1 Title Page

- Hello, Thank you all for attending
- I am Trever Hallock, studying under Steven Billups, and I am here to talk about some algorithms we developed.
- I hope you enjoy the talk.

2 Introduction

- Let's start with what this presentation covers.
- For my thesis, I developed local search, model-based algorithms, for constrained derivative-free optimization.
- The significance of our work is that we address the case of **unrelaxable constraints**.
- Unrelaxable constraints are constraints that provide no meaningful information at infeasible points.
- This means the algorithm must construct accurate models with limited sample point choices.
- In the first draft I provided, I used the word "partially-quantifiable"; however, after receiving feedback, the correct terminology is unrelaxable.

3 Formulation

- We begin by describing Derivative Free Optimization, or DFO for short.
- We are interested in minimizing an objective f, subject to several constraints c sub i.
- The problem is derivative free, because no derivative information is available for some functions: these functions are called black box functions.
- For example, this may arise when the objective and constraints are outputs from an expensive simulation.
- In our case, we assume that **all** functions are black box.
- Not only are they black-box but each constraint is also unrelaxable.
- Because of this, we have removed the usual equality constraints within the formulation.

4 Applications

- Problems with unrelaxable constraints ocurr in several situations.
- Digabel and Wild contributed a taxonomy of constraint types, in which they promoted the term unrelaxable constraint, and provide examples such as the following scenarios.
- Sometimes physical models are not meaningful outside the feasible region, or a simulation may take a long time to converge.
- For example, simulation results may not be meaningful when a quantity representing a fluid's concentration level becomes negative, or when a quantity representing an amount of water being pumped is negative.
- Also, some decision variables such as chronological events in time must be ordered.

- One interesting example is the inverse transport problem solved by Armstrong and Favorite in 2016 [?].
- They note that one of the numerical integration methods used within their algorithm produces large errors or simply does not converge in some regions.
- These hidden constraints are also unrelaxable, because no information is provided when the subroutine does not converge.
- However, the error may quantifiably increase while approaching infeasibility, meaning it would possible to model the constraint's boundary.

5 Why are unrelaxable constraints hard?

- Well, ok, unrelaxable constraints exist, what makes them difficult?
- We will discuss this in more detail within the background section, but **one** issue is that model-based methods need feasible sample points to construct model functions.
- These images illustrate the challenge.
- Without constraints, classic algorithms can choose sample points within a spherical region to ensure these accurate models.
- However, on the right, we see that with unrelaxable constraints, sample points may not be allowed to spread out over the trust region, which can hurt the model's accuracy.
- This challenge is even present within our first algorithm, which only handles linear constraints.

- When we extend the algorithm for linear constraints to general constraints, another problem arises: the algorithm must account for errors in the modelled feasible region.
- In this image, we see that the by constructing red linearizations of the black constraints, we create an inaccurate model of the feasible region, which lies between the two black constraints.
- The linearization overestimates the convex constraint on the left, meaning an algorithm may believe a point is feasible, when it is not.
- Conversely, the linearized feasible region does not allow evaluating some points that are feasible with respect to the concave function on the right.
- In a perfect world, we would avoid all infeasible evaluation attempts.
- However, because of these innaccuracies, our algorithm may attempt to evaluate infeasible points.
- When it does, because the constraints are unrelaxable, it will not have access to the objective or constraint values.
- Therefore, it is preferrable to avoid these infeasible evaluation attempts if possible.

6 Table of Contents

- So I have just given the introduction.
- Next we will cover background material relevant for our algorithm.
- Then we will describe the algorithms before making some concluding remarks.

7 Model-based Trust Region Framework

- We will start by listing the key steps within a Model-based, trust region algorithm.
- First, we construct a model near the current iterate by choosing sample points from some sample region.
- Next, the algorithm checks its stopping conditions by comparing a criticality measure and the trust region radius to user defined thresholds.
- We then compute and evaluate a trial point by minimizing the objective's model function over some search region.
- If the trial point is feasible, and provides the expected reduction, we accept it as the new iterate.
- Otherwise, we decrease the trust region radius about the current iterate for more accurate models during the next iteration.
- Let's go into each of these steps in more detail.

8 Model based trust region method

- Now, the first step was building models.
- This means that rather than using the true function while searching for a trial point or checking feasibility, we use some approximation, presumably more efficient or easier to work with.
- We build this model by choosing coefficients for a set of basis functions that we make agree with the true function on a sample set.
- We construct a model for the objective and each of the constraints from the same sample points, and we potentially have a different model for each iteration.
- Here, we denote the iteration with k.
- As you can see, we use a quadratic model, m sub f, for the objective and a linear model, m sub c sub i, for constraint c sub i.
- Once we have these models, we can, for example, choose the trial point is by minimizing the modeled objective over a modeled feasible region.
- In theory, solving optimization sub-problems using these models is simpler than using the original functions.
- The accuracy of the models depends on the sample points used to construct the models.
- We turn to this topic next.

9 Geometry

- The model's accuracy is not only related to the proximity of the sample set, but also to the relative positions of the points in the sample set.
- When the shape, or geometry, of the sample set will create an accurate model of the true function, we describe the sample set as "well poised".

- Here, we can see an illustration of how the poisedness of the sample set affects the model's accuracy.

- The plot on the left shows a model from six points that are nearly perfectly poised.
- Both the model and the true function are plotted, however they are so close that we only see one contour.
- On the right, we see a model of the same function, however the sample points are nearly colinear.
- This results in a model inaccurate near the edges because there are fewer sample points nearby.

- There is a well known model improving algorithm within DFO that can construct well poised points within an ellipsoidal shape.
- However, if a sample region has constraints that are narrow, there may be limited sample point choices.
- Sometimes, there may not even be a set of points that are sufficiently poised over the entire trust region.

10 Stopping Conditions

- The next step of the algorithm computes a criticality measure.
- This is a non-negative value which roughly measures how far a point x is from satisfying the first order necessary conditions for optimality.
- This makes it natural to use within stopping criteria: while the criticality measure is large, the algorithm can still make progress minimizing the objective.
- When it is small, the point may be nearly optimal.
- In our case, we measure the criticality by computing the objective's negative gradient's projection onto the constraints.
- If the negative gradient is zero, or points outside the feasible region, then no progress can be made, and we would have a small criticality measure.
- Our convergence analysis will ensure that the criticality measure tends to zero.

11 Trust region subproblem

- When the current iterate is not optimal, we compute a step bringing us closer to optimality.
- Trust region methods choose the next iterate in the **proximity** of the model's sample points, so that we "trust" our model's accuracy.
- To do this, we constrain the trial point to be within an L infinity ball around the current iterate.
- An L 2 ball is more frequently used, but with linear constraint models, it is convenient to let this be a an L infinity ball so that the trust region subproblem is a minimization over a polytope.
- The width of this ball, labeled delta, is the trust region radius.
- Aside from this additional trust region constraint, this problem is the original DFO problem with the true functions replaced by their model functions.

12 Evaluating the Trial Point

- That last step of the algorithm is to evaluate the trial point.
- If the models were not accurate, the true function value may be either larger or smaller than the models prediction.
- Of course, if it is much smaller, we want to accept this unexpectedly good point.
- But if the objective is larger, we may need more accurate models.
- The quantity we use to measure this is rho sub k: the true reduction divided by the expected reduction.
- When the models agree with the true function values, this is 1.
- However, when rho is small or negative, we need to decrease the trust region radius to provide more accurate models during the next iteration.

13 Feasible Derivative Free Algorithm

- With that background, We now focus on our algorithms.
- Our algorithm for linear constraints implements a template described by Conejo, Karas, Pedroso, Ribeiro, and Sachine in 2013.
- Their framework assumes quadratic or linear models that satisfy certain conditions.
- And their development is general, without specifying details such as how to construct models, or how to solve the trust region subproblem.
- We provide these details using **only feasible function evaluations**.
- These conditions include an efficiency and an accuracy condition, which we will talk about next.
- Also, for the criticaliy measure, the algorithm uses a projection onto an explicitly known, convex feasible set.
- This is great for linear constraints, but it needed modifications for general black-box constraints.

14 Algorithm Assumptions

- Here, we describe these conditions in more detail.
- The efficiency condition requires the algorithm to find a good solution to the trust region subproblem.
- In theory, even if each iterate decreases the objective, such as the blue points in the figure, the iterates may not reach a minimum.
- If each trial point satisfies the efficiency condition, this cannot happen
- Because the efficiency condition ensures not just that each iterate decreases the objective value, but that the reduction is large enough that the iterates do coverge to a critical point such as the green star.
- Notice that the required decrease in the objective depends on the criticality measure: we cannot expect much reduction when we are near a critical point.

- However, this condition is only on the model functions, so we need these to be accurate, to translate this information to the true functions.
- The accuracy condition requires the model's gradient to be close to the true function's gradient.
- By the model's construction, the function value at each sample point is already equal to the model's value, so the correspoding bound on the function value comes for free.

15 The Algorithm for Linear Constraints

- First, we discuss the algorithm for linear constraints.
- Because the linear version is a particular implementation of the algorithm described by Conejo et al, we can show convergence by satisfying the hypothesis we just presented.
- There are well-known algorithms for computing a trial point satisfying the the efficiency condition, so our main concern for linear constraints was the accuracy condition.
- To construct accurate models, we allow sample points to be chosen from any ellipsoidal sample region satisfying certain conditions.
- Because it is ellipsoidal, we can use classic model improving algorithms to construct the sample set after a simple transformation to a spherical region.

16 Feasible Derivative Free Trust Regions

- In the paper, we discuss a number of different strategies for implementing Conejo et al's algorithm.
- We distinguish them by their choices in the three trust regions we use.
- The first of these trust regions is the outer trust region.
- It is determined by the the trust region radius capitol delta sub k, and contains both the other trust regions.
- The sample region is used to construct sample points while building the models.
- A well chosen sample region produces accurate model functions, while still being feasible.
- In the trust region subproblem, we search for a trial point over the search region.
- A good search region should allow reduction in the objective, and must also be feasible.

17 Example

- So here we see an example of the three trust regions.
- The outer trust region is in blue, and the sample region is in black.
- The linearized feasible region is in red, and when we intersect that with the outer trust region we get the search region in dotted magenta.
- To construct sample points, we map the sample region onto the unit ball.
- We then call a model improvement algorithm to find well poised points in the unit ball, and map them back to the sample region for evaluation

18 Sample Region Requirements

- We showed convergence for one particular choice of these trust regions.
- Namely, when then sample region is ellipsoidal, and the search region is the outer trust region intersected with the constraint linearization.
- More precisely, the sample region is an ellipsoid built with a positive definite, symmetric matrix q, a center c, and a radius determined by delta.
- In Lemma 3.4 of our paper, we show the accuracy of models interpolated from a sample set in this ellipsoid depends on the condition number of q.
- Thus, one requirement to satisfy the accuracy condition is a bound on q's condition number independent of the iteration.
- This enusres the ellipsoid is not too skinny.
- Also, the accuracy condition is explicitly stated on the current iterate, so we need the ellipsoid to be near the current iterate.
- This is done by requiring the current iterate to be in the ellipsoid formed without the one half present in this equation.
- Alternatively, we could have required the ellipsoid to be large enough.
- Namely, if the ellipsoid had an axis as long as some percentage of the outer trust region radius, the model would be accurate over the entire trust region.
- The ellipsoid we constructed would have also satisfied this more restrictive condition.
- Lastly, we require the ellipsoid to be feasible.
- This is easier for linear constraints, but when we discuss more general constraints, we only make this requirement for sufficiently small delta.

19 Example Ellipsoid

- In this image, we see an example iteration showing the sample region.
- The outer trust region is in blue, constraint linearizations are in red, and the sample region is in black.
- Usually, the sample region is centered at the current iterate, but notice that in our construction, the ellipsoid may not even include the current iterate.
- In fact, it would be impossible for any feasible ellipsoid to contain this iterate.
- However, it must be close enough that when it is scaled to the dotted ellipsoid, it does.
- Also, the condition number of the matrix defining this ellipsoid must be bounded.

20 Ellipsoid Construction

- There are several possible ellipsoid constructions satisfying our conditions.
- In our paper, we provide a method for one such construction.
- We can compute a direction u, shown in black, that is feasible with respect to all these nearly active constraints.

- We then measure the smallest angle between the feasible direction, depicted in black, and any infeasible direction.
- We construct a second order cone, depicted in yellow for the 2d picture, and blue in the 3d picture, of all that directions that make a smaller angle with the feasible direction.
- The direction is chosen to maximize the width of this cone.
- Although this cone potentially removes several feasible points, it simplifies the expression for the ellipsoid without hurting the condition number of q unnecessarily.
- We then construct the ellipsoid within this cone.
- This construction sets the foundation for non-linear constraints as well, which we will turn to next.

21 Linear Algorithm Summary

- So let's review.
- We created an algorithm that does not use information from any infeasible evaluation, although it assumes linear constraints.
- We harnessed the analysis provided by Conejo et al to show its convergence.
- The primary obsticle was creating accurate model functions
- We did this by finding a feasible ellipsoid so we can run classic model-improving algorithms on a transformed sample region.
- We generalized this ellipsoid by providing a set of sample region requirements sufficient for our accuracy analysis.
- Lastly, we experimented with other methods within our numerical results section.

22 Nonlinear algorithm

- We turn our attention to general, non-linear constraints.
- Because we no longer know the precise boundary of the feasible region, we chose to buffer our trust regions with second order cones.
- We construct one second-order cone for each nearly-active constraint, and then force each sample point and the trial point to lie within the intersection of these cones.
- Note, we could have used any number of other shapes to buffer the infeasible region, and we considered some within our numerical results.
- So in this image, we have several black constraint linearizations, and blue second order cones buffering them.
- The region in yellow is buffered by all the constraints, and we could choose our sample region and search region inside.

23 Nearly Active Constraints

- We begin by determining the set of what we call nearly active constraints.
- Nearly active means that the constraint linearization has a zero near the current iterate.
- In this image, one constraint is actually active while another constraint does not even intersect the current outer trust region.
- However, another constraint intersects the trust region, although it is not active at the current iterate.
- The set of active constraints and those that intersect the current trust region are considered nearly active.
- If a constraint is not nearly active, all the nearby points will be feasible with respect to that constraint anyway.
- // We actually buffer the outer trust region slightly....

24 Buffering Cones

- The method for constructing the buffering cones is as follows.
- Suppose we are at an iterate in green, and the true, nonlinear constraint is in black.
- We first construct the linearization of the constraint, which is depicted in blue.

- We then compute the linearization's zero along the constraint gradient and scale it towards the current iterate.
- The scaled point is the red star.
- We then construct a second order cone opening towards the current iterate, with its vertex at the red star.

- As the trust region decreases, the cone widens and approaches the linearization of the constraint.
- This ensures that the buffered feasible region approaches the linearized feasible region.

- This is done for each nearly active constraint, and the intersection of all cones provides a buffered feasible region.
- We show that for small enough trust region radaii, the intersection of each constraint's cone and the outer trust region is feasible.

- Namely, theorem 4.24 states that the feasible region contains the intersection of the cones intersected with both the current outer trust region and the next outer trust region.
- We use both trust regions, so the sample region we construct during iteration k used to construct the next iterate's models is also feasible.

- Within our paper, we show that a similar construction as used for linear constraints can be applied to find a sample region within this buffered region.
- This is called the convervative ellipsoid within our paper, and it lies within the recession cone of the intersection of each constraint's buffering cones.
- The recession cone is in green in this image, while the constraint's cones are in blue.
- there may be better....

25 Sufficient Reduction

- That construction ensures that the sample region is buffered, but we also use the buffered region for our search region.
- However, limiting the trial point to lie within the buffered region, limits the amount of objective reduction possible.
- For example, in this image, the negative gradient is magenta, and the red buffering cone for the active, black constraint removes all descent directions.
- This means that in some scenarios, there may not be sufficient reduction within the buffered region.

- By choosing a trial point within this smaller region, we can no longer rely on common trust region subproblem algorithms because they may choose a trial point anywhere in the entire linearized feasible region.
- Remember, we need a point satisfying the efficiency condition, which projects onto the linearized constraints, not the buffered region.
- We show that such a point does exist within Theorem 4.27.
- However, because it depends on a sufficiently small trust region, we must explicitly check for reduction within our algorithm.
- Notice that checking the efficiency condition does not require evaluating the objective or constraints.

- In this image the search region is depicted in dark red, which is the intersection each constraint's buffering cone
- Because the negative gradient is at the magenta circle, it is likely that the solution to the trust region subproblem is at the top most vertex.
- However, it is cumbersome to show that this point satisfies the efficiency condition, and we don't need to show this particular point does: we just needed to find some point.
- Here, I illustrate the point that we show satisfies efficiency.
- We first use any classic method to find a point within the constraint's linearization, here depicted in black, that does satisfy the efficiency condition, **but we use half the trust region radius**
- This is the inner blue square.
- This is because we still need to move the solution at the blue arrow into the buffered region.

- We do this by adding the green arrow, the feasible direction, to ensure that we are once again within the buffered region.
- Using the smaller trust region radius ensures we are still within the outer trust region after adding the green arrow.
- In the picture, the feasibility component actually moves us closer, but it need not in general.
- We show that this point satisfies the efficiency condition for the linearized region, not just the buffered region.

26 Criticality Measure

- Conejo et al's criticality measure projects a point onto the true feasible region.
- However, because we do not know the true feasible region, and, in fact, have nonlinear constraints, we must replace the feasible region with a modelled feasible region.
- We use linear constraints, which ensures that the projection is still well defined.
- We were able to show that the model criticality measure converges to the true criticality measure.

- This result is complicated by the fact that the linearized feasible region changes across iterations.
- As you can see, this is the same image we showed earlier, which motivated the issues with inaccurate model functions.
- However, these inaccuracies change each iteration, possibly along with the set of nearly active constraints.

27 Bounded Projection

- Thus, to bound our criticality measure, we must bound how far our projection moves as the constraint models change.
- The key insight to bounding the projection, is that how much the projection changes with pertubations of the constraints depends on how similar the constraint normals are.
- Consider the two dimensional case first.
- In the following image, when two constraint linearizations meet at something close to a 90 degree angle, perturbing the black constraints to the blue constraints does not move the green x's projection by much.

- However, when the constraints make a smaller angle, and we add the **exact** same purtubation, the projection moves much farther.

- Thus, we can bound the difference in projections onto two similar 2-d polytopes by measuring the "angles" the constraints make with eachother.
- Suppose we can bound how far the negative gradient, namely the point we wish to project, is from the the current iterate.

- That distance is shown with the outer, magenta circle.
- Then any constraints that are violated by the point we wish to project, but are also far from the current iterate - atleast farther than the inner circle, cannot make a small angle with each other.
- For example the blue lines: no matter how we situate them such that they intersect within the magenta circle and outside of the green circle, they will make a large angle with eachother.
- It is only the constraints that are close to the current iterate that can be a problem, namely, those that are in the green circle.
- We use a regularity assumption to bound the distance moved by the projection along these nearly active constraints.

- My convergence proof required me to bound the distance our criticality measure's projection moves along these nearly active constraints.
- My initial approach relied on a quantity called the Hoffman constant, which uses this quantity to make the insight we developed for 2d constraints explicit.
- However, this approach required that I make "messy assumptions" such as requiring a bounded feasible set.
- I ended up abandoning this approach in favor of proposing a reguarity assumption inspired by the Mangasarian-Fromovitz constraint qualification.
- The analysis culminates in Theorem 4.41.
- Theorem 4.41 says that the difference between projections of the k-th iterates negative gradient onto the k-th linearized feasible region and the true linearization at x k goes to zero.
- Likewise, it says that the difference in projections on to feasible regions across iterates goes to zero.
- Next, we discuss the regularity assumption used for these nearly active constraints.

28 Regularity Assumption

- Our regularity assumption is loosely based on the Mangasarian-Fromovitz constraint qualification.
- This, ensures the existence of a feasible direction for any critical point.
- To bound the projection onto the linearized constraints, and ensure a bound on the condition number of q, we strengthened this qualification.
- Firstly, we assume that there is a feasible direction for each feasible x.
- However, rather than just using the active constraints, we use the nearly active constraints.

- It would be difficult to satisfy if we use all constraints, because there may be feasible regions that have parallel linearized constraints.
- However, we can't use only the active contraints, because some constraints arbitrarly close to active, may seem active with slightly inaccurate model functions.
- Thus, we use the nearly-active constraints: when the trust region becomes small enough, it cannot contain zeros of both constraint's linearizations and the green x.

- Thus, we require a feasible descent direction for the nearly active constraints.

- But this version is not strong enough.
- For example, consider two constraints depected on the left.
- The feasible region lies between the two constraints.
- The width of the feasible region, as measured by two infeasible directions on either side of the feasible region, is depicted on the right.
- The points become arbitrarly close as x moves farther from the viewer.
- If you imagine fitting an ellipsoid within these directions, it would become arbitrarly poorly conditioned.
- Thus, we created a uniform bound across all feasible points by introducing a small negative epsilon instead of the zero.

29 Ellipsoid Recovery

- One last topic important to the discussion of our algorithm, is feasible ellipsoid recovery.
- To set the stage, imagine that we started the algorithm with a only single point, and we needed to construct an entire feasible sample set.
- We know there is some feasible direction headed away from that point, if we are close enough, but it could be very thin.
- For example, in the image.
- We would have no model functions to even approximate the feasible direction.
- This is why we require a feasible sample region, rather than only a point.
- However, some sample sets along the way may also be infeasible, and when this happens, we decrease the trust region radius.
- Now, as the trust region radius decreases about the current iterate, the sample points required for constructing models become relatively further away.
- This means that the algorithm can reach a sticky situation with only one sample point.
- There are no other sample points to create models to even guess which direction is feasible.
- We do show the existence of a feasible ellipsoid for general non-linear constraints, but in pathological cases, finding a sample set with no model information could be computationally expensive.
- Thus, we assume a subroutine capable of finding some feasible sample set.
- Of course, we only need to call this subroutine until the trust region radius is small enough that the buffered region is feasible.
- Also, for convex constraints, this becomes easier, and we provide such a subroutine.

30 Convergence results

31 Numerical Results

- We compared our algorithm to three other solvers on the Hock-Schittkowski problem set.
- This is a performance profile, where the y-axis is the ratio of problems solved, and the x-axis is the number of evaluations used divided by the number used by the algorithm fewest evaluations.
- Algorithms that solved a larger percentage of the problems are higher on the chart, while algorithms that used fewer function evaluations are further to the left.
- PDFO is a collection of algorithms written by Powell, some of these are even used within SCIPY.optimize, which is a standard collection of optimization routines for python.
- However, the library we found which explicitly supports general, unrelaxable constraints is NOMAD.
- NOMAD does not use derivative information, and therefore does not model the constraint boundary.
- This means we required fewer infeasible evaluation attempts than NOMAD.
- However, this also makes NOMAD is more robust, so it can converge on several poorly conditioned problems.

32 Contributions

- So that is our algorithm.
- We believe it is the first model-based DFO algorithm for unrelaxable general, constraints.
- There are other algorithms that avoid infeasible evaluations,....
- To show convergence of our algorithm, we explicitly constructed feasible ellipsoids within linear constraints, and then within a buffered region, which we showed to be feasible.
- We also showed that this buffered region contains a point satisfying the efficiency condition.
- We showed convergence in criticality measure, which required bounding how far the negative gradient's projection can move across iterates.
- To do this, we created a novel regularity assumption inspired by the
- Mangasarian-Fromovitz constraint qualification for inequality constraints.
- Finally, in the paper, we provide some numerical results that suggest our algorithm makes fewer infeasible evaluation attempts, while still converging with a comparable number of total evaluations.

33 Extensions

- - Within the linear chapter of our paper, we discuss using polyhedral sample regions.
  - Although this outperforms our ellipsoidal method, we did not derive error bounds for such models.
  - We believe there should be corresponding error bounds over any convex shape.
- - Also, after creating this version of the model improvement algorithm, we conjecture that some narrow trust regions may not require the same number of sample points as a spherical or box-constrained region.

  - In fact, for these trust regions, the accuracy may harmed by close, nearly-redundant sample points.
  - Other researchers have done related work, such as choosing sample points that extend futher along directions for which the objective has more variance.
  - This is a similar idea, only it uses the feasible region shape rather than objective evalutations.

- - Lastly, our regularity assumption references model functions directly.
  - It would be cleaner to only make assumptions about the true functions, and we do provide a result that suggests it may be possible.
  - Namely, we show that after the trust region radius decreases to a certain threshold, if the ellipsoid is ever well conditioned, it stays that way.
  - This removes the accuracy condition's dependency on the regularity assumption, so the accuracy condition can translate assumptions about the true constraints to statements about the model functions.