Convex optimization

Simplices

- Set of convex combinations of vectors $x^1, \ldots, x^n = \{\lambda_1 x^1 + \ldots + \lambda_n x^n : \lambda_1, \ldots, \lambda_n \geq 0 \text{ and } \lambda_1 + \ldots + \lambda_n = 1\}$ — this corresponds to the set of vectors in the space enclosed by the line segments x^1 to x^2, …, x^1 to x^n, x^2 to x^3, …, x^2 to x^n, …
  - We denote these x^1, x^2 (don't confuse with $(x)^1$, $(x)^2$) as x_1, x_2 are reserved for the components of x
- The 0-simplex is a point
- The 1-simplex is the set of convex combinations of 2 linearly independent 1D vectors i.e. a line segment
- The 2-simplex is the set of convex combinations of 3 linearly independent 2D vectors i.e. a triangle
- The 3-simplex is the set of convex combinations of 4 linearly independent 3D vectors i.e. a tetrahedron
- The n-simplex is the n dimensional shape obtained by the set of convex combinations of n+1 linearly independent n dimensional vectors

Convex sets

- **A set X is convex iff $\forall x^1, x^2 \in X.\ \forall \lambda \in [0,1].\ \lambda x^1 + (1-\lambda)x^2 \in X$** or equivalently for every pair of vectors in X the line segment between them is a subset of X
- It can easily be shown that if A and B are convex then $A \cap B$ is convex but $A \cup B$ is not necessarily convex
- *The set of feasible solutions to an LP is convex but the set of feasible solutions to an IP is not in general convex*

Convex functions

- **A function f(x) is a convex function on a set X iff $\forall x^1,x^2 \in X.\ \forall \lambda \in [0,1].\ f(\lambda x^1 + (1-\lambda)x^2) \leq \lambda f(x^1) + (1-\lambda)f(x^2)$ or equivalently for every pair of points on f(x) the line joining them is not below the curve f(x) at any point between them**
- **For strictly convex we replace ≤ with < and not below at any with above at every**
- **f(x) is (strictly) concave iff −f(x) is (strictly) convex**
- **A function may be neither concave nor convex** *but will be able to made to be by a suitable domain restriction*
- *A function is both concave and convex iff it is linear*
- If f(x) is a convex or increasing function on a convex set X, $\forall k.\ \{x: f(x) \leq k\}$ is a convex set
  - To see this deduce that in the case $X \subseteq \mathbb{R}$ the set is an interval
- **Any linear combination with <u>all non-negative coefficients</u> of convex functions is a convex function**

Multivariable differentiation, and differential conditions for convexity

- **When generalising to multivariable functions the first derivative is a vector** of partial derivatives
- **When generalising to multivariable functions the second derivative is a matrix (called the Hessian matrix of the function)** of partial derivatives of partial derivates
  - **f_{ij}'' = partial derivative of (partial derivative of f with respect to j) with respect to i and is element ij in matrix** — is symmetric, $f_{ij}'' \equiv f_{ji}''$
- **A minor of a matrix A is a principal minor iff it was obtained by deleting rows and columns with the same index**
  - Recall minor is <u>determinant</u> of matrix obtained by deleting <u>(possibly no)</u> rows and/or columns
- **A** (symmetric) **matrix is positive semi-definite iff all its principal minors are non-negative**
- **A minor of order k of an nxn matrix A is a leading principal minor iff it was obtained by keeping only the first n-k rows and columns of A**
- **A** (symmetric) **matrix is positive definite iff all its leading principal minors are <u>strictly</u> positive**
- **A** (symmetric) **matrix A is negative (semi-)definite iff −A is positive (semi-)definite**
- **f(x) is convex on X iff Hessian is positive semi-definite $\forall x \in X$**
- **If Hessian is positive definite $\forall x \in X$, then f(x) is strictly convex on X <u>but the converse does not hold</u>** e.g. $f(x)=x^4$ is strictly convex but $f''(x) = 0$ at $x=0$
- **$f'(x)$ = the zero vector $\Leftrightarrow$ is a stationary point**
  - **$f''(x)$ positive definite** (f strictly convex) **at a stationary point $\Rightarrow$ is a local minimum**
  - **$f''(x)$ negative definite** (f strictly concave) **at a stationary point $\Rightarrow$ is a local maximum**
  - **Otherwise, inconclusive so test values around stationary point**
  - *This is very similar to single variable functions*

Non-linear optimization

- **We can generalise linear programming (LP) to mathematical programming (MP):**
  min f(x)
  S.t. $g_i(x) \leq 0 \ \forall i \in \{1, \ldots, m\}$
  $x \in \mathbb{R}^n$
  No requirement for f or g_i to be linear or x to be non-negative anymore
- Neighbourhood of a point x with radius R = **U(x, R) = ball centred at x with radius R**

Let F be the feasible region then:

- **x is an interior point iff $\exists R > 0$: $U(x, R) \subseteq F$**
- **x is a boundary point iff $x \in F$ and x is not an interior point**
  - *In an LP optima are boundary points*
- **If $\exists R > 0$: $\forall x \in U(x^*, R) \cap F$. $f(x^*) \leq f(x)$, then x^* is a locally optimal solution**
- **If $\forall x \in F$. $f(x^*) \leq f(x)$, then x^* is a globally optimal solution**
- **If $f''(x^*) > 0$ and $f'(x^*)=0$, then x^* is a locally optimal solution**
- **An MP problem in the above canonical form is called convex iff F is convex and f and g_i are convex on F**
- **If an MP is convex, then local optima are global optima** e.g. in an LP local optima are global optima
- **If x^* is an interior point and the MP is convex and $f'(x^*)=0$, then x^* is a globally optimal solution**
  - *Recall from single variable functions that exterma are either turning points or endpoints*
- If an MP is convex and f is strictly convex on F, then the optimal solution is unique if it exists

Integer programming

Constructing IPs

- As when constructing any LP, all variables must be defined
- **$x_i = \neg y_j \Rightarrow x_i = 1 - y_j$**
- **$x_i = y_1 \vee \ldots \vee y_j \Rightarrow jx_i \leq y_1 + \ldots + y_j$ and $x_i \geq y_1, \ldots, y_j$**
- **$x_i = y_1 \wedge \ldots \wedge y_j \Rightarrow x_i \geq y_1 + \ldots + y_j - j + 1$ and $x_i \leq y_1, \ldots, y_j$**
- **If $y_j = 1$, then $x_i \geq k \Rightarrow x_i \geq k - M(1-y_i)$**
- **If $y_j = 1$, then $x_i \leq k \Rightarrow x_i \leq k + M(1-y_i)$**
- At least k of $f_i(x_1, \ldots, x_n) \leq d_i$ hold $\Rightarrow f_i(x_1, \ldots, x_n) \leq d_i + M(1-y_i)$ and $\Sigma y_i \geq k$ where y_i are auxiliary variables introduced for this purpose
- $f_i(x_1, \ldots, x_n) \in \{d_1, \ldots, d_j\} \Rightarrow f_i(x_1, \ldots, x_n) = d_1y_1 + \ldots + d_jy_j$ and $y_1 + \ldots + y_j = 1$ where y are auxiliary variables introduced for this purpose
- Although M is in principle arbitrarily large, we in practice use the smallest sufficient M to save on computation
  - M = max { min { b_i/a_{ij} over j} over i} for canonical form (max with i^{th} constraint sum over j of $a_{ij}x_j \leq b_i$)

Set cover/partition as IP

- *SET-COVER is NP-hard and we are going to demonstrate a reduction from it to IP*
- SET-COVER: Call $S \subseteq 2^X$ a covering of X iff $X = \cup x$ over $x \in S$. Let c be a function from 2^X to the reals. Find a covering S^* s.t. total cost of S^* is minimal
- Let m=|X| and define a bijection between X and {1, …, n}
  Let $T = 2^X$ with an arbitrary ordering and |T|=n
  Let $[A]_{mxn}$ be the matrix s.t. $d_{ij} = 1$ if $i \in T[j]$, 0 otherwise — constant
  Let $c_i = c(x_i)$ — constant
  Define binary decision variable x_j=1 if $T_j \in S$, 0 otherwise
  min $z=c_1x_1+...+c_nx_n$
  s.t. $a_{i1}x_1+...+a_{in}x_n \geq 1$ for all i
  $x_1, \ldots, x_n \in \{0, 1\}$
- SET-PARTITION: SET-COVER but each member of X is in exactly one member of S
  - Replace ≥ 1 with = 1 in the IP

Solving IPs: Branch and bound

- **BB(prob, largest_LB)**
  sol = lp_solver(lp_relax(prob))
  if problem was infeasible then return ⊥
  UB = z in sol
  feas_sol = round_to_feasible(sol, prob)
  LB = z in feas_sol
  If UB == LB then return feas_sol
  If LB > largest_LB then largest_LB = LB
  if UB < largest_LB then return ⊥
  Pick arbitrary x_j in sol that is non-integral
  Let β be the value of x_j in sol
  return max_z {BB(prob ∪ {$x_j \leq$ floor(β)}, largest_LB), BB(prob ∪ {$x_j \geq$ ceil(β)}), largest_LB}
- Using largest_LB is called fathoming and is not necessary for correctness but allows us to prune fruitless branches and is considered part of the canonical algorithm
- We can multiply UB by some slightly smaller than 1 constant before comparing to LB for a speed up if we don't need a perfect solution
- **For a canonical (i.e. max) problem, as we go down the tree LB (non-strictly) increases and UB (non-strictly) decreases**

Solving IPs: Branch and cut

- **Branch and cut:**
  1. Fix variables
  2. Eliminate redundant constraints
  3. Generate and add cutting planes
  4. Use branch and bound on simplified problem
- Fix variables — find which variables can only take one value based on the constraints and rewrite these as the appropriate constants
- Eliminate redundant constraints — while there exists a constraint that can be removed without altering feasible region { remove that constraint }
- Cutting planes alter the feasible region without removing any feasible solutions by adding extra constraints that are implied by the limited set of values the variables are allowed to range over

Solving IPs: Branch and cut: BIPs

- In this module we only cover branch and cut on pure binary IPs
- **For a ≤ constraint: If sum of most positive coefficient and all negative coefficients > RHS then variable with most positive coefficient can be fixed to 0**
  - **Keep on applying this rule until the condition is no longer true**
- For a ≥ constraint: If sum of most negative coefficient and all positive coefficients < RHS then variable with most negative coefficient can be fixed to 0
  - Keep on applying this rule until the condition is no longer true
- **For a ≤ constraint: If constraint holds when all variables with positive coefficients are 1 and all others are 0 it can be removed**
- For a ≥ constraint: If constraint holds when all variables with negative coefficients are 1 and all others are 0 it can be removed
- **Cutting planes:**

  Write in canonical form (all constraints ≤)

  for each constraint with all non-negative coefficients

  Find a set of variables V (if one exists) s.t. if all in V are 1 and all other variables are 0 the constraint is violated but if all but (any, universal quantification) one variable in V are 1 and all other variables are 0 the constraint is satisfied

  Add constraint sum of variables in V ≤ |V| − 1

  Do this for all such sets (including subsets of ones already done)

Solving IPs: Dynamic programming

- Let x_i = i^{th} decision variable
- **Let c_i = capacity remaining after decisions 1 to i−1 have been made — c_1 = total capacity available**
- **Let $f_i(x_i)$ = value of x_i + max $f_{i+1}(x_{i+1})$ s.t. capacity constraints**
  - **To find the second term: read off in the previous table (table for x_{i+1}) the max cell of the row with the capacity remaining after x_i**
- **Tabulate for c_n, x_n then c_{n-1}, x_{n-1} … then c_1, x_1**
  - x_n will likely be straightforward as we will probably want to use all the remaining capacity we can

c_i =	$f_i(x_i)$ when x_i =		max	argmax
	C	D		
A	E	F	max E, F	C and/or D
B	G	H	max G, H	C and/or D

Network optimization

Networks, and minimum cost network flow

- Network = a directed weighted graph
- In a flow network each node has a (possibly negative) supply and each arc has a capacity (maximum flow) and a cost (per unit flow)
  - Positive capacity = supply node
  - Negative capacity = demand node
  - Zero capacity = transshipment node
- Minimum cost network problem: Decide the flows through each arc to minimize total cost s.t. net flow through each node = supply of that node and flows are non-negative and don't exceed capacities
  - May be IP or LP depending on the context for the flows
- Shortest path problem can be reduced to IP minimum cost flow by taking capacities to be unbounded, start to have supply 1, end to have supply -1, and all other nodes to have supply 0, and edge weights to be flow costs
  - We use Dijkstra because it is tractable and IP isn't

Dijkstra's algorithm

- Create a bijection from nodes to naturals 1…
- **As dynamic programming: OPT(i) = 0 if i=0, min { OPT(j) + w_{ji} | 0 ≤ j ≤ i } otherwise**
- Algorithm:

```
Input G = (N, A)
Update(i)
    Let A(i) = set of all arcs with source i
    For j ∈ {j | (i, j) ∈ A}
        if d(j) > d(i) + c_ij
            d(j) := d(i) + c_ij
            pred(j) := i
d(1) := 0
pred(1) := ⊥
d(j) = +∞ for j=2,...
update(1)
S := {1}
T := N \ {1}
While S ≠ N
    Pick i from T s.t. d(i) is minimal
    update(i)
    S := S ∪ {i}
    T := T \ {i}
```

- Can show Dijkstra as a DP style table with an * next to a node once it has moved to S

Minimum spanning tree algorithms

- MINIMUM-SPANNING-TREE: Given an undirected weighted graph find a spanning (same node set as the graph) tree (connected acyclic subgraph) with minimal total cost
- Kruskal's algorithm: Start with T empty. while T is not spanning add an arc that does not introduce a cycle to T with minimal weight to T
  - *Cycles can be (efficiently) detected using dfs*
- **Prim's algorithm: Start with T as an arbitrary node. While T is not spanning add an arc from a node in the tree to a node outside the tree with minimal weight to T**
  - This is much the same greedy approach as Kruskal but the anti-cycling is implicit this time