NUMERICAL OPTIMIZATION PROJECT

CME 304, WINTER QUARTER 2016

1 The Problem Statement

Enclose an ellipse with N rectangles so as to minimize the area between the ellipse and the rectangles.

In this project, we implement a computer program that solves the optimization problem stated above. The problem as stated above in its full generality is extremely hard to solve and hence we will solve a simpler version of the problem, which will be discussed. We will study several aspects of the optimization routine, for e.g how the choice of the descent direction impacts its performance, performance dependence on the choice of the linesearch algorithm, runtime dependence on the number of rectangles N and many more. Finally, we will also study a more general version of the problem, where we try to enclose different super-ellipses with rectangles.

2 Setting Up The Optimization Problem

We begin by defining the optimization problem mathematically and setting up a cartesian coordinate system that allows us to study it easily. Next, we briefly discuss the problem in the general case and why it is difficult to solve, and introduce certain assumptions that lead to an easier problem. It will be seen that this simpler problem can be further reduced to the solution of the optimization problem, where the ellipse is the unit circle.

2.1 Choice of the coordinate system

Let us consider an ellipse with semi-major axis a and semi-minor axis b. For our convenience, we will study the problem in a cartesian coordinate system and choose the coordinate axes to be parallel to the axes of the ellipse, and the origin to be coincident with the center of the ellipse. This is depicted in Figure 1. Hence, with our choice of the coordinate system, the equation of the ellipse is given by:

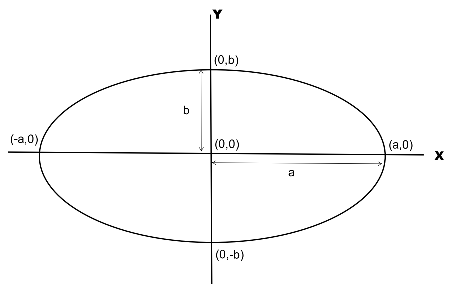
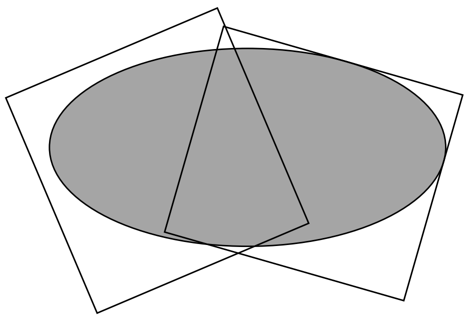


Figure 1: Ellipse and the coordinate system

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1; \quad a \geq b > 0 \tag{1}$$

2.2 The optimization problem in the general case

The optimization problem in the general case involves finding a configuration of N rectangles that completely encloses the ellipse, i.e any point inside or on the ellipse in Figure 1 is contained inside atleast one rectangle, such that the configuration minimizes the area outside the ellipse. The optimization problem posed in this broad general form does not impose any restrictions on the orientation of the rectangles with respect to the ellipse or to each other. It also does not impose any restrictions on whether the rectangles can overlap with each other or not. For example, for $N = 2$, we depict two possible configurations of rectangles that completely enclose the ellipse in Figures 2 (a) and (b). In the former case, the sides of the rectangles are not parallel to the axes of the ellipse or to the sides of each other, while in the latter case the sides of the rectangle are parallel to the axes of the ellipse and to each other.


(a)

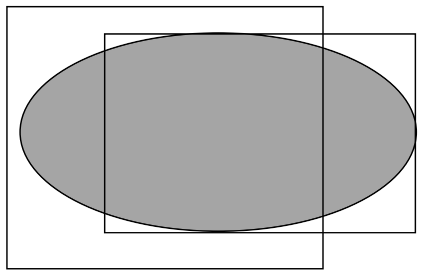

(b)

Figure 2: For $N = 2$, two possible configurations of rectangles that enclose the ellipse.

Trying to set up an optimization problem that can be solved on a computer in the general case is not very easy. Several difficulties arose during this effort and as a result the general case was abandoned. In all likelihood, it appears to be a non-trivial research problem. *However, it is also equally likely that my lack of knowledge of mathematics in this particular field prevented me from being able to formulate the problem.* Nevertheless, it is worth pointing out some of the key observations that were made during this unsuccessful attempt:

1. Each rectangle has 5 degrees of freedom which means that we need 5 independent variables to completely specify each rectangle. These are the X and Y coordinates of the center of each rectangle, the length and breadth of each rectangle and the angle by which one of the sides of each rectangle is rotated relative to the X axis. This means that for N rectangles there are $5N$ independent variables.
2. The first difficulty that arose was in determining a way to calculate the objective function for a given *valid set* of $5N$ independent variables. *Valid set* means that the N rectangles corresponding to the $5N$ independent variables completely enclose the ellipse. Assuming a valid set, one could calculate the area enclosed by all the rectangles first and then subtract the area of the ellipse to get the objective

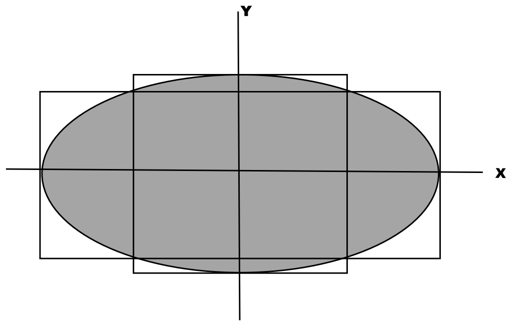
function value. The calculation of the area enclosed by all the rectangles is non-trivial but possible with a suitable algorithm. It involves computing a set of disjoint(non-intersecting) polygons that represent the union of all the rectangles, and then calculating the area of each disjoint polygon, followed by summing the areas.

3. The next difficulty that was encountered was in computing the gradient of the objective function with respect to the $5N$ independent variables. The same difficulty was encountered in the hessian computation. The lack of an analytical formula for the objective function and consequently both the gradient and the hessian meant that one had to rely entirely on finite difference schemes to evaluate these quantities of interest while solving the optimization problem. *There is a possibility that there may be risks associated with this idea as well, as there is a suspicion that the objective function is non-differentiable, but this has not been properly investigated.*

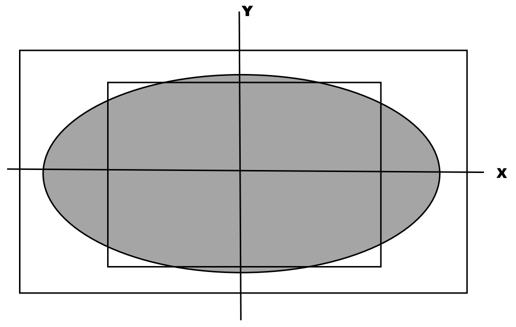
4. However, the most important difficulty was encountered in trying to come up with a set of functional constraints that must be satisfied in order for the ellipse to be completely enclosed by N rectangles, given by a set of $5N$ numbers. *No way was found to overcome this difficulty, but again it may be due to my ignorance in this field.*

Due to these observations, it was decided to restrict ourselves to a subset of possible configurations for the rectangles, that makes the optimization problem particularly amenable to an easy computer implementation. This is what we discuss next.

2.3 Imposing symmetry on the solution



(a)

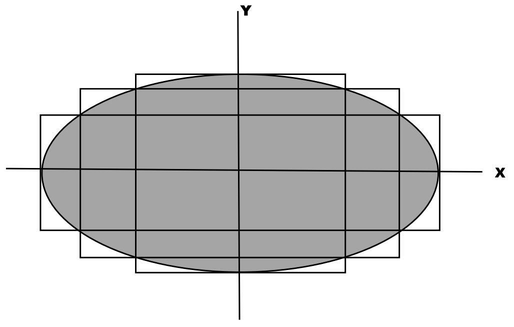


(b)

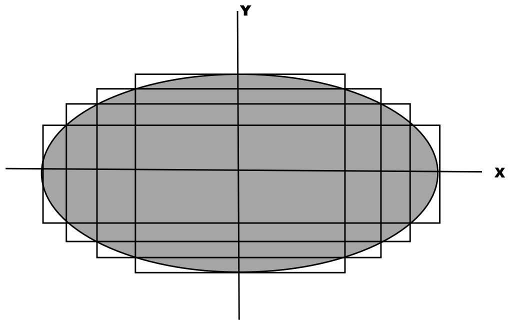
Figure 3: For $N = 2$, two possible symmetric configurations of rectangles that enclose the ellipse.

Instead of trying to solve the optimization problem in the general case, we impose extra conditions on the possible configurations that the rectangles can achieve in terms of their positions and orientations. These conditions are that the sides of each rectangle must be parallel to the coordinate axes and the center of each rectangle must coincide with the origin. Imposing these extra conditions leads to a symmetry requirement that the rectangles must be invariant under inversion about the origin ($x \leftarrow -x,\ \ y \leftarrow -y$), and reflection about the X axis ($x \leftarrow x, \;\; y \leftarrow -y$) and Y axis ($x \leftarrow -x, \;\; y \leftarrow y$). We will call such configurations of rectangles that cover the ellipse as *"symmetric covering configurations (SCC)."* Two such configurations are shown for $N = 2$ in Figure 3. It turns out that although the configuration in Figure 3(b) is a SCC, it cannot be one that minimizes the area outside the ellipse. For instance, the outermost rectangle can always be shrunk so that its sides are tangential to the ellipse, thereby decreasing the area outside the ellipse.

In general, it can be proved that a SCC that minimizes the area outside the ellipse must be of the form shown in Figure 3(a). We skip the proof here, but present a geometrical argument for why this must be true for $N = 2$. Let us start by noting that if there exists a SCC where any one or both rectangles completely enclose the ellipse individually, then each one of the rectangles can be shrunk so that they are tangential to the four apexes of the ellipse, thereby decreasing the area outside the ellipse. The configuration thus obtained has an area outside the ellipse that is larger than any configuration of the form in Figure 3(a). Next note that it must also be true that for the minimum configuration, atleast one rectangle must be tangential to the apexes joined by the minor axis of the ellipse. But this rectangle cannot be tangential to the other two apexes as well, because it will then fully enclose the ellipse and there exists configurations of the form in Figure 3(a) with a smaller area. This rectangle then must intersect the ellipse in the minimum configuration at four other additional points. It then follows that the other rectangle must be tangential to the remaining two apexes and in addition must intersect the ellipse at the same four non-apex points, leading to the configuration of Figure 3(a). This argument can be easily generalized to cases for $N \geq 2$, but is more involved. In Figure 4, we show possible SCC for $N = 3$ and 4.



(a)



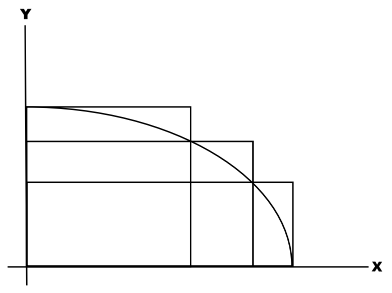
(b)

Figure 4: (a) A SCC of rectangles for $N = 3$. (b) A SCC of rectangles for $N = 4$.

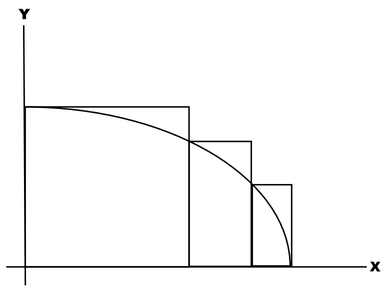
In what follows, we will be only concerned with finding solutions to the optimization problem where the configuration of the rectangles is a SCC of the form descibed in Figure 4. We will see that this leads to an enormous simplification of the optimization problem as we will be able to write an analytical expression for the objective function. This in turn will allow us to find easy analytical expressions for the gradient and hessian of the objective function. But first, we reduce the problem to that on the first quadrant by exploiting the symmetry of the solution.

2.4 Reducing the problem to the first quadrant

We can exploit the symmetry of the solution to reduce the optimization problem to one on only the first quadrant. To do that, let us note that both the SCC and the ellipse itself are invariant under inversion about the origin and reflections about the X and Y axes. Thus, the area outside the ellipse is also the same in each of the four quadrants. We can then reduce the optimization problem to one of minimizing the area outside the ellipse in only the first quadrant as shown in Figure 5(a). The rectangles in this figure are the one-fourth part of the full rectangles over all the quadrants, and they have the property that the lower left vertex coincides with the origin, and two sides of each rectangle are coincident with the coordinate axes. The new problem then is to find a configuration of rectangles of the form in Figure 5(a) such that the area outside the ellipse in the first quadrant is minimized. It is easily seen that the problem is equivalent to minimizing the area outside the ellipse for a simpler arrangement of rectangles as shown in Figure 5(b). In Figure 5(b), two adjacent rectangles have coincident edges. Solving this problem yields a solution to the problem on all the four quadrants through a symmetric geometrical extension, through the crucial intermediate step of constructing the form in Figure 5(a) from the one in Figure 5(b).



(a)



(b)

Figure 5: (a) This figure represents the picture in the first quadrant for a SCC of rectangles for $N = 3$. Note that now all the rectangles have the lower left vertex coinciding with the origin. (b) This figure represents the configuration of the rectangles leading to an equivalent optimization problem as in (a).

2.5 The objective function

We are finally in a position to write down the objective function corresponding to the minimization problem of the area outside the ellipse in the first quadrant. Consider again the ellipse given by equation (1) and let the number of rectangles be N. We will look for a configuration of rectangles in the first quadrant of the form similar to Figure 5(b). Let the i^{th} rectangle have width ah_i. It follows then that the first rectangle has a height b, the second rectangle has a height $b(1 - h_1^2)^{1/2}$ and so on. The combined area for all the N rectangles is then given by:

$$A = ab \sum_{i=1}^{N} h_i \left[1 - \left(\sum_{j=1}^{i-1} h_j \right)^2 \right]^{1/2} \tag{2}$$

Now, the area of the ellipse in the first quadrant is $\frac{\pi ab}{4}$. The area outside the ellipse is then the difference between the combined area of the rectangles and the area of the ellipse, and hence the objective function is given by:

$$A = ab \sum_{i=1}^{N} h_i \left[1 - \left(\sum_{j=1}^{i-1} h_j \right)^2 \right]^{1/2} - \frac{\pi ab}{4} \tag{3}$$

We now have a minimization problem in the N variables $h_1, h_2, ..., h_N$. However, we have certain constraints. First, since the rectangles cover the ellipse it must be true that $\sum_{i=1}^{N} h_i = 1$. We would also like to enforce that the solution always involves positive h_i, i.e $h_i > 0$. This is essential as otherwise the objective function becomes unbounded below and complex for $h_i \to -\infty$. Also, the objective function defined in equation (3) is infinitely differentiable in the domain specified by these constraints. As an example, consider Figure 6 which shows the arrangement of the rectangles for $N = 4$. Each rectangle has an edge coincident with an edge of the next rectangle, and each adjacent pair of rectangles also intersect the ellipse at the same point. In fact, the top left vertex of each rectangle lies on the ellipse.

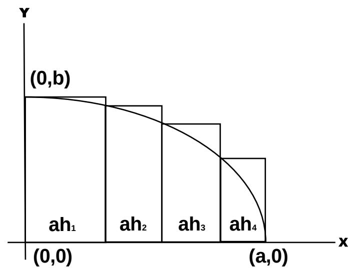


Figure 6: This figure shows the configuration of the rectangles for $N = 4$. Each adjacent pair of rectangles have coincident edges and intersect the ellipse at the same point.

2.6 Reducing the problem to the unit circle

A really interesting consequence of the above formulation is that the optimization problem can simply be solved for the case $a = b = 1$. To see this, consider equation (3). The important aspect in this equation is that the semi-major and semi-minor axes lengths appear as a product in the objective function and can be factored out. Hence, minimizing A is equivalent to minimizing $A/(ab)$. But $A/(ab)$ is nothing but the objective function A for the case $a = b = 1$. With this insight, we can now define the objective function to minimize as:

$$\begin{aligned} \min \;\; F &= \sum_{i=1}^{N} h_i \left[1 - \left(\sum_{j=1}^{i-1} h_j \right)^2 \right]^{1/2} - \frac{\pi}{4} \\ \text{s.t} \;\; &\sum_{i=1}^{N} h_i = 1, \;\; h_i > 0 \;\; \forall i = 1, \ldots, N \end{aligned} \tag{4}$$

If we can solve the above problem for the unit circle in (4), we can get a solution for the problem for the ellipse in (3) for any arbitrary a and b by simply scaling the X and Y axes appropriately ($x \leftarrow ax, \; y \leftarrow by$). That in turn can be used to construct a solution for the whole ellipse in all the four quadrants. This is great progress !

2.7 The gradient of the objective function

To calculate the gradient of the objective function, we differentiate F with respect to h_k for $k = 1, \ldots, N$. The result is given below:

$$\begin{aligned}
\frac{\partial F}{\partial h_1} &= 1 - \sum_{i=2}^{N} h_i \left(\sum_{j=1}^{i-1} h_j \right) \left[1 - \left(\sum_{j=1}^{i-1} h_j \right)^2 \right]^{-1/2} ; \\
\frac{\partial F}{\partial h_k} &= \left[1 - \left(\sum_{j=1}^{k-1} h_j \right)^2 \right]^{1/2} - \sum_{i=k+1}^{N} h_i \left(\sum_{j=1}^{i-1} h_j \right) \left[1 - \left(\sum_{j=1}^{i-1} h_j \right)^2 \right]^{-1/2} ; \quad k = 2, \ldots, N-1 \\
\frac{\partial F}{\partial h_N} &= \left[1 - \left(\sum_{j=1}^{N-1} h_j \right)^2 \right]^{1/2}
\end{aligned} \tag{5}$$

2.8 The hessian of the objective function

The hessian of the objective function can similarly be calculated, but the calculation is tedious. The results are given below:

$$\begin{aligned}
\frac{\partial^2 F}{\partial h_m \partial h_k} &= \begin{cases} -\left(\sum_{j=1}^{k-1} h_j \right) \left[1 - \left(\sum_{j=1}^{k-1} h_j \right)^2 \right]^{-1/2} & - \; \sum_{i=k+1}^{N} h_i \left[1 - \left(\sum_{j=1}^{i-1} h_j \right)^2 \right]^{-3/2}, \\ & \text{for } k = 2, \ldots, N-1; \; m < k \\ -\left(\sum_{j=1}^{k-1} h_j \right) \left[1 - \left(\sum_{j=1}^{k-1} h_j \right)^2 \right]^{-1/2}, & \text{for } k = N; \; m < k \end{cases} \\
\frac{\partial^2 F}{\partial h_k^2} &= \begin{cases} -\sum_{i=k+1}^{N} h_i \left[1 - \left(\sum_{j=1}^{i-1} h_j \right)^2 \right]^{-3/2}, & \text{for } \; k = 1, \ldots, N-1 \\ 0, & \text{for } \; k = N \end{cases} \\
\frac{\partial^2 F}{\partial h_m \partial h_k} &= \begin{cases} -\left(\sum_{j=1}^{m-1} h_j \right) \left[1 - \left(\sum_{j=1}^{m-1} h_j \right)^2 \right]^{-1/2} & - \; \sum_{i=m+1}^{N} h_i \left[1 - \left(\sum_{j=1}^{i-1} h_j \right)^2 \right]^{-3/2}, \\ & \text{for } k = 2, \ldots, N-1; \; m > k \\ -\sum_{i=m+1}^{N} h_i \left[1 - \left(\sum_{j=1}^{i-1} h_j \right)^2 \right]^{-3/2}, & \text{for } k = 1; \; m > k \end{cases}
\end{aligned} \tag{6}$$

The hessian is symmetric and it can be directly verified that $\frac{\partial^2 F}{\partial h_m \partial h_k} = \frac{\partial^2 F}{\partial h_k \partial h_m}$ from the relationships in equation (6). Surprisingly, it turns out that the hessian calculation is not very expensive for the objective function. One needs to calculate the diagonal entries of the hessian and the first sub-diagonal only. All the other non-diagonal elements of the hessian belonging to the same row are equal below the diagonal. This can again be verified from equation (6). To get the entries of the hessian matrix above the diagonal, one can just use symmetry of the hessian to get them from the computed entries below the diagonal.

3 Numerical Implementation

In this section, we present some of the key aspects of the numerical implementation for solving the optimization problem in (4). We will briefly discuss the following aspects:

1. Efficient computation of *the objective function*, *the gradient* and *the hessian*.
2. Null space active set method to enforce the constraints.
3. Determining the direction of descent based on *steepest descent* or *modified Newton*.
4. The linesearch algorithm based on *Goldstein* or *Strong-Wolfe* criteria.
5. The complete high level algorithm to find the solution.

3.1 Efficient computation of the objective function

If we are given an input vector $\mathbf{h} := \{h_1, h_2, \ldots, h_N\}$ that satisfies the constraints in (4), the algorithm to compute the objective function efficiently is given in Algorithm 1. The computational complexity of the algorithm is $O(N)$, counting all the additions, multiplications and exponentiation.

3.2 Efficient computation of the gradient

If we are given an input vector $\mathbf{h} := \{h_1, h_2, \ldots, h_N\}$ that satisfies the constraints in (4), the algorithm to compute the gradient $\mathbf{g} := \{g_1, g_2, \ldots, g_N\}$ where $g_i = \partial F / \partial h_i$, is given in Algorithm 2. The computational complexity of the gradient computation is also $O(N)$, counting all the additions, multiplications and exponentiation. The space complexity of the algorithm is $O(N)$.

3.3 Efficient computation of the hessian

If we are given an input vector $\mathbf{h} := \{h_1, h_2, \ldots, h_N\}$ that satisfies the constraints in (4), the algorithm to compute the hessian $\mathbf{H} := \{H_{ij}, \ \forall\ i, j \in [1, \ldots, N]\}$ where $H_{ij} = \partial^2 F / \partial h_i \partial h_j$, is given in Algorithm 2. The computational complexity of the hessian computation is again $O(N)$, counting all the additions, multiplications and exponentiation. As mentioned previously, it is a nice surprise that the Hessian computation is very cheap for the problem formulation in (4). In fact, it is no more expensive than the gradient computation. However, the space complexity for the hessian is $O(N^2)$.

Algorithm 1 Computing The Objective Function

```
1: procedure OBJECTIVE FUNCTION(h)
2:     $F = h_1$
3:     $s = 0$
4:     for $i = 2, ..., N$ do
5:         $s = s + h_{i-1}$
6:         $F = F + h_i(1 - s^2)^{1/2}$
7:     end for
8:     $F = F - \frac{\pi}{4}$
9: end procedure
```

Algorithm 2 Computing The Gradient

```
procedure GRADIENT($\mathbf{h}$)
    $\mathbf{s} = 0$
    $s_1 = h_1$
    for $i = 2, ..., N$ do
        $s_i = s_{i-1} + h_i$
    end for
    $\mathbf{g} = 0$
    for $i = N, ..., 2$ do
        $g_i = (1 - s_{i-1}^2)^{1/2}$
    end for
    $g_1 = 1$
    $f = 0$
    for $i = N - 1, ..., 1$ do
        $f = f + h_{i+1} s_i (1 - s_i^2)^{-1/2}$
        $g_i = g_i - f$
    end for
end procedure
```

Algorithm 3 Computing The Hessian

```
procedure HESSIAN($\mathbf{h}$)
    $\mathbf{s} = 0$
    $s_1 = h_1$
    for $i = 2, ..., N$ do
        $s_i = s_{i-1} + h_i$
    end for
    $\mathbf{H} = 0$
    for $i = N - 1, ..., 1$ do
        $H_{ii} = H_{i+1\ i+1} - h_{i+1}(1 - s_i^2)^{-3/2}$
    end for
    for $i = 2, ..., N$ do
        $f = H_{ii} - s_{i-1}(1 - s_{i-1}^2)^{-1/2}$
        for $j = 1, ..., i - 1$ do
            $H_{ij} = f$
            $H_{ji} = f$
        end for
    end for
end procedure
```

3.4 Strategy to enforce the constraints

We now briefly discuss how to enforce the constraints appearing in the optimization problem in (4). Let us first note that there are two classes of constraints: equality constraints and inequality constraints. They are stated again below.

$$\begin{aligned} &\sum_{i=1}^{N} h_i = 1 && \text{(Equality Constraint)} \\ &h_i > 0 \ \ \forall i \in \{1, \ldots, N\} && \text{(Inequality Constraints)} \end{aligned} \tag{7}$$

At any feasible point, the constraints in (7) must be true. The strategy that we adopt to solve the problem is to always be feasible with respect to the strict equality constraint. So initially, we will start from a point that is feasible with respect to all the constraints and then search for the minimizer in the null space of only the equality constraint.

The strategy to maintain feasibility with respect to the inequality constraints is different. The key observation that motivates this approach is that *the solution to the optimization problem in* (4) *cannot be such that any of the inequality constraints are satisfied exactly*. This can be proved easily with a geometric argument. Suppose for N rectangles the solution to the optimization problem involves some $h_m = 0$ for $m \in \{1, \dots, N\}$. But this cannot be true because one can split any of the other rectangles with non-zero width into two and decrease the area outside the ellipse, thereby leading to a contradiction. Hence, we cannot have $h_m = 0$ at the minimizer. Applying this argument recursively, one can prove that there can be no i such that $h_i = 0$ at the minimizer. Therefore, at the minimizer all the inequality constraints must be inactive. Also note that *the equality and inequality constraints in equation* (7) *together specify a bounded convex domain.*

These observations suggest that as long as we stay feasible with respect to the hyperplane $\sum_{i=1}^{N} h_i = 1$, and in addition remain inactive with respect to the inequality constraints we should be able to converge to a minimizer inside the bounded convex domain. This motivates the following strategy to remain feasible with respect to the inequality constraints: starting from a feasible point at the current iterate we will determine a search direction, and then inside the linesearch routine we will choose the maximum step length so that none of the inequality constraints are hit. This will lead to a feasible point inactive with respect to the inequality constraints, to repeat the same process for the next iteration.

With this strategy our working set will always consist of only the row vector $[1\ 1 \dots 1\ 1]$. Therefore, using the notation in Walter's notes, we have:

$$A_{1\times N} = \begin{bmatrix} 1 & 1 & . & . & . & 1 & 1 \end{bmatrix} \tag{8}$$

For such an A, the orthogonal matrix Z that spans the null space of A is obtained by performing a QR decomposition of the matrix M in equation (9).

$$M_{N\times(N-1)} = \begin{bmatrix} 1 & . & . & . & . & . & . \\ -1 & 1 & . & . & . & . & . \\ . & -1 & 1 & . & . & . & . \\ . & . & -1 & . & 1 & . & . \\ . & . & . & . & -1 & 1 & . \\ . & . & . & . & . & -1 & 1 \\ . & . & . & . & . & . & -1 \end{bmatrix} \tag{9}$$

It is easy to check that M spans the null space of A, i.e $AM = 0$, and is full column rank. The matrix Z is then given by $ZR = M$, where R is a non-singular upper-triangular matrix. We then have $AZ = 0$ and $Z^T Z = I$. This QR factorization only needs to be performed once and does not need to be changed from one iteration to the next for a fixed number of rectangles N. The initial feasible point with respect to both the equality and inequality constraints can be taken for example to be $\mathbf{h^{(0)}} = [1/N \ 1/N \ \ldots 1/N \ 1/N]^T$. Once Z is calculated, one can see that any search direction of the form $\mathbf{p} = Zp_z$ will ensure that the successive iterates satisfy the equality constraint exactly. Finally, the reduced gradient and the reduced hessian can be calculated as in equation (10). Calculating these quantities are $O(N^2)$ and $O(N^3)$ operations respectively, in terms of computational complexity.

$$\begin{aligned} \text{Reduced Gradient} &= \hat{\mathbf{g}} = Z^T\mathbf{g} \\ \text{Reduced Hessian} &= \hat{H} = Z^T H Z \end{aligned} \tag{10}$$

3.5 Determining the direction of descent

We will compare the performance of the optimization program for two different cases of descent directions based on *1) steepest descent* and *2) modified Newton*. The numerical details of their implementations are briefly discussed next.

1. *Steepest descent:* The direction of steepest descent is computed using the relation $\mathbf{p} = -Z\hat{\mathbf{g}} = -Z(Z^T\mathbf{g})$. Given $\hat{\mathbf{g}}$ and Z, this is an $O(N^2)$ operation. This needs to be done once per iteration in the optimization program.

2. *Modified Newton:* The modified Newton approach is just an improved version of the Newton method for determining the search direction that safeguards against the indefiniteness of the Hessian. In our implementation, once the reduced hessian $\hat{H}$ has been calculated, to get the modified newton update we first perform a modified cholesky decomposition of $\hat{H}$. Following the notation in Walter's notes, we choose the parameter $\epsilon = 10^{-6}$, and the parameter $\beta = \max_i(|\hat{H}_{ii}|^{1/2})$ for $i = 1, \ldots, N$. The modified cholesky decomposition computes a non-singular upper-triangular matrix $\hat{R}$ that satisfies equation (11). The algorithm used for the decomposition is the same as in Walter's notes, and so we do not repeat it here.

$$\hat{H} + E = \hat{R}^T\hat{R} \tag{11}$$

Once we have $\hat{R}$, we can compute the search direction $\mathbf{p}$ as follows in equation (12). The process involves solving two triangular systems of linear equations where the first one is solved using forward substitution and the second one is solved using backward substitution.

$$\begin{aligned} &\text{Forward solve} &&: \hat{R}^T\mathbf{u} = -\hat{\mathbf{g}} = -Z^T\mathbf{g} \\ &\text{Backward solve} &&: \hat{R}\mathbf{v} = \mathbf{u} \\ &\text{Compute descent direction} &&: \mathbf{p} = Z\mathbf{v} \end{aligned} \tag{12}$$

Given $\hat{\mathbf{g}}$, $\hat{H}$ and Z, the modified cholesky algorithm can be used to compute the descent direction in $O(N^3)$ computational complexity. Even though this is costlier as compared to steepest descent, it will be seen that it more than justifies itself due to the property of quadratic convergence of the Newton method, close to the true solution.

3.6 The linesearch algorithm

We implemented and tested the linesearch algorithm using both the Goldstein and Strong-Wolfe conditions. The most critical aspect in implementing them is to determine a maximum step length parameter α_{max} that prevents too large of an update. This is important in the context of this particular problem as choosing α_{max} too large can make our solution become violated with respect to the inequality constraints, as has been mentioned previously. The strategy that we adopt here is to first calculate the parameter α_{max} and then pass it as a parameter to either of the linesearch routines. We present how to determine α_{max} in Algorithm 4, given an initial feasible point $\mathbf{h}$ and a search direction $\mathbf{p}$. In the last step of Algorithm 4, we set $\alpha_{max} = 0.99\alpha_{max}$. This sets the maximum step length to be slightly smaller than what would cause one to hit the nearest inequality constraint if one moved along the direction $\mathbf{p}$ starting from $\mathbf{h}$. Algorithm 4 determines α_{max} in $O(N)$ computational complexity, which needs to be performed once every iteration.

Algorithm 4 Computing α_{max}

```
1: procedure DETERMINE ALPHA MAX($\mathbf{h}, \mathbf{p}$)
2:     $\alpha_{max} = \infty$
3:     for $i = 1, \dots, N$ do
4:         if $p_i < 0$ then
5:             $\alpha = -h_i/p_i$
6:             if $\alpha < \alpha_{max}$ then
7:                 $\alpha_{max} = \alpha$
8:             end if
9:         end if
10:    end for
11:    $\alpha_{max} = 0.99\alpha_{max}$
12: end procedure
```

Let us next briefly discuss the choice of parameters for the linesearch algorithms and some of the main implementation details.

1. *Goldstein linesearch:* For a search direction $\mathbf{p}$, the current iterate $\mathbf{h}$ and a gradient $\mathbf{g}$ at the current iterate, the Goldstein linesearch algorithm proceeds by finding an initial interval $\mathbf{D} = [\alpha_1 \ \alpha_2] \subset [0 \ \alpha_{max}]$ such that equation (13) is true, for some choice of μ_1 and μ_2 such that $0 < \mu_1 \leq \mu_2 < 1$.

$$F(\mathbf{h} + \alpha_1\mathbf{p}) < F(\mathbf{h}) + \mu_2\alpha_1\mathbf{g}^{\mathbf{T}}\mathbf{p}\ ,\ F(\mathbf{h} + \alpha_2\mathbf{p}) > F(\mathbf{h}) + \mu_1\alpha_2\mathbf{g}^{\mathbf{T}}\mathbf{p} \tag{13}$$

   Once such an interval is determined, we use bisection to reduce the interval iteratively till we find a point satisfying the Goldstein conditions in the interval $\mathbf{D}$. The Goldstein condition is satisfied when equation (14) holds. Also note that it may be the case that while searching for an interval satisfying equation (13), we might find a point that satisfies equation (14) before an interval is found. In that case we have already found a valid α satisfying Goldstein conditions and nothing further has to be done.

$$F(\mathbf{h}) + \mu_2\alpha\mathbf{g}^{\mathbf{T}}\mathbf{p} \leq F(\mathbf{h} + \alpha\mathbf{p}) \leq F(\mathbf{h}) + \mu_1\alpha\mathbf{g}^{\mathbf{T}}\mathbf{p} \tag{14}$$

In our implementation, we choose $\mu_1 = 0.4,\ \mu_2 = 0.6$ and $\alpha_1 = 0$. Also there is a special case that needs to be handled separately which occurs when no initial interval satisfying equation (13) can be found. This may happen because depending on the value of α_{max}, it might be the case that there exists no $\alpha, \alpha_2 \in [0\ \alpha_{max}]$ such that equation (13) or (14) holds. It may also be due to the fact that such a point exists, but we have not checked all possible points α_2 in the interval $[0\ \alpha_{max}]$ that satisfies equation (13). The later possibility arises because we do not want to spend too much computational effort in doing the linesearch at every iteration, and in any case there are infinite such points to check - so we will never be able to check all of them. The solution that we chose is to uniformly check $2^6 = 64$ points in the interval $[0\ \alpha_{max}]$ till either equation (13) or (14) is satisfied. However, if no such points are found, then we choose the value of α to be the one that achieves the minimum value of the objective function among all the points checked.

2. *Strong-Wolfe linesearch:* The Strong-Wolfe linesearch proceeds in a manner similar to Goldstein linesearch. For a search direction $\mathbf{p}$, the current iterate $\mathbf{h}$ and a gradient $\mathbf{g}$ at the current iterate, the Strong-Wolfe linesearch algorithm proceeds by finding an initial interval $\mathbf{D} = [0\ \alpha_2] \subset [0\ \alpha_{max}]$ such that either of the conditions in equation (15) is true, for some choice of μ and η such that $0 < \mu \leq \eta < 1$, and $0 < \mu < 1/2$.

$$F(\mathbf{h} + \alpha_2 \mathbf{p}) > F(\mathbf{h}) + \mu \alpha_2 \mathbf{g}^{\mathbf{T}} \mathbf{p} \ \text{ or } \ \nabla F(\mathbf{h} + \alpha_2 \mathbf{p})^T \mathbf{p} > -\eta \alpha_2 \mathbf{g}^{\mathbf{T}} \mathbf{p} \tag{15}$$

Once such an interval is found, we use bisection to reduce the interval iteratively till a point α in $\mathbf{D}$ is found that satisfies the Strong-Wolfe conditions given in equation (16). Also note that as in the case of Goldstein linesearch, it may happen that a point α satisfying equation (16) may be found before an interval satisfying equation (15) is found. In that case, we have found a point satisfying the Strong-Wolfe conditions.

$$\begin{aligned} &F(\mathbf{h} + \alpha \mathbf{p}) \leq F(\mathbf{h}) + \mu \alpha \mathbf{g}^{\mathbf{T}} \mathbf{p} \\ &\eta \alpha \mathbf{g}^{\mathbf{T}} \mathbf{p} \leq \nabla F(\mathbf{h} + \alpha \mathbf{p})^T \mathbf{p} \leq -\eta \alpha \mathbf{g}^{\mathbf{T}} \mathbf{p} \end{aligned} \tag{16}$$

For Strong-Wolfe linesearch, we choose $\mu = 0.3$ and $\eta = 0.5$. We check a maximum of $2^6 = 64$ points in the interval $[0\ \alpha_{max}]$ in the step to find an interval satisfying equation (15), and in exact analogy to Goldstein linesearch if no such interval is found, we set α to be the value that achieves the minimum of the objective function among all the points checked.

3.7 The complete algorithm (high level pseudocode)

We can finally present the high-level algorithm to solve the optimization problem. We have developed the relevant components over the previous sections and refer to them again in the full algorithm presented next, without elaborating each component. It should be noted that although the algorithm structure presented in Algorithm 5 remains the same, the execution of each component may be different. For example, we have linesearch based on Goldstein or Strong-Wolfe conditions, determination of descent direction based on steepest descent or modified-Newton etc.

Algorithm 5 The Full Optimization Algorithm

```
1: procedure OPTIMIZE(N)
2:     Determine initial feasible point $\mathbf{h^{(0)}}$
3:     Construct $M$
4:     Perform QR factorization: $ZR = M$
5:     Calculate initial reduced gradient and reduced hessian: $\hat{\mathbf{g}}^{(0)}$, $\hat{H}^{(0)}$
6:     $i = 0$
7:     while $||\hat{\mathbf{g}}^{(i)}||_2/N > 10^{-10}$ or smallest pivot in modified cholesky of $\hat{H}^{(i)} < 0$ do
8:         Determine direction of descent $\mathbf{p}^{(i)}$ (Steepest Descent/Modified Newton)
9:         Calculate $\alpha^{(i)}_{max}$
10:        Perform linesearch to get $\alpha^{(i)}$ (Goldstein/Strong-Wolfe)
11:        $\mathbf{h}^{(i+1)} = \mathbf{h}^{(i)} + \alpha^{(i)}\mathbf{p}^{(i)}$
12:        $i = i + 1$
13:        Calculate $\hat{\mathbf{g}}^{(i)}$, $\hat{H}^{(i)}$
14:        Perform modified Cholesky of $\hat{H}^{(i)}$ (if not calculated before, i.e if using steepest descent)
15:    end while
16: end procedure
```

Algorithm 5 takes as input the number of rectangles N. The program terminates upon satisfaction of the first and second order optimality conditions. To account for finite numerical accuracy achievable with a computer, we put a small tolerance (10^{-10}) on the value of the reduced gradient scaled by N at the optimal point. Similarly, the positive definiteness of the reduced Hessian is checked by performing a modified cholesky decomposition on it, and checking if the smallest pivot is negative. At the termination of the program, we get the vector $\mathbf{h}$ which is our solution to the optimization problem. $F(\mathbf{h})$ gives us the value of the objective function, i.e the minimum area outside the unit circle for N rectangles.

4 Results

In this section, we present the results obtained by the optimization program. All results will be presented on the unit circle, although at the end we will show an example of how the result looks for the case of an ellipse. As we shall see from the results, solving the problem on the unit circle reveals some very interesting properties of the solution. We begin by studying some of these properties generated using the optimization program outlined in Algorithm 5. The Strong-Wolfe linesearch and modified Newton search direction were used inside the algorithm for all the results discussed in the sections 4.1 - 4.2, while the initial feasible point was taken as $\mathbf{h^{(0)}} = [1/N \; 1/N \; \dots 1/N \; 1/N]^T$ for solving each problem for different values of N.

4.1 Properties of the solution for different N

We plot how the solutions look for different values of N in Figure 7. As can be seen, the solution behaves as expected. Every time we add a new rectangle, it is intuitively clear that it is possible to decrease the area outside the circle. This can be easily seen through a geometrical argument. Suppose we have found the optimal solution F^*_N for

N rectangles, and we want to find the solution for $N+1$ rectangles. One can start by dividing any rectangle at the optimal solution for N into two rectangles to yield a feasible point for the case $N+1$ (as an example, consider the diagrams in Figure 7 for $N=1$ and $N=2$ to see how such a division could be done). Let the objective function value in this configuration be F'_{N+1}. If at the optimal solution for $N+1$ rectangles, the value of the objective function is F^*_{N+1}, then it must be true that $F^*_{N+1} \leq F'_{N+1} < F^*_N$. Therefore as N increases, the area outside the unit circle decreases *monotonically*. In the limit as $N \to \infty$, this area converges to 0.

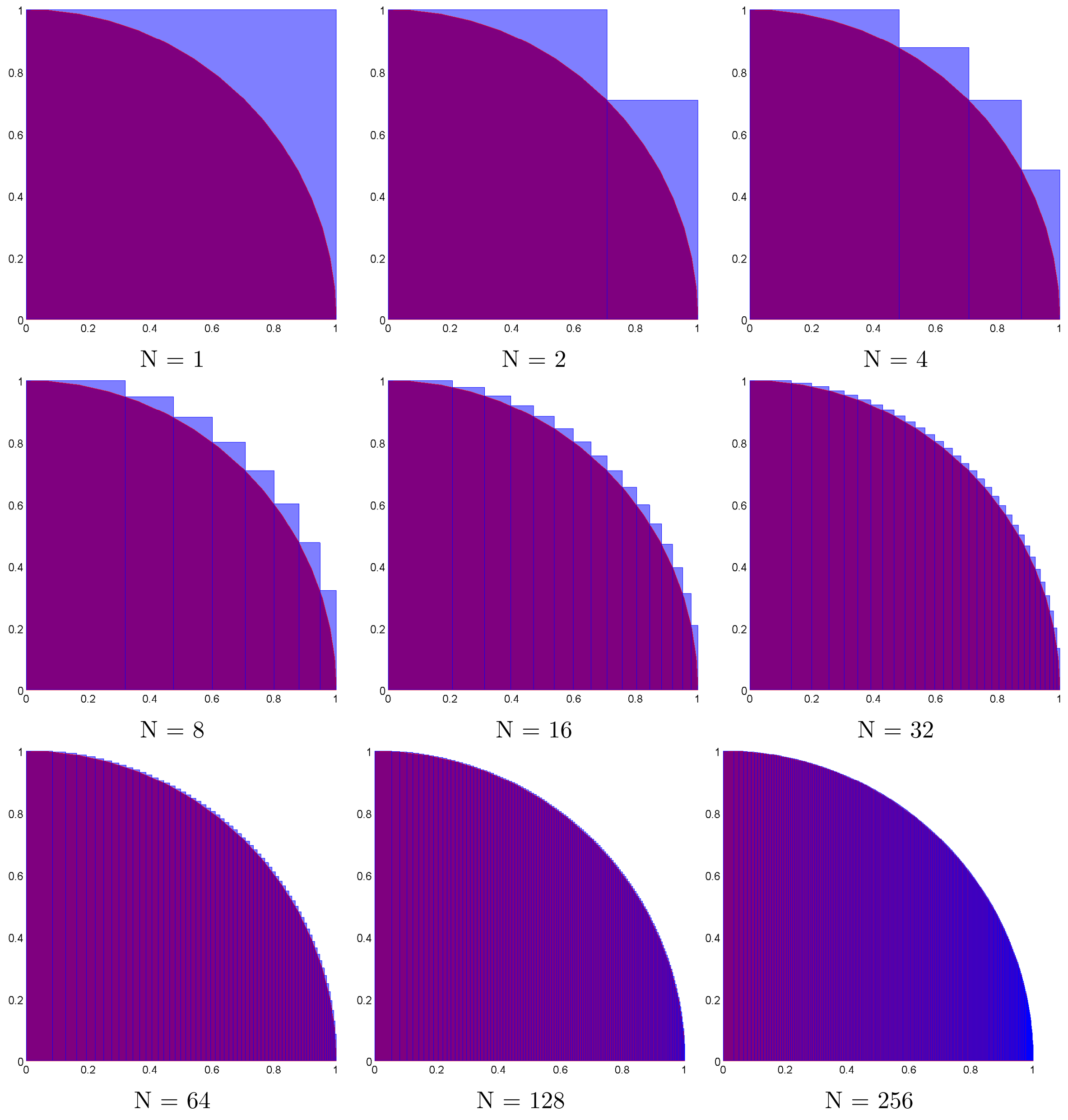


Figure 7: The configuration of the rectangles at the optimal solution point for different values of N.

The minimum area outside the unit circle (F_N^*) determined by solving the optimization problem numerically, as a function of the number of rectangles N is plotted in Figure 8. To be precise, we have plotted F_N^* on a logarithmic axis versus $\log_2 N$. The interesting result here is that the plot seems to be linear for large N.

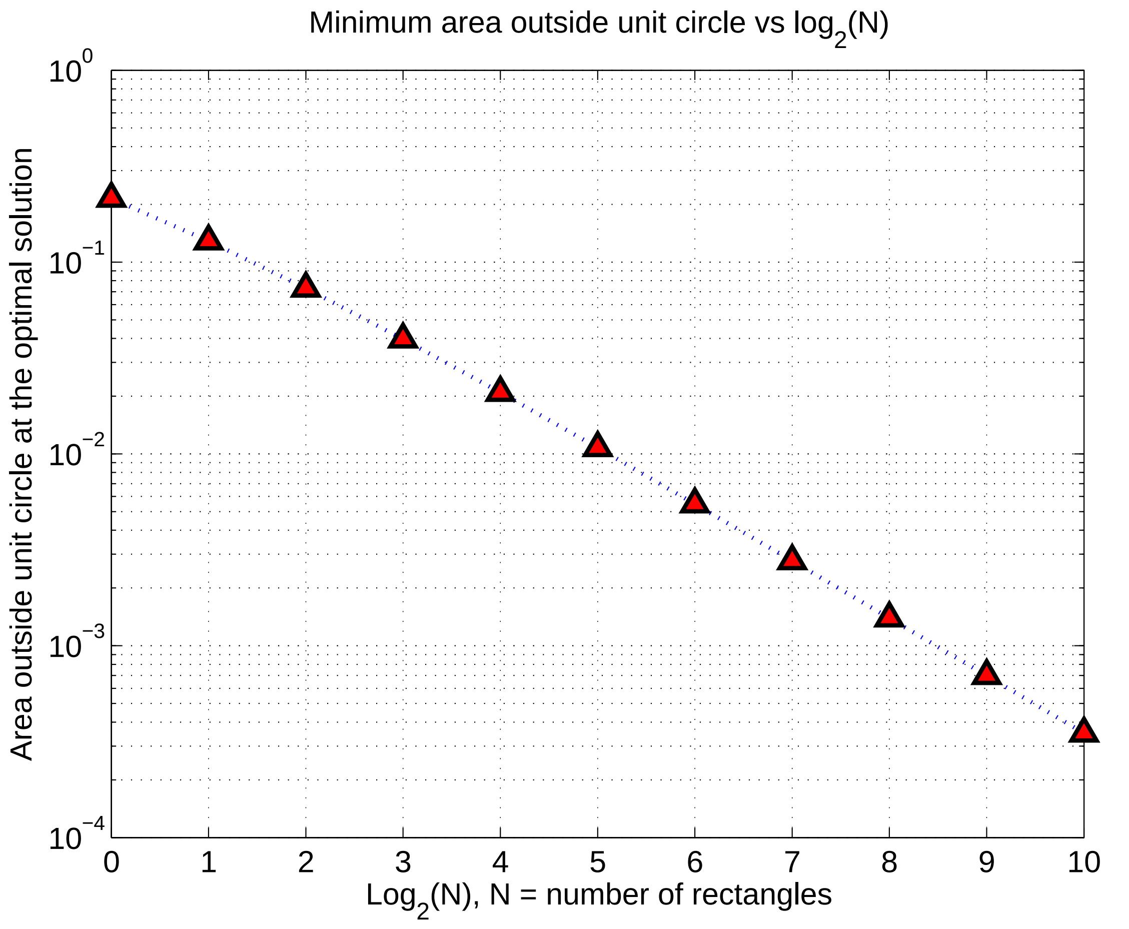


Figure 8: This figure plots F_N^* on a logarithmic axis as a function of $\log_2 N$.

Another interesting quantity of interest is the area outside the unit circle for each rectangle corresponding to the solution for a total of N rectangles. We plot this quantity for $N = 32, 64$ and 128 in Figure 9. The easy observation from these figures is that the area outside each rectangle decreases as the number of rectangles N increases. This is completely expected. However, there is another more interesting fact about these rectangles that is a little harder to see. This is an important result and we state it below:

For a fixed value of N, the area outside the unit circle at the optimal solution for each of the N rectangles is a symmetrical function about $N/2$.

To see this, look at Figure 10 which plots the same figures as in Figure 9 for $N = 32$ and 128 over a smaller dynamic range suitable for each figure. The symmetry is easily seen. However, this is predictable from a geometrical argument on the circle. Imagine how the figures shown previously in 5 (a) and (b) would look like for the unit circle, the latter case being the optimization problem we are solving. The equivalence of the problems characterized by Figures 5 (a) and (b) (as explained in section 2.4) leads to an important conclusion, which is that *Figure 5 (a) at the optimal point for the unit circle must be invariant under a permutation of the coordinate axes $y \leftarrow x, x \leftarrow y$, due to symmetry for any N.* Note that the symmetry argument only holds for the configuration of the rectangles at the optimal solution with the circle playing a key role in the argument. What this implies is that if one swapped the X and Y axes, the entire image (all the rectangles and circle combined) would resemble the image before the swapping took place.

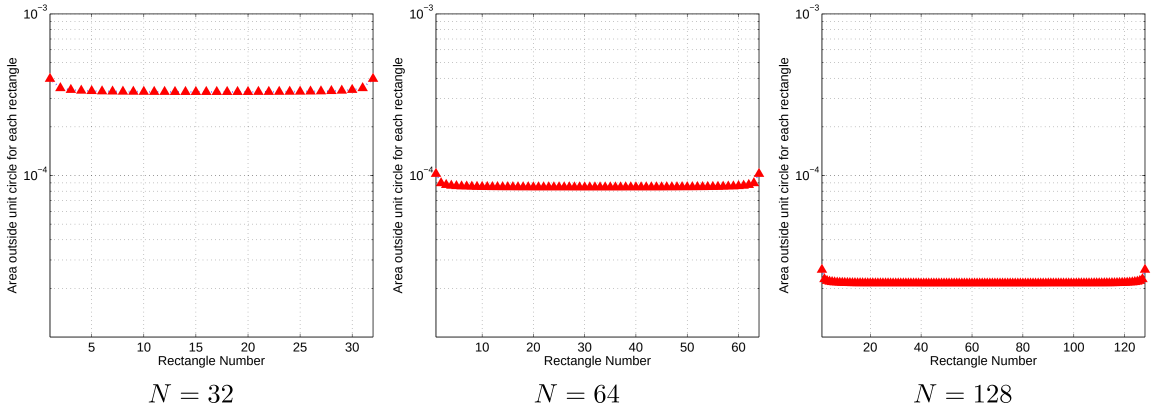


Figure 9: This figure plots the area outside each rectangle at the optimal solution determined by the optimization program for $N = 32$, 64 and 128. The vertical axis here is plotted in a logarithmic scale, and the range of the vertical axis is kept the same.

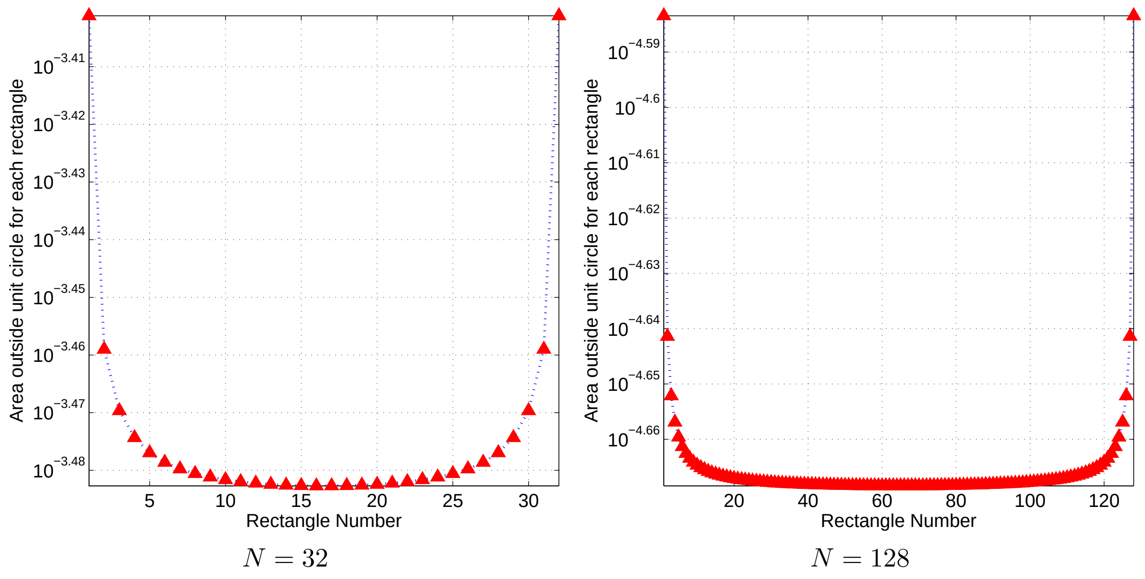
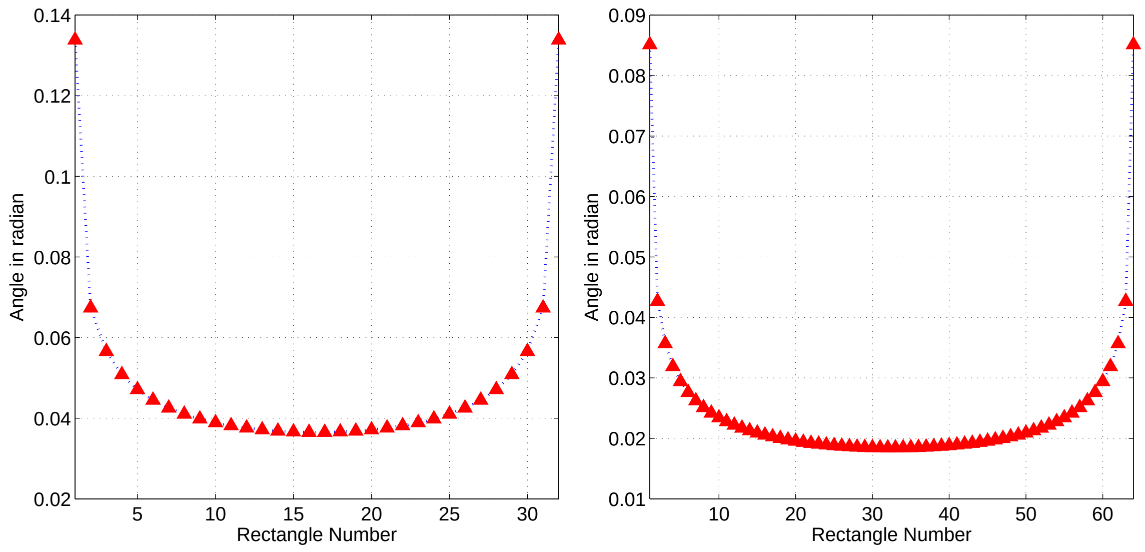


Figure 10: Same plot as in Figure 9 for $N = 32$ and 128 with reduced range on the vertical axis.

An exactly similar symmetry argument also suggests the following result. We state this below:

The angles subtended by the circumference of the circle contained inside each of the N rectangles at the optimal solution is also a symmetrical function about $N/2$.

This result is shown in Figure 11 for the cases $N = 32$ and 64. The angle being plotted here is the angle subtended by the arc at the center of the circle, where the end points of the arc for each rectangle are the two points where each rectangle intersects the circumference of the circle.



$N = 32$ $N = 64$

Figure 11: Plot of the angle subtended by the part of the circumference contained inside each rectangle at the center of the circle, for the cases $N = 32$ and 64.

Both the symmetry results stated above follow because of symmetries of the circle. This is an important point because we would have missed these nice properties of the solution if we had solved our optimization problem on the ellipse directly, as then the same quantities would have no longer been symmetrical.

In the next section, we study how the performance of the algorithm such as run time, number of iterations and other key performance indicators vary as a function of the the number of rectangles N.

4.2 Performance analysis versus the number of rectangles N

1. *Number of function and gradient evaluations:* In Figure 12, we plot the total number of function and gradient evaluations for the entire run of the optimization program for different values of N. The vertical axis is plotted in a logarithmic scale, while along the horizontal axis we plot the value of $\log_2 N$. As we can see, the number of function and gradient evaluations decrease as the value of N increases (except near $N = 2^6 = 64$, I'm not sure what is causing this...needs more

investigation). There are several desirable aspects about these figures that deserve some discussion.

(a) The figures illustrate that increasing the dimension of the problem does not lead to an ever increasing amount of work in terms of function and gradient evaluations. *In fact, it should be pointed out that this is a consequence of using modified Newton to find the search direction at every iteration, which is not the case if the search directions were determined using steepest descent!!.*

(b) Numerical tests done with steepest descent did not even converge in a reasonable amount of time for $N = 2^{10}$ rectangles on a windows PC on which all computing was done for this project. In fact, the number of function and gradient evaluations increase with N.

(c) Another aspect that could be behind these figures is how we choose the initial feasible point. Everything we discussed so far is based on choosing the initial point as $\mathbf{h}^{(\mathbf{0})} = [1/N \ 1/N \ \ldots \ 1/N \ 1/N]^T$. It may be that this is a very good starting solution and changing the initial feasible point to something else might lead to a degraded performance.

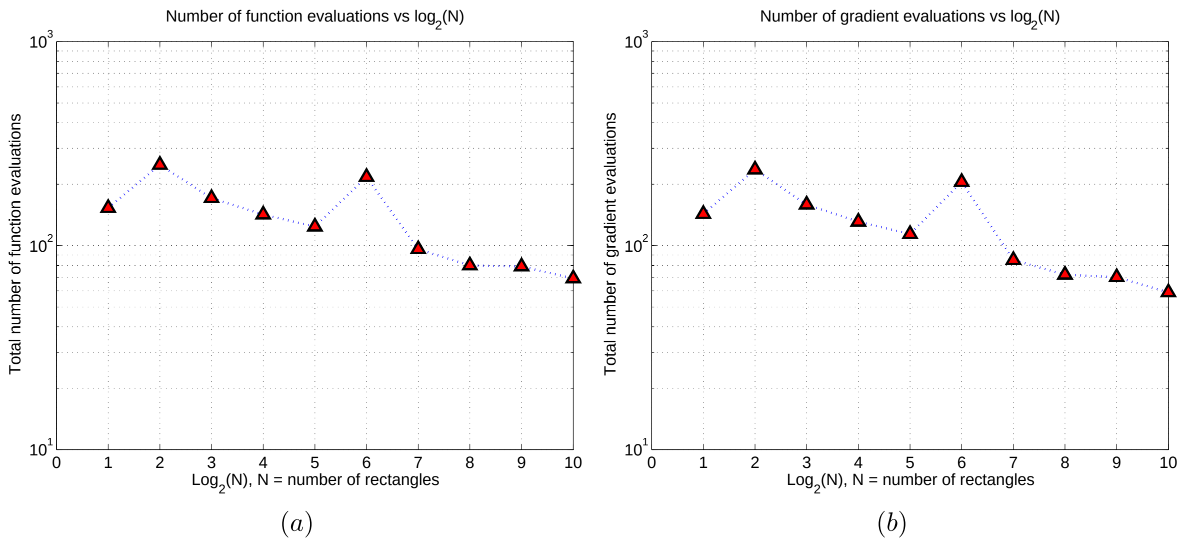


Figure 12: (a) Plot of the total number of function evaluations vs $\log_2 N$. (b) Plot of the total number of gradient evaluations vs $\log_2 N$. The vertical axis is plotted on a logarithmic scale.

2. *Number of iterations and total run time:* We next study the total number of iterations needed to converge to the true solution as a function of N and also how the run time varies with N. These results are plotted in Figure 13 (a) and (b) respectively. We make the following comments about these plots below:

   (a) The optimization algorithm with modified Newton search direction is extremely robust in terms of generating search directions that quickly converge to the optimal solution. The number of iterations appear to be approximately 10 across the entire range of N tried in this project $2^0 - 2^{10}$. However, this again might be influenced by the fact that the initial feasible solution is a good starting point. *An important comment regarding this is that the conclusions would be completely different had one used steepest descent to generate the search directions. With steepest descent the number of iterations needed to converge goes up drastically.*

(b) Figure 13 (b) is the more practical plot of interest that shows that the actual run time does increase with increasing the value of N. Neglecting the initial part of the curve for small values of N where overhead effects are probably influencing the behavior of the curve, we see that for large N the run time increases exponentially (since the horizontal axis plots $\log_2 N$).

(c) A big reason for the increase in run time involves forming the hessian matrix, accessing its elements, performing the modified cholesky factorization and associated increases in computational complexity for matrix vector multiplications with increasing N. So this dependence of the total run time on N is expected.

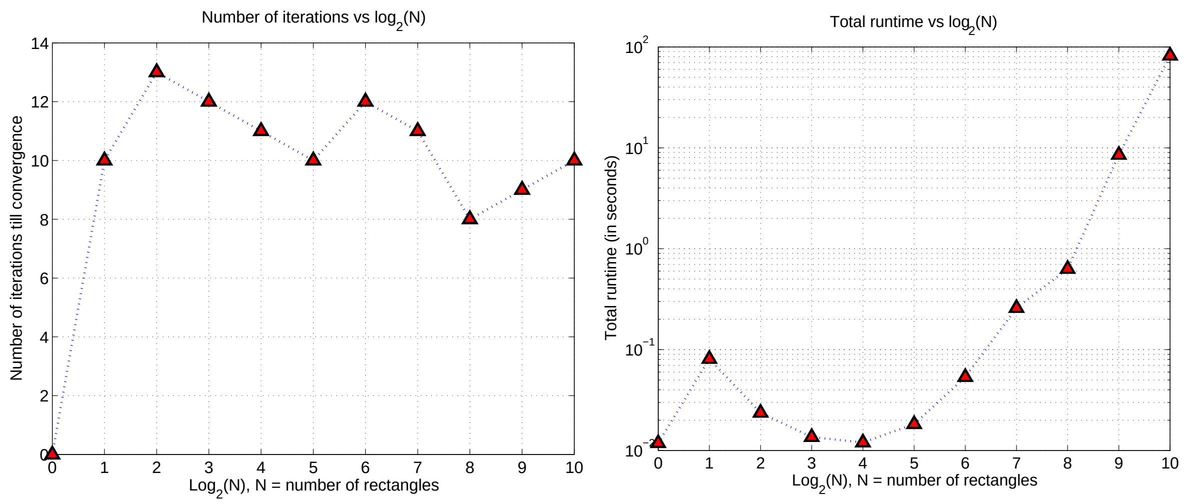


Figure 13: (a) This figure is a plot of the total number of iterations taken to converge to the solution vs $\log_2 N$. (b) This figure is a plot of the total run time of the optimization program vs $\log_2 N$. The vertical axis in (b) is plotted in a logarithmic scale.

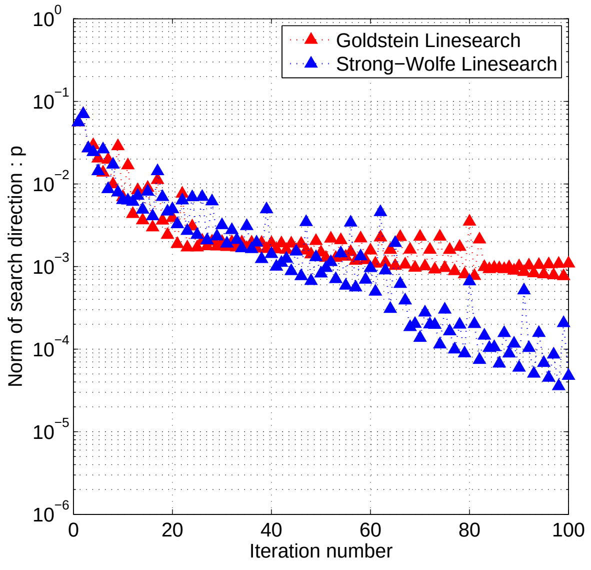
4.3 Performance analysis of the algorithm for different line search strategies and for different descent directions for a fixed value of N

So far we have looked at different results by using descent directions generated using the modified Newton algorithm and the Strong-Wolfe line search criteria inside the optimization program. In this section, we will study how the performance of the optimization program depends on the choice of the line search algorithm and the choice of using different descent directions. We choose $N = 100$ for this study and do not vary it.

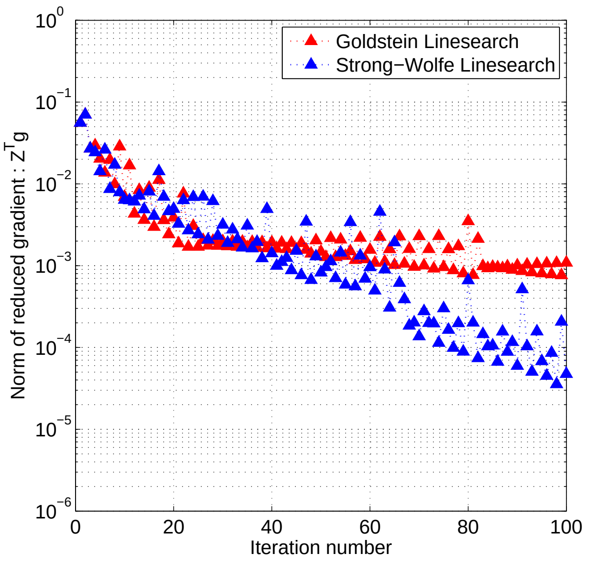
Comparison of Goldstein and Strong-Wolfe line search implementations

In the first test, we use steepest descent to generate our descent directions and then compare some key performance parameters using these directions for different line search methods - Goldstein and Strong-Wolfe, as a function of the iteration number. For the purpose of this study, we fix the total number of iterations at 100. In fact, the solution does not converge in 100 iterations (because we are using steepest descent), but is enough to illustrate the main differences. The following quantities are of interest as a function of the iteration number:

- Norm of reduced gradient $||\hat{\mathbf{g}}||$
- Norm of search direction $||\mathbf{p}||$
- Objective function F
- The step length α
- Number of function and gradient evaluations at every iteration
- Number of cumulative function and gradient evaluations



(a) $||\mathbf{p}||_2$

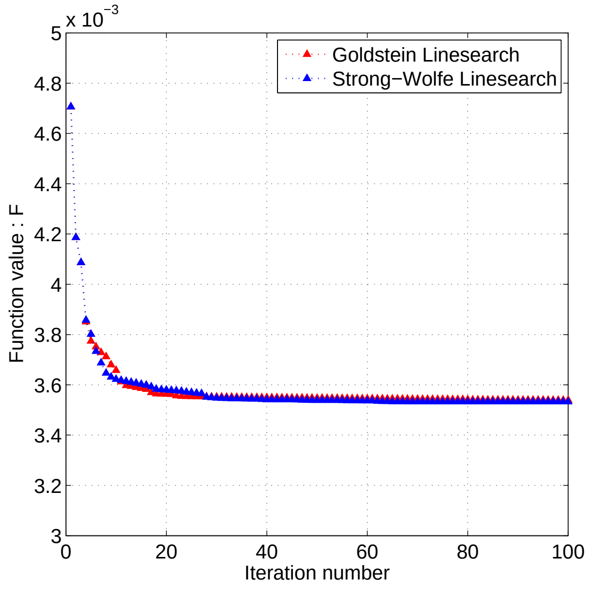


(b) $||\hat{\mathbf{g}}||_2$

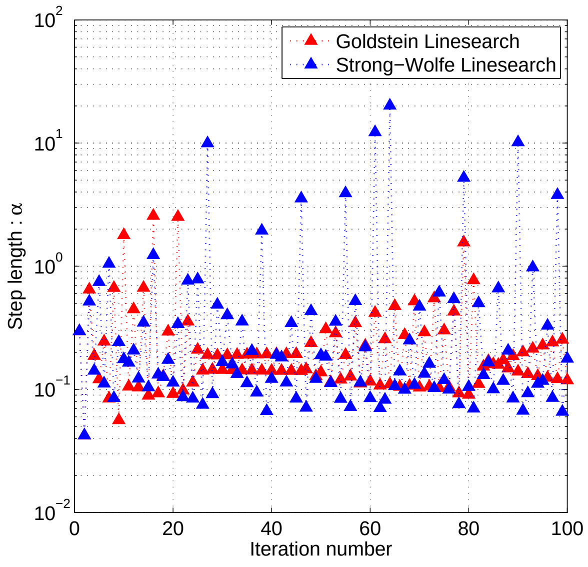
Figure 14: In this figure we plot the norm of the search direction vector $||\mathbf{p}||_2$ and the norm of the reduced gradient $||\hat{\mathbf{g}}||_2$ as a function of the iteration number, in (a) and (b) respectively. The vertical axis is plotted in a logarithmic scale.

In Figures 14 (a) and (b), we plot the norm of the search direction vector $||\mathbf{p}||_2$ and the norm of the reduced gradient $||\hat{\mathbf{g}}||_2$ as a function of the iteration number. As we can see, we are very far from achieving convergence as $||\hat{\mathbf{g}}||_2 \sim 10^{-4}$ after 100 iterations. However, the plots suggest that Strong-Wolfe line search is achieving an order of magnitude better convergence compared to Goldstein line search.

In Figure 15 (a), we plot the function value F versus the iteration number. The plot suggests that in terms of achieving the optimal value of the objective function, both the line search algorithms are good overall. The main differences are in how many digits after the decimal point do the answers match the true solution. However, as we can see, after 40 iterations the algorithms with either line search match the true solution with less than 1% error. In Figure 15 (b), we plot the step length α as a function of the iteration number. It appears that α for either line search spans 3 orders of magnitude $(10^{-1} - 10^{1})$, but there is no pattern in this figure of relevance.



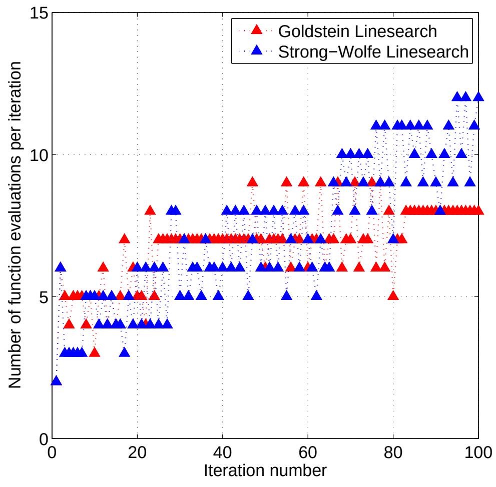
(a) F



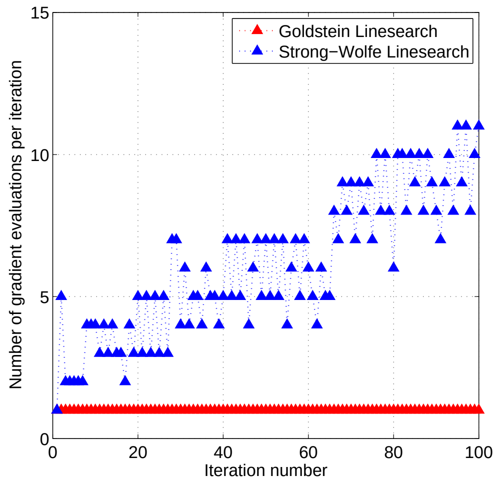
(b) α

Figure 15: In this figure we plot the function value F and the step length α as a function of the iteration number, in (a) and (b) respectively. The vertical axis in (b) is plotted in a logarithmic scale.

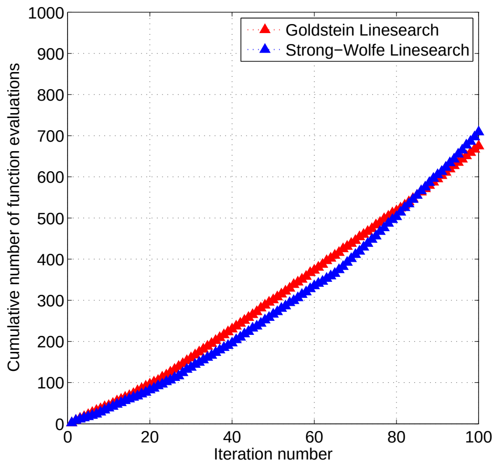
Next, in Figure 16 we plot some useful statistics of the number of function and gradient evaluations as a function of iteration number for the Goldstein and Strong-Wolfe line search algorithms. Figures 16 (a) and (b) show the exact number of function and gradient evaluations respectively, taking place per iteration. As one can see, the number of gradient evaluations in the optimization algorithm with Goldstein line search is just one per iteration. This is because as one can see from equations (13) and (14), the conditions check for only the function values $F(\mathbf{h}+\alpha\mathbf{p})$ at the new points. The quantity $\mathbf{g^T p}$ appearing in the conditions only need to be computed for the first point being checked and for any new point being checked it does not change. However, for Strong-Wolfe line search, we need to compute the gradient of the function at each point we check which explains Figure 16 (b). Finally for Figures 16 (a) and (b), one needs to note that for the Strong-Wolfe line search both the number of function and gradient evaluations are showing an overall increasing tendency with increasing iteration number. *(This is worrying because we expect the amount of computation involved in the line search to decrease as we converge closer and closer to the solution. This probably points to inefficiencies in the line search code that we have currently !)*



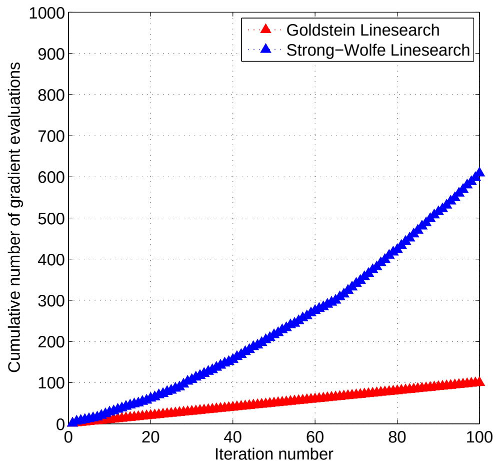
(a)



(b)



(c)



(d)

Figure 16: In this figure we plot : (a) Number of function evaluations per iteration, (b) Number of gradient evaluations per iteration, (c) Cumulative number of function evaluations, (d) Cumulative number of gradient evaluations, versus the iteration number.

In Figures 16 (c) and (d), we plot the cumulative number of function and gradient evaluations as a function of the iteration number. There is no more information in them than in Figures 16 (a) and (b). However, it is still nice to visualize these results. In particular it reveals that the number of function evaluations tend to be the same for both Goldstein and Strong-Wolfe line search algorithms, while the number of gradient evaluations is clearly much greater for Strong-Wolfe line search than for Goldstein line search.

Comparison of steepest descent and modified Newton search directions

In this section we study the superiority of the modified Newton algorithm as compared to the steepest descent algorithm when it comes to generating good search directions. We know of the theoretical result that close to the true solution, Newton iterates exhibit quadratic convergence. This is really what is behind some of the spectacular results that follow. The number of rectangles in this study is again fixed at $N = 100$, while the optimization program is run for 100 iterations, without an explicit convergence criteria enforced to stop execution. We have already demonstrated that with steepest descent, we do not achieve convergence in 100 iterations, but we will be pleasantly surprised to see that modified Newton converges in ~ 50 iterations. For this study, we will use the Strong-Wolfe conditions in the line search. We plot the same quantities plotted in the Figures 14, 15 and 16, but this time we compare the results obtained using modified Newton versus steepest descent algorithms.

We begin by plotting the norm of the search direction vector $||\mathbf{p}||_2$ and the norm of the reduced gradient $||\hat{\mathbf{g}}||_2$ in Figures 17 (a) and (b) respectively, as a function of the iteration number using modified Newton and steepest descent directions. The results show that both these quantities $||\mathbf{p}||_2$ and $||\hat{\mathbf{g}}||_2$ reach machine precision $\sim 10^{-15}$ by iteration 50 for modified Newton. Contrast this with the dismal performance of steepest descent which only manages to reduce these quantities to about $\sim 10^{-5}$ after 100 iterations. This clearly is a consequence of the quadratic convergence properties of the Newton iterates close to the optimal point, something that the steepest descent iterates lack completely.

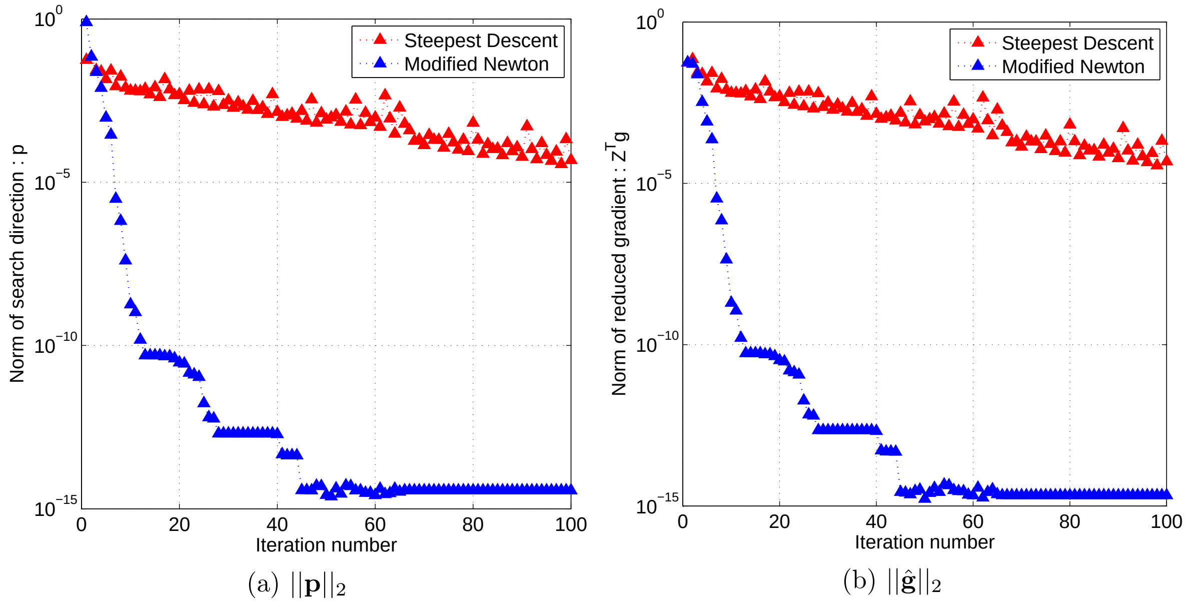
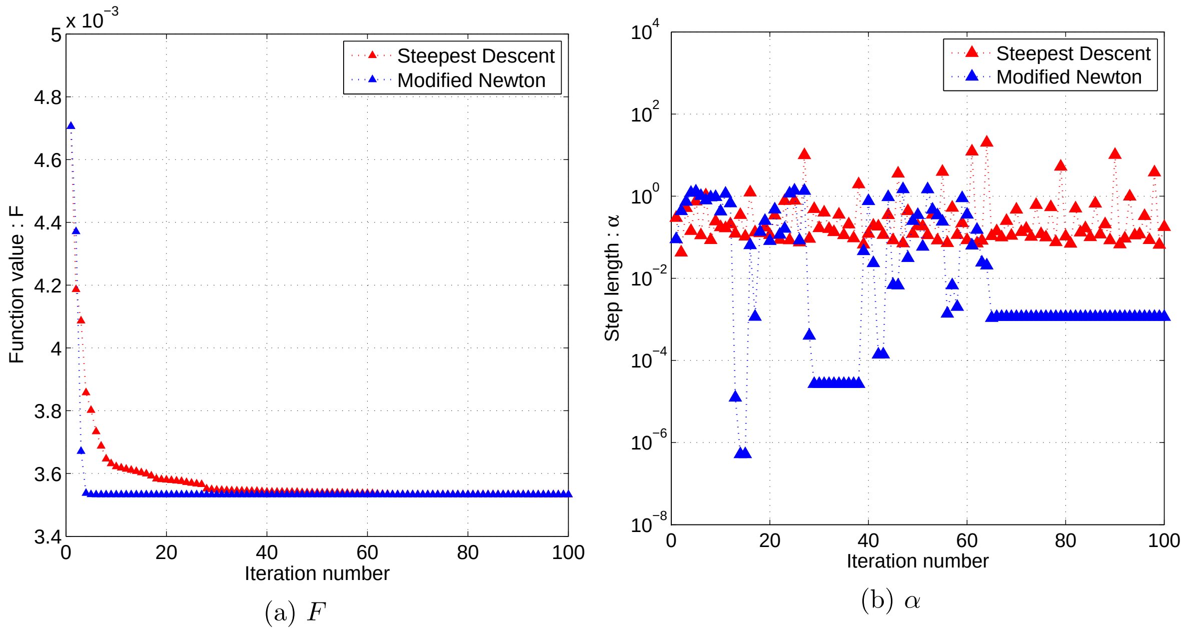


Figure 17: In this figure we plot the norm of the search direction vector $||\mathbf{p}||_2$ and the norm of the reduced gradient $||\hat{\mathbf{g}}||_2$ as a function of the iteration number, in (a) and (b) respectively. The vertical axis is plotted in a logarithmic scale.

In Figure 18 (a), we plot the objective function F versus the number of iterations for the optimization program run using modified Newton and steepest descent directions. The rate at which the modified Newton iterates converge to the true solution is spectacular - in fact we reach the true solution within an error tolerance of 1% of the objective function value at the optimal solution, in just 5 iterations. Comparatively, the steepest descent iterates take about 40 iterations to reach the same tolerance in terms of the objective function value. The amazing convergence of the modified Newton iterates is a consequence of quadratic convergence of the Newton iterates near the optimal solution. In Figure 18 (b), we plot the value of the step length α as a function of the iteration number. The general observation is that for steepest descent, α is in the range $\sim 10^{-1} - 10^{1}$. For modified Newton iterates, the value of α seems to have a bigger dynamic range $\sim 10^{-6} - 10^{0}$, but there is also an undesirable feature for $N \sim 65$ or larger. It seems that the value of alpha becomes equal to 10^{-3} in this range. *This shouldn't be, as from theory we know that α should be close to 1 near the optimal solution, for search directions generated using the Newton method. This probably is due to inefficiencies in the line search routine and needs more investigation ! However, it is equally likely that this effect is a manifestation of the fact that we cannot compute very small numbers ($\sim 10^{-15}$) on the computer with high accuracy, and the lack of precision in being able to compute the function value and the gradient interferes with the quadratic convergence of the Newton iterates.*

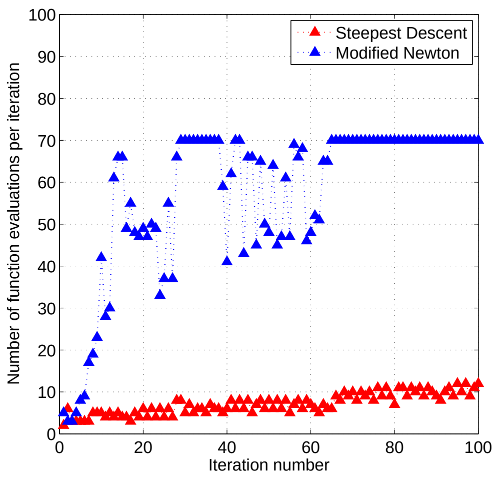


(a) F (b) α

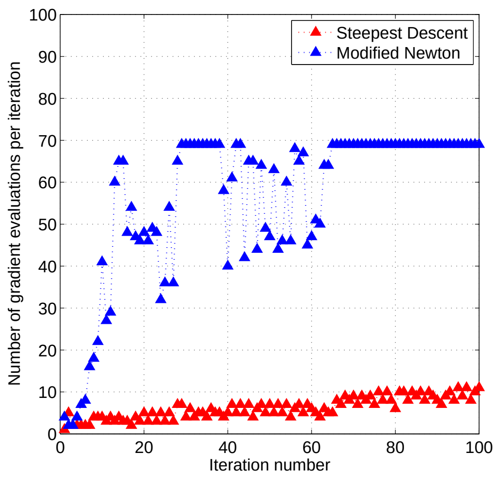
Figure 18: In this figure we plot the function value F and the step length α as a function of the iteration number, in (a) and (b) respectively. The vertical axis in (b) is plotted in a logarithmic scale.

Finally, we plot the number of function and gradient evaluations per iteration, for steepest descent and modified Newton iterates in Figures 19 (a) and (b) respectively. We see that for modified Newton search directions, the Strong-Wolfe line search needs to perform many more function and gradient evaluations compared to the steepest descent search directions. *The unpleasant aspect about these plots is that for modified Newton search directions, the line search routine does too many computations even after convergence is reached. This can either be due to the inability to compute and compare very small numbers ($\sim 10^{-15}$) accurately on a computer, and due to inefficiencies in the line search implementation.*

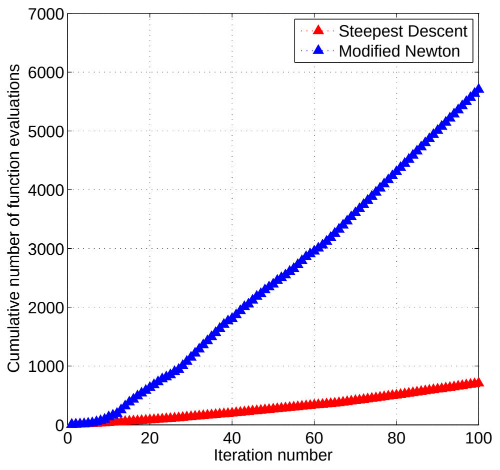
Figures 19 (c) and (d) plot the cumulative number of function and gradient evaluations respectively for the optimization program with descent directions generated using modified Newton or steepest descent algorithm, versus the iteration number. Although we expect the plots to be increasing with the number of iterations (which is the case for both algorithms), it is clear that the total number of function and gradient computations become very large for the modified Newton search direction implementation. This may be also true for the steepest descent directions once the solution gets "close enough" to the true solution, which the steepest descent algorithm fails to do in 100 iterations and so we don't see the effect in these plots.



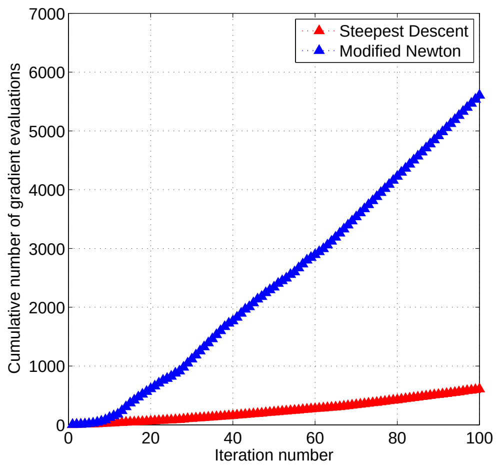
(a)



(b)



(c)



(d)

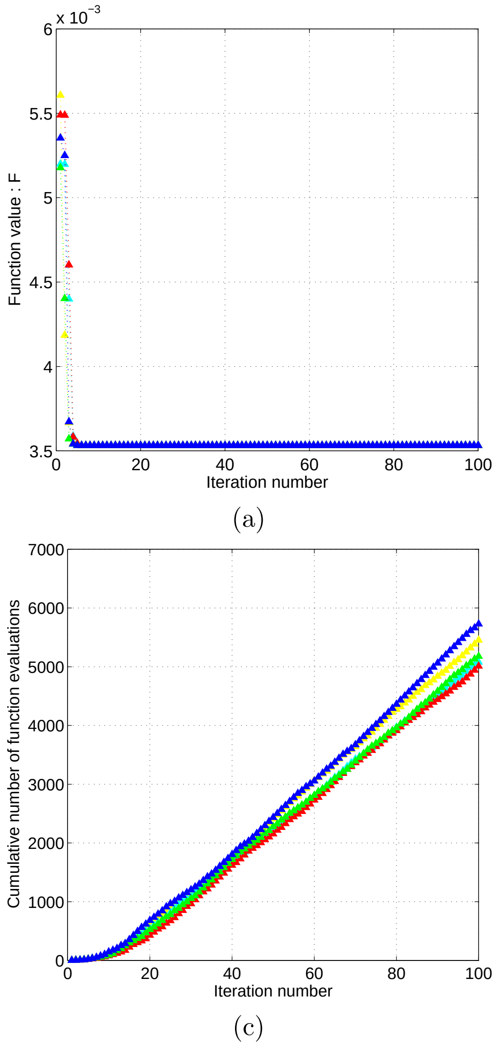
Figure 19: In this figure we plot : (a) Number of function evaluations per iteration, (b) Number of gradient evaluations per iteration, (c) Cumulative number of function evaluations, (d) Cumulative number of gradient evaluations, versus the iteration number.

Finally, we should note that even though we plotted 100 iterations with the modified Newton algorithm being used to generate the search directions inside the optimization program, Figure 12 says that convergence is reached in only about 10-15 iterations.

So all the issues pointed out about the line search routine, are never really encountered in practice as the program stops much before due to convergence being achieved. Nevertheless, the plots in Figures 18 and 19 are important to consider in order to be able to find computer bugs.

4.4 Impact of choosing different starting solutions on performance

In this section we study how choosing different starting solutions impact the performance of the optimization program for a fixed value of $N = 100$.



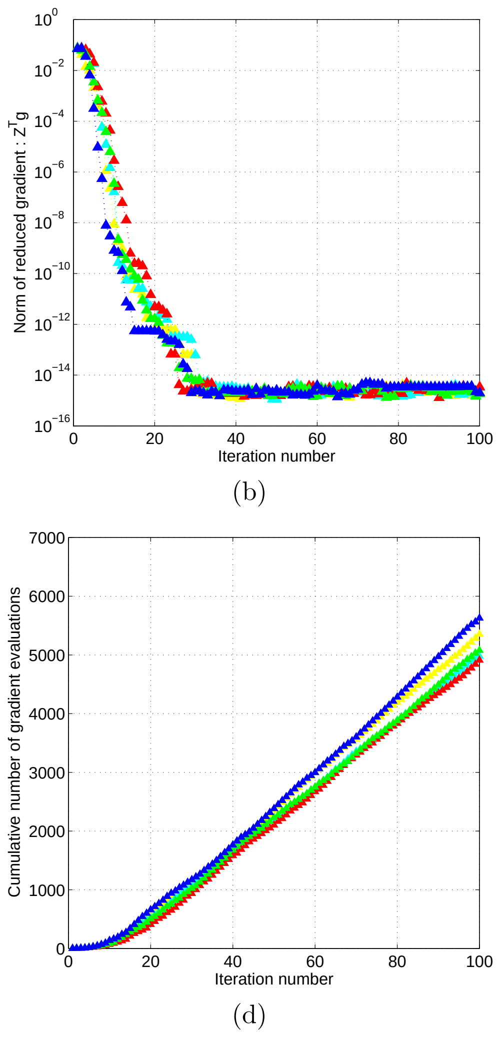
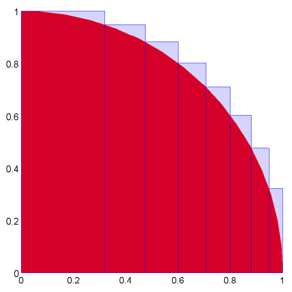


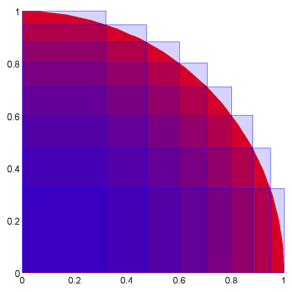
Figure 20: In this figure we plot for different randomly generated starting feasible solutions : (a) function value F, (b) norm of the reduced gradient $||\hat{\mathbf{g}}||_2$, (c) Cumulative number of function evaluations, (d) Cumulative number of gradient evaluations, versus the iteration number. The vertical axis is (a) and (b) are plotted in a logarithmic scale. Each color represents a different starting solution and the colors across all the four figures are consistent.

Again, we have plotted some key performance indicators for 100 iterations, like function value F, the norm of the reduced gradient $||\hat{\mathbf{g}}||_2$, cumulative number of function and gradient evaluations as a function of the iteration number. We use the modified Newton algorithm to generate search directions and Strong-Wolfe line search for this study. The results are plotted in Figure 20. Each of the colors represent the results for a randomly generated initial feasible point. As one can see, there is not much difference in the performance of the algorithm when we use different starting feasible points, and all of them exhibit similar convergence characteristics and computational effort in terms of function and gradient evaluations.

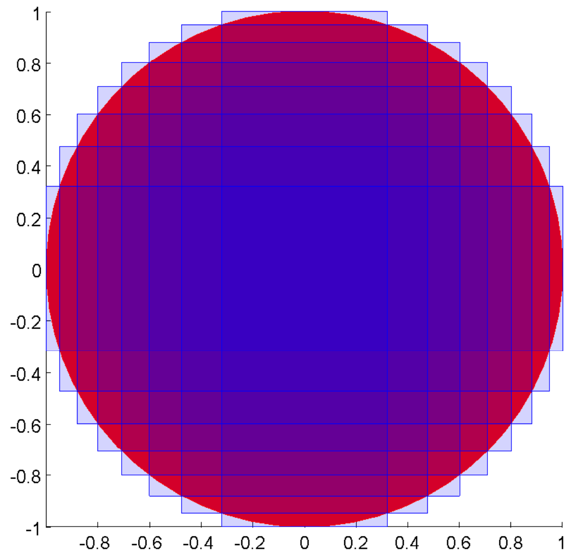
4.5 Extending the solution to the full ellipse



(a)



(b)



(c)

Figure 21: This figure illustrates how the solution obtained can be extended to obtain the configuration of the rectangles for the unit circle on all the four quadrants. (a) This is the solution of the optimization problem that we solve for the unit circle on the first quadrant. (b) Next, note how the solution in (a) remains unchanged for this new configuration of the rectangles. (c) Finally, the configuration in (b) can be extended to all the four quadrants to give the solution of the ellipse covering problem in all four quadrants. Here we have plotted the results for $N = 8$ rectangles.

We finish the analysis by showing how the problem solved on only the first quadrant for a unit circle directly yields a SCC solution for the whole ellipse in all the four quadrants. The result is through a simple geometrical extension. We first consider Figures 21 (a), (b) and (c). Here we look at results using $N = 8$ rectangles. Figure 21 (a) plots the configuration of the rectangles that cover the unit circle in the first quadrant in our formulation of the optimization problem. Next, we note that Figure 21 (b) represents another configuration of the rectangles that leaves the optimal solution unchanged as in Figure 21 (a). Also note that by doing so, we have not changed the number of rectangles, as was discussed in section 2.4 where we first formulated our problem stating the equivalence between the forms in Figures 5 (a) and (b). Finally note that how we can directly obtain the solution for the unit circle in all the four quadrants by first adding the reflection of Figure 21 (b) about the plane $x = 0$ and then the adding the reflection the resulting figure about the plane $y = 0$ to get the full configuration. This is plotted in Figure 21 (c).

Once the SCC using N rectangles is known for the unit circle on all the four quadrants, we can easily find the SCC for any ellipse of semi-major and semi-minor axes given by a and b respectively. One can do this by simply performing a scaling of the X and Y axes appropriately ($x \leftarrow ax$, $y \leftarrow by$). For example, in Figure 22 we show how such an extension can be done for $a = 2$, $b = 1$. As a result of this transformation, the unit circle transforms into the desired ellipse and the rectangles also get stretched appropriately. Needless to say, *this transformation preserves the SCC property of the rectangles and is in fact the optimal solution for the desired ellipse that minimizes the area outside the ellipse.* This fact was discussed before.

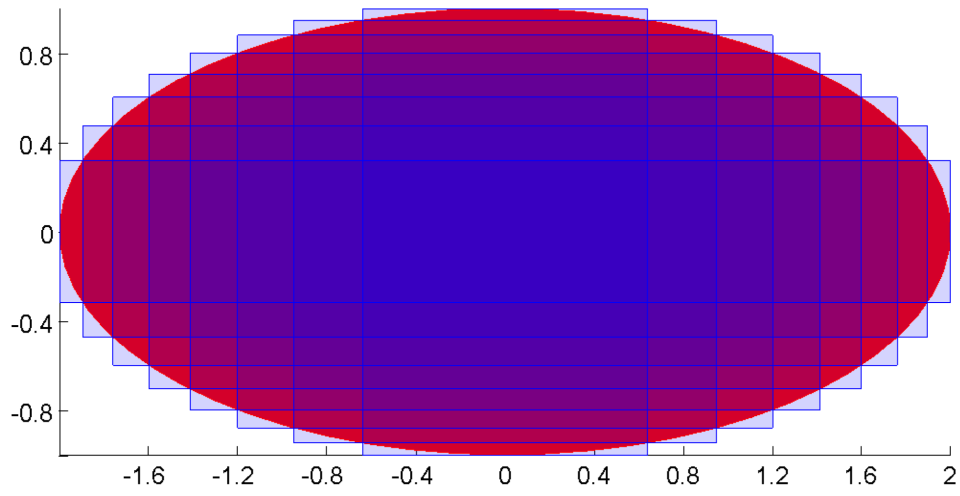


Figure 22: The SCC using $N = 8$ rectangles for an ellipse with semi-major and semi-minor axes lengths $a = 2$ and $b = 1$ respectively, that minimizes the area outside the ellipse.

5 Future Work

Some cool results have been generated by running the program on super-ellipses which are described by equation (17). However, the results are in unstructured form and more study needs to be done. Hence we avoid putting them here. Some pictures will be sent at a later date.

$$\left|\frac{x}{a}\right|^{\alpha} + \left|\frac{y}{b}\right|^{\beta} = 1 \;\;,\;\; \alpha > 0, \beta > 0 \tag{17}$$

6 Acknowledgements

I'd like to thank Carlos for excellent exposition of some very important and difficult concepts pertinent to numerical optimization in his office hours for the course CME 304. I'd also like to thank Prof. Walter Murray for making this project an essential component of the course. Trying to implement computer programs to solve the problem really taught me a great deal about some of the major difficulties that one encounters during program execution like limits of finite precision floating point arithmetic, performance issues with inefficient line search routines, trade-offs necessary to solve problems in a reasonable amount of time with limited memory and computing resources etc.

References

[1] Nocedal, Jorge and Wright, Stephen, 2006, Numerical optimization: Springer Science & Business Media

[2] Gill, Philip E and Murray, Walter and Wright, Margaret H, 1981, Practical optimization: Academic press

[3] Murray, Walter, 2016, Numerical Optimization CME 304 course lecture notes