CS 520: Computational Methods in Optimization

Optimization on the Surface of the (Hyper)-Sphere

Course Project Report

Due on Friday, May 2, 2014

Parameswaran Raman*
Jiasen Yang†

[1]Ph.D. Student, Department of Computer Science, Purdue University
[2]Ph.D. Student, Department of Statistics, Purdue University

Contents

1 Introduction

1.1 The Thompson Problem

The *Thompson problem* was posed by physicist J. J. Thompson to determine the minimum electrostatic potential energy configuration of n electrons on the surface of a unit sphere that repel each other with a force given by *Coulomb's law* [1].

It is clear that for the case of $n = 2$, the optimal configuration consists of electrons at antipodal points. For $n = 3$, electrons reside at the vertices of an equilateral triangle about a great circle. Minimum energy configurations have also been rigorously identified for the cases of $n = 4, 5, 6, 12$ [1]. However, no analytical solutions have been found in more general cases. This provides a good opportunity for us to study the problem using methods of numerical optimization.

1.2 Mathematical Formulation

In this project, our goal is to study the following modification of the Thompson problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{i-1} \frac{1}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2} \\ \text{subject to} \quad & \boldsymbol{x}_i \in \mathbb{R}^k,\ \|\boldsymbol{x}_i\|_2 = 1,\ 1 \le i \le n. \end{aligned} \tag{1}$$

where each $\boldsymbol{x}_i$ represents the coordinates of a point lying on the k-dimensional hyper-sphere.

For convenience, we can stack all the $\boldsymbol{x}_i$'s into a $k \times n$ matrix denoted by

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_n \end{bmatrix},$$

and rewrite the optimization problem in the form

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{X}) := \sum_{i=1}^{n} \sum_{j=1}^{i-1} \frac{1}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2} \\ \text{subject to} \quad & \boldsymbol{X} \in \mathbb{R}^{k \times n},\ \operatorname{diag}\{\boldsymbol{X}\boldsymbol{X}^{\mathsf{T}}\} = \boldsymbol{e}, \end{aligned} \tag{2}$$

where $\boldsymbol{e} = [1, 1, \cdots, 1]^{\mathsf{T}} \in \mathbb{R}^k$. In this report, our discussion will primarily focus on the case of $k = 3$ since the solutions are easy to visualize. However, most of the methods we discuss can be directly applied to cases where k is moderately large as well.

The remainder of this report is organized in the following manner. In Section 2, we apply a sequence of optimization methods to study Problem (2). Apart from applying well-known optimization methods including the penalty method, the augmented Lagrangian method, the interior-point method, stochastic gradient descent, and the Nelder-Mead method, we also discuss unconstrained optimization via spherical coordinates and creatively introduce a *Coulomb force method* to study the problem. We provide visualizations of the obtained solutions and compare their accuracy using the converged values of the objective function. In Section 3, we explore more creative ideas to study the problem. In particular, we develop a convex reformulation of the constraint, and briefly review the *spherical packing problem.*

2 Optimization Methods

2.1 Unconstrained Optimization via Spherical Coordinates

For the case of $k = 3$, problem (2) becomes minimization of the objective function on the surface of the unit sphere in $\mathbb{R}^3$. In this case, a natural way of converting the problem into an unconstrained optimization problem is to denote the points using the spherical coordinates

$$\begin{cases} x = \sin\phi \cos\theta; \\ y = \sin\phi \sin\theta; \\ z = \cos\phi, \end{cases}$$

where $\phi \in [0, \pi]$ and $\theta \in [0, 2\pi]$. Figure 1 provides an illustration of the angles ϕ and θ.

Under the spherical coordinates, problem (2) becomes the unconstrained problem

$$\text{minimize} \quad f(\phi, \theta) := -\sum_{i=1}^{n} \sum_{j=1}^{i-1} \frac{1}{2[\sin\phi_i \sin\phi_j \cos(\theta_i - \theta_j) + \cos\phi_i \cos\phi_j - 1]}. \tag{3}$$

Differentiating with respect to ϕ_i and θ_i yields the gradients

$$\frac{\partial f}{\partial \phi_i} = \sum_{j \neq i} \frac{\cos\phi_i \sin\phi_j \cos(\theta_i - \theta_j) - \sin\phi_i \cos\phi_j}{2[\sin\phi_i \sin\phi_j \cos(\theta_i - \theta_j) + \cos\phi_i \cos\phi_j - 1]^2}, \tag{4}$$

$$\frac{\partial f}{\partial \theta_i} = -\sum_{j \neq i} \frac{\sin\phi_i \sin\phi_j \sin(\theta_i - \theta_j)}{2[\sin\phi_i \sin\phi_j \cos(\theta_i - \theta_j) + \cos\phi_i \cos\phi_j - 1]^2}. \tag{5}$$

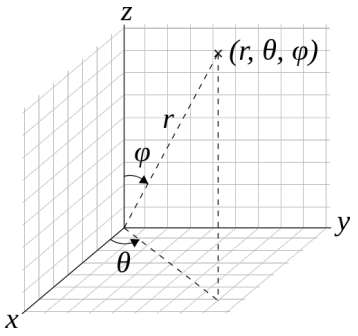


Figure 1: Illustration of the spherical coordinates [2].

We check the correctness of our gradient computations using the forward finite differences method implemented in the `gradientcheck` function contained in the Poblano toolbox. The MATLAB wrapper code and output is shown as follows:

```
% Gradient Check for spherical_main.m
n = 2;
phi = 2*pi*rand(1,n);
```

```
theta = pi*rand(1,n);
out = gradientcheck(@(x) spherical_obj(x, n), [phi theta]',
          'DifferenceType','forward');
out
out.G
out.GFD
```

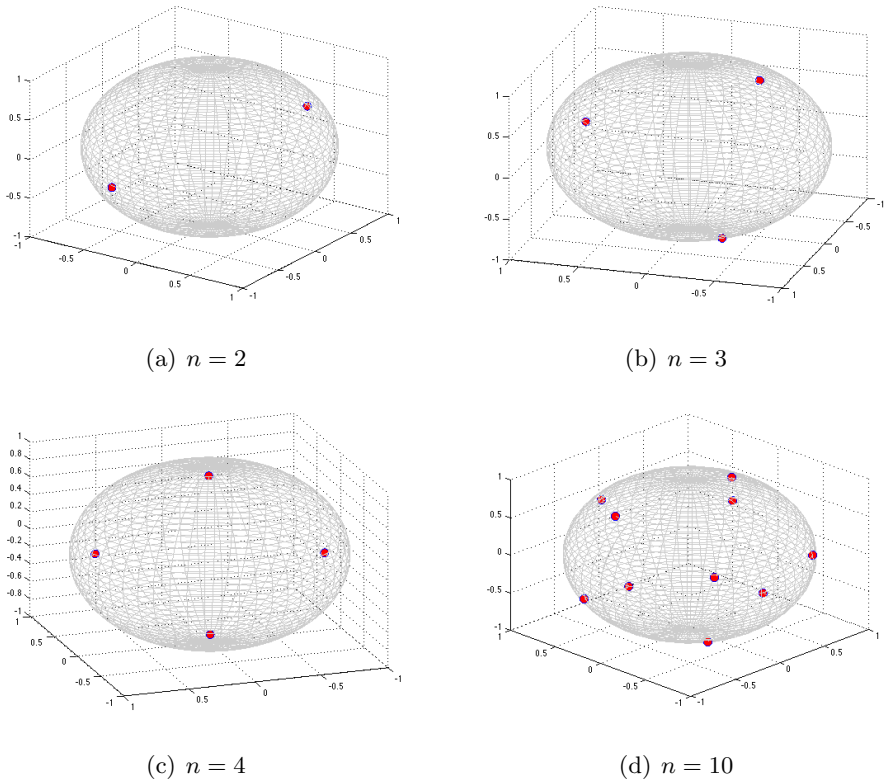
```
>> gradcheck
out =
                    G: [4x1 double]
                  GFD: [4x1 double]
              MaxDiff: -7.6871e-06
           MaxDiffInd: 1
    NormGradientDiffs: 9.4885e-06
        GradientDiffs: [4x1 double]
               Params: [1x1 struct]

ans =
 -148.9590
   55.3360
  -37.4119
   37.4119

ans =
 -148.9590
   55.3360
  -37.4119
   37.4119
```

From the above output, we verify that the numerical gradients matches our analytically computed gradients with high precision.

To solve the optimization problem, we notice that evaluation of the objective function and its gradients takes $\mathcal{O}(n^2)$ computation, whereas evaluation of the Hessian would take $\mathcal{O}(n^3)$ computation. We therefore utilize Quasi-Newton methods such as L-BFGS to solve the optimization problem. The `lbfgs` function in Poblano is used to solve problem (3) for various values of n. Figure 2 visualizes the solutions for the cases of $n = 2, 3, 4, 10$.



(a) $n = 2$ (b) $n = 3$

(c) $n = 4$ (d) $n = 10$

Figure 2: Visualization of the solutions for the unconstrained spherical coordinates method.

2.2 Projected Gradient Descent

The methods discussed beyond this section can be directly applied to cases where $k \geq 3$. Differentiation of the objective function in (2) yields the gradients

$$\frac{\partial f(\boldsymbol{X})}{\partial x_{ij}} = -2 \sum_{l \neq i} \frac{x_{ij} - x_{lj}}{\|\boldsymbol{x}_i - \boldsymbol{x}_l\|_2^4}, \quad 1 \leq i, j \leq n; \tag{6}$$

which can be written more compactly as

$$\nabla_{\boldsymbol{x}_i} f(\boldsymbol{X}) = -2 \sum_{l \neq i} \frac{\boldsymbol{x}_i - \boldsymbol{x}_l}{\|\boldsymbol{x}_i - \boldsymbol{x}_l\|_2^4}, \quad 1 \leq i \leq n. \tag{7}$$

There are many ways to reformulate the constraints in (2) in order to convert it into an unconstrained problem, as we shall demonstrate in subsequent sections. However, let us start by considering a simple and intuitive algorithm, in which we perform gradient descent on the original objective function in (2), and after each iteration we project the updated points $\boldsymbol{x}_1, \cdots \boldsymbol{x}_n$ back onto the surface of the (hyper)-sphere (which corresponds to normalizing the columns of $\boldsymbol{X}$). Clearly, the convergence of this algorithm could not be guaranteed; yet this algorithm appears to work empirically under careful initializations.

2.3 Penalty Method

In this section, we convert problem (2) into an unconstrained problem by relaxing the constraint into the objective function using a regularization parameter λ which could be tuned by cross validation.

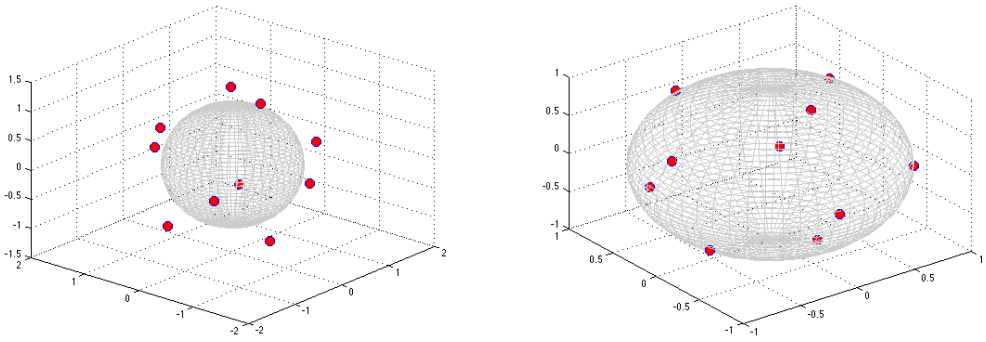
The penalized objective function takes the form

$$f(\boldsymbol{X}) := \sum_{i=1}^{n} \sum_{j=1}^{i-1} \frac{1}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2} + \frac{\lambda}{2} \sum_{i=1}^{n} \left(\|\boldsymbol{x}_i\|^2 - 1 \right)^2. \tag{8}$$

Here, we modified the constraint to $(\|\boldsymbol{x}_i\|^2 - 1)^2/2 = 0$ in order to avoid cancellation of positive/negative errors. The gradients of the penalized objective function (8) are given by

$$\nabla_{\boldsymbol{x}_i} f(\boldsymbol{X}) = -2 \sum_{l \neq i} \frac{\boldsymbol{x}_i - \boldsymbol{x}_l}{\|\boldsymbol{x}_i - \boldsymbol{x}_l\|_2^4} + \lambda \left(\|\boldsymbol{x}_i\|^2 - 1 \right) \boldsymbol{x}_i, \quad 1 \leq i \leq n. \tag{9}$$

The intuition behind the penalty method is that as $\lambda \to \infty$, minimizing the objective function (8) should be equivalent to solving the original constrained optimization problem (2). In other words, for large values of λ, minimizers of (8) should approximately lie on the surface of the unit sphere; this might not be the case for small values of λ. However, since accurately solving the penalized problem with a large value of λ give rise to computational difficulties, we instead start by setting the value of λ to be quite small, say $\lambda = 1$, and then solve the penalized problem; after which we gradually increase the value of λ in small increments, solving each more heavily penalized problem with $\boldsymbol{X}$ initialized as the solution to the previous problem. Solving this sequence of penalized problems would eventually provide us with a solution with high precision.



(a) $\lambda = 1$ (b) $\lambda = 100$

Figure 3: Visualization of the solutions for the penalty method ($n = 10$).

Figure 3 illustrates the intuition we described. In the figure, we observe that for the penalized problem with $\lambda = 1$, the constraint is poorly enforced in that none of the points

lie on the surface of the unit sphere. However, for the penalized problem with $\lambda = 100$, the constraint appears to be very well approximated.

As before, we verify the correctness of our gradient computations using the forward finite differences method as follows:

```
% Gradient Check for penalty_main.m
N = 2;
p = 3;
X = 2*rand(n, p)-1;
x = X; x = x(:);
out = gradientcheck(@(x) penalty_obj(x, n, p, 1), x,
          'DifferenceType', 'forward');
out
out.G
out.GFD
```

```
>> gradcheck
out =
                    G: [6x1 double]
                  GFD: [6x1 double]
              MaxDiff: -7.1876e-07
           MaxDiffInd: 4
    NormGradientDiffs: 1.0145e-06
        GradientDiffs: [6x1 double]
               Params: [1x1 struct]

ans =
   -5.0305
    4.4829
  -19.0028
   19.1971
   -1.1199
    1.2200

ans =
   -5.0305
    4.4829
  -19.0028
   19.1971
   -1.1199
    1.2200
```

Having verified the gradients, we utilize the `lbfgs` function in Poblano to minimize the objective function (8) for various values of n. Some converged values are listed in Table 1.

n	10	20	30	40
$f(\boldsymbol{X}^*)$	24.7424	129.9554	342.4396	659.5119

Table 1: Converged values of the objective function for the penalty method.

2.4 Augmented Lagrangian Method

The augmented Lagrangian method can be viewed as an extension of the penalty method, and the objective function takes on a similar form:

$$f(\boldsymbol{X}) := \sum_{i=1}^{n}\sum_{j=1}^{i-1}\frac{1}{w\|\boldsymbol{x}_i-\boldsymbol{x}_j\|_2^2} + \frac{\lambda}{2}\sum_{i=1}^{n}\left(\|\boldsymbol{x}_i\|^2-1\right)^2 - \sum_{i=1}^{n}\mu_i\left(\|\boldsymbol{x}_i\|^2-1\right), \tag{10}$$

with its gradients given by

$$\nabla_{\boldsymbol{x}_i} f(\boldsymbol{X}) = -2\sum_{l\neq i}\frac{\boldsymbol{x}_i-\boldsymbol{x}_l}{\|\boldsymbol{x}_i-\boldsymbol{x}_l\|_2^4} + 2\lambda\left(\|\boldsymbol{x}_i\|^2-1\right)\boldsymbol{x}_i - 2\mu_i\boldsymbol{x}_i, \quad 1\leq i\leq n. \tag{11}$$

We note that in both the penalty method and the augmented Lagrangian method, since the constraint is only loosely approximated in the beginning, we do not need to enforce a very stringent tolerance level as the solutions are will only be serving as initial values for the next problem in the sequence. Therefore, computationally it would be more efficient to gradually decrease the tolerance setting as λ increases, which would save us time in solving the initial problems.

As always, we verify the correctness of our gradient computations using the forward finite differences method as follows:

```
% Gradient Check for augmented_lagrangian_main.m
N = 3;
p = 3;
X = 2*rand(n, p)-1;
for i = 1:n
    X(i,:) = X(i,:)/norm(X(i,:));
end
x = X; x = x(:);
lambda = 1;
mu = zeros(n,1);
out = gradientcheck(@(x) augmented_lagrangian_obj(x,n,p,lambda,mu), x,
          'DifferenceType', 'forward');
out
out.G
out.GFD
```

```
>> gradcheck
out =
                    G: [9x1 double]
                  GFD: [9x1 double]
              MaxDiff: 4.6926e-08
           MaxDiffInd: 4
    NormGradientDiffs: 7.6902e-08
        GradientDiffs: [9x1 double]
               Params: [1x1 struct]

ans =
    1.1957
    1.4398
   -2.6354
   -1.0928
    1.8569
   -0.7641
   -0.7383
   -1.5598
    2.2981

ans =
    1.1957
    1.4398
   -2.6354
   -1.0928
    1.8569
   -0.7641
   -0.7383
   -1.5598
    2.2981
```

Having verified the gradients, we utilize the `lbfgs` function in Poblano to minimize the objective function (10) for various values of n. Some converged values are listed in Table 2.

n	10	20	30	40
$f(\boldsymbol{X}^*)$	24.7452	129.9907	337.3002	655.3411

Table 2: Converged values of the objective function for the augmented Lagrangian method.

2.5 Interior-Point Method

Although the interior-point method is commonly known as an effective method for solving optimization problems with inequality constraints, we also applied it to solve problem (2). In particular, we utilize the MATLAB implementation of the interior-point method by calling the `fmincon` function.

We apply the interior-point method to minimize the objective function (8) for various values of n, and some converged values are listed in Table 3. We notice that the converged function values for the penalty method, the augmented Lagrangian method, and the interior-point method are quite close to each other.

n	10	20	30	40
$f(\boldsymbol{X}^*)$	25.0678	132.8934	338.0972	640.3781

Table 3: Converged values of the objective function for the interior-point method.

Additionally, to compare the unconstrained spherical coordinates method, the penalty method, and the interior-point method, Figure 4 plots the value of the objective function against iteration number for the cases $n = 10$ and $n = 30$, while Figure 5 illustrates the time it takes until convergence for varying values of n.

2.6 Stochastic Gradient Descent

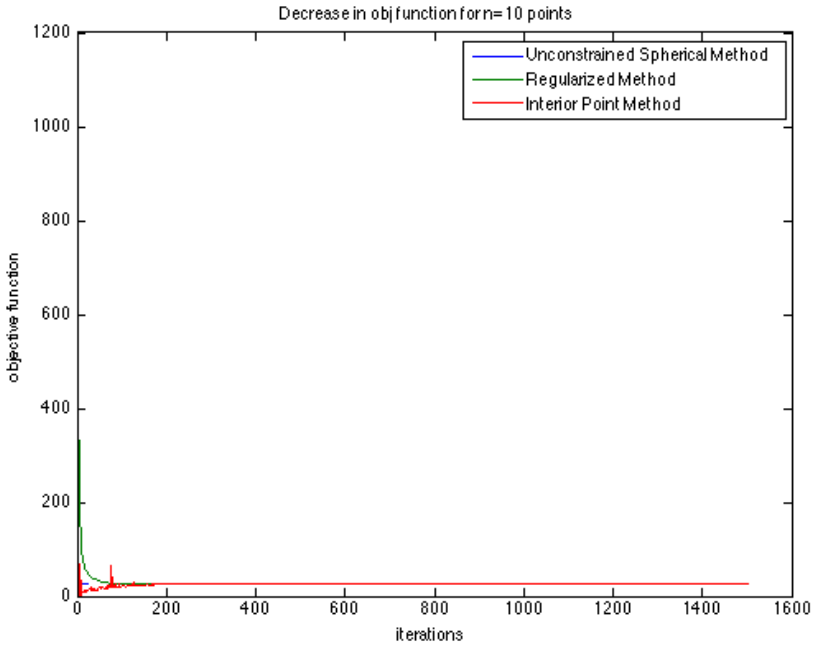
We remark that the evaluation of the gradient $\nabla_{\boldsymbol{x}_i} f(\boldsymbol{X})$ in equation (14) requires a summation over $n-1$ points. Thus, if n becomes very large, it can be quite costly to evaluate the gradient. In this case, instead of performing *batch* gradient descent updates which takes $\mathcal{O}(n)$ computation, we could turn to utilize *stochastic* gradient descent updates which takes $\mathcal{O}(1)$ computation instead. Even though the stochastic gradient descent algorithm would need a lot more iterations to converge, it can take much less running time in total as compared to batch gradient descent. In addition, stochastic gradient descent would also be an appealing algorithm if we wish to consider an *online* version of the Thompson problem.

The objective function we use for stochastic gradient descent is the same as that of the penalty method, in which we relax the constraint using a regularization parameter λ.

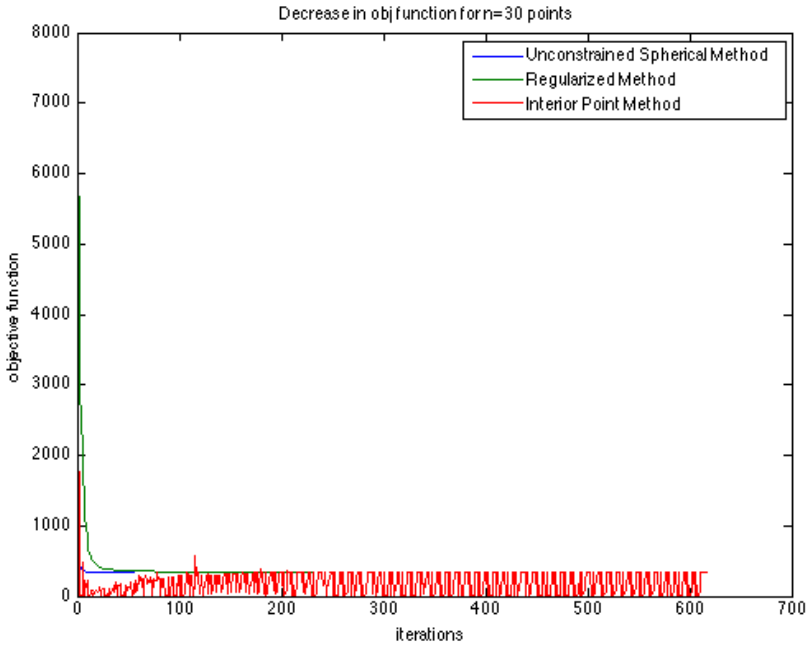
$$f(\boldsymbol{X}) := \sum_{i=1}^{n} \sum_{j=1}^{i-1} \frac{1}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2} + \frac{\lambda}{2} \sum_{i=1}^{n} \left(\|\boldsymbol{x}_i\|^2 - 1 \right)^2. \tag{12}$$

The gradients of the penalized objective function (12) are given by

$$\nabla_{\boldsymbol{x}_i} f(\boldsymbol{X}) = -2 \sum_{l \neq i} \frac{\boldsymbol{x}_i - \boldsymbol{x}_l}{\|\boldsymbol{x}_i - \boldsymbol{x}_l\|_2^4} + \lambda \left(\|\boldsymbol{x}_i\|^2 - 1 \right) \boldsymbol{x}_i, \quad 1 \leq i \leq n. \tag{13}$$

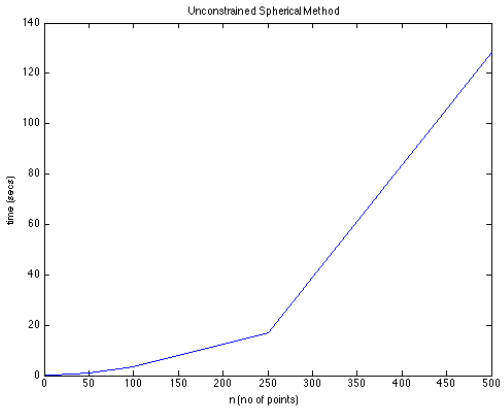


(a) $n = 10$

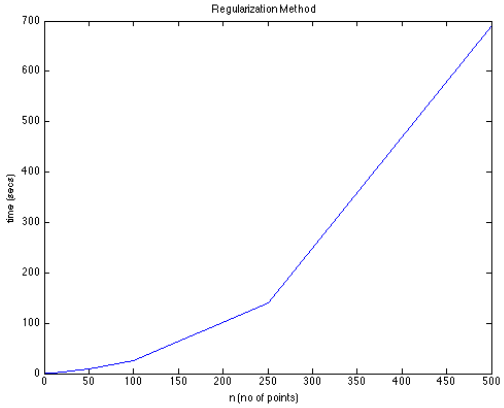


(b) $n = 30$

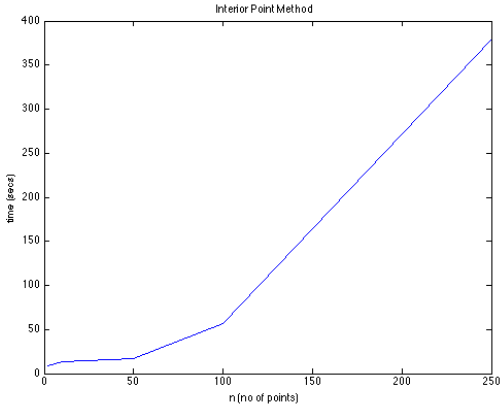
Figure 4: Plots of the objective function value vs iteration number for the spherical method, the penalty (regularized) method, and the interior-point method.



(a) Spherical method



(b) Regularization method



(c) Interior-point method

Figure 5: Plots of the time until convergence for varying number of points.

To derive the stochastic gradient descent algorithm, we introduce the notation

$$\nabla_{\boldsymbol{x}_i,\boldsymbol{x}_l} f(\boldsymbol{X}) = -2\frac{(\boldsymbol{x}_i - \boldsymbol{x}_l)}{\|\boldsymbol{x}_i - \boldsymbol{x}_l\|_2^4} + \frac{\lambda}{n-1}\left(\|\boldsymbol{x}_i\|^2 - 1\right)\boldsymbol{x}_i, \quad 1 \le i \le n. \tag{14}$$

The stochastic gradient descent algorithm to minimize the above objective function is outlined in Algorithm 1. Instead of summing over $n-1$ points to perform each gradient update, here we randomly sample a pair of points and perform stochastic updates only for that pair.

Algorithm 1 Stochastic Gradient Descent

Require: n (# of points), p (dimension), λ (regularization parameter), γ (step size), $iters$.

for $t = 1$ to $iters$ **do**

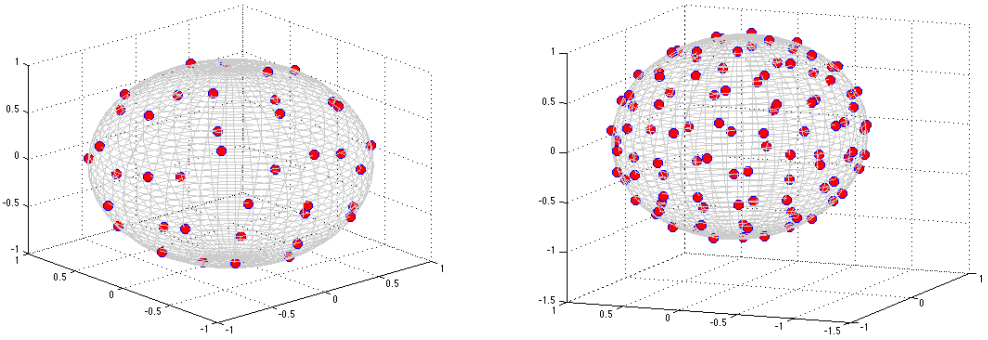
Randomly sample a pair of different points $\boldsymbol{x}_i, \boldsymbol{x}_l$ from the n points.

Update $\boldsymbol{x}_i, \boldsymbol{x}_l$ by scaling up the stochastic gradients:

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \gamma(n-1)\nabla_{\boldsymbol{x}_i,\boldsymbol{x}_l} f(\boldsymbol{X});$$
$$\boldsymbol{x}_l \leftarrow \boldsymbol{x}_l - \gamma(n-1)\nabla_{\boldsymbol{x}_i,\boldsymbol{x}_l} f(\boldsymbol{X});$$

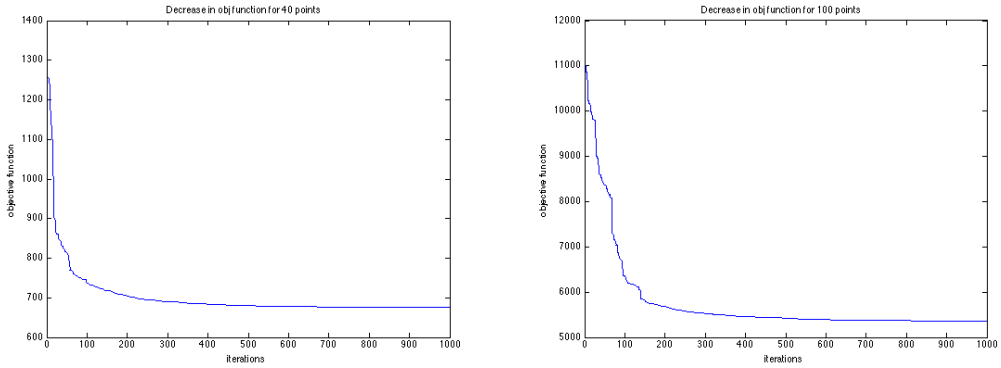
end for

Our experiments show that the algorithm converges to a local optima after carefully tuning the parameters. Compared to previous methods, stochastic gradient descent clearly wins out in terms of execution time when the number of data points n is large. The solutions obtained for $n = 40$ and $n = 100$ are shown in Figure 6, while Figure 7 illustrates how the value of the objective function decreases over time.



(a) $n = 40$ (b) $n = 100$

Figure 6: Plots of the solutions for stochastic gradient descent.



(a) $n = 40$ (b) $n = 100$

Figure 7: Plots of the objective function value vs iteration for stochastic gradient descent.

2.7 Nelder-Mead Method

During the project, we also applied the Nelder-Mead method, a popular derivative-free method for unconstrained optimization, to minimize the penalized objective function (8). In particular, we utilize the MATLAB implementation of the Nelder-Mead method by calling the `fminsearch` function. Our experiments show that the method behaves fairly well.

2.8 Coulomb Force Method

All the methods we have considered so far are from an optimization perspective. In this section, we attempt to consider the problem using the original physical interpretation. According to Coulomb's law, electrons i and j repel each other with a force with magnitude proportional to $1/\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2$. In what we term as the *Coulomb force method*, we first randomly arrange the n points on the surface of the sphere, and then for each point, we simulate the effects of Coulomb's force that the other $n-1$ points exerted to this point by shifting it slightly in the corresponding direction. The summation of Coulomb's forces would clearly move the point out of the surface, and thus we project the point back to the surface of the sphere after each motion. We iterate this process through all the data points, and then carry out several passes through the data.

The intuition behind Coulomb's force method is that in each step we move a single point according to Coulomb's law while enforcing the spherical constraint; and then we iterate through all the data points several times in the hope that the process eventually stabilizes and we arrive at a state where the energy attains a local minimum. By implementing different initializations, there is also a chance that we might arrive at the global minimum. Clearly, the convergence of this method could not be guaranteed, yet our experiments show empirically that its performance is quite satisfactory, although its speed of convergence is rather slow as compared to previously discussed methods.

3 Exploratory Ideas

3.1 Convex Reformulation of the Constraint

In this section, we consider a convex reformulation of the constraint $\|\boldsymbol{x}_j\|_2 = 1$. Inspired by [3], we note that the following fact holds:

$$\mathbb{E}\|\boldsymbol{A}\boldsymbol{x}_j\|_1 = \sum_{i=1}^{m} \mathbb{E}|\boldsymbol{a}_i^\top \boldsymbol{x}_j| = c\,m\|\boldsymbol{x}_j\|_2,$$

where $c = \sqrt{2/\pi}$ is a constant, $\boldsymbol{A}$ is a Gaussian random matrix of size $m \times k$ with its entries a_{ij} *i.i.d.* $\sim \mathcal{N}(0,1)$, and $\boldsymbol{a}_i$ denotes the i-th row of $\boldsymbol{A}$. The expectation is taken with respect to the elements of $\boldsymbol{A}$, and the last equation follows from the fact that the first absolute moment of the standard Normal distribution equals c.

The above observation indicates that the linear constraint $\|\boldsymbol{A}\boldsymbol{x}_j\|_1 = c\,m$ is equivalent to the original constraint $\|\boldsymbol{x}_j\|_2 = 1$ in expectation. Thus, when n or k becomes very large, instead of solving (2) we can work with the following simpler problem instead:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{i-1} \frac{1}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2} \\ \text{subject to} \quad & \boldsymbol{x}_i \in \mathbb{R}^k,\ \|\boldsymbol{A}\boldsymbol{x}_i\|_1 = c\,m,\ 1 \le i \le n. \end{aligned} \tag{15}$$

3.2 The Sphere Packing Problem

We note that Thompson's problem is closely related to the *sphere packing problem*, which aims to find the largest diameter of n identical circles that can be placed on the sphere without overlapping [4]. Figure 8 provides an illustration of the problem.

A solution to the sphere packing problem should possess the following properties:

1) Each point $\boldsymbol{x}_i$ is equally distant to its three nearest neighbors.

2) There exists d such that $\min_{i\neq j} \|\boldsymbol{x}_i - \boldsymbol{x}_j\| = d$ for all i.

According to [4], there is a simple algorithm for solving the problem:

Step 1: Put each point in the centre of its three nearest neighbors as long as Property 1) is not satisfied.

Step 2: Calculate $d_i = \min_{i\neq j} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ for all i. If the d_i's are not equal, take the point $\boldsymbol{x}_i$ with the largest value of d_i and move its three nearest neighbors closer to $\boldsymbol{x}_i$. Then repeat Step 1.

Step 3: Use small random perturbations and search for the arrangement with greater minimum distance. When one is found, return to Step 1.

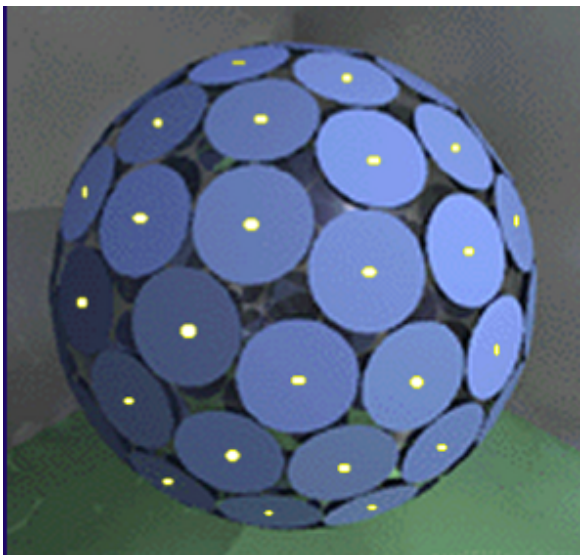


Figure 8: The Sphere Packing Problem [4].

References

[1] http://en.wikipedia.org/wiki/Thomson_problem.

[2] http://en.wikipedia.org/wiki/Spherical_coordinate_system.

[3] Y. Plan and R. Vershynin. One-bit compressed sensing by linear programming. *Communications on Pure and Applied Mathematics*, 66:1275–1297, 2013.

[4] http://www-lp.fmf.uni-lj.si/plestenjak/talks/preddvor.pdf.

[5] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

[6] D. Gleich. https://www.cs.purdue.edu/homes/dgleich/cs520-2014/.