CS 520 Project Report

Ahmed Elbagoury and Rania Ibrahim

April 28, 2017

In this report, we present our results of using different optimization techniques to solve the Thompson problem. First, in section 1 we explain the definition of Thompson problem and its mathematical formulation. Then, we explain various techniques to solve this problem, we start in section 2 by explaining how to use the penalty method and in section 3 we briefly discuss the use of Nelder Mead method. After that in section 4, we use Augmented Lagrangian Method, then in section 5, we show the use of projected gradient method. After that, we show in section 6 how to solve the Thompson problem using spherical coordinates method. Section 7 shows how to use the Interior point method to solve the Thompson problem. In section 8, we show the experimental results of comparing all the aforementioned methods. We then show our endeavors in exploring different techniques, in section 9, we discuss possible convex reformulations. Finally, we try to solve this problem using Homotopy methods in section 10 and 11 respectively. Section 11 discusses using Randomized methods as a future work and section 12 gives a conclusion for the report.

1 The Thompson Problem

Thompson problem is the problem of finding the optimal locations of n electrons on the surface of the unit sphere, such that the repel force between these electrons is minimized. More formally the problem is defined as:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{i<j} \frac{1}{||x_i - x_j||^2} \\ \text{subject to} \quad & x_i \in \mathbb{R}^k, ||x_i|| = 1, 1 \leq i \leq n \end{aligned} \tag{1}$$

Where k is the dimension of the points and n is the number of points.

2 Penalty Method

The penalty method provides a way to utilize unconstrained optimization techniques to solve constrained optimization problems.

More formally, for the following optimization problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & q_i(x) \leq 0, i = 1, \cdots, m \\ & h_i(x) = 0, i = 1, \cdots, n \end{aligned} \tag{2}$$

We define $p(\lambda, y)$ to be a penalty function as:

$$p(\lambda, y) = \begin{cases} 0, & \text{if } y < 0 \\ \lambda y^n, & \text{if } y \geq 0 \end{cases}$$

- p is continuous.
- $p(\lambda, y) \geq 0, \forall \lambda$ and y.
- $p(\lambda, y) = 0$ for $y \leq 0$ and p is strictly increasing for both $\lambda > 0$ and $y > 0$.

The optimization function in 2 can be converted to:

$$\tilde{f}(x) = f(x) + \sum_{i=1}^{m} p(\alpha_i, q_i(x)) + \sum_{i=1}^{n} p(\beta_i, h_i^2(x)) \tag{3}$$

The first term penalizes points that violate at least one of the inequality constraints. The second term penalizes the points that violate at least one of the equality constraints, $h_i^2(x)$ is used so that both positive and negative values are penalized. Note that, α_i and β_i are positive constants that control to what extend the inequality and equality constraints will be enforced, respectively.

Problem 1 has equality constraints, so it can be written as:

$$f(X) = \sum_{i \neq j} \frac{1}{||x_i - x_j||_2^2} + \frac{\alpha}{2} \sum_{i=1}^{n} (||x_i||_2 - 1)^2 \tag{4}$$

Note that we used $p(\alpha, h(x)) = \frac{\alpha}{2} \sum_{i=1}^{n} (||x_i||_2 - 1)^2$. The penalty method starts by a small value for α and solves the unconstrained problem, then it increases the value of α, after that it resolves the unconstrained problem again and keeps iterating till it reaches a sufficiently accurate point.

To implement the penalty method, we used scipy "`optimize`" package for unconstrained optimization. The derivative of 4 is:

$$\nabla_{\mathbf{x}_{ik}} f(X) = -2 \sum_{i \neq j} \frac{x_{ik} - x_{jk}}{||x_i - x_j||_2^4} + \alpha(||x_i|| - 1) \frac{x_{ik}}{||x_i||}$$

Central difference technique is used to verify the correctness of the gradient (the code is attached)

3 Nelder Mead

Nelder Mead Method is a derivative free method. Its basic idea is to define a simplex with n+1 points and then use the simplex slope to find a descent direction for reducing your function.

In order to employ Nelder Mead method in solving the Thompson problem, first we use the penalty method to convert the problem from a constrained optimization problem to an unconstrained optimization problem and then we solve the penalty method using Nelder Mead.

4 Constrained Problem with Augmented Lagrangian Method

4.1 Overview

Augmented Lagrangian Method replaces a constrained problem with:

- Series of unconstrained problems.
- An additional term for each one of the unconstrained problems. This term is an estimate of the Lagrangian multiplier.

Given an unconstrained problem: $\min_{\mathbf{x}} f(\mathbf{x})$, Subject to: $c_i(\mathbf{x}) = 0 \ \forall i \in I$. Augmented Lagrangian Method converts this unconstrained problem to the following series of unconstrained problems: $\min_{\mathbf{x}} f(\mathbf{x}) + \frac{\mu_k}{2} \sum_{i \in I} c_i^2(\mathbf{x}) - \sum_{i \in I} \lambda_i^k c_i(\mathbf{x})$. Where λ_i is an estimate of the Lagrangian Multiplier and its accuracy of approximating the Lagrangian Multiplier improves as the number of iterations increases, and μ_k is a parameter that is chosen at each iteration.

4.2 Algorithm

We have implemented the Augmented Lagrangian Method (ALM) mentioned in Numerical Optimization Book by Jorge Nocedal and Stephen J. Wright in Framework 17.3.

4.3 Comparison

Augmented Lagrangian Multiplier differs from the standard Lagrangian by the presence of the squared terms of $\frac{\mu_k}{2} \sum_{i \in I} c_i^2(\mathbf{x}_k)$. This term penalizes the objective function if $c_i(\mathbf{x}_k) \neq 0$, where k is the iteration number.

Additionally, Augmented Lagrangian Multiplier differs from the quadratic penalty method by the presence of the summation containing λ. Therefore, it reduces the possibility of ill conditioning. Unlike the penalty method, Augmented Lagrangian Multiplier doesn't need to take $\mu \to \infty$ in order to solve the constrained optimization.

4.4 Thompson Problem

Given the Thompson problem, we have written its Augmented Lagrangian Method function as follows:

$$L_A(\mathbf{x}, \lambda; \mu) = \sum_{i=1}^{n} \sum_{j=1:i \neq j}^{i-1} \frac{1}{||\mathbf{x}_i - \mathbf{x}_j||^2} + \frac{\mu}{2} \sum_{i=1}^{n} (||\mathbf{x}_i|| - 1)^2 - \sum_{i=1}^{n} \lambda_i (||\mathbf{x}_i|| - 1)$$

Also, we have computed the gradient for the Augmented Lagrangian Method function, which is the following:

$$\nabla_{\mathbf{x}_{ik}} L_A(\mathbf{x}, \lambda; \mu) = -2 \sum_{i \neq j} \frac{\mathbf{x}_{ik} - \mathbf{x}_{jk}}{||\mathbf{x}_i - \mathbf{x}_j||^4} + \mu (||\mathbf{x}_i|| - 1) \frac{\mathbf{x}_{ik}}{||\mathbf{x}_i||} - \lambda_i \frac{\mathbf{x}_{ik}}{||\mathbf{x}_i||}$$

We have double checked our gradient by implementing the central difference method and comparing it to our gradient. You can find the code written in python attached with the report.

5 Projected Gradient Descent

5.1 Overview

Although gradient descent is used to solve unconstrained optimization problems, it is utilized by Projected Gradient Descent (PGD) in solving constrained optimization problems. PGD uses Gradient descent as a first step to find the optimal x and then chooses the nearest x' to x that satisfies the constraints.

5.2 Thompson Problem

In Thompson problem, the gradient of the objective function is:

$$\nabla_{\mathbf{x}_{ik}} f(x) = -2 \sum_{i \neq j} \frac{\mathbf{x}_{ik} - \mathbf{x}_{jk}}{||\mathbf{x}_i - \mathbf{x}_j||^4}$$

Then, we use gradient descent and update $x_{ik}^{k+1} \leftarrow x_{ik}^k - \alpha \nabla_{\mathbf{x}_{ik}} f(x)$ until the first order optimality conditions for this unconstrained problem is satisfied. Note that, we have double checked our gradient by implementing the central difference method and comparing it to our gradient.

Instead of projecting x^{k+1} to the unit sphere after each update, we project only at the end the optimal x^* to the unit sphere in order to find the nearest x to it that satisfies the constraints. This variation has yielded a better result in Projected Gradient Descent.

6 Unconstrained Optimization using Spherical Coordinates

The Thompson Problem can be transformed to unconstrained optimization problem by representing the points using spherical coordinate system. For instance in case of $k = 3$, equation 1 becomes:

$$\underset{x}{\text{minimize}} \quad \sum_{i<j} \frac{1}{2(\sin(\beta_i)\sin(\beta_j)\cos(\theta_i - \theta_j) + \cos(\beta_i)\cos(\beta_j) - 1)} \tag{5}$$

Then, the coordinates of the point P_i can be represented as $x_i = \sin(\beta_i)\cos(\theta_i), y_i = \sin(\beta_i)\sin(\theta_i)$, and $z_i = \cos(\theta_i)$ for $1 \leq i \leq n$. This guarantees that $||[x, y, z]^T||_2 = 1$.

We use scipy "`optimize`" package to solve the unconstrained optimization 5.

7 Interior point method

7.1 Overview

Interior Point method (IP) is a class of optimization algorithms that are used to solve both linear and nonlinear optimization problems. The basic idea of the interior point method is to solve a sequence of optimization problems by replacing the inequality constraints with a log barrier functions that penalize the objective function if the constraints are violated.

7.2 Thompson Problem

We have tried to use interior point method to solve this problem. In order to use interior point method, we have first searched for a Python library for Interior point method, and have found Ipopt (Interior Point OPTimizer) library. However, it was not supported in Python. After searching carefully, we have found a Python interface for the library called pyipopt. We wrote a class to optimize the Thompson problem using this interface. Although, we were able to run the code, the results were bad.

After investigating the reasons behind this bad results, we found that the number of iterations were too small, however, we were unable to tune the parameters of the Interior point method due to the inflexibility of the Python interface and its poor documentation. As a result, we decided to move to MATLAB and we were able to run interior point method using fmincon function.

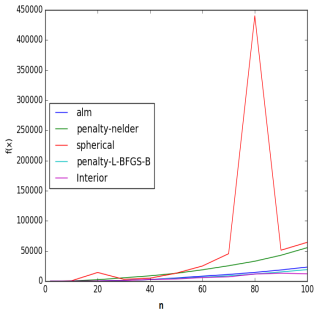

Figure 1: Value of $f(x)$,k=2

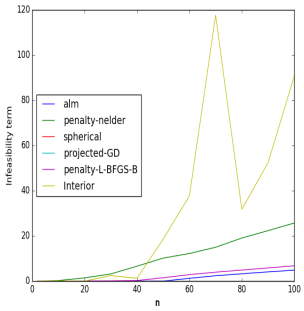

Figure 2: Inf term,k=2

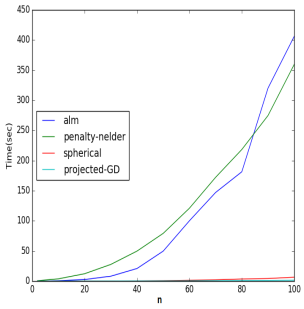

Figure 3: Running time,k=2

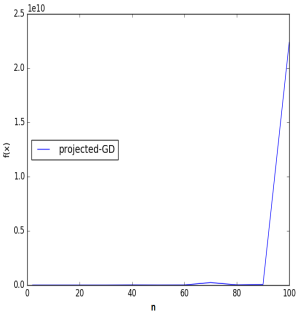

Figure 4: Value of $f(x)$ for PGD, k = 2

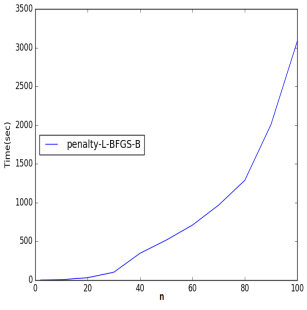

Figure 5: Time for penalty with -L-BFGS-B, k = 2

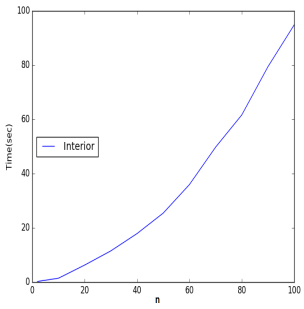

Figure 6: Time for IP, k = 2

8 Experimental Results

To assess the performance of the different methods, we use the running time, the objective function and the infeasibility term (inf term), which is defined as the sum of the absolute

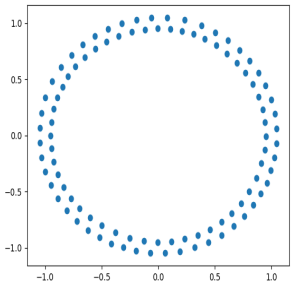

Figure 7: Augmented Lagrange Multiplier (n = 100, k = 2)

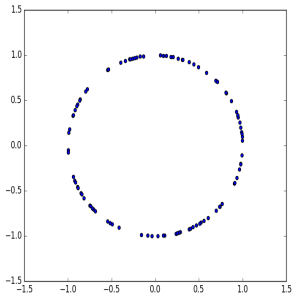

Figure 8: Projected Gradient Descent (n = 100, k = 2)

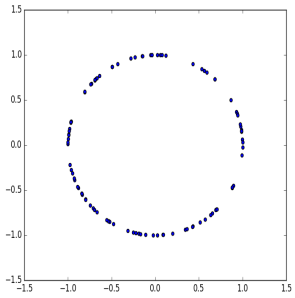

Figure 9: Spherical Unconstrained Optimization (n = 100, k = 2)

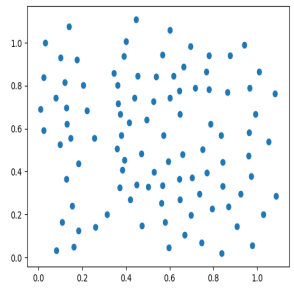

Figure 10: Penalty Method using Nelder-mead (n = 100, k = 2)

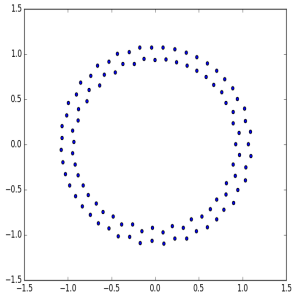

Figure 11: Penalty Method using L-BFGS-B (n = 100, k = 2)

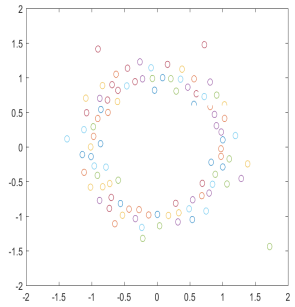

Figure 12: Interior Point Method (n = 100, k = 2)

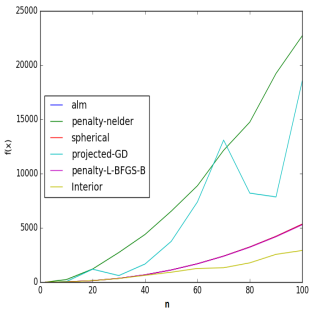

Figure 13: Value of $f(x)$, k=3

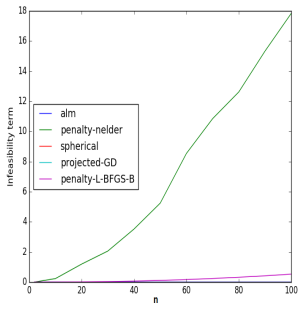

Figure 14: Inf term, k=3

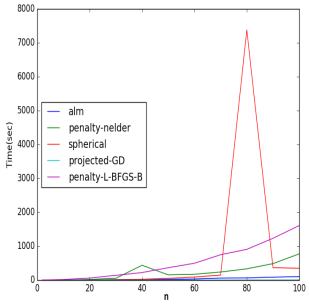

Figure 15: Running time, k=3

difference of the norm of x_i and 1. The goal of using the infeasibility term is showing to what extend the placement of the points violates the constraint of being on the unite sphere, the higher the infeasibility term the more the constraints are violated.

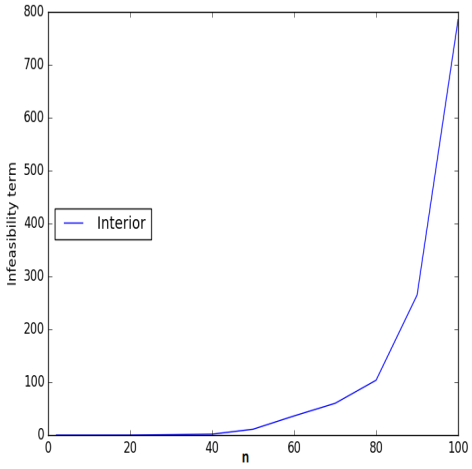

Figure 16: Inf for IP, k = 3

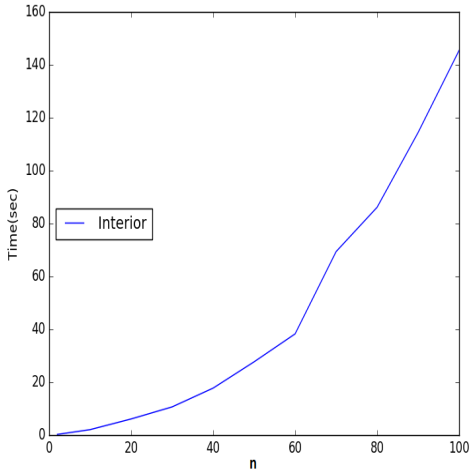

Figure 17: Time for IP, k = 3

The average of 5 runs, each with a random starting point, is reported for every method. The Augmented Lagrangian Method was run using $\mu = 1$ and increasing μ value by 100 in each iteration. The λ was initialized with all 10. These values were selected empirically so that the running time is reasonable. For penalty method, α was initialized to 1 and incremented at each iteration by 100.

Figures 1, 2 and 3 show the value of the objective function, infeasibility term and the running time of every method for different values of n and $k = 2$, respectively. We plotted the objective function of the projected gradient descent and the running time of penalty method `L-BFGS-B` in two different figures figure 4 and 5, respectively as they have large values on the y-axis compared to other methods, so including them with other methods will not illustrate the difference between the other methods. Additionally, we have drawn the time for the interior point method in a separate figure 6 as it is implemented in MATLAB, while the other methods are implemented in Python. Note that, in the figures we refer to Augmented Lagrange multiplier as alm.

For the running time, projected gradient descent is the fastest method, then spherical method and interior point method. Augmented Lagrangian Method is faster than penalty method with Nelder-Mead Method until $n = 80$, then Nelder-Mead Method is slightly faster. Finally, the penalty method without Nelder-Mead Method has the largest running time.

For the interior point method, it has the lowest objective function value, but it violates the equality constraints significantly by having the infeasibility term equals 120 when $n = 70$.

For the objective function value, Augmented Lagrangian Method and penalty method with "L-BFGS-B" have the second lowest objective function, then the penalty method with Nelder-Mead Method and then the Spherical method. The projected gradient descent has the worst objective function.

For the infeasibility term, which is defined as the sum of the absolute difference of the norm of x_i and 1. Spherical method and projected gradient descent method are able to satisfy the constraint of placing the points on the unit sphere, as they guarantee that the norms equal to 1. However, Augmented Lagrangian Method violates the constraint of

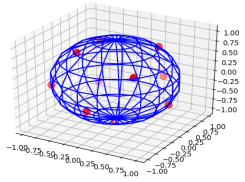

Figure 18: Spherical Coordinates in 3D, n = 10

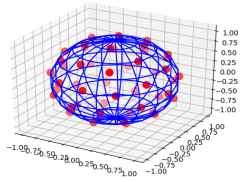

Figure 19: Spherical Coordinates in 3D, n = 50

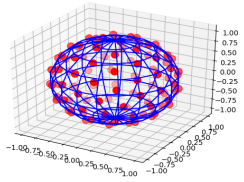

Figure 20: Spherical Coordinates in 3D, n = 100

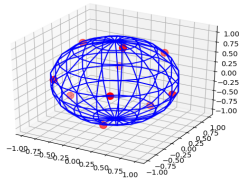

Figure 21: ALM in 3D, n = 10

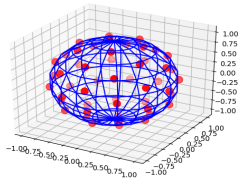

Figure 22: ALM in 3D, n = 10

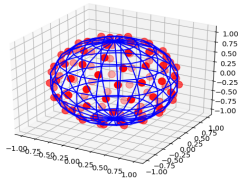

Figure 23: ALM in 3D, n = 50

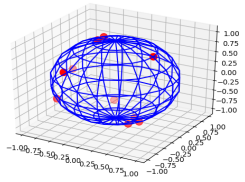

Figure 24: Projected Gradient Descent in 3D, n = 10

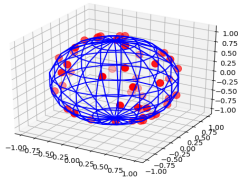

Figure 25: Projected Gradient Descent in 3D, n = 50

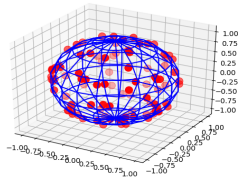

Figure 26: Projected Gradient Descent in 3D, n = 100

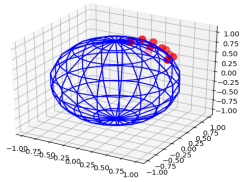

Figure 27: Penalty Method with Nelder Mead in 3D, n = 10

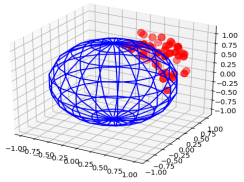

Figure 28: Penalty Method with Nelder Mead in 3D, n = 50

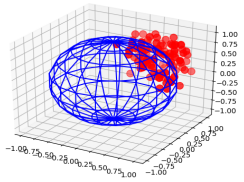

Figure 29: Penalty Method with Nelder Mead in 3D, n = 100

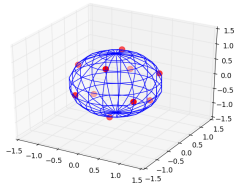

Figure 30: Penalty Method in 3D, n = 10

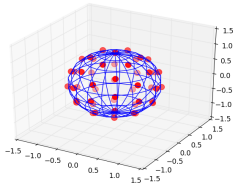

Figure 31: Penalty Method in 3D, n = 50

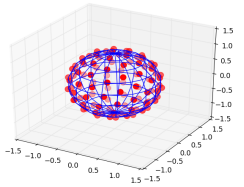

Figure 32: Penalty Method in 3D, n = 100

placing the points on the unit sphere and has an infeasibility term up to 4.8 when n=100 and penalty method with Nelder-Mead Method also violates these constraints and has an infeasibility term up to 25.6 when n = 100. The penalty method with "L-BFGS-B" violates these constraints and its infeasibility term is between the Augmented Lagrangian Method and penalty method with Nelder-Mead.

Figures 7, 8, 9, 10 and 11 show the results of positioning 100 points in 2 dimensional space. On one hand both Spherical and the projected gradient descent methods, as expected, place the points on the unite circle as both of them ensure the satisfiability of this constraint, however, the value of the objective function for these two methods is high as shown in figures 1 and 4. On the other hand, the Penalty method using Nelder-Mead fails to place the points on the unit circle and the placing of the points do not follow a certain pattern, yet it achieves a lower value for the objective function compared to the Spherical and the projected gradient descent methods. The Augmented Lagrange Multiplier and Penalty using L-BFGS do not place the points on the unit sphere, that is said, they assign the points on two concentric circles, while they achieve the second lowest value for the objective function after the interior point. Interior point results are shown in 12, note that, interior point is placing some points far away from the unit circle, we are omitting these outlier points on the figure to show the circle and they are measured by the infeasibility term in the previous figures.

Furthermore, we have extended k to 3 to evaluate how the methods perform in three dimensional space. Figures 13, 14 and 15 show the value of the objective function, infeasibility term and the running time of every method for different values of n and $k = 3$, respectively. Figure 16 shows the interior point method infeasibility term separately as it is the highest and adding it with the other methods scales the y axis and hence doesn't show the difference between the other methods. Moreover, figure 17 shows the interior point separately as it is implemented in MATLAB, while the other methods are implemented in Python.

For the objective function, figure 13 shows that interior point method achieves the lowest objective function. The second best are the augmented Lagrangian method, penalty method with L-BFGS-B, and spherical method as their performance is comparable. At the end, penalty method with Nelder-Mead is the worst. For the infeasibility term, figure 14 and 16 show that interior point has the worst infeasibility term. It is violating the constraints by 800 when n = 100. The second worse is the penalty method with Nelder-Mead. The other methods results are comparable. Finally, for the time, figures 15 and 17 show that interior point and the projected gradient descent are the fastest, then augmented Lagrangian method, then penalty method with Nelder-Mead, and finally penalty method with L-BFGS-B. Spherical method is faster than the penalty methods except at n = 80.

Figures from 18 to 35 show the placement of the points on the unit sphere for all the methods, when n = 10, 50 and 100 and k = 3. These figures show that projected gradient descent placement of points is not close to evenly distributed. Additionally, Penalty method with Nelder-Mead placement of points is only concentrated in a small portion of the unit sphere and this bad placement was reflected in having the penalty method with Nelder-Mead achieve the worst objective function. We can also see that the placement of points for the interior point method is not good either, many points are placed away from the unit sphere, this is reflected in having the worst infeasibility term. Other methods placement of points was reasonable.

9 Convex and Semi-definite Programming Reformulations

9.1 Convex Optimization

A Convex Optimization problem has the following form:

$$\min_x f_0(x)$$

$$\text{Subject to: } f_i(x) \leq i \text{ for } i = 1, ..., m$$

$$Ax = b$$

Where $f_i(x)$ is a convex function $\forall i$, and the equality constraints are affine.

9.2 Disciplined Convex Programming (DCP)

Disciplined Convex Programming (DCP) defines a set of rules to construct mathematical expressions with known curvature. By following these rules, you will be able to tell whether the final expression is convex, concave, affine or unknown. The detailed DCP rules can be found in `http://dcp.stanford.edu/rules`.

9.3 Thompson Problem

We have found two packages for convex optimization in Python. The first one is CVXOPT[1] and it needs the gradient and the hessian of the objective function and of the constraints. The second one is CVXPY[2] and it doesn't need the gradient or the hessian.

First, we wanted to validate whether the Thompson problem's objective function and its constraints form a convex function or not. Lets, first look at the objective function, recall that the objective function of the Thompson problem is:

$$\sum_{i<j} \frac{1}{||x_i - x_j||^2}$$

This objective function doesn't follow the DCP rules and hence we don't know its curvature. Although $||x_i - x_j||^2$ is a convex function, the inverse of the norm is not convex as according to DCP: $f(\texttt{expr})$ is convex if f is convex and either one of the following holds:

- f increases when `expr` increases and `expr` is convex.
- f decreases when `expr` decreases and `expr` is concave.

[1] http://cvxopt.org/
[2] http://www.cvxpy.org/en/latest/

- `expr` is affine.

In our case $f = \frac{1}{x}$ is convex $\forall x > 0$ and expr=$||x_i - x_j||^2$ is convex too. However, f increases when `expr` decreases and therefore it doesn't follow the DCP rules and we can't conclude what is the curvature of the objective function. We have validated that the objective function is indeed not convex using the following DCP analyzer [3]. Figure 36 shows the analysis of this function.

Now, lets look at the constraints, recall the constraints of Thompson problem are:

$$||x_i||_2 = 1 \ \forall i$$

In convex optimization, we need the equality constraint to be affine and in this case the constraints are not affine.

To conclude, we need to reformulate both the objective function and the constraints to be able to apply convex optimization methods.

9.4 Thompson Problem Convex Reformulations

9.4.1 Trial 1

We have reformulated the Thompson problem to the following:

$$\max_x \sum_{i=1}^{n} \sum_{j=1}^{i-1} ||x_i - x_j||^2 - \alpha \sum_{i=1}^{n} (||x_i||^2 - 1)^2$$

$$\text{Subject to: } ||x_i||^2 \leq 1 \ \forall i$$

We calculated the derivative and the hessian of this objective function and tried to optimize it using CVXOPT and CVXPY. However, CVXPY didn't run stating that we violated the DCP rules. Therefore, after closely looking at the objective function, we realized that the second term is not convex and this is why we have read about the DCP rules in detail.

9.4.2 Trial 2

We have reformulated the Thompson problem to the following:

$$\max_x \sum_{i=1}^{n} \sum_{j=1}^{i-1} ||x_i - x_j||^2 + \alpha \sum_{i=1}^{n} ||x_i||^2 \tag{6}$$

$$\text{Subject to: } ||x_i||^2 \leq 1 \ \forall i$$

Again we calculated the derivative and the hessian of this objective function and tried to optimize it using CVXOPT and CVXPY. However, we were unable to do that as we are maximizing a convex function and convex optimization standard form is either minimizing a convex function or the opposite i.e., maximizing a concave function.

In our search on how to solve the previous objective function, we found a theorem stating that when we maximize a convex function subject to convex constraints, then a solution exists on a vertex of the feasible region, we leave this step as a future work.

[3] `http://dcp.stanford.edu/analyzer`

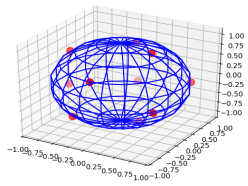

Figure 33: Interior Point Method in 3D, n = 10

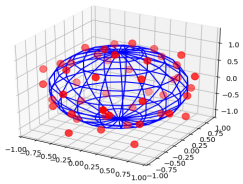

Figure 34: Interior Point Method in 3D, n = 50

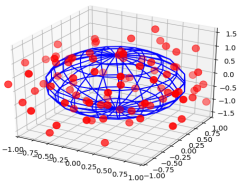

Figure 35: Interior Point Method in 3D, n = 100

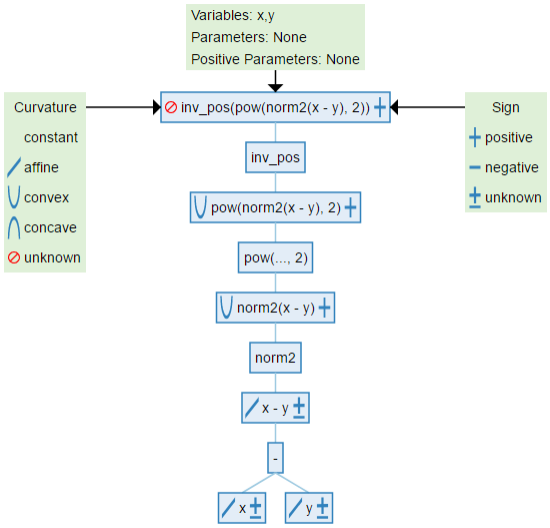

Figure 36: Analyzing $f(x)$

9.4.3 Trial 3

In this trial, we view the problem as minimizing the similarity between the data points and hence re-formulated the problem as:

$$\min_{S} ||S||_F^2$$
$$\text{Subject to: } S_{ii} = 1 \ \forall i$$

Which can be written as:

$$\min_{S} \sum_{i=1}^{n} \sum_{j=1}^{n} S_{ij}^2$$
$$\text{Subject to: } S_{ii} = 1 \ \forall i$$

Which is on the standard form of convex optimization. Giving it to CVXPY, the returned solution was always the identity matrix. Although the identity matrix is indeed the optimal solution for this problem, it is a trivial solution that does not allow the similarity to be negative as the objective function is sum of squared terms, this problem is fixed in the next formulation of the optimization problem.

9.5 Semi-definite Programming

Semidefinite Programming is a subfield of convex optimization, it has the following standard form:

$$\min_{X} C \bullet X$$

$$\text{Subject to: } A \bullet X = b \ , \ X \geq 0$$

Where $A \bullet B = \sum_{i=1}^{n} \sum_{j=1}^{m} A_i j B_i j$ and $X \geq 0$ means that X is a semidefinite matrix.

9.6 Semi-definite Programming Reformulations

Recall that Thompson problem objective function is:

$$\max_{x} \sum_{i<j} ||x_i - x_j||^2$$

$$\text{Subject to: } ||x_i||^2 = 1 \ \forall i$$

Let $S = X^T X$, the similarity matrix between the data points and therefore our objective function becomes:

$$\max_S \sum_{i<j} (S_{ii} - 2S_{ij} + S_{jj})$$

$$\text{Subject to: } S_{ii} = 1 \ \forall i$$

And to make the previous objective function on the semi-definite programming form, we add the constraints that $S \geq 0$, we get:

$$\max_S \sum_{i<j} (S_{ii} - 2S_{ij} + S_{jj})$$

$$\text{Subject to: } S_{ii} = 1 \ \forall i \ , \ S \geq 0$$

The previous formulation is an SPD formulation.

When we tried to code it in CVXPY, it yields infinity as an answer, to prevent that we add an extra constraint that $S \leq 1$ and our objective function becomes:

$$\max_S \sum_{i<j} (S_{ii} - 2S_{ij} + S_{jj})$$

$$\text{Subject to: } S_{ii} = 1 \ \forall i \ , \ S \geq 0 \ , \ S \leq 1$$

To retrieve the X after solving the previous objective function for S, the first method we tried is to do cholesky factorization of S. As the returned X is $n \times n$ and our dimensions is k, we used PCA to project X into a lower dimension space k that has the maximum variations and then we normalized X to the unit sphere, we call this method **"chol,≥ 0"**.

Another method to retrieve X is to note that S=$U\Sigma U^T$, the best low rank k approximation of S is $\widehat{S} = \widehat{U}\widehat{\Sigma}\widehat{U}^T$, where $\widehat{\Sigma} \in \mathbb{R}^{k \times k}$ and $\widehat{U} \in \mathbb{R}^{n \times k}$. As $S = X^T X$, therefore we set X $= \widehat{\Sigma}^{\frac{1}{2}}\widehat{U}^T$ and get $X \in \mathbb{R}^{n \times k}$, we call this method **"SVD,≥ 0"**.

As our similarity function can have negative values, as if two points in two opposite positions, then their similarities will be -1. We have relaxed the previous optimization problem to the following:

$$\max_S \sum_{i<j} (S_{ii} - 2S_{ij} + S_{jj})$$

$$\text{Subject to: } S_{ii} = 1 \ \forall i \ , \ S \geq -1 \ , \ S \leq 1$$

Note that the previous optimization function is no longer semidefinite programming but it is still a convex optimization. Therefore, we still use CVXPY library to solve it. Again, we use both Cholesky factorization then PCA and we call this method **"chol,≥ -1"**. We also use SVD and we call this method **"SVD,≥ -1"**.

We try another version with SVD as after running the experiment, the points were concentrated in a portion of the unit circle. We change our optimization problem to the following:

$$\max_S \sum_{i<j} (S_{ii} - 2S_{ij} + S_{jj})$$

$$\text{Subject to: } S_{ii} = 1 \ \forall i \ , \ S \geq -1 \ , \ S \leq 1 \ , \ \sum_{j=1}^{n} S_{ij} \leq 0 \ \forall i$$

The objective of the constraints $\sum_{j=1}^{n} S_{ij} \leq 0 \ \forall i$ is to ensure that the similarities between point i and the other points are negative and hence dissimilar. This was added to force the points to stay away from each other. We call this method **"SVD-strict,≥ -1"**.

9.7 Experimental Results

In this section, we compare the five variations of convex and semidefinite programming reformulations that we discussed in the previous section. The running time, objective function and the infeasibility term are reported for every method. Each method was run 5 times using a random starting point and then the mean is reported. Figures 37, 38, 39 and 40 show the comparison results. Figure 37 and 38 show that **SVD-stric, ≥ -1** is the best method by having the lowest objective function and **SVD, ≥ 0** has the worst objective function. Additionally, figure 40 shows the infeasibility term, as we can see all of the methods infeasibility terms are less that 10^{-14} as we project the points back to the unit circle by normalizing them. Furthermore, figure 39 shows the running time. The running time of the methods is comparable, however, **SVD-stric, ≥ -1** has slightly longer running time when $n = 100$.

Figures 41 to 55 show the points arrangements for the methods using $n = 10, 50$ and 100. As shown in these figures, **"chol,≥ -1"** well distributes the points around the circle. On the other hand, SVD methods is concentrating the points around a small region of the unit circle. **SVD-stric, ≥ -1** places the points equally distanced from each other, however, the method is only considering a portion of the unit circle and it is leaving the upper part of the unit circle empty!.

Comparing the convex and SPD reformulation methods to the methods compared in section 8, we can see that the convex and SPD reformulation methods have worse objective function values, but they are faster than the methods in section 8.

10 Homotopy Method

10.1 Overview

Homotopy map $H(x, \lambda)$ is defined as:

$$H(x, \lambda) = \lambda r(x) + (1 - \lambda)(x - a)$$

Where λ is a scalar value. Our objective is to use this homotopy map to get the solution of $r(x) = 0$. Lets note that solving $H(x, \lambda)$ when $\lambda = 0$, will get the solution $x = a$, and when $\lambda = 1$ will get the solution $r(x) = 0$.

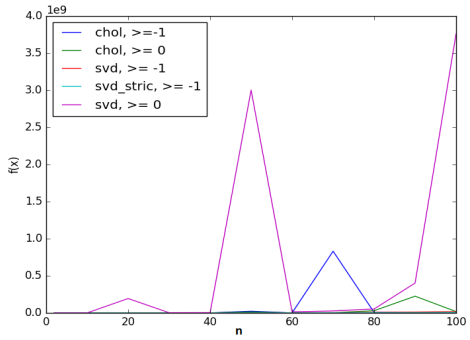

Figure 37: Objective Function Comparison

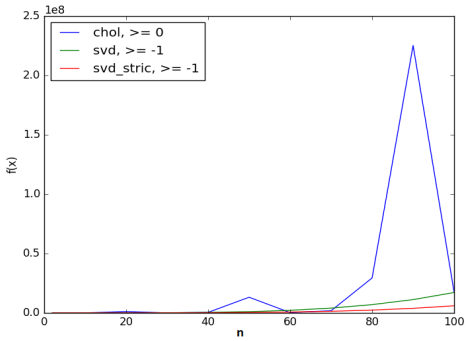

Figure 38: Objective Function Zoomed

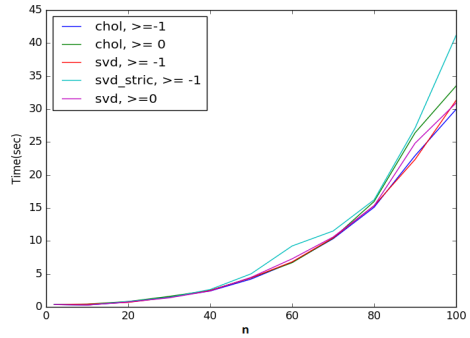

Figure 39: Time Comparison

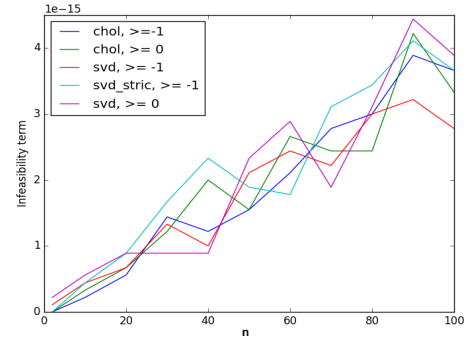

Figure 40: Infeasibility Term Comparison

The basic idea of the homotopy method is to start with $\lambda = 0$ and solution is $x = a$. Then, increase λ by small steps, finally when we reach $\lambda = 1$, we will get our desired solution that $r(x) = 0$. Therefore, by tracking this path, we will be able to solve our problem.

10.2 Thompson Problem

To solve the Thompson problem, we note that at our minimizer point we need the gradient of the objective function to be zero and $||x_i||^2 - 1 = 0$ for all i. Therefore, we define our function $r(x)$ to be:

$$r(x) = \begin{bmatrix} g(x) \\ ||x_i||^2 - 1 \end{bmatrix}$$

Where $g(x) \in \mathbb{R}^{nk}$ is the gradient of the objective function and hence $r(x) \in \mathbb{R}^{nk+n}$.

Note that we only consider the first order optimality conditions, hence we can not guarantee that we will reach a minimizer, we can only guarantee that we will reach a stationary point.

We used the following code `https://math.berkeley.edu/~mgu/MA128BSpring2010/` for Homotopy in MATLAB and we have only considered the case of $n = 2$ and $k = 2$. The

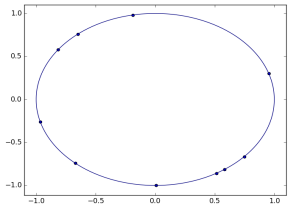

Figure 41: **chol,**≥ 0 n $= 10$

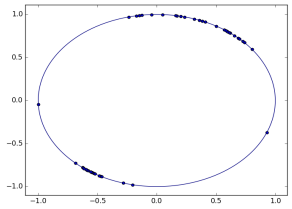

Figure 42: **chol,**≥ 0 n $= 50$

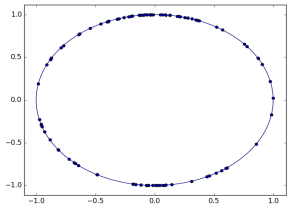

Figure 43: **chol,**≥ 0 n=100

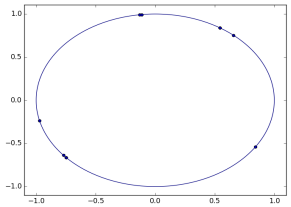

Figure 44: **chol,**≥ -1 n $= 10$

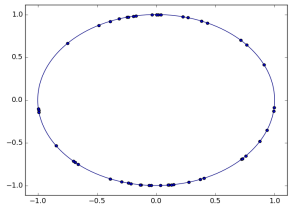

Figure 45: **chol,**≥ -1 n $= 50$

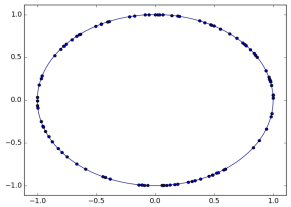

Figure 46: **chol,**≥ -1 n=100

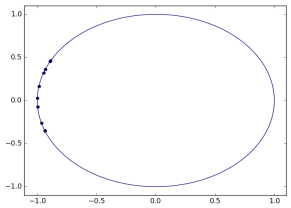

Figure 47: **SVD,**≥ 0 n $= 10$

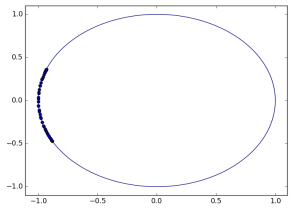

Figure 48: **SVD,**≥ 0 n $= 50$

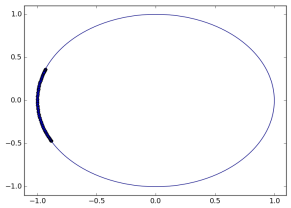

Figure 49: **SVD,**≥ 0 n=100

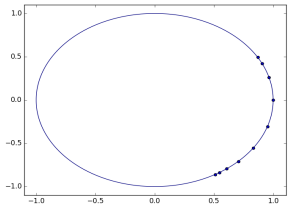

Figure 50: **SVD,**≥ -1 n $= 10$

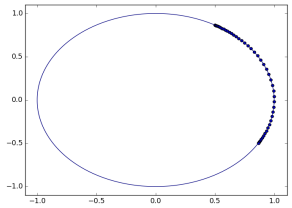

Figure 51: **SVD,**≥ -1 n $= 50$

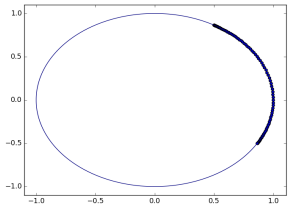

Figure 52: **SVD,**≥ -1 n=100

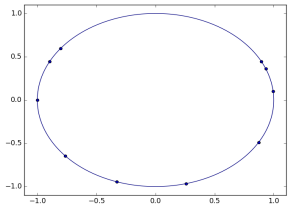

Figure 53: **SVD-stric,**≥ -1 n $= 10$

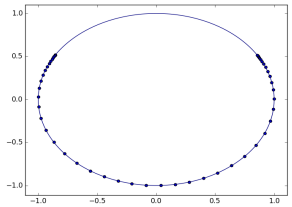

Figure 54: **SVD-stric,**≥ -1 n $= 50$

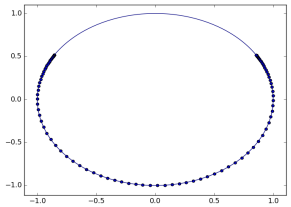

Figure 55: **SVD-stric,**≥ -1 n $= 100$

code takes as an input both the homotopy function and the Jacobian of the function. Our function in the case of $n = 2$ and $k = 2$ is:

$$H(x, \lambda) = \begin{bmatrix} -2 \times (x - z) \times 1.0/((x - z)^2 + (y - m)^2)^2 \\ -2 \times (y - m) \times 1.0/((x - z)^2 + (y - m)^2)^2 \\ -2 \times (z - x) \times 1.0/((x - z)^2 + (y - m)^2)^2 \\ -2 \times (m - y) \times 1.0/((x - z)^2 + (y - m)^2)^2 \\ x^2 + y^2 - 1 \\ z^2 + m^2 - 1 \end{bmatrix}$$

Where $X_1 = \begin{bmatrix} x \\ y \end{bmatrix}$ and $X_2 = \begin{bmatrix} z \\ w \end{bmatrix}$.

By running the code in this special case, we get the following solution:

$$X_1 = \begin{bmatrix} -4.0339 \\ 3.4443 \end{bmatrix}$$

$$X_2 = \begin{bmatrix} 3.4443 \\ -4.0339 \end{bmatrix}$$

Therefore, the code did place the two points far from each other in opposite directions but they are not on the unit circle.

11 Randomized Method

One possible extension is to use a randomized algorithm to place the points. The main idea is to solve a sequence of easier problems, which are given the locations of $n - 1$ points, find the location of the n^{th} point. To achieve this we start with random positions of the n points on the unit sphere. Then, iterate over the points and at each iteration choose a point and given the positions of the other $n - 1$ points assign this point a location, a good heuristic to this is placing it in the middle between the pair of points that has the highest distance. The usage of Lovasz local lemma can be investigated in order to prove a bound on the quality of this random algorithm.

12 Conclusion

In this report, we have investigated several techniques for optimizing the Thompson problem. We have found that the Augmented Lagrange Multiplier method had a low objective function value and a low infeasibility term while maintaining a good running time. While the Projected Gradient Descent method had the lowest running time and had a zero infeasibility term, it had the highest objective function. The Spherical Coordinates method had a zero infeasibility term while maintaining good running time and slightly high objective function. For the Interior Point method, it had a low running time and had a low objective function, however, it had a high infeasibility term. In addition, the convex formulation we proposed had a low running time and a zero infeasibility term, but it had a high objective function.

Based on that, if fast running time is the most important property then Projected Gradient Descent is the most suitable, but if the value of the objective function is the main criterion then the Interior point method would be a better choice. In case of enforcing the constraints (i.e, placing the points on the unit sphere) is more important, then Projected Gradient Descent, Spherical coordinates method or convex formulation are the best options. Finally, Augmented Lagrange Multiplier is a good compromise that does fairly well in all the metrics.

13 References

- Nocedal, Jorge, and Stephen J. Wright. Numerical optimization 2nd., 2006.
- UIUC Big Data Optimization Course Lecture 10: `http://niaohe.ise.illinois.edu/IE598/pdf/IE598-lecture10-projected%20gradient%20descent.pdf`
- `https://en.wikipedia.org/wiki/Spherical_coordinate_system`
- Gleich, D. F., Rasmussen, M., Lang, K., & Zhukov, L. The world of music: User ratings; spectral and spherical embeddings; map projections. Online report, 2006.
- Boyd, Stephen, and Lieven Vandenberghe. Convex optimization. Cambridge university press, 2004.
- `https://en.wikipedia.org/wiki/Semidefinite_programming`
- CMU Linear and Semidefinite Programming (Advanced Algorithms) Course Lecture 10: `https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f11/www/notes/lecture10.pdf`
- CMU Advanced Approximation Algorithms Course Lecture 14: `http://www.cs.cmu.edu/~anupamg/adv-approx/lecture14.pdf`
- Watson, Layne T., and Raphael T. Haftka. Modern homotopy methods in optimization. Computer Methods in Applied Mechanics and Engineering 74.3 (1989): 289-305.
- Alloula, Karim, Jean-Pierre Belaud, and Jean-Marc Le Lann. An homotopy method for global optimization of continuous models. Chem. Eng. Trans., 24 (2011): 325-330.
- Beck, Jf. "An algorithmic approach to the Lov local lemma. I." Random Structures & Algorithms 2.4 (1991): 343-365.