Derivative Free Model-Based Methods for Local Constrained Optimization

Trever Hallock

June 20, 2017

Contents

1 Introduction

This paper will discuss research topics for my research in derivative free optimization (DFO). It begins with an introduction to derivative free optimization supplemented by some of the recent advancements. It then details several future research directions and one in particular (filter methods) that has been studied. The focus is on model-based trust region algorithms for local search within constrained derivative free optimization.

1.1 Introduction to Derivative Free Optimization

1.1.1 What it is

Derivative free optimization refers to mathematical programs in which derivative information is not explicitly available. Derivatives cannot be calculated directly, and function evaluations are computationally expensive. One of the primary goals within derivative free optimization is to solve programs while avoiding as many expensive function evaluations as possible.

1.1.2 Problem formulations

This proposal aims to develop derivative-free algorithms for solving constrained nonlinear optimization problems of the form

$$\begin{array}{ll} \min & f(x) \\ \text{subject to} & c_i(x) \leq 0 \quad i \in \mathcal{I} \\ & c_i(x) = 0 \quad i \in \mathcal{E} \end{array}$$

where at least one of the functions $f, c_i, i \in \mathcal{I} \cup \mathcal{E}$ is a black box function, meaning that we have no information about its derivatives.

We will introduce several different forms this problem can take noting those which we will discuss algorithms for. For a more comprehensive characterization of the types constraints, consult [1].

1.1.3 Where problems come from

laziness and parameter tuning were circled.

There are a growing number of applications for DFO. For example, derivative free methods can be useful when the objective is the result of a simulation that does not admit automatic differentiation. As the popularity of complicated simulations increase, so does the demand for optimizing over black box software codes. which may be copyrighted. Derivative free optimization has also been a popular method for parameter tuning, as simulations may have several parameters with unidentified relationships to their output.

Sometimes user laziness can preclude derivative information. Even when it would be possible to compute derivative information, it may be prohibatively time consuming.

A trend within derivative free optimization is the permission for larger tolerances within solutions. Their functions are frequently expensive to evaluate, so we can only ask for a small number of significant figures. This implies slightly less concern for asymptotic convergence rates.

1.1.4 Noise. Deterministic versus stochastic

One branch of DFO is concerned with noisy evaluations of S. Noisy functions can be categorized as either deterministic or stochastic. Roundoff error, truncation error and finite termination errors can result in what is called deterministic noise. This means that although a function is not evaluated accurately, the error will not change across multiple function calls with the same input. On the other hand, stochastic noise means that each point in the domain is associated with a distribution of possible values S may return. In this paper we assume S is not noisy.

1.1.5 Types of constraints

Sometimes the derivatives of c are known, so the objective is the only derivative free function. One common such case is to include bound constraints of the form $b_L \leq x \leq b_U$ for some $b_L < b_U$, which gives rise to Box Constrained DFO (BCDFO). This is one of the better studied cases, with several software packages available, SPBOX, PSwarm, all NLopt algorithms except COBYLA (BOBYQA, NEWUOA, PRAXIS, Sbplx).

We are primarily interested in the case where no derivative information of c is known: for example, if they are also output from a simulation used to evaluate the objective. This means that each call to the objective gives values of the constraints as well, and vice versa.

This means that we have as many points for which we know the constaints as points for which we know the objective. This creates an interesting situation within model-based approaches which use different orders of models for the objective than the constraints. The algorithm designer must decide how to choose a subset of these points, use a higher order model, or fit an overdetermined model.

If the constraints can be evaluated at points outside the feasible region, the constraints are called *relaxable* constraints. Some problems additionally contain "hidden" constraints which are not explicit in the model but merely result in a notification that the objective could not be evaluated at the requested point. For example, this can arise when simulation software fails. This may mean that it is not possible to tell how close to a "hidden" constraint an iterate lies.

Another area that received attention recently is that of imposing structure on f, and c. For example, a method called Practical Optimization Using No Derivatives for sums of Squares is developed within [2] when f takes the form of a least squares error $f(x) = \frac{1}{2}\|F(x)\|^2$ over bound constraints where F is a nonlinear, vector valued function.

1.1.6 Importance of Derivatives

The lack of derivative information means that DFO methods are at a disadvantage when compared to their counterparts in nonlinear optimization. First and second derivative information is explicit in algorithms with quadratic convergence such as Newton's method. They are also present in conditions for convergence results such as Wolf's, Armijo or Goldstien for line search methods. Additionally, stopping criteria usually involve a criticality test involving derivatives. When derivatives are known, they should be used.

1.2 Proposed Thesis Direction

1.2.1 Main idea

The focus of the research will be on developing model-based trust region methods. Our strategy will be to adapt classical (derivative-based) methods for constrained nonlinear programming to the derivative-free context. The main idea will be to use function values calculated on a set of sample points to construct local models of the black box functions. Then, wherever derivatives are required in the classical algorithm, derivatives of the model functions will be used instead. Of course, other modifications will be needed to achieved desired convergence properties.

In the following section, I give an overview of fundamental concepts of model based methods, particularly trust region methods, along with a review of recent literature. I will then describe preliminary work on a trust region sequential quadratic programming (SQP) filter method based on [3]. Finally, I describe several ideas for future research.

1.2.2 Potential NLP algorithms to adapt

There are several considerations to make when selecting an NLP method. One of the primary concerns is how well it is extended to the trust region framework: for example, this conflicts with several goals of line searches as discussed in (2.1.6).

Another concern is that some algorithms only produce feasible iterates once the algorithm has converged. This is different than "any time algorithms" that maintain feasibility so that the algorithm can be stopped at any time to yield feasible guess. The longer an any time algorithm is run, the better the returned value is, until optimality is reached.

Here we outline several potential nonlinear programming algorithms of interest. For some of these algorithms, others have made progress in translating to a DFO context.

Line Search Line search is one type of algorithm researchers have made progress in adapting to the derivative free context [4].

Filter method The first filter methods in DFO were introduced in 2004 within the context of pattern searches [5]. Since then, other papers have made progress in a trust region framework including [6] and [7].

Active Set Active set methods have been considered for DFO [8], although the focus has been on bound constrained optimization.

Augmented Lagrangian A popular method for NLP is the Augmented Lagrangian method, which has also been adapted for both general and box constrained derivative free optimization [**?**].

Penalty Methods Several researchers have considered derivative free algorithms for penalty methods including [9]. There are many flavors of these algorithms.

Interior Point

2 Background

2.1 Derivative Free Algorithm classes

2.1.1 Automatic Differentiation

When S is the result of a simulation for which the source code is available, one convenient approach is to perform automatic differentiation. Although derivatives of complicated expressions resulting from code structure are difficult to work with on paper, the rules of differentiation can be applied algorithmically. However, the nature of the code or problem can make this very difficult: for example with combinatorial problems that rely heavily on if statements.

2.1.2 Direct search

Another approach is to use direct search methods that do not explicitly estimate derivative information but evaluate the objective on a pattern or other structure to find a descent direction. Examples of this include Coordinate descent, implicit filtering and other pattern based search methods. One of the most popular direct search method is Nelder Mead, which is implemented in fminsearch in Matlab. It remains popular although it is proven to not converge in pathological cases unless modifications are made.

These methods can be robust in that they are insensitive to scaling and often converge to a local minimum even when assumptions such as smoothness or continuity are violated. However, they ignore potentially helpful information because they do not use derivative information provided through the function evaluations. This means that they can also lack fast convergence rates.

2.1.3 Finite difference methods

Finite difference methods can be used to approximate the derivative of a function f. One common approximation called the symmetric difference is given by $\nabla f(x) \approx (\frac{f(x+he_i)-f(x-he_i)}{2h})_i$ for some small h where $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)^T \quad \forall\, 1 \leq i \leq n$ is the unit vector with 1 in its ith component. However, the number of function evaluations tends to grow large with the dimension and number of iterations the algorithm performs. This is because derivative information is only gathered near the current iterate when h is small, which is required for accurate derivative calculations. Because of the large number of function evaluations required for finite difference schemes, it may preferable to spread sample points out over the entire region where we may expect to step. Also, function evaluations may be subject to noise, making the finite difference approximations of derivative problematic.

2.1.4 Model based methods

In this paper, we are concerned with model based methods. These typically minimize a model function that only approximates the objective and constraints. The model functions are chosen to accurately represent the original function, but allow for derivative information to be calculated easily. They work by evaluating functions on a set of sample points to construct local models of the functions.

This allows the algorithm to minimize these easier model functions over a trust region, rather than working with the original function. When derivatives are given in the original function, model-based methods can also be used to approximate derivative information. We will see several examples in what follows.

Interpolation/regression methods Within Interpolation methods, we construct our model by regressing basis functions onto a set of sampled points. For example, given a function $f(x) : \mathbb{R}^n \to \mathbb{R}$ we can use a set of basis functions $\phi_i : \mathbb{R}^n \to \mathbb{R} \quad \forall 1 \leq i \leq d_1$ to construct a model function $m(x) = \sum_{i=1}^{d_1} \lambda_i \phi_i(x)$ approximating $f(x)$ by selecting appropriate $\lambda_i \in \mathbb{R}$. This is done by choosing a set of sample points $Y = \{y^1, y^2, \ldots, y^{d_2}\}$, evaluating $f = (f_1 = f(y^1), f_2 = f(y^2), \ldots, f_d = f(y^{d_2}))^T$ and forcing model agreement with the original function $f(x)$ by ensuring

$$\begin{bmatrix} \phi_1(y^1) & \phi_2(y^1) & \ldots & \phi_{d_1}(y^1) \\ \phi_1(y^2) & \phi_2(y^2) & \ldots & \phi_{d_1}(y^2) \\ & & \vdots & \\ \phi_1(y^{d_2}) & \phi_2(y^{d_2}) & \ldots & \phi_{d_1}(y^{d_2}) \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_{d_1} \end{bmatrix} \approx \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{d_2} \end{bmatrix}. \tag{1}$$

When $d_1 = d_2$ this is called interpolation and equality is desired within (1). When $d_1 < d_2$ this is called underdetermined interpolation. This can be handled by requesting and a minimum norm solution. Finally, when $d_1 > d_2$ this is called regression and only a least squares solution can be requested. Note that in practice, the set Y is shifted and scaled.

Basis functions The choice of model functions ϕ_i can have some affect on the convergence rate, as Powell showed in [10]. One common choice of basis functions is the Lagrange polynomials, in which we select polynomials satisfying $\phi_i(y^j) = \delta_{ij}$, the kroneker delta function. This reduces the matrix within (1) to an identity matrix. Lagrange polynomials of order p can be computed by starting with the monomial basis $\prod_{i=1}^{n} x_i^{n_i}$ for all choices of $0 \leq n_i \leq p$ with $\sum_{i=1}^{n} n_i \leq p$ and inverting the corresponding Vandermonde matrix.

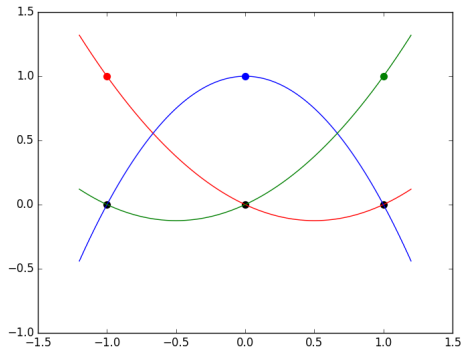
Newton's Fundamental polynomials are also used, and follow a similar pattern. However, they maintain different orders of polynomials within the basis: a single constant value, a set of $n+1$ linear polynomials, $n + \binom{n}{2}$ quadratic functions, and more for higher order polynomials. Radial basis functions may have some intuitive advantage because the algorithm makes claims about the accuracy of the function over a trust region.

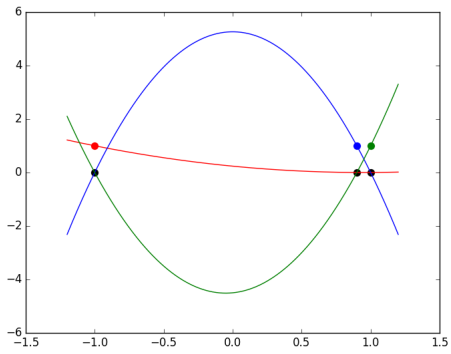
Model functions are usually chosen to be fully linear or fully quadratic: terms describing how the model's error grows as a function of the trust region radius.

Sampling issues These methods have issues with poor sampling choices. The two most important aspects of the sample points are their geometry and proximity.

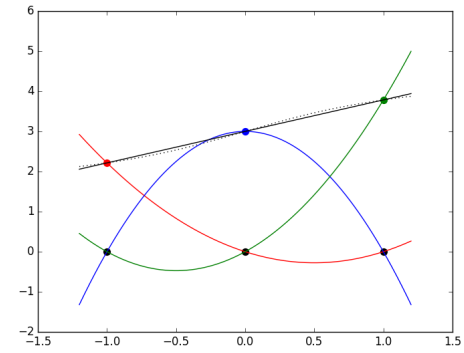
Geometry For geometry, regression based methods require that the set the function is evaluated at must be Λ-poised for a fixed constant Λ. If the set is Λ-poised for a certain trust region radius, then the maximum absolute value of any Lagrange polynomial over the trust region is one. This ensures that the Vandermonde matrix used to find the coefficients used to express the model function in terms of a basis of Lagrange polynomials is well conditioned. Although we do not go into the details here, problems become apparent when comparing the Lagrange polynomials associated with a poised set with those of an ill poised set.

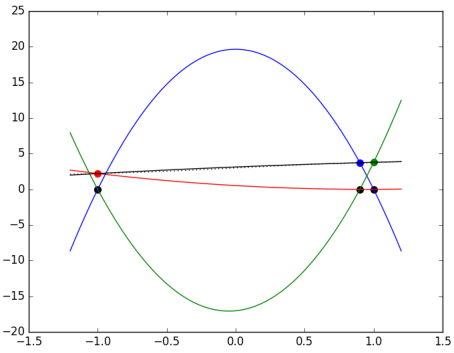
Within the first set of pictures below we see the set of quadratic Lagrange polynomials on the interval $[-1, 1]$. The maximum value of these polynomials over the trust region is simply 1. However, if we use sample points $\{-1, .9, 1\}$ instead of the points $\{-1, 0, 1\}$, we find the second set of polynomials.





If we use these to approximate $3 + \tan^{-1}(x)$, the first basis functions do not vary far from the objective value over the trust region, and the maximum difference between the function and the model function is 0.0711. However, the second basis functions jump far away from the actual function, and the maximum difference between the model function and actual function is 0.1817.





Proximity Proximity refers to the trust region radius. The trust region must go to zero if we are to be sure that we have reached a critical point. In general, the smaller the trust region, the closer to linear or quadratic the original function will look. This is because the model's error term given by Taylor's expansion is proportional to the trust region radius.

2.1.5 Model-based, Trust Region Methods

The overall description of the trust region framework is that a set of poised points are chosen for some radius $\Delta > 0$ about the current iterate. The objective and constraints are then evaluated at these points to construct a model function as a linear combination of some set of basis functions. Next, the model is minimized over this trust region and the argument minimum becomes the trial point. The objective is evaluated at the trial point and a measure of reduction ρ is computed. If ρ implies that sufficient reduction has been made and that the model approximates the function well, the trial point is accepted as the new iterate. Otherwise, the trust region is reduced to increase model accuracy.

For unconstrained optimization, the algorithmic framework can be described with these steps:

1. Create a model function $m_k(x)$.
   - In classical trust region methods, the following quadratic model function is used:
   $$m_k(x) = f(x^{(k)}) + \nabla f(x^{(k)})^T(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T\nabla^2 f(x^{(k)})(x - x^{(k)})$$
   - $\nabla f(x^{(k)})$ and $\nabla^2 f(x^{(k)})$ must be approximated
   - Geometric properties of the sample set that must be satisfied
2. If $\nabla m_k(x) < \tau$ stop, where τ is some tolerance
3. Solve the Trust region subproblem: $s^{(k)} = \arg\min_{s \in B_{x^{(k)}}(\Delta_k)} m_k(x^{(k)} + s)$
   - $B_{x^{(k)}}(\Delta_k) = \{x \in \mathbb{R}^n : \|x - x^{(k)} \leq \Delta_k\}$ is the ball of radius Δ_k centered at $x^{(k)}$
4. Test for improvement
   - Compute
   $$\rho_k = \frac{f(x^{(k)}) - f(x^{(k)} + s^{(k)})}{m_k(x^{(k)}) - m_k(x^{(k)} + s^{(k)})} \tag{2}$$
   which measures the actual improvement over predicted improvement
   - If ρ is small, $x^{(k+1)} = x^{(k)}$ (reject) and decrease radius
   - If ρ is intermediate, $x^{(k+1)} = x^{(k)} + s^{(k)}$ (accept) and decrease radius
   - If ρ is large, $x^{(k+1)} = x^{(k)} + s^{(k)}$ (accept) and either increase the radius or decrease if $\nabla m_k(x_k)$ is small
5. Go to step 1

Our goal is generalize this framework to handle constraints, where we must reduce constraint violation while ensuring the accuracy of the models of the constraints.

2.1.6 Trust region versus Linesearch

Within derivative free optimization, we can ensure the accuracy of our model function by sampling points over a small enough trust region. However, reducing the trust region implies more points must be evaluated. Linesearch methods rely on the the ability to calculate a descent direction that will be accurate in a small enough region around the current iterate: small enough that the trust region must be reduced to ensure the model's accuracy.

This means trust region framework fits into derivative free optimization more naturally than line search methods. Not only do the trust regions arise naturally, but many line search algorithms exploit how much easier it is to find a descent direction than solve a trust region subproblem. However, in derivative free optimization, solving a costly trust region subproblem is acceptable if it allows us to avoid even more expensive function evaluations.

2.1.7 Literature Background

Derivative free methods Within [11] derivative-free methods are developed in detail. This is the first text book devoted to derivative free optimization. It contains a good explanation of ensuring geometry of the current set with poisedness for unconstrained problems and also covers other derivative-free methods including direct-search and line search.

A good review of derivative free algorithms and software libraries can be found in [12]. This compares several software libraries, and reviews the development of derivative free optimization since it started. Another recent review can be found in [13].

Within [14] and [15] Biegler uses a filter method to ensure global convergence within a line search framework.

Filter methods The original filter method was proposed by Gould and Toint in [16]. The motivation for the filter method was that the algorithm does not need to tune any parameters as in penalty or merit methods. We will discuss filter methods in more detail below.

In the 2006 paper [17], Fletcher reviews how filter methods have developed. The only references to derivative-free versions of the filter method are those applied to pattern based (direct) methods. However, the authors outline trust region filter methods along with several other variants.

Within [18] a sequential quadratic programming method is applied to the filter method?

I based my algorithm on the trust region filter method described in [3].

Derivative free filter methods In Brekelman's paper [7], a trust region filter method is developed to minimize function evaluations by constructing linear model functions. This algorithm evaluates the underlying functions at points chosen by an experimental design, and moves the design towards otpimality according the filter while making any necessary geometry improvement steps.

A recent paper from September 2016 [6] implements a derivative-free trust region filter method for solving the following intriguing program:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x,y,z) \\ \text{subject to} \quad & g(x,y,z) \leq 0 \\ & h(x,y,z) = 0 \\ & y = S(x) \end{aligned}$$

where $S : \mathbb{R}^n \to \mathbb{R}^m$ is a black box function, $f : \mathbb{R}^{n+m+k} \to \mathbb{R}$, $g : \mathbb{R}^{n+m+k} \to \mathbb{R}^{|\mathcal{I}|}$, and $h : \mathbb{R}^{n+m+k} \to \mathbb{R}^{|\mathcal{E}|}$.

Biegler assumes that the dimension of x is small compared to the dimension of y and z. This permits the algorithmic approach of relaxing $y = S(x)$.

This is more convenient for some problems than the algorithm we consider as it allows for the objective to depend on other glass box functions of the input and multiple outputs of the black box function. Glass box functions, as opposed to black box, are functions for which derivative information is known. The authors compare their algorithm to finite difference methods as well as kriging on three different applications from Chemistry. Within the algorithm, the current iterate is always feasible with respect to all inequalities except for the constraint $y = d(z)$.

Colson has also applied filter techniques to derivative-free optimization in his 2004 Ph.D. thesis [19]. This focuses on bilevel programming.

Beyond the filter Within nonlinear programming, some new techniques have been proposed extending the filter method. One new algorithm Toint proposes in [20] generalizes the filter method with the notion of a trust funnel. This is an interior point algorithm that is suitable for large scale problems, as it is matrix free. This could be generalized as it is only for glass box functions with no derivative-free methods.

3 Progress

In this section we discuss one algorithm that has been converted to a derivative free context. This is the filter method, which has been independently adapted for derivative free optimization by [7].

3.1 SQP Filter method

We will be considering problems of the form

$$\begin{aligned} \min_{x} \quad & f(x) \\ c_i(x) \leq 0 \quad & \forall i \in \mathcal{I} \\ c_i(x) = 0 \quad & \forall i \in \mathcal{E} \end{aligned} \tag{3}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, and each $c_i : \mathbb{R} \to \mathbb{R}$. It will be convenient to write $c_{\mathcal{I}}(x) = (c_1(x), c_2(x), \ldots, c_{|\mathcal{I}|})^T$, $c_{\mathcal{E}}(x) = (c_{|\mathcal{I}|+1}(x), c_{|\mathcal{I}|+2}(x), \ldots, c_{|\mathcal{I}|+|\mathcal{E}|})^T$ and $c(x) = (c_{\mathcal{I}}^T(x), c_{\mathcal{E}}^T(x))^T$.

We work within a trust region, sequential quadratic programming framework with interpolating model functions that uses a filter method introduced by Gould and Toint [**?**]. For our algorithm, we assume that all functions are derivative-free: all functions are evaluated by a single call to a black box function: $S(x) = (f(x), c_{\mathcal{I}}(x)^T, c_{\mathcal{E}}(x)^T)^T$.

We first compute an interpolation set poised for regressing a set of model functions, which we choose to be quadratic functions. Although we have enough sample points to construct a quadratic model of the constraints, we only construct linear models to avoid the complexity of Quadratically Constrained Quadratic Programming which is NP-hard.

3.1.1 Model functions

At an iteration k, we first construct a model function $m_f^k(x)$ that we use to approximate the first and second derivatives of $f(x)$. We also construct $m_{\mathcal{I}}^k(x)$ and $m_{\mathcal{E}}^k(x)$ to approximate the first derivatives of $c(x)$. Namely, at an iterate x^k, we let $f^k = f(x^k)$, $g^k = \nabla m_f^k(x^k)$. We also define the $c_{\mathcal{I}}^k = c_{\mathcal{I}}^k(x^k)$, $A_{\mathcal{I}}^k = \nabla m_{\mathcal{I}}^k(x^k)$, $c_{\mathcal{E}}^k = c_{\mathcal{E}}(x^k)$, and $A_{\mathcal{E}}^k = \nabla m_{\mathcal{E}}^k(x^k)$.

We will also need to compute the Hessian of the Lagrangian for (3). The Lagrangian is defined as $L(x, \mu, \lambda) = f(x) + \sum_{i\in\mathcal{I}} \mu_i c_i(x) + \sum_{i\in\mathcal{E}} \lambda_i c_i(x)$. To compute its second derivative, we let A^k and c^k contain $A_{\mathcal{E}}^k$ and $c_{\mathcal{E}}^k$ as well as any rows of $A_{\mathcal{I}}^k$ and $c_{\mathcal{E}}^k$ corresponding to active constraints at x^k. The set of active constraints $\mathcal{A} \subseteq \mathcal{I} \cup \mathcal{E}$ includes $\mathcal{E}$ and any $i \in \mathcal{I}$ for which $c_i(x^k) \geq 0$. We then solve the system

$$\begin{aligned} \nabla^2 m_k(x_k)d + {A^k}^T\lambda &= g^k \\ A^k d &= c^k \end{aligned}$$

for λ and compute

$$H^k = \nabla^2 m_k(x^k) + \sum_{i\in\mathcal{E}} \lambda_i \nabla^2 m_{g,i}^k(x^k) + \sum_{i\in\mathcal{A}\setminus\mathcal{E}} \lambda_i \nabla^2 m_{h,i}^k(x^k).$$

With these definitions, we define the quadratic subproblem as

$$\begin{aligned} \arg\min_s f(x^k + s) = f^k + (g^k)^T s + \frac{1}{2}(s^k)^T H^k s^k \\ c_{\mathcal{I}}^k + A_{\mathcal{I}}^k s \leq\ & 0 \\ s.t. \qquad c_{\mathcal{E}}^k + A_{\mathcal{E}}^k s =\ & 0 \\ \|s\| \leq\ & \Delta_k. \end{aligned}$$

3.1.2 Step decomposition

In each iteration, we attempt to compute a step s^k that will decrease either the constraint violation or the function value. To this end, we decompose the step s^k into a normal step n^k intended to decrease constraint violation and a tangential step t_k intended to reduce the objective. The step n^k projects the current iterate onto the feasible region. Currently, we project x^k onto only the linear model of the feasible region. This gives rise to the following definition:

$$\begin{aligned} n_k = \arg\min_n \|n\|^2 \qquad (4) \\ c_{\mathcal{I}}^k + A_{\mathcal{I}}^k n \leq\ & 0 \\ s.t. \qquad c_{\mathcal{E}}^k + A_{\mathcal{E}}^k n =\ & 0 \\ \|n\|^2 \leq\ & \Delta_k^2 \qquad (5) \end{aligned}$$

The constraints ensure that the point $x^k + n^k$ will lie within the trust region and the linearization of the feasible region at the current point x^k. The objective ensures that this is the projection of the current iterate onto this region.

However, we wish for more than this: we also want $x^k + s^k$ to lie within the feasible region. We need to know there is enough space to provide sufficient decrease within the tangential step. Therefore, we require the stronger condition that

$$\|n\| \leq \kappa_\Delta \Delta_k \min\{1, \kappa_\mu \Delta_k^\mu\} \qquad (6)$$

for some fixed constants $0 < \kappa_\Delta \leq 1$, $\kappa_\mu > 0$, $0 < \mu < 1$.

If this stronger condition is satisfied, we say that the program is *compatible*. We will discuss what happens when the program is not compatible under feasibility restoration. When the current iterate and trust region radius are compatible, we are able to compute a tangential step t^k

$$\begin{aligned} t^k = \arg\min_t \quad (g^k + H^k n^k)^T t + \frac{1}{2} t^T H^k t \qquad (7) \\ s.t. \qquad A_{\mathcal{E}}^k t =\ & 0 \\ c_{\mathcal{I}}^k + A_{\mathcal{I}}^k (n_k + t) \leq\ & 0 \\ \|n^k + t\|^2 \leq\ & \Delta_k^2 \end{aligned}$$

to decrease the objective value while staying within the trust region and the feasible region. Up to the constant term $\frac{1}{2}(n^k)^T H^k n^k$, this is a restatement of the quadratic subproblem.

3.1.3 The Filter

The filter is the tool used to ensure convergence to a feasible point, by ensuring that we never accept an iterate with both a worse objective value and a worse constraint violation. We introduce a new quantity

$$\theta^k = \theta(x^k) = \max\{\max_{i\in\mathcal{E}}\{|c_i(x^k)|\}, \max_{i\in\mathcal{I}}\{[c_i(x^k)]_+\}\}$$

which measures the current constraint violation. We have also experimented with

$$\theta^k = \theta(x^k) = \sum_{i\in\mathcal{E}} |h_i(x^k)| + \sum_{i\in\mathcal{I}} [g_i(x^k)]_+.$$

The filter itself is set of nondominated points $(\theta_i, f_i) \in \mathbb{R}^2$ from some previous iterations. A point $(x_1, x_2, \ldots, x_n)$ is said to dominate another point $(y_1, y_2, \ldots, y_n)$ if $x_i \leq y_i \forall 1 \leq i \leq n$ and $x_i < y_i$ for at least one $1 \leq i \leq n$. If there are no points in the filter that dominate a point x, then x is said to be nondominated. The algorithm ensures that all new iterates are nondominated with respect to all previous iterates when the problem is viewed as a multi-criteria optimization problem $\min(\theta, f)$. To ensure that no accumulation points are infeasiable, we must also ensure that the objective decreases by a greater amount when the current iterate is far from the feasible region. Therefore, we introduce a constant $0 < \gamma_\theta < 1$ and require either

$$\theta(x_k) \leq (1 - \gamma_\theta)\theta_i \tag{8}$$

or

$$f(x_k) \leq f_i - \gamma_\theta \theta_i \tag{9}$$

for all (θ_i, f_i) that are currently in the filter. When this condition is satisfied, we say that the new iterate x_k is *acceptable* to the filter. As the algorithm iterates, we add values of θ^k and f^k to the filter. When new points are added to the filter, the algorithm removes all points dominated by the new point.

3.1.4 Restoration Step

When the current iterate is not compatible, we can do nothing but call a restoration method. The goal of the feasibility restoration step is to find a new iterate and trust region radius that allows the current iterate to be compatible.

While performing the restoration step, we minimize the squared norm of θ with no regard to the objective values. That is, we wish to approximately compute $\arg\min_x \theta(x)$. This is now an unconstrained problem, for which we can use classical derivative free trust region techniques. In each iteration of the restoration algorithm we minimize the quadratic model of the constrained violation θ, and update the trust region based on the new function value. However, we only seek a point which is acceptable to the filter. This means that rather than continuing until a critical point is found, we check during each iteration if the new iterate and trust region radius are compatiable within the filter algorithm.

It is possible that the restoration step is unsuccessful if the iterates approach an infeasible local minimum of the constraints. In this case, the algorithm will fail to find a feasible local minimum, and will need to be restarted. This possibility is inherent within local search algorithms for non linear optimization.

3.1.5 Criticality Measure

In order to construct stopping criteria, we introduce a criticality measure χ which goes to zero as the iterates approach a first order critical point. Once this has reached small enough threshold, and the trust region is small enough, we can terminate the algorithm.

That is, if $x^\star$ is a first order critical point satisfying

$$\begin{aligned}
\nabla f(x^\star) + \sum_{i\in\mathcal{I}} \lambda_i \nabla c_i(x^\star) + \sum_{i\in\mathcal{E}} \mu_i \nabla c_i(x^\star) &= 0 \\
c(x^\star)_i \mu_i &= 0 \quad \forall i \in \mathcal{I} \\
c(x^\star) &\leq 0 \quad \forall i \in \mathcal{I} \\
c(x^\star) &= 0 \quad \forall i \in \mathcal{E}
\end{aligned}$$

for some $\lambda_i \in \mathbb{R}$, and $\mu_i \geq 0$, then we need $\lim_{x\to x^\star} \chi(x) = 0$.

This suggests the following definition:

$$\chi = |\min_t \langle g_k + H_k n_k, t\rangle| \tag{10}$$

$$A_\mathcal{E} t = 0 \tag{11}$$

$$c_\mathcal{I} + A_\mathcal{I} t \leq 0 \tag{12}$$

$$\|t\| \leq 1 \tag{13}$$

Notice that this must be computed after the normal step n^k has been computed.

3.1.6 Sufficient reduction of model

It is not enough to simply ask for decrease in the objective every iteration. We also require *sufficient* reduction, so that any accumulation point of the sequences of iterates is feasible. To this end, we introduce constants $0 < \kappa_\theta < 1$ and $\psi > \frac{1}{1+\mu}$ and require

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_\theta \theta_k^\psi. \tag{14}$$

3.2 The Algorithm

With these tools, we can now describe the algorithm. The overall steps are as follows:

---

1. **Initialize** each constant introduced so far
2. **Compute model functions** m_f, m_g, m_h, the active constraints $\mathcal{A}$, and the Hessian of the Lagrangian H_k
3. **Compute the normal step** n_k according to (4)

   If the feasible region of (4) is empty or (6) is not satisfied, then

   - Add (θ_k, f_k) to the filter
   - Call the feasibility restoration routine
   - Increment k
   - Go to step 2
4. **Compute criticality measure** χ from (10)

   - If χ is small, but the trust region radius is larger than the tolerance, then decrease the trust region, increment k and return to step 2
   - Otherwise, if χ and the trust region radius is smaller than the tolerance, return success
   - If χ is zero, then return success
5. **Compute tangential step** t_k according to (7)

   Define $s_k = n_k + t_k$.
6. Check for **sufficient reduction** in the model function

   If (14) is satisfied

   - Add (θ_k, f_k) to the filter
   - Increment k
   - Go to step 2
7. **Evaluate** the function and constraints at the trial point $x_k + s_k$

   If the new point is not acceptable to the filter according to either (8) or (9), then

   - Add (θ_k, f_k) to the filter
   - Increment k
   - Go to step 2
8. **Compute** ρ according to (2)

   - If $\rho \leq \gamma_1$ is small, then decrease the trust region radius and go to step 2
   - If $\gamma_1 < \rho \leq \gamma_2$ is intermediate, then accept the trial point $x_{k+1} = x_k + s_k$.
   - If $\rho > \gamma_2$, and χ is small, then decrease the trust region radius.
   - If $\rho > \gamma_2$ is large, and χ is large, then increase the trust region radius.
9. increment k and go to step 2

3.3 Changes needed for DFO

This algorithm is identical to the classical filter algorithm for NLP, except for the replacement of derivative information of the original functions with derivative information computed by the model function as well as one more small modifications mentioned here.

One issue with applying the original algorithm within a DFO context was that the trust region radius is not required to go to zero. However, within DFO we must also require that the trust region goes to zero as we approach a stationary point to ensure model convergence to the original function. One way of ensuring this is to decrease the trust region radius when the current step lies well within the trust region radius. However, a better approach may be to introduce a tolerance on the criticality measure and decrease the trust region whenever the criticality falls below the threshold. Thus we introduce a new constant η_χ.

We also placed the check for sufficient decrease (Step 6) before evaluating the function at the trial point (Step 7). This means that we can avoid evaluating the function if the model does not predict a decrease. However, this also means that we do not accept some points previously accepted in the classical algorithm.

3.4 Computational results and Testing libraries

3.4.1 Algorithms

In order to compare the algorithms we develop, we can compare our algorithm with one of a few current software libraries for derivative free optimization. One such library is SPDEN, which is a GPL library implementing a sequential penaly DFO method written by Prof. Klaus Truemper from the University of Texas at Dallas. The other main library for testing our algorithms will be NLOpt, which is a nonlinear optimization library contained within Coin-OR.

Other potential libraries include Tomlab/LGO and Tomlab/EGO. However, these algorithms are for global optimization. Although there are several algorithms for bound constrained derivative free optimization, only a small number of algorithms are accessible for nonlinear constraints.

3.4.2 Test Functions

The Cute library has collected several interesting functions for testing nonlinear optimization algorithms, and has developed into CUTEr and CUTEst.n addition to these, we will used dfovec which is a Matlab implementation of several test functions.

Most of these functions do not have naturally constraints, but constraints can be introduced by using some output variables or other functions as constraints.

4 Future Work

In this section, I will discuss what our next steps are.

4.1 Refining our implementation of the SQP filter method

We began with the filter method, and recent progress has been done by other researchers with this method [7].

We will continue to refine our implementation, there are several topics to explore here:

- Different infeasibility measures
  - We can generalize (relax) the possible steps by only requiring that new iterates are only acceptable with respect to a multi objective filter of the form $(f(x), [c_1(x)]_+, \dots [c_{|\mathcal{I}|}(x)]_+, |c_{|\mathcal{I}|+1}(x)|, \dots, |c_{||\mathcal{I}|+|\mathcal{E}|}|)$.
  - The L_2 for θ given in (3.1.3) is promising on our current test functions, and we can experiment with L_1
- Different strategies for fitting model functions.
  - The authors only considered linear models, there are several ways to generalize this.
  - This is particularly interesting when modelling quadratic constraint functions.
- Ellipsoidal trust regions.
- Relaxing the condition that all iterates have to remain feasible with respect to all constraints except $y = d(z)$, which is assumed within [7].
- Considering Parallelization

  Modern super computer design can be exploited in several different ways, including parallelization of linear algebra kernels or of function and/or derivative evaluations within the algorithm. More interestingly is from the modification of the basic algorithms which increase the degree of intrinsic parallelism. This can come from performing multiple function and/or derivative evaluations asynchronously. Other current parallel algorithms include NOMAD and SNOBFIT.

4.2 Expand test libraries

Through our research, we also hope to expand the set of test libraries for derivative free optimization. Currently, constraints are artificially imposed on problems from the current set of test functions. We would like to have several different classes of programs based on how these constraints affect algorithms. Distinguishing some types of constraints may promote progress in specific areas of derivative free optimization.

Types of constraints For example, one area that may not have received sufficient attention is that of NLP responses to constraints that leave narrow feasible regions near a critical point. This is of particular interest to DFO because reducing the dimension near a critical point could significantly harm the geometry of the poised set. One approach for algorithms attempting to handle these constraints is to redefine the poisedness as the maximum value of the model function over only the feasible region intersected with the trust region.

Varied runtimes Another type of problem that to our knowledged is not handled within the literature is programs with highly varied runtimes. Algorithms that hope to do well on this type of problem may consider explicitly modeling the runtime of evaluating the objective function. We would assume that the runtime varies continuously with x. For example, in some problems from PDE's, the runtime of a simulation may depend on its "stiffness," so the runtime of an algorithm optimizing over this type of similation may depend highly on choices of sample points. In these situations, it may be advisable to consider the runtime cost of the function when evaluating new points: choosing points that remain poised with respect to the geometry but minimize evaluation time instead of simply solving the trust region sub problem.

4.3 Exploit Structure

Formulations In addition to experimenting with these generalizations, we would like to work within the general problem statement:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x, S(x)) \\ \text{subject to} \quad & c_i(x, S(x)) \leq 0, \ \forall i \in \mathcal{I} \\ & c_i(x, S(x)) = 0, \ \forall i \in \mathcal{E} \end{aligned}$$

where $S : \mathbb{R}^n \to \mathbb{R}^m$, $f : \mathbb{R}^{n+m} \to \mathbb{R}$, and $c_i : \mathbb{R} \to \mathbb{R}$. Here, $S(x)$ is the black-box function, We will assume that f, and $c_i, i \in \mathcal{I} \cup \mathcal{E}$ are a priori functions, which can be directly evaluated. Although derivatives of S are not known, we will continue to assume assume that it has nice properties, such as being twice continuously differentiable. This ignores applications in which S may be corrupted by noise.

Sparsity Further structure that may be exploited is that of sparse Hessians or Jacobians. While using a Newton based approach, the calculation of the Hessian can be expensive due to the dimensions involved. Research can be done studying for exploiting sparsity within the derivatives of the black box functions.

For example, functions that reduce as

$$f(x, y, z) = f_1(x) + f_2(y, z)$$

will have sparse Hessian matrices as $\frac{\partial^2}{\partial y \partial x} f(x, y, z) = \frac{\partial^2}{\partial x \partial x} f(x, y, z) = 0$. Algorithms that take advantage of this sparsity can be more memory efficient.

References

[1] S. L. Digabel and S. M. Wild, "A taxonomy of constraints in simulation-based optimization," 2015.

[2] S. M. Wild, "Solving derivative-free nonlinear least squares with POUNDERS," April 2014 (Revised June 2015).

[3] R. Fletcher, N. Gould, S. Leyffer, P. Toint, and A. Wchter, "Global convergence of a trust-region sqp-filter algorithm for general nonlinear programming," *SIAM Journal on Optimization*, vol. 13, no. 3, pp. 635–659, 2003.

[4] G. Fasano, G. Liuzzi, S. Lucidi, and F. Rinaldi, "A linesearch-based derivative-free approach for nonsmooth constrained optimization," *SIAM Journal on Optimization*, vol. 24, no. 3, pp. 959–992, 2014.

[5] M. A. Abramson, C. Audet, G. Couture, and J. E. D. Jr., "Filter pattern search algorithms for mixed variable constrained optimization problems," *Pacific Journal of Optimization*, vol. 17, pp. 477–500, 2007.

[6] J. P. Eason and L. T. Biegler, "A trust region filter method for glass box/black box optimization," *AIChE Journal*, vol. 62, no. 9, pp. 3124–3136, 2016.

[7] R. Brekelmans, L. Driessen, H. Hamers, and D. den Hertog, "Constrained optimization involving expensive function evaluations: A sequential approach," *European Journal of Operational Research*, vol. 160, no. 1, pp. 121 – 138, 2005. Applications of Mathematical Programming Models.

[8] S. Gratton, P. L. Toint, and A. Trltzsch, "An active-set trust-region method for derivative-free nonlinear bound-constrained optimization," *Optimization Methods and Software*, vol. 26, no. 4-5, pp. 873–894, 2011.

[9] G. Liuzzi, S. Lucidi, and M. Sciandrone, "Sequential penalty derivative-free methods for nonlinear constrained optimization.," *SIAM Journal on Optimization*, vol. 20, no. 5, pp. 2614–2635, 2010.

[10] M. J. D. Powell, *Recent research at Cambridge on radial basis functions*, pp. 215–232. Basel: Birkhäuser Basel, 1999.

[11] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to Derivative-Free Optimization*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2009.

[12] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations," *Journal of Global Optimization*, vol. 56, no. 3, pp. 1247–1293, 2013.

[13] A. Custodio, K. Scheinberg, and L. Vicente, "Methodologies and software for derivative-free optimization," 2017.

[14] A. Wächter and L. T. Biegler, "Line search filter methods for nonlinear programming: Motivation and global convergence," *SIAM Journal on Optimization*, vol. 16, no. 1, pp. 1–31, 2005.

[15] A. Wächter and L. T. Biegler, "Line search filter methods for nonlinear programming: Local convergence," *SIAM Journal on Optimization*, vol. 16, no. 1, pp. 32–48, 2005.

[16] N. I. M. Gould and P. L. Toint, "Nonlinear programming without a penalty function or a filter," *Mathematical Programming*, vol. 122, no. 1, pp. 155–196, 2010.

[17] R. Fletcher, S. Leyffer, R. Fletcher, and S. Leyffer, "A brief history of filter methods," tech. rep., 2006.

[18] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming for large-scale nonlinear optimization," *Journal of Computational and Applied Mathematics*, vol. 124, no. 12, pp. 123 – 137, 2000. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

[19] B. Colson, "Trust-region algorithms for derivative-free optimization and nonlinear bilevel programming," *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 2, no. 1, pp. 85–88, 2004.

[20] R. Sampaio and L. Toint, "A trust-funnel method for nonlinear optimization problems with general nonlinear constraints and its application to derivative-free optimization, most information in this cite is wrong," *NAXYS Namur Center for Complex Systems*, vol. 61, no. 5000, pp. 1–31, 2015.

[21] J. Nocedal and Y. Yuan, "Combining trust region and line search techniques,"