APPLIED OPTIMIZATION METHODS ASSIGNMENT GROUP 4

Authors: u1716865, u1720269, u1715856, u1703501,u1714860
March 19, 2020

1 Question 1

The purpose of this exercise is to solve the Thomson problem. This involves finding the set of positions of n electrons in d-dimensional space that minimises the potential energy of the system. Our aim is to solve this problem numerically using the proposed sequential quadratic programming (SQP) technique. This is a local direction search method that builds on Newton's method for a system with quadratic equality constraints.

1.1 Task 1: Deriving the KKT conditions for SQP

The SQP approach is based on iterative direction changes to the electron positions given by $\mathbf{d}$ (not to be confused with the number of dimensions d) that minimises the following objective function

$$\min_{\mathbf{d}} \nabla f(x^{(k)})^T \mathbf{d} + \frac{1}{2}\mathbf{d}^T \left(H_f(x^{(k)}) + \sum_{i=1}^{m} u_i^k H_{h_i}(x^{(k)}) \right) \mathbf{d}$$

subject to $m = n$ constraints of the form

$$\nabla h_i(x^{(k)})^T \cdot \mathbf{d} + h_i(x^{(k)}) = 0.$$

The H_f is the Hessian matrix of second derivatives of the objective function $f(\mathbf{x})$ and H_{h_i} is the Hessian matrix of the constraint for electron i which is

$$h_i(\mathbf{x}_i) = \sum_{k=1}^{d} x_{ik}^2 - 1 = 0.$$

The Lagrangian for this problem is given by

$$\mathcal{L}(\mathbf{d}, \lambda) = \nabla f(x^{(k)})^T \mathbf{d} + \frac{1}{2}\mathbf{d}^T \left(H_f(x^{(k)}) + \sum_{i=1}^{m} u_i^k H_{h_i}(x^{(k)}) \right) \mathbf{d} + \lambda^T \left(\nabla h(x^{(k)})^T \mathbf{d} + h(x^{(k)}) \right).$$

The KKT conditions are given by taking the derivatives with respect to $\mathbf{d}$ and λ. Deriving with respect to $\mathbf{d}$ gives

$$\nabla f(x^{(k)})^T + \left(H_f(x^{(k)} + \sum_{i=1}^{m} u_i^k H_{h_i}(x^{(k)}) \right) d + \lambda^T \nabla h(x^{(k)}) \quad = \quad 0.$$

Deriving with respect to λ gives back the constraints

$$\nabla h(x^{(k)})^T d + h(x^{(k)}) \quad = \quad 0.$$

The KKT condition and the constraint can be written in matrix form as follows:

$$\begin{bmatrix} Q_k & \nabla h(x^{(k)}) \\ \nabla h(x^{(k)})^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{d}_k \\ \lambda^k \end{bmatrix} = - \begin{bmatrix} \nabla f(x^{(k)} \\ h(x^{(k)}) \end{bmatrix}$$

where

$$Q_k = H_f(x^{(k)}) + \sum_{i=1}^{m} u_i^k H_{hi}(x^{(k)})$$

So both the direction vector for updating the positions of the electrons is given by d_k and the Lagrange multipliers λ^k can be obtained by solving the system of linear equation as below:

$$\begin{bmatrix} d_k \\ \lambda^k \end{bmatrix} = - \begin{bmatrix} Q_k & \nabla h(x^{(k)}) \\ \nabla h(x^{(k)})^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} \nabla f(x^{(k)}) \\ h(x^{(k)}) \end{bmatrix}$$

If we compare the form of this equation to the Quadratic Newton approach in Nocedal and Wright (Section 18.1), we can set $u^{(k+1)} = \lambda^{(k)}$. This gives us an iteration scheme which will minimise $f(\mathbf{x})$ subject to quadratic equality constraints.

1.2 Task 2: Implementing the SQP algorithm code in Matlab

We have implemented the SQP algorithm (see files submitted along with this report). The necessary inputs are:

- n: the number of points.
- d: the dimension of the space occupied by the electrons.
- ϵ: the magnitude of vector d below which we terminate the iteration scheme. This is set to $1e-6$.
- max iterations: the maximum number of iterations (as a fall-back termination condition in case convergence fails).
- seed: the random number seed used to generate the initial electron positions and values of u.

We wrote the following 3 functions, used in the final function `thompsonSolve.m` which solves the whole problem:

- `circle.m`: to draw a circle which is to be used to analyse the solution
- `func.m`: represents the minimisation problem f(x): $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{\|x_i - x_j\|}$
- `grad_f.m`: returns the gradient of $f(x)$, written as $\nabla f(x)$

The steps in the algorithm are as follows:

1. We set $k = 0$ and initialise the algorithm by randomly generating n d-length vectors $\mathbf{x}^{(0)}$. For each electron $i = 1, \cdots, n$, we draw the position in each dimension x_{ik} for $k = 1, \ldots, d-1$ as a random uniform on $[-1, 1]$. For the final element x_{id} we solve for the value that fulfills the constraint $\sum_{k=1}^{d} x_{ik}^2 = 1$. This starts the iterations with a set of position vectors that satisfy all of the constraints. We initialise $u^{(0)}$ as an n-length vector of uniform random draws on interval$[0, 1]$.
2. We calculate $f(\mathbf{x}^{(k)})$ and $\nabla f(x^{(k)})$. We calculate the Hessian matrix $H_f(x^{(k)})$ numerically[1].
3. We calculate $h_i(\mathbf{x}^{(k)})$, and its grad vector $\nabla h(\mathbf{x}^{(k)})$ and the Hessian matrix $H_h(x^{(k)})$ analytically[2].
4. These values are plugged into the system of linear equations derived above

$$\begin{bmatrix} \mathbf{d}^{(k)} \\ \lambda^{(k)} \end{bmatrix} = - \begin{bmatrix} Q_k & \nabla h(x^{(k)}) \\ \nabla h(x^{(k)})^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} \nabla f(x^{(k)}) \\ h(x^{(k)}) \end{bmatrix}$$

   which are then solved for $\mathbf{d}^{(k)}$ and $\lambda^{(k)}$.
5. If $|\mathbf{d}| < \epsilon$ we terminate the algorithm. Otherwise $x^{(k+1)} = x^{(k)} + d_k$, $u^{(k+1)} = \lambda^k$, increment k and return to (2).

1.3 Task 3: Solving Thompson Problem with d = 2 and n = 20

The aim is to find the optimal configuration of 20 electrons on a 2 dimensional circle of unit radius. The results are shown in Figure 1. The black dots represent the initial values of the electrons $\mathbf{x}^{(0)}$. The red dots represent the final positions of the electrons. They are all equally spaced and as far apart from one-another on the circle as possible. This appears to be a correct solution to the Thompson problem. Running the algorithm also gives a minimum potential energy solution of $f(x) = 131.8695$.

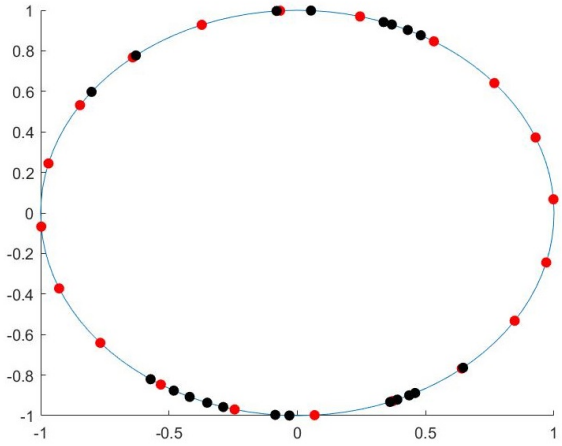

Figure 1: Graphical solution using values $n = 20$, $d = 2$, $\epsilon = 1e^{-6}$.

[1] This is simpler than calculating the analytical derivative of the objective function and more generic as it would allow us to change the objective function $f(\mathbf{x})$ without having to change the Hessian function.

[2] As these equations are quadratic and unlikely to change, it is straightforward to do these calculations analytically.

We repeated this using different values of the random seed in order to change the initial conditions and found differemt solutions with the same minimum potential energy. This reflects the rotational symmetry of this problem that means that there is an infinity of solutions. With regards to convergence, figures 2 and 3, show the values of $f(x)$ and $\|d\|$ increase and decrease significantly over iterations, but $f(x)$ converges to 198.7, and $\|d\|$ converges to near 0. This is due to the many local minima of the energy function $f(x)$ in this problem. The amount of time spent converging depends on the quality of the initial estimate of $x^{(0)}$ and $u^{(0)}$. If these start out close to the optimal solution, the convergence can be a lot faster. By choosing the use a randomised starting points we have imposed a significant challenge to the optimiser. With more time we could possibly develop alternative starting conditions that may permit a faster and more optimal solution.

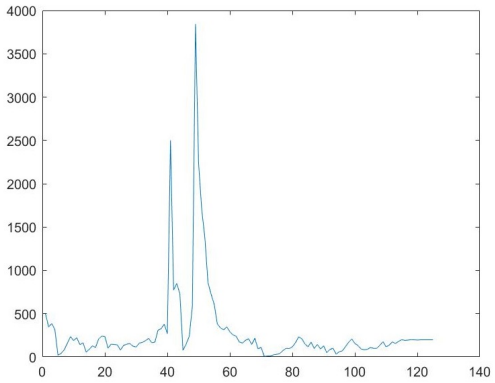

Figure 2: Evolution of $f(\mathbf{x})$ over time using values $n = 20$, $d = 2$ and epsilon = $1e^{-6}$

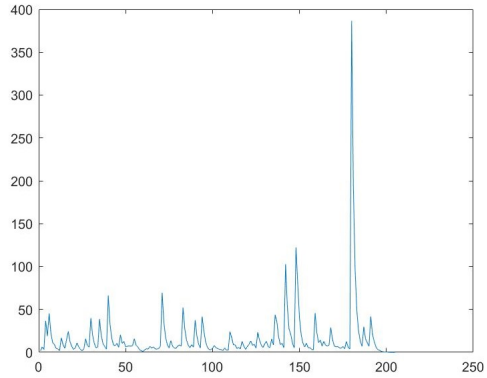

Figure 3: Evolution of $\|d\|$ over time for n = 20, d = 2 and epsilon = $1e^{-6}$

1.4 Task 4: Modifying the problem for a unique global solution

If we want to obtain one global optimal solution for the Thompson problem, we would need to keep one of the points stationary over the whole process. This breaks the aforementioned rotational symmetry. We do this in the algorithm `thompsonSolve_Stationar` where we only compute the direction of change for the first $n-1$ points and keep the direction of point $x_n^{(k)}$ zero, ie $d_n^{(k)} = 0$. Figure 5 shows the end points on a 3D plot.

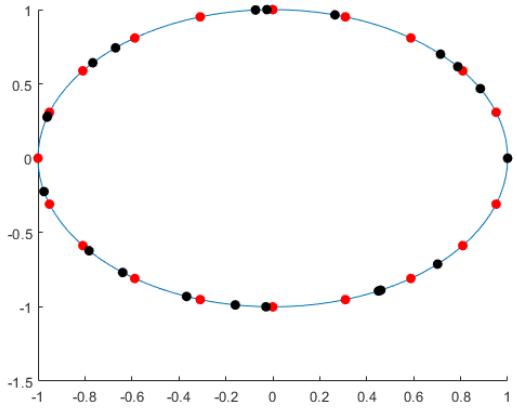

Figure 4: Unique global solution using values $n = 20$, $d = 2$, $\epsilon = 1e^{-6}$, and with the stationary point $x_n^{(k)} = (1, 0)$.

Now we solve the problem with $n = 10$ and $d = 3$.

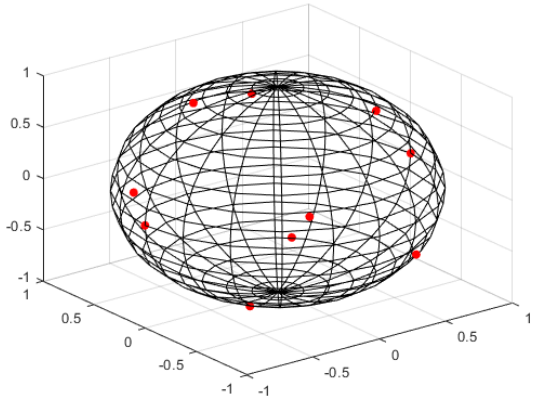

Figure 5: Solution using values $n = 10$, $d = 3$, $\epsilon = 1e^{-6}$, max iteration = 1000

We see in Figure 5 that the points are equally spaced and we have confirmed this by calculating their pairwise Euclidean distances.

2 Question 2

The purpose of this exercise is determine the optimal path for selecting n different products in a warehouse where the same product may be located at multiple locations.

2.1 Task 1: Plotting locations of all products

A set of Matlab files have been provided that reads in the products and their locations and plots these. See Figure 6. The depot is marked by a blue star at $(0, 50)$. The numbers next to the points represent the number of the item at the location.

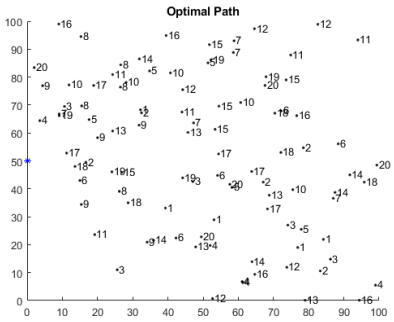

Figure 6: The product locations in the warehouse. Each numbered point is a collection point, each number is an item. The blue star is the depot.

2.2 Task 2: Develop a simple construction heuristic for this problem in Matlab

The heuristic is split into several stages and we will use it to identify the shortest tour from the depot to one of the storage locations of every product and then back to the depot. Our objectives are:

1. Read the data and modify it into matrices containing the x and y coordinates of the storage locations respectively.
2. Initialise the path with the location of the depot.
3. Consider all the storage locations of previously non-collected items and calculate and store their distance from the current location.
4. Find the coordinates of the product location which has the minimum distance from the current location and add this distance to the total distance, and add the coordinates of this storage location to the path. This is the new current location.

5. Return to step 2 unless all items in the order have been collected.

The algorithm is straightforward to implement. It is an example of a travelling salesman problem using the nearest neighbour heuristics. It is simple to implement. This strategy was effective because we essentially followed Prim's algorithm which is a tried and proven method to approach this problem. The solution found from this heuristic is a total distance of 268.0947.

2.3 Task 3: Modify the program from 2.1 such that it displays the found solution graphically

It is straightforward to plot out the returned path by plotting xPosVect and yPosVect that are generated by the algorithm. This has been done.

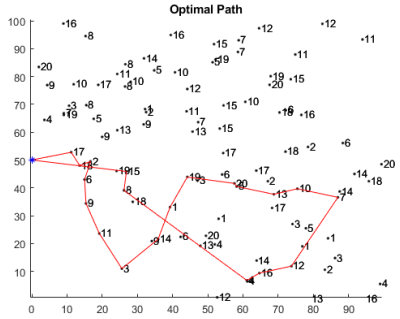

Figure 7: The path is the solution found by the algorithm which is 268.1 units.

2.4 Task 4: Difference between solutions when each product is stored in only one location

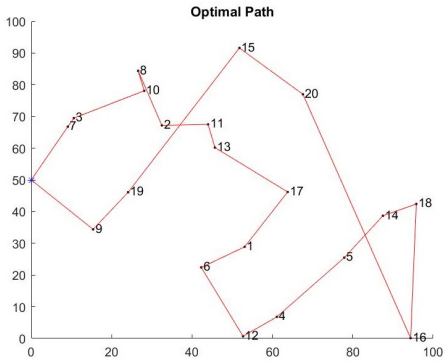

Figure 8: The optimal path when each product is stored in one location. In this case the resulting path length is 461.2 units.

We see that the difference in the two tours is 461.216 - 268.0947 = 193.1213 and therefore there is a significant increase in the estimated distance needed to cover when each product is only stored in one location. This was to be expected because the driver now has less choice of which node to travel to at each iteration to collect all items, and thus will be forced down a path that is less optimal than in the first case.

2.5 Task 5: Develop an Evolutionary Algorithm to solve the problem

The steps in the algorithm are as follows:

1. Read the data and modify it into two matrices of the x and y coordinates of the item locations respectively.
2. Randomise the items to collect at each step making sure that no item is repeated. Randomise the piles to collect the items from by creating a vector of integer values normally distributed between each pile. Repeat this μ times.
3. Evaluate the distances for each of these parent paths and rank them. Create weights based on the ranking.
4. COMBINATION: Select 2 parent paths randomly based on the weighting. Cut out a random section out of the first parent path and complete it with the second parent path to create a child path.
5. MUTATION 1: select two random items in the child path and swap them.

6. MUTATION 2: select a random element of the piles vector of the child paths and reassign it to a random integer between 1 and 5.
7. Evaluate the distance of the child path.
8. Repeat Steps 2 to 6 λ times.
9. Combine the parent and children paths into one matrix and select the μ best solutions as the new parents.
10. Return to step 3 until no child paths are added to the new parent path matrix `noNewCount`times.
11. plot and return the distance of the optimal solution.

A big advantage of Evolutionary algorithms is that it can allow continuous adaptation and perform a simulation-based optimisation using realistic conditions. Drawbacks include the long processing time and no optimality guarantee. The objective function also becomes increasingly complicated to evaluate. It also does not give us our best strategy; it requires further information on outcome of certain events and its behaviour. We try to randomise as much as possible to eliminate bias. When the algorithm was first designed and implemented, we had scenarios where we only had one parent receiving a mutation and no offspring would then get this mutation due to our earlier program only reading the first element of the offspring's values. To solve this we randomised the element picked to ensure there are some offspring with mutant characteristics.

Our model gave a result of 456.9101 which is a good result considering the number of products in each location and the number of items to collect were randomised.

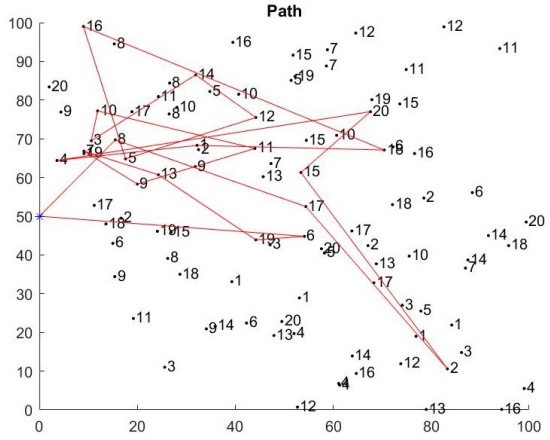

Figure 9: A solution of the Evolutionary Algorithm where the length of the path is 456.9 units.