PRINCIPLES OF OPTIMUM DESIGN - FINAL PROJECT: WELDED BEAM DESIGN OPTIMIZATION

AMIR MOHAMMAD VAHEDI

---

1. Problem Statement

The welded beam design optimization problem is a classical engineering optimization problem that aims to minimize the fabrication cost of a welded steel beam used in industrial applications [1]. The problem is formulated as a continuous optimization problem, where the dimensions of the beam are varied to find the cost-minimal design while satisfying a set of mechanical constraints.

1.1. **Design Variables.** The design variables are the dimensions of the welded beam: the width of the weld $h = x_1$, the length of the attached part of the beam $l = x_2$, the height of the beam $t = x_3$, and the thickness of the beam $b = x_4$. These dimensions are illustrated in Figure 1, which shows the beam and the position of the applied load.

1.2. **Objective Function.** The objective is to minimize the total cost of the beam, which includes the costs of materials and labor. The cost function is defined as:

$$\text{Cost} = (C_1 + C_3) \cdot h \cdot l^2 + C_2 \cdot t \cdot b \cdot (L + t)$$

where C_1, C_2 and C_3 are cost coefficients associated with the weld and the beam materials, respectively.

1.3. **Constraints.** The design must adhere to the following constraints to ensure structural integrity and manufacturability:

- Shear stress (τ) in the weld must not exceed the allowable shear stress ($\tau_{\max}$).

$$\tau = \sqrt{(\tau_d)^2 + 2 \cdot \tau_d \cdot \tau_{dd} \cdot \frac{x_2}{2R} + (\tau_{dd})^2} \leq \tau_{\max}$$

  - Primary stress acting over the weld throat τ_d is:

$$\tau_d = \frac{P}{\sqrt{2} \cdot x_1 \cdot x_2}$$

  - Secondary torsional stress τ_{dd} is:

$$\tau_{dd} = \frac{M \cdot R}{J}$$

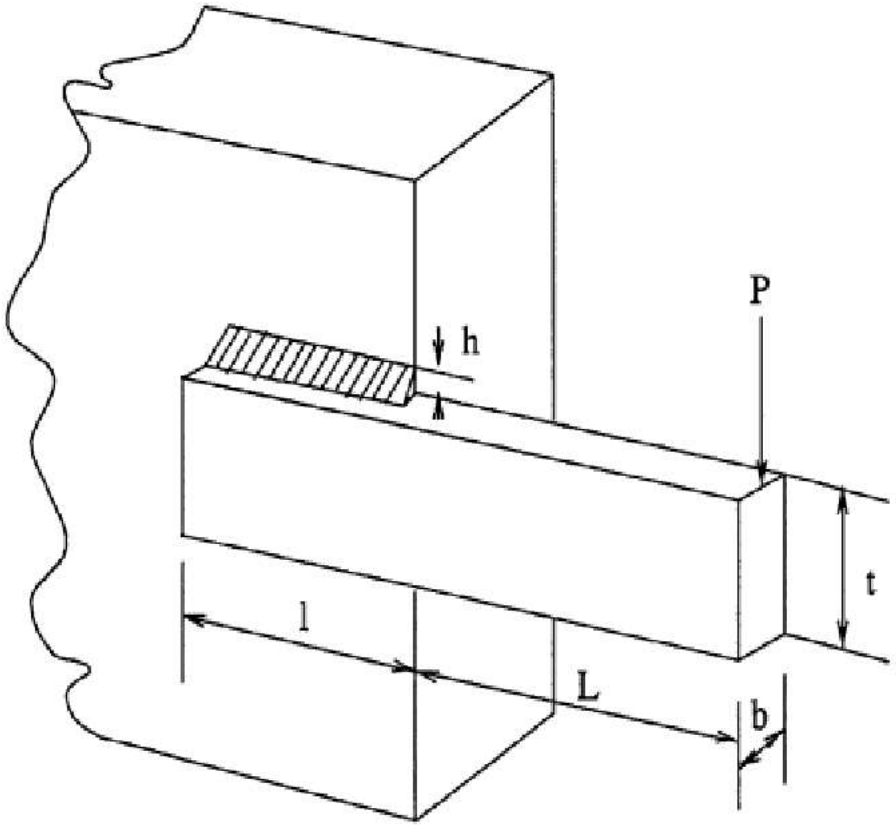

FIGURE 1. Schematic representation of a welded beam design problem, showcasing the key dimensions (h, l, t, b) subject to optimization.

– Moment M of load P about the center of gravity of the weld setup is:

$$M = P\left(L + \frac{x_2}{2}\right)$$

– Polar radius R is:

$$R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2}$$

– Polar moment of inertia J of the weld is:

$$J = 2\left(x_1 \cdot x_2 \cdot \sqrt{2}\left(\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right)\right)$$

- Bending stress (σ) in the beam must not exceed the allowable bending stress ($\sigma_{\max}$).

$$\sigma = \frac{6 \cdot P \cdot L}{b \cdot t^2} = \frac{6 \cdot P \cdot L}{x_4 \cdot x_3^2} \leq \sigma_{\max}$$

- Buckling load (P) should be greater than the applied load [2].

$$P_{\mathrm{c}} = \frac{4.013 \cdot E \cdot \sqrt{\frac{x_3^2 \cdot x_4^6}{36}}}{L^2} \cdot \left(1 - \frac{x_3}{2 \cdot L} \cdot \sqrt{\frac{E}{4 \cdot G}}\right) > P$$

- End deflection (δ) of the beam should not exceed the allowable deflection ($\delta_{\max}$).

$$\delta = \frac{4 \cdot P \cdot L^3}{E \cdot b \cdot t^3} = \frac{4 \cdot P \cdot L^3}{E \cdot x_4 \cdot x_3^3} \leq \delta_{\max}$$

- The weld thickness should not be greater than the beam thickness. (This is a practical manufacturing constraint because the weld typically needs to be at least as thick as the material it is joining to ensure a strong bond and to prevent weak points that could lead to structural failure.)

$$h \leq b$$

- The weld thickness can't be less than the minimum thickness value.

$$0.125 \leq h$$

- Cost-related constraint (The whole expression is likely aiming to keep the cost of the weld and beam material below a certain threshold)

$$C_1 \cdot l^2 + C_2 \cdot t \cdot b \cdot (L + t) \leq 5$$

1.4. **Non-Designable Parameters.** The parameters that are fixed and not subject to optimization include:

- Young's modulus of the beam material, $E = 30 \times 10^6$ psi.
- Shearing modulus, $G = 12 \times 10^6$ psi.
- Length of the overhang, $L = 14$ in.
- Maximum allowable shear stress, $\tau_{\max} = 13,600$ psi.
- Maximum allowable bending stress, $\sigma_{\max} = 30,000$ psi.
- Maximum allowable deflection, $\delta_{\max} = 0.25$ in.
- Applied load, $P = 6,000$ lbs.
- Cost per unit volume of the weld material, $C_1 = 0.10471\,\mathrm{in}^{-3}$.
- Cost per unit volume of the bar, $C_2 = 0.04811\,\mathrm{in}^{-3}$.
- Labor cost per unit weld volume, $C_3 = 1\,\mathrm{in}^{-3}$.

1.5. **Bounds on Design Variables.** Each design variable is subject to specific bounds to ensure practical manufacturability and to meet engineering standards. The bounds for the variables are as follows:

- Width of the weld ($h = x_1$): The height of the beam is constrained to be within a range that ensures structural stability while optimizing for cost and weight. The bounds are set as:
$$0.1 \leq x_1 \leq 2 \,\text{inches}$$
- Length of the attached part of the beam ($l = x_2$): This dimension affects the leverage and thus the bending moments experienced by the beam. It is bounded by:
$$0.1 \leq x_2 \leq 10 \,\text{inches}$$
- Height of the beam ($t = x_3$): The thickness is crucial for resisting bending and shear stresses. Adequate thickness is necessary to prevent excessive deflection and to ensure the beam's integrity under load. The bounds are:
$$0.1 \leq x_3 \leq 10 \,\text{inches}$$
- Thickness of the beam ($b = x_4$): The width of the weld must be sufficient to safely transfer the load between the beam and the supporting structure without failing in shear. The bounds are:
$$0.1 \leq x_4 \leq 2 \,\text{inches}$$

These bounds are chosen based on typical industry standards and past experimental data, which suggest that they strike an effective balance between cost, manufacturability, and safety.

2. Optimization Problem in Standard Form

The objective of the welded beam design optimization problem is to minimize the total cost of the welded beam construction, which includes the costs of materials and labor. The optimization is formulated as follows:

Objective Function. Minimize the cost function $f(x)$, where $x = (x_1, x_2, x_3, x_4)$ represents the design variables:

$$f(x) = C_1 \cdot x_1^2 \cdot x_2 + C_2 \cdot x_3 \cdot x_4 \cdot (L + x_2)$$

Design Variables. The design variables are:

x_1 : Width of the weld (inches)

x_2 : Length of the attached part of the beam (inches)

x_3 : Height of the beam (inches)

x_4 : Thickness of the beam (inches)

Constraints. Subject to the following constraints:

$g_j(x)$: Inequality constraints on shear stress, bending stress, deflection, etc. $\leq 0, \quad j = 1, \ldots, 7$

$$g_1(x) = \tau - \tau_{\max} \leq 0$$
$$g_2(x) = \sigma - \sigma_{\max} \leq 0$$
$$g_3(x) = x_1 - x_4 \leq 0$$
$$g_4(x) = C_1 \cdot x_1^2 + C_2 \cdot x_3 \cdot x_4 \cdot (L + x_2) - 5 \leq 0$$
$$g_5(x) = 0.125 - x_1 \leq 0$$
$$g_6(x) = \delta - \delta_{\max} \leq 0$$
$$g_7(x) = P - P_c \leq 0$$

Bounds. The design variables are also bounded by the following:

$$0.1 \leq x_1 \leq 2 \,\text{inches}$$
$$0.1 \leq x_2 \leq 10 \,\text{inches}$$
$$0.1 \leq x_3 \leq 10 \,\text{inches}$$
$$0.1 \leq x_4 \leq 2 \,\text{inches}$$

3. Optimality Conditions

In determining the optimal design for the welded beam, we consider the Karush-Kuhn-Tucker (KKT) conditions, which provide necessary conditions for a solution to be optimal. Given the non-linear nature of the problem with inequality and equality constraints, the following conditions must be satisfied at a point $\mathbf{x}^*$:

- **Stationarity:** The gradient of the objective function, $f(\mathbf{x})$, and the gradients of any active constraints must be orthogonal to the feasible direction at the optimal point. This condition can be mathematically expressed as:

  $$\nabla f(\mathbf{x}^*) + \sum_{j=1}^{n_g} \sigma_j \nabla g_j(\mathbf{x}^*) + \sum_{k=1}^{n_h} \lambda_k \nabla h_k(\mathbf{x}^*) = 0$$

  where σ_j and λ_k are the Lagrange multipliers for the inequality and equality constraints, respectively. The σ symbol as a Lagrange multiplier can be taken wrong with the bending stress, so we will change its notation to μ. In this problem we don't have any equality constraint so the stationary conditions will be described as follows:

  $$\nabla f(\mathbf{x}^*) + J_g(\mathbf{x}^*)^T \mu = 0$$

- **Primal Feasibility:** All design constraints must be satisfied at the optimal point:

  $$g_j(\mathbf{x}^*) + s_j^2 = 0, \quad \forall j \in \{1, \ldots, n_g\}$$
  $$h_k(\mathbf{x}^*) = 0, \quad \forall k \in \{1, \ldots, n_h\}$$

- **Dual Feasibility:** The Lagrange multipliers associated with the inequality constraints must be non-negative:

$$\mu_j \geq 0, \quad \forall j \in \{1, \ldots, 7\}$$

- **Complementary Slackness:** For each inequality constraint, the product of the Lagrange multiplier and the slack variable must be zero:

$$\mu_j s_j = 0, \quad \forall j \in \{1, \ldots, 7\}$$

The gradients of the objective and constraint functions will be defined based on the specified cost and stress analyses. For instance, the gradient of the cost function, $\nabla f(\mathbf{x})$, will involve the partial derivatives with respect to each design variable, $\frac{\partial f}{\partial x_i}$, computed at the optimal point $\mathbf{x}^*$. Similarly, the gradients of the constraint functions, $\nabla g_j(\mathbf{x})$ will be derived from their respective mathematical expressions and form the Jacobian matrix J_g. The analytical derivation of the gradients of the cost functions, $\nabla f(\mathbf{x})$ and the Jacobian matrix of the constraints J_g are expressed in the following:

- **Gradient of the Objective Function:**

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial x_4}\right]^T$$

$$\frac{\partial f}{\partial x_1} = 2\, x_1\, x_2\, (C_1 + C_3)$$

$$\frac{\partial f}{\partial x_2} = (C_1 + C_3)\, {x_1}^2 + C_2\, x_3\, x_4$$

$$\frac{\partial f}{\partial x_3} = C_2\, x_4\, (L + x_2)$$

$$\frac{\partial f}{\partial x_4} = C_2\, x_3\, (L + x_2)$$

$$\nabla f(\mathbf{x}) = \begin{bmatrix} 2\, x_1\, x_2\, (C_1 + C_3) \\ (C_1 + C_3)\, {x_1}^2 + C_2\, x_3\, x_4 \\ C_2\, x_4\, (L + x_2) \\ C_2\, x_3\, (L + x_2) \end{bmatrix}$$

- **Jacobian matrix of the constraints:**

$$J_g = [\nabla g_1, \nabla g_2, \nabla g_3, \nabla g_4, \nabla g_5, \nabla g_6, \nabla g_7]^T$$

$$\nabla g_i(\mathbf{x}) = \left[\frac{\partial g_i}{\partial x_1}, \frac{\partial g_i}{\partial x_2}, \frac{\partial g_i}{\partial x_3}, \frac{\partial g_i}{\partial x_4}\right]^T$$

$$dom_{g_1} = 2\sqrt{\frac{P^2}{2\,{x_1}^2\,{x_2}^2} + \frac{P^2\left(L+\frac{x_2}{2}\right)}{4\,{x_1}^2\,x_2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)} + \frac{P^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{4}\right)\left(L+\frac{x_2}{2}\right)^2}{8\,{x_1}^2\,{x_2}^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)^2}}$$

$$\frac{\partial g_1}{\partial x_1} = \frac{1}{dom_{g_1}}\Bigg(\frac{P^2}{{x_1}^3\,{x_2}^2} + \frac{P^2\left(L+\frac{x_2}{2}\right)}{2\,{x_1}^3\,x_2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)} + \frac{P^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{4}\right)\left(L+\frac{x_2}{2}\right)^2}{4\,{x_1}^3\,{x_2}^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)^2}$$

$$-\frac{P^2\left(\frac{x_1}{2}+\frac{x_3}{2}\right)\left(L+\frac{x_2}{2}\right)^2}{8\,{x_1}^2\,{x_2}^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)^2} + \frac{P^2\left(\frac{x_1}{2}+\frac{x_3}{2}\right)\left(L+\frac{x_2}{2}\right)}{4\,{x_1}^2\,x_2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)^2} + \frac{P^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{4}\right)\left(\frac{x_1}{2}+\frac{x_3}{2}\right)\left(L+\frac{x_2}{2}\right)^2}{4\,{x_1}^2\,{x_2}^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)^3}\Bigg)$$

$$\frac{\partial g_1}{\partial x_2} = \frac{-1}{dom_{g_1}}\Bigg(\frac{P^2}{{x_1}^2\,{x_2}^3} - \frac{P^2}{8\,{x_1}^2\,x_2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)} + \frac{P^2\left(L+\frac{x_2}{2}\right)}{24\,{x_1}^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)^2} -$$

$$\frac{P^2\left(L+\frac{x_2}{2}\right)^2}{16\,{x_1}^2\,x_2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)^2} + \frac{P^2\left(L+\frac{x_2}{2}\right)}{4\,{x_1}^2\,{x_2}^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)} + \frac{P^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{4}\right)\left(L+\frac{x_2}{2}\right)^2}{24\,{x_1}^2\,x_2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)^3} +$$

$$\frac{P^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{4}\right)\left(L+\frac{x_2}{2}\right)^2}{4\,{x_1}^2\,{x_2}^3\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)^2} - \frac{P^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{4}\right)\left(L+\frac{x_2}{2}\right)}{8\,{x_1}^2\,{x_2}^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)^2}\Bigg)$$

$$\frac{\partial g_1}{\partial x_3} = \frac{1}{dom_{g_1}}\Bigg(\frac{P^2\left(\frac{x_1}{2}+\frac{x_3}{2}\right)\left(L+\frac{x_2}{2}\right)}{4\,{x_1}^2\,x_2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)^2} - \frac{P^2\left(\frac{x_1}{2}+\frac{x_3}{2}\right)\left(L+\frac{x_2}{2}\right)^2}{8\,{x_1}^2\,{x_2}^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)^2} +$$

$$\frac{P^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{4}\right)\left(\frac{x_1}{2}+\frac{x_3}{2}\right)\left(L+\frac{x_2}{2}\right)^2}{4\,{x_1}^2\,{x_2}^2\left(\left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2+\frac{{x_2}^2}{12}\right)^3}\Bigg)$$

$$\frac{\partial g_1}{\partial x_4} = 0$$

$$\nabla g_1(\mathbf{x}) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} \\ \frac{\partial g_1}{\partial x_2} \\ \frac{\partial g_1}{\partial x_3} \\ 0 \end{bmatrix}$$

$$\nabla g_2(\mathbf{x}) = \begin{bmatrix} 0 \\ 0 \\ -\frac{12\,L\,P}{{x_3}^3\,x_4} \\ -\frac{6\,L\,P}{{x_3}^2\,{x_4}^2} \end{bmatrix}$$

$$\nabla g_3(\mathbf{x}) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}$$

$$\nabla g_4(\mathbf{x}) = \begin{bmatrix} 2\,C_1\,x_1 \\ C_2\,x_3\,x_4 \\ C_2\,x_4\,(L+x_2) \\ C_2\,x_3\,(L+x_2) \end{bmatrix}$$

$$\nabla g_5(\mathbf{x}) = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\nabla g_6(\mathbf{x}) = \begin{bmatrix} 0 \\ 0 \\ -\frac{12\,L^3\,P}{\mathrm{E}\,{x_3}^4\,x_4} \\ -\frac{4\,L^3\,P}{\mathrm{E}\,{x_3}^3\,{x_4}^2} \end{bmatrix}$$

$$\nabla g_7(\mathbf{x}) = \begin{bmatrix} 0 \\ 0 \\ -\frac{4.013\,\mathrm{E}\left({x_3}^2\,{x_4}^6\sqrt{\frac{\mathrm{E}}{4\,G}}-L\,x_3\,{x_4}^6\right)}{6\,L^3\sqrt{{x_3}^2\,{x_4}^6}} \\ \frac{4.013\,\mathrm{E}\,{x_3}^2\,{x_4}^5\left(2\,L-x_3\sqrt{\frac{\mathrm{E}}{4\,G}}\right)}{4\,L^3\sqrt{{x_3}^2\,{x_4}^6}} \end{bmatrix}$$

$$J_g = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \frac{\partial g_1}{\partial x_3} & 0 \\ 0 & 0 & -\frac{12\,L\,P}{{x_3}^3\,x_4} & -\frac{6\,L\,P}{{x_3}^2\,{x_4}^2} \\ 1 & 0 & 0 & -1 \\ 2\,C_1\,x_1 & C_2\,x_3\,x_4 & C_2\,x_4\,(L+x_2) & C_2\,x_3\,(L+x_2) \\ -1 & 0 & 0 & 0 \\ 0 & 0 & -\frac{12\,L^3\,P}{\mathrm{E}\,{x_3}^4\,x_4} & -\frac{4\,L^3\,P}{\mathrm{E}\,{x_3}^3\,{x_4}^2} \\ 0 & 0 & -\frac{4.013\,\mathrm{E}\left({x_3}^2\,{x_4}^6\sqrt{\frac{\mathrm{E}}{4\,G}}-L\,x_3\,{x_4}^6\right)}{6\,L^3\sqrt{{x_3}^2\,{x_4}^6}} & \frac{4.013\,\mathrm{E}\,{x_3}^2\,{x_4}^5\left(2\,L-x_3\sqrt{\frac{\mathrm{E}}{4\,G}}\right)}{4\,L^3\sqrt{{x_3}^2\,{x_4}^6}} \end{bmatrix}$$

4. Python Implementation using `scipy.optimize.minimize`

4.1. **Choice of the algorithm.** For the Welded Beam Design problem, which involves both non-linear objective functions and constraints, a suitable choice of algorithm from `scipy.optimize.minimize` is Sequential Least Squares Quadratic Programming (SLSQP), implemented in `SciPy` as method=`'SLSQP'`. This algorithm is well-suited for problems with both equality and inequality constraints and is capable of handling non-linearities in the objective and constraint functions, and bounds on the design variables. SLSQP approximates the problem locally by a quadratic programming subproblem at each iteration, making it effective for engineering design problems where precision in handling constraints is crucial (Other methods are also evaluated but they can't handle the constraints or the bounds on the design variables).

4.2. **Code and Output.** The optimization code implemented for the Welded Beam Design problem is presented below. This code utilizes the `scipy.optimize.minimize` function with the SLSQP method to minimize the objective function subject to the given constraints. It has been structured to start from three different initial guesses to evaluate the robustness and consistency of the optimization process.

The optimization code for the Welded Beam Design project is meticulously crafted, adhering to a structured programming approach. Initially, all necessary libraries and modules are imported, setting the foundation for the subsequent computational tasks.

4.3. **Parameter Initialization.** The problem parameters are defined explicitly, laying out the material properties, geometrical dimensions, and other constants necessary for formulating the optimization model.

4.4. **Objective and Constraints Formulation.** The objective function is formulated to capture the cost minimization goal of the welded beam design. Constraints 1 through 7 are constructed to enforce mechanical requirements and design standards, such as stress limits and deflection criteria.

4.5. **Gradient Calculation.** Given the complexity of constraint functions, especially for Constraint 1, the analytical derivation of gradients could be prone to errors. To mitigate this, symbolic computation capabilities of the `sympy` library are leveraged, utilizing its symbolic gradient calculation feature. This provides a reliable and error-free approach to determining the gradients of the objective function and constraints. During the optimization iterations, the `.subs()` method substitutes the current design variables into the symbolic gradient expressions to evaluate their numerical values.

4.6. **Constraint Evaluation and Progress Callback.** A custom function, `cons_check(x, cons)`, is implemented to evaluate the status of each constraint, categorizing them as active, inactive, or violated based on their evaluated sign and magnitude, in accordance with the standard optimization form.

Furthermore, a callback function is integrated into the optimization routine. Its dual purpose is to monitor the progress of the optimizer, providing real-time updates during the computation, and to compile the history of the objective function values, facilitating the creation of convergence plots post-optimization.

4.7. **Optimization Execution.** The bounds for design variables are set, and the constraints, equipped with their respective Jacobians, are defined in a format compatible with `scipy.optimize.minimize`. The SLSQP method is selected as the optimization algorithm due to its adeptness at handling problems of this nature. Initially verified with a single starting design, the optimization is then robustly tested by executing it from three different seeds, thereby ensuring consistency across various initial conditions.

4.8. **Convergence Analysis.** Upon completion of the optimization runs, convergence graphs are plotted to visually illustrate the optimization trajectory for each seed. These plots are instrumental in confirming the convergence behavior and the reliability of the solutions obtained.

This comprehensive coding structure and methodical approach underscore the reliability and accuracy of the optimization process implemented for the welded beam design project.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 28 22:01:54 2024

@author: vahed
"""

import numpy as np
from scipy.optimize import minimize
from scipy.optimize import differential_evolution, NonlinearConstraint
import matplotlib.pyplot as plt
from sympy import symbols, diff, Matrix

# Parameters
E = 30e6 #Young's modulus (psi)
G = 12e6 #Shearing modulus for the beam material (psi)
L = 14 #Overhang length of the member (inch)
tau_max = 13600 #Design stress of the weld (psi)
sigma_max = 30000 #Design normal stress for the beam material (psi)
delta_max = 0.25 #Maximum deflection (inch)
P = 6000 #Load (lb)
C1 = 0.10471 #Cost per unit volume of the weld material ($ inch-3)
C2 = 0.04811 #Cost per unit volume of the bar ($ inch-3)
C3 = 1 #Labor cost per unit weld volume ($ inch-3)
Nfeval = 1
Objective_history=[]
n_variables= 4 # Number of variables
```

```
28
29
30 # Objective Function
31 def objective(x):
32     x1, x2, x3, x4 = x
33     V_weld = x1**2 * x2 # Volume of weld material (inch3)
34     V_bar = x3 * x4 * (L + x2) # Volume of bar (inch3)
35     f = (C1+C3) * V_weld + C2 * V_bar #Total material cost to be
       minimized.
36     return f
37
38 # Constraints
39 def constraint1(x):
40     # The shear stress at the beam support location cannot exceed
41     # maximum allowable for the material
42     x1, x2, x3, x4 = x
43     # Primary stress acting over the weld throat
44     tau_d = P /(np.sqrt(2)*x1*x2)
45
46     # Moment of P about center of gravity of weld setup
47     M = P * (L + x2/2)
48     R = np.sqrt((x2**2)/4 + ((x1+x3)/2)**2)
49
50     # Polar moment of inertia of weld
51     J = 2 * (x1*x2 * np.sqrt(2)* ((x2**2)/12 + ((x1+x3)/2)**2))
52
53     # Secondary torsional stress.
54     tau_dd= M*R/J
55
56     # Weld stress
57     tau = np.sqrt(tau_d**2 + 2* tau_d * tau_dd * x2/2/R + tau_dd**2)
58     return -(tau - tau_max)
59
60 def constraint2(x):
61     # The normal bending stress at the beam support location cannot
       exceed
62     #the maximum yield strength for the material
63     x1, x2, x3, x4 = x
64     # Bar bending stress
65     sigma = 6 * P * L /(x4 * x3**2)
66     return -(sigma - sigma_max)
67
68 def constraint3(x):
69     # Side Constraint
70     # The member thickness is greater than the weld thickness
71     x1, x2, x3, x4 = x
72
```

```
    return -(x1 - x4)

def constraint4(x):
    # Side Constraint
    # The member thickness is greater than the weld thickness
    x1, x2, x3, x4 = x

    return -(C1 * x1**2 + C2 * x3*x4 * (L+ x2) - 5)

def constraint5(x):
    # Side Constraint
    # The weld thickness must be larger than a defined minimum
    x1, x2, x3, x4 = x

    return -(0.125 - x1)

def constraint6(x):
    # The deflection cannot exceed the maximum deflection
    x1, x2, x3, x4 = x

    # Bar Deflection. To calculate the deflection, assume the bar to be a cantilever of length L
    delta = 4 * P * L**3 /(E * x4 * x3**3)
    return -(delta - delta_max)

def constraint7(x):
    # The buckling load is greater than the applied load
    x1, x2, x3, x4 = x

    # For narrow rectangular bars, the bar buckling load is approximated by
    Pc = 4.013 * E / L**2 * np.sqrt(x3**2 * x4**6 /36)* (1 - x3/2/L * np.sqrt(E/4/G))
    return -(P - Pc)

### Problem's Gradients
# Define symbols
x1, x2, x3, x4 = symbols('x1 x2 x3 x4')

# objective_derivative
V_weld = x1**2 * x2 # Volume of weld material (inch3)
V_bar = x3 * x4 * (L + x2) # Volume of bar (inch3)
f = (C1+C3) * V_weld + C2 * V_bar #Total material cost to be minimized.
df_dx1 = diff(f, x1)
df_dx2 = diff(f, x2)
df_dx3 = diff(f, x3)
df_dx4 = diff(f, x4)
```

```
df= [df_dx1, df_dx2, df_dx3, df_dx4]

def objective_derivative(x):
    xx1, xx2, xx3, xx4 = x
    fprime= np.zeros(n_variables)
    # fprime= np.array([2*(C1+C3) * xx1 * xx2,
    #                   (C1+C3) * xx1**2 + C2 * xx3  * xx4,
    #                   C2 * xx4 * (L + xx2),
    #                   C2 * xx3 * (L + xx2)])
    for i in range(n_variables):
        fprime[i] = df[i].subs({x1:xx1, x2: xx2, x3:xx3, x4:xx4})

    return fprime

# Constraint1_derivative
tau_d = P /(np.sqrt(2)*x1*x2)
# Moment of P about center of gravity of weld setup
M = P * (L + x2/2)
R = ((x2**2)/4 + ((x1+x3)/2)**2)**0.5
# Polar moment of inertia of weld
J = 2 * (x1*x2 * np.sqrt(2)* ((x2**2)/12 + ((x1+x3)/2)**2))
# Secondary torsional stress.
tau_dd= M*R/J
# Weld stress
tau = (tau_d**2 + 2* tau_d * tau_dd * x2/2/R + tau_dd**2)**0.5
g1= tau - tau_max
dg1_dx1 = diff(g1, x1)
dg1_dx2 = diff(g1, x2)
dg1_dx3 = diff(g1, x3)
dg1_dx4 = diff(g1, x4)
dg1= [dg1_dx1, dg1_dx2, dg1_dx3, dg1_dx4]

def constraint1_derivative(x):
    xx1, xx2, xx3, xx4 = x
    gprime= np.zeros(n_variables)
    for i in range(n_variables):
        gprime[i] = dg1[i].subs({x1:xx1, x2: xx2, x3:xx3, x4:xx4})

    return -gprime

# Constraint2_derivative
sigma = 6 * P * L /(x4 * x3**2)
g2= sigma - sigma_max
dg2_dx1 = diff(g2, x1)
dg2_dx2 = diff(g2, x2)
dg2_dx3 = diff(g2, x3)
dg2_dx4 = diff(g2, x4)
```

```
dg2= [dg2_dx1, dg2_dx2, dg2_dx3, dg2_dx4]

def constraint2_derivative(x):
    xx1, xx2, xx3, xx4 = x
    gprime= np.zeros(n_variables)
    for i in range(n_variables):
        gprime[i] = dg2[i].subs({x1:xx1, x2: xx2, x3:xx3, x4:xx4})

    return -gprime

# Constraint3_derivative
g3= x1 - x4
dg3_dx1 = diff(g3, x1)
dg3_dx2 = diff(g3, x2)
dg3_dx3 = diff(g3, x3)
dg3_dx4 = diff(g3, x4)
dg3= [dg3_dx1, dg3_dx2, dg3_dx3, dg3_dx4]

def constraint3_derivative(x):
    xx1, xx2, xx3, xx4 = x
    gprime= np.zeros(n_variables)
    for i in range(n_variables):
        gprime[i] = dg3[i].subs({x1:xx1, x2: xx2, x3:xx3, x4:xx4})

    return -gprime

# Constraint4_derivative
g4= C1 * x1**2 + C2 * x3*x4 * (L+ x2) - 5
dg4_dx1 = diff(g4, x1)
dg4_dx2 = diff(g4, x2)
dg4_dx3 = diff(g4, x3)
dg4_dx4 = diff(g4, x4)
dg4= [dg4_dx1, dg4_dx2, dg4_dx3, dg4_dx4]

def constraint4_derivative(x):
    xx1, xx2, xx3, xx4 = x
    gprime= np.zeros(n_variables)
    for i in range(n_variables):
        gprime[i] = dg4[i].subs({x1:xx1, x2: xx2, x3:xx3, x4:xx4})

    return -gprime

# Constraint5_derivative
g5= 0.125 - x1
dg5_dx1 = diff(g5, x1)
dg5_dx2 = diff(g5, x2)
dg5_dx3 = diff(g5, x3)
```

```
dg5_dx4 = diff(g5, x4)
dg5= [dg5_dx1, dg5_dx2, dg5_dx3, dg5_dx4]

def constraint5_derivative(x):
    xx1, xx2, xx3, xx4 = x
    gprime= np.zeros(n_variables)
    for i in range(n_variables):
        gprime[i] = dg5[i].subs({x1:xx1, x2: xx2, x3:xx3, x4:xx4})

    return -gprime

# Constraint6_derivative
delta = 4 * P * L**3 /(E * x4 * x3**3)
g6= delta - delta_max
dg6_dx1 = diff(g6, x1)
dg6_dx2 = diff(g6, x2)
dg6_dx3 = diff(g6, x3)
dg6_dx4 = diff(g6, x4)
dg6= [dg6_dx1, dg6_dx2, dg6_dx3, dg6_dx4]

def constraint6_derivative(x):
    xx1, xx2, xx3, xx4 = x
    gprime= np.zeros(n_variables)
    for i in range(n_variables):
        gprime[i] = dg6[i].subs({x1:xx1, x2: xx2, x3:xx3, x4:xx4})

    return -gprime

# Constraint7_derivative
Pc = 4.013 * E / L**2 * (x3**2 * x4**6 /36)**0.5 * (1 - x3/2/L * (E/4/G)
    **0.5)
g7= P - Pc
dg7_dx1 = diff(g7, x1)
dg7_dx2 = diff(g7, x2)
dg7_dx3 = diff(g7, x3)
dg7_dx4 = diff(g7, x4)
dg7= [dg7_dx1, dg7_dx2, dg7_dx3, dg7_dx4]

def constraint7_derivative(x):
    xx1, xx2, xx3, xx4 = x
    gprime= np.zeros(n_variables)
    for i in range(n_variables):
        gprime[i] = dg7[i].subs({x1:xx1, x2: xx2, x3:xx3, x4:xx4})

    return -gprime

```

```
# Checking the status of each constraint according to its sign by standard form.
def cons_check(x,cons):
    constraint1,constraint2,constraint3, constraint4, constraint5, constraint6, constraint7 =cons
    print('Constraint Evaluation:')
    g1 = constraint1(x)
    print('g1 =',-g1)
    if g1<0:
        print('Constraint1 is violated')
    elif g1>0:
        print('Constraint1 is inactive')
    else:
        print('Constraint1 is active')

    g2 = constraint2(x)
    print('g2 =',-g2)
    if g2<0:
        print('Constraint2 is violated')
    elif g2>0:
        print('Constraint2 is inactive')
    else:
        print('Constraint2 is active')

    g3 = constraint3(x)
    print('g3 =',-g3)
    if g3<0:
        print('Constraint3 is violated')
    elif g3>0:
        print('Constraint3 is inactive')
    else:
        print('Constraint3 is active')

    g4 = constraint4(x)
    print('g4 =',-g4)
    if g4<0:
        print('Constraint4 is violated')
    elif g4>0:
        print('Constraint4 is inactive')
    else:
        print('Constraint4 is active')

    g5 = constraint5(x)
    print('g5 =',-g5)
    if g5<0:
        print('Constraint5 is violated')
    elif g5>0:
```

```
        print('Constraint5 is inactive')
    else:
        print('Constraint5 is active')

    g6 = constraint6(x)
    print('g6 =',-g6)
    if g6<0:
        print('Constraint6 is violated')
    elif g6>0:
        print('Constraint6 is inactive')
    else:
        print('Constraint6 is active')

    g7 = constraint7(x)
    print('g7 =',-g7)
    if g7<0:
        print('Constraint7 is violated')
    elif g7>0:
        print('Constraint7 is inactive')
    else:
        print('Constraint7 is active')

    print('====================================')


# Callback function to display the optimizer's progress
def callbackf(x):
    global Nfeval
    global Objective_history
    print ('{0:4d}   {1: 3.6f}   {2: 3.6f}   {3: 3.6f}  {4: 3.6f}  {5: 3.6f}'.format(Nfeval, x[0], x[1], x[2], x[3], objective(x)))
    Objective_history.append([Nfeval, objective(x)])
    Nfeval += 1
    # print(f"Current solution: {x} Objective: {objective(x)}")


# Bounds
bounds = [(0.1, 2), (0.1, 10), (0.1, 10), (0.1,2)]

# Constraints dictionary
cons = [{'type': 'ineq', 'fun': constraint1, 'jac': constraint1_derivative},
        {'type': 'ineq', 'fun': constraint2, 'jac': constraint2_derivative},
        {'type': 'ineq', 'fun': constraint3, 'jac': constraint3_derivative},
```

```
        {'type': 'ineq', 'fun': constraint4, 'jac':
    constraint4_derivative},
        {'type': 'ineq', 'fun': constraint5, 'jac':
    constraint5_derivative},
        {'type': 'ineq', 'fun': constraint6, 'jac':
    constraint6_derivative},
        {'type': 'ineq', 'fun': constraint7, 'jac':
    constraint7_derivative}]
cons2 = [constraint1, constraint2, constraint3,constraint4, constraint5,
    constraint6
         ,constraint7]
# Initial guess
x0 = [0.18, 4, 9, 0.22]

# Optimize
print('{0:4s}   {1:9s}   {2:9s}   {3:9s}   {4:9s}  {5:9s}'.format('Iter'
    , ' X1', ' X2', ' X3', 'X4', 'f(X)'))

result = minimize(objective, x0, method='SLSQP',jac=
    objective_derivative, bounds=bounds, constraints=cons, callback=
    callbackf, tol = 1e-6)

print('x*=',result.x)
print('f(x*)=',result.fun)
print('Number of Iterations:',result.nit)
print('Number of Function Evaluations:',result.nfev)

x_o= result.x
cons_check(x_o,cons2)


# Part b
x01 = [[0.3, 5, 8, 0.3],
       [0.8, 1, 7, 1.5],
       [0.2, 3, 9, 0.5]]

fig,ax=plt.subplots(1,1)
for i in range(3):
    Nfeval = 1
    Objective_history=[]
    print('Results of Set', i+1)
    print('{0:4s}   {1:9s}   {2:9s}   {3:9s}   {4:9s}  {5:9s}'.format('
    Iter', ' X1', ' X2', ' X3', 'X4', 'f(X)'))
    result = minimize(objective, x01[i], method='SLSQP',jac=
    objective_derivative, bounds=bounds, constraints=cons, callback=
    callbackf)
    print('x*=',result.x)
```

```
    print('f(x*)=',result.fun)
    print('Number of Iterations:',result.nit)
    print('Number of Function Evaluations:',result.nfev)
    Iter = [item[0] for item in Objective_history]
    f_values =  [item[1] for item in Objective_history]
    plt.plot(Iter , f_values, label=f"run {i+1}")

    x_o= result.x
    cons_check(x_o,cons2)
ax.set_xlabel('Iterations')
ax.set_ylabel('Cost')
plt.legend()
# Save the plot as SVG (high quality, vector format)
plt.savefig('Convergencegraph.png', dpi=300)
```

```
Optimization code output using scipy.optimize.minimize
==================================
Iter    X1          X2          X3          X4          f(X)
   1    0.187440    3.924022    8.996485    0.206815    1.756748
   2    0.190824    3.820547    9.036370    0.205733    1.747563
   3    0.199275    3.603630    9.036624    0.205730    1.732579
   4    0.205730    3.465165    9.036624    0.205730    1.724127
   5    0.205730    3.470481    9.036624    0.205730    1.724851
   6    0.205730    3.470483    9.036624    0.205730    1.724852
   7    0.205730    3.470489    9.036624    0.205730    1.724852
   8    0.205730    3.470489    9.036624    0.205730    1.724852
x*= [0.20572964 3.47048867 9.03662391 0.20572964]
f(x*)= 1.724852308596435
Number of Iterations: 9
Number of Function Evaluations: 11
Constraint Evaluation:
g1 = 1.9339495338499546e-08
Constraint1 is violated
g2 = 2.0736479200422764e-09
Constraint2 is violated
g3 = 7.13318293321663e-15
Constraint3 is violated
g4 = -3.4329837853628886
Constraint4 is inactive
g5 = -0.08072963978607528
Constraint5 is inactive
g6 = -0.2355403225847533
Constraint6 is inactive
g7 = 1.0168150765821338e-09
Constraint7 is violated
```

4.9. **Results for 3 different initial designs.** The optimization process was initiated from three distinct starting points to assess the influence of initial conditions on the convergence and robustness of the solution. Table 1 encapsulates the outcomes of these runs, including the initial and optimal designs, the optimal values of the objective function, the number of iterations, and the number of function evaluations.

The assessment of the constraints for each run is documented in Tables 2, 3, and 4. These tables detail the values of the constraint functions and their respective statuses—active, inactive, or violated—for each initial design configuration.

As evident from the reported optimization outputs, all three runs, notwithstanding their differing initial designs, have converged to the same optimal design with no constraints violated. For all runs, constraints 1, 2, 3, and 7 were identified as active, while the remaining constraints were classified as inactive. It should be noted that, due to the inherent numerical errors associated with the optimization algorithm, values on the order of $1e$-6 are regarded as approximately equivalent to zero. This tolerance level is taken into consideration when evaluating the activeness of constraints and determining the convergence of the solution.

The convergence of all three trials is substantiated by the successful termination message provided by the optimization algorithm. Further corroboration is derived from the convergence plots shown in Figure 2, where all three distinct runs exhibit convergence to equivalent values and demonstrate a cessation of progress beyond a certain point, reinforcing the reliability and stability of the obtained solution.

Initial Design	Optimal Design	Optimal Value	Iterations	Function Evaluations
$(0.3, 5, 8, 0.3)$	$(0.2057, 3.4704, 9.0366, 0.2057)$	1.7248	8	9
$(0.8, 1, 7, 1.5)$	$(0.2057, 3.4704, 9.0366, 0.2057)$	1.7248	14	15
$(0.2, 3, 9, 0.5)$	$(0.2057, 3.4704, 9.0366, 0.2057)$	1.7248	10	10

TABLE 1. Optimization results for the welded beam design problem.

Constraint	Value	Status
g_1	-1.27e-11 ≈ 0	Active
g_2	-1.16e-10 ≈ 0	Active
g_3	2.49e-16 ≈ 0	Active
g_4	-3.4329	Inactive
g_5	-0.0807	Inactive
g_6	-0.2355	Inactive
g_7	7.27e-12 ≈ 0	Active

TABLE 2. Constraint Evaluations for the welded beam design problem, run 1.

Constraint	Value	Status
g_1	-5.12e-10 ≈ 0	Active
g_2	-5.55e-9 ≈ 0	Active
g_3	-2.91e-14 ≈ 0	Active
g_4	-3.4329	Inactive
g_5	-0.0807	Inactive
g_6	-0.2355	Inactive
g_7	-3.09e-9 ≈ 0	Active

TABLE 3. Constraint Evaluations for the welded beam design problem, run 2.

Constraint	Value	Status
g_1	4.76e-8 ≈ 0	Active
g_2	7.85e-8 ≈ 0	Active
g_3	-2.31e-14 ≈ 0	Active
g_4	-3.4329	Inactive
g_5	-0.0807	Inactive
g_6	-0.2355	Inactive
g_7	-2.61e-8 ≈ 0	Active

TABLE 4. Constraint Evaluations for the welded beam design problem, run 3.

4.10. **Analysis.**

(a) **Convergence Plot**:
The convergence plot of the objective function (vertical axis) versus iteration (horizontal axis) for all three runs is shown in Figure 2.

(b) **Robustness of `scipy.minimize`**:
The robustness of an optimization algorithm is fundamentally characterized by its ability to consistently arrive at the same solution, particularly in the context of non-linear problems with a complex landscape of local minima and maxima. A robust optimization algorithm should ideally converge to the global optimum regardless of initial conditions or slight changes in the problem setup.

In the case of the welded beam design optimization using the SLSQP algorithm from `scipy.minimize`, we observed all runs with three different initial designs all converging to the same optimal design without violating any constraints, there is an indication of robustness in the algorithm. Here's why:

– **Consistency Across Initial Designs:** A robust optimization algorithm should yield consistent results despite variations in initial conditions. The fact that `scipy.minimize` found the same solution from three distinct initial designs suggests that it is not overly sensitive to the starting point and is therefore robust in this regard.

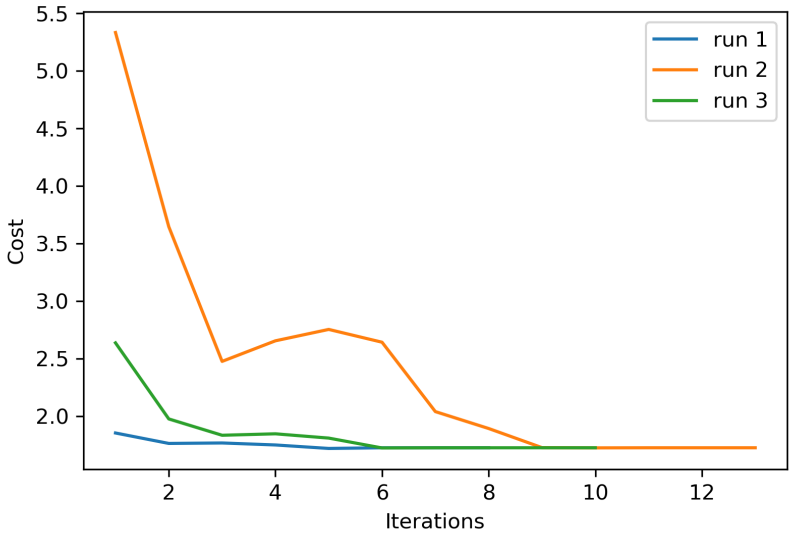

FIGURE 2. Convergence plot of the objective function vs. iteration for all three runs.

- **Constraint Handling:** The algorithm managed to find solutions that satisfy the problem's constraints, indicating that it can effectively navigate the feasible region defined by the constraints.
- **Successful Termination:** If the algorithm reported a successful termination status for all optimization runs, this suggests that it has met its convergence criteria consistently, reinforcing the notion of robustness.
- **Convergence Plots:** The convergence plots further support the robustness of the algorithm by visually demonstrating that the optimization process converges to the same values across different runs, indicating stable behavior of the algorithm.

4.11. **Analytical Solution Verification.** In this section, the optimality condition (KKT condition) of the converged solution is investigated analytically.

$$x^* = \begin{bmatrix} 0.20572964 \\ 3.47048867 \\ 9.03662391 \\ 0.20572964 \end{bmatrix}$$

$$f(x^*) = 1.7248523085953442$$

As constraints 1, 2, 3, and 7 are active and constraints 4, 5, and 6 are inactive we have:

$$\mu_4 = \mu_5 = \mu_6 = 0$$

$$s_1 = s_2 = s_3 = s_7 = 0$$

From the Primal Feasibility condition we have:

$$s_j = \pm\sqrt{-g_j(\mathbf{x}^*)}, \quad \forall j \in \{1, \ldots, n_g\}$$

$$s_4 = \pm 1.8528312889640355$$

$$s_5 = \pm 0.28412961793248254$$

$$s_6 = \pm 0.4853249659606267$$

From the Stationary condition we have:

$$\nabla f(\mathbf{x}^*) + J_g(\mathbf{x}^*)^T \mu = 0$$

As we know, constraints 4, 5, and 6 are inactive and can be removed from the J_g to reduce the dimensionality of the problem and ease up the calculations. The Stationary condition equations will become like below:

$$\begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \\ \frac{\partial f}{\partial x_4} \end{bmatrix} + \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \frac{\partial g_1}{\partial x_3} & \frac{\partial g_1}{\partial x_4} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \frac{\partial g_2}{\partial x_3} & \frac{\partial g_2}{\partial x_4} \\ \frac{\partial g_3}{\partial x_1} & \frac{\partial g_3}{\partial x_2} & \frac{\partial g_3}{\partial x_3} & \frac{\partial g_3}{\partial x_4} \\ \frac{\partial g_7}{\partial x_7} & \frac{\partial g_7}{\partial x_2} & \frac{\partial g_7}{\partial x_3} & \frac{\partial g_7}{\partial x_4} \end{bmatrix}^T \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_7 \end{bmatrix} = 0$$

$$\begin{bmatrix} 2\,x_1\,x_2\,(C_1 + C_3) \\ (C_1 + C_3)\,{x_1}^2 + C_2\,x_3\,x_4 \\ C_2\,x_4\,(L + x_2) \\ C_2\,x_3\,(L + x_2) \end{bmatrix} +$$

$$\begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \frac{\partial g_1}{\partial x_3} & 0 \\ 0 & 0 & -\frac{12\,L\,P}{{x_3}^3\,x_4} & -\frac{6\,L\,P}{{x_3}^2\,{x_4}^2} \\ 1 & 0 & 0 & -1 \\ 0 & 0 & -\frac{4.013\,\mathrm{E}\left({x_3}^2\,{x_4}^6\sqrt{\frac{\mathrm{E}}{4\,G}} - L\,x_3\,{x_4}^6\right)}{6\,L^3\sqrt{{x_3}^2\,{x_4}^6}} & \frac{4.013\,\mathrm{E}\,{x_3}^2\,{x_4}^5\left(2\,L - x_3\sqrt{\frac{\mathrm{E}}{4\,G}}\right)}{4\,L^3\sqrt{{x_3}^2\,{x_4}^6}} \end{bmatrix}^T \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_7 \end{bmatrix} = 0$$

$$\begin{bmatrix} 1.57748696 \\ 0.13619787 \\ 0.17291683 \\ 7.59532947 \end{bmatrix} + \begin{bmatrix} -67309.99076198 & -3127.41889517 & -1203.81367362 & 0 \\ 0 & 0 & -6639.64779275 & -145822.44945947 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & -436.52738288 & -87493.46967583 \end{bmatrix}^T \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_7 \end{bmatrix} = 0$$

By solving the system of linear equations, we have:

$$\begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_7 \end{bmatrix} = \begin{bmatrix} 4.3550\text{e-}5 \\ 1.5113\text{e-}5 \\ 1.3538 \\ 4.6148\text{e-}5 \end{bmatrix}$$

By having all μ as positive, the Dual Feasibility condition is satisfied and the given solution is verified analytically using the KKT condition.

5. Python Implementation using `scipy.optimize.differential_evolution`

The implementation and subsequent evaluation of the Kernel Principal Component Analysis (KPCA) as compared to other algorithms such as linear PCA, the Parzen window algorithm, and one-class SVM, have demonstrated that KPCA offers superior performance in terms of accuracy. This enhanced accuracy is particularly evident in its ability to handle complex, non-linear data transformations more effectively, thus making it highly suitable for applications in novelty detection where distinguishing subtle anomalies is crucial.

5.1. **Definition of Nonlinear Constraints.** In the implemented optimization routine, 'NonlinearConstraint' objects compatible with `scipy.optimize.differential_evolution` are meticulously defined. These constraints adhere to the Python optimization standard which stipulates that constraints should yield non-negative values for feasible solutions. Therefore, the lower bounds of the constraints are set to zero or a suitable positive value to reflect this requirement, while the upper bounds are typically set to 'np.inf', indicating no upper limit. Additionally, the Jacobian of the constraints—a matrix of first-order derivatives of the constraint functions—is provided to facilitate the optimization process.

5.2. **Optimization Execution with Multiple Seeds.** To validate the robustness and consistency of the optimization algorithm, three different seed values are selected for the random number generator used within `differential_evolution`. This ensures that the starting population of solutions varies with each run. A `for` loop is constructed to iterate over these seed values, executing the optimization process for each one. By doing so, we can analyze the impact of the stochastic nature of the algorithm on the optimization outcome and confirm the solution's stability across multiple runs.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 28 22:01:54 2024

@author: vahed
"""

# Part c Differential Evolution
# No lower limit on constr_fun
lb =  0
```

```
11
12 # Upper limit on constr_fun
13 ub= np.inf
14
15 # Nonlinear Constraints
16 nlc1 = NonlinearConstraint(constraint1, lb, ub, jac=
       constraint1_derivative)
17 nlc2 = NonlinearConstraint(constraint2, lb, ub, jac=
       constraint2_derivative)
18 nlc3 = NonlinearConstraint(constraint3, lb, ub, jac=
       constraint3_derivative)
19 nlc4 = NonlinearConstraint(constraint4, lb, ub, jac=
       constraint4_derivative)
20 nlc5 = NonlinearConstraint(constraint5, lb, ub, jac=
       constraint5_derivative)
21 nlc6 = NonlinearConstraint(constraint6, lb, ub, jac=
       constraint6_derivative)
22 nlc7 = NonlinearConstraint(constraint7, lb, ub, jac=
       constraint7_derivative)
23 Cons_diff = [nlc1, nlc2, nlc3, nlc4, nlc5, nlc6, nlc7]
24 print('Results of differential_evolution')
25 sol = differential_evolution(objective, bounds=bounds, constraints=
       Cons_diff, tol= 1e-10, maxiter=8000)
26 print('x*=',sol.x)
27 print('f(x*)=',sol.fun)
28 print('Number of Function Evaluations:',sol.nfev)
29 x_o= sol.x
30 cons_check(x_o,cons2)
31
32 # Part d
33 seeds= [157, 4921, 753]
34
35 for i in range(3):
36     print('Results of differential_evolution set', i+1)
37     sol = differential_evolution(objective, bounds=bounds, seed = seeds[
       i], constraints=[nlc1, nlc2, nlc3, nlc4, nlc5, nlc6, nlc7], tol= 1e
       -10, maxiter=8000)
38     print('x*=',sol.x)
39     print('f(x*)=',sol.fun)
40     print('Number of Generations:',sol.nit)
41     print('Number of Function Evaluations:',sol.nfev)
42     x_o= sol.x
43     cons_check(x_o,cons2)
```

```
Optimization code output using scipy.optimize.differential_evolution
==================================
Results of differential_evolution
```

```
x*= [0.20572964 3.47048867 9.03662391 0.20572964]
f(x*)= 1.7248523086235628
Number of Function Evaluations: 11421
Constraint Evaluation:
g1 = -1.0274379746988416e-07
Constraint1 is inactive
g2 = -1.0268813639413565e-06
Constraint2 is inactive
g3 = -1.3623546735175296e-12
Constraint3 is inactive
g4 = -3.432983785334403
Constraint4 is inactive
g5 = -0.08072963978506309
Constraint5 is inactive
g6 = -0.23554032258548466
Constraint6 is inactive
g7 = -9.446739568375051e-08
Constraint7 is inactive
```

5.3. **Results for 3 different seeds.** The optimization process was initiated from three distinct random seeds to assess the influence of randomness on the convergence and robustness of the solution. Table 5 encapsulates the outcomes of these runs, including the initial and optimal designs, the optimal values of the objective function, the number of generations, and the number of function evaluations.

The assessment of the constraints for each run is documented in Tables 6, 7, and 8. These tables detail the values of the constraint functions and their respective statuses—active, inactive, or violated—for each initial design configuration.

As evident from the reported optimization outputs, all three runs, with their differing seeds, have converged to the same optimal design with no constraints violated. For all runs, constraints 1, 2, 3, and 7 were identified as active, while the remaining constraints were classified as inactive. It should be noted that, due to the inherent numerical errors associated with the optimization algorithm, values on the order of $1e$-6 are regarded as approximately equivalent to zero. This tolerance level is taken into consideration when evaluating the activeness of constraints and determining the convergence of the solution.

The convergence of all three trials is substantiated by the successful termination message provided by the optimization algorithm. All three distinct runs exhibit convergence to equivalent values and demonstrate a cessation of progress beyond a certain point, reinforcing the reliability and stability of the obtained solution.

5.4. **Analysis.**

Seed	Optimal Design	Optimal Value	Generations	Function Evaluations
157	$(0.2057, 3.4704, 9.0366, 0.2057)$	1.7248	408	11010
4921	$(0.2057, 3.4704, 9.0366, 0.2057)$	1.7248	414	11217
753	$(0.2057, 3.4704, 9.0366, 0.2057)$	1.7248	446	11589

TABLE 5. Optimization results for the welded beam design problem.

Constraint	Value	Status
g_1	-8.47e-8 ≈ 0	Active
g_2	-2.57e-6 ≈ 0	Active
g_3	-3.18e-12 ≈ 0	Active
g_4	-3.4329	Inactive
g_5	-0.0807	Inactive
g_6	-0.2355	Inactive
g_7	-5.63e-7 ≈ 0	Active

TABLE 6. Constraint Evaluations for the welded beam design problem, seed 1.

Constraint	Value	Status
g_1	-1.17e-7 ≈ 0	Active
g_2	-2.31e-6 ≈ 0	Active
g_3	-1.81e-12 ≈ 0	Active
g_4	-3.4329	Inactive
g_5	-0.0807	Inactive
g_6	-0.2355	Inactive
g_7	-1.92e-7 ≈ 0	Active

TABLE 7. Constraint Evaluations for the welded beam design problem, seed 2.

Constraint	Value	Status
g_1	-2.95e-8 ≈ 0	Active
g_2	-1.001e-7 ≈ 0	Active
g_3	-6.71e-12 ≈ 0	Active
g_4	-3.4329	Inactive
g_5	-0.0807	Inactive
g_6	-0.2355	Inactive
g_7	-3.80e-9 ≈ 0	Active

TABLE 8. Constraint Evaluations for the welded beam design problem, seed 3.

(a) **Comparison of Evaluation Numbers**:
The `differential_evolution` function in SciPy is a global optimization algorithm that does not require the gradient of the objective function. Instead, it

uses a stochastic sampling approach based on evolutionary algorithms that iterate over a population of solution candidates. This process is inherently different from algorithms used in `scipy.minimize`, such as Sequential Least Squares Quadratic Programming (SLSQP), which utilize the gradient information to converge to a solution. Consequently, `differential_evolution` may take more iterations and function evaluations to reach an optimum because it relies on evolutionary strategies rather than gradient descent mechanisms to minimize the objective function. This leads to a larger number of function evaluations, especially in cases where the landscape of the objective function is complex with multiple local minima.

(b) **Similarity of Optimal Designs**:
Upon examining the optimal designs obtained from different runs of the optimization algorithm, it is observed designs are very close to each other. This consistency suggests that the optimization algorithm is converging to a solution that is either the global optimum or a local optimum of high quality. The similarity of the designs also implies that the solution space of the problem is well-structured around the optimum, allowing the algorithm to repeatedly identify the same or very similar solutions, even when starting from different random seeds.

References

[1] Kenneth M. Ragsdell and Don T. Phillips. "Optimal Design of a Class of Welded Structures Using Geometric Programming". In: *Journal of Engineering for Industry* 98 (1976), pp. 1021–1025. URL: https://api.semanticscholar.org/CorpusID:109072095.

[2] Stephen P Timoshenko and James M Gere. *Theory of elastic stability*. Courier Corporation, 2009.