Unit Topics

- Linear Programming
  - Graphical solution
  - Solution by enumeration
  - Simplex method
  - Dual form of a linear program
- Integer Programming
  - Cutting planes method
  - Branch and bound method
- Nonlinear Programming
  - Convexity
  - Golden Section Search
  - Downhill Simplex
  - Gradient Descent
  - Conjugate Gradient
  - Newton's Method and the Levenberg-Marquardt Extension
  - Lagrange Multipliers
  - Penalty functions
  - Dynamic Programming

Optimization Theory and its Applications

Jerome Wynne

18 November 2017

Linear Programming

A linear program is a problem where a linear function must be optimized with respect to a set of inequality constraints, i.e.:

$$\begin{aligned} \max \quad & \mathbf{c}^T\mathbf{x} = z \\ \text{such that} \quad & \mathbf{a_1}^T\mathbf{x} \leq b_1 \\ & \vdots \\ & \mathbf{a_n}^T\mathbf{x} \geq b_n \\ & \mathbf{x} \succeq 0 \end{aligned}$$

An example problem might be a version of the Italian Job. Imagine you find yourself in a truck hanging - perilously - over the edge of a cliff, and must quickly remove a fixed mass of precious metals from the truck's bed if you are to pull back onto the road. These metals are valuable, however, and you want to retain the combination of them that will yield the highest payoff. Given that: the metal's masses are x_1, x_2, and x_3; their respective values per unit mass are c_1, c_2, and c_3; and that the overall retained mass must not exceed b_1 kilos, what linear program represents your dilemma?

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + c_3x_3 = z \\ \text{such that} \quad & x_1 + x_2 + x_3 \leq b_1 \\ & x_1,\ x_2,\ x_3 \geq 0 \end{aligned}$$

A Linear Program in Standard Form

The form of a linear program affects how we solve it - whether we are maximizing or minimizing, and what type of constraints the problem is subject to, both effect the approach we take. For this reason, we convert our problem into a *standard form* so that our solution technique is consistent between problems. A linear program in standard form has several characteristics:

- The objective, which must be linear, is to be maximized.
- The constraints are linear equalities.
- The decision variables are all greater than or equal to zero.

Converting a given program to standard form is reasonably straightforward. There are several possible ways in which a program might deviate from standard form:

- The objective needs to be minimized.
- Some of the decision variables may be able to take on negative values.
- The constraints may be inequalities rather than equalities.

The first of these potential problems can be resolved by maximizing the negative of the objective function. If z must be minimized by altering $x_1, ..., x_n$, then the same values of x_i will yield a maximum of $-z$.

Why must the decision variables be positive?

The second problem, where we may have variables that can take on positive values, can be resolved by decomposing the negative variable into two positive variables. For example, let x denote a real-valued variable to be decomposed. Let $x^+ = \max(x, 0)$ and $x^- = |\min(x, 0)|$. Then

$$x = x^+ - x^-$$

Since only x^+ *or* x^- can be non-zero, it is necessary to introduce a constraint to reflect this,

$$x^+ + x^- = |x|$$

To resolve the third problem, we first ensure that all inequalities are 'less than or equal to': To flip the direction of g.t. inequalities, we simply multiply both sides through by -1. For example, the inequality

$$x_1 + x_2 \geq b_1$$

reflects the same constraints as

$$-(x_1 + x_2) \leq -b_1$$

Now that the constraints are all l.t. inequalities, we can introduce non-negative *slack variables* that convert them into equality constraints. For simplicity, consider the inequality

$$x_1 + x_2 \leq b_1$$

To this, we can introduce the slack variable $s_1 \geq 0$ such that:

$$x_1 + x_2 + s_1 = b_1$$

Using these three techniques - converting min to max, signed variables to unsigned ones, and inequalities to equalities - we can

arrive at a linear programme in standard form, i.e. one of the type:

$$\begin{aligned} \max \quad & \mathbf{c}^T\mathbf{x} = z \\ \text{such that} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \succeq 0 \end{aligned}$$

Geometric Interpretation of a Linear Programme

A linear programme in standard form is a set of constraints upon the vector $\mathbf{x}$. For each constraint introduced, one degree of freedom is lost, relative to a maximum of n such that $\mathbf{x} \in \mathcal{R}^n_+$. To make this really clear, consider a set of two linear equations in three variables x_1, x_2, x_3,

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 &= b_1 \\ x_2 \quad + \quad x_4 &= b_2 \end{aligned}$$

We can re-write this to demonstrate that several variables can be specified in terms of other variables:

$$\begin{aligned} x_4 &= \frac{1}{2}\Big(b_1 - b_2 - x_1 - x_3\Big) \\ x_2 &= b_2 - x_4 \end{aligned}$$

Notice that it's only necessary to specify x_1 and x_3 for x_4 and x_2 to be known. Alternatively, we could derive x_1 and x_3 just by specifying x_2 and x_4. If we were to start talking in terms of sets, then we would be considering the set:

$$\{\ \mathbf{x} = (x_1, x_2, x_3, x_4) : x_i \geq 0, \text{ and our constraints are satisfied }\}$$

But what would this set look like? To answer this question, consider the constraints in isolation: Each of them defines a $n-1$ dimensional hyperplane. Since all of the constraints must be satisfied at once, we only retain the points at the intersection between these hyperplanes. Provided that the two hyperplanes aren't collinear, their intersection will form a set that has one fewer dimension than the planes.

We can see that the intersection of our two hyperplanes will have one fewer dimension than the hyperplanes themselves by re-expressing the set of points satisfying the first inequality. Let (y_1, y_2, y_3) such that $x_4 = b_1 - y_1 - y_2 - y_3,\ x_1 = y_1,\ x_2 = y_2,\ x_3 = y_3$ in the original set. We then can represent our movement around the first plane using (y_1, y_2, y_3). To constrain movement in this coordinate system using the second inequality, we need to specify that inequality in terms of the new coordinate system:

$$y_1 + y_3 = b_1 - b_2$$

This allows us to reduce the number of coordinate we're using one degree further by the observation that:

$$z_1 = b_1 - b_2 - y_3$$
$$z_2 = y_2$$

Where z_1 and z_2 are set independently of one another. And there we have it - the intersection of our two 3-dimensional hyperplanes forms a 2-dimensional hyperplane - a line.

So that's a framework for understanding the equality constraints in $\mathbf{A}$, but we also have $\mathbf{x} \succeq 0$. This inequality truncates those hyperplanes at the o in all axes. Now, the resulting plane fragment could be in one of two states: It could be leant against the axes, or it could point away from the origin, extending endlessly into magnificent nothingness.

Dramatics aside, that plane fragment's orientation affects the nature of the optimal solution. If the plane is oriented such that it spans out into space without ceasing, then it's possible that the objective vector can be increased without bound. By contrast, if the plane is abutted by the axes, then one of its corners will correspond to the optimal solution (provided that the plane's normal vector is linearly independent of the objective vector).

With the problem in standard form and a geometric understanding firmly in hand, let's look at a method for solving a linear program.

Solution by Enumeration

For small linear programs - ones with few decision variables - we can find a solution by listing all of the corner points, and their associated objective values. If we have m constraints and n variables, then we can set $n - m$ variables to zero, and deduce the remaining variables values. If those values are all greater than or equal to zero, then we know that the point is feasible. This approach locks in on points that are corners of the simplex. Note however, that

$$\binom{n}{n-m} = \frac{n!}{(n-m)!\, m!}$$

solutions must be enumerated.

The Simplex Method

The simplex method is a technique for solving a linear program in standard form. The method has several steps:

1. Determine an initial basic feasible solution.

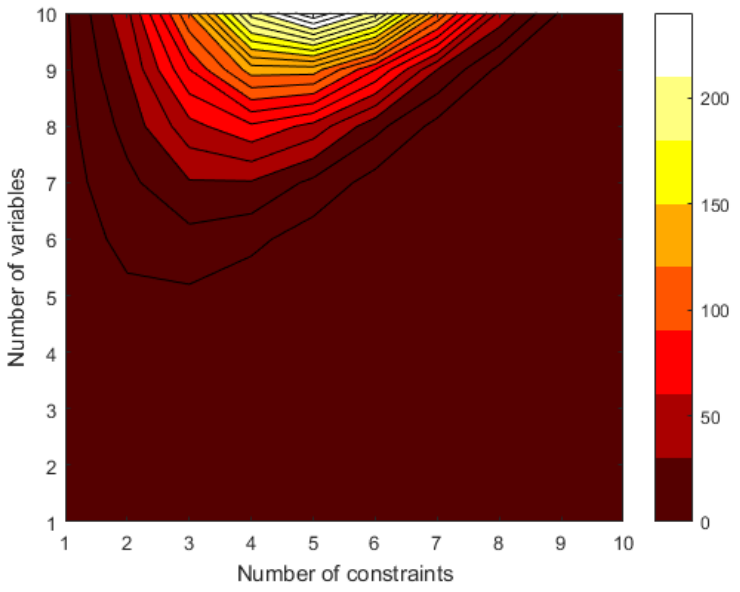

Figure 1: Growth in the number of points that must be enumerated with a problem's number of constraints and variables.

2. Introduce an entering variable, and maximize it until it pushes one another variable to zero.
3. The variable pushed to zero is known as the leaving variable.
4. Repeat steps 2 and 3 until the objective function can be maximized no further.

We'll demonstrate this using an example. Consider the LPP

$$\begin{aligned} \max \quad & \mathbf{c}^T\mathbf{x} && = z \\ \text{s.t.} \quad & \begin{bmatrix}\mathbf{A} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{x} \\ \mathbf{s}\end{bmatrix} && = \mathbf{b} \\ & \mathbf{x},\ \mathbf{s} && \succeq 0 \end{aligned}$$

Where $\mathbf{I}$ is an $m \times m$ identity matrix (i.e. all constraints were originally inequalities, and we've introduced m slack variables).

To apply the simplex method, we first need a initial basic feasible solution. In the case we've set up this is straightforward: Simply initialize the slack variables as being equal to each element of $\mathbf{b}$, and set $\mathbf{x}$ to zero. If we had fewer slack variables than constraints (i.e. some of the original constraints were equalities), then we'd need to trial values of $\mathbf{x}$ for which a subset was set to zero, and the remaining non-zero values were positive.

Collapsing the objective function into a constraint $z - \mathbf{c}^T\mathbf{x} = 0$, we

can we-write this problem as:

$$\begin{bmatrix} 0 & \mathbf{A} & \mathbf{I} \\ -1 & \mathbf{c}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} z \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ 0 \end{bmatrix}$$

Where we've collapsed the slack variables $\mathbf{s}$ into the vector $\mathbf{x}$. To solve this equation for the optimal $\mathbf{x}$, we begin by writing it in augmented form:

$$\left[\begin{array}{ccc|c} 0 & \mathbf{A} & \mathbf{I} & \mathbf{b} \\ -1 & \mathbf{c}^T & \mathbf{0} & 0 \end{array}\right]$$

This form is known as the initial simplex tableau for the LPP. A dictionary for a linear programming problem defines a subset of variables x and z in terms of the remaining variables, and has the same solution set as the initial dictionary. The variables defined are known as basic variables, whereas the variables they're defined in terms of are known as non-basic. Every dictionary identifies a particular solution to the linear system obtained by setting the non-basic variables equal to zero, yielding basic variables that form a *basic feasible solution* if they are non-negative.

The simplex algorithm moves from one feasible dictionary representation of a system to another while increasing the value of z. It does this by increasing one of the non-basic variables from its value of zero. The amount by which we can increase a non-basic variable is constrained by the current dictionary we have, and which variable is pushed to zero first. All that remains is to rearrange the intial dictionary such that the entering variable is basic, and the leaving variable is non-basic. Having done this, the entering variable must then be eliminated from the other equations in the dictionary by substituting its definition into the other elements. .

See the University of Washington's Maths Department's *Solving LPs: The Simplex Algorithm of George Dantzig*

By considering the new dictionary's expression for z, we can determine which variable should enter next. We can again consider what variable leaves, then re-arrange the dictionary to reflect the change in basis. Once z's expressions contains only terms with negative coefficients, we know that we've reached a maximum.

1. Propose an initial basic feasible solution.
2. Set up an initial dictionary for that solution.
3. Identify the entering and leaving variables.
4. Rearrange the initial dictionary to yield a new basic feasible solution.
5. Repeat (2) - (4) until the coefficients for z in the dictionary are all negative.

In Gaussian elimination, we derive the combination of basic vectors in one space that is necessary to yield a specific combination of basic vectors in another space under a given transformation.

To derive this particular combination of vectors, we consider how the basis of the destination space might be re-expressed. We work towards a unit basis by adding and subtracting rows of the transformation's matrix. Eventually, by changing the basis of the destination space, we wind up with an identity mapping. This implies that we've applied a step-wise inversion of the linear mapping. If we apply it to the r.h.s. of the equation too, then we yield the combination of the domain's basis vectors necessary for the desired transformation.

Applying Gaussian elimination to the augmented matrix: Each step of Gaussian elimination corresponds to one simplex pivot. Identify the pivot column by considering the coefficients in $\mathbf{x}$ for z. Identify the pivot row by writing out the ratio between the values in $\mathbf{b}$ and the pivot column, then selecting the row with the smallest value. Use row elimination to make the column/row combination unitary.

It's worth pointing out that an optimization can feasibly have redundant constraints, or constraints that prevent there from being a solution. Either of these situations must occur when there are more constraints than decision variables, but they might occur when the situation is reversed. To test whether a given $m \times n$ matrix $\mathbf{A}$ contains a redundant constraint, perform Gaussian elimination on $\mathbf{A}$: If, under row elimination, a row of $\mathbf{A}$ is zeroed, then the corresponding element of $\mathbf{b}$ *must* be zero, otherwise the equation is nonsense. If it *is* zero, then that constraint must have been redundant.

Duality

The solution to the dual of a linear programming problem is also a solution to the original problem. Example of dual statements:

- Two points determine a line.
- Two lines determine a point.

The idea here is that we can define one entity in terms of another, or another entity in terms of the first. Here's another example I made up:

- 'I' denotes who I am.
- I denotes what 'I' is.

See hand-written notes for an example.

The classic primal-dual pair of linear programs is as follows. Define the primal problem as the what we're accustomed to, with the

exception of being a minimization problem

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x} = z_x \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \ \geq \mathbf{b} \\ & \mathbf{x} \quad \succeq \mathbf{0} \end{aligned}$$

Then define the dual of this problem as that represented by

$$\begin{aligned} \max_{\mathbf{y}} \quad & \mathbf{b}^T\mathbf{y} \ = z_y \\ \text{s.t.} \quad & \mathbf{A}^T\mathbf{y} \leq \mathbf{c} \\ & \mathbf{y} \quad \succeq \mathbf{0} \end{aligned}$$

Okay, pointers:

- In the original problem, the decision variables are in $\mathbf{x}$. In the dual, they are represented by $\mathbf{y}$. How do $\mathbf{x}$ and $\mathbf{y}$ relate to one another?
- In the dual problem, $\mathbf{A}$ is transposed.
- The objective vector in the dual is the constraint of the primal, and the constraint vector of the dual is the objective vector of the primal, but with the inequality flipped.
- In the primal, we minimize: In the dual, we maximize. The dual is more recognizable here, but I suppose we could equally well flip the notation if we wanted to.

  When we get back: Do the example. Done!

Duality theorems

Weak duality theorem

Begin by letting

$$\begin{aligned} \mathbf{A}\mathbf{x} &\geq \mathbf{b} \\ \mathbf{A}^T\mathbf{y} &\leq \mathbf{c} \end{aligned}$$

Such that $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$. We can then take dot products to yield a result that makes the two expressions equivalent:

$$\mathbf{y}^T\mathbf{A}\mathbf{x} = \mathbf{x}^T\mathbf{A}^T\mathbf{y} = \text{ Scalar}$$

To see this, we need to use the fact that the transpose of a scalar is itself. Defining $\mathbf{y}^T\mathbf{b} = z_y$, $\mathbf{x}^T\mathbf{c} = z_x$, we arrive at the conclusion

$$z_y = \mathbf{y}^T\mathbf{b} \leq \mathbf{y}^T\mathbf{A}\mathbf{x} \leq \mathbf{x}^T\mathbf{c} = z_x$$

It follows that if equality holds, and $\mathbf{x}$ and $\mathbf{y}$ are both feasible, then $\mathbf{x} = \mathbf{x}^*$ and $\mathbf{y} = \mathbf{y}^*$. If we assume that equality does hold for some

$\mathbf{x}$ and $\mathbf{y}$, then we can ask what happens when $\mathbf{x}$ is altered such that $\mathbf{x}^T\mathbf{c}$ decreases: This would imply that $\mathbf{y}^T\mathbf{A}\mathbf{x}$ decreases, which further means $\mathbf{y}^T\mathbf{b}$ must decrease. The opposing line of reasoning can be used to show that $\mathbf{x}$ is at its optimal value.

This theorem presumes that those $\mathbf{x}$ and $\mathbf{y}$ are feasible, however. How we can we be sure that they are? Does the feasibility of a solution to one problem imply something about the feasibility of the other?

Duality Theorem

For any dual pair of linear programming problems such that a finite optimal solution exists for one problem, the objective functions of both problems have an identical optimal value.

$$\min\ z_x = \max\ z_y$$

Complementary slackness theorem

If $\mathbf{x}$ and $\mathbf{y}$ are feasible solutions to a primal-dual pair of linear programming problems, then both are optimal if and only if

$$\mathbf{y}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) = 0$$
$$\mathbf{x}^T(\mathbf{A}^T\mathbf{y} - \mathbf{c}) = 0$$

This can be proved directly from the duality theorem. This theorem implies that, at the optimum of our two problems, either $(\mathbf{A}\mathbf{x})_i - \mathbf{b}_i = 0$ or $\mathbf{y}_i = 0$.

If the optimal solution to the primal problem has been found by the simplex method, then the optimal value of the i^{th} primal variable is found in the objective function row under under the slack variable corresponding to the i^{th} constraint. The surplus variables in the primal problem given the negative of the dual variable.

Integer Programming

An integer programming problem in which all variables are integers is called a pure integer programming problem. If some variables are restricted to be integers, and others not, then the problem is said to be a *mixed integer programming problem*.

Example 1: Capital budgeting

Assume that you have £14,000 to invest. How should you distribute it across the following investments if you wish to maximize your return?

Investment 1 £5,000 up-front, £8,000 return.

Investment 2 £7,000 up-front, £11,000 return.

Investment 3 £4,000 up-front, £6,000 return.

Investment 4 £3,000 up-front, £4,000 return.

Let x_i be an indicator variable indicating whether investment i is made. It follows that our problem can be expressed as

$$\begin{aligned} \max \quad & 8x_1 + 11x_2 + 6x_3 + 4x_4 = z \\ \text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \end{aligned}$$

To test feasible solutions, we can permute through each choice of investment: We can see that the last three investments yield the highest

x_1	x_2	x_3	x_4	z
1	1	0	0	19
1	0	1	1	18
0	1	1	1	21
⋮	⋮	⋮	⋮	⋮
0	0	0	1	4

return, however they also use the entirety of the available funds.

Example 2: Multiperiod Capital Budgeting

Now imagine we must choose among projects that require investments of different amounts in each of several periods, with different budgets available in each period. We wish to invest £14,000, £12,000, £15,000 in each month of the next quarter. The investments opportunities available to use are:

Investment 1 Invest £5,000, £8,000, £2000 in each month, and receive an overall return of £8000.

Investment 2 Invest £7,000 in month 1, £10,000 in month 3, and receive a return of £11,000.

Investment 3 Invest £4,000 in month 2, £6,000 in month 3, and receive a return of £6,000.

We have specific resources for each period, each period has specific demands, and the return is only received if we invest across all periods. Denoting each investment using an indicator variable, we can represent the periodic investments as constraints, and the overall

returns as an objective

$$\begin{aligned} \max \quad & 8x_1 + 11x_2 + 6x_3 = z_x \\ \text{s.t.} \quad & 5x_1 + 7x_2 \leq 14 \\ & 8x_1 + 4x_3 \leq 12 \\ & 2x_1 + 10x_2 + 6x_3 \leq 15 \\ & x_i \in \{0, 1\} \end{aligned}$$

Example 3: The Knapsack Problem

The knapsack problem is a classic integer programming problem. If we have a rucksack of a fixed size, and several items to put in it each of a distinct utility, which items should we choose to put in the bag? Let s be a constant represent the bag's size, and let $\mathbf{c}$ be a vector containing the integer utility of each item. let $\mathbf{a}$ be a vector containing the sizes of each item, and $\mathbf{x}$ be a vector containing indicator variables specifying whether an item is or is not in the bag. The problem is then the program

$$\begin{aligned} \max \quad & \mathbf{c}^T\mathbf{x} = u \\ \text{s.t.} \quad & \mathbf{a}^T\mathbf{x} \leq s \end{aligned}$$

Example 4: Fire Station Problem

A city council is trying to determine in which neighborhoods to place fire stations. The neighborhoods have distinct boundaries, and a fire station in one neighborhood can attend to fires in all adjacent neighborhoods, including its own. If the council wants to minimize the number of fire stations it uses, where should it place the stations?

This type of problem is known as a *set covering problem*. We can represent it by letting x_i be an indicator variable specifying whether a station is placed in neighborhood i. We want to minimize the sum over x_i, and can represent the full-coverage constraint by considering neighborhood adjacency. For example, in 2, we can specify that neighborhoods 11 & 10 fire-stopping needs must be met by the constraints:

$$\begin{aligned} x_9 + x_{10} + x_{11} &\geq 1 \\ x_8 + x_9 + x_{10} + x_{11} &\geq 1 \end{aligned}$$

Example 5: The Travelling Salesman Problem

A travelling salesman must visit 20 cities before returning home. He knows the distance between each pair of cities and wishes to mini-

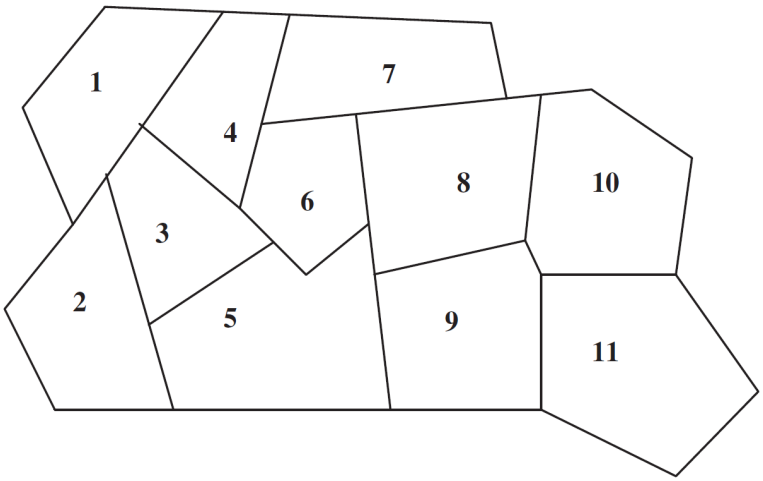

Figure 2: An archetypal domain for a set covering problem.

mize the total distance travelled. In what order should the salesman visit the cities?

Denote the cost of travelling between city i and city j as c_{ij}. Let x_{ij} be a variable indicating whether the salesman uses the route from city i to city j. The constraints on this problem are that

$$\sum_{j=1;j\neq i}^{n} x_{ij} = 1 \quad \text{for } 1 \leq i \leq n$$

i.e. Each city is only departed from once, and

$$\sum_{i=1;i\neq j}^{n} x_{ij} = 1 \quad \text{for } 1 \leq j \leq n$$

i.e. Each city is only arrived at once. These constraints are sufficient for the route to be connected.

The objective is to minimize the overall cost of the route

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Solving Integer Programming Problems

One of three methods are commonly used to solve integer programming problems:

- Enforcing constraints that yield integrality - the *cutting planes* method.
- Dynamic programming - splitting the problem into smaller subproblems via *branch and bound*.

- Linear relaxation - convert the program into a linear programming problem, then solve for an approximate solution.

Relaxing an Integer Program

We can convert an integer program into a linear program by dropping the integrality restriction on the decision variables, i.e.

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x} && = z \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} && = b \\ & \mathbf{x} \geq 0, \in \mathbb{Z} \end{aligned}$$

becomes

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x} && = z \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} && = b \\ & \mathbf{x} \geq 0, \in \mathbb{R} \end{aligned}$$

Solving the relaxed form of an integer programming problem is useful: Its solution provides a bound on the integer cost, and gives an approximately optimal solution to the true integer problem.

Branch and Bound

Branch and bound is a method for solving integer programming problems. We will explain it in the context of the following problem:

$$\begin{aligned} \max \quad & 8x_1 + 11x_2 + 6x_3 + 4x_4 && = z \\ \text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 && \leq 14 \\ & x_i \in \{0,1\}, \quad i = 1,\ldots,4 \end{aligned}$$

The linear relaxation solution of this problem is $x_1 = 1$, $x_2 = 1$, $x_3 = 0.5$, $x_4 = 0$, with a return of $z = 22$. The integer solution can therefore not have a value of more than 22. To force x_3 to be an integer, we branch on it: We create two new problems, one in which we constrain $x_3 = 3$, and one in which we constrain $x_3 = 1$. We solve both of these, then if their solutions are still not integers we branch them again. To choose which problem to branch

- Choose a sub-problem that has not been chosen before.
- Choose the sub-problem with the largest (or smallest) solution value.

Branch a variable that is not an integer, keeping the previous constraints intact. Stop branching when an integral solution is obtained. Only when all branches have terminated, either through integrality of

infeasibility of their solution, can we evaluate which solution is optimal. It's possible to pursue an entire branch, obtain a feasible integral solution, then compare it with the value of a fractional solution at the root of another branch (fathoming).

In summary, branch and bound is applied as follows

- Solve the linear relaxation of the problem. If the solution is an integer, return the solution. Otherwise, create two new subproblems by branching on a fractional variable - constrain its value to possible integers.
- A subproblem is not active when any of the following occur:
  - The subproblem was previously branched on.
  - Its solution consists of integers only.
  - The subproblem has an infeasible solution.
  - The subproblem can be fathomed by a bounding argument.
- Choose an active subproblem and branch on a fractional variable. Repeat until there are no more active subproblems.

Note that if you're using non-binary integers, you should branch on inequality constraints adjacent to the fractional variable (e.g. if $x = 1.23$ then the branch subproblem constraints would be $x \leq 1$ and $x \geq 2$).

Non-linear Programming

The non-linear optimization techniques will cover how to solve programs in which both the objective funciton and the constraints are non-linear. We can categorize nonlinear optimization algorithms as either

- gradient-free or gradient-based,
- constrained or unconstrained,
- and with either continuous or discrete decision variables.

The methods we will cover are as follows.

- Direct search using line search in one dimension - a gradient-free, continuous, unconstrained algorithm.
- Downhill simplex - an algorithm that is also gradient-free, continuous, and unconstrained.
- Steeped descent - gradient-based, continuous, unconstrained.

- Conjugate gradient methods - gradient-based, continuous, unconstrained.
- Newton's method - gradient-based, continuous, unconstrained.
- Gauss-Newton - gradient-based, continuous, unconstrained.
- Lagrangian multipliers - gradient-based, continuous, unconstrained.
- Penalty function methods - gradient-based, continuous, unconstrained.
- Dynamic programming - gradient-free, discrete, constrained.

Definitions of Optimality

We'll use several definitions of optimality in this part of the course. A *strict global minimum*, x^*, of a function is defined as

$$f(x^*) < f(y) \qquad \text{for all } y \neq x^* \in V(x), \text{ the set of all feasible values.}$$

A *global* minimum permits equality. We may wish to talk about minima in some region of a function, as opposed to globally. To be able to do this, we introduce a *strong local minimum* y^*

$$f(y^*) < f(y) \qquad \text{for all } y \in N(y^*, \eta), y \neq y^*$$

Where $N(y^*, \eta)$ is the set of all points within distance η of y^*. A *weak* local minimum is defined similarly, with the inequality relaxed.

Testing the nature of stationary points

Define a stationary point x_s of the function $f(x)$ such that $\nabla f(x_s) = \mathbf{0}$, where ∇ denotes the element-wise derivative operator. The *nature* of a stationary point tells us what f's surface is like in the vicinity - $f(x_s)$ may be some form of optima, or may alternatively be a point of inflection, or further still a saddle point.

We can determine the nature of a stationary point by evaluating the Hessian of f at x_s. 'Hessian' is shorthand for the matrix of second derivatives,

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Consider a maximum of $f(x)$ first, either local or global. At that point, the Hessian of f must evaluate as being strictly positive definite, i.e. it must satisfy

$$u^T H u > 0 \qquad \text{for all } u \neq \mathbf{0} \in \mathbb{R}$$

Equivalently, the eigenvectors of H must all be strictly positive.

Now consider if x_s corresponds to a minimum. Then H must be a negative-definite matrix, such that

$$u^T H u < 0 \qquad \text{for all } u \neq \mathbf{0} \in \mathbb{R}$$

This is the same as saying that all eigenvalues of H must be negative.

As an alternative to determining H's eigenvectors, we can test the sign of each of its upper left sub-matrix determinants.

Convex sets and convex functions

A convex set is any set X that satisfies the condition

$$\lambda x_1 + (1-\lambda)x_2 \in X$$

for all $x_1, x_2 \in X, \lambda \in [0,1]$. A convex function is any function for which the line between two points on its graph sits above the function itself, i.e. $f(x)$ is convex if and only if

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2) \quad \lambda \in [0,1]$$

Operations that preserve convexity include

- non-negative linear combinations of convex functions,
- and affine compositions.

We can test the convexity of a function using the Hessian matrix: Provided that it is positive definite over the entire set of feasible points, the function is convex. By contrast, if the Hessian is negative definite over the domain then $f(x)$ is concave.

$$H = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}$$

It is possible that a function is neither convex or concave, and is instead both over specific ranges.

If the function and set under study are both convex, then a local minimum must also be a global minimum.

Direct Search in One Dimension

Golden section line search uses function calls to locate the minimum of function in one variable. The algorithm is as follows:

1. Take a triplet (a, b, c) and evaluate f at a point x between b and c.
2. If $f(x) > f(b)$ then form the new triplet (a, b, x).
3. If $f(x) < f(b)$ then the new triplet is (b, x, c).

4. Repeat 2 and 3 until the distance between the first and third elements of the triplet is below some tolerance.

Select x such that if the ratio of the two lengths $b - a$ and $c - a$ is w,

$$\frac{b-a}{c-a} = w$$

then the next point x should be a fraction 0.38 into the larger of the two intervals, as measured from the central point of the triplet.

Nelder Mead and Introduction to Gradient Descent

Today we're covering:

- Downhill simplex - an unconstrained algorithm for M-dimensions.
- An example of downhill simplex being run on Himmelblau's function.
- An introduction to gradient descent.

Nelder-Mead or *downhill* simplex is also known as the amoeba algorithm. A simplex is a geometric object with $n + 1$ vertices in n dimensions together with planar faces connecting these vertices. Why use downhill simplex?

- It is robust, and will sometimes succeed where other methods fail.
- It does not use gradients, only function calls.
- The method is geometric and simple to use.
- It 'crawls downhill with few assumptions'.

The idea of the downhill simplex algorithm is that it starts with a set of $N + 1$ points defining a simplex that encloses the minimum. The values of the function are then evaluated at all vertices of the simplex, and are put into order. A series of reflections, expansions, or contractions of the simplex are pursued which consistently move the simplex away from the highest-valued regions towards the lowest-valued regions. This sequence of maneuvers will always converge on the minimum of the function.

1. For a function in two variables, evaluate at three points x_1, x_2, x_3 to form a simplex (i.e. a triangle here).
2. Reorder the points' indices such that $f(x_1) < f(x_2) < f(x_3)$.
3. Calculate the centroid $x_0 = \frac{x_1+x_2}{2}$.

4. Try a new point by *reflecting* away from the worst, $x_r = x_0 + (x_0 - x_3)$ (See figure below).

5. If f(x_1) $\leq$f(x_r) $\leq f(x_2)$, then replace x_3 with x_r to form a new set of three points (x_1, x_r, x_2).

6. If $f(x_r) \leq f(x_1)$, then *extend* the reflection as $x_r' = x_0 + \lambda(x_r - x_0)$, where x_0 is an extension scaling factor (usually $\lambda = 2$). If $f(x_r') < f(x_r)$, then the new triple is (x_1, x_2, x_r'); otherwise the triple is (x_1, x_2, x_r).

7. If $f(x_r) \geq f(x_2)$, *contract*. Form $x_c = x_0 + \rho(x_3 - x_0)$, with $\rho = 0.5$.

SLIDE 14 of LECTURE 2

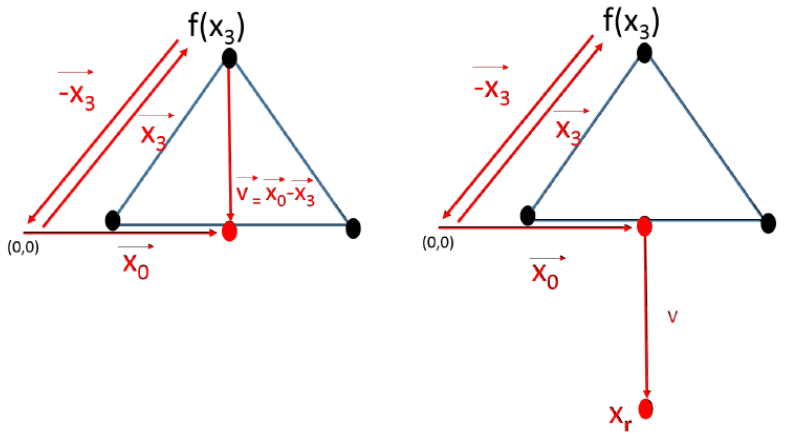

Figure 3: Looking down onto the 2D x-plane, the reflection step of the downhill simplex method.