An Intrinsic Approach to Manifold Optimization

Benjamin Shaw
College of Science
Utah State University
Logan, UT, United States
ben.shaw@usu.edu

***Abstract*—An intrinsic approach to manifold optimization is outlined, specifically within the context of constrained regression. For two simple examples, the explicit formulas needed to implement Riemannian gradient descent intrinsically are given. These formulas are implemented in python and used to apply intrinsic Riemannian constrained regression in two distinct scenarios.**

***Index Terms*—Manifold Optimization, Machine Learning, Constrained Regression.**

I. Introduction

In 1-dimensional linear regression, one wishes to apply the model $y = mx + b$ to a dataset containing points x_i and target variables y_i. The "best line of fit" is found by optimizing the parameters m and b. Although a closed-form solution may exist, the best parameters are often merely approximated due to the potentially high computational cost of finding the exact solution. At any rate, the process of finding the optimal parameters m and b is an unconstrained optimization problem, since there are no apparent constraints imposed upon the parameters.

The function (of the parameters) that is to be optimized–minimized, more specifically–is known as the loss function. The loss function measures the closeness of the parameters to the optimal parameters: optimal parameters are found when the loss function is minimized. One example of a loss function is known as the Mean Squared Error loss (MSE loss) [1], which is defined as follows, for a model $y = f(x; w)$ with model parameters $w = \{w_j\}$:

$$L = \frac{1}{N} \sum_{i=1}^{N} ||y_i - f(x_i; w)||^2. \tag{1}$$

The integer N is the number of datapoints in the dataset.

One simple method used in unconstrained optimization is gradient descent. In this method, a point is updated as follows, given a loss function L:

$$x_1 = x_0 - \eta \nabla L(x_0), \tag{2}$$

where x_0 is the initial point, $\nabla L(x_0)$ is the gradient of the loss function at the point x_0, η is a hyperparameter known as the learning rate, and x_1 is the updated point. The iterative process of gradient descent is repeated until desired, which is typically when the change in the loss function values appears to halt or else when the desired number of iterations has been realized.

In principle, constraints can be introduced on the parameters of the loss function. Examples for the simple model $y = mx + b$ include $m^2 + b^2 = 1$, $m > 1$, and $m - b^2 > 0$. The first example is an equality constraint, whereas the second two examples are inequality constraints. If a constraint is introduced, the optimization of the parameters subject to the given constraint(s) is called a constrained optimization problem. In constrained optimization, one seeks to minimize the loss function subject to the given constraints.

One known method of handling constrained optimization problems is called manifold optimization. In this technique, the constraints are interpreted as defining a surface (or region)–a manifold, really–in the ambient parameter space: if the optimization parameters are forced to update only on the surface itself, the parameters will always satisfy the given constraints. Moreover, if one is able to define operations which move a point on the manifold to another point on the manifold, the constrained optimization has the interpretation of an unconstrained optimization problem on the manifold. For our linear model, for example, the constraint $m^2 + b^2 = 1$ defines the surface of a (1-dimensional) sphere: with this constraint, the problem can be recast as an unconstrained optimization problem on the 1-dimensional sphere.

Once the constrained optimization problem is cast as an unconstrained optimization problem on the manifold, a technique such as gradient descent can be applied: however, caution must be used. In general, the components of the gradient of a function are not merely the partial derivatives with respect to the variables of the function: although this is the case in Euclidean space in Cartesian coordinates, this is not the case in general. What is needed is called the Riemannian gradient, which involves not only the partial derivatives, but also the use of an object called the Riemannian metric. Additionally, the motion from one point to another must be along a curve known as a Geodesic curve, which curve is the generalization of a straight line on manifolds. Thus, in manifold optimization, the update of a point in the parameter space is a more complicated operation than in the case with Euclidean space in Cartesian coordinates.

The existing implementations of manifold optimization are extrinsic: that is, they rely on operations in the ambient space [2] [3]. Take, for example, the $m^2 + b^1 = 1$ constraint, which places the model parameters on the unit circle: with this constraint, one can calculate the gradient in the ambient 2-dimensional Euclidean space, then define an operation which "projects" the gradient onto the surface of the sphere.

Another approach to manifold optimization is intrinsic. For the $m^2+b^2=1$ constraint, one can introduce local coordinates on the manifold—the 1-dimensional surface—from which the Riemannian metric is explicitly computed, the Geodesic curves given (or approximated), and the operations take place strictly in the 1-dimensional space. For example, one can define the coordinate t such that $m=\cos(t)$ and $b=\sin(t)$: the example constraint is now trivially satisfied. These equations allow one to explicitly calculate the Riemannian metric and give the equations which define the Geodesic curves, allowing one to optimize on a strictly 1-dimensional surface.

In this paper, the intrinsic approach has been implemented in two simple examples where regression parameters are constrained. There is one known attempt to implement the intrinsic approach [4]: however, this implementation relies heavily on symbolic software [5] and numeric approximations for the Geodesic curves. In this paper, the intrinsic approach will be implemented in python without the use of symbolic software and using exact solutions to the geodesic equation.

This paper is organized as follows. In section 2, we will present an overview of the formal mathematical setting in which Riemannian gradient descent occurs. The section immediately following will describe the Riemannian gradient descent algorithm, as well as offer the specifics of how the algorithm in this paper differs from others. Section 4 briefly discusses the notion of constrained regression, along with potentially motivating applications [6]. In section 5, we apply Riemannian gradient descent to the model $y=mx+b$ subject to $m>1$, and in section 6 we apply Riemannian gradient descent to the model $\frac{a}{\sqrt{x}}+\frac{b}{x^2}+c$ subject to $a^2+b^2=1$. The paper then ends with concluding remarks.

II. A Brief Description of the Differential Geometry

This section is intended to give a brief overview of the formal setting in which Riemannian gradient descent occurs. In 3-dimensional Euclidean space, one can visualize many 2-dimensional surfaces, such as the surface of a sphere or a flat plane. These surfaces are examples of Manifolds. A manifold M is a set with a topology–that is, some notion of what "open sets" are–that is said to be "locally similar" to $\mathbb{R}^n$. Local similarity means that open sets in M can be mapped to open sets in $\mathbb{R}^n$ in a smooth, invertable manner (with a smooth inverse). If the mapping from open sets in M to open sets in $\mathbb{R}^n$ is sufficiently smooth, M is said to be a differentiable manifold. There are a few other formal, topological properties of differentiable manifolds which we do not address in this paper, but which can be found in many textbooks on differential geometry. [7] For this paper, it suffices to consider that many surfaces in 3-dimensional Euclidean space ($\mathbb{R}^3$) are examples of manifolds.

With examples of surfaces in $\mathbb{R}^3$ fresh in the reader's mind, we consider also the subject of tangent planes to a particular surface. For the surface defined by the 2-dimensional sphere, the tangent plane at the North pole would appear something like a plate on top of a basketball: the tangent plane at a point p on the surface of the sphere is a plane which, when visualized in $\mathbb{R}^3$, touches the sphere only at p. The tangent space at a point, denoted T_pM, has the mathematical structure of a vector space: that is, one can add two vectors in T_pM and get another vector in T_pM, as well as scale vectors in T_pM.

A Riemannian metric is an object that defines an inner product (the generalization of the dot product) on the tangent space at each point on the manifold. The term "metric" is also sometimes used to describe a function that allows one to calculate distances between two points in a set; however, the Riemannian metric is not such a function. It can, however, be used to define such a "distance function."

A Riemannian manifold is a pair (M,g), where M is a differentiable manifold and g is a Riemannian metric. A connection on M is an object that allows one to take derivatives of vectors in the tangent space. A "metric connection" is a connection on M that is defined, in a particular way, by a Riemannian metric.

A geodesic curve is the generalization of straight lines in Euclidean space. If one is on (or in) a manifold and throws a baseball, the ball will follow the path of a geodesic (as long as there is no "air resistance" or analogous features). The geodesic curves are defined by a second-order system of ordinary differential equations, generally non-linear. The Geodesic equations are defined by a connection–for our purposes, the metric connection. Thus, specifying a Riemannian metric, for our purposes, is the key to obtaining the geodesic curves. Of course, to obtain exact solutions to the geodesic equations, one must solve a system of non-linear, second order ordinary differential equations.

The exponential map is a function that maps tangent vectors (at a point) to the manifold itself. It is defined in terms of geodesics: if $\gamma(t)$ is the unique geodesic satisfying $\gamma(0)=p\in M$ and $\gamma'(0)=v\in T_pM$, then $\exp_p(v)=\gamma(1)$. That is, to apply the exponential map to a vector v in the tangent space at p, one solves the geodesic equation with initial values $\gamma(0)=p$ and $\gamma'(0)=v$, then assigns v to $\gamma(1)$, the result of which is a point on the Manifold.

III. Riemannian Gradient Descent

The update rule for ordinary gradient descent is given in (2). In words, the point x_0 in Euclidean space is mapped to a new point by following the straight-line path from x_0 in the direction of $-\nabla L(x_0)$ (the gradient at x_0) an amount controlled by the learning rate η. [8]

The update rule for Riemannian gradient descent is a generalization of the ordinary rule. In words, the point p on the manifold is mapped to a new point by following the geodesic with initial point p and initial direction $\nabla L(p)$ (now the Riemannian gradient) an amount controlled by the learning rate η:

$$q=\exp_p(-\eta\nabla L(p)).$$

The Riemannian gradient differs from the gradient in Cartesian coordinates in Euclidean space. One begins, as in the Cartesian case, by computing the partial derivatives of the loss function L. However, the components of the gradient in Cartesian coordinates are simply these partial derivatives, whereas in the Riemannian case this is generally not true. In the Riemannian case, one applies the inverse of the Riemannian metric, which can be realized as a matrix when coordinates are chosen, to the components of the "vector" of partial derivatives. The result is a vector which is associated with a bona fide tangent vector in the tangent space at a point.

A. Riemannian GD and Constrained Optimization

Naturally, one may consider the question of why the parameters in a given loss function should be interpreted as coordinates on a Riemannian manifold, rather than the traditional interpretation of the parameters as Cartesian coordinates in Euclidean space. One reason for the interpretation of the Loss function parameters as coordinates of a Riemannian manifold stems from constrained optimization. Consider, for example, the following function of three variables:

$$f(x, y, z) = -e^{-(x-1)^2-(y-1)^2-(z-1)^2}. \tag{3}$$

The function f is minimized at the point $(1, 1, 1) \in \mathbb{R}^3$, which point could be discovered by means of ordinary gradient descent. However, we now consider the constrained optimization problem of minimizing f subject to the following constraint:

$$x^2 + y^2 + z^2 = 1. \tag{4}$$

The point $(1, 1, 1)$ does not satisfy the constraint given in (4), and is thus not a valid solution to the constrained optimization problem. What is needed is a method for handling constrained optimization.

Techniques for handling constrained optimizations abound. One such technique is known as manifold optimization. As the constraints define a valid subset of points in the ambient Euclidean space, the subset of points is viewed as a manifold, embedded in Euclidean space. For example, the constraint given in (4) defines the 2-dimensional surface of the sphere of radius 1. On the manifold, the optimization is unconstrained, allowing one to employ unconstrained optimization techniques such as Riemmanian gradient descent.

B. The Extrinsic Approach

Manifold optimization is most commonly implemented extrinsically: that is, the mathematical operations typically take place in the ambient Euclidean space. A summary of the extrinsic algorithm is as follows [8].

The gradient of the loss function is first computed in the original coordinates, where the components of the gradient vector are simply the partial derivatives. This gradient vector is reimagined as a vector in the tangent space at a point, which tangent vector is them mapped back down to the manifold. Often, this mapping–the exponential map–is approximated by means of projecting the tangent vector to the nearest point on the manifold.

It has been shown that this method has the property of "almost sure convergence," meaning the success of this algorithm is comparable to the success of ordinary gradient descent. It is believed that there are, however, drawbacks to this approach, which will be discussed shortly.

C. The Coordinate-based, Intrinsic Approach

Another approach is intrinsic: that is, operations do not rely on the ambient space. In this method, local coordinates are made explicit, which coordinates allow one to give the Riemannian metric explicitly. The metric connection can then be defined, allowing one to subsequently give the geodesic equations: the algorithm for Riemannian gradient descent then proceeds as described previously.

Take, for example, the constraint given in (4). One can give coordinates (ϕ, θ) such that the following is true:

$$x = \cos(\phi)\sin(\theta), \quad y = \sin(\phi)\sin(\theta), \quad z = \cos(\theta). \tag{5}$$

Substituting (5) into (4), we see that the constraint is satisfied. Thus, in (ϕ, θ) coordinates, the minimization of (3) is unconstrained. Of course, the function f must be realized in our new coordinates:

$$f(\phi, \theta) = -e^{-(\cos(\phi)\sin(\theta)-1)^2-(\sin(\phi)\sin(\theta)-1)^2-(\cos(\theta)-1)^2}. \tag{6}$$

All there is left to do is apply Riemannian gradient descent to $f(\phi, \theta)$ in order to approximate the minimum. The Riemannian metric is needed. In the original (x, y, z) coordinates, it is assumed that the metric is simply the Euclidean metric, given as

$$g = dx^2 + dy^2 + dz^2, \tag{7}$$

the components of which can be represented as the identity matrix. With the transformation given in (5), we find that the metric in (ϕ, θ) is given as

$$\tilde{g} = \sin(\theta)^2 d\phi^2 + d\theta^2. \tag{8}$$

The components of the inverse metric can be represented as the following matrix:

$$g^{-1} = \begin{bmatrix} \frac{1}{\sin(\theta)^2} & 0 \\ 0 & 1 \end{bmatrix}, \tag{9}$$

Allowing us to compute the Riemannian gradient of $f(\phi, \theta)$. The Riemannian metric also allows us to define the metric connection, which in turn allows us to give the geodesic equation. The solutions–the geodesic curves–allow us to define the exponential map, and our Riemannian gradient descent algorithm is defined.

D. Intrinsic vs. Extrinsic: a Brief Comparison

With both the extrinsic and intrinsic approaches described, it is worth a brief discussion as to the potential strengths and weaknesses of each. It is believed that both methods have advantages that allow one to be preferred over the other, depending on the circumstance.

One clear advantage of the extrinsic approach is that there is no need to select local coordinates. Moreover, the operations that take place are comparatively simple: tangent vectors and points on the manifold alike can be interpreted as points/vectors in the ambient vector space, which can be added and scaled.

The intrinsic approach, on the other hand, suffers from complex operations. To make matters worse, the formula for the Riemannian gradient descent, and the solutions to the geodesic equations, are coordinate-dependent, which is an issue since there is no unique choice of acceptable local coordinates for a given manifold. Thus, if the user is to be allowed to select local coordinates manually, any computer program which implements Riemannian gradient descent must incorporate symbolic software.

The intrinsic approach also requires solutions to the geodesic equations: solutions to second-order, non-linear differential equations. In general, the problem is hopeless. Thus, the solutions to the geodesic equations must most likely be numerically approximated.

The challenges of the intrinsic approach can be addressed, however. In order to optimize a particular loss function subject to a particular constraint, a consistent, sensible choice of coordinates could be given: thus, a computer program which optimizes a particular loss function subject to a particular constraint could be explicitly coded without symbolic software if the mathematics was worked out beforehand. In fact, this is the approach taken in this paper.

The need to select local coordinates is, at first, an apparent disadvantage of the intrinsic approach. However, it also offers the advantage of flexibility. For the extrinsic approach, it appears that the manifold interpretation of the constraints must be known; however, with the intrinsic approach, one needs only to select local coordinates such that the given constraints are satisfied. Of course, this could be difficult in general, but it allows certain problems to be addressed that may not be addressable using the extrinsic approach.

Lastly, the need to use the Riemannian gradient and the need to propagate along a geodesic curve also have the potential to be relaxed. While this is still under investigation, there is reason to believe that the "poor man's gradient," meaning the vector whose components consist merely of the partial derivatives, may point in a direction which is always within 90° of the Riemannian gradient, so that, perhaps, the poor man's gradient may be a suitable approximation. Secondly, if the learning rate is taken to be sufficiently small, it is possible that the geodesic curves can be approximated as other curves, the form of which would be explicitly known without having to solve the Geodesic equations exactly.

IV. Constrained Regression

Of course, one is free to question the practical application of constrained optimization in the context of data science. As it happens, the applications in data science are plentiful, including in the implementation of the support vector machine [1]. For this paper, we examine the application to regression: that is, constrained regression.

For a model $f(x;w)$ parameterized by $w = \{w_j\}$, a regression task seeks to minimize a loss function involving f, such as the loss function given in (1). The result of the optimization algorithm is a set of parameters $\{w_j\}$ which are believed to (approximately) minimize the loss function and are therefore taken to be the "best parametersvector" for the model.

In principle, the model parameters can be constrained. This occurs in deep learning when the weight matrix of a given fully connected layer are endowed with a notion of "orthogonality"–the resulting neural network is thought to have superior properties in certain circumstances [3].

Constrained optimization is also thought to be applicable in the context of Economics, where the model constraints stem from desirable attributes of the model. For example, in a dataset where the natural log of the number of sales is being analyzed as a function of the natural log of the average price of a given product as follows [6]:

$$\ln(\text{sales}) = m \ln(\text{price}) + b. \tag{10}$$

The m parameter has the interpretation of "elasticity," or the change in the product demand as a function of price. The elasticity is thought to be strictly outside of the interval $[-1, 1]$: a constraint.

The remainder of this paper will be dedicated to the application of manifold-constrained optimization in two cases motivated by economics.

V. Application 1: a linear model

Our first model will be a simple, 1-dimensional linear model $y = mx + b$ where the slope m is taken to be greater than 1: $m > 1$. It has been suggested that, for a dataset consisting of two features, namely the natural log of the price and the natural log of the number of sales, the model $y = mx + b$ can be applied, and the interpretation of m is the "elasticity" of the product in question–the change in the demand of the product as a function of the change in price. Ideally, the elasticity is strictly greater than 1 or strictly less than -1. Thus, we seek to develop the tools to optimize the mean square error (1) with $f(x) = mx + b$ subject to the inequality constraint $m > 1$.

To ensure that $m > 1$, one can choose coordiantes (u, v) in the following way:

$$m = e^u + 1, \qquad b = v. \tag{11}$$

In this choice of coordinates, u is free and we always have $m > 1$: thus, the problem is unconstrained in (u, v) coordinates.

We are, however, required to give the formula for the following: the loss function in our new coordinates, the Riemannian gradient, and the solutions to the geodesic equations. The formula for the loss function comes from simply evaluating (1) with $f(x) = (e^u + 1)x + v$. The components for the gradient are found to be, respectively the following:

$$2e^{-u}\sum_{i=1}^{N}[(e^u+1)x_i + v - y_i]x_i, \quad 2\sum_{i=1}^{N}(e^u+1)x_i + v - y_i. \tag{12}$$

The parametric geodesic curves are found to be the following:

$$u(t) = \ln(c_1 t + c_2), \quad v(t) = c_3 t + c_4, \tag{13}$$

where, given initial position coordinates (p_1, p_2) and initial gradient components (v_1, v_2), we find that

$$c_1 = v_1 e^{p_1}, \;\; c_2 = e^{p_1}, \;\; c_3 = v_2, \;\; c_4 = p_2. \tag{14}$$

Having computed the Riemannian gradient and given the geodesic curves, the algorithm of Riemannian gradient descent is now well-defined.

A. Testing on Synthetic Data

We first test our algorithm, coded in python, on a simple, synthetic dataset. The data is generated by the formula $y = x + 1$, where the datapoints x_i are chosen to be uniformly spaced between 0 and 10: in all, 100 points are chosen. Our gradient descent algorithm is initialized at the point $(u, v) = (2, 0)$, which, in the original (m, b) coordinates is $(e^2 + 1, 0)$.

A visualization of the decrease in loss function values is given in Fig. 1 below.

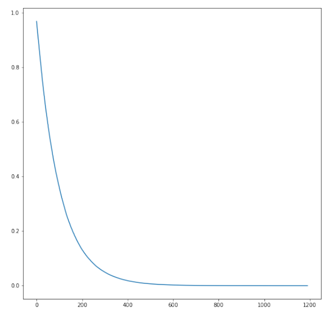

Fig. 1. The loss function values as a function of the number of trainig epochs. On the horizontal axis, we have the number of epochs (starting at epoch 10), and on the vertical axis are the loss function values.

B. The Avocado Dataset

Next, we attempt our algorithm using a more realistic dataset. The dataset contains various attribues dealing with the sale of avocados. From the dataset, we extract the price and the average number of sales at a given price. A visualization of our best line of fit is given in Fig 2 below.

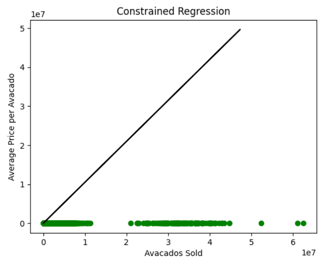

Fig. 2. Our best line of fit for the avocado dataset, subject to the slope being greater than 1.

Obviously, it seems that the constraint for $m > 1$ is not suited for this particular dataset. However, this result offers additional assurance that our algorithm for gradient descent respects the constraints without compromise.

VI. Application 2: the Phillips curve

We now turn to our second model, which is based on the Phillips curve. The Economic theory behind the Phillips curve is that as the unemployment rate increases, the interest rates decrease; conversely, as the interest rates increase, unemployment rates decrease. Thus, the Phillips curve assumes that the unemployment rate and the interest rate are somewhat "inversely related."

Our second model is therefore the following, where y represents the inflation rate and x represents the unemployment rate:

$$y = \frac{a}{\sqrt{x}} + \frac{b}{x^2} + c, \qquad a^2 + b^2 = 1. \tag{15}$$

The motivation behind this model is that it is a mixture of the functions $\frac{1}{\sqrt{x}}$ and $\frac{1}{x^2}$, both exhibiting "inversely related" shapes; however, the former decays slower while the latter increases much more sharply. Thus, our model exhibits what we believe to be a balanced mixture of two curves, one which sharply increases as $x \to 0$ and is negligible at larger values of x, and one which is less significant at low values of x but more significant for larger values of x. An additional constant parameter is included for increased flexibility with the curve fit.

As in the previous model, we will give the relevant formulas for Riemannian gradient descent. Coordinates (α, β) are chosen with

$$a = \cos(\alpha), \quad b = \sin(\alpha), \quad c = \beta. \tag{16}$$

This allows us to defined the Riemannian gradient in our new coordinates. The components are, respectively,

$$-2\sum_{i=1}^{N}\frac{-\cos(2\alpha) + \sin(\alpha)(\beta - y_i)x_i^2}{x_i^{5/2}} \tag{17}$$

$$+2\sum_{i=1}^{N}\frac{\cos(\alpha)((1 - x_i^3)\sin(\alpha) + x_i^2(\beta - y_i))}{x_i^4},$$

and

$$2\sum_{i=1}^{N}\frac{(\beta - y_i)x_i^2 + \cos(\alpha)x_i^{3/2} + \sin(\alpha)}{x_i^2}. \tag{18}$$

Despite the increased complexity of the Riemannian gradient, the geodesic curves are much more simple:

$$\alpha(t) = c_1 t + c_2, \quad \beta(t) = c_3 t + c_4, \tag{19}$$

where for initial position coordinates (p_1, p_2) and initial velocity components (v_1, v_2), we have

$$c_1 = v_1, \;\; c_2 = p_1, \;\; c_3 = v_2, \;\; c_4 = p_2. \tag{20}$$

In fact, the geodesic curves are simply straight lines, as in the case of Euclidean space. The formulas given above allow us to implement Riemannian gradient descent.

Due to a discovered inability to work with the originally-intended dataset, our application is limited to synthetically-generated data. We generate 100 numbers x_i evenly distributed from 1 to 11, and generate values y_i with $y_i = \dfrac{1}{\sqrt{x_i}} + 1$. Initializing our gradient descent at the point $(\pi/4, 0)$ in (α, β) coordinates, which corresponds to the point $(1/\sqrt{2}, 1/\sqrt{2}, 0)$ in (a, b, c) coordinates, we apply our algorithm. As before, the loss function decreases over the course of the training process–500 iterations, in this case–and we now make a plot to depict the path of the optimization algorithm in the (a, b) plane (Fig 3):

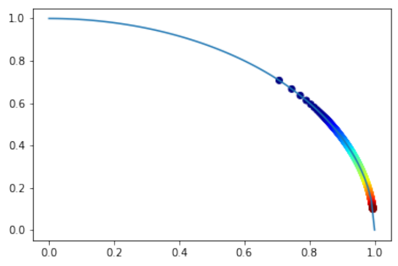

Fig. 3. The path of our algorithm in the (a, b) plane. The colored points, beginning with blue points, show the path of the gradient descent. The blue curve shows the line which defines the constraint $a^2 + b^2 = 1$.

Our final estimated parameters are approximately $a = 0.995$, $b = 0.098$, and $c = 0.993$.

VII. Conclusion

We have implemented intrinsic Riemannian gradient descent in python in two distinct scenarios. The first scenario sought to minimize the mean square error with model function $y = mx + b$ subject to the constraint $m > 1$. The second scenario sought to minimize the mean square error as well, but with model function $y = \frac{a}{\sqrt{x}} + \frac{b}{x^2} + c$ subject to $a^2 + b^2 = 1$. We have given the algorithm for intrinsic gradient descent explicitly in both cases, giving the geodesic curves as well as the formula for the Riemannian gradient in our respective choice of coordinates. We have shown using simple, synthetic data, that our algorithm tends to converge.

The challenges of implementing the intrinsic approach to Riemannian gradient descent are apparent. For a given constraint, local coordinates will need to be chosen so as to render the problem as unconstrained in the new coordinates: subsequently, a formula for the Riemannian gradient must be given in that particular choice of coordinates, and the geodesic curves must also be given. Notwithstanding the apparent challenges to implementing the intrinsic approach, we conclude that implementation in python is viable in at least certain cases. We also advocate for the advancement and expansion of this implementation, due to the potential advantages of the intrinsic approach.

A. *Changes made in the project proposal*

In large part, the project[1] proceeded as planned, though there were notable deviations. First of all, we proposed implementing symbolic-free Riemannian gradient descent on the 2-dimensional surface of a sphere: this was abandoned since the exact solution to the associated geodesic curves was elusive. Due to time constrains, no remedial efforts were made and the 2-dimensional sphere example was simply abandoned.

We also proposed fitting our Phillips curve model to real, macroeconomic data, which was not done. The data we obtained was found to be more difficult to work with, and remedial efforts were abandoned for the sake of time.

Lastly, we proposed further investigating the application of constrained regression within the context of economic- and business-related applications. The purported reason for this deficiency is due to the inability to source data for individual business entities.

Notwithstanding the scope of the project being slightly reduced, the main objective was completed, being to give a symbolic- and approximation-free implementation of Riemannian gradient descent in python for some simple examples.

References

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Springer, 2017.

[2] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre, "Manopt, a Matlab toolbox for optimization on manifolds," *Journal of Machine Learning Research*, vol. 15, no. 42, pp. 1455–1459, 2014. [Online]. Available: https://www.manopt.org

[3] M. Meghwanshi, P. Jawanpuria, A. Kunchukuttan, H. Kasai, and B. Mishra, "Mctorch, a manifold optimization library for deep learning," arXiv preprint arXiv:1810.01811, Tech. Rep., 2018.

[4] B. Shaw, "An intrinsic approach to equality- and inequality-constrained optimization on riemannian manifolds." April 2023, unpublished.

[5] I. M. Anderson and C. G. Torre, "The differential geometry software project," May 2022. [Online]. Available: https://digitalcommons.usu.edu/dg{_}downloads/4

[6] H. Vardhan, "Constrained linear regression." [Online]. Available: https://towardsdatascience.com/constrained-linear-regression-aaf488296d93

[7] D. Lovelock and H. Rund, *Tensors, Differential Forms, and Variational Principles*. Dover Publications inc., 1989.

[8] S. Bonnabel, "Stochastic gradient descent on riemannian manifolds," *IEEE Transactions on Automatic Control*, vol. 58, no. 9, pp. 2217–2229, sep 2013. [Online]. Available: https://doi.org/10.1109%2Ftac.2013.2254619

[1]Code is available at https://github.com/benshaw2/CS6830_FinalProject