Wind Turbine Blade Optimisation

Design Optimisation Project

Alexander Evans Moncloa

UNIVERSITY OF BATH

Brief

This report entails the process of developing a wind turbine blade for a small-scale wind turbine. The principal metric that the overall turbine would be judged by is the total power extracted from the wind by the wind turbine (P), measured in watts. In order to increase P, the Wind Power Equation will be used and all relevant variables increased.

$$P = \frac{1}{2}\rho A v^3 C_p \qquad (1)$$

As seen in Equation 1, the relevant variables are ρ, air density (kg/m^3), A, swept area of the wind turbine blades (m^2), v, wind velocity (m/s) and Cp, coefficient of power (dimensionless).

By adhering to the General Permitted Development Order 2015, the blade length must have a maximum length of 1.1m. The turbine itself is to be placed nearby the University of Bath in southern England. This area has a low average windspeed of 5m/s (Shu et al. 2021), however major advantages include little daily or seasonal variance and a low altitude, therefore air density (ρ) is very high as it experiences more atmospheric pressure. Lesser speed variance allows the turbine to be geared to reach an optimal tip speed ratio (TSR), which will likely stay at the optimal velocity without needing dynamic wing angle adjustments, which are often the weak points in turbines and drive up manufacturing and repair costs.

This report will focus purely on aerodynamic performance, therefore the most critical analysis element is the Coefficient of Power (Cp). This number dictates how much power is being generated relative to the total power moving through the turbine area, however turbines cannot capture more than 59.3% of this power due to Betz's law, a derivation from the principles of conservation of mass and momentum (Betz 1966). Therefore, an optimal design will be judged as a percentage of this maximum coefficient.

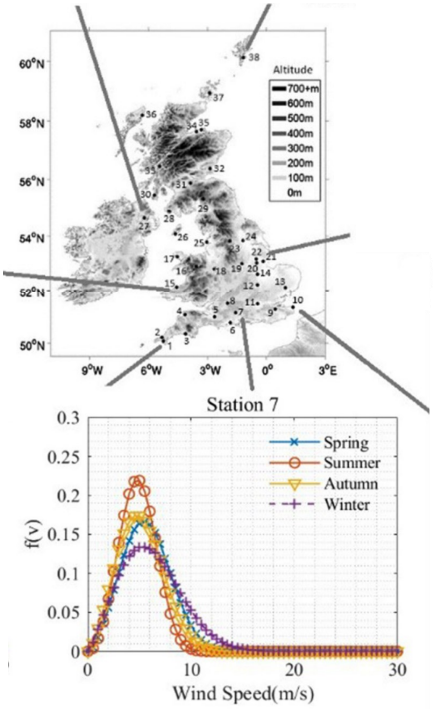


Fig 1: Weibull distribution for Station 7, nearby Bath, Somerset. Source: Shu et al. 2021

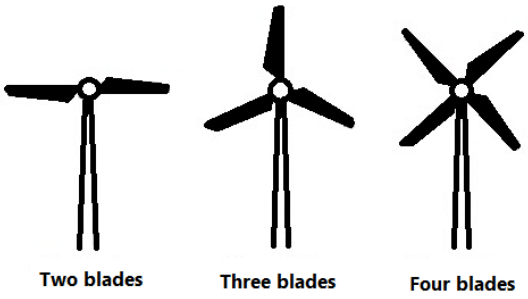
Constraints

Blade Count

To begin with, a reasonable blade count had to be decided. Options varying from 1 up to theoretically even 10 were on the table, however the main range considered was 2 to 4 blades. This was because a single blade has very poor balancing and has a history of disasters, along with 5 or more blades being too near to each other, causing closely aligned turbines to begin slipstreaming (lowering efficiency) (Adeseye 2021).

What ruled out a 2 bladed design was the fact that they are sensitive to gyroscopic precision, resulting in wobbling (Adeseye 2021). This leads to stability problems for the turbine, putting pressure on turbine components which reduce efficiency and lifetime. 4 bladed rotors actually can produce slightly more power at lower TSRs (~4) than a 3 bladed rotor, however the difference is very miniscule and even number of blades still causes stability problems if in a rigid frame. To ensure longevity, a 3 blade design was selected.

Fig 2: Illustration of different turbines



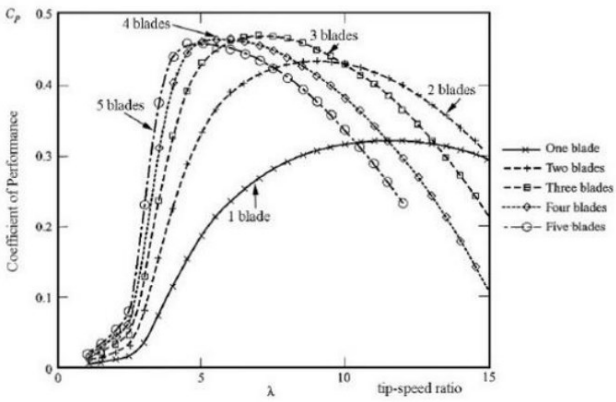


Fig 3: Cp values for different turbines (Adeseye 2021)

Blade Radius

In order to maximise power generated (P), the swept area of turbine blades (A) must also be increased. The legal limit is 1.1m per blade, therefore the blade length should be 1.1m. This generates a value for A of 3.8m^2.

Blade Elements

Selecting too little blade elements may lead to a misrepresentation of the true Cp values, however selecting too many blade elements would double or triple compute time for near-identical values.

With some experimentation, it was discovered that 25 elements was the sweet spot between accuracy and compute time.

Airfoil Choice

When selecting airfoils, there were 3 main contenders to select from. These were: S2091-101-83, SD7034 and S1210.

The reason for there being a variety of airfoils to select from is due to a paper, having analysed a large selection of airfoils for small wind turbines using XFOIL, having concluded that these 3 airfoils were most aerodynamic in conditions with a Re (Reynold's number) range present in low windspeed conditions (Salgado 2016).

The paper clearly states that mechanical strength and manufacturing process were not analysed, which is a positive sign as the airfoil selection criterion is entirely based on ability to maximise Cp.

All 3 airfoils were compared in the paper based on their maximum glide ratio (GR) relative to different Reynold's numbers. This metric is valuable because GR relates the lift coefficient (Cl) with the drag coefficient (Cd), essentially being a measure of efficiency.

Within the paper, the maximum GR of S1210 was highest compared to the 5 other analysed airfoils for 3/6 different Reynold's number ranges. The airfoils that had the greatest GR in the other 3 categories (S4061 and S4320) were omitted as viable candidates by the authors due to a very high decay in GR at values nearby their maximum values, essentially reducing the reliability that these airfoils would remain near their maximal GR.

This left S1210 as the greatest option for airfoil choice by a large margin. The airfoil was included in the Ashes database, therefore was kept as a constraint for all simulations.

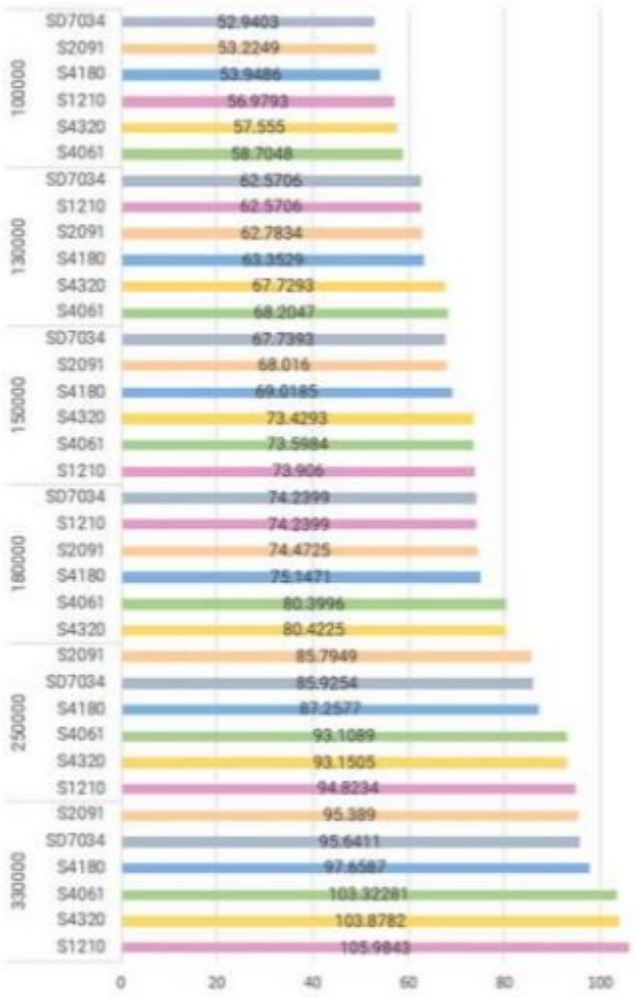
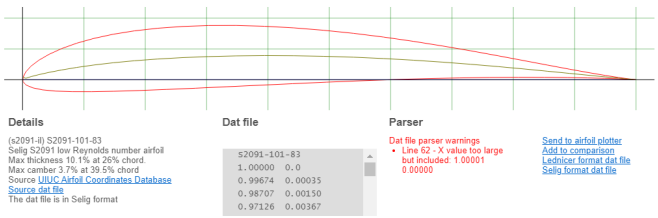


Fig 4: Value for Max-GR for selected airfoils at different Re (Salgado 2016)

S2091-101-83 (s2091-il)

S2091-101-83 - Selig S2091 low Reynolds number airfoil



Details

(s2091-il) S2091-101-83
Selig S2091 low Reynolds number airfoil
Max thickness 10.1% at 26% chord.
Max camber 3.7% at 39.5% chord.
Source UIUC Airfoil Coordinates Database
Source dat file
The dat file is in Selig format

Dat file

S2091-101-83
1.00000 0.0
0.99674 0.00035
0.98707 0.00150
0.97126 0.00367

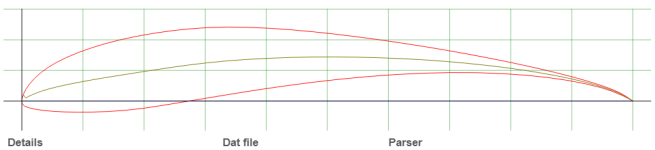
Parser

Dat file parser warnings
- Line 62 - X value too large but included: 1.00001 0.00000

Send to airfoil plotter
Add to comparison
Lednicer format dat file
Selig format dat file

S1210 12% (s1210-il)

S1210 12% - Selig S1210 high lift low Reynolds number airfoil



Details

(s1210-il) S1210 12%
Selig S1210 high lift low Reynolds number airfoil
Max thickness 12% at 21.4% chord.
Max camber 6.7% at 51.1% chord.
Source UIUC Airfoil Coordinates Database
Source dat file
The dat file is in Selig format

Dat file

S1210 12%
1.00000 0.00000
0.99837 0.00101
0.99398 0.00397
0.98753 0.00832
0.97908 0.01317

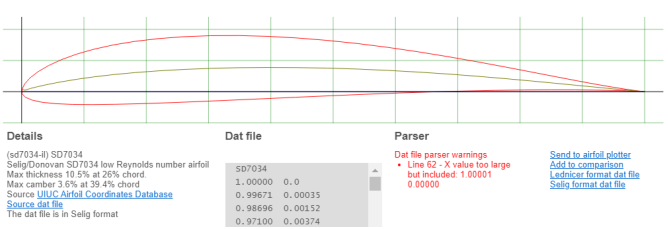
Parser

No parser warnings

Send to airfoil plotter
Add to comparison
Lednicer format dat file
Selig format dat file

SD7034 (sd7034-il)

SD7034 - Selig/Donovan SD7034 low Reynolds number airfoil



Details

(sd7034-il) SD7034
Selig/Donovan SD7034 low Reynolds number airfoil
Max thickness 10.5% at 26% chord.
Max camber 3.6% at 39.4% chord.
Source UIUC Airfoil Coordinates Database
Source dat file
The dat file is in Selig format

Dat file

SD7034
1.00000 0.0
0.99671 0.00035
0.98696 0.00152
0.97100 0.00374

Parser

Dat file parser warnings
- Line 62 - X value too large but included: 1.00001 0.00000

Send to airfoil plotter
Add to comparison
Lednicer format dat file
Selig format dat file

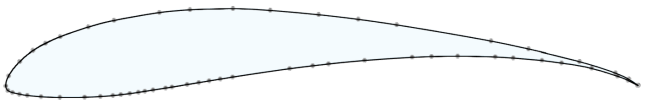


Fig 5: Airfoil shapes of 3 considered airfoils (Airfoil Tools 2024), followed by Fusion360 render of S1210 using the spline tool

Parameters

Twist

Angle of Attack

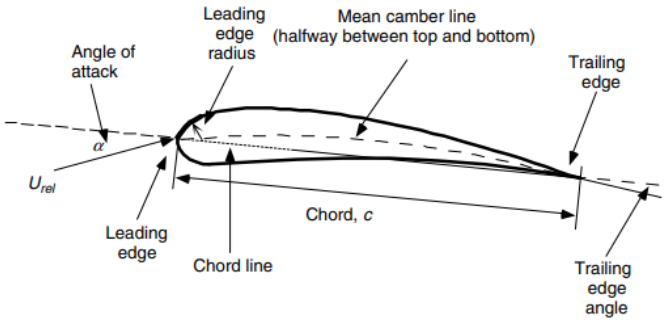


Fig 6: Airfoil nomenclature.

Source: Manwell 2009

Suboptimal: If the blade is twisted incorrectly, it can cause a head-on collision with the wind, resulting in zero or negative torque (stall condition).
Optimal: Proper twist ensures the blade experiences lift rather than drag, maximising power generation.
Excessive: At extreme twists, the blade becomes aerodynamic to the wind again but fails to generate torque, entering a "no-movement" regime.

Wind Speed

The relative wind velocity (V_{rel}) experienced by a blade segment depends on the vector sum of the wind speed (V_{wind}) and the tangential velocity of the blade ($r \times \omega$), where r is the radial position along the blade and ω is the angular velocity:

$$V_{rel} = \sqrt{(V_{wind})^2 + (r \cdot \omega)^2} \quad (2)$$

Since V_{rel} changes with radius, the blade must have a greater twist at the root and a smaller twist at the tip to ensure a near-optimal angle of attack across its span.

Chord Length

The chord length affects C_P by influencing the lift-to-drag ratio of the blade. A longer chord generates more lift but also increases drag. The ideal chord length balances these effects:

Lift Generation: A larger chord increases the effective lifting area, boosting aerodynamic force.
Drag Reduction: A thinner chord reduces drag, which becomes significant at high tip speeds.

The chord length distribution was parameterised to vary between maximum of 50 cm at the root (for structural strength and higher lift generation) and minimum of 10 cm at the tip to reduce drag, as the tip operates at higher relative velocities and drag penalties increase quadratically with speed.

TSR (Tip Speed Ratio)

$$TSR = \frac{\omega \cdot R}{V_{wind}} \quad (3)$$

Where R is the blade length. The C_P value varies significantly with TSR, as shown in the results section.

Studies in the literature often disagree on the ideal TSR due to variations in blade design (number of blades, chord length, twist), wind conditions (steady vs. turbulent wind profiles) and aerofoil selection.

For three-bladed turbines, the optimal TSR typically falls between 4 and 8, depending on design specifics and target operational ranges (Manwell 2009).

The optimisation algorithm generated C_P values at TSRs of 2, 4, 6, 8, and 10. By plotting these data points, the TSR that maximised C_P could be identified. For this project, the optimal TSR was definitively found to be 4, aligning with theoretical expectations for small wind turbines operating in low wind-speed conditions.

Optimisation

Genetic Algorithm

newMain.m

Fig 7: MATLAB script responsible for constant twist/chord

```
newTwist = ones(25, 1) * X(1); % Creates a constant twist distribution
newChord = ones(25, 1) * X(2); % Creates a constant chord distribution
```

In order to find the optimal values of twist and chord length, a genetic algorithm was set up in MATLAB to attempt to find the combination of values with the highest Cp output. The way this algorithm works is by creating a population of data points, each data point being defined by its Cp value with a genetic code compromising of its twist and chord lengths. These populations undergo a simulated version of natural selection, where only the fittest (those with the highest Cp) get to survive to the next generation. Data points with high fitness are combined to make offspring, which are then introduced into the population.

The critical aspect of this algorithm is the ability for genes to mutate. Increasing mutation rate minimises the chance of falling into a local minima, as even with a very fit population going in a certain direction, the genetic mutations have the chance to explore different aspects of the function that normally would be ignored. In the results it will be clear to see the mutations, they often are seen as significant downgrades in Cp at a late stage of evolution, however the a few cause a significant jump up in Cp, driving the model's progress forwards.

The initial algorithm shown here only optimised for the entire blade's twist and chord, solving a relatively simple graph in a costly manner. This was because newMain.m was just for testing purposes, ensuring that adequate values actually could be generated. Only 2-3 generations were ran before moving onto a more complex optimisation space.

newerMain.m

Fig 8: Main code change in MATLAB

```
% Ensure linear variation in twist and chord
twistDist = linspace(X(1), X(2), 25)'; % Linearly varying twist from root (X(1)) to tip (X(2))
chordDist = linspace(X(3), X(4), 25)'; % Linearly varying chord from root (X(3)) to tip (X(4))

% Ensure constraints: root twist > tip twist and root chord > tip chord
if X(1) <= X(2) || X(3) <= X(4)
    F = Inf; % Penalise invalid configurations
    return;
end
```

This was the next iteration of the genetic algorithm. This version was ran for a culmination of 100 generations, however this had to be done in stages with human supervision to avoid wasting compute on a poorly set up system.

This algorithm took 4 input variables, the root twist angle, the root chord length as well as the tip twist angle and the tip chord length. Any combination of these 4 variables would lead to different Cp values as outputs from Ashes, however there was more potential here for an optimal blade design due to previously discussed potential for keeping the angle of attack optimal at different parts of the blade.

The change in twist and chord length from root to tip was mapped segment by segment using the linspace function, sending all 25 segments to Ashes for each data point. By adding an infinate penalty to inputs where twist angle or chord length were greater at the tip than the base, the algorithm generated relevant data points from the get-go, which lead to less time taken to converge on a maximum Cp value.

newestMain.m

A linear change in twist and chord may not be the optimal solution for maximising Cp in a turbine blade according to Manwell, 2009. This book suggests that to make a Betz optimum blade, the twist and chord at the base must exponentially decrease, levelling off around 1/3 through the blade length. The rationale for this being that drag increases exponentially with velocity, therefore the tip is subject to many times more drag than the root.

This profile promises to maximise Cp values, therefore the new 4 input variables were root twist angle, root chord length, twist coefficient and chord coefficient. The effect these coefficients had on values at each segment were governed by the functions below:

$$\text{Twist}(r) = \text{Root Twist} \cdot e^{-\text{Twist Coefficient} \cdot r} \quad (4)$$

$$\text{Chord}(r) = \text{Root Chord} \cdot e^{-\text{Chord Coefficient} \cdot r} \quad (5)$$

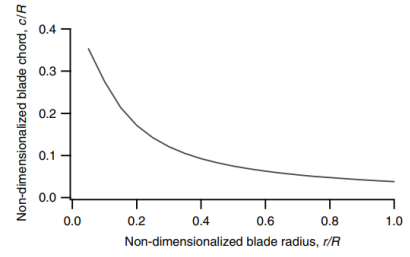


Figure 3.25 Blade chord for the example Betz optimum blade

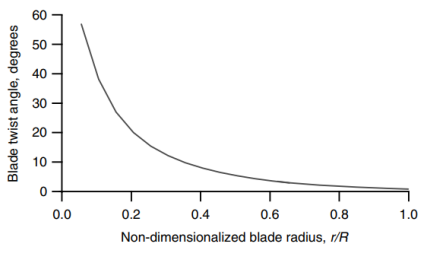


Figure 3.26 Blade twist angle for sample Betz optimum blade

Fig 9: Betz optimum blade changes

Innovative Approaches

parpool

Fig 10: parpool() being used in MATLAB

```
parpool('local');

% Genetic Algorithm Options
options = optimoptions('gamultiobj', ...
    'UseParallel', true, ... % Parallel computation
```

As most MATLAB optimisation algorithms automatically run on a single CPU core, there was room for algorithm optimisation to drastically reduce compute time. The usage of the parpool() function in MATLAB was attempted, in order to utilise all 8 cores + threads on the host computer. This would theoretically reduce compute times by a maximum 8 -fold.

The implementation was unsuccessful in MATLAB for usage with the ga function. A slew of error messages with no definitive solution led to trials with a different function, perhaps better optimised for usage with parpool(). This led to the algorithm being rewritten in order to use gamultiobj, a version of ga that is designed for multiple objectives.

There is only 1 objective, maximise Cp, however as gamultiobj worked with parpool(), therefore by plotting Cp against a dummy objective (set to 0) MATLAB would use the full CPU functionality to find the optimal function.

This approach worked for a few seconds, blitzing through datapoints, until MATLAB would inevitably crash. This resulted in persistent crashes, even with parpool set to use only 2 cores (very perplexing).

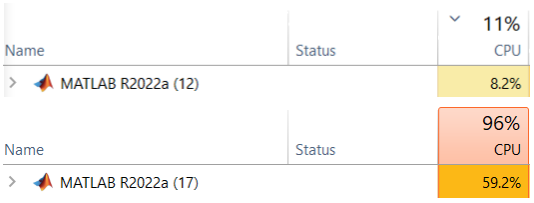


Fig 11: (Above) CPU usage, (Below) MATLAB code

gpuArray

Due to the unsuccessful parallelisation with various CPU cores, workarounds to involve the powerful on-board Nvidia graphics card were considered. Graphics cards are currently very trendy for solving optimisation problems, however usually the optimisation algorithms aren't genetic.

```
function F = optimiseBladeGPU(X, Params, numSegments)
    % Move parameters to GPU
    X = gpuArray(X);

    % Define twist and chord distributions
    newTwist = gpuArray(linspace(X(1), X(2), numSegments)'); % Twist distri
    newChord = gpuArray(generateChordDistribution(X(3), X(4), numSegments))

    % Run simulation on GPU
    [rotor, ~] = runSIMGPU(Params, newTwist, newChord);
```

There existed no easy way to use the graphics card for the MATLAB ga function, however experimentation with offloading arrays onto the GPU were attempted. By using the gpuArray() function, stored values of twist and chord were saved onto the graphics card successfully.

Unfortunately, this did not lead to any reduction in time taken. This is because the primary time-taking tasks were still running on the CPU (with no way to run it in parallel on GPU cores), therefore this speed reduction attempt was akin to giving your sibling an unplugged controller and telling them they helping you beat a video game.

Deep Learning Agent

In order to effectively use the GPU, a switch in optimisation algorithm was considered. Deep learning algorithms have increased in popularity recently and more importantly, MATLAB have made the function compatible with Nvidia GPUs.

The idea behind using a deep learning algorithm is to train an agent to move around the solution space, being rewarded for finding areas with a greater Cp (envision a robot climbing up a hill) and punished when it wanders into areas with a lower Cp (Dell'Aversana 2022).

This method can be more prone to being stuck at local maxima, but parameters can be changes to encourage the agent to explore nearby areas with "curiosity" despite it going in the wrong direction.

Unfortunately, this method did not properly run. The most likely reason might be related to the graphics card not being able to communicate effectively with Ashes.

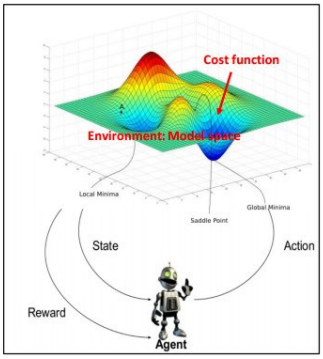


Fig 12: Diagram of DRL agent in function (Dell'Aversana 2022)

Simulated Annealing

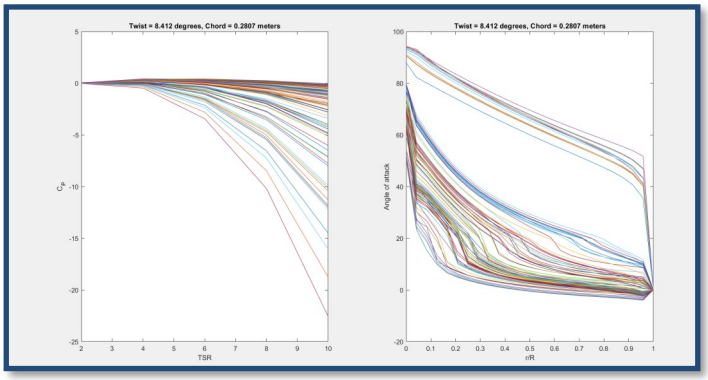
This method models the physical process of heating a material then lowering the temperature, slowly, to decrease defects. This method is highly effective at finding a global minima/maxima, therefore was used in conjunction with the genetic algorithm to cross reference values found. This method did not have the same accuracy as a genetic algorithm, being worse for pinpointing a precise value.

Discussion of Results

Genetic Algorithm

newMain.m

Fig 13: Initial MATLAB outputs



These results were mostly for testing to see if the genetic algorithm could converge around values that seemed reasonable.

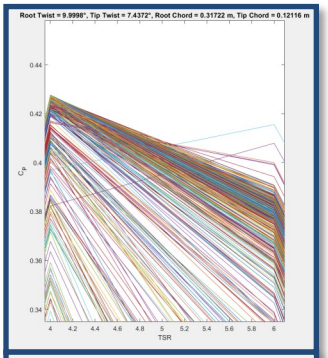
These blades had universal twist and chord to them, therefore the optimal value would likely have converged below 0.42, as shown by a side-by-side comparison of a 2 generation run along with a 3 generation run. If the differenced weren't in the scale of 0.0002, then perhaps throwing more compute at the task would have had more promise.

These simulations gained insight on the angle of attack for various designs, the vast majority having a very steep change in angle of attack due to the nature of drag increasing exponentially by the tip.

newerMain.m

```
Optimal values found:
Root Twist: 20.9893 degrees
Tip Twist: 4.3643 degrees
Root Chord: 0.34535 meters
Tip Chord: 0.13136 meters
Maximum C_P: 0.44394
```

Fig 14: MATLAB outputs



This simulation consisted of varying twist angle and chord length as the turbine changes length. The resultant Cp values after 50 generations were really promising, approaching 0.444.

This algorithm was ran overnight, taking 10hrs to converge. The first run actually had the root twist cap out at a value of 19.9998 degrees, therefore the simulation had to be re-ran with twist bounds between 20 and 30, causing a convergence around 21 degrees to be optimal.

Each combination was also graphed with varying TSR values. With a fine zoom it can be seen that the vast majority of high achievers in the population would perform optimally at a TSR of 4, influencing the parameters and making optimising Cp for a TSR of 4 to be the new goal.

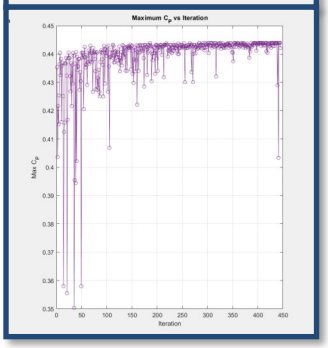


Fig 15: More MATLAB outputs

newestMain.m

```
Optimal values found:
Twist Coefficient: 0.99854
Chord Coefficient: 0.66468
Root Twist: 19.1669 degrees
Root Chord: 0.30838 meters
Maximum C_P: 0.44292
```

```
>> tiptwistandchord
Tip Twist: 7.0614 degrees
Tip Chord: 0.15864 meters
```

This final simulation used an exponential function to vary the twist and chord of the turbine blade. Very interestingly, this did not perform as expected.

The maximum Cp value converged on was slightly less than the value newerMain.m converged on, being 0.2% less efficient relative to the Betz coefficient.

The most optimal twist coefficient turned out to be just under 1... meaning a linear decrease in twist was deemed optimal! This result was in contrast with the literature. The optimal chord coefficient was 2/3, which aligns closely with what can be seen in most turbine design.

The function also had no signs of ever reaching 0.444 either, as shown in Figure X, the logarithmic nature of progress with the ga function makes it highly improbable that a few lucky mutations get this design to jump up 0.001 values of Cp, when it took 20 generations to go up 0.001 previously.

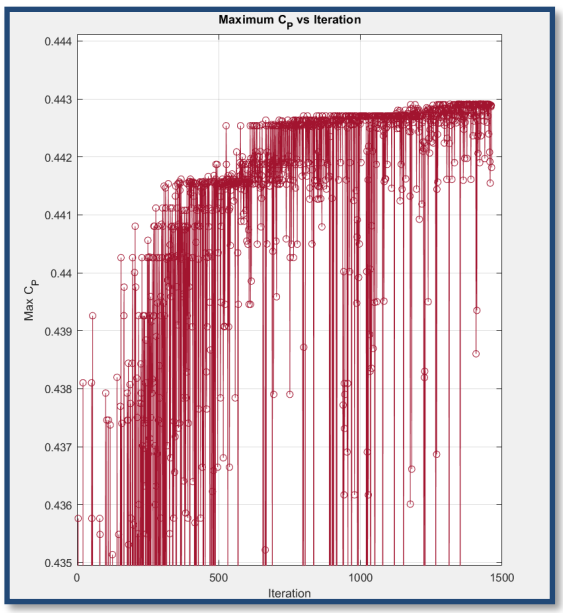


Fig 16: Max Cp against Iteration

Visualisation

The greatest Cp values were found to be for a root twist of 20.9893 degrees, a tip twist of 4.3643 degrees, a root chord of 0.34535 metres and a tip chord of 0.13136 metres.

As these values were in the linear twist and chord distributions, they were modelled in Fusion360 in order to help the reader visualise the final design.

Highlighted in blue is the optimised blade itself.

The Cp for this model is 0.44394, equating to **74.863%** of Betz's coefficient (0.593). Practical utility-scale wind turbines achieve at peak 75%-80% of the Betz limit (Burton 2001), therefore this result is very promising and appears to be near a professional standard.

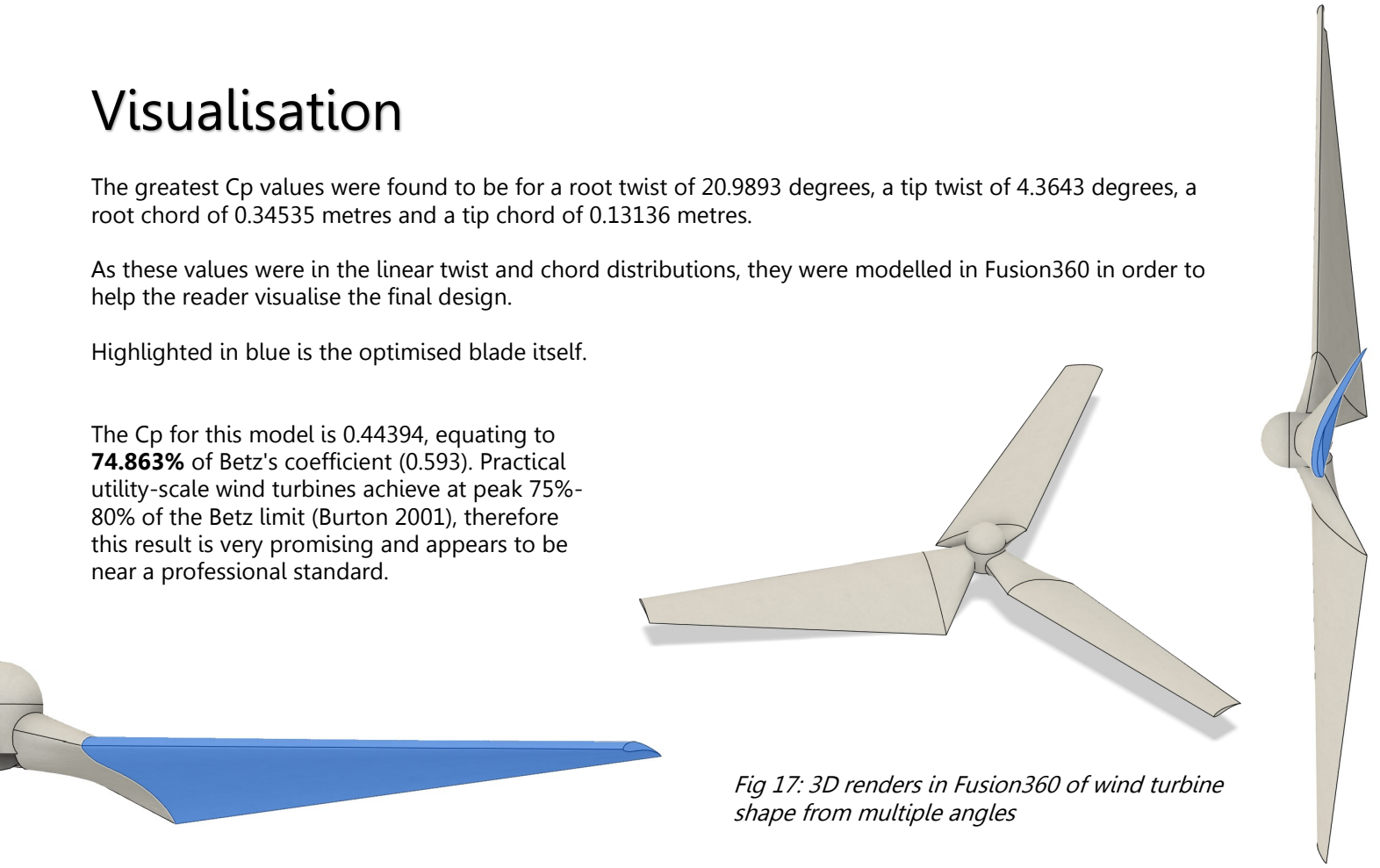


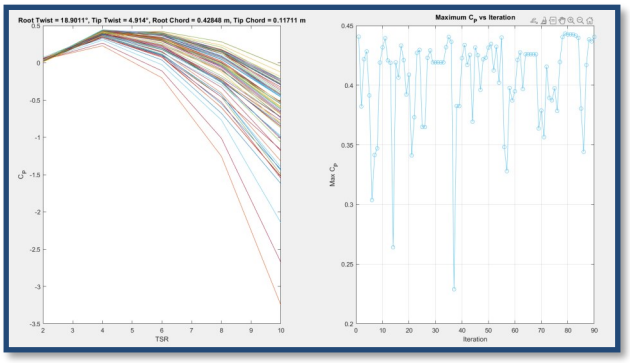
Fig 17: 3D renders in Fusion360 of wind turbine shape from multiple angles

Simulated Annealing

This was attempted at quite a late stage, contrary to how it should be used (ideally to roughly hone in on a global minima with a really wide range of values). This algorithm ran a lot faster than the genetic algorithm, however it did not do a very great job of honing in on the top values for Cp and iterating.

As can be seen in the Max Cp graph against Iterations, the algorithm would continue to waste multiple guesses on very far off data points, preventing it from finding better values.

The algorithm also had the luxury of pre-honed in values from the previous genetic algorithm testing, however it was simply used for a task it is not designed to do. This still performed better than the genetic algorithm with exponentially changing twist and chord.




```
Optimal values found:
Root Twist: 23.5021 degrees
Tip Twist: 4.5068 degrees
Root Chord: 0.46242 meters
Tip Chord: 0.10867 meters
Maximum C_P: 0.44324
```

```
Optimal values found:
Root Twist: 24.9866 degrees
Tip Twist: 4.0027 degrees
Root Chord: 0.50004 meters
Tip Chord: 0.1 meters
Maximum C_P: 0.44341
```

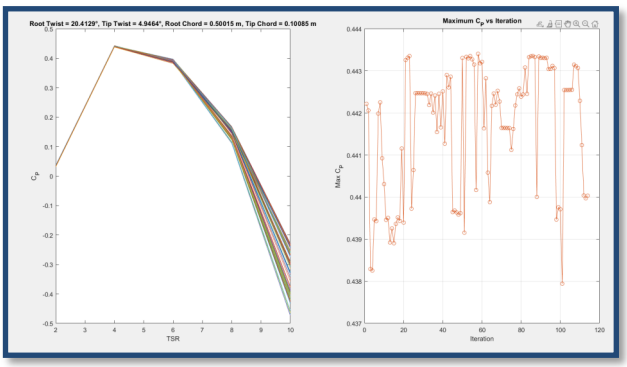


Fig 18: Simulated annealing results in MATLAB

Reflection

Constraints

With more time, revisiting the airfoil constraints could have lead to a superior Cp output. Re-running the genetic algorithm for 50 generations on airfoils SD7034 and S2091 may have resulted in a superior combination for maximising Cp, despite their inferior performance in Salgado's study.

A mixture of airfoils would likely have yielded the best result. Most conventional turbine designs have a lift-optimised airfoil at the root and a drag-optimised airfoil at the tip. Perhaps rerunning the simulation with a combination of the best airfoils at their respective tasks could have reached a higher Cp than using the best all-rounder airfoil available (the S1210).

Experimentation with a 4 blade design may have yielded in higher Cp values, as these airfoils do better in low TSR ranges (Adeseye 2021). A significant enough increase in Cp might be noteworthy or desirable if the use case of the turbine is short-term.

Parameters

There exist more non-linear functions than a simple exponential decrease. Testing a multitude of functions may have lead to a higher Cp value to be found.

Trying the genetic algorithm again, with a very high mutation rate and a very high range for root chord lengths may have yielded in an optimal design very far off the maxima found, however this approach was not done due to a lack of time.

Finding a metric other than maximum Cp may have been a superior approach to determining an ideal wind turbine blade design, as a design optimised for a TSR of 4 may fare poorly if the wind speeds alter slightly. A design that accounts for efficiency at other TSRs and weights that effectively against probability of such TSRs occurring (if the turbine is geared for a TSR of 4 at 5m/s) would likely create a design with greater power output over a meaningful period of time, such as a week.

Improving GA Model

The greatest improvement in run time for the genetic algorithm, assuming equal hardware specifications, would be to run the model in Python using open source add-ons that allow the model to be ran on a GPU, fully utilising the parallel computing power of the computer.

If that failed, finding a variant of the genetic algorithm in Python that allows for CPU parallel processing without crashing would also be a drastic improvement in run times.

Finally, continuing to refine the algorithm with parpool() may have led to a breakthrough where at least a 2x improvement in compute speed could have been attained.

Utilising Other Models

The deep learning agent attempt was very interesting and had the potential to run very rapidly if set up adequately. Finding a method to communicate with Ashes using GPU cores would be a first step in getting this method to work, however much more time would need to be spent in general understanding this algorithm to ensure an appropriate set up.

A different approach to prevent falling into local maxima may have been using Particle Swarm Optimisation (PSO). This may have been a good benchmark to cross-examine results from GA and SA, helping zone in on the best value of Cp possible.

References

Adeseye, K. A., et al., 2021. Estimation of Weibull parameters for wind energy analysis across the UK. IOP Conference Series: Earth and Environmental Science, [online] 801, p.012020. Available at: https://iopscience.iop.org/article/10.1088/1755-1315/801/1/012020/pdf [Accessed 6 Dec. 2024].

Betz, A., 1966. Introduction to the Theory of Flow Machines. Translated by D. G. Randall. Oxford: Pergamon Press.
Burton, T., Sharpe, D., Jenkins, N. and Bossanyi, E., 2001. Wind Energy Handbook. 1st ed. Chichester: John Wiley & Sons, p.65.

Dell'Aversana, P., 2022. Reinforcement learning in optimization problems. Applications to geophysical data inversion. AIMS Geosciences, [online] 8(3), pp.488-502. Available at: https://doi.org/10.3934/geosci.2022027 [Accessed 6 Dec. 2024].

Manwell, J.F., McGowan, J.G. and Rogers, A.L., 2009. Wind Energy Explained: Theory, Design and Application. 2nd ed. Chichester: John Wiley & Sons. Available at: https://ee.tlu.edu.vn/Portals/0/2018/NLG/Sach_Tieng_Anh.pdf [Accessed 6 Dec. 2024].

Salgado, V., Troya, C., Moreno, G. and Molina, J., 2016. Airfoil selection methodology for small wind turbines. International Journal of Renewable Energy Research, [online] 6(4). Available at: https://doi.org/10.20508/ijrer.v6i4.4642.g6930 [Accessed 6 Dec. 2024].

Shu, Z.R. and Jesson, M., 2021. Estimation of Weibull parameters for wind energy analysis across the UK. Journal of Renewable and Sustainable Energy, [online] 13(2), p.023303. Available at: https://doi.org/10.1063/5.0038001 [Accessed 6 Dec. 2024].

Sørensen, J.N. and Myken, A., 1992. Unsteady actuator disc model for horizontal axis wind turbines. Journal of Wind Engineering and Industrial Aerodynamics, [online] 39(1–3), pp.139–149. Available at: https://doi.org/10.1016/0167-6105(92)90540-Q [Accessed 6 Dec. 2024].

Tony Burton et al., (ed), 2001. Wind Energy Handbook, John Wiley and Sons.