Chapter 1

NLP Algorithms

1.1 Algorithms Introduction

We will begin with unconstrained optimization and consider several different algorithms based on what is known about the objective function. In particular, we will consider the cases where we use

- Only function evaluations (also known as *derivative free optimization*),
- Function and gradient evaluations,
- Function, gradient, and hessian evaluations.

We will first look at these algorithms and their convergence rates in the 1-dimensional setting and then extend these results to higher dimensions.

1.2 1-Dimensional Algorithms

We suppose that we solve the problem

$$\min f(x) \tag{1.2.1}$$
$$x \in [a, b]. \tag{1.2.2}$$

That is, we minimize the univariate function $f(x)$ on the interval $[l, u]$.
For example,

$$\min(x^2 - 2)^2 \tag{1.2.3}$$
$$0 \leq x \leq 10. \tag{1.2.4}$$

Note, the optimal solution lies at $x^* = \sqrt{2}$, which is an irrational number. Since we will consider algorithms using floating point precision, we will look to return a solution $\bar{x}$ such that $\|x^* - \bar{x}\| < \epsilon$ for some small $\epsilon > 0$, for instance, $\epsilon = 10^{-6}$.

1.2.1 Golden Search Method - Derivative Free Algorithm

https://www.youtube.com/watch?v=hLm8xfwWYPw

Suppose that $f(x)$ is unimodal on the interval $[a, b]$, that is, it is a continuous function that has a single minimizer on the interval.

Without any extra information, our best guess for the optimizer is $\bar{x} = \frac{a+b}{2}$ with a maximum error of $\epsilon = \frac{b-a}{2}$. Our goal is to reduce the size of the interval where we know x^* to be, and hence improve our best guess and the maximum error of our guess.

Now we want to choose points in the interior of the interval to help us decide where the minimizer is. Let x_1, x_2 such that

$$a < x_2 < x_1 < b.$$

Next, we evaluate the function at these four points. Using this information, we would like to argue a smaller interval in which x^* is contained. In particular, since f is unimodal, it must hold that

1. $x^* \in [a, x_2]$ if $f(x_1) \leq f(x_2)$,
2. $x^* \in [x_1, b]$ if $f(x_2) < f(x_1)$,

After comparing these function values, we can reduce the size of the interval and hence reduce the region where we think x^* is.

We will now discuss how to chose x_1, x_2 in a way that we can

1. Reuse function evaluations,
2. Have a constant multiplicative reduction in the size of the interval.

We consider the picture:

To determine the best d, we want to decrease by a constant factor. Hence, we decrease be a factor γ, which we will see is the golden ration (GR). To see this, we assume that $(b - a) = 1$, and ask that $d = \gamma$. Thus, $x_1 - a = \gamma$ and $b - x_2 = \gamma$. If we are in case 1, then we cut off $b - x_1 = 1 - \gamma$. Now, if we iterate and do this again, we will have an initial length of γ and we want to cut off the interval $x_2 - x_1$ with this being a proportion of $(1 - \gamma)$ of the remaining length. Hence, the second time we will cut of $(1 - \gamma)\gamma$, which we set as the length between x_1 and x_2.

Considering the geometry, we have

$$\text{length } a \text{ to } x_1 \ + \text{length } x_2 \text{ to } b = \ \text{total length } + \ \text{length } x_2 \text{ to } x_2$$

hence

$$\gamma + \gamma = 1 + (1 - \gamma)\gamma.$$

Simplifying, we have

$$\gamma^2 + \gamma - 1 = 0.$$

Applying the quadratic formula, we see

$$\gamma = \frac{-1 \pm \sqrt{5}}{2}.$$

Since we want $\gamma > 0$, we take

$$\gamma = \frac{-1+\sqrt{5}}{2} \approx 0.618$$

This is exactly the Golden Ratio (or, depending on the definition, the golden ration minus 1).

Example:

We can conclude that the optimal solution is in $[1.4, 3.8]$, so we would guess the midpoint $\bar{x} = 2.6$ as our approximate solution with a maximum error of $\epsilon = 1.2$.

Convergence Analysis of Golden Search Method:
After t steps of the Golden Search Method, the interval in question will be of length

$$(b-a)(GR)^t \approx (b-a)(0.618)^t$$

Hence, by guessing the midpoint, our worst error could be

$$\frac{1}{2}(b-a)(0.618)^t.$$

1.2.2 Bisection Method - 1st Order Method (using Derivative)

Minimization Interpretation

Assumptions: f is convex, differentiable

We can look for a minimizer of the function $f(x)$ on the interval $[a, b]$.

Root finding Interpretation

Instead of minimizing, we can look for a root of $f'(x)$. That is, find x such that $f'(x) = 0$.
Assumptions: $f'(a) < 0 < f'(b)$, OR, $f'(b) < 0 < f'(a)$. f' is continuous
The goal is to find a root of the function $f'(x)$ on the interval $[a, b]$. If f is convex, then we know that this root is indeed a global minimizer.
Note that if f is convex, it only makes sense to have the assumption $f'(a) < 0 < f'(b)$.

Convergence Analysis of Bisection Method:
After t steps of the Bisection Method, the interval in question will be of length

$$(b-a)\left(\frac{1}{2}\right)^t.$$

Hence, by guessing the midpoint, our worst error could be

$$\frac{1}{2}(b-a)\left(\frac{1}{2}\right)^t.$$

1.2.3 Gradient Descent - 1st Order Method (using Derivative)

Input: $f(x)$, $\nabla f(x)$, initial guess x^0, learning rate α, tolerance ϵ
Output: An approximate solution x

```
1. Set $t = 0$
2. While $\|f(x^t)\|_2 > \epsilon$:
    (a) Set $x^{t+1} \leftarrow x^t - \alpha \nabla f(x^t)$.
    (b) Set $t \leftarrow t + 1$.
3. Return $x^t$.
```

1.2.4 Newton's Method - 2nd Order Method (using Derivative and Hessian)

Input: $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)$, initial guess x^0, learning rate α, tolerance ϵ
Output: An approximate solution x

```
1. Set $t = 0$
2. While $\|f(x^t)\|_2 > \epsilon$:
    (a) Set $x^{t+1} \leftarrow x^t - \alpha[\nabla^2 f(x^t)]^{-1} \nabla f(x^t)$.
    (b) Set $t \leftarrow t + 1$.
3. Return $x^t$.
```

1.3 Multi-Variate Unconstrained Optimizaition

We will now use the techniques for 1-Dimensional optimization and extend them to multi-variate case. We will begin with unconstrained versions (or at least, constrained to a large box) and then show how we can apply these techniques to constrained optimization.

1.3.1 Descent Methods - Unconstrained Optimization - Gradient, Newton

Outline for Descent Method for Unconstrained Optimization:
Input:

- A function $f(x)$
- Initial solution x^0
- Method for computing step direction d_t
- Method for computing length t of step
- Number of iterations T

Output:

- A point x_T (hopefully an approximate minimizer)

Algorithm

1. For $t = 1, \dots, T$,
$$\text{set } x_{t+1} = x_t + \alpha_t d_t$$

Choice of α_t

There are many different ways to choose the step length α_t. Some choices have proofs that the algorithm will converge quickly. An easy choice is to have a constant step length $\alpha_t = \alpha$, but this may depend on the specific problem.

Choice of d_t using $\nabla f(x)$

Choice of descent methods using $\nabla f(x)$ are known as *first order methods*. Here are some choices:

1. **Gradient Descent:** $d_t = -\nabla f(x_t)$
2. **Nesterov Accelerated Descent:** $d_t = \mu(x_t - x_{t-1}) - \gamma \nabla f(x_t + \mu(x_t - x_{t-1}))$

Here, μ, γ are some numbers. The number μ is called the momentum.

1.3.2 Stochastic Gradient Descent - The mother of all algorithms.

A popular method is called *stochastic gradient descent* (SGD). This has been described as "The mother of all algorithms". This is a method to **approximate the gradient** typically used in machine learning or stochastic programming settings.

Stochastic Gradient Descent:
Suppose we want to solve

$$\min_{x \in \mathbb{R}^n} F(x) = \sum_{i=1}^{N} f_i(x). \tag{1.3.1}$$

We could use *gradient descent* and have to compute the gradient $\nabla F(x)$ at each iteration. But! We see that in the **cost to compute the gradient** is roughly $O(nN)$, that is, it is very dependent on the number of function N, and hence each iteration will take time dependent on N.

Instead! Let i be a uniformly random sample from $\{1,\dots,N\}$. Then we will use $\nabla f_i(x)$ as an approximation of $\nabla F(x)$. Although we lose a bit by using a guess of the gradient, this approximation only takes $O(n)$ time to compute. And in fact, in expectation, we are doing the same thing. That is,

$$N \cdot \mathbb{E}(\nabla f_i(x)) = N \sum_{i=1}^{N} \frac{1}{N} \nabla f_i(x) = \sum_{i=1}^{N} \nabla f_i(x) = \nabla \left(\sum_{i=1}^{N} f_i(x) \right) = \nabla F(x).$$

Hence, the SGD algorithm is:

1. Set $t = 0$
2. While ...(some stopping criterion)
   (a) Choose i uniformly at random in $\{1,\dots,N\}$.
   (b) Set $d_t = \nabla f_i(x_t)$
   (c) Set $x_{t+1} = x_t - \alpha d_t$

There can be many variations on how to decide which functions f_i to evaluate gradient information on. Above is just one example.

Linear regression is an excellent example of this.

Example 1: Linear Regression with SGD Given data points $x^1,\dots,x^N \in \mathbb{R}^d$ and output $y^1,\dots,y^N \in \mathbb{R}$, find $a \in \mathbb{R}^d$ and $b \in \mathbb{R}$ such that $a^\top x^i + b \approx y^i$. This can be written as the optimization problem

$$\min_{a,b} \quad \textstyle\sum_{i=1}^N g_i(a,b) \tag{1.3.2}$$

where $g_i(a,b) = (a^\top x^i + b)^2$.

Notice that the objective function $G(a,b) = \sum_{i=1}^N g_i(a,b)$ is a convex quadratic function. The gradient of the objective function is

$$\nabla G(a,b) = \sum_{i=1}^{N} \nabla g_i(a,b) = \sum_{i=1}^{N} 2x^i(a^\top x^i + b)$$

Hence, if we want to use gradient descent, we must compute this large sum (think of $N \approx 10,000$).

Instead, we can **approximate the gradient!**. Let $\tilde{\nabla} G(a,b)$ be our approximate gradient. We will compute this by randomly choosing a value $r \in \{1,\dots,N\}$ (with uniform probability). Then set

$$\tilde{\nabla} G(a,b) = \nabla g_r(a,b).$$

It holds that the expected value is the same as the gradient, that is,

$$\mathbb{E}(\tilde{\nabla} G(a,b)) = G(a,b).$$

Hence, we can make probabilistic arguments that these two will have the same (or similar) convergence properties (in expectation).

Choice of Δ_k using the hessian $\nabla^2 f(x)$

These choices are called *second order methods*

1. **Newton's Method:** $\Delta_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
2. **BFGS (Quasi-Newton):** $\Delta_k = -(B_k)^{-1} \nabla f(x_k)$

Here

$$\begin{aligned} s_k &= x_{k+1} - x_k \\ y_k &= \nabla f(x_{k+1}) - \nabla f(x_k) \end{aligned}$$

and

$$B_{k+1} = B_k - \frac{(B_k s_k)(B_k s_k)^\top}{s_k^\top B_k s_k} + \frac{y_k y_k^\top}{y_k^\top s_k}.$$

This serves as an approximation of the hessian and can be efficiently computed. Furthermore, the inverse can be easily computed using certain updating rules. This makes for a fast way to approximate the hessian.

1.4 Constrained Convex Nonlinear Programming

Given a convex function $f(x)\colon \mathbb{R}^d \to \mathbb{R}$ and convex functions $f_i(x)\colon \mathbb{R}^d \to \mathbb{R}$ for $i = 1, \ldots, m$, the *convex programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \le 0 \quad \text{for } i = 1, \ldots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{1.4.1}$$

1.4.1 Barrier Method

Constrained Convex Programming via Barrier Method:
We convert (**??**) into the unconstrained minimization problem:

$$\begin{aligned} \min \quad & f(x) - \phi \sum_{i=1}^{m} \log(-f_i(x)) \\ & x \in \mathbb{R}^d \end{aligned} \tag{1.4.2}$$

Here $\phi > 0$ is some number that we choose. As $\phi \to 0$, the optimal solution $x(\phi)$ to (**??**) tends to the optimal solution of (**??**). That is $x(\phi) \to x^*$ as $\phi \to 0$.

Constrained Convex Programming via Barrier Method - Initial solution:
Define a variable $s \in \mathbb{R}$ and add that to the right hand side of the inequalities and then minimize it in the objective function.

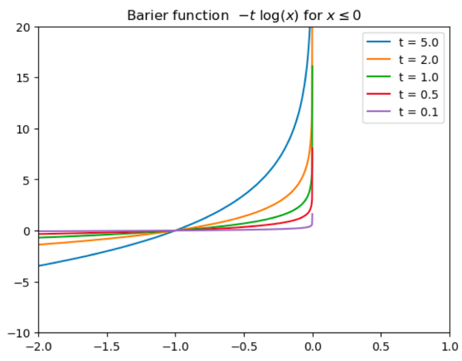
$$\begin{aligned} \min \quad & s \\ \text{s.t.} \quad & f_i(x) \le s \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d, s \in \mathbb{R} \end{aligned} \tag{1.4.3}$$

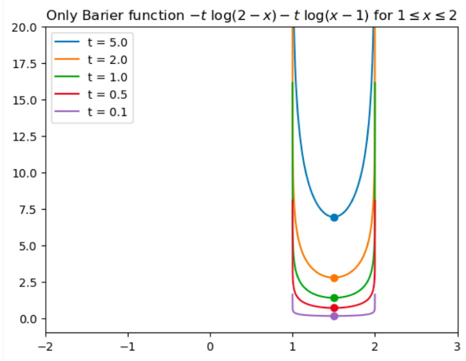
Note that this problem is feasible for all x values since s can always be made larger. If there exists a solution with $s \le 0$, then we can use the corresponding x solution as an initial feasible solution. Otherwise, the problem is infeasible.
Now, convert this problem into the unconstrained minimization problem:

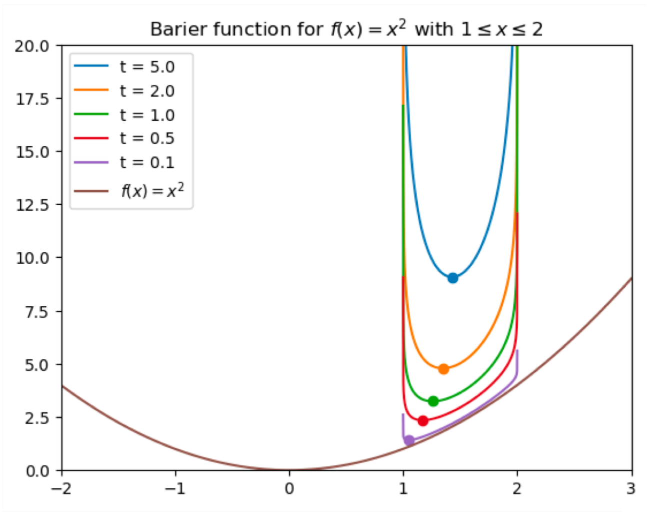
$$\begin{aligned} \min \quad & f(x) - \phi \sum_{i=1}^{m} \log(-(f_i(x) - s)) \\ & x \in \mathbb{R}^d, s \in \mathbb{R} \end{aligned} \tag{1.4.4}$$

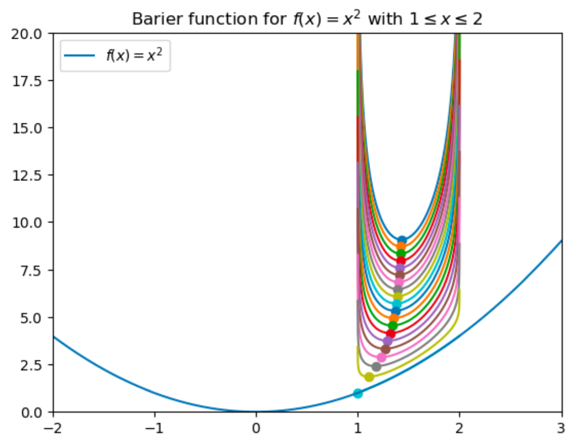
This problem has an easy time of finding an initial feasible solution. For instance, let $x = 0$, and then $s = \max_i f_i(x) + 1$.

Images below: the value t **is the value** ϕ **discussed above**

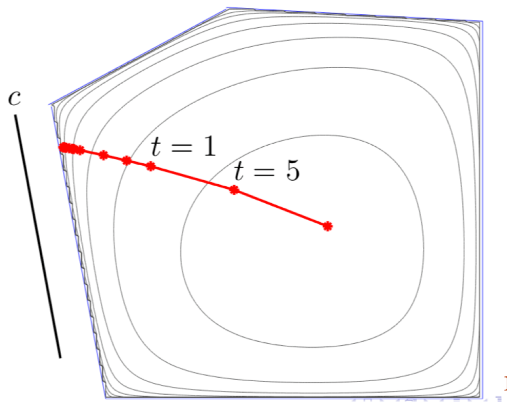








Minimizing $c^T x$ subject to $Ax \leq b$.



[1]

[1]Image taken from unknown source.

Chapter 2

Computational Issues with NLP

We mention a few computational issues to consider with nonlinear programs.

2.1 Irrational Solutions

Consider nonlinear problem (this is even convex)

$$\begin{array}{rl} \min & -x \\ \text{s.t.} & x^2 \leq 2. \end{array} \tag{2.1.1}$$

The optimal solution is $x^* = \sqrt{2}$, which cannot be easily represented. Hence, we would settle for an **approximate solution** such as $\bar{x} = 1.41421$, which is feasible since $\bar{x}^2 \leq 2$, and it is close to optimal.

2.2 Discrete Solutions

Consider nonlinear problem (not convex)

$$\begin{array}{rl} \min & -x \\ \text{s.t.} & x^2 = 2. \end{array} \tag{2.2.1}$$

Just as before, the optimal solution is $x^* = \sqrt{2}$, which cannot be easily represented. Furthermore, the only two feasible solutions are $\sqrt{2}$ and $-\sqrt{2}$. Thus, there is no chance to write down a feasible rational approximation.

2.3 Convex NLP Harder than LP

Convex NLP is typically polynomially solvable. It is a generalization of linear programming.

Convex Programming:
Polynomial time (P) (typically)

Given a convex function $f(x)\colon \mathbb{R}^d \to \mathbb{R}$ and convex functions $f_i(x)\colon \mathbb{R}^d \to \mathbb{R}$ for $i = 1, \dots, m$, the *convex programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \le 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{2.3.1}$$

Example 2: Convex programming is a generalization of linear programming. This can be seen by letting $f(x) = c^\top x$ and $f_i(x) = A_i x - b_i$.

2.4 NLP is harder than IP

As seen above, quadratic constraints can be used to create a feasible region with discrete solutions. For example

$$x(1-x) = 0$$

has exactly two solutions: $x = 0, x = 1$. Thus, quadratic constraints can be used to model binary constraints.

Binary Integer programming (BIP) as a NLP:

NP-Hard

Given a matrix $A \in \mathbb{R}^{m\times n}$, vector $b \in \mathbb{R}^m$ and vector $c \in \mathbb{R}^n$, the *binary integer programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \le b \\ & \cancel{x \in \{0,1\}^n} \\ & x_i(1-x_i) = 0 \quad \text{for } i = 1, \dots, n \end{aligned} \tag{2.4.1}$$

2.5 Karush-Huhn-Tucker (KKT) Conditions

The KKT conditions use the augmented Lagrangian problem to describe sufficient conditions for optimality of a convex program.

KKT Conditions for Optimality:

Given a convex function $f(x)\colon \mathbb{R}^d \to \mathbb{R}$ and convex functions $g_i(x)\colon \mathbb{R}^d \to \mathbb{R}$ for $i = 1, \dots, m$, the *convex programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{2.5.1}$$

Given $(\bar{x}, \bar{\lambda})$ with $\bar{x} \in \mathbb{R}^d$ and $\bar{\lambda} \in \mathbb{R}^m$, if the KKT conditions hold, then $\bar{x}$ is optimal for the convex programming problem.

The KKT conditions are

1. (Stationary).

$$-\nabla f(\bar{x}) = \sum_{i=1}^{m} \bar{\lambda}_i \nabla g_i(\bar{x}) \tag{2.5.2}$$

2. (Complimentary Slackness).

$$\bar{\lambda}_i g_i(\bar{x}) = 0 \text{ for } i = 1, \dots, m \tag{2.5.3}$$

3. (Primal Feasibility).

$$g_i(\bar{x}) \leq 0 \text{ for } i = 1, \dots, m \tag{2.5.4}$$

4. (Dual Feasibility).

$$\bar{\lambda}_i \geq 0 \text{ for } i = 1, \dots, m \tag{2.5.5}$$

If certain properties are true of the convex program, then every optimizer has these properties. In particular, this holds for Linear Programming.

2.6 Gradient Free Algorithms

2.6.1 Needler-Mead

https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method https://youtube/NI3W1lrvWoc?t=96

Chapter 3

Material to add...

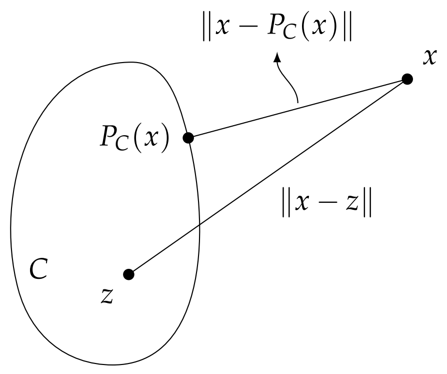
3.0.1 Bisection Method and Newton's Method

See section 4 of the following nodes: http://www.seas.ucla.edu/~vandenbe/133A/133A-notes.pdf

3.1 Gradient Descent

Recap Gradient and Directional Derivatives: https://www.youtube.com/watch?v=tIpKfDc295M
https://www.youtube.com/watch?v=_-02ze7tf08
https://www.youtube.com/watch?v=N_ZRcLheNv0
https://www.youtube.com/watch?v=4RBkIJPG6Yo
Idea of Gradient descent: https://youtu.be/IHZwWFHWa-w?t=323
Vectors: https://www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=2&t=0s

3.2 Projection Gradient Methods



h