Derivative Free Model-Based Methods for Local Constrained Optimization

Trever Hallock

May 18, 2017

Contents

1 Introduction **2**
1.1 Introduction to Derivative Free Optimization . . . 2
1.1.1 What it is . . . 2
1.1.2 Where problems come from . . . 2
1.1.3 Problem formulations . . . 2
1.1.4 Noise. Deterministic versus stochastic . . . 2
1.1.5 Types of constraints . . . 3
1.1.6 Importance of Derivatives . . . 3
1.2 Proposed Thesis Directioin . . . 3
1.2.1 Main idea . . . 3
1.2.2 Potential NLP algorithms to adapt . . . 3
1.2.3 Overview of the rest of the paper . . . 4

2 Background **4**
2.1 Derivative free classes . . . 4
2.1.1 Automatic Differentiation . . . 4
2.1.2 Direct search . . . 4
2.1.3 Model based methods . . . 5
2.1.4 Model-based, Trust Region Methods . . . 6
2.1.5 Trust region versus Linesearch . . . 7
2.1.6 Literature Review Should this still be Background? Many of the names in the outline look like they were intended to be changed. . . . 7

3 Progress **8**
3.1 SQP Filter method . . . 8
3.1.1 Model functions . . . 8
3.1.2 Step decomposition . . . 8
3.1.3 The filter . . . 9
3.1.4 f-type versus θ-type . . . 9
3.1.5 Restoration Step . . . 9
3.1.6 Criticality Measure . . . 10
3.1.7 Not here yet . . . 10
3.2 The Algorithm . . . 10
3.3 Changes needed for DFO . . . 11
3.4 Pseuodcode of algorithm . . . 11
3.5 Computational results and Testing libraries . . . 13
3.6 I added:Generalizations . . . 13

4 Potential research **13**
4.1 I added:Goals . . . 13
4.2 Types of constraints . . . 13
4.2.1 Thin constraints . . . 13
4.2.2 Hidden constraints . . . 13
4.3 Parallelization . . . 13
4.4 Sparse Hessian or Jacobian . . . 14

1 Introduction

This paper will discuss research topics for my research in derivative free optimization (DFO). It begins with an introduction to the goals of derivative free optimization supplemented by some of the recent advancement made in the field. 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 programs programs algorithms in which derivative information is unknown, deceptive or otherwise impractical to compute. The number of function evaluations can grow when approximating derivatives that are not given explicitly. Thus, 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 Where problems come from

There are a growing number of such applications. For example, derivative free methods can 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 popular within parameter tuning, as simulations may have several parameters having 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 prohibitively 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 regard for asymptotic convergence rates.

Your outline mentions iterative methods. Should I say something about them? I think this is good enough, unless you had more specifics...

1.1.3 Problem formulations

Derivative free methods consider nonlinear, constrained optimization problems of the form

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

which give rise to different classes of derivative-free optimization based on properties of $S : \mathbb{R}^n \to \mathbb{R}^m$, $f : \mathbb{R}^{n+m} \to \mathbb{R}$, $g_i : \mathbb{R}^{n+m} \to \mathbb{R}$, and $h_i : \mathbb{R}^{n+m} \to \mathbb{R}$. Here, $S(x)$ is a black-box function, meaning that we have no information about its derivatives. Although derivatives of S are not known, we will assume that S, f, g, and h are all continuously twice differentiable. We will introduce several different forms this problem can take before narrowing in on our interest.

An intriguing form of the previous program is presented by [1] and given here:

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

where $S : \mathbb{R}^n \to \mathbb{R}^m$, $f : \mathbb{R}^{n+m+k} \to \mathbb{R}$, $g : \mathbb{R}^{n+m+k} \to \mathbb{R}^{|\mathcal{I}|}$, 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 $y = S(x)$.

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

If $g(x, S(x)) = g(x)$ and $h(x, S(x)) = h(x)$, then the derivatives of g and h 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: CITE.

We are primarily interested in the case where no derivative information of g and h are 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 produces $g(x, S(x)) = g(S(x))$ and $h(x, S(x)) = h(S(x))$.

(This means that we have as many sample points for our constraints as our objective. This creates an interesting situation within model-based approaches in which we 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. 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, g, and h. For example, when f takes the form of a least squares error, some improvements can be made. CITE Stephen Wild

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. For this reason, it is desirable for n, the dimension of x, to be small.

1.2 Proposed Thesis Directioin

Within this section, we will discuss the details of what I propose to study. I will focus strictly on model-based methods for local optimization.

1.2.1 Main idea

Future work includes converting nonlinear algorithms to a derivative free context. We start by choosing a nonlinear optimization algorithm, such as the ones listed below. The algorithm will contain steps that reference derivatives, for example, in either computation of a step direction or in a criticality measure. To convert the algorithm to a DFO context, the derivatives are replaced with derivatives of the corresponding model function over a trust region. These model functions are constructed from function values at selected sample points, and are trusted to be accurate locally.

However, once this has been done, the algorithm may require further modifications to ensure convergence. For example, in the example we give later of converting the filter method, we had to change trust region management rules to ensure convergence. Not only did the trust region simply not go to zero in the filter method which is required in the DFO context, but we needed to decrease the trust region with a weakened criticality measure.

1.2.2 Potential NLP algorithms to adapt

There are several consideration 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.

Another concern is that filter methods only produce feasible iterates once the algorithm has converged. This is different than so called "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.

Any time algorithms is not my term.

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 (CITE)
- Filter method

  - The first filter methods in DFO were introduced in 2004 CITE within the context of pattern searches
  - Since then, progress has also been made in a trust region framework CITE, CITE, CITE

- Active Set (CITE)
- Augmented Lagrangian (CITE)
- Penalty
  - Abramson & Audet, 2006
  - Abramson et al. 2009c
  - audet et al. 2008b
  - Audet & Dennis, 2006
  - sequential penalty merit functions Liuzzi et al., 2010
  - smoothed exact $l - \infty$ penalty function Liuzzi & Lucidi, 2009
  - exact penalty merit function Fasano, Liuzzi, Lucidi, & Rinalsdi, 2014; Gratton & Vicente, 2014
- Progressive Barrier
  - Audet & Dennis, 2009
- Interior Point (CITE)

1.2.3 Overview of the rest of the paper

In the remaining pages, we will discuss DFO background by discussing high level characterizations of DFO methods and some of the issues these methods face. We will finish the background with a brief literature review before discussing one classical NLP algorithm I converted to a derivative free context. Finally, we will briefly mention some of the interesting specializations that can be pursued.

2 Background

2.1 Derivative free 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 to (symbolically?) calculate derivatives. However, the nature of the code or problem can make this very difficult: for example with combinatorial problems that rely heavily on if statements. Is this accurate?

2.1.2 Direct search

Another approach is to use direct search methods that do not explicitly or implicitly 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. Another direct search method is Nelder Mead-it, 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 information because they do not use derivative information provided through the function evaluations. This means that they can also lack fast convergence rates.

(0th derivative)

2.1.3 Model based methods

In this paper, we are concerned with model based methods. These typically minimize a model function that only approximate the objective and constraints. The model functions are chosen to accurately represent the original function, but allow for derivative information to be calculated easily.

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.

They work by evaluating functions on a set of sample points to construct local models of the functions. We will see several examples in what follows.

Kriging seems to be another popular model function, but I usually see it used within global optimization.

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 $\nabla f(x) \approx \left(\frac{f(x+he_i)-f(x-he_i)}{2h}\right)_i$ for some small h where $e_i = (0,\dots,0,1,0,\dots,0)^T \quad \forall\ 1 \le i \le n$ is the unit vector with 1 in its ith component. This may work well, but can have issues with unlucky iterates (CITE). I will remove this unless I find there reference that claims this. 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.

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 $\phi_i : \mathbb{R}^n \to \mathbb{R} \quad \forall 1 \le i \le d_1$ 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 $Y = \{y^1, y^2, \dots, y^{d_2}\}$, evaluating $f = (f_1 = f(y^1), f_2 = f(y^2), \dots, f_d = f(y^{d_2}))^T$ and forcing model agreement with the original function $f(x)$ by ensuring

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

When $d_1 = d_2$ this is called interpolation and equality is desired with . When $d_1 < d_2$ this is called underdetermined interpolation and a minimum norm solution is frequently requested. 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 CITE. 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 2.1.3 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 $0 \le n_i \le p$ with $\sum_{i=1}^n n_i \le p$ and inverting the corresponding Vandermonde matrix.

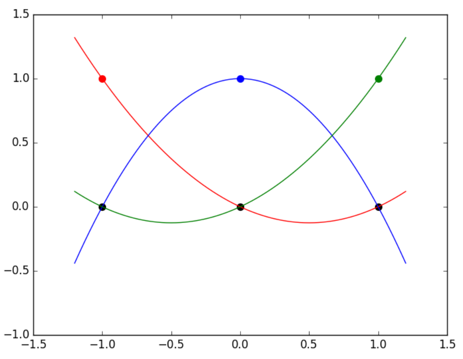
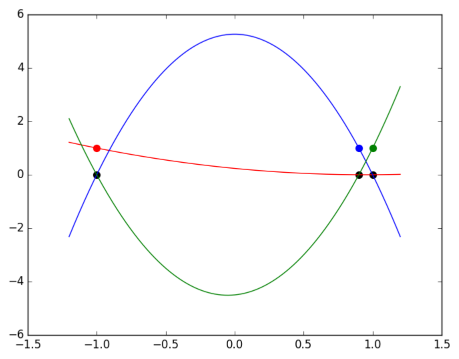
Newton's Fundamental polynomials are also used, and follow a similar pattern. However, they maintain different orders of polynomials within the basis: a single constant value, a set of $n+1$ linear polynomials, $n + \binom{n}{2}$ quadratic functions, and so on. Radial basis functions may have some intuitive 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.

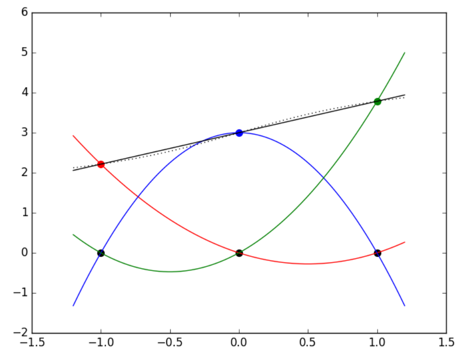
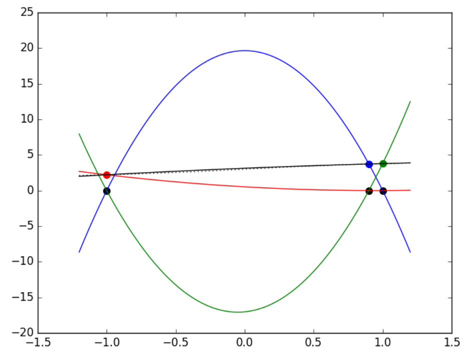
Sampling issues These methods have issues with poor sampling choices. The two most important aspects of the sample points are their geometry and proximity.

Geometry For geometry, regression based methods require that set the function is evaluated at must be Λ-poised for a fixed constant Λ. 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 a poised set with those of an ill poised set.

Within the first set of pictures 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 see that the Lagrange polynomials we find the second set of polynomials.

We that 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 (, or the volume more generally?).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.

Within noisy optimization, this gives rise to several more problems. If you think there is something else about proximity that I should mention, please let me know.

2.1.4 Model-based, Trust Region Methods

I know: still hard to follow.

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 minimum becomes the trial point. The objective is evaluated at the trail(why not?)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. Define a model function $m_k(x)$.
   One such choice is $m_k(x) = f(x^{(k)}) + \nabla f(x^{(k)})^T(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T \nabla^2 f(x^{(k)})(x - x^{(k)})$
   - $\nabla f(x^{(k)})$ and $\nabla^2 f(x^{(k)})$ must be approximated
   - There are 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: $s^{(k)} = \arg\min_{s \in B_{x^{(k)}}(\Delta_k)} m_k(x^{(k)} + s)$
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

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

2.1.5 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 steepest 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 an costly trust region subproblem is acceptable if it allows us to avoid even more expensive function evaluations.

2.1.6 Literature Review Should this still be Background? Many of the names in the outline look like they were intended to be changed.

The original filter method was proposed by Gould in [2]. The motivation for the filter method was that the algorithm does not need to tune any parameters as in penalty or merit methods.

A recent paper from September 2016 [1] implements a derivative-free trust region filter method for solving the the program given above (REFERENCE). 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 $y = d(z)$.

In the 2006 paper [3], 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.

In Brekelman's paper [4], a trust region filter method is developed to minimize function evaluations by constructing linear model functions. This paper also uses experimental designs to choose following iterates.

Within [5] and [6] Biegler uses a filter method to ensure global convergence within a line search framework. We experimented with this before deciding combining line search with the derivative-free trust region approach was not a natural approach. (There have been attempts to employ both of these frameworks in [7].)

Within [8] derivative-free methods are developed in detail. This 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.

Within [9] Toint generalizes the filter method with the notion of a trust funnel. This is for glass box functions as it does not include any derivative-free methods.

Within [10] a sequential quadratic programming method is applied to the filter method?

Colson has also applied filter techniques to derivative-free optimization in his 2004 Ph.D. thesis [11]. This focuses on bilevel programming.

I based my algorithm on the trust region filter method described in [12]

the two additional review papers.

Classical NLP Algorithm	Model based approach	Pattern Search
SQP Filter	Yes	Maybe
Trust Funnel	Maybe	Maybe
Penalty Methods	Maybe	Maybe
Barrier Methods	Maybe	Maybe
Augmented Lagrangian	Maybe	Maybe
Interior Point Algorithms	Maybe	Maybe
Filter Methods	Maybe	Maybe

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 converted by (). (delete this: We first describe the classical algorithm before introducing the changes required for derivative free optimization.)

3.1 SQP Filter method

We will be considering problems of the form

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

where $f : \mathbb{R}^n \to \mathbb{R}$, and each $g_i : \mathbb{R}^n \to \mathbb{R}$, $h_i : \mathbb{R}^n \to \mathbb{R}$. It will be convenient to write $g(x) = (g_1(x), g_2(x), \ldots, g_{|\mathcal{I}|})^T$ and $h(x) = (h_1(x), h_2(x), \ldots, h_{|\mathcal{E}|})^T$.

We work within a trust region, sequential quadratic programming framework with interpolating model functions that uses a filter method introduced by Fletcher. For our algorithm, we assume that all functions are derivative-free: all functions f, g, h are evaluated by a single call to a black box function $d(x) = (f(x), g(x)^T, h(x)^T)^T$.

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.

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 $m_g^k(x)$ and $m_h^k(x)$ to approximate the first derivatives of $g(x)$ and $h(x)$. Namely, at an iterate x^k, we let $g^k = \nabla m_f^k(x^k)$ be the gradient of the objective at x^k. We also define the $c_{ineq}^k = g(x^k)$ and let $A_{ineq}^k = \nabla m_g^k(x^k)$ be the jacobian of the inequality constraints. Likewise, we let $c_{eq}^k = h(x^k)$ and let $A_{eq}^k = \nabla m_h^k(x^k)$ be the jacobian of the inequality constraints.

We will also need to compute the hessian of the Lagrangian. To find the Lagrangian, we let A^k and c^k contain A_{eq}^k and c_{eq}^k as well as any rows of A_{ineq}^k and c_{eq}^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 $g_i(x^k) \geq 0$. We then solve the system

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

for λ and compute

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

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 attempts to project 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:

$$\begin{aligned} n_k = \arg\min_n \|n\|^2 & \\ s.t. \qquad c_{eq}^k + A_{eq}^k n = & \ 0 \\ c_{ineq}^k + A_{ineq}^k n \leq & \ 0 \\ \|n\|^2 \leq & \ \Delta_k^2 \end{aligned} \tag{4}$$

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

$$\|n\| \leq \kappa_\Delta \Delta_k \min\{1, \kappa_\mu \Delta_k^\mu\} \tag{5}$$

for some fixed constants $? < \kappa_\Delta <?$, $? < \kappa_\mu <?$, $? < \mu <?$.

If this stronger condition is satisfied, we say that the program is *compatible*. In this case, we are able to compute a tangential step t_k

$$\begin{aligned} t^k = \arg\min_t \quad & (g^k + H_k n^k)^T t + \frac{1}{2} t^T H_k t \\ s.t. \quad & c_{eq}^k + A_{eq}^k t = 0 \\ & c_{ineq}^k + A_{ineq}^k t \leq 0 \\ & \|n^k + t\|^2 \leq \Delta_k^2 \end{aligned} \tag{6}$$

to decrease the objective value while staying within the trust region and the feasible region.

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

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

which measures the current constraint violation.

The filter works by ensuring that all new iterates are nondominated with respect to all previous iterates when the problem is viewed as a multi-criteria optimization problem min (θ, f). However, simply ensuring that new points are nondominated does not provide sufficient progress, we must ensure that the objective decreases by a greater amount when the current iterate is far from the feasible region. Therefore, we introduce a constant $? < \gamma_\theta <?$ and require either

$$\theta(x_k) \leq (1 - \gamma_\theta)\theta_i \tag{7}$$

or

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

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 *admissible* to the filter. As we iterate, we add values of θ^k and f^k to the filter.

We must also ensure the we never add a pair (θ_k, f_k) with $\theta_k = 0$, as this requires all subsequent points to remain feasible.

3.1.4 f-type versus θ-type

When steps decrease f significantly more than θ the steps are considered f type. Likewise, θ-type steps are steps that significantly reduce the constraint violation.

I thought that the paper I read about the convergence rate said they introduced a new criteria to this that made it more efficient. I guess it is not really needed, I should probably delete this.

3.1.5 Restoration Step

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

While performing the restoration step, we place constraints in the objective by minimizing the squared norm of θ with no regard to $f(x)$. 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. At the end of each iteration, we also check if the new iterate and trust region radius is 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 the problem we are trying to solve.

3.1.6 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, we can terminate the algorithm.

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

$$\begin{aligned} \nabla f(x^\star) + \sum_{i\in\mathcal{I}} \lambda_i \nabla g_i(x^\star) + \sum_{i\in\mathcal{E}} \mu_i \nabla h_i(x^\star) &= 0 \\ g(x^\star)_i \mu_i &= 0 \quad \forall i \in \mathcal{I} \\ g(x^\star) &\le 0 \\ h(x^\star) &= 0 \end{aligned}$$

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

This suggests the following definition:

$$\begin{aligned} \chi = \quad & |\min_t \langle g_k + H_k n_k, t\rangle| \\ A_{eq} t \quad & = 0 \\ c_{ineq} + A_{ineq} t \quad & \le 0 \\ \|t\| \quad & \le 1 \end{aligned} \tag{9}$$

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

3.1.7 Not here yet

Talk about

$$m_k(x_k) - m_k(x_k + s_k) \ge \kappa_\theta \theta_k^\psi \tag{10}$$

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 list coming soon
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 5 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 9

   - 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 6

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

   If 10 is satisfied, then add (θ_k, f_k) to the filter,

   - Add (θ_k, f_k) to the filter
   - Increment k

- Go to step 2

decrease the trust region radius and 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 7 or 8, then

   - Add (θ_k, f_k) to the filter
   - Increment k
   - Go to step 2

8. Compute ρ according to 2

   - If ρ is small, then decrease the trust region radius and go to step 2
   - If ρ is intermediate, then accept the trial point $x_{k+1} = x_k + s_k$.
   - If ρ is large, and χ is small, then decrease the trust region radius.
   - If ρ 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 changed something else! We changed:

- the order of evaluated the expected decrease and evaluating ρ
- the projection on to the feasible region (this was ambiguous in the original paper, but they did not solve it exactly...)

3.4 Pseuodcode of algorithm

Psuedo code

Algorithm 1 Filter Trust Region Search

```
1: procedure TRUST REGION FILTER
2:     initialize
3:     $k = 0$
4:     choose an $x_0$
5:     while $k < maxit$ do
6: main loop:
7:         ensure poisedness, possibly adding points to the model
8:         Compute $m_k, g_k = \nabla m_k(x_k), c_k, A_k, f_k = f(x_k), \mathcal{A}, \theta_k$
9:         Solve:
10:                  $\nabla^2 m_k(x_k)d + A_k^T\lambda = g_k$
                     $A_k d \quad = c_k$
11:        $H_k \leftarrow \nabla^2 m_k(x_k) + \sum_i \lambda_i \nabla^2 c_{ik}$
12:        $n_k \leftarrow \arg\min_n\{\|n\|^2 | c_{eq} + A_{eq}n = 0 \wedge c_{ineq} + A_{ineq}n \leq 0 \wedge \|n\|^2 \leq \Delta_k\}^2$
13:        $\chi_k \leftarrow |\min_t\{\langle g_k + H_k n_k, t\rangle | A_{eq}t = 0 \wedge c_{ineq} + A_{ineq}t \leq 0 \wedge \|t\| \leq 1\}|$
14:        if constraint violation $= 0 \wedge \chi = 0$ then
15:            if $tol < \Delta_k$ then
16:                reduce $\Delta$: $\Delta_{k+1} \leftarrow$ some $\in [\gamma_0\Delta_k, \gamma_1\Delta_k]$
17:                $k \leftarrow k + 1$
18:                go to main loop
               success
19:        if Feasible region $\neq \emptyset \wedge \|n\| \leq \kappa_\Delta \Delta_k \min\{1, \kappa_\mu \Delta_k^\mu\}$ then
20:            $t_k \leftarrow \arg\min_t\{(g_n + H_k n_k)^T t + \frac{1}{2}t^T H_k t | c_{eq} + A_{eq}t = 0 \wedge c_{ineq} + A_{ineq}t \leq 0 \wedge \|s\| \leq \Delta_k\}$
21:            $s_k \leftarrow t_k + n_k$
22:            if $m_k(x_k) - m_k(x_k + s_k) \geq \kappa_\theta \theta_k^\psi$ then
23:                add $x_k$ to filter
24:                reduce $\Delta$: $\Delta_{k+1} \leftarrow$ some $\in [\gamma_0\Delta_k, \gamma_1\Delta_k]$
25:                go to main loop
26:            // Here we evaluate new $c$ and $f$ at $x_k + s_k$
27:            if $x_k + s_k$ is acceptable: $\theta(x_k + s_k) \leq (1 - \gamma_\theta)\theta' \vee f(x_k + s_k) \leq f' - \gamma_\theta\theta' \forall (f', \theta') \in$ Filter then
28:                $\rho = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}$
29:                if $\rho < \eta_1$ then
30:                    reduce $\Delta$: $\Delta_{k+1} \leftarrow$ some $\in [\gamma_0\Delta_k, \gamma_1\Delta_k]$
31:                    $k \leftarrow k + 1$
32:                    go to main loop
33:                else if $\rho > \eta_2$ then
34:                    if $\|s\| < \frac{\Delta_k}{2}$ then
35:                        reduce $\Delta$: $\Delta_{k+1} \leftarrow$ some $\in [\gamma_0\Delta_k, \gamma_1\Delta_k]$
36:                    else
37:                        increase $\Delta$: $\Delta_{k+1} \leftarrow$ some $\in [\Delta_k, \gamma_2\Delta_k]$
38:                $x_{k+1} \leftarrow x_k + s_k$
39:            else
40:                reduce $\Delta$: $\Delta_{k+1} \leftarrow$ some $\in [\gamma_0\Delta_k, \gamma_1\Delta_k]$
41:                $k \leftarrow k + 1$
42:                go to main loop
43:        else
44:            add $x_k$ to filter
45:            compute new $r$ (restoration step) and $\Delta$
46:            if impossible to restore then fail
47:            $x_{k+1} \leftarrow x_k + r$
48:        $k \leftarrow k + 1$
```

3.5 Computational results and Testing libraries

In order to compare the algorithms we develop, we will use the

- SDPEN A Sequential Penalty Derivative-free Method for Nonlinear Constrained Optimization problems Copyright (C) 2011 G.Liuzzi, S.Lucidi, M.Sciandrone
- DAKOTA/PATTERN
- tomlab lgo, ego?
- nomad

on test sets

- dfovec
- cute/st

http://thales.cheme.cmu.edu/dfo/comparison/dfo.pdf

3.6 I added:Generalizations

Although we began with the filter method, recent progress has been done by other researchers with this method CITE. However, there are some generalizations we could pursue with this:

- We can generalize (relax) the possible steps by only requiring that new iterates are admissable with respect to a multi objective filter of the form $(f(x), [g_1(x)]_+, \dots [g_{|\mathcal{I}|}(x)]_+, |h_1(x)|, \dots, |h_{|\mathcal{E}|}|)$. The positive part might be bad here (do not add points that are zero.)
- We could relax the condition that all iterates have to remain feasible with respect to all constraints except $y = d(z)$.
- The authors only considered linear models, we can use quadratic models.

4 Potential research

4.1 I added:Goals

We could 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 involve the "stiffness" of .... 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.

4.2 Types of constraints

4.2.1 Thin constraints

Again, we only consider local search algorithms for this problem that seek a first or second order stationary point. One interesting topic within the research is different shapes of the feasible regions near critical points. Although constrained DFO has attracted much interest in terms of papers, there are still few libraries for constrained derivative free optimization. An area that may not have seen much progress is that of the 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 may be to redefine the poisedness as the maximum value of the model function over only the feasible region intersected with the trust region.

4.2.2 Hidden constraints

Although hidden constraints have been considered, we are not aware of much work done to solve these. No library?

4.3 Parallelization

Parallelization is another topic receiving attention. 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.

NOMAD? SNOBFIT hopstack?

CITE Schnabel [99]

4.4 Sparse Hessian or Jacobian

While using a Newton based approach, the calculation of the hessian can be expensive due to the dimensions involved. Research could be done studying for exploiting sparsity within the derivatives of the black box functions.

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

This currently has no place in the paper:

Within heuristics behind Applying DFO techniques to algorithms that currently use random sampling to tune parameters Machine learning techniques are a hot topic, and some techniques have very complicated objectives for which derivative information is not known.

Should probably include:

$$-A^T[AA^T]^{-1}c$$

References

[1] 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.

[2] 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.

[3] R. Fletcher, S. Leyffer, R. Fletcher, and S. Leyffer, “A brief history of filter methods,” tech. rep., 2006.

[4] 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.

[5] 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.

[6] 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.

[7] J. Nocedal and Y. Yuan, “Combining trust region and line search techniques,”

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.