Some popular nonlinear programming algorithms

Mohamed Tarek[1,2]

[1]PumasAI Inc., USA
[2]UNSW Canberra, Australia

October 2021

Many design and decision-making tasks can be formulated as a constrained nonlinear optimization problem with an objective, equality constraints, inequality constraints and bound constraints on the decision variables. The problem can then be solved with any of the many constrained nonlinear optimization (aka nonlinear programming) algorithms available in literature and implemented in software. In this document, I attempt to explain three common mathematical optimization algorithms for solving constrained nonlinear optimization problems. These algorithms are:

- The original method of moving asymptotes (Svanberg, 1987).
- The primal-dual interior point optimization algorithm available in IPOPT as explained in Wächter and Biegler (2006).
- The augmented Lagrangian family of algorithms.

These are all extremely popular optimization algorithms with one thing in common: there is no sufficiently general yet somewhat accessible explanation of these algorithms (to my knowledge) beyond their original papers or some large textbook which might be too far out of reach for the general user base of these algorithms. For the benefit of the readers who are not familiar with nonlinear programming (NLP) concepts and terminology, I also present a short primer on nonlinear programming concepts and formulations in this document.

All of the algorithms discussed here are available in the Nonconvex.jl[1] software package in the Julia programming language. I maintain Nonconvex.jl and this document might grow to become a technical companion to the software. For code examples, please refer to the documentation of Nonconvex.jl.

[1] `https://github.com/JuliaNonconvex/Nonconvex.jl`

Contents

1 Target audience and disclaimers

The target audience of this document is people with a bit of a mathematical background but with little to no constrained optimization background, and people who are (future) users or (future) developers of nonlinear optimization software. This document was originally part of my PhD thesis on topology optimization but I decided to present it separately here because then it might be useful to a larger audience. For this reason, there will be references to challenges in topology optimization problems which the general readers are free to ignore. For the benefit of the readers who are not familiar with nonlinear programming (NLP) concepts and terminology, I also present a short primer on nonlinear programming concepts and formulations. For readers familiar with basic concepts of NLP, feel free to jump straight to section 3.

In this document, I attempt to make my explanations:

1. Accessible but without sacrificing generality as much as I can, and
2. Correct and detailed but without too many derailing proofs and obscure mathematical language.

Correctness and accessibility are tough to combine in one document. This is my attempt.

Completeness is not necessarily a goal here if it comes at the cost of accessibility (in my subjective opinion) and if I find that there is a sufficiently clear reference I can point the interested readers to. Finally, in this document I focus on "what the algorithms are" more than the reasons "why they are what they are", although I also give reasons and proofs whenever the reasons/proofs can be concisely described (mostly) in words.

In this document, the terms *linear* and *affine* will be used inter-changeably in the context of classifying a function/constraint as either a linear/affine or a nonlinear function/constraint. Even though there is a distinction between the 2 terms, optimization literature commonly uses these 2 terms inter-changeably for the same purpose.

2 Background

2.1 Convex vs. non-convex

Optimization is a field filled with seemingly cryptic but fundamentally simple and useful concepts. Two such concepts are:

1. Convexity
2. Conic constraints

2.1.1 Convexity

A set of points $S = \{\boldsymbol{x}\}$ is said to be convex if every weighted average of some points inside the set is also inside the set. For sets in 2D and 3D, one can visually check for the convexity of a set S. If you can never draw a line of finite length with a start point and an end point inside the set S but part of the line lies outside the set, this set is convex. Your inability to draw such a line implies that no such line exists which implies that the set is convex.

However, set convexity generalizes to more dimensions than 2 or 3. More generally, one can formally define convexity as follows. A set S is convex if for every $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S$, $\alpha\boldsymbol{x}_1 + (1-\alpha)\boldsymbol{x}_2 \in S$ for all $0 < \alpha < 1$. This is a straightforward generalization of the line test in 2D/3D. It is also simple to show that the intersection of any number of convex sets is either the empty set or another convex set.

That's set convexity. Function convexity is a slightly different but related concept. A function $f(\boldsymbol{x})$ is called convex if the set of points satisfying the constraint $f(\boldsymbol{x}) \leq c$ is either an empty or a convex set for every $c \in \mathbb{R}$. Note that the set of such points doesn't have to be a compact set, that is it can extend to $\pm\infty$ along one or more directions. Showing that a function is convex can be done by proving the above property. This however is only one way to check if a function is convex. Another common check for twice differentiable functions is to check if the function's Hessian is positive semidefinite everywhere in the domain of the function. Identifying and building convex functions in a disciplined way is a well-studied field, often termed "disciplined convex programming" (DCP). For more on DCP, the readers are advised to check the excellent online resources available in `https://dcp.stanford.edu`.

In optimization, a constraint is said to be a convex constraint if the set of points satisfying this constraint is a convex set. One common way to construct a convex constraint is using a convex function $f(\boldsymbol{x})$. If f is a convex function, the following constraint is convex:

$$f(\boldsymbol{x}) \leq c \tag{1}$$

for any constant c, assuming there exists at least 1 point satisfying this constraint. Note that it has to be a $\leq$ constraint. Other common convex constraints are linear constraints:

$$f(\boldsymbol{x}) = 0 \tag{2}$$

where f is an affine function of $\boldsymbol{x}$, e.g. $f(\boldsymbol{x}) = \boldsymbol{b}^T\boldsymbol{x} + a$ for some constants $\boldsymbol{b}$ and a.

A constrained optimization problem with a convex *minimization* objective function and convex constraints is called a "convex program". Formulating decision problems as convex programs and solving them with convex optimization algorithms is often called "convex programming" which is a sub-field of "nonlinear programming". Convex programs are theoretically appealing because under mild assumptions, they have efficient algorithms that can find the so-called *global optimal solution*, which is a (not necessarily unique) feasible point $\boldsymbol{x}$ that absolutely minimizes the objective function, i.e. no other solution can do strictly better.

Note that it has to be a minimization problem. Maximizing a general convex function is NP-hard! If your objective $f(\boldsymbol{x})$ is a quantity you want to maximize, that's equivalent to minimizing $-f(\boldsymbol{x})$ instead. If $-f(\boldsymbol{x})$ is a convex function, f is known as a concave function. Therefore for a maximization optimization problem to be convex, the objective function must be concave and the constraints must be convex. This might be counter-intuitive that the maximization objective function must be *concave* for the optimization problem to be *convex*, but hey no one said optimization jargon was intuitive!

Similarly, the following constraint:

$$f(\boldsymbol{x}) \geq c \tag{3}$$

is a convex constraint if f was a concave function since this constraint is equivalent to:

$$-f(\boldsymbol{x}) \leq -c \tag{4}$$

where $-f(\boldsymbol{x})$ is convex which has the correct $\leq$ form we used above. Again, it's slightly counter-intuitive but just go with it.

2.1.2 Conic constraints

Among convex constraints, there are some special classes of constraints that are generally considered nicer than "generic" convex constraints. These constraints have a specific structure which can be exploited in more efficient algorithms. These are broadly known as "*structured constraints*". A simple class of structured constraints are the so-called linear constraints, e.g.:

$$\boldsymbol{b}^T\boldsymbol{x} + a = 0 \tag{5}$$

There are a number of algorithms that can handle linear constraints more efficiently than generic inequality ($\leq/\geq$) or equality ($=$) constraints. In fact, linear equality constraints are the only convex equality constraints! Any other equality constraint where the function in the constraint is not linear can never be convex. Linear constraints are therefore among those special classes of so-called *structured constraints*.

Another broad class of structured constraints is conic constraints. A conic constraint is a convex constraint whose feasible set is a special convex set known as a cone. Consider almost every child's dream, an infinitely tall ice cream cone. The set of points inside this cone comprise a convex set, recall the line drawing test. This however is only one type of cone. Just like with convex sets, mathematicians generalized the concept of a cone to multiple dimensions.

A convex set S is considered a cone if for every $\boldsymbol{x}$ inside the set S, $c\boldsymbol{x}$ was also inside the set S for all $c \geq 0$. Note how a cone must be a non-compact set since c is allowed to go to ∞. A nice way to visualize this is to think of the tip of the ice cream cone as the origin. Then extend an arrow from the origin to any point $\boldsymbol{x}$ inside the cone. Stretching or contracting this arrow is equivalent to changing the value of c, where the tip of the new arrow is $c\boldsymbol{x}$. For S to be a cone, the tip of every such stretched/contracted arrow must be inside the set S for any point $\boldsymbol{x} \in S$.

Considering the new generalized definition of a cone, one can come up with various types of cones beside the famous ice cream cone. The technical term of the ice cream cone is a second-order cone by the way (leave it to mathematicians to ruin a perfectly delicious *cone*cept). A structured conic constraint can be written as:

$$\boldsymbol{x} \in C \tag{6}$$

where C is a cone. There are various efficient algorithms that can handle conic constraints more efficiently than regular convex constraints. It is therefore often appealing to convert regular mathematical constraints that are known to be convex to their conic equivalents to make use of these efficient algorithms. For instance, the following convex constraint:

$$\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} \leq c \tag{7}$$

where $\boldsymbol{A}$ is a positive semi-definite matrix is equivalent to a second-order conic constraint. For more on conic constraints and transforming regular convex constraints to their conic structured equivalent, the readers are referred to the excellent online book titled: "MOSEK Modeling Cookbook" by MOSEK ApS (ApS, 2018).

2.1.3 Non-convexity

The vast majority of research literature on continuous mathematical optimization seems to be published on different variations and flavours of convex and conic programming, with a few exceptions. This is perhaps not surprising given the attractive theoretical properties of these classes of problems that enable mathematicians to reason about them and prove interesting, non-obvious results. There is however another extremely important class of continuous optimization problems that doesn't nearly get as much attention from mathematicians and is often treated as a "lost cause". These are the non-convex continuous optimization problems, i.e. problems where:

- The decision variables are continuous, and
- At least one constraint is not a convex constraint or the minimization (maximization) objective is not a convex (concave) function.

This is perhaps not surprising given that even some simple, non-convex, continuous optimization problems are known to be "NP-hard", where the existence of "efficient" algorithms to "solve" this class of problems is extremely unlikely by some definitions of "efficient" and "solve". This often means that efficiently solving an instance of this class requires many heuristics and is often almost as much an art as it is a science.

That said, "efficient" and "solve" are contextual terms. Often mathematicians consider the so-called "worst-case complexity" of a class of problems when talking about the computational efficiency of solving this class of problems. Worst-case complexity means that if one instance of this class is extremely difficult to "solve" for 100s of variables, we assume the whole class is extremely difficult to solve for 100s of variables. There are various classes of *problem difficulty*, aka "*complexity classes*", that are well studied in a field known as "*complexity theory*". For more on complexity theory and problem complexity analysis, the readers are referred to the excellent online book titled "Computational Complexity: A Modern Approach" by Arora and Barak (2007).

Worst-case complexity aside, if one chooses to define "efficient" and "solve" as finding a good (not necessarily the best) solution to a specific non-convex optimization problem with a reasonable number of variables and constraints in a reasonable amount of time, one can find many problem-algorithm pairings where a real-life problem can be efficiently solved, even though there would be little to no theoretical guarantees. This is generally the goal of non-convex optimization algorithms, to solve as many real-life, non-convex optimization problems as possible, as fast as possible.

In this document, a few such algorithms will be covered, those algorithms targeting generic non-convex optimization problems where no structure can be assumed. Special structured (e.g. conic) optimization algorithms on the other hand will not be covered here. That said, things aren't so black and white and there are some non-convex optimization algorithms that can efficiently handle special structured constraints when they exist while still handling generic non-convex objectives and/or constraints. Some examples of such algorithms are the Frank-Wolfe family of algorithms (Besançon et al., 2021) and the proximal family of algorithms (Parikh and Boyd, 2013). These however are also not going to be discussed in this document.

2.2 Formulations

There are generally a number of ways to formulate the same optimization or decision problem using mathematical functions and decision variables. For instance, consider the following inequality constrained optimization problem:

$$\begin{aligned} &\underset{\boldsymbol{x}}{\text{minimize}} && f_0(\boldsymbol{x}) \\ &\text{subject to} && \\ &f_i(\boldsymbol{x}) \leq 0 && \forall i = 1..I, \\ &l_j \leq x_j \leq u_j && \forall j = 1..V \end{aligned} \tag{8}$$

where I is the number of constraints, V is the number of decision variables, f_i are potentially nonlinear scalar-valued function of $\boldsymbol{x}$ and $\boldsymbol{l}$ and $\boldsymbol{u}$ are finite vectors of lower and upper bounds on the decision vector $\boldsymbol{x}$ respectively. Two formulations are considered equivalent for optimization purposes if every optimal solution of one formulation maps to an optimal solution in the other formulation. In the following sub-sections, 2 relevant re-formulation tricks will be presented. For more re-formulation techniques, the readers are referred to the excellent online book titled: "MOSEK Modeling Cookbook" by MOSEK ApS (ApS, 2018).

2.2.1 A linear objective is all you need

For instance, consider the alternative formulation:

$$\begin{aligned} &\underset{\boldsymbol{x}, c}{\text{minimize}} && c \\ &\text{subject to} \\ &f_0(\boldsymbol{x}) \leq c, \\ &f_i(\boldsymbol{x}) \leq 0 && \forall i = 1..I, \\ &l_j \leq x_j \leq u_j && \forall j = 1..V \end{aligned} \tag{9}$$

where we added a new decision variable c and a new constraint $f_0(\boldsymbol{x}) \leq c$ to the previous formulation and changed the objective to minimizing c which is a linear function of the decision variables.

It is clear that if $\boldsymbol{x}^*$ is an optimal solution of formulation 8, then $(\boldsymbol{x}, c) = (\boldsymbol{x}^*, f_0(\boldsymbol{x}^*))$ is optimal in formulation 9. This is because $f_0(\boldsymbol{x}^*)$ is the lowest value $f_0(\boldsymbol{x})$ can take for any feasible $\boldsymbol{x}$ according to the remaining inequality constraints, and $c = f_0(\boldsymbol{x}^*)$ is the lowest value the objective c can take without violating the constraint on c.

Conversely, if $(\boldsymbol{x}^*, c^*)$ is an optimal solution of formulation 9, $\boldsymbol{x}^*$ must be optimal in formulation 8. Since c shows up in only one constraint:

$$f_0(\boldsymbol{x}) \leq c \tag{10}$$

in formulation 9, for $(\boldsymbol{x}^*, c^*)$ to be optimal, c^* must be equal to $f_0(\boldsymbol{x}^*)$. And since this is lowest value c can take, it must be the lowest value $f_0(\boldsymbol{x})$ can take without violating any of the other constraints on $\boldsymbol{x}$, which in turn makes $\boldsymbol{x}^*$ an optimal solution to formulation 8. This completes the proof.

2.2.2 Slack variables

Another common formulation transformation is changing all the inequality constraints to equality constraints except the variable bounds. This can be done by introducing additional slack variables. Let $\boldsymbol{s}$ be a vector of so-called slack variables with length I, where s_i is the i^{th} element of the vector associated with constraint i for $i \in 1..I$. Formulation 8 can be re-written as:

$$\begin{aligned} &\underset{\boldsymbol{x}, \boldsymbol{s}}{\text{minimize}} && f_0(\boldsymbol{x}) \\ &\text{subject to} \\ &f_i(\boldsymbol{x}) + s_i = 0 && \forall i = 1..I, \\ &l_j \leq x_j \leq u_j && \forall j = 1..V, \\ &s_i \geq 0 && \forall i = 1..I \end{aligned} \tag{11}$$

Since $\boldsymbol{s}$ doesn't show up in the objective, in order to prove that formulations 8 and 11 are equivalent for optimization purposes, it suffices to show that every feasible solution $\boldsymbol{x}$ to formulation 8 can be mapped a feasible solution $(\boldsymbol{x}, \boldsymbol{s})$ in formulation 11 and vice versa.

It is simple to show that if $\boldsymbol{x}$ is a feasible solution in formulation 8, that $(\boldsymbol{x}, \boldsymbol{s})$ where $\boldsymbol{s}$ such that $s_i = -f_i(\boldsymbol{x})\ \forall i \in 1..I$ is a feasible solution in formulation 11. This is because for $\boldsymbol{x}$ to be feasible in formulation 8, $f_i(\boldsymbol{x})$ must be non-positive for all i which makes s_i non-negative. Conversely, if $(\boldsymbol{x}, \boldsymbol{s})$ is a feasible solution to formulation 11, $\boldsymbol{x}$ is clearly feasible in formulation 8 because s_i will be non-negative which implies that $f_i(\boldsymbol{x}) \leq 0$ for all i. This completes the proof.

2.2.3 Equality constraints and empty interiors

While the following equality constraint:

$$f(\boldsymbol{x}) = 0 \tag{12}$$

is in theory equivalent to the following 2 inequality constraints:

$$0 \leq f(\boldsymbol{x}) \leq 0 \tag{13}$$

this transformation does not change the shape or nature of the feasible domain. Not all NLP algorithms that can handle inequality constraints can also handle problems with 2 inequality constraints derived from an equality constraint like this. For instance, the convergence proof of the globally convergent method of moving asymptotes (MMA) algorithm (Svanberg, 2002) assumes that the feasible domain must have a non-empty interior. Equality constraints introduce an "empty interior". A domain $D \subseteq \mathbb{R}^V$ is said to have a non-empty interior if $\exists(\boldsymbol{x}_c \in D, r > 0)$ such that $\boldsymbol{x}_c + \boldsymbol{u} \in D$ for all $\boldsymbol{u}$ in $\{\boldsymbol{u} : \boldsymbol{u} \in \mathbb{R}^V \wedge ||\boldsymbol{u}||_2 \leq r\}$. Since equality constraints change the feasible domain to a lower dimensional manifold embedded in $\mathbb{R}^V$, there exist no such $(\boldsymbol{x}_c, r)$ in equality constrained problems. Therefore, the MMA algorithm will fail if the interior of the feasible domain is empty.

Linear equality constraints are generally an exception though. While technically still introducing an empty interior, most NLP algorithms can either natively handle linear equality constraints or the linear equality constrained NLP can be re-parameterized such that the interior of the feasible domain is no longer empty. The primal-dual interior point optimizer (IPOPT) discussed in this document and the augmented Lagrangian algorithm (Bertsekas, 1996) can handle linear and even nonlinear equality constraints natively. Sequential quadratic programming (SQP) methods (Boyd and Vandenberghe, 2009) can only handle linear equality constraints natively whereas nonlinear ones must get locally approximated by linear ones in an iterative process. MMA however doesn't handle either linear or nonlinear equality constraints natively. But it can be made to support linear ones with a simple nullspace re-parameterization trick.

Let the following be the linear equality constrained NLP with a nonlinear objective and inequality constraints:

$$\begin{aligned}
&\underset{\boldsymbol{x}}{\text{minimize}} && f_0(\boldsymbol{x}) \\
&\text{subject to} && \\
&f_i(\boldsymbol{x}) \leq 0 && \forall i = 1..I, \\
&l_j \leq x_j \leq u_j && \forall j = 1..V, \\
&\boldsymbol{A}\boldsymbol{x} = b &&
\end{aligned} \tag{14}$$

where $\boldsymbol{A}$ is a constant matrix of size $(M \times V)$ and $\boldsymbol{b}$ is a constant vector. Let $\boldsymbol{x}_0$ be an arbitrary point that satisfies the $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ constraint, e.g. $\boldsymbol{x}_0 = \boldsymbol{A}^+\boldsymbol{b}$ where $\boldsymbol{A}^+$ is the Moore-Penrose pseudoinverse of $\boldsymbol{A}$ (Golub and Loan, 1996). Let $\boldsymbol{N}$ be the nullspace matrix of $\boldsymbol{A}$ such that $\boldsymbol{A} \times \boldsymbol{N} = \boldsymbol{0}$, where $\boldsymbol{N}$ is of size $V \times U$. Every feasible solution $\boldsymbol{x}$ to the constraints $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ can now be written as:

$$\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{N}\boldsymbol{y} \tag{15}$$

for some $\boldsymbol{y} \in \mathbb{R}^U$, where $U < V$. Substituting for $\boldsymbol{x}$ in formulation 14, we get:

$$\begin{aligned} &\underset{\boldsymbol{y}}{\text{minimize}} \quad f_0(\boldsymbol{x}_0 + \boldsymbol{N}\boldsymbol{y}) \\ &\text{subject to} \\ &f_i(\boldsymbol{x}_0 + \boldsymbol{N}\boldsymbol{y}) \leq 0 \quad \forall i = 1..I, \\ &\boldsymbol{l} \leq \boldsymbol{x}_0 + \boldsymbol{N}\boldsymbol{y} \leq \boldsymbol{u} \end{aligned} \tag{16}$$

The re-parameterization essentially limits the feasible domain to the nullspace of the linear constraints since the "interior" is non-empty once we limit ourselves to the nullspace. A similar re-parameterization trick can also be used for some nonlinear manifolds, e.g. optimization on the surface of a hyper-sphere.

Beside the iterative linear approximation of nonlinear equality constraints, MMA can also be made to approximately support nonlinear equality constraints by relaxing it as follows:

$$-\epsilon \leq f(\boldsymbol{x}) \leq \epsilon \tag{17}$$

where $\epsilon > 0$ which creates a non-empty interior.

2.3 Regularity conditions

There are a number of optimization algorithms that can solve a general NLP without assuming convexity. Most NLP algorithms tend to converge to a locally optimal solution without global optimality guarantees and this suffices in many applications. For simplicity, assume all the inequality constraints have been converted to equality constraints using slack variables. Let the NLP with equality constraints be:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad f(\boldsymbol{x}) \tag{18a}$$

$$\text{subject to}$$

$$\boldsymbol{c}(\boldsymbol{x}) = \boldsymbol{0}, \tag{18b}$$

$$\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \tag{18c}$$

where $\boldsymbol{c}(\boldsymbol{x})$ is of length E. Additionally, let f and $\boldsymbol{c}$ be continuous and twice differentiable functions.

A point $\boldsymbol{x}$ is called a regular point if it satisfies one of the so-called constraint qualification conditions. Three common constraint qualifications for NLPs are:

1. Linear constraint qualification (LCQ): $\boldsymbol{c}(\boldsymbol{x})$ is an affine function.

2. Linear independence constraint qualification (LICQ): the rows of the Jacobian $\nabla \boldsymbol{c}(\boldsymbol{x})$ and the gradients of the active (i.e. satisfied at equality) bound constraints are linearly independent at $\boldsymbol{x}$.

3. Mangasarian-Fromovitz constraint qualification (MFCQ): the rows of $\nabla \boldsymbol{c}(\boldsymbol{x})$ are linearly independent at $\boldsymbol{x}$ and there exists a direction vector $\boldsymbol{d} \in \mathbb{R}^V$ where $d_i > 0$ if $x_i = l_i$, $d_i < 0$ if $x_i = u_i$, and $\nabla \boldsymbol{c}(\boldsymbol{x})^T \boldsymbol{d} = \boldsymbol{0}$.

Alternatively, if the problem is convex and the nonlinear inequality constraints are not converted to equality constraints, every feasible point is regular if $\exists \boldsymbol{x}$ such that all the inequality constraints are satisfied but not active (i.e. satisfied at a strict inequality) and all the linear equality constraints are satisfied. This condition is known as Slater's condition (Boyd and Vandenberghe, 2009).

2.4 Sufficient optimality conditions for regular points

Let:

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, z_+, z_-) = f(\boldsymbol{x}) + \boldsymbol{c}(\boldsymbol{x})^T \boldsymbol{\lambda} + (\boldsymbol{x} - \boldsymbol{u})^T z_+ - (\boldsymbol{x} - \boldsymbol{l})^T z_- \tag{19}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^E$ is the vector of Lagrangian multipliers of the nonlinear constraints, $z_- \in \mathbb{R}^V_+$ is the vector of Lagrangian multipliers of the $\geq$ bound constraints, and $z_+ \in \mathbb{R}^V_+$ is the vector of Lagrangian multipliers of the $\leq$ bound constraints.

If $\boldsymbol{x}$ is regular and is a local optimum of the NLP, then $\exists(\boldsymbol{\lambda}, z_+, z_-)$ such that:

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}, z_+, z_-) = \boldsymbol{0} \tag{20}$$
$$\boldsymbol{c}(\boldsymbol{x}) = 0 \tag{21}$$
$$\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \tag{22}$$
$$z_+ \geq \boldsymbol{0} \tag{23}$$
$$z_- \geq \boldsymbol{0} \tag{24}$$
$$(\boldsymbol{x} - \boldsymbol{u})^T z_+ = 0 \tag{25}$$
$$(\boldsymbol{x} - \boldsymbol{l})^T z_- = 0 \tag{26}$$
$$\nabla \boldsymbol{c}(\boldsymbol{x})^T \nabla^2_{\boldsymbol{xx}} L(\boldsymbol{x}, \boldsymbol{\lambda}, z_+, z_-) \nabla \boldsymbol{c}(\boldsymbol{x}) \succcurlyeq \boldsymbol{0} \tag{27}$$

These conditions are known as the KKT sufficient conditions for optimality and $\boldsymbol{x}$ would be called a KKT point (Boyd and Vandenberghe, 2009). Condition 20 is known as the stationarity condition which generalizes the 0 gradient condition for unconstrained NLPs. Conditions 21 and 22 are known as primal feasibility conditions. Conditions 23 and 24 are known as dual feasibility conditions. Finally, condition 27 is known as the second order KKT optimality condition which generalizes the second order optimality condition for unconstrained NLPs, where $\boldsymbol{A} \succcurlyeq \boldsymbol{0}$ when $\boldsymbol{A}$ is a matrix means that $\boldsymbol{A}$ must be positive semi-definite.

Note that not every local optimal solution to the NLP must be a regular point or by consequence a KKT point for that matter. Therefore, these conditions are not necessary conditions for optimality for general NLPs. However for convex problems,

if Slater's condition is satisfied, these conditions are both necessary and sufficient and the local/global optimum is guaranteed to be a regular and a KKT point (Boyd and Vandenberghe, 2009). All the optimization algorithms presented next seek to find a locally optimal KKT point.

3 Method of moving asymptotes

3.1 Separable convex approximation

Consider the following inequality constrained optimization problem:

$$\begin{aligned} &\underset{\boldsymbol{x}}{\text{minimize}} \quad f_0(\boldsymbol{x}) \\ &\text{subject to} \\ &f_i(\boldsymbol{x}) \leq 0 \qquad \forall i = 1..I, \\ &l_j \leq x_j \leq u_j \quad \forall j = 1..V \end{aligned} \tag{28}$$

where I is the number of inequality constraints, V is the number of decision variables, f_i is a potentially nonlinear scalar-valued function of $\boldsymbol{x}$ and $\boldsymbol{l}$ and $\boldsymbol{u}$ are finite vectors of lower and upper bounds on the decision vector $\boldsymbol{x}$ respectively.

The first MMA algorithm proposed by Svanberg (1987) relied on a separable convex approximation of the objective and constraint functions. A function $f(\boldsymbol{x})$ is linearly approximated with respect to either $t_{l,j}$ or $t_{u,j}$, where:

$$t_{l,j} = t_l(x_j; L_j) = \frac{1}{x_j - L_j} \tag{29}$$

$$t_{u,j} = t_u(x_j; U_j) = \frac{1}{U_j - x_j} \tag{30}$$

for all $j \in 1..V$ for some constants L_j and U_j, where $l_j \leq L_j < U_j \leq u_j$, and L_j and U_j are known as the asymptotes of the approximation. The choice of which function, $t_{l,j}$ or $t_{u,j}$, to approximate f with respect to, for each variable x_j, depends on the sign of the partial derivative $\frac{\partial f}{\partial x_j}$ such that the approximation is convex.

More specifically, let the MMA approximation of $f(\boldsymbol{x})$ around $\bar{\boldsymbol{x}}$ be:

$$\bar{f}(\boldsymbol{x}; \bar{\boldsymbol{x}}) = f(\bar{\boldsymbol{x}}) + \Sigma_j \bar{f}_j(x_j; \bar{\boldsymbol{x}}) \tag{31}$$

Let $\bar{f}_j(x_j; \bar{\boldsymbol{x}})$ be:

$$\bar{f}_j(x_j; \bar{\boldsymbol{x}}) = \begin{cases} \left(t_l(x_j; L_j) - t_l(\bar{x}_j; L_j)\right) \frac{\partial f}{\partial t_{l,j}}(\bar{\boldsymbol{x}}), & \frac{\partial f}{\partial x_j}(\bar{\boldsymbol{x}}) < 0 \\ \left(t_u(x_j; U_j) - t_u(\bar{x}_j; U_j)\right) \frac{\partial f}{\partial t_{u,j}}(\bar{\boldsymbol{x}}), & \frac{\partial f}{\partial x_j}(\bar{\boldsymbol{x}}) \geq 0 \end{cases} \tag{32}$$

where $\bar{x}_j$ is the j^{th} element of $\bar{\boldsymbol{x}}$ and $\frac{\partial f}{\partial t_{l,j}}$ and $\frac{\partial f}{\partial t_{u,j}}$ are:

$$\frac{\partial f}{\partial t_{l,j}} = \frac{\partial f}{\partial x_j} \Big/ \frac{dt_{l,j}}{dx_j} = -(x_j - L_j)^2 \frac{\partial f}{\partial x_j} \tag{33}$$

$$\frac{\partial f}{\partial t_{u,j}} = \frac{\partial f}{\partial x_j} \Big/ \frac{dt_{u,j}}{dx_j} = (U_j - x_j)^2 \frac{\partial f}{\partial x_j} \tag{34}$$

$\bar{f}_j(x_j;\bar{\boldsymbol{x}})$ can therefore be written as:

$$\bar{f}_j(x_j;\bar{\boldsymbol{x}}) = \begin{cases} -\Big(t_l(x_j;L_j) - t_l(\bar{x}_j;L_j)\Big)(\bar{x}_j - L_j)^2 \frac{\partial f}{\partial x_j}(\bar{\boldsymbol{x}}), & \frac{\partial f}{\partial x_j}(\bar{\boldsymbol{x}}) < 0 \\ \Big(t_u(x_j;U_j) - t_u(\bar{x}_j;U_j)\Big)(U_j - \bar{x}_j)^2 \frac{\partial f}{\partial x_j}(\bar{\boldsymbol{x}}), & \frac{\partial f}{\partial x_j}(\bar{\boldsymbol{x}}) \geq 0 \end{cases} \tag{35}$$

The signs of the gradient and Hessian of $\bar{f}(\boldsymbol{x};\bar{\boldsymbol{x}})$ therefore depend mostly on $t_l(x_j;L_j)$ and $t_u(x_j;U_j)$. Since the approximation is separable in $\boldsymbol{x}$, the Hessian of $\bar{f}(\boldsymbol{x};\bar{\boldsymbol{x}})$ wrt $\boldsymbol{x}$ is a diagonal matrix where the j^{th} diagonal element is:

$$\frac{\partial^2 \bar{f}}{\partial x_j^2} = \begin{cases} -\frac{d^2 t_{l,j}}{dx_j^2}(\bar{x}_j - L_j)^2 \frac{\partial f}{\partial x_j}(\bar{\boldsymbol{x}}), & \frac{\partial f}{\partial x_j}(\bar{\boldsymbol{x}}) < 0 \\ \frac{d^2 t_{u,j}}{dx_j^2}(U_j - \bar{x}_j)^2 \frac{\partial f}{\partial x_j}(\bar{\boldsymbol{x}}), & \frac{\partial f}{\partial x_j}(\bar{\boldsymbol{x}}) \geq 0 \end{cases} \tag{36}$$

Since $\frac{d^2 t_{l,j}}{dx_j^2}$ and $\frac{d^2 t_{u,j}}{dx_j^2}$ are both positive for all x_j, it is clear that the MMA approximation is always going to be convex.

In each iteration of the MMA algorithm, convex approximations of the objective and all the constraint functions are formed around the current solution and the approximate problem is solved to optimality while restricting the decision variables to be between the asymptotes $\boldsymbol{L}$ and $\boldsymbol{U}$ instead of the original bounds $\boldsymbol{l}$ and $\boldsymbol{u}$. This is similar to the trust region approach. Let the restricted convex approximation of the original nonlinear program be:

$$\begin{aligned} &\underset{\boldsymbol{x}}{\text{minimize}} \quad \bar{f}_0(\boldsymbol{x};\bar{\boldsymbol{x}}) \\ &\text{subject to} \\ &\bar{f}_i(\boldsymbol{x};\bar{\boldsymbol{x}}) \leq 0 \qquad \forall i = 1..I, \\ &\alpha_j \leq x_j \leq \beta_j \quad \forall j = 1..V \end{aligned}$$

where each decision variable x_j is restricted to be between $\alpha_j = max(l_j, L_j)$ and $\beta_j = min(u_j, U_j)$ and $\bar{f}_i(\boldsymbol{x};\bar{\boldsymbol{x}})$ is the MMA approximation of $f_i(\boldsymbol{x})$ around $\bar{\boldsymbol{x}}$. In order to form the approximation above, the gradient of the objective and the full Jacobian of the constraint functions need to be computed first. This typically limits the MMA algorithm's scalability in handling many constraints where the full Jacobian can be expensive to form, e.g. in stress-constrained topology optimization. However if only a few constraints exist, the MMA algorithm is usually quite robust when starting from a feasible solution. Most importantly is that once the approximate problem is formed, no more calls to the objective or constraint functions are required by the primal-dual algorithm to solve the approximate problem to optimality.

3.2 Efficient primal-dual algorithm

Once the approximation is formed, the separable nature of the MMA convex approximation then allows the approximate nonlinear program to be solved to optimality using an efficient primal-dual Lagrangian optimization algorithm (Svanberg, 1987). The dual

of the convex approximation problem above is the following lower bound constrained nested optimization problem:

$$\underset{\boldsymbol{\lambda} \geq \mathbf{0}}{\text{maximize}} \quad \min_{\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = r_0 + \sum_{j=1}^{V} \mathcal{L}_j(x_j, \boldsymbol{\lambda})$$

where

$$\mathcal{L}_j(x_j; \boldsymbol{\lambda}) = \frac{p_{0,j} + \sum_i \lambda_i p_{i,j}}{U_j - x_j} + \frac{q_{0,j} + \sum_i \lambda_i q_{i,j}}{x_j - L_j} \tag{37}$$

$$r_0 = f_0(\bar{\boldsymbol{x}}) - \sum_{j=1}^{V} \frac{p_{0,j}}{U_j - \bar{x}_j} + \frac{q_{0,j}}{\bar{x}_j - L_j} \tag{38}$$

$$p_{i,j} = \begin{cases} (U_j - \bar{x}_j)^2 \frac{\partial f_i}{\partial x_j}(\bar{\boldsymbol{x}}) & \frac{\partial f_i}{\partial x_j}(\bar{\boldsymbol{x}}) > 0 \\ 0 & \frac{\partial f_i}{\partial x_j}(\bar{\boldsymbol{x}}) \leq 0 \end{cases} \tag{39}$$

$$q_{i,j} = \begin{cases} 0 & \frac{\partial f_i}{\partial x_j}(\bar{\boldsymbol{x}}) \geq 0 \\ -(\bar{x}_j - L_j)^2 \frac{\partial f_i}{\partial x_j}(\bar{\boldsymbol{x}}) & \frac{\partial f_i}{\partial x_j}(\bar{\boldsymbol{x}}) < 0 \end{cases} \tag{40}$$

The primal optimal solution $\boldsymbol{x}^*$ of the convex approximation has an analytic form as a function of the dual solution $\boldsymbol{\lambda}$:

$$x_j^*(\boldsymbol{\lambda}) = \begin{cases} \alpha_j & \frac{\partial \mathcal{L}_j}{\partial x_j}(\alpha_j; \boldsymbol{\lambda}) \geq 0 \\ \beta_j & \frac{\partial \mathcal{L}_j}{\partial x_j}(\beta_j; \boldsymbol{\lambda}) \leq 0 \\ \frac{(p_{0,j} + \sum_i \lambda_i p_{i,j})^{1/2} L_j + (q_{0,j} + \sum_i \lambda_i q_{i,j})^{1/2} U_j}{(p_{0,j} + \sum_i \lambda_i p_{i,j})^{1/2} + (q_{0,j} + \sum_i \lambda_i q_{i,j})^{1/2}} & otherwise. \end{cases} \tag{41}$$

3.3 Algorithm strengths

Since the optimization of the approximate nonlinear program requires no calls to the objective or constraint functions, its performance is independent of the computational complexity of computing the objective value, its gradient, the constraint values and their Jacobian.

One of the biggest numerical challenges in topology optimization is that the scale of the objective and constraints can often be vastly different. For example in the volume constrained compliance minimization problem, the objective value scales up with the volume of the design and the Young's modulus while the volume fraction constraint does not. Moreover, the ℓ_∞-norm of the gradient of the volume fraction constraint scales down as the number of elements increases while that of the gradient of the compliance function does not. This usually results in problems where the objective and constraints span multiple orders of magnitude. This difference in scale usually means that more iterations are needed to converge to sufficiently large or sufficiently small elements of the dual solution $\boldsymbol{\lambda}$ that satisfies the first order KKT optimality conditions. More specifically in the MMA algorithm, this mostly translates to requiring more iterations to solve the approximate problem to optimality and usually not many more subproblems to be solved. However as mentioned earlier, the additional iterations needed to solve

the approximate subproblem do not require any additional calls to the objective or constraint functions. This is an especially attractive feature of the MMA algorithm in topology optimization since scaling issues are common and the computational time is usually dominated by the time it takes to compute the objective, the constraints and their gradients.

Another attractive feature of the MMA algorithm is the low sensitivity of the algorithm's performance to most of the parameters. In Nonconvex.jl, the lower bound constrained dual of the convex approximation is solved using a log-barrier method with the nonlinear conjugate gradient algorithm (Nocedal and Wright, 2006). The parameters of the subproblem optimizer generally have little effect on the performance of the MMA algorithm for the reason mentioned above. The main parameters which tend to affect the performance of the algorithm by changing the number of subproblems that need to be solved are: s_{init} used to specify the initial asymptotes, s_{incr} used to widen the asymptotes of each variable and s_{decr} used to tighten the asymptotes for each variable. These parameters are described in details by Svanberg (1987). Finally, a tolerance tol must be picked to terminate the algorithm.

4 Primal-dual interior point method

In this section, the primal-dual interior point optimizer (IPOPT) as described in Wächter and Biegler (2006) and implemented in the IPOPT software will be described. This algorithm and its implementation are well tested in practice. However, the main disadvantage of the IPOPT algorithm is its complexity and the large number of hyper-parameters as will be seen next.

4.1 Original problem

Consider the following equality constrained optimization problem:

$$\begin{aligned} &\underset{\boldsymbol{x}}{\text{minimize}} \quad f(\boldsymbol{x}) \\ &\text{subject to} \\ &\boldsymbol{c}(\boldsymbol{x}) = \boldsymbol{0}, \\ &\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \end{aligned} \tag{42}$$

where the length of $\boldsymbol{x}$ is V, E is the number of elements in the output of the vector-valued function $\boldsymbol{c}(\boldsymbol{x})$, f is a potentially nonlinear scalar-valued function of $\boldsymbol{x}$ and $\boldsymbol{l}$ and $\boldsymbol{u}$ are the lower and upper bounds on the variables. If there are inequality constraints, they can be converted to equality constraints by adding slack variables. Additionally, let $\mathcal{I}_{\boldsymbol{l}}$ be the set of variable indices with a finite lower bound and $\mathcal{I}_{\boldsymbol{u}}$ be the set of variable indices with a finite upper bound. Additionally, assume that the lower and upper bounds are not equal for any variable. If a variable's lower and upper bounds are equal, it can be fixed and removed from the optimization.

4.2 Barrier problem

In the IPOPT algorithm by Wächter and Biegler (2006), a so-called log barrier method (Boyd and Vandenberghe, 2009) is used to guarantee that $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$ remains satisfied at every intermediate solution if the initial solution is within the bounds. This is achieved using a so-called barrier function such as:

$$-\mu\Big(\sum_{i \in \mathcal{I}_{\boldsymbol{l}}} \log\left(x_i - l_i\right) + \sum_{i \in \mathcal{I}_{\boldsymbol{u}}} \log\left(u_i - x_i\right)\Big) \tag{43}$$

for some $\mu > 0$ which would go to ∞ if any of the decision variables approaches one of its finite bounds. This creates a barrier stopping the optimizer from every reaching the finite bound. It can be shown that if a decreasing geometric series of values μ are used that the solutions of the sub-problems will follow a so-called critical path converging to a KKT point in convex problems satisfying Slater's condition (Boyd and Vandenberghe, 2009). However, if the optimal solution is exactly on a boundary, the problem can become numerically unstable near the optimal solution. Therefore, the variable bounds are typically relaxed slightly. In particular, the lower bound l_i where $i \in \mathcal{I}_{\boldsymbol{l}}$ is relaxed by $tol \times \max\left(1, |l_i|\right)$, and the upper bound u_i where $i \in \mathcal{I}_{\boldsymbol{u}}$ is relaxed

by $tol \times \max(1, |u_i|)$. $\boldsymbol{l}$ and $\boldsymbol{u}$ will refer to the relaxed lower and upper bounds from now on.

One final catch in the barrier problem formulation is that if the set of optimal points to the original problem does not consist of isolated points, but contains an unbounded connected subspace, e.g. a line to ∞ or $-\infty$ for one or more variables, the log-barrier term may become $-\infty$. This is only possible if a variable is bounded from one side only where it's allowed to go to $\pm\infty$ from the other side. If this happens, the minimum objective value of the barrier problem becomes $-\infty$ even when the original problem's objective value is bounded. For this reason, the following additional term is added to the barrier objective:

$$\kappa_d \mu \sum_{i \in \mathcal{I}_{\boldsymbol{l}} \setminus \mathcal{I}_{\boldsymbol{u}}} \log(x_i - l_i) + \kappa_d \mu \sum_{i \in \mathcal{I}_{\boldsymbol{u}} \setminus \mathcal{I}_{\boldsymbol{l}}} \log(u_i - x_i) \tag{44}$$

for some small constant $\kappa_d \in (0, 1)$. This way, divergence of variables having only one bound is penalized. The effect of this additional term is reduced as μ decreases and can be shown to not affect the local convergence proof (Wächter and Biegler, 2006). The barrier sub-problem is therefore defined as:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \phi_\mu(\boldsymbol{x}) = f(\boldsymbol{x}) - \mu \sum_{i \in \mathcal{I}_{\boldsymbol{l}}} \log(x_i - l_i) - \mu \sum_{i \in \mathcal{I}_{\boldsymbol{u}}} \log(u_i - x_i) \\ & + \kappa_d \mu \sum_{i \in \mathcal{I}_{\boldsymbol{l}} \setminus \mathcal{I}_{\boldsymbol{u}}} \log(x_i - l_i) + \kappa_d \mu \sum_{i \in \mathcal{I}_{\boldsymbol{u}} \setminus \mathcal{I}_{\boldsymbol{l}}} \log(u_i - x_i) \\ \text{subject to} \quad & \\ & \boldsymbol{c}(\boldsymbol{x}) = \boldsymbol{0} \end{aligned} \tag{45}$$

The KKT stationarity condition is therefore:

$$\nabla f(\boldsymbol{x}) + \nabla \boldsymbol{c}(\boldsymbol{x})^T \boldsymbol{\lambda} - \boldsymbol{z}_{\boldsymbol{l}} - \boldsymbol{z}_{\boldsymbol{u}} = \boldsymbol{0} \tag{46}$$

where $\boldsymbol{\lambda}$ is the vector Lagrangian multipliers associated with the equality constraint $\boldsymbol{c}(\boldsymbol{x}) = \boldsymbol{0}$, $\boldsymbol{z}_{\boldsymbol{l}}$ is a vector whose i^{th} element is:

$$z_{l_i} = \begin{cases} \frac{\mu}{x_i - l_i} & i \in \mathcal{I}_{\boldsymbol{l}} \cap \mathcal{I}_{\boldsymbol{u}} \\ \frac{(1-\kappa_d)\mu}{x_i - l_i} & i \in \mathcal{I}_{\boldsymbol{l}} \setminus \mathcal{I}_{\boldsymbol{u}} \\ 0 & otherwise, \end{cases} \tag{47}$$

$\boldsymbol{z}_{\boldsymbol{u}}$ is a vector whose i^{th} element is:

$$z_{u_i} = \begin{cases} \frac{\mu}{u_i - x_i} & i \in \mathcal{I}_{\boldsymbol{l}} \cap \mathcal{I}_{\boldsymbol{u}} \\ \frac{(1-\kappa_d)\mu}{u_i - x_i} & i \in \mathcal{I}_{\boldsymbol{u}} \setminus \mathcal{I}_{\boldsymbol{l}} \\ 0 & otherwise, \end{cases} \tag{48}$$

$\mathbf{1}_{\boldsymbol{l}}$ is a vector whose i^{th} element is:

$$\begin{cases} 1 & i \in \mathcal{I}_{\boldsymbol{l}} \\ 0 & otherwise, \end{cases} \tag{49}$$

and $\mathbf{1}_{\boldsymbol{u}}$ is a vector whose i^{th} element is:

$$\begin{cases} 1 & i \in \mathcal{I}_{\boldsymbol{u}} \\ 0 & otherwise. \end{cases} \tag{50}$$

Additionally, let $\boldsymbol{Z_l}$ be the diagonal matrix whose diagonal is $\boldsymbol{z_l}$, $\boldsymbol{Z_u}$ be the diagonal matrix whose diagonal is $\boldsymbol{z_u}$, $\boldsymbol{I_l}$ be the diagonal matrix whose diagonal is $\mathbf{1}_{\boldsymbol{l}}$, $\boldsymbol{I_u}$ be the diagonal matrix whose diagonal is $\mathbf{1}_{\boldsymbol{u}}$, $\boldsymbol{X_l}$ be the diagonal matrix whose i^{th} diagonal element is:

$$\begin{cases} x_i - l_i & i \in \mathcal{I}_{\boldsymbol{l}} \cap \mathcal{I}_{\boldsymbol{u}} \\ \frac{x_i - l_i}{1-\kappa_d} & i \in \mathcal{I}_{\boldsymbol{l}} \setminus \mathcal{I}_{\boldsymbol{u}} \\ 0 & otherwise, \end{cases} \tag{51}$$

and $\boldsymbol{X_u}$ be the diagonal matrix whose i^{th} diagonal element is:

$$\begin{cases} u_i - x_i & i \in \mathcal{I}_{\boldsymbol{l}} \cap \mathcal{I}_{\boldsymbol{u}} \\ \frac{u_i - x_i}{1-\kappa_d} & i \in \mathcal{I}_{\boldsymbol{u}} \setminus \mathcal{I}_{\boldsymbol{l}} \\ 0 & otherwise. \end{cases} \tag{52}$$

The first order KKT sufficient conditions for optimality assuming the constraint qualifications are satisfied can therefore be written as:

$$\nabla f(\boldsymbol{x}) + \nabla \boldsymbol{c}(\boldsymbol{x})^T \boldsymbol{\lambda} - \boldsymbol{z_l} - \boldsymbol{z_u} = \mathbf{0} \tag{53}$$

$$\boldsymbol{c}(\boldsymbol{x}) = \mathbf{0} \tag{54}$$

$$\boldsymbol{X_l Z_l} \mathbf{1}_{\boldsymbol{l}} - \mu \mathbf{1}_{\boldsymbol{l}} = \mathbf{0} \tag{55}$$

$$\boldsymbol{X_u Z_u} \mathbf{1}_{\boldsymbol{u}} - \mu \mathbf{1}_{\boldsymbol{u}} = \mathbf{0} \tag{56}$$

$$\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \tag{57}$$

$$\boldsymbol{z_l} \geq \mathbf{0} \tag{58}$$

$$\boldsymbol{z_u} \geq \mathbf{0} \tag{59}$$

Conditions 55 and 56 ensure that the relationship between $\boldsymbol{X_l}$, $\boldsymbol{Z_l}$, $\boldsymbol{X_u}$ and $\boldsymbol{Z_u}$ is maintained according to the definitions of $\boldsymbol{z_l}$ and $\boldsymbol{z_u}$. Assume l_i and u_i are both finite for some index i. Given that μ is positive, condition 55 guarantees that $x_i - l_i$ and z_{l_i} will either be both positive or both negative. However even if we start from a value for $x_i > l_i$ and $z_{l_i} > 0$, during the intermediate line search steps, there is a non-zero chance that both $x_i - l_i$ and z_i may become negative simultaneously for some i, hence the need for the explicit non-negativity constraints on $\boldsymbol{z_l}$ and $\boldsymbol{z_u}$, and the bounds constraints on $\boldsymbol{x}$.

The primal-dual interior point optimizer presented in Wächter and Biegler (2006) computes an approximate solution to the barrier problem for a fixed value of μ then decreases μ and continues the solution of the next barrier problem from the approximate

solution of the previous one. The optimality error is defined as:

$$E_\mu(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{z_l}, \boldsymbol{z_u}) := \max\left\{ \frac{||\nabla f(\boldsymbol{x}) + \nabla \boldsymbol{c}(\boldsymbol{x})^T \boldsymbol{\lambda} - \boldsymbol{z_l} - \boldsymbol{z_u}||}{s_d}, ||\boldsymbol{c}(\boldsymbol{x})||_1, \frac{||\boldsymbol{X_l Z_l 1_l} - \mu \boldsymbol{1_l}||_\infty}{s_c}, \frac{||\boldsymbol{X_u Z_u 1_u} - \mu \boldsymbol{1_u}||_\infty}{s_c} \right\} \tag{60}$$

with scaling parameters $s_d, s_c \geq 1$ defined below. $E_0(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{z_l}, \boldsymbol{z_u})$ is the optimality error for the original problem. Let $(\tilde{\boldsymbol{x}}_*, \tilde{\boldsymbol{\lambda}}_*, \tilde{\boldsymbol{z}}_{\boldsymbol{l},*}, \tilde{\boldsymbol{z}}_{\boldsymbol{u},*})$ be the approximate solution of the barrier problem. The algorithm terminates if $E_0(\tilde{\boldsymbol{x}}_*, \tilde{\boldsymbol{\lambda}}_*, \tilde{\boldsymbol{z}}_{\boldsymbol{l},*}, \tilde{\boldsymbol{z}}_{\boldsymbol{u},*}) \leq tol$ for some tolerance tol.

It is possible that the magnitudes of $\boldsymbol{\lambda}$, $\boldsymbol{z_l}$ and $\boldsymbol{z_u}$ might become very large, e.g. if the gradients of the active constraints are nearly linearly dependent in a solution to the original problem. This is why the scaling factors s_d and s_c are used to make it easier to satisfy the terminating condition around these solutions. s_d and s_c are therefore chosen as:

$$s_d = \max\left\{ s_{max}, \frac{||\lambda||_1 + ||\boldsymbol{z_l}||_1 + ||\boldsymbol{z_u}||_1}{E + V} \right\} / s_{max} \tag{61}$$

$$s_c = \max\left\{ s_{max}, \frac{||\boldsymbol{z_l}||_1 + ||\boldsymbol{z_u}||_1}{V} \right\} / s_{max} \tag{62}$$

Using these scaling factors, a component of the optimality error will be scaled, whenever the average value of the multipliers becomes larger than a fixed number $s_{max} \geq 1$

Denoting with j the outer iteration counter of barrier sub-problems. Each barrier problem j is terminated when:

$$E_{\mu_{j-1}}(\tilde{\boldsymbol{x}}_{*,j}, \tilde{\boldsymbol{\lambda}}_{*,j}, \tilde{\boldsymbol{z}}_{\boldsymbol{l},*,j}, \tilde{\boldsymbol{z}}_{\boldsymbol{u},*,j}) \leq \kappa_\epsilon \mu_j \tag{63}$$

for a constant $\kappa_\epsilon > 0$. The new barrier parameter μ_j is then obtained using:

$$\mu_j = \max\left\{ \frac{tol}{10}, \min\left\{ \kappa_\mu \mu_{j-1}, \mu_{j-1}^{\theta_\mu} \right\} \right\} \tag{64}$$

with constants $\kappa_\mu \in (0, 1)$ and $\theta_\mu \in (1, 2)$. In this way, μ is eventually decreased at a superlinear rate. On the other hand, this update rule stops μ from becoming smaller than necessary given the desired tolerance tol.

4.3 Damped Newton's method

In order to solve the barrier problem for a given value $\mu = \mu_j$, damped Newton's method is applied to the primal-dual optimality conditions. Here we use k to denote the iteration counter for the inner iterations when solving the barrier problem. Given an iterate $(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{z}_{\boldsymbol{l},k}, \boldsymbol{z}_{\boldsymbol{u},k})$ with $\boldsymbol{l} < \boldsymbol{x}_k < \boldsymbol{u}$ and $\boldsymbol{z}_{\boldsymbol{l},k}, \boldsymbol{z}_{\boldsymbol{u},k} > \boldsymbol{0}$, some search directions

$(\boldsymbol{d}_k^{\boldsymbol{x}}, \boldsymbol{d}_k^{\boldsymbol{\lambda}}, \boldsymbol{d}_k^{\boldsymbol{z_l}}, \boldsymbol{d}_k^{\boldsymbol{z_u}})$ are obtained using the regularized linearization of the optimality conditions (excluding the inequality constraints):

$$\begin{bmatrix} \boldsymbol{W}_k + \delta_w \boldsymbol{I} & \boldsymbol{A}_k & -\boldsymbol{I_l} & -\boldsymbol{I_u} \\ \boldsymbol{A}_k^T & -\delta_c \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{Z}_{\boldsymbol{l},k} & \boldsymbol{0} & \boldsymbol{X}_{\boldsymbol{l},k} & \boldsymbol{0} \\ \boldsymbol{Z}_{\boldsymbol{u},k} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{X}_{\boldsymbol{u},k} \end{bmatrix} \begin{pmatrix} \boldsymbol{d}_k^{\boldsymbol{x}} \\ \boldsymbol{d}_k^{\boldsymbol{\lambda}} \\ \boldsymbol{d}_k^{\boldsymbol{z_l}} \\ \boldsymbol{d}_k^{\boldsymbol{z_u}} \end{pmatrix} = - \begin{pmatrix} \nabla f(\boldsymbol{x}_k) + \boldsymbol{A}_k^T \boldsymbol{\lambda}_k - \boldsymbol{z}_{\boldsymbol{l},k} - \boldsymbol{z}_{\boldsymbol{u},k} \\ \boldsymbol{c}(\boldsymbol{x}_k) \\ \boldsymbol{X}_{\boldsymbol{l},k} \boldsymbol{Z}_{\boldsymbol{l},k} \boldsymbol{1_l} - \mu_j \boldsymbol{1_l} \\ \boldsymbol{X}_{\boldsymbol{u},k} \boldsymbol{Z}_{\boldsymbol{u},k} \boldsymbol{1_u} - \mu_j \boldsymbol{1_u} \end{pmatrix} \tag{65}$$

where δ_w and δ_c are non-negative constants and $\boldsymbol{A}_k := \nabla \boldsymbol{c}(\boldsymbol{x}_k)^T$ and $\boldsymbol{W}_k := \nabla^2_{\boldsymbol{xx}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k)$ which is the Hessian of the Lagrangian function of the original problem, where:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) := f(x) + \boldsymbol{c}(\boldsymbol{x})^T \boldsymbol{\lambda} \tag{66}$$

Note that the Lagrangian terms from the bounds constraint $\boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{u}$ in the original problem can be ignored since their contribution to the Hessian is 0. When the Hessian of the Lagrangian is not available, an l-BFGS approximation (Nocedal and Wright, 2006) of the Hessian can be used instead. This changes the IPOPT algorithm from a second order algorithm to a first order one. And the Newton update becomes a quasi-Newton update.

When δ_w and δ_c are 0s, we get the linearization of the primal-dual first order KKT conditions. However, since the Hessian of the Lagrangian function isn't necessarily a positive definite matrix when the NLP is non-convex and in order to guarantee that the search directions obtained are descent directions (Wächter and Biegler, 2006), a positive value for δ_w is used. Additionally, if the gradients of the active constraints are (nearly) linearly dependent, the matrix on the LHS will be (nearly) singular even if the Hessian of the Lagrangian is positive definite. To stop the matrix from becoming singular in this case, a positive value for δ_c can be used. If the matrix is so ill-conditioned even with large values for δ_w and δ_c, the algorithm gives up on finding a search direction and attempts a feasibility restoration step, hoping that the matrix has better properties close to feasible points. For more on the feasibility restoration phase or on the heuristic used to choose δ_w and δ_c, the readers are referred to section 3.1 in Wächter and Biegler (2006).

Instead of solving the non-symmetric system of equations above, one can instead change the system as such:

$$\begin{bmatrix} \boldsymbol{W}_k + \delta_w \boldsymbol{I} & \boldsymbol{A}_k & -\boldsymbol{I_l} & -\boldsymbol{I_u} \\ \boldsymbol{A}_k^T & -\delta_c \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{X}^+_{\boldsymbol{l},k} \boldsymbol{Z}_{\boldsymbol{l},k} & \boldsymbol{0} & \boldsymbol{I_l} & \boldsymbol{0} \\ \boldsymbol{X}^+_{\boldsymbol{u},k} \boldsymbol{Z}_{\boldsymbol{u},k} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I_u} \end{bmatrix} \begin{pmatrix} \boldsymbol{d}_k^{\boldsymbol{x}} \\ \boldsymbol{d}_k^{\boldsymbol{\lambda}} \\ \boldsymbol{d}_k^{\boldsymbol{z_l}} \\ \boldsymbol{d}_k^{\boldsymbol{z_u}} \end{pmatrix} = - \begin{pmatrix} \nabla f(\boldsymbol{x}_k) + \boldsymbol{A}_k^T \boldsymbol{\lambda}_k - \boldsymbol{z}_{\boldsymbol{l},k} - \boldsymbol{z}_{\boldsymbol{u},k} \\ \boldsymbol{c}(\boldsymbol{x}_k) \\ \boldsymbol{z}_{\boldsymbol{l},k} - \mu_j \boldsymbol{X}^+_{\boldsymbol{l},k} \boldsymbol{1_l} \\ \boldsymbol{z}_{\boldsymbol{u},k} - \mu_j \boldsymbol{X}^+_{\boldsymbol{u},k} \boldsymbol{1_u} \end{pmatrix} \tag{67}$$

where $\boldsymbol{X}^+$ is the Moore-Penrose pseudoinverse of $\boldsymbol{X}$ (Golub and Loan, 1996). Adding the third and fourth equations to the first one, the third and fourth blocks of coefficients of the first equation will be eliminated.

$$\begin{bmatrix} \boldsymbol{W}_k + \delta_w \boldsymbol{I} + \boldsymbol{\Sigma}_k & \boldsymbol{A}_k & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{A}_k^T & -\delta_c \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{X}^+_{\boldsymbol{l},k} \boldsymbol{Z}_{\boldsymbol{l},k} & \boldsymbol{0} & \boldsymbol{I_l} & \boldsymbol{0} \\ \boldsymbol{X}^+_{\boldsymbol{u},k} \boldsymbol{Z}_{\boldsymbol{u},k} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I_u} \end{bmatrix} \begin{pmatrix} \boldsymbol{d}_k^{\boldsymbol{x}} \\ \boldsymbol{d}_k^{\boldsymbol{\lambda}} \\ \boldsymbol{d}_k^{\boldsymbol{z_l}} \\ \boldsymbol{d}_k^{\boldsymbol{z_u}} \end{pmatrix} = - \begin{pmatrix} \nabla f(\boldsymbol{x}_k) + \boldsymbol{A}_k^T \boldsymbol{\lambda}_k - \mu_j \boldsymbol{X}^+_{\boldsymbol{l},k} \boldsymbol{1_l} - \mu_j \boldsymbol{X}^+_{\boldsymbol{u},k} \boldsymbol{1_u} \\ \boldsymbol{c}(\boldsymbol{x}_k) \\ \boldsymbol{z}_{\boldsymbol{l},k} - \mu_j \boldsymbol{X}^+_{\boldsymbol{l},k} \boldsymbol{1_l} \\ \boldsymbol{z}_{\boldsymbol{u},k} - \mu_j \boldsymbol{X}^+_{\boldsymbol{u},k} \boldsymbol{1_u} \end{pmatrix} \tag{68}$$

where $\boldsymbol{\Sigma}_k = \boldsymbol{X}^+_{\boldsymbol{l},k}\boldsymbol{Z}_{\boldsymbol{l},k} + \boldsymbol{X}^+_{\boldsymbol{u},k}\boldsymbol{Z}_{\boldsymbol{u},k}$. Therefore, one can now solve for $\boldsymbol{d}^{\boldsymbol{x}}_k$ and $\boldsymbol{d}^{\boldsymbol{\lambda}}_k$ by solving the following symmetric linear system:

$$\begin{bmatrix} \boldsymbol{W}_k + \delta_w \boldsymbol{I} + \boldsymbol{\Sigma}_k & \boldsymbol{A}_k \\ \boldsymbol{A}^T_k & -\delta_c \boldsymbol{I} \end{bmatrix} \begin{pmatrix} \boldsymbol{d}^{\boldsymbol{x}}_k \\ \boldsymbol{d}^{\boldsymbol{\lambda}}_k \end{pmatrix} = -\begin{pmatrix} \nabla f(\boldsymbol{x}_k) + \boldsymbol{A}^T_k \boldsymbol{\lambda}_k - \mu_j \boldsymbol{X}^+_{\boldsymbol{l},k}\mathbf{1}_{\boldsymbol{l}} - \mu_j \boldsymbol{X}^+_{\boldsymbol{u},k}\mathbf{1}_{\boldsymbol{u}} \\ \boldsymbol{c}(\boldsymbol{x}_k) \end{pmatrix} \tag{69}$$

then use the value of $\boldsymbol{d}^{\boldsymbol{x}}_k$ to find $\boldsymbol{d}^{\boldsymbol{z_l}}_k$ and $\boldsymbol{d}^{\boldsymbol{z_u}}_k$ using:

$$\boldsymbol{d}^{\boldsymbol{z_l}}_k = -\boldsymbol{z}_{\boldsymbol{l},k} + \mu_j \boldsymbol{X}^+_{\boldsymbol{l},k}\mathbf{1}_{\boldsymbol{l}} - \boldsymbol{X}^+_{\boldsymbol{l},k}\boldsymbol{Z}_{\boldsymbol{l},k}\boldsymbol{d}^{\boldsymbol{x}}_k \tag{70}$$

$$\boldsymbol{d}^{\boldsymbol{z_u}}_k = -\boldsymbol{z}_{\boldsymbol{u},k} + \mu_j \boldsymbol{X}^+_{\boldsymbol{u},k}\mathbf{1}_{\boldsymbol{u}} - \boldsymbol{X}^+_{\boldsymbol{u},k}\boldsymbol{Z}_{\boldsymbol{u},k}\boldsymbol{d}^{\boldsymbol{x}}_k \tag{71}$$

Note that the elements in $\boldsymbol{z_l}$ that are always 0 because their indices correspond to decision variables $\boldsymbol{x}$ unbounded from below can be ignored in implementation. The same can be done for the elements of $\boldsymbol{z_u}$ that are always 0 because their indices correspond to decision variables $\boldsymbol{x}$ unbounded from above.

4.4 Line search and filter

4.4.1 Maximum step size

Having computed the solution to the search directions linear system of equations to obtain $(\boldsymbol{d}^{\boldsymbol{x}}_k, \boldsymbol{d}^{\boldsymbol{\lambda}}_k, \boldsymbol{d}^{\boldsymbol{z_l}}_k, \boldsymbol{d}^{\boldsymbol{z_u}}_k)$, now two step sizes $\alpha_k, \alpha^z_k \in (0, 1]$ have to be determined in order to obtain the next iterate using:

$$\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \alpha_k \boldsymbol{d}^{\boldsymbol{x}}_k \tag{72}$$

$$\boldsymbol{\lambda}_{k+1} := \boldsymbol{\lambda}_k + \alpha_k \boldsymbol{d}^{\boldsymbol{\lambda}}_k \tag{73}$$

$$\boldsymbol{z}_{\boldsymbol{l},k+1} := \boldsymbol{z}_{\boldsymbol{l},k} + \alpha^z_k \boldsymbol{d}^{\boldsymbol{z_l}}_k \tag{74}$$

$$\boldsymbol{z}_{\boldsymbol{u},k+1} := \boldsymbol{z}_{\boldsymbol{u},k} + \alpha^z_k \boldsymbol{d}^{\boldsymbol{z_u}}_k \tag{75}$$

When the step sizes are chosen, the bounds on $\boldsymbol{x}$, $\boldsymbol{z_l}$ and $\boldsymbol{z_u}$ are enforced. Let $\tau_j \in (0, 1)$ be the so-called fraction-to-the-boundary parameter given by:

$$\tau_j = \max\{\tau_{min}, 1 - \mu_j\} \tag{76}$$

where $\tau_{min} \in (0, 1)$ is its minimum value. The step sizes are then chosen using the following fraction-to-the-boundary rule:

$$\alpha^{max}_k := \max\left\{\alpha \in (0, 1] : \left(\tau \boldsymbol{l} + (1 - \tau_j)\boldsymbol{x}_k\right) \leq \left(\boldsymbol{x}_k + \alpha \boldsymbol{d}^{\boldsymbol{x}}_k\right) \leq \left(\tau \boldsymbol{u} + (1 - \tau_j)\boldsymbol{x}_k\right)\right\} \tag{77}$$

$$\alpha^z_k := \max\left\{\alpha \in (0, 1] : \left(\boldsymbol{z}_{\boldsymbol{l},k} + \alpha \boldsymbol{d}^{\boldsymbol{z_l}}_k \geq (1 - \tau_j)\boldsymbol{z}_{\boldsymbol{l},k}\right) \wedge \left(\boldsymbol{z}_{\boldsymbol{u},k} + \alpha \boldsymbol{d}^{\boldsymbol{z_u}}_k \geq (1 - \tau_j)\boldsymbol{z}_{\boldsymbol{u},k}\right)\right\} \tag{78}$$

where the actual step size $\alpha_k \in (0, \alpha^{max}_k]$ is determined using a backtracking line search procedure exploring a decreasing sequence of trial step sizes:

$$\alpha_{k,l} = 2^{-l}\alpha^{max}_k \tag{79}$$

with $l = 0, 1, 2, \ldots$.

4.4.2 Line search

Choosing the step sizes is done using a so-called line search filter method. During line search, a step size is considered acceptable if it leads to an acceptable reduction in the objective value of the barrier problem $\phi_\mu(\boldsymbol{x})$, and/or the constraint violation $\theta(\boldsymbol{x}) = ||\boldsymbol{c}(\boldsymbol{x})||_1$ with a certain emphasis on the latter quantity. Let $\boldsymbol{x}_k(\alpha_{k,l})$ be:

$$\boldsymbol{x}_k(\alpha_{k,l}) := \boldsymbol{x}_k + \alpha_{k,l}\boldsymbol{d}_k^{\boldsymbol{x}} \tag{80}$$

There are 2 acceptable criteria used to accept a trial step size. The following conditions are called the "switching conditions":

$$\nabla\phi_{\mu_j}(\boldsymbol{x}_k)^T\boldsymbol{d}_k^{\boldsymbol{x}} < 0 \tag{81}$$

$$\alpha_{k,l}[-\nabla\phi_{\mu_j}(\boldsymbol{x}_k)^T\boldsymbol{d}_k^{\boldsymbol{x}}]^{S_\phi} > \delta[\theta(\boldsymbol{x}_k)]^{S_\theta} \tag{82}$$

for some constants $\delta > 0$, $S_\phi > 1$ and $S_\theta \geq 1$. If $\theta(\boldsymbol{x}_k) \leq \theta^{min}$, for some constant $\theta^{min} \in (0, \infty)$, and the following so-called "switching conditions" are satisfied, a step size $\alpha_{k,l}$ is considered acceptable if the following so-called Armijo condition is satisfied:

$$\phi_{\mu_j}(\boldsymbol{x}_k(\alpha_{k,l})) \leq \phi_{\mu_j}(\boldsymbol{x}_k) + \eta_\phi\alpha_{k,l}\nabla\phi_{\mu_j}(\boldsymbol{x}_k)^T\boldsymbol{d}_k^{\boldsymbol{x}} \tag{83}$$

for some constant $\eta_\theta \in (0, 0.5)$ and $(\theta(\boldsymbol{x}_k(\alpha_{k,l})), \phi_{\mu_j}(\boldsymbol{x}_k(\alpha_{k,l}))) \notin \mathcal{F}_k$ where $\mathcal{F}_k$ is a filter set (defined below) used to reject a trial solution if its constraint violation is too high or if it has a close barrier objective value and constraint violation value as certain previous iterates $\boldsymbol{x}_k$. This filter method ensures that the algorithm cannot cycle between 2 points that alternately decrease the constraint violation and barrier objective value. If the trial step is rejected, a so-called second order correction (SOC) step is performed. The SOC step is explained below.

If either $\theta(\boldsymbol{x}_k) > \theta^{min}$ or the switching conditions are violated, a step size $\alpha_{k,l}$ is considered acceptable if either:

$$\theta(\boldsymbol{x}_k(\alpha_{k,l})) \leq (1 - \gamma_\theta)\theta(\boldsymbol{x}_k) \tag{84}$$

$$(\theta(\boldsymbol{x}_k(\alpha_{k,l})), \phi_{\mu_j}(\boldsymbol{x}_k(\alpha_{k,l}))) \notin \mathcal{F}_k \tag{85}$$

or

$$\phi_{\mu_j}(\boldsymbol{x}_k(\alpha_{k,l})) \leq \phi_{\mu_j}(\boldsymbol{x}_k) - \gamma_\phi\theta(\boldsymbol{x}_k) \tag{86}$$

$$(\theta(\boldsymbol{x}_k(\alpha_{k,l})), \phi_{\mu_j}(\boldsymbol{x}_k(\alpha_{k,l}))) \notin \mathcal{F}_k \tag{87}$$

is satisfied for some constants $\gamma_\theta, \gamma_\phi \in (0, 1)$. If the trial step is rejected, the SOC step is performed.

4.4.3 Filter set

The filter set $\mathcal{F}_k$ is initialized as:

$$\mathcal{F}_0 = \{(\theta, \phi) \in \mathbb{R}^2 : \theta \geq \theta^{max}\} \tag{88}$$

Then after every iteration k if an accepted trial point $\boldsymbol{x}_k$ doesn't satisfy either the switching conditions or the Armijo rule, the filter set is updated as:

$$\mathcal{F}_{k+1} = \mathcal{F}_k \cup \left\{ (\theta, \phi) \in \mathbb{R}^2 : \left(\theta \geq (1-\gamma_\theta)\theta(\boldsymbol{x}_k)\right) \wedge \left(\phi \geq \phi_{\mu_j}(\boldsymbol{x}_k) - \gamma_\phi \theta(\boldsymbol{x}_k)\right) \right\} \tag{89}$$

Every time the barrier parameter μ is decreased in the outer iterations of the algorithm, the filter set is reset to its definition at $k = 0$.

If the backtracking procedure (SOC steps included) fails to find an acceptable trial step $\alpha_{k,l} \geq \alpha_k^{min}$, where:

$$\alpha_k^{min} := \gamma_\alpha \times \begin{cases} \min\left\{\gamma_\theta, \frac{\gamma_\theta \theta(\boldsymbol{x}_k)}{-\nabla\phi_{\mu_j}(\boldsymbol{x}_k)^T \boldsymbol{d}_k^{\boldsymbol{x}}}, \frac{\delta[\theta(\boldsymbol{x}_k)^{S_\theta}]}{[-\nabla\phi_{\mu_j}(\boldsymbol{x}_k)^T \boldsymbol{d}_k^{\boldsymbol{x}}]^{S_\phi}}\right\} & \left(\nabla\phi_{\mu_j}(\boldsymbol{x}_k)^T \boldsymbol{d}_k^{\boldsymbol{x}} < 0\right) \wedge \left(\theta(\boldsymbol{x}_k) \leq \theta^{min}\right) \\ \min\left\{\gamma_\theta, \frac{\gamma_\phi \theta(\boldsymbol{x}_k)}{-\nabla\phi_{\mu_j}(\boldsymbol{x}_k)^T \boldsymbol{d}_k^{\boldsymbol{x}}}\right\} & \left(\nabla\phi_{\mu_j}(\boldsymbol{x}_k)^T \boldsymbol{d}_k^{\boldsymbol{x}} < 0\right) \wedge \left(\theta(\boldsymbol{x}_k) > \theta^{min}\right) \\ \gamma_\theta & otherwise, \end{cases} \tag{90}$$

for some safety factor $\gamma_\alpha \in (0, 1]$, the algorithm reverts to a feasibility restoration phase. For details on the feasibility restoration phase, please refer to section 3.3 in Wächter and Biegler (2006).

4.4.4 Second order correction

If a trial step $\alpha_{k,l}$ is rejected for some iteration l in the backtracking procedure and $\theta(\boldsymbol{x}_k(\alpha_{k,l})) \geq \theta(\boldsymbol{x}_k)$, before moving on to the next trial, an SOC step is performed. The SOC step aims to reduce the infeasibility by applying an additional Newton-type step for the constraints at the point $\boldsymbol{x}_k + \tilde{\boldsymbol{d}}_k^{\boldsymbol{x}}$ where $\tilde{\boldsymbol{d}}_k^{\boldsymbol{x}} = \alpha_{k,l} \boldsymbol{d}_k^{\boldsymbol{x}}$ using the Jacobian $\boldsymbol{A}_k^T$ at $\boldsymbol{x}_k$ to obtain a direction update $\boldsymbol{d}_k^{\boldsymbol{x},soc}$ by solving:

$$\boldsymbol{A}_k^T \boldsymbol{d}_k^{\boldsymbol{x},soc} + \boldsymbol{c}(\boldsymbol{x}_k + \alpha_{k,l} \boldsymbol{d}_k^{\boldsymbol{x}}) = \boldsymbol{0} \tag{91}$$

The new corrected search direction is then obtained from:

$$\boldsymbol{d}_k^{\boldsymbol{x},cor} = \alpha_{k,l} \boldsymbol{d}_k^{\boldsymbol{x}} + \boldsymbol{d}_k^{\boldsymbol{x},soc} \tag{92}$$

The details of efficiently and carefully calculating $\boldsymbol{d}_k^{\boldsymbol{x},soc}$ can be found in section 2.4 in Wächter and Biegler (2006). Once the corrected search direction $\boldsymbol{d}_k^{\boldsymbol{x},cor}$ has been computed, we again apply the fraction-to-the-boundary rule:

$$\alpha_k^{soc} := \max\left\{\alpha \in (0, 1] : \left(\tau \boldsymbol{l} + (1-\tau_j)\boldsymbol{x}_k\right) \leq \left(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k^{\boldsymbol{x},cor}\right) \leq \left(\tau \boldsymbol{u} + (1-\tau_j)\boldsymbol{x}_k\right)\right\} \tag{93}$$

and check if the resulting trial point:

$$\boldsymbol{x}_k^{soc} := \boldsymbol{x}_k + \alpha_k^{soc} \boldsymbol{d}_k^{\boldsymbol{x},cor} \tag{94}$$

is acceptable to the filter and satisfies the acceptance criteria set earlier replacing $\boldsymbol{x}(\alpha_{k,l})$ with $\boldsymbol{x}_k^{soc}$ while keeping $\boldsymbol{d}_k^{\boldsymbol{x}}$ as-is. If the new iterate is still rejected, the correction step is repeated (replacing $\alpha_{k,l}$ with α_k^{soc}, and $\boldsymbol{d}_k^{\boldsymbol{x}}$ with $\boldsymbol{d}_k^{\boldsymbol{x},cor}$ in the SOC step) unless the correction step has not decreased the constraint violation by a fraction $\kappa_{soc} \in (0, 1)$ or a maximum number p^{max} of SOC steps has been performed. In that case, the original search direction $\boldsymbol{d}_k^{\boldsymbol{x}}$ is reverted to and the regular backtracking line search is resumed with a shorter step size.

4.4.5 Accelerating heuristics

When multiple trial steps are rejected because the filter set is too strict or the progress made in the objective or constraint violation is insufficient, 2 accelerating heuristics are further employed to relax the acceptance criteria under some conditions. More on this can be found in section 3.2 in Wächter and Biegler (2006).

4.4.6 Clamping

Finally after taking a step successfully, each variable z_{l_i} in $\boldsymbol{z_l}$ and each variable z_{u_i} in $\boldsymbol{z_u}$ then get clamped to an interval to avoid their extreme deviation:

$$z_{l_i} \in \begin{cases} \left[\frac{\mu_j}{\kappa_\Sigma (x_i - l_i)}, \frac{\kappa_\Sigma \mu_j}{(x_i - l_i)}\right] & i \in \mathcal{I}_{\boldsymbol{l}} \cap \mathcal{I}_{\boldsymbol{u}} \\ \left[\frac{(1-\kappa_d)\mu_j}{\kappa_\Sigma (x_i - l_i)}, \frac{\kappa_\Sigma (1-\kappa_d)\mu_j}{x_i - l_i}\right] & i \in \mathcal{I}_{\boldsymbol{l}} \setminus \mathcal{I}_{\boldsymbol{u}} \\ [0, 0] & otherwise, \end{cases} \tag{95}$$

$$z_{u_i} \in \begin{cases} \left[\frac{\mu_j}{\kappa_\Sigma (u_i - x_i)}, \frac{\kappa_\Sigma \mu_j}{u_i - x_i}\right] & i \in \mathcal{I}_{\boldsymbol{l}} \cap \mathcal{I}_{\boldsymbol{u}} \\ \left[\frac{(1-\kappa_d)\mu_j}{\kappa_\Sigma (u_i - x_i)}, \frac{\kappa_\Sigma (1-\kappa_d)\mu_j}{u_i - x_i}\right] & i \in \mathcal{I}_{\boldsymbol{u}} \setminus \mathcal{I}_{\boldsymbol{l}} \\ [0, 0] & otherwise. \end{cases} \tag{96}$$

This clamping step creates a safeguard needed for the global convergence proof of the algorithm (Wächter and Biegler, 2006). A large value for $\kappa_{\boldsymbol{\Sigma}} = 10^{10}$ is used to only minimally interfere with the Newton or quasi-Newton update.

5 Augmented Lagrangian algorithm

Both the MMA and IPOPT algorithms discussed previously require the full Jacobian of the constraints. This can be computationally prohibitive in some applications. In some cases, computing the Jacobian is much more expensive than calling the operator $\boldsymbol{v} \rightarrow \boldsymbol{J}'\boldsymbol{v}$ where $\boldsymbol{J}$ is the Jacobian of the constraints and $\boldsymbol{v}$ is a vector. The first order augmented Lagrangian algorithm only requires this operator and not the full Jacobian matrix.

Consider the following inequality constrained optimization problem:

$$\begin{aligned} &\underset{\boldsymbol{x}}{\text{minimize}} && f_0(\boldsymbol{x}) \\ &\text{subject to} && \\ &f_i(\boldsymbol{x}) \leq 0 && \forall i = 1..I, \\ &f_i(\boldsymbol{x}) = 0 && \forall i = I+1..I+E, \\ &l_j \leq x_j \leq u_j && \forall j = 1..V \end{aligned} \tag{97}$$

where I is the number of inequality constraints, E is the number of equality constraints and V is the number of decision variables, f_i is a potentially nonlinear scalar-valued function of $\boldsymbol{x}$ and $\boldsymbol{l}$ and $\boldsymbol{u}$ are finite vectors of lower and upper bounds on the decision vector $\boldsymbol{x}$ respectively.

The augmented Lagrangian algorithm (AugLag) is not a single algorithm but a family of algorithms in which the main idea is to reduce the above minimization problem to the following max-min formulation:

$$\underset{\boldsymbol{\lambda} \in \boldsymbol{Y}_\lambda}{\text{maximize}} \quad \min_{\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}} \left\{ \mathcal{L}_c(\boldsymbol{x}, \boldsymbol{\lambda}) = f_0(\boldsymbol{x}) + \sum_{i=1}^{I+E} \lambda_i f_i(\boldsymbol{x}) + c \sum_{i=1}^{I} \max\{f_i(\boldsymbol{x}), 0\}^2 + c \sum_{i=I+1}^{I+E} |f_i(\boldsymbol{x})|^2 \right\} \tag{98}$$

for some constant $c \geq 0$, where $\boldsymbol{Y}_\lambda = \{\boldsymbol{\lambda} : \lambda_i \geq 0 \, \forall i \in [1, I], \lambda_i \in \mathbb{R} \, \forall i \in [I+1, I+E]\}$ and $\mathcal{L}_c$ is known as the augmented Lagrangian function. Assume all the functions used are continuous and twice differentiable.

Note how the inequality and equality constraints were relaxed and added to the objective but the bounds constraints were not. That is the inner minimization still needs to respect the bounds constraints. The choice of which constraints to relax and which to handle directly, as well as which optimizer to use for the inner optimization problem and which one to use for the outer problem can lead to different variants of the augmented Lagrangian family of algorithms. The optimal value of the augmented Lagrangian function subject to some relaxed constraints is known to be always less than or equal to the optimal value of the original optimization problem. It is also known that the outer maximization problem is always concave (i.e. tractable) even if the objective and constraint functions were non-convex (Bertsekas, 1996).

It is known (Bertsekas, 1996) that if:

1. The original problem has an isolated set of local minima $\boldsymbol{X}^*$ which is compact,
2. The constant c is increased in a sequence $\{c_k\}$ such that $0 < c_k < c_{k+1}$, and

3. The Lagrangian multipliers $\boldsymbol{\lambda}$ follow an arbitrary bounded sequence $\{\boldsymbol{\lambda}_k\}$ where $\lambda_{k,i}$ is the i^{th} element of $\boldsymbol{\lambda}_k$

then there exists a sub-sequence of points $\{\boldsymbol{x}_k\}_K$ converging to a point $\boldsymbol{x}^* \in \boldsymbol{X}^*$ such that $\boldsymbol{x_k}$ is a local minimum for the following augmented Lagrangian sub-problem:

$$\underset{\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}}{\text{minimize}} \quad \left\{ \mathcal{L}_{c_k}(\boldsymbol{x}, \boldsymbol{\lambda}_k) = f_0(\boldsymbol{x}) + \sum_{i=1}^{I+E} \lambda_{k,i} f_i(\boldsymbol{x}) + c_k \sum_{i=1}^{I} \max\{f_i(\boldsymbol{x}), 0\}^2 + c_k \sum_{i=I+1}^{I+E} f_i(\boldsymbol{x})^2 \right\} \tag{99}$$

If additionally each $\boldsymbol{x}_k$ is a global minimum of the augmented Lagrangian sub-problem with c_k and $\boldsymbol{\lambda}_k$, and the feasible set is compact, e.g. $\boldsymbol{l}$ and $\boldsymbol{u}$ are finite, then every limit point of the sequence $\{\boldsymbol{x}_k\}$ is a global minimum of the original problem. Note that it is possible for the original problem to have a (unique) global minimum but the augmented Lagrangian sub-problem is unbounded from below. This is a weakness of the AugLag algorithm that needs to be addressed in implementation. However if the feasible domain is compact, this weakness is eliminated.

If additionally the sub-sequence $\{\boldsymbol{x}_k\}_K$ converges to a point $\boldsymbol{x}^* \in \boldsymbol{X}^*$ that satisfies one of the constraint qualification conditions, then the sequence $\{\text{proj}_{Y_{\boldsymbol{\lambda}}}(\boldsymbol{\lambda}_k + c_k \boldsymbol{G}_k)\}$ converges to a point $\boldsymbol{\lambda}^*$ such that $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ satisfy the first order sufficient KKT optimality conditions, where $\boldsymbol{G}_k$ is a vector of length $I + E$ and whose i^{th} element is $f_i(\boldsymbol{x}_k)$ (Bertsekas, 1996). Another weakness of the AugLag algorithm is that the sub-problem's optimizer may converge to a point that doesn't satisfy any of the constraint qualification conditions for the original problem. One case where this is guaranteed to happen is if the original problem is infeasible so the quadratic term dominates the augmented Lagrangian function as $c \to \infty$ and every (approximate) KKT point of the sub-problem must (approximately) violate the constraint qualification conditions of the original problem.

Instead of choosing $\boldsymbol{\lambda}_k$ arbitrarily, if $\boldsymbol{\lambda}_{k+1}$ is chosen as:

$$\boldsymbol{\lambda}_{k+1} = \text{proj}_{Y_{\boldsymbol{\lambda}}}(\boldsymbol{\lambda}_k + \alpha_k \boldsymbol{G}_k) \tag{100}$$

where $\alpha_k = c_k$, convergence can often be significantly accelerated and the algorithm typically converges at lower values of c_k. The dual update step size α_k can also be improved to a more "optimal" choice than c_k or a Newton/quasi-Newton direction can be used instead of $\boldsymbol{G}_k$. These can lead to faster convergence but come at an increased computational cost. For more on the augmented Lagrangian algorithm, see Bertsekas (1996).

In the context of topology optimization, one challenge with AugLag is that the Hessian of the augmented Lagrangian function becomes rather ill-conditioned for large values of c where the condition number of the Hessian matrix goes to ∞ as $c \to \infty$. Given that computing the Hessian of the augmented Lagrangian in topology optimization is generally intractable, one must rely on first order methods to optimize the sub-problem. However even quasi-Newton methods may very well encounter difficulty converging if the Hessian approximation is not good enough and/or the starting point is not near a solution. Adding to this the scaling issues that are commonly found in topology optimization and this makes AugLag extremely difficult to fine-tune in practice

when using it in topology optimization. The convergence speed and numerical stability of AugLag can be extremely sensitive to the initial value of c, its increment rate, the initial solution, and the sub-problem's optimizer's terminating conditions.

The main computational advantage of AugLag is that it can handle block constraints more efficiently. Let $\boldsymbol{G}(\boldsymbol{x})$ be the vector-valued function whose components are the constraint functions $f_i(\boldsymbol{x}) \, \forall i \in [1, I+E]$ and let $\boldsymbol{M}(\boldsymbol{x}) = \text{proj}_{\boldsymbol{Y}_\lambda} \boldsymbol{G}(\boldsymbol{x})$ be the projection of $\boldsymbol{G}(\boldsymbol{x})$ on $\boldsymbol{Y}_\lambda$. The augmented Lagrangian formulation can therefore be written as:

$$\underset{\boldsymbol{\lambda} \in \boldsymbol{Y}_\lambda}{\text{maximize}} \quad \min_{\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}} \{\mathcal{L}_c(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + (\boldsymbol{\lambda} + c\boldsymbol{M}(\boldsymbol{x}))^T \boldsymbol{G}(\boldsymbol{x})\} \tag{101}$$

Note that the gradient of $\mathcal{L}_c(\boldsymbol{x}, \boldsymbol{\lambda})$ wrt $\boldsymbol{x}$, $\nabla_{\boldsymbol{x}} \mathcal{L}_c(\boldsymbol{x}, \boldsymbol{\lambda})$, can be written in terms of the gradient of $f(\boldsymbol{x})$, $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$, and the Jacobian of $\boldsymbol{G}(\boldsymbol{x})$, $\nabla_{\boldsymbol{x}} \boldsymbol{G}(\boldsymbol{x})$, as:

$$\nabla_{\boldsymbol{x}} \mathcal{L}_c(\boldsymbol{x}, \boldsymbol{\lambda}) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} \boldsymbol{G}(\boldsymbol{x})^T (\boldsymbol{\lambda} + 2c\boldsymbol{M}(\boldsymbol{x})) \tag{102}$$

To compute the above gradient, the full Jacobian of the constraints need not be built. Only the operator $\boldsymbol{w} \rightarrow \nabla_{\boldsymbol{x}} \boldsymbol{G}(\boldsymbol{x})^T \boldsymbol{w}$ needs to be defined. For the purposes of this paper, the term *block constraint* will be used to refer to constraints on functions $\boldsymbol{G}(\boldsymbol{x})$ where the operator $\boldsymbol{w} \rightarrow \nabla_{\boldsymbol{x}} \boldsymbol{G}(\boldsymbol{x})^T \boldsymbol{w}$ can be defined more efficiently than simply calculating the entire Jacobian, $\nabla_{\boldsymbol{x}} \boldsymbol{G}(\boldsymbol{x})$, and then multiplying its transpose by $\boldsymbol{w}$.

References

M. ApS. Mosek modeling cookbook, 2018. URL `https://docs.mosek.com/MOSEKModelingCookbook-v2.pdf`.

S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. 2007. URL `https://theory.cs.princeton.edu/complexity/book.pdf`.

D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, 1996.

M. Besançon, A. Carderera, and S. Pokutta. Frankwolfe.jl: a high-performance and flexible toolbox for frank-wolfe algorithms and conditional gradients, 2021.

S. Boyd and L. Vandenberghe. *Convex Optimization*. 2009.

G. H. Golub and C. F. V. Loan. *Matrix Computations*, volume 10. 1996.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Sc, 2006. ISBN 354035445X. doi: 10.1002/lsm.21040.

N. Parikh and S. Boyd. *Proximal Algorithms*. 2013.

K. Svanberg. The method of moving asymptotes - a new method for structural optimization. *International Journal for Numerical Methods in Engineering*, 24(2):359–373, 1987.

K. Svanberg. A Class of Globally Convergent Optimization Methods Based on Conservative Convex Separable Approximations. *SIAM Journal on Optimization*, 12(2): 555–573, 2002.

A. Wächter and L. T. Biegler. *On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming*, volume 106. 2006. ISBN 1010700405.