Examples paper 2/3 notes

Nicholas Wong

1 Golden sections

Golden sections is an instance of *interval search*, which is a gradient-free method to find a local minimum of a function with one parameter. For multivariate functions, interval search can be used to find the local minimum along a certain search direction.

All interval search methods require 4 function evaluations at each iteration. For ease of exposition, we can restrict our attention to $I_1 = [x_1, x_4] = [0, 1]$ without loss of generality (because we can rescale the problem). From here, we need to evaluate the function twice more at x_2, x_3 such that $0 < x_2 < x_3 < 1$. Then, compare $f(x_2)$ and $f(x_3)$. If $f(x_2) > f(x_3)$, let the new interval be $[x_2, 1]$. If $f(x_2) < f(x_3)$, the new interval should be $[0, x_3]$. (If $f(x_2) = f(x_3)$, then it doesn't matter which interval we choose)

The question is then: where to place the points x_2 and x_3? The idea here is that we want to choose x_2 and x_3 such that a function evaluation from the previous iteration can be *reused*. That is, suppose that we reduced our interval from $[x_1, x_4]$ to $[x_2, x_4]$. The new interval would need two new test points, such that $x_2 < x_5 < x_6 < x_4$.

Now, suppose we simply place our points at 1/3 and 2/3 of the way in between, so $x_2 = 1/3$, $x_3 = 2/3$. Thus,

$$\begin{aligned} x_5 &= x_2 + \frac{x_4 - x_2}{3} = \frac{5}{9} \\ x_6 &= x_2 + \frac{2(x_4 - x_2)}{3} = \frac{7}{9} \end{aligned} \tag{1}$$

Neither of these points match any points from the previous iteration. The idea of Golden sections is that

$$\begin{aligned} x_2 &= \frac{3 - \sqrt{5}}{2} \approx 0.382 \\ x_3 &= \frac{\sqrt{5} - 1}{2} \approx 0.618 \end{aligned} \tag{2}$$

Then, you will find that $x_5 = x_3$, which means that we can reuse $f(x_3)$ for $f(x_5)$. Why does this work? Try to recall why from the figure in the lecture notes.

2 Recap on Simplex algorithm

To make sense of the simplex algorithm, recall that we make some fundamental (correct) assumptions about the nature of the solution of a *linear program*. Suppose that the number of decision variables is n and there are m equality constraints, where $m < n$[1]. In addition to the equality constraints, we implicitly have

$$x_i \geq 0 \tag{3}$$

for all $i = 1, 2, ..., n$. If the cost function is linear and constrains affine, this is what we call a linear program (where we by convention always assume a minimisation problem without loss of generality). For all linear programs, the following holds:

[1] What happens if $m \geq n$?

- The feasible set is the faces/edges/vertices (i.e. the boundary) of a (convex) polygon in n dimensions (aka a *polytope*).
- The solution to the linear program is at one of the vertices. At each vertex, m variables are positive (the *basic variables* and $n-m$ are zero (the *free variables*).

The goal of the simplex method is two-fold:

1. Get a feasible solution that is on a vertex, i.e. find one $\mathbf{x}$ such that m variables are positive and $n-m$ are zero. This is called *Phase 1* of the simplex method. [2]
2. Hop from one vertex to another adjacent one by walking along an edge that connects the current vertex to a new vertex. In this process, the objective function should decrease. This is *Phase 2* of the simplex method.

The mechanics of the simplex method in both phases are similar, involving the basic maneuvers of:

- Identifying an entering variable from the free variables. This is the variable we start to increase away from zero, determining which edge we choose to traverse among all possible ones emanating from the current vertex.
- Identifying the leaving variable. This is the basic variable that first hits zero upon increasing the entering variable. Geometrically, once the leaving variable hits zero, we've hit a vertex along the edge we are walking along.

So,

1. *Which is the entering variable?* The one which decreases the objective function the fastest when we increase it. This is given by the *reduced cost* (and we want to pick the most negative one among the free variables).
2. *Which is the leaving variable?* The one which has the smallest positive ratio when you divide the final column entry of the tableau with the entry in the constraint equation corresponding to the entering variable. Each row gives you a ratio corresponding to one basic variable. Try to convince yourself why this works (This has a lot to do about how we *standardise* the tableau).
3. *How do we know we are done?* When all reduced costs are positive, such that no edge gives us further reduction in the cost function.
4. *What about inequality constraints?* Convert them to equality constraints via slack variables.
5. *What's special about Phase 1?* In phase 1, we are solving a partially different problem. Suppose the original problem is

$$\begin{aligned} \text{minimise } & f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \\ \text{subject to } & \mathbf{a}_i^T\mathbf{x} = b_i, \quad i = 1,2,...,m \\ & x_j \geq 0, \quad j = 1,2,...,n \end{aligned} \tag{4}$$

The phase 1 problem should be

$$\begin{aligned} \text{minimise } & \sum_i q_i = \mathbf{1}^T\mathbf{q} \\ \text{subject to } & \mathbf{a}_i^T\mathbf{x} + q_i = b_i, \quad i = 1,2,...,m \\ & x_j \geq 0, \quad j = 1,2,...,n \\ & q_k \geq 0, \quad k = 1,2,...,m \end{aligned} \tag{5}$$

[2] Food for thought: It seems that an obvious way of bypassing this seemingly troublesome step is to just randomly pick a subset of basic variables and declare the rest as free variables. Think about why this may not be the best idea.

When all the q's (the *auxiliary variables*) are driven to 0, which should be the minimum of the auxiliary cost function $\mathbf{1}^T\mathbf{q}$, we have arrived at a vertex point lying on the feasible set, from which we begin Phase 2. Good thing about Phase 1 is: We are already given a feasible starting point on the vertex of the constraint polytope for (5). You just need to do the simplex maneuvers to finish the job.

3 Question 3 in constrained optimisation

Here's my take on this. Let the production over the next 3 months be quantified by

$$\mathbf{x} = [x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23}, x_{31}, x_{32}, x_{33}]^T, \tag{6}$$

where x_{ij} is the amount produced in month i to satisfy the demand in month j. The cost function is

$$f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} = [16, 19, 22, 23, 17, 20, 30, 24, 18]^T\mathbf{x}, \tag{7}$$

subject to

$$\begin{aligned} x_{11} + x_{21} + x_{31} &= 150 \\ x_{12} + x_{22} + x_{32} &= 200 \\ x_{13} + x_{23} + x_{33} &= 250 \\ x_{11} + x_{12} + x_{13} &\leq 200 \\ x_{21} + x_{22} + x_{23} &\leq 200 \\ x_{31} + x_{32} + x_{33} &\leq 200 \\ x_{ij} &\geq 0. \end{aligned} \tag{8}$$

From discussions in supervisions, it seems that one can think about it logically and arrive at an intuitive solution... but in general the best way of solving this is through network optimisation, which is not in the syllabus anymore.

4 Penalty and barrier functions

The idea of these methods is to replace a constrained optimisation problem with an unconstrained one, but we inflate the function value at places in the input space where the original problem is infeasible. The rationale is that a minimisation algorithm will then prefer places where the function value is not inflated (hence tend to be lower).

How does one decide on how intensely one should penalise infeasible points? If the penalty is too mild, that does not do much to discourage searching in the infeasible areas. However, too severe of a penalty and we produce a function where steep gradients are present (i.e. the gradients are *ill-conditioned*—small changes in the input causes a very large change in function values, bad news for curve fitting methods and gradient search methods!).

A clever way of negotiating this trade-off is:

- Begin with a mild penalty, optimise to get a minimum which likely is quite far from the desired minimum (because the distortion to the original problem is large).
- Using this distorted minimum as a warm start, incrementally increase the penalty magnitude and optimise again. Even if the penalty is steep at some places, hopefully the warm start allows the optimisation algorithm to avoid these areas.