8936

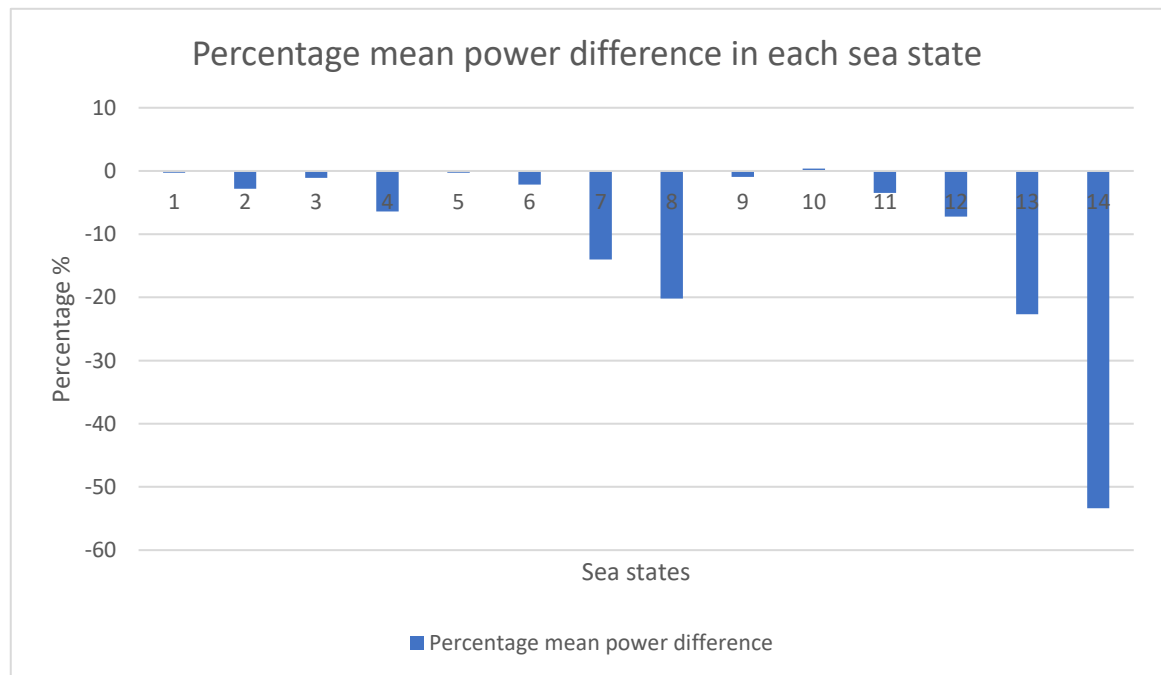


Figure 6. 4 - Percentage mean power difference between using constant control parameters or neural network parameters

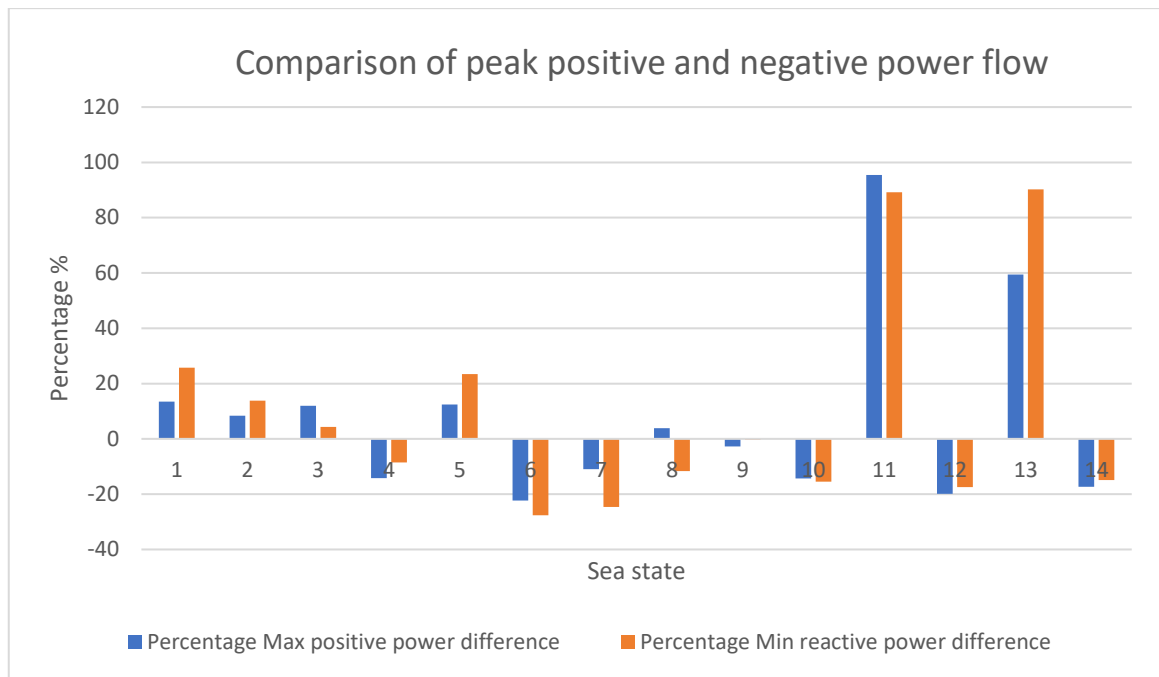


Figure 6. 5 – Max and min peak power difference when using constant control parameters or neural network parameters

As the above tables and graphs clearly show, the performance in terms of mean absorbed power is better in all but one case when using constant control parameters.

Despite this, in most cases the difference in the mean absorbed power is actually pretty small, with differences of only a few percent. On the other hand, some sea state scenarios, particularly scenarios 8, 13 and 14, showed a tremendous drop in performance, up to the point where in sea state number 14 the mean absorbed power with the neural network configuration was less than half of what was obtained with constant control parameters.

Although these results may seem disappointing, it is interesting to point out how in most sea states, a basic implementation of a simple feed forward neural network is able to produce a time varying control signal which is able to nearly match the performance of a carefully optimized pair of constant control parameters and in one of the sea states (number 10), it is even able to produce a control signal which leads to a slight increase in the mean absorbed power. Additionally, the neural network was also able to produce a control input which resulted in a smaller overall peak to peak power output in seven out of the fourteen sea states, which brings the obvious benefits of hardware downsizing and also the need to handle a smaller reactive power flow.

Another interesting result produced by the neural network is the shape of the produced control signals. In the next few graphs, some examples of the produced signals, together with the corresponding input heave force will be presented and analyzed.



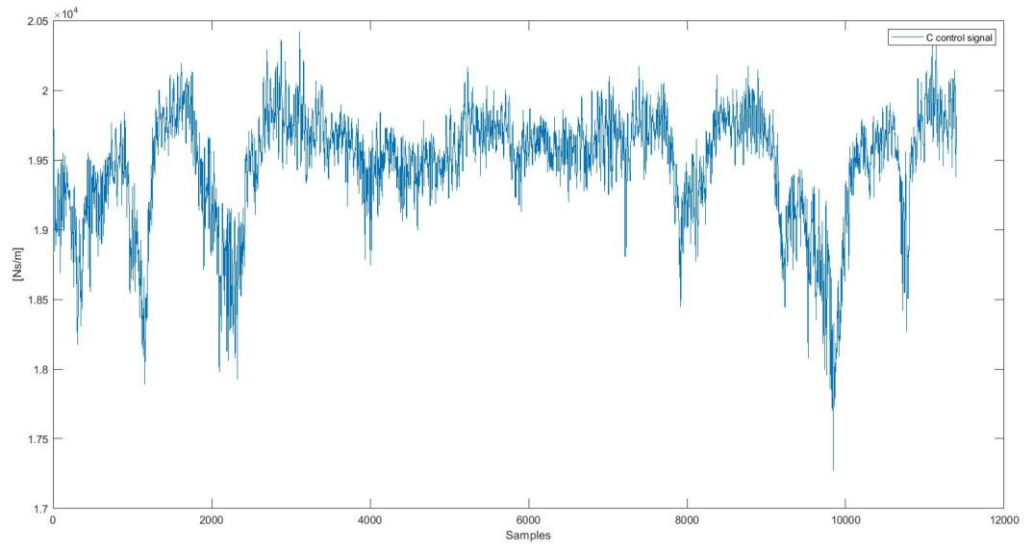


Figure 6. 6 - Control signal of damping control parameter 'C' generated for sea state number 5

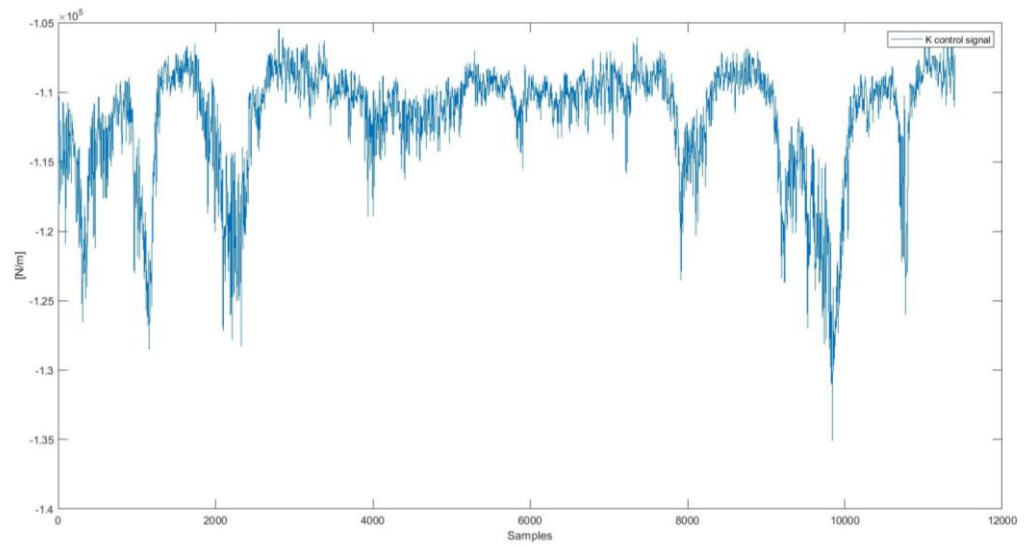


Figure 6. 7 - Control signal of stiffness control parameter 'K' generated for sea state number 5

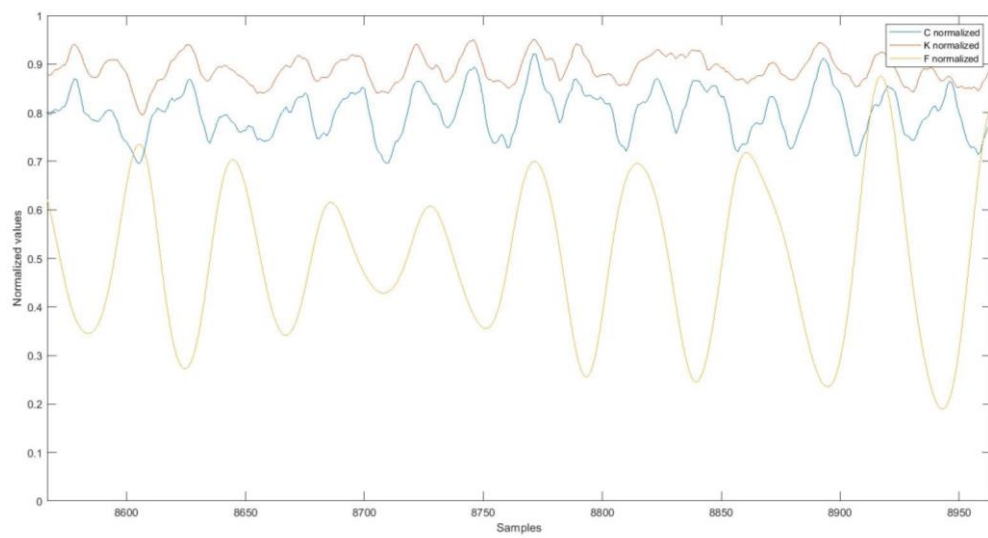


Figure 6. 8 - Normalized damping, stiffness and heave force signal for sea state number 5

Analyzing figure 6.6 it is possible to notice how the two control signals seem to be strongly related to one another, often following each other creating two nearly symmetrical shapes. This is surprising since the two control signals were not linked in any way during the training phase.

Another interesting feature to point out is how the control signals have a principal frequency component which seems to be quite similar to the principal frequency of the wave heave force represented in yellow. This suggests that, to some extent, the neural network is actually able to follow the incoming wave force and select an appropriate control signal on a sample-by-sample basis.

To confirm this, another plot was drawn using the data collected from sea state number 4, which compared to sea state number 5 has a wave energy period more than twice as long. The following plots show the control signals and force reading corresponding to sea state number 4.

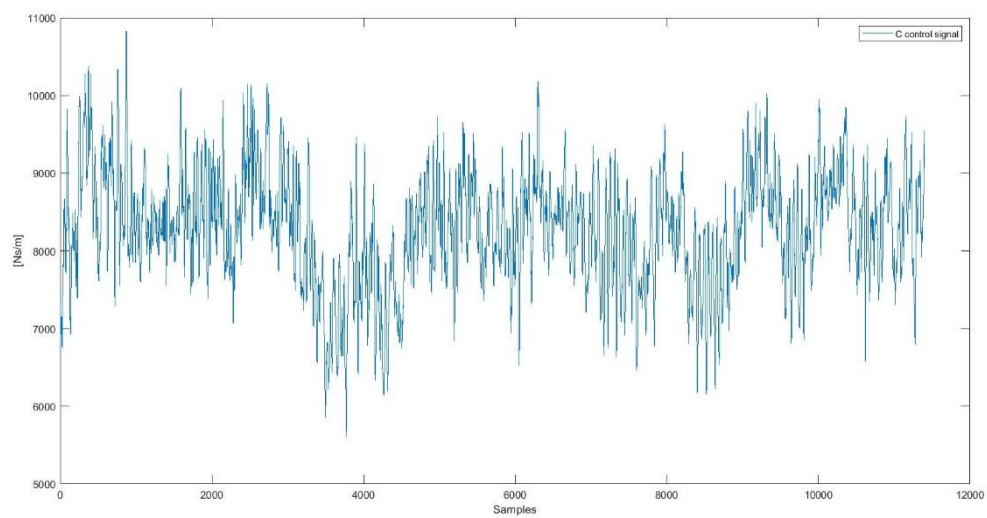


Figure 6. 9 - Control signal of damping control parameter 'C' generated for sea state number 4

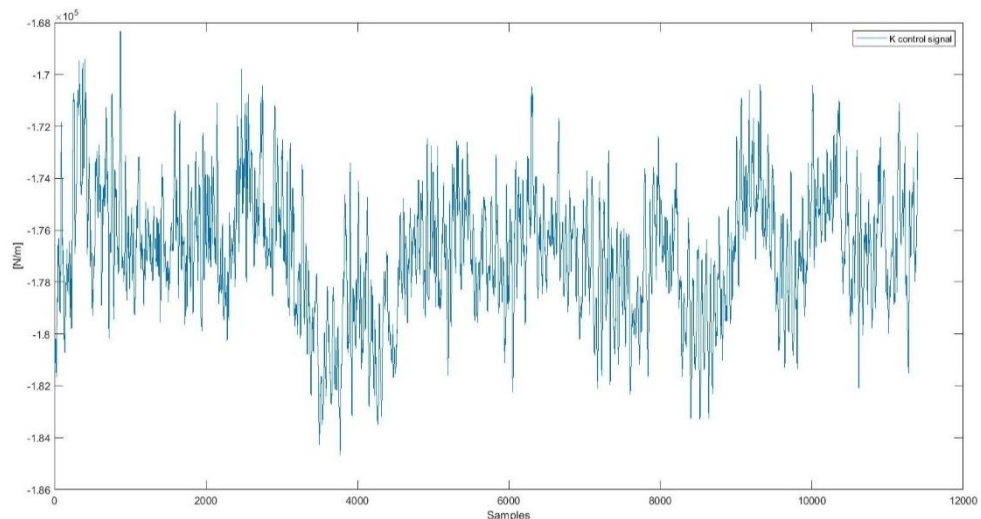


Figure 6. 10 - Control signal of stiffness control parameter 'K' generated for sea state number 4

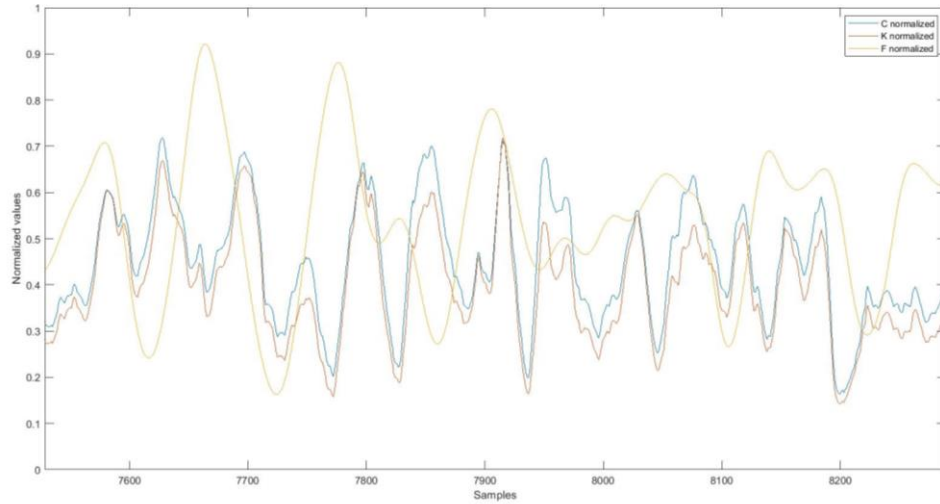


Figure 6. 11 - Normalized damping, stiffness and heave force signal for sea state number 4

As confirmed by figure 6.9, when the input heave force has a larger period, also the respective control signal generated by the neural network will have a corresponding larger oscillating period.

This result is something that might seem quite unremarkable at first sight, but it must be noted that at training time, each oscillating wave force signal was only coupled with a corresponding constant pair of optimal control parameters obtained from the genetic algorithms. This means that no information was given on how the control parameters should change in time on a wave-by-wave basis. It is thus quite remarkable that the neural network was able to produce a control signal which varied in time with a frequency very similar to the corresponding input wave heave force and that additionally, the produced control signal actually oscillated close to the optimal value obtained during the heuristic optimization, signifying that the neural network was actually able to distinguish wave scenarios and use both the optimal static control information as well as the time varying force to produce the control signal.

## 6.3 The Long Short Term Memory Neural Network

A long-short term memory neural network (LSTM neural network) is a type of recurrent neural network which, unlike feed forward neural networks, is able to store information and even feedback such information instead of only being able to pass information forwards from the input to the output nodes. LSTM neural networks were specifically developed to handle sequential data inputs and to solve problems related to these kinds of inputs, such as weather forecasting, stock market predictions, text translation, and many other similar applications. The main idea behind a LSTM neural network is to create a network which is able to operate in a manner which is close to how a human mind operates. As humans process information, they do not start their thinking from scratch at every time step, but instead, information is stored, and in some cases reused in order to process the current information not only on the basis of the current input, but also on the basis of past inputs which help to give the information context. Traditional feed forward networks cannot do this since they have no memory storing capability and also no feedback capability. On the other hand, LSTM neural networks have the ability to store, forget and reuse past information at each time step in order to help the process of deciding the current predicted output.

These processes are achieved through a system of gates, each with its own specific task, which ultimately allow the LSTM unit to hold on to what is considered as relevant past information and to discard other information which is not as relevant and to use such information to make decisions on the upcoming outputs. For a more detailed explanation of the workings of a LSTM neural network, please refer to the appropriate chapter in appendix B.

### 6.3.1 The network structure

When dealing with LSTM neural networks, the classic shape and structure seen for feed forward neural networks is no longer valid. Instead, each LSTM layer is composed of cells.

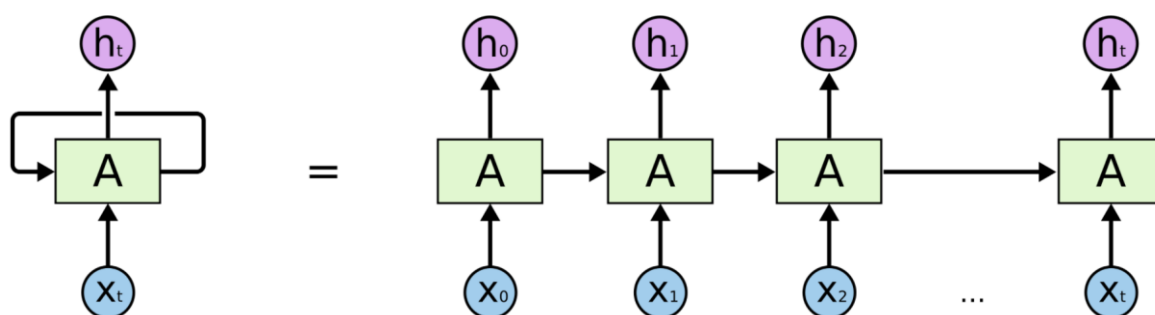


Figure 6. 12 - An unrolled LSTM network layer [75].

Each cell represents the LSTM layer for a given time instant. For this reason, LSTM layers are said to *unroll* to match the length of the input vector. This is simply an easier manner to visualize a LSTM layer. So instead of thinking of it as a layer which feeds back information to itself, we can imagine an LSTM cell for each time step, where information is passed from one cell to the next as time passes.

The number of cells in a LSTM layer, is thus a variable which does not need to be explicitly defined since it will depend on the length of the input vector.

What does need to be defined instead, is the number of *hidden units*.

Within each LSTM cell, the user can define how many hidden units are needed for the problem at hand. Hidden units can be thought of as basic LSTM building blocks where each hidden unit contains the basic elements needed for a LSTM network to work.

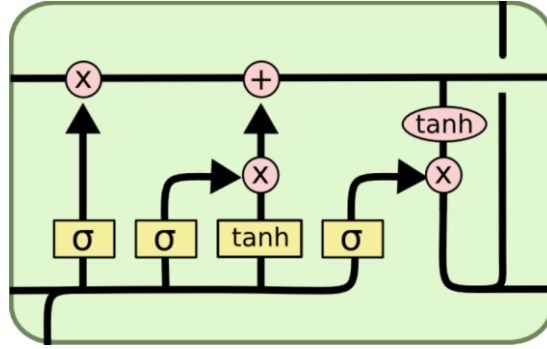


Figure 6. 13 - Basic LSTM unit

Within each cell, the hidden units are concatenated, so that information is processed by all the units at each time instant.

Each LSTM cell can work perfectly with just one hidden unit per cell, in fact in appendix B, each cell is only shown to have one hidden unit within it. But just as a feed forward neural network can work with only one neuron per layer, this is often not enough to have a structure which is able to solve more complex problem. So just as in feed forward neural networks the user might decide to increase the number of neurons, for LSTM layers the number of hidden units can be increased as well based on the problem complexity. For this work, each LSTM layer was equipped with 128 hidden units.

Just as for the feed forward network, the Experiment Manager tool was used to setup tests to gauge the performance of the network on a sequence regression task while varying some network hyperparameters.

The chosen network structure was composed of:

- a sequence input layer with rescale-symmetric normalization
- 3 bidirectional LSTM layers, each having 128 hidden units and set to sequence output mode
- a fully connected output layer with 2 outputs and a tanh activation function
- a scaling layer to scale the outputs so to match the bounds on the stiffness and damping control parameters
- a regression layer

Just as for the feed forward neural network, the LSTM network was built in MATLAB using the “layers” function to define the different network layers.

### 6.3.2 Training the LSTM network

Just as for the feed forward neural network, training was performed by gathering data every seven days from the top three performing point absorbers for each sea state during the optimization performed by the genetic algorithm. This meant that the used training set was exactly the same as for the feed forward network, with the difference that LSTM networks are inherently able to store past information, so there was no need to rearrange the heave force data into matrix form.

With the network structure, training set and training strategy defined, only the hyperparameters used for training had to be defined before training could commence.

As for the network structure, different network training hyperparameter values were evaluated using the Experiment Manager tool available on MATLAB in the Machine Learning and Deep Learning toolbox. The main hyperparameters tested were:

- Initial learning rate
- Solver type
- Mini-Batch size
- Input normalization type

The final hyperparameters chosen for training were:

<b>Initial learning rate</b>	<b>Solver type</b>	<b>Mini-Batch Size</b>	<b>Input normalization</b>	<b>Shuffle</b>	<b>Max Epochs</b>
0.001	Adam	1	rescale-symmetric	Never	200

Where all the other many hyperparameters for training a neural network were kept standard according to the standard MATLAB ‘trainingOptions’ function when training a neural network.

The training process was performed using the ‘trainNetwork’ function and simply involved the definition of the input training input sequence, output sequence, layer types and training options defined in this chapter. The training process was performed on a workstation belonging to the MOREnergy research lab because of the computation weight was too much for a standard PC.

### 6.3.3 Testing the LSTM network

To test the neural network, the same Simulink model used for the feed forward neural network test was used. A wave scenario was generated in MATLAB and the resulting heave force was fed to the neural network as an input, thus generating a corresponding vector of control parameters. This vector was then used in Simulink together with the original wave heave force vector to simulate how the point absorber would behave under such a wave scenario if it adopted as control parameters, the parameters created by the neural network.

The Simulink environment setup was as follows.

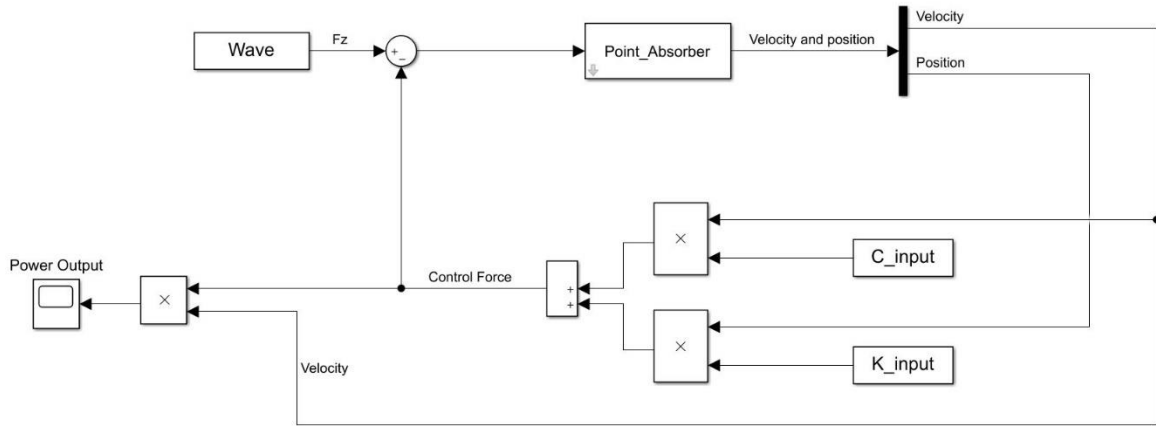


Figure 6. 14 - Simulink model for point absorber and neural network simulation

Tests were performed for each of the fourteen sea states to compare the behavior of the model when using the control vectors generated by the neural network and when using the constant control parameters obtained during the optimization phase involving the genetic algorithms.

The tests were conducted one sea state at a time, and for each sea state, a force vector  $F_z$  was generated. This force vector was then used both directly in the Simulink environment and also as an input to the neural network so to obtain the corresponding control parameter vectors to be used in the Simulink simulation. A comparison was drawn between either using the generated control parameter vectors as control inputs in the model or using the best constant control parameters for the sea state under test found during the previous optimization using genetic algorithms.

The generated force vector corresponded to a 20-minute representation of the sea state, which is the same time span used by each generation of the genetic algorithm to evaluate a population.

The analysis of the results will mainly focus on the comparison of the mean power output between the two methods, while also taking into account features like maximum and minimum power peaks.

Sea state number	Te & Hs values [s]; [m]	Mean power constant C & K [W]	Mean power with N.N control [W]	Max power constant C & K [W]	Max power with N.N control [W]	Min power constant C & K [W]	Min power with N.N control [W]
1	4,17; 0,1	7,870* 10 <sup>-1</sup>	1,523* 10 <sup>-1</sup>	2,013* 10 <sup>-3</sup>	1,131* 10 <sup>-3</sup>	-1,354* 10 <sup>-3</sup>	-9,868* 10 <sup>-2</sup>
2	6,13; 0,1	2,185* 10 <sup>-2</sup>	5,647* 10 <sup>-1</sup>	1,901* 10 <sup>-4</sup>	5,979* 10 <sup>-3</sup>	-1,596* 10 <sup>-4</sup>	-5,225* 10 <sup>-3</sup>
3	8,08; 0,1	4,097* 10 <sup>-2</sup>	1,435* 10 <sup>-2</sup>	6,916* 10 <sup>-4</sup>	2,281* 10 <sup>-4</sup>	-6,768* 10 <sup>-4</sup>	-2,119* 10 <sup>-4</sup>
4	10,04; 0,1	6,384* 10 <sup>-2</sup>	3,043* 10 <sup>-2</sup>	1,745* 10 <sup>-5</sup>	5,258* 10 <sup>-4</sup>	-1,571* 10 <sup>-5</sup>	-4,823* 10 <sup>-4</sup>

<b>5</b>	4,17; 1,08	9,391* 10 <sup>3</sup>	1,724* 10 <sup>3</sup>	2,400* 10 <sup>5</sup>	1,325* 10 <sup>5</sup>	-1,482* 10 <sup>5</sup>	-1,094* 10 <sup>5</sup>
<b>6</b>	6,13; 1,08	2,549* 10 <sup>4</sup>	6,583* 10 <sup>3</sup>	2,256* 10 <sup>6</sup>	6,960* 10 <sup>5</sup>	-1,895* 10 <sup>6</sup>	-6,086* 10 <sup>5</sup>
<b>7</b>	8,08; 1,08	4,753* 10 <sup>4</sup>	1,672* 10 <sup>4</sup>	9,794* 10 <sup>6</sup>	2,669* 10 <sup>6</sup>	-9,684* 10 <sup>6</sup>	-2,471* 10 <sup>6</sup>
<b>8</b>	10,04; 1,08	7,489* 10 <sup>4</sup>	3,543* 10 <sup>4</sup>	2,324* 10 <sup>7</sup>	6,157* 10 <sup>6</sup>	-2,117* 10 <sup>7</sup>	-5,638* 10 <sup>6</sup>
<b>9</b>	4,17; 2,06	3,542* 10 <sup>4</sup>	5,849* 10 <sup>3</sup>	1,100* 10 <sup>6</sup>	4,443* 10 <sup>5</sup>	-6,295* 10 <sup>5</sup>	-3,523* 10 <sup>5</sup>
<b>10</b>	6,13; 2,06	9,295* 10 <sup>4</sup>	2,396* 10 <sup>4</sup>	8,333* 10 <sup>6</sup>	2,531* 10 <sup>6</sup>	-6,981* 10 <sup>6</sup>	-2,213* 10 <sup>6</sup>
<b>11</b>	8,08; 2,06	1,738* 10 <sup>5</sup>	6,070* 10 <sup>4</sup>	3,047* 10 <sup>7</sup>	9,772* 10 <sup>6</sup>	-3,056* 10 <sup>7</sup>	-8,997* 10 <sup>6</sup>
<b>12</b>	6,13; 3,04	2,215* 10 <sup>5</sup>	4,753* 10 <sup>4</sup>	2,431* 10 <sup>7</sup>	7,590* 10 <sup>6</sup>	-2,167* 10 <sup>7</sup>	-6,617* 10 <sup>6</sup>
<b>13</b>	8,08; 3,04	3,779* 10 <sup>5</sup>	1,318* 10 <sup>5</sup>	6,979* 10 <sup>7</sup>	2,145* 10 <sup>7</sup>	-6,755* 10 <sup>7</sup>	-1,963* 10 <sup>7</sup>
<b>14</b>	8,08; 4,02	6,993* 10 <sup>5</sup>	2,258* 10 <sup>5</sup>	1,914* 10 <sup>8</sup>	4,638* 10 <sup>7</sup>	-1,833* 10 <sup>8</sup>	-4,323* 10 <sup>7</sup>

<b>Sea state number</b>	<b>Percentage mean power difference</b>	<b>Percentage max positive power difference</b>	<b>Percentage min reactive power difference</b>
<b>1</b>	-80,6480	-43,8152	-27,1196
<b>2</b>	-74,1556	-68,5481	-67,2619
<b>3</b>	-64,9744	-67,0185	-68,6909
<b>4</b>	-52,3340	-69,8682	-69,2998
<b>5</b>	-81,6420	-44,7917	-26,1808
<b>6</b>	-74,1742	-69,1489	-67,8839
<b>7</b>	-64,8222	-72,7486	-74,4837
<b>8</b>	-52,6906	-73,5069	-73,3680
<b>9</b>	-83,4867	-59,6091	-44,0349
<b>10</b>	-74,2227	-69,6268	-68,2997
<b>11</b>	-65,0748	-67,9291	-70,5596
<b>12</b>	-78,5418	-68,7783	-69,4647
<b>13</b>	-65,1230	-69,2649	-70,9400
<b>14</b>	-67,7106	-75,7680	-76,4157



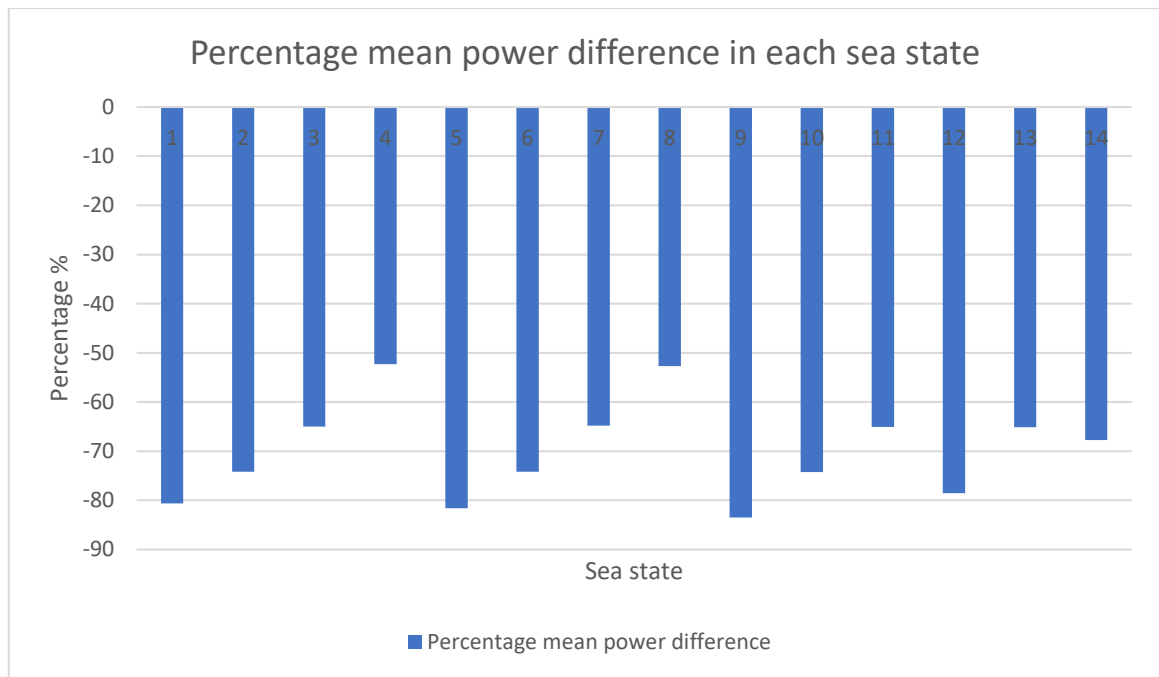


Figure 6. 15 - Percentage mean power difference between using constant control parameters or LSTM network parameters

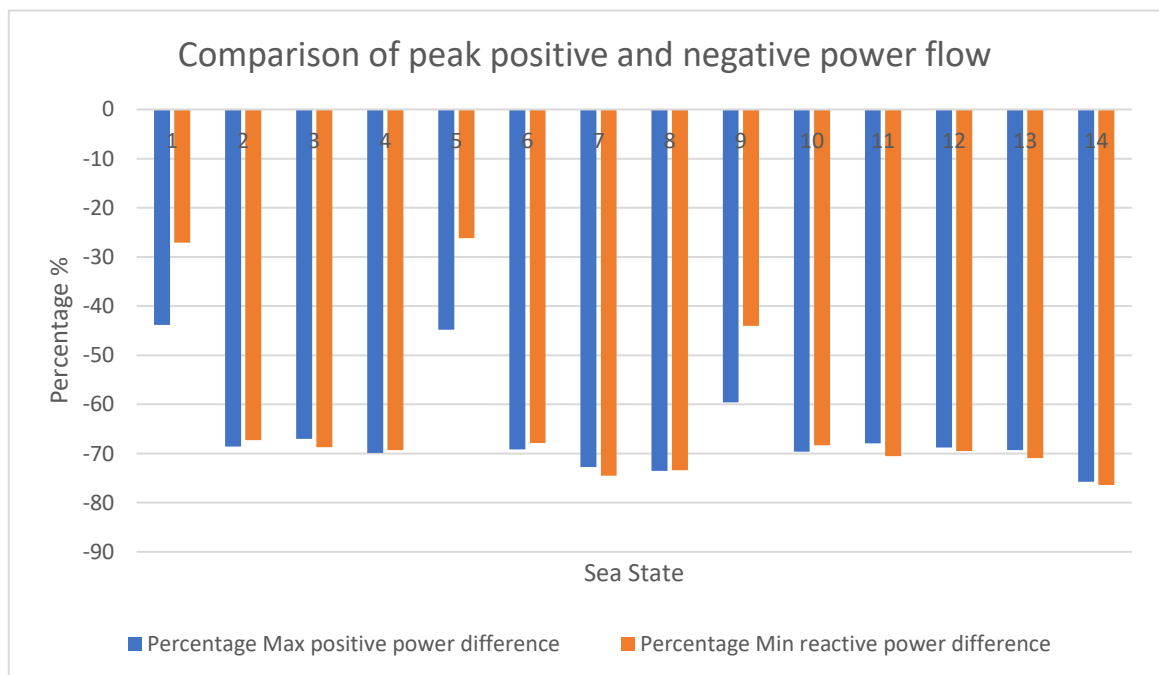


Figure 6. 16 - Max and min peak power difference when using constant control parameters or neural network parameters

From the above graphs it is clear that the performance obtained with the LSTM configuration is by far the worst between the two neural network configurations.

Analyzing the mean power difference graph, it is possible to pick out a noticeable efficiency trend as the wave energy period changes. As the period increases, the relative loss decreases, signifying that this current configuration of the neural network has difficulty in producing a control signal for wave forces with higher energetic frequencies.

To better understand this phenomenon, the control signals produced by the neural network and the corresponding wave heave force which was used as an input to produce the control signal will be plotted together to better understand if there may be any relationship between the input force frequency and the control signal frequency which ultimately led to a drop in mean absorbed power as the force frequency increased.

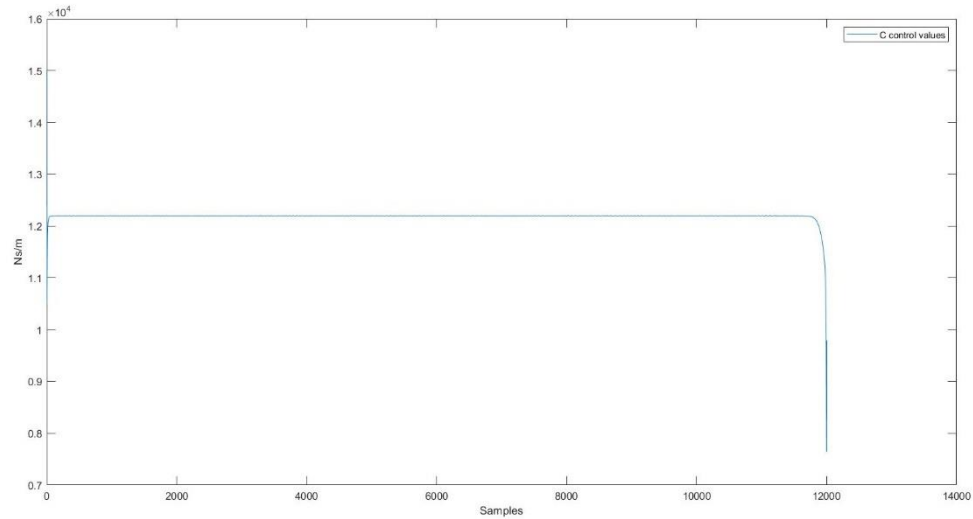


Figure 6. 17 - Control signal of damping control parameter 'C' generated for sea state number 1

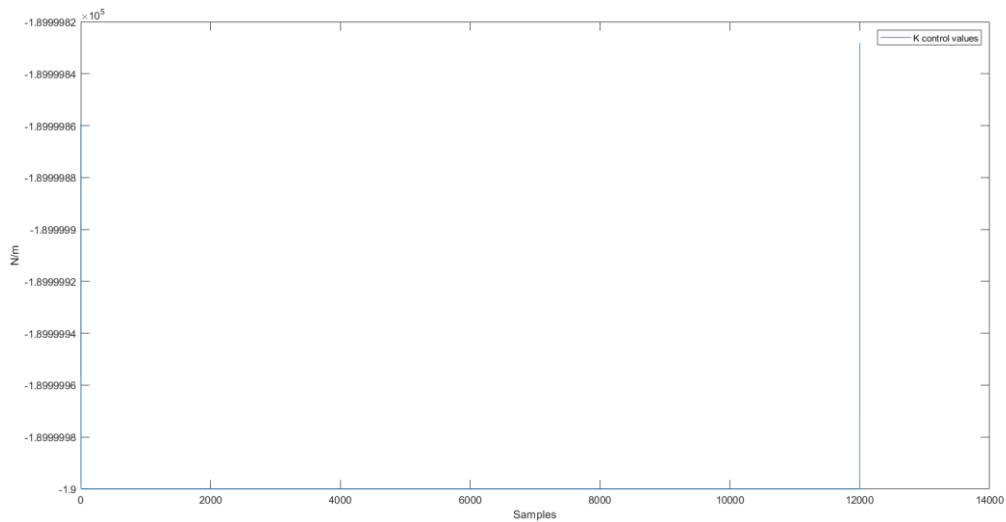


Figure 6. 18 - Control signal of stiffness control parameter 'K' generated for sea state number 1

For both the damping and the stiffness control signals, the LSTM neural network produces a control signal which has a large transient spike both at the beginning and at the end of the run, which spoils the graphical analysis of the plotted control signals. The next two plots will show a close up of the two control signals once the transient sections have been removed.

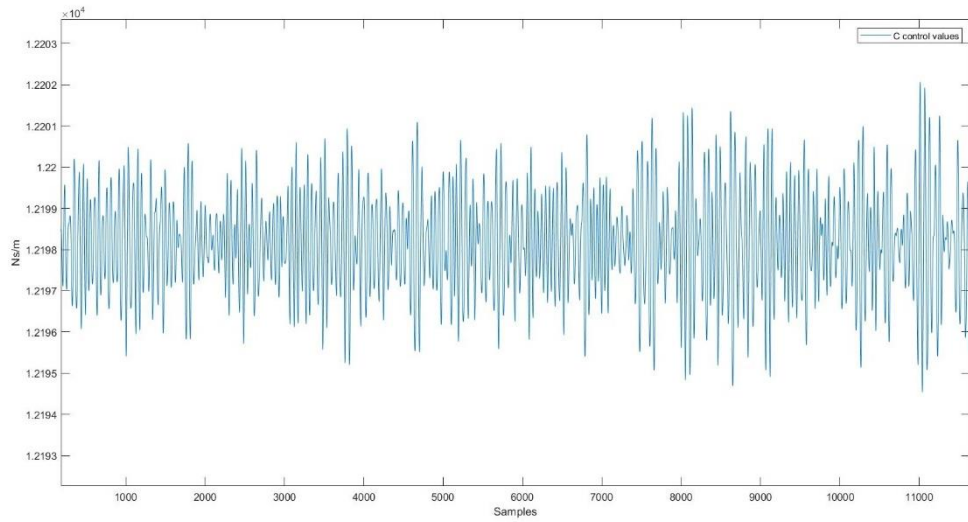


Figure 6. 19 – Close-up of the control signal of the damping control parameter 'C' generated for sea state number 1

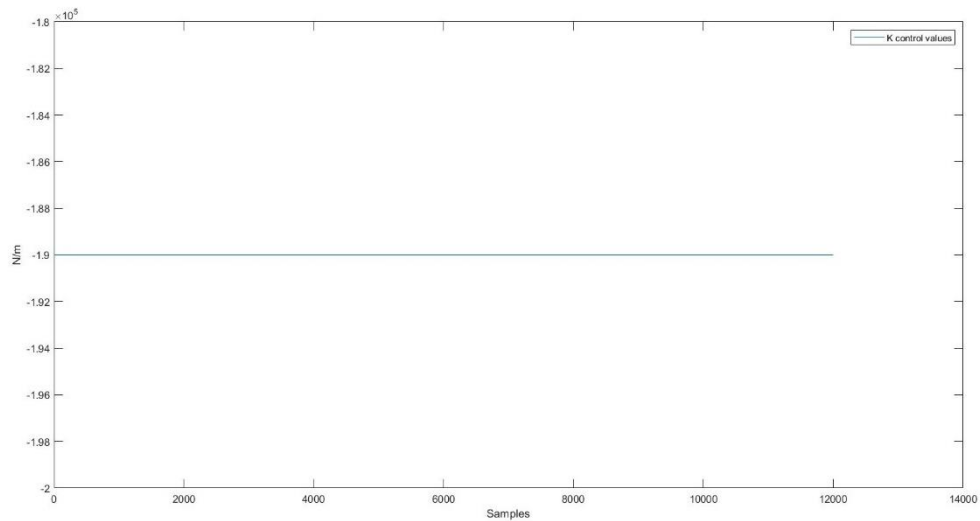
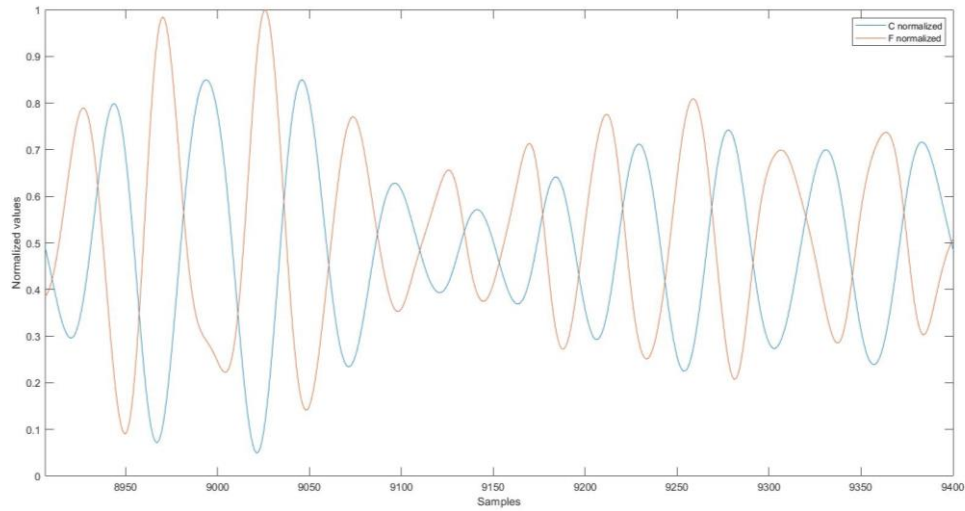


Figure 6. 20 – Close-up of the control signal of the stiffness control parameter 'K' generated for sea state number 1

The close-up graphs reveal how the LSTM network produced a constant control signal for the stiffness and a time varying signal for the damping values. This is quite in contrast to what was produced by the feed forward neural network which produced time varying signals for both control variables. Additionally, the produced signals from the LSTM do not seem to be anywhere near the corresponding optimal fixed control signals found by the genetic algorithm optimization.

To get a better understanding of how the control signal is produced and how it matches the force input, a closeup plot of the control and the heave force is presented next for both the damping and stiffness control values.

As for the feed forward neural network, the following plots will show the normalized values of the heave force and control signals. To better appreciate the variations on the control signals, the transient sections were truncated.



*Figure 6. 21 - Normalized damping and heave force signal for sea state number 1*

The above image suggests that, even for sea states with a small energetic period such as sea state number 1, for which the energy losses were much worse compared to sea states with a larger energy period, the LSTM network is able to produce a control signal for the damping parameter which is completely able to follow the oscillating input heave force, while also varying the magnitude of such control signal according to the magnitude of the input force.

This leads to the conclusion that it is not a problem related to the ability to follow the input heave force, but rather most likely linked to the magnitude itself of the control signals for both the damping and stiffness parameters.

A quick look at the optimal static control parameters obtained for sea state 1 by the genetic algorithm optimization shows how far both the constant stiffness and the mean of the varying damping produced by the LSTM network are for the parameters found by the genetic algorithm.

This is most likely the reason for the poor performance since, although a variation on a wave-by-wave basis was expected for the control parameter vectors produced by the neural network, the general working area should probably be in the same region as the static parameters obtained from the optimization process.

Further proof of this is the trend of the power loss in figure 6.15 where it's clear that performance improves as the sea state energy period increases. This is not because of some intrinsic problem with the frequency of the generated control signal for the damping control parameter, but it's simply due to how far the magnitude of the generated stiffness control signal is from the different static optimal stiffness signals for each sea state.

The next image shows all the different control signals for all the 14 sea states plotted on a single graph for the damping and stiffness control parameters with the transients removed.

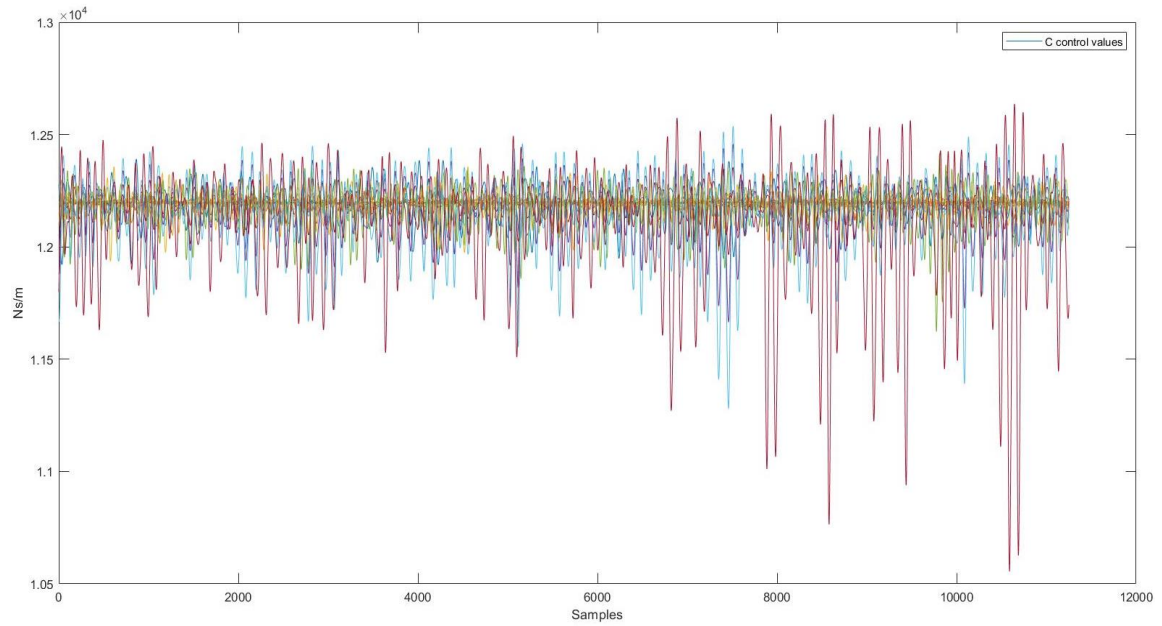


Figure 6. 22 – Damping control signals generated by LSTM neural network for all 14 sea states

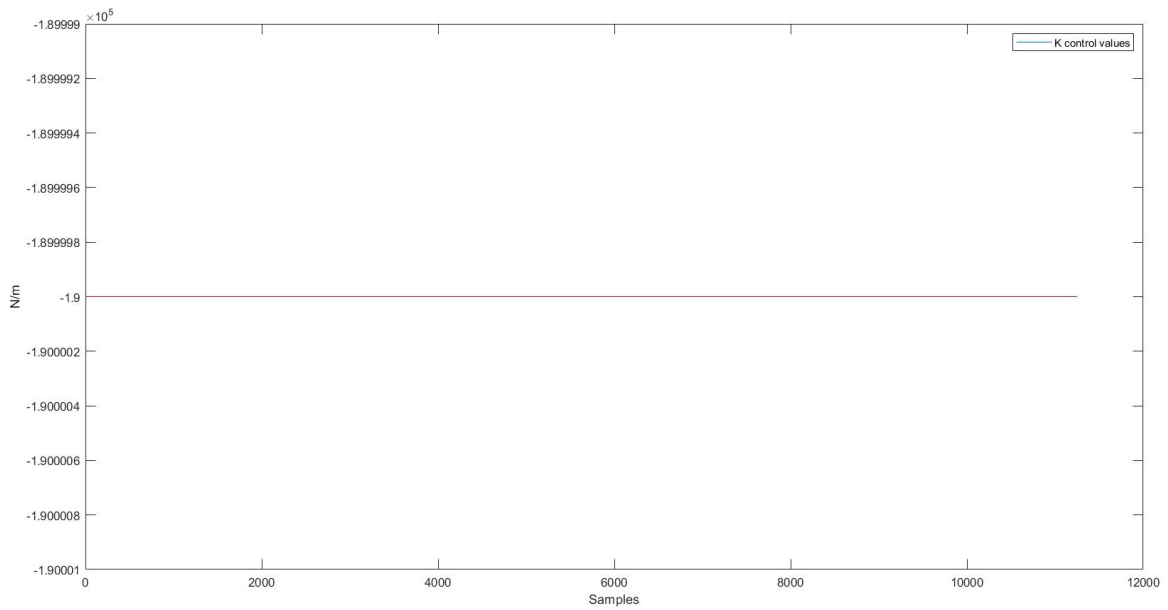


Figure 6. 23 – Stiffness control signals generated by LSTM neural network for all 14 sea states

From the two graphs it's clear how the LSTM network somehow had bound all the control signals to be centered around two very specific locations for both the varying damping control signal and the constant stiffness signal; was now clear that this was the main driver of the poor performance. In particular, the greatest contribution to this poor performance was given by the constant stiffness signal, which for some sea states was very far from the optimal constant stiffness found by the genetic algorithms, while for other sea states (see table at page 78) it was much closer.

It was now clear how the main cause of the bad performance and of the power loss trend which varied as the sea state energy period varied was the produced stiffness control signal and that although also

the damping signal seemed to be bound to a region in which it most likely should not be, it was not the main driver of the bad performance.

This theory was put to the test by shifting the damping control signal so that it's mean would lie around the optimal value produced by the genetic algorithm and by amplifying the control signal so that the oscillations would be more prominent. A simulation was then run with the modified damping control alongside with the constant optimal stiffness produced by the genetic algorithm. This mix of control signals was done to ensure that the comparison between the constant control signals and the time varying damping control would be a fair comparison without taking in to account the value of the stiffness produced by the neural network. The results showed that with a constant stiffness but a time varying damping factor the mean absorbed power actually increased with respect to the control using constant control variables in some sea states.

Sea state number	Te & Hs values [s]; [m]	Mean power constant C & K [W]	Mean power with N.N control [W]	Max power constant C & K [W]	Max power with N.N control [W]	Min power constant C & K [W]	Min power with N.N control [W]
1	4,17; 0,1	7,987* 10 <sup>1</sup>	8,031* 10 <sup>1</sup>	2,013* 10 <sup>3</sup>	3,015* 10 <sup>3</sup>	-1,354* 10 <sup>3</sup>	-2,291* 10 <sup>3</sup>
2	6,13; 0,1	2,048* 10 <sup>2</sup>	2,048* 10 <sup>2</sup>	1,901* 10 <sup>4</sup>	1,713* 10 <sup>4</sup>	-1,596* 10 <sup>4</sup>	-1,413* 10 <sup>4</sup>
3	8,08; 0,1	3,982* 10 <sup>2</sup>	3,921* 10 <sup>2</sup>	6,916* 10 <sup>4</sup>	7,837* 10 <sup>4</sup>	-6,768* 10 <sup>4</sup>	-6,545* 10 <sup>4</sup>
4	10,04; 0,1	5,783* 10 <sup>2</sup>	5,809* 10 <sup>2</sup>	1,376* 10 <sup>5</sup>	1,675* 10 <sup>5</sup>	-1,366* 10 <sup>5</sup>	-1,730* 10 <sup>5</sup>
5	4,17; 1,08	9,213* 10 <sup>3</sup>	9,234* 10 <sup>3</sup>	2,400* 10 <sup>5</sup>	3,015* 10 <sup>5</sup>	-1,482* 10 <sup>5</sup>	-1,885* 10 <sup>5</sup>
6	6,13; 1,08	2,390* 10 <sup>4</sup>	2,393* 10 <sup>4</sup>	2,256* 10 <sup>6</sup>	2,096* 10 <sup>6</sup>	-1,900* 10 <sup>6</sup>	-1,765* 10 <sup>6</sup>
7	8,08; 1,08	4,514* 10 <sup>4</sup>	4,522* 10 <sup>4</sup>	9,794* 10 <sup>6</sup>	1,062* 10 <sup>7</sup>	-9,684* 10 <sup>6</sup>	-9,512* 10 <sup>6</sup>
8	10,04; 1,08	6,799* 10 <sup>4</sup>	6,802* 10 <sup>4</sup>	1,893* 10 <sup>7</sup>	1,915* 10 <sup>7</sup>	-1,884* 10 <sup>7</sup>	-1,908* 10 <sup>7</sup>
9	4,17; 2,06	3,642* 10 <sup>4</sup>	3,644* 10 <sup>4</sup>	1,086* 10 <sup>6</sup>	1,052* 10 <sup>6</sup>	-6,173* 10 <sup>5</sup>	-6,68* 10 <sup>5</sup>
10	6,13; 2,06	8,723* 10 <sup>4</sup>	8,747* 10 <sup>4</sup>	8,333* 10 <sup>6</sup>	8,270* 10 <sup>6</sup>	-6,981* 10 <sup>6</sup>	-6,880* 10 <sup>6</sup>
11	8,08; 2,06	1,639* 10 <sup>5</sup>	1,669* 10 <sup>5</sup>	3,047* 10 <sup>7</sup>	3,765* 10 <sup>7</sup>	-3,056* 10 <sup>7</sup>	-3,130* 10 <sup>7</sup>
12	6,13; 3,04	2,282* 10 <sup>5</sup>	2,287* 10 <sup>5</sup>	2,431* 10 <sup>7</sup>	2,856* 10 <sup>7</sup>	-2,167* 10 <sup>7</sup>	-2,441* 10 <sup>7</sup>
13	8,08; 3,04	3,599* 10 <sup>5</sup>	3,639* 10 <sup>5</sup>	6,979* 10 <sup>7</sup>	1,056* 10 <sup>8</sup>	-6,755* 10 <sup>7</sup>	-9,140* 10 <sup>7</sup>
14	8,08; 4,02	6,862* 10 <sup>5</sup>	6,879* 10 <sup>5</sup>	1,914* 10 <sup>8</sup>	2,074* 10 <sup>8</sup>	-1,833* 10 <sup>8</sup>	-2,003* 10 <sup>8</sup>

Sea state number	Percentage mean power difference	Percentage max positive power difference	Percentage min reactive power difference
<b>1</b>	0,5509	49,7765	69,2024
<b>2</b>	0	-9,8895	-11,4662
<b>3</b>	-1,5319	13,3169	-3,2949
<b>4</b>	0,4496	21,7297	26,6471
<b>5</b>	0,2279	25,6250	27,1930
<b>6</b>	0,1255	-7,0922	-7,1053
<b>7</b>	0,1772	8,4337	-1,7761
<b>8</b>	0,0441	1,1622	1,2739
<b>9</b>	0,0549	-3,1308	8,2132
<b>10</b>	0,2751	-0,7560	-1,4468
<b>11</b>	1,8304	23,5642	2,4215
<b>12</b>	0,2191	17,4825	12,6442
<b>13</b>	1,1114	51,3111	35,3072
<b>14</b>	0,2477	8,3595	9,2744

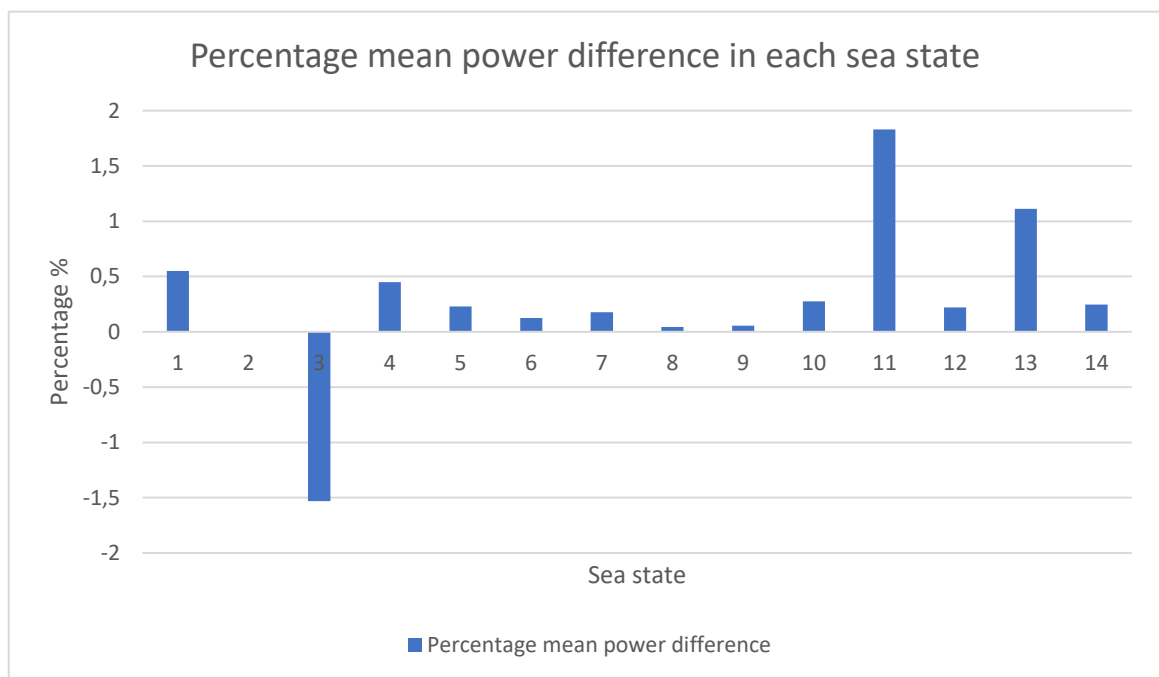


Figure 6. 24 - Percentage power difference when using shifted and rescaled LSTM network damping and constant stiffness

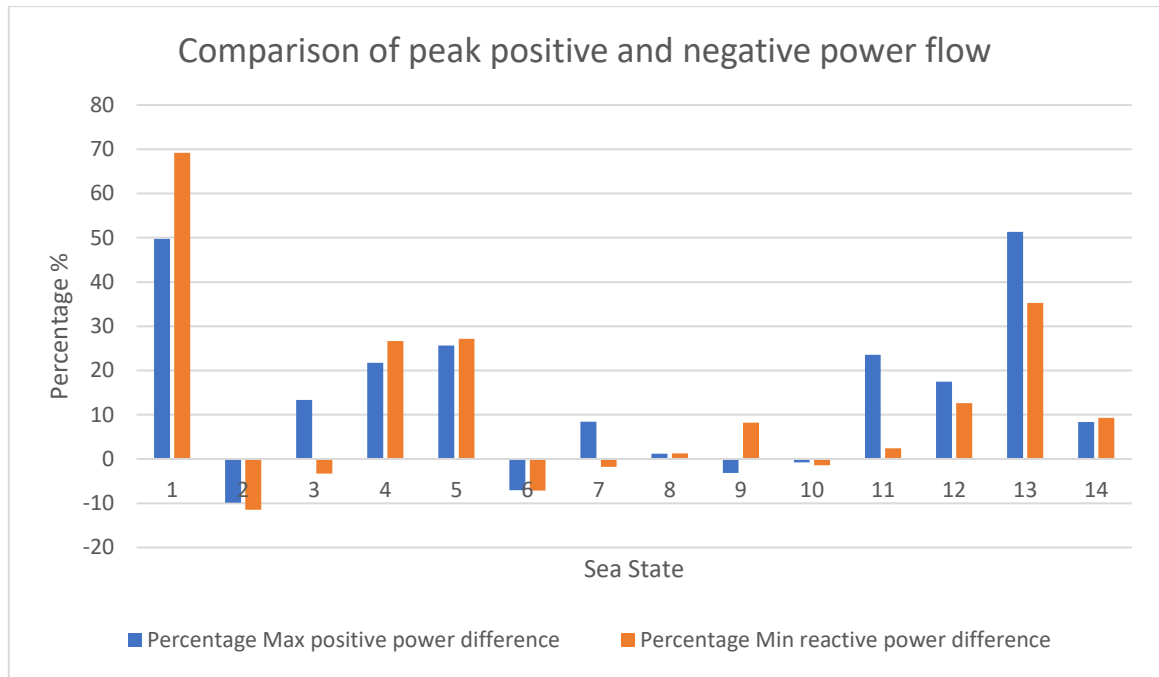


Figure 6. 25 - Max and min power difference when using shifted and rescaled LSTM network damping and constant stiffness

The above graphs suggest that the problems of the control signals produced by the neural network were mainly two:

- Firstly, both the damping and stiffness signal were centered around a single value for all sea states. This is particularly detrimental especially considering that, since the produced stiffness signal was constant, this meant that, whatever the sea state, the stiffness would not change. This problem was fixed in the last test by shifting the signals produced by the LSTM network to be centered around the optimal vales found by the genetic algorithm optimization procedure.
- Secondly, the variation in the damping control signal actually seemed to produce some positive effects in terms of produced mean power in most of the sea state scenarios tested. Thus, it might have also been positive to have a varying stiffness signal instead of a constant one.

Although the above results were obtained by shifting the damping control signal around a working point which was known a priori to be efficient, it is still a positive result to see that in all sea states but one, the varying damping signal produced by the neural network actually managed to achieve a higher mean power reading than when using a constant damping signal. So, despite having to manually manipulate the signal position, this result still shows that a neural network does have the potential to create a time varying control signal able to increase the power absorption of a point absorber.



## 7 – Conclusions and future work

In this thesis, a collaborative learning strategy for model-free control of an array of wave energy converters has been analyzed and tested in a simulation environment.

The aim of this work was to test a model-free learning strategy which would allow an array of heaving point absorbers to collaborate to reach the common goal of optimizing the control strategy variables and to use the acquired data to further learn how to change the control parameters in a continuous manner in order to adapt the control strategy on a wave-by-wave basis.

The simulations involved 14 separate sea states, each characterized by its own couple of significant wave height ( $H_s$ ) and wave energy period ( $T_e$ ). Each sea state was considered to last 20 minutes in order to get a robust statistical evaluation of the point absorber performance.

The control strategy chosen was of the reactive type which implied the use of two independent control parameters to be tuned, namely the damping factor ( $C$ ) and the stiffness ( $K$ ).

Genetic algorithms were initially used as a metaheuristic collaborative learning strategy to optimize the two control parameters for each sea state. Each point absorber in the array represented a single individual in the population of the genetic algorithm carrying as genes the control parameters used for the current generation and as fitness, the mean absorbed power over the simulation time window of 20 minutes.

Before deploying the genetic algorithm, a tuning procedure was used which would promote algorithm structures whose internal hyperparameters resulted in algorithms with enough exploratory behavior to not trap in local minima but enough exploitative behavior so to not take too much time to converge.

A ninety-minute simulation comprised of a mixture of all the sea states taking in to account their specific occurrence was created and used as a testing ground for the genetic algorithm evolution.

A memory management system was then used to create individual meta populations of a single genetic algorithm structure where each population was linked to a single sea state and could evolve independently to reach optimal control parameters for the given sea state.

The results showed how a population of 16 individuals was able to converge to optimal static control parameters for a given sea state in just over 12 generations, which in turn meant that the array would only need about 4 hours in a given sea state to find the optimal damping and stiffness parameters which would maximize the mean power output of the device.

With the genetic algorithm structure complete, the next step was to find a means to achieve a model-free continuous control over the control variables of the reactive control law so to achieve a control signal which could vary on a wave-by-wave basis so to continuously adapt to the current scenario.

The adopted strategy tested both feed-forward neural networks and LSTM neural networks as means to learn how to map the input heave wave force into a meaningful continuous control signal for both the damping factor and the stiffness parameters of the reactive control law.

The two network structures showed remarkably different behavior.

The feed forward neural network was fed, at each time instant, with multiple inputs which represented the current and past 600 heave force values read and as output, the vectors corresponding to the two control parameters had to be produced.

The network showed that it was able to produce signals with a similar frequency to the input heave force, signifying that it was actually able to correctly follow the oscillating wave input force.

Besides this positive remark, in all but one wave scenario the mean produced power was lower than what was obtained with the constant control parameters and the peak power flow values were in some cases lower and in other higher.

Summarizing, the feedforward neural network showed potential in its ability to map the force signal in to the two control signals, but the results showed that the produced signals didn't actually cause any appreciable increase in the produced mean power.

A second but similar test was carried out using a long-short term memory (LSTM) neural network. This kind of network has the ability to create feedback loops within itself and to store past information in order to make more significant predictions about the current input. For these reasons there was no need to feed the network with a matrix of inputs, instead a simple single force vector of the time series heave force was used.

Unfortunately, the network by its own wasn't able to improve on the feed forward neural network, but actually produced worst results in terms of the mean absorbed power. This was mainly due to the produced control signals all being bound in a region which was sub optimal in terms of produced power. The reason behind this most likely resides in the chosen network structure.

Additionally, to having all the signals bound to a fixed common mean value, an unexpected behavior from the LSTM network was the constant stiffness control signals it produced. This was particularly surprising since, at least to the authors knowledge, no bias was given between one output and the other and both outputs were treated in the same manner in the final processing phases of the neural network. These two major malfunctions ultimately caused the LSTM network to perform poorly with respect to the feed forward network.

To understand if the network could have potentially still produced a control signal which could have increased the mean absorbed power of the device, the damping control signals were manually shifted so that their mean would match the optimal control values produced by the genetic algorithm

optimization. The results showed that in this scenario the mean absorbed power actually increased in all but one sea state signifying that the varying damping signal produced by the LSTM network could have had the potential to improve the performance over the constant control signals. Unfortunately, since the stiffness signal produced by the network was constant, the same could not be said for the stiffness parameter, although one might speculate that also a varying stiffness control signal might also increase the systems performance.

As a final consideration, the first half of the proposed strategy, involving the online optimization of the control parameters through a metaheuristic approach employing genetic algorithms can be considered a success. The designed genetic algorithm proved to be a valid tool to optimize the parameters of a reactive control law over multiple sea state scenarios in a repeatable, fast and reliable manner while never using any model of the system during the optimization process.

On the other hand, the second half of this work, which involved the use of neural networks to further learn interdependencies between the input heave force and the corresponding control parameters to try and achieve a continuous control over such parameters did not produce exceptional results.

The main problem with the neural network approach will obviously lie either in the training set used or in the structure of the neural nets themselves. Both of these fields may be subject of further study to try and accomplish better results.

Future work which may spawn from this preliminary research might include:

- Further work on the genetic algorithms to make the optimization process even faster and more reliable by trying to implement adaptive control over the genetic algorithm structure and self-adaptation [72], [73], [74].
- Testing and developing neural network structures to try and achieve better performance and to try and uncover the possible causes of the shapes of the signals produced by the LSTM network.
- Additional work may also try and achieve continuous data driven control through other machine learning techniques besides from neural networks.
- Finally, the actual dataset used to train the neural networks might be revised and modified to try and give the neural networks, or any other tool used, more information about how the input signals are linked to the desired output control signals, and to try and gather more meaningful data about what the networks are actually trying to achieve.



## Appendix A: Evolutionary algorithms

Evolutionary computing is a branch of computer science which, as the name suggests, takes inspiration from evolutionary processes occurring in nature. More precisely we can think of evolutionary computing as a parallel to a scenario where in nature, a population of individuals in a given environment is competing for survival and reproduction.

The competing individuals can be somehow ranked based on their fitness. In evolutionary computing such fitness relates to how well an individual is performing at achieving the desired goal. Just as in nature, the probability of survival and reproduction of an individual depends on how fit such individual is.

When talking about evolutionary computing, we still use the term individual and fitness, but it must be noted what such terms mean. For *individual* we intend a candidate solution to our optimization problem, this solution may be a single parameter, or a set of parameters based on the dimension of the optimization problem. For *fitness* of an individual, we mean the quality (how well it solves the problem at hand) of a solution that uses the given parameters that the individual carries.

Under the main category of Evolutionary Computing, over the years many branches stemming from the same basic concepts have emerged.

In the 1960s three different families of Evolutionary Computing emerged:

- In the USA, Fogel, Owens and Walsh introduced **Evolutionary Programming (EP)**.
- In the USA, Holland introduced his version called **Genetic Algorithm (GA)**.
- In Germany, Reichenberg and Schwefel introduced **Evolution Strategies (ES)**.

Each of these methods use the same basic principles of evolution but differ in their implementation.

The basic underlying idea behind all Evolutionary Computing algorithms is the same: given a population of individuals and a measure to evaluate the fitness of such individuals, as generations pass and natural selection takes place, thanks to the principle of survival of the fittest, the average fitness of the population will grow as populations elapse.

This concept can easily be translated into function maximization (using a fitness function) or minimization (using a cost function). We can imagine a simple example with a 3D surface and only 2 optimization variables, while the vertical axis corresponds to the fitness of the candidate solutions (individuals). An initial population of candidate solutions is initialized randomly, as populations elapse and reproduction and mutation between solutions occurs, the average fitness of the population will increase thanks to reproductive and survival selection strategies which favour the fittest individuals. Eventually the population should move to a position in the landscape which is correlated to high fitness (a peak in the fitness function), leading to a solution very close to the function's maximum. Thus, it's easy to see that in an Evolutionary Algorithm scenario, the fitness function which gives a measure of how fit each individual is, must correspond to the function we want to maximise (or minimise if using a cost function).

In the next chapter, a detailed view of how an Evolutionary Algorithm operates is given referencing the main operators that are used. Later, focus will be shifted on specific operators and mechanisms for Genetic Algorithms, which as stated earlier, are a specific family of Evolutionary Algorithms.

## A.1 How does and Evolutionary Algorithm work?

As stated in the previous paragraphs, an Evolutionary Algorithm takes inspiration from evolutionary theory by using an analogy of survival of the fittest to a population of candidate solutions, in order to possibly reach the optimum solution to our underlying problem.

At the beginning of an algorithm run, a set of candidate solutions is randomly produced in the space of optimization variables specific to the problem, and within the predefined bounds of such variables. To give an example, if the function we were trying to minimize was the sphere function, each individual would carry in its genome the X and Y coordinates of its position in the X-Y plane. Depending on the problem dimensionality, the number of optimization variables will vary.

It must be noted that the number of candidates solutions randomly generated is chosen a priori and is referred to as **population size**. Generally, the population size will remain constant throughout the run, while the individuals making up the population will change.

Once the initial population has been initialised, each member of the population is assigned a fitness value which depends on the function value we are trying to minimise/maximise if the optimization parameters of such individual were plugged in to such function. Usually, the fitness is chosen to be exactly equal to the function value.

At this point the initial individuals will be ranked by fitness and the first generation is now complete. All further steps are needed to give life to the next generation of individuals which will either replace or will in some cases compete against the current generation for survival.

Firstly, from the pool of current individuals, parents are picked for reproduction, this is known as **parent selection**. Usually, each couple of parents gives life to 2 offspring so that the number of offspring can easily match the population size. This is not always the case since in some implementations the total number of offspring is larger than the population size. Competition between offspring or between offspring and current population is then used to reduce the number of survivors to the selected population size. In other cases (steady state algorithms) at each generation only one offspring is produced. Whatever the total number of chosen offspring may be, once such number has been reached, the offspring undergo **mutation**. Mutation allows the genotype of an individual to mutate randomly in order to create a new mutated version of the initial gene. In other words, the optimization parameters which an individual is carrying are mutated, forming a new individual which replaces the non-mutated child. Mutation is not always applied to all the offspring, usually the offspring undergo mutation with a certain probability often referred to as **mutation probability** or **mutation rate**. Once the offspring have been created, their fitness is evaluated.

Now that a new set of offspring has been created and mutated, competition for survival must occur.

Different kind of **survival selection** methods are available but generally, the offspring can either:

- Directly form the next generation (if the number of offspring is equal to the population size).
- Compete against each other only (if the number of offspring is larger than the population size).
- Compete against each other and the previous generation.
- No fitness-based competition occurs, the age of the solutions is used to determine who continues to the next generation (youngest) and who is left out (oldest).

Competition can be either completely deterministic, only choosing the best candidates at each cycle, or it can have some stochasticity introduced, thus giving a chance to suboptimal solutions to also pass in the next generation.

The process that has just been described then continues until either a predefined **termination condition** is met or when a solution of sufficient quality is found.

A pseudo code for an EA might look something like:

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

Figure A. 1 - Pseudo code for a generic EA [62].

A simple high-level explanation of the main forces acting on an Evolutionary Algorithm can help explain how each operation within the above pseudocode can help the population to increase its average fitness and finally reach an adequate solution

The main forces acting on an Evolutionary Algorithm are Variation and Selection:

- Recombination and Mutation are both variation operators. Their purpose is to create new genetic material thus allowing the exploration of new solutions and thus of the optimization space.
- Selection is instead the driving force which utilises the fitness information of each individual to drive the increase of the average population fitness, thus allowing the population to exploit the fitness information that is known.

A balance between exploration and exploitation is usually needed to properly reach a good solution. Typically, at the beginning of a run, exploration must be predominant in order to allow the collection of data about the optimization landscape and to explore new solutions. As time passes, exploitation usually takes over in order to use the gathered information to reach an optimal solution.

The balance and trade-off between exploration and exploitation obviously depends on the type of landscape we are facing (unimodal/multimodal, simple/complex etc) and on the specific implementation details of the EA.

In general, if exploration is kept too high, the algorithm will act as a random search algorithm, losing all the good traits of an EA, if instead exploitation is too high, the algorithm might make ill-suited decisions based on too little information and may end up stuck in a local minimum.

## A.2 Evolutionary Algorithms: Why?

When looking for new ideas on how to solve problems, scientists and engineers have often resorted to nature, looking for inspiration from natural processes or living beings.

Regarding problem optimization and problem solving, two obvious candidates from which we might take inspiration from are:

- The human brain.
- The naturally occurring evolutionary processes.

The first candidate led to the evolution of the field of neurocomputing, while the second led to the field of evolutionary computing. Thus, as a first motivation for how and why evolutionary algorithms came to be we can simply say, nature inspired human curiosity. This curiosity can be then directly linked to the use of evolutionary computing not only for problem solving but used directly to better understand natural evolution processes. Evolutionary processes can be simulated using a wide range of parameters, simulating different population traits and different set of initial circumstances within a matter of a few hours or days, giving researchers insights of how evolutionary processes might have shaped passed populations or how they might shape future ones.

Obviously, such simulations have to be performed very carefully, paying attention to all implementation details in order to get a realistic performance. But even so, this will not always ensure that the obtained results can be directly linked to real world processes.

A second motivation for the use of evolutionary algorithms is the fast pace at which new problems, requiring new solutions, are emerging. This fast growth pace although is accompanied by a smaller and smaller time window to solve these problems in, thus preventing ad hoc solutions to be developed, which would obviously take time and resources to implement. Thus, the trend is to look for a robust general solver which can perform well under a variety of problems with only minor adjustments. This is exactly what evolutionary algorithms are capable of, making them ideal



candidates for a wide variety of problems that can then be solved with minor adjustments in a relatively short time.

### A.3 Components of Evolutionary Algorithms

The main components that make up the most important parameters to be chosen when constructing an Evolutionary Algorithm are:

- Representation
- Population
- Initialization
- Parent selection
- Recombination (or Crossover)
- Mutation
- Survival selection
- Termination condition
- Performance measure

#### A.3.1 Representation

The first step to take when designing an Evolutionary Algorithm is to decide which representation type will be used to define the population members. Representation consists in finding a way to map the solutions of our real-world problem into the population of our Evolutionary Algorithm, such that this population can then be manipulated by the algorithm to evolve in time.

Objects forming the solution space of the original “real world” problem are referred to as *phenotypes* while the encoded solutions that are then manipulated by the Evolutionary Algorithm are called *genotypes*.

In this stage it is important to select a proper encoding in order to assure firstly that all possible solutions to our problem may be explored without limitations (unless specified) and also that the final solution of our Evolutionary Algorithm (the final genotype) then can be meaningfully translated back into a solution of our optimization problem (a phenotype).

The type of representation used mainly depends on the problem at hand, even though in some cases, different types of representations may work on the same problem.

#### Binary representation

Binary representation is one of the earliest types of representation used, especially when dealing with Genetic Algorithms (GAs).

Binary representation simply consists in encoding a solution in a bitstring of predetermined length.

This type of encoding is quite a natural type of encoding when dealing with Boolean decision problems where the complete solution is a simple set of a certain number of either yes or no. An

example of these kind of problems is the Knapsack problem, which is a sort of generalization of many industrial problems. Imagine having a set of  $n$  items each with its own *value*  $v_i$  and *cost*  $c_i$ . The problem consists in selecting a subset of such items that will maximise the sum of the single values, while keeping the sum of the costs below a certain a priori defined threshold  $C_{\max}$ . It is thus natural to use binary encoding for such kind of problems where each candidate carries a binary string where a 1 means to keep the corresponding item while a 0 means to discard such item.

Binary encoding was also heavily used in early genetic algorithms whose phenotype solution space was made of real numbers. For example, an individual might carry a binary string of length 32 where each 8 bits represented an encoded real valued variable, thus in this specific example, allowing up to 256 possible values to each variable in a range defined a priori.

One of the main problems in representing real numbers using a binary encoding is that not all bits have the same importance, thus when mutation occurs (random change of individual's value) the magnitude of such mutation strongly depends on which bit is mutated. This can be reformulated in the fact that the Hamming distance between consecutive integers, mapped as binary strings, is often not equal to one.

Ideally, during mutation, the probability of changing a 5 in to a 6 or in to a 4 should be the same, however changing a 0101 to 0110 requires 2 bits to be flipped while changing it in to a 0100 only requires 1 bit to be flipped. Thus, when dealing with real valued phenotypic variables, the extra work of mapping into binary is usually not worth the effort.

### Integer representation

Integer representation can be naturally used if our problem deals with the optimization of variables which take on integer values. An example might be of moving along a path on a square grid, we might encode *North, East, South, West* as  $\{0, 1, 2, 3\}$ .

As can be seen from the example above, integer representation can be used both when our variables actually represent real integers, and thus phenotype to genotype mapping is straight forward, or it can also be used to represent a list of actions or a list of attributes on which our optimization problem is based on.

### Real valued representation

When the values we need to represent come from a continuous distribution instead of a discrete one, it is obvious that neither binary nor integer representations are well suited to our needs. The most natural representation in this case is a real valued representation where variables might represent continuous physical quantities like a length, temperature, power etc.

In the early days of Evolutionary Algorithms, especially when dealing with Genetic Algorithms, problems whose phenotypic variables were real valued numbers were often encoded using a binary

representation instead of a real valued representation. This was often done simply because historically Genetic Algorithms were born using a binary representation scheme. Encoding real valued variables as binary strings obviously causes a loss in precision since a binary string can only be decoded in an integer number. This number can then be used to represent a position within a predefined range, which can then lead to a floating-point number in some cases, but still, only a limited number of floating point numbers in the predefined range can be represented in such manner, causing a loss in precision. The other problem with binary representations of real valued numbers was already presented previously when it was shown that the Hamming distance of two consecutive binary numbers is not always 1, thus leading to favourable directions of mutation.

### Permutation Representation

Permutation representation is used when the problem at hand requires the optimiser to decide an ordering in which a sequence of events should happen. Permutation encodes the events as a fixed set of integer values but differs from integer representation since it does not allow a given integer to be repeated twice in the sequence (so any given event can only occur once in the sequence).

A classic example of this type of representation may be used when dealing with a production scheduling problem where the problem may ask to decide in which order should some components be produced based on the set-up times, production times etc.

### A.3.2 Population

The population of Evolutionary Algorithms can be seen as a group of individuals whose role is to hold the representation of possible solutions. Initially, the population is usually initialized at random, thus the values of the variable that each individual carries in its genes are randomly chosen within a predefined range. The population can then evolve in time leading to an increasing fitness and finally a near optimal solution to the n dimensional fitness landscape on which it's evolving on.

The population size is a key parameter to be chosen when dealing with any Evolutionary Algorithm. The population size is usually chosen to be fixed during a single run, thus creating the competition for survival, but in some case studies, researchers have experimented with varying population size in order to obtain better performance from the given algorithm. Generally, a fixed population size is a good choice for most problems.

Usually, a larger population size is preferred for complex landscapes and multimodal problems in order to allow for better exploration. A smaller population is instead preferred for simpler landscapes, which also allows a smaller computational cost with respect to a large population.

Population *diversity* is a measure of how diverse the population is, or in other words, how many different solutions are present in the population and how different such solutions are from one another. It is usually desirable to keep population diversity high at the beginning of a run, in order to

better explore the landscape and to avoid getting trapped in local minima. As the run progresses, population diversity should drop as the population converges to a minimum, hopefully being a global minimum.

### A.3.3 Initialization

Initialization is a procedure used only once at the beginning of the run of a genetic algorithm to create the first population of individuals. All the methods used in future generations do not apply to the first generation since they are all based on evolutionary concepts, while the first generation has nothing to evolve from.

Generally, a good initial population can help the algorithm to locate the optima while a bad guess may hinder the evolution. While this topic may be intuitive, some researchers sustain that the effort put in to finding an optimal initialization strategy may be in some sense not particularly well placed since evolutionary algorithms can in general increase the average fitness of the population very quickly in the first few generations. Thus, a good initialization strategy, which would cause the initial average fitness of the first population to be higher, may only save a few elapsed generations with respect to any other initialization technique.

Although the above statement may be true, choosing the right initialization technique may help in solving particularly hard and large-scale problems with a relatively small population.

For a problem regarding a black-box optimization, no information about the optimization landscape can be used by the initialization procedure to select favourable location in which to initialize the population. In these cases, the most common initialization procedure involves a random initialization of the initial individuals. An example is the use of pseudo-random number generators.

Many other techniques and categorizations of initialization procedures exist, and an extensive survey can be found in the work from Borhan et al. [75].

### A.3.3 Parent selection

Parent selection is the procedure of selecting a number of parents for reproduction which will yield their offspring which will then form the next generation. Parent selection is usually not completely stochastic, in other words, individuals with higher fitness have a higher chance of being selected with respect to individuals with a lower fitness level. The probability distribution of parent selection over the whole population determines what is known as *selection pressure*.

A high selection pressure means that low fitness candidates have a low probability of being selected while a lower selection pressure starts to even out the field and makes parent selection tend towards a stochastic process. It is usually preferable to avoid the extremes of selection pressure since a pressure which is too high will lead to an algorithm which is too greedy and not very explorative, causing the population to get stuck in a local minimum, while a selection pressure which is too low will cause parent selection to be stochastic and will not drive the population to increase its fitness

since individuals with poor performance have the same chance of reproducing as individuals with high performance.

Selection pressure can usually be tuned based on the parent selection mechanism chosen.

The parameter through which selection pressure can be tuned is usually chosen to be constant throughout a run, but research indicates that often, a varying selection pressure may lead to better performance both in terms of quality and in terms of speed. The commonly chosen option when varying selection pressure is to increase selection pressure in time since this will allow the population to initially explore a larger solution space and not get trapped in local minima, then in time, selection pressure can be increased in order to focus on a specific location in the search space where the optimum probably lies.

Different parent selection schemes exist, and most of these can also be used for survival selection.

### Fitness proportional selection (Roulette wheel selection)

In fitness proportional selection the probability that an individual  $i$  is chosen for mating depends on its absolute fitness value related to the absolute fitness values of the rest of the individuals in the population.

Given an individual  $i$  with fitness  $f_i$  the selection probability of such individual using fitness proportional selection is:

$$P_{FPS}(i) = \frac{f_i}{\sum_{j=1}^u f_j}$$

Over the years this selection method has been studied intensively and some of the problems which have been discovered include:

- When fitness values are very close to one another, selection pressure is practically null. Thus, when the run has passed the initial phase and the population starts to converge, population fitness will usually increase very slowly because of vanishing selection pressure.
- Outstandingly fit individuals can take over the population very quickly. This causes the search to focus on a specific location of the search space where the very fit individual lies, causing the population to initially converge quickly thus reducing the explorative traits of the algorithm which are desirable early on in order to properly assess the search space. This will lead to premature convergence.
- The selection pressure changes if the fitness function is transposed. Because the fitness proportional selection works on the relative difference of fitness between individuals, if for

example the fitness function is transposed, the relative difference in fitness of each individual with respect to the others will change.

### Ranking selection

Ranking selection was conceived to try and fix some of the issues present in fitness proportional selection.

In ranking selection, the selection probability is not directly proportional to the absolute fitness of the individual, instead, individuals are ranked based on their fitness level. This ordered list of individuals can then be used to allocate selection probabilities to each individual based on their rank, not directly on their fitness value.

Mapping a rank number to a selection probability can be done in different ways, the most common are a linearly decreasing probability or an exponentially decreasing probability of selection. Exponentially decreasing selection pressure provides a stronger selection pressure with respect to linear selection.

Considering a population of  $\mu$  individuals the selection probability of an individual with rank  $i$  where the best individual has rank  $\mu-1$  and the worst has rank 0 can be expressed using either linear ranking or exponential ranking:

$$P_{linear}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$$

$$P_{exp}(i) = \frac{1 - e^{-i}}{c}$$

Where  $1 < s \leq 2$  ensures that the worst individual does not have a probability below 0.

C instead ensures that the sum of the probabilities is 1, thus C is a function of population size.

### Tournament selection

Tournament selection works by picking a predefined number of individuals randomly from the population and then selecting the best individual from such group. The size of the selected sample is called tournament size.

Tournament selection does not require any global knowledge of the whole population and it does not need an ordered population to work. It is therefore conceptually simple to understand and to implement.

Selection pressure can easily be tuned in tournament selection by tuning the size of the tournament. A large tournament will most likely include the best individual in it; thus, a high selection pressure will result since the best individual will often be chosen. A small tournament instead will have a

lower chance of including the best individual, and might even only include poor individuals, thus allowing for a lower selection pressure.

An individual within a tournament can be selected depending on the following factors:

- It's rank in the population. This does not need to be known a priori, but it is obvious that a higher-ranking individual has a higher chance of being selected if it is included in a given tournament.
- The tournament size. As stated above, the tournament size can be used to vary the selection pressure.
- The probability that the most fit member of the tournament is selected. In the classic tournament selection format, the probability that the most fit member of the tournament is selected is 1 (Deterministic tournament), but variants exist where the probability is  $<1$  (Stochastic tournament).

The ease of implementation and the ability to control selection pressure simply through the tournament size make tournament selection one of the most widely used selection methods.

### Stud selection

Stud selection is a selection strategy which involves always selecting the fittest member of the population as one of the parents used for reproduction. In other words the fittest member of each generation acts as the *Stud*. Once the Stud has been identified, the next generation created by mating the stud with all the remaining individuals in the current population.

This mating procedure will cause the number of offspring to be one less than the current number of individuals in the population. If we want the populations to have constant size, one option is to simply carry the stud to the next generation directly, in this way all generations will have equal size.

This method of selection and reproduction obviously has its roots in animal breeding where the fittest individual is chosen to mate in order to hopefully produce the fittest offspring as possible.

#### A.3.4 Recombination (or crossover)

After parent selection has been performed, the selected parents are then combined to produce offspring, such procedure is known as recombination or crossover. The crossover operator takes the genotypes from two parents and combines them to form a given number of children. Often this number is equal to two in order to have the population size equal to the offspring number, such that the offspring directly become the new population.

Crossover is not a deterministic operation in the sense that the portions of genetic information exchanged by the two parents are not chosen a priori, instead the genetic information exchanged is randomly chosen. The structure itself of the crossover operation, meaning how the genetic

information is chosen and later combined to form the offspring, depends on the chosen crossover operator.

The structure of a crossover operator mainly depends on the representation chosen for the population. Here are a few examples of crossover operators depending on the chosen representation:

### 1) Recombination for Binary Representation

- **One Point Crossover:** initially a random number is selected in the range  $[1, l - 1]$  where  $l$  is the length of the binary string encoding. This random number indicated the position where the binary string of the two parents will be cut. Two children can be then created by exchanging the tails of the binary encoding of the two parents after the cutting point. The random number indicating the cutting point is chosen at random within the given range for each set of parents. Obviously for a given couple of parents the cutting point must be equal otherwise this would cause children with encodings of different length.

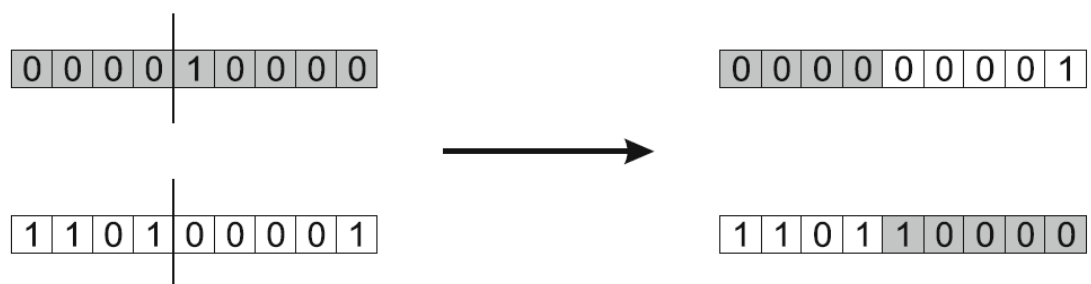


Figure A. 2 - One point crossover [62]

- **n Point Crossover:** n point crossover is simply the variant of one point crossover in which instead of having only one cutting point, multiple cutting points are used. Offspring are then created by taking alternate segments from the two parents.

The most common form of n point crossover is two-point crossover ( $n = 2$ ).

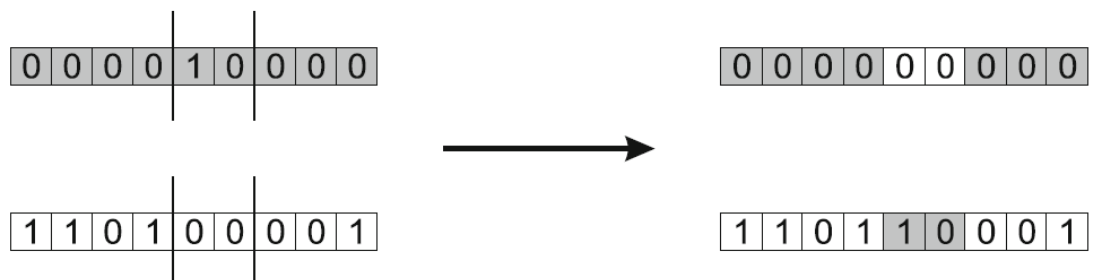


Figure A. 3 - Two-point crossover [62]



- **Uniform Crossover:** Uniform crossover differs from n point crossover in such that it does not consider sections of the genomes to be exchanged between parents, but it randomly selects single genes from each parent that will then form the new offspring.

This procedure can be implemented by creating a random binary bitstring of the same length of the parent's bitstring, which is then used to decide from which parent each gene should come from. If at a given position in the newly generated bitstring a 1 is present, then the first child will inherit such gene from the first parent, if instead a 0 is present, the first child will inherit the gene from the second parent.

A second child can be created from the two parents by either creating a new bitstring and repeating the procedure, or by simply flipping the already generated bitstring so that the second child takes its genes in a mirror like manner with respect to the first child.

In the image below, uniform crossover is performed using the second option (only one bitstring)

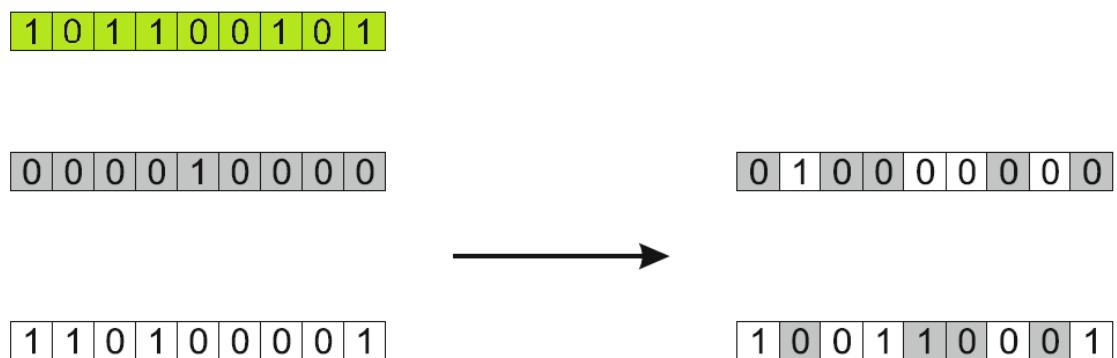


Figure A. 4 - Uniform crossover [62]

By analysing the proposed crossovers for binary representations, we can notice that each crossover presents a so-called **Bias**.

As can be noticed, n Point crossover has an inherent bias in that it keeps together genes that are close to each other in the original parents. This effect is known as **Positional Bias**. Differently, Uniform Crossover does not display any Positional Bias since there is no mechanism that inherently prevents or favours a group of genes from sticking together and being passed on to the offspring. However, Uniform Crossover does present a tendency of transmitting approximately 50% of genes from each parent and usually hinders a large number of genes coming from the same parent from being transmitted to the offspring. This is known as **Distributional Bias**.

Generally, it is not directly possible to state which crossover operator works best on any given problem. But knowing how the crossover operators work and knowing their biases can

sometimes give an insight on which crossover operator might work best for a given kind of problem.

For example, in ordering problems where an ordering of actions or items is the output of our problem, we might consider using n Point Crossover since it might help keep together parts of the sequence which work well together. This is then obviously subject to the number of objects in the list, the weight/impact each object has on the fitness function, the number of crossover points etc.

## 2) Recombination for Real Valued Representation

For what concerns Real Valued Representations, three main families of recombination operators are available.

The first option consists in recombination operators which simply assign to each of the child's gene, one of the corresponding genes from one of the two parents. Thus, a given gene of a child is either the gene of one parent or the gene of the other. This is then repeated for all genes an individual might carry. This option has the disadvantage that no new values can be added to a gene through crossover, but only through mutation. Crossover in this case acts kind of like a selector of genes and does not produce any new genetic material.

Recombination operators of this kind are called **discrete recombination** operators.

The second option are recombination operators which create new genes whose values lie **between** the parent genes used for crossover.

This can be seen as creating a new gene  $z$  from the genes  $x$  &  $y$  of the two parents as:

$$z_i = \alpha x_i + (1 - \alpha) y_i$$

Where  $\alpha \in [0,1]$ .

This method has the advantage of being able to create new gene values but has the drawback that this new value is restricted to be in between the values of the genes used from the two parents.

Recombination operators of this kind are called **intermediate** or **arithmetic recombination** operators.

The third kind of operator involves creating new genetic material from the parent genes with values which may also lie outside the range defined by the parent's genes.

Recombination operators of this kind are called **blend recombination** operators.

Next, some of the most common operators for real valued representations are illustrated:

- **Simple Arithmetic Recombination:** Simple Arithmetic Recombination is a mixture of discrete recombination strategy and an arithmetic recombination strategy. It involves simply picking a random number between one and j-1 where j is the number of genes in each individual, that is, the number of optimization variables (since each optimization variable will be represented by a real valued number for real valued representations). To perform crossover, simply perform the arithmetic average of all genes after the selected point, all genes before such point come from the 2 parents.

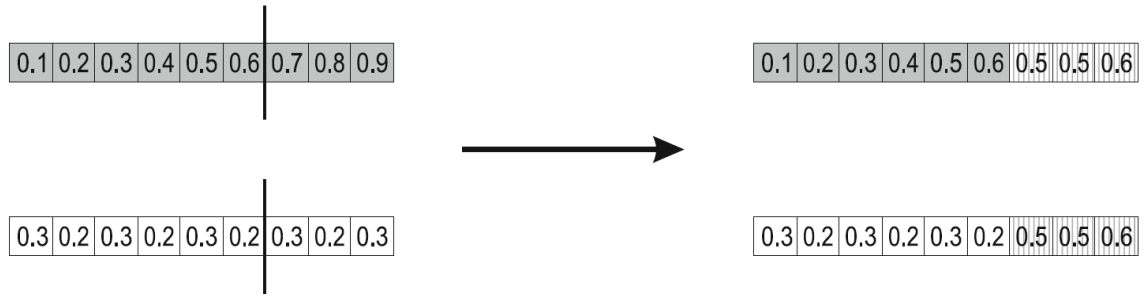


Figure A. 5 - Simple arithmetic crossover [62]

- **Single Arithmetic Recombination:** Single Arithmetic Recombination is again a mixture of discrete recombination strategy and an arithmetic recombination strategy. For this recombination, simply pick a random gene, at that gene perform an arithmetic average of the two parents, the other genes come directly from the parents.

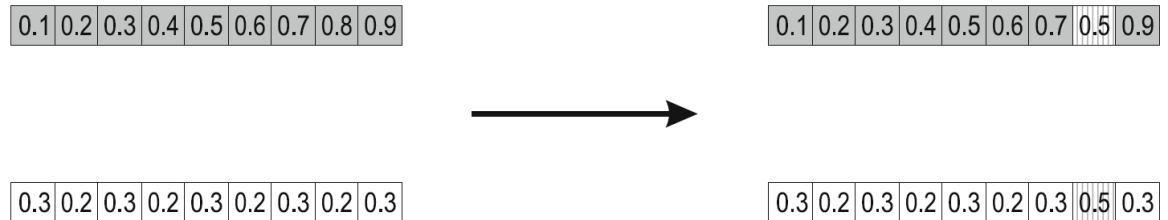


Figure A. 6 - Single arithmetic crossover [62]

- **Whole Arithmetic Recombination:** Whole Arithmetic Recombination is a fully arithmetic recombination strategy. It simply involves taking the weighted sum of each gene from the two parents. This weighted sum is controlled by the parameter  $\alpha$  which tells which parent will influence the weighted sum more greatly. Two offspring can be produced by using:

$$\text{Child 1} = \alpha \cdot \bar{x} + (1 - \alpha) \cdot \bar{y}, \quad \text{Child 2} = \alpha \cdot \bar{y} + (1 - \alpha) \cdot \bar{x}$$

If  $\alpha = 1/2$  the two offspring generated will be identical, thus usually  $\alpha$  is chosen different from  $1/2$ .

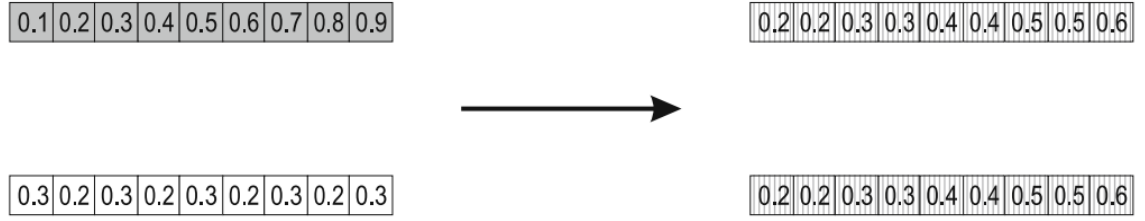


Figure A. 7 - Whole arithmetic crossover [62]

- **Blend Recombination:** Blend recombination was created in order to have the possibility to create offspring with genes whose values can lie outside of the range of the parent's genes. To create an offspring, we firstly need to sample a random number  $u$  from  $[0, 1]$ , then we can calculate

$$\gamma = (1 - 2\alpha)u - \alpha$$

The final offspring gene value can be calculated as:

$$z_i = (1 - \gamma)x_i + \gamma y_i$$

The term  $\alpha$  can be used to control how likely it is for the child's gene to fall in a range within that of the parent's genes or outside. Using  $\alpha = 1/2$  gives equal probabilities of the two events happening.

### A.3.5 Mutation

In a conventional Evolutionary Algorithm, after the crossover operation, mutation is performed on either the whole offspring population or on only part of it based on a given probability of mutation. Mutation in the process of taking a single genotype and slightly modifying it according to a predefined mutation strategy. It must be noted that mutation is a stochastic operator in the sense that the change it causes to the genotype is random and unbiased.

What can be controlled in some sense while performing mutation is with what probability will mutation occur, and, if mutation takes place, approximately how large will this mutation be. These two parameters are usually referred to as **mutation probability** and **mutation step size**.

Mutation probability can vary widely from problem to problem, but in general, different Evolutionary Algorithms use mutation differently, and thus have a different take on what ranges should the mutation probability be in. For example, Genetic Algorithms use mutation as a secondary search operator, leading to low mutation probabilities, while in Evolutionary Programming, mutation is

considered it is used as the main search operator, solely responsible for the generation of new individuals.

Another argument that can be made while choosing the mutation probability is whether the specific problem requires all the members of the population to reach the optimum or if only one member (the fittest) is required to reach the optimum solution. In the former case, a smaller mutation probability might be preferred in order to avoid disruption of a good solution of the whole population, while in the latter case, a larger mutation might allow a better exploration of the search space.

Mutation step size instead varies according to the problem to be solved and to the ranges of the optimization variables in question. Obviously, a large mutation step will be preferred when the search space is very large and when the optimum might have a “soft” shape, while is the search space has a very small range, small mutation steps are needed.

The operator through which mutation takes place depends on the type of representation used. Some of the main mutation operators for different representation classes are presented next.

### 1) Mutation for Binary Representation

- **Bitwise mutation:** Bitwise mutation consists in creating a vector of random numbers uniformly distributed between 0 and 1 of the same length as the genotype binary vector. This random vector is then compared with the mutation probability  $p_m$ . At each position in which the randomly generated number is smaller than the mutation probability, a bit flip is performed in the binary genotype. Thus, on average, an increase in mutation probability will cause an increase in the number of bits flipped. An example is presented with a probability of mutation of 50%.

$$p_m = 0.5$$

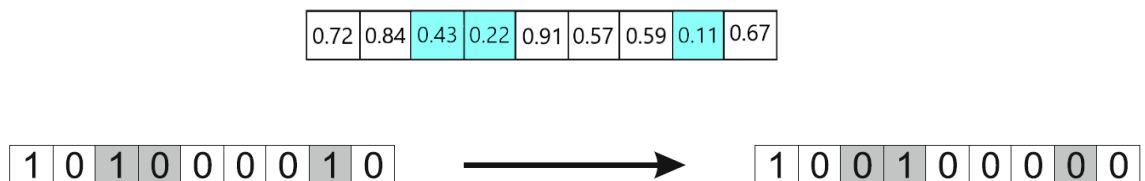


Figure A. 8 - Bitwise mutation

## 2) Mutation for Real Valued Representation

For real valued representations, the value that a genotype can take is no longer discrete, thus mutation operators simply involve mutating each original genotype to a value within a predefined domain given by a lower bound and an upper bound for each gene.

The lower and upper bounds are obviously problem dependent.

$$\langle x_1, \dots, x_n \rangle \rightarrow \langle x'_1, \dots, x'_n \rangle, \quad \text{where } x_i, x'_i \in [L_i, U_i].$$

- **Uniform Mutation:** For uniform mutation, the mutated values ( $x'_i$ ) are drawn from a uniform random distribution with bounds  $[L_i, U_i]$ . This is very simple to implement but has the disadvantage that the size of the mutation cannot be controlled since is equally probable to pick any value within the range. Thus, depending on what the original gene value was, the step caused by mutation might be very small or very large and anywhere in between.
- **Non-Uniform Mutation:** Non uniform mutation is designed so that the size of the mutation step can somehow be controlled and is most likely to be small than very large. This is achieved by adding to the current gene a value drawn randomly from a Gaussian distribution with zero mean. The size of the step can be controlled by appropriately choosing the standard deviation of the Gaussian distribution. Increasing the standard deviation will increase the likelihood of picking larger mutation values while reducing the standard deviation will ensure that the mutation step size will most likely be small. In the literature, the standard deviation of the Gaussian distribution is in fact referred to as *mutation step size*.

After adding the value picked from the Gaussian distribution, if necessary, a truncation can be performed to respect the predefined gene bounds  $[L_i, U_i]$ .

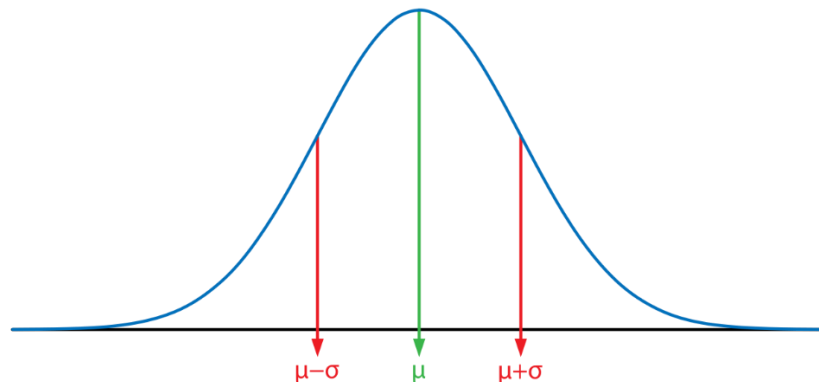


Figure A. 9 - Non uniform mutation with gaussian distribution of step size

### A.3.6 Survivor Selection

Survivor selection is used in order to decide which members of the current child population and parent population are going to go and form the next parent population, thus survival selection is used at the end of a single evolutionary cycle, before a new cycle begins.

Selection is needed in order to prevent the population size from continuously growing and can be also used to try and keep the better individuals while discarding the worst ones.

The process of discarding the worst individuals although might need to be carefully regulated since in some scenarios, simply discarding all the lesser fit individuals and only allowing the fittest to survival may cause premature convergence to a local optimum.

In principle any of the mechanisms used for parent selection can also be used for survival selection but over the years many tailored solutions for survival selection have been proposed.

The main driving factors that are used for survival selection are either age or fitness. Age can be used to discard the “n” oldest individuals each time selection has to be performed while fitness-based selection usually discards the “n” less fit individuals. Stochasticity can be added if needed to make the processes nondeterministic.

#### 1) Age-Based Survival Selection

Age-based survival mechanisms are designed to ensure that each individual exists in the population for the same number of EA iterations. This kind of survival mechanism does not use any fitness information of the individuals, and thus might seem to be detrimental to the goal of reaching an optimum through fit individuals. But as long as it is coupled with a sufficient selection pressure from the parent selection stage and with a variation operator (mutation and crossover) which is not too disruptive, a steady increase in the average population fitness should be observed.

Age based survival selection may take different forms based on the number of offspring ( $\lambda$ ) and the number of members of the population ( $\mu$ ).

**$\mu = \lambda$  :** When the number of offspring is exactly equal to the number of members of the population, at each cycle the parents are discarded, and the offspring form the new population. Each individual exists for one generation only.

**$\lambda < \mu$  :** When the number of offspring is smaller than the number of members of the generation the  $\lambda$  offspring are simply inserted in each generation by eliminating  $\lambda$  individuals to make space. Initially the individuals eliminated are picked at random until the initial population disappears. After  $\mu/\lambda$  cycles, it will be possible to deterministically eliminate the oldest  $\lambda$  individuals at each cycle.

## 2) Fitness Bases Selection

Fitness based survival selection mechanisms use the fitness information of the  $\mu$  individuals plus the  $\lambda$  offspring to decide which individuals should pass to the next generation.

Some examples of fitness-based selection strategies are presented next.

- **Elitism:** This technique is usually used in conjunction with age based survival selection schemes in order to prevent the loss of the fittest member in each generation. Simply, each time survival selection takes place, whatever method for selection is used, the currently fittest member of the population is always inserted in the next generation.
- **$(\mu + \lambda)$  Selection:** For this selection scheme, the parent and offspring population are merged together and ordered according to their fitness. The top  $\mu$  individuals of the newly merged population will then go and form the next generation.

This kind of selection introduces a very large selection pressure since only the best individuals are kept. Because of this it is usually coupled with parent selection methods which offer relatively low selection pressure. Furthermore, this kind of selection is often detrimental if a self-adaptation of the EA parameters is in place.

- **$(\mu, \lambda)$  Selection:** This selection scheme is used when  $\lambda \geq \mu$ , thus when the number of offspring is larger or equal to the number of members of the population. This strategy involves discarding the parents and only keeping the offspring. If the number of offspring is equal to  $\mu$ , then no further computation for the selection is needed, instead, if the number of offspring is larger than  $\mu$ , the offspring are ranked according to their fitness and only the top  $\mu$  are selected as the next generation. Because at each generation, all the parents are discarded, this kind of selection may be useful for optimization problems involving multimodal landscapes, allowing the population to escape a local minimum more easily.

### A.3.7 Termination Condition

The above steps that were previously discussed form the backbone of a single cycle/generation of an evolutionary algorithm. The algorithm will then continuously run through generations until a termination criterion is met. Ideally such termination criterion would be that the algorithm has found the solution leading to the optimum of the function it is trying to optimize. But since EAs are stochastic by nature, there is no guarantee that the optimum will be reached, and furthermore, such optimum may not be known a priori. For these reasons a termination condition is needed in order to stop the algorithm from running indefinitely. The main termination conditions used are:



- The number of generations reaches a certain limit.
- The fitness improvement remains below a predefined threshold for a predefined time.
- The population diversity is below a predefined threshold.

The first condition is usually the most widely used condition for termination.

If the function optimum was known, another termination condition may be the arrival to the target solution.

### A.3.8 Performance Measure

As a user designs an EA, a need to assess its performance compared to other optimization algorithms or compared to different implementations of the same EA is needed.

Because EAs are stochastic by nature, a single run cannot be used to empirically determine the performance of an EA, instead a number of experiments needs to be performed to gain sufficient experimental data to assign a certain performance measure to the EA.

The main solution measures are based on:

- Success rate
- Solution quality
- Speed
- **Success rate (SR)** involves defining or knowing a priori a target solution to the problem at hand. This is usually straight forward in standard academic problems where the optimum solution is known, while it can also be defined for a real-world problem if enough information about the problem is known.

An example of a possible target for a real-world problem in which the optimum is not exactly known is to mark as a successful run, a run ending below a certain fitness threshold within a given number of generations. If a run manages to stay within the predefined conditions, it can be marked as successful, if it does not it is not successful.

The success rate is then simply the rate of successful runs within all the runs tested. As stated earlier, a good number of runs must be performed to obtain a reliable performance measure. In general, SR is used when the solution to our problem is known a priori so that the success rate is the rate of successful runs that have found the optimum of the selected function.

- **Mean best fitness (MBF)** is another performance measure which is instead directly linked to solution quality. To calculate MBF, the fitness of the best individual at each complete run of an EA is recorded, the MBS is then the average of such values over all runs.

- **Average number of Evaluations to a Solution (AES)** considers as a performance measure the speed of an algorithm by using the time it has taken, on average, to reach a predefined solution. It must be noted that only successful runs (runs which manage to reach the predefined solution) are used to calculate the AES since if also the runs which did not find the solution were used, their number of evaluations would depend on the stopping criterion and not at all on the target solution set, thus skewing the AES result.
- **Best Ever Fitness and Worst Ever Fitness:** Additionally, to the previously mentioned fitness measures, which are the most used fitness measures since they give a statistical result over a number of runs, best ever fitness and worst ever fitness can be two additional performance measures which may be particularly useful in scenarios in which a single excellent or terrible run may be important to log. For example, for one off design problems, the best ever fitness may be the most important measure since one very good solution is all we are looking, on the other hand, the worst ever fitness may give an insight on how bad the worst case scenario might be for an algorithm that is running a repetitive problem and needs to run multiple optimization cycles in real time, in which each obtained result must be used and cannot simply be discarded.

Regarding SR and MBF it must be noted that both these measures can be used to compare algorithm runs and implementations for a predefined limit on the computational time. If the maximum computation time is changed, obviously the SR and MBF will change, not allowing a fair comparison. This can be intuitively seen in the graph below where two stopping times are chosen and based on the stopping time, algorithm A and B actually swap position in their performance ranking. Obviously both results are valid if we are comparing A and B with the same maximum computation time, what must be avoided in order to have a fair comparison, is to compare the two algorithms with different maximum computation times.

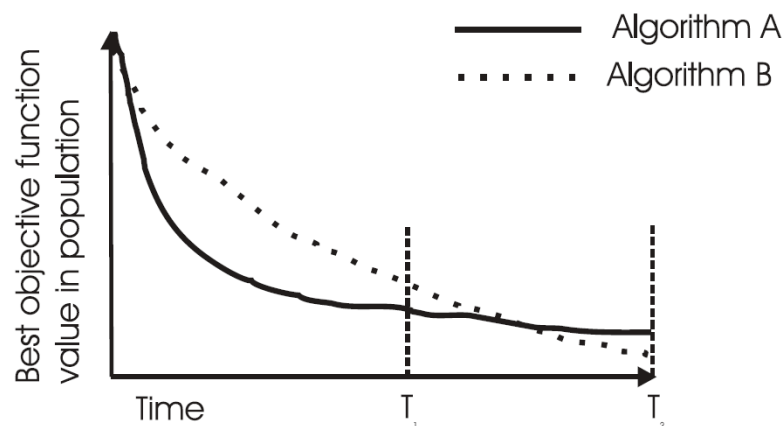


Figure A. 10 - Comparison of different stopping times [62].

Thus recapping, for SR and MBF we define a priori a maximum computation time (e.g. generations) and measure the algorithms effectiveness within this time window. For AES instead, the user must first define a suitable level of target fitness, AES then considers how much time is needed on average to reach such predefined level of fitness.

Although AES can generally give a good indicator of how fast an algorithm is, it can sometimes be misleading depending on the structure of the EA.

Since in general AES only takes in to account the number of evaluations or of generations elapsed until a solution is found, the total algorithm run time may vary widely depending on how long a single evaluation (or generation) takes. This will obviously depend on how the EA is structured and if any 'hidden labour' is present in any of the operators of the EA (e.g., some form of local search in the mutation operator).

An argument can be made of then replacing the number of evaluations to solution for another time dependent parameter, for example the CPU run time to solution, but this would cause problems in comparing EA on different platforms since the run time would now depend on the hardware on which the EA is running and on how such hardware is being currently utilised.

- **Progress Plot:** Another interesting performance measure which can be used to analyse the performance of an EA is a plot showing the progress of the fitness measure of the most fit individual over time. The plot can either represent a single run or can represent an averaged value of fitness in time over multiple runs, giving more robustness to the comparison. This kind of plot can be very useful to compare the performance of EAs since it can give a lot of information to the user such as the number of runs until convergence, or the steepness of the curve at the end of the run, giving an indication if a possible improvement could be made by extending the run time, etc.

## Appendix B: Neural Networks

Artificial Neural Networks (ANNs) are a tool which can be used to perform machine learning, such that an algorithm learns to perform a certain task by analysing training examples of the same class of the task we later need it to perform autonomously.

ANNs are used anywhere from speech recognition, image classification, to data regression.

Their name stems from the fact that their structure is inspired by the human brain, mimicking the way that biological neurons transmit signals to one another.

Many types of neural networks currently exist based on their structure and internal workings. The *Feed Forward Neural Network* can be considered as the simplest neural network and was the first kind of Artificial Neural Network to be conceived.

Feed Forward Neural Networks are networks in which information can only be passed forwards, so from input to output and not vice versa.

The basic building block of any neural network is a *Neuron*. In Feed Forward neural networks neurons are stacked to produce what is called a *Layer*.

Layers can then be placed in a sequential manner in order to produce the basic structure of a Feed Forward Neural Network.

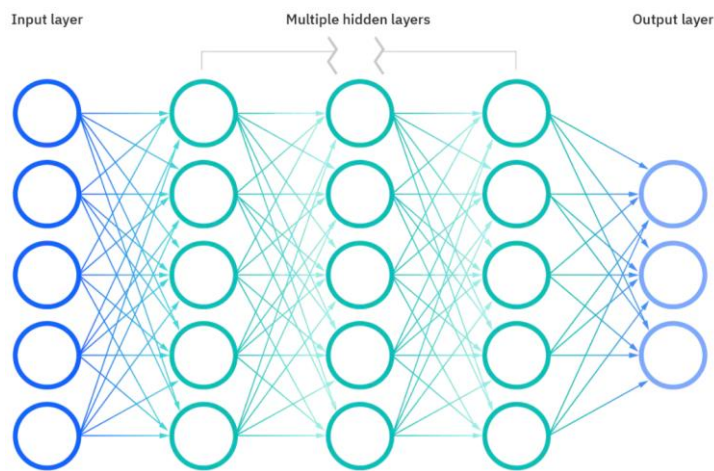


Figure B. 1 - Basic structure of a feed-forward neural network

As can be seen from the above image, in Feed Forward Neural Networks (FFNNs) the output of each neuron is passed on as information to each neuron on the next layer. This produces what is known as a *Fully Connected Layer* (each neuron passes its output to all neurons in the next layer).

Notice that the number of neurons in any given layer does not depend on the number of neurons in the other layers. The number of neurons in a given hidden layer is a design specification which can be set in order to create a Neural Network with the desired working characteristics.

What is important though, is the number of neurons in the input layer and in the output layer. As can be assumed from their names, the neurons in the input and output layers represent the inputs we feed

to the neural network and the corresponding outputs we need the net to produce. Thus, it is obvious that the number of neurons in the input and output layers is problem dependant.

## B.1 Neurons

As stated earlier, the basic building block of an Artificial Neural Network is called a neuron.

Neurons are simply nodes whose purpose is to take an input and produce an output based on the *activation function* and *bias* assigned to such neuron.

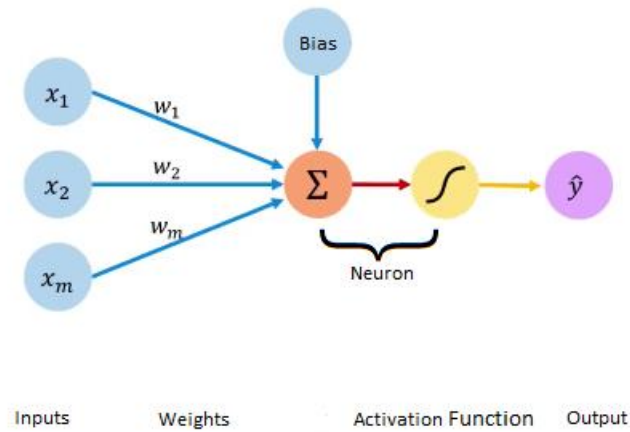


Figure B. 2 - Working principle of an ANN neuron

As we saw earlier, the input to any given neuron is coming from the neurons in the previous layer. To produce the output of a neuron, firstly the *weighted sum* of all the inputs is performed. This implies having a certain weight assigned to each connection between neurons. To such sum a *bias*, which is simply a number, can be added in order to shift the result of the sum preferentially towards a given direction or in order to create a threshold based on such bias. After the weighted sum is performed, the resulting number is passed through an *Activation Function* whose purpose is to add a nonlinear behaviour to the algorithm or also to simply bound the output value of the neuron within a certain range. The resulting output from the activation function is then the final output of the neuron, this output is then passed on to all neurons in the next layer and the cycle continues until the information gets to the final layer, the output layer.

The equation for the output of a neuron with activation function  $g$  and bias  $w_0$  can be written as:

$$\hat{y} = g \left( w_0 + \sum_{i=1}^m x_i w_i \right)$$

## B.2 Activation functions

Many types of activation functions have been proposed through the years [76]. Each has its own advantages and disadvantages, and many are only suited for a particular class of problems.

### B.2.1 Hidden layer activation functions

Typically, nonlinear and differentiable functions are used as activation functions in hidden layers. Nonlinear functions allow the network to have non linearities in its governing equations and in turn allows to learn more complex functions. Differentiability is key since the derivative of the activation function will be later used by the backpropagation algorithm.

The most commonly used activation functions for a hidden layer are:

- Rectified Linear Unit (ReLU)
- Sigmoid
- Hyperbolic Tangent (Tanh)

#### 1) ReLU

The rectified linear activation function is one of the most commonly used for the hidden layers in modern Neural Networks and was popularized in 2010 by Nair and Hinton for Restricted Boltzmann machines [77] .

Its recent popularity is due to the fact that it is less susceptible to the *vanishing gradient* problem which prevents deep networks from improving their learning abilities [78] [79].

For all its advantages, the ReLU function does suffer from what is called the *dying ReLU problem* [76].

This occurs when a large number of ReLU neurons only output 0 and thus the gradient will fail to flow during back propagation (which is how Neural Networks learn) since the function's gradient is zero in the left half plane. This will cause a large part of the network to become inactive or *die*.

This phenomenon can be prevented by either using smaller learning rates or by simply replacing the ReLU function with a Leaky ReLU.

The Leaky ReLU adds a small slope for negative inputs. This allows the output to be non-zero and thus it prevents neurons from dying off.

The mathematical definition of the ReLU function is as follows

$$g(x) = \max(0, x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

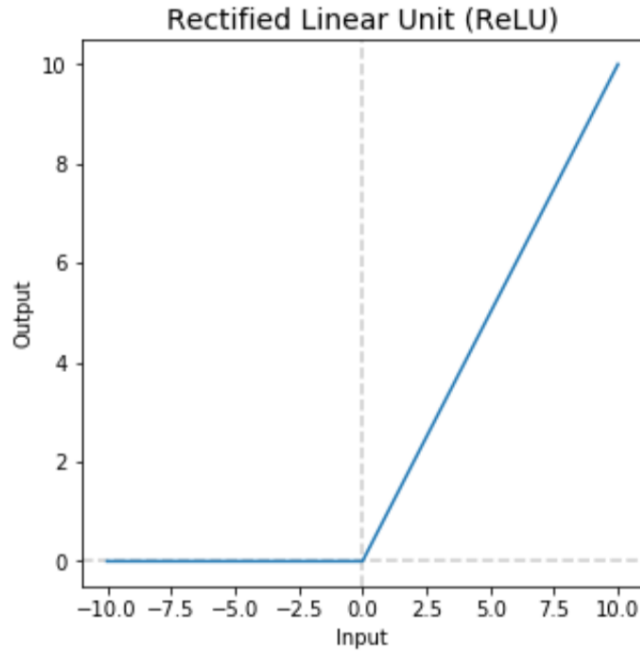


Figure B. 3 - ReLU activation function

The mathematical definition of the Leaky ReLU function is as follows

$$g(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha \cdot x & \text{if } x < 0 \end{cases}$$

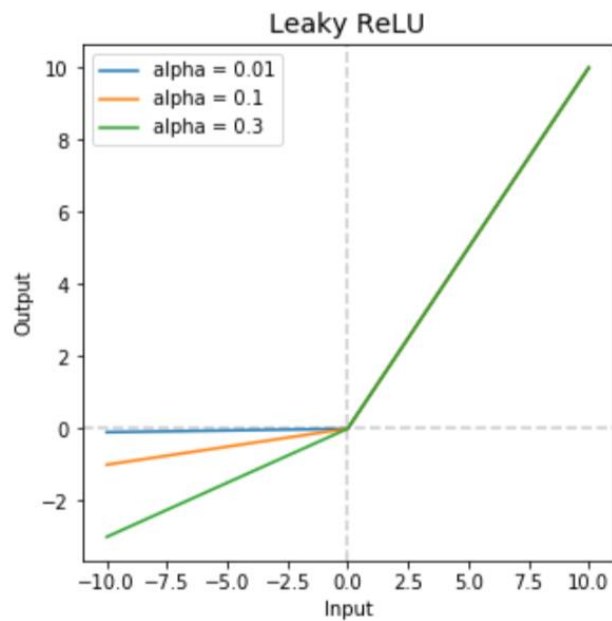


Figure B. 4 - Leaky ReLU activation function

Some of the advantages of using this activation function can be [80]:

- Neural networks using ReLU activation functions are computationally cheaper than networks using sigmoid or tanh activation functions.
- Neural networks that use ReLU activation functions usually converge much faster than networks using saturating activation functions with gradient descent.
- The derivative of the ReLU activation function is equal to 1 in the right-hand plane of the function. This can help avoid trapping into local optima and resolves the vanishing gradient problem.

The main problem which accompanies ReLU functions is that the function derivative in the left plane is equal to 0, meaning it's left-hard-saturating. This may lead to what is known as the *dying neuron* phenomenon which causes the affected neurons to shut down and their weights and biases to not be updated and the neurons will not be activated any longer.

## 2) Sigmoid

The sigmoid activation function, sometimes called a “squishing” function, maps all input values in a range between 0 and 1. This activation function is inspired from the activation functions found in the neurons of our brain where often neurons fire following a sigmoidal trend.

As can be seen, the sigmoid is a non-symmetric function, thus all neurons will have only positive outputs. This can sometimes be a problem which can be addressed by using a Tanh function.

The mathematical definition of the sigmoid function is as follows.

$$g(x) = \frac{1}{1 + e^{-x}}$$

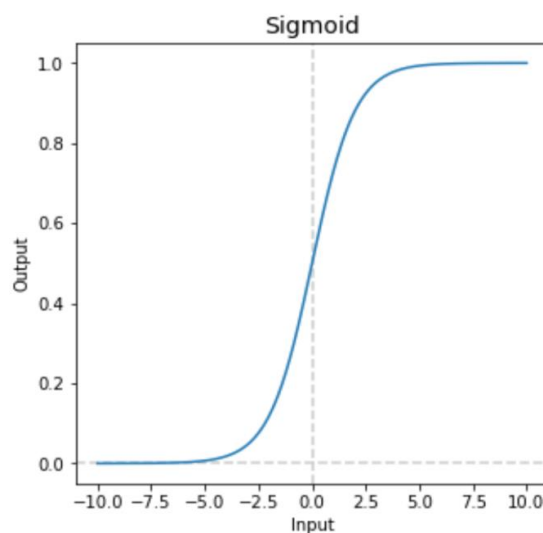


Figure B. 5 - Sigmoid activation function



Because of its inherent saturation for large magnitudes of inputs, the sigmoid (and also tanh) function is seldomly used for the hidden layers of deep neural networks because of the vanishing gradient problem it can cause. The sigmoid function is thus usually used for shallower networks or even as an output layer activation function because of its bounding abilities with a smooth transition between 0 and 1.

### 3) Tanh

The Hyperbolic Tangent (Tanh) activation function is very similar to the sigmoid function, the only difference is that it's symmetric with respect to the origin. This allows the output of neurons to take both positive and negative values between -1 and +1.

The mathematical definition of the tanh function is as follows.

$$g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

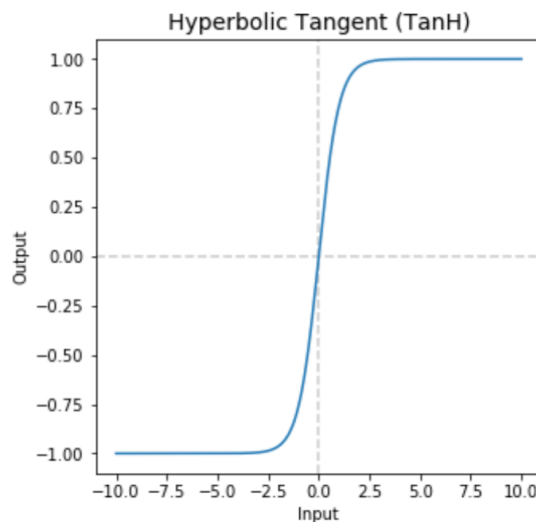


Figure B. 6 - Hyperbolic tangent

The hyperbolic tangent activation function is usually preferred over the sigmoid function because of its symmetry with respect to 0 which means that most of its outputs will usually be small. In addition, nets using the tanh function converge faster than those using sigmoid activation functions [81].

### B.2.2 Output layer activation functions

The most commonly used activation functions for the output layer are:

- Linear activation function
- Sigmoid activation function
- Softmax activation function

#### 1) Linear activation function

The linear activation function, as its name suggests, linearly transposes the input to the output. When using this function, the output is not bonded and thus can take any value.

This function is only usually used in output layers since as stated earlier, to map complex data, a non-linearity must be introduced in our net, and this being a linear function it could not serve such purpose. It is instead often used as an output activation function in regression problems where we don't want the output data to be altered in scale.

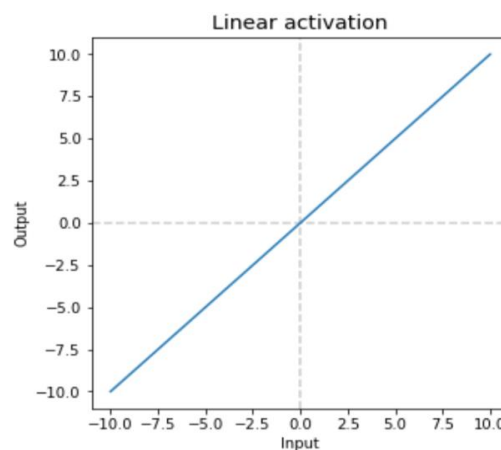


Figure B. 7 - Linear activation function

#### 2) Sigmoid activation function

This function was already presented earlier as a candidate for a hidden layer activation function. It can also be used as an output function when we need our output to be scaled between 0 and 1. This might happen when for example we are facing a multilabel classification problem and need to know to which categories our inputs belong to, or when we need to for example predict a probability

#### 3) Softmax activation function

A softmax activation function outputs a vector of values which sum to 1. The size of the output vector is equal to the size of the input vector thus, this function is especially useful in multiclass classification when we need to classify an object and select the appropriate class. Softmax thus returns the single probabilities that such object belongs to a certain class, a decision can then be made from the obtained probability values.

## B.3 How neural networks are trained

Before ANNs can be used to solve actual problems, ANNs must be *trained*.

Training involves using a training dataset which is comprised of a series of inputs and a series of known desired outputs that we wish the net to produce when it is fed the corresponding inputs.

An example might be a classification problem in which we show the net a series of images of cats and dogs, each of which is labelled with the correct name, either cat or dog. We then show the neural network such image inputs and we also tell the net which are the correct outputs it should produce, so which name corresponds to which image. This is known as the training phase of the network.

During this phase, the network updates its internal weights and biases, based on the training set it is provided with, in order to learn whatever task it was assigned to solve. In the simple case above it learns to sort images of cats and dogs.

The process of updating the biases and weight during training occurs through an optimization of the *loss function* or *error function* or *performance function* which tells the user how well (or badly) the network is doing on the training data, so how close is the output of the network to the desired output. Since Artificial Neural Networks learn to map inputs to desired outputs from the examples proposed during the training session, an adequate loss function must be used depending on the specific problem at hand, may it be for example a classification or a regression problem.

Choosing the right loss function for the problem at hand is a critical step, a bad choice of the loss function might lead to unsatisfactory results simply because the loss function is not adequately representing the performance of our neural network.

In this text, only loss functions suited for regression problems will be analysed since no classification problems are faced in this discussion.

### B.3.1 Loss functions for regression problems

In a regression problem, the network is tasked to learn how to predict real valued quantities. One example might be the prediction of the remaining milage in a car given a set of signals coming from the accelerator, how much fuel is left in the tank etc. The most common loss functions used for regression problems are:

- Mean absolute error loss (MAE)
- Mean squared error loss (MSE)
- Root mean square error loss (RMSE)

### Mean absolute error (MAE)

The mean absolute error loss is probably the easiest error function to implement because it is simply the mean of the absolute error (difference) between the actual and predicted output values during training.

The equation for the MAE can be written as:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

From the above formula we can see that the MAE increases linearly with the size of the error.

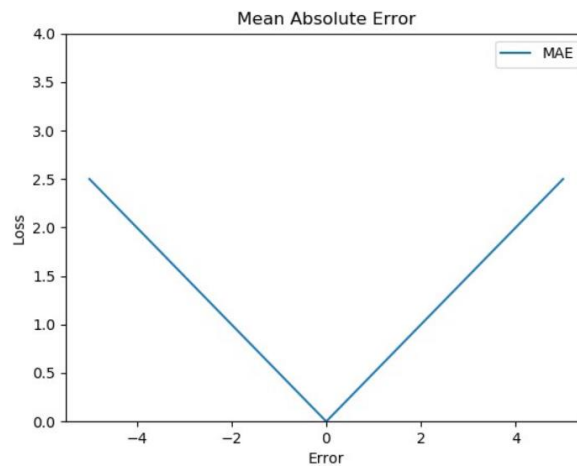


Figure B. 8 - Mean absolute error

### Advantages

- Easy to implement
- Computationally inexpensive
- Robust to outliers

### Drawbacks

- MEA does not consider the order of magnitude of the outputs. If the Neural Network under consideration has multiple outputs and they have different orders of magnitude, when computing the MEA this will not be considered. This is obviously a drawback since MAE of 1 on a scale of 10 is very different than on a scale of 1000.
- MAE presents a large gradient even for small error vales. This is detrimental for the learning process in ANNs.

## Mean squared error (MSE)

The mean squared error (MSE) is the go-to loss function when dealing with regression problems. The MSE is calculated by computing the square of the errors and then taking its mean.

This created a quadratic scoring method which is not proportional to the error as in MAE but it's proportional to the square of the error. This causes the loss to be much greater for relatively larger errors while smaller errors are not penalized so much.

This behaviour can be understood by looking the MSE equation and the graph correlating MSE to loss.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

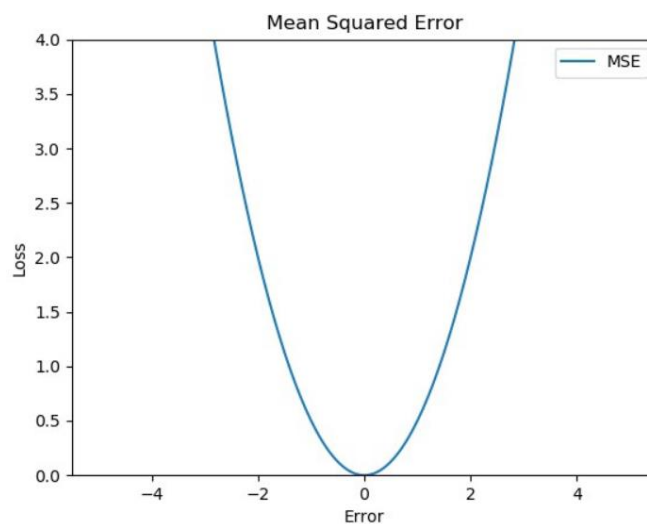


Figure B. 9 - Mean squared error

### Advantages

- The gradient reduces gradually as the error shrinks, thus allowing the optimization algorithm to converge to the minimum efficiently.

### Drawbacks

- The very large loss caused by a large error might cause drastic jumps during backpropagation and the update of weights and biases, which is usually undesired.
- MSE is sensitive to outliers

## Root mean squared error (RMSE)

As the name suggests, RMSE is simply the MSE with an additional square root applied.

This renders RMSE a linear loss function, similar to MAE.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

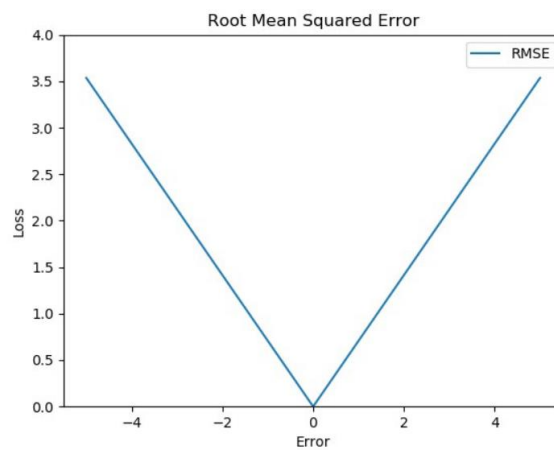


Figure B. 10 - Root mean squared error

### Advantages

- Penalizes larger errors more than MAE. This is a desirable trait if we are facing a problem where the seriousness of the error grows faster than the error itself. So having an error of 10 is more than double as bad as having an error of 5.

### Drawbacks

- Just like MAE, it's a linear function of the error, thus its gradient is large even when small errors are present.

## B.4 Optimization techniques for neural networks

Once a performance function/loss function has been defined, an optimization procedure must be performed to minimise the function in order for the network to increase its performance and learn how to solve a specific set of tasks. This process is an iterative process where many training examples are fed to the network and where the network's parameters (weights, biases etc) are modified in order to minimise the chosen performance/loss function which in turn reflects the performance of the network.

There are several different classes of network learning laws, such as associative, competitive learning and performance learning. Many different learning laws exist that fall under the category of performance learning. These learning laws are distinguished by the fact that during training the network parameters are adjusted in an effort to optimize the “performance” of the network.

The general setup for performance learning involves two steps.

The first step is to choose the performance/loss function which will define what is meant by network “performance”. The performance index must be chosen so that it guarantees that the corresponding performance surface has a minimum point (the target optimum).

The second step is to perform a search of the parameter space of the network in order to try and reduce the performance index and hopefully reach a minimum of the performance function.

Note that the performance function will be a function of the network parameters, i.e., the weights and biases.

### B.4.1 Conditions for optimality

Now that the general setting of the problem at hand has been defined it is possible to define some necessary conditions for optimality of a candidate point on the performance surface.

Considering  $F(x)$  as a multivariable function representing the performance function whose variables represent the variables in the neural network structure (weights and biases) and considering  $x^*$  as a candidate point for a minimum of the performance function, we may write multiple conditions in order to ensure that such point is a minimum of the performance function.

#### 1) First order conditions

A first order condition is for the gradient at  $x^*$  to be equal to zero. This is a necessary but not sufficient condition for  $x^*$  to be a local minimum point.

$$\nabla F(x)|_{x=x^*} = 0$$

Any points satisfying the above equation are called stationary points.

## 2) Second order conditions

Assuming now that point  $x^*$  is a stationary point a new condition is needed to verify that such point is a minimum point.

- A necessary condition for  $x^*$  to be a minimum (strong or weak minimum) point is that the Hessian matrix related to  $F(x)$  must be positive semidefinite. This condition can be tested by verifying that all the eigenvalues of the matrix are non-negative.

$$\nabla^2 F(x)|_{x=x^*} = \text{positive semidefinite}$$

- A sufficient condition for  $x^*$  to be a strong minimum point is that the Hessian matrix related to  $F(x)$  must be positive definite.

$$\nabla^2 F(x)|_{x=x^*} = \text{positive definite}$$

### B.4.2 Performance optimization: basic optimization algorithms

In the first part of this chapter, the main basic optimization strategies will be presented, namely gradient descent with steepest gradient, conjugate gradient and the stochastic/minibatch variants.

Again, in this chapter we consider the performance function as a given without specifying which performance function is used.

Given a performance function  $F(x)$  which in some sense reflects the performance of the neural network, the objective of the optimization algorithm is find a value of  $x$  (where  $x$  represents the vector variables on which  $F$  is dependent on) that minimizes  $F(x)$ .

Most optimization algorithms are iterative algorithms. These algorithms begin their search from an initial guess  $x_0$  and then update the guess based on an equation of the form:

$$x_{k+1} = x_k + \alpha_k p_k$$

Where  $p_k$  is a vector representing a search direction and  $\alpha_k$  is a positive scalar called learning rate which determines the size of the step in the  $p_k$  direction.

The algorithms discussed from here on are distinguished by how each chooses the search direction  $p_k$ .



## Steepest Descent

The steepest descent algorithm entails moving at each step always in the direction of steepest descent. Calling  $g_k$  the gradient evaluated at the “old” guess

$$g_k = \nabla F(x)|_{x=x_k}$$

Any vector  $p_k$  that satisfies:

$$g_k^T p_k < 0$$

Is called a descent direction. In practice this means that if we take a small enough step in this direction the function will decrease. Although taking enough adequate steps in a given set of generic descent directions would bring to the minimization of the performance function, a better solution would be to choose the descent direction of steepest descent at each iteration. The direction of steepest descent occurs when  $g_k^T p_k$  is most negative.

Therefore, the vector  $p_k$  that points in the direction of steepest descent is given by:

$$p_k = -g_k$$

From this definition it is possible to define the method of *steepest descent*:

$$x_{k+1} = x_k - \alpha_k g_k$$

Regarding the *learning rate*  $\alpha_k$  different techniques can be used to select its value. The simplest methods use either a fixed learning rate throughout the run or a variable learning rate that changes according to a predetermined *rule* (e.g.,  $\alpha_k = 1/k$ ).

The size of the learning rate is an important hyperparameter that must be chosen carefully in order to obtain good performance from the algorithm.

Consider an example of a 2-variable performance function  $F(x_1, x_2)$  for which we can draw the contour plot. For small learning rates, the steepest descent trajectory will follow a path that is practically always orthogonal to the contour lines. This will guarantee a precise and efficient path towards the minimum but might take a long time because of the very small steps taken at each iteration.

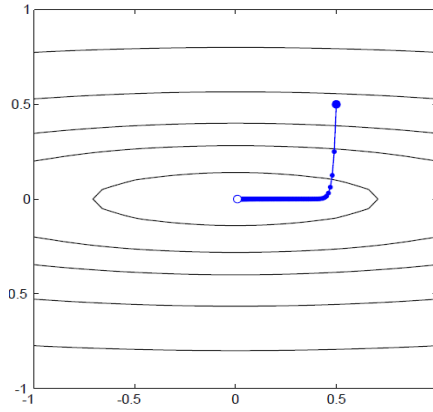


Figure B. 11 - Trajectory for steepest descent with small  $\alpha$

Although a larger learning rate might be desired to speed up convergence, if the learning rate was set too high the result might be an oscillating trajectory which in turn will not save time.

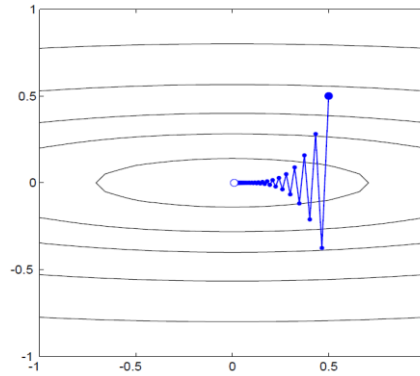


Figure B. 12 - Trajectory for steepest descent with larger  $\alpha$

If the learning rate is still increased at this point, an unstable learning rate might eventually be reached for which convergence will not be possible and the algorithm's oscillations will not decay but will grow.

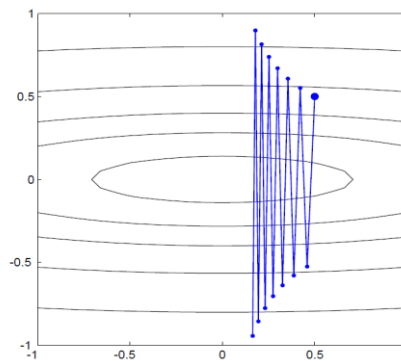


Figure B. 13 - Trajectory for steepest descent with unstable  $\alpha$

The  $\alpha$  value which guarantees an upper stability limit to the algorithm cannot be found for any arbitrary performance function, but it is possible to show that for quadratic functions such limit is:

$$\alpha < \frac{2}{\lambda_{max}}$$

Where  $\lambda_{max}$  is the largest eigenvalue of the Hessian matrix of the performance function  $F(x)$ .

## Conjugate Gradient

The conjugate gradient method is similar to the steepest descent method but instead of using directional vectors pointing in the direction of steepest descent it uses conjugate vectors.

It can be shown that if a sequence of linear searches is performed along a set of conjugate directions, then the exact minimum of any given quadratic function will be reached in at most  $n$  steps, where  $n$  corresponds to the number of parameters characterizing the quadratic function. This property is known as *quadratic termination*. This property makes the conjugate gradient method much more efficient than the simple steepest descent method.

Other search algorithms such as Newton's Method also possess this characteristic, but the conjugate gradient method possesses the advantage (compared to Newton's Method) that it does not need to calculate and store the second derivatives of the performance function. This is particularly advantageous when dealing with functions of a large number of variables as in neural networks since the Hessian of a function of  $n$  elements requires the calculation of  $n^2$  elements.

It can be shown that search directions are conjugate if they are orthogonal to the changes in the gradient at successive iterations of the algorithm:

$$\Delta g_k^T p_j = 0 \quad k \neq j$$

It must be noted that the first search direction is chosen arbitrarily. It is thus common practice to choose the first search direction in the direction of steepest descent:

$$p_0 = -g_0$$

As for the steepest gradient method, the step at each iteration can always be defined as:

$$\Delta x_k = \alpha_k p_k$$

The size of the learning rate  $\alpha_k$  must be chosen so that we minimise the performance index with respect to  $\alpha_k$  at each iteration. For a generic performance function this requires a line search, but for a quadratic function it can be shown that given a certain direction  $p_k$ , the value of  $\alpha_k$  which minimises  $F(x_k + \alpha_k p_k)$  is:

$$\alpha_k = -\frac{g_k^T p_k}{p_k^T A_k p_k}$$

Where:

- $A_k$ : Hessian matrix evaluated at old position  $x_k$
- $g_k$ : Gradient direction
- $p_k$ : direction vector for  $k^{\text{th}}$  step

Finally, at the successive iterations, it is necessary to compute the next vector  $p_k$  which must be orthogonal to  $\{\Delta g_0, \Delta g_1, \Delta g_2, \dots, \Delta g_{k-1}\}$ .

It can be shown that such a vector can be constructed as:

$$p_k = -g_k + \beta_k p_{k-1}$$

The values of  $\beta_k$  can be canonically chosen from the following most common choices:

$$\beta_k = \frac{\Delta g_{k-1}^T g_k}{\Delta g_{k-1}^T p_{k-1}}$$

*from Hestenes and Stiefel*

$$\beta_k = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}}$$

*from Fletcher and Reeves*

$$\beta_k = \frac{\Delta g_{k-1}^T g_k}{g_{k-1}^T g_{k-1}}$$

*from Polak and Ribière*

## Stochastic gradient descent / mini batch stochastic gradient descent

Gradient descent in its simplest form uses the gradient information of the individual losses of each training example in the whole training dataset before making a single well-informed step in the direction of steepest descent. Although this allows for a precise step based on all training information, it increases training time. A solution to the problem, is to use either stochastic gradient descent or mini batch stochastic gradient descent where the gradient is not calculated using the whole dataset, but only a subset of it (mini-batch) or one single training example (stochastic).

Stochastic gradient descent works just as gradient descent does, but instead of using the whole dataset before making a decision, weights and biases are updated at each training example.

This in turn makes the whole training process take steps much more often.

The drawback is that since each step is taken while considering only a single training example, the direction of the step is only based on the gradient delivered by such training example. This causes the step to very likely be in a direction that is not exactly optimal for the whole dataset.

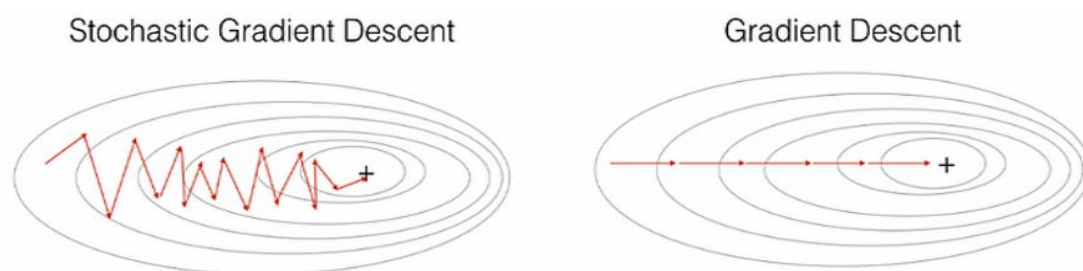


Figure B. 14 - Examples of trajectories taken from SGD and simple GD

In general, this is not a problem since the combined effect of many steps will eventually lead in the correct direction. Thus, in general, the gradient descent method used to train neural networks is still Stochastic Gradient Descent since it delivers huge time savings with respect to simple GD.

Mini batch stochastic gradient descent is middle option between stochastic gradient descent and simple gradient descent. At each iteration, a group of training examples called mini batch is used to update the weights and biases of the net. This in turn allows the average effect of more than one training example to be used to make a decision in which direction to move. This allows for a slightly longer time to train with respect to stochastic gradient descent, but a more informed decision when taking a step.

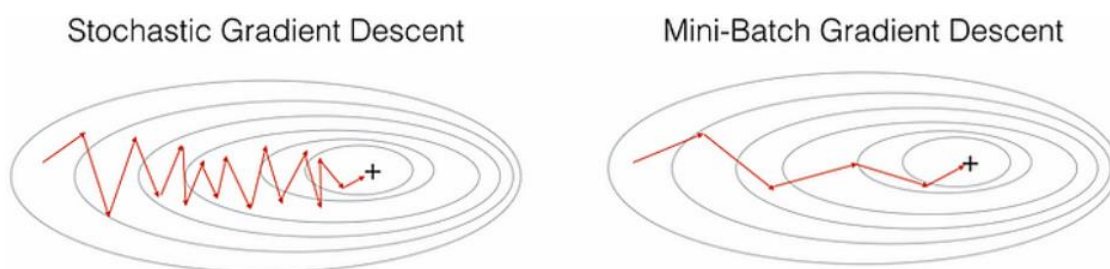


Figure B. 15 - Examples of trajectories taken from SGD and simple Mini-Batch GD

### B.4.3 Performance optimization: improvements on basic optimization algorithms

Over the years, many improvements have been suggested to increase the performance of the basic implementations of gradient descent proposed in the previous chapter. The main improvements presented here fall within two main categories: momentum and adaptive learning rate. From these two improvements, different algorithms implementing these two main ideas have spawned.

#### Momentum

Momentum was added to the simple gradient descent by borrowing the concept of momentum from physics. This is achieved by enforcing each step in the newly calculated direction to also have a contribution coming from the direction of the previous timestep. In practical terms this is achieved by calculating what is known as *velocity* and by adding a term which acts as *friction*.

Velocity  $v$  is simply computed by adding to the currently calculated step (learning rate \* gradient) a portion of the previously computed step weighted by the *friction* factor  $\gamma$ .

$$v_k = \gamma \cdot v_{k-1} + \alpha_k g_k$$

The descent step is then calculated as:

$$x_{k+1} = x_k - v_k$$

The addition of momentum helps to prevent large oscillations in stochastic gradient descent thanks to the fact that there is now also a contribution coming from the previous training example.

Note that if the friction parameter  $\gamma$  is set to zero, the algorithm returns to simple steepest gradient descent.

Other more complicated implementations of the same basic idea of momentum have also been developed throughout the years such as the Nesterov Accelerated Gradient Descent (NAG)

#### Adaptive learning rate

The second main addition to the basic steepest descent algorithm was to adapt the learning rate on the fly based on how often a given parameter actually influences the performance function. Some parameters may be “active” in the function minimisation process more often than others, by adapting the learning rate to how often a given parameter is used/updated allows to dynamically optimise the size of the steps based on the single current optimization step. This is a powerful tool for data sets containing data ranging from sparse to very dense.

Some of the main algorithms that use adaptive learning rates are Adagrad, RMSProp and Adam.

## Adaptive gradient descent (Adagrad)

The key idea behind Adagrad is to have an adaptive learning rate for each of the weights. It performs large updates for parameters which are less frequent, and small steps for parameters which are frequently *observed*.

Adagrad adjusts the learning rate in time, not by considering the overall elapsed time or the overall iterations, but it proposes a schedule based on the number of times a particular feature is seen.

Parameters associated with infrequent features could potentially only receive updates whenever these features occur. Thus, if a learning rate is decreased based on a total elapsed time, parameter control for the infrequent features might reach a very low learning rate before the infrequent feature can be observed properly. The advantage of counting the number of times a feature is observed is now obvious.

In Adagrad, instead of an actual count, an aggregate of the squares of the previously observed gradients is used. We define  $s_t$  as a variable to accumulate past gradient variance. In the following example the per-parameter update formulations are proposed which can then simply be put in vector form to account for all the remaining variables.

$$s_{k,i} = s_{k-1,i} + g_{k,i}^2$$

The weights can then be updated as:

$$x_{k+1,i} = x_{k,i} - \frac{\alpha}{\sqrt{s_{k,i}} + \varepsilon} g_{k,i}$$

Where  $\alpha$  is the initial learning rate,  $\varepsilon$  is an addition constant so that it's impossible to divide by zero,  $k$  is the time step and  $i$  is the variable index.

### Advantages

- Learning rate is different for each parameter independently
- No need for manual tuning of the learning rates
- Parameters which are infrequent won't risk having small learning rates prematurely

### Drawbacks

- Computationally expensive
- Since the learning rate is continuously decreasing, if the number of computations needed to find a solution is very large, training might get very slow.
- For deep learning problems, Adagrad might decrease learning rate too fast

## RMSProp

One of the problems with Adagrad is that the learning rates might become very small when the number of iterations is large. Also, for deep learning problems, Adagrad might reduce the learning rate far too quickly. The problem is that Adagrad accumulates the squares of the gradient  $g_k$  into a vector  $s_k = s_{k-1} + g_k^2$  which causes  $s_t$  to keep growing due to the lack of normalization. In turn the weights keep decreasing. To fix this problem RMSProp uses a leaky average to accumulate the squares of the gradient.

$$s_k = (1 - \gamma)g_k^2 + \gamma s_{k-1}$$
$$x_{k+1} = x_k - \frac{\alpha}{\sqrt{s_k} + \varepsilon} g_k$$

Where  $\gamma$  is  $> 0$ . This additional freedom allows to prevent the denominator beneath the learning rate  $\alpha$  to become too large and cause learning to become extremely slow.

## Adaptive Moment Estimation (Adam)

Adam can be considered as a combination of RMSProp and stochastic gradient descent with momentum. Adam uses leaky averaging to obtain an estimate of both the momentum and also the second moment of the gradient. In addition to storing the decaying average of past gradients, it also stores a decaying average of past gradients used for momentum. This is done individually for each variable, allowing to have individual learning rates and individual momentum changes.

$$v_k = (1 - \beta_1)g_k + \beta_1 v_{k-1}$$
$$s_k = (1 - \beta_2)g_k^2 + \beta_2 s_{k-1}$$

The new step becomes:

$$x_{k+1} = x_k - \frac{\alpha}{\sqrt{s_k} + \varepsilon} v_k$$

Where  $\beta_1$  and  $\beta_2$  are non-negative weighting parameters often set to  $\beta_1 = 0.9$  and  $\beta_2 = 0.999$ .

Thanks to the combination of the best different aspects from different optimization algorithms, Adam presents itself as one of the most robust and effective optimization algorithms used in deep learning.

### Advantages

- Fast convergence to minima
- Robust

### Drawbacks

- Possible failure to converge when second moment estimate  $s_t$  blows up
- Computationally costly



## B.5 Long Short-Term Memory (LSTM) neural networks

LSTM neural networks are a particular kind of neural networks belonging to the class of recurrent neural networks. These kinds of networks are particularly useful at processing data linked to a timeseries such as video, speech data or any other sequential data. The peculiarity of these kind of networks with respect to traditional neural networks is their ability to store or discard past information that has been fed to the network up until a certain moment. This can be seen as a sort of memory of past events used to better process current event which is similar to how a human would take decisions. When we think or when we process information, we do not only rely on the current information being fed to our brains, but we use past information, both long and short term, in order to take better decisions.

From this point of view, LSTM networks can be seen as networks which have internal loops within them to allow a given set of information of past events to persist.

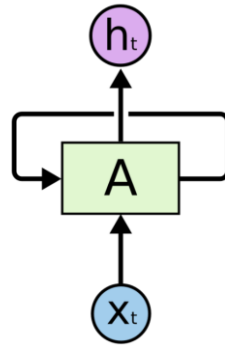


Figure B. 16 – Basic idea behind recurring neural networks and LSTM networks [82].

In the above image an input  $x_t$  is fed to a cell  $A$  which then produces an output  $h_t$ , with the addition of an internal loop to feed information back to the cell. This concept may be hard to grasp, so it can be easier to imagine the looping network as a succession of multiple copies of the same network, each copy dealing with the inputs at a given time step and then passing on information to the next network used for the next time step.

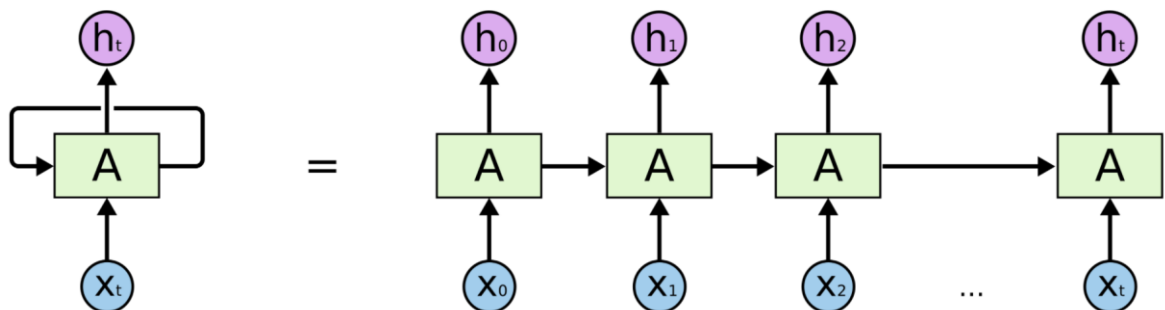


Figure B. 17 - An unrolled LSTM network [82].

As can be seen in the above image, an LSTM network, or any other recurrent network for that matter, naturally “unrolls” or “unfolds” to the length of the input sequence fed to the network.

The peculiarity of LSTM networks with respect to standard recurrent neural networks lies within the repeating cell.

Standard recurring neural networks usually possess very simple repeating modules, such as modules only containing a single tanh network layer.

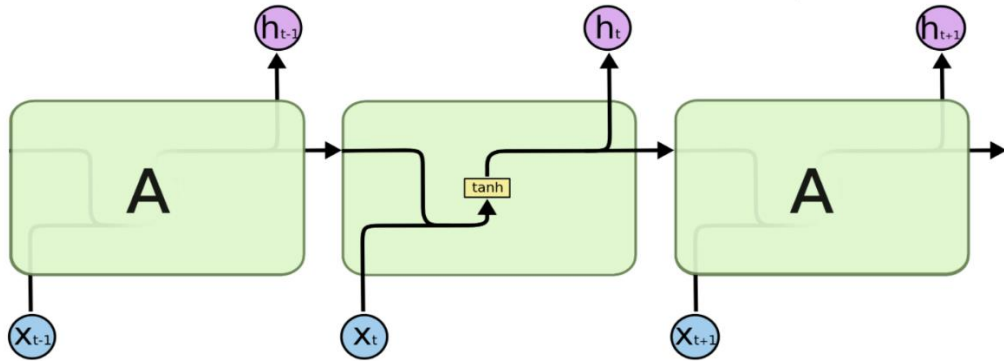


Figure B. 18 - Repeating module or cell for a RNN [82].

This simplicity of the repeating module or cell is one of the reasons why RNNs suffer from long-term dependency learning difficulties where they struggle to connect relevant information from previous time steps to current events if the time gap to the past information is too large.

LSTM networks instead were conceived with this problem in mind, and what results is a much more complex repeating module than what is found in standard RNNs.

LSTM networks were introduced by Hochreiter and Schmidhuber in 1997 [83] and through the years have been refined by many others to avoid the long-term dependency problem of standard RNNs, making them extremely well suited to tackle problems related to long time series, especially when relevant information needs to be stored and kept relevant for long periods of time.

LSTM networks achieve this behavior thanks to the specially designed repeating module composed of four neural network layers and different pointwise operations at different points in the information flow.

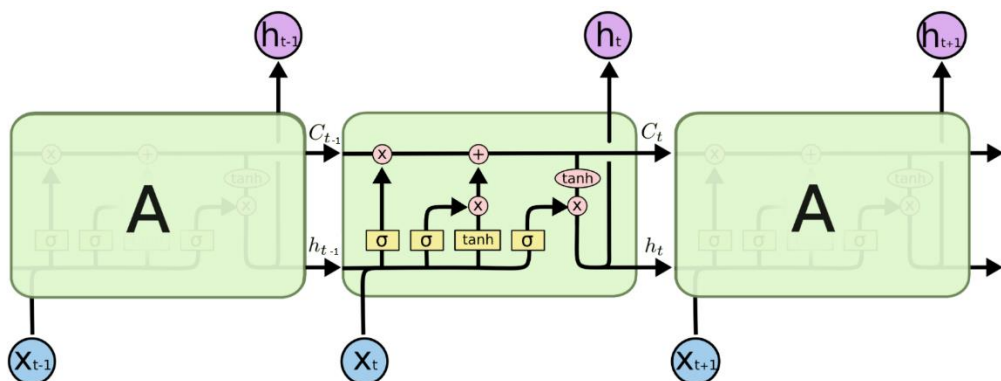


Figure B. 19 - Repeating module of a LSTM neural network [82].

Where in the above diagram we have:

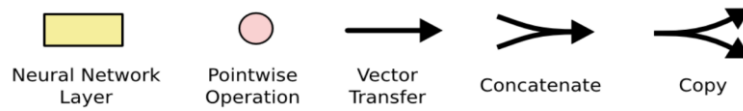


Figure B. 20 - List of symbols for LSTM module [82].

The information that is passed from one cell to the next (so from one time step to the next) is both the current output  $h_t$  and what is known as the *cell state*  $C_t$ .

The cell state can be considered as the mechanism used to keep memory of past information, or past outputs/predictions and let it flow to the next cells, carefully regulated by structures called *gates*.

Gates allow to select which information and how much of it to let through, so they act as a sort of memory management where they decide which information is relevant and thus should be kept and which information can be discarded.

As can be seen in figure B.19, the processed and gated information in a concatenation of the past output and the current input. This information is then passed simultaneously to multiple gates, each performing a specific operation in order to produce the next output and next cell state.

### B.5.1 Forget gate layer

The forget gate is responsible to interpret the current input and past output information in order to decide what information needs to be discarded from the cell state.

This operation is achieved thanks to a sigmoid layer which produces a value between 1 and 0 for each variable in the merged input and past output vector. This vector then multiplies the past cell state, effectively gating each value. If the sigmoid layer outputs a value equal to 1 for a given position in the vector, the subsequent multiplication will translate in to letting all the information through in that position, while if a 0 is outputted from the sigmoid layer, the vector wise multiplication will translate in to forgetting all the information in the corresponding position of the cell state vector.

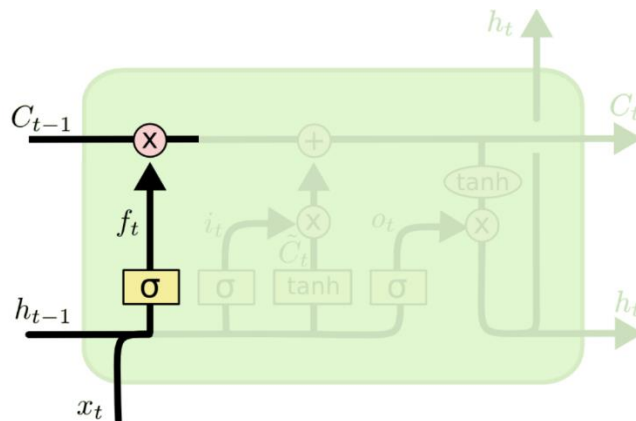


Figure B. 21 – Forget gate and operation on old cell state [82].

### B.5.2 Ignore gate layer or input gate layer

The ignore gate layer is used to gate the newly created or predicted cell state so to ignore part of this new vector. First, the new vector  $\tilde{C}_t$  of candidate values for the cell state is created through the use of a tanh layer. In parallel, a sigmoid layer is used to produce a vector of values spanning between 0 and 1 which are then used in a vector multiplication operation to decide how much of each of the new candidates in  $\tilde{C}_t$  to let through. This operation of using a vector from a sigmoid layer to gate another vector is the basis of the gating operations used in LSTM neural networks.

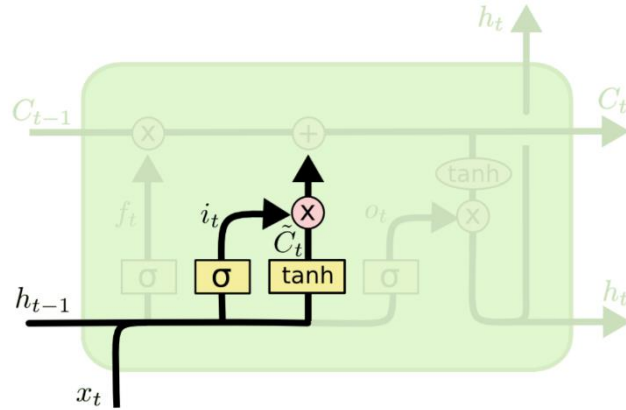


Figure B. 22 - Ignore gate operation [82].

### B.5.3 Updating the cell state

Finally, after forgetting the unwanted information from the old cell state and creating new gated candidates, the cell state can be updated. This is simply achieved through simple vector addition of the old, gated cell state, rid of what was deemed forgettable, and the new gated candidates.

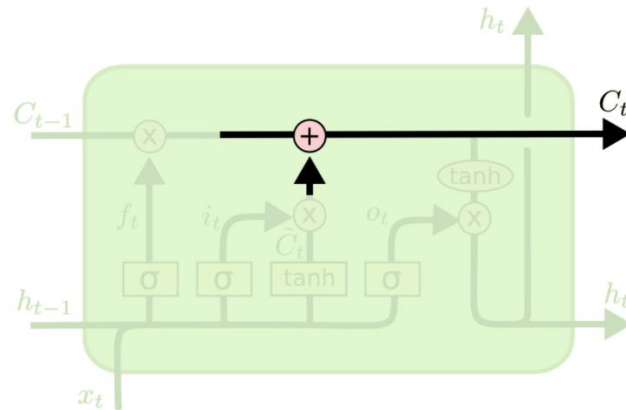


Figure B. 23 - Updating the cell state [82].

#### B.5.4 Creating the output of the cell

With the cell state created it is now possible to create the output of the cell, which is the actual output that our neural network is trained to produce. The output is based on the cell state, but it will be a filtered version of it.

Again, similarly to what was done in the forget and ignore gates, a sigmoid layer is used to produce a vector of values ranging from 0 to 1 which will be used to decide which and how much of each value in the cell state is going to be used to produce the output  $h_t$ .

The cell state is then passed through a tanh pointwise operation in order to compress all the values within the cell state between -1 and 1. The resulting squashed cell state is then multiplied by the vector coming from the sigmoid layer. The resulting filtered or gated values represent the output  $h_t$ .

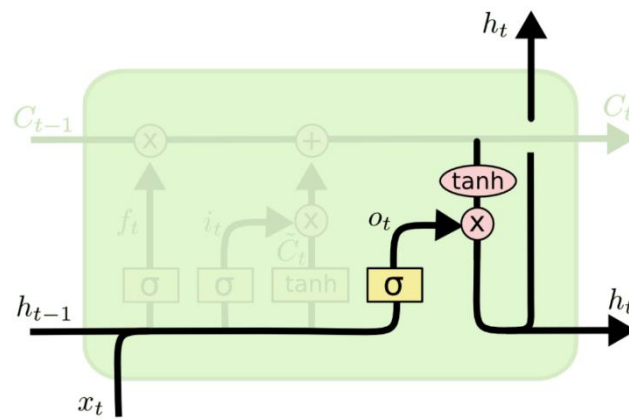


Figure B. 24 - Creating the output vector  $h_t$  [82].



## Bibliography

- [1] IRENA, "Offshore renewables: An action agenda for deployment," International Renewable Energy Agency, Abu Dhabi, 2021.
- [2] IRENA, "Renewable capacity statistics," International Renewable Energy Agency , Abu Dhabi, 2021.
- [3] IRENA, "Innovation outlook: Ocean energy technologies," International Renewable Energy Agency, Abu Dhabi, 2020.
- [4] U. Nations, "The Ocean Conference," in *Factsheet: People and Oceans*, New York, 2017.
- [5] IRENA, "irena.org," 2020. [Online]. Available: <https://www.irena.org/Statistics/View-Data-by-Topic/Finance-and-Investment/Investment-Trends>.
- [6] J. A. S. C. I. M. d. A. I. K. Iraide Lòpez, "Review of wave energy technologies and the necessary power-equipment," in *Renewable and Sustainable Energy Reviews*, volume 27, 2013, pp. 413-437.
- [7] A. R. P. M. N. S. B. Drew, "A review of wave energy converter technology.," *Institution of Mechanical Engineers, Part A: Journal of Power and Energy*, pp. 887-902, 2009.
- [8] P. B. Balazs Czech, "Wave energy converter concepts: design challenges and classification," *IEEE Industrial electronics magazine*, pp. 4-16, 2012.
- [9] C. S.-W. Kester Gunn, "Quantifying the global wave power resource," *Renewable Energy*, vol. 44, pp. 296-304, 2012.
- [10] G. Komen, "Guide to Wave Analysis and Forecasting," *Journal of fluid mechanics*, vol. 234, no. 1, 1992.
- [11] B. L. Méhauté, *An Introduction to Hydrodynamics & Water Waves*, Springer Science, 1976.
- [12] A. F. Molland, *The Maritime Engineering Reference Book*, Butterworth Heinemann, 2008.
- [13] L. M. Willard J. Pierson Jr, "A proposed spectral form for fully developed wind seas based on the similarity theory of S. A. Kitaigorodskii," *Journal of Geophysical Research* , vol. 69, no. 24, pp. 5181-5190, 1964.
- [14] e. a. Klaus Hasselmann, "Measurement of wind-wave growth and swell decay during the joint North Sea wave project (JONSWAP)," 1973.
- [15] J. P. K. Arthur Pecher, *Handbook of Ocean Wave Energy*, Springer Open, 2017.

- [16] S. L. M. L. C. D. D. S. D. Z. N. L. S. Z. Zhanhui Qi, "Research on the Algorithm Model for Measuring Ocean Waves Based on Satellite GPS Signals in China," *Sensors - High-Precision GNSS in Remote Sensing Applications*, 2019.
- [17] E. Pasta, *Model-Free Control System Architecture for the ISWEC*, 2020.
- [18] IEC/TS 62600-101: Marine energy. Wave, tidal and other water current converters. Wave energy resource assessment and characterization, 2015.
- [19] J. V. R. Alexis Mérigaud, "Free-Surface Time-Series Generation for Wave Energy Applications," *IEEE JOURNAL OF OCEAN ENGINEERING*, vol. 43, no. 1, pp. 19-35, 2018.
- [20] C. Leavitt, "Mechanism for Utilizing Wave-Power". U.S.A Patent US321229A, 30 June 1885.
- [21] "The Liquid Grid," [Online]. Available: <https://theliquidgrid.com/marine-clean-technology/wave-energy-converters/>.
- [22] R. Z. X. W. Elie Al Shami, "Point Absorber Wave Energy Harvesters: A Review of Recent Developements," *energies*, vol. 12, no. 1, 2018.
- [23] J. A. O. S. R. W. Valeria Castellucci, "Algorithm for the calculation of the translator position in permanent magnet linear generators," *Journal of Renewable and Sustainable Energy*, no. 6, 2014.
- [24] E. Anderlini, *Control of wave energy converters using machine learning strategies*, 2017.
- [25] M. M. Nick J. Baker, "Direct drive wave enrgy converters," *Revue des Energies Renouvelables: Power Engineering*, vol. 36, pp. 1-7, 2001.
- [26] T. V. G. P. F. C. G. B. G. M. Edoardo Pasta, "Collaborative strategy for model-free control of arrays of wave energy converters: A genetic algorithm approach," in *OCEANS 2021:San Diego - Porto*, 2021.
- [27] P. D. E. P. G. B. P. N. G. M. A. P. S. B. Luca Parrinello, "An adaptive and energy-maximizing control optimization of wave energy converters using an extremum-seeking approach," *Physics of Fluids*, vol. 32, no. 11, 2020.
- [28] Y. P.-S. a. J. V. R. N. Faedo, "Finite-order hydrodynamic model determination for wave energy applications using moment matching," *Ocean Engineering*, vol. 163, pp. 251-263, 2018.
- [29] U. Korde, "Efficient primary energy conversion in irregular waves," *Ocean Engineering*, vol. 26, no. 7, pp. 625-651, 1999.
- [30] J. Falnes, *Ocean Waves and Oscillating Systems: Linear Interactions Including Wave Energy Extraction*, Cambridge University Press, 2002.
- [31] J. R. M. T. N. J. C. S. H. Salter, "Power conversion mechnisms for wave energy," *Journal of Engineering for the Maritime Environment*, vol. 216, 2002.



- [32] J. F. K. Budal, "A resonant point absorber of ocean-wave power," *Nature*, pp. 478-479, 1975.
- [33] J. F. T. M. Jørgen Hals, "A Comparison of Selected Strategies for Adaptive Control of Wave Energy Converters," *Journal of offshore mechanics and Arctic engineering*, vol. 133, no. 3, 2011.
- [34] G. B. F. F. John V. Ringwood, "Energy-Maximising Control of Wave-Energy Converters," *IEEE CONTROL SYSTEMS MAGAZINE*, 2014.
- [35] K. F. J. Budal, "The Norwegian wave-power buoy project," in *Second international Symposium on Wave energy Utilisation*, 1992.
- [36] K. F. J. H. T. I. L. C. a. O. T. Budal, "Model Experiment with a Phase Controlled Point Absorber," in *Proceedings of the Second International Symposium on Wave and Tidal Energy*, 1981.
- [37] G. D. A. C. A. Babarit, "Comparison of latching control strategies for a heaving wave energy device in random sea," *Applied ocean research*, vol. 26, no. 5, pp. 227-238, 2004.
- [38] A. F. O. Falcão, "Phase Control Through Load Control of Oscillating Body Wave Energy COnverters With Hydraulic PTO System," *Ocean Engineering*, vol. 35, no. 3, pp. 358-366, 2008.
- [39] J. H. R. G. T. M. L. G. A. O. F. M.F.P. Lopes, "Experimental and numerical investigation of non-predictive phase-control strategies for a point-absorbing wave energy converter," *Ocean Engineering*, vol. 36, no. 5, pp. 386-402, 2009.
- [40] M. W. T. Folley, "The control of wave energy converters using active bipolar damping," *Journal of engineering for the maritime environment*, vol. 223, no. 4, pp. 479-487, 2009.
- [41] D. F. E. B. M. A. E. Anderlini, "Reactive control of a wave energy converter using artificial neural networks," *International Journal of Marine Energy*, 2017.
- [42] J. V. R. Francesco Fusco, "Short-Term Wave Forecasting for Real-Time Control of Wave Energy Converters," *IEEE transactions on sustainable energy*, vol. 1, no. 2, pp. 99-106, 2010.
- [43] P. Gieske, *Model predictive control of a wave energy converter: Archimedes Wave Swing*, 2007.
- [44] M. E. M. O. S. T. K. B. Markus Richter, "Power optimisation of a point absorber wave energy converter by means of linear model predictive control," *IET Renewable Power Generation* , vol. 8, no. 2, pp. 203-215, 2011.
- [45] J. F. T. M. J. Hals, "Constrained Optimal Control of a Heaving Buoy Wave-Energy Converter," *Journal of Offshore Mechanics and Arctic Engineering*, vol. 133, no. 1, 2011.

- [46] H. N. N. G. S. Y. C. Paolino Tona, "An Efficiency-Aware Model Predictive Control Strategy for a Heaving Buoy Wave Energy Converter," in *11th European Wave and Tidal Conference*, Nantes, 2015.
- [47] A. M. J. V. R. Yeraí Peña-Sanchez, "Short-Term Forecasting of Sea Surface Elevation for Wave Energy Applications: The Autoregressive Model Revised," *IEEE Journal of Oceanic Engineering*, vol. 45, no. 2, pp. 462-471, 2020.
- [48] M. J. M. P. B. J. S. d. C. Duarte Valério, "Identification and control of the AWS using neural network models," *Applied Ocean Research*, vol. 30, no. 3, pp. 178-188, 2008.
- [49] J. H. Holland, *Adaptation in Natural and Artificial Systems*, 1975.
- [50] A. S. A.E. Eiben, "On evolutionary exploration and exploitation," *Fundamenta Informaticae*, vol. 35, pp. 1-16, 1998.
- [51] M. Z. Huayang Xie, "Tuning Selection Pressure in Tournament Selection," 2009.
- [52] D. P. K. Reuven Y. Rubinstein, *Simulation and the Monte Carlo Method*, III ed., John Wiley & Sons, 2016.
- [53] F. C. P. B. L. P. G. M. E. Pasta, "A Model-Free Control Strategy Based on Artificial Neural Networks for PeWEC," in *14th European Wave and Tidal Energy Conference (EWTEC)*, Plymouth, UK, 2021.
- [54] H. R. T. M. M. S. Shahryar Rahnamayan, "Quasi oppositional differential evolution," in *IEEE Congress on Evolutionary Computation*, Singapore, 2007.
- [55] M. Clerc, "clerc.maurice.free.fr," 24th December 2008. [Online]. Available: <http://clerc.maurice.free.fr/psso/Initialisations.pdf>.
- [56] S. S. A. E. Volker Nannen, "Costs and Benefits of Tuning Parameters of Evolutionary Algorithms," in *Parallel Problem Solving from Nature - PPSN X*, Springer-Verlag, 2008, pp. 528-538.
- [57] J. F. A. H. K.L. Mills, "Determining Realative Importance and Effective Settings for Genetic Algorithm Control Parameters," *Evolutionary computation*, vol. 23, no. 2, pp. 309-342, 2015.
- [58] J. H. Holland, *Adaptation in Natural and Artificial Systems*, 1975.
- [59] A. Brindle, *Genetic algorithms for function optimization*, Ph.D dissertation, Department of Computing Science, University of Alberta, 1981.
- [60] P. j. F. Wael Khatib, "The Stud GA: A Mini Revolution?," in *Lecture Notes in Computer Science*, 1998.
- [61] D. E. G. Brad L. Miller, "Genetic Algorithms, Tournament Selection, and the Effect of Noise," *Complex Systems*, vol. 9, no. 3, pp. 193-212, 1995.

- [62] J. S. A.E. Eiben, Introduction to Evolutionary Computing, Second Edition ed., Springer, 2015.
- [63] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.
- [64] S. S. A.E. Eiben, "Parameter tuning for configuring and analysing evolutionary algorithms," *Swarm and Evolutionary Computation*, vol. 1, pp. 19-31, 2011.
- [65] K. D. Jong, "Parameter Setting in EAs: a 30 Year Perspective," in *Parameter Setting in Evolutionary Algorithms*, Springer, 2007, pp. 1-18.
- [66] M. J. A.E. Eiben, "A critical note on experimental research methodology in EC," in *Congress on Evolutionary Computation*, Piascataway, 2002.
- [67] M. H. Bernd Freisleben, "Optimization of Genetic Algorithms by Genetic Algorithms," in *Artificial Neural Nets and Genetic Algorithms*, 1993, pp. 392-399.
- [68] J. J.Grefenstette, "Optimization of Control Parameters for Genetic Algorithms," in *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS*, 1986.
- [69] A. E. S.K. Smith, "Comparing Parameter Tuning Methods for Evolutionary Algorithms," in *IEEE Congress on Evolutionary Computation*, Trondheim, Norway, 2009.
- [70] W. P. Warren S. McCulloch, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.
- [71] "intel newsroom," 21 May 2018. [Online]. Available: <https://newsroom.intel.com/news/many-ways-define-artificial-intelligence/#gs.p31uvw>.
- [72] M. H. a. A. E. E. Giorgos Karafotias, "Parameter Control in Evolutionary Algorithms: Trends and Challenges," *IEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, vol. 19, no. 2, 2015.
- [73] J. M. C. O. a. F. M. Brian Mc Ginley, "Maintaining Healthy Population Diversity Using Adaptive Crossover, Mutation, and Selection," *IEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, vol. 15, no. 5, 2011.
- [74] R. K. \*. P. C. Sandip Aine, "Adaptive parameter control of evolutionary algorithms to improve quality-time trade-off," *Applied soft computing*, pp. 527-540, 2009.
- [75] X. L. A. K. Q. Borhan Kazimipour, "A Review of Population Initialization Techniques for Evolutionary Algorithms," *IEEE Congress on Evolutionary Computation*, pp. 2585-2592, 2014.
- [76] T. Szandala, "Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks," *Studes in Computational Intelligenece; Bio-inspired Neurocomputing*, no. 903, pp. 203-224, 2020.

- [77] G. E. H. V.Nair, "Rectified linear units improve restricted boltzmann machines," in *International Conference on Machine Learning*, 2010.
- [78] M. Nielsen, *Neural Networks and Deep Learning*.
- [79] J. Brownlee, "Machine Learning Mastery," 11 January 2019. [Online]. Available: <https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/>.
- [80] Y. B. Xavier Glorot, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research*, vol. vol. 9, pp. 249-256, 2010.
- [81] L. B. G. B. O. K. M. Y. Lecun, "Efficient Backprop," in *Neural Networks: Tricks of the trade*, 1998, pp. 9-50.
- [82] C. Olah, "colah's blog," 27 August 2015. [Online]. Available: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- [83] J. S. Sepp Hochreiter, "LONG SHORT-TERM MEMORY," *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

*Grazie Yummy*