Removing optimization problems' constraints with two-step geometric transformations

Oleksandr Kaleniuk*

National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

Abstract

From the geometric perspective, every continuous fragment of N-dimensional space constrained by strict inequalities is homeomorphic to the N-dimensional space itself. In this regard, every constrained optimization problem, where constraints are a set of inequalities, could be theoretically solved as a set of unconstrained problems.

For any practical purposes, however, applying an abstract transformation is not yet a method. Some of the most productive methods of numerical optimization rely on the differential properties of the target function, which might not survive the transformation. In this paper, we examine a two-step transformation that is not only continuous but also regular, allowing us to use Newtonian and quasi-Newtonian methods on the transformed target functions. The first step is the transformation from $\mathbb{R}^N$ to $[-1,1]^N$. The second is the transformation from $[-1,1]^N$ to either an N-dimensional linear hypercube mapping, an N-dimensional sphere, or an N-simplex. The list of second-step transformations can be continued, but this work only focuses on the three mentioned above.

Problem statement

Let's say we have a target N-ary function T and a set of constraints C. The constrained space forms an inner space of either a hypercube, a simplex, or a unit sphere. We want to find a local optimum for the function T within the constraints C using a Newtonian or a quasi-Newtonian method of unconstrained optimization. For this, we want to define a transformation, or a composition of transformations fg, that maps $\mathbb{R}^N$ to C. The transformation $f(g(argopt_x T(f(g(x)))))$ is the optimum of T constrained in C.

Literature review

There are different approaches for turning an unconstrained optimization algorithm into a constrained one. The simplest approach would be to limit the type of constraints to per-coordinate boundaries. For instance, the implementation of the Nelder-Mead in the popular Python SciPy library keeps the simplex coordinates within the requested bounds by hard limiting their values [1].

Similar to that, the Bounded Optimization BY Quadratic Approximations (BOBYQA) [2] extends the unconstrained derivative-free optimization method NEWUOA [3] by introducing per-coordinate bounds.

The Constrained Optimization BY Linear Approximations (COBYLA) uses linear approximation of non-linear constraints and then applies a parametric penalty function to ensure that the approximation of the optimum found is adequately precise [4].

The Constrained Optimization BY Quadratic Approximations (COBYQA) method allows both linear and non-linear constraints while using an optimization approach similar to bounded-only BOBYQA. COBYQUA way of constraint handling generalizes the Byrd-Omojokun approach [5, 6] that allows defining equality constraints with inequality constraints [7]. The method uses a parametric penalty function where the penalty parameter is variable. Using penalty functions is overall typical for the trust region-based optimization algorithms [8].

A similar approach could be used in the particle swarm optimization (PSO), although there are other methods typical for this class of optimization, namely bisection and preserving feasibility [9]. However, a popular PSO library, Pyswarm, only allows per-coordinate bounds [10].

This paper proposes a different approach for turning a constrained optimization problem into an unconstrained one, and it involves mapping $\mathbb{R}^N$ onto the constrained area.

Mappping $\mathbb{R}^N$ to $[-1,1]^N$

Let's presume we have a bounded optimization problem where $T(x,y)$ is the target function, and the boundaries are:

$$\begin{gathered} x > -1 \\ x < 1 \\ y > -1 \\ y < 1 \end{gathered} \tag{1}$$

*Corresponding author: o.kaleniuk@kpi.ua

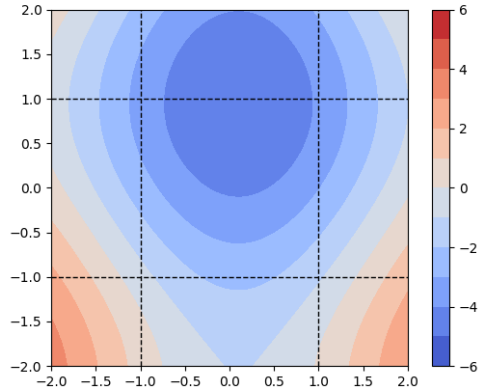

Figure 1: *An example of a bounded target function before any transformation*

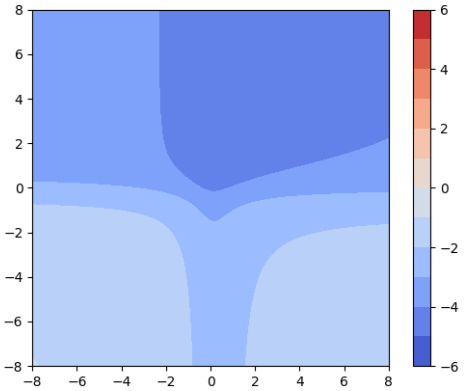

Figure 2: *An example of the unbounded target function after the arctangent transformation*

For example, if we define T like this:

$$T(x,y) = -3cos(x-0.1) + 2cos(y+2.2) \qquad (2)$$

Then the plot of this function and the constraints look like in figure 1.

The boundaries constrain the searching space for the optimization problem in a $[-1,1]^2$ square. Let's introduce the transformation from $\mathbb{R}^N$ to $[-1,1]^N$. One possible way to do so would be by using the arctangent function.

$$f_i(x,y) = (\frac{arctan(x)}{\pi/2}, \frac{arctan(y)}{\pi/2}) \qquad (3)$$

Now the function $Ti(x,y) = T(f_i(x,y))$ is defined on the whole $\mathbb{R}^N$ but it only contains values from the $T(x,y)$ defined on the set of boundaries (1). A plot of this function (now, of course, without constraints) is shown in figure 2.

A solution of the unconstrained optimization problem on $T(f_i(x,y))$ will be equivalent to the solution of

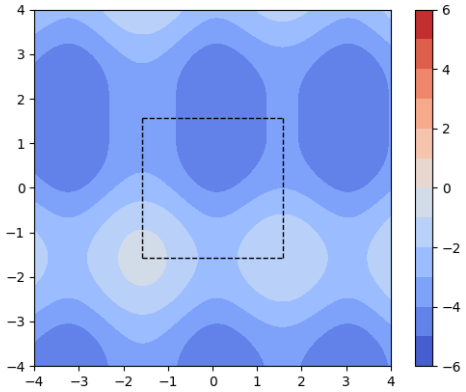

Figure 3: *An example of the unbounded target function after the sine transformation*

the bounded problem on $T(x,y)$ up to the transformation f_i. So given that the solution for the $T(f_i(x,y))$ is obtainable by any possible algorithm, and could be expressed as x_i, y_i, the equivalent solution for the $T(x,y)$ would be simply $f_i(x_i.y_i)$.

The transformation f_i is continuous and regular. For the most part, it retains enough similarity with the original $T(x,y)$ function so that if a Newtonian or a quasi-Newtonian method was able to find a solution on unconstrained $T(x,y)$ within the bounded region, it would also be able to do so on the unbound $T(f_i(x,y))$. Near the boundaries, however, every function tends to degenerate into a plateau due to the nature of the arctangent transformation. So the iterative methods tend to converge very slowly.

Another possible way to remove the bounds for the $T(x,y)$ would be to use a periodic function such as sine.

$$f_p(x,y) = (sin(x), sin(y)) \qquad (4)$$

An example of a transformed function $T(f_p(x,y))$ is shown in figure 3.

Such a transformation normally transforms a convex function into a non-convex one. A single optimum then becomes an infinite set of optimums. However, if there was one and only one local optimum in the bounded space, then all the optimums in the transformed space become equivalent under the f_p transformation. Not only does this allow us to use Newtonian and quasi-Newtonian methods, but it also removes the complications with near-boundary solutions that were present with the arctangent transformation f_i.

Mapping $[-1,1]^N$ to C

The cube $[-1,1]^N$ can be transformed into a manifold of arbitrary shape. Let's call this transformation $g(x_1, x_2, ..., x_N)$. This means that if this manifold corresponds to the constraint space for optimization, the non-constrained method can be used to optimize a target function $T(x_1, x_2, ..., x_N)$ on the constraint space by using a combination of transformations f from above and g. First we find the optimal point $x_uc = (x_{uc1}, x_{uc2}, ..., x_{ucN})$ in the unconstrained space $\mathbb{R}^N$, then we transform this point into the space the original T is defined to get the optimal point x_o.

$$\begin{aligned} x_{uc} &= argoptT(g(f(x))) \\ x_o &= g(f(x_{uc})) \end{aligned} \tag{5}$$

In practice, we're not interested in abstract manifolds. We need a specific set of pragmatically applicable transformations with their differential properties known and their implications on the optimization problem-solving understood. This work proposes three such transformations:

1. A transformation from $[-1,1]^N$ to a linear transformation of a hypercube $[-1,1]^N$.
2. A transformation from $[-1,1]^N$ to an N-simplex.
3. And a transformation from $[-1,1]^N$ to an N-dimensional sphere.

Linear transformation of an N-hypercube

Let's start from a 2-dimensional case. Let's say we have a target function $T(x,y)$, and a set of linear constraints that form some convex quadrilateral:

$$\begin{aligned} y &< a_1 x + b_1 \\ x &< a_2 y + b_2 \\ y &> a_3 x + b_3 \\ x &> a_4 y + b_4 \end{aligned} \tag{6}$$

These constraints form a quadrilateral C^2:

$$\begin{aligned} C^2 = ((x_{min1}, y_{min1}), (x_{min2}, y_{max2}), \\ (x_{max3}, y_{max3}), (x_{max4}, y_{min4})) \end{aligned} \tag{7}$$

An example of such rectangle constraining a target function is shown in figure 4.

Let g be a bilinear transformation.

$$\begin{aligned} g(x,y) = (a_x xy + b_x y + c_x x + d_x, \\ a_y xy + b_y y + c_y x + d_y) \end{aligned} \tag{8}$$

The inner space of the rectangle $[-1,1]^2$ can then be transformed into the inner space of the quadrilateral C^2 in a way that:

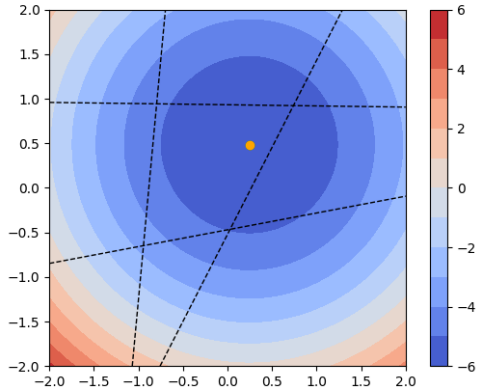

Figure 4: *An example of a binary target function constrained by a quadrilateral*

$$\begin{aligned} g(-1,-1) &= (x_{min1}, y_{min1}) \\ g(-1,+1) &= (x_{min2}, y_{max2}) \\ g(+1,+1) &= (x_{max3}, y_{max3}) \\ g(+1,-1) &= (x_{max4}, y_{min4}) \end{aligned} \tag{9}$$

This correspondence allows us to define 4 linear equations per every coordinate x and y. Solving these equations will provide us with all the coefficients a_x, b_x, c_x, d_x and a_y, b_y, c_y, d_y respectively.

Let's define matrix A as:

$$\begin{bmatrix} +1 & -1 & -1 & +1 \\ -1 & -1 & +1 & +1 \\ +1 & +1 & +1 & +1 \\ -1 & +1 & -1 & +1 \end{bmatrix} \tag{10}$$

Now let's define vectors B_x and B_y:

$$\begin{aligned} B_x &= [x_{min1}, x_{min2}, x_{max3}, x_{max4}] \\ B_y &= [y_{min1}, y_{max2}, y_{max3}, y_{min4}] \end{aligned} \tag{11}$$

Now solving these two equations will give us the coefficients for the g transformation.

$$\begin{aligned} A[a_x, b_x, c_x, d_x] &= B_x \\ A[a_y, b_y, c_y, d_y] &= B_y \end{aligned} \tag{12}$$

The transformation g now transforms the space in a way that the inner space of the rectangle $[-1,1]^2$ bijects the inner space of the quadrilateral C^2 but only if the quadrilateral is convex.

An example of such a bijection, that also transforms the function along with its optimum point, is shown in Figure 5.

Now let f be an arctangent transformation:

$$f(x,y) = \left(\frac{arctan(x)}{\pi/2}, \frac{arctan(y)}{\pi/2}\right) \tag{13}$$

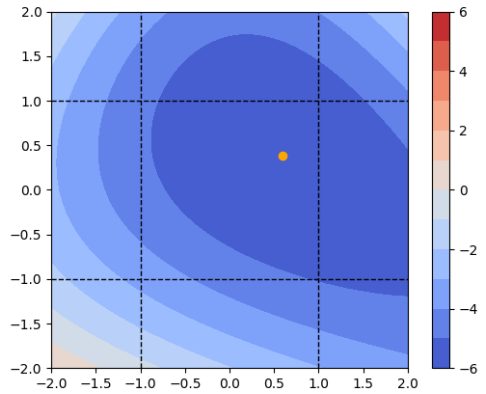

Figure 5: *An example of the bilinear transformation that makes a constrained optimization problem into a bounded one*

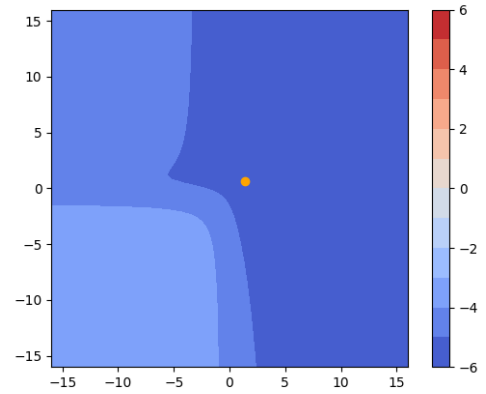

Figure 6: *An example of the target function mapped onto the unconstrained space*

This transformation maps the whole $\mathbb{R}^2$ onto the inner space of $[-1, 1]^2$. The combination of transformations f and g transforms $\mathbb{R}^2$ onto the inner space of C^2. To optimize the target function constrained on C^2 we can now optimize the unconstrained function $T(f(g(x, y)))$, find the optimum point in the unconstrained space x_{uc} and then transform the found optimum onto the C^2 with the same combination to find the true optimum point x_o of the constraint target function T: $x_o = f(g(x_{uv}))$.

An example of a target function defined on the unconstrained space is shown in figure 6.

This approach generalizes fairly easily to C^N. The rectangle then becomes a linearly transformed hypercube; the coefficients of the N-dimensional linear transformations form a vector of 2^N elements, so the computational complexity of the transformation also rises exponentially. But the numeric values of the coefficients for the bilinear transformation are always deducible from the corners' coordinates of the bounding hypercube. And the coefficients of each corner point can easily be found by putting the corresponding linear equations that form the constraints into a linear system on N equations and solving the system as we do when computing the planes' intersection.

For instance, the coordinates $(x_{min1}, x_{min2}, ..., x_{minN})$ can be found by gathering all the constraints of type $x_i > a_{i_N} * x_N + ... + a_{i_{i+1}} * x_{i+1} + a_{i_{i-1}} * x_{i-1} + ... + a_{i_2} * x_2 + a_{i_1} * x_1 + a_{i_0}$, where a_{i_j} is the j-th coefficient of the i-th linear constraint, and making them into a system of linear equations:

$$\begin{gathered} a_{1_N}x_N + ... + a_{1_{i+1}}x_{i+1} - x_i + a_{1_{i-1}}x_{i-1} + \\ + ... + a_{1_2}x_2 + a_{1_1}x_1 = -a_{1_0} \\ a_{2_N}x_N + ... + a_{2_{i+1}}x_{i+1} - x_i + a_{2_{i-1}}x_{i-1} + \\ + ... + a_{2_2}x_2 + a_{2_1}x_1 = -a_{2_0} \\ ... \\ a_{i_N}x_N + ... + a_{i_{i+1}}x_{i+1} - x_i + a_{i_{i-1}}x_{i-1} + \\ + ... + a_{i_2}x_2 + a_{i_1}x_1 = -a_{i_0} \\ ... \\ a_{1_N}x_N + ... + a_{N_{i+1}}x_{i+1} - x_i + a_{N_{i-1}}x_{i-1} + \\ + ... + a_{N_2}x_2 + a_{N_1}x_1 = -a_{N_0} \end{gathered} \tag{14}$$

Solving this system gives us the coordinates for one of the corners of the linearly transformed hypercube formed by the linear constraints. Solving 2^N such systems will give us enough values to form N linear systems of 2^N equations that will represent the transformations of all the cube's corners. For the 2D, the corresponding equations are shown in formula (12).

Solving these equations will give us the coefficients for the transformation g. The transformation f then remains almost the same as in formula (13):

$$f(x_1, x_2, ..., x_N) = (\frac{arctan(x_1)}{\pi/2}, \frac{arctan(x_2)}{\pi/2}, ..., \frac{arctan(x_N)}{\pi/2}) \tag{15}$$

And the optimum point of the function T constrained by C^N could still be found by solving the unconstrained optimization problem on the transformed function as shown in formula (5).

N-simplex

Simplices are used heavily in geometric modeling and finite element analysis. A triangle mesh representing a surface for 3D graphics is an example of a simplicial complex. A tetrahedral mesh of a 3D model prepared for FEA simulation is another. We can find an optimum of a function constrained by an N-dimensional simplex border using a principle similar to that we used for the hypercube constraints.

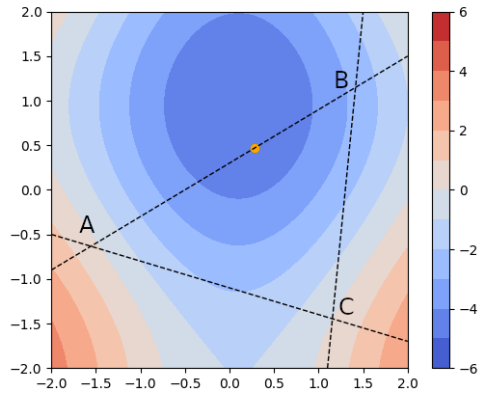

Figure 7: *An example of a target function constrained in a 2-simplex*

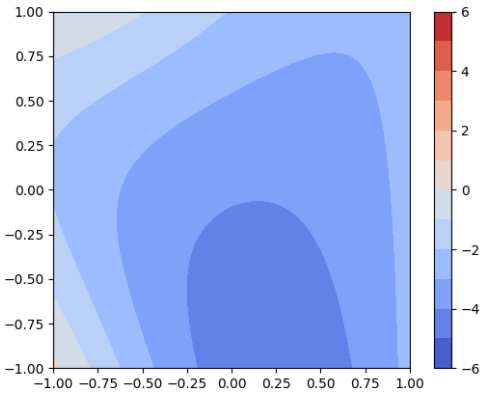

Figure 8: *An example of a square to triangle transformation applied to the target function from figure 7*

Of course, we need another variant of the g transformation.

Technically, there is no single best transformation from an N-simplex onto an N-dimensional cube $[-1,1]^N$. There are at least $N!$ transformations that are equivalent for any pragmatical usage.

Let's look at the 2-dimensional example.

Let's say we have three linear constraints (16) that, while intersecting in pairs, define a triangle ABC on the $\mathbb{R}^N$ plane as seen in figure 7. We want to find the optimum of a function T in the triangle ABC.

$$\begin{aligned} x &> a_1 y + b_1 \\ y &> a_2 x + b_2 \\ y &< a_3 x + b_3 \end{aligned} \tag{16}$$

The triangle's ABC vertices appear where the lines representing the constraints intersect.

Let's A be the intersecting points of the 1st and the 2nd constraint, B - the 2nd and the 3rd, C - the 3rd and the 1st. The specific coordinates of each point could then be obtained by solving 3 systems of 2 linear equations each.

$$\begin{cases} x_A - a_1 y_A = b_1 \\ y_A - a_2 x_A = b_2 \end{cases} \\ \begin{cases} y_B - a_2 x_B = b_2 \\ y_B - a_3 x_B = b_3 \end{cases} \\ \begin{cases} y_C - a_3 x_C = b_3 \\ x_C - a_1 y_C = b_1 \end{cases} \tag{17}$$

One way to define the $[-1,1]^2$ to the 2-simplex transformation for the three vertices A, B, and C would be this:

$$\begin{aligned} g(x,y) = \\ (x_A + (x_B - x_A)\frac{x+1}{2} + \\ (x_C - x_A)\frac{y+1}{2}(1 - \frac{x+1}{2}), \\ y_A + (y_B - y_A)\frac{x+1}{2} + \\ (y_C - y_A)\frac{y+1}{2}(1 - \frac{x+1}{2})) \end{aligned} \tag{18}$$

Here the $\frac{x+1}{2}$ and $\frac{y+1}{2}$ transform $[-1,1]^2$ into $[0,1]^2$. Vectors $B - A$ and $C - A$ form a linear basis. The $[0,1]^2$ square, if interpreted as coordinates in this basis, corresponds to the parallelogram that covers the simplex ABC and an equivalent triangle we have no interest in. To limit $[0,1]^2$ to the ABC only, we multiply the second coordinate of the square by one minus the first coordinate.

The formula (18) can be generalized onto a set of 6 transformations given that A, B, and C are interchangeable. Not all of these 6 will be identical, but there is no good measure to assess the practical applicability of one transformation over another, so in this work, only the default transformation, the one described in (18) is further used.

An example of such a transformation transforming the $[-1,1]^2$ to the triangle ABC the target function from figure 7 is defined in is shown in figure 8.

The formula (18) can be rewritten to clearly show that it is a special case of a bilinear transformation.

$$
\begin{aligned}
g(x,y) = \\
(xy(\frac{x_A}{4} - \frac{x_C}{4}) + x(\frac{-x_A}{4} + \frac{x_B}{2} - \frac{x_C}{4}) + \\
y(\frac{-x_A}{4} + \frac{x_C}{4}) + \frac{x_A}{4} + \frac{x_B}{2} + \frac{x_C}{4}, \\
xy(\frac{y_A}{4} - \frac{y_C}{4}) + x(\frac{-y_A}{4} + \frac{y_B}{2} - \frac{y_C}{4}) + \\
y(\frac{-y_A}{4} + \frac{y_C}{4}) + \frac{y_A}{4} + \frac{y_B}{2} + \frac{y_C}{4})
\end{aligned} \tag{19}
$$

It can also be generalized to the N-dimensional case. Let's call the simplex vertices v_i where $i = 1..N+1$. The coordinates of the point inside the cube $[-1,1]^2$ will then be x_j where $j = 1..N$, and the point itself $x = (x_1, x_2, ..., x_N)$. The formula (18) then becomes:

$$
\begin{aligned}
g(x) = \\
v_1 + \sum_{i=1}^{N}(v_{i+1} - v_1)\frac{x_i + 1}{2}\prod_{j=1}^{i-1}(1 - \frac{x_j + 1}{2})
\end{aligned} \tag{20}
$$

Note that while we were only referring to the constrained space and not the constraints themselves before while talking about the unconstraining transformation, with simplices it is possible, given their recurrent structure, to find an optimum not only in the N-simplex itself, but on its border too. This is possible because the border of an N-simplex is exactly $N+1$ $(N-1)$-simplices, which also have their borders as $N-2$-simplices, which in turn have their borders as $N-3$ simplices, etc. A 0-simplex is a point. So we can compute the target function in the simplex vertices, use unbounding transformations to find target function's optimums on the simplex edges, use similar transformations to find potential optimums on the simplex triangles, etc, until we have a set of potential optimums found on all possible borders of a given simplex. Then we can pick a minimum or a maximum with a simple discrete search.

Whether looking for the optimum recurrently in all N dimensions is practically applicable is questionable. Normally, the approximation of an optimum inside a simplex is good enough even if the true optimum lies exactly on the simplex border.

N-dimensional sphere

Once again, let's start from a 2-dimensional case. Let's say we have a spherical constraint $x^2 + y^2 < 1$ and a function T we want to find an optimum for in that sphere. The simplest way to make a transformation from a square $[-1,1]^2$ to the sphere would be to linearly transform a pair of polar coordinates r and ϕ. An r would correspond to the square's first coordinate, and the ϕ to the second.

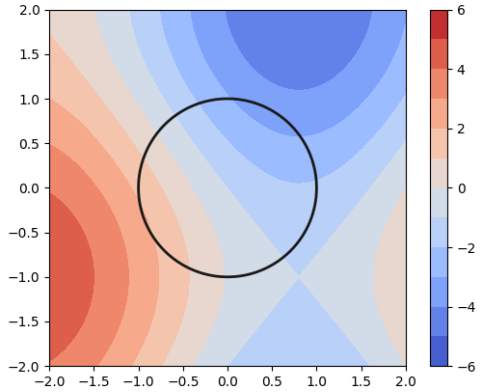

Figure 9: *An example of a function constrained within a unit circle*

$$
g(x,y) = (\frac{x+1}{2}cos(\pi y), \frac{x+1}{2}sin(\pi y)) \tag{21}
$$

This already enables us to use the composition (5) and use unconstrained optimization to find the optimum constrained by the sphere. But for the spherical constraints, we can make the composition simpler if we choose our unbounding transformation f to be periodic, such as (4).

$$
fg(x,y) = (sin(x)sin(y), cos(x)sin(y)) \tag{22}
$$

This function alone grants the transformation from $\mathbb{R}^2$ to the unit circle. The resulting function $T(fg(x,y))$ becomes periodic and loses the convex property if it has been convex before (as in figure 9) but with this transformation, most of the unconstrained optimization algorithms are still able to find the optimum on the source function T because, firstly, the transformation itself is regular so it doesn't prohibit differentiation anywhere enabling Newtonian and quasi-Newtonian methods to work, secondly, even if the function $T(fg(x,y))$ has the infinite amount of optimal points, they are all equal, and they all correspond to the single point in the constrained space under the same transformation fg.

You can see this in figure 10: the inner space of the circle corresponds to the square marked with a dashed line in the figure, but the rest of the function is a repeated pattern everywhere on $\mathbb{R}^2$.

Experiments

To explore the practical applicability of the proposed method, a series of computational experiments were conducted. Each series was dedicated to a particular combination of transformations:

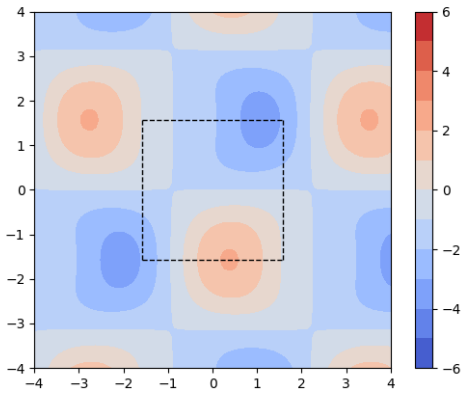

Figure 10: *An example of a function being "unconstrained" by a periodic transformation*

1. The arctangent transformation from $\mathbb{R}^2$ to $[-1,1]^2$ (3) combined with the transformation from $[-1,1]^2$ to a bilinear transformation of a square (8) to minimize a quadratic function with 4 linear constraints.
2. The periodic transformation from $\mathbb{R}^2$ to $[-1,1]^2$ (4) combined with the transformation from $[-1,1]^2$ to a bilinear transformation of a square (8) to minimize a quadratic function with 4 linear constraints.
3. The arctangent transformation from $\mathbb{R}^2$ to $[-1,1]^2$ (3) combined with the transformation from $[-1,1]^2$ to a 2-simplex (18) to minimize a quadratic function with 3 linear constraints.
4. The periodic transformation from $\mathbb{R}^2$ to $[-1,1]^2$ (4) combined with the transformation from $[-1,1]^2$ to a 2-simplex (18) to minimize a quadratic function with 3 linear constraints.
5. The bi-periodic transformation from $\mathbb{R}^2$ to a unit circle (22) to minimize a quadratic function on a unit circle.

The target function for all series was made to have a random minimum in the $(-1,1)^2$ square's inner space.

For the first 4 series, the initial configuration of constraints had been randomized along with the target function.

For the first 2 series, the initial constraints were given by four points of their intersections. The points were randomly generated in $(-1,0)^2$, $(-1,0)\times(0,1)$, $(0,1)^2$, and $(0,1)\times(-1,0)$ squares. In this way, the linear constraints always form a convex quadrilateral, which is a necessary condition for the bilinear transformation.

For series 3 and 4, the constraints were given by the three points of intersection randomized in: $(-1,0)\times(-0.5,0.5)$, $(0,1)^2$, and $(0,1)\times(-1,0)$.

The experiments concluded compared relative effectiveness of COBYLA with constraints [4] and unconstrained BFGS [11] on transformed target functions. The main factor that was considered as a measure of effectiveness was the number of target function evaluation.

The experiments have shown that BFGS with arctangent transformation f (3) usually loses in performance to COBYLA if the minimum points lies outside the constraints. It is also observed that the chance of BFGS outperforming COBYLA goes down as the minimum point gets closer to the constraints even while technically staying inside. This may be attributed to the nature of arctangent transformation that creates quasi-plateau near the constraints as the space within contraints gets transformed disproportionately: a small translation in the origin space of the target function corresponds to the larger and larger translation in the $\mathbb{R}^N$ space as the point in question approaches the constraints.

This effect is observable both for the square (8) and 2-simplex (18) transformations g.

With the periodic transformation (4) and bilinear transformation (8), BFGS outperforms COBYLA in roughly 2/3 of all the test runs. However, it is worth noticing that the total amount of target function evaluations differs between algorithms by less than 10%, so the difference in pragmatic efficiency seems negligible.

With the periodic transformation (4) and 2-simplex transformation (8), BFGS outperforms COBYLA in about 1/2 of all the test runs. The total amount of target function evaluations is 10%-20% more for BFGS, so COBYLA shows slightly more performant.

With the bi-periodic transformation (22) to the unit circle, BFGS outperforms COBYLA in about 2/3 of all the run cases. The difference in total function evaluation counts is about 10% in favor of BFGS.

Regarding the precision, BFGS with the bi-periodic transformation has the upper hand when the randomized minimum lies outside the unit circle, not in. BFGS gives a better approximation in about 99% of all the test runs; however, the difference in total error between the two algorithms is negligible. In 1024 test runs, the difference in total error is only about 1e-4.

Considering all the evidence, BFGS with the proposed transformation gives a small performance and precision boost in some of the problems explored. However, the scale of improvement doesn't seem to be game changing for any practical problem.

The source code for all the experiments is available on GitHub: `https://github.com/akalenuk/unconstraining_transformations`.

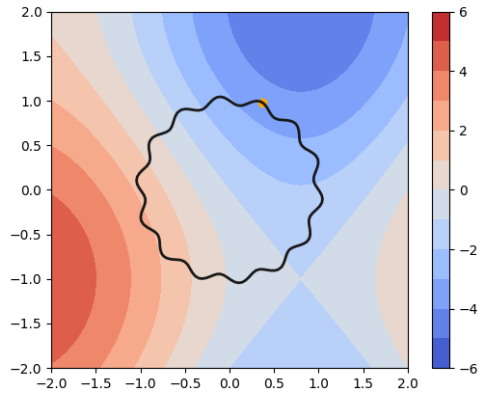

Figure 11: *A constraint given as a sinusoidal function in polar coordinates*

Discussion

The proposed method may not show any drastic improvement over existing algorithms on lightweight target functions and primitive constraint configurations we examined. But the experimental results show promise, especially regarding the non-linear constraints. It is worth noticing that while the traditional approach segregates linear and non-linear constraints, for the transformation-based approach, any transformation that maps $\mathbb{R}^N$ to any constrained area is inherently non-linear, so there is no difference whether the constraints themselves are linear or not.

Effectively, this enables employing non-linear transformations from $[-1, 1]^N$ to the constrained area. We have already seen one in the context of cube to sphere transformation (22), but we can also use quadratic or cubic transformations for handling quadratic or cubic constraints, respectively.

There is also space for improvement regarding setting constraints in polar coordinates. So far, we only looked at the unit circle; however, it is easy to generalize the approach for optimizing a target function constrained by another function c in polar coordinates so that $c : \phi \to r$. An example of solving such an optimization problem is shown in Figure 11.

Solving this problem with a traditional approach would be difficult. The computational complexity of the linear approximation of a sinusoid in polar coordinates depends on its period, while the computational complexity of a bi-periodic transformation doesn't.

There might also be problems where there is already a mapping function we can use as the transformation from a unit cube to the constrained space. For instance, if we want to minimize something on a surface given as a spline patch. In this case, the natural parameterization of the spline is already a mapping function we can use as an g part of the fg transformation.

It would be interesting to investigate such problems in the future, but that goes beyond the scope of this particular paper.

Conclusions

This paper proposes an approach to solve optimization problems with constraints by mapping the $\mathbb{R}^N$ onto the constrained area and solving the unconstrained optimization problem on that map. To make the approach modular, the transformation is split into a 2-step transformation. First the $\mathbb{R}^N$ to $[-1, 1]^N$ transformation, then the $[-1, 1]^N$ to the constrained area. In this paper, two transformations of the former type and three of the latter are examined.

The experiments show that the proposed approach offers little to no benefit in solving problems of mathematical optimization with constraints compared to other well-established methods. There are small gains in terms of performance and precision that it can provide in one context or another, but the scale of these benefits is small. However, the general approach is also applicable to problems that would be hard to tackle with more traditional methods.

References

[1] Fuchang Gao and Lixing Han. "Implementing the Nelder-Mead simplex algorithm with adaptive parameters". In: *Computational Optimization and Applications* 51.1 (2012), pp. 259–277. DOI: https://doi.org/10.1007/s10589-010-9329-3.

[2] M. J. D. Powell. *The BOBYQA algorithm for bound constrained optimization without derivatives*. Tech. rep. DAMTP 2009/NA06. University of Cambridge, 2009. URL: https://www.damtp.cam.ac.uk/user/na/NA_papers/NA2009_06.pdf.

[3] M. J. D. Powell. "The NEWUOA software for unconstrained optimization without derivatives". In: *Advances in Optimization and Numerical Analysis*. Ed. by G. Di Pillo and M. Roma. New York: Large-Scale Optimization, 2006, pp. 255–297. URL: https://link.springer.com/chapter/10.1007/0-387-30065-1_16.

[4] M. J. D. Powell. "A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation". In: *Advances in Optimization and Numerical Analysis*. Ed. by Susana Gomez and Jean-Pierre Hennart. Dordrecht: Springer Netherlands, 1994, pp. 51–

67. ISBN: 978-94-015-8330-5. DOI: 10.1007/978-94-015-8330-5_4. URL: https://doi.org/10.1007/978-94-015-8330-5_4.

[5] R. H. Byrd. "Robust trust region methods for constrained optimization". TheThird SIAM Conference on Optimization. 1987.

[6] E.O. Omojokun. "Trust Region Algorithms for Optimization with Nonlinear Equality and Inequality Constraints". PhD thesis. Boulder, CO: University of Colorado, 1989.

[7] T. M. Ragonneau. "Model-Based Derivative-Free Optimization Methods and Software". PhD thesis. Hong Kong, China: Department of Applied Mathematics, The Hong Kong Polytechnic University, 2022. URL: https://theses.lib.polyu.edu.hk/handle/200/12294.

[8] A.R. Conn, N.I.M. Gould, and P.L. Toint. *Trust Region Methods*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, 2000. ISBN: 9780898714609. URL: https://books.google.com.ua/books?id=BPFWLwTysSAC.

[9] Mauro S. Innocente and Johann Sienz. *Constraint-Handling Techniques for Particle Swarm Optimization Algorithms*. 2021. arXiv: 2101.10933 [cs.NE]. URL: https://arxiv.org/abs/2101.10933.

[10] Lester James Miranda. "PySwarms: a research toolkit for Particle Swarm Optimization in Python". In: *Journal of Open Source Software* 3.21 (2018), p. 433. DOI: 10.21105/joss.00433. URL: https://doi.org/10.21105/joss.00433.

[11] R. Fletcher. *Practical Methods of Optimization*. A Wiley-Interscience publication v. 1. Wiley, 1987. ISBN: 9780471915478. URL: https://books.google.com.ua/books?id=rTnWGpxfKAUC.