BIDA407 - Machine Learning Fundamentals

Final Project

Modelling the Performance of a Heave Constrained Point Absorber

prepared by

Anthony Truelove MASc, P.Eng.

PRIMED - Research Engineer

prepared for

Dr. Hossen Teimoorinia

AI Specialist - Herzberg Astronomy and Astrophysics Research Centre

April 6, 2024

Drafted using LaTeX

Abstract

In this work, perturbation theory is applied to the modelling and simulation of a heave constrained point absorber wave energy converter (WEC). To that end, a problem with related but reduced dynamics is first set up and solved in order to form the exact solution part of the complex dynamics.

The complex dynamics of the WEC are then simulated over a large number (∼35,000) of sea state and WEC design combinations in order to generate a data set of the "true" WEC behaviour. The complex dynamics are then compared to the reduced dynamics in order to determine an appropriate form for the perturbation. In this case, a separable perturbation is found to be most appropriate.

A dense neural network (DNN[1]) is then chosen to serve as a perturbation machine, and optimal hyperparameters for the DNN are evolved using a differential evolution algorithm.[2] Following the hyperparameter evolution, the optimal DNN architecture is then trained to act as the perturbation machine. What results is a perturbation machine with decent performance; that is, the expected power output predictions made using the perturbation machine fall within ±35% of the corresponding "true" values, 19 times out of 20.

With the perturbation machine thus trained, two example applications are presented; namely

1. WEC design optimization.

2. WEC performance mapping.

In both cases, the results that emerge are encouraging.

1. For the case of a sea state with 3 m significant wave height H_s and 10 s spectral peak period T_p, an optimal design for a heave constrained point absorber is sought (with the design variables being float diameter D and power takeoff damping b). Using a simulated annealing approach[3], an optimal design of $D = 81.02$ m and $b = 72.29 \times 10^6$ N.s/m is found, for which expected power output is a little less than 2 MW. In the course of optimization, 13,920 candidate designs were assessed in a little less than 30 minutes.

[1]Specifically a `tensorflow.keras.Sequential`.
[2]Specifically `scipy.optimize.differential_evolution`.
[3]Specifically `scipy.optimize.dual_annealing`.

2. For the optimal design previously found, a WEC performance map is generated over the sea state space $H_s \in [0, 8]$ m and $T_p \in [5, 16]$ s. In the course of generating the performance map, expected power output for the WEC was predicted for 4,096 different sea states in a little less than nine minutes.

Contents

List of Figures

1 Motivation and Intent

Wave energy converter (WEC) technology is an emerging field within the domain of renewable energy, especially for regions with a high wave energy potential such as Vancouver Island. However, compared to other renewable energy technologies such as solar and wind, WEC dynamics are highly complex and expensive to model and simulate, and so methods of reducing this computational expense (without sacrificing an excessive amount of detail) are desirable. This is precisely the intent of this work

To that end, the approach taken in this work is inspired by the historical successes of the so-called perturbation methods. Essentially, this is the technique of obtaining a solution (albeit approximate) to a complex dynamics problem by first obtaining an exact solution to a related, reduced problem, and then seeking an appropriate correction (or *perturbation*) of this exact solution to better approximate the true, complex dynamics.[1] The canonical example of the power of these methods are their contribution to the discovery of the planet Neptune [1]:

Perturbation theory has been investigated by prominent mathematicians (e.g., Laplace, Poisson, Gauss), and as such the computations can be performed with very high accuracy. For example, the existence of the planet Neptune was postulated in 1848 by mathematicians J.C. Adams and U. le Verrier, with this postulation being based on the deviations (i.e. perturbations) in the observed motion of the planet Uranus. The existence of Neptune was then confirmed by way of observation by the astronomer J.G. Galle (who received coordinates from le Verrier). This represented a triumph of perturbation theory.

The approach taken in this work can be itemized as follows

1. The related, reduced problem is set up and solved.
2. The corresponding complex dynamics are simulated using industry standard software (namely Proteus DS).
3. A perturbation form is chosen, and a dense neural network is trained to act as a perturbation machine.
4. Some example applications using the trained perturbation machine are presented.

[1] This could also be a sequence of corrections, rather than simply one.

2 Reduced Problem

2.1 Definition

While there are several classes of WEC technology currently under development, for example [2, 3]

1. point absorbers (like the AquaBuoy WEC)
2. attenuators (like the Pelamis WEC)
3. terminators (like the Oyster WEC)
4. oscillating water columns (like the Mutriku WEC)
5. overtopping devices (like the Wave Dragon WEC)

the point absorber class appears to be emerging as a particularly promising candidate. As such, this work will focus exclusively on this class of tech.

That said, consider a point absorber WEC which is a cylindrical float riding a bottom-fixed piling as illustrated in Figure 2.1.[1]

[1] Think a WEC design which is intended to be installed at the waterline of existing offshore infrastructure, like wind turbines or oil rigs.

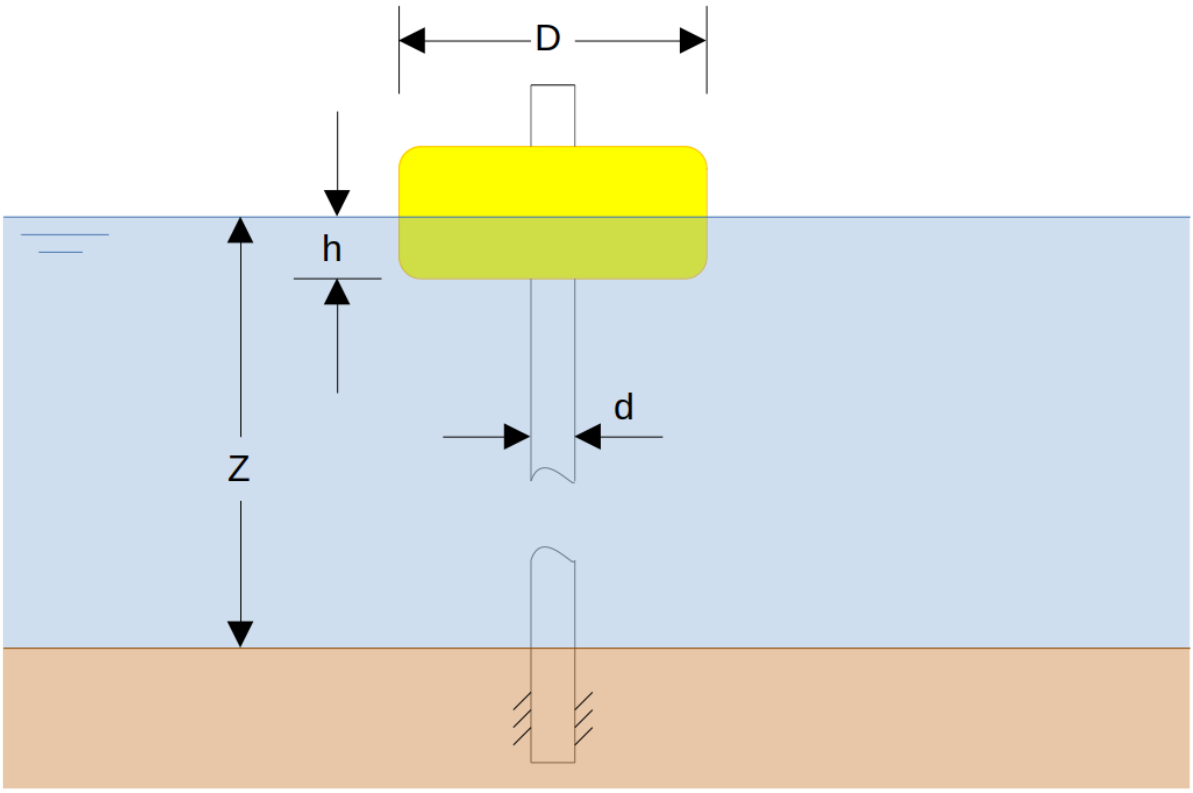

Figure 2.1: A point absorber as a cylindrical float riding a bottom-fixed piling.

That is, a float of diameter $D > d$ and with resting draft $h > 0$ is riding a bottom-fixed piling of diameter $d > 0$ in a sea of depth $Z > 0$. The basic operating principle here is that wave action causes the float to oscillate, and the relative motion between the float and piling can then drive some power takeoff device.

2.2 Simplifying Assumptions

To begin assembling the reduced problem, a number of simplifying assumptions are made; namely

1. Inviscid fluid (i.e., no drag).
2. Forces due to wave incidence, wave diffraction, and wave radiation are all negligible compared to buoyancy forces.
3. Added mass effects can be ignored.
4. Fluid memory effects can be ignored.
5. Power takeoff (PTO) dynamics are linear.

6. The WEC is heave constrained (i.e., it is a single degree-of-freedom system).

As such, the only forces acting on the WEC in this reduced case are weight, buoyancy, and the reaction from the PTO. Specifically, the WEC can only move in heave (i.e. up and down).

2.3 Constructing the Differential Equation of Motion

Consider the free-body diagram illustrated in Figure 2.2.

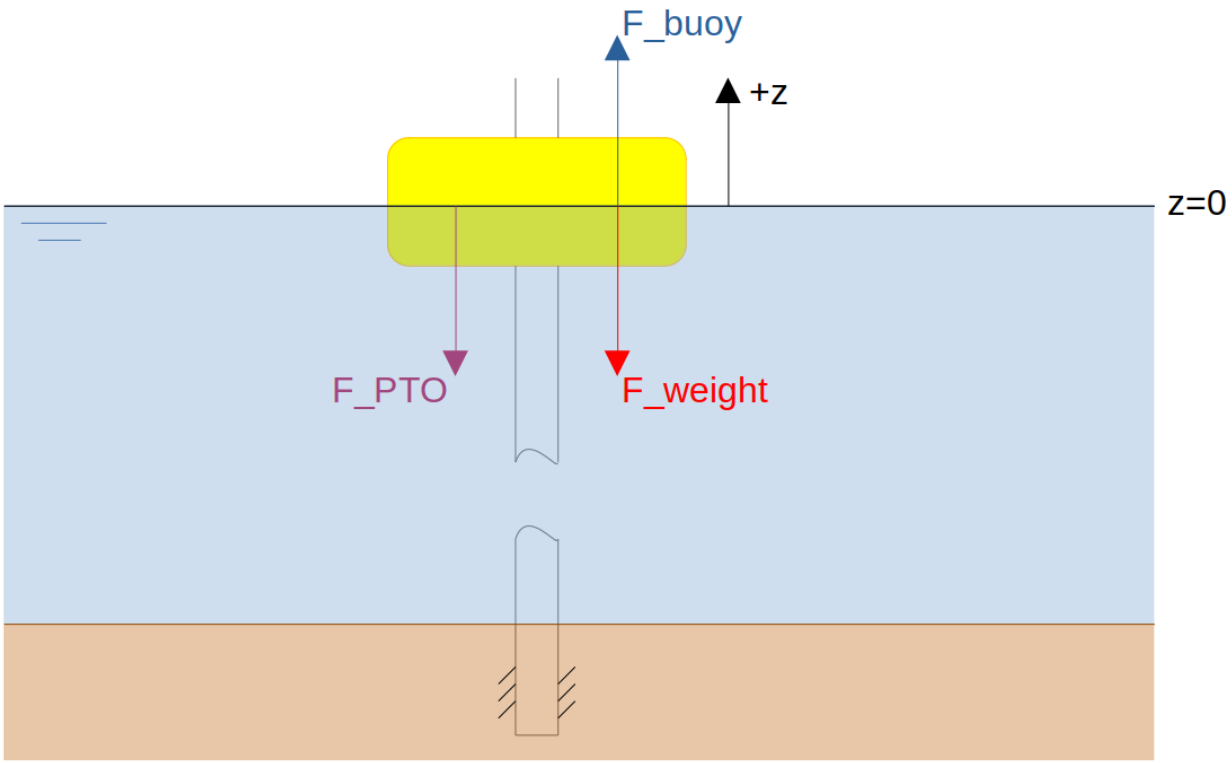

Figure 2.2: Free-body diagram for the reduced WEC dynamics.

That is, define positive float motion ($+z$) as upward, and define the origin ($z = 0$) as the mean sea level. Therefore, for positive float motion, the driving buoyancy force has positive orientation and the PTO reaction has negative orientation. Of course, weight always has negative orientation in this case.

As per Newton's Second Law (Newton II), the sum of external forces acting on a body is equal to the time rate-of-change of the body's momentum. So, in this case, application of Newton II yields

$$F_{\text{buoy}} + F_{\text{weight}} + F_{\text{PTO}} = \frac{d}{dt}\{m\dot{z}\} \tag{2.1}$$

Assuming a constant float mass $m > 0$, it then follows that

$$F_{\text{buoy}} + F_{\text{weight}} + F_{\text{PTO}} = m\ddot{z} \tag{2.2}$$

Now, the force due to weight is given simply by the product of float mass m and acceleration due to gravity g; namely

$$F_{\text{weight}} = -mg \tag{2.3}$$

As for the reaction from the PTO, if one assumes constant stiffness and damping values $k \geq 0$ and $b \geq 0$ respectively, then it follows that

$$F_{\text{PTO}} = -kz - b\dot{z} \tag{2.4}$$

Finally, the driving buoyancy force can be obtained by way of Archimedes' Principle, which states that a floating body experiences an upward buoyancy force equal to the weight of the fluid displaced. Therefore, for a float with a resting draft of h in a fluid of density $\rho > 0$, it follows in this case that

$$F_{\text{buoy}} = \frac{\pi\rho g}{4}(D^2 - d^2)(h + \overline{\eta} - z) \quad \text{for} \quad h + \overline{\eta} - z \geq 0 \tag{2.5}$$

where $\overline{\eta}$ is the deviation in sea surface elevation, from $z = 0$, averaged over the waterplane area of the float. But then, the fact that the float has a resting draft of h implies (again, by Archimedes) that

$$m = \frac{\pi\rho h}{4}(D^2 - d^2) \tag{2.6}$$

as this ensures that (2.3) and (2.5) are equal and opposite when the system is at rest (i.e., $z = \dot{z} = \ddot{z} = 0$ and $\overline{\eta} = 0$), as one might logically expect. From this, it follows that

$$F_{\text{weight}} = -\frac{\pi\rho g h}{4}(D^2 - d^2) \tag{2.7}$$

Finally, substituting (2.4) - (2.7) into (2.2) and re-arranging yields (after simplifying) the following differential equation of motion

$$m\ddot{z} + b\dot{z} + (k + k_D)\, z = k_D\overline{\eta} \quad \text{for} \quad h + \overline{\eta} - z \geq 0 \tag{2.8}$$

with

$$k_D = \frac{\pi\rho g}{4}(D^2 - d^2) \tag{2.9}$$

being the buoyancy stiffness. Note that (2.8) is a linear, second-order, ordinary differential equation with constant coefficients (and as such, can be solved exactly using classical methods).

2.4 Solving the Differential Equation of Motion

2.4.1 General Solution to the Homogeneous Equation

First, a general solution to the homogeneous equation is sought. That is, determine $z(t)$ such that

$$m\ddot{z} + b\dot{z} + (k + k_D)\, z = 0 \tag{2.10}$$

The classical approach here is by way of the trial solution

$$z(t) = C \exp[rt] \tag{2.11}$$

Substitution of (2.11) into (2.10) then yields the second-order characteristic equation (in r), which in turn then leads to the general solution

$$z(t) = C_1 \exp\left[\frac{1}{2m}\left(-b + \sqrt{b^2 - 4(k+k_D)m}\right) t\right] + C_2 \exp\left[\frac{1}{2m}\left(-b - \sqrt{b^2 - 4(k+k_D)m}\right) t\right] \tag{2.12}$$

where C_1 and C_2 are arbitrary constants (see `Maple/pdf/ODE_general_solution.pdf`). That said, assuming zero initial conditions (i.e., $z(0) = \dot{z}(0) = 0$) leads to $C_1 = C_2 = 0$. This assumption is made in this work, and so the general solution is here omitted.

2.4.2 Expressing Average Sea Surface Elevation

Since the differential equation of motion has constant coefficients, it follows that a particular solution to the nonhomogeneous equation can be obtained by way of the method of undetermined coefficients. However, in order to do so, an appropriate expression for $\overline{\eta}$ must first be obtained.

As per [4], the deviation in the sea surface elevation (from mean sea level) over an area of sea can be expressed as a cosine series (in polar form) as follows

$$\eta(r, \theta, t) = \sum_{n=1}^{\infty} a_n \cos\left(\frac{2\pi n t}{T} - k_n r \cos(\psi_n - \theta) - \phi_n\right) \tag{2.13}$$

where $a_n \in \mathbb{R}$ is component amplitude, $T > 0$ is some fundamental period, $k_n > 0$ is component wave number, $\psi_n \in [-\pi, \pi]$ is component direction, and $\phi_n \in [-\pi, \pi]$ is component phase (random, with uniform distribution). Applying the trig identity

$$\cos(\alpha - \beta) = \cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta)$$

then expands (2.13) as follows

$$\eta(r, \theta, t) = \sum_{n=1}^{\infty} a_n \left[\cos\left(\frac{2\pi n t}{T} - \phi_n\right)\cos(k_n r \cos(\psi_n - \theta)) + \sin\left(\frac{2\pi n t}{T} - \phi_n\right)\sin(k_n r \cos(\psi_n - \theta))\right] \tag{2.14}$$

Now, $\overline{\eta}$ was introduced as the deviation in the sea surface elevation (from mean sea level) averaged over the waterplane area of the float. That is

$$\overline{\eta}(t) = \frac{1}{A_{\text{water}}} \iint_{A_{\text{water}}} \eta(r,\theta,t) dA \tag{2.15}$$

where A_{water} is the waterplane area of the float. For the case of the cylindrical float considered in this work, (2.15) can be expressed as

$$\overline{\eta}(t) = \frac{4}{\pi(D^2 - d^2)} \int_0^{2\pi} \int_{\frac{d}{2}}^{\frac{D}{2}} \eta(r,\theta,t) r dr d\theta \tag{2.16}$$

Next, approximation using a *finite* ($N < \infty$) cosine series allows one to express (2.16), by way of substitution of (2.14), as

$$\begin{aligned}
\overline{\eta}(t) \cong & \\
& \frac{4}{\pi(D^2 - d^2)} \sum_{n=1}^{N} \left[a_n \cos\left(\frac{2\pi n t}{T} - \phi_n \right) \int_0^{2\pi} \int_{\frac{d}{2}}^{\frac{D}{2}} r \cos(k_n r \cos(\psi_n - \theta)) dr d\theta \right] + \\
& \frac{4}{\pi(D^2 - d^2)} \sum_{n=1}^{N} \left[a_n \sin\left(\frac{2\pi n t}{T} - \phi_n \right) \int_0^{2\pi} \int_{\frac{d}{2}}^{\frac{D}{2}} r \sin(k_n r \cos(\psi_n - \theta)) dr d\theta \right]
\end{aligned} \tag{2.17}$$

Unfortunately, the integrals in (2.17) defy exact solution,[2] and so the best that can be done here is some partial and asymptotic analyses to get a sense of the nature of (2.17). That said, such results (luckily) are forthcoming.

To begin, attempting to solve the integrals in (2.17) exactly yields the following partial results (see `Maple/pdf/eta_overline_integrals.pdf`).

$$\int_0^{2\pi} \int_{\frac{d}{2}}^{\frac{D}{2}} r \cos(k_n r \cos(\psi_n - \theta)) dr d\theta = \frac{1}{2k_n^2} \int_0^{2\pi} [\cdots] d\theta \tag{2.18a}$$

$$\int_0^{2\pi} \int_{\frac{d}{2}}^{\frac{D}{2}} r \sin(k_n r \cos(\psi_n - \theta)) dr d\theta = \frac{1}{2k_n^2} \int_0^{2\pi} [\cdots] d\theta \tag{2.18b}$$

So then, presumably the dimensionless term

$$\Pi_{k_n} = k_n^2 (D^2 - d^2) \tag{2.19}$$

is of significance in any exact solution to the integrals in (2.17).

Next, for the limiting case of $d = 0$, it can be shown (see `Maple/pdf/eta_overline_integrals.pdf`) that

[2] To the best of the author's ability.

$$\lim_{D\to 0^+} \frac{4}{\pi D^2} \int_0^{2\pi} \int_0^{\frac{D}{2}} r\cos(k_n r\cos(\psi_n - \theta))drd\theta = 1 \tag{2.20a}$$

$$\lim_{D\to 0^+} \frac{4}{\pi D^2} \int_0^{2\pi} \int_0^{\frac{D}{2}} r\sin(k_n r\cos(\psi_n - \theta))drd\theta = 0 \tag{2.20b}$$

And so, as $D \to 0^+$ in this case,

$$\lim_{D\to 0^+} \overline{\eta}(t) \cong \sum_{n=1}^{N} a_n \cos\left(\frac{2\pi nt}{T} - \phi_n\right) \tag{2.21}$$

as one might logically expect (as this is the expression for deviation in sea surface elevation at a point, as per [4]; simply let $r = 0$ in (2.13)).

Next, for the limiting case of $d = 0$, it can be shown (see `Maple/pdf/eta_overline_integrals.pdf`) that

$$\lim_{D\to\infty} \frac{4}{\pi D^2} \int_0^{2\pi} \int_0^{\frac{D}{2}} r\cos(k_n r\cos(\psi_n - \theta))drd\theta \sim \lim_{D\to\infty} \pm\frac{L_1}{\pi k_n^2 D} = 0 \tag{2.22a}$$

$$\lim_{D\to\infty} \frac{4}{\pi D^2} \int_0^{2\pi} \int_0^{\frac{D}{2}} r\sin(k_n r\cos(\psi_n - \theta))drd\theta \sim \lim_{D\to\infty} \mp\frac{L_2}{\pi k_n^2 D} = 0 \tag{2.22b}$$

for some constants (bounded) $L_1 \geq 0$ and $L_2 \geq 0$. And so, as $D \to \infty$ in this case,

$$\lim_{D\to\infty} \overline{\eta}(t) \cong 0 \tag{2.23}$$

as one might logically expect (over a large enough area of sea, the average deviation from mean sea level approaches zero as per [4]).

Finally, given the results laid out in (2.19) - (2.23), one *potential* form for $\overline{\eta}$ that satisfies (for small d) is

$$\overline{\eta}(t) \cong \sum_{n=1}^{N} a_n I_{\cos} \cos\left(\frac{2\pi nt}{T} - \phi_n\right) \quad \text{for} \quad D \geq d \tag{2.24}$$

where $I_{\cos}$ is given by

$$I_{\cos} = \frac{4}{\pi(D^2 - d^2)} \int_0^{2\pi} \int_{\frac{d}{2}}^{\frac{D}{2}} r\cos(k_n r\cos(\psi_n - \theta))drd\theta \tag{2.25}$$

To facilitate this, a numerical approximation function was written for $I_{\cos}$ and is implemented as part of `Python/pyx/wave_energy_converter.pyx`.[3]

[3]Furthermore, evidence is presented in `Maple/pdf/eta_overline_integrals.pdf` that the corresponding $I_{\sin}$ is, in fact, everywhere zero (or very nearly).

2.4.3 Particular Solution to the Nonhomogeneous Equation

If one applies the same trig identity as was used previously, it follows that (2.24) can be expanded as follows

$$\overline{\eta}(t) \cong \sum_{n=1}^{N} a_n I_{\cos} \left[\cos(\phi_n) \cos\left(\frac{2\pi n t}{T}\right) + \sin(\phi_n) \sin\left(\frac{2\pi n t}{T}\right) \right] \quad \text{for} \quad D \geq d \tag{2.26}$$

or, more compactly,

$$\overline{\eta}(t) \cong \sum_{n=1}^{N} \left[\alpha_n \cos\left(\frac{2\pi n t}{T}\right) + \beta_n \sin\left(\frac{2\pi n t}{T}\right) \right] \tag{2.27a}$$

$$\alpha_n = a_n I_{\cos} \cos(\phi_n) \tag{2.27b}$$

$$\beta_n = a_n I_{\cos} \sin(\phi_n) \tag{2.27c}$$

From this, it follows that the nonhomogeneous equation can be expressed as

$$m\ddot{z} + b\dot{z} + (k + k_D)\, z \cong k_D \sum_{n=1}^{N} \left[\alpha_n \cos\left(\frac{2\pi n t}{T}\right) + \beta_n \sin\left(\frac{2\pi n t}{T}\right) \right] \tag{2.28}$$

Therefore, given the form of the right-hand side of (2.28), the method of undetermined coefficients can be invoked by assuming a particular solution of the form

$$z(t) \cong \sum_{n=1}^{N} \left[A_n \cos\left(\frac{2\pi n t}{T}\right) + B_n \sin\left(\frac{2\pi n t}{T}\right) \right] \tag{2.29}$$

where the A_n and B_n are the undetermined coefficients. This then implies that

$$\dot{z}(t) \cong \sum_{n=1}^{N} \left[-A_n \left(\frac{2\pi n}{T}\right) \sin\left(\frac{2\pi n t}{T}\right) + B_n \left(\frac{2\pi n}{T}\right) \cos\left(\frac{2\pi n t}{T}\right) \right] \tag{2.30a}$$

$$\ddot{z}(t) \cong \sum_{n=1}^{N} \left[-A_n \left(\frac{2\pi n}{T}\right)^2 \cos\left(\frac{2\pi n t}{T}\right) - B_n \left(\frac{2\pi n}{T}\right)^2 \sin\left(\frac{2\pi n t}{T}\right) \right] \tag{2.30b}$$

Substitution of (2.29) and (2.30a-b) into (2.28) then yields

$$m\sum_{n=1}^{N}\left[-A_n\left(\frac{2\pi n}{T}\right)^2\cos\left(\frac{2\pi nt}{T}\right)-B_n\left(\frac{2\pi n}{T}\right)^2\sin\left(\frac{2\pi nt}{T}\right)\right]+$$
$$b\sum_{n=1}^{N}\left[-A_n\left(\frac{2\pi n}{T}\right)\sin\left(\frac{2\pi nt}{T}\right)+B_n\left(\frac{2\pi n}{T}\right)\cos\left(\frac{2\pi nt}{T}\right)\right]+$$
$$(k+k_D)\sum_{n=1}^{N}\left[A_n\cos\left(\frac{2\pi nt}{T}\right)+B_n\sin\left(\frac{2\pi nt}{T}\right)\right]\cong$$
$$k_D\sum_{n=1}^{N}\left[\alpha_n\cos\left(\frac{2\pi nt}{T}\right)+\beta_n\sin\left(\frac{2\pi nt}{T}\right)\right] \quad (2.31)$$

Equating the sine and cosine terms in (2.31), by index n, then reveals the following system of linear equations

$$\begin{bmatrix} k+k_D-m\left(\frac{2\pi n}{T}\right)^2 & b\left(\frac{2\pi n}{T}\right) \\ -b\left(\frac{2\pi n}{T}\right) & k+k_D-m\left(\frac{2\pi n}{T}\right)^2 \end{bmatrix}\begin{bmatrix} A_n \\ B_n \end{bmatrix}\cong\begin{bmatrix} k_D\alpha_n \\ k_D\beta_n \end{bmatrix} \quad (2.32)$$

Now, provided that the determinant of the 2×2 in (2.32) is non-zero, namely

$$\left(k+k_D-m\left(\frac{2\pi n}{T}\right)^2\right)^2+b^2\left(\frac{2\pi n}{T}\right)^2\neq 0 \quad (2.33)$$

(which is guaranteed for $b>0$ since $n\geq 1$) it follows that the unique solution to (2.32) is, by Cramer's rule,

$$A_n\cong\frac{k_D\alpha_n\left(k+k_D-m\left(\frac{2\pi n}{T}\right)^2\right)-k_D\beta_n b\left(\frac{2\pi n}{T}\right)}{\left(k+k_D-m\left(\frac{2\pi n}{T}\right)^2\right)^2+b^2\left(\frac{2\pi n}{T}\right)^2} \quad (2.34a)$$

$$B_n\cong\frac{k_D\beta_n\left(k+k_D-m\left(\frac{2\pi n}{T}\right)^2\right)+k_D\alpha_n b\left(\frac{2\pi n}{T}\right)}{\left(k+k_D-m\left(\frac{2\pi n}{T}\right)^2\right)^2+b^2\left(\frac{2\pi n}{T}\right)^2} \quad (2.34b)$$

(2.34a-b) into (2.29) thus defines a particular solution to (2.28) (see `Maple/pdf/ODE_particular_solution.pdf`).

2.4.4 Computing Expected Power

Observe that the power captured (or dissipated) by the WEC PTO is given by

$$P = b\dot{z}^2 \tag{2.35}$$

And so, substitution of (2.30a) yields

$$\begin{aligned} P \cong b\left(\sum_{n=1}^{N}\left[-A_n\left(\frac{2\pi n}{T}\right)\sin\left(\frac{2\pi nt}{T}\right)+B_n\left(\frac{2\pi n}{T}\right)\cos\left(\frac{2\pi nt}{T}\right)\right]\right)^2 = \\ b\sum_{n=1}^{N}\sum_{j=1}^{N}A_nA_j\left(\frac{2\pi n}{T}\right)\left(\frac{2\pi j}{T}\right)\sin\left(\frac{2\pi nt}{T}\right)\sin\left(\frac{2\pi jt}{T}\right)- \\ 2b\sum_{n=1}^{N}\sum_{j=1}^{N}A_nB_j\left(\frac{2\pi n}{T}\right)\left(\frac{2\pi j}{T}\right)\sin\left(\frac{2\pi nt}{T}\right)\cos\left(\frac{2\pi nt}{T}\right)+ \\ b\sum_{n=1}^{N}\sum_{j=1}^{N}B_nB_j\left(\frac{2\pi n}{T}\right)\left(\frac{2\pi j}{T}\right)\cos\left(\frac{2\pi nt}{T}\right)\cos\left(\frac{2\pi jt}{T}\right) \end{aligned} \tag{2.36}$$

Now, the expected value (over the fundamental period T) of (2.36) can be expressed as

$$\begin{aligned} \mathrm{E}\{P\} \cong b\mathrm{E}\left\{\left(\sum_{n=1}^{N}\left[-A_n\left(\frac{2\pi n}{T}\right)\sin\left(\frac{2\pi nt}{T}\right)+B_n\left(\frac{2\pi n}{T}\right)\cos\left(\frac{2\pi nt}{T}\right)\right]\right)^2\right\} = \\ b\sum_{n=1}^{N}\sum_{j=1}^{N}A_nA_j\left(\frac{2\pi n}{T}\right)\left(\frac{2\pi j}{T}\right)\mathrm{E}\left\{\sin\left(\frac{2\pi nt}{T}\right)\sin\left(\frac{2\pi jt}{T}\right)\right\}- \\ 2b\sum_{n=1}^{N}\sum_{j=1}^{N}A_nB_j\left(\frac{2\pi n}{T}\right)\left(\frac{2\pi j}{T}\right)\mathrm{E}\left\{\sin\left(\frac{2\pi nt}{T}\right)\cos\left(\frac{2\pi nt}{T}\right)\right\}+ \\ b\sum_{n=1}^{N}\sum_{j=1}^{N}B_nB_j\left(\frac{2\pi n}{T}\right)\left(\frac{2\pi j}{T}\right)\mathrm{E}\left\{\cos\left(\frac{2\pi nt}{T}\right)\cos\left(\frac{2\pi jt}{T}\right)\right\} \end{aligned} \tag{2.37}$$

or, in integral form,

$$\mathrm{E}\{P\} \cong b\mathrm{E}\left\{\left(\sum_{n=1}^{N}\left[-A_n\left(\frac{2\pi n}{T}\right)\sin\left(\frac{2\pi nt}{T}\right)+B_n\left(\frac{2\pi n}{T}\right)\cos\left(\frac{2\pi nt}{T}\right)\right]\right)^2\right\}=$$
$$b\sum_{n=1}^{N}\sum_{j=1}^{N}A_nA_j\left(\frac{2\pi n}{T}\right)\left(\frac{2\pi j}{T}\right)\left(\frac{1}{T}\int_0^T\sin\left(\frac{2\pi nt}{T}\right)\sin\left(\frac{2\pi jt}{T}\right)dt\right)-$$
$$2b\sum_{n=1}^{N}\sum_{j=1}^{N}A_nB_j\left(\frac{2\pi n}{T}\right)\left(\frac{2\pi j}{T}\right)\left(\frac{1}{T}\int_0^T\sin\left(\frac{2\pi nt}{T}\right)\cos\left(\frac{2\pi nt}{T}\right)dt\right)+$$
$$b\sum_{n=1}^{N}\sum_{j=1}^{N}B_nB_j\left(\frac{2\pi n}{T}\right)\left(\frac{2\pi j}{T}\right)\left(\frac{1}{T}\int_0^T\cos\left(\frac{2\pi nt}{T}\right)\cos\left(\frac{2\pi jt}{T}\right)dt\right) \tag{2.38}$$

177 But then, by orthogonality (see `Maple/pdf/orthogonality.pdf`), namely

$$\frac{1}{T}\int_0^T\sin\left(\frac{2\pi nt}{T}\right)\sin\left(\frac{2\pi jt}{T}\right)dt=\frac{\delta_{nj}}{2} \tag{2.39a}$$

178

179

$$\frac{1}{T}\int_0^T\sin\left(\frac{2\pi nt}{T}\right)\cos\left(\frac{2\pi nt}{T}\right)dt=0 \tag{2.39b}$$

180

181

$$\frac{1}{T}\int_0^T\cos\left(\frac{2\pi nt}{T}\right)\cos\left(\frac{2\pi jt}{T}\right)dt=\frac{\delta_{nj}}{2} \tag{2.39c}$$

182

183

$$\delta_{nj}=\begin{cases}1 & \text{if} \quad n=j\\ \\ 0 & \text{otherwise}\end{cases} \tag{2.39d}$$

184

185 it follows that (2.38) reduces significantly to

$$\mathrm{E}\{P\} \cong \frac{b}{2}\sum_{n=1}^{N}\left(A_n^2+B_n^2\right)\left(\frac{2\pi n}{T}\right)^2 \tag{2.40}$$

2.5 Summary of Results

Given environmental and WEC design parameters, together with

$$m = \frac{\pi \rho h}{4}(D^2 - d^2)$$

$$k_D = \frac{\pi \rho g}{4}(D^2 - d^2)$$

$$\overline{\eta}(t) \cong \sum_{n=1}^{N} \left[\alpha_n \cos\left(\frac{2\pi n t}{T}\right) + \beta_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

$$\alpha_n = a_n I_{\cos} \cos(\phi_n)$$

$$\beta_n = a_n I_{\cos} \sin(\phi_n)$$

$$I_{\cos} = \frac{4}{\pi(D^2 - d^2)} \int_0^{2\pi} \int_{\frac{d}{2}}^{\frac{D}{2}} r \cos(k_n r \cos(\psi_n - \theta)) dr d\theta$$

it can be shown (see above) that

$$z(t) \cong \sum_{n=1}^{N} \left[A_n \cos\left(\frac{2\pi n t}{T}\right) + B_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

$$A_n \cong \frac{k_D \alpha_n \left(k + k_D - m\left(\frac{2\pi n}{T}\right)^2\right) - k_D \beta_n b \left(\frac{2\pi n}{T}\right)}{\left(k + k_D - m\left(\frac{2\pi n}{T}\right)^2\right)^2 + b^2 \left(\frac{2\pi n}{T}\right)^2}$$

$$B_n \cong \frac{k_D \beta_n \left(k + k_D - m\left(\frac{2\pi n}{T}\right)^2\right) + k_D \alpha_n b \left(\frac{2\pi n}{T}\right)}{\left(k + k_D - m\left(\frac{2\pi n}{T}\right)^2\right)^2 + b^2 \left(\frac{2\pi n}{T}\right)^2}$$

$$\mathrm{E}\{P\} \cong \frac{b}{2} \sum_{n=1}^{N} \left(A_n^2 + B_n^2\right) \left(\frac{2\pi n}{T}\right)^2$$

3 Simulating a Heave-Constrained Point Absorber

Recall that the intent of this work is to apply perturbation theory to the modelling of point absorbers in order to reduce computational expense. The reduced problem has already been solved, and so what remains is deriving an appropriate perturbation of the reduced dynamics with the goal of approaching the complex dynamics.

The approach taken in this work is to first simulate the complex dynamics using established modelling software, namely Proteus DS (PDS) [5]. The complex dynamics are then compared to the reduced dynamics in order to determine an appropriate form for the perturbation. Finally, a perturbation machine is designed and trained.

3.1 Simulation Design

An example PDS simulation is provided at

`ProteusDS/WEC_sim_example.zip`

For additional details, refer to the example.

Each PDS simulation considered a heave constrained point absorber as illustrated in Figure 2.1. An example of the corresponding representation within PDS (for the case $D = 30$ m) is illustrated in Figures 3.1 and 3.2.

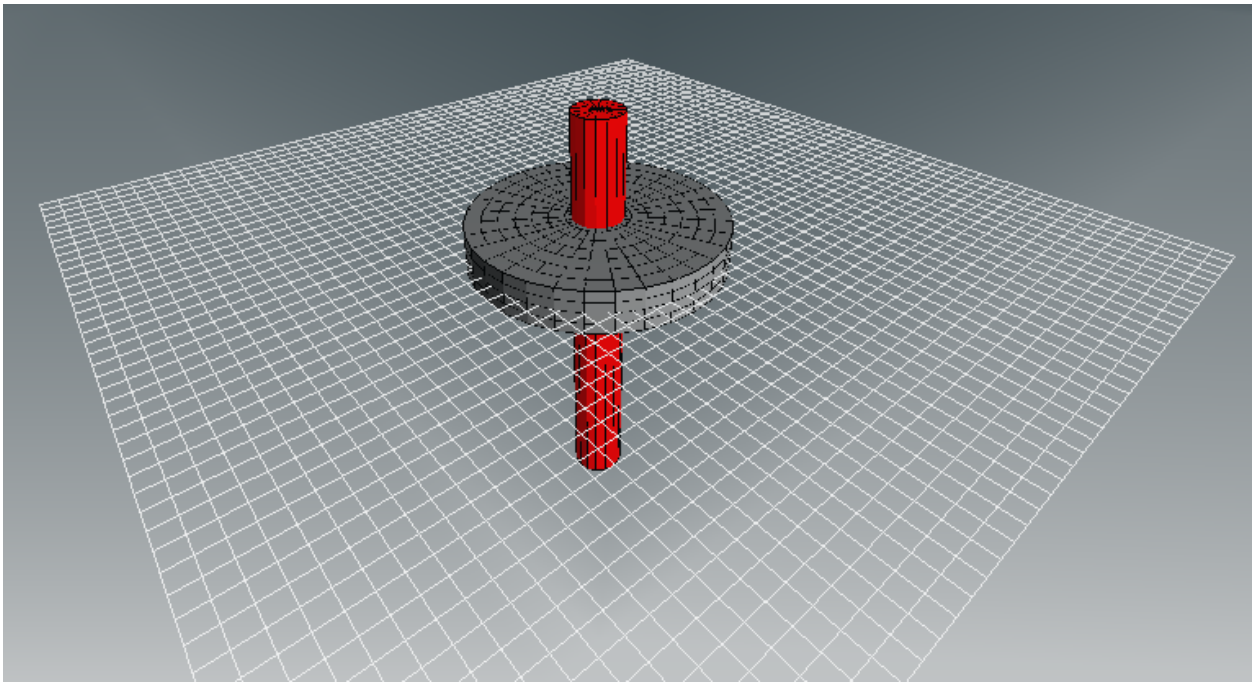

Figure 3.1: WEC model within PDS (isometric view).

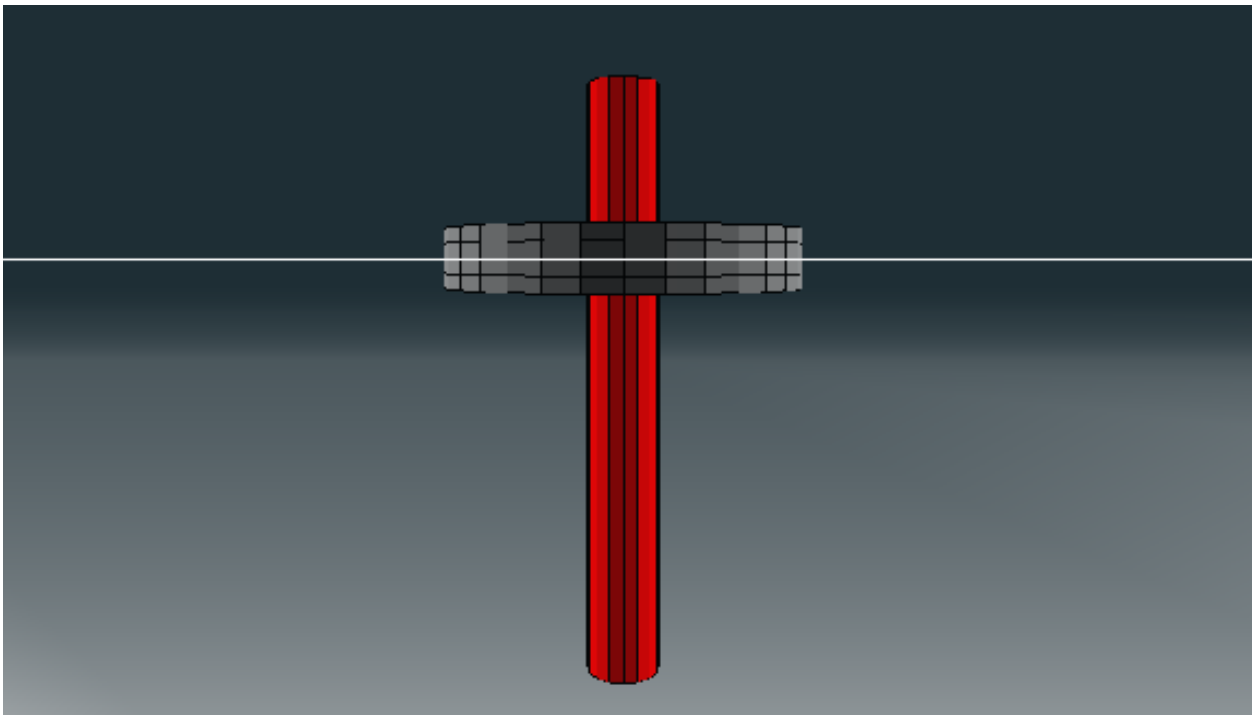

Figure 3.2: WEC model within PDS (orthometric view).

Each simulation had the following key features

1. Only waves were considered. Currents and winds were neglected.

2. An irregular sea surface was generated using a Pierson-Moskowitz spectrum under given significant wave height (H_s) and wave spectral peak period (T_p) values. For more details, see [4].

3. The sea state was simulated as initially being at rest, and was then gradually ramped up to the desired state over a period of 100 seconds. After ramping, the simulation then continued for another 900 seconds.

4. Simulation state was logged at a frequency of 2 Hz. Simulation state was updated at whatever frequency was appropriate (an adaptive time step integrator was used within PDS).

5. External forces due to buoyancy, weight, PTO reaction, drag, wave incidence, and added mass were all captured. Wave diffraction, wave radiation, and fluid memory effects were neglected.[1]

An example animation of the simulated WEC response is provided at

`animations/WEC_sim_example_D30m.gif`

That said, there is one simulation caveat to keep in mind here. That is, the float is modelled in PDS as a simple cylinder primitive (a `RigidBodyCylinder` feature, to use the PDS terminology), and so it does not have a 6 m diameter hole in its centre; the float and piling simply clip through each other! This was done to make the simulations simple, for the sake of run times, and easily scaleable, for the sake of data generation. As such, the simulated expected power output values are no doubt an overestimate of what the true WEC would produce. However, the degree of overestimation is bounded and presumably proportional to the ratio of waterplane areas (simulated to true). In the worst case scenario of $d = 6$ m and $D = 10$ m it would be

$$\frac{D^2}{D^2 - d^2} \cong 1.5625$$

or about a 1.6× overestimation. In the best case scenario of $d = 6$ m and $D = 50$ m, the overestimation shrinks to

$$\frac{D^2}{D^2 - d^2} \cong 1.0146$$

hardly any overestimation at all. Considering values of D between 10 m and 50 m yields the results illustrated in Figure 3.3.

[1] Although neglecting diffraction is arguably not a big deal in this case, for the lower values of D. Small-body approximation, etc; see [6].

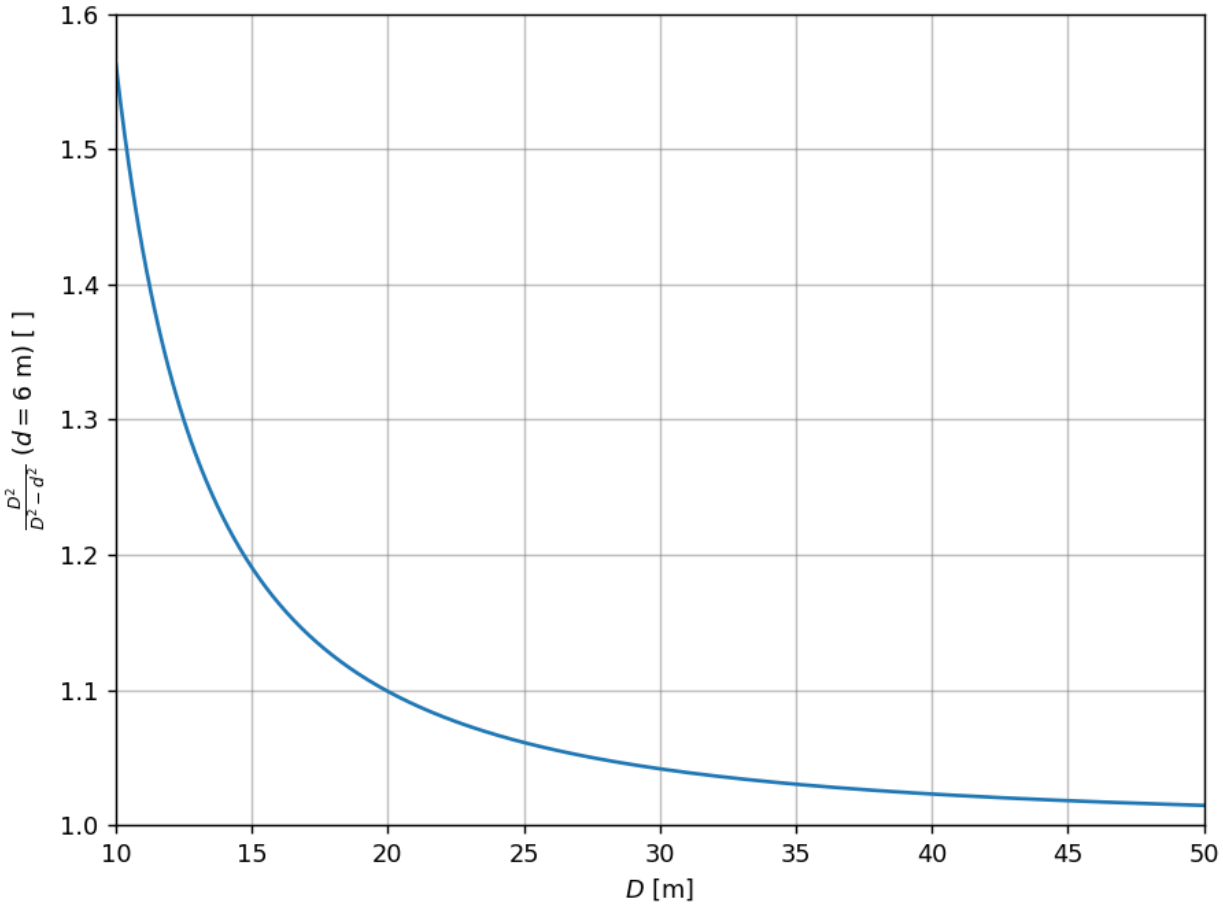

Figure 3.3: Ratio of waterplane areas (simulated to true).

As a simple post-processing means of correcting for this overestimation, *the expected power outputs from PDS were scaled by a factor of*

$$\frac{D^2 - d^2}{D^2}$$

3.2 Data Generation

The method of exhaustion was applied here. A data table consisting of 35,346 rows (one per PDS simulation) was generated by considering the following feature values (selected so as to achieve a simulation space filling lattice)

$$Z \in \{20, 30, 40, \cdots, 100\} \text{ m} \tag{3.1}$$

$$T_p \in \{5, 6.25, 7.5, \cdots, 15\} \text{ s} \tag{3.2}$$

$$D \in \{10, 15, \cdots, 50\} \text{ m} \tag{3.3}$$

$$k \in \{0, 125, 250, \cdots, 1000\} \text{ N/m} \tag{3.4}$$

$$b \in \{10^4, 5 \times 10^4, 10^5, 5 \times 10^5, \cdots, 10^8\} \text{ N.s/m} \tag{3.5}$$

Significant wave height (H_s) was also varied, but in a physically constrained manner. The following deep water wave breaking heuristic was applied [7][2]

$$\{H_s\}_{\text{break}} \cong 0.0204 g T_p^2 \tag{3.6}$$

and then H_s values in the closed interval $[0.05\,\{H_s\}_{\text{break}}, \{H_s\}_{\text{break}}]$ were considered. All other features were held fixed; namely

1. Float inner diameter d was fixed at 6 m.
2. Float draft h was fixed at 2.5 m. Float mass m then follows from (2.6).
3. Fluid density ρ was fixed at 1,025 kg/m^3 (nominal density of seawater).
4. Gravity g was fixed at 9.81 m/s^2.

For each PDS simulation run, the following outputs were logged in the data table

1. The expected power output (E$\{P\}$, [kW]) of the WEC over the last 900 seconds of simulation.
2. The deep water wave power transport [kW/m], according to [4, 7][3]

$$J \cong \frac{\rho g^2}{64\pi} H_s^2 T_p \tag{3.7}$$

3. The deep water hydrodynamic efficiency [ ], given by

$$\eta_{\text{hydro}} = \frac{\text{E}\{P\}}{JD} \tag{3.8}$$

4. The simulation terminal condition (either `COMPLETE` or `ERROR`).

The resulting data table was saved (after post-processing for overestimation) to

`Python/data/PDS_simulation_results.csv`

[2]Taking $T_e \cong T_p$, where T_e is the spectral energy period. Also, note that if $g = 9.81$ m/s^2, then $0.0204g \cong 0.2$ m/s^2. Compare this to the factor of $0.14 \times 1.56 = 0.2184$ m/s^2 given on p. 9 of [7].

[3]Again, taking $T_e \cong T_p$, where T_e is the spectral energy period.

3.3 Data Mining

For implementation details, see

`Python/mine_PDS_simulation_results.py`

For the sake of mining the PDS simulation results (and for subsequently training the desired perturbation machine) the features were taken to be water depth (Z), significant wave height (H_s), wave peak period (T_p), float outer diameter (D), PTO stiffness (k), and PTO damping (b). Likewise, the target was taken to be expected power output ($\mathrm{E}\{P\}$).

To begin, the data was cleaned by first dropping all rows which either contain a `NaN`, or for which the simulation terminal condition is `ERROR`. Doing so resulted in 98.35% of the original data being retained. Then, rows in which the significant wave height to sea depth ratio was excessive were dropped. In this case, "excessive" was taken to be

$$H_s > \frac{Z}{2} \tag{3.9}$$

Dropping all such rows resulted in 91.3% of the data being retained.

After cleaning, the following target statistics were extracted

```
min target = 0.001 kW
max target = 78806.231 kW
mean target = 1592.799 kW
standard deviation of target = 4639.114 kW
median target = 115.84 kW

target percentiles:
    5-percentile: 0.603 kW
    20-percentile: 8.371 kW
    40-percentile: 55.913 kW
    60-percentile: 245.014 kW
    80-percentile: 1389.752 kW
    90-percentile: 4284.355 kW
    95-percentile: 8376.364 kW
    99-percentile: 23374.501 kW
```

Plotting a histogram of the target values yields Figure 3.4.

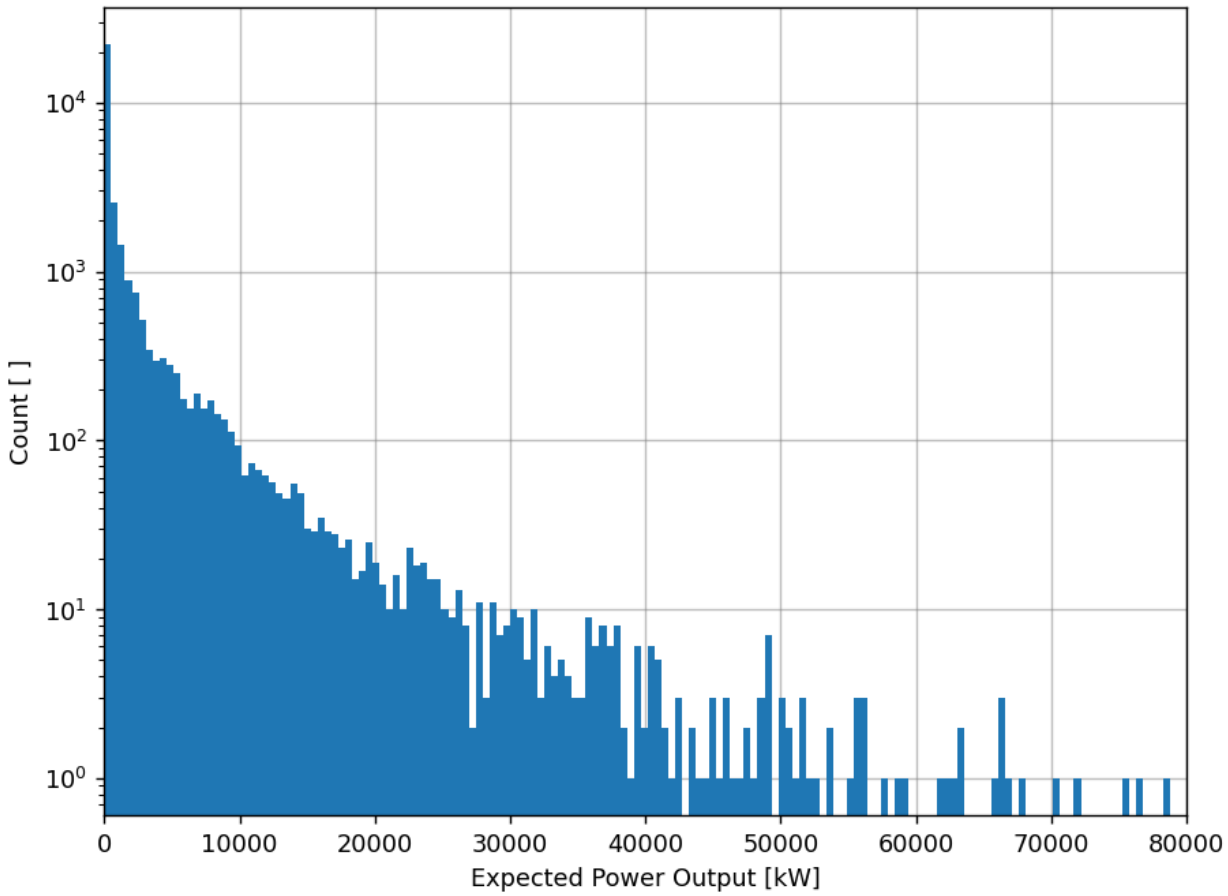

Figure 3.4: Histogram of simulated expected power output (target) values.

Of note in the extracted statistics, as well as in Figure 3.4, is the wide range of target values simulated. This then indicates what one might logically expect; the dynamic response of the WEC is quite sensitive to the selected features.

Scatter plots of the selected features (after cleaning) were generated and are provided in Appendix A. Refer to these plots for an illustration of the resulting simulation space filling lattice. The cleaned feature and target arrays were saved to

`Python/data/feature_array.npy`
and
`Python/data/target_array.npy`

3.4 Adding Some Dimensionless Features

In addition to the six features enumerated above, four additional dimensionless features were considered (the inspiration here being the historical successes of the Buckingham Π Theorem; see [8] for details); namely

1. An aggregate form of Π_{k_n}, referred to here as simply Π_0.

2. The damping ratio of the WEC, referred to here as simply Π_1.

3. The ratio of the wave spectral peak period to the damped natural period of the WEC, referred to here as simply Π_2.

4. A dimensionless feature involving ρ, g, H_s, T_p, and b, referred to here as simply Π_3.

3.4.1 First Dimensionless Feature

For the first dimensionless feature, introduce the Pierson-Moskowitz wave spectrum as presented in [4].

$$S(f) \cong \frac{5}{16} H_s^2 T_p^{-4} f^{-5} \exp\left[-\frac{5}{4}\left(\frac{1}{T_p f}\right)^4\right] \tag{3.10}$$

Next, introduce the dispersion relation for Airy waves, given by [4]

$$(2\pi f_n)^2 = g k_n \tanh(k_n Z) \tag{3.11}$$

which defines an implicit relation between component wave frequency f_n and component wave number k_n as a function of gravity g and sea depth Z. Next, introduce characteristic wave number $\overline{k}$ as

$$\overline{k} = \frac{\sum_{n=1}^{N} k_n S_n}{\sum_{n=1}^{N} S_n} \tag{3.12}$$

with

$$S_n = \frac{1}{\Delta f} \int_{f_n - \frac{\Delta f}{2}}^{f_n + \frac{\Delta f}{2}} S(f) df \tag{3.13}$$

Finally, Π_0 is given by

$$\Pi_0 = \overline{k}^2 \left(D^2 - d^2\right) \tag{3.14}$$

3.4.2 Second Dimensionless Feature

For the second dimensionless feature, this is simply a result from classical dynamics (the motion of a damped harmonic oscillator). For the WEC considered here, Π_1 is the damping ratio given by

$$\Pi_1 = \frac{b}{2\sqrt{m(k + k_D)}} \tag{3.15}$$

3.4.3 Third Dimensionless Feature

For the third dimensionless feature, this is simply a result from classical dynamics (the motion of a damped harmonic oscillator). For the WEC considered here, the damped natural frequency f_b is given by

$$f_b = \begin{cases} \frac{\sqrt{1-\Pi_1^2}}{2\pi}\sqrt{\frac{k+k_D}{m}} & \text{if } |\Pi_1| < 1 \\ \\ \frac{1}{2\pi}\sqrt{\frac{k+k_D}{m}} & \text{otherwise} \end{cases} \tag{3.16}$$

Π_2 is then defined as the product of wave peak period and the damped natural frequency (or the ratio of the wave peak period to the damped natural period of the WEC); namely

$$\Pi_2 = T_p f_b \tag{3.17}$$

3.4.4 Fourth Dimensionless Feature

For the fourth dimensionless feature, this is assembled as follows

$$\Pi_3 = \frac{b}{\rho g H_s^2 T_p} \tag{3.18}$$

The intent of this dimensionless feature is to expose the feature space to changes in ρ and g, as well as to capture the ratio of damping to power available in the sea state.

3.5 Data Mining Continued

Appropriate values for each of the dimensionless features, corresponding to the rows of the cleaned data table, were generated and saved to

`Python/data/dimensionless_feature_array.npy`

For implementation details, see

`Python/generate_dimensionless_features.py`

Scatter plots of the dimensionless features were generated and are provided in Appendix B. Of note in these plots is the tendency of the target values to be clustered in the dimensionless feature space; for example, see Figure 3.5 below (reprinted here for ease of reference).

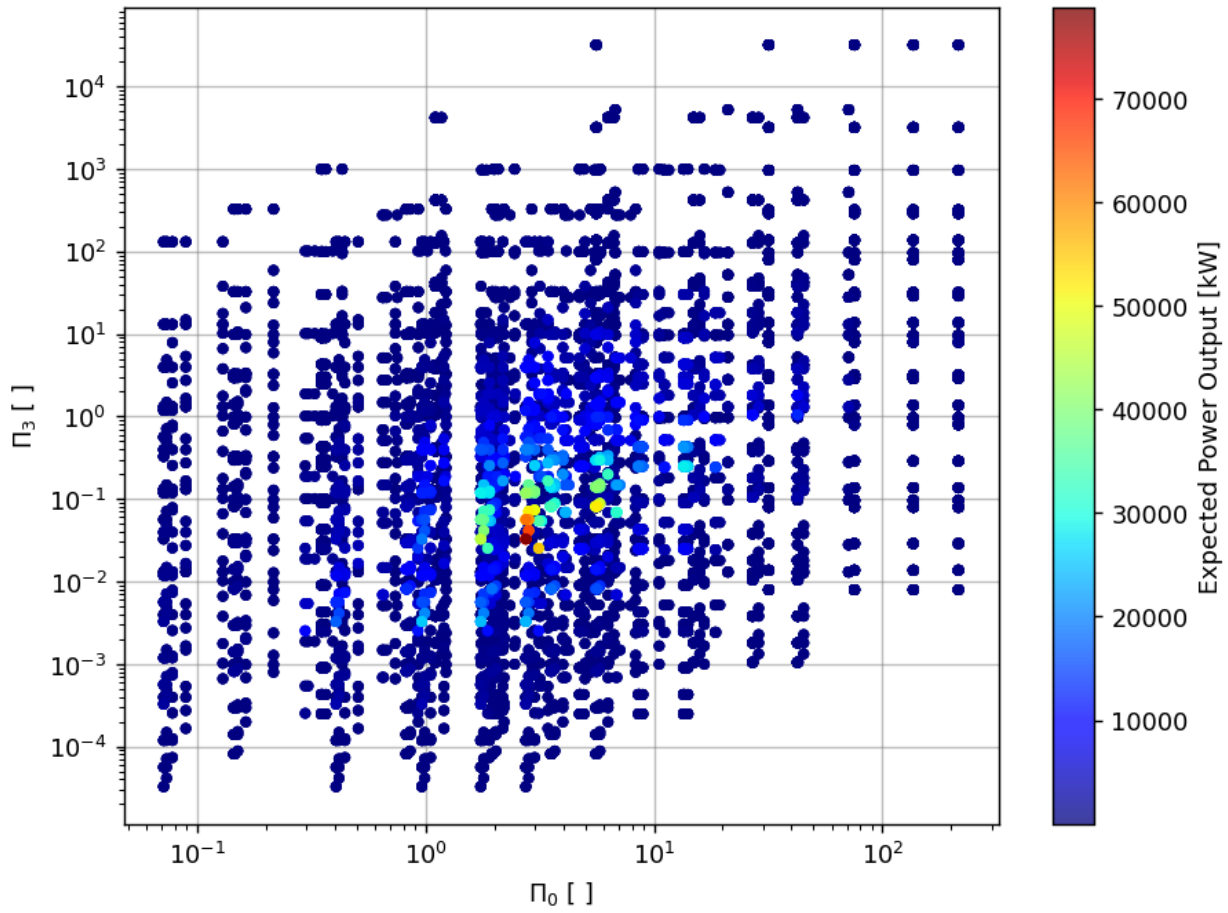

Figure 3.5: PDS simulation results mining. Dimensionless Feature 3 vs Dimensionless Feature 0. Plotted with "hottest" points visible.

This then implies that the dimensionless features likely contain useful information for the purpose of training the desired perturbation machine.

The features, dimensionless features, and target were then merged into a single array, and the b, Π_0, Π_1, and Π_3 columns were $\log_{10}$ scaled (so, transform $x \to \log_{10}(x)$ to linearize their distributions). Clustering was then performed by way of the t-distributed stochastic neighbour embedding (t-SNE) technique (as implemented in `sklearn.manifold.TSNE`; see [9]). The result of this clustering is illustrated in Figure 3.6

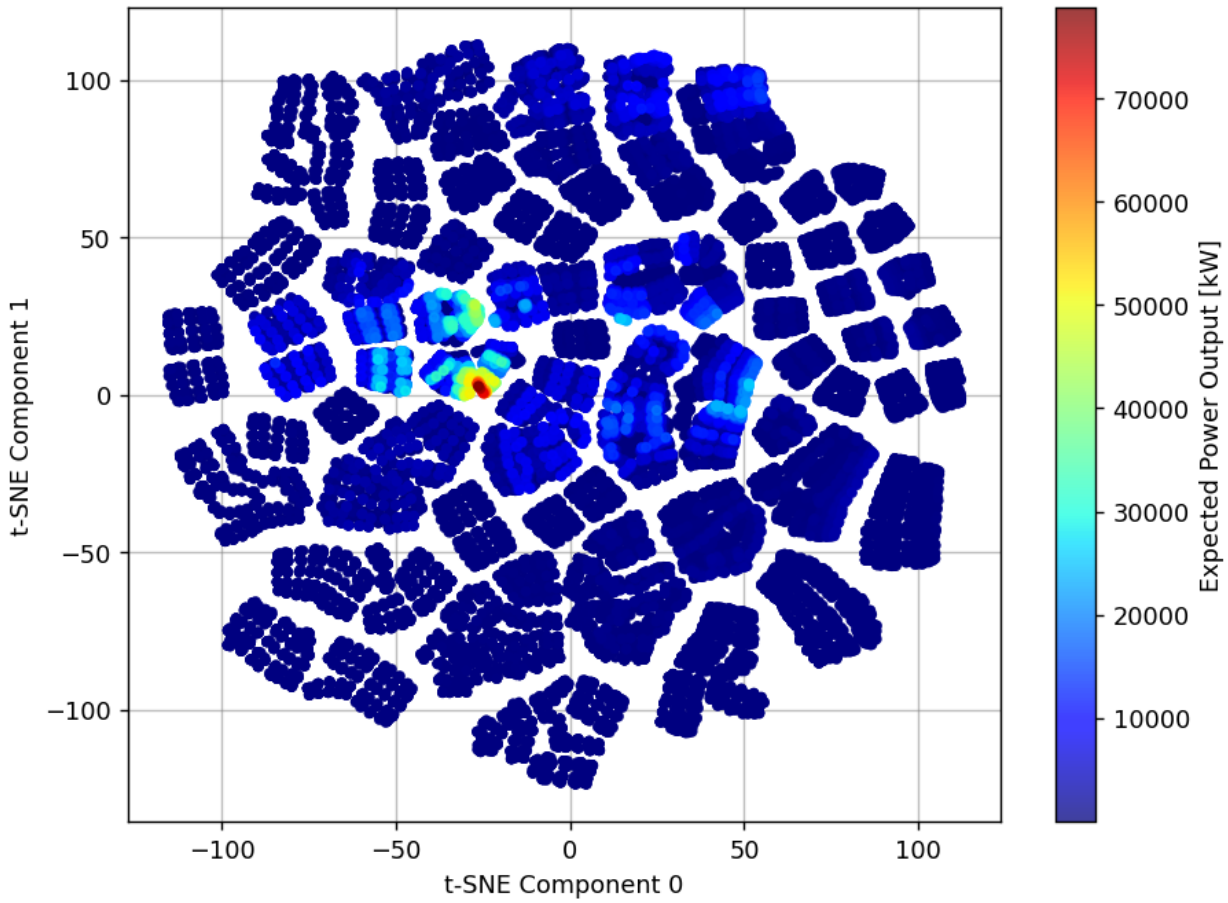

Figure 3.6: t-SNE clustering of the merge of the features, dimensionless features, and target. Plotted with "hottest" points visible.

Of note in Figure 3.6 is the fact that the high magnitude target values are narrowly distributed in the t-SNE space, with them being spread over only a few clusters (about two). This then reveals that there evidently is structure in the extended feature space which can be exploited by the desired perturbation machine.

4 Machine Learning: Building the Perturbation Machine

4.1 Data Generation

For implementation details, see

`Python/generate_reduced_dynamics_power.py`

Using the results summarized at the end of Chapter 2, the expected power output according to the reduced dynamics was computed under the following assumptions

1. A fundamental period of $T = 900$ seconds is used, so as to align with the PDS simulations.

2. The wave spectrum is Pierson-Moskowitz, as defined in (3.10). This aligns with the PDS simulations.

3. The component waves are Airy waves, and so the dispersion relation defined in (3.11) applies. (This also aligns with the PDS simulations.)

4. The wave component amplitudes a_n can be computed by way of

$$a_n = \sqrt{2S_n(\Delta f)_n} \tag{4.1}$$

   with S_n computed by way of (3.13).

5. The component phases are random numbers uniformly distributed over the closed interval $[-\pi, \pi]$. That is, a random amplitude, random phase model is applied [4].[1]

6. The component directions are random numbers normally distributed about $\psi = 0$ with a standard deviation of $\frac{\pi}{36}$ (so 95% of waves are within $\pm 10^\circ$ of $\psi = 0$). Given the symmetry of the WEC float, this can be done without loss of generality.

[1] It was also verified that using random phases like this does not result in wildly random expected power outputs (on the contrary, the outputs are very stable). This is the commented out portion in `Python/generate_reduced_dynamics_power.py`.

4.2 Data Mining

Plotting a superposition of histograms, namely the expected power outputs from the PDS simulations and from the reduced dynamics, yields Figure 4.1.

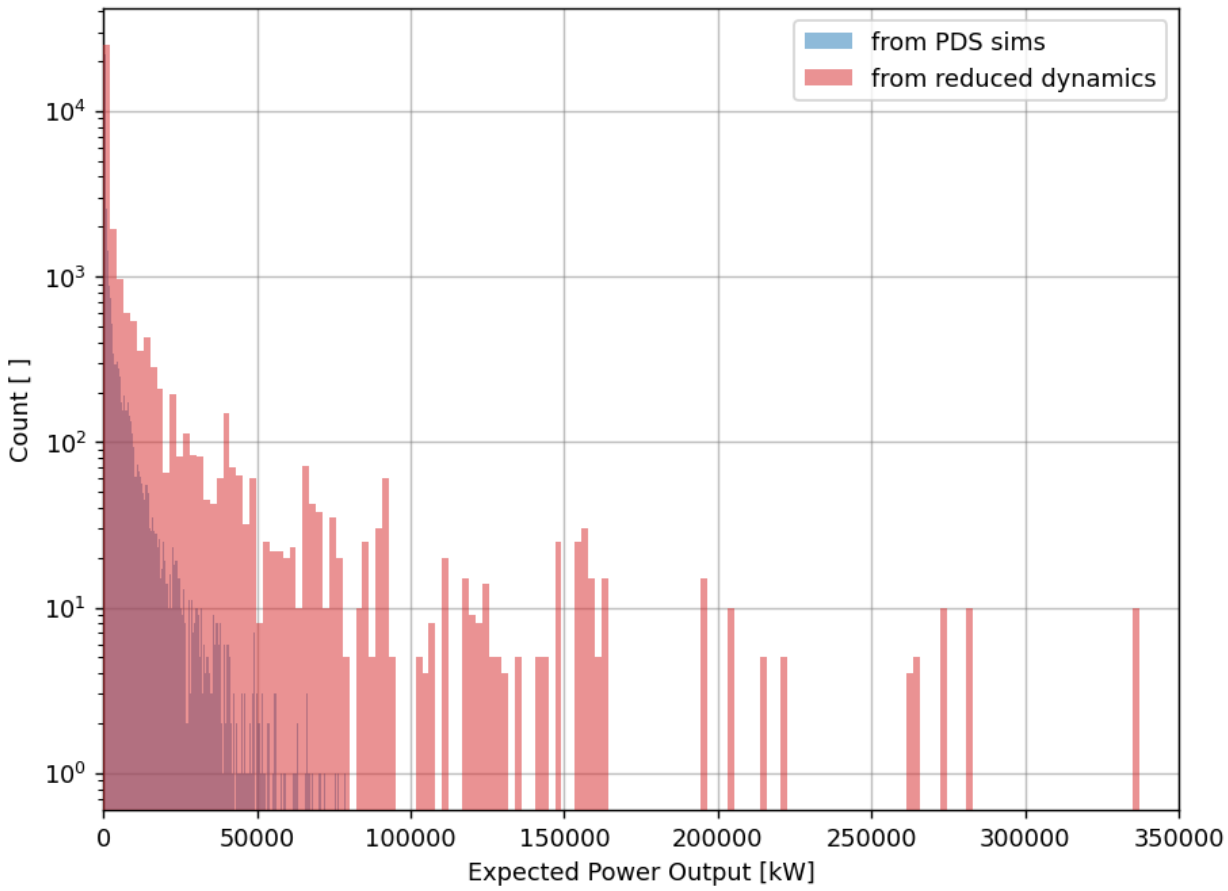

Figure 4.1: Superposition of the distributions in expected power output from the PDS sims and from the reduced dynamics.

Of note in Figure 4.1 is the fact that the expected power output under the reduced dynamics is similar in shape to the values derived from the PDS sims. That said, the scale is not similar, with the expected power output under the reduced dynamics tending to over-estimate WEC performance.[2] Fortunately, the similarity in shape is key, and it indicates that a perturbation machine could work well here. It just remains to decide whether a linear (additive) or separable (multiplicative) perturbation is more appropriate in this case.

To begin, define $\mathrm{E}_{\mathrm{PDS}}\{P\}$ to be the expected power output simulated by PDS, and define $\mathrm{E}_{\mathrm{reduced}}\{P\}$ to be the corresponding expected power output under the reduced dynamics.

[2]Which makes sense, since many important dynamics like drag and added mass are neglected in the reduced dynamics.

4.2.1 Considering a Linear Perturbation

Consider a linear perturbation of the expected power under the reduced dynamics

$$\mathrm{E}\{P\} = \mathrm{E}_{\text{reduced}}\{P\} + \mathcal{P}(\cdots) \tag{4.2}$$

That is, the "true" expected power output is the expected power output under the reduced dynamics plus a correction factor (the linear perturbation $\mathcal{P}$). To get an initial sense of what such a linear perturbation might look like, the distribution of $\mathrm{E}_{\text{PDS}}\{P\} - \mathrm{E}_{\text{reduced}}\{P\}$ was plotted (since, ideally, $\mathcal{P}(\cdots) = \mathrm{E}_{\text{PDS}}\{P\} - \mathrm{E}_{\text{reduced}}\{P\}$, if $\mathrm{E}_{\text{PDS}}\{P\}$ is taken to be "true"). The result is illustrated in Figure 4.2.

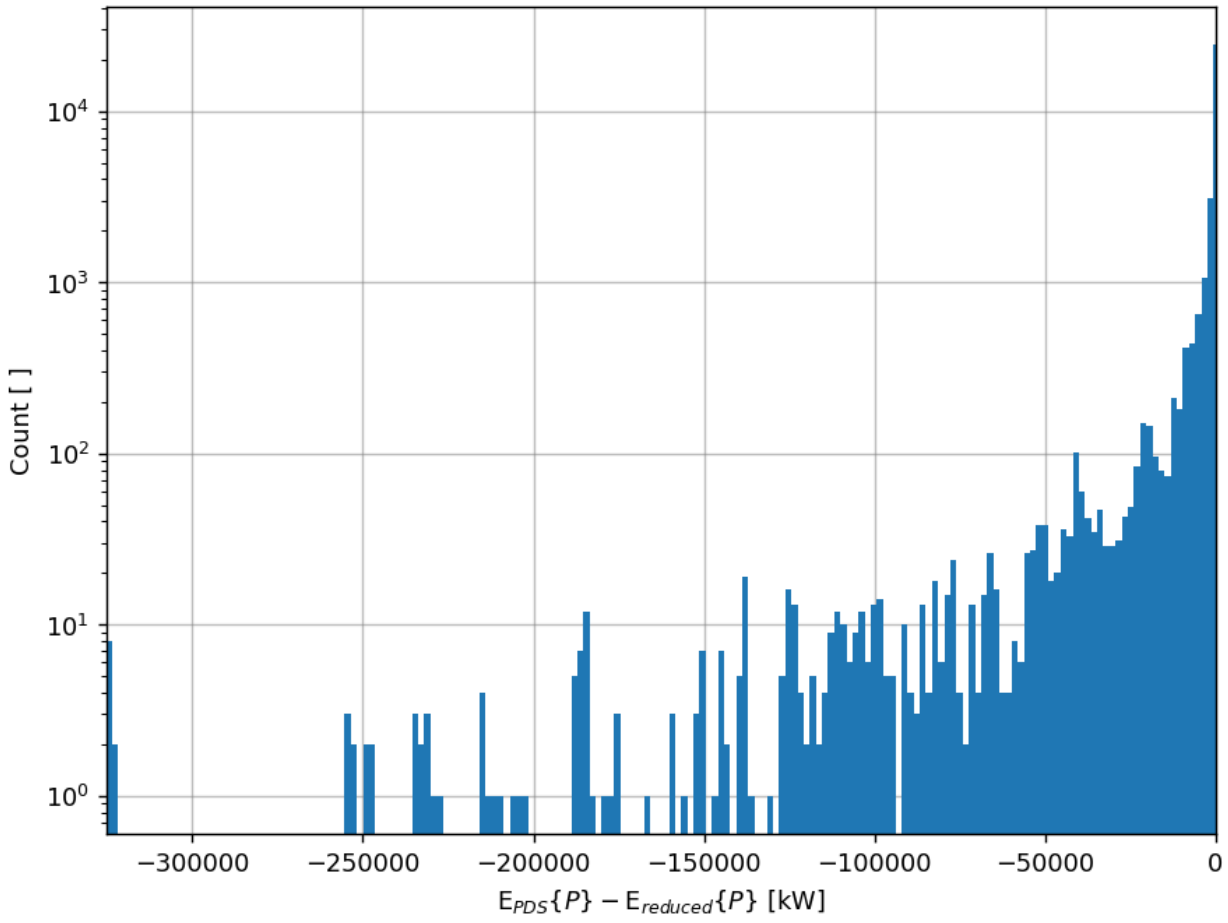

Figure 4.2: Histogram of target differences $\mathrm{E}_{\text{PDS}}\{P\} - \mathrm{E}_{\text{reduced}}\{P\}$.

Of note in Figure 4.2 is the wide range of values for the difference.

Applying the same t-SNE clustering technique that was employed previously yields Figure 4.3.

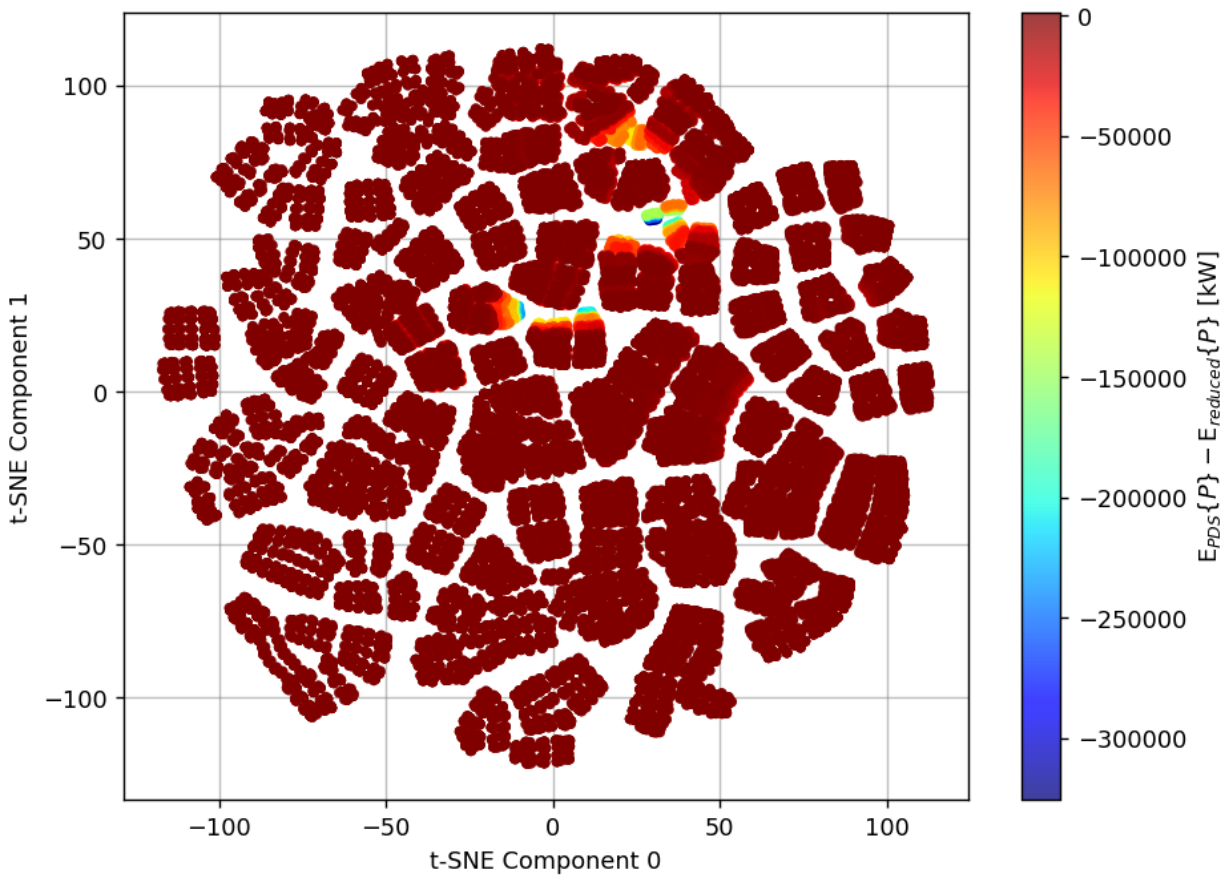

Figure 4.3: t-SNE clustering of the merge of the features, dimensionless features, and target differences. Plotted with "hottest" points visible.

Of note in Figure 4.3 is the fact that the large magnitude differences are a bit more widely distributed, with them being evenly spread over about six different clusters (compare to Figure 3.6). That said, while there is evident structure in the extended feature space, it would be more difficult to exploit.

4.2.2 Considering a Separable Perturbation

Consider a separable perturbation of the expected power under the reduced dynamics

$$\mathrm{E}\{P\} = \mathcal{P}(\cdots)\mathrm{E}_{\text{reduced}}\{P\} \tag{4.3}$$

That is, the "true" expected power output is the expected power output under the reduced dynamics multiplied by a correction factor (the separable perturbation $\mathcal{P}$). To get an initial sense of what such a separable perturbation might look like, the distribution of

$$\frac{\mathrm{E}_{\text{PDS}}\{P\}}{\mathrm{E}_{\text{reduced}}\{P\}}$$

was plotted (since, ideally, $\mathcal{P}(\cdots) = \mathrm{E}_{\mathrm{PDS}}\{P\}/\mathrm{E}_{\mathrm{reduced}}\{P\}$, if $\mathrm{E}_{\mathrm{PDS}}\{P\}$ is taken to be "true"). The result is illustrated in Figure 4.4.

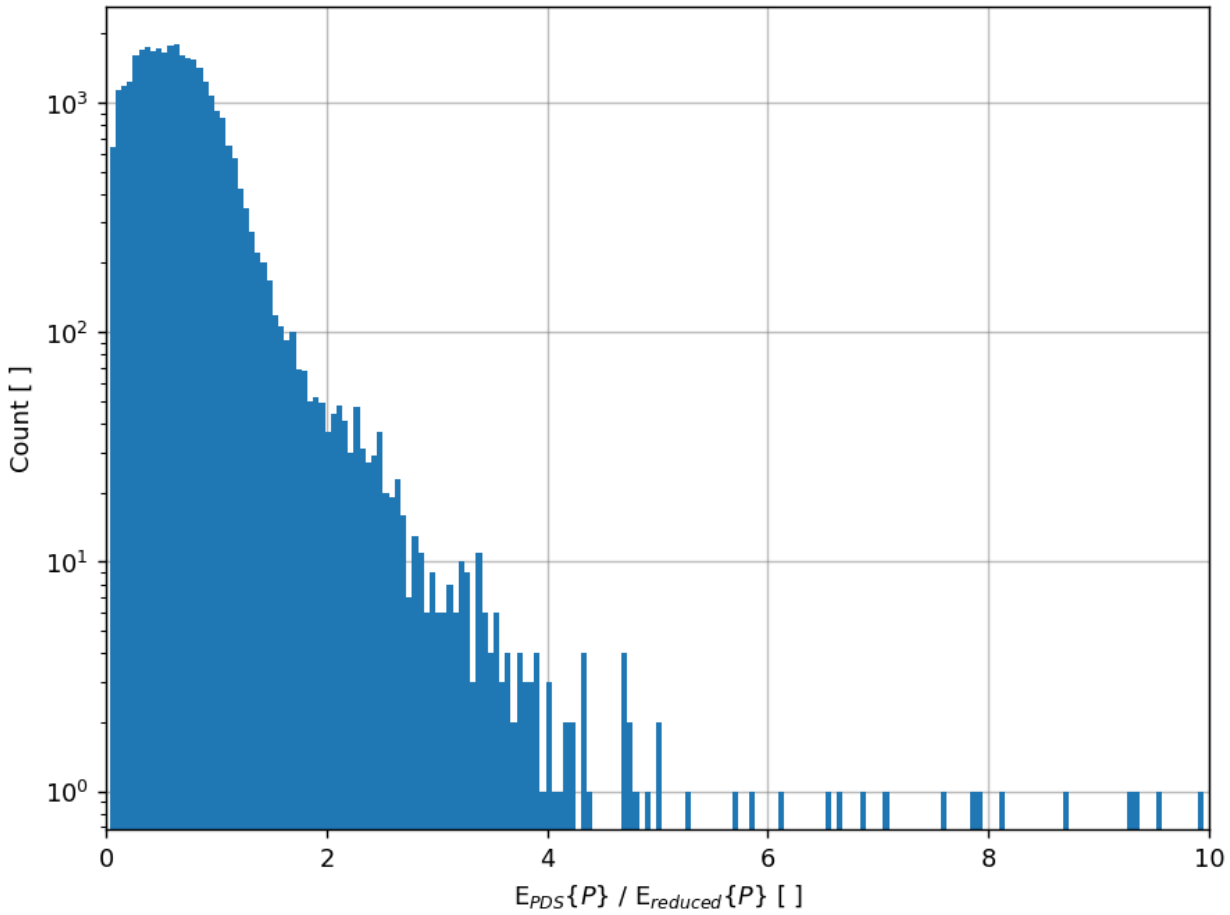

Figure 4.4: Histogram of target ratios $\mathrm{E}_{\mathrm{PDS}}\{P\}/\mathrm{E}_{\mathrm{reduced}}\{P\}$.

Of note in Figure 4.4 is the comparatively narrow range of values for the ratio, with most being contained in the closed interval $[0, 2]$.

Applying the same t-SNE clustering technique that was employed previously yields Figure 4.5.

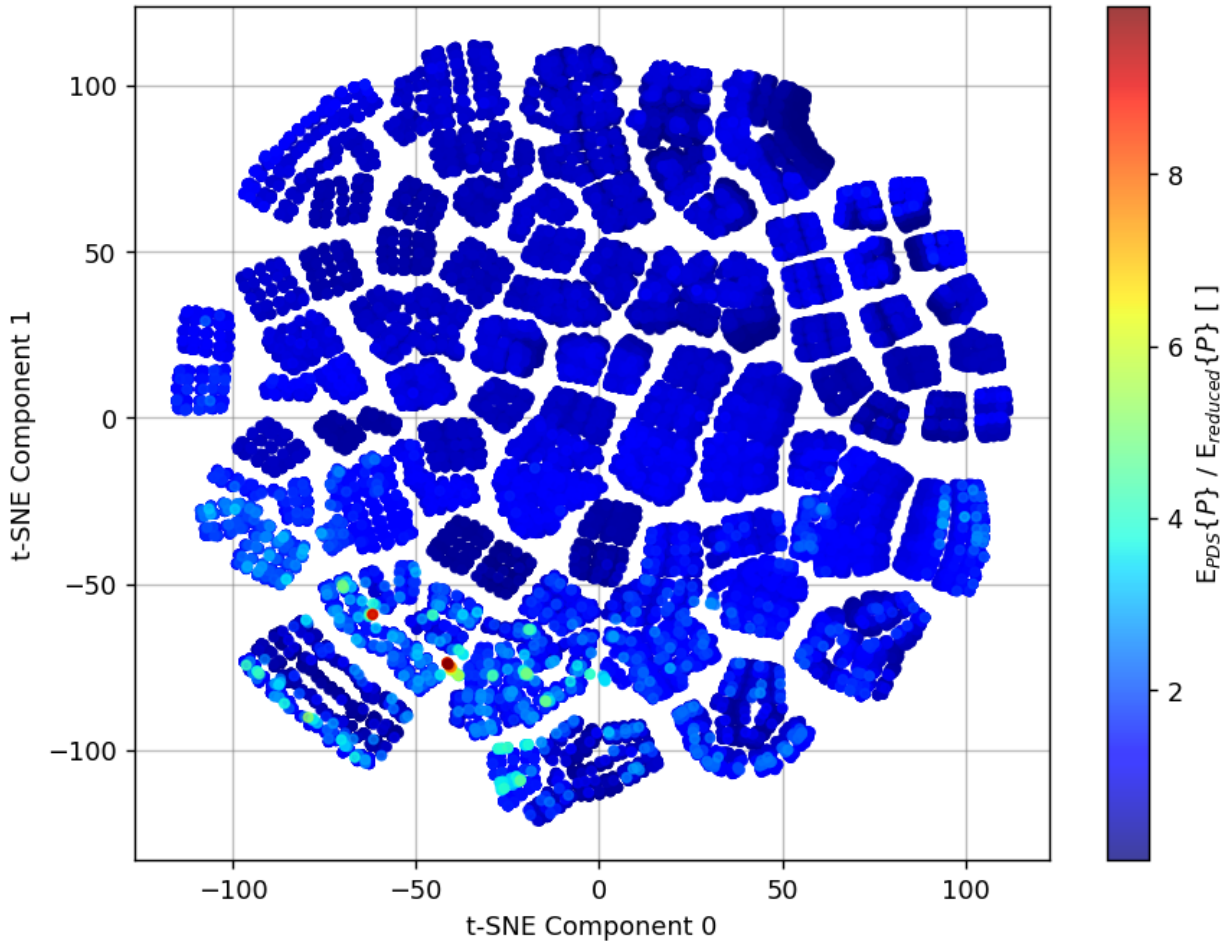

Figure 4.5: *t*-SNE clustering of the merge of the features, dimensionless features, and target ratios. Plotted with "hottest" points visible.

Of note in Figure 4.5 is the fact that the large magnitude ratios are narrowly distributed, with them being spread over only a few clusters (about two). That said, there is evident structure in the extended feature space which can be exploited by the desired perturbation machine.

4.3 Deep Learning

Given the results illustrated in Figures 4.2 - 4.5, it seems that a separable perturbation would be more appropriate in this case. This then suggests a possible way forward; namely

Machine Learning: Train a *perturbation machine* that maps from the extended features (i.e. features + dimensionless features) to appropriate $\mathcal{P}(\cdots)$ values, with the machine being designed as a plug in for (4.3).

As such, the extended feature array for this training exercise is the merge of

`Python/data/feature_array.npy`
and
`Python/data/dimensionless_feature_array.npy`

and the target array is the ratios illustrated in Figure 4.4. These ratios were saved to

`Python/data/target_ratios_array.npy`

4.3.1 Pre-Processing

For implementation details, see

`Python/perturbation_machine_preprocessing.py`

Given the complex nature of the extended feature array in play here, it was decided to train a dense network (namely, a `tensorflow.keras.Sequential`; see [10]) to serve as the perturbation machine; that is to say, an $\mathbb{R}^{10} \to \mathbb{R}$ regressor. In preparation for this, the following pre-processing steps were taken

1. The b, Π_0, Π_1, and Π_3 columns of the extended feature array were $\log_{10}$ scaled (to "linearize" their distribution). In other words, extended features 5, 6, 7, and 9 were replaced by $\log_{10}(b)$, $\log_{10}(\Pi_0)$, $\log_{10}(\Pi_1)$, and $\log_{10}(\Pi_3)$, respectively.

2. Outliers were trimmed from the target ratios array using a 95-percentile cutoff scheme (and as such, 95% of the data was retained). As a result, the corresponding rows of the extended feature array were also dropped. The impact of this is illustrated in Figures 4.6 and 4.7, and the results were saved to

`Python/data/extended_feature_array_trimmed.npy`
and
`Python/data/target_ratios_array_trimmed.npy`

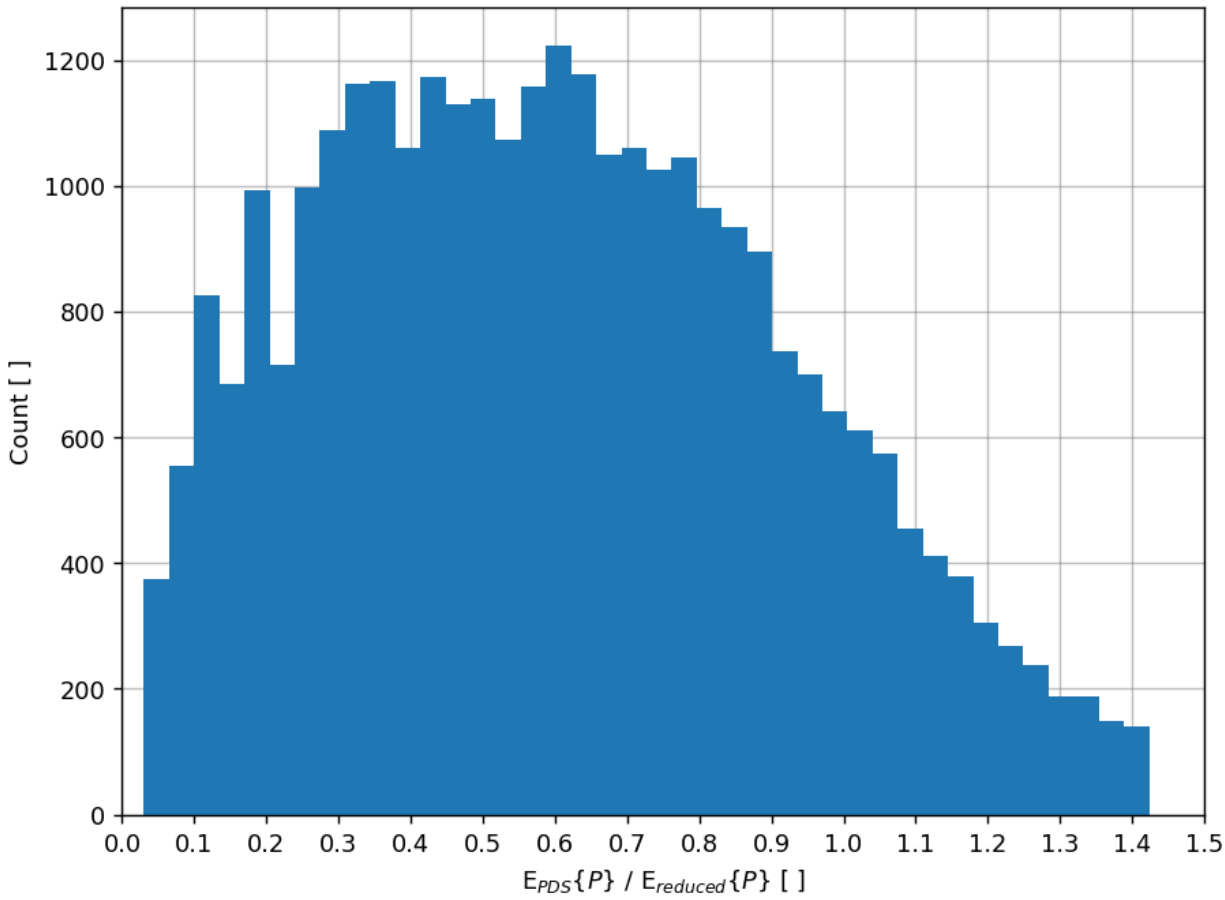

Figure 4.6: Histogram of target ratios $\mathrm{E_{PDS}}\{P\}/\mathrm{E_{reduced}}\{P\}$. Outliers were trimmed using a 95-percentile cutoff.

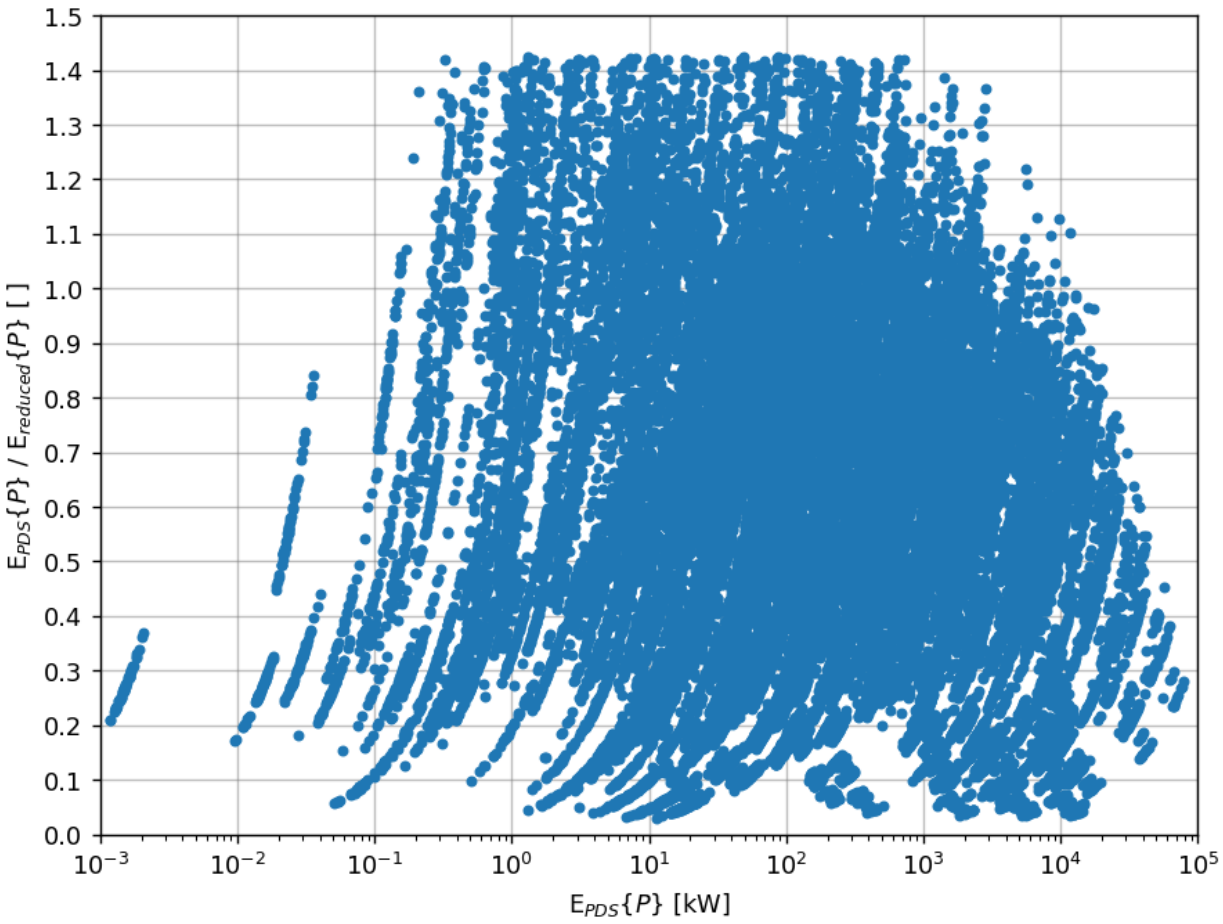

Figure 4.7: Correlation between target ratios $\mathrm{E_{PDS}}\{P\}/\mathrm{E_{reduced}}\{P\}$ and "true" power production $\mathrm{E_{PDS}}\{P\}$ (after outliers were trimmed).

Of note in Figure 4.6 is the continuous and mostly normal nature of the target ratios distribution over the closed interval [0, 1.4]; this is practically ideal for machine learning purposes.

4.3.2 Hyperparameter Optimization

For implementation details, see

`Python/evolve_perturbation_machine_hyperparameters.py`

In order to tune the hyperparameters for the dense network, a co-optimization approach was used as illustrated in Figure 4.8.

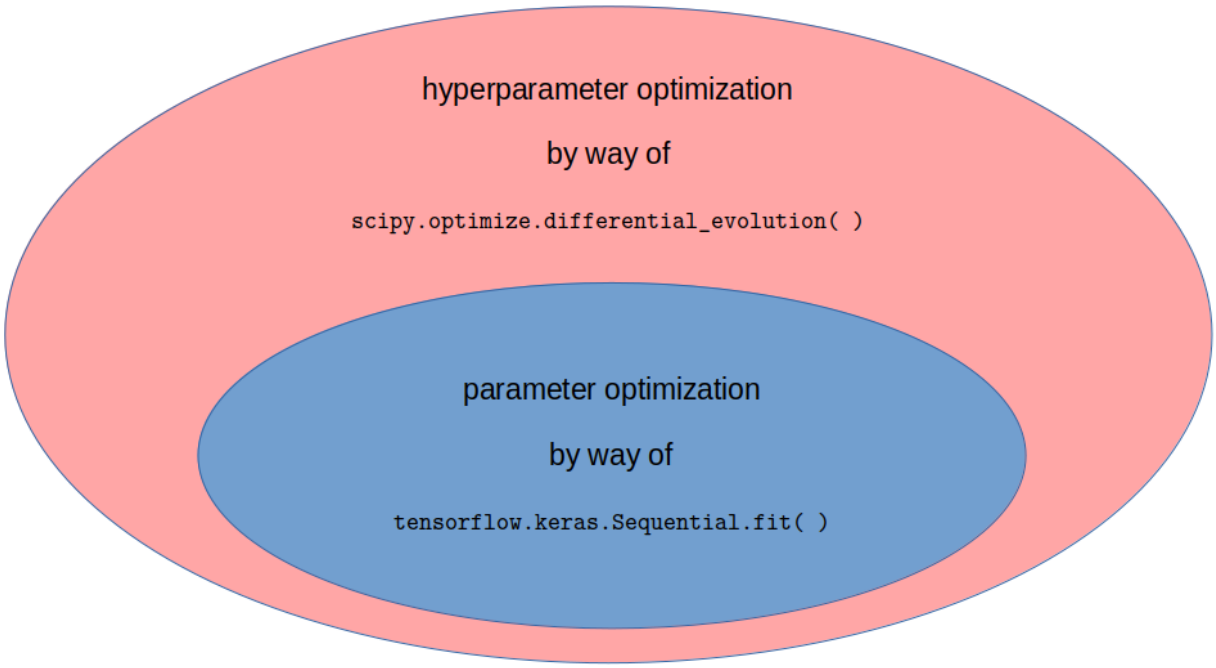

Figure 4.8: Sequential co-optimization approach.

That is, for every candidate set of hyperparameters, the dense network was fit to a random train/test split of the data (i.e., parameter optimization) and then its performance was assessed. The objective of the outer layer of Figure 4.8 was then to maximize the performance of the inner layer dense network by intelligently exploring the hyperparameter space.

For the purpose of this work, the following hyperparameter space was chosen [10]

1. number of hidden layers: up to 6, and this choice was driven by hardware limitations.

2. number of neurons in each hidden layer: $\in \{0, 1, 2, \cdots, 512\}$, and this choice was driven by hardware limitations. Note that if a hidden layer has zero neurons in it, then it is omitted from the dense network (hence *up to* six hidden layers).

3. activation function: $\in$ { `"elu"`, `"gelu"`, `"hard_silu"`, `"leaky_relu"`, `"linear"`, `"relu"`, `"selu"`, `"silu"` }, as per [11]. Note that only linear units were considered in this work, with the intent being to avoid the vanishing gradients problem as much as possible. In addition, network layers were subject to a `MaxNorm` constraint, so as to avoid the exploding gradients problem.

All other hyperparameters were left to take their default values, as defined in [10]. This choice results in a seven dimensional hyperparameter optimization problem. That said, since the hyperparameter space is, in this case, discrete and of mixed type, a derivative-free optimizer is needed to drive the outer layer of the co-optimization. To that end, the `differential_evolution` optimizer (as implemented in `scipy.optimize`) was selected [12], its multiprocessing capabilities were employed, and hyperparameter evolution was carried out

over a sequence of generations of 448 candidates each (and with an evolution real time limit (soft) of 48 hours).

Finally, the objective for the outer layer was defined as a scalar objective (to be minimized) which seeks to balance over-fitting with training and test set performance. That is, if

$$\mu_{\text{train}} = \text{mean}_i\{y_{\text{train},i} - \widehat{y}_{\text{train},i}\}$$
$$\sigma_{\text{train}} = \text{std}_i\{y_{\text{train},i} - \widehat{y}_{\text{train},i}\}$$
$$\text{median}_{\text{train}} = \text{median}_i\{y_{\text{train},i} - \widehat{y}_{\text{train},i}\}$$

$$\mu_{\text{test}} = \text{mean}_i\{y_{\text{test},i} - \widehat{y}_{\text{test},i}\}$$
$$\sigma_{\text{test}} = \text{std}_i\{y_{\text{test},i} - \widehat{y}_{\text{test},i}\}$$
$$\text{median}_{\text{test}} = \text{median}_i\{y_{\text{test},i} - \widehat{y}_{\text{test},i}\}$$

where the y_i are target data and the $\widehat{y}_i$ are predictions, then the scalar objective H is given by

$$\begin{aligned} H = |\mu_{\text{test}} - \mu_{\text{train}}| + |\sigma_{\text{test}} - \sigma_{\text{train}}| + |\text{median}_{\text{test}} - \text{median}_{\text{train}}| + \\ |\mu_{\text{train}}| + \sigma_{\text{train}} + |\text{median}_{\text{train}}| + \\ |\mu_{\text{test}}| + \sigma_{\text{test}} + |\text{median}_{\text{test}}| \end{aligned} \quad (4.4)$$

Note that, by definition, $H \geq 0$. Furthermore, observe that each term has the following influence on H

1. $|\mu_{\text{test}} - \mu_{\text{train}}|$: Deviation between the training and test set μ values causes H to increase. This penalizes over-fitting.

2. $|\sigma_{\text{test}} - \sigma_{\text{train}}|$: Deviation between the training and test set σ values causes H to increase. This penalizes over-fitting.

3. $|\text{median}_{\text{test}} - \text{median}_{\text{train}}|$: Deviation between the training and test set medians causes H to increase. This penalizes over-fitting.

4. $|\mu_{(\,)}|$: Deviation in absolute μ values away from their minimum of zero causes H to increase. This penalizes poor training and test set performance.

5. $\sigma_{(\,)}$: Deviation in σ values away from their minimum of zero causes H to increase. This penalizes poor training and test set performance.

6. $|\text{median}_{(\,)}|$: Deviation in absolute medians away from their minimum of zero causes H to increase. This penalizes poor training and test set performance.

Following termination of the hyperparameter evolution (which terminated on running out of memory after about 24 hours[3]), the optimal hyperparameters that emerged were

1. hidden layers: $[381, 191, 466, 131, 197, 470]$
2. activation: `"relu"`

These optimal hyperparameters were saved to

`Python/data/perturbation_machine_hyperparams.npy`

for later use. A summary of the resulting model architecture is illustrated in Figure 4.9

Model: "sequential"

Layer (type)	Output Shape	Param #
hidden_layer_1 (Dense)	(None, 381)	4,191
hidden_layer_2 (Dense)	(None, 191)	72,962
hidden_layer_3 (Dense)	(None, 466)	89,472
hidden_layer_4 (Dense)	(None, 131)	61,177
hidden_layer_5 (Dense)	(None, 197)	26,004
hidden_layer_6 (Dense)	(None, 470)	93,060
output_layer (Dense)	(None, 1)	471

Total params: 1,042,013 (3.97 MB)
Trainable params: 347,337 (1.32 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 694,676 (2.65 MB)

Figure 4.9: Perturbation machine model architecture under the evolved hyperparameters.

It therefore seems that the hyperparameter evolution was attracted towards a network architecture that might be loosely described as an encoder [381, 191] feeding an autoencoder [466, 131, 197, 470].

[3]During the course of the evolution, it was noted that building a large number of `tensorflow.keras` models in a loop (even over multiple processes) apparently remains leaky, even under version 2.16 (the version used in this work). The maximum memory footprint of this evolution was huge; in excess of 64 GB. This has been previously noted elsewhere, for example

`https://stackoverflow.com/questions/76527878/memory-leak-in-tensorflow`
and
`https://github.com/tensorflow/tensorflow/issues/44711`

Perhaps time to consider switching to `PyTorch` ... see

`https://thenextweb.com/news/why-tensorflow-for-python-is-dying-a-slow-death`

Over the course of evolution, the hyperparameter objective evolved as illustrated in Figure 4.10.

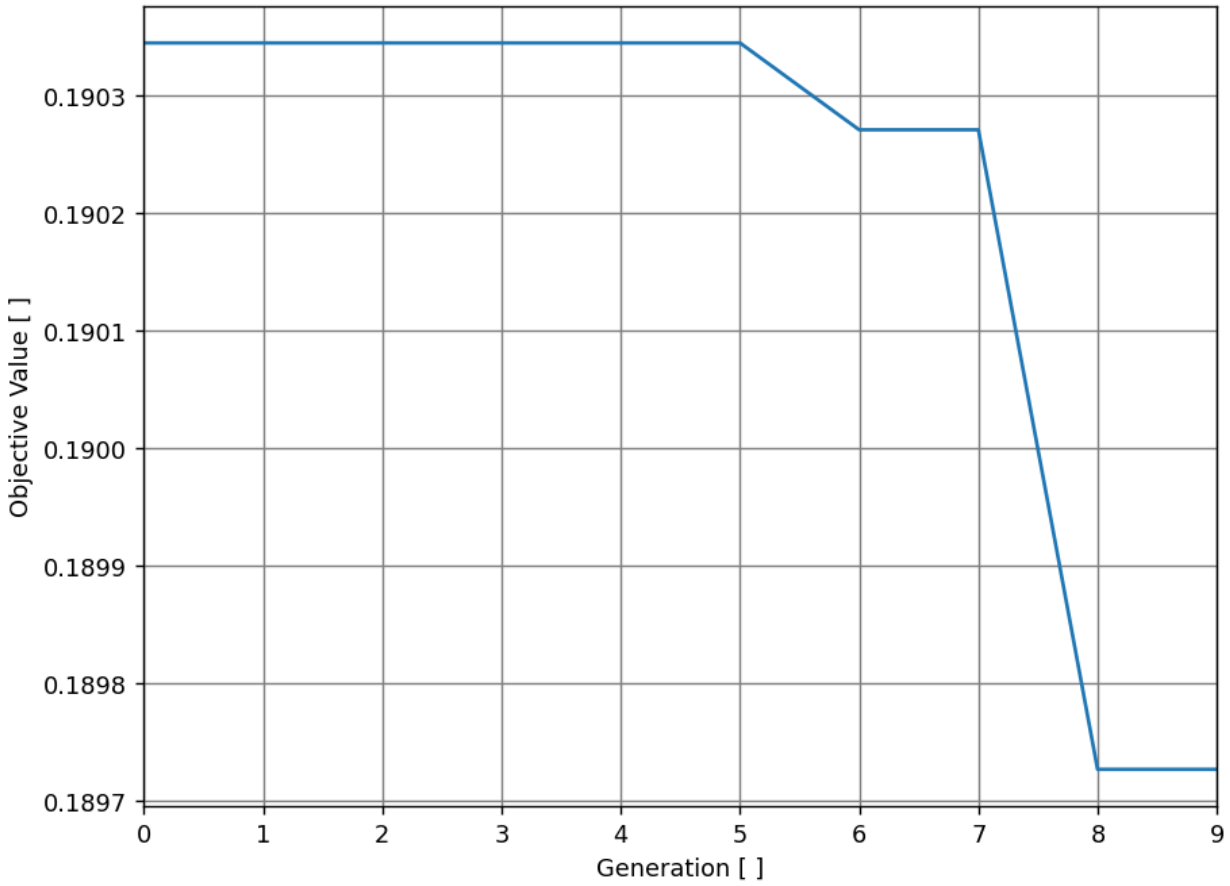

Figure 4.10: Evolution in the perturbation machine hyperparameter objective H.

That is, `differential_evolution` was able to make two major improvements to the modelling performance over the course of 10 generations. However, these result should be taken with a grain of salt since the evolution terminated on running out of memory (rather than halting on a convergence criteria).

4.3.3 Training

For implementation details, see

`Python/train_perturbation_machine.py`

The resulting perturbation machine was saved to

`Python/data/perturbation_machine.keras`

During the course of training, the training and test set performances varied over the training epochs as illustrated in Figure 4.11.

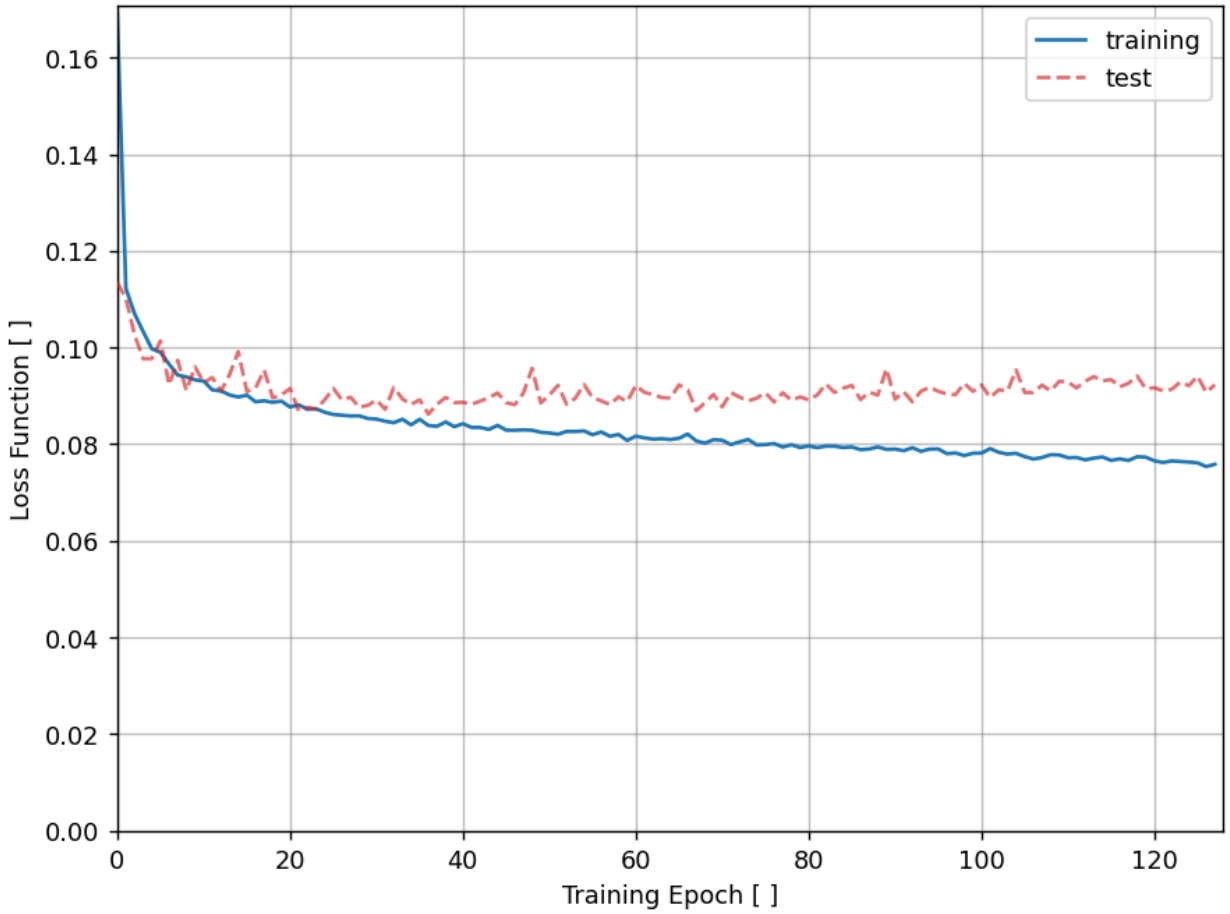

Figure 4.11: Training and test set performance of the perturbation machine over the training epochs.

Of note in Figure 4.11 is the fact that the training set performance smoothly approaches an apparent performance asymptote (loss) of a little less than 0.08. Additionally, the test set performance closely tracks the training set performance for about 20 epochs, and then begins to display over-fitting after about 40 epochs; this indicates that the hyperparameter optimization is working as expected, although results are not quite as impressive as hoped. Note that in this work, the DNN parameters associated with the best *test* set performance were retained for later application.

4.4 Performance

4.4.1 Target Ratios

For implementation details, see

`Python/train_perturbation_machine.py`

After evolution of the hyperparameters, the resulting perturbation machine had the following test set performance metrics

$$\mu_{\text{test}} = 0.00977$$
$$\sigma_{\text{test}} = 0.09525$$
$$\text{median}_{\text{test}} = 0.00249$$

From this, it follows that

1. The test set error of the perturbation machine is in the closed interval

$$[\mu_{\text{test}} - 2\sigma_{\text{test}}, \mu_{\text{test}} + 2\sigma_{\text{test}}] = [-0.1807, 0.2003]$$

   19 times out of 20.

2. The perturbation machine has a slight tendency to under-predict target ratios.

Further illustration of perturbation machine performance is shown in Figures 4.12 - 4.14.

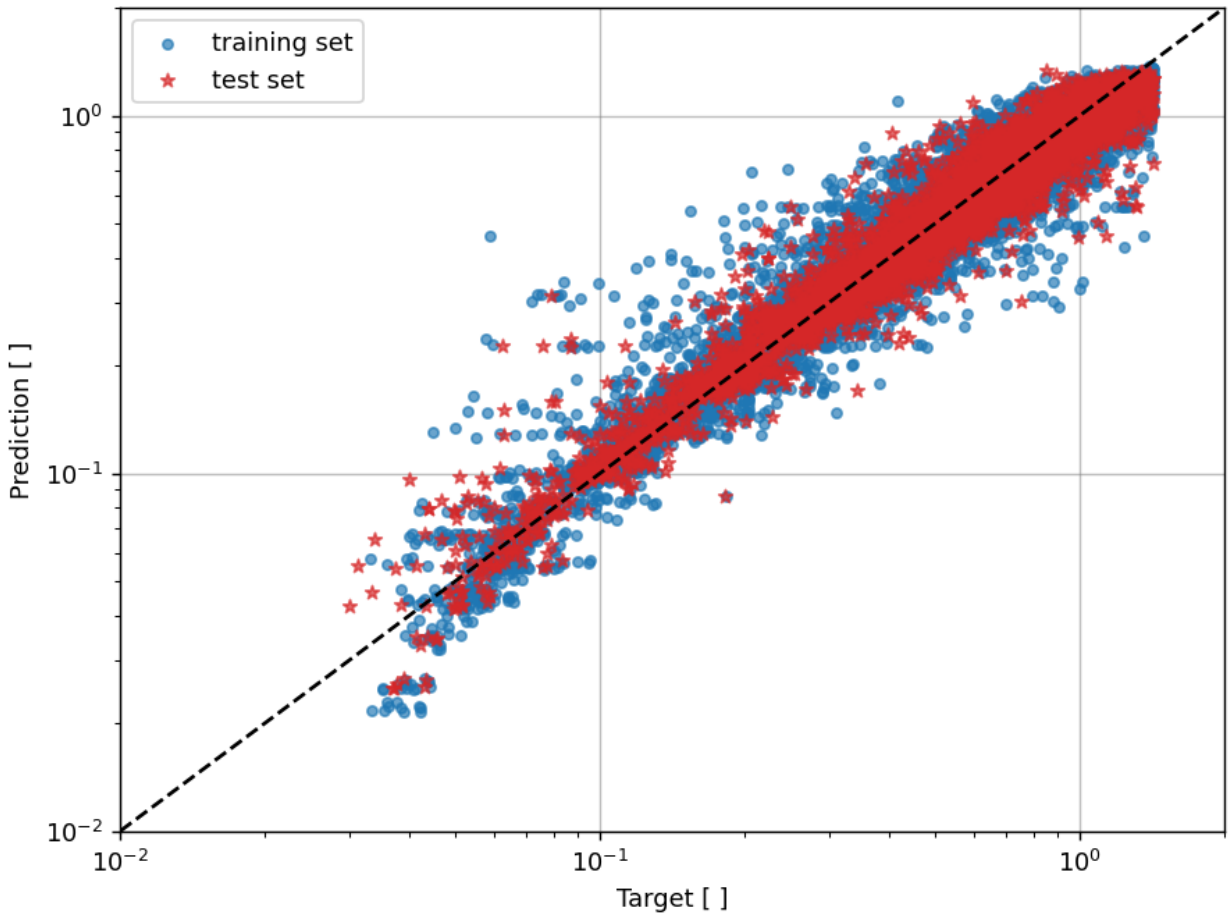

Figure 4.12: Perturbation machine performance: training/test correlation.

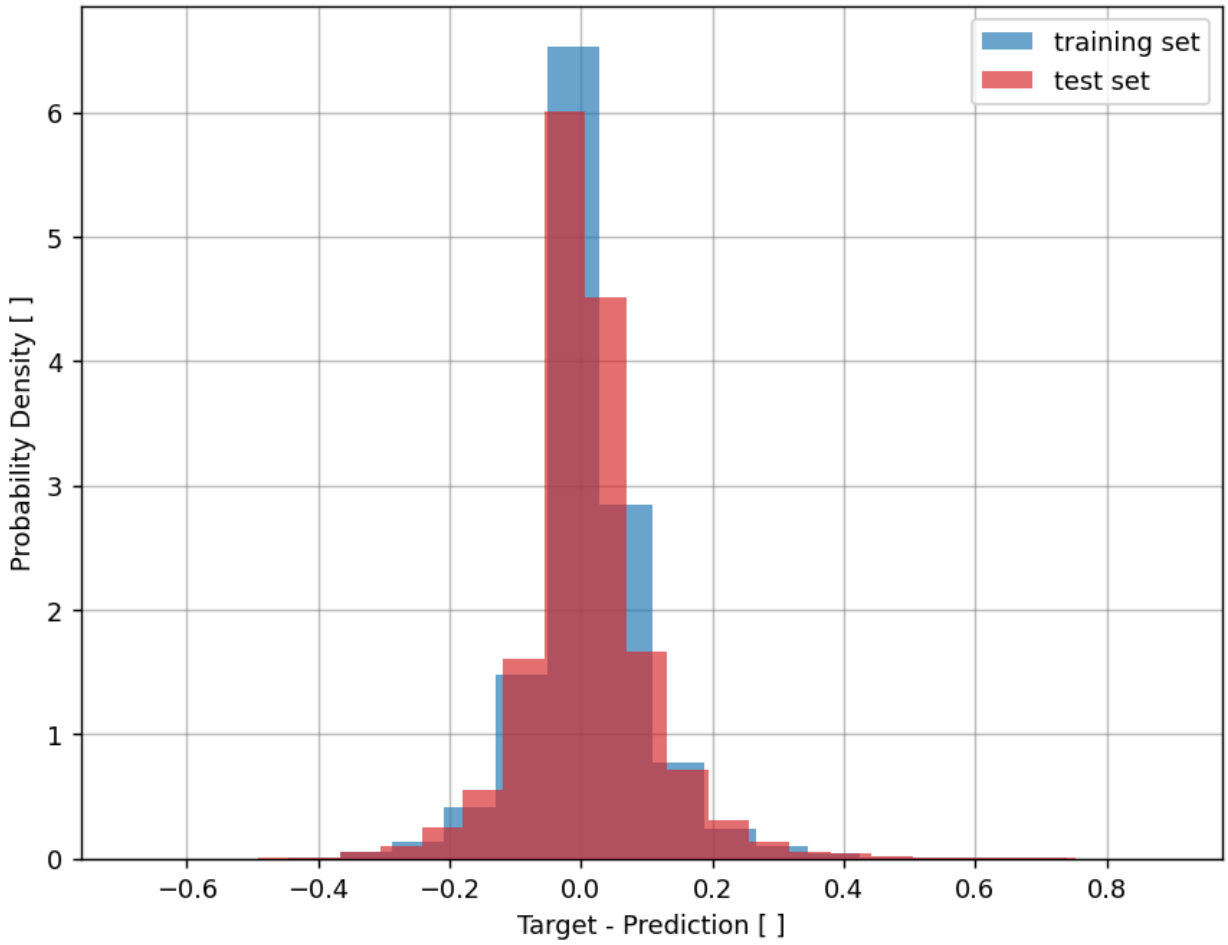

Figure 4.13: Perturbation machine performance: training/test distribution.

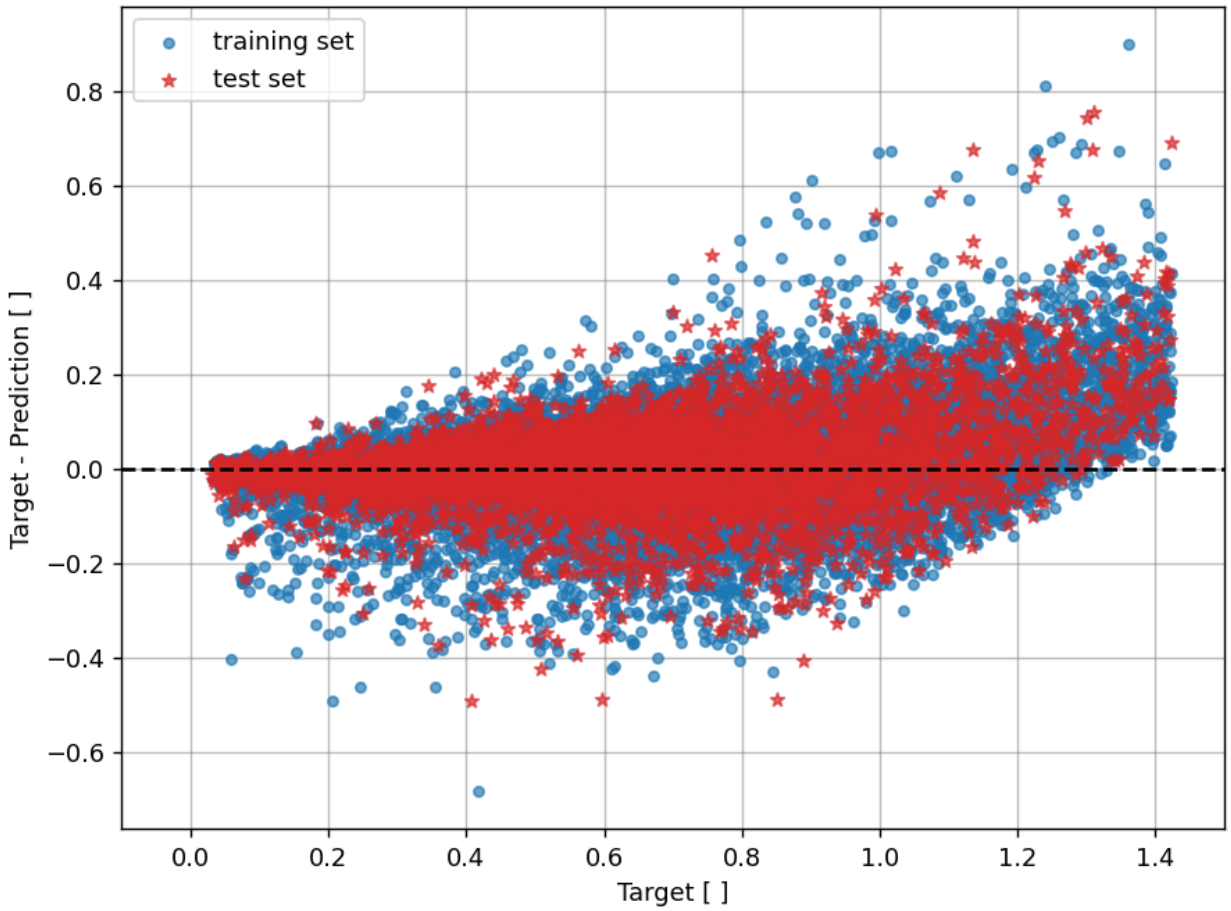

Figure 4.14: Perturbation machine performance: training/test residuals.

Of note in Figures 4.12 - 4.14 is evidence of good model generality as well as reasonably good performance in a majority of cases.

4.4.2 Expected Power Output

For implementation details, see

`Python/test_perturbation_machine.py`

Applying the perturbation machine as part of (4.3) yields the following performance metrics (for all data)

$$\mu_{\text{power}} = 47.199 \text{ kW}$$
$$\sigma_{\text{power}} = 488.189 \text{ kW}$$
$$\text{median}_{\text{power}} = 0.0853 \text{ kW}$$

and from this, it follows that

1. The expected power output prediction error of the perturbation machine is in the closed interval

$$[\mu_{\text{power}} - 2\sigma_{\text{power}}, \mu_{\text{power}} + 2\sigma_{\text{power}}] = [-929, 1024] \text{ kW}$$

532 19 times out of 20.

533 2. The perturbation machine has a slight tendency to under-predict expected power out-
534 put.

535 Further illustration of perturbation machine performance is shown in Figures 4.15 - 4.17.

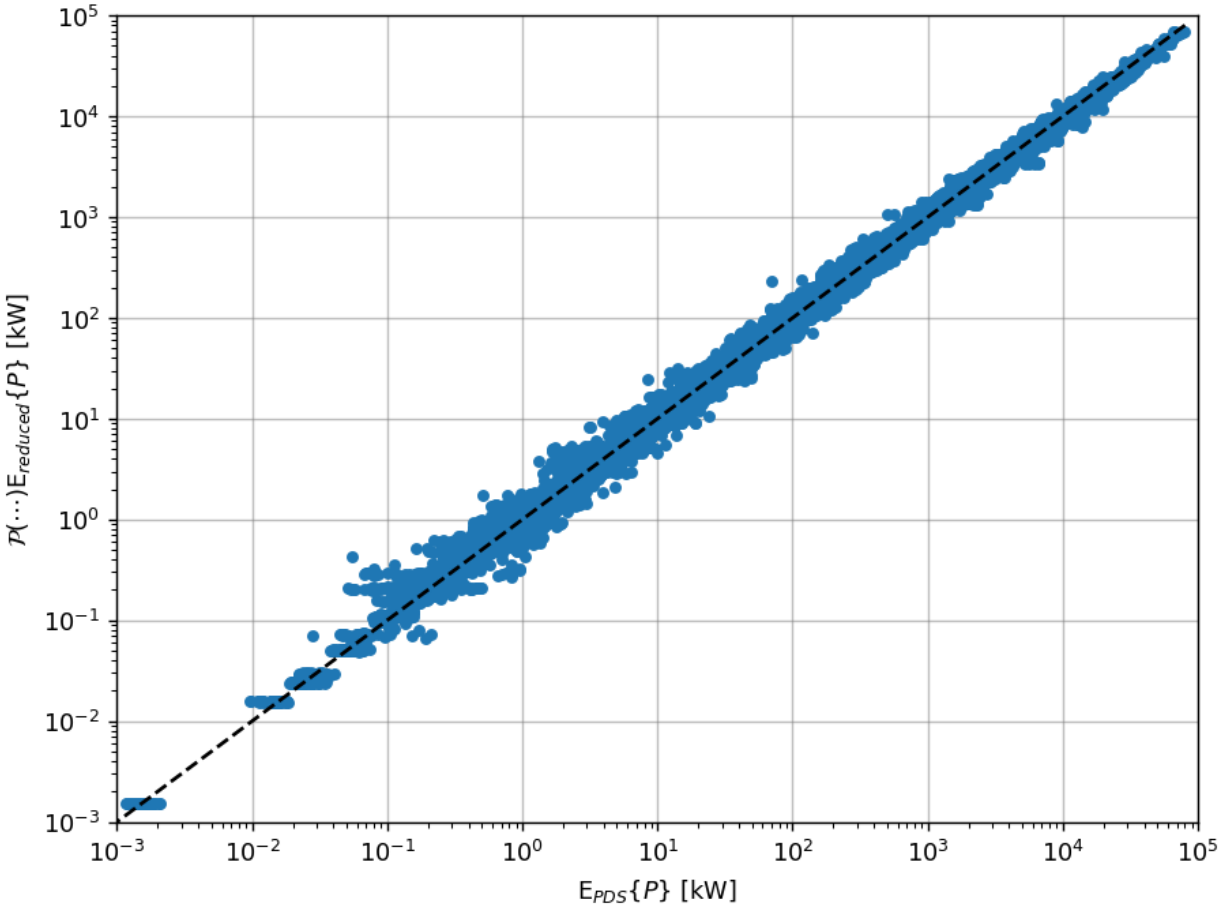

Figure 4.15: Perturbation machine power performance: correlation.

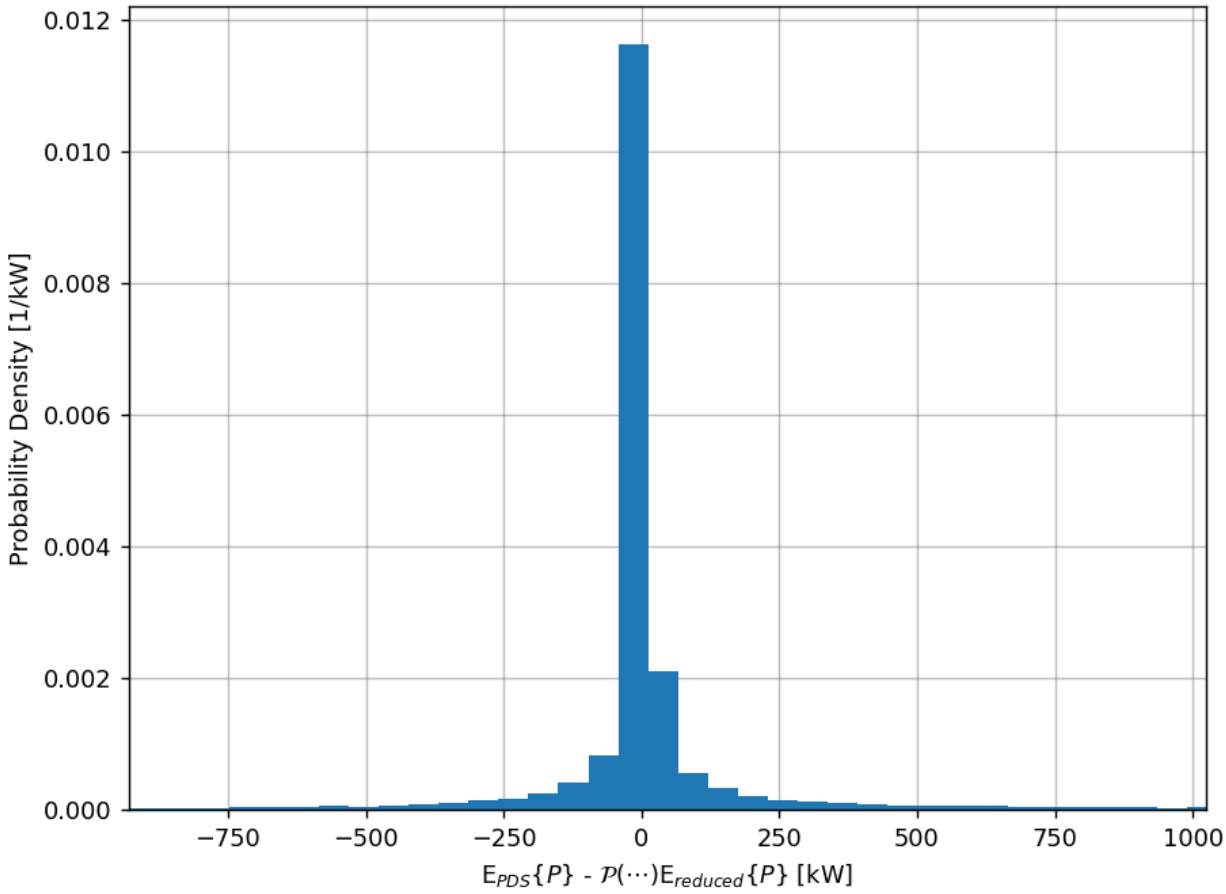

Figure 4.16: Perturbation machine power performance: distribution.

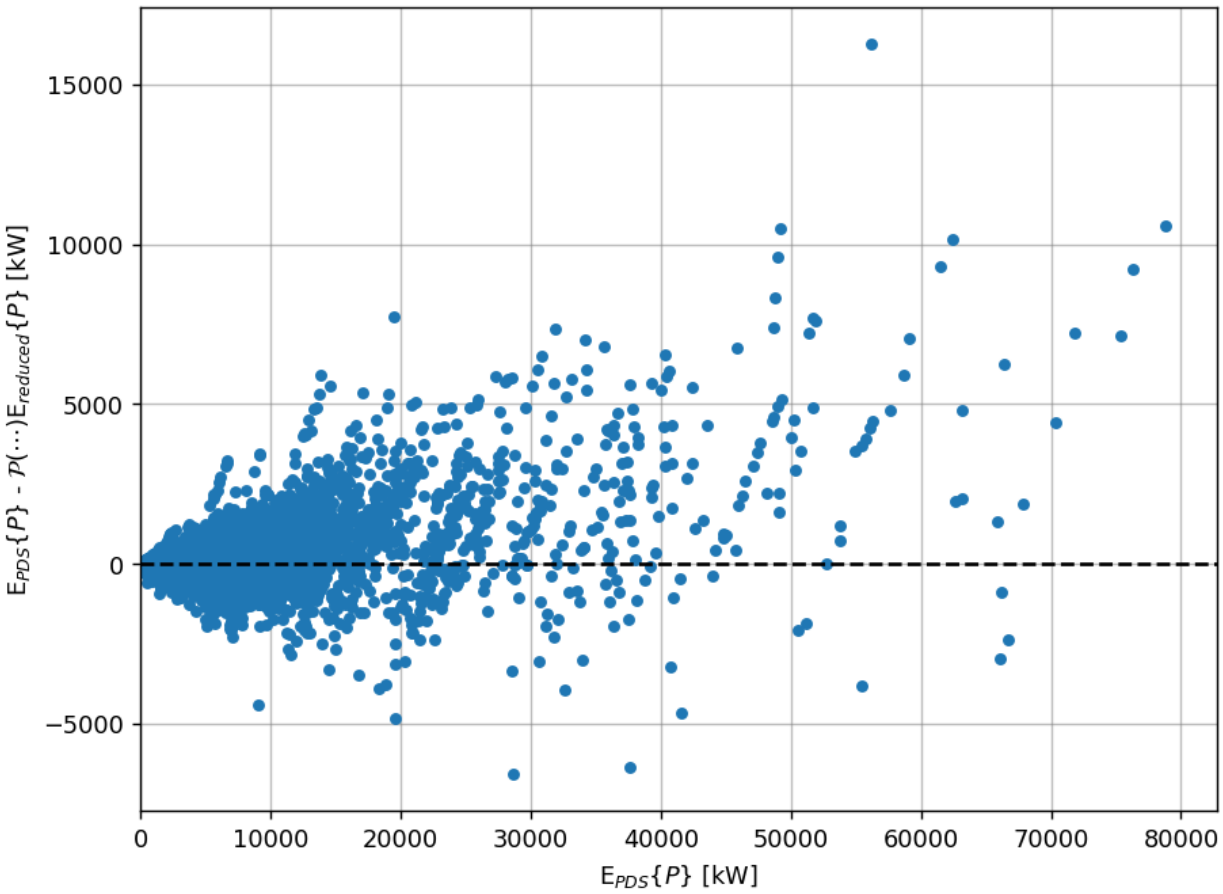

Figure 4.17: Perturbation machine power performance: residuals.

Of note in Figures 4.15 - 4.17 is the fact that the expected power output prediction errors tend to have magnitudes appropriate in scale relative to the underlying "true" value (see Figures 4.15 and 4.16, in particular).

Considering the percent error in expected power output prediction, namely

$$\text{percent error} = \frac{\mathrm{E}_{\mathrm{PDS}}\{P\} - \mathcal{P}(\cdots)\mathrm{E}_{\mathrm{reduced}}\{P\}}{\mathrm{E}_{\mathrm{PDS}}\{P\}}$$

yields the following performance metrics (for all data)

$$\mu_{\text{percent}} = -0.0071$$
$$\sigma_{\text{percent}} = 0.1753$$
$$\text{median}_{\text{percent}} = 0.0090$$

From this it follows that the percent error in expected power output prediction is in the closed interval

$$[\mu_{\text{percent}} - 2\sigma_{\text{percent}}, \mu_{\text{percent}} + 2\sigma_{\text{percent}}] = [-0.3577, 0.3435]$$

and so is like ±35%, 19 times out of 20. This performance is further illustrated in Figures 4.18 and 4.19.

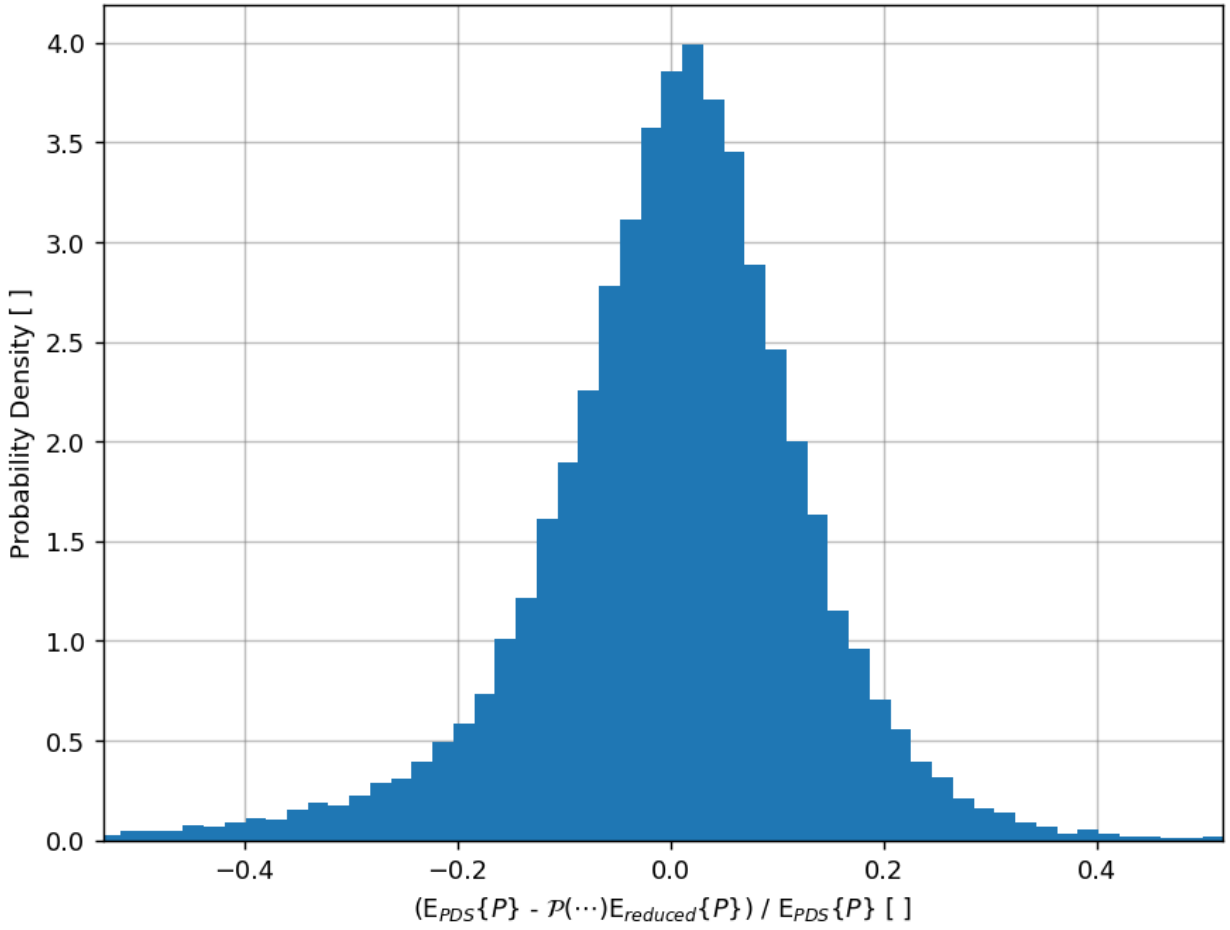

Figure 4.18: Perturbation machine percent error performance: distribution.

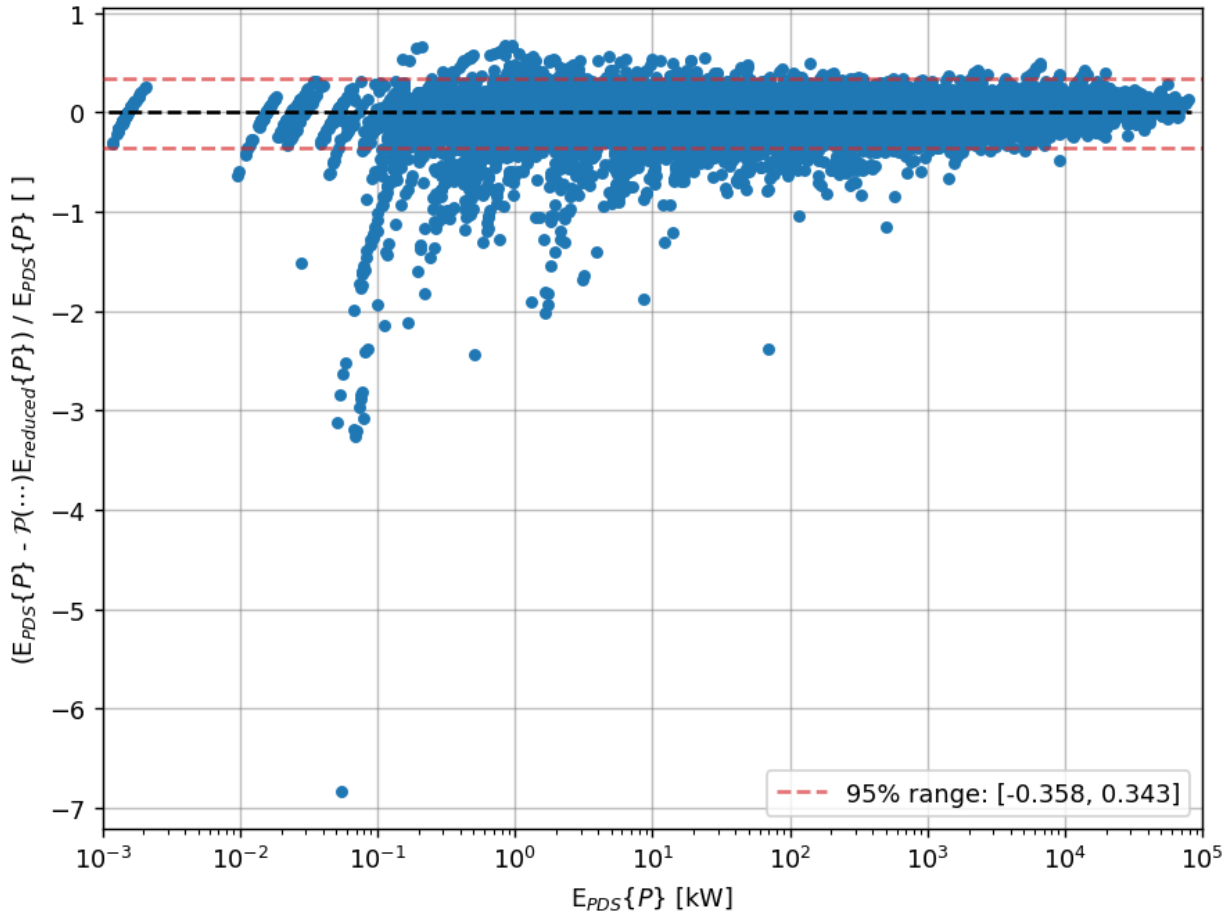

Figure 4.19: Perturbation machine percent error performance: residuals. 95% range shown.

Figure 4.19 is particularly interesting, as it indicates that the larger percent errors are generally concentrated in the lower power range, less than 1 MW (thus limiting the magnitude of the absolute prediction errors). This indicates that the perturbation machine, as presently trained, has practical utility. This is especially true when considering differences in runtime performance. Generating the PDS simulation results took 10 days running in parallel on a 12 core 13$^{\text{th}}$ Gen Intel i7-13700 machine. Contrast this with the predictions made using (4.3), which took about 90 minutes to generate running in serial on a single core of the same machine. Therefore, (4.3) yields about a 1,900× speed up at the cost of a roughly ±35% prediction error, 19 times out of 20.[4]

[4]Since, if run in serial on a single core, the PDS simulations would have taken more like 120 days to finish.

5 Some Example Applications

5.1 WEC Design Optimization

For implementation details, see

`Python/example_application_optimization.py`

The perturbation machine developed in the course of this work was set to the task of facilitating a design optimization of a heave constrained point absorber like that illustrated in Figure 2.1. For the purpose of this example application, the follow features were fixed

$$Z = 30 \text{ m}$$
$$H_s = 3 \text{ m}$$
$$T_p = 10 \text{ s}$$
$$d = 6 \text{ m}$$
$$h = 2.5 \text{ m}$$
$$k = 0$$

The features varied in the course of optimization were the float outer diameter D and the PTO damping b, and the objective was to maximize expected power output (as computed using (4.3)). For reference, Airy waves with periods around 10 seconds have wavelengths in the range 150 m - 175 m (depending on water depth).

First, a 64×64 point grid search was performed over the following feature intervals

$$D \in [20, 120] \text{ m}$$
$$b \in \left[10^5, 10^8\right] \text{ N.s/m}$$

in order to get an initial mapping of the design space. The result of this is illustrated in Figure 5.1.

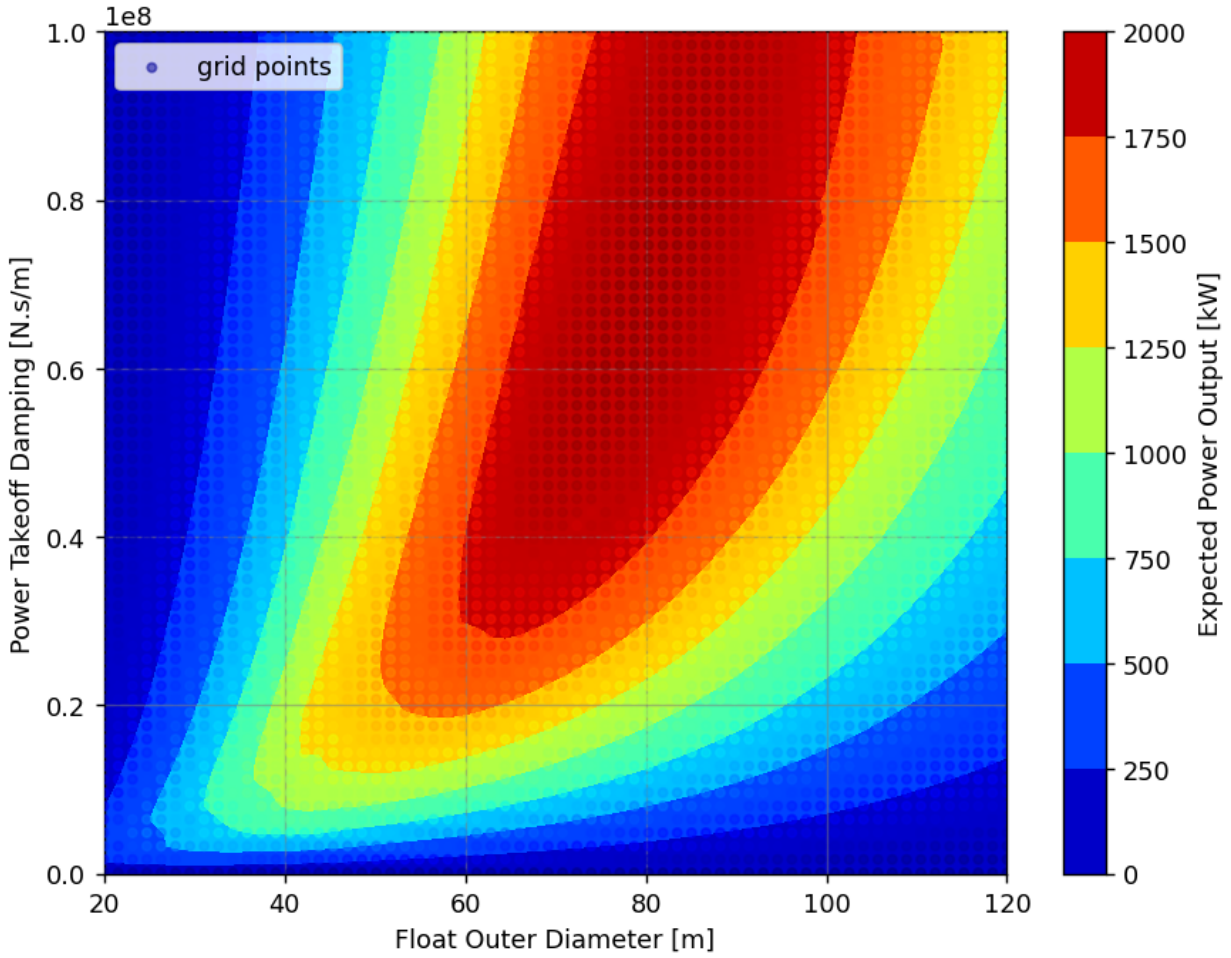

Figure 5.1: Results of initial 64×64 point grid search of the WEC design space. A filled contour plot was generated by interpolating between the points.

Of note in Figure 5.1 is the fact that the design space is exhibiting non-linear behaviour as one might expect. Furthermore, there appears to be a clearly optimal design contained somewhere in the red region of the plot.

From the points assessed in the initial 64×64 point grid search, the best design found was $D = 80.32$ m and $b = 71.46 \times 10^6$ N.s/m for which $\mathrm{E}\{P\} \cong 1,995$ kW. For reference, this corresponds to a deep water hydrodynamic efficiency of

$$\eta_{\text{hydro}} = \frac{1,995 \text{ kW}}{(44.15 \text{ kW/m})(80.32 \text{ m})} \cong 0.563$$

or about 56% (perhaps rather optimistic, but ultimately less than 100% which is encouraging to see; also, this is not really a deep water case).

Design optimization was then carried out using `scipy.optimize.dual_annealing` together with (4.3) and the candidate design as an initial guess [13]. The resulting optimal design was $D = 81.02$ m and $b = 72.29 \times 10^6$ N.s/m for which $\mathrm{E}\{P\} \cong 1,996$ kW. For reference, this corresponds to a deep water hydrodynamic efficiency of

$$\eta_{\text{hydro}} = \frac{1,996 \text{ kW}}{(44.15 \text{ kW/m})(81.02 \text{ m})} \cong 0.558$$

or about 56%. The points explored by the optimizer are illustrated in Figure 5.2.

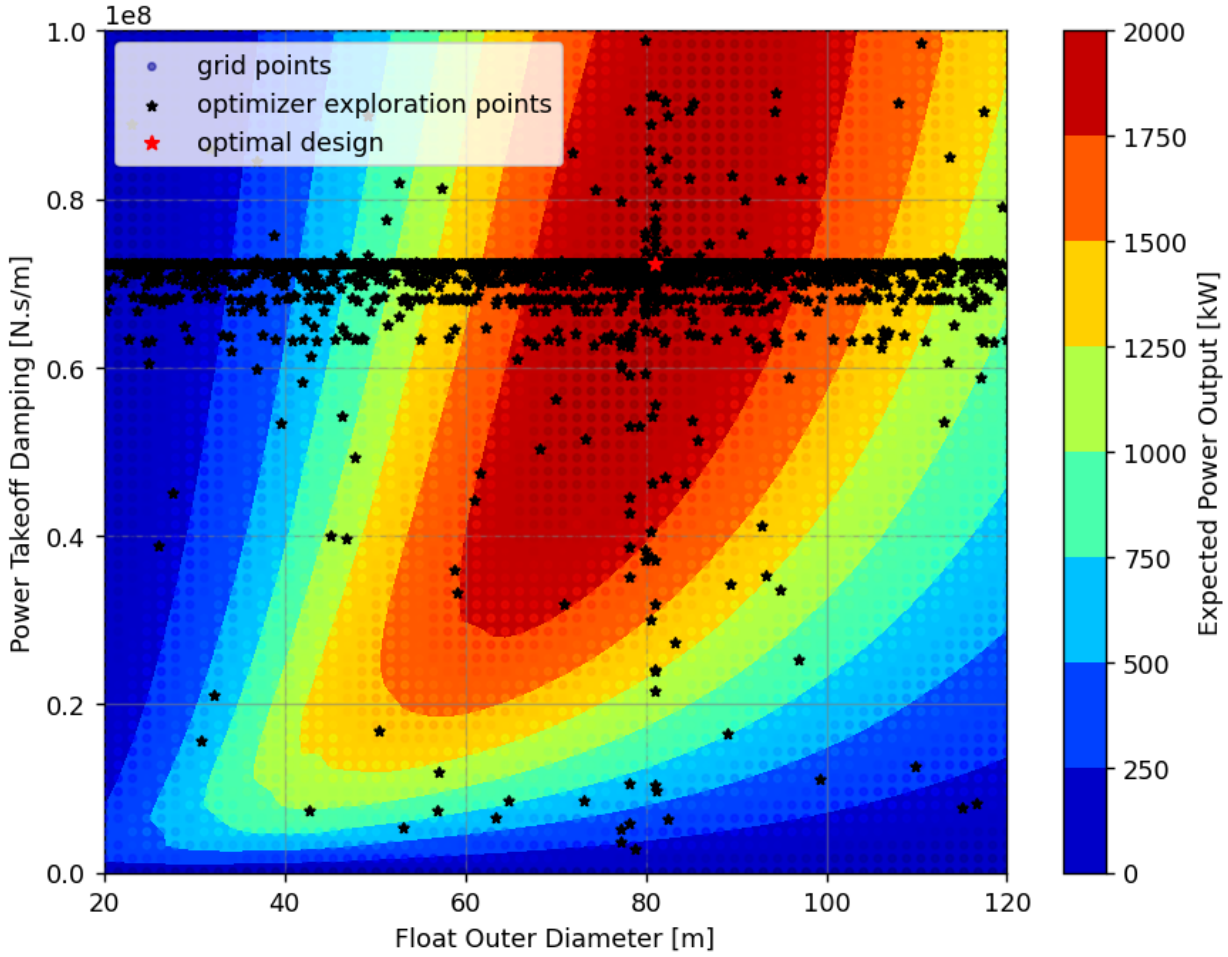

Figure 5.2: Results of initial 64×64 point grid search of the WEC design space. A filled contour plot was generated by interpolating between the points. Optimizer exploration points shown.

Therefore, one might conclude that the design region around the "crosshairs" of the exploration points is a region warranting a closer look using more detailed (and expensive) modelling tools.

Finally, it is worth noting that this entire optimization example was run in serial (i.e., no parallel processing was used) on a 13$^{\text{th}}$ Gen Intel i7-13700 machine. In all, 13,920 candidate WEC designs were assessed in this case (4,096 in the original grid search, plus another 9,824 in the design optimization). Total run time was 1,721 seconds (a little less than 30 minutes),

and this serves to highlight the magnitude of speed up that can be achieved by using machine learning models like this.[1]

5.2 WEC Performance Mapping

For implementation details, see

`Python/example_application_performance.py`

The optimal WEC design previously found ($D = 81.02$ m and $b = 72.29 \times 10^6$ N.s/m) was retained for this example, along with the values for Z, d, h, and k used previously. The goal was then to generate a performance map[2] for this WEC over a range of (H_s, T_p) pairs. As such, a 64×64 point grid search was performed over the following feature intervals

$$H_s \in [0, 8] \text{ m}$$
$$T_p \in [5, 16] \text{ s}$$

in order to generate the performance map. Note that the wave breaking heuristic defined by (3.6) was enforced by assuming that expected power output is zero whenever $H_s > \{H_s\}_{\text{break}}$. Likewise expected power output is assumed to be zero whenever $H_s = 0$.

The result of the grid search is illustrated in Figure 5.3.

[1] Performing the same design optimization using PDS simulations (running in parallel) would, evidently, take about four days.

[2] Think dense performance matrix presented as a filled contour plot.

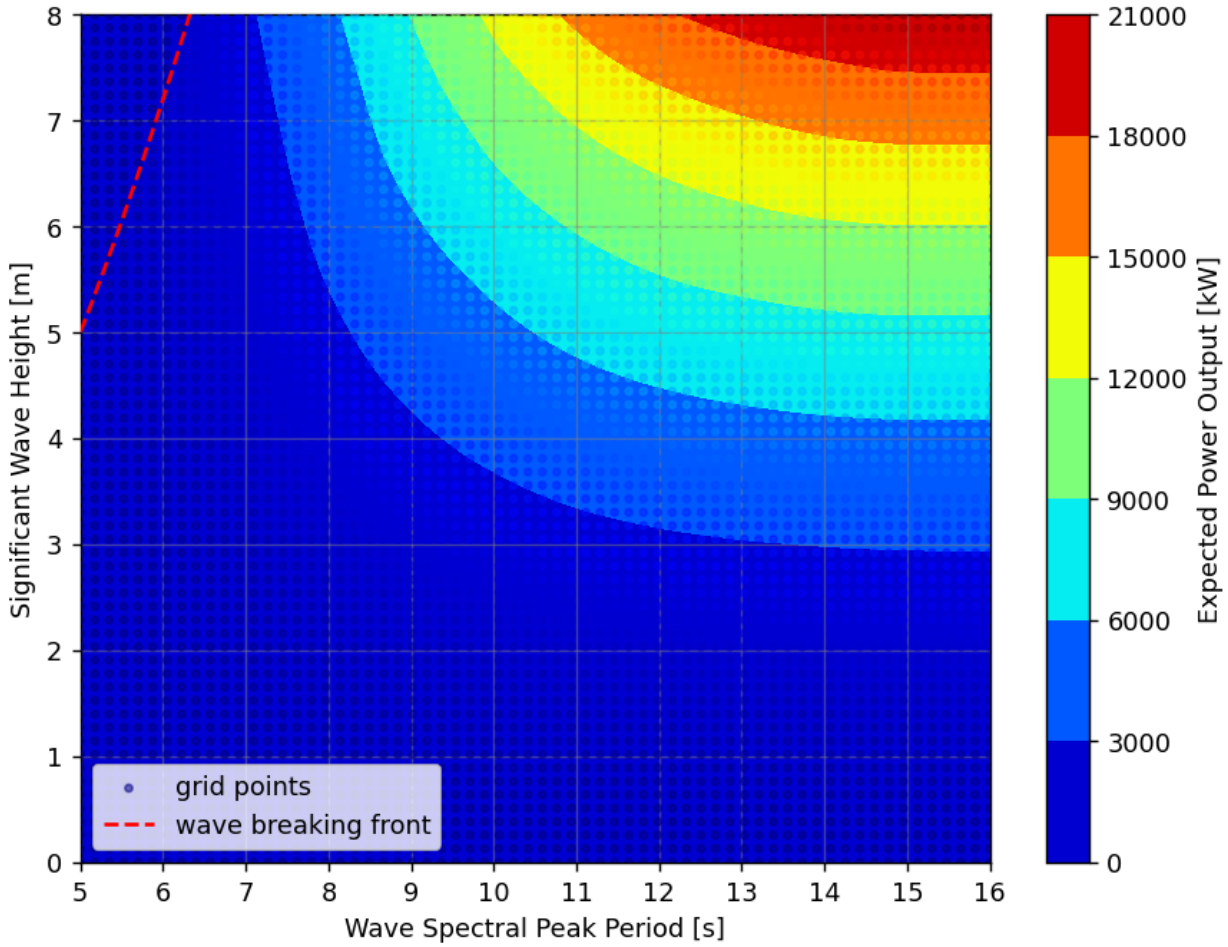

Figure 5.3: WEC performance map, which is the result of a 64×64 point grid search. A filled contour plot was generated by interpolating between the points.

Of note in Figure 5.3 is the fact that WEC performance tends to increase exponentially as H_s increases (as one might expect, since $J \propto H_s^2$).

Finally, it is worth nothing that this entire performance mapping example was run in serial (i.e., no parallel processing was used) on a 13$^{\text{th}}$ Gen Intel i7-13700 machine. Total run time was 504 seconds (less than nine minutes), and this serves to highlight the magnitude of speed up that can be achieved by using machine learning models like this.[3]

[3]Generating the same performance map using PDS simulations (running in parallel) would, evidently, take somewhere in the range of 28 - 30 hours.

6 Future Work

In order to build the perturbation machine demonstrated in this work, Proteus DS was used to simulate WEC dynamics in order to determine "true" values for expected power output. However, even these simulations were simplified for the sake of run time, and so the steps laid out in this work could be repeated using improved simulation data which includes

1. More accurate float meshing (including the centreline hole).
2. Diffraction and radiation effects where applicable.
3. Fluid memory effects where applicable.

Of course, this requires coordination between meshing software, boundary element method (BEM) software, and Proteus DS, and so is largely a modelling pipeline project. Needless to say, it would also be a computationally expensive undertaking, and so deploying on advanced hardware would probably be necessary. However, the potential speed up relative to those simulations could be orders of magnitude more than what is observed here (and possibly for about the same prediction error, depending on the quality of machine learning model constructed).

References

[1] N. Bogolyubov, "Perturbation Theory," 2024. [Online]. Available: https://encyclopediaofmath.org/index.php?title=Perturbation_theory

[2] Y. Zhang, Y. Zhao, W. Sun, and J. Li, "Ocean wave energy converters: Technical principle, device realization, and performance evaluation," *Renewable and Sustainable Energy Reviews*, vol. 141, 2021.

[3] B. Guo, T. Wang, S. Jin, S. Duan, K. Yang, and Y. Zhao, "A Review of Point Absorber Wave Energy Converters," *Journal of Marine Science and Engineering*, vol. 10, 2022. [Online]. Available: https://www.mdpi.com/2077-1312/10/10/1534

[4] L. Holthuijsen, *Waves in Oceanic and Coastal Waters*. Cambridge University Press, 2010, iSBN-13: 978-1-13-946252-5.

[5] DSA Ocean, "Proteus DS: Purpose-Built Marine Dynamic Analysis Software," 2024. [Online]. Available: https://proteusds.com/

[6] S. Beatty, "Self-Reacting Point Absorber Wave Energy Converters," Ph.D. dissertation, University of Victoria, 2015. [Online]. Available: https://www.uvic.ca/research/centres/iesvic/assets/docs/dissertations/Dissertation-Beatty1.pdf

[7] B. Robertson, H. Bailey, M. Leary, and B. Buckham, "A methodology for architecture agnostic and time flexible representations of wave energy converter performance," *Applied Energy*, vol. 287, 2021.

[8] J. Logan, *Applied Mathematics*, 4th ed. John Wiley & Sons, 2013, iSBN-13: 978-1-118-47580-5.

[9] scikit learn, "`sklearn.manifold.TSNE`," 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

[10] tensorflow, "`tensorflow.keras.Sequential`," 2024. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/Sequential

[11] ——, "`tensorflow.keras.activations`," 2024. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/activations

[12] scipy, "`scipy.optimize.differential_evolution`," 2024. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html

[13] ——, "`scipy.optimize.dual_annealing`," 2024. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.dual_annealing.html

A Feature Plots

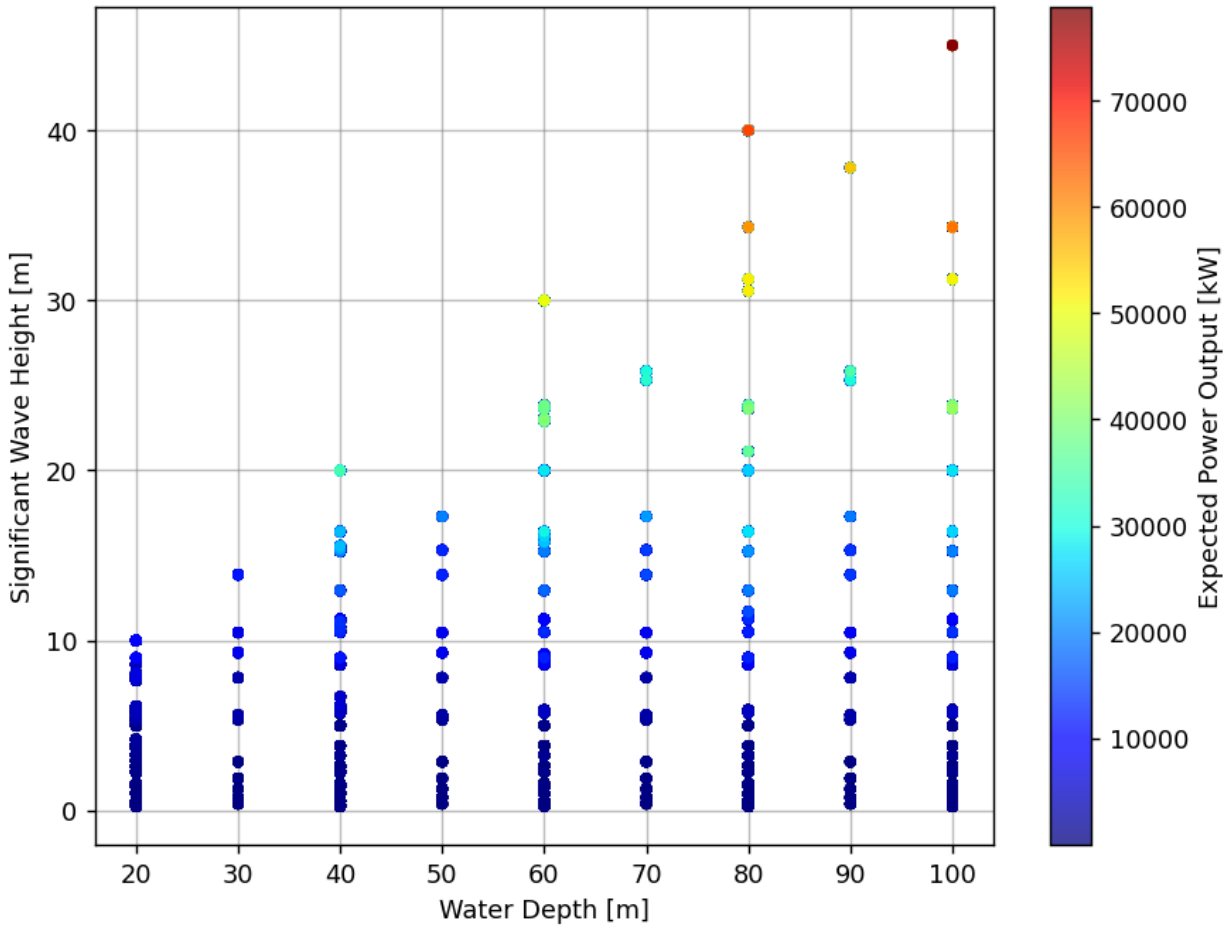

Figure A.1: PDS simulation results mining. Feature 1 vs Feature 0. Plotted with "hottest" points visible.

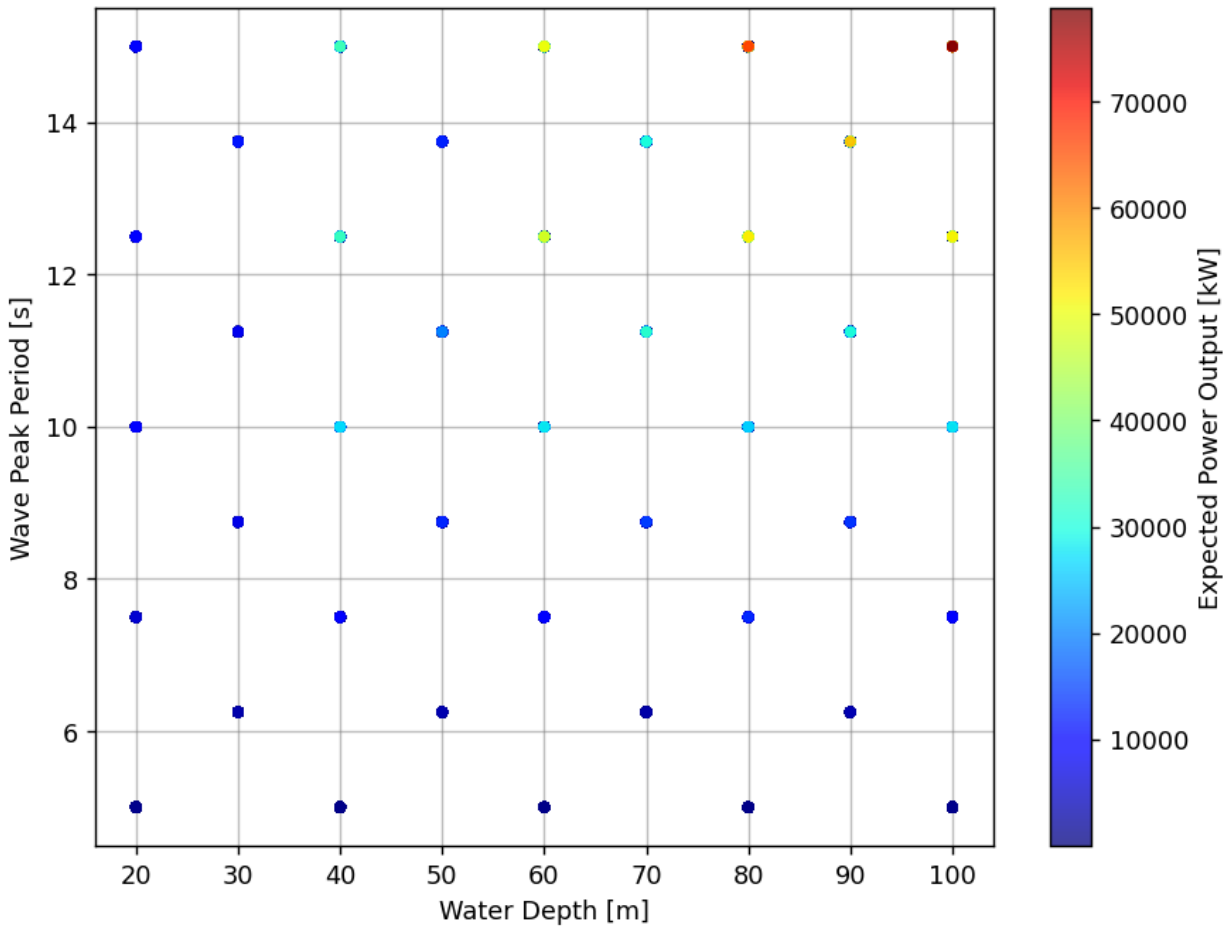

Figure A.2: PDS simulation results mining. Feature 2 vs Feature 0. Plotted with "hottest" points visible.

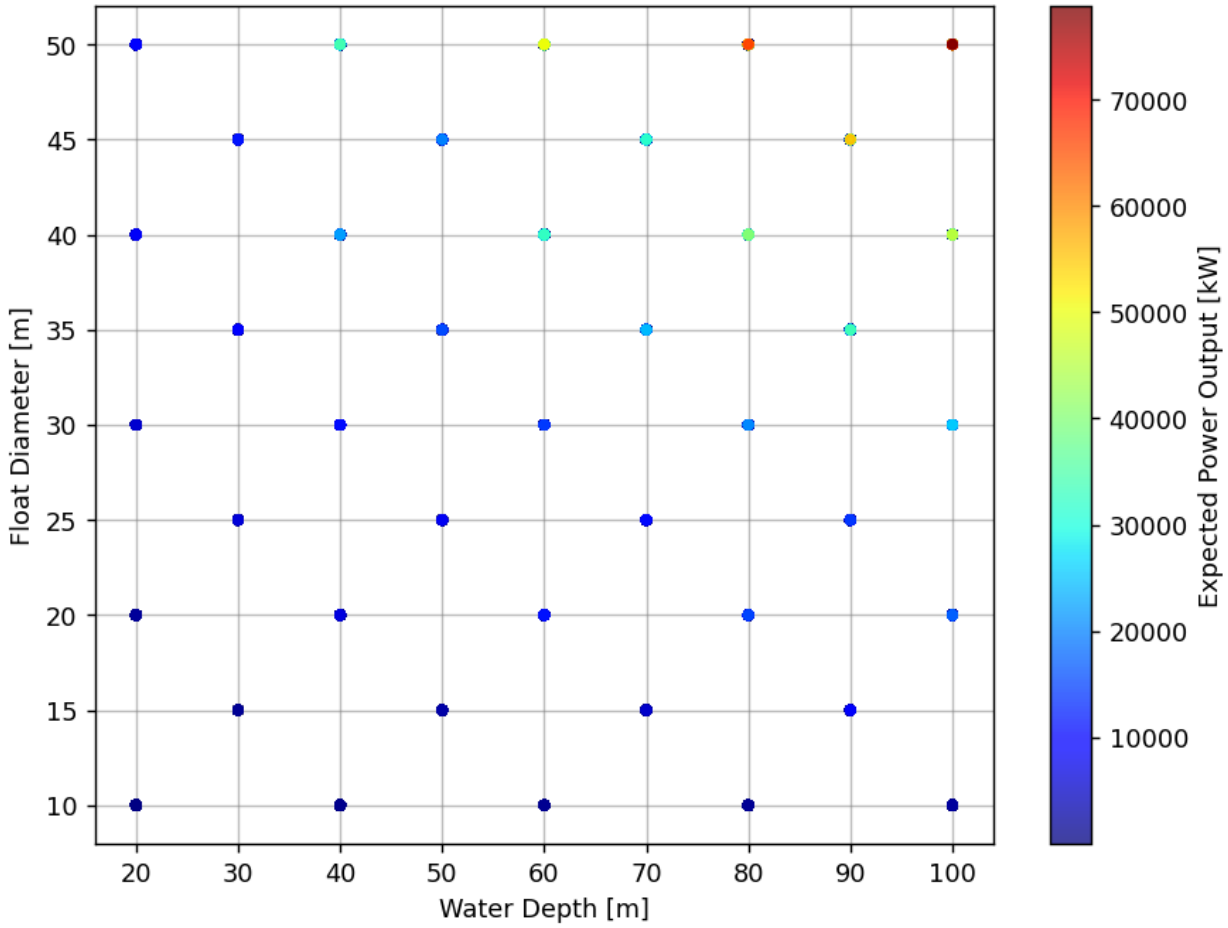

Figure A.3: PDS simulation results mining. Feature 3 vs Feature 0 Plotted with "hottest" points visible..

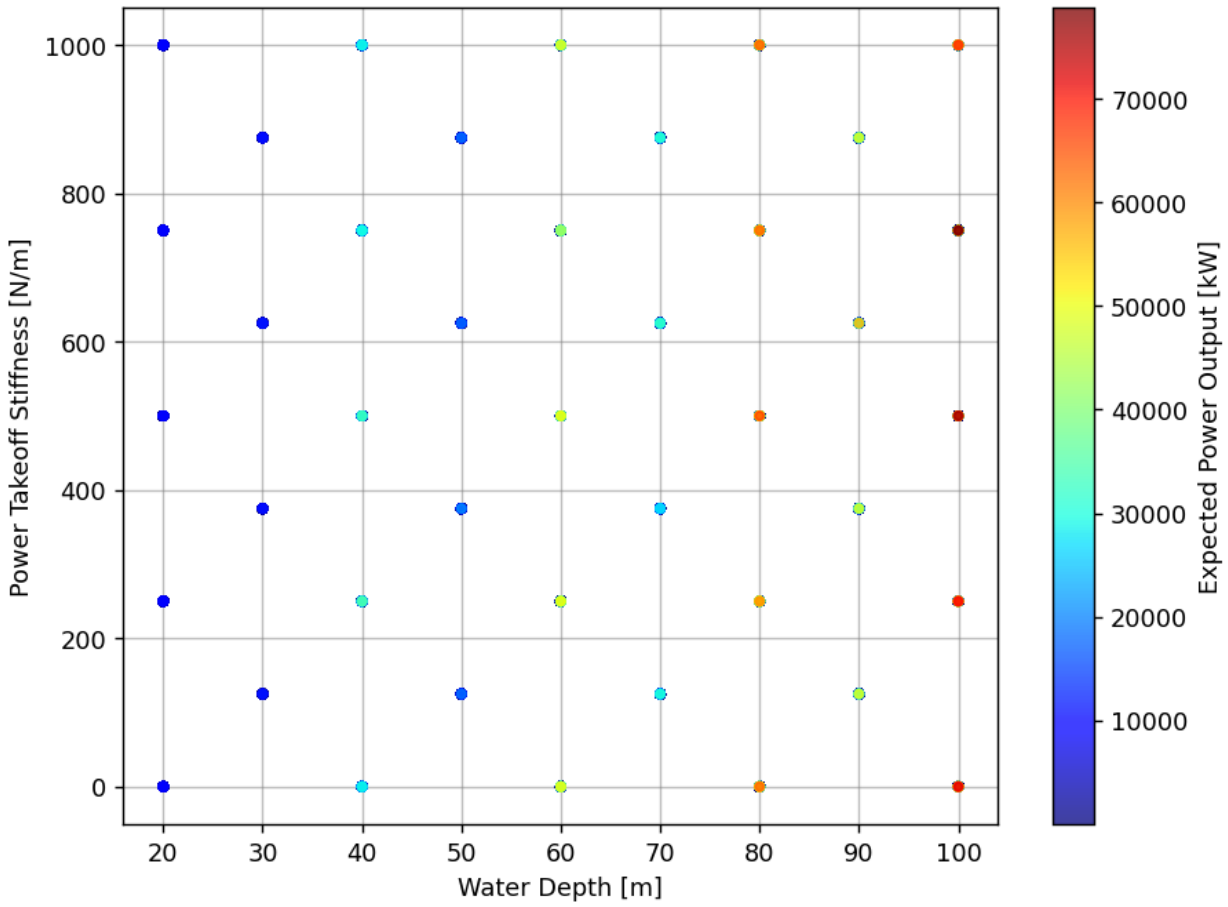

Figure A.4: PDS simulation results mining. Feature 4 vs Feature 0. Plotted with "hottest" points visible.

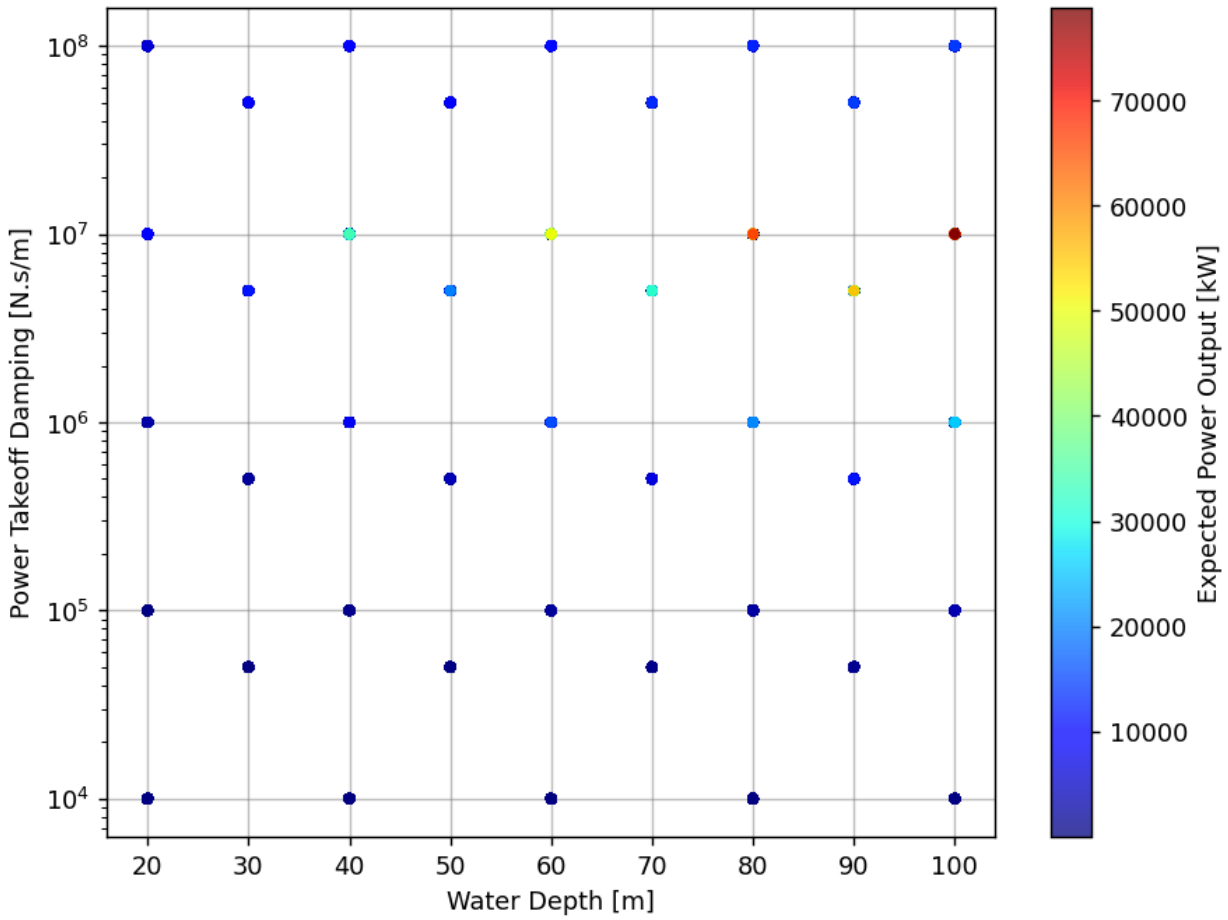

Figure A.5: PDS simulation results mining. Feature 5 vs Feature 0. Plotted with "hottest" points visible.

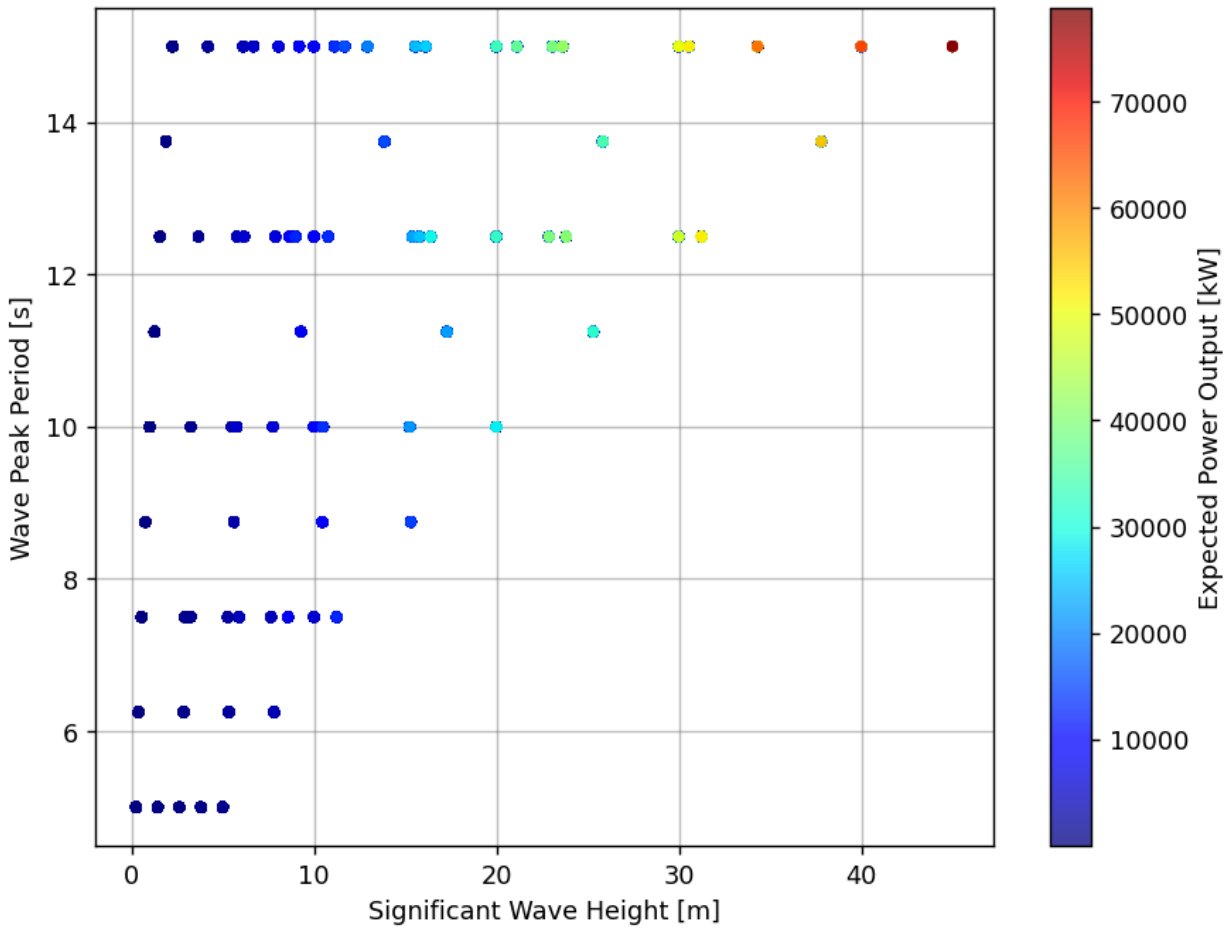

Figure A.6: PDS simulation results mining. Feature 2 vs Feature 1. Plotted with "hottest" points visible.

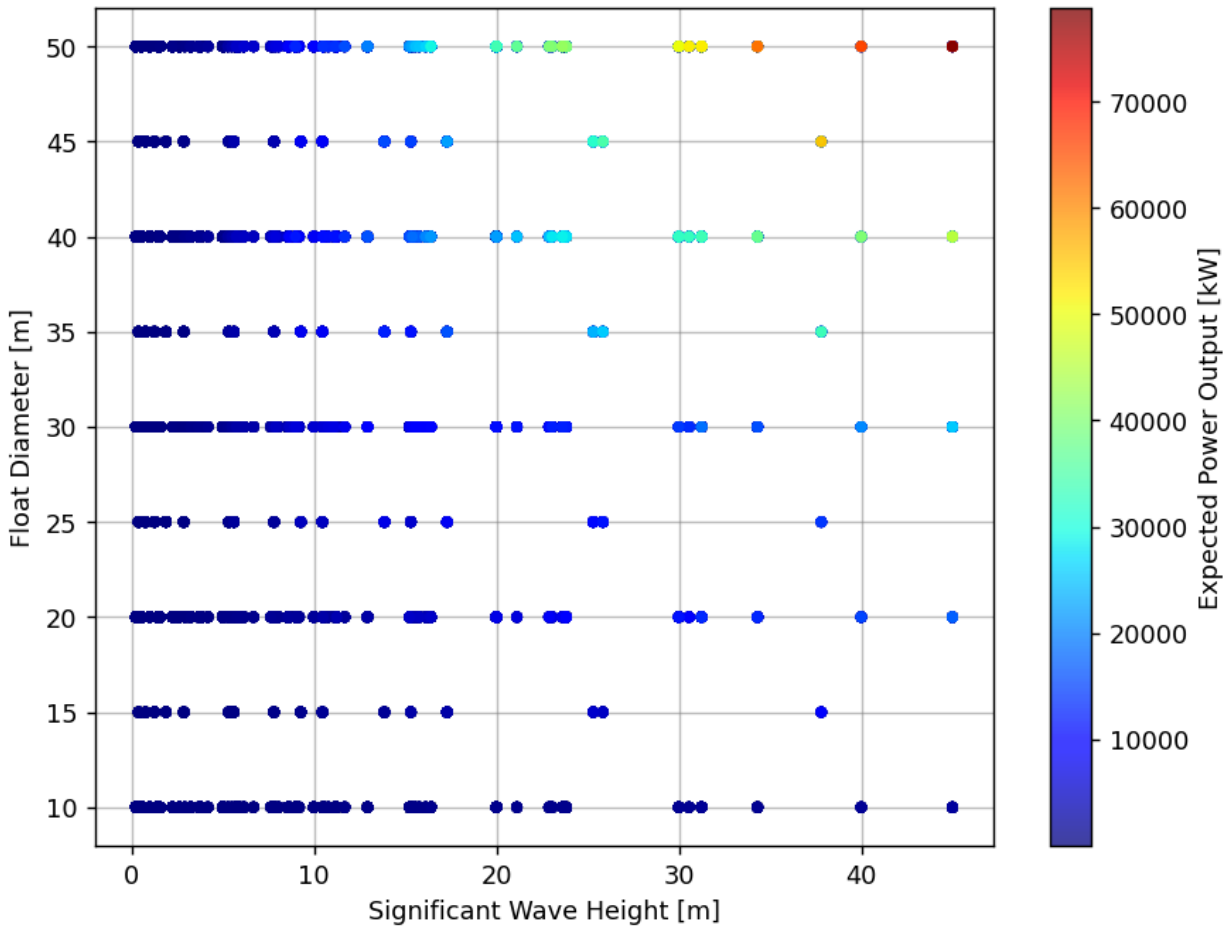

Figure A.7: PDS simulation results mining. Feature 3 vs Feature 1. Plotted with "hottest" points visible.

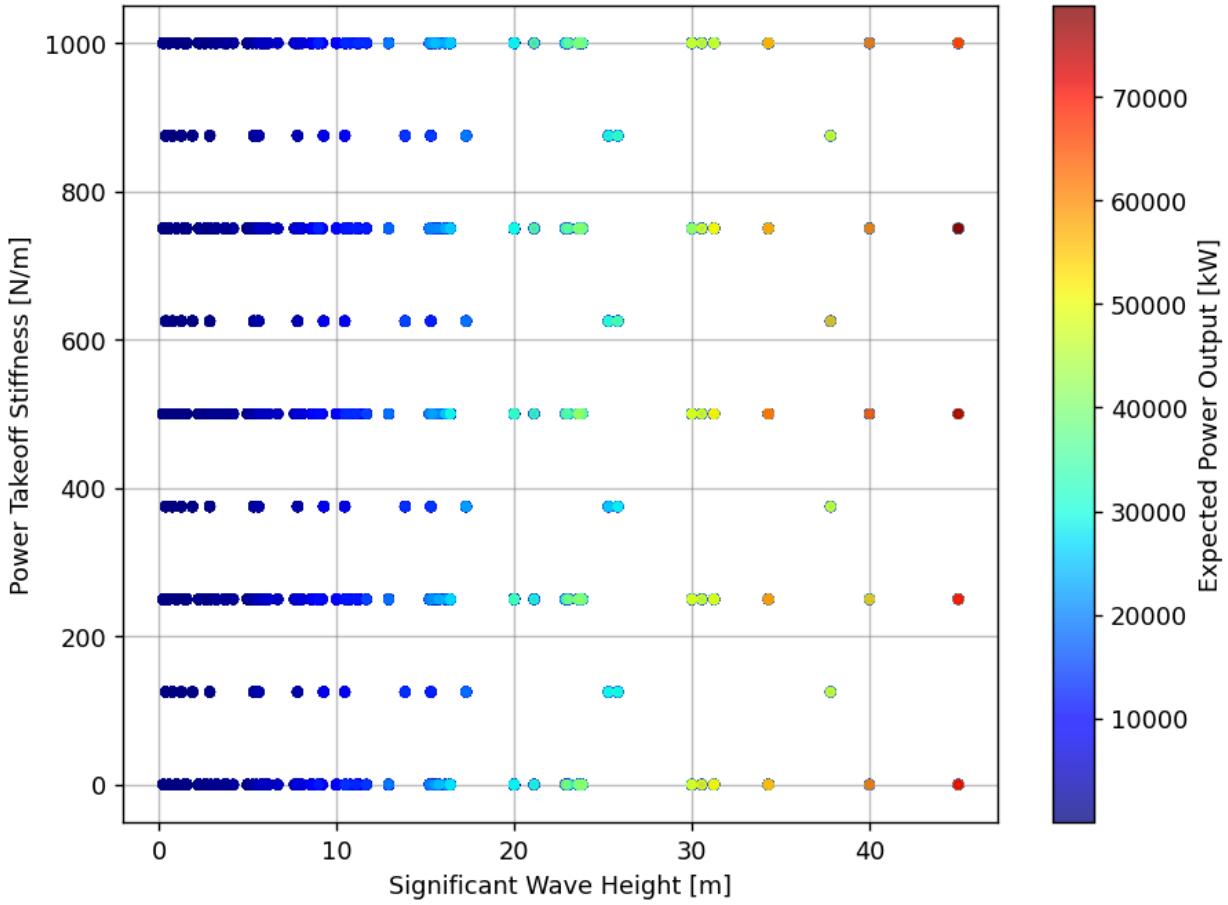

Figure A.8: PDS simulation results mining. Feature 4 vs Feature 1. Plotted with "hottest" points visible.

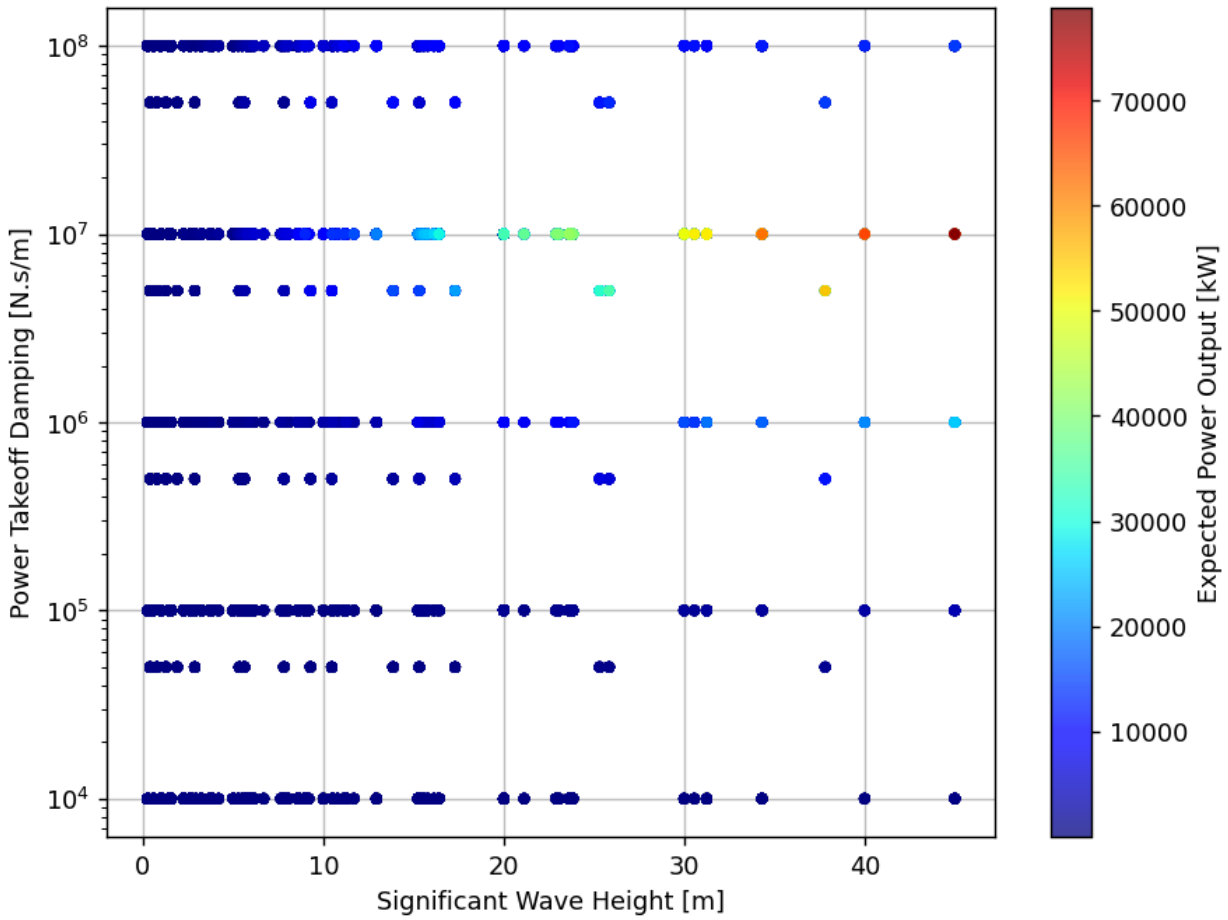

Figure A.9: PDS simulation results mining. Feature 5 vs Feature 1. Plotted with "hottest" points visible.

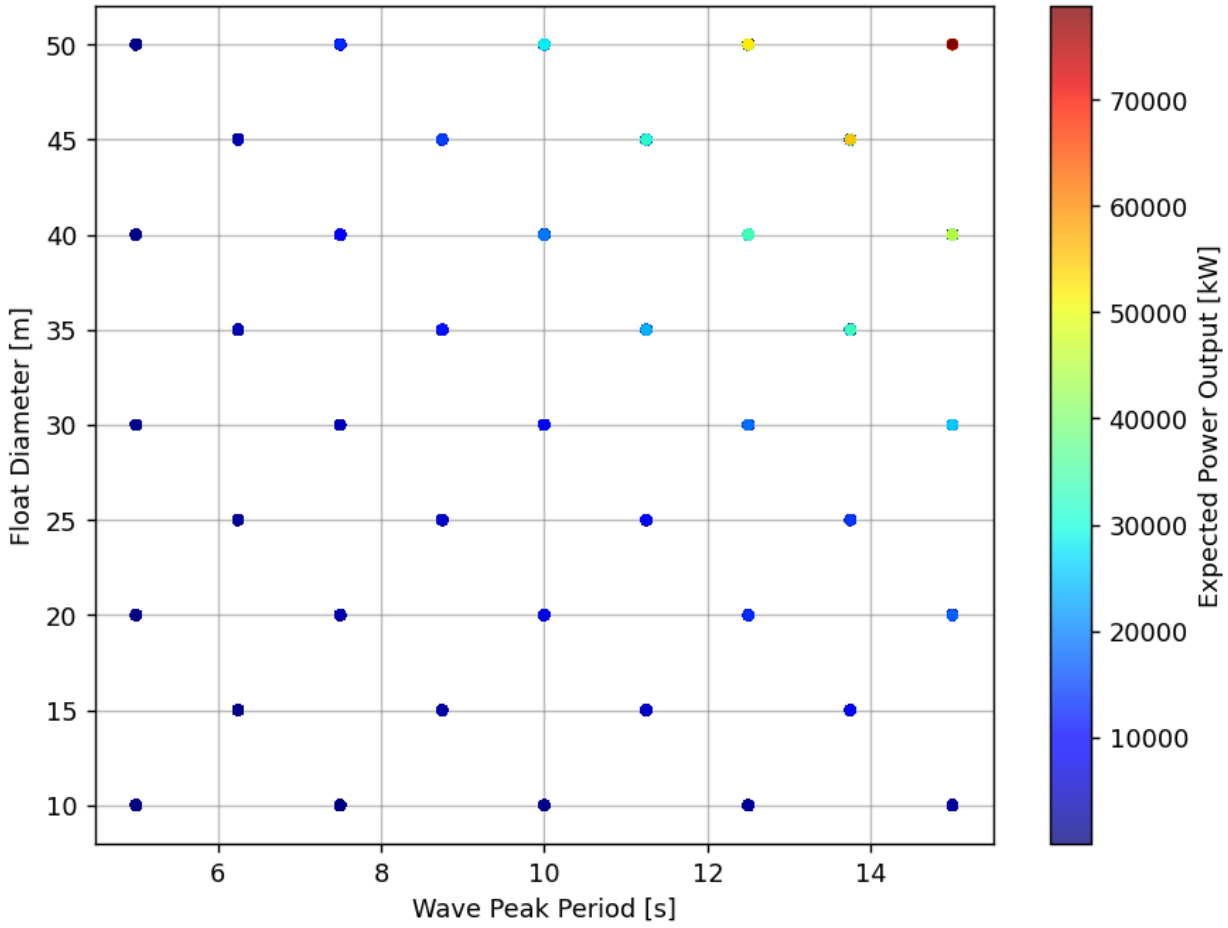

Figure A.10: PDS simulation results mining. Feature 3 vs Feature 2. Plotted with "hottest" points visible.

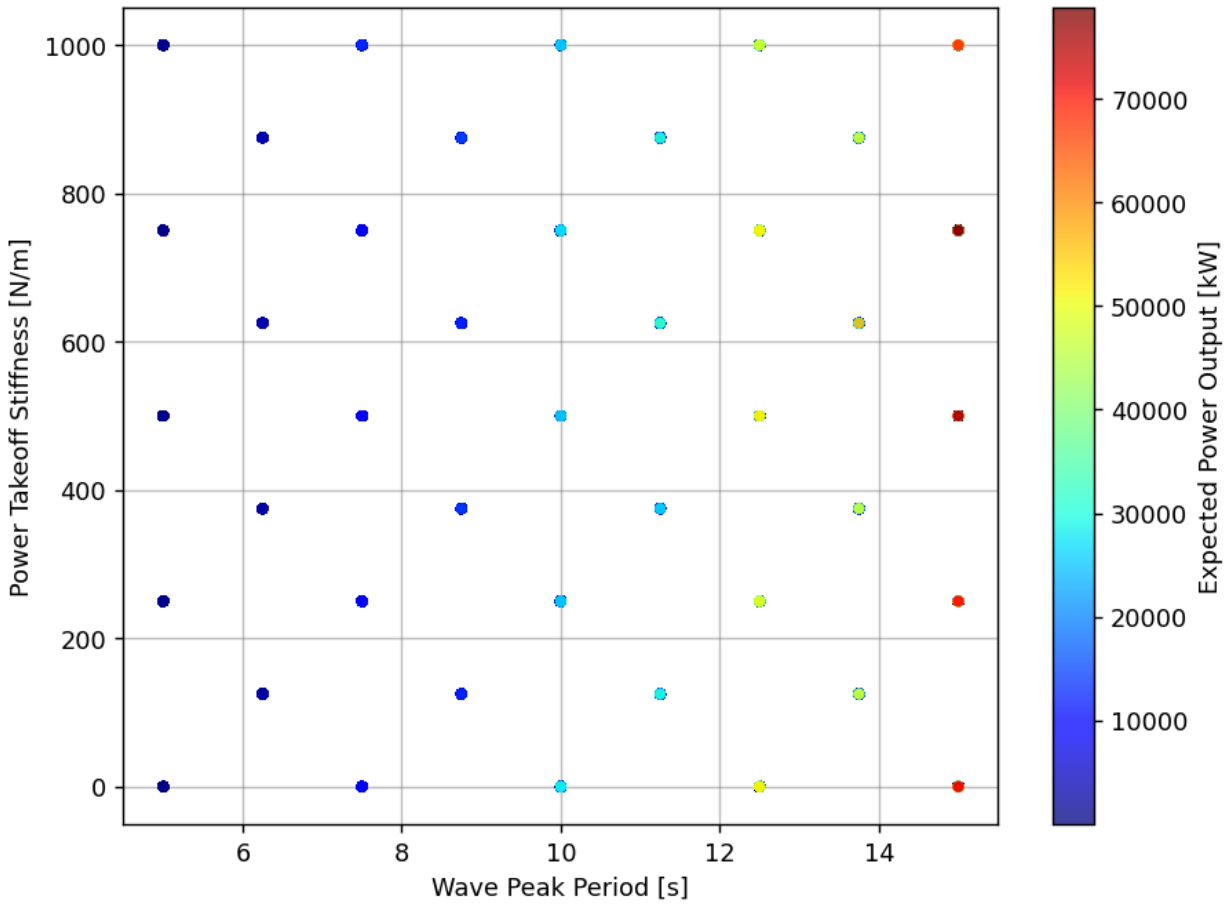

Figure A.11: PDS simulation results mining. Feature 4 vs Feature 2. Plotted with "hottest" points visible.

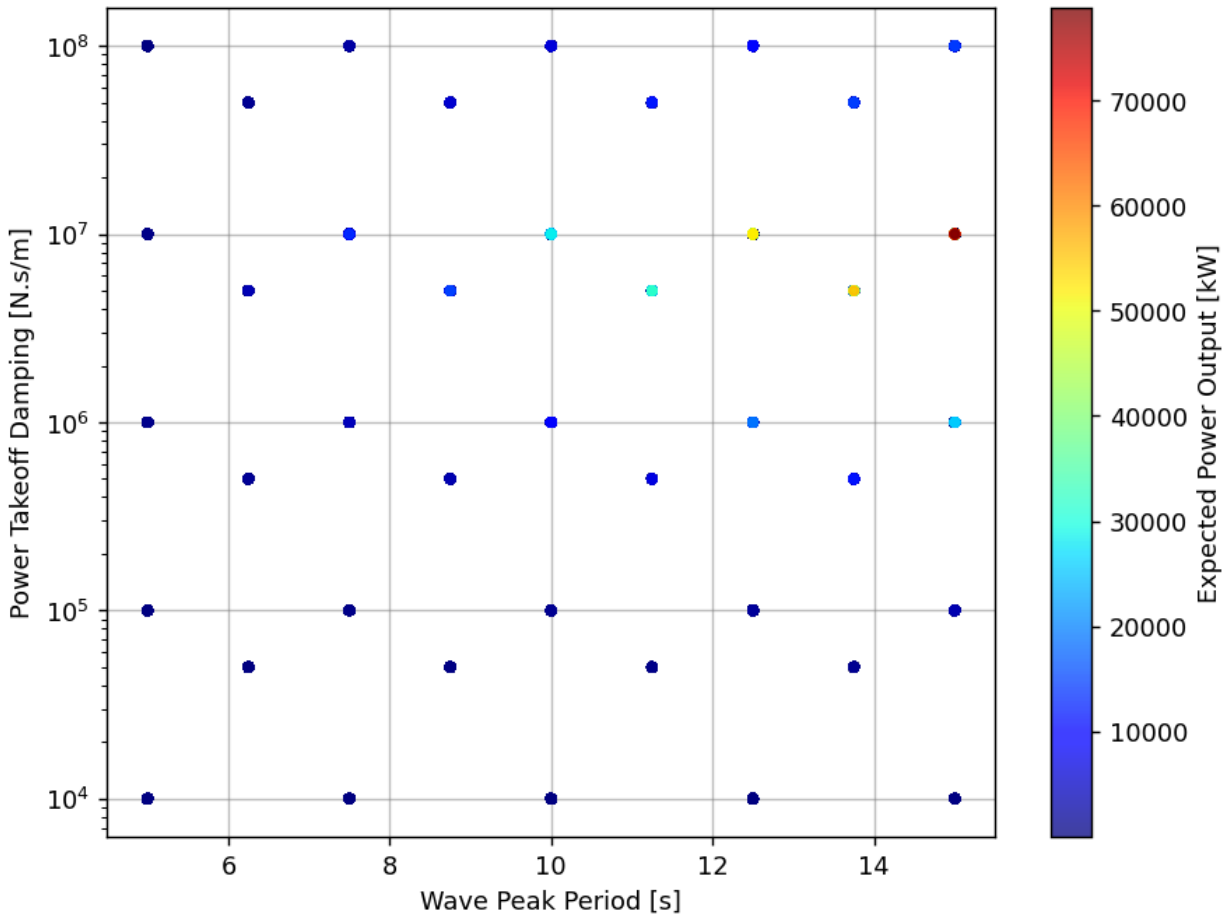

Figure A.12: PDS simulation results mining. Feature 5 vs Feature 2. Plotted with "hottest" points visible.

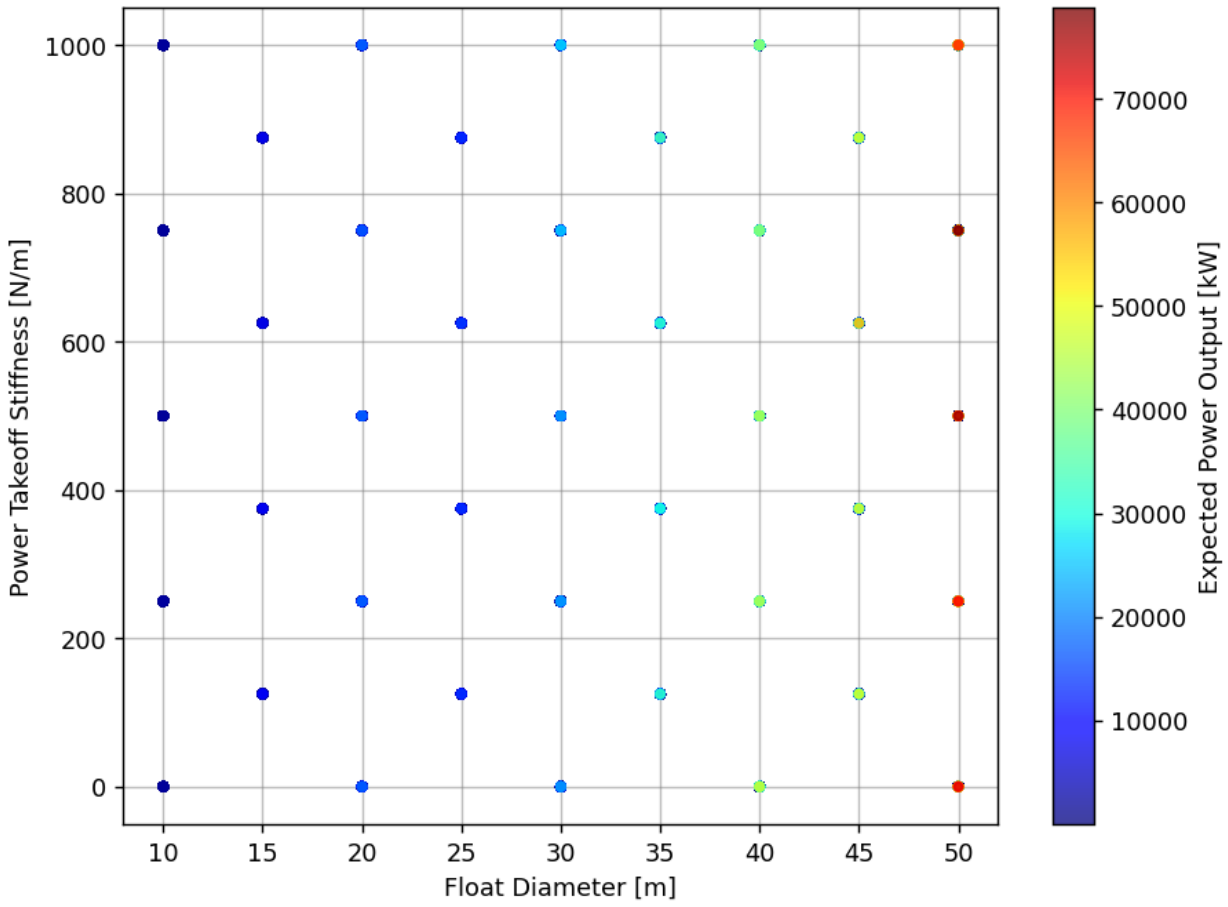

Figure A.13: PDS simulation results mining. Feature 4 vs Feature 3. Plotted with "hottest" points visible.

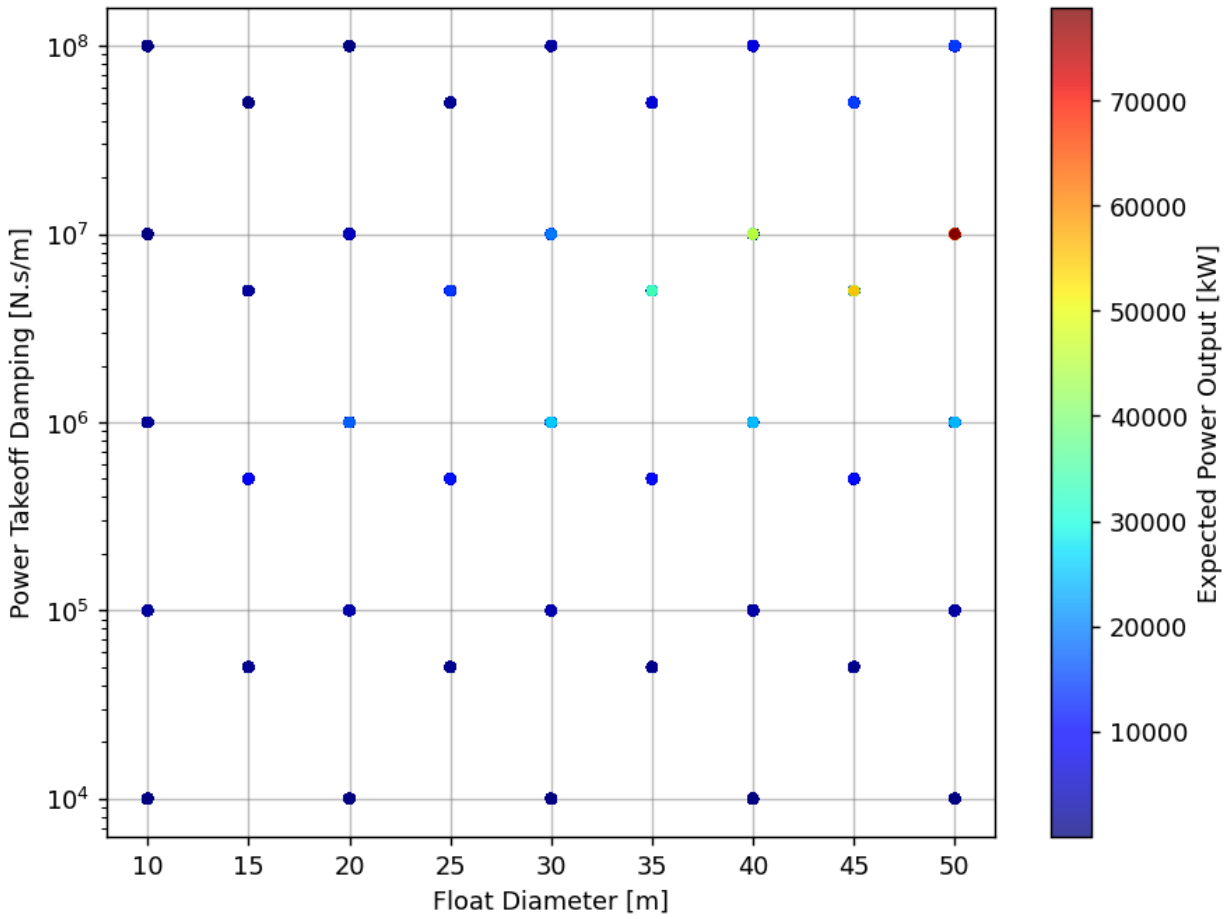

Figure A.14: PDS simulation results mining. Feature 5 vs Feature 3. Plotted with "hottest" points visible.

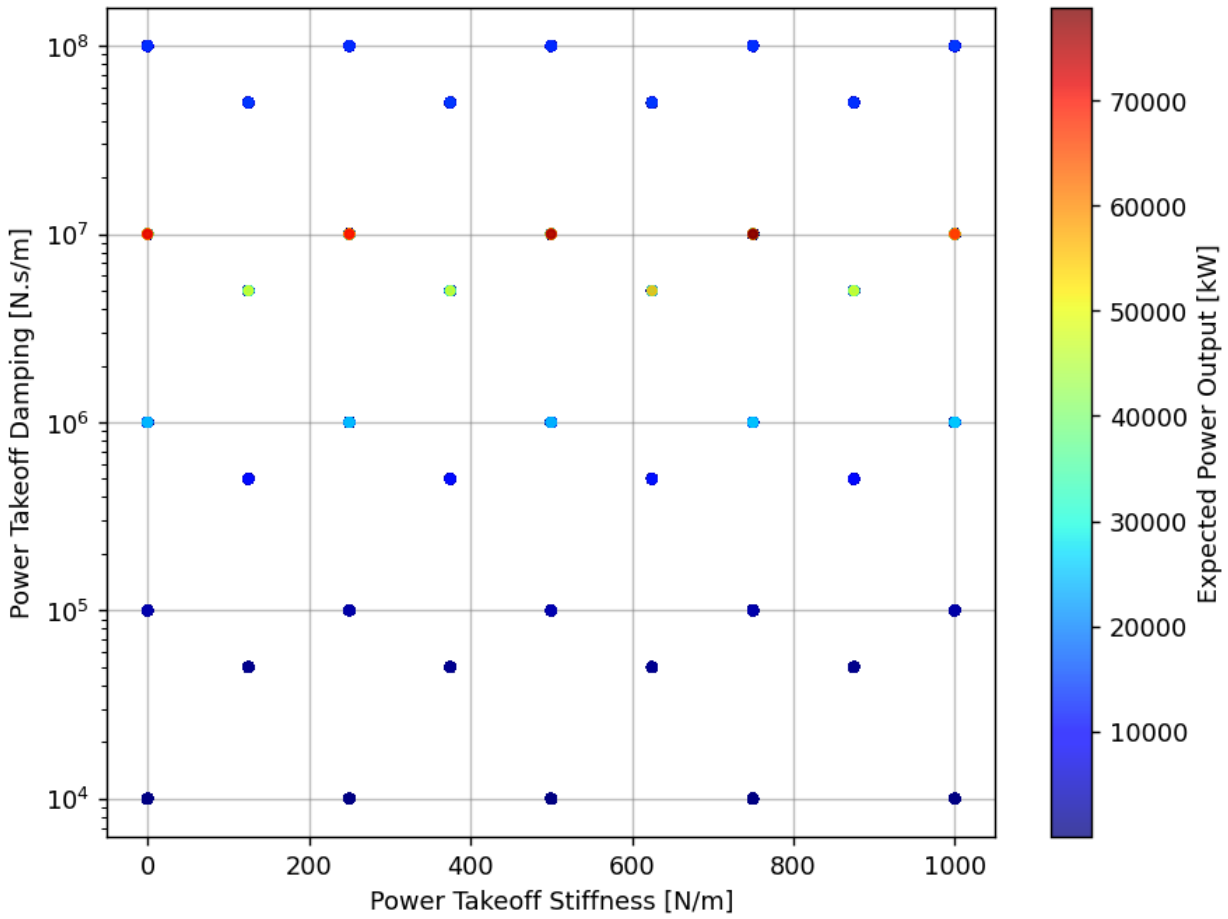

Figure A.15: PDS simulation results mining. Feature 5 vs Feature 4. Plotted with "hottest" points visible.

B Dimensionless Feature Plots

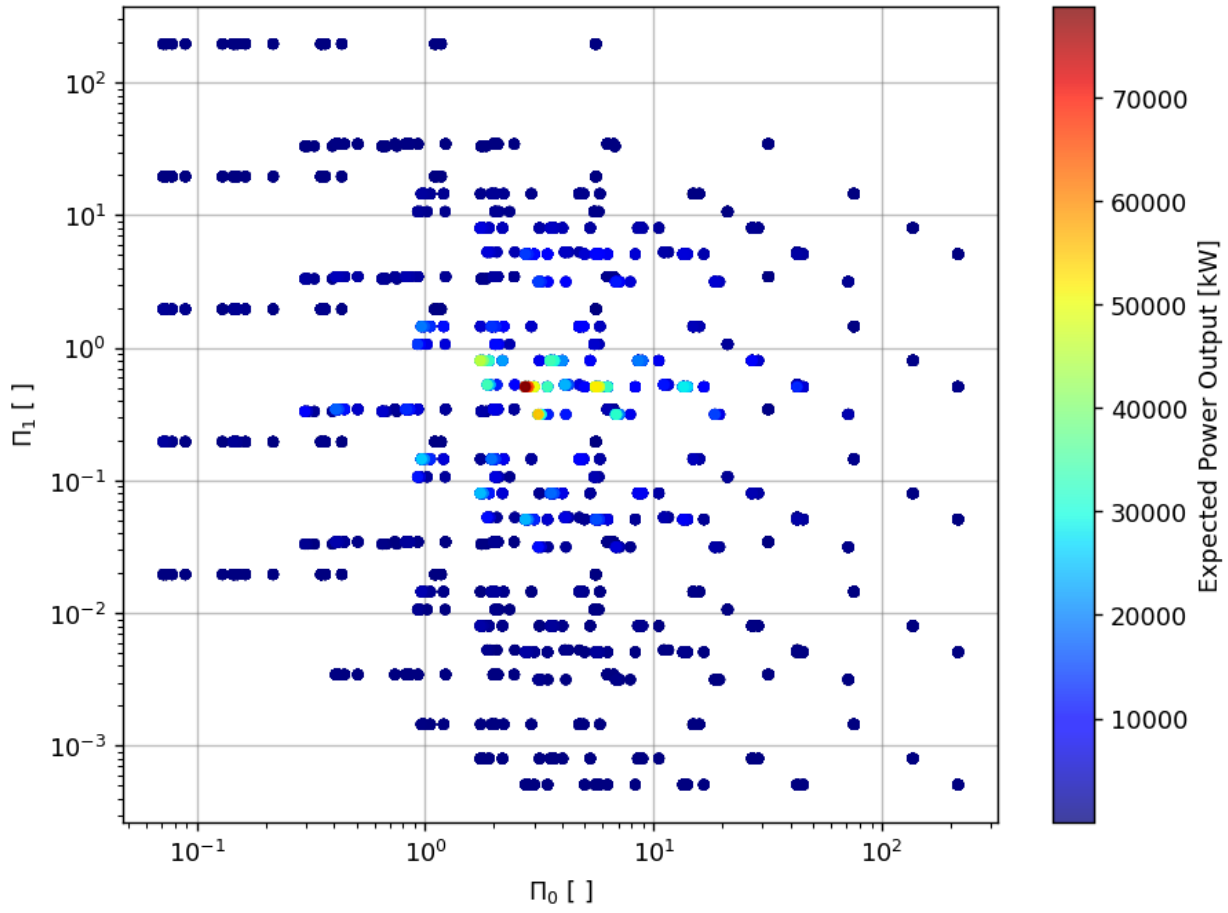

Figure B.1: PDS simulation results mining. Dimensionless Feature 1 vs Dimensionless Feature 0. Plotted with "hottest" points visible.

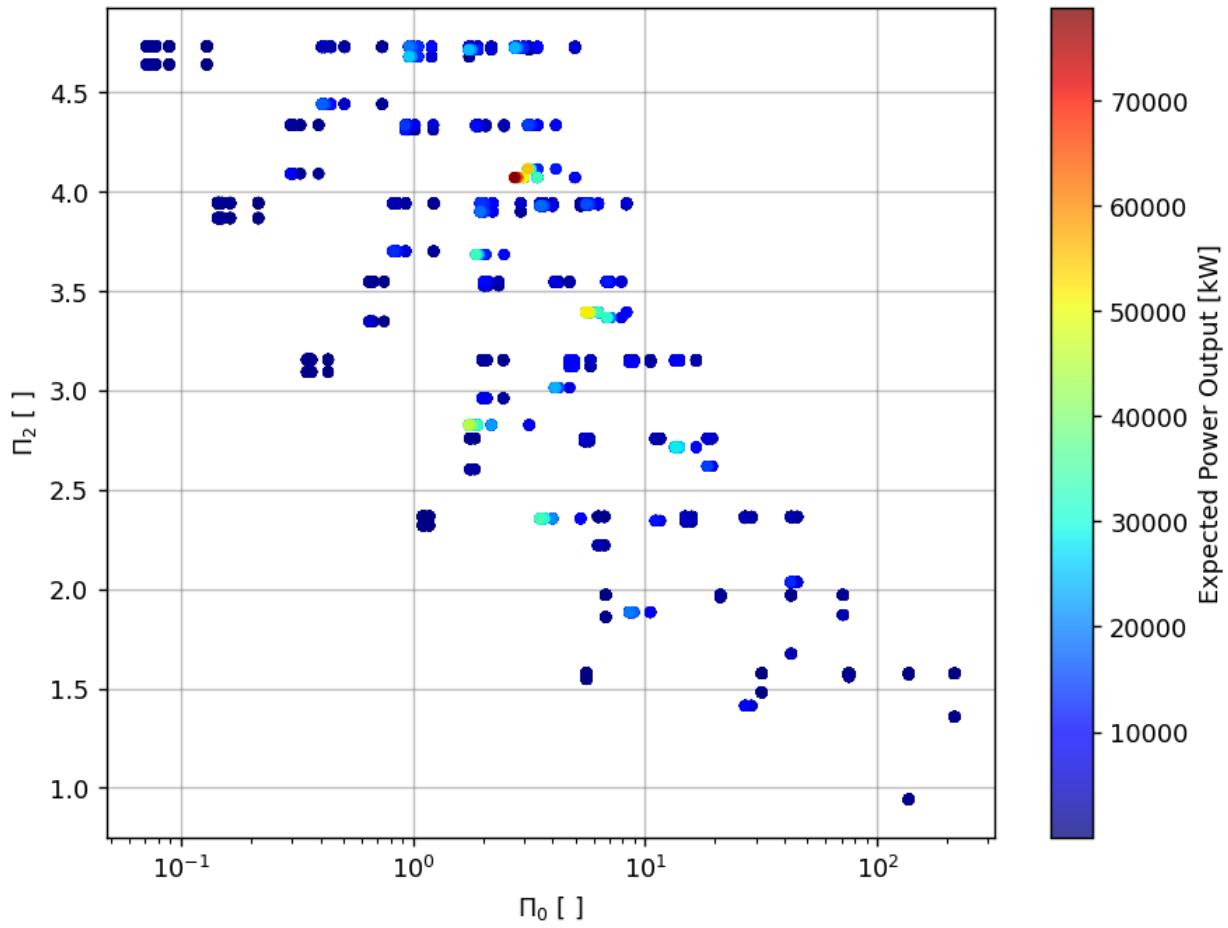

Figure B.2: PDS simulation results mining. Dimensionless Feature 2 vs Dimensionless Feature 0. Plotted with "hottest" points visible.

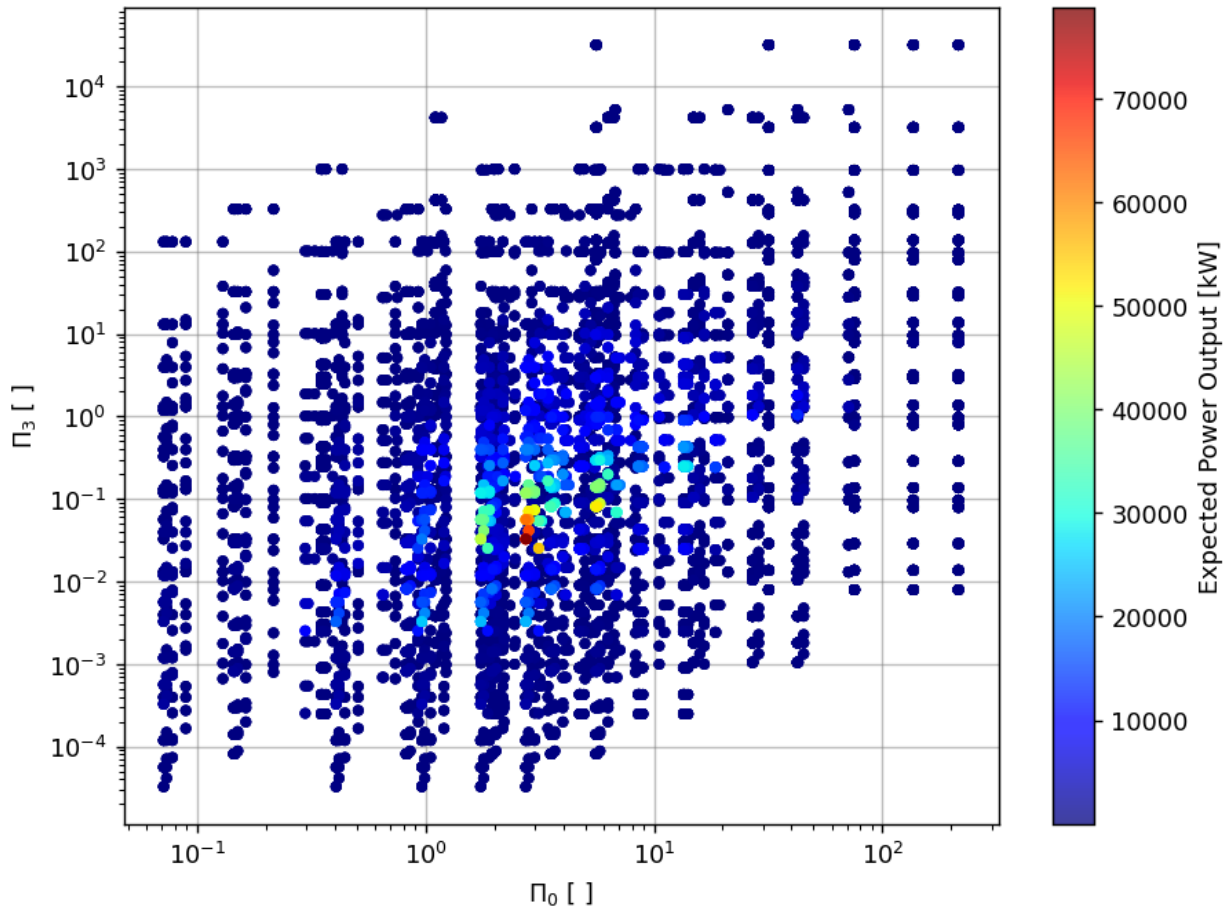

Figure B.3: PDS simulation results mining. Dimensionless Feature 3 vs Dimensionless Feature 0. Plotted with "hottest" points visible.

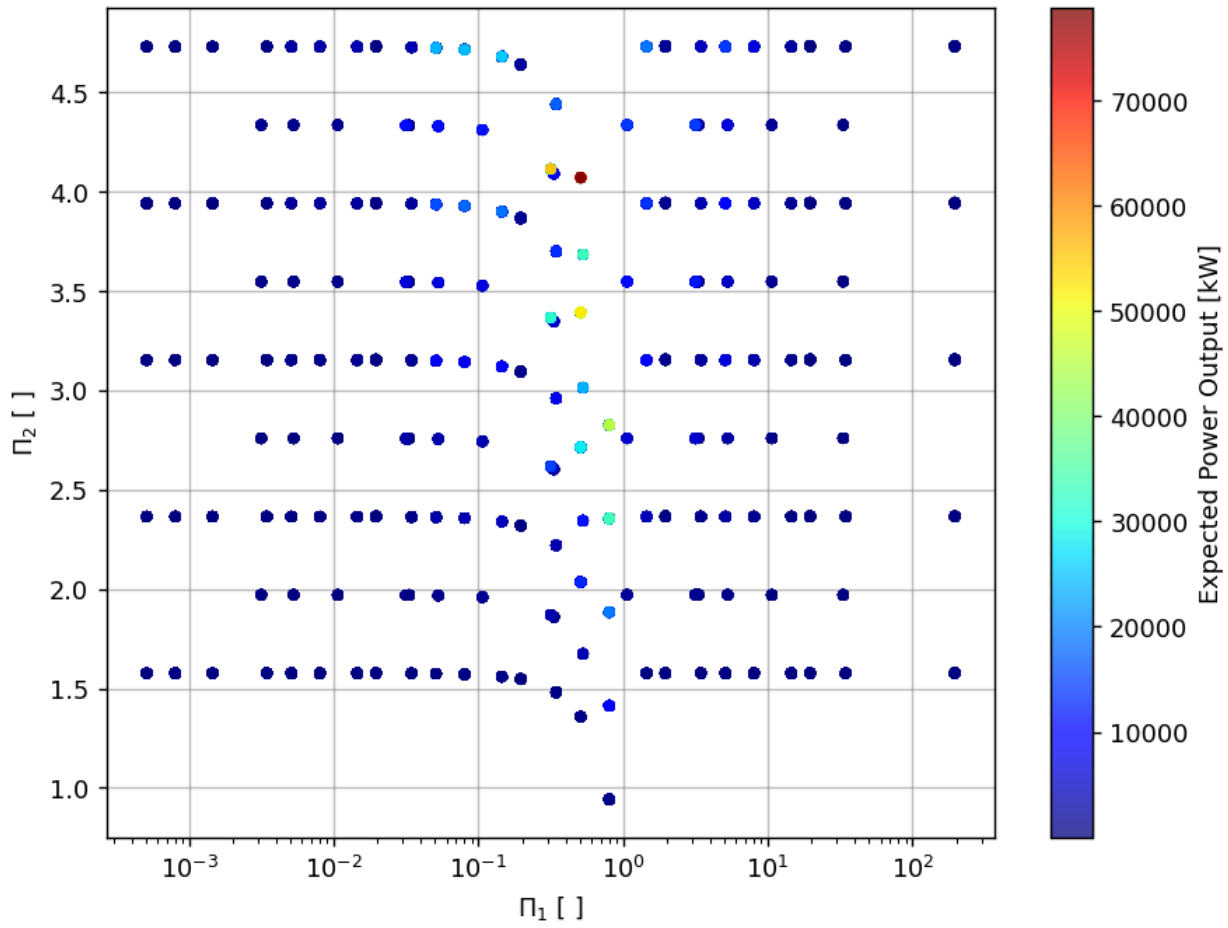

Figure B.4: PDS simulation results mining. Dimensionless Feature 2 vs Dimensionless Feature 1. Plotted with "hottest" points visible.

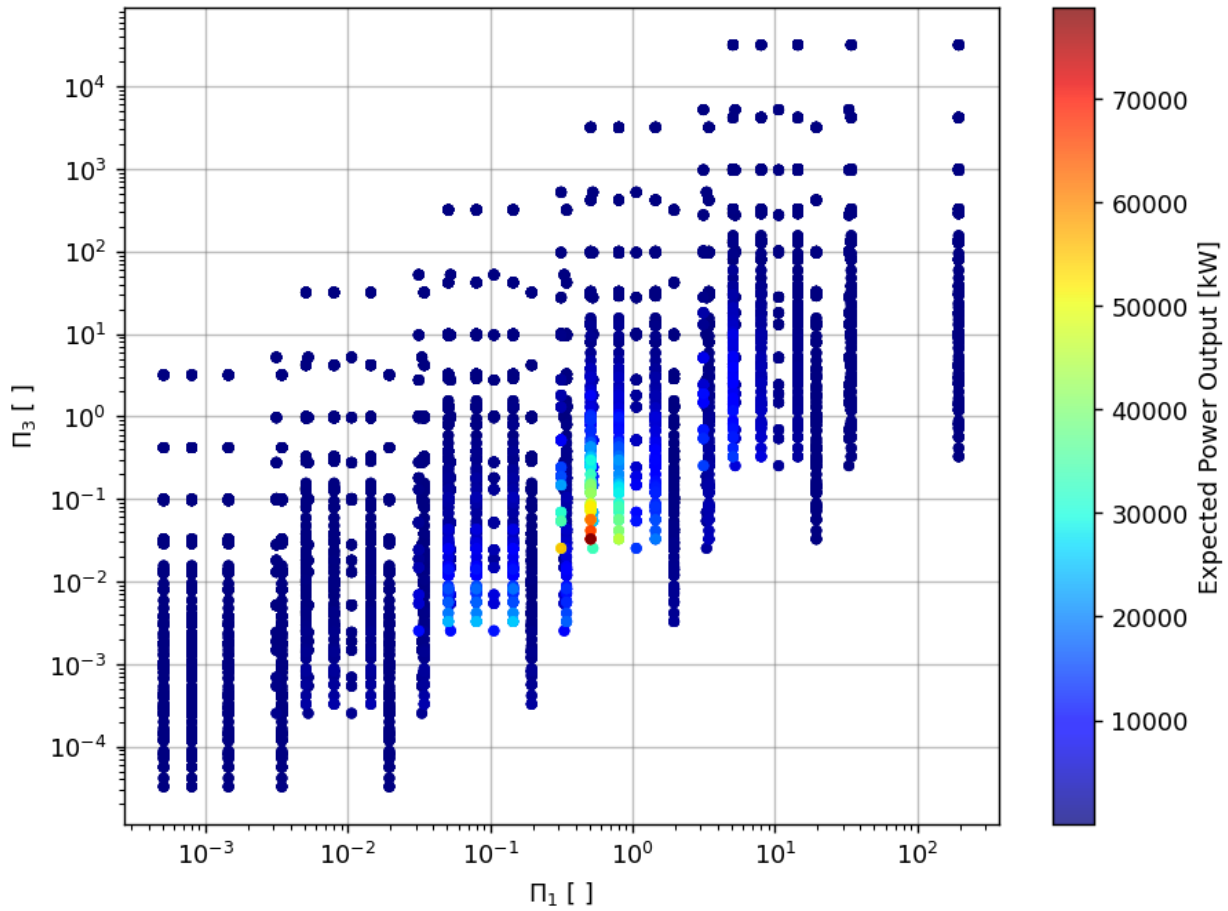

Figure B.5: PDS simulation results mining. Dimensionless Feature 3 vs Dimensionless Feature 1. Plotted with "hottest" points visible.

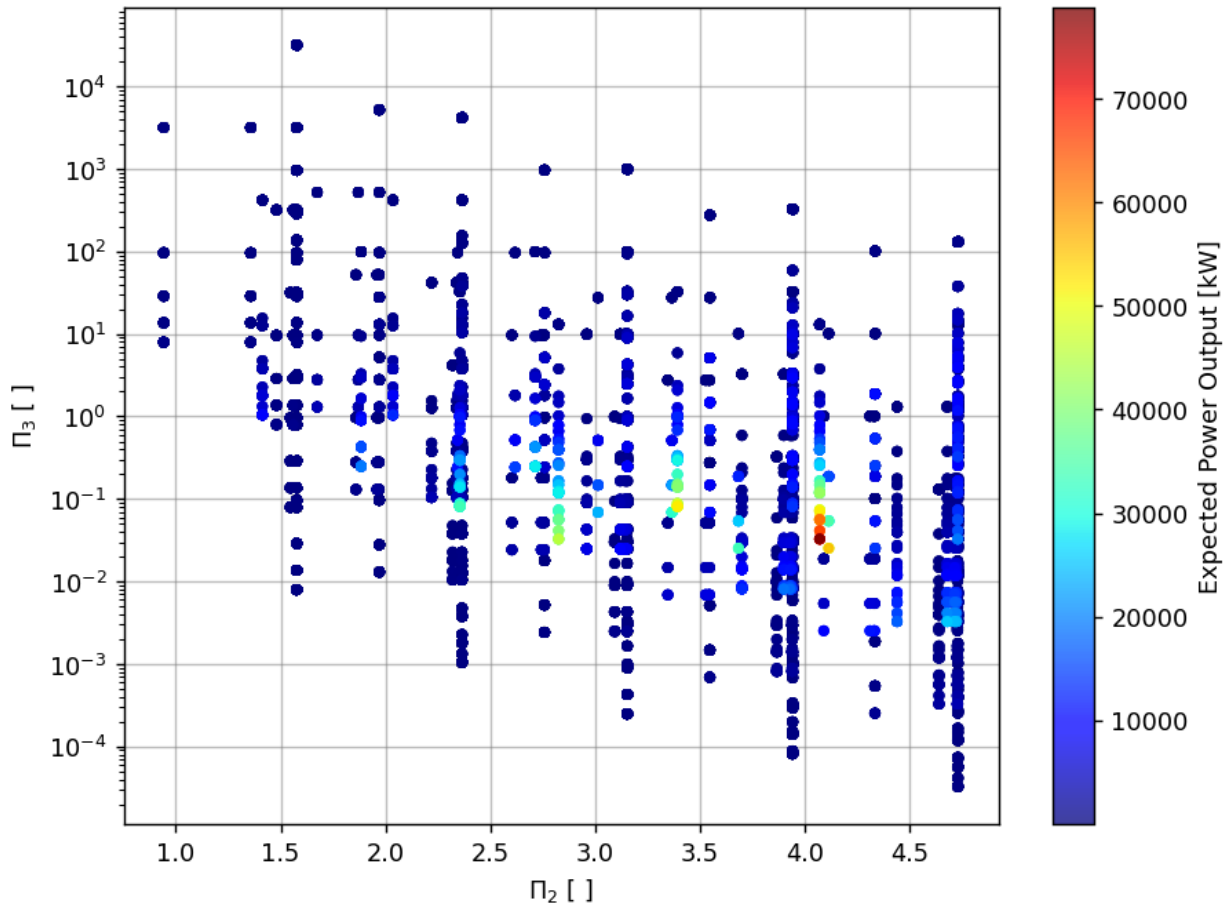

Figure B.6: PDS simulation results mining. Dimensionless Feature 3 vs Dimensionless Feature 2. Plotted with "hottest" points visible.