This problem and task contained within this report was formulated by Dr. Zhangming Wu of Cardiff University in 2025. The solution and the code are the author's own.

## Table of Contents

## Table of Figures and Tables

## Nomenclature

| Term | Meaning |
|---|---|
| $A_1$ | Cross-Sectional Area of Member 1 and 3 |
| $A_2$ | Cross-Sectional Area of Member 2 |
| X | Tuple with $A_1$ and $A_2$ |
| $f_1(X)$ | Objective 1 |
| $f_2(X)$ | Objective 2 |
| P | Load |
| H | Truss Dimension |
| E | Young's Modulus |
| $\sigma_i,\ i = 1,2,3$ | Stress Constraints for Members 1,2,3. |
| SLSQP | Sequential Least Squares Quadratic Programming |
| SQP | Sequential Quadratic Programming |
| BFGS | Broyden–Fletcher– Goldfarb–Shanno (algorithm) |
| KKT | Karush-Kuhn-Tucker (conditions) |

## 1. Introduction (385 Words)

Engineering design often requires making informed trade-offs between conflicting constraints. Typically, optimisation seek to minimise an objective function defined by design variables, subject to certain constraints. For example, in aerospace engineering, efficiency can be improved by reducing weight although this might compromise structural strength (Poll 2014). Thus, optimisations, like that performed by Wong et al. (2018), might aim to reduce weight via topology optimisation subject to constraints such as maximum allowable stress or mass retention. Similarly, optimisation algorithms can be used in non-structural applications such as in logistics processes. Pečený et al. (2020) discuss the use of optimisation in transport logistics to decrease costs or improve efficiencies.

This report concerns the optimisation of a three-member truss shown in Figure 1.1. The design variables are $A_1$ and $A_2$ which represent the cross-sectional areas of members 1 and 3, and member 2, respectively. (Wu 2025a)

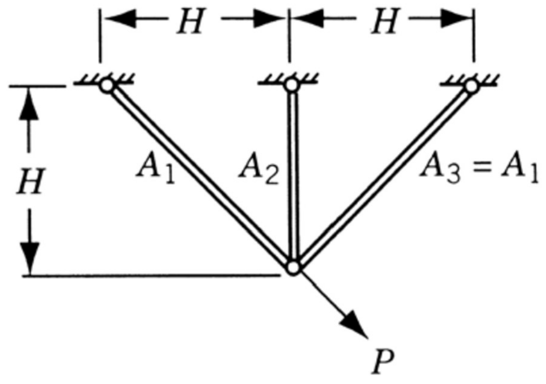


**Figure 1.1** – The Truss Subject to Optimisation

The truss is subject to a load (*P*) and there were two objective functions $f_1(\boldsymbol{X})$ and $f_2(\boldsymbol{X})$, which concern weight and vertical deflection respectively. Three optimisations were carried out with the following objectives: minimise weight [$f_1(\boldsymbol{X})$], minimise vertical deflection [$f_2(\boldsymbol{X})$], and minimise both weight and vertical deflection with equal weighting [$0.5 \cdot f_1(\mathbf{X}) + 0.5 \cdot f_2(\boldsymbol{X})$]; herein referred to as optimisations 1, 2, and 3, respectively. The optimisation was subject to stress constraints for each member and bounds for the design variables $A_1$ *and* $A_2$, giving the optimisation problem as follows (Wu 2025a):

Find:

$$\boldsymbol{X} = \begin{Bmatrix} A_1 \\ A_2 \end{Bmatrix}$$

which minimises:

$$f_1(\boldsymbol{X}) = 2\sqrt{2}A_1 + A_2 \qquad 1.1$$

or/and

$$f_2(\boldsymbol{X}) = \frac{PH}{E}\left[\frac{1}{A_1 + \sqrt{2}A_2}\right] \qquad 1.2$$

subject to:

$$\sigma_1 \le \sigma^{(u)}$$

$$\sigma_2 \le \sigma^{(u)}$$

$$\sigma_3 \le \sigma^{(l)}$$

$$A_i^{(u)} \ge A_i \ge A_i^{(l)}, i = 1,2$$

where:

$$\sigma_1(\boldsymbol{X}) = \text{stress in member 1} = P\left[\frac{A_2 + \sqrt{2}A_1}{\sqrt{2}A_1^2 + 2A_1A_2}\right] \qquad 1.3$$

$$\sigma_2(\boldsymbol{X}) = \text{stress in member 2} = P\left[\frac{1}{A_1 + \sqrt{2}A_2}\right] \qquad 1.4$$

$$\sigma_3(\boldsymbol{X}) = \text{stress in member 3} = P\left[\frac{A_2}{\sqrt{2}A_1^2 + 2A_1A_2}\right] \qquad 1.5$$

and

$$P = 20, \quad H = 1, \quad E = 1, \quad \sigma^{(u)} = 20, \quad \sigma^{(l)} = -15, \quad A_i^{(u)} = 5.0, \quad A_i^{(l)} = 0.1$$

(Wu 2025a)

The optimisation was performed and results were presented using the various Python libraries, including SciPy's scipy.optimize.minimize() for the non-linear optimisation. The program also presented the solution and iteration data, calculated the active constraints, and presented figures to enable visualisation of convergence performance and the optimal solution in relation to the feasible region and objective function contours.

The aim of this report is to present the optimisation methodology used, present the optimal solutions for this problem, evaluate the performance of each objective function and the program more generally, and discuss the results and how the optimisation improved the design of the truss. The report will also consider how optimisation can be used in future designs and how this optimisation might have been improved.

## 2. Methodology and Implementation (514 Words)

### 2.1. Main Algorithms, Libraries, Data Structures, and Techniques

The core functionality of the programme was the numerical optimisation performed by the scipy.optimize.minimize() function. Within minimize(), the sequential least squares quadratic programming (SLSQP) method was used (SciPy Community 2023). SLSQP is a variation of the sequential quadratic programming (SQP) first introduced by Kraft (1988). Which remains one of the most modern and efficient techniques for solving medium-scale, smooth, and constrained optimisation problems (Nocedal and Wright 2006).

SQP solves nonlinear optimisation problems by applying Newton's method to a system of nonlinear equations derived from the Karush–Kuhn–Tucker (KKT) conditions and the Lagrangian of the constrained problem (Kraft 1988, Wu 2025b). In SLSQP, the quadratic subproblem is reformulated as a least-squares problem, which improves numerical efficiency, stability, and implementation (Kraft 1988). The search direction is obtained from this subproblem, and the Hessian of the Lagrangian is updated iteratively using a quasi-Newton BFGS approach (Kraft 1988, Wu 2025c).

The remainder of the programme was designed to prepare input data for the numerical optimisation and present the results. NumPy was used for efficient array handling and mathematical operations across the design region, matplotlib.pyplot was used for high-quality visualisation, and pandas.DataFrame was used to store iteration history and summarise results. This structure enabled clear and structured storage, simple plotting, and easy inspection of raw data.

### 2.2. The Programme Structure, Key Functions, and Implementation Techniques

The Python script was structured with clear separation between global variable definition, objective and constraint function declarations, optimisation execution, and plotting functions. This modular approach supported readability, maintainability, and reuse.

Three objective functions (two single-objective and one multi-objective) were defined to represent the optimisation goals, they returned the objective values for given design variables. The behaviour constraints were implemented as a single function returning a list of behvaiour constraint values, all written in inequality form and rearranged to meet SciPy's expectation of the KKT condition where $g_j(X) \geq 0$. The geometric constraints were implemented directly through the bounds argument of the minimize function, improving numerical efficiency.

Two plotting functions were implemented: contourplotfunc() generated contour plots of the objective function with shaded feasible regions and marked optima, while convergenceplotfunc() displayed objective value versus iteration to evaluate convergence. Contour plots used a custom colour map with manual label placement to clearly highlight contour levels, constraint boundaries, and the optimal solution. Both were tailored for clarity and informative visual output.

The minimise function was called from within a wrapper function, optifunc(), which managed the optimiser setup; It also passed relevant data to the plotting functions, contained a callback function to store iteration data, and returned the results to the main programme. The callback function logged the design variables and objective values at each iteration, enabling convergence analysis and graphical plotting via panda data frames. This enabled simple repeatability for multiple optimisations.

The main programme looped over the defined objective functions, calling optifunc() for each, determining active constraints, and calculating the values for $f_1(\mathbf{X})$ *and* $f_2(\mathbf{X})$ at each optimum. The final results were summarised and displayed via a data frame. A flowchart describing the code logic is shown in Figure 2.1.

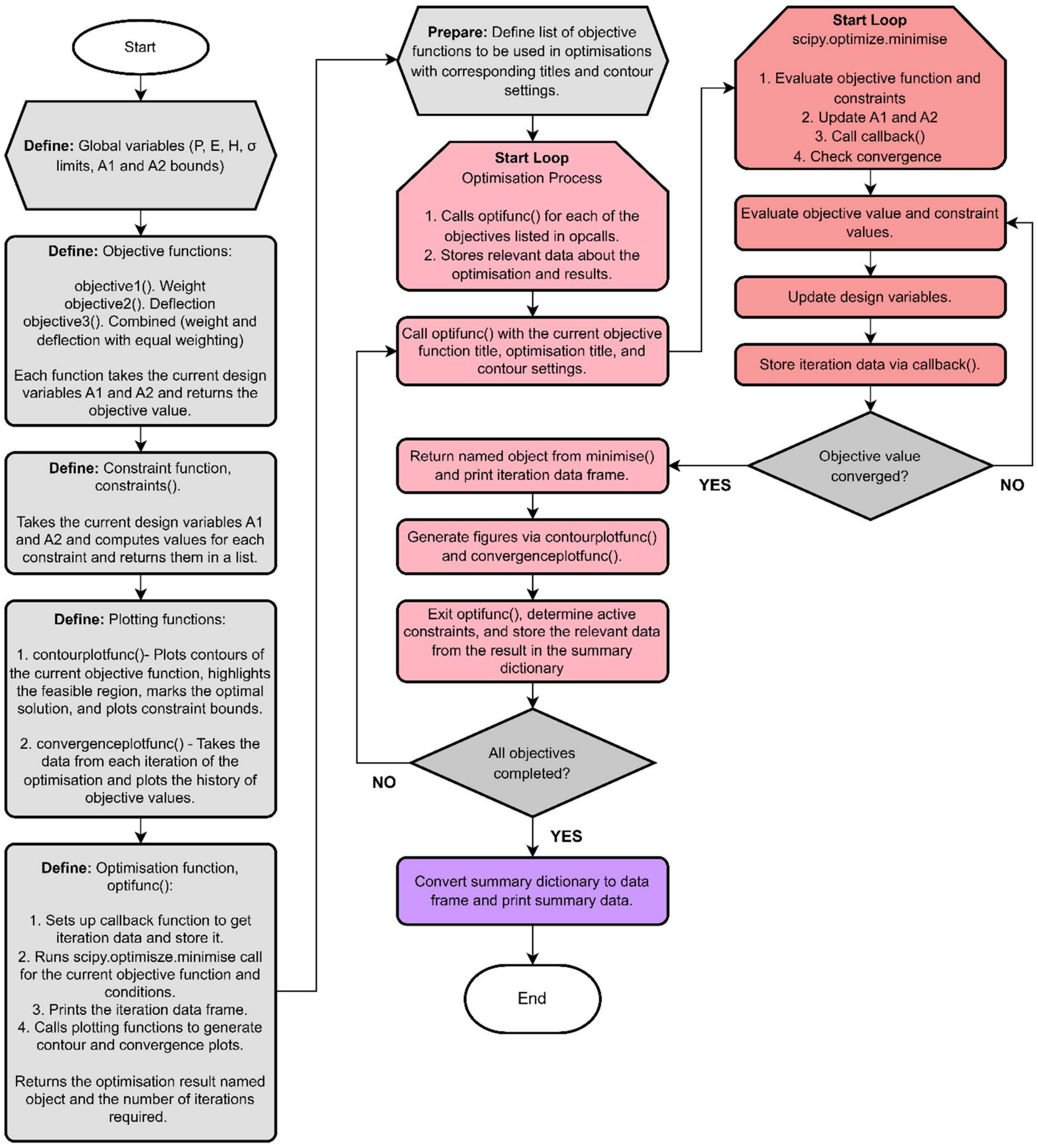

Start
**Define:** Global variables (P, E, H, σ limits, A1 and A2 bounds)
**Define:** Objective functions:
objective1(). Weight
objective2(). Deflection
objective3(). Combined (weight and deflection with equal weighting)
Each function takes the current design variables A1 and A2 and returns the objective value.
**Define:** Constraint function, constraints().
Takes the current design variables A1 and A2 and computes values for each constraint and returns them in a list.
**Define:** Plotting functions:
1. contourplotfunc()- Plots contours of the current objective function, highlights the feasible region, marks the optimal solution, and plots constraint bounds.
2. convergenceplotfunc() - Takes the data from each iteration of the optimisation and plots the history of objective values.
**Define:** Optimisation function, optifunc():
1. Sets up callback function to get iteration data and store it.
2. Runs scipy.optimiscze.minimise call for the current objective function and conditions.
3. Prints the iteration data frame.
4. Calls plotting functions to generate contour and convergence plots.
Returns the optimisation result named object and the number of iterations required.
**Prepare:** Define list of objective functions to be used in optimisations with corresponding titles and contour settings.
**Start Loop**
Optimisation Process
1. Calls optifunc() for each of the objectives listed in opcalls.
2. Stores relevant data about the optimisation and results.
Call optifunc() with the current objective function title, optimisation title, and contour settings.
**Start Loop**
scipy.optimize.minimise
1. Evaluate objective function and constraints
2. Update A1 and A2
3. Call callback()
4. Check convergence
Evaluate objective value and constraint values.
Update design variables.
Store iteration data via callback().
Objective value converged?
YES
NO
Return named object from minimise() and print iteration data frame.
Generate figures via contourplotfunc() and convergenceplotfunc().
Exit optifunc(), determine active constraints, and store the relevant data from the result in the summary dictionary
All objectives completed?
NO
YES
Convert summary dictionary to data frame and print summary data.
End

**Figure 2.1** – Programme Logic Flowchart

## 3. Results (541 Words)

A summary of the results of the three optimisations is shown in Table 1. Figures 3.1, 3.2, and 3.3 show contour plots and convergence plots side by side for optimisations 1, 2, and 3, respectively.

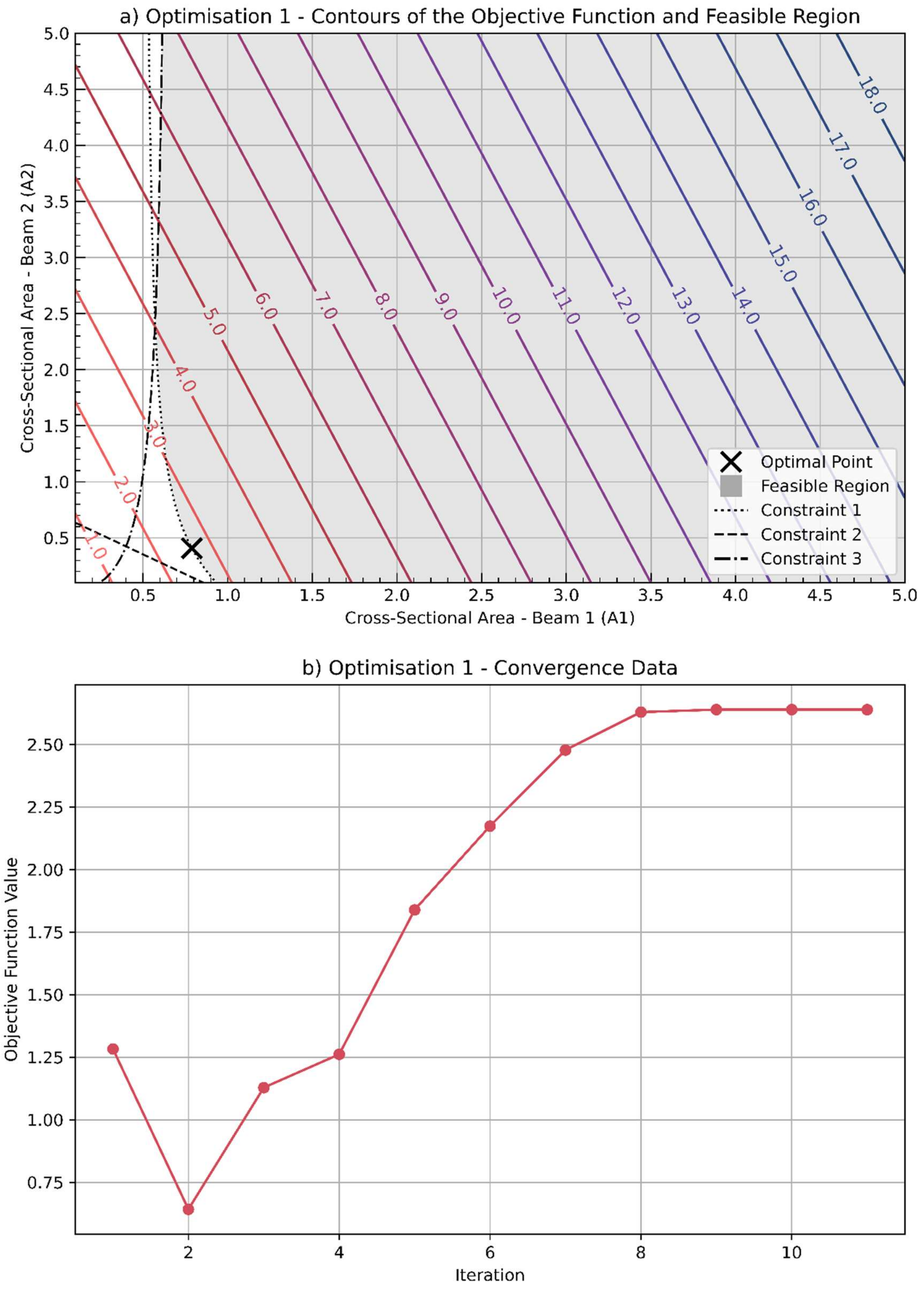
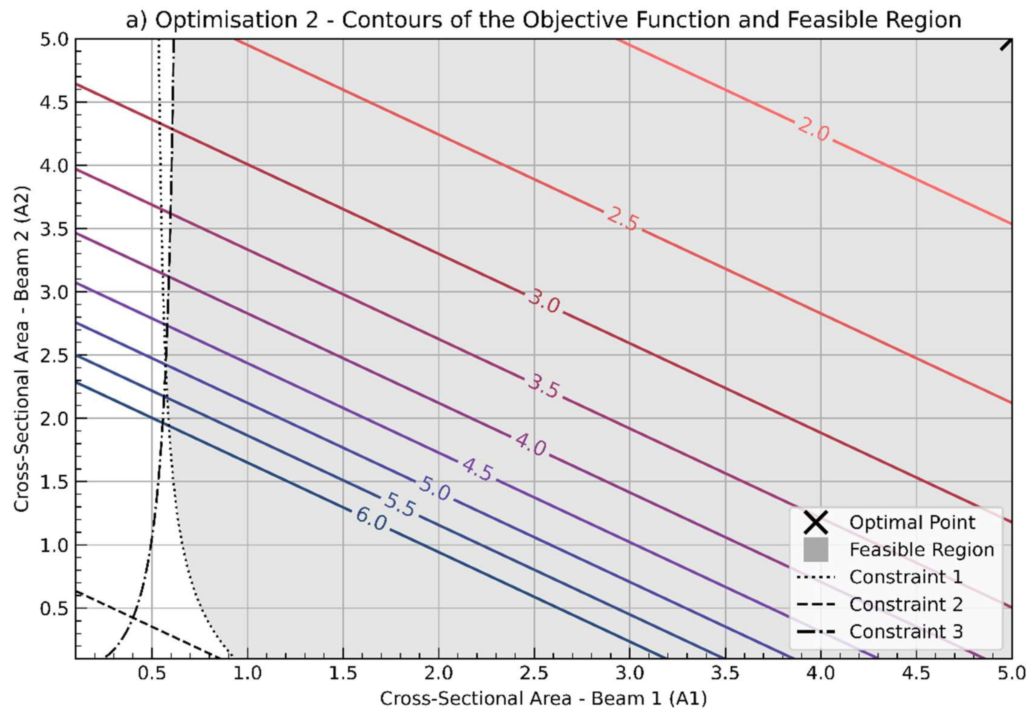


**Figure 3.1** – Optimisation 1 Plots



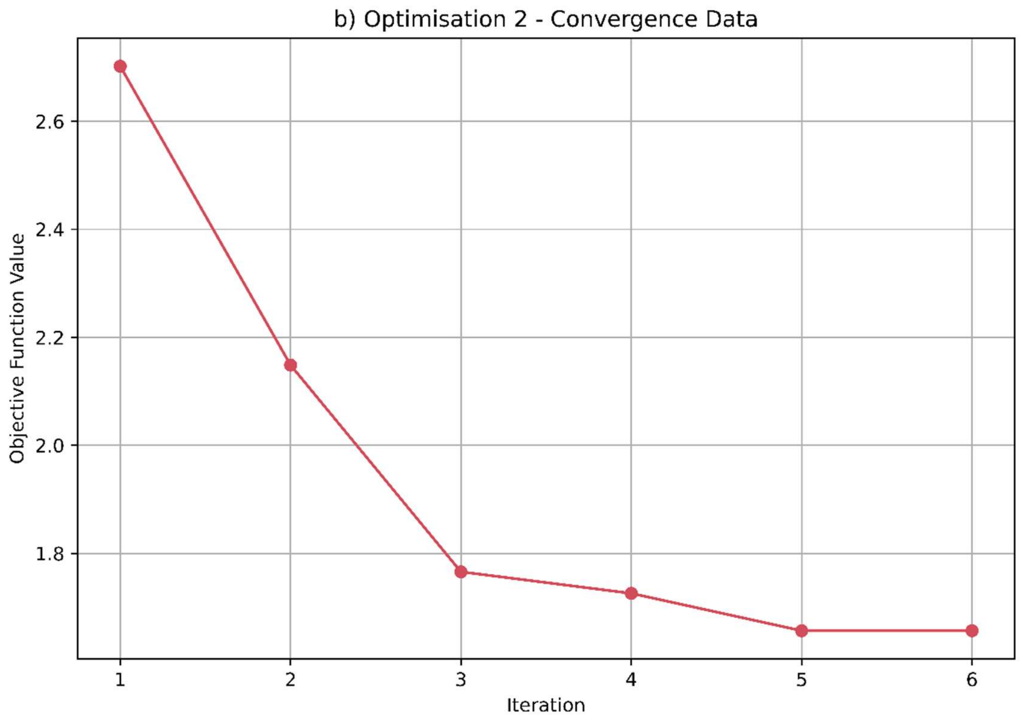
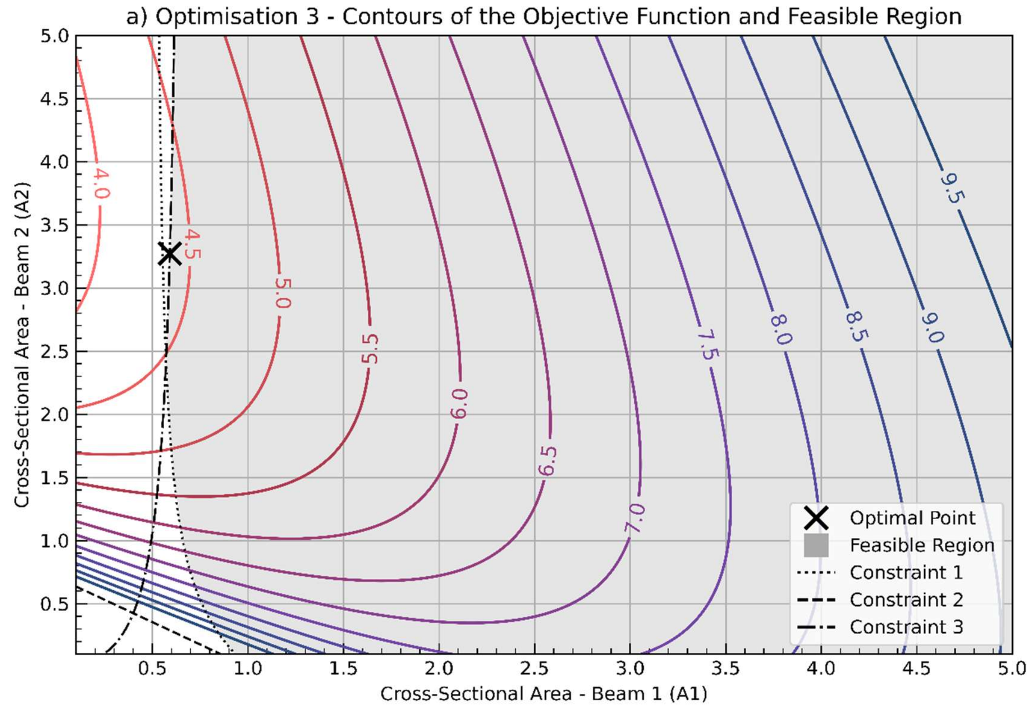


**Figure 3.2** – Optimisation 2 Plots



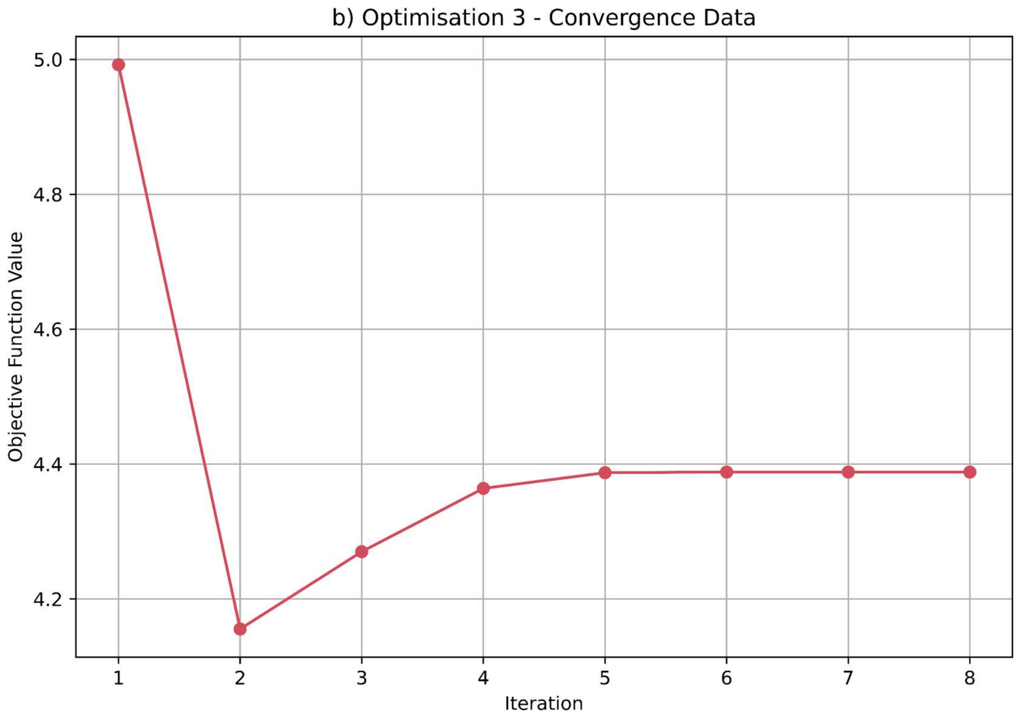


**Figure 3.3** – Optimisation 3 Plots

**Table 1** – Optimisation Results Summary

| Optimisation Number | $A_1$ | $A_2$ | Objective Value | Iterations | Active Constraints | $f_1(X)$ | $f_2(X)$ |
|---|---|---|---|---|---|---|---|
| 1 | 0.7887 | 0.4082 | 2.6390 | 10 | Constraint 1 | 2.6390 | 14.6410 |
| 2 | 5.0000 | 5.0000 | 1.6569 | 6 | None | 19.4121 | 1.6569 |
| 3 | 0.5912 | 3.2722 | 4.3883 | 8 | Constraint 3 | 4.9442 | 3.8324 |

Three optimisations were run on the truss structure using different objective functions: (1) minimise weight, (2) minimise deflection, and (3) minimise weight and deflection with equal weighting. Each optimisation used the same constraints and an initial guess of 2 for both the design variables $A_1$ and $A_2$; This initial guess was arbitrary, the only requirement was that the initial guess be feasible.

Table 1 shows that optimisation 1 resulted in cross-sectional area values of 0.7887 and 0.4082 for $A_1$ and $A_2$, respectively, with an objective value of 2.6390. The solution converged after 10 iterations and Figure 3.1b shows that, despite initially searching in the wrong direction, the optimisation iterations were stable. Figure 3.1a and Table 1 show that the active constraint in optimisation 1 was constraint 1 (the stress in member 1). The values of $f_1(\mathbf{X})$ and $f_2(\mathbf{X})$ were 2.6390 and 14.6410.

Similarly, from Table 1, optimisation 2 resulted in cross-sectional area values of 5.0000 for both $A_1$ and $A_2$. The objective value in this case was 1.6569. The solution converged after 6 iterations, and Figure 3.2b shows that the optimisation was entirely smooth. Table 1 shows that optimisation 2 was not limited by any of the three constraints, however, it is clear from logical interpretation and Figure 3.2a that the optimisation was limited by the upper bound of 5 for both design variables $A_1$ and $A_2$. The values of $f_1(\mathbf{X})$ and $f_2(\mathbf{X})$ were 1.6569 and 19.4121.

Table 1 also shows that optimisation 3 observed values of 0.5912 and 3.2722 for $A_1$ and $A_2$, respectively, with an objective value of 4.3883. The solution converged after 8 iterations and Figure 3.3b shows that, despite an initial overshoot of the optimum value, the optimisation iterations were stable. Figure 3.3a and Table 1 show that the active constraint in optimisation 3 was constraint 3 (the stress in member 3). The values of $f_1(\mathbf{X})$ and $f_2(\mathbf{X})$ were 4.9442 and 3.8324.

In all optimisations, convergence was relatively quick and stable, always achieving convergence within 10 iterations. In optimisations 1 and 2, large objective values for deflection and weight, respectively, were observed and they contrasted small values of the corresponding weight and deflections. Optimisation 3 struck a middle ground, with no extreme values for $f_1(\mathbf{X})$ and $f_2(\mathbf{X})$. Furthermore, a comparison of the design variables $A_1$ and $A_2$ in optimisations 1 and 3 show that $A_1$ decreased by 25.06% and $A_2$ increased by 701.63% in the multi-objective optimisation. Figures 3.1a, 3.2a, and 3.3a show that constraint 2 does form part of the boundary of the feasible region, indicating that it was never active and that stress in member 2 was not limiting. In terms of the design, optimisation 3 suggests that the balanced design, compromising between weight and deflection, had a small area $A_1$ and a relatively large cross-sectional area $A_2$.

## 4. Discussion (435 Words)

The optimum results showed that in optimisation 1, the minimisation of weight resulted in a large deflection, in optimisation 2, the minimisation of deflection resulted in a large weight, and in optimisation 3, the balancing of both weight and deflection resulted in an optimum that was a middle ground between weight and deflection. These results were as expected when considering fundamental solid mechanics.

Optimisation 2 was somewhat limited in its contribution to the understanding of the problem. From Figure 3.2a, it can be seen that the optimum point would only ever find the upper bounds of the design variables – an intuitive result because, to minimise deflection, the largest possible beams are required. This highlighted the effectiveness of optimisation 3; By considering weight alongside deflection, it was ensured that the design would not be unduly heavy nor have large deflection.

As constraint 2 was never active, this could have been removed from the optimisation with no effect on the optimum result and a reduction in computational time. Furthermore, this result indicates that the stress in member 2 is not limiting and that member 2 primarily serves to stiffen the truss.

In terms of the physical meanings of the results, optimisation 1 had small values of $A_1$ and $A_2$ which meant the truss, although lightweight, was weak. In optimisation 2, the values of $A_1$ and $A_2$ reached the maximum of 5, a design which was rigid but less practical than desired and not an efficient use of material. The balance struck in optimisation 3 resulted in a slightly lower value of $A_1$ but increased the value of $A_2$ by 706%. This indicated that the most balanced design should use a large cross-sectional area for member 2, to reduce deflection, and a low cross-sectional area for members 1 and 3, to reduce weight. This shows that the program ensured good material efficiency in the truss design because the smaller member, which could be widened with the lowest impact on weight, was used for stiffening the truss. Between optimisations 1 and 3, the active constraint switched from $\sigma_1$ to $\sigma_3$, suggesting that the design in optimisation 1 supported the load mostly by tension of member 1, and in optimisation 3, member 2 supported most of load and the limiting factor became the compressive stress in member 3.

Overall computational performance was good, as indicated by the low iteration numbers in each optimisation. The plots aided the interpretation of the results by providing good visualisations. However, the programme could have been improved by incorporating additional objectives such as minimise cost or manufacturing considerations such as prefer beams of similar dimensions.

## 5. Conclusion (227 Words)

This study successfully optimised the three-member truss for the three objectives, with the cross-sectional areas of the beams as design variables and member stresses as constraints. It was shown that minimising weight alone causes a significant increase in deflection and that minimising deflection alone causes a significant increase in the weight. Optimisations 1 and 2, whilst feasible, exhibited real-world impracticalities that were balanced by the introduction of objective 3, used in optimisation 3, which assigned equal weighting to the weight and deflection objective functions. The resulting values of design variables $A_1$ and $A_2$ were 0.5912 and 3.2722, respectively, meaning the optimum truss had a large cross-sectional area in member 2, relative to that in members 1 and 3.

The findings confirmed fundamental structural mechanics principles and the presence of a trade-of between stiffness and weight. Furthermore, the study demonstrated how the design variables affected the load distribution, with a notable observation being the shift in the active constraint from $\sigma_1$ to $\sigma_3$.

From this study, it was shown that optimisation offers a powerful, systematic method for exploring design trade-offs. Future work could enhance realism by introducing cost as a constraint, preferring consistent beam sizing, or modelling more complex load cases. Alternative solvers could also be explored to test the robustness of the solution or different initial guesses could be used to test the efficiency of the programme.

## 6. References

Kraft, D. 1988. *A software package for sequential quadratic programming*.  Köln, Germany.

Nocedal, J. and Wright, S. J. 2006. *Numerical optimization*. Springer series in operations research. 2nd ed. New York: Springer.

Pečený, L., Meško, P., Kampf, R. and Gašparík, J. 2020. Optimisation in Transport and Logistic Processes. *Transportation Research Procedia* 44, pp. 15-22. Available at: https://www.sciencedirect.com/science/article/pii/S2352146520300533doi: https://doi.org/10.1016/j.trpro.2020.02.003

Poll, D. I. A. 2014. On the application of light weight materials to improve aircraft fuel burn – reduce weight or improve aerodynamic efficiency? *The Aeronautical Journal* 118(1206), pp. 903-934. Available at: https://www.cambridge.org/core/product/E33367DAFEB6491D16415D9D5EBE2E53doi: 10.1017/S0001924000009611

SciPy Community. 2023. scipy.optimize.minimize — SciPy v1.10.1 Manual. Available at: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html [Accessed: 19/04/2025].

Wong, J., Ryan, L. and Kim, I. Y. 2018. Design optimization of aircraft landing gear assembly under dynamic loading. *Structural and multidisciplinary optimization* 57(3), pp. 1357-1375. doi: 10.1007/s00158-017-1817-y

Wu, Z. 2025a. EN3098 Engineering Optimisation with Python Coursework 2024/25. Cardiff: Cardiff University, School of Engineering.

Wu, Z. 2025b. EN3098 Engineering Optimization with Python – Lecture 4: Nonlinear Programming I. Cardiff: Cardiff University, School of Engineering.

Wu, Z. 2025c. EN3098 Engineering Optimization with Python – Lecture 5: Nonlinear Programming II. Cardiff: Cardiff University, School of Engineering.