Derivative Free Model-Based Methods for Local Constrained Optimization

Trever Hallock

May 4, 2018

Contents

1 Introduction

This paper will discuss 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 approaches for optimization over thin feasible regions. The focus is on model-based trust region algorithms for local search within constrained derivative free optimization. Specifically, we are interested in problems with thin, hiddenconstraints. Narrow constraints introduce numerical instability and make it hard to to model the relavent functions.

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

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

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 citation

1.1.3 Where Problems come from

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 citationwhen 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

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 $e_i = (0, \dots, 0, 1, 0, \dots, 0)^T \quad \forall\, 1 \le i \le n$ 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 $\phi_i : \mathbb{R}^n \to \mathbb{R} \quad \forall 1 \le i \le d_1$ 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 $Y = \{y^1, y^2, \ldots, y^{d_2}\}$, 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 citationOne 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 $0 \le n_i \le p$ with $\sum_{i=1}^n n_i \le p$ and inverting the corresponding Vandermonde matrix.

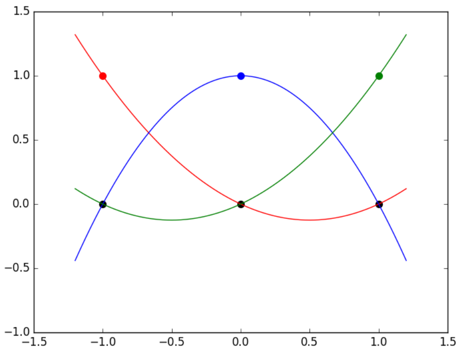
Newton's Fundamental polynomials are also used, and follow a similar pattern. However, they maintain different orders of polynomials within the basis: a single constant value, a set of $n+1$ linear polynomials, $n + \binom{n}{2}$ quadratic functions, and more for higher order polynomials. Radial basis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

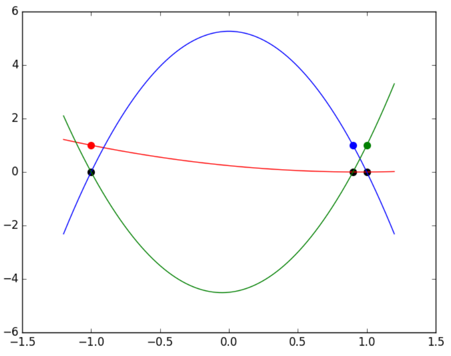
Model functions are usually chosen to be fully linear or fully quadratic: terms describing how the model's error grows as a function of the trust region radius.

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

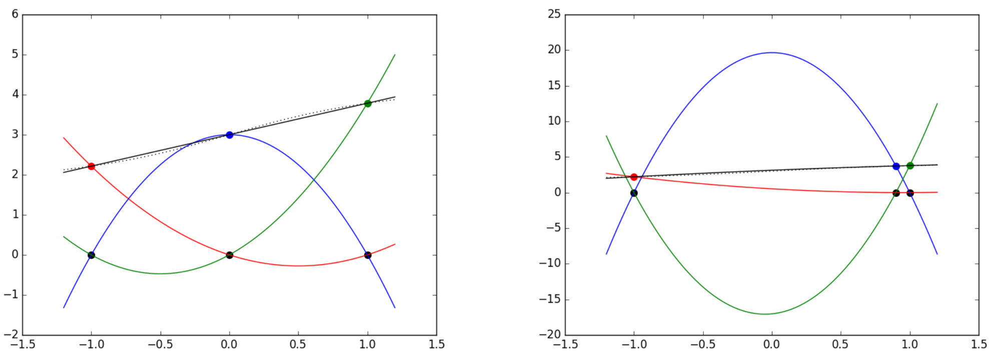
Geometry For geometry, regression based methods require that the set the function is evaluated at must be Λ-poised for a fixed constant Λ. If the set is Λ-poised for a certain trust region radius, then the maximum absolute value of any Lagrange polynomial over the trust region is one. This ensures that 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

3 Experiments

3.1 Setting

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 $c(x) = (c_{\mathcal{I}}^T(x), c_{\mathcal{E}}^T(x))^T$.

It will also be convenient to let $F^k = \{x \in \mathbb{R}^n | c_i(x) \leq 0 \forall i \in \mathcal{I} \wedge c_i(x) = 0 \forall i \in \mathcal{E}\}$ be the feasible region.

We work within a trust region, sequential quadratic programming framework with interpolating model functions. 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 also assume that we are not able to evaluate points outside the feasible region. This introduces a new type of constraint: it is not quite a hidden constraint because we do observe function values within the feasible region. However, it does implies similar difficulty to construct the model function near the boundary of the feasible region because we have limited sampling ability.

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 $A_{\mathcal{E}}^k = \nabla m_{\mathcal{E}}^k(x^k)$.

3.1.2 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:

$$\begin{aligned} \chi = & \quad |\min_t \langle g_k + H_k n_k, t\rangle| && (4) \\ A_{\mathcal{E}} t & = 0 && (5) \\ c_{\mathcal{I}} + A_{\mathcal{I}} t & \leq 0 && (6) \\ \|t\| & \leq 1 && (7) \end{aligned}$$

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

3.1.3 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

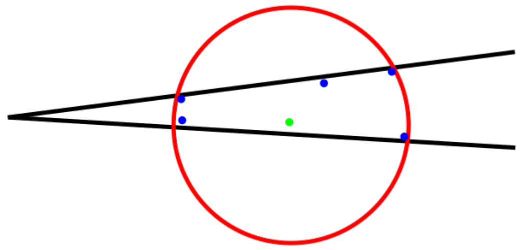
$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_\theta \theta_k^\psi. \tag{8}$$

4 Algorithms

5 naive

One simple approach to handling hidden constraints is to simply avoid letting points fall outside the feasible region. Within this model, we maintain the same policies for updating the trust region radius. However, within the LU factorization algorithm to select new points, we add to the trust region subproblem that the new points must also lie within the current model of the trust region.

This can happen when the feasible region intersect the trust region is small compared to the trust region. As the number of dimensions grows, this can become worse.



6 Bumping ξ

Within the classical algorithm, we used ξ as a lower bound of the pivot values of the Vandermode matrix. When requiring points to live within the trust region intersect the feasible region, it can happen that even after replacing a point, we still have not satisfied this bound. One way to handle this is to simply allow ξ to decrease until a threshold is reached. (The threshold is for maintaining a fixed lambda.)

7 Ellipse

We decided to keep the trust region entirely within the feasible region because of the high lambda when only requiring the points to lie in the feasible region. In order to deal with the numerical problems from having a circular trust region, we can map the feasible region back to a circle. More specifically, at iteration k, we select an ellipse center μ^k, a scaling factor π^k and positive definite matrix Q^k to define and ellipse $E_k = \{x \in \mathbb{R}^n \| \pi^k - \frac{1}{2}(x-\mu^k)^T Q^k (x - \mu^k) \geq 0\}$. We then map this back to the origin with the affine transformation $T^k : \mathbb{R}^n \to R$ $T^k(x) = \frac{\sqrt{2}}{\pi^k} L^k(x - \mu^k)$ where $L = cholesky(Q)$. This will allow us to construct a lambda poised set within narrow trust regions.

There are a number of issues to be solved to define this ellipse:

- How do we ensure that $x^k \in E^k$?
- How do we choose the center of the ellipse μ^k?
- Is $E_k \subset F_k$?

We have tried a couple of ways of ensuring the current iterate is within the trust region, $x^k \in E^k$. If we do not include the current iterate within the interior , we likely lose sufficient reduction. This can be done by either expanding the radius of the ellipse, or by including the original point as a constraint for the ellipse problem.

In order to include the original point as a constraint, we simply add the following constraint to the definition of the ellipse:

$$\pi^k - \frac{1}{2}(x^k - \mu^k)^T Q^k (x^k - \mu^k) \geq 0.$$

However, the optimization problem we construct to find Q^k solves for ${Q^k}^{-1}$, so that adding this constraint is a constraint on the inverse of Q^k.

The alternative is to scale Q by a constant. To do this, we use the scaling factor π^k by defining it to be

$$\pi^k = \max\{1, \frac{1}{2}(x^k - \mu^k)^T Q^k (x^k - \mu^k)^T\}$$

and let the ellipse be:

$$\{x \in |1 - \frac{1}{2\pi^k}(x - \mu^k)^T Q (x - \mu^k) \geq 0\}$$

However, this means that we do not ensure $E_k \subset F_k$ so that the trust region subproblem must contain constraints for both the ellipse and the feasible region.

The are some concerns we have to consider while defining the ellipse. One major concern is that we can accidently define the ellise in a way that forces the trust region away from the desired drection. Also, unless we include points outside the ellipse, we cannot find a second order solution because the ellipse can only be tangent to the constraint. We also want consecutive ellipse to share volume, in order to avoid evaluating as many points as possible. (So far, we have been re-evaluating the set of trial points each iteration.) Finally, we have the problem of computing the ellipse once given a suitable definition.

7.1 Finding the maximal E_k given μ^k

Within this paper, we first solve the problem of finding the maximal ellipse given the center, and then perform a simple search over different centers of the ellipse. Because of this, we will first show how to find an ellipse with maximal volume given a fixed center. For now, we also let $s^k = 1$.

Given a polyhedron P defined by an $m \times n$ matrix A,

$$P = \{x \mid Ax \leq b\},$$

and we wish to find the largest ellipse $E \subset P$ centered at a point μ^k within this polyhedron.

We can shift the polyhedron by letting $\bar{b} = b - A\mu^k$ and letting $x \to x - \mu^k$ so that the polyhedron becomes

$$P = \{x \mid Ax \leq \bar{b}\}$$

The ellipse can then be centered at zero, and defined by a positive semi-definite matrix $Q \succeq 0$:

$$E = \{d \mid \frac{1}{2}d^T Q d \leq 1\}.$$

where $Q = Q^T$. We can define the auxiliary function

$$f(x) = \frac{1}{2}x^T Q x$$

so that this becomes

$$E = \{d \mid f(d) \leq 1\}.$$

At the locations where the ellipse intersects the polyhedra with minimum value of f, the gradients of f must be orthogonal to the faces. Each face is defined by at least one $A_i x \leq \bar{b}_i$ where A_i is the ith row of A for $1 \leq i \leq m$. This implies that if $d^{(i)}$ is such a point, then

$$\nabla f(d^{(i)}) = \lambda_i A_i \quad \forall 1 \leq i \leq m$$

for some λ_i. This means

$$Qd^{(i)} = \lambda_i A_i \quad \forall 1 \le i \le m$$
$$d^{(i)} = \lambda_i Q^{-1} A_i \quad \forall 1 \le i \le m$$

We also know that

$$A_i^T d^{(i)} = \bar{b} \quad \forall 1 \le i \le m$$
$$A_i^T \lambda_i Q^{-1} A_i = \bar{b} \quad \forall 1 \le i \le m$$
$$\lambda_i = \frac{\bar{b}}{A_i^T Q^{-1} A_i} \quad \forall 1 \le i \le m$$

so that

$$d^{(i)} = \lambda_i Q^{-1} A_i \forall 1 \le i \le m$$
$$d^{(i)} = \frac{\bar{b}}{A_i^T Q^{-1} A_i} Q^{-1} A_i \quad \forall 1 \le i \le m.$$

Finally, we need that for each i, $f(d^{(i)}) \le 1$:

$$\frac{1}{2}(d^{(i)})^T Q d^{(i)} \le 1$$
$$\frac{1}{2}(\frac{\bar{b}}{A_i^T Q^{-1} A_i} Q^{-1} A_i)^T Q \frac{\bar{b}}{A_i^T Q^{-1} A_i} Q^{-1} A_i \le 1$$
$$\frac{1}{2} \frac{1}{A_i^T Q^{-1} A_i} \bar{b}^T A_i^T Q^{-1} Q \frac{\bar{b}}{A_i^T Q^{-1} A_i} Q^{-1} A_i \le 1$$
$$\frac{1}{2} \frac{1}{A_i^T Q^{-1} A_i} \bar{b}^T \frac{\bar{b}}{A_i^T Q^{-1} A_i} A_i^T Q^{-1} A_i \le 1$$
$$\frac{1}{2} \bar{b}^T \frac{\bar{b}}{A_i^T Q^{-1} A_i} \le 1$$
$$\frac{1}{2} \bar{b}^T \bar{b} \le A_i^T Q^{-1} A_i$$
$$A_i^T Q^{-1} A_i \ge \frac{1}{2} \bar{b}^T \bar{b}$$

Thus, the maximal ellipse is defined by

$$\sup_{Q \succeq 0} \det(Q^{-1})$$
$$s.t. \quad A_i^T Q^{-1} A_i \ge \frac{1}{2} \bar{b}^T \bar{b}$$

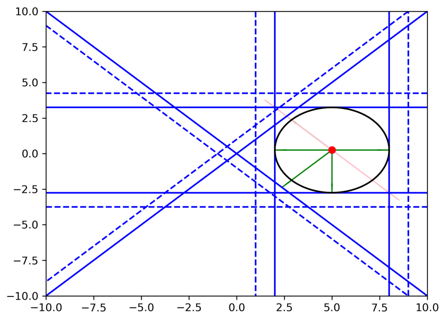
This problem includes a maximization of a determinant. In order to ensure that the trust region goes to zero, we must also ensure that the maximum eigenvalue of the matrix is bounded. Thus, for some bound M^k we define Q^k as

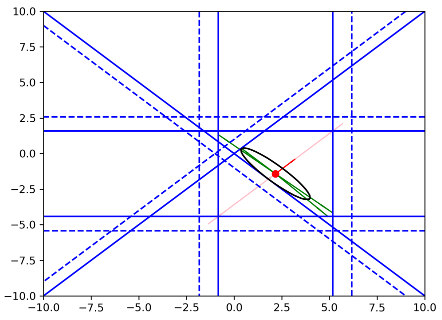
$$Q^k = V(\mu_k) = \sup_{Q \succeq 0} \det(Q^{-1}) \tag{9}$$
$${A^k}_i^T Q^{-1} {A^k}_i \ge \frac{1}{2}(b^k - A^k \mu^k)^T (b^k - A^k \mu^k) \tag{10}$$
$$eig(Q)_i \le M^k \qquad \forall i \in \mathcal{I} \cup \mathcal{E} \tag{11}$$
$$\pi^k - \frac{1}{2}(x^k - \mu^k)^T Q^k (x^k - \mu^k) \ge 0 \tag{12}$$

if we wish to include the original point explicitly. This gives rise to the trust region sub problem of

7.2 Stationary center

The simplest approach is to not change the center of the ellipse, but instead let $\mu^k = x^k$. In the example below, we begin with a trust region that must be very small to lie within the feasible region, and note that if we even keep the same center, we are still able to search towards the vertex of the feasible region. After solving the trust region subproblem in the first image, the iterate is near a constraint within the next iteration. However, the ellipse elongates along an axis parallel to this constraint.



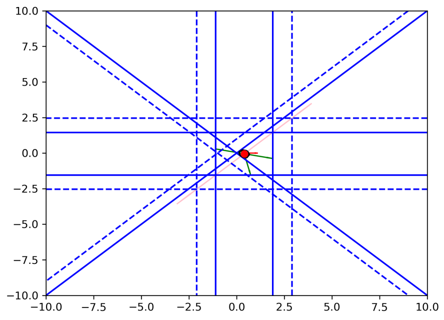


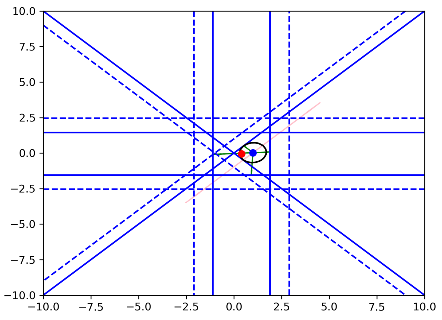
7.3 Search everything

Another simple approach is to define the ellipse as to maximize the area within the ellipse. That is,

$$\mu^k = \sup_{\mu \in F^k} V(\mu))$$

This has the advantage that it captures much of the feasible region. However, one problem with this search is that it can force the trust region away from the desired direction.





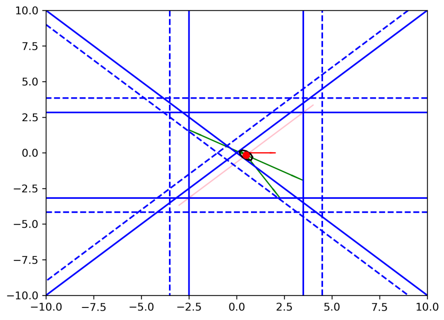
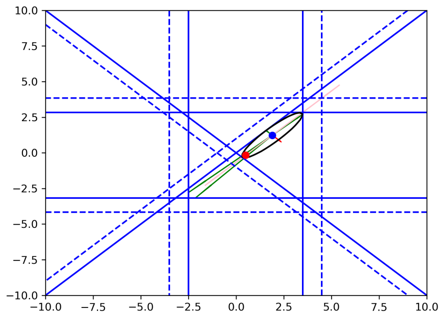
One attempt to fix this problem is by limiting the search direction for the center of the ellipse.

7.4 Line searches

Although the minimum of problem P can appear anywhere within the trust region, there are some reasons for expecting it to be at a "vertex." If it lies on the interiorer, there is little need for using constrained approaches once near the solution.

The ellipse with maximum volume, however, tends to lead away from vertices because there is more space away from vertices. One way of trying to leave the direction towards a vertex within the trust region, while still allowing to incease the ellipse volume is by limiting the search for the new center to lie on line segments starting at the current iterate.

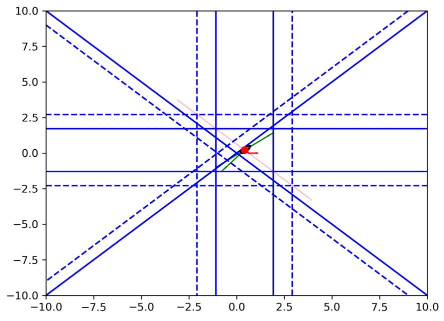
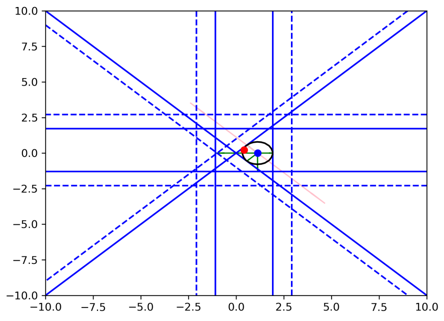
For example, our first attempt was to simply search a line directed away from the closest constraint. This has obvious problems though, as we should avoid letting the new center get closer to another constraint:

To fix this, we break the line into segments based on the nearest constraint. We find the center that maximizes the volume of the ellipse after restricting it to lie on a ray perpendicular to the nearest constraint. Nearest here means means the constraint with the smallest distance from the iterate to the projection of that iterate onto the constraint. If there are two constraints that are both closer than all others, then we follow a ray whose points are equidistant from both of these constraints.

We search along this away from the current iterate until we reach a point that has the same distance to another constraint as the nearest to the iterate. We could then in general continue to follow line segments by always travelling away from the nearest constraints. However, after we include enough constraints, we will eventually once again head away from a vertex.

This means that we can define the a class of searches that each use a different number of line segments to search. In this paper, we consider one and two line segments.

8 Results

the example problem I have been working with linear constraints

Algorithm	Iterations	Evaluations
Search everything (include the original)	24	169
Search everything	19	106
One line segment	21	86
Two line segments	23	115