Delft University of Technology
Master of Aerospace Engineering
Control & Simulation

An Independent, Generic, User-Commanded, Sequential Quadratic Programming Module for Solving the Aircraft Trim Problem

Michael Visser
michaelvisser1991@gmail.com

Abstract:
The Delft University of Technology ('TU Delft') developed a real-time distributed system for scientific and educational purposes. Because of the high level of expertise required to learn from- and work in a real-time environment, TU Delft created a middleware layer, DUECA (Delft University Environment for Communication and Activation), and a simulation-specific addition framework: DUSIME (Delft University SIMulation Environment). A common practice is embedding the numerical optimization tool in an aircraft model and retrieving the starting conditions, referred to as the initial trim set. Setting up such an embedded tool for every aircraft model is very labor-intensive. For over 20 years, these issues have limited the overall user experience in DUECA. Hence, the research created an independent, generic, User-commanded, Sequential Quadratic Program (SQP) module capable of solving the aircraft trim problem in DUECA. The trim module works by a user selecting a desired steady-state aircraft trajectory through a Graphic User Interface (GUI) and then commands the trim module to search for the set of initial trim conditions. The advised flight trajectory found so far by minimizing the DASMAT trim problem are the straight-and-level descending, pull-up, and turning flight. The calculated initial trim sets allow the starting up of an aircraft simulation in a steady-state, stable enough such that a pilot can take over manual control.

July 19, 2022

Table of Contents

1 Introduction

1.1 Background

Real-time distributed systems are used worldwide and find many applications. Real-time systems not only depend on correct computation results, but also on the time needed for these results to be computed. The distribution of such systems is necessary for fault tolerance, data processing of actuators, using sensors at specific locations, and performance issues. One application where these system properties are convenient is in flight simulation systems.

The Delft University of Technology ('TU Delft') adopted real-time distributed systems for scientific and educational purposes. Because of the high level of expertise required to learn from- and work in a real-time environment, TU Delft created a middleware layer, DUECA (Delft University Environment for Communication and Activation), and a simulation-specific addition framework: DUSIME (Delft University SIMulation Environment). DUECA is established to support and develop real-time distributed processes by hiding the complex network and synchronization of computation processes. DUECA offers users services DUECA base, DUECA configure, DUECA creation, DUECA control. The DUECA base service level is accessible for modules written in C++ and provides the following. Ensuring unique naming, assigning identity, and registry is one base level service. Providing a communication network and activation are other base level services. Communication between (possibly) distributed devices flows through 'CHANNELS'. A channel is defined by its possibly distributed property and has predefined types of data. The channel communication types are either an event, a type of data that refreshes once triggered, or stream data, that is refreshed regularly. DUECA provides activation execution that does not require a global schedule where modules in a fixed order. Instead, modules describe activation conditions that determine when to schedule a module. To provide User-friendly control of dynamic systems in DUECA, the developers created the top layer DUSIME [39].

DUSIME provides capabilities that are required for real-time simulating. DUSIME acts as a simulation state machine that controls the DUSIME classes HardwareModule and SimulationModule. The class HardwareModule controls the simulation states Down, Neutral, Calibrate and Active, essential for control hardware systems such as the motion or control loading system of the SIMONA research simulator. The class SimulationModule controls the simulation states HoldCurrent, Advance, and CalculateInco. The simulation state HoldCurrent assures maintaining the current states and Advance allows time steps by the dynamic model. During both phases, a set of starting points for which a dynamic model starts in a steady-state condition referred to as initial trim conditions are required. The simulation state CalculateInco is intended to find initial trim conditions by solving the aircraft trim problem for some user-defined conditions. This initial trim condition is the simulation model's starting point, enabling the computation of all other model-dependent outputs. Currently, the SimluationModule does support facilities for calculating initial conditions, but isn't linked to the real-time environment and is only available for a limited number of dynamic models. This means most initial trim conditions of the dynamic models are labor-intensive predetermined sets of initial trim conditions which limit the DUECA/DUSIME starting capability. By extending the DUECA/DUSIME software and creating an independent generic initial trim module based on an extensive literature study, the environment is extended in such a way that a user can select the initial conditions that are desired for flight simulations through an interface [39].

In literature, the aircraft trim problem is solved in a number of ways. Baghdadi et al. [5] solves the problem using a bifurcation method. Millidere et al. [35] solves the problem using a Newton-step method. Marco, Duke, and Berndt [32] solve the problem more analytical. Baghdadi et al. [5] points out a common way to solve the aircraft trim problem is using Sequential Quadratic Programming (SQP) where the other two solve the problem using a Newton-step using for Millidere et al. [35] a line-search framework. Marco, Duke, and Berndt [32] used the DirectSearch C++ library to obtain all results given their research. The common, logical, and safe option is using one of these guidelines to fill the research gap which is implementing a trim module for all dynamic models in DUECA. The paper of Millidere et al. [35] shows that it's possible to trim an aircraft using basic methods.

The novelty of the current study is creating a facility in DUECA that not only trims regular aircraft, but also other models. An additional novelty is the implementation of a generic solver in a real-time distributed environment. This further increases the demand for a facility in general. Within the constraints of a master thesis project, the research gap will be (partially) filled. The focus of this research is on creating an independent, generic, User-commanded, numerical optimization module capable of solving the aircraft trim problem in DUECA. If this is satisfied, the model can be used in follow-up research to also solve other dynamic models in DUECA. Making the trim module is not only an urgent need for the DUECA, but also of scientific importance.

1.2 Problem Definition

The generic trim routines are not available in the framework of DUECA/DUSIME. This missing feature causes users to derive and submit their own initial trim conditions for their specific dynamic models. This is a lacking feature in the framework starting capability, therefore overall user experience is compromised. Most simulations implemented in the current environment hold a few initial conditions and can be extended to many possible equilibrium points. The aim of the project is to extend the fixed set of equilibrium points for any simulation model in DUECA by making starting conditions selectable. By creating an independent generic scientifically substantiated initial trim module based and establishing a user interface for selecting initial conditions, DUECA's initial performance and user experience will be increased, making the environment more complete. Hence, the problem definition is defined as:

How can an independent generic trim module in DUECA use numerical optimization to find user-defined steady-state aircraft conditions that are selectable through a graphic user interface?

The project aims to increase the initial capabilities of DUECA by allowing the user to select/customize their preferred initial condition. The research goal is to develop an optimization module in DUECA that is capable of finding steady-state initial conditions for aircraft models based on user preferences that are selected through a graphic user interface. Based on the found research gap and objective, the aim is divided into several sub-aims and defined below.

- Section 2 gathers input in the field of aircraft mechanics and dynamics regarding popular aircraft parameters and their representations, common flight scenarios, and aircraft model complexity.
- Section 3 gathers input in the field of numerical optimization regarding best practices for defining and solving such a problem.
- Section 3 develops candidate numerical optimization algorithms that are capable of solving generic aircraft models subjected to their predefined flight scenario and parameters settings.
- Section 4 performs a case study on the performance of the implemented algorithm by solving relevant benchmark functions and comparing results with similar optimization strategies.

The desired research sub-aims are achieved by completing all sub-goals. The sub-goals are reflected in the subsidiary questions.

1. What generic parameter settings define a common steady-state trim condition for the aircraft trim problem?
   1.1. What generic parameter quantification defines general steady-state flight for the aircraft trim problem?
   1.2. What generic parameter settings and formulations concerning common spatial representations define general trim conditions?
2. What numerical optimization methods are effective at solving the aircraft trim problem independently?
   2.1. What are industry best practices for solving the aircraft trim problem in the field of numerical optimization?
   2.2. What practical and relevant optimization algorithms solve the aircraft trim problem?
   2.3. Which of these methods provides a safe structure such that as algorithm complexity increases the performance increases but is capable of operating at any intermediate complexity step?

3. How do different numerical optimization features perform on aircraft trim problem and benchmark problems?
   3.1. What algorithm from two: iteration frameworks, step methods, first derivative approaches, and second derivative approaches solves the aircraft trim problem the most robust, accurate, and fastest?
   3.2. What constraints and objective function is necessary, and do any other features prove determinate to achieve a successful algorithm outcome?

The generic trim module will operate independently. DUECA's base service makes it possible to create modules that communicate through CHANNELS. The goal is to create a numerical algorithm in one Module, and a Graphic User Interface in another module that both connects to an executable simulation model. The user fills in the flight setting using the GUI, then presses a start trim button. After pressing start, the GUI transforms necessary parameters, then sends it off to the minimization algorithm for its first iteration x_k. The trim module dictates the execution of the Blackbox aircraft model. In the Blackbox model, only read/write code is appended, necessary for communication. When the minimization algorithm finds a solution that is lower than a set tolerance, then a solution presents to the user. Figure 1 visualizes the flow of data.

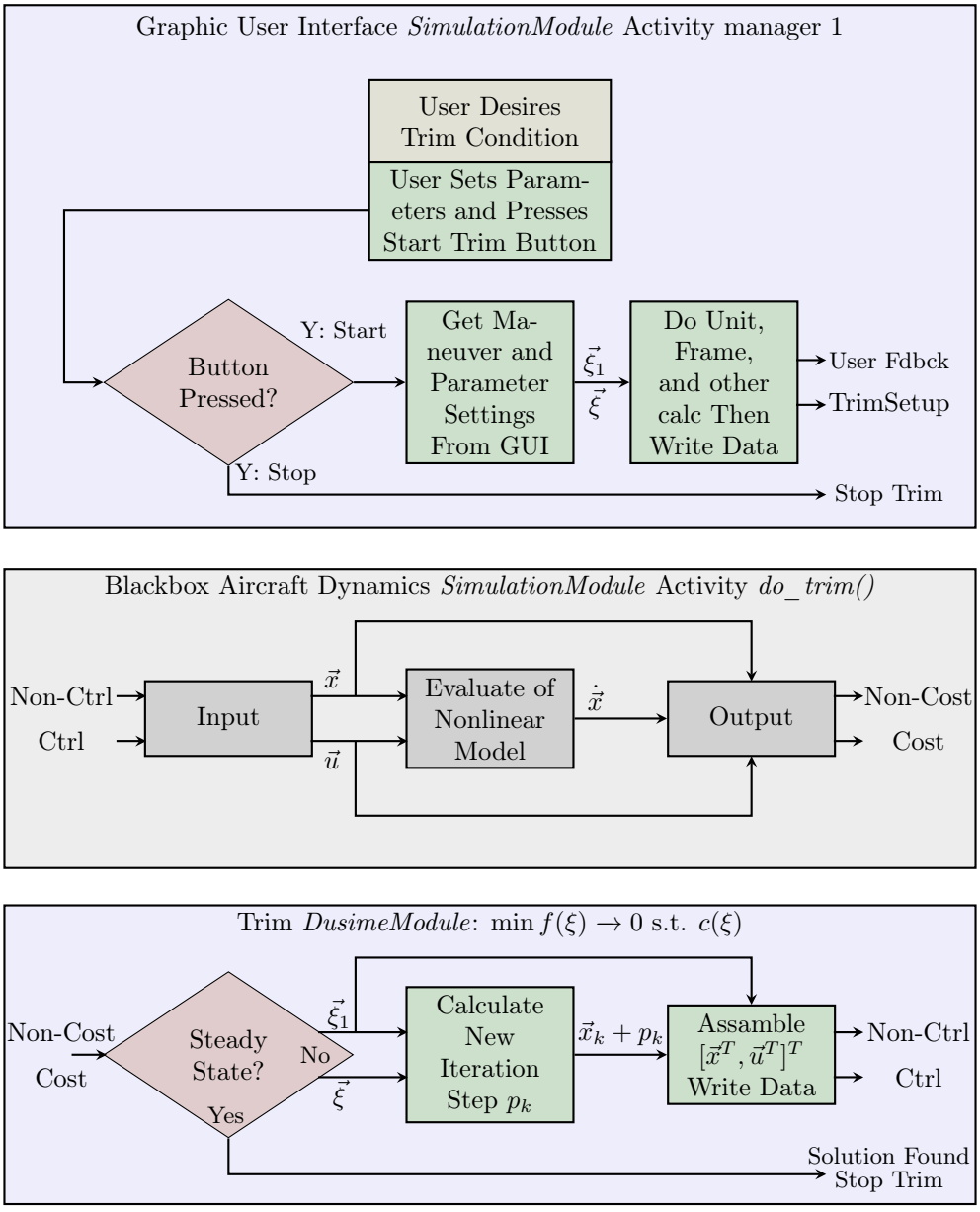


Figure 1: Flowchart for the independent generic trim routine in DUECA

Section 2 describes the generic aircraft trim problem definition. This section then works toward parameter settings suggestions necessary to efficiently find a specific type of trim condition. A specific type of trim condition allows steady-state motion of an aircraft following a specific flight trajectory. This trajectory is also referred to as a flight maneuver, in a steady-state condition. The GUI module uses these suggesting settings to develop a visual design, create the static Parameter Transformation layer settings, and dynamic transformation equations inside the trim algorithm. Section 3 explores proven numerical optimization methods and their performance criteria, then describes an algorithm that is capable of solving the aircraft trim problem. Section 4 finds appropriate benchmark problems, after which the information provided in section 2 and 3 is used to solve selected popular benchmark and aircraft trim problems. Near the end of section 4 a more optimal algorithm form is provided. Discussable steps and results are described in Section 5. The conclusion of the results and progress made so far is given in section 6.

2 Aircraft Trim Problem

This section solves the aircraft trim problem and describes the process step by step. The generic trim problem is formulated and describes what aircraft motion quantities define a valid generic trim problem solution.

2.1 Generic Trim Problem

Before establishing a generic trim module, a generic description for aircraft behavior has to be created. In general, a generic aircraft state system can be expressed by the *implicit system* below [16, 17, 32, 41]

$$g(\dot{x}, x, u) = 0 \tag{2.1}$$

Here, x defines the aircraft *state* vector; The vector g contains n_s scalar nonlinear functions g_i describing 6-DOF aircraft state equations projected in a reference frame; and the column vector u contains n_c control variables [32]. Generally, the system of equations Eq. (2.1) is implicit. This means a set of g_i equations or a single n_s scalar equations contains a component of the vector $\dot{x}$ state variable time derivatives. The state time derivatives that cannot explicitly be written as separate quantities [32]. Many system states for simulating aircraft states are available. There are distinctions made between system states as follows [16, 17, 32, 41]:

$$x = [x_d^T, x_k^T]^T \tag{2.2}$$

Here, the aircraft state column vector x expresses a kinematic column state vector x_k and a dynamic column state vector x_d. Common kinematic and dynamic state parameters are $x_k = [x, y, z, (\phi, \theta, \psi) \vee (q_1, q_2, q_3, q_4), ...]^T$ and $x_d = [(V, \alpha, \beta) \vee (u, v, w), p, q, r, ...]^T$. Here, the kinematic state variables describe the position x, y, z and the angles $(\phi, \theta, \psi) \vee (q_1, q_2, q_3, q_4)$ of the aircraft's center of mass with respect to an inertial frame of reference. The dynamic state variables $(V, \alpha, \beta) \vee (u, v, w)$ describe translational velocity, and (p, q, r) formulates rotational velocity of the aircraft's center of gravities. Both kinematic and dynamic state vectors must include subsystem state variables when present in a specific aircraft. If a specific aircraft includes a propulsion system and/or aerodynamic control surface actuator or another system that expresses its motion by dynamic state variables, Then a user must add the dynamic state variables to the appropriate vector. System of equations Eq. (2.1) describes a control vector u that defines a row-column of control inputs. The control inputs may consist of aerodynamic control surfaces and engine manipulators. The number of control states depends on the type of aircraft. An example of a minimum conventional aircraft configuration's input arrangement gives the following:

$$\vec{u} = [\delta_T, \delta_e, \delta_a, \delta_r, \cdots]^T \tag{2.3}$$

Here, δ_T describes the throttle settings; δ_e the angular deflection of the elevator; δ_a the angular deflection of the ailerons and δ_r the angular deflection of the rudder. Important to note is that most inputs are constrained within a certain interval. A classical concept in the field of nonlinear system theory, in this case, a generic trim module of any airplane model given by Eq. (2.1), is finding an equilibrium point or also referred to as a trim point. For an autonomous process where the system input is not dependent on external control inputs, the time-invariant system trim point state vector $\vec{x_{eq}}$ as a particular set of solutions for $\vec{x}$ which satisfies the equation below [17, 25, 32]

$$g(0, \vec{x}_{eq}, \vec{u}_{eq}) = \vec{0} \qquad \text{with} \quad \dot{\vec{x}}_{eq} \equiv 0 \quad \text{and} \quad \vec{u}_{eq} = (0 \text{ or } \vec{u}_0). \tag{2.4}$$

Here, $\vec{u_{eq}}$ defines the control settings to ensure a steady state. The vector $\dot{\vec{x}}_{eq}$ must be selected appropriately. Selecting $\dot{\vec{x}}$ from Eq. (2.2) and setting $\dot{\vec{x}}_{eq} = \dot{\vec{x}} = \vec{0}$ corresponds to a generalized idea of rest of a system. This limits the trim application to an aircraft in standstill on the ground optimization only. The appropriate form suggests the generalized idea of equilibrium where $\dot{\vec{x}}_d = \vec{x}_{eq} = \vec{0}$. Dynamic systems in equilibrium allow rotational and translational motion, making flight possible. The content of $\vec{x}_{eq}$ and $\vec{u}_{eq}$ is selected such that a numerical optimizer manipulates the variables $\vec{x}_{eq}$ and $\vec{u}_{eq}$ resulting in system equilibrium. Table 1 describes motion requirements and all possible generic trim problem solutions.

Accelerations	$\Rightarrow$	$\dot{u}, \dot{w}, \dot{w}\,(\text{or}\,\dot{V}, \dot{\alpha}, \dot{\beta})$	$\equiv$	$\dot{p}, \dot{q}, \dot{r}\,(\text{or}\,\dot{p_B}, \dot{q_B}, \dot{r_B}) \equiv 0$
Linear velocities	$\Rightarrow$	$u, v, w\,(\text{or}\,V, \alpha, \beta)$	$=$	predefined constant value
Angular velocities	$\Rightarrow$	$p, q, r\,(\text{or}\,p_B, q_B, r_B)$	$=$	predefined constant value
Aircraft controls	$\Rightarrow$	$\delta_T, \delta_e, \delta_a, \delta_r$	$=$	predefined constant value

Table 1: Valid motion and control values that solve the generic trim problem.

The system achieves steady-state flight if the dynamic state derivatives $(\dot{p}, \dot{q}, \dot{r})$ and $(\dot{u}, \dot{v}, \dot{w})$ or $(\dot{V}, \dot{\alpha}, \dot{\beta})$ are zero. This allows predefining the dynamic state variables (p, q, r) and (u, v, w) or (V, α, β) constant or zero. Table 1 shows acceptable aircraft motion quantities that provide a valid solution to the generic trim problem. The next step formulates assumptions and defines a strategy in Section 2.2 such that Section 2.3 can specify steady-state flight maneuvers.

2.2 Assumptions and Strategy

Reducing the complexity of the aircraft trim problem is common practice. Marco, Duke, and Berndt [32] describe that when the unsteady aerodynamic $\dot{\alpha}$ and $\dot{\beta}$ rate dependencies are completely general, the implicit system of equations Eq. (2.1) contains equations of unsteady motion. The trim condition must derive from equilibrium. During this state of equilibrium, the dynamic state derivatives $\dot{\alpha} = \dot{\beta} = 0$ or $\dot{u} = \dot{v} = \dot{w} = 0$. This makes the following explicit state equation sufficient for solving the aircraft trim problem [32]:

$$\dot{x} = f(x, u) \tag{2.5}$$

Here, each separate function in the set $f = [f_1, f_2, \cdots, f_n]^T$ explicitly describes one state derivative per equation. Although all equations are valid for trimming, not all relations can be used. Relations that contain aircraft model-specific parameters are unusable. The aircraft model-specific parameters, such as mass or gravity, may vary per aircraft model, where the generic trim module must work for all aircraft models. Because aircraft-specific state equations are unusable, a numerical routine must model the state equations that contain aircraft-specific parameters. The explicit state equations that rely only on state transformations are usable.

Assumptions are often necessary when searching for trim conditions because the trajectories that describe true equilibrium are very limited. Marco, Duke, and Berndt [32] describes that it's considered satisfactory to assume *flat-Earth equations* for control system design, therefore also for finding trim conditions. This assumption allows the *wing-level horizontal flight* and *constant altitude turning flight* trim conditions. Marco, Duke, and Berndt [32] also describes neglecting changes in atmospheric density. Combining the *flat-Earth* assumption and neglecting changes in atmospheric density allow for *wing-level climb/descending flight* and *climbing/descending turning flight* [32]. The types of trim condition described in the current and Section 2.3 may only result in transitory conditions. Imposing constraining equations on the flight conditions results in trajectory-specific trim conditions for the aircraft trim problem. Generally, implementing constraining equations leads to specific types of acceptable trim conditions in simulation or linearization programs [32].

The trim algorithm Eq. (2.5) describes 12 explicit equations of motion. The 12 functions return a set of kinematic state derivatives $\dot{x}_k$, and a set of dynamic state derivatives $\dot{x}_d$. The sets of state derivatives are a function of kinematic state variables x_k, dynamic state variables x_d and control settings u. The trim algorithm only uses the aircraft dynamic model for execution and doesn't solve the differential equation to provide a propagation step in time. Therefore, the trim algorithm only needs the set of dynamic state derivatives for calculating a state and input combination that provides generic trim. If a solver finds a condition where $\dot{x}_d = 0$ using x_d, $\dot{x}_k$, x_k, and u without any other constraining criteria, the user ends up finding a random steady-state flight condition. This is because finding the condition $\dot{x}_d = 0$ using all states and input has infinitely many solutions. By targeting user-defined state quantities, using state transformations, and imposing state constraints, the trim module is capable of finding the user-specified trim condition. In general, the following procedure ensures trajectory-specific trim.

The system is reduced by first deciding a target trajectory, then assigning values to a set of state variables and control quantities. For a rectilinear trajectory example, the flight specifics create a set of initial values

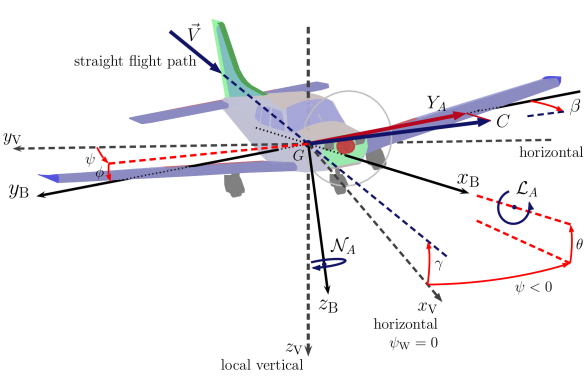
giving $\xi_1 = [V, h, \gamma, \psi_w, \dot{\gamma}_w = 0]^T$. Constraining the flight to achieve the rectilinear trajectory, a set of derived state quantities gives $\xi_2 = [p = 0, q = 0, r = 0]$. The remaining set becomes the set of trim algorithm's manipulating variables, containing state and control variables $\xi = [\phi, \theta, \psi, \delta_T, \delta_e, \delta_a, \delta_r]^T$. The vectors ξ_1 and ξ_2 are different subsets of $[x^T, u^T]^T$, resulting in the remaining set ξ of $[x^T, u^T]^T$. In general, the trim equations of motion considers the following set of state equations:

$$f_n(\xi) = 0, \qquad \text{for } n = 1, \ldots, 6. \tag{2.6}$$

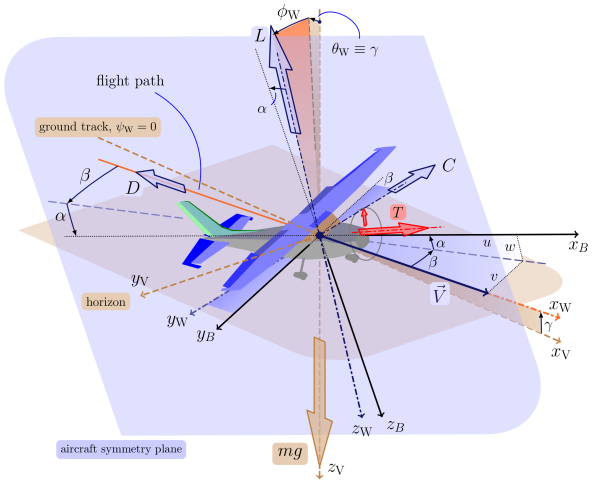
Here, the set of functions $f_n(\xi)$ with respect to some unknown set of variables ξ. Marco, Duke, and Berndt [32] refers to the vector ξ as trim controls. The next section formulates the types of trajectories and suggests parameter settings for achieving them.

2.3 General Steady-State Trim Conditions

In this section, Table 2 displays suggestions on trim settings. In a Graphic-User-Interface (GUI) a user chooses a flight trajectory and quantifies values for the specific parameters in the form $(\cdot)_0$. Targets for parameters in a back-end program ensure that the minimization algorithm searches for the trajectory the user desires. The *Target Initial Parameters* (TIP) describes the user and back-end set of target parameters. The trim algorithm uses the target numeric values to initialize the search. An optimization algorithm minimizes a cost function f by changing its variables that are subjected to constraining equations given by the *Parameter Constraints* (PC). From now, the *Trim Control Parameters* (TCP) refers to the content of the vector $\vec{\xi}$ that the trim algorithm uses to search for steady-state. An explanation of the *Derived Parameters* (DP) starts under table 2. In conclusion, the goal is finding the user custom TIP resulting in the trim condition $f(\dot{x}_d(\xi)) \to 0$ using the TCP set ξ subjected to the set of PC $c(\xi) = 0$.

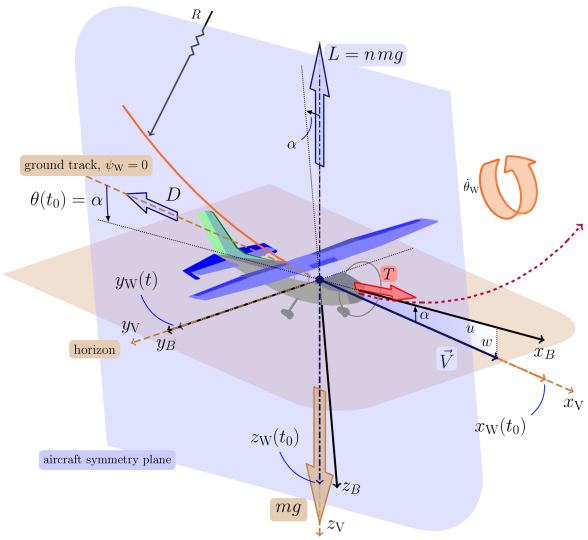


(a) Generic asymmetric condition that visualizes with red lines the variable trim attitudes $\alpha, \beta, \phi, \theta, \psi$ with $\gamma(\alpha, \beta, \phi, \theta)$ for the straight-and-level flight condition [32]. Determining quantities for straight ψ_w and level $\gamma = 0$ flight.

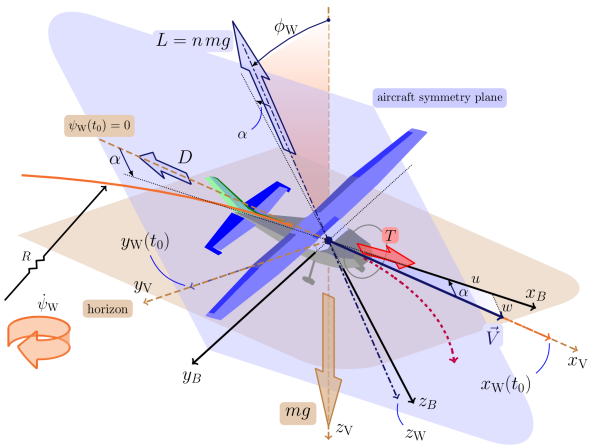


(b) Straight Crabbing Flight with no climb angle that directs north [32].

Figure 2: The steady-state straight-and-level and straight crabbing flight condition.



(a) The wings-level in pull-up trim condition valid at one instantaneous moment in time [32]. Determining quantities for wings-level $\phi = 0$, and pull-up $\dot{\theta}_W$ = positive non-zero flight.



(b) The steady-state coordinated and level turning flight condition [32]. Determining quantities for coordinated $\beta = 0$, level $\gamma = 0$, and turning $\dot{\psi}_W$ = non-zero flight.

Figure 3: The steady-state pull-up and coordinated turning flight condition [32].

The definition of a generic aircraft trim condition is $\dot{x}_d = 0$, but a specific type of trim maneuver also satisfies PCs set by TIPs. Common aircraft initial type of trim maneuver are straight flight and straight-and-level flight, push-over/pull-up, and steady-state turning flight. Straight flight is a steady-state condition where an aircraft moves along a straight line with a given heading and flight path angle. In addition, straight-and-level flight is a maneuver where this flight path angle and body roll angle is zero. Pull-over/pull-up flight is a maneuver that starts straight-and-level, then a positive flight path angular velocity $\dot{\gamma}_w > 0$ defines a pull-up and a negative $\dot{\gamma}_w < 0$ defines a push-over. In both cases, only allowing change in the vertical plane. Steady-state coordinated and level turning flight defines a maneuver where the flight path angle $\gamma = 0$ is zero, the load factor $n > 1$, and only allowing movement in the horizontal plane. However, fewer restricting choices for the turning flight maneuver are possible. Advised state settings for the maneuvers are summarized below w.r.t. the common aircraft state parameters.

Straight flight [10, 32, 43]		
TIP		$V = V_0$, $h = h_0$, $\gamma = \gamma_0$ and $p = q = r = 0$
PC		$\gamma = \sin^{-1}(\cos\alpha\cos\beta\sin\theta - \sin\beta\sin\phi\cos\theta - \sin\alpha\cos\beta\cos\phi\cos\theta)$
TCP		$\vec{\xi} = [\alpha, \beta, \phi, \theta, \psi, \delta_T, \delta_e, \delta_a, \delta_r]^T$
Straight and level flight [32, 43]		
TIP		$V = V_0$, $h = h_0$, $\gamma = \phi = p = q = r = 0$
PC		$\gamma = \sin^{-1}(\cos\alpha\cos\beta\sin\theta - \sin\beta\sin\phi\cos\theta - \sin\alpha\cos\beta\cos\phi\cos\theta)$
TCP		$\vec{\xi} = [\alpha, \beta, \theta, \psi, \delta_T, \delta_e, \delta_a, \delta_r]^T$
push-over/pull-up flight [10, 32, 43, 44]		
TIP		$V = V_0$, $h = h_0$,$\dot{\gamma} = \dot{\gamma}_0$, and $\gamma = \phi = 0$
PC		$\gamma = \sin^{-1}(\cos\alpha\cos\beta\sin\theta - \sin\beta\sin\phi\cos\theta - \sin\alpha\cos\beta\cos\phi\cos\theta)$
DP		$\phi_w = \tan^{-1}\frac{\cos\alpha\sin\beta\sin\theta+\cos\beta\cos\theta\sin\phi+\sin\alpha\sin\beta\cos\theta\cos\phi}{\sin\alpha\cos\theta\cos\phi+\sin\alpha\sin\phi}$ $\begin{bmatrix}p\\q\\r\end{bmatrix} = \begin{bmatrix}\cos\alpha\cos\beta & -\cos\alpha\sin\beta & -\sin\alpha\\ \sin\beta & \cos\beta & 0\\ \sin\alpha\cos\beta & -\sin\alpha\sin\beta & \cos\alpha\end{bmatrix}\begin{bmatrix}1 & 0 & -\sin\gamma\\ 0 & \cos\phi_w & \cos\gamma\sin\phi_w\\ 0 & -\sin\phi_w & \cos\gamma\cos\phi_w\end{bmatrix}\begin{bmatrix}\dot{\phi_w} = 0\\ \dot{\gamma} = \dot{\gamma}_0\\ \dot{\psi_w} = 0\end{bmatrix}$
TCP		$\vec{\xi} = [\alpha, \beta, \phi, \theta, \psi, \delta_T, \delta_e, \delta_a, \delta_r]^T$
Steady-State Turn [16, 32, 43, 44]		
TIP		$V = V_0$, $h = h_0$, $\dot{\psi}_w = \dot{\psi}_{w_0}$, $\gamma = \gamma_0$, and $\dot{\phi}_w = \dot{\gamma} = 0$
PC		$\gamma = \sin^{-1}(\cos\alpha\cos\beta\sin\theta - \sin\beta\sin\phi\cos\theta - \sin\alpha\cos\beta\cos\phi\cos\theta)$
DP		$\pm\phi_w = \tan^{-1}\left[\frac{(n^2-\cos^2\gamma)^{\frac{1}{2}}}{\cos\gamma}\right]$ $\phi_w = \tan^{-1}\frac{\cos\alpha\sin\beta\sin\theta+\cos\beta\cos\theta\sin\phi+\sin\alpha\sin\beta\cos\theta\cos\phi}{\sin\alpha\cos\theta\cos\phi+\sin\alpha\sin\phi}$ $\begin{bmatrix}p\\q\\r\end{bmatrix} = \begin{bmatrix}\cos\alpha\cos\beta & -\cos\alpha\sin\beta & -\sin\alpha\\ \sin\beta & \cos\beta & 0\\ \sin\alpha\cos\beta & -\sin\alpha\sin\beta & \cos\alpha\end{bmatrix}\begin{bmatrix}1 & 0 & -\sin\gamma\\ 0 & \cos\phi_w & \cos\gamma\sin\phi_w\\ 0 & -\sin\phi_w & \cos\gamma\cos\phi_w\end{bmatrix}\begin{bmatrix}\dot{\phi_w} = 0\\ \dot{\gamma} = 0\\ \dot{\psi_w} = \dot{\psi}_{w_0}\end{bmatrix}$
TCP		$\vec{\xi} = [\alpha, \beta, \phi, \theta, \psi, \delta_T, \delta_e, \delta_a, \delta_r]^T$
Minimizing $f(\xi) \to 0$		

Table 2: Advised Steady-state settings for the given maneuver

Achieving a trim maneuver requires additional dynamic system information by formulating PCs. When a user desires a target for a certain parameter (e.g., flight path angle $\gamma_0 = 0$), that is not explicitly part of the equations of motion it writes. Then the trim algorithm introduces the flight path angle as a PC. More general, *Primary parameters* are fundamental parameters to a simulation model and their numerical value must be known when evaluating a dynamic model. *Secondary parameters* are parameters that are combinations of primary parameters and not essential for model computing. The trim algorithm achieves the target quantity $\gamma_0 = 0$ by minimizing the difference between an expression of the secondary parameter γ as a function of the primary parameters, $\alpha, \beta, \phi, \theta$ as given in Eq. (2.7). An example PC that achieves the target the user desires may have the form $c(\xi)_1 = |\gamma_0 - \gamma(\alpha, \beta, \phi, \theta)| = 0$. PCs that derive p, q, r or alternatively quaternion derivative components $\dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{q}_4$ as a function of kinematic state variables, kinematic state derivatives, control input, and/or load factor makes sense from a user's perspective, as these are interpretable parameters. An additional advantage is implementing PCs directly in the trim module reduces data flow between modules. Implementing PCs directly doesn't come without a consequence.

$$\gamma = \sin^{-1}(\cos\alpha\cos\beta\sin\theta - \sin\beta\sin\phi\cos\theta - \sin\alpha\cos\beta\cos\phi\cos\theta) \quad \vee \quad \gamma = \theta - \alpha \iff \gamma, \phi = 0 \tag{2.7}$$

$$\phi_W = \tan^{-1}\frac{\cos\alpha\sin\beta\sin\theta + \cos\beta\cos\theta\sin\phi + \sin\alpha\sin\beta\cos\theta\cos\phi}{\sin\alpha\cos\theta\cos\phi + \sin\alpha\sin\phi} \tag{2.8}$$

Consider the following example that sets up the steady-state turning flight condition. Enforcing this maneuver requires PCs on the kinematic state variables $\dot{\phi} = \dot{\psi} = 0$. The goal is using the transformation of Eq. (2.11) to derive values for (p, q, r). Consider the following set of five subsidiary parameters $\phi_w, \theta_w, \dot{\phi}_w, \dot{\theta}_w = \gamma, \dot{\psi}_w$, that Eq. (2.11) uses, for achieving this goal. Here, $\dot{\theta}_w = \gamma$ already have a PC, leaving 4 free variable to target, derive or constrain. This set must find dependencies using PCs as a function of primary state variables or a targeting values that a user assigns. Otherwise, more variables than equations exist, making it less likely for the algorithm to find a solution. Secondly, the solution may become unrealistic because there exists a relation in the literature, but the trim modules don't describe the dependency. Two approaches consider a more and less user-friendly approach. The more user-friendly approach introduces an interpretable parameter, such as load factor n, which a user must

assign a target value to. The load factor and PC γ derive ϕ_w by solve for Eq. (2.10a). The trim module must use ϕ_w to solve Eq. (2.10b) and derive $\dot{\psi}_w$. Now, the assumptions Eq. (2.10a) uses are that all forces in z direction come from lift only. This focuses the trim module to select a value for the gravitation acceleration that Eq. (2.10b) uses. Equation (2.8) increases complexity and makes the target parameter less interpretable $\dot{\psi}_w = \dot{\psi}_{w_0}$, but makes fewer assumptions.

$$\begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \rightarrow \frac{1}{2} \begin{bmatrix} -q_2 & -q_3 & -q_4 \\ q_1 & -q_4 & q_3 \\ q_4 & q_1 & -q_2 \\ q_3 & q_2 & q_1 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{bmatrix} \tag{2.9}$$

$$\pm\,\phi_W = \tan^{-1}\left[\frac{(n^2 - \cos^2\gamma)^{\frac{1}{2}}}{\cos\gamma}\right] \tag{2.10a}$$

$$\dot{\psi}_W = \frac{g}{V}\tan\phi_W \tag{2.10b}$$

$$\begin{bmatrix} \cos\alpha\cos\beta & -\cos\alpha\sin\beta & -\sin\alpha \\ \sin\beta & \cos\beta & 0 \\ \sin\alpha\cos\beta & -\sin\alpha\sin\beta & \cos\alpha \end{bmatrix} \begin{bmatrix} 1 & 0 & -\sin\phi_w \\ 0 & \cos\gamma & \cos\phi_w\sin\gamma \\ 0 & -\sin\gamma & \cos\phi_w\cos\gamma \end{bmatrix} \begin{bmatrix} \dot{\phi_w} \\ \dot{\gamma} \\ \dot{\psi_w} \end{bmatrix} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.11}$$

The derived parameters set (DP) is the last set of parameter abbreviations. In general, the derived parameter set is necessary for secondary parameters that undergo a primary parameter conversion such that the (TIP) and/or (PC) connects to the simulation model.

2.4 Problem Exploration and Strategy

The problem size affects the aircraft trim problem size to some extent. First, the typical size and a specific number of variables in aircraft models are discussed, after which the effect of size on solving is presented. The work of Duke, Antoniewicz, and Krambeer [17] gives a very complete view of the generalized equations of motion and complexity regarding the aircraft trim problem and is highly recommended.

Stevens, Lewis, and Johnson [44], Hess [23], and Cook [11] describe 6 degrees of freedom (DOF) aircraft models that work with 12 state variables and 4 input variables. The 12 state variables describe the translational and rotational information of an aircraft's rigid body in three-dimensional space. The four input variables consist of the flight control surface and drive-train/engine manipulators. Peters and Barwey [42] describe a rotorcraft model that serves in their research as a demonstration program. The program consists of a fuselage, a four-bladed main rotor, a four-bladed tail rotor, inflow, and an engine/drive-train system that count of 12, 16, 8, 4, 0-4/0-2 state variables respectively. Dependent on the complexity of the drive-train and engine, the model consists of 40 to 46 system states. The DASMAT model contains 18 groups, resulting in using up to a total of 240 variables to simulate a 6DOF aircraft model [45]. The work of Guimarães Neto et al. [21] formulates flexible flight dynamics that uses (6+n)-DOF and is only valid for small deformations. Resulting in a 6+n problem subjected to n-6 constraints using n = 120 in the FEM model for the elastic degree of freedom. Cooke et al. [12] presents a quaternion based 6DOF aircraft model containing 14 state variables and 2 necessary quaternion constraining equations. Two necessary quaternion constraining equations ensure that the quaternion and quaternion derivative magnitude remains equal to 1. Holzapfel, Sturhan, and Sachs [25] apply for all components, except the flight simulating system, an implicit nonlinear first-order state-space model for plant dynamics modelling. This low-cost PC-based flight simulator executes in pseudo real-time using Microsoft Windows that incorporates a template-based trimming approach.

As explained in section 2.3, a trim algorithm finds a trim condition by minimizing the dynamic state derivatives using the kinematic state-, dynamic state- and control input variables. The state variables consist of the translational and rational positions and velocities of an aircraft's center of gravity and its engine model if included. The flight control surfaces and engine controls make up the list of input

variables. A trim algorithm minimizes the dynamic state derivatives of the aircraft's rigid body and its engine model [45]. But more dynamic states are possible, as described in Stevens, Lewis, and Johnson [44] (p. 191) where leading-edge flap actuator dynamics are additionally considered when trimming the F16 model. Chen [9] defines its aircraft model in steady-state when the control derivatives: lateral, collective, longitudinal, and directional control displacements are zero [9].

When minimizing an optimization problem, the problem should be a fully determined system. This means that there are as many independent relations as variables. The problem may also be under-determined, meaning fewer independent relations than variables. The trim algorithm must therefore have at least as many TCP, as dynamic state derivatives plus the secondary parameter-based constraint equations. Resulting in a trim problem size of at least 6 variables using the models given by presented by Stevens, Lewis, and Johnson [44], Hess [23], Cook [11]. The DASMAT trim problem minimizes 6 body acceleration states, 4 engine acceleration states, and 1 constraint, resulting in 11 equations. Under realistic flight conditions, the DASMAT model preferably uses 11 states trim control parameters to solve the problem. Creating a determined problem can still have more than one solution. As Marco, Duke, and Berndt [32] mention, if a trim goal is satisfied by some non-zero sideslip, combinations of non-zero rudder and aileron deflection may also be possible. Also, the opposite sign of the found parameters is a possibility. Choosing the search-space carefully cancels out their possibilities.

3 Algorithm Selection

The field of numerical optimization contains a large variety of techniques for solving complex problems. In this study, a framework is required that solves the aircraft trim problem. Here, an optimization algorithm optimizes a multi-objective function by manipulating its variables while considering the constraints. Before selecting a candidate numerical optimization framework, defining a set of demands is essential. The research scope describes the demands that narrow down the algorithm selection. The preliminary research viewed many candidate numerical frameworks. Constrained by the time limit of the master thesis phase and noticing in section 1.1 that basic method are capable of solving the aircraft trim problem, results in the following choice. The candidate numerical optimization framework must be capable of solving a basic aircraft trim problem at a basic level of algorithm complexity. As the algorithm becomes more complex, it must be capable of solving a more complex aircraft trim problem. This research considers an algorithm only a candidate if it's capable of working at a basic and more advanced level of complexity. This guarantees some working algorithm implementation at the end of this research.

3.1 Performance Indicators

The research goal of solving the aircraft trim problem is to find a close-to equilibrium condition that allows users to take over manual control. The upcoming definitions formulate general performance indicators.

- A *robust* algorithm performs satisfactorily on a wide spectrum of problems within its class while starting from a reasonable initial guess,
- An *efficient* algorithm minimizes excessive computing time and memory,
- An *accurate* algorithm finds a solution with precision without being too error sensitive which includes the effect of arithmetic rounding errors that occur during iterations on computers.

From the general numerical optimization definitions [38], a more trim problem-oriented description follows. Accuracy determines the closeness to equilibrium of the found initial condition. A high order of accuracy allows a user more time to construct a mental image and manually take over control of the simulation. Robustness assures that the algorithm performs well on aircraft with low and high maneuverability, but also on stable and unstable aircraft. Translating in being capable of trimming small single-engine and large multi-engine airplanes, but also rotorcraft and fighter aircraft. An efficient algorithm finds a solution while minimizing time and storage. The research considers robustness the most important and efficiency more important than accuracy. The definition of robustness describes the research goal without further explanation. Efficiency reduces pressure on the simulating environment and reduces the time a user waits on the desired initial condition. A high level of accuracy is important, but finding a quasi-equilibrium that allows users to take over manual control is not pushing numerical bounds.

3.2 Proven Numerical Methods

As mentioned in section 1.1, Baghdadi et al. [5] solves the aircraft trim problem using a bifurcation method. Millidere et al. [35] solves the problem using the Newton-Raphson method. Marco, Duke, and Berndt [32] used the DirectSearch C++ library. Using DirectSearch was their only choice as it minimizes the effort spent on coding. Thus, the experience reported in the literature would suggest using a Newton-based method for these problems. Stevens, Lewis, and Johnson [44] finds solutions to their trim problems using the fminsearch function in Matlab that essentially uses Nelder and Mead Simplex algorithm. The paper of Peters and Barwey [41] describes a general theory of rotorcraft trim appended with control augmentation. The trim routine uses a type of Newton-Raphson method.

Mastinu and Plöchl [33] explain different optimization techniques that are capable of solving automotive systems. Here, the efficiency and accuracy of uniformly distributed sequences, evolutionary strategies, and sequential quadratic programming techniques are visualized. In chapter 5.4 of Van Kampen [46] a 6 DOF F-16 model is trimmed using an Interval Algorithm (IA) and a SQP algorithm. The trim algorithm

minimizes the objective function using 7 TCP. Both algorithms are capable of finding the solution to the problem presented. The author concludes that the IA is guaranteed to find the global solution to a given problem, but is 200 times slower in finding a solution than the SQP method. A preliminary research inspected earlier used numerical solvers and found that the SQP had very appealing properties. Nocedal and Wright [38] referred to the SQP approach as a very effective method for solving nonlinear constrained optimization problems. The approach shows its strength when solving significant nonlinearities in its constraints, and allows a build-up process. The SQP approach can be constructed by starting with a basic unconstrained minimizing algorithm based on a type of Newton method. After assembling the basic algorithm, features are added that convert the unconstrained solver to a more complex constrained optimization algorithm.

3.3 Candidate Algorithm Framework

Based on the findings and considering the requirements, the SQP approach is a good choice as an optimization approach. The algorithm's basic form, choosing a type of Newton method, must be capable of solving a basic aircraft trim problem. The simplified aircraft model available in DUECA and familiar to the researcher is the aircraft model described by Hess [23]. For this research, the more complex model available in DUECA is the DASMAT aircraft model [45]. Section 4.6 provides a detailed description of both aircraft models. Again, The SQP approach emphasizes its feature selection first on robustness, then efficiency, and finally accuracy. The goal is to create a *generic* trim module which shares the description of robustness. But if only robustness was the goal, then the IA proposed by Van Kampen [46] that guarantees to find the global solution might suit better. The IA was 200 times slower on 7 variables, which may become worse when solving the DASMAT trim problem that contains at least 10 variables. The work of Peters and Barwey [42] solves the rotorcraft trim and stability problem using the Newton-Raphson method that led to 90+ state variables and took 360 minutes. Adding speed by including efficiency is therefore of great importance. Keeping these goals in mind, the next section explains the general numerical algorithm working principles and feature selection of Newton-type SQP algorithms.

3.4 Working Principle, Candidate Methods, and Features

The basic process of Newton-type methods and SQP approaches share many common steps. As all numerical optimization algorithms do, they work iteratively. During an iteration, evaluating the objective function and its constraints is part of most processes. The algorithm uses the information to construct a local model that approximates the actual problem. The local model consists of the problem's objective and constraint cost at the evaluating point, first-order derivative information, and sometimes second-order derivative information. After constructing the local model, a local optimization takes place to approximate the best step. Consideration of taking the step depends on the iteration method. Regardless if a method is taking the step or not, the decision will lead to the next iteration that repeats this cycle. No matter the order of execution and other deciding factors, the following points are present in a SQP Newton-type framework:

- Iteration methods available are Trust-region (T) and Line-search (L);
- Objective modeling subjected to constraint may use penalty methods, filtering, merit functions, or the augmented Lagrangian.
- First-order derivatives exploration is possible using finite differences;
- Second-order derivatives information gathering may use finite differences, Quasi-Newton methods, or least-squares methods;
- Step methods may execute Newton step methods, least-square methods, or conjugate gradient methods.

3.4.1 Iteration Methods

In general, iterative methods in numeric optimizations consist of two categories: trust-region methods and line search methods. Trust-region methods start an iteration with a fixed maximum step length, then find the optimal step direction. Line search methods start the iteration by first determining its step

direction, then reducing its step length. Figure 4 illustrates this process, where the trust-region step is constrained, and the line search step is the solution to the subproblem m_k.

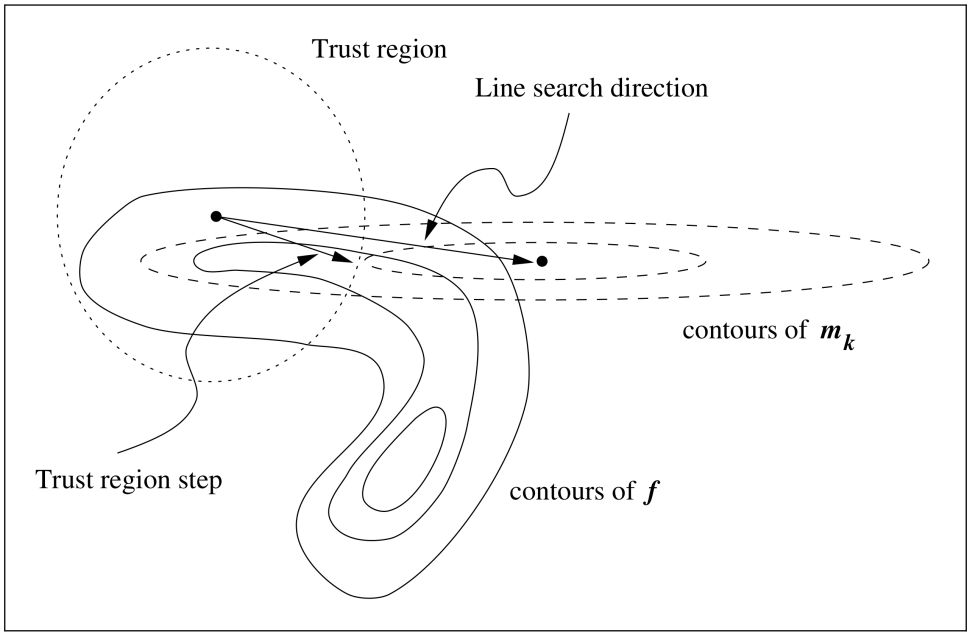


Figure 4: Line search and trust-region iteration method [37]. A graphical explanation of both iteration methods. Here, the 'full' step of the line search method and the constrained trust-region method step are visible.

Besides this difference, line search commonly uses a first-order prediction model while trust-region uses a second-order prediction model and is constrained to a maximum step length. The second-order model and maximum step length constraint make it possible for the trust-region method to handle an indefinite Hessian. This may make trust-region methods more reliable. At the same time, adding the trust-region constraint increases computational complexity and may cause an infeasible subproblem. Both line search and trust-region algorithms have their trade-offs. Implementing and undergoing numerical experiments is therefore the only possible way of finding their robustness and efficiency performance indicators.

The specific condition and steps that are crucial for making the algorithm converge define the section 3.5.

3.4.2 Objective Modelling Methods

Measuring the progress of a SQP algorithm towards the objective solution requires constant monitoring. A common way to track progress is by constructing a merit function. Candidate objective modelling functions are the penalty and augmented Lagrangian merit functions. Depending on the change of a merit function's magnitude and line-search or trust-region conditions, A step accepts or declines, making iterating possible. Popular penalty functions are the ℓ_1, ℓ_2, and ℓ_∞-norm functions. Other popular merit functions are the Fletcher augmented Lagrangian and (standard) Augmented Lagrangian function. The advantage of the ℓ_1 function is that the functions are easy to extend. Work well on problems that contain complementarity constraints and may serve as a safeguard when other merit functions fail. The augmented Lagrangian function is popular because of its simplicity but is more difficult to extend. They incorporate the Lagrangian parameter into the merit function and requiring more computational effort [6, 38], but may lead to faster converging speed [2]. The penalty functions are candidates because of their easy implementation. They must be safeguarded against an unrealistically high penalty for achieving the level of robustness required.

3.4.3 First-Order Derivative Methods

The first-order derivative describes gradient information essential for modelling the objective gradient behavior. Using the model proves useful when evaluating the next attempt in the neighborhood of an evaluation point. To provide a trim module that is generally applicable, the involvement of the trim mod-

ule in the aircraft models must be kept to a minimum. Besides minimum involvement, aircraft models may use data tables, analytical formulas, or any other representation to model aircraft behavior in some frame of reference. Therefore, using some analytical set of state equation derivatives to model all aircraft is not possible. Instead, the algorithm uses finite difference methods to approximate and gather first-order derivative information. In general, finite difference methods approximate the first-order derivative at some point by using a Taylor expansion up to the desired accuracy. The finite difference methods in this research are the forward Euler (E) and central difference (C) formulas. Both methods can increase their accuracy and robustness by implementing a Richardson Extrapolation method. Increasing robustness is also possible when implementing a second-order Lagrange interpolation polynomial. Richardson's extrapolation techniques make non-optimal step-size disturbances less influential when considering more evaluation points, while Lagrange interpolation performs well on unequally spaced data within the interpolation region as explained e.d. by Chapra and Canale [7] in Chapter 23. Because the generic application is the main focus, obtaining derivatives with Lagrange interpolation assistance is preferable. Richardson's extrapolation techniques will obtain more accuracy, but less robustness when dealing with unequally spaced data. As an addition to forward Euler and central difference, a second-order Lagrange interpolation polynomial implementation when extending the algorithmic complexity.

According to Nocedal and Wright [38] (p. 197), the accuracy of central difference is less impressive in practice, but it may still increase performance near the solution [38]. If assuming the optimal choice for finite difference step size under conditions given by Chapra and Canale, Nocedal and Wright [8, 38], the error becomes (E) $u^{1/3}$ and (C) $u^{2/3}$ with $u = (1/2)^{53}$, assuming a well-scaled problem. Calculating the optimal step size is not part of the research scope, but the error will affect the algorithm's performance. The more accurate (C) requires two additional function evaluations per variable against the more efficient (E) which uses one additional function evaluation per variable. The robustness of these techniques is limiting and depends on the finite differences between optimal step size (h) methods and aircraft model data spacing.

3.4.4 Second-Order Derivative Methods

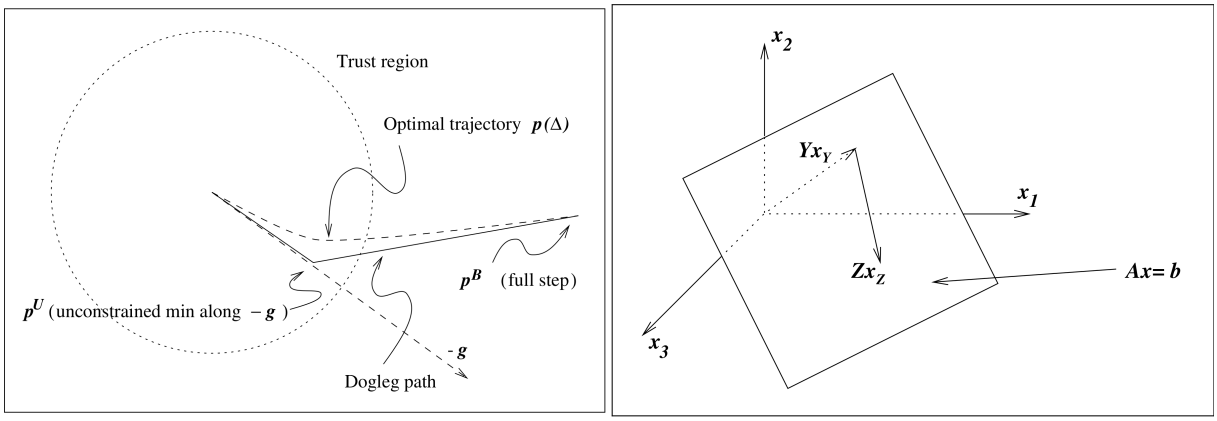
The second-order derivative matrix, known as the Hessian, models the changes in gradient. Constructing the Hessian has advantages for small to medium size problems. A major advantage of constructing a Hessian is being able to implement second-order optimality conditions in an algorithm. Optimality conditions, see Section **??**, may use the Hessian to verify and/or recognize a solution. A second major advantage is accelerating the search process. Implementing the Hessian increases the convergence rate near the solution compared to steepest descent methods dramatically, especially when solving difficult problems [38]. However, there exists more than one method to construct the Hessian. Constructing the Hessian is possible using finite differences, Quasi-Newton updating formulas, or least-squares techniques. Finite difference techniques can construct a Hessian accurately but are also inefficient. The trim module allows little involvement in dynamic models, making finite differences an obvious choice when using a Newton method. Yet again, efficiency is higher rated than accuracy, resulting in more suitable choices in the Quasi-Newton class or the least-squares class.

Quasi-Newton methods (QN) use gradient information to construct and update a Hessian approximation. They are proven to be practical and very effective for small to medium size problems. QN methods have a lower arithmetic cost than Newton's methods, are practically robust, and may achieve a super-linear convergence rate. Mixed approaches exist that construct a second-order derivative (Hessian) approximate matrix such are the BFGS, SR1, and the DFP, a mixed method called the Broyden class, see Nocedal and Wright [38] Chapter 6. The BFGS methods are one of the most effective classes because of their self-correcting properties. This guaranteed a positive semi-definite Hessian update if the initial BFGS matrix is positive semi-definite. The damped-BFGS is more effective at maintaining its update well-defined than a regular BFGS method. According to Nocedal and Wright [37] (p. 541), many SQP programs implement the damped-BFGS updating formula and perform well on many problems. This makes the damped-BFGS method a more favorable candidate feature than the unmodified BFGS method. The SR1 algorithm is less robust than BFGS methods and requires more safeguards, especially when combining this method with a line search method. The SR1 update formula is still useful for its cheap and efficient computation, and often reassembling the actual Hessian is more accurate than the BFGS updating formula. The effectiveness of Quasi-Newton methods is well-understood and proven by [1, 13, 15, 26, 27, 38, 40] and many others.

Least-square routines are in practice very effective solvers for nonlinear problems. They usually require only the evaluation of the Jacobian of a given problem and can retrieve first-order and approximated second-order derivative information. This process is fundamental in the Gauss-Newton (GN) line search method. The Levenberg-Marquardt method applies a similar process to a trust-region routine which copes with (GN)'s main weakness, which is its behavior its Jacobian becomes near rank-deficient. Local convergence of these techniques may become superlinear in the neighborhood of the solution [38].

3.4.5 Step methods

Conjugate gradient (CG) methods use conjugate sets to find solutions to general linear and nonlinear problems. In the CG inexact Newton class, the trust-region CG-Steihaug and line search Newton–CG are candidate step methods. Both candidate step methods are able of solving a constrained optimization problem by incorporating projected conjugate gradient (PCG). A second addition to both step methods, the residual minimizer proposed by Gould, Hribar, and Nocedal [19], which removes significant rounding errors during iteratively solving. Al these considerations make the step method very robust. The step method is capable of handling an indefinite Jacobian or Hessian matrix, solving unconstrained and constrained problems, and reducing rounding errors. Other candidate step methods for trust-region are the Cauchy-point, the Dogleg Method, and the Two-Dimensional Subspace Minimization. The line-search frame uses the Armijo backtracking condition to control step length [38].



(a) Dogleg step method minimizing a problem subjected to a trust-region constraint [37]

(b) Minimizing the objective function in the null-space of the constraints [37]

Figure 5: The steady-state straight-and-level and straight crabbing flight condition.

3.4.6 Summary

Selecting the methods and features emphasizes the performance indicator in the order of first robustness, then efficiency, and finally accuracy. Besides the performance indicators, practicality of implementation, proven aircraft trim numerical methods, and extendability in the level of complexity were leading factors during the preselecting stage. The trim product up until now uses the SQP framework, containing the following methods and features.

- Iteration methods available are Trust-region (T) and Line-search (L);
- Objective model uses ℓ_1-norm for line-search methods and trust-region uses the ℓ_2-norm;
- First-order derivatives approximation uses Central differences (C) and forwards Euler (E);
- Second-order derivatives use the Quasi-Newton damped-BFGS (B) or SR1 formulas (S) to construct the Hessian matrix;
- Step methods are the Projected conjugate gradient (P) and the Dogleg method (D) that is only available for trust-region iteration methods.

These choices define the basis of the evaluation framework. Section 3.5 describes the structuring of execution order, necessary conditions, and considerations for SQP algorithms in general.

3.5 Sequential Quadratic Programming Algorithm

Before going in-depth into the numerical procedure, the algorithm features are given in the following table

Section	reference	Section
Lagrangian Function	$\mathcal{L}(x, \lambda)$	Optimality Conditions 3.5.1
First-Order Necessary Conditions	KKT	Optimality Conditions 3.5.1
Gradient of the Lagrangian function	$\nabla_x \mathcal{L}(x, \lambda)$	Optimality Conditions 3.5.1
Second-Order Necessary Conditions	$w \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*) w \geq 0$	Optimality Conditions 3.5.1
Second-Order Sufficient Condition	$w \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*) w > 0$	Optimality Conditions 3.5.1
Auxiliary problem	Eq. 3.15	Objective Modeling Methods 3.5.2
Quadratic Model	Eq. 3.15	Objective Modeling Methods 3.5.2
Central Differences	$\partial f / \partial x_i$	First-Order Derivatives 3.5.3
Forward Euler	$\partial f / \partial x_i$	First-Order Derivatives 3.5.3
Jacobian of the objective	$J(x)$	First-Order Derivatives 3.5.3
Jacobian of the constraints	$A(x)$	First-Order Derivatives 3.5.3
Hessian Damped BFGS	B	The Damped-BFGS Method 3.5.5
Hessian SR1	B	The SR1 Method 3.5.6
Lagrange/(least squares) multiplier	λ	Lagrange Multiplier 3.5.8
Penalty value	μ	Penalty Updating Strategy 3.5.9
Projected Conjugate Gradient	Algorithm 1	Projected Conjugate Gradient Method 3.5.10
Second-Order Step Correction	$\hat{p}_k$	Second-Order Correction 3.5.11
Trust-region quadratic Model	q_μ	Trust-region subproblem 3.6.1
Trust-region merit function	$\phi_2(x, \mu)$	Trust-region subproblem 3.6.1
discrepancy parameter	ρ	Trust-region subproblem 3.6.1
Dogleg	p	The dogleg method 3.6.1
Trust-region pseudocode	Algorithm 3	Trust-Region Pseudocode3.9.1
line search merit function	$\phi_1(x, \mu)$	The Armijo Condition 3.7.1
Line search directional derivative	$\mathcal{D}(\phi_1(x_k; \mu_k))$	The Armijo Condition 3.7.1
Line search backtracking algorithm	Algorithm 2	Backtracking 3.7.1
Line search step acceptance	Condition 3.52	Line search Acceptance Condition 3.7.1
Line Search pseudocode	Algorithm 4	Line Search Pseudocode 3.9.2
Generic objective function	f	Aircraft Trim specifications 3.8
Generic constraining functions	c in Table 2	General Steady-State Trim Conditions 2.3

Table 3: The symbols, conditions and equations integrated in the SQP Algorithm

The general nonlinear objective function subjected to constraints defines the following problem,

$$\min_{x \in \Omega} f(x) \tag{3.1a}$$

$$c_i(x) = 0, \qquad i \in \mathcal{E} \tag{3.1b}$$

$$c_i(x) \geq 0, \qquad i \in \mathcal{I} \tag{3.1c}$$

Here, f is an objective function that is subjected to constraints c both as a function of the variables x. The goal of such a problem is finding the set of points x that results in the lowest possible value for f and satisfies all constraints c. This set of points is called the global minimizer of the problem (3.1). A local solution is noted as x^* where $(\cdot)^*$ indicates the minimizer and $(\cdot)^{**}$ a maximizer. The imposed constraints span the feasible region of $f(x)$, and limit the search space for the minimizer as Eq. (3.2) describes.

$$\Omega = \{x | c_i = 0, i \in \mathcal{E}; c_i(x) \geq 0, i \in \mathcal{I}\} \tag{3.2}$$

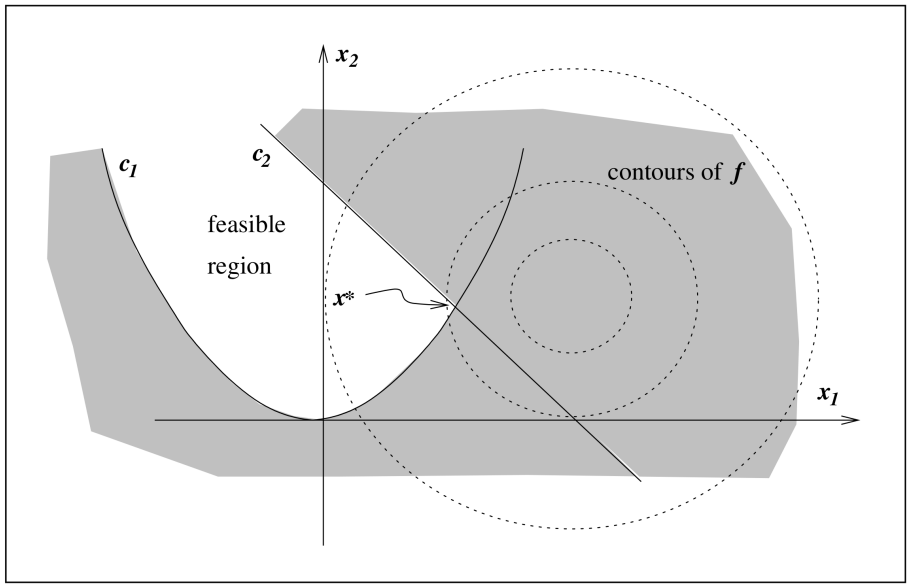


Figure 6: Geometrical representation of a constrained optimization problem [37]

3.5.1 Optimality Conditions

Regardless of the specific algorithm features, the algorithm needs mathematical conditions for checking and verifying solution points. Two types of optimality conditions exist: Necessary and Sufficient Conditions. Necessary conditions must be satisfied by any solution point. Sufficient conditions guarantees that x^* is, in fact, a solution [38]. The difference between the two conditions is the derivation perspective regarding the objective function and constraints. Necessary conditions assume x^* is a local solution, then formulates properties of f and c. Sufficient conditions are conditions that ensure f and c are local solutions.

Before introducing the advantages and mathematical conditions of optimality conditions, an example is provided that explains the difference between the necessary and sufficient conditions. Consider a valley where a local minimum must be found without depending on our eyes and ears. The valley surface area is very smooth, free of obstacles, and safe for such experiments. We use a walking stick to poke around and feel how the surface of the valley directly around us behaves. With our feet, we can feel the slope of the landscape. We will start at some point x_k in the valley, where the valley is our feasible search space Ω. The first question is: How do we know that we have found a local minimum? Deriving conditions that define the properties of a local minimum point x^* in the search area is necessary for recognizing if a point is or is not a local minimum. The first condition uses the information from the slope of the surface on which we stand. We state that to consider our position a local minimum, it is necessary that the slope of the surface on which we stand is zero $\nabla f(x^*) = 0$. Looking critically at our initial problem, we recognize that a local minimum in a valley is not necessarily a point. It may also be a line or a flat region in some part of the valley. A local point that agrees with $\nabla f(x^*) = 0$ is a saddle point, which is not a local minimum. Providing more conditions makes our claim that we have found a local minimum stronger. When we find a point that satisfies $\nabla f(x^*) = 0$, we use the walking stick to feel the surrounding. This way we can feel the slope we stand on change, giving us insight into the curvature of the surrounding. We suppose that if x^* is a local minimum, then it is necessary that the curvature stays zero or positive $\nabla^2 f(x^*) \geq 0$ and $\nabla f(x^*) = 0$ is satisfied. This makes calling a line or flat region a local minimum possible. However, if we find ourselves in a position where $\nabla f(x^*) = 0$ and we find that poking around us only leads upwards $\nabla^2 f(x^*) > 0$, then it's sufficient to guarantee that position x^* a strict local minimum.

In the example above, the first-order optimality conditions uses function information up to the first-order derivative. A second-order optimality condition uses information up to the second-order derivative. The second-order necessary condition accepts more possibilities if a point satisfies $\nabla^2 f(x^*) \geq 0$ and $\nabla f(x^*) = 0$. The second-order sufficient condition guarantees a stronger local minimum if a point

satisfies $\nabla^2 f(x^*) > 0$ and $\nabla f(x^*) = 0$, but may not capture all points or may even fail (consider the Hessian at $x = 0$ for minimization problem: $\min x^6$).

The example above has relatively straightforward optimality conditions because it's an unconstrained optimization problem. However, this study requires optimality conditions for a constrained optimization problem. A common approach in constrained optimization is using the Lagrangian, which is capable of describing the objective and constraints in one function. The general constrained optimization problem (3.1) uses the following Lagrangian function:

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cap \mathcal{I}} \lambda_i c_i(x). \tag{3.3}$$

Here, the Lagrangian $\mathcal{L}(x, \lambda)$ describes the objective $f(x)$, $c_i(x)$ the constraints, and λ_i the Lagrange multiplier. The set of constraint active for all feasible points x is the union of the equality constraints $\mathcal{E}$ and active inequality constraints. This union set defines the *Active set* of constraints and is given below.

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} | c_i(x) = 0\}. \tag{3.4}$$

The first fundamental set of conditions is called the *First-Order Necessary Conditions*. This set of conditions determines that x^* is a local minimizer if the conditions hold. The necessary condition is called first-order, as it uses gradient information about the objective and constraint functions. This set of conditions is also known as the *Karush-Kuhn-Tucker* (KKT) conditions and formulates the following statements:

$$\begin{aligned}
\nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, && && \text{(3.5a)}\\
c_i(x^*) &= 0, && i \in \mathcal{E} && \text{(3.5b)}\\
c_i(x^*) &\geq 0, && i \in \mathcal{I} && \text{(3.5c)}\\
\lambda_i^* &\geq 0, && i \in \mathcal{I} && \text{(3.5d)}\\
\lambda_i^* c_i(x^*) &= 0, && i \in \mathcal{I} \cup \mathcal{E}. && \text{(3.5e)}
\end{aligned}$$

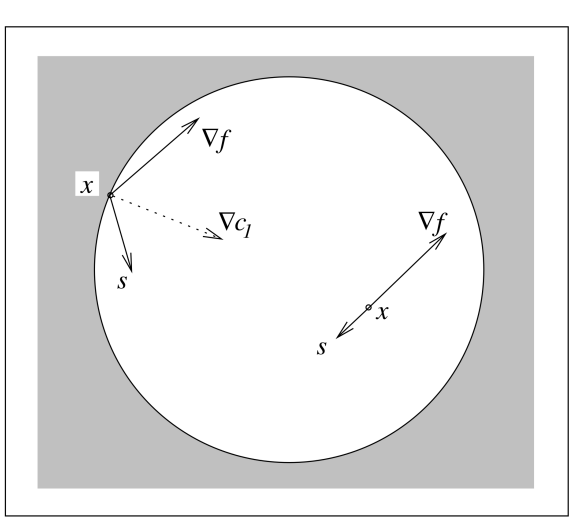
Here, $\nabla \mathcal{L}(x^*, \lambda^*)$ is the gradient of the Lagrangian function and λ^* the Lagrange multiplier. The gradient of the Lagrangian function contains derivative information for the objective function and constraints. More specific, the active set only contains the set of equality and violated inequality constraints. Omitting the rest $i \notin \mathcal{A}(x^*)$ replacing Eq. (3.5a) by

$$0 = \nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla f(x^*) - \sum_{i \in \mathcal{A}(x^*)} \lambda_i^* \nabla c_i(x^*) \tag{3.6}$$

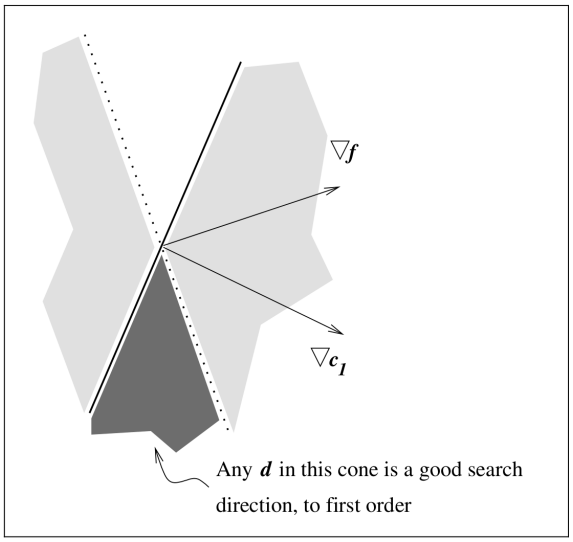
These conditions provide one of the stopping criteria of the algorithm, making it fundamental for the numerical scheme. It also shows the role of the Lagrange multiplier λ_i. The Lagrange Multiplier describes the sensitivity of the objective function value to each constraint. Each λ_i indicate the amount that f is pushing or pulling on each individual c_i. The work of Nocedal and Wright [38] page 342 describes a sensitivity analysis that shows the effect of small perturbations on the Lagrange multiplier. It shows, if $\lambda_i^* ||\nabla c_i(x^*)||$ is large, the sensitivity of the optimal value to placement of i^{th} constraint depend on the quantity. Small quantities show that the dependence is not too strong. If $\lambda_i^* = 0$, small perturbations to c_i will result in insignificant changes of the optimal objective value. Using active constraint set $\mathcal{A}(x)$ and feasible set x, the linearized feasible directions set $\mathcal{F}(x)$ is

$$\mathcal{F}(x) = \left\{ \quad d \middle| \begin{array}{ll} d^T \nabla c_i(x) = 0, & \text{for all } i \in \mathcal{E} \\ d^T \nabla c_i(x) \geq 0, & \text{for all } i \in \mathcal{A}(x) \cap \mathcal{I} \end{array} \right\} \tag{3.7}$$

Here, the First-Order feasible set $\mathcal{F}(x)$ is a cone, shares geometric properties with the set Ω, and does not rely on the algebraic specifications. The set $\mathcal{F}(x)$ does depend on the definition of the constraints.



(a) Problem: $\min x_1 + x_2$ s.t. $-x_1^2 - x_2^2 + 2 \geq 0$ starting with and without constraints active (Numerical Optimization page 311 [38])



(b) The cone region that combines the objective goal by descending in the direction $-\nabla f$ and satisfies $\mathcal{F}(x)$ simultaneously [37]

Figure 7: An example constrained optimization problem [37]. Figure 7a Visualizes and describes a constrained minimization problem. Figure 7b illustrates the linearized feasible set and objective goal of the top-left point x in figure 7a.

The first-order conditions give insight to the relation between $\nabla f(x)$ and $\nabla c_i(x)$ at x^*. When the first-order conditions are satisfied, movement of any w from $\mathcal{F}(x)$ results in one of two scenarios. Either the first-order approximate increases $w^T \nabla f(x^*) > 0$, or kept the same $w^T \nabla f(x^*) = 0$. First-order alone cannot determine if the function value will increase or decrease along the direction of movement. This makes the second derivative especially convenient, as it captures curvature information of the Lagrangian function. However, the second-order derivative requires stronger smoothness assumptions. Additionally, it assumes that the functions f and c_i in Ω are twice continuously differentiable. Here, the *critical cone* definition gives more insight than directions from $\mathcal{F}(x)$, while λ^* satisfies the first-order necessary conditions (3.5a) as defined as follows:

$$w \in \mathcal{C}(x^*, \lambda^*) \iff \begin{cases} \nabla c_i(x^*)^T w = 0, & \forall i \in \mathcal{E}, \\ \nabla c_i(x^*)^T w = 0, & \forall i \in \mathcal{A}(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* > 0, \\ \nabla c_i(x^*)^T w \geq 0, & \forall i \in \mathcal{A}(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* = 0, \end{cases} \tag{3.8}$$

Curvature information that the Hessian of the Lagrangian provides at the local solution x^* are non-negative critical directions. These directions belong in the set $\mathcal{C}(x^*, \lambda^*)$. It is a necessary condition that if x^* is a local solution and LICQ holds while λ^* satisfies the KKT conditions, the Hessian of the Lagrangian has non-negative curvature in $\mathcal{C}(x^*, \lambda^*)$. These *Second-Order Necessary Conditions* are mathematically given below

$$w \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*) w \geq 0, \qquad \forall w \in \mathcal{C}(x^*, \lambda^*). \tag{3.9}$$

Under these conditions, the Hessian of the Lagrangian $\nabla^2_{xx} \mathcal{L}(x^*, \lambda^*)$ is positive semi-definite. This means that none of the eigenvalues of $\nabla^2_{xx} \mathcal{L}(x^*, \lambda^*)$ are negative. The following conditions define the *Second-Order Sufficient Conditions*: If for some $x^* \in R^n$ there exists λ^* such that the first-order necessary conditions (3.5) are satisfied and condition (3.11) holds, then x^* is a strict local minimizer.

$$w \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*) w > 0, \qquad \forall w \in \mathcal{C}(x^*, \lambda^*), w \neq 0. \tag{3.10}$$

Under these conditions, the Hessian of the Lagrangian $\nabla^2_{xx} \mathcal{L}(x^*, \lambda^*)$ is positive definite. This means that none of the eigenvalues of $\nabla^2_{xx} \mathcal{L}(x^*, \lambda^*)$ are negative or zero.

Second-order conditions may also describe conditions on alternative projections. A popular approach is using a two-sided projection of $\nabla^2_{xx} \mathcal{L}(x^*, \lambda^*)$ onto the relating subspace of $\mathcal{C}(x^*, \lambda^*)$. When the

multiplier λ^* satisfies and is a unique solution of the KKT conditions and strict complementarity holds, with $A(x^*)^T = [\nabla c_i(x^*)]_{i \in \mathcal{A}(x^*)}$ and Z the full column Null space of the $A(x^*)$ spanning $\mathcal{C}(x^*, \lambda^*)$ with its columns which describes

$$\mathcal{C}(x^*, \lambda^*) = \{Zu | u \in R^{|\mathcal{A}(x^*)|}\}. \tag{3.11}$$

then restate condition (3.9) by

$$u^T Z^T \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*) Zu \geq 0. \tag{3.12}$$

and restate condition (3.11) by

$$u^T Z^T \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*) Zu > 0. \tag{3.13}$$

Mathematical formulations are essential for finding the solution to the problem by taking iterative steps. Solving the problem by iterative steps requires a local model of the objective function to obtain an iteration step. The following part describes the objective model that the algorithm uses.

3.5.2 Objective Modeling Methods

The general nonlinear problem which is given in Eq. (3.1) requires a model to achieve the objective. Again, the objective is finding a point x that results in the lowest value of the objective function $f(x)$ while satisfying all constraints $c(x)$. A method to model the behavior is using an approximation model valid close to the current evaluation point x_k. By finding the minimizing step p of an approximation model, new iteration points $x + p$ are calculated that minimize the problem (3.1) in a sequence of steps. The quadratic program below models problem Eq. (3.1) locally.

$$\min_p \quad f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla^2_{xx} \mathcal{L}_k p \tag{3.14a}$$

$$\text{subjected to} \quad \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E} \tag{3.14b}$$

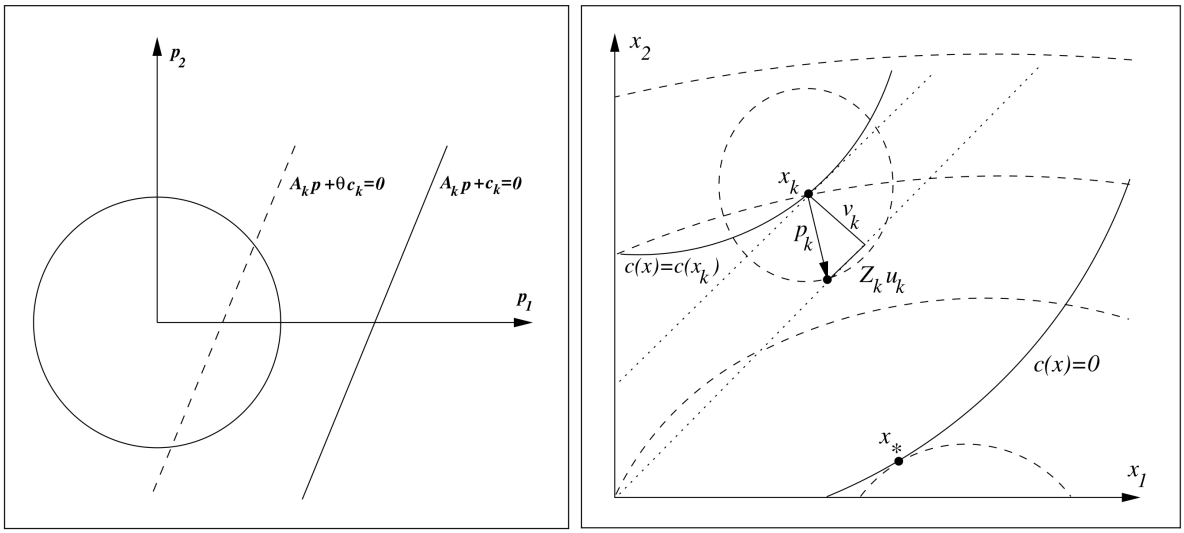
$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I} \tag{3.14c}$$

Here, the notation $(\cdot)_k$ indicates the iteration sequence number, p indicates the iteration step, ∇f_k is the gradient and $\nabla^2_{xx} \mathcal{L}_k$ is the Lagrangian Hessian. The matrix A_k is the Jacobian of the constraints in $\mathcal{A}_k$, as formulation (3.4) describes. This program consists of a quadratic approximation model Eq. (3.15a) that reflects the objective function and approximation models of the constraints both near x_k. The equality and inequality constraints use the linear approximation models (3.15b) and (3.14c) to simulation the constraint behavior. The quadratic program solves the problem sequentially using the *Equality-constrained Quadratic Programming* (EQP) approach. This approach uses an active set $\mathcal{A}_k$ that imposes all violated constraints as equality constraints and ignores all others. One of the algorithms of Section 3.5.7 minimizes the quadratic problem below resulting in a new iteration step p

$$\min_p \quad f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla^2_{xx} \mathcal{L}_k p \tag{3.15a}$$

$$\text{subjected to} \quad A_k p + c_k = r_k, \tag{3.15b}$$

Here, r_k is the residual vector [19, 38, 49]. The minimization of the residual vector depends on the iteration scheme. When using the line search method, this vector is minimized without step length limitations. The trust-region iteration method must take an alternative approach because of its trust-region constraint. The trust-region method must operate within its trust-region radius, but must also satisfy the active constraints. Satisfying both goals might not always be possible because the minimizing step length p is larger than the trust-region radius length Δ. Figure 8a illustrates an approach known as the relaxation method. By making the constraint violation less severe, finding a step that satisfies both the trust-region radius constraint and objective constraints is possible. The goal of the relaxation method is to satisfy all constraints while using the smallest value θ. Figure **??** shows the minimization step p_k consisting of relaxation step v_k and optimal objective step $Z_k u_k$ while stays in the trust-region (if using trust-region method). Section 3.5.7 elaborates what algorithm this research uses to find the optimal step p_k.



(a) Relaxation of the linearized constraining equation $A_k p + c_k = 0$. Parameter θ reduces the violation preventing trust-region radius and problem constraining equation inconsistency [37]

(b) Minimizing both $f(x_k)$ and $c(x_k)$ simultaneously with optimal constraining step v_k and optimal null-space projection step $Z_k u_k$, giving combining to step p_k [37]

Figure 8: Minimizing a trust-region constrained optimization problem [37].

The information provided in the previous part shows the similar and different objectives of the line search and trust-region iteration method. Both methods search for a sequence of steps that results in a feasible optimal point of the constrained optimization problem (3.1) by repeatedly solving a local quadratic program (3.15). One of the differences between line search and trust-region is the step accepting criteria. The line-search method checks if the iteration step p_k from solving (3.15) meets descent conditions at $(x_k + p_k)$. The trust-region method checks the quality of its local model.

3.5.3 First-Order Derivatives

Finding the first-order derivative of a black-box aircraft model requires the evaluation of specific data points, also called nodes. Formulas that originate from Taylor's theorem can approximate derivative information with finite precision. A machine can't compute infinitesimal steps to measure a function's sensitivity to variable change. The computer rather perturbs a function by changing its variable with small but finite steps. The small difference in variable results in a function value difference. Formulas using the ratio between function value differences and finite variable differences approximate the derivative. Labeling these derivative approximation techniques as finite differences formulas. The accuracy of the approximation depends on the step size h, truncation error, rounding error, and method of data representation that a black-box model uses. For example, the central difference formula contains the following errors

$$\begin{array}{ccccccc} f'(x_i) & = & \frac{f(x_{i+1})-f(x_{i-1})}{2h} & + & \frac{e_{i+1}-e_{i-1}}{2h} & - & \frac{f''(\xi)}{6}h^2 \\ \text{True} & & \text{Finite difference} & & \text{Round-off} & & \text{Truncation} \\ \text{value} & & \text{approximation} & & \text{error} & & \text{error} \end{array} \tag{3.16}$$

The formulas use two nodes at some step size h away from the evaluation point to construct an approximation of the derivative. This finite approximation results in a second-order accurate approximation $\mathcal{O}(h^2)$. Controlling the step size controls the magnitude of the round-off and truncation error. Equation (3.16) demonstrates how the step size controls both types of errors. A relatively smaller h makes the round-off error the dominant error, while a relatively larger h increases the truncation error. Besides the different types of errors a finite difference strategy provides, enumerate strategies exist for fitting derivatives. See Chapter 4.3 of Chapra and Canale [8] for more information. Besides Taylor oriented methods, interpolation, and regression methods may also model a derivative, with each their advantages. Approximating the derivative of continuous, smooth, equally spaced data-based models may use the

Euler forward Eq. (3.17a) or finite differences Eq. (3.17b) formulas. Chapra and Canale [8] suggests in Chapter 23.3 using second order Lagrange interpolation polynomial Eq. (3.17c) for approximating the derivative of unequally spaced data-based models. If the data-based model is noisy, regression techniques such as Vandermonde [29] or least square fit of a sinusiod [8] are recommended.

$$\frac{\partial f}{\partial x_i} = \frac{f(x + he_i) - f(x)}{h} + \mathcal{O}(h) \tag{3.17a}$$

$$\frac{\partial f}{\partial x_i} = \frac{f(x + he_i) - f(x - he_i)}{2h} + \mathcal{O}(h^2) \tag{3.17b}$$

$$\frac{\partial f}{\partial x_i} = f(x_{i-1}) \frac{2x - x_i - x_{i+1}}{(x_{i-1} - x_i)(x_{i-1} - x_{i+1})} + f(x_i) \frac{2x - x_{i-1} - x_{i+1}}{(x_i - x_{i-1})(x_i - x_{i+1})} + f(x_{i+1}) \frac{2x - x_{i-1} - x_i}{(x_{i+1} - x_{i-1})(x_{i+1} - x_i)} \tag{3.17c}$$

When considering an equally spaced step size h, the second order Lagrange interpolation formula (3.17c) becomes the central difference formula (3.17b). Table 4 provides a summary per approximation method the data point requirements, additional specialization, and the order of approximation error. The regression method is excluded from this research, but Chapra and Canale [8], Khoury and Harder [29], and Hauser [22] cover the basic concepts regarding this topic.

Method	Requirements	Specialization	Eq.	Approx. Error	Ref
Euler forward	Current + 1 nodes	equally spaced	(3.17a)	$\mathcal{O}(h)$	[38] p. 195
Central difference	2 nodes	equally spaced	(3.17b)	$\mathcal{O}(h^2)$	[38] p. 196
Lagrange interpolation	3 nodes	unequally spaced	(3.17c)	Error $\leq \mathcal{O}(h^2)$	[8] p. 658
Regression	n nodes	noisy	-	-	[8, 22, 29]

Table 4: Summary: Numerical derivative methods

Using one of these methods makes mapping derivative information possible. The first-order derivative information may construct a gradient vector or a Jacobian matrix. Depending on the representation of the objective function, capturing the derivative information in a gradient vector $\nabla f(x)$ or a Jacobian matrix $J(x)$ is possible. Modeling of the constraints requires the use of the Jacobian of the constraints $A(x)$.

$$\nabla f(x) = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}]^T, \qquad \nabla c(x) = [\frac{\partial c}{\partial x_1}, \frac{\partial c}{\partial x_2}, \ldots, \frac{\partial c}{\partial x_n}]^T \tag{3.18}$$

The gradient of the merit function and constraints is given by $\nabla f(x)$ and $\nabla c(x)$ in Eq. (3.18), respectively. Both vectors are dependent on the same number of variables n in this research.

$$J(x) = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(x)}{\partial x_1} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix}, \qquad A(x) = \begin{bmatrix} \frac{\partial c_1(x)}{\partial x_1} & \cdots & \frac{\partial c_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial c_i(x)}{\partial x_1} & \cdots & \frac{\partial c_i(x)}{\partial x_n} \end{bmatrix} \tag{3.19}$$

Here, $J(x)$ and $A(x)$ are the Jacobian matrices of the merit function and constraints, respectively. The number of optimization parameters is given by m, and the number of constraint equations is given by $i \in \mathcal{I} \cup \mathcal{E}$.

Numerical Errors
Before implementing numerical techniques in practice, a solid understanding of the working principles is necessary. Constructing a numerical iteration-based algorithm may result in many types of errors. Hauser [22] states (on page 27) seven sources of errors that occur in any numerical models.

3.5.4 Second-Order Derivative

The corrected second-order derivative matrix $\nabla^2_{xx}\mathcal{L}_k$ known as the Hessian of the Lagrangian contains curvature information of the function $f(x)$ corrected by the curvature information of the constraints. This process makes it less attractive to leave the feasible space far from the solution and not attractive to leave the feasible space close to the solution. The matrix $\nabla^2_{xx}\mathcal{L}_k$ estimates the full Quasi-Newton approximations, SR1 or damped-BFGS, by using the following adjustments

$$s_k = x_{k+1} - x_k, \qquad y_k = \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1}). \tag{3.20}$$

3.5.5 The Damped-BFGS Method

The damped-BFGS method performs an unmodified BFGS update when θ_{B_k} or θ_{H_k} are equal to 1. The damped-BFGS method performs an interpolation when $\theta_{B_k} \in (0,1)$ or $\theta_{H_k} \in (0,1)$. This interpolation makes sure that the BFGS update remains positive definite by staying close enough to the current BFGS approximation. If the value of θ_{B_k} or θ_{H_k} are zero, the current approximation skips updating, preventing an indefinite matrix [14, 37, 38].

$$r_{B_k} = \theta_{B_k} y_k + (1 - \theta_{B_k}) B_k s_k \tag{3.21}$$

$$r_{H_k} = \theta_{H_k} s_k + (1 - \theta_{H_k}) H_k y_k \tag{3.22}$$

Eq. (3.23) and (3.24) defines the parameters with $\hat{\theta} = 0.2$.

$$\theta_{B_k} = \begin{cases} 1 & \text{if } s_k^T y_k \geq \hat{\theta} s_k^T B_k s_k \\ (1-\hat{\theta}) s_k^T B_k s_k / (s_k^T B_k s_k - s_k^T y_k) & \text{if } s_k^T y_k < \hat{\theta} s_k^T B_k s_k \end{cases} \tag{3.23}$$

$$\theta_{H_k} = \begin{cases} 1 & \text{if } s_k^T y_k \geq \hat{\theta} y_k^T H_k y_k \\ (1-\hat{\theta}) y_k^T H_k y_k / (y_k^T H_k y_k - s_k^T y_k) & \text{if } s_k^T y_k < \hat{\theta} y_k^T H_k y_k \end{cases} \tag{3.24}$$

Replacing the vectors (s_k, y_k) of the standard BFGS Hessian approximation with (s_k, r_{B_k}) gives the damped-BFGS Hessian approximation.

$$\text{(Damped BFGS)} \qquad B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{r_{B_k} r_{B_k}^T}{r_{B_k}^T s_k} \tag{3.25}$$

Substituting s_k for r_{H_k} in the standard BFGS inverse Hessian formula gives the damped-BFGS inverse Hessian updating formula.

$$\text{(Damped BFGS)} \qquad H_{k+1} = (I - \frac{r_{H_k} y_k^T}{y_k^T r_{H_k}}) H_k (I - \frac{y_k r_{H_k}^T}{y_k^T r_{H_k}}) + \frac{r_{H_k} r_{H_k}^T}{y_k^T r_{H_k}} \tag{3.26}$$

The parameter s_k contains step information and y_k contains the corrected gradient difference. The Quasi-Newton Hessian matrices used are the SR1 and damped BFGS updating formulas. The damped BFGS is well proven in practice and guarantees to stay positive definite when its initial choice is positive definite. The SR1 updating formula is a good choice according to Nocedal and Wright [38] for trust-region methods when the necessary safeguards are implemented. Both iteration methods use the SR1 updating but correct the Hessian when the matrix is indefinite. The correction by subtracting the identity matrix multiplied by a factor times the most negative eigenvalue from the Hessian.

3.5.6 The SR1 Method

in the formulas of BFGS and DFP the (inverse) approximation of the Hessian B_{k+1} (or H_{k+1}) are updated using B_k (or H_k) resulting in a rank two modification. There exist a simpler scheme to update these matrices by using a rank-one modification. This modification produces a symmetric rank-one modification that is known as the SR1 method. The SR1 update produces a symmetric rank-one update that satisfies the secant equation but does not guarantee a positive definite matrix update. This part describes the possibilities and necessary safeguards of the SR1 method. The following sources describe the derivation of the method [37, 38]. The upcoming SR1 formula describes the Hessian approximate that satisfies the secant equation.

$$\text{(SR1)} \qquad B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} \tag{3.27}$$

Solving the Sherman-Morrison formula results in the inverse SR1 updating formula H_{k+1} that is capable of approximating the inverse Hessian.

$$\text{(SR1)} \qquad H_{k+1} = H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k} \tag{3.28}$$

In contrast to the BFGS method, the SR1 updating formula does not guarantee a positive definite update B_{k+1}. The SR1 formula has the tendency of becoming indefinite. This property is a major drawback

when implementing the SR1 formula in a line search framework and requires a set of safeguards. Trust-region methods provide a framework where the SR1 method is very useful. The trust-region framework regards the property that allows an indefinite Hessian approximation as a chief advantage. A major disadvantage of the SR1 method is the singularity found in the denominator of Eq. (3.27) and Eq. (3.28). The denominator can become very small or even zero in very specific situations. according to Nocedal and Wright [38], three cases describe the denominator of both SR1 formula's (3.27) and Eq. (3.28).

1. If $(y_k - B_k s_k)^T s_k \neq 0$, A unique one-rank update for Eq. (3.27) exist that satisfies the secant equation;
2. If $y_k = B_k s_k$, no unique one-rank update besides $B_{k+1} = B_k$ exist that satisfies the secant equation;
3. If $y_k \neq B_k s_k$ and $(y_k - B_k s_k)^T s_k = 0$ no unique symmetric one-rank update exists that satisfies the secant equation.

Although the rank two BFGS update formula guarantees a positive definite (non-singular) matrix, the rank one SR1 updating formula comes with its advantages:

(i) Safeguarding the SR1 methods that prevent numerical instabilities and breaking down updating formula is achievable using the items in list Eq. (3.29);

(ii) The SR1 formula is capable of generating matrices that are a good approximation of the true Hessian and often more accurate than the BFGS approximation;

(iii) Constrained optimization based on a quasi-Newton method or optimization methods for partially separable functions may not satisfy $y_k^T s_k > 0$. An (inverse) Hessian approximation update using the BFGS method is not recommended, while $y_k^T s_k > 0$ is not true. Here, An Indefinite Hessian approximation is convenient because they reflect indefiniteness in the true Hessian.

To safeguards the SR1 formula from (i), The Hessian approximation is skipped if the denominator is very small. This safeguard ensures Eq. (3.27) is only updated if

$$|s_k^T (y_k - B_k s_k)| \geq r ||s_k|| \, ||y_k - B_k s_k|| \tag{3.29}$$

The parameter $r \in (0, 1)$ is a small number of approximately $r = 10^{-8}$. If inequality (3.29) is not satisfied, the Hessian approximation sets $B_{k+1} = B_k$. This process is one of the major advantages when using SR1 over BFGS. The condition that safeguards to skip a step $s_k^T (y_k - B_k s_k) \approx 0$ occurs infrequently in the SR1 method. This specific condition only occurs when the vectors align in a special way, and skipping the update does not negatively affect the iteration process. In contrast, the BFGS method condition that must be satisfied $s_k^T y_k > 0$ fails easily. Because this condition fails easily, the update cannot be skipped. Too many missing updates may degrade the Hessian approximation.

3.5.7 Step Method

The quadratic function that Eq. (3.15) formulates models of the objective locally. The solution to this problem is still unknown. However, the optimality conditions describe what an algorithm must look for. First, the algorithm defines a local model using the first- and second-order approximation techniques, then the optimality conditions test the candidate solution. More specifically, the algorithm accepts the candidate solution when the solution of the local quadratic problem satisfies the first-order-necessary conditions (see Eq. (3.5)), and Second-Order Necessary Conditions or satisfies only the Second-Order Sufficient Conditions. The sequential quadratic program solves problems in a sequence to construct an ideal step. However, finding the ideal step is too arbitrary. The goal is to find a step that satisfies the optimality conditions, which have many forms. This is why this section describes multiple-step methods. The main step methods are the dogleg step method for the trust-region framework and the projected conjugate gradient method for the line-search and the trust-region framework. Both frameworks incorporate the least square step method to find the Lagrange multiplier in 3.5.8 and second-order-correction step in 3.5.11. Section 3.5.9 describes the penalty updating strategy that puts more weight on finding the constraint step method.

3.5.8 Lagrange Multiplier

In optimization, two methods exist for identifying the Lagrange multiplier. Procedures can derive the parameter directly, or a routine may estimate the Lagrange multiplier. Each method has its advantages and disadvantages. The trim module uses an estimation of the Lagrange multiplier using a least-squares approximation. The literature commonly refers to the least-squares estimation of the Lagrange multiplier as the least-squares multiplier. The Least-squares multiplier approximates the Lagrange multiplier equation using the smallest euclidean norm.

$$\hat{\lambda}_{k+1} = (A_k A_k^T)^{-1} A_k \nabla f_k, \tag{3.30a}$$

$$\min_{\lambda} \ ||\nabla_x \mathcal{L}(x_k, \lambda)||_2^2 \ = \ ||\nabla f_k - A_k^T \lambda||_2^2. \tag{3.30b}$$

here, A_k is the Jacobian matrix of the constraints and has full row rank. Equation (3.30a) calculates the least-squares multiplier that satisfies the first-order optimality condition.

3.5.9 Penalty Updating Strategy

One method of minimizing the constraint violation is introducing a penalty parameter. The penalty parameter artificially increases the magnitude of the constraint violation. This makes the algorithm effective at avoiding the infeasible search space, as it will result in a significant increase in objective value. Following this analogy too literally might suggest that starting with a large penalty function might be favorable. Figure 7a illustrates minimizing both objective and constraint functions simultaneously may be more advantageous than minimizing first the constraint than the objective separately. Working with a large penalty value may result in an ill-defined problem, as small changes in constraint violation may result in a large perturbation of the solution. An adaptive penalty function is therefore a more effective strategy. Equation (3.31) formulates a quadratic model that agrees with both trust-region and line search for computing an adaptive penalty value.

$$\mu \geq \frac{\nabla f_k^T p_k + (\sigma/2) p_k^T \nabla_{xx}^2 \mathcal{L}_k p_k}{(1-\rho)||c_k||_1} \qquad \sigma = \begin{cases} 1 & \text{if } p_k^T \nabla_{xx}^2 \mathcal{L}_k p_k > 0, \\ 0 & \text{Otherwise} \end{cases} \tag{3.31}$$

If the step direction agrees with the secondary sufficient condition, a quadratic model computes the penalty value. Otherwise, a linear model computes the penalty value [38](p. 543).

3.5.10 Projected Conjugate Gradient Method

The projected conjugate gradient method is a robust type of step-generating algorithm. This approach implicitly computes a step direction using an orthogonal null-space matrix Z. Ill-conditioning of A or poor estimations of its null-space Z do not affect this method [38]. The main difference between a conventional conjugate gradient method and the projected CG method lay in the usage of the residual. Both approaches start solving the model (3.15) resulting in some residual vector r. The conjugate gradient approach iteratively minimizes the negative residual along a conjugate search direction. This search direction is unconstrained and may result in a constraint violation. The projected conjugate gradient approach uses a projection matrix to project the residual into the null-space of the constraints, then minimizes along this projected conjugate search direction.

Algorithm 1: Step method: Projected Conjugate Gradient

Result: Set computation of p_k that minimizes subproblem _ in the nullspace of the constraints

Choose $H = \{I$ or diag$(|G_{ii}|)$ or G or other choice$\}$
Calculate the positive semi-definite matrix $W_{zz} = Z^T H Z$
Solve $Ax = b$ and find initial point x ;
Compute $r = Hx + c$, find $\vec{y}$ that minimizes $||r - A^T y||_{G^{-1}}$ than set $r \leftarrow r - A^T y$;
Compute $P = Z(W_{zz})^{-1} Z^T$, $g = Pr$, and set $p = -g$
Repeat until convergence test is satisfied
while $\epsilon > |r^T g - r^T P r|$ **do**
    $\alpha \leftarrow r^T g / p^T H p$
    $x \leftarrow x + \alpha p$
    $r^+ \leftarrow r + \alpha H p$
    $g^+ \leftarrow P r^+$
    $\beta \leftarrow (r^+)^T g^+ / r^T g$
    $p \leftarrow -g^+ + \beta p$
    $g \leftarrow g^+$ and find y that satisfies $||r^+ - A^T y||_{G^{-1}}$
    $r \leftarrow r^+ - A^T y$
end

3.5.11 Second-Order Correction

Executing the PCG and dogleg routine solves the local model described in Eq. (3.15) giving the local minimization step p_k. The next line search $x_{k+1} = x_k + \alpha p_k$ or for trust-region $x_{k+1} = x_k + p_k$ iteration point is checked by its acceptation criteria and accepted if satisfied. The acceptation criteria per iteration method are explained in section 3.7 and 3.6 respectively. Updating the active set in an iterative process for finding the solution to the subproblem might be expensive for large or complex problems. Here, the active set is determined at the current point, and a solution to this current point is found using the approximated model. If an increase in the norm of the constraints is detected, a secondary step correction is executed to avoid the Maratos effect, as explained in Nocedal and Wright [38] chapter 15.5.

$$\hat{p}_k = -A_k^T (A_k A_k^T)^{-1} c(x_k + p_k). \tag{3.32}$$

3.6 SQP Trust-Region

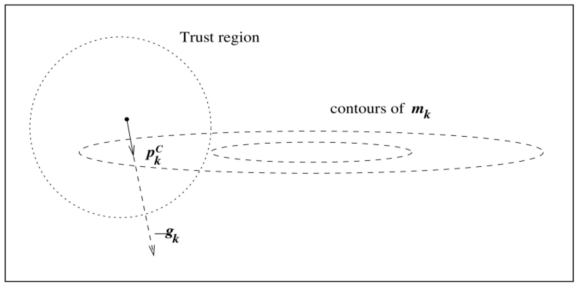
Starting from some iteration point x_k, the trust-region algorithm trusts its quadratic model up to some distance Δ_k. The trust-region iteration step length $||p_k||$ may therefore only search for an optimum that is not larger than the trust-region radius $||\Delta_k||$. This not only limits the step length p, but also alters the step direction of p. The additional step length constraint that the trust-region method adds to the auxiliary subproblem (3.15) is $||p||_2 \leq \Delta_k$ forming the subproblem below.

$$\min_v \quad ||A_k v + c_k||_2^2 \tag{3.33a}$$

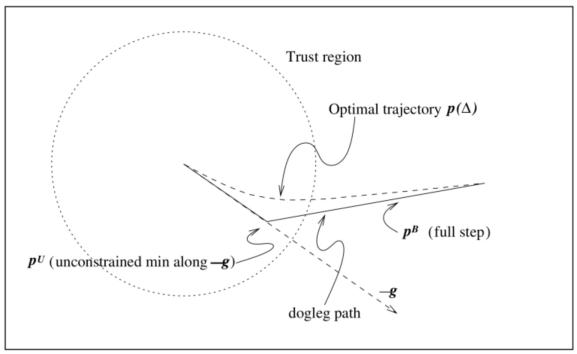
$$\text{subjected to} \quad ||p||_2 \leq 0.8\Delta_k \tag{3.33b}$$

3.6.1 Cauchy Point Methods

The Cauchy point is a steepest descent-based method and computes a step direction that contains sufficient reduction. The Cauchy point method can find global minima, but it performs poorly even if it computes an ideal step length. In Figure 10a illustrates the Cauchy point step direction p_k^c. The Cauchy point calculation is based on a first-order approximation, meaning it is a steepest descent-based search method. These kinds of search methods can find global minima but perform poorly even if the ideal step length is calculated. Variations of these methods have been developed over the years, and some are described in this section.



(a) Cauchy point method [38]



(b) Dogleg approximation vs optimal trajectory [38]

Figure 9: Acceptable intervals visualized where desired values for step length α satisfy the inequalities and a combined procedure define the fundamental Wolfe condition [27, 38]

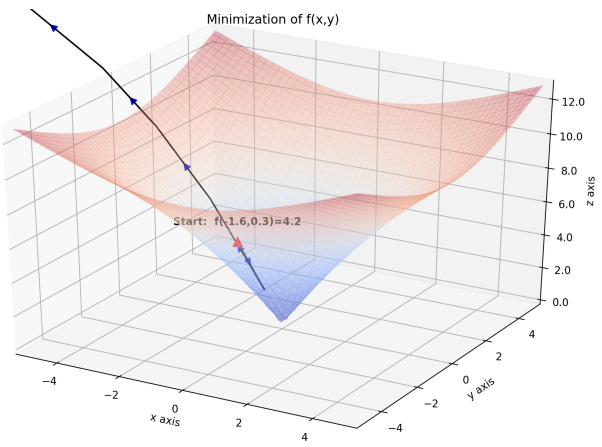
The Dogleg Method

Figure 10b visualizes a different approach based on the Cauchy point method. The method here is called the dogleg method, named after its distinctive shape. The most appropriate application is a Newton-dogleg method that searches for an optimum step of a convex objective function. Other problems that consist of an indefinite Hessian are solvable using the dogleg method, but applying a dogleg method makes not much point. The full step p^B that the dogleg method uses is not the unconstrained minimizer of the model. Methods that are more capable to cope with such problems are two-dimensional subspace minimization [38].

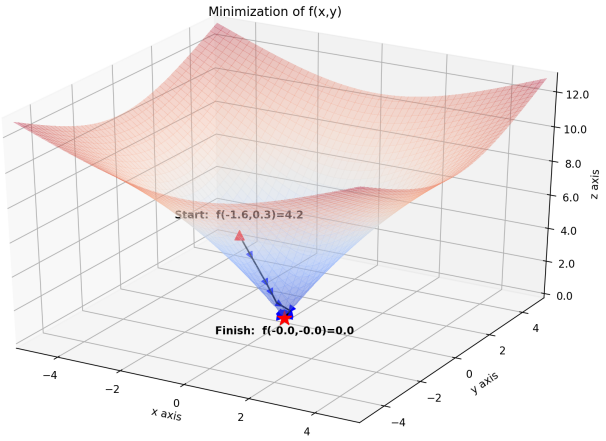
For this reason, the procedure switches from dogleg to two-dimensional subspace method when the Second-Order-Sufficient conditions don't hold. Two-dimensional subspace minimization executes when the B is indefinite but requires an estimate of the most negative eigenvalue of B. If the eigenvalues of B are negative, the following subspace for (3.34).

$$\text{span}[g, (B + \alpha I)^{-1} g], \qquad \text{for some } \alpha \in (-\lambda_l, -2\lambda_l] \tag{3.34}$$

Here, λ_l defines the most negative eigenvalues of B and α is used to make the B positive definite. The flexible interval allows numerical procedures, like the Lanczos method, to find a value for α.



(a) Dogleg method using an indefinite matrix



(b) Dogleg method switching to the two-dimensional subspace method when the Hessian becomes indefinite

Figure 10: Finding the minimum of the unconstrained optimization problem: $f(x, y) = 20(1 - e^{-0.2\sqrt{0.5(x^2+y^2)}})$

The minimization scheme of the characteristic subproblem (3.33a) depends on the outcome of the second-order-sufficient condition. If the second-order-sufficient conditions don't hold, the scheme searches for a

solution in a two-dimensional subspace according to Eq. (3.34). When the condition holds, the dogleg solves the below given sequence to obtain p^U and p^B.

$$p^U = -\frac{g^T g}{g^T B g} g \tag{3.35}$$

$$p^B = -Hg \tag{3.36}$$

Assembling the dogleg trajectory uses the combination of the two steps p^U and p^B according to the magnitude of the parameter τ. The optimal magnitude of τ defines the following step problem as a function of τ formulating $p(\tau)$. The function $p(\tau)$ describes the following condition:

$$p(\tau) = \begin{cases} \tau p^U, & 0 \leq \tau \leq 1, \\ p^U + (\tau - 1)(p^B - p^U), & 1 \leq \tau \leq 2. \end{cases} \tag{3.37}$$

When the trust-region uses the ball-shaped trust-region constraint, the quadratic condition below must hold. Solving the quadratic equation derives the parameter τ which concludes the dogleg.

$$||p^U + (\tau - 1)(p^B - p^U)||^2 = \Delta^2 \tag{3.38}$$

Besides the ball-shape constraint, the box-shape trust-region is a popular choice. The box-shape limits the length of the elements in a vector instead of limiting the length of the vector.

Trust-Region Subproblem
The SQP trust-region algorithm has an additional trust-radius constraint imposed on the problem. Satisfying both the set of constraints imposed on the problem and the trust-region radius constraint is not always possible. Figure 8a illustrates the complications that arrive from having both constraints active, and the solution the relaxation strategy offers. The relaxation method artificially reduces the constraint violation, avoiding inconsistent constraints. Besides minimizing the relaxed constraint set, a sequential subproblem solver can minimize the objective while considering the trust-region constraint. Figure 8b illustrates this process. The relaxation vector r_k is the smallest residual for the below-given subproblem:

$$\min_v \quad ||A_k v + c_k||_2^2 \tag{3.39a}$$

$$\text{subjected to} \quad ||v||_2 \leq 0.8\Delta_k \tag{3.39b}$$

$$r_k = A_k v_k + c_k \tag{3.39c}$$

The subproblem is solved by selecting the smallest Euclidean norm that is found by the least-squares approximation. The parameter vector v_k is used as starting point for the Dogleg and Projected CG method. After calculating the initial step, the local quadratic model given by Eq. (3.40) solved iteratively.

$$q_\mu = f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L} p + \mu ||c_k + A_k p||_2 \tag{3.40}$$

Solving the quadratic model q_μ results in a step that is checked to ensure if the local quadratic model provides a satisfactory accuracy. The merit function used by this algorithm is given by Eq. (3.41).

$$\varphi_2(x, \mu) = f(x) + \mu ||c(x)||_2 \tag{3.41}$$

Here, $\varphi_2(\cdot)$ contains the quadratic sum of the dynamic derivative x_d represented by $f(x)$ and $||c(x)||_2$ computes the norm of the constraints. The accuracy of the local function is tested by Eq. (3.42).

$$\rho = \frac{ared_k}{pred_k} = \frac{\varphi_2(x_k, \mu) - \varphi_2(x_k + p_k, \mu)}{q_\mu(0) - q_\mu(p_k)} \tag{3.42}$$

Here, ρ quantifies the agreement between the actual reduction $ared_k$ and the predicted reduction $pred_k$. If ρ is small or negative, the local model is of poor quality, resulting in a reduction of the trust-region. A poor local model may cause a rejection of the iteration step. A ratio of $\rho = 1$ quantifies a good agreement between the actual and predicted reduction. A good agreement may lead to an increase in trust-region radius.

3.7 SQP Line Search

Line search method first obtain a step direction p_k by executing a program that minimizes a local model. The algorithm then determines how far to step in this search direction by solving the scalar length α. If the set of conditions agree at some candidate point $x_k + \alpha p_k$, then the algorithms accepts the candidate point, and the iteration point $x_k + \alpha p_k$ becomes the next iteration point x_{k+1}. Equation (3.43) describes this procedure mathematically

$$x_{k+1} = x_k + \alpha_k p_k \tag{3.43}$$

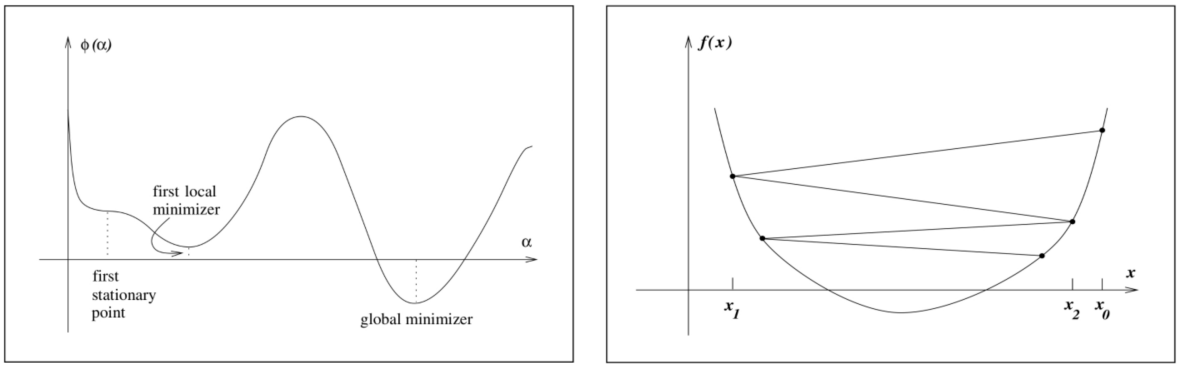
The parameter α_k is a positive value and defined as the step length $\alpha_k \in (0, 1]$. A line search method success depends on the step length α and search direction p_k. Both parameters can be found using multiple methods, and dependent on the application, one method is more effective than the other. Most line search minimization schemes contain conditions that only accept the step αp_k if a new point $x_k + \alpha p_k$ is a descending direction such that $f(x_k + \alpha p_k) < f(x_k)$. In a way, the parameters p_k and α_k contribute a major part to the success of the line search methods. Considering this knowledge in line with the methods of previous sections, the upcoming part presents conditions that identify acceptation criteria for a candidate point $x_k + \alpha p_k$.

3.7.1 Step Length

The goal is to find a step length α along the descending direction p_k that reduces f substantially. However, identifying the quantity α that results in significant reduction may require too much time. Defining the computation process that finds an adequate step length α_k results in a trade-off between ideal step length and computation time. Effective methods for finding the step length do not necessarily find a step length near the minimizer of some model, but rather focus on a step length that delivers an adequate reduction of f. Meaning the line search algorithm searches for some α that satisfies:

$$f(x_k + \alpha_k p_k) < f(x_k) \qquad \alpha \in (0, 1]. \tag{3.44}$$

This statement alone does not confine the problem enough such that it promotes efficient convergence to the minimizer of f at some point x^*. This statement only enforces a reduction constraint on the algorithm. Many choices for α exist and only considering the condition (3.44) as figure 11a illustrates. These figures show the many possible choices of α for some model $\phi(\alpha)$ the next iteration point $x_k + \alpha p_k$. Again, as figure 11a illustrates, many points exist where the condition (3.44) is true. Only considering condition (3.44) may lead to many iterations resulting in slow convergence, as figure 11b demonstrates.



(a) The ideal length is the global minimizer [38]

(b) Insufficient reduction of f [38]

Figure 11: Step length identification and minimization using line search algorithms

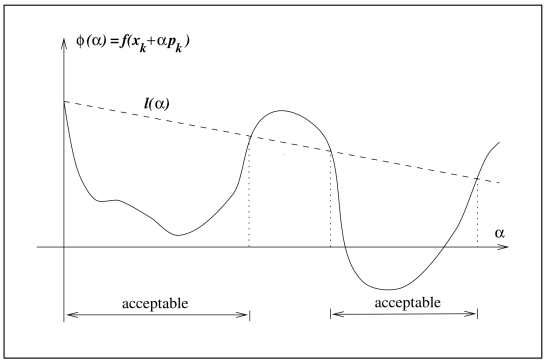
The Armijo Condition

The Armijo condition describes a statement that accepts the step length α if the step αp_k results in sufficient reduction of f. The condition accepts the step length α, if the candidate objective value $f(x_k + _k)$ is less or equal than a reduction prediction. The inequality below defines the Armijo condition

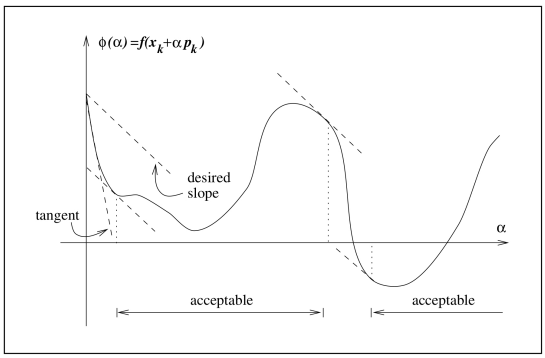
$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f_k^T p_k. \tag{3.45}$$

From inspecting the above presented inequality, it is clear that this condition uses local information of f. The condition restricts possible choices of α using the weight c_1 and directional derivative $\nabla f_k^T p_k$.

The search directional p_k along the gradient ∇f_k predicts descent of one point. Involving all possible choices α creates a line from starting point $\alpha = 0$ up to $\alpha = 1$. The weight c_1 controls the slope of the line $\alpha \nabla f_k^T p_k$. Using a linear form $\phi(\alpha) = f(x_k + \alpha p_k)$ and $l(\alpha) = f(x_k) + c_1 \alpha \nabla f_k^T p_k$, acceptable α exist when $\phi(\alpha) \leq l(\alpha)$. Figure 12a reveals how the Armijo condition affects the set of acceptable choices of α. According to Nocedal and Wright [38], typical values for c_1 are small which may approximately be $c_1 = 10^{-4}$ [38].



(a) regions for $l(\alpha)$ that satisfy Eq. (3.46) [38]



(b) regions for $l(\alpha)$ that satisfy Eq. (3.46) [38]

Figure 12: Acceptable intervals visualized where desired values for step length α satisfy the inequalities and a combined procedure define the fundamental Wolfe condition [27, 38]

The Armijo condition, that Eq. (3.46) describes, formulates a usable condition for unconstrained minimization methods. In essence, the armijo condition describes sufficient reduction of the next iteration point. The algorithm that this research employs uses the following Armijo condition:

$$\phi(x_k + \alpha p_k; \mu_k) \leq \phi(x_k; \mu_k) + c_1 \alpha \mathcal{D}(\phi(x_k; \mu_k)). \tag{3.46}$$

Here, $\mathcal{D}$ is the directional derivative of $\phi(x_k; \mu_k)$. For more information regarding the directional derivative, see *Numerical optimization* theorem 18.2 page 541 and page 628 [38].

$$\varphi_1(x, \mu) = f(x) + \mu ||c(x)||_1 \tag{3.47}$$

$$\mathcal{D}(\varphi_1(x, \mu)) = \nabla f^T(x) p - \mu ||c(x)||_1 \tag{3.48}$$

Curvature Condition
To promote a more effective scheme, another inequality is introduced that evaluates a gradient-based condition. A curvature condition is defined by the following inequality

$$\nabla f(x_k + \alpha p_k)^T p_k \geq c_2 \nabla f_k^T p_k \tag{3.49}$$

Where c_2 is a constant between $c_2 \in (c_1, 1)$ and according to [27, 38] this value is in practical application commonly set to $c_2 = 0.9$ when p_k is based on a (quasi)-Newton methods or $c_2 = 0.1$ for a nonlinear conjugate gradient method. This inequality is satisfied when the directional derivative of the next iteration is larger than the current directional derivative. This seems strange because it might indicate that finding a minimization suggests searching for a steeper slope as where the slope at x^* is flat. Because of the negative directional derivative $\nabla f_k^T p_k < 0$, larger states a more positive directional derivative. A sequence of less negative directional derivatives results in finding a local minimizer, global minimizer, or stationary. When this inequality can no longer be satisfied, the line search terminates, indicating one of these points.

Wolfe Condition
Using both inequalities (3.46) and (3.49) defines the wolfe conditions and is a effective approach when searching for a candidate step length α. The Wolfe Condition regarding $\phi(x_k; \mu)$ becomes

$$\phi(x_k + \alpha p_k; \mu_k) \leq \phi(x_k; \mu_k) + c_1 \alpha \mathcal{D}\phi(x_k; \mu_k) \tag{3.50a}$$

$$\mathcal{D}(\phi(x_k + \alpha p_k; \mu_k)) \geq c_2 \alpha \mathcal{D}(\phi(x_k; \mu_k)). \tag{3.50b}$$

where $(0 \leq c_1 \leq c_2 \leq 1)$. As can be observed in 15, the interval of choosing a *alpha* candidate still allows some movement across a minimizes or stationary point of $\phi(\alpha)$ because larger slopes are accepted. The set of inequalities known as the strong Wolfe condition restricts this behavior by imposing a change in the gradient inequality, which states:

$$\phi(x_k + \alpha p_k; \mu_k) \leq \phi(x_k; \mu_k) + c_1 \alpha \mathcal{D}\phi(x_k; \mu_k) \tag{3.51a}$$

$$|\mathcal{D}(\phi(x_k + \alpha p_k; \mu_k))| \geq c_2 \alpha \mathcal{D}(\phi(x_k; \mu_k)). \tag{3.51b}$$

Backtracking
A method that are proven competitive uses besides the Armijo inequality (3.46) without the curvature inequality (3.49). By introducing a new parameter $\rho \in (0, 1)$, the search direction is made more adaptive. An initial trial step length $\alpha = \hat{\alpha} > 0$ is executed where condition (3.46) is tested. If the condition is false, a procedure changes the step length according to $\alpha = \alpha\rho$ which makes the step smaller and tests the condition again until the condition is true. A typical backtracking algorithm takes the form:

Algorithm 2: Backtracking [2, 27, 38]

Set $\alpha_k \leftarrow 1$, $\rho \in (0, 1)$, $c_1 \in (0, 1)$
while $\phi(x_k + \alpha_k p_k; \mu_k) > \phi(x_k; \mu_k) + c_1 \alpha \mathcal{D}\phi(x_k; \mu_k)$ **do**
  $\alpha_k \leftarrow \alpha_k \rho$
end
Terminate with α_k

The used variant is a modified variant compared to [38]

Line Search Acceptance Condition
In contrast to the trust-region algorithm that makes its step decision based on first and second derivative information guarded by its trust-region, line search only checks for sufficient reduction based on the first derivative. Without including a filter, secondary step correction, Lagrangian strategy, or large enough penalty value, the algorithm may find a solution in the infeasible region. A cost value at this point may be more attractive with additional constraint cost than a feasible point without a penalty. Because the secondary correction step is less accurate for large steps and the Lagrangian is approximated, an additional constraint is employed based on observations. Nocedal and Wright [38] on page 545 presents the pseudocode for a SQP line search algorithm. It is suggested that the penalty parameter μ is chosen such that inequality (3.31) holds. Far from the solution with the parameter μ working with inequalities constraint converted to equality constraints valid for some reasonable distance from the current point, a line search could make such a large step that is of the actual objective function outside the feasible region becomes attractive. Normally, the Armijo condition is appended with a condition that demands some (absolute) gradient reduction defined by the (strong) Wolfe condition, which may prevent this from happening. Because this scheme uses finite differences to calculate the gradient including this condition for every backtracking value of α, is very expensive. Executing a second-order correction step to prevent this from happening was not recommended, as it is only valid for a small step near the solution. Based on parameters already compute during the evaluation of the Armijo condition, the Armijo condition was appended with an additional condition that allows change in the constraint value $c(x_k + \alpha p_k)$ resulting in $\theta||c(x_k + \alpha p_k)|| > ||c(x_k)||$. This hard constraint is appended with a softening in the Armijo condition, resulting in $f(x_k + \alpha p_k) \leq f(x_k) + \mu|c(x_k)| + \eta\alpha(\nabla f_k^T p_k - \mu * |c(x_k)|)$ defining the new condition to be

$$\begin{gathered}\theta||c(x_k + \alpha p_k)|| > ||c(x_k)|| \text{ and } f(x_k + \alpha p_k) \leq f(x_k)(1/\theta)\mu|c(x_k)| + \eta\alpha(\nabla f_k^T p_k - \mu * |c(x_k)|) \\ \theta \in (0, 1).\end{gathered} \tag{3.52}$$

This condition is implemented in all results provided in chapter 4.

3.8 Aircraft Trim Specifications

The objective function that describes the aircraft trim problem minimizes the squared sum of the dynamic state derivatives and its constraints. Here, only the objective function is without its constraints is

considered. Equation (3.53) describes the objective mathematically.

$$f = \dot{x}_d^T W \dot{x}_d \tag{3.53}$$

Where f describes the cost function, the set of states $\dot{x}_d$ describes all dynamic state derivatives, and the matrix W is a diagonal non-negatief weight matrix. For now, let's consider the aircraft dynamic state derivatives $\dot{x}_d = [\dot{V}, \dot{\alpha}, \dot{\beta}, \dot{p}, \dot{q}, \dot{r}]$ first. The function is rather simple, but varies per aircraft trim problem and maneuver. The authors McFarland [34] explain that the weight must be according to the magnitude of influence they have [34]. The body acceleration units they use are ft/s^2 and rad/s^2 with the weights on the diagonal $W = [1, 0.1, 0.1, 0.1, 0.1, 0.1]$ respectively (page 9, Eq. 15). The work of Van der Linden [45] uses the diagonal weights $W = [5, 5, 5, 1, 2, 10]$ regarding the body acceleration units m/s^2, rad/s^2, rad/s [45]. The author Stevens, Lewis, and Johnson [44] illustrates on page 191 a 6-DOF F16 model that uses weights $[1, 100, 100, 10, 10, 10]$ for ft/s^2, rad/s^2, rad/s. The authors state that the weights are uncritical, which makes sense as they can reduce the cost function to 10^{-10} or lower. However, this reports finds that for less significant trim accuracy, scaling may be critical. The work of Friedman and Rand [18] describes that the stopping criteria of $\sqrt{\dot{u}^2 + \dot{v}^2 + \dot{w}^2} < 0.001\ m/s^2$ and $\sqrt{\dot{p}^2 + \dot{q}^2 + \dot{r}^2} < 0.001\ rad/s^2$ guarantees adequate trim for rotorcraft configurations. Because this research searches a solutions that are steady-state enough such that a pilot can take over manual control, these bounds ($\sqrt{\dot{u}^2 + \dot{v}^2 + \dot{w}^2} < 0.001\ m/s^2$ and $\sqrt{\dot{p}^2 + \dot{q}^2 + \dot{r}^2} < 0.001\ rad/s^2$) are considered by this study.

3.9 Concluding SQP-framework

Combining all theory provided in this chapter results in a pseudocode regarding the trust-region and line search method. In the following, a summary briefly describes the role and placing of each theoretical subpart, after which the pseudocodes are introduced.

The optimality conditions verify solutions on several levels in the algorithm, as it checks if a candidate point indeed is a local solution. In other words, do the candidate settings solve the aircraft trim problem. Secondly, it checks if all solutions of the local models, problems, and subproblems are valid for the sake of computing steps.

3.9.1 Trust-Region Pseudocode

Algorithm 3: Sequential quadratic programming: Trust-region Pseudo-code

Result: The steady-state parameter set x^* that satisfy constraint set c

Set: $\epsilon > 0$, $\eta \in (0, 0.5)$ and $\gamma \in (0, 1)$;
Choose starting point x_0, initialize active set $\mathcal{A}_0$ and set Hessian approximation $B_0 \leftarrow I$;
Calculate c_0, f_0, A_0, ∇f_0 and approximate $\hat{\lambda}_0$ by using equation 3.30a ;
for *k = 1, 2, ...* **do**
    if $||f_k - A_k^T \hat{\lambda}_k||_\infty > \epsilon$ ***and*** $||c_k||_\infty > \epsilon$ **then**
        Stop with approximate solution x_k ;
    end
    Calculate p_k by projected CG or dogleg method ;
    Choose μ_k to satisfy 3.31 ;
    if $||c_{k+1}|| > ||c_k||$ ***and*** $||f_{k+1}|| > ||f_k||$ **then**
        Invoke secondary step correction $\hat{p}_k$ by 3.32 ;
        Set $p_k \leftarrow p_k + \hat{p}_k$;
    end
    Evaluate $\nabla f(x_k + p_k)$;
    Run trust-region updating algorithm to obtain Δ_{k+1} ;
    if $\rho_k > \eta$ **then**
        Set $x_{k+1} \leftarrow x_k + p_k$;
    else
        Set $x_{k+1} \leftarrow x_k$;
    end
    Set $\mathcal{A}_{k+1}$ then evaluate $c_{k+1} \leftarrow c(x_{k+1})$, $A_{k+1} \leftarrow A(x_{k+1})$,
    and approximate $\hat{\lambda}_{k+1}$ by using equation 3.30a ;
    Set $s_k \leftarrow p_k$ and evaluate $y_k \leftarrow \nabla \mathcal{L}(x_k + p_k, \hat{\lambda}_{k+1}) - \nabla \mathcal{L}(x_k, \hat{\lambda}_{k+1})$
    to obtain B_{k+1} by updating B_k using a quasi-Newton formula ;
    Set $\nabla f_{k+1} \leftarrow \nabla f(x_{k+1})$, $\nabla \mathcal{L}_{k+1} \leftarrow \nabla \mathcal{L}(x_k + p_k, \hat{\lambda}_{k+1})$;
end

3.9.2 Line Search Pseudocode

Algorithm 4: Sequential quadratic programming: Line-search Pseudo-code

Result: The steady-state parameter set x^* that satisfy constraint set c

Set: $\epsilon > 0$, $\eta \in (0, 0.5)$, $\theta \in [0.9, 1]$ and $\gamma \in (0, 1)$;
Choose starting point x_0, initialize active set $\mathcal{A}_0$ and set Hessian approximation $B_0 \leftarrow I$;
Calculate c_0, f_0, A_0, ∇f_0 and approximate $\hat{\lambda}_0$ by using equation 3.30a ;
for *k = 1, 2, ...* **do**
    if $||f_k - A_k^T \hat{\lambda}_k||_\infty > \epsilon$ ***and*** $||c_k||_\infty > \epsilon$ **then**
        Stop with approximate solution x_k ;
    end
    Calculate p_k by projected conjugate gradient method (or any other) ;
    Choose μ_k to satisfy 3.31 ;
    while $\theta c(x_k + \alpha p_k) > c_k$ ***and*** $f(x_k + \alpha p_k) > f_k + \eta\alpha(\nabla f_k^T p_k - \mu c_k)$ **do**
        Set: $\alpha \leftarrow \gamma\alpha$;
    end
    if $||c(x_k + \alpha p_k)|| > ||c_k||$ ***and*** $||f(x_k + \alpha p_k)|| > ||f_k||$ **then**
        Invoke secondary step correction $\hat{p}_k$ by 3.32 ;
        Set $p_k \leftarrow \alpha p_k + \hat{p}_k$;
    else
        Set $p_k \leftarrow \alpha p_k$;
    end
    Set $x_{k+1} \leftarrow x_k + p_k$ and $\alpha \leftarrow 1$;
    Set $\mathcal{A}_{k+1}$ then Evaluate c_{k+1}, f_{k+1}, A_{k+1}, ∇f_{k+1}
    and approximate $\hat{\lambda}_{k+1}$ by using equation 3.30a ;
    Set $s_k \leftarrow p_k$ and evaluate $y_k \leftarrow \nabla\mathcal{L}(x_k + p_k, \hat{\lambda}_{k+1}) - \nabla\mathcal{L}(x_k, \hat{\lambda}_{k+1})$;
    Obtain B_{k+1} by updating B_k using a quasi-Newton formula ;
end

3.10 Implementation and Verification Process

Implementing the mathematics requires some C++ template library able to execute vectors, matrices, numerical solvers, linear algebra operations, and more advanced relevant algorithms. The EIGEN template library satisfies the implementation needs for this thesis. Other libraries may suit better, but finding the optimal library is outside of the scope of this project. Many numerical optimization projects use the EIGEN library and functionality. Advanced numerical computation extensions implement Eigen and make use of their open-source policy. Relevant extensions that use Eigen are IFOPT [48], CppNumericalSolvers [47] referred to by Nocedal and Wright [38], Google's Ceres [4], and many others see [20].

Re-implementing existing parts of existing libraries was considered but was found not to be a practical solution. The trust-region and line-search pseudocodes in section 3.5 are straightforward to implement. The algorithm only requires functions that EIGEN supports, and doesn't require parts of existing libraries. Using parts of existing libraries will most definitely increase the performance of the algorithm. However, it will increase the complexity or make it impossible to verify the result at each development stage. During the development of the trim module, multiple platforms were involved. The development process set implementation targets involving the development platforms: Python, DUECA and Matlab Simulink.

The first stage of development constructs a trust-region and line-search minimization implementation in Python. Testing and verifying the working principle and outcome of the basic implementation uses literature findings. More specifically, the python script minimizes the Rosenbrock problem and compares iteration steps and results to the findings of Nocedal and Wright [38]. When iteration steps and minimization results agree with the literature, the next development stage starts.

The second stage constructs the same basic minimization scheme in DUECA. When the results agree with the literature findings and Python script, the next development stage starts. The next development stage adds a function that reassembles basic helicopter dynamics in python and DUECA. The python script and DUECA implementation both minimize the helicopter dynamics, targeting the straight and level flight condition. A Matlab Simulink and DUECA implementation of the helicopter model verifies the straight and level flight initial condition.

The third stage detaches the dependencies of all problems in DUECA making the basic trim module generic capable of executing through a Graphic User Interface (GUI). The basic generic trim module again finds the straight and level flight condition and compares the result to previous steady-state results.

The fourth stage upgrades the basic trim module to a SQP trim module and verifies the results. In the final development stage, the independent generic trim module connects to the DASMAT aircraft model and solves the aircraft trim problem in multiple steady-state conditions selectable through a GUI.

The techniques of interest that Eigen provides are the dense (matrix) decompositions. Eigen provides an overview of benchmark results[1] of Eigen-specific generic linear algebra decomposition information[2]. The least-squares multiplier Eq. (3.30a) and the secondary correction step correction both use the least-square operation that the Eigen library provides. All functions that the algorithm uses influence the robustness, efficiency, and accuracy of the algorithm. The choice of Eigen decomposition must stay in line with these performance indicators, therefore the choice must be first the most robust, then the most efficient, then the most accurate. To achieve sufficient this, two choices influence the performance indicators. The form of the least-squares equation that a decomposition solves, and the type of decomposition that executes the least-squares operation. The first consideration is manipulating the form of the linear system. Manipulating the form of the linear system $Ax = b$ computes the problem by solving the normal equation $A^TAx = A^Tb$. This form reduces the problem size when the matrix $A^{m\times n}$, has more rows than columns $m > n$, giving a more efficient but less stable and accurate result. Eigen provides the option of computing x directly by using the least-squares solver. This procedure solves a larger linear system, hence is less efficient but has provided more stability and accuracy. Eigen describes direct solving the linear system using least-squares as the preferred method. Eigen argues that an ill-conditioned matrix affects the normal equation method as A^TA results in a squared condition number and overall loses twice as many digits compared to direct least-squares or other methods[3]. The direct least-squares approach provides the most control regarding the performance parameters, making it for this study the favorite choice for small to medium problems. Eigen suggests several decomposition methods capable of solving the least-squares problem. The compositions relevant for this research are: the Householder and Single Value Decomposition (SVD). The HouseHolder decomposition provides three classes. These classes from least to most reliable, time-consuming, and accurate are the HouseholderQR, ColPivHouseHolderQR, and FullPivHouseHolderQR. According to Eigen[4], the slowest but most reliable and accurate direct least-squares solving is the Singular Value Decomposition *MatrixBase::bdcSvd()*. The current implementation uses the HouseholderQR far from the solution, ColPivHouseHolderQR near the solution, and FullPivHouseHolderQR close to the solution.

[1] https://eigen.tuxfamily.org/dox/group__DenseDecompositionBenchmark.html
[2] https://eigen.tuxfamily.org/dox/group__TopicLinearAlgebraDecompositions.html
[3] https://eigen.tuxfamily.org/dox/group__LeastSquares.html
[4] https://eigen.tuxfamily.org/dox/group__LeastSquares.html

4 Benchmark Problems

In this section, the algorithm is tested on well-known benchmark problems presented in section 4.1 and 4.6 to assess the performance of the proposed methods on different types of problems. This process will define the relative performance of each algorithm. The following list provides the features that make up the upcoming algorithms:

- Iteration methods available are Trust-region (T) and Line-search (L);
- First-order derivatives are determined using Central differences (C) and forwards Euler (E);
- Second-order derivatives are approximated using the Quasi-Newton damped-BFGS (B) or SR1 formulas (S);
- Step methods are calculated by first-order, second-order Projected conjugate gradient (P), or dogleg method. The Dogleg method (D) is only available for trust-region iteration methods, and the first-order projected conjugate gradient can only be used with the line-search iteration method.

An algorithm based on trust-region (T), Central difference (C), damped-BFGS (B), and second-order Projected conjugate gradient method (P) is abbreviated to TCBP. The first-order projected conjugate gradient method requires no Hessian approximate, meaning (B) or (S) is not mentioned in the abbreviation. The candidate optimization algorithms for solving the benchmark problems are expressed by their abbreviations below.

Algorithm number	abbre-viation	iteration framework	First-order derivative	Second-order derivative	Step method
1:	TCBP	Trust-region	Central difference	Damped-BFGS	Projected conjugate gradient
2:	TCBD	Trust-region	Central difference	Damped-BFGS	Dogleg
3:	TCSP	Trust-region	Central difference	SR1	Projected conjugate gradient
4:	TCSD	Trust-region	Central difference	SR1	Dogleg
5:	TEBP	Trust-region	Euler forward	Damped-BFGS	Projected conjugate gradient
6:	TEBD	Trust-region	Euler forward	Damped-BFGS	Dogleg
7:	TESP	Trust-region	Euler forward	SR1	Projected conjugate gradient
8:	TESD	Trust-region	Euler forward	SR1	Dogleg
9:	LCBP	Line search	Central difference	Damped-BFGS	Projected conjugate gradient
10:	LCSP	Line search	Central difference	SR1	Projected conjugate gradient
11:	LCP	Line search	Central difference		Projected conjugate gradient
12:	LEBP	Line search	Euler forward	Demped BFGS	Projected conjugate gradient
13:	LESP	Line search	Euler forward	SR1	Projected conjugate gradient
14:	LEP	Line search	Euler forward		Projected conjugate gradient

Table 5: SQP Algorithms

Section 4.1 explains what benchmark functions are applied to test the performance of each proposed algorithm. Section 4.3 elaborates on the hardware and software setup during the experiments. Section 4.4 describes the algorithm's safeguards necessary for functioning and the limitations they may bring. Section 4.5 show how different algorithm features which table 5 abbreviates behave on the popular benchmark selection. Section 4.6 shows the performance of different algorithm features on multiple steady-state flight conditions. Section 4.2 formulates the aircraft trim solution settings for the DASMAT trim problem.

4.1 Benchmark Problem Selection

The complexity of the selected benchmark problems must match the complexity of the generic trim problem. Categorization of benchmark problems are organized and explained by Jamil and Yang [28], a C++ test library is presented by Adorio [3]. Tested algorithms and results are gathered in the footnote[5]

[5] http://infinity77.net/global_optimization/test_functions.html#test-functions-index
http://www5.zzu.edu.cn/ecilab/Benchmark.htm
https://www.sfu.ca/~ssurjano/index.html

for some specialized algorithms. During this research, it is observed that the majority of the aircraft simulation models are continuous, differentiable, nonlinear, and non-convex. Therefore, the candidate benchmark problems should be continuous, differentiable, nonlinear, and non-convex. Additional characteristics that categorize function complexity are modality, separability, and dimensionality. The modality of a candidate function describes the number of ambiguous peaks in its landscape. Multimodal indicates that a function has one or more local solutions besides the global solution, whereas a unimodal function only has a global solution. Separability and dimensionality control the complexity of a function. Separability gives a measure of variable relation among themselves. Separable functions are in general easy to solve, whereas non-separable functions are not. Dimensionality influences the difficulty of a problem by the number of variables. The search space is directly affected by the dimensions and grows exponentially as the number of dimensions increases [28].

The benchmark problem candidate consists of functions that reassemble the generic aircraft trim problem and gives insight to how the algorithm performs against literature findings. However, selecting a number of generic benchmark problems that reassembles the generic aircraft trim problem proved hard to identify. Many studies that implement routines to solve the aircraft trim problem set out the logical goal of only solving the aircraft trim problem. Besides achieving this goal is mostly a conclusion of the research, most trim modules aren't generic making it very time-consuming to create, implement and experiment on such a list of representative benchmark aircraft trim problems. This research did create a generic trim module, making it possible to construct such a list. The list of reassembling benchmark problems only becomes apparent after solving the problems. The trim module of this research minimizes a list of popular benchmark problems that match the aircraft trim problem criteria. After solving the popular problems and comparing them to the aircraft trim problem result, such a reassembling list becomes apparent.

Considering the criteria above, table 6 lists a selection of benchmark problems. In appendix B a selection of small to medium-sized nonlinear problems containing several nonlinear constraints made it to the list. The problems are G1 to G13.

Abbriv	Problem	optimum	search space	constraints	Dim	source
ROS	$f_{105}(\mathbf{x}) = \sum_{i=1}^{n-1}[100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2]$	$f(\mathbf{x}^*) = 0$	$30 \leq x_i \leq 30$	N	20 & 40 & 60	[28]
POW	$f_{92}(\mathbf{x}) = \sum_{i=1}^{D-3}\{(x_i + 10x_{i+1})^2 + 5(x_{i+2} - x_{i+3})^2 + (x_{i+1} - 2x_{i+2})^4 + 10(x_i - x_{i+3})^4\}$	$f(\mathbf{x}^*) = 0$	$4 \leq x_i \leq 5$	N	20 & 40 & 60	[28]
PERM	$f_{\text{Perm}}(\mathbf{x}) = \sum_{k=1}^{n}\left\{\sum_{i=1}^{n}(i^k + 0.5)\left[\left(\frac{x_i}{i}\right)^k - 1\right]\right\}^2$	$f(\mathbf{x}^*) = 0$	$-n \leq x_i \leq n$	N	2 & 4 & 6	[3]

Table 6: search space constrained problems

4.2 Implementation and Settings for DASMAT

This section demonstrates the parameter settings and implementation using the DASMAT aircraft trim problem. The aircraft trim problem settings depend on available DASMAT initial conditions. If there exist initial conditions for the steady-state flight conditions: straight, straight-and-level, straight-and-wing level, push-over, pull-up, steady-state turning flight then the initial conditions are reproduced. If there are no specific initial flight conditions available, the trim settings consider common use flight conditions such as standard glides slopes or standard turn rates. Abbreviations indicate the specific aircraft trim problem that the algorithm solves. For example, the abbreviation FL280KIAS190 indicates an aircraft trim problem at a Flight Level (FL) of 280 traveling at a Knots Indicated Air Speed (KIAS) of 190. If a straight-flight (S) initial condition follows a glide path of 3^o, the abbreviation becomes FL280KIAS190S3. The general form of this abbreviation is FLh_0KIASV_0Sγ_0 which the enumeration below explains. This construction follows an abbreviation order that indicates part 1 by FLh_0, part 2 by KIASV_0 and part 3 by Sγ_0, but many other forms for part 3 are possible, see below.

1. (FLh_0) Flight Level
2. (KIASV_0) Knots Indicated Air Speed

`http://www.benchmarkfcns.xyz/fcns`
`http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm`

3. (Sγ_0) Straight-flight **or** (SLγ_0) straight-and-level-flight **or** (PO$\dot{\gamma}_0$)/(PU$\dot{\gamma}_0$) push-over/pull-up **or** (T$\dot{\psi}_w$Gγ_0) steady-state Turning flight

Table 2 in section 2.3 describes general settings for specific initial conditions. Table 7 describes the specific DASMAT aircraft trim conditions using the abbreviations, and Table 2

Straight flight: FL70KIAS200G3	
TIP	$h = 70$, $V_{kias} = 200$, $\gamma_0 = 3^o$ and $p = q = r = 0$
PC	$\gamma = \sin^{-1}(\cos\alpha\cos\beta\sin\theta - \sin\beta\sin\phi\cos\theta - \sin\alpha\cos\beta\cos\phi\cos\theta)$
$c(\xi)_1$	$\gamma - \gamma_0$
TCP	$\vec{\xi} = [\alpha, \beta, \phi, \theta, \psi, \delta_T, \delta_e, \delta_a, \delta_r]^T$
Straight and level flight: FL280KIAS190WLG0	
TIP	$h = 280$, $V_{kias} = 190$, $\gamma_0 = \phi = p = q = r = 0$
PC	$\gamma = \sin^{-1}(\cos\alpha\cos\beta\sin\theta - \sin\beta\sin\phi\cos\theta - \sin\alpha\cos\beta\cos\phi\cos\theta)$
$c(\xi)_1$	$\gamma - \gamma_0$
TCP	$\vec{\xi} = [\alpha, \beta, \theta, \psi, \delta_T, \delta_e, \delta_a, \delta_r]^T$
pull-up flight: FL200KIAS200PU3G0	
TIP	$h = 200$, $V_{kias} = 200$, $\dot{\gamma}_0 = 3$, and $\gamma = \phi = 0$
PC	$\gamma = \sin^{-1}(\cos\alpha\cos\beta\sin\theta - \sin\beta\sin\phi\cos\theta - \sin\alpha\cos\beta\cos\phi\cos\theta)$
DP	$\phi_w = \tan^{-1}\frac{\cos\alpha\sin\beta\sin\theta+\cos\beta\cos\theta\sin\phi+\sin\alpha\sin\beta\cos\theta\cos\phi}{\sin\alpha\cos\theta\cos\phi+\sin\alpha\sin\phi}$
	$\begin{bmatrix}p\\q\\r\end{bmatrix} = \begin{bmatrix}\cos\alpha\cos\beta & -\cos\alpha\sin\beta & -\sin\alpha\\ \sin\beta & \cos\beta & 0\\ \sin\alpha\cos\beta & -\sin\alpha\sin\beta & \cos\alpha\end{bmatrix}\begin{bmatrix}1 & 0 & -\sin\gamma\\ 0 & \cos\phi_w & \cos\gamma\sin\phi_w\\ 0 & -\sin\phi_w & \cos\gamma\cos\phi_w\end{bmatrix}\begin{bmatrix}\dot{\phi_w} = 0\\ \dot{\gamma} = \dot{\gamma}_0\\ \dot{\psi_w} = 0\end{bmatrix}$
$c(\xi)$	$[\gamma - \gamma_0, p - p_{DP}, q - q_{DP}, r - r_{DP}]$
TCP	$\vec{\xi} = [p, q, r, \alpha, \beta, \phi, \theta, \psi, \delta_T, \delta_e, \delta_a, \delta_r]^T$
Coordinated Turning flight: FL100KIAS250T3G0	
TIP	$h = 100$, $V_{kias} = 250$, $\dot{\psi}_w = 3^o$, $\gamma_0 = 0$, and $\dot{\phi}_w = \dot{\gamma} = 0$
PC	$\gamma = \sin^{-1}(\cos\alpha\cos\beta\sin\theta - \sin\beta\sin\phi\cos\theta - \sin\alpha\cos\beta\cos\phi\cos\theta)$
DP	$\pm\phi_w = \tan^{-1}\left[\frac{(n^2-\cos^2\gamma)^{\frac{1}{2}}}{\cos\gamma}\right]$
	$\phi_w = \tan^{-1}\frac{\cos\alpha\sin\beta\sin\theta+\cos\beta\cos\theta\sin\phi+\sin\alpha\sin\beta\cos\theta\cos\phi}{\sin\alpha\cos\theta\cos\phi+\sin\alpha\sin\phi}$
	$\begin{bmatrix}p\\q\\r\end{bmatrix} = \begin{bmatrix}\cos\alpha\cos\beta & -\cos\alpha\sin\beta & -\sin\alpha\\ \sin\beta & \cos\beta & 0\\ \sin\alpha\cos\beta & -\sin\alpha\sin\beta & \cos\alpha\end{bmatrix}\begin{bmatrix}1 & 0 & -\sin\gamma\\ 0 & \cos\phi_w & \cos\gamma\sin\phi_w\\ 0 & -\sin\phi_w & \cos\gamma\cos\phi_w\end{bmatrix}\begin{bmatrix}\dot{\phi_w} = 0\\ \dot{\gamma} = 0\\ \dot{\psi_w} = \dot{\psi_{w_0}}\end{bmatrix}$
TCP	$\vec{\xi} = [p, q, r, \alpha, \beta, \phi, \theta, \psi, \delta_T, \delta_e, \delta_a, \delta_r]^T$
Descending Turning Flight: FL70KIAS200T3G3	
TIP	$h = 70$, $V_{kias} = 200$, $\dot{\psi}_w = 3^o$, $\gamma_0 = 3^o$, and $\dot{\phi}_w = \dot{\gamma} = 0$
PC	$\gamma = \sin^{-1}(\cos\alpha\cos\beta\sin\theta - \sin\beta\sin\phi\cos\theta - \sin\alpha\cos\beta\cos\phi\cos\theta)$
DP	$\pm\phi_w = \tan^{-1}\left[\frac{(n^2-\cos^2\gamma)^{\frac{1}{2}}}{\cos\gamma}\right]$
	$\phi_w = \tan^{-1}\frac{\cos\alpha\sin\beta\sin\theta+\cos\beta\cos\theta\sin\phi+\sin\alpha\sin\beta\cos\theta\cos\phi}{\sin\alpha\cos\theta\cos\phi+\sin\alpha\sin\phi}$
	$\begin{bmatrix}p\\q\\r\end{bmatrix} = \begin{bmatrix}\cos\alpha\cos\beta & -\cos\alpha\sin\beta & -\sin\alpha\\ \sin\beta & \cos\beta & 0\\ \sin\alpha\cos\beta & -\sin\alpha\sin\beta & \cos\alpha\end{bmatrix}\begin{bmatrix}1 & 0 & -\sin\gamma\\ 0 & \cos\phi_w & \cos\gamma\sin\phi_w\\ 0 & -\sin\phi_w & \cos\gamma\cos\phi_w\end{bmatrix}\begin{bmatrix}\dot{\phi_w} = 0\\ \dot{\gamma} = 0\\ \dot{\psi_w} = \dot{\psi_{w_0}}\end{bmatrix}$
TCP	$\vec{\xi} = [p, q, r, \alpha, \beta, \phi, \theta, \psi, \delta_T, \delta_e, \delta_a, \delta_r]^T$
Minimizing $f(\xi) \to 0$	

Table 7: Test conditions that led to the results

The User selects the desired steady-state flight conditions by using the following GUI.

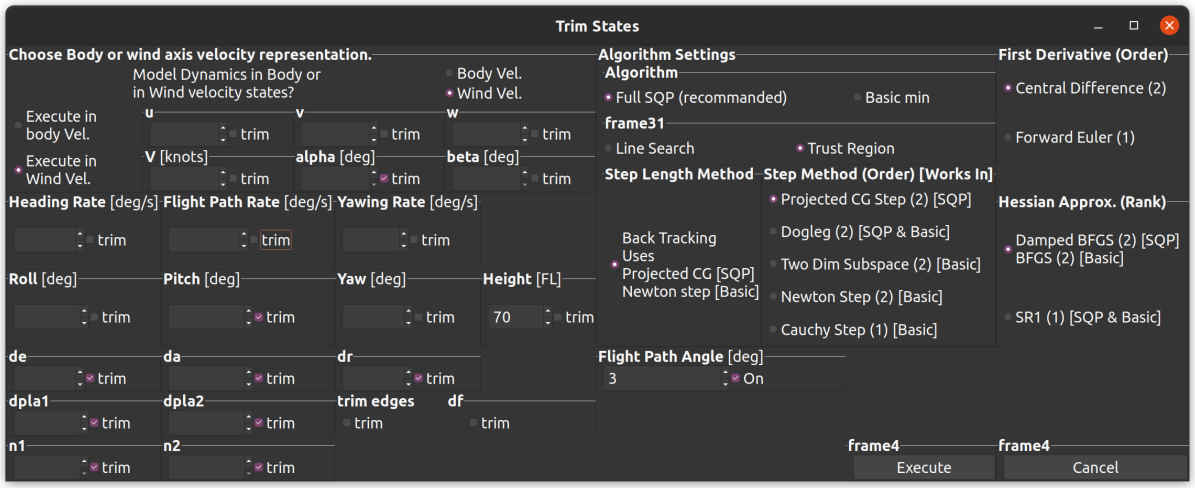


Figure 13: The trim module its Graphic User Interface in its development stage.

The GUI both controls the steady-state aircraft maneuver settings and the selection of the trim algorithm features. The left side is dedicated to flight maneuvering selection and the right side algorithm settings are selected. In addition to the general steady-state flight trajectories defined in Section 2.3, the GUI allows a user to disable rudder trim, simulate engine failure, or any other input failure. If a user wants to include a variable in the Trim Control Parameter set, then the user ensures the box labeled 'trim' is unchecked. In the setup that figure 13 visualizes, the parameters labeled 'Roll' is not added to the TCP vector, but 'Pitch' is. If a field is left empty while not added to the TCP vector, it's added to the TIP set and put to zero by default. This setup will find a solution to the steady-state straight and wing-level descending flight maneuver at an altitude of 70 FL. By default, the trim module searches the flight maneuver using the TCBP algorithm, but other combinations are selectable.

4.3 Experimental Setup and Method

The experiments are executed on a Dell XPS 15 7000 Series-7590 with a CPU: i9-9980HK (16 MB Cache, Base 2.40 GHz, Max. Turbo 5.0 GHz, 8 cores), Disk: 1 TB M.2 PCIe NVMe SSD, Graphics: GTX 1650 4 GB GDDR5, RAM: 32 GB DDR4-2666MHz. The Linux distribution Ubuntu 18.04 is chosen as OS platform, the program is written in C++ and executed as a supporting module in the real-time distributed middleware layer DUECA/DUSIME. Before starting the experiments all non-essential background apps were closed, the laptop was set to flight mode, the laptop power adapter connection was ensured, the 4k-OLED display is dimmed, laptop and room temperature were checked, external cooling fans were turned on, and the cache was cleared. During the start-up of the execution program, the *dueca_run.x* executor priority was increased. During the experimental execution, the room temperature is kept as similar as possible and external cooling fans are used to cool the machine. In between problem executions, no algorithm features are to change.

Each iteration sequence from start to finish is called a trial. In general, the trial is successful when it finds the global minima for the given minimization problem and fails otherwise. In this research, a trial is said to be successful if all items given below are satisfied.

1. The benchmark tolerance $\epsilon \leq 10^{-4}$ or Aircraft trim problem tolerance $\epsilon \leq 10^{-5}$ is satisfied for problem gradient of the Lagrangian $||f_k - A_k^T \hat{\lambda}_k||_\infty \leq \epsilon$ and constraint magnitude $||c_k||_\infty \leq \epsilon$;
2. The found trial optima must be within $\epsilon \leq 10^{-4}$ or $\epsilon \leq 10^{-5}$ absolute accuracy of the actual generic benchmark or aircraft trim problem solution. This logical test is described by formulation (4.1)

$$f(x^*) - \epsilon(|f(x^*)|+f(x^*) == 0) \quad \leq f(x^*_{trial}) \leq \quad f(x^*) + \epsilon(|f(x^*)|+f(x^*) == 0) \tag{4.1}$$

The trial is only stopped if one of the build-in safeguards is violated or criterion 1 is satisfied. If criterion 1 is satisfied, the trial solution is checked whether it's a local or global solution. If criterion 2 is satisfied,

then the trial solution is a global solution. If criteria 1 and 2 are not satisfied in some prescribed number of iterations, the trial attempt is ended, and a new attempt is initiated at some random point in the feasible search space.

All 14 methods are executed simultaneously for a period of 30 minutes. Starting at some pseudo-random point in the search-space of the problem. This random seed one initiates the function *srand* which selects a starting position in the search-space of the problem. The number of trial solutions is logged and analyzed after the experiment. Criteria 1 and 2 are tested for the found trial solutions for all algorithms and then for all problems. The algorithm that finds the most amount of global solutions in the given time is considered the best-performing algorithm. The best performing algorithm for the given problem becomes the benchmark algorithm, and its number of found solutions is defined as 100%. Other algorithms are expressed relative to the best algorithm. The generic benchmark results are described in section 4.5 and the aircraft trim problem results are visualized in section 4.6.

4.4 Safeguards and Limitations

This section describes safeguards that must be applied to prevent crashing of the algorithm under specific conditions. These conditions include the spreading of NaNs, preventing stalls, and enforcing positive definiteness if necessary.

The current limitations of the algorithms will reduce overall performance. The algorithm does not yet have a restoration phase. If the algorithm stalls, a new random point initiates until the convergent criteria are satisfied. The finite differences step size is a predetermined fixed value and is used regardless of the magnitude of the problem. The computed value $\hat{f}(\cdot)$ is equal to the summation of the actual value $f(\cdot)$ and truncation error $\tau(\cdot)$ defining $\hat{f}(x) = f(x) + \tau(x)$. The computed value may contain a small truncation error, but the derivative may experience a large error. This potential large error results in an inaccurate first-order derivative approximation. The first-order derivative is used to construct the second-order derivative, which further increases the potential error.

The Jacobian of active constraints is not yet suited with a feature that checks if the Jacobian matrix is of full row rank. The quality of the numerical derivative matrices is not checked on their condition, which may result in an ill-conditioned matrix. An ill-conditioned and/or not full-row ranked matrix may result in an inaccurate local solution or even result in a local maximizer instead of a minimizer. Positive definiteness cannot be guaranteed for the SR1 matrix, which is the common problem of using the SR1 Quasi-Newton update formula. Multiple methods exist for forcing a positive definite Hessian approximation. Here, positive definiteness is forced by applying $B_k = B_k + \alpha I$ for $\alpha \in (\lambda_1, 2\lambda_1]$ where λ_1 denotes the most negative eigenvalue of B_k. All SR1 algorithms use this implementation. This adjustment is very expensive for larger problems but is worthwhile for smaller problems. Bad or ill-conditioned matrices may reduce convergence speed near the solution. To further increase convergent speed near the solution of the problem, an eigenvalue clustering method should be implemented.

All methods contain a calculation and application of the Lagrangian estimate according to (3.30b). the implementation of the second-order correction method (3.32) is essential to avoid the Maratos effect and activates when an increase in constraint cost is detected. The central difference method assembles the constraint Jacobian matrix is offline using the set of active constraints. The finite difference methods (C,E) make use of the fixed step size based on Nocedal and Wright [38].

4.4.1 Important Limitations on Benchmark Results

Between the popular benchmark results and aircraft trim problem results, the trim algorithm underwent an important update. For this reason, the popular benchmark results can't be compared with the aircraft trim problem result. During the execution of all algorithm configurations solving different aircraft trim problem settings, the results were very inconsistent. The trim module gave many (thousands) solutions for one TIP configuration, but couldn't find a single solution for a slightly different setting. After a period of debugging, the problem became obvious. The standard **.inverse()** option gave rise to significant numerical instabilities under certain conditions. These conditions are not apparent when the objective function of the problems is only analytical formulas. ironically, these types of problems become the implementation testing problem during the implementation process because of their ease of debugging

and open/traceable structure. The problem does become apparent when a problem, like the DASMAT model, uses numerical procedures, data tables, or any other complex underlying structures to calculate a function. All algorithm combinations now implement the householder decomposition to extract an inverse matrix or solve a least-squares problem. This implementation procedure describes the last part of section 3.10. It describes that the algorithm selects different householder decomposition techniques depending on the magnitude of the gradient of the Lagrangian $\mathcal{L}$.

4.4.2 Aircraft Trim Implementation Limitations

Section 2.4 and Table 7 both show a set that contains the label derived parameters (DP). The goal of the derived parameters is to directly overwrite the parameter set $[p, q, r]^T$. This results in a smaller problem that invokes fewer TCPs. However, imposing the DP set $[p, q, r]^T$ as constraints were the least time-consuming approach. Creating yet another update did not fit in the time window of the graduation period. The consequence of this implementation increases the number of arithmetic operations, resulting in a less efficient algorithm. Another parameter that is not yet generic is the GUI parameter V_{KIAS}. In the current implementation, a conversion process in the trim GUI module converts the desired V_{KIAS} to the dynamics model usable V. In a future expansion, This precalculation may estimate the starting point for V_{KIAS} after which the algorithm minimizes the error by adding a TCP.

4.5 Results of Popular Benchmark Problems

All selected benchmark problems are executed regarding the conditions described in section 4.3. The algorithm with the most global solutions becomes the benchmark algorithm for a specific benchmark problem. The number of solutions of all other algorithms that solve the same problem is divided by the number of solutions found by the best algorithm multiplied by 100%. This result shows the relative performance % per algorithm. Label (All) in figure 14 sums all popular benchmark problem results, then defines the best as 'the benchmark' converting its score to 100%.

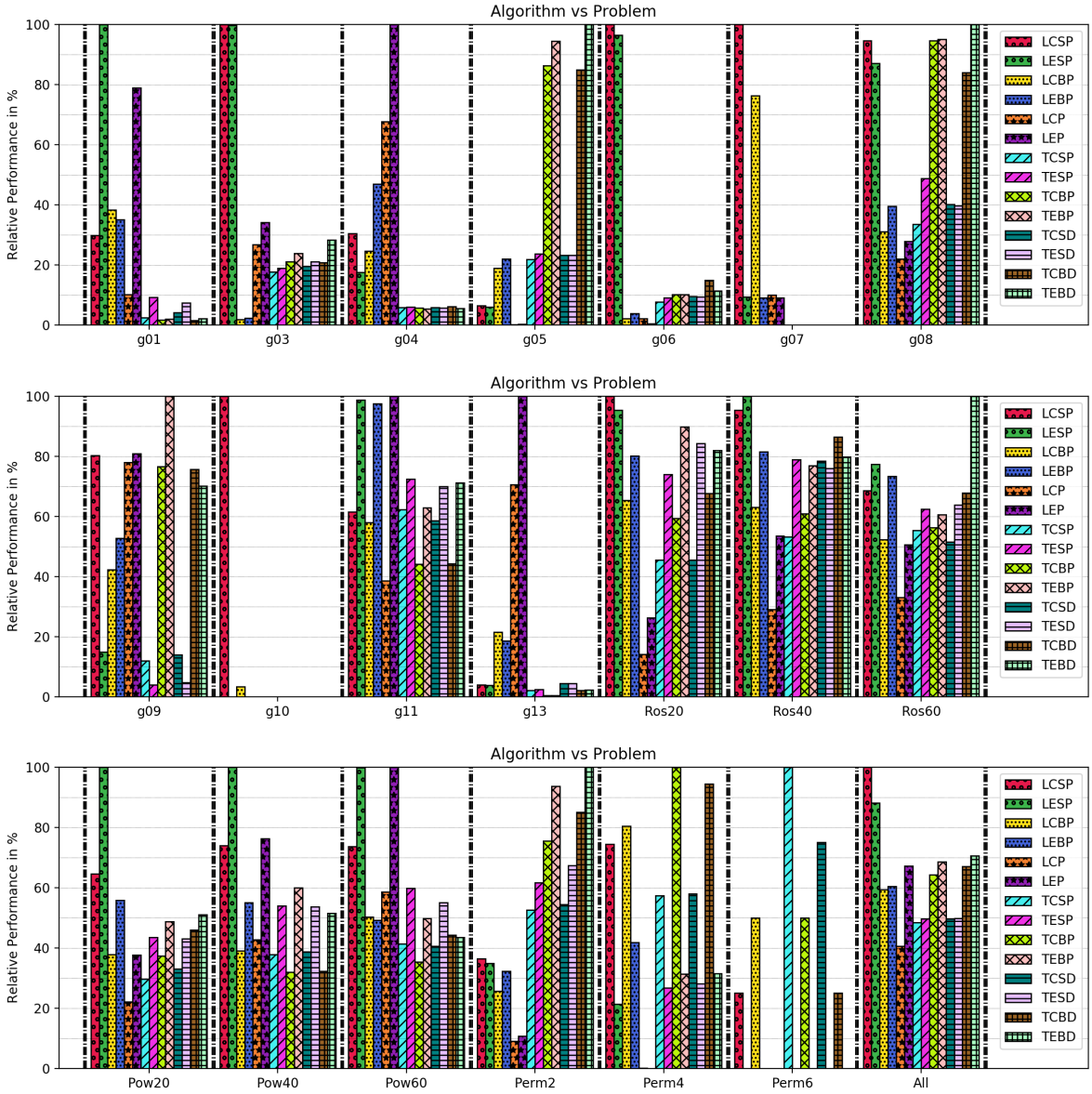


Figure 14: SQP algorithm with different active features evaluating analytical benchmark problems. Striped bars belong to the trust-region method, and line search results contain symbols.

The LCSP and LESP algorithms show the most reliable and best overall performance in the line-search class. The LCSP and LCBP are the only feature combinations that successfully find the global minima for all problems. This means that, for this set of problems, the feature combinations LCSP and LCBP are the most robust. In the trust-region class, all damped-BFGS feature combinations show better performance than the trust-region SR1 feature combinations. Figure 14 illustrates the overall performance of all algorithms using the label 'All'. The overall performance of all results doesn't deviate by an order of magnitude. The overall best algorithm LCSP finds 250% more solutions than the overall worst algorithm LCP.

4.6 Results of Aerospace Benchmark Problems

The aerospace benchmark problems consist of a helicopter model and the DASMAT model. The helicopter model is written in state-space format according to Mitchell et al. [36] with parameters described in [23]. The system is derived using a body frame of reference, the state vector contains eight states and

the input vector contains four inputs. The DASMAT model is built using Matlab Simulink and used through its C export code. This model uses the wind frame of reference velocity components. Both models are implemented in DUECA and depend on a complex grid of communication channels and activities handled by DUECA.

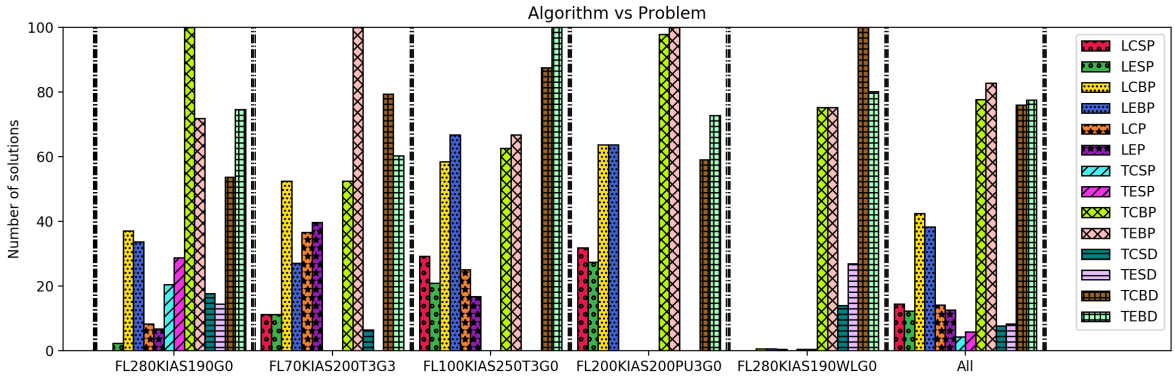


Figure 15: Performance on different types of tests.

The trust-region, damped-BFGS algorithm combinations show the best performance when solving the aircraft trim problem. The trust-region, SR1 algorithm combinations show the worst overall performance. This trend is also observable when inspecting the line search algorithm's performance. The line search, damped BFGS class performs overall better at solving the aircraft trim problem than the line search, SR1 dependent methods. For now, the most favorable combinations for solving the aircraft trim problem are: the TEBP, TCBP, TEBD, and TCBD algorithm combinations.

5 Discussion

When inspecting the results that Section 4.5 illustrates, the algorithm combinations LCSP and LESP have the best overall performance on the benchmark problems. The same cannot be said for the performance of these algorithms on the aerospace trim problems in Section 4.6. Here, the best performing are the trust-region, damped-BFGS algorithm combinations. However, when inspecting the results more closely, the results in Section 4.5 and Section 4.6 do show agreement. When inspecting the content of the results in the plot labeled 'All' in both sections, the damped-BFGS algorithm combinations show satisfactory performance. Important to note, and explained more thoroughly in this discussion, the results in Section 4.5 use a different inverse function than the results obtained in Section 4.6 because of numerical issues. The performance of the SR1 algorithm seems most affected. Because the damped-BFGS updating formula is generally more robust than the SR1 updating formula Nocedal and Wright [38], chances are that the SR1 encountered numerical issues earlier in its iteration process. When an algorithm's implemented safeguard detects a numerical issue, the safeguard resets the search sequence, meaning an algorithm starts with a cleaned memory at a new random point. Again, because the SR1 method is less stable than the damped-BFGS method, and therefore requires more safeguards (see Section 3.5.6 and Nocedal and Wright [38]), this worked favorably for the SR1 algorithm combinations. But in reality, the SR1 algorithm was stopped faster because it deteriorated faster, which explains the drop in performance when a different, but more stable inverse calculator was introduced. To test this hypothesis, it's recommended to investigate this in a followup research. If this hypothesis holds, the damped-BFGS updating strategy becomes the recommended Hessian updating strategy for solving the selected set of analytical benchmark problems and the DASMAT aircraft trim problem. If no followup research tests this hypothesis, the user is advised to select the damped-BFGS updating strategy, although all other algorithm combinations stay available for a user to select.

The implemented trim module uses the graphic user interface (GUI) to target specific initial conditions. Selecting an appropriate set of parameters results in a specific type of aircraft trajectory. The parameter where the user may enter a target for, is the velocity parameter V_{KIAS}. The relation between the Knots indicated airspeed and the true airspeed is aircraft dependent. A generic trim module must solve for this parameter, adding a variable to the TCP. However, this study uses a conversion to calculate the true airspeed from the user targeting parameter V_{KIAS}. This choice reduces the number of variables by one and hence increases performance. However, it limits the generic property of the trim module. A future expansion may add the parameter as a variable and may use the conversion as a starting point.

To achieve a simpler implementation, the trim module imposes the derived parameter set $[p, q, r]^T$ as a constraint. Marco, Duke, and Berndt [32] suggest a different approach by deriving the parameter set $[p, q, r]^T$ using the Eq. 2.11, and then updating these parameters, which is recommended for a further improvement of the trim algorithm. The implementation now imposes the derived parameter set $[p, q, r]^T$ as constraints. Minimizing these constraints is only possible if the parameters are part of the TCP set, increasing its size. Although the trim module is successful at finding a solution using the larger TCP set, it makes finding a solution unnecessarily complex.

The parameters in section 2.3 describe settings for maneuvers described in table 2 that refer to problems presented in Eq. (2.10). The goal is to create a generic initial trim module that solves the aircraft trim problem for user-customizable and common flight maneuvers. Implementing analytical constraints based on aircraft parameter relations allows a direct constraint evaluation in the trim module. The direct evaluation removes the need of sending the data from the trim module to the flight dynamics module, executing an aircraft model with the trim module's found parameter settings, and sending the result back to the trim module for optimization evaluation. The direct implementation reduces the computational cost and data traffic between the trim module and the flight dynamics module. Consequently, the generic application is compromised in some sense. The flight path angle, wind-axis bank attitude, and parameter set $[p, q, r]^T$ for now specifically depend on the Euler and wind angles. By only adding an additional transformation layer between the parameter set and the constraint calculator, and creating a GUI spacial representation selector, this problem is resolved.

The numerical routine has implemented some stopping criteria to prevent cycling, deterioration of the step method, or any event that triggers stalling of the algorithm. One event that triggers stalling is not yet having a good predictor, slack variables, and updating system for the set of active constraints.

Resulting in switching on and off of active constraints near the solution providing the minimizing step p_k at $\mathcal{A}(x)$ and not at $\mathcal{A}(x^*)$ as mentioned in 4.4. The switching on and off only happens when the constraints are inequality constraints. This may explain the performance failure on problems $g07$ and $g10$, as these problems both impose a relatively large set of inequality constraints on their variables. This missing feature may only become problematic for the aircraft trim problem when inequality conditions are appended or solutions lie on the edge of the search space of some TCP's. All aircraft trim tests performed until this point didn't find a solution close enough to the search space edge to cause this shortcoming.

the results of the popular benchmark problems in section 4.5 use a different Eigen implementation than the results of aerospace benchmark problems in section 4.6. All calculations in section 4.5 use the computation *.inverse()* function to calculate important step information. However, during the execution of the DASMAT aircraft problem, the results became inconsistent, as section 4.4.1 further elaborates. After some debugging, it was found that the *.inverse()* function gave rise to numerical issues and was replaced by householder decompositions. A simple strategy based on experimental observations for choosing the appropriate decomposition is set to be dependent on the closeness to the stopping criteria. For a large, small, and very small at magnitude difference from $|\nabla f_k - A_k^T \hat{\lambda}|_\infty$ the less expensive HouseholderQR, more stable ColPivHouseholderQR and the well-proven FullPivHouseholderQR decomposition was used respectively.

The HouseholderQR decomposition may still experience numerical issues far from the solution, as nonlinearity may induce challenging conditioned matrices. This implementation accepts the numerical issues and restarts them when they get too severe. Later implementations may develop routines to monitor the numerical behavior of the matrices involved. By incorporating the *info()* function, the numerical issues can be monitored. However, nothing can be said about the quality of the matrix itself. Ill-conditioned matrices and large condition numbers cannot be monitored without significant computational expenses. The SR1-based algorithms may benefit from monitoring its matrix condition, but more research is needed to confirm this.

6 Conclusion

The Delft University of Technology ('TU Delft') developed a real-time distributed system for scientific and educational purposes. Because of the high level of expertise required to learn from- and work in a real-time environment, TU Delft created a middleware layer, DUECA (Delft University Environment for Communication and Activation), and a simulation-specific addition framework: DUSIME (Delft University SIMulation Environment). Application programmers such as students or educators may develop a complex aircraft simulation module, but must supply their models with precalculated starting conditions, which is a labor-intensive process. A common practice is embedding the numerical optimization tool in an aircraft model and retrieving the starting conditions, referred to as the initial trim set. Setting up such an embedded tool for every aircraft model is also very labor-intensive. For over 20 years, these issues have limited the overall user experience in DUECA.

Hence, the research goal is to create an independent, generic, User-commanded, numerical optimization module capable of solving the aircraft trim problem in DUECA. This research created a generic and independent SQP (initial trim) module that reduces embedding labor and optimization knowledge. The module is generic such that it connects to many different aircraft models and is defined independently for its limited involvement in aircraft models. The trim module works by a user selecting a desired steady-state aircraft trajectory through a Graphic User Interface (GUI) and then commands the trim module to search for the set of initial trim conditions. The trim module finds the initial trim conditions by executing its SQP algorithm. This initial trim set allows the starting up of an aircraft simulation in a steady-state, that is stable enough such that a pilot can take over manual control.

The stability and possible steady-state conditions depend on the parameters that define equilibrium flight and the constraining equations. An system in motion or at rest achieves steady-state flight if the dynamic state derivatives $(\dot{p}, \dot{q}, \dot{r})$ and $(\dot{u}, \dot{v}, \dot{w})$ or $(\dot{V}, \dot{\alpha}, \dot{\beta})$ are zero. This allows predefining the dynamic state variables (p, q, r) and (u, v, w) or (V, α, β) constant or zero. Besides the dynamic state derivatives of the aircraft's rigid body, a trim algorithm must also minimize the engine model, the leading-edge flap actuator dynamics, or any other component that may introduce an acceleration. If the squared sum of the accelerations is smaller than 10^{-5}, the steady-state flight conditions are considered sufficiently stable. However, without a proper GUI and parameter constraints, users may find themselves solving for random steady-state flight conditions. The GUI accepts quantities for the Flight level h, roll angle ϕ, the knots indicated airspeed V_{KIAS}, glide path angle γ, glide path rate $\dot{\gamma}$ and wind-axis yaw rate $\dot{\psi}_w$. Selecting quantities for these GUI parameters actives trim routines for: straight flight, straight and wing-level flight, pull-up/push-over flight, steady-state turning flight, or any combination of these flight conditions. The maneuver straight flight describes the steady-state movement of an aircraft in a straight line without a body roll angle and selectable flight path angle, straight-and-level is a maneuver that is also straight and flies parallel relative to flat earth, Push-Pull presents a user with a condition that is ascending-descending maneuver from a straight-and-level departing instantaneous moment in time, and steady-state turning describes an ascending, level, or descending turning flight. The underlying transformations make it possible to connect aircraft models using wind axis of reference, body axis of reference, or quaternion-based flight systems. The algorithm so far is successful at solving the DASMAT aircraft trim model for the given stability. The DASMAT model is a wind-axis frame of reference derived aircraft model that contains additional dynamic engine states. By connecting more layers of frame of reference transformations in the future, more spatial representations can be solved.

The initial trim algorithm used during this research is a trust-region and line search Equality Constrained approach Sequential Quadratic Programming (SQP) algorithm. This algorithm was tested on commonly used benchmark problems in C++, where its possible weaknesses were exposed. The mathematics of the initial trim algorithm is verified by using literature substantiated methods based on pseudocodes. The trim algorithm is validated by simulating the found initial trim parameters and checking if it's indeed the appropriate flight maneuver.

The most important performance indicator of this research is the robustness of the algorithm. The algorithm combination that has a consistent overall performance on a wide range of problems in its class becomes the most favorable algorithm. The performance of these test features governs all combinations of: the line search and trust-region framework, first-order derivative functions Euler forward and central differences, the second-order derivative updating formula SR1, and the damped BFGS, the step method

PCG, and only for trust-region a projected dogleg method. the experiments only test for the relative performance between the framework and features. No external results were used for comparison in this research. External results were only used to choose the right algorithm methods, framework, and features. Overall, the trust-region method showed to be more robust than the line search method because of its consistent and stable performance on all benchmark test problems. The line search methods showed a relative superior efficiency under certain circumstances. These results are in line with recommendations found in the literature. The literature points out that the trust-region algorithm overall may be slightly more stable, while the line search may be slightly more aggressive. The central difference method showed better overall performance, but the first-order performance can be optimized by switching from Euler to central difference close to the solution, as advised by [38]. This also depends on the quality of the matrix decomposition. The PCG method turns out to be the most superior, but the derived projected dogleg showed compatibility on smaller problems. This makes sense as the dogleg takes two leaps towards a solution where the PCG keeps iterating until some condition is satisfied. This may be less important for smaller problems, but more significant for larger problems. The damped BFGS algorithm is not only more favorable according to literature, but also shows a better performance than the SR1 algorithm. Concluding its stability and guaranteeing definite properties is more important than the more reassembling and cheaper Hessian SR1 estimation for the given set of problems and conditions.

The feature combinations successfully solve the DASMAT aircraft trim problem. The algorithms most effective for solving the common benchmark problems are the trust-region damped BFGS algorithms. The relative best common benchmark trust-region solver performed 28% less effectively than the best line search algorithm. The relative best aircraft trim line search method solves 85% less effectively than the best trust-region method. According to the data so far, the BFGS method proves overall more robust than the SR1 updating strategy. The results of this study suggest that the trust-region method is more robust than the line-search method. for now, the data also suggests that both trust-region BFGS dogleg and PCG are both robust choices when implemented in an aircraft trim module. however, more investigation is needed to assess the performance on other aircraft models.

7 Recommendations

This section identifies a number of possible improvements over the current approach:

Reducing the number of trim control parameters: although the current implementation is capable of solving the aircraft trim problem for all maneuvers, the algorithm is not optimized when solving the pull-up, push-down, or turning flight aircraft trim problem. More generally, the TCP set is larger than necessary when the algorithm is solving the aircraft trim problem when $(p, q, r) \neq \vec{0}$. This problem can be avoided by following the implementation method that Marco, Duke, and Berndt [32] uses. For the current trim module implementation, this means adjusting the following. First, the dynamic states (p, q, r) are no longer found using constraints but derived. Using the user-defined *Target Initial Parameters* (TIP) $(\dot{\phi}_w, \dot{\gamma}_w, \dot{\psi}_w)$ and the kinematic state in the TCP, the set (p, q, r) is derived and updated using Eq. 2.11 just before evaluating the aircraft model. This way, the set of constraints, when solving the pull-up, push-down, or turning flight aircraft trim problem, consists of 1 constraint instead of 4, and the TCP set is reduced by 3. Implementing this recommendation will reduce the degrees of freedom and the number of constraints, which improves the algorithm's performance significantly.

Implementing the target initial parameter V_{KIAS}: the current implementation calculates the user-defined velocity parameter by using the equations formulated by Van der Linden [45] (p. 201-203). The equations that calculate the required atmospheric parameters and velocity conversions that result in V_{KIAS} are not wrong, but they might be differently employed by some other dynamic model, making this calculation DASMAT specific, not generic. If some dynamic model is capable of calculating its own V_{KIAS}, the parameter must be added to the list of constraints or to the list of dynamic state derivatives. The least labor extensive solution would be adding the user-defined V_{KIAS_0} minus the calculated V_{KIAS} squared making $(V_{KIAS_0} - V_{KIAS_0})^2$ to the list of dynamic derivative constraints. The correct method would be converting the offline constraint evaluation to an online calculation method. This way, the step method minimizes in the null-space of the constraints, making the process a lot more effective. Secondly, the adaptive penalty value will ensure that the difference between V_{KIAS_0} and V_{KIAS} is kept at a minimum without hindering the minimization process too much. However, both methods will be successful. Using the procedure that calculates V_{KIAS} currently, will give minimizing the difference between $(V_{KIAS_0} - V_{KIAS_0})^2$ a warm start, making the method more effective, but by far not as effective as the correct method.

Improving the adaptive decomposition routine: the current algorithm version has an adaptive Eigen HouseHolder decomposition procedure that switches between the least stable but fastest decomposition *HouseholderQR*, the well-balanced decomposition *ColPivHouseholderQR*, and the proven but slowest decomposition *FullPivHouseholderQR*. These decomposition template functions are used to calculate the inverse of the Hessian or to solve a least-squares problem. Switching between the decomposition types is based on the magnitude of the gradient of the Lagrangian $\nabla_x \mathcal{L}(x, \lambda)$. Selecting the current switch values that determine which decomposition type executes the decomposition based on $\nabla_x \mathcal{L}(x, \lambda)$ is found using experiments. Because the decomposition type has a significant influence on the performance of the algorithm, this feature must further be optimized.

Indicating benchmark problems: because the popular benchmark problems and the aircraft trim problem use another solver, the results cannot be compared. Finding a set of problems that's comparable to the aircraft trim problem would be a nice to have secondary research outcome. Currently, no information can be provided as to which popular benchmark problem poses a similar level of complexity as solving the DASMAT aircraft trim problem. A followup research should take this opportunity to explore this uncharted field. The goal of the followup research will be searching for a set of candidate popular benchmark problems with similar complexity levels as the aircraft trim problem. After finding a set of problems, the performance of the current algorithms to other studies. This process may lead to trim module updates and may help other developers to construct a generic trim module.

References

[1] "4. The BFGS Method". In: *Iterative Methods for Optimization*, pp. 71–86. DOI: 10.1137/1.9781611970920.ch4. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611970920.ch4. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611970920.ch4.

[2] P. -A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton, UNITED STATES: Princeton University Press, 2007. ISBN: 978-1-4008-3024-4. URL: http://ebookcentral.proquest.com/lib/delft/detail.action?docID=457711.

[3] Ernesto P. Adorio. *MVF - Multivariate Test Functions Library in C for Unconstrained Global Optimization*. URL: http://www.geocities.ws/eadorio/mvf.pdf.

[4] Sameer Agarwal, Keir Mierle, et al. *Ceres Solver*. http://ceres-solver.org.

[5] Baghdadi et al. "Robust Methods for Aircraft Trim Computation and Analysis". In: Oct. 2009.

[6] Paul T. Boggs and Jon W. Tolle. "Sequential Quadratic Programming". In: *Acta Numerica* 4 (1995), pp. 1–51. DOI: 10.1017/s0962492900002518.

[7] Steven C. Chapra and Raymond P. Canale. *Numerical Methods for Engineers*. McGraw-Hill Education, 2015.

[8] Steven C. Chapra and Raymond P. Canale. *Numerical methods for engineers*. 6th ed. McGraw-Hill Higher Education, 2010.

[9] Robert T. N. Chen. *Efficient Algorithms for Computing Trim and Small-Disturbance Equations of Motion of Aircraft in Coordinated and Uncoordinated, Steady, Steep Turns*. Tech. rep. National Aeronautics and Space Administration Ames Research Center, 1983.

[10] Gary John Jr. Coleman. "A generic stability and control tool for flight vehicle conceptual design: Aeromech software develoment". PhD thesis. University of Texas, 2007.

[11] Michael V. Cook. *Flight Dynamics Principles: A Linear Systems Approach to Aircraft Stability and Control*. Third Edition. The Boulevard, Langford Lane, Kidlington, Oxford, United Kingdom: Elsevier, 2012.

[12] Joseph M. Cooke et al. "NPSNET: Flight Simulation Dynamic modeling using quaternions". In: *Presence: Teleoperators and Virtual Environments* 1.4 (1992), pp. 404–420. DOI: 10.1162/pres.1992.1.4.404.

[13] Frank E. Curtis, Tim Mitchell, and Michael L. Overton. "A BFGS-SQP method for nonsmooth, nonconvex, constrained optimization and its evaluation using relative minimization profiles". In: *Optimization Methods and Software* 32.1 (2017), pp. 148–181. DOI: 10.1080/10556788.2016.1208749. eprint: https://doi.org/10.1080/10556788.2016.1208749. URL: https://doi.org/10.1080/10556788.2016.1208749.

[14] Frank E. Curtis and Xiaocun Que. "A quasi-newton algorithm for nonconvex, nonsmooth optimization with Global Convergence Guarantees". In: *Mathematical Programming Computation* 7.4 (2015), pp. 399–428. DOI: 10.1007/s12532-015-0086-2.

[15] Yu-Hong Dai. "A perfect example for the BFGS method". In: *Mathematical Programming* 138 (Apr. 2012). DOI: 10.1007/s10107-012-0522-2.

[16] Eugene L Duke, Brian P Patterson, and Robert F Antoniewicz. "User's Manual for LINEAR, a FORTRAN Program to Derive Linear Aircraft Models". In: *NASA Technical Paper 2768* (1987). DOI: 10.2514/6.2007-6703. URL: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19880012356.pdf.

[17] Eugine.L. Duke, Robert. F. Antoniewicz, and Keith. D. Krambeer. *Derivation and Definition of a Linear Aircraft Model*. Tech. rep. National Aeronautics and Space Administration Ames Research Center, 1988.

[18] Chen Friedman and Omri Rand. "Robust trim procedure for rotorcraft configurations". In: *Aerospace Science and Technology* 45 (2015), pp. 442–448. ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2015.06.007. URL: https://www.sciencedirect.com/science/article/pii/S1270963815001844.

[19] Nicholas I. Gould, Mary E. Hribar, and Jorge Nocedal. "On the Solution of Equality Constrained Quadratic Programming Problems Arising in Optimization". In: *SIAM Journal on Scientific Computing* 23.4 (2001), pp. 1376–1395. DOI: 10.1137/s1064827598345667.

[20] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[21] Antônio B. Guimarães Neto et al. "Formulation of the Flight Dynamics of Flexible Aircraft Using General Body Axes". In: *AIAA Journal* 54.11 (2016), pp. 3516–3534. DOI: 10.2514/1.j054752.

[22] John R. Hauser. *Numerical Methods for Nonlinear Engineering Models*. Springer, 2009.

[23] Ronald Hess. "Analytical Assessment of Performance, Handling Qualities, and Added Dynamics in Rotorcraft Flight Control". In: *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 39 (Feb. 2009), pp. 262–271. DOI: 10.1109/TSMCA.2008.2007943.

[24] Willi Hock and Klaus Schittkowski. "Test examples for nonlinear programming codes". In: *Lecture Notes in Economics and Mathematical Systems* (1981). DOI: 10.1007/978-3-642-48320-2.

[25] Florian Holzapfel, Ingo Sturhan, and Gottfried Sachs. "Low-Cost PC Based Flight Simulator for Education and Research". In: *AIAA Modeling and Simulation Technologies Conference and Exhibit*. 2002. DOI: 10.2514/6.2002-4862. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2002-4862. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2002-4862.

[26] Mohd Asrul Hery Ibrahim, Mustafa Mamat, and Wah Leong. "BFGS method: A new search direction". In: *Sains Malaysiana* 43 (Oct. 2014), pp. 1591–1597.

[27] Aragón Francisco J. et al. *Nonlinear optimization*. Springer, 2019.

[28] Momin Jamil and Xin She Yang. "A literature survey of benchmark functions for global optimisation problems". In: *International Journal of Mathematical Modelling and Numerical Optimisation* 4.2 (2013), p. 150. DOI: 10.1504/ijmmno.2013.055204.

[29] Richard Khoury and Douglas Wilhelm Harder. *Numerical methods and modelling for engineering*. Springer, 2016.

[30] Slawomir Koziel and Zbigniew Michalewicz. "Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization". In: *Evolutionary Computation* 7.1 (1999), pp. 19–44. DOI: 10.1162/evco.1999.7.1.19.

[31] Jing Liang et al. "Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization". In: *Nangyang Technological University, Singapore, Tech. Rep* 41 (Jan. 2006).

[32] Agostino De Marco, Eugene Duke, and Jon Berndt. "A General Solution to the Aircraft Trim Problem". In: *AIAA Modeling and Simulation Technologies Conference and Exhibit* (2007). DOI: 10.2514/6.2007-6703.

[33] Giampiero Mastinu and Manfred Plöchl. *Road and off-road vehicle system dynamics: handbook*. CRC Press/Taylor amp; Francis Group, 2014.

[34] Richard McFarland. "Trimming an aircraft model for flight simulation". In: (Nov. 1987).

[35] Murat Millidere et al. "Further Investigations on Newton-Raphson Methods in Aircraft Trim". In: *AIAA Scitech 2021 Forum*. DOI: 10.2514/6.2021-0411. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2021-0411. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2021-0411.

[36] David Mitchell et al. "Determination of Maximum Unnoticeable Added Dynamics". In: *AIAA Atmospheric Flight Mechanics Conference and Exhibit*. DOI: 10.2514/6.2006-6492. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2006-6492. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2006-6492.

[37] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. New York, NY, USA: Springer, 1999.

[38] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer, 2006.

[39] M.M. van Paassen and Olaf Stroosma. "DUECA - DATA-DRIVEN ACTIVATION IN DISTRIBUTED REAL-TIME COMPUTATION". In: *AIAA Modeling and Simulation Technologies Conference and Exhibit* (2000). DOI: 10.2514/6.2000-4503.

[40] G.A. Periçaro et al. "HLRF–BFGS optimization algorithm for structural reliability". In: *Applied Mathematical Modelling* 39.7 (2015), pp. 2025–2035. ISSN: 0307-904X. DOI: https://doi.org/10.1016/j.apm.2014.10.024. URL: http://www.sciencedirect.com/science/article/pii/S0307904X14004971.

[41] David Peters and Dinesh Barwey. "A general theory of rotorcraft trim". In: *36th Structures, Structural Dynamics and Materials Conference* (1995). DOI: 10.2514/6.1995-1451.

[42] David Peters and Dinesh Barwey. "A general theory of rotorcraft trim". In: *Mathematical Problems in Engineering* 2 (Jan. 1996). DOI: 10.1155/S1024123X9600021X.

[43] J. R. Raol and Jatinder Singh. *Flight mechanics modeling and analysis*. Taylor & Francis, 2009.

[44] Brian L. Stevens, Frank L. Lewis, and Eric N. Johnson. *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons, 2016.

[45] C. A. A. M. Van der Linden. *Dasmat: Delft University aircraft simulation model and analysis tool*. University of Technology, 1996.

[46] Erik-Jan Van Kampen. "Global Optimization using Interval Analysis: Interval Optimization for Aerospace Applications". PhD thesis. Sept. 2010.

[47] Patrick Wieschollek. *CppOptimizationLibrary*. https://github.com/PatWie/CppNumericalSolvers. 2016.

[48] Alexander W Winkler. *Ifopt - A modern, light-weight, Eigen-based C++ interface to Nonlinear Programming solvers Ipopt and Snopt*. 2018. DOI: 10.5281/zenodo.1135046. URL: https://doi.org/10.5281/zenodo.1135046.

[49] Yuan. Ya-xiang. *A review of trust region algorithms for optimization*. Tech. rep. Institute of Computational Mathematics and Scientific/Engineering Computing, 2008.

Appendix A

The G01 Function

$$f(x) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - 5\sum_{i=5}^{13} x_i$$
$$g_1(x) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0$$
$$g_2(x) = 2x_1 + 2x_3 + x_{10} + x_{11} - 10 \leq 0$$
$$g_3(x) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0$$
$$g_4(x) = -8x_1 + x_{10} \leq 0$$
$$g_5(x) = -8x_2 + x_{11} \leq 0$$
$$g_6(x) = -8x_3 + x_{12} \leq 0$$
$$g_7(x) = -2x_4 - x_5 + x_{10} \leq 0$$
$$g_8(x) = -2x_6 - x_7 + x_{11} \leq 0$$
$$g_9(x) = -2x_8 - x_9 + x_{12} \leq 0$$

where $0 \leq x_i \leq 1 (i = 1, ..., 9)$, $0 \leq x_i \leq 100 (i = 10, 11, 12)$ and $0 \leq x_{13} \leq 1$. The optimum solution is $f(x^*) = -15$. [30].

The G03 Function

$$f(x) = -(\sqrt{n})^n \prod_{i=1}^{n} x_i$$
$$h(x) = \sum_{i=1}^{n} x_i^2 - 1 = 0$$

where $n = 10$ and $0 \leq x_i \leq 1$ $(i = 1, ..., n)$. The optimum solution is $f(x^*) = -1.00050010001000$ [30, 31].

The G04 Function

$$f(x) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$
$$g_1(x) = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \leq 0$$
$$g_2(x) = -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5 \leq 0$$
$$g_3(x) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \leq 0$$
$$g_4(x) = -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \leq 0$$
$$g_5(x) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \leq 0$$
$$g_6(x) = -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \leq 0$$

where $78 \leq x_1 \leq 102$ and $78 \leq x_i \leq 102$ $(i = 3, 4, 5)$. The optimum solution is $f(x^*) = -3.066553867178332e+004$ [30, 31].

The G05 Function

$$f(x) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3$$
$$g_1(x) = -x_4 + x_3 - 0.55 \leq 0$$
$$g_2(x) = -x_3 + x_4 - 0.55 \leq 0$$
$$h_3(x) = 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 = 0$$
$$h_4(x) = 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0$$
$$h_5(x) = 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 = 0$$

where $0 \leq x_1 \leq 1200$, $0 \leq x_2 \leq 1200$, $-0.55 \leq x_3 \leq 0.55$ and $-0.55 \leq x_4 \leq 0.55$. The optimum solution is $f(x^*) = 5126.4967140071$ [30, 31].

The G06 Function

$$f(x) = (x_1 - 10)^3 + (x_2 - 20)^3$$
$$g_1(x) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0$$
$$g_2(x) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0$$

where $13 \leq x_1 \leq 100$ and $0 \leq x_2 \leq 100$. The optimum solution is $f(x^*) = -6961.81387558015$ [30, 31].

The G07 Function

$$\begin{aligned} f(x) =& x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 \\ &+ 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 \\ &+ 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \end{aligned}$$
$$g_1(x) = -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0$$
$$g_2(x) = 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0$$
$$g_3(x) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0$$
$$g_4(x) = 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0$$
$$g_5(x) = 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0$$
$$g_6(x) = x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \leq 0$$
$$g_7(x) = 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \leq 0$$
$$g_8(x) = -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0$$

where $-10 \leq x_i \leq 10 \quad (i = 1, ..., 10)$. The optimum solution is $f(x^*) = 24.30620906818$ [30, 31].

The G08 Function

$$f(x) = -\frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1 + x_2)}$$
$$g_1(x) = x_1^2 - x_2 + 1 \leq 0$$
$$g_2(x) = 1 - x_1 + (x_2 - 4)^2 \leq 0$$

where $0 \leq x_1 \leq 10$ and $0 \leq x_2 \leq 10$. The optimum solution is $f(x^*) = -0.0958250414180359$ [30, 31].

The G09 Function

$$f(x) = (x_1 - 10)^2 + 5\,(x_2 - 12)^2 + x_3^4 + 3\,(x_4 - 11)^2$$
$$+ 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$
$$g_1(x) = -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0$$
$$g_2(x) = -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0$$
$$g_3(x) = -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0$$
$$g_4(x) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0$$

where $-10 \leq x_i \leq 10$ for $i = 1, ..., 7$. The optimum solution is $f(x^*) = 680.630057374402$ [30, 31].

The G10 Function

$$f(x) = x_1 + x_2 + x_3$$
$$g_1(x) = -1 + 0.0025\,(x_4 + x_6) \leq 0$$
$$g_2(x) = -1 + 0.0025\,(x_5 + x_7 - x_4) \leq 0$$
$$g_3(x) = -1 + 0.01\,(x_8 - x_5) \leq 0$$
$$g_4(x) = -x_1x_6 + 833.33252x_4 + 100x_1 - 83333.333 \leq 0$$
$$g_5(x) = -x_2x_7 + 1250x_5 + x_2x_4 - 1250x_4 \leq 0$$
$$g_6(x) = -x_3x_8 + 1250000 + x_3x_5 - 2500x_5 \leq 0$$

where $100 \leq x_1 \leq 10000$, $1000 \leq x_i \leq 10000$ $(i = 2, 3)$ and $10 \leq x_i \leq 1000$ $(i = 4, ..., 8)$. The optimum solution is $f(x^*) = 7049.24802052867$ [30, 31].

The G11 Function

$$f(x) = x_1^2 + (x_2 - 1)^2$$
$$h(x) = x_2 - x_1^2 = 0$$

where $-1 \leq x_1 \leq 1$ and $-1 \leq x_2 \leq 1$. The optimum solution is $f(x^*) = 0.7499$ [30, 31].

The G13 Function

$$f(x) = e^{x_1x_2x_3x_4x_5}$$
$$h_1(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0$$
$$h_2(x) = x_2x_3 - 5x_4x_5 = 0$$
$$h_3(x) = x_1^3 + x_2^3 + 1 = 0$$

where $-2.3 \leq x_i \leq 2.3$ $(i = 1, 2)$ and $-3.2 \leq x_i \leq 3.2$ $(i = 3, 4, 5)$. The optimum solution is $f(x^*) = 0.053941514041898$ [24, 31].

Appendix B

Example Problem

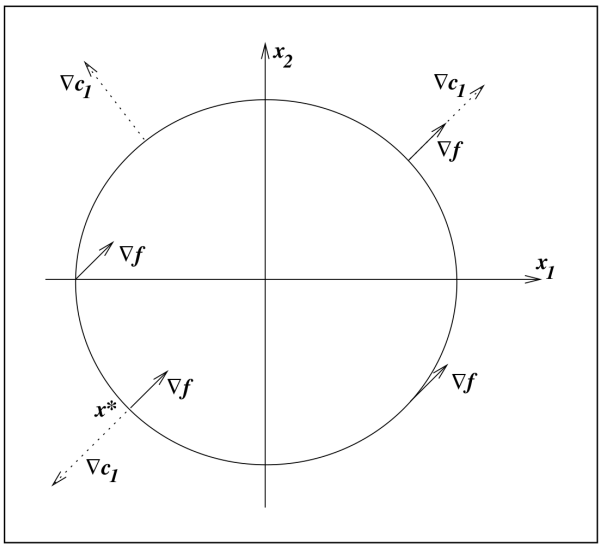


Figure 16: Minimizing a constrained optimization problem (Numerical Optimization: Example 12.1 page 308 [38])

Example problem:

$\min x_1 + x_2 \qquad \text{s.t.} \quad x_1^2 + x_2^2 - 2 = 0$

Defining:

$f(x) = x_1 + x_2$ and $c_1(x) = x_1^2 + x_2^2 - 2$

Lagragian function:

$\mathcal{L}(x, \lambda) = x_1 + x_2 - \lambda_1(x_1^2 + x_2^2 - 2)$

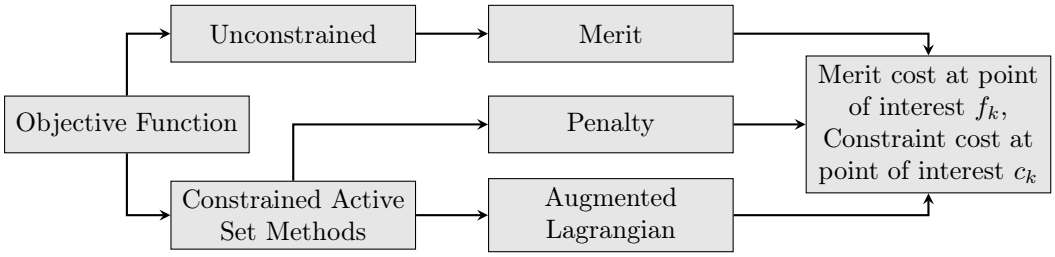
Gradient Lagrangian function:

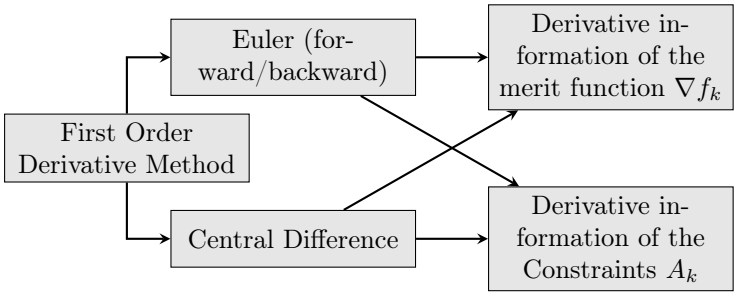
$$\nabla_x \mathcal{L}(x, \lambda) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \lambda_1 \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

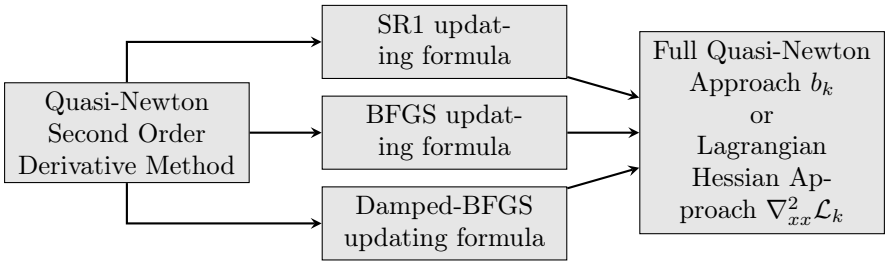
Solution that satisfies KKT condition Eq. (3.5)

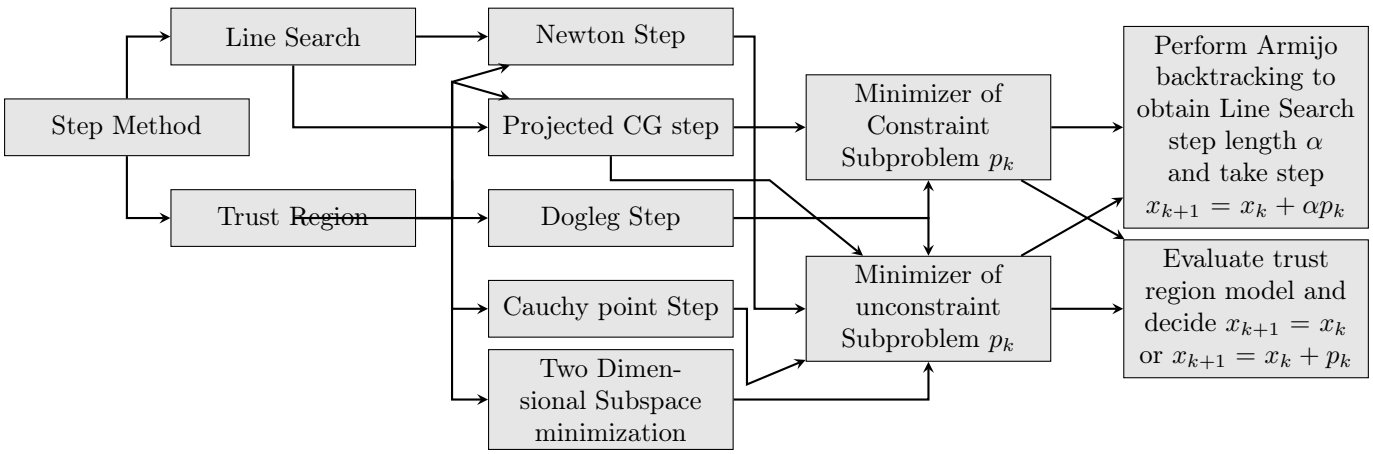
$x^* = (-1, -1)^T$ and $\lambda^* = 1/2$

Appendix C


Unconstrained
Merit
Objective Function
Penalty
Merit cost at point of interest f_k, Constraint cost at point of interest c_k
Constrained Active Set Methods
Augmented Lagrangian


Euler (forward/backward)
Derivative information of the merit function ∇f_k
First Order Derivative Method
Central Difference
Derivative information of the Constraints A_k


SR1 updating formula
Full Quasi-Newton Approach b_k or Lagrangian Hessian Approach $\nabla^2_{xx}\mathcal{L}_k$
Quasi-Newton Second Order Derivative Method
BFGS updating formula
Damped-BFGS updating formula


Line Search
Newton Step
Perform Armijo backtracking to obtain Line Search step length α and take step $x_{k+1} = x_k + \alpha p_k$
Step Method
Projected CG step
Minimizer of Constraint Subproblem p_k
Trust ~~Region~~
Dogleg Step
Minimizer of unconstraint Subproblem p_k
Evaluate trust region model and decide $x_{k+1} = x_k$ or $x_{k+1} = x_k + p_k$
Cauchy point Step
Two Dimensional Subspace minimization

Appendix D

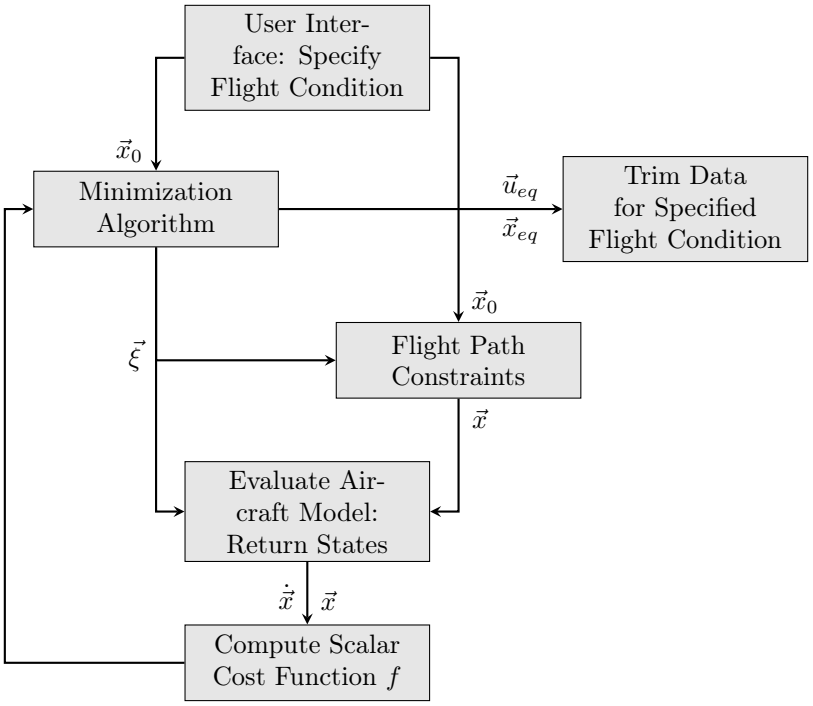
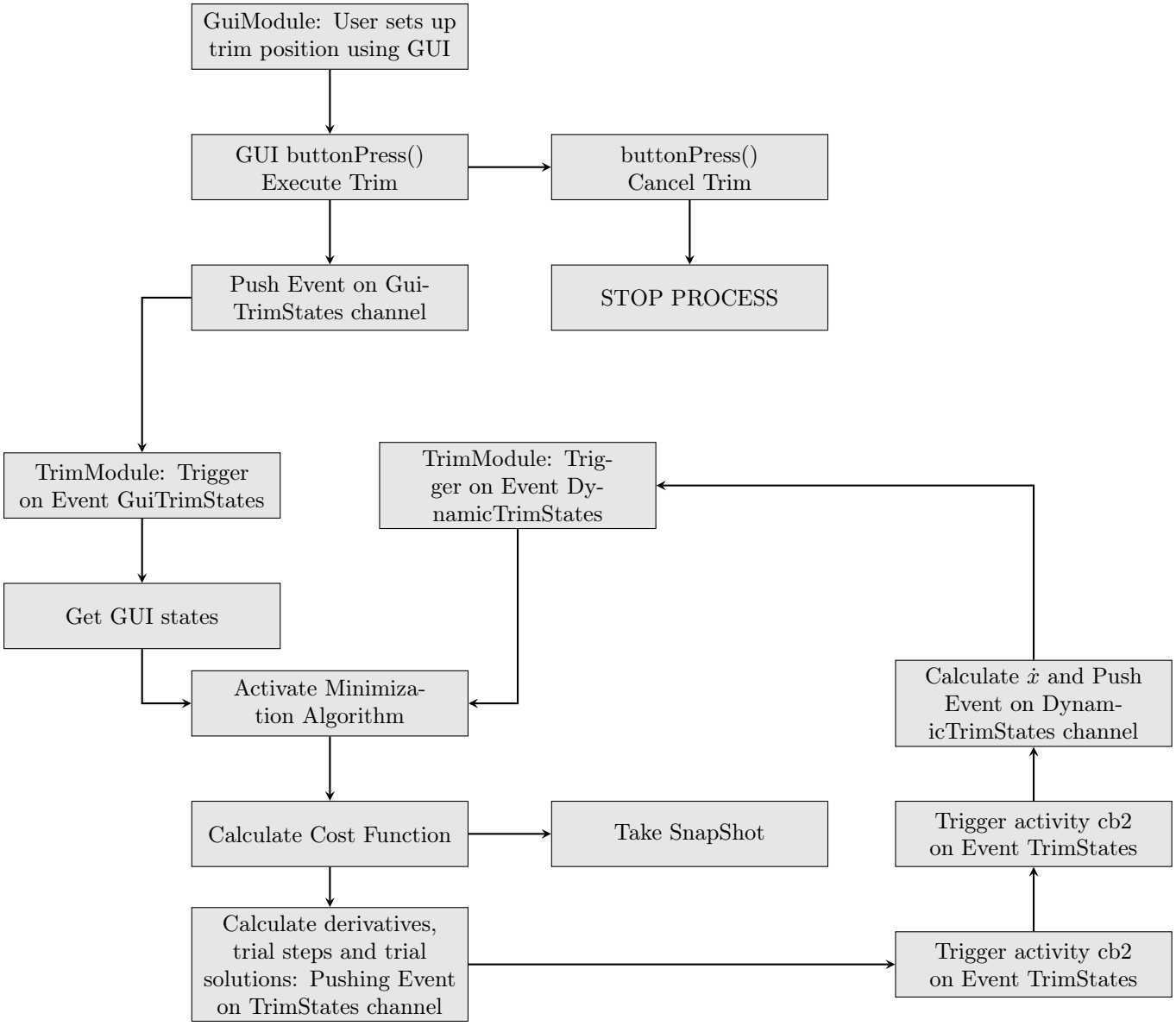
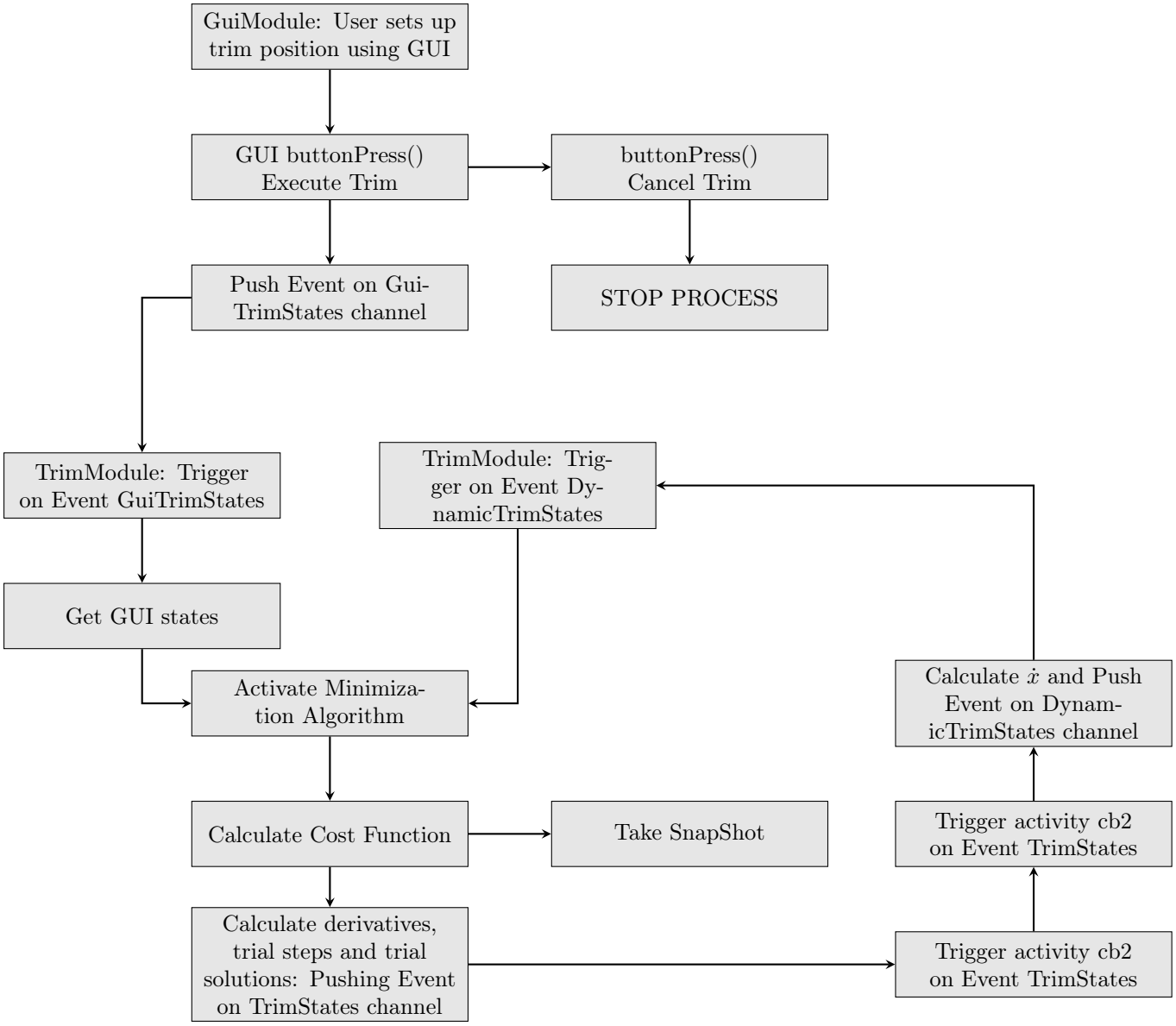


Figure 17: Flowchart for a generic trim routine [43, 44]


GuiModule: User sets up trim position using GUI
GUI buttonPress() Execute Trim
buttonPress() Cancel Trim
Push Event on Gui-TrimStates channel
STOP PROCESS
TrimModule: Trigger on Event GuiTrimStates
TrimModule: Trig-ger on Event Dy-namicTrimStates
Get GUI states
Activate Minimiza-tion Algorithm
Calculate $\dot{x}$ and Push Event on Dynam-icTrimStates channel
Calculate Cost Function
Take SnapShot
Trigger activity cb2 on Event TrimStates
Calculate derivatives, trial steps and trial solutions: Pushing Event on TrimStates channel
Trigger activity cb2 on Event TrimStates


GuiModule: User sets up trim position using GUI
GUI buttonPress() Execute Trim
buttonPress() Cancel Trim
Push Event on Gui-TrimStates channel
STOP PROCESS
TrimModule: Trigger on Event GuiTrimStates
TrimModule: Trig-ger on Event Dy-namicTrimStates
Get GUI states
Activate Minimiza-tion Algorithm
Calculate $\dot{x}$ and Push Event on Dynam-icTrimStates channel
Calculate Cost Function
Take SnapShot
Trigger activity cb2 on Event TrimStates
Calculate derivatives, trial steps and trial solutions: Pushing Event on TrimStates channel
Trigger activity cb2 on Event TrimStates

Appendix E

Note! float and double in dco

u	99.9982
w	-0.598303
theta	-5.81674 (degrees)
v	0
phi	0
delta b	0
delta c	0.0854101
delta a	0
delta p	0
cost	0.0018839
iteratios	491

Table 8: Results using float

u	99.9227
w	-4.01752
theta	-5.8307 (degrees)
v	0
phi	0
delta b	0
delta c	0.573972
delta a	0
delta p	0
cost	9.82303e-05
iteratios	68

Table 9: Results using double