Department of Aerospace and Geodesy
TUM School of Engineering and Design
Technical University of Munich

Chapter 7:

Advanced Topics

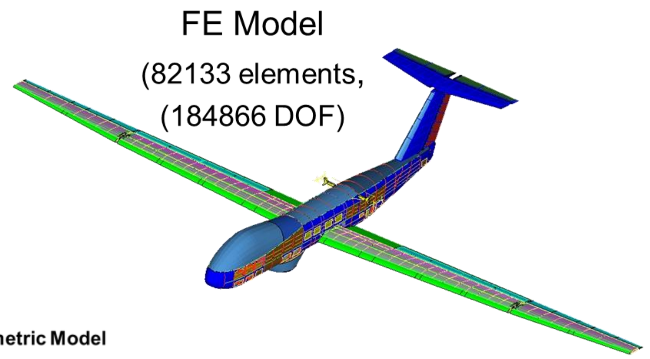
Layout

- Active Set

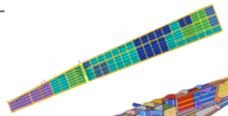
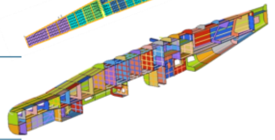
- Dealing with non-smooth functions for gradient algorithms (kinks, steps etc.)

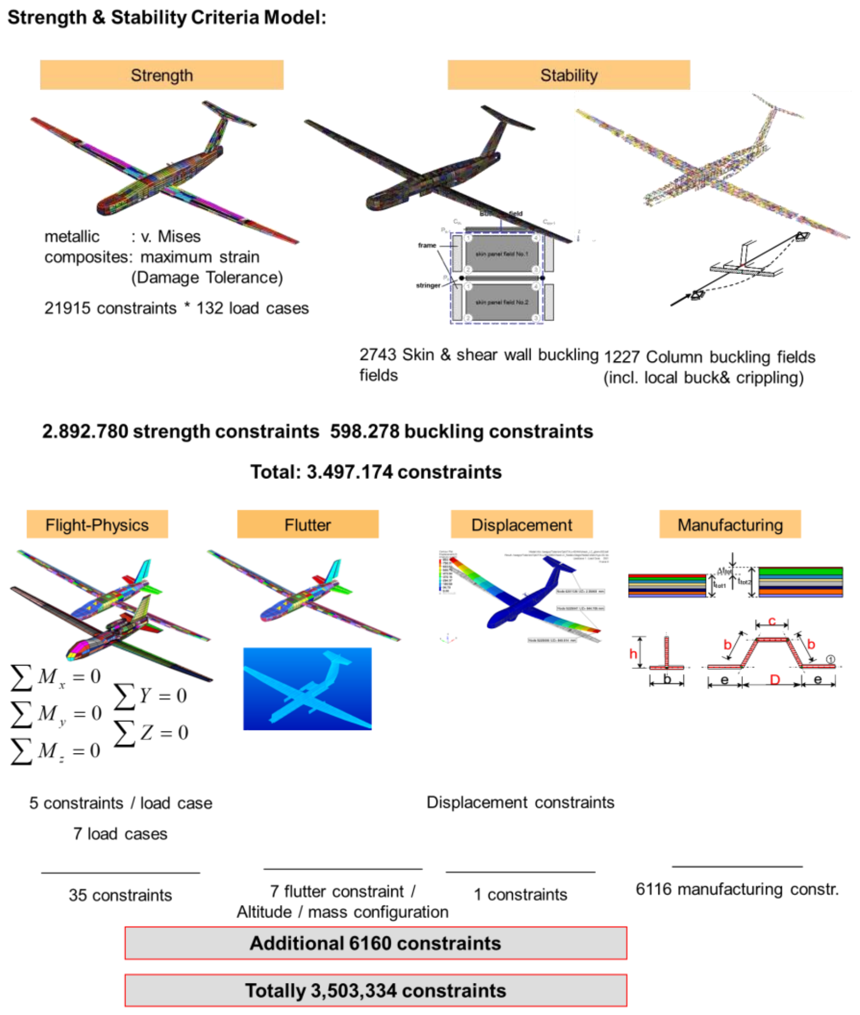
The Active Set Strategy

- In larges-scale optimisation problems the big number of primarily inequality constraints looms up to serious issue e.g. in structure optimisation as can be seen in the following representative multidisciplinary optimisation:



Full Parametric Model

Component	Design Parameter	No. Design Variables	
Wing	Skin: composite ply thicknesses Stringers: height and ply thicknesses Spars: cap cross-sectional areas, web thickness	372	
Fuselage	Skin: metallic skin thicknesses Metallic shear walls, frame, longeron thicknesses	358	
Empennage	Skin: composite ply thicknesses Stringers: height and ply thicknesses Spars: cap cross-sectional areas, web thickness	241	
Total		**971**	



Strength & Stability Criteria Model:

Strength | Stability

metallic : v. Mises
composites: maximum strain (Damage Tolerance)
21915 constraints * 132 load cases

2743 Skin & shear wall buckling fields

1227 Column buckling fields (incl. local buck& crippling)

2.892.780 strength constraints 598.278 buckling constraints

Total: 3.497.174 constraints

Flight-Physics | Flutter | Displacement | Manufacturing

$\sum M_x = 0$ $\sum M_y = 0$ $\sum M_z = 0$ $\sum Y = 0$ $\sum Z = 0$

5 constraints / load case
7 load cases

Displacement constraints

35 constraints | 7 flutter constraint / Altitude / mass configuration | 1 constraints | 6116 manufacturing constr.

Additional 6160 constraints

Totally 3,503,334 constraints

The Active Set Strategy – Dealing with inequality constraints

- The problems caused by the huge number of inequality constraints is related to the following:
  - Though we know that at the optimum only n constraints can be active at maximum, with n being the number of design variables, the set of active constraints at the optimum is unknown in advance
  - Considering a big number of constraints during the gradient-based optimisation means calculating and saving their derivatives in the Jacobian:
    - Example: 1000 Design Variables, 100,000,000 Constraints: $\frac{1000 \cdot 100{,}000{,}000 \cdot 8\ Byte}{1024 \cdot 1024 \cdot 1024} \approx 745\ GB$ RAM memory only for Jacobian
  - The time required for calculating the Jacobian with all explicit and implicit components will exceed by far any acceptable timeframe
- Thus, the need to reduce the number of constraints dealt with in one optimisation iteration is great – we need an active set strategy!

The Active Set Strategy – Dealing with inequality constraints

- Inequality constraints differ from equality constraints in so far that they possess several statuses, beside being active or violated, inequality constraints can be additionally satisfied (and redundant)
- When an inequality constraint is satisfied especially at the optimum it is obsolete i.e., it could've been removed from the optimisation problem without any effect on the final result.
- For the sake of developing a numerically efficient algorithm that handles inequality constraints during the iterative solution process, the following important criterion is utilised:

  $\lambda_j \geq 0$ for $j \in [1, m]$ with m the number of inequality constraints
- This criterion enables identifying the inactive i.e. satisfied inequality constraints – these are the constraints with $\lambda_j < 0$ during the iterative process, beside the constraint value itself $g_j(\boldsymbol{x})$
  – It is important to mention that having $\lambda_j < 0$ in one of the intermediate steps (iterations), doesn't mean that this constraint can't be active at the optimum – it is just the status of this constraint at the current design variables vector

The Active Set Strategy – Dealing with inequality constraints

- Based on the sign of the Lagrange multiplier and the sign of the constraint, four cases can be distinguished:
  1. $g_j(\boldsymbol{x}) < 0$ and $\lambda_j < 0$
  2. $g_j(\boldsymbol{x}) < 0$ and $\lambda_j \geq 0$
  3. $g_j(\boldsymbol{x}) \geq 0$ and $\lambda_j < 0$
  4. $g_j(\boldsymbol{x}) \geq 0$ and $\lambda_j \geq 0$
- According to the understanding of the role of the sign of the Lagrange multiplier of an inequality constraint, following algorithm is established:
  – Case 1: Satisfied constraints with negative Lagrange multiplier: If both $g_j(\boldsymbol{x}) < 0$ and $\lambda_j < 0$, the constraint is satisfied and doesn't influence the minimisation i.e., it is currently obsolete. Removing it from the active set i.e., setting $\lambda_j = 0$ is the best approach, as this lets the algorithm focus on "binding" constraints
    - Since the constraint is marked inactive and is removed from the active set, this constraint is not considered in the next sensitivity analysis and is removed from the Jacobian (see "fixed size working array concept" in the following)

The Active Set Strategy – Dealing with inequality constraints

- Case 2: Satisfied constraints with a positive Lagrange multiplier: When $g_j(\boldsymbol{x}) < 0$ but $\lambda_j \geq 0$, this is a "dual feasibility" violation, implying that although the constraint is currently satisfied, it contributes to the objective's minimisation. In this case, it is recommended to keep the constraint in the active set. This approach preserves the influence of the constraint while the algorithm seeks a stationary solution. Usually, the next iterations lead to diminishing value of the Lagrange multiplier until being zero or also negative.
  - The question arises, however, whether is not possible to support convergence by setting the Lagrange multiplier to zero
  - It is also a good numerical practice to count the number of iterations with this constraint “in conflict” in order to remove the constraint from the active set
- Case 3: Violated constraints with a negative Lagrange multiplier: When $g_j(\boldsymbol{x}) \geq 0$ but $\lambda_j < 0$, this indicates that the inequality constraint is not correctly aligned with the direction of improvement (since $\lambda_j < 0$ shouldn't happen for active inequalities). By keeping it in the active set, you maintain feasibility while iteratively adjusting the constraint.
  - In this case – assuming the constraint keeps violated – the value of the Lagrange multiplier must increase in the next iteration and in best cases getting positive (switching to Case 4)
  - However, the question arises whether it is not possible to shorten the duration of this conflict ($g_j(\boldsymbol{x}) \geq 0$ but $\lambda_j < 0$) by setting the Lagrange multiplier to a small positive number ε (why not zero?)

The Active Set Strategy – Dealing with inequality constraints

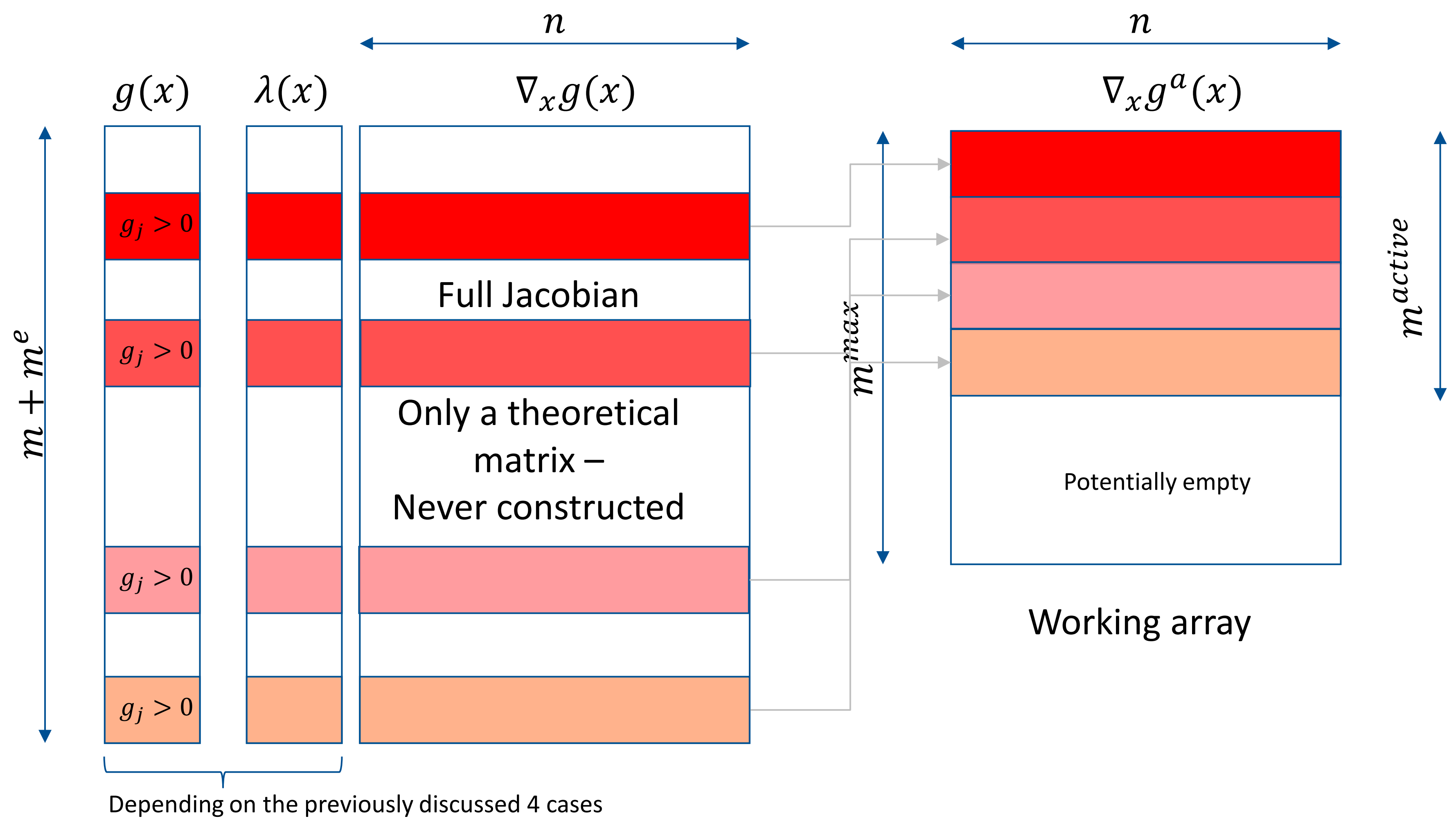
- Case 4: Violated constraints with a positive Lagrange multiplier: When $g_j(\boldsymbol{x}) \geq 0$ but $\lambda_j \geq 0$, this is the most "natural" case, where the constraint is active and influences the search direction calculation correctly. In this case the next iteration should lead to a smaller violation of the constraint i.e., the value $g_j(\boldsymbol{x})$ gets closer to zero

The Active Set Strategy – Software implementation

- In practice, setting small tolerances on $g_j(\boldsymbol{x})$ and λ_j can help mitigating numerical instability. I.e. also, constraints that are close to zero should be kept in the active set avoiding oscillatory behaviour
- Reactivating Constraints: If a removed constraint later nears the boundary ($g_j(\boldsymbol{x}) \geq 0$), it is reintroduced to the active set.
- Finally, a very important software implementation detail must be mentioned. In the previously mentioned four cases, the constraints are added and removed from the active set. The question is how to manage this from software-technical point of view?
  - Managing the memory dynamically i.e., changing the size of the arrays e.g., the Jacobian, is forbidden.
  - To remedy the operating on a dynamic Jacobian array, the “fixed size working array concept” is introduced

Active set strategy – Fixed size working array concept

- Fixed size array concept



- m^{max} is a user-defined parameter depending on the number of design variables, number of constraints – and on the available computer memory
- The optimisation algorithm operates only on $m^{active} \leq m^{max}$

The Active Set Strategy – Fine-tuning

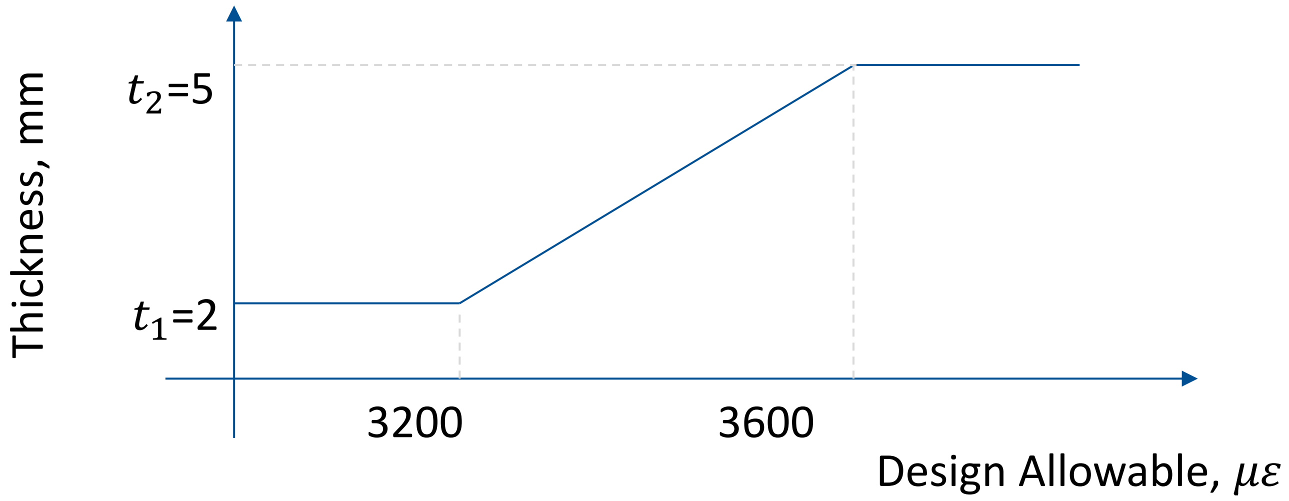
- For large-scale problems, it's efficient to adopt a selective active set strategy that manages constraints to reduce computational overhead. Here's how to develop a scalable approach:
  - Define and manage an Active Set:
    - Maintain two sets of constraints: the active set (binding or nearly binding constraints) and the inactive set (satisfied with some margin).
    - At each iteration, only consider constraints in the active set for sensitivity analysis, Lagrange multiplier updates, and search direction computations.
  - Inactive Constraints:
    - For constraints that are sufficiently satisfied (e.g., $g_j(\boldsymbol{x}) \leq -\varepsilon$ for a small margin ε and $\lambda_j \leq 0$), remove them from the active set entirely, skipping any associated calculations.
    - This saves on both computation and memory requirements, especially with numerous constraints (large-scale optimisation)
  - Dynamic Active-Set Updates:
    - At each iteration, check if any currently inactive constraint $g_j(\boldsymbol{x})$ is nearing violation ($g_j(\boldsymbol{x}) > -\varepsilon$). If so, add it back to the active set and assign it a zero Lagrange multiplier.
    - Conversely, for active constraints where $g_j(\boldsymbol{x}) \leq -\varepsilon$ and the Lagrange multiplier $\lambda_j < 0$, remove them from the active set.
    - In order to avoid numerical instabilities caused by a volatile active set i.e., inequality constraints getting added to the active set in one iteration i, getting removed in the following iteration $i + 1$ and then getting added once more in the next iteration $i + 2$, it is recommended to damp this process by keeping active constraints in the active set for a couple of iterations, though they became inactive.

Dealing with non-smooth functions

- Gradient-based optimisation algorithms are extremely efficient in solving large-scale optimisation problems, however a couple of prerequisites need to be fulfilled
  - Continuous design variables
  - Smooth functions, at least C^1
  - Availability of sensitivities
- In engineering practice, the requirements (leading to the constraints) are very often formulated as non-smooth relationships like
  - Case conditions leading to steps
  - Linear interpolations leading to kinks
- In the following, strategies for dealing with both cases are presented

Dealing with non-smooth functions - Kinks

- A typical example for introducing kinks into the mathematical formulation of engineering requirements in structure mechanics is related to linear interpolations:
  - E.g., the damage tolerance design requirement is based on testing. Since test campaigns are very expensive, it is a common practice to determine the design allowables (strain allowables) for specific thicknesses and to interpolate for any other material thickness in between

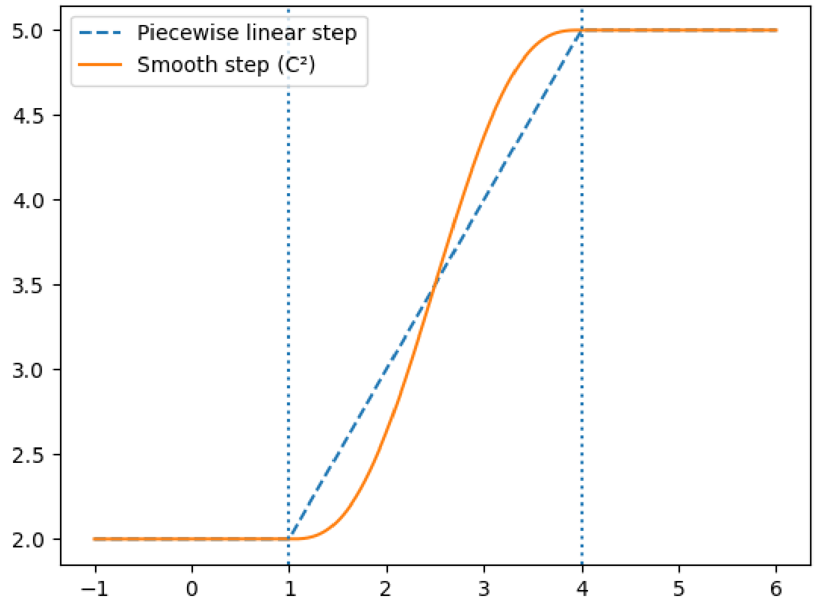


  - This seemingly harmless function causes major difficulties in engineering practice when optimising with gradient methods.

Dealing with non-smooth functions - Kinks

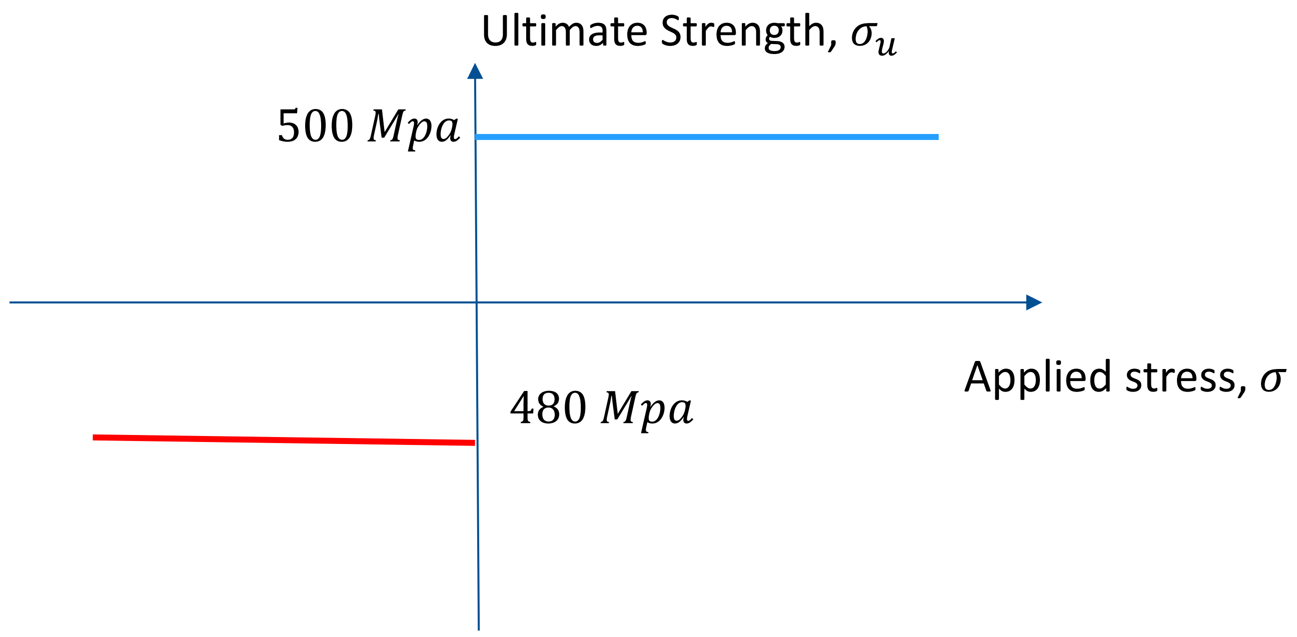
- The sudden change in gradients at the kinks causes oscillations in the optimisation i.e., the design variables might jump from one design area (e.g. ~2mm) to another area (e.g. ~3mm) and jumping back in the subsequent iteration
- To avoid such oscillation, it is recommended to smooth the kinks by a C^n continuous function with $n \geq 1$
- We substitute the original function with a Smoothstep-based ramp:
  - Required behaviour:
    - $f(x) = f_1$ for $x \leq x_1$, $f(x) = f_2$ for $x \geq x_2$, $f(x) = f_1 + (f_2 - f_1)\frac{x - x_1}{x_2 - x_1}$ for $x_1 \leq x \leq x_2$
    - No kinks at x_1 and x_2, i.e. value and slope must match the original function
  - The Smoothstep-based ramp is constructed as follows:
    - $f(x) = f_1 + (f_2 - f_1)\mathrm{R(t)}$ with $t = \frac{x - x_1}{x_2 - x_1}$
    - The ramp function satisfies: $\mathrm{R}(0) = 0$, $\mathrm{R}(1) = 1$, $\acute{R}(0) = 0$ and $\acute{R}(1) = 0$
    - A quintic smoothstep interpolation can be chosen: $\mathrm{R(t)} = \begin{cases} 0, \ t \leq 0 \\ 6t^5 - 15t^4 + +10t^3 \\ 1, \ t \geq 1 \end{cases}$
    - It is possible to improve the interpolation by wrapping the parameter t:

$$\hat{t} = \frac{t^p}{t^p + (1-t)^p}$$



Dealing with non-smooth functions – Step Function

- Step functions occur in structure mechanics e.g. when the sign of the function changes the mechanical behaviour
  - E.g. material strength in tension is different from the one in compression
- The most effective way to account for step functions in the optimisation is to define two separate constraints
  - Though this doubles the number of constraints, however, using an active step strategy mitigates this drawback and provides smooth convergence


Ultimate Strength, σ_u
$500\ Mpa$
Applied stress, σ
$480\ Mpa$