("Optimisation" theoretical vs "OR" practical)

Optimisation:

1. Identify problem (general)
2. Formulate (specific) & assess viability
3. Observe system (find out constraints etc)
4. Meet parties

} Phase 1 (items 1–4)

5. Mathematical model e.g. LP
6. Preparations: check alg., pilot study, clean data, etc
7. S/w + s/w tests
8. Solve!

} Phase 2 (items 5–8)

9. Check solutions
10. Sensitivity analysis // sub-optimal but more practical solutions?

} Phase 2a (items 8–10)

11. Present results (present 3 options / "sell" one)
12. Implement chosen soln // evaluate // modifications needed?
13. Follow-up etc

} Phase 3 (items 11–13)

- **Hard vs** Soft: course focuses on "hard" but also need soft
- **Deterministic** vs stochastic: course focuses on deterministic
- Goal (Hard, det): maximise objective function / subject to constraints

Choosing software: "crucial" to pick the right algorithm for large and/or complex problems. Speed? Differentiability? Etc. Try different methods. Try different starting values. DON'T just plug in default

Factors to consider:

Problem	Software
problem size	speed
structure	price
requirements e.g. sensitivity analysis	ease of use
For NLPs: starting point,	vendor support, training
good algo important! (diff. etc?)	compatibility

1 Linear Programming

Two variables: graphical solution $\rightsquigarrow$ varying z gives a family of parallel lines

Standard form LP:

$$\begin{aligned} \max\ \mathbf{c}^T\mathbf{x} \text{ s.t.} \\ A\mathbf{x} \leq \mathbf{b}; \\ \mathbf{x} \geq 0 \end{aligned}$$

$\rightsquigarrow$ **Standard form**

- ***feasible***: $\mathbf{y}$ is feasible if $A\mathbf{y} \leq \mathbf{b}$ and $\mathbf{y} \geq 0$ and
- ***optimal***: if it is feasible and maximises $\mathbf{c}^T\mathbf{x}$;
- ***feasible region*** is set of all feasible vectors
- ***value*** of LP is maximum value for $z = \mathbf{c}^T\mathbf{x}$

❀ LP in standard form – three options:

1. Infeasible
2. Feasible but unbounded
3. Unique solution

❀ Solution (of LP) never in interior of feasible region $\rightarrow$ always at a vertex. But there can be a lot of vertices... $\left[\text{up to } \dfrac{(m+n)!}{m!\,n!}\right]$

❀ Simplex: logical approach for moving between vertices

Algorithm 1 Simplex algorithm

Write in standard form

$\Longrightarrow$ Convert to slack form (creates an "identity structure")

$\Longrightarrow$ Write as tableau

CHECK feasibility // preliminary pivots if necessary

LOOP:

(a) ID column with most negative value in bottom row

(b) ID row with min {RHS/entry} (only consider entries > 0) $\Longrightarrow$

(c) PIVOT by adding multiples of the pivot row to each target row in turn

(NOT more general row ops)

UNTIL all bottom row coeffs are non-neg

General idea: if $RHS > 0$ and tableau has identity structure ("basic variables"), this corresponds to a ***basic feasible solution***. From feasible solution pivot to new, better, feasible solution.

Initial tableau may not correspond to a BFS: use preliminary pivots.

If: (1) row entries are all 0 and RHS $\neq 0$, or

(2) row entries are all negative and RHS is positive (or vice versa)]

$\Longrightarrow$ infeasible!

If: b.r. entry is negative but no plausible pivot

$\Longrightarrow$ unbounded!

1.1 Simplex: beyond the basics

Degeneracy and cycling

Degeneracy: sequence of pivots transforms some b_i to 0 $\iff$ some basic variable = 0

$\rightsquigarrow$ Generally NBD

Cycling $\implies$ degeneracy // Degeneracy $\not\Rightarrow$ cycling

If cycling occurs (rare!): change pivot rule. If problem still not solved, perturbation method.

Initialisation

Recall "basic feasible solution" = Identity structure, with bottom coeffs zero; all RHS > 0. Can force the ID structure (slacks) but may land up with -ve RHS (or converting to > 0, lose ID structure) $\implies$ *initialisation*: two options covered:

1. Big M

$$\max x_1 + bx_2 \ldots + kx_n \to \max\ x_1 + bx_2 \ldots + kx_n - MR$$

Include R in constraint i where the usual slack variable comes up negative (for +ve RHS b_i):

$$C + \ldots - s_i \ldots = +b_i \to \qquad C + \ldots - s_i \ldots + R = +b_i$$

M assumed to be very large

R replaces s_i as "basic variable", but the R column has M on the bottom row, so pivot it out (one step: subtract M times R row from bottom row)

$\rightsquigarrow$ Tableau now in BFS form: solve: if R ends up in the basis, no FS to original problem

2. Two-phase

1. Replace objective function with artificial variable R, to be minimised (s.t. constraints)

2. Simplex: if $R = 0$, no feasible solution; else, take solution with R as starting point for orig. problem. (= delete R col, refill bottom row, now in BFS form)

✓ Big-M simpler

✗ But harder to implement by computer ("very big" × small number → FP errors etc.)

$\rightsquigarrow$ two-phase more common

❀ (Either) can be used to determine existence of FS

1.2 More cool stuff: Duality

Primal	$\leftrightarrow$	Dual
A	$\leftrightarrow$	A^T
$\mathbf{b}$	$\leftrightarrow$	$\mathbf{c}$
$\leq$	$\leftrightarrow$	$\geq$
max	$\leftrightarrow$	min

dual of eq. constraint $\rightarrow$ free var

(constraints only; $\mathbf{x} \geq 0 \rightarrow \mathbf{y} \geq 0$)

1 primal variable : 1 dual constraint
(⇝ two constraints: two-variable dual $\Longrightarrow$ graphical solution possible!)

Various results:

1. Dual of dual is original (proof: easy)
2. Weak duality: if $\mathbf{y}$ is feasible for the dual (=min) and $\mathbf{x}$ for the primal (=max), $\mathbf{c}^T\mathbf{x} \leq \mathbf{b}^T\mathbf{y}$
   $\Longrightarrow$ value of objective function for any FS to primal is lower bound for minimum value of dual
   $\Longrightarrow$ if primal is feasible but unbounded, dual is not feasible
   $\Longrightarrow$ if primal and dual both feasible, then they are both bounded
3. Could also have both infeasible (proof by e.g.)
4. Strong duality: if one problem has an optimum, so does the other and it's the same (*no proof*)
   ⇝ *(Corollary)* Options: (1) Both feasible&bounded, with same optimal solution
   (2) Feasible&unbounded // infeasible
   (3) Both infeasible
5. Complementary slackness:
   if constraint has strict inequality at optimum (slack var is non-zero, constraint is "slack"),
   the matching variable of dual is zero
   $\Longleftrightarrow$ if variable is non-zero, then matching constraint is equality (*proof by algebra*)
   [remember: always only as many non-zero vars as constraints]
   ⇝ can use to solve hard LPs where dual is easy
   [solution already gives us optimum; equality constraints give us simultaneous equations]

Always sanity-check results!

1.2.1 Dual simplex

❀ use when: all bottom row coefficients non-negative // RHS has some negative entries
❀ typically for: updating a solution (new constraint, changes to parameters)
❀ drive to make all RHS non-negative (then done)

Algorithm 2 Dual simplex

```
LOOP:
    Pick row with most negative RHS
    If all column entries in this row are ≥ 0 → INFEASIBLE
    Pick negative column entry for bottom-row/row ratio closest to 0 [analogous to regular]
     ⟹ PIVOT
```

1.3 Sensitivity analysis

"Changes in production" **[force value of a variable x_i or slack s_i]**

(Works same whether forcing "units of x_i to be made" (x_i or "units of resource to be left over" (s_j))

(a) x_i not part of identity structure (not a "basic variable"):

$\rightsquigarrow$ would be 0 in optimal solution;

- Take column for this var, multiply by required value, glue to RHS (subtract)
- RHS still ≥ 0: still optimal, no change;

  otherwise, may have to pivot to get RHS ≥ 0 (e.g. dual simplex)

(b) x_i in the identity structure ("basic variable"):

- Look at constraint line where $x_i = 1$: other basic variables are 0 on this line, so equation works out to $1.x_i + bn_1 + cn_2 + \ldots = C$ where the n_j are non-basic variables, i.e. normally 0
- reset x_i to required value and assume we will change *one* of the n_j to be > 0

  [can show algebraically this is always best, proof not included]
- Test each n_j of appropriate sign (multiply new n_j by the value of this var on z line and subtract from optimum), to see which one has least negative effect on objective function (*reality check: modifying the solution returns less optimal result!*)
- Glue chosen column (& multiplier) into RHS column (subtract) to see effect on other basic vars

"Changes in resources" **[change RHS]**

- Find slack variable for resource (constraint) being changed (e.g. constraint 3, variable s_3)
- DON'T remove column from tableau, but DO glue it (same sign) to RHS, multiplier a (amount of change)
- Is tableau still optimal? If so, done; if not, pivot time (dual simplex, since non-optimal will mean something on RHS < 0

"Changes in selling prices" **[change bottom line]**

- Replace bottom row entry (of soln) with change amount
- Pivot into identity-matrix form

  *(= do nothing if q was not in basic var, else add q * (that var's row) to bottom line)*
- q unspecified: whether this tableau is optimal will depend on q, can read off range for which it's optimal (i.e. for which z row is all ≥ 0): in this range optimum vector stays same and can see effect of q on optimum value
- q specified: if outside the range above, may need to pivot further to find new optimal form

“New constraints” [what it says on the tin]

1. Compare constraint with current optimal solution: if constraint is already met, done
2. Otherwise, add constraint to tableau, pivot (= first pivot constraint out of basic var cols, hopefully this gets into dual simplex form)

1.4 Interior point methods

[No detail] IP/Simplex: both iterative, both start from feasible solution

☆ Alternative for (usually) large problems

⇝ “polynomial” time (vs simplex worst case exponential)

⇝ but one I PT iteration is longer than one simplex iteration

☆ Convergence criterion, “close to” optimum (cf gradient descent)

⇝ convergence criterion not always ideal: duality gap to assess proximity to optimum (remember at optimum primal and dual have same value)

✗ No handy tableau for post analysis

☆ Possibility of combining with simplex for final stage

❀ Tricks:

⇝ transform/scale feasible region (keep current iterate near centre $\implies$ ensures large steps)

⇝ barrier function $\implies$ penalty for points close to boundary of feasible region

⇝ but uses log term = non-linear!

Klee-Minty

$$\begin{aligned} \max\ & x_d \\ \text{s.t.}\ & 0 \le x_1 \le 1 \\ & \dots \\ & \epsilon x_{i-1} \le x_i \le 1 - \epsilon x_{i-1} \\ & \dots \\ & \epsilon x_{d-1} \le x_d \le 1 - \epsilon x_{d-1} \end{aligned}$$

❀ known to require $2^d - 1$ simplex iterations ⇝ exponential

1.5 Quadratic programming: Lemke

Can write in form:
$\min \frac{1}{2}\mathbf{x}Q\mathbf{x} + \mathbf{c}^T\mathbf{x}$ s.t.
$A\mathbf{x} \leq \mathbf{b}$; $\mathbf{x} \geq \mathbf{0}$

☆ Problem: is a quadratic; constraints: linear

☆ Objective function is convex

❀ "quite restrictive but does occur quite often"

(*Like LPs, useful not just for quadratic problems but often an approx. for more complex NLPs*) (Taylor series!)

Algorithm 3 Lemke

Init: Get into standard form, i.e. set up $Q, A, \mathbf{c}, \mathbf{b}$

$= n$ variables ; m constraints

0. Check objective function is convex (principal minors of Q)
1. Build tableau (see below) y, v form an identity matrix structure: **basic** variables
2. Start in z column; pick row with most -ve entry in constant column, pivot so z becomes basic
3. LOOP:

    3a. Find variable that left basic structure. If z, then STOP.

    Else: $\implies$ it has a complement:. Identify *the complement*

    3b. Pivot on this complement ("minimum ratio rule") [RHS/col: min +ve].

    If no pivot possible, also STOP (*no solution*)

Lemke tableau:

$\mathbf{x}$	$\mathbf{u}$	$\mathbf{y}$	$\mathbf{v}$	$\mathbf{z}$	
$-Q$	$-A^T$	I_n	0	$(-1\ \ -1\ldots\ \ -1)^T$	$\mathbf{c}$
A	0	0	I_m	$(-1\ \ -1\ldots\ \ -1)^T$	$\mathbf{b}$

x_i, y_i are complementary pairs; u_j, v_j are complementary pairs

✓ (No proof) will terminate (providing obj. function is convex);

✓ efficient

❀ can use quadratic approximations of more general problems to use this method as efficient approximation technique

Quadratic programming: Lemke

2 Integer Programming

- Pure or mixed
- Often (mixed) binary
- No universal algo
  - $\rightsquigarrow$ Can't (necessarily) just round LP solution
  - $\rightsquigarrow$ Bounded $\implies$ finitely many solutions ... but might be impractically many!
  - $\rightsquigarrow$ Some "good" approaches but none known that are not exponential
  - $\rightsquigarrow$ More specific approaches for specific problems

❀ Neat trick: use NLP constraint $x = x^2 \implies x \in \{0, 1\}$

2.1 Strategies

General tool = ☆ branch-and-bound ☆

But sometimes, can find shortcuts:

- Small problems: exhaustive enumeration
- Heuristics, (a) as "good enough" solution (b) to whittle down possibilities

  $\rightsquigarrow$ If problems from the real world are being posed because we want a solution in the real world, can apply real-world common sense!

  (e.g. aiming to optimise total suitability when matching people to jobs; rule out in advance any matches with "poor" suitability)
- May be able to Use Algebra on constraints to whittle down possibilities (combine constraints, etc)

  $\rightsquigarrow$ but, relies on spotting possibilities // not general
- "Logical constraints" (build in as algebraic constraints)

  ! (not sure how this is a "strategy")
- Cutting planes:

  (1) Solve LP

  (2) If solution is optimum, done; else, find "cutting plane" separating optimum from feasible region $\implies$ new feasible region

  (3) Rinse & repeat

  [initially thought not to be efficient; more recent methods discovered making it viable]

Branch & Bound

- Solve LP ("LP relaxation").
- Pick a variable that is non-integer in the solution, so $a < x_i < a + 1$ for some integer a
- Set up two new LPs, one with new constraint $x_i \leq a$, one with new constraint $x_i \geq a+1$ (branch)
  [NB $a = 0 \implies$ new constraint is $x_i = 0$ due to non-negativity constraints]
- Solve these: each solution establishes an upper bound on objective value

---

Algorithm 4 Branch-and-bound

Initialise; set L to "any" feasible integer solution (largest known, or a large negative value)

LOOP:

$\rightsquigarrow$ Branch: set up new problems | NB: only ever 2 subproblems per node |

$\rightsquigarrow$ Bound: calculate optimum for them

$\rightsquigarrow$ Fathom: if this branch has $z < L$, or is infeasible; ditch it

else, $z > L$; if solution is integer, reset L and cut off branch here

$\rightsquigarrow$ Test: is there anywhere else to go, if not, current L ("incumbent") is solution

---

✓ Conceptually simple

✓ Can be adapted easily to NLPs

Notes:

- Choosing order of operations has major impact (e.g. breadth-first, depth-first, node orderings...)
- Might make sense to stop at sub-optimal solution [LP optimum is an upper bound: are we close?]
- Spend time on initialisation to get a good starting point
- Might be able to exploit problem structure, e.g. branch on constraints for binary problems *[TODO: check what he means by this??]*
- Can also use branch & bound for non-linear integer problems, etc.
- Can be exponential
- Strategies:

Strategies

  - if integer variable has a lot of possible values (e.g. > 20), consider treating it as continuous; try and keep down total number of integer variables
  - make upper/lower bounds on integer variables as tight as possible
  - the more constraints the better! (opposite to LPs)
  - order in which integer variables are processed is critical. choose "based on economic significance and user experience"
  - stop within 3% of continuous optimum, if allowed
  - consider whether rounded LP solution is practical

3 Graphs and networks

Terminology (for this course)

graph = (nodes + arcs /) vertices + edges

directed = digraph / undirected

multigraph (multiple edges; loops) / simple (assume unless otherwise stated)

empty → complete = K_n (n vertices)

isomorphic graphs

subgraph

spanning [subgraph, tree]

adjacent (vertices)

neighbourhood $N(v)$ of a vertex v

degree of a vertex

edge sequence → chain → *circuit/cycle*

connected vertices, graph, digraph: strongly, weakly

acyclic → tree (with leaves)

network = digraph with no loops or multiple edges & each edge has a weight/capacity, sources & sinks are identified (at least 1 of each). Assume weakly connected

cut [in network]

weighted graph [cf network capacity]

minimum spanning tree (=minimal connector)

================================

Handshaking lemma: sum of the vertex degrees is equal to twice the number of edges

[proof by double counting]

⟹ total number of odd-degree vertices in a graph is even

⟹ if several people shake hands at a party, the total number of hands shaken must be even

Result on trees (no proof) Following are equivalent:

1. G is a tree
2. any two vertices in G are connected by a unique path
3. G is acyclic with $|E| = |V| - 1$

Max-flow min-cut Theorem (no proof):

value of any max flow (in a network) equals minimum capacity of any cut

Graphs and networks

Types of problem: often could be expressed as LP but useful to exploit network structure.

3.1 Shortest path (source to sink)

[capacities/weights = distances]

Algorithm 5 Dijkstra's algorithm

1. Label source vertex
2. LOOP:
   - (a). Consider last permanently labelled vertex, say X; look at all Y adjacent to X:

     if more efficient route than the current temp label on Y (∞ if none), update temp label.
   - (b). Make vertex with shortest-dist temp label into permanent label
   - (c). If reached destination, STOP
3. Construct shortest path

3.2 Maximum flow through a directed network

Could be an LP but we can manage a lot more efficiently using max flow alg:

Algorithm 6 Maximum flow algorithm

INIT: find a feasible flow [make it as good as you can by inspection, saves time over zero flow...]

LOOP:

(define sets of edges I, R in which flow can be increased and decreased)

[just a concept: don't explicitly calculate sets]

a. find a chain source → sink by adding vertices from I or R

(never add a vertex that's already in the chain (terminology for LNO: "labelled")).

if no chain possible: STOP

b. increase flow along chain as much as possible

Current flow (before last loop, in which attempt to find chain failed) is optimal

(4. Sanity check by finding a cut)

Extensions

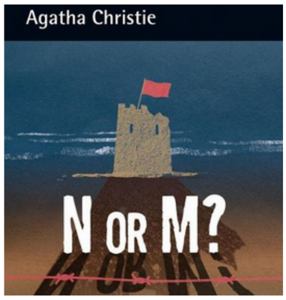
- Multiple sources and/or sinks: create artificial "supersource,sink"
- Two-way flow: add edge
- Node capacities: split node in two, insert edge
- Costs as well as capacities
- Gains/losses [e.g. electrical circuits heat, money can be taxed...]
- Contractual obligations to use certain routes

3.3 Minimum (=capacity) spanning tree of an (undirected) graph

Example problems: (1) New underground system, stations, tracks between them? (2) Central heating system / minimise piping (3) Telecomms

(a) Kruskal

Repeatedly add minimum weight edge, providing no cycles



(b) Prim

Repeatedly add min weight edge ***that links with a vertex in the tree***!

====================================

⇝ look similar, but Kruskal is $O(m \log m)$, Prim $O(n^2)$ (n nodes, m edges). Is graph sparse ($m \approx n$) or dense? ($m \approx n^2$)

[See also Comp Opt re smart algos!]

Minimum (=capacity) spanning tree of an (undirected) graph

4 Complexity

❀ usually want to minimise

☆ but in cases such as crypto might want to guarantee minimum is not too low!

Factors:
- the algorithm [key]
- the hardware
- the code
- the inputs (think of best case vs worst case ["most useful"?] vs avg case)
- constraints for space, time [time most interesting in modern world]

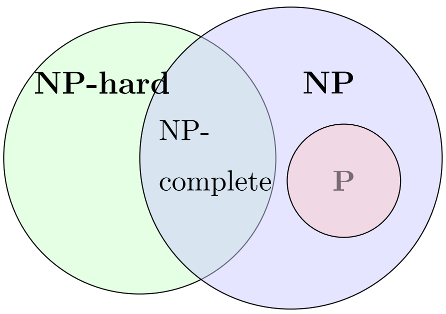
❀ runtime?

❀ memory?

❀ Time – Moore's law – Quantum?? –

⇝ proportional to steps

⇝ in terms of problem size (input parameters)



O-notation: typically

*: const → log → linear → quadratic → poly (degree k) → exponential

Sum of functions: take the fastest growing one, drop the rest

For this course: worst case;

P: Polynomial time // NP : Check in P time [else: is it exponential]

[may also need to know: class U: Undecidable]

NP-hard [any NP problem can be transformed into it in P time] // NP-complete [NP, and NP-hard]

5 Non-linear programming

5.1 Convexity / concavity

Set

Def: Convex = points on line segment are in set

Def: Concave = not convex [nothing more]

☆ intersection of convex sets is convex (proof by algebra)

☆ union of convex sets is not necessarily convex (proof by example)

☆ a hyperplane in $\mathbb{R}^n$ divides the space into two convex sets (proof from definition)

☆ feasible region for an LP is convex (proof from definition as intersection)

Function

Def: (strictly) Convex = $f(c\mathbf{x}_1 + (1-c)\mathbf{x}_2)\{\leq, <\}cf(\mathbf{x}_1) + (1-c)f(\mathbf{x}_2)$ $0 \leq c \leq 1$

Def: (strictly) Concave = $f(c\mathbf{x}_1 + (1-c)\mathbf{x}_2)\{\geq, >\}cf(\mathbf{x}_1) + (1-c)f(\mathbf{x}_2)$ $0 \leq c \leq 1$

☆ A function f is convex $\iff$ $-f$ concave (proof direct from defs)

☆ A linear function is both convex and concave (proof from definitions)

☆ f, g convex $\implies$ $f + g$ convex

Univariate: if $f''(x)$ exists for all x in a convex set S then

☆ $f(x)$ is a convex function $\iff$ $f''(x) \geq 0$ for all $x \in S$

☆ $f(x)$ is a concave function $\iff$ $f''(x) \leq 0$ for all $x \in S$

(Notice S is convex both times)

Multivariate

Compute Hessian $\frac{\partial^2 f}{\partial x_i \partial x_j}$ [NB symmetric]:

☆ f is concave if $H(\mathbf{x})$ is negative semidefinite for all $\mathbf{x}$

☆ f is strictly concave if $H(\mathbf{x})$ is negative definite

☆ mutatis mutandis for convexity

☆ The following statements about a *symmetric* matrix A are equivalent:

1. A is positive semidefinite
2. All eigenvalues of A are nonnegative
3. $A = ZZ'$ for some real matrix Z

$\implies$ we can go from eigenvalues or other linear algebra methods to definiteness of H and thus to convexity/concavity

5.2 Principal minors & co

- ❀ ith principal minor: determinant of an $i \times i$ submatrix (can be several ith principal minors)
- ❀ kth leading principal minor: delete the last $n \times k$ rows/columns
- ❀ given a multivariate function f, H_k: kth *leading* principal minor of the Hessian

Theorem: Assume f has continuous second order derivatives. [test f : concave?]

1. f is convex on S $\iff$ for all $\mathbf{x}$, all principal minors are non-negative
2. f is concave on S $\iff$ for all $\mathbf{x}$, all kth non-zero principal minors have the same sign as $(-1)^k$

$\rightsquigarrow$ NB: function can be neither!

- ❀ Stationary points: could be local maxima, local minima, ... or saddle points

Theorem: n-variable problem, $k = 1, \dots n$ [test nature of stat pt]

1. if $H_k(\mathbf{x}) > 0$ for all k then $\mathbf{x}$ is a local minimum
2. if $H_k(\mathbf{x}) \neq 0$ and has the same sign as $(-1)^k$ for all k, $\mathbf{x}$ is a local maximum
3. if $H_n(\mathbf{x}) \neq 0$ but neither 1 nor 2 applies, $\mathbf{x}$ is not a local extremum 4. if $H_n(\mathbf{x}) = 0$, no conclusions can be drawn.

Theorem: [local $\leftrightarrow$ global]

If (1) NLP is a *maximisation* and (2) the feasible region S is a convex set:

Objective function f_0 concave on S $\implies$ any local maximum is an optimum

(Proof by contradiction)

Corollary: [*See also:* f, all g convex $\implies$ KKT pt is global opt]

If the NLP is a *minimisation*, S still *convex*:

Objective function f_0 convex on S $\implies$ any local minimum is an optimum

---

NLPs:

- ❀ max $f_0(\mathbf{x})$ s.t. $f_i(\mathbf{x}) \leq 0$ (note formulation as ≤ 0 for all constraints)

[Can always get into this form $\to$ but in fact need min for most algs!]

vs LPs

- feasible region has generally curved boundaries
- optimum not necessarily at vertex
- not necessarily at boundary at all
  (e.g. with $x^2 + y^2 \leq 1$, opt at (0,0))
- might be local optimum but not global
  [necessary vs sufficient...]
- LPs = special case! Other special cases can also be exploited
- potentially multiple disconnected feasible regions
  (e.g. $\sin(x) \geq \sin(x + \pi)$)

Classification

- univariate vs multivariate
- constrained vs unconstrained
- exact vs approx [methods]

$\rightsquigarrow$ $2^3 = 8$ categories

5.3 Univariate

(*constrained/unconstrained: usually reduce to constrained = find interval of interest*)

Points to check:
- endpoints of interval
- does derivative exist everywhere?
- is $f'(x) = 0$ solvable?
- distinguish max/min & global/local

❀ Can typically find an interval with optimum in [but careful about optima at ends...]

❀ May be subproblems to multivariate methods

❀ Basic solution is to look for $f'(x) = 0$

Proposed strategy:

1. if $f'(x)$ doesn't exist in many places or is hard to solve for zero, use numerical method; else

2 (a) plot curve to get a basic idea

(b) Evaluate f at (i) local optima by differentiation; (ii) points of non-differentiability; (iii) end-points $\implies$ choose optimum from (i), (ii), (iii)

$\rightsquigarrow$ ✗ not always possible // (i), (ii), (iii) means *ad-hoc* methods

$\rightsquigarrow$ more commonly: Approximate methods

- for computer implementation
- to a required degree of accuracy
- point vs. interval

☆ NB: always consider rate of convergence!

E.g., point method: Newton

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

❀ continue until $|x_{n+1} - x_n| < \epsilon$

✓ conceptually simple

✓ easy to implement

✓ can converge fast

✗ f ☆ must be twice differentiable ☆ — can fail if $f'' \approx 0$

✗ might converge to local but not global optimum

✗ badly behaved functions (could diverge or wander)

Univariate

E.g., interval method: line search

☆ Start with sketch!

☆ Divide interval in half each time; consider $f'(x_n)$ (x_n is division point)

☆ Continue until interval is small enough (*can take* $f \approx \frac{1}{2}(f(a) + f(b))$ *as point sol.*)

✓ simple

✓ easy computation

✗ slow convergence ($\log_2\left(\frac{a-b}{\epsilon}\right)$ divisions)

✗ f must be differentiable

✗ need a single optimum in the interval

5.4 Multivariate

1: Unconstrained

Exact methods

$\nabla f(\mathbf{x})$ the gradient vector of first partial derivatives

$\implies$ a system of n equations when all are zero

Solve to find stationary points

Then need to determine nature of stationary points (use thms above)

...
✓ simple
✓ can often find local optima
...

✗ often not applicable
✗ need differentiability
✗ can be hard/impossible to solve simultaneous eqn.

Approximate methods

Newton (cf univariate) for two-variables ("Obvious extensions for $n > 2$ variables")

[string of algebra = derivation]

$$\mathbf{x}_{n+1} = \mathbf{x}_n - H^{-1}\nabla f$$

At each iteration, evaluate H^{-1} and f at $\mathbf{x}_n$ ☆ Good starting point is crucial ☆

✓ Fast convergence in some cases.

✗ ☆ BOOM ☆ but "not a viable practical tool":

✗ Needs a lot of computing power ✗ If H has a singularity (between starting point and true optimum), ☆ BOOM ☆

✗ Can be badly behaved // not robust //

⇝ sensitive to starting point

⇝ can reach stationary iteration point

⇝ or get stuck in a cycle

✗ Convergence might not be to an optimum (local; saddle point...)

✗ Need:

H invertible and well conditioned

NB H may be invertible for only some $\mathbf{x}$

f twice differentiable with explicit analytic form of derivatives

✗ Remember this is an approximation method and we're discarding quadratic (& higher power) terms of a series ... but that can actually cause problems for convergence

Multivariate

❀ sooooo: Quasi-Newton $\boxed{\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha_{n+1} H_n \nabla f}$

argument to f is $\begin{pmatrix} x_1 + \alpha y_1 \\ x_2 + \alpha y_2 \\ \dots \end{pmatrix}$ where
x, y are known (prev. step) so it's an eqn in α

⇝ α_n is a 'step length'

⇝ $\{H_i\}$ is a sequence of matrices typically with $H_0 = I$

e.g.:

$H_n = (H + \lambda_n I)^{-1}$ where the λ_i are constants; BFGS; DFP (non-examinable)

steepest descent (=minimisation) (or ascent for maximisation)

⇝ $H_n = I$: $\mathbf{x}_{n+1} = \mathbf{x} \boxed{+} \alpha_{n+1} \nabla f(\mathbf{x}_n)$

⇝ α_{n+1} found by univariate search to minimise $f(\mathbf{x}_n + \alpha_{n+1} \nabla f(\mathbf{x}_n))$

(x_n known $\implies$ equation in α: differentiate and solve for zero gradient)

Usual convergence strategy:
(1) Good starting point
(2) iterate
(3) until convergence criterion
(apply to all cpts of vector $\mathbf{x}$)

☆ Zigzagging ⇝ slow convergence is often a problem

✓ Simple idea
✓ Usually converges

✗ Often slowly (compared to other approaches)
✗ Tricky computing, including univariate search
✗ Could be a *local* optimum
✗ Require differentiability

☆ Not generally recommended (except well conditioned problems) ☆

! But: many methods suffer from "tricky computing", local optima, need for differentiability

Adapt steepest ascent to fix zigzagging & slow convergence? $\implies$ bring it in line with other methods?

⇝ change step size e.g. 0.9α

⇝ modify direction e.g. $\alpha(\frac{1}{2}(\nabla f(\mathbf{x}_n) + \nabla f(\mathbf{x}_{n-1})))$ [no further discussion on these]

Multivariate

2a: Equality constrained

1. Sketch
✗ 2 variables only (possibly 3)
❀ approximate

2. Substitution ("not to be despised, it can be useful")
❀ Algebra with constraints [*equality* constrained] to get "reduced objective function"
(*constraints are sim. eqns*) ❀ Solve reduced objective function by appropriate means

3. Lagrange multipliers: the fun stuff

Algorithm 7 Lagrange multipliers

1. define L Lagrangian by munging constraints with obj. function z;
   ⇝ Constraint C is stuff $= 0$, $\boxed{L = z + \lambda(C)}$
2. Diff. w.r.t. x_i **and** λ $\implies$ system of sim. eqns (all derivatives zero)
3. Solve!
4. Check what kind of a stationary point it is...

All optima are Lagrange pts
! not all L. pts are even stat. pts!

⇊

If sol. is unique and opt. exists, have found it

2b: Inequality constrained

"The most general type of NLP"

(rearrange to) min $f_0(\mathbf{x})$ s.t. $f_i(\mathbf{x}) \leq \mathbf{0}$

Method:

- m constraints: build 2^m subproblems, each one with i of the constraints ($0 \leq i \leq m$), treated as equalities [rest ignored]
- solve the 2^m equality constrained problems [choose method from above]
- see if solutions violate other constraints (if so, bin)
- compare optimum for non-binned solutions

☆ Sloooooooooooooooooooooooooooooooooooooooow ☆

KKT conditions (*Karush–Kuhn–Tucker*):

❀ Lagrangian: $f_0(\mathbf{x}) + \sum_j u_j f_j(\mathbf{x})$ (remember the f_j ($j > 0$) are the constraints)

1.	$\frac{\partial L}{\partial x_i} = 0$	**Gradient**
2.	$u_i f_i(\mathbf{x}) = 0$	**Orthogonality**
3.	$f_i(\mathbf{x}) \leq \mathbf{0}$	**Feasibility**
4.	$u_i \geq 0$	**Non-negativity**

4 *sets* of conditions: way more than 4 things to test!

$\mathbf{x}$ is a (*local*) optimum $\implies$ all conditions are satisfied

KKT Method

- get into min f_0 s.t. $\ldots \leq 0$
- Set up a bunch of equations corresponding to KKT conditions
- Solve 'em to find local optima [typically works out as branching technique: pick one equation that narrows down options, and try these options in another equation...]
- Test to see if it's global $\implies$ is f convex?

Thm: (no proof)

$\rightsquigarrow$ if f_j is convex for all j then any such point ("KKT point") is a global minimum

$\rightsquigarrow$ a few other similar tests (not covered)

✗ but general case have to examine each point

✗ tedious

✓ still more promising than 2^m constraints method

Multivariate

5.5 Penalty and Barrier methods

5.5.1 Penalty

Move to feasible region from outside it (sequence of infeasible points)

Set up unconstrained problem $\min f(\mathbf{x}) + cP(\mathbf{x})$

Defining P:

Equality constraints: $P(x) = \sum(h(\mathbf{x})^2)$

Inequality constraints: ≤ 0: $P(x) = \sum_i (\max\{0, g(\mathbf{x})\})^2$

[square term ensures differentiable. So we are told]

$\{c_k\}$ is an *increasing sequence tending to infinity*

❀ in rare cases may be able to solve $P'(x) = 0$ analytically (then let $M \to \infty$)

Commonly: use iterative method with $\mathbf{x}_k$ as starting point for step $k+1$

> Theorem: A limit point of any sequence $\{\mathbf{x}_k\}$ generated by the penalty method (as $c \to \infty$) is a solution to the problem $\min f(\mathbf{x})$ s.t. $\mathbf{x} \in S$

5.5.2 Barrier

"Prevent" the search procedure from leaving the feasible region

Set up unconstrained problem $\min f(\mathbf{x}) + \epsilon B(\mathbf{x})$

E.g. $B = -\sum_i \frac{1}{g_i(x)}$ or $-\sum_i log\left(-g_i(x)\right))$

($g_i \leq 0$; barrier methods always feas. pt)

> Theorem: A limit point of any sequence $\{\mathbf{x}_k\}$ generated by the barrier method (as $\epsilon \to 0$) is a solution to the problem $\min f(\mathbf{x})$ s.t. $\mathbf{x} \in S$

5.5.3 Both:

✓ normally converge; handle cusps & other anomalies well

✓ easier programming (only unconstrained functions)

✗ working with more complex functions

✗ can be issues with slow convergence

Barrier vs Penalty:

B: even if you don't reach convergence, all solutions are feasible

B: typically require fewer function evaluations $\Longrightarrow$ faster

P: good with equality constraints (barrier methods are complicated)

P: barrier methods need feasible start point, could be hard to find

5.6 NLP methods: Summary

		Univariate	Multivariate
Exact		Always sketch graph!	
	constr.	$f'(x) = 0$ constraints → intervals: check: - non-diff. pts - endpts of interval	*Eq. constrained* - try subst. to create "reduced obj. fn" ❀ Lagrange multiplier method *Eq. constrained* - set up 2^m constraints (*slow!*) - KKT method: set up sim eqns...
	unconstr.	-	$\nabla f(\mathbf{x}) \implies$ set up sim. eqns - use thms to determine nature of stationary pts ✗ impractical!
Approx	*constr.*	point: = Newton: $$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$ interval = line search *lots of issues!*	Penalty or barrier $\implies$ convert to unconstrained ⇊
	unconstr.	-	Newton $$\mathbf{x}_{n+1} = \mathbf{x}_n - H^{-1}(\mathbf{x_n})\nabla f(\mathbf{x}_n)$$ ... lots of issues $\implies$ quasi-Newton $$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha_{n+1}A_n(\mathbf{x_n})\nabla f(\mathbf{x}_n)$$ here A is not (necessarily) the Hessian (inverted Hessian), e.g. could just be an identity matrix → steepest {a,de}scent. (still has issues)

6 Proofs & derivations

LP: dual $\leftrightarrow$ original

let L: primal: $\max \mathbf{c}^T\mathbf{x}$ s.t. $A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0$

$$\begin{aligned} L^*(\text{dual}) &= \min \mathbf{b}^T\mathbf{y} \text{ s.t. } A^T\mathbf{x} \geq \mathbf{c}, \mathbf{y} \geq 0 \\ &= \max(-\mathbf{b}^T\mathbf{y}) \text{ s.t. } -A^T\mathbf{x} <= \mathbf{c}, \mathbf{y} \geq 0 \\ (L^*)^* &= -\min(\mathbf{c}^T\mathbf{x}) \text{ s.t. } -(A^T)^T\mathbf{x} \geq -\mathbf{b}, \mathbf{x} \geq 0 \\ &= \max \mathbf{c}^T\mathbf{x} \text{ s.t. } A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0 \end{aligned}$$

---

dualising equality constraints

Let $L = \min \mathbf{c}^T\mathbf{x}$ s.t. $A\mathbf{x} = b, x \geq 0$.

$$\begin{aligned} L &= \min \mathbf{c}^T\mathbf{x} & &\text{s.t. } \begin{pmatrix} A \\ -A \end{pmatrix}\mathbf{x} & &\geq \begin{pmatrix} \mathbf{b} \\ -\mathbf{b} \end{pmatrix}, \mathbf{x} \geq 0 \\ L* &= \max \begin{pmatrix} \mathbf{b} \\ -\mathbf{b} \end{pmatrix}^T \mathbf{y} & &\text{s.t. } \begin{pmatrix} A \\ -A \end{pmatrix}^T \mathbf{y} & &\leq \mathbf{c},\ \mathbf{y} \geq 0 \\ &= \max \begin{pmatrix} \mathbf{b}^T & -\mathbf{b}^T \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} & &\text{s.t. } \begin{pmatrix} A^T & -A^T \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} & &\leq \mathbf{c}, \mathbf{y} \geq 0 \\ &= \max \mathbf{b}^T(\mathbf{u} - \mathbf{v}) & &\text{s.t. } A^T(\mathbf{u} - \mathbf{v}) & &\leq \mathbf{c}, \\ &= \max \mathbf{b}^T\mathbf{z} & &\text{s.t. } A^T\mathbf{z} & &\leq \mathbf{c}, \end{aligned}$$

where $\mathbf{z} = \mathbf{u} - \mathbf{v}$ is a vector of free variables $(\mathbf{u}, \mathbf{v} \geq 0)$

cor: dual variable defined by an equality constraint is unrestricted

---

Weak duality

$\mathbf{x}, \mathbf{y}$ feasible for primal, dual: $\mathbf{c}^\mathbf{T}\mathbf{x} \leq \mathbf{b}^\mathbf{T}\mathbf{y}$

$\mathbf{x}$ feasible $\implies A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0$

$\implies (A\mathbf{x})^T \leq (\mathbf{b})^T \implies \mathbf{x}^T A^T \leq \mathbf{b}^T \implies \mathbf{x}^T A^T \mathbf{y} \leq \mathbf{b}^T\mathbf{y}$

$\mathbf{y}$ feasible $\implies A^T\mathbf{y} \geq \mathbf{c}, \mathbf{x} \geq 0$

$\implies \mathbf{x}^T A^T \mathbf{y} \geq \mathbf{x}^T\mathbf{c}$

combining, we have $\mathbf{x}^T\mathbf{c} \leq \mathbf{b}^T\mathbf{y}$

cor: any feasible solution for maximum problem is lower bound to minimum value of minimum problem

cor: any feasible solution for minimum problem is lower bound to maximum value of maximum problem

cor: if maximum problem is feasible/unbounded, minimum has no feasible solution

cor: if minimum problem is feasible/unbounded, maximum has no feasible solution

cor: if both problems are feasible, both are bounded

Possible to have primal and dual both infeasible

$$\begin{aligned} \max\ & 2x_1 - x_2 \\ \text{s.t. } & x_1 - x_2 \leq 1 \\ & -x_1 + x_2 \leq -2 \qquad \& \ x_1, x_2 \geq 0 \end{aligned}$$

Complementary slackness:

Write L as:

$$P: \max \sum_j c_j x_j = z$$
$$\text{s.t. } \sum_j a_{ij} x_j + s_i = b_i \text{ for all } i$$
$$x_j, s_i \geq 0 \text{ for all } j$$

write dual as

$$D: \min \sum_i b_i y_i = z$$
$$\text{s.t. } \sum_i a_{ij} y_i - t_j = c_j \text{ for all } i$$
$$y_i, t_j \geq 0 \text{ for all } j$$

Both feasible (by assumption), with optimal solutions $w^* = z^*$ (equality by duality):

$$\begin{aligned} w^* - z^* &= \sum_i b_i y_i - \sum_j c_j x_j \\ &= \sum_i (\sum_j a_{ij} x_j + s_i) y_i - \sum_j (\sum_i a_{ij} y_i - t_j) x_i \\ &= \sum_j s_i y_i + \sum_j t_j x_j = 0 \end{aligned}$$

since all vars ≥ 0 : $s_i y_i = 0 = t_j x_j$

---

intersection of convex sets is convex:

Let $\mathbf{x}_1, \mathbf{x}_2 \in S_1 \cap S_2$; $c \in [0,1]$. Now $c\mathbf{x}_1 + (1-c)\mathbf{x_2} \in S_1$ & similar for S_2. So $c\mathbf{x}_1 + (1-c)\mathbf{x_2} \in S_1 \cap S_2$. Extend by induction

Union of convex sets is not necessarily convex

E.g.: $S_1 = \{(x,y) \in \mathbb{R}^2 | 0 \leq x \leq 2, 0 \leq y, \leq 1\}$, $S_2 = \{(x,y) \in \mathbb{R}^2 | 0 \leq x \leq 1, 0 \leq y, \leq 2\}$

A hyperplane in $\mathbb{R}^n$ divides the space into two convex sets

Take two points in either set and apply the definition of convexity

The feasible region for an LP is convex: Combine result re. hyperplane and result re. intersections.

A function f is convex $\iff$ $-f$ is concave: Follows directly from dfns

A linear function is both convex and concave: Let $f(\mathbf{x}) = a\mathbf{x} + b$ and consider

$$\begin{aligned} f(c\mathbf{x}_1 + (1-c)\mathbf{x_2}) &= a(c\mathbf{x}_1 + (1-c)\mathbf{x}_2) + b \\ &\quad + cb + (1-b)b \\ &= c(a\mathbf{x}_1 + b) + (1-c)(a\mathbf{x}_2 + b) \\ &= cf(\mathbf{x}_1) + (1-c)f(\mathbf{x}_2) \end{aligned}$$

f, g convex $\implies$ $f + g$ convex:Apply def of convexity and add resulting inequalities

---

$f''(x) \geq 0$ for all $x \in$ some convex set S $\implies$ $f(x)$ convex

If $f(x)$ is convex, the line joining any two points is never below the curve, so the slope of $f(x)$ must be non-decreasing for all x.

Proofs & derivations

Newton's method

$$\begin{aligned}
\text{Taylor series :} f(x) &= \sum_i \frac{f^{(n)}(a)}{n!}(x-a)^n \\
&\approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 \\
\left(\frac{d}{dx}\right) \quad f'(x) &\approx 0 + f'(a) + \frac{2}{2}(x-a)f''(x) \\
f'(x) = 0 \implies f'(a) &\approx -xf''(x) + af''(x) \\
\implies x &\approx \frac{af''(x) - f'(a)}{f''(x)} = a - \frac{f'(a)}{f''(x)}
\end{aligned}$$

As iterative scheme, set $x_n = a, x_{n+1} = x$: $\implies \boxed{x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}}$

Newton's method: multivariate Taylor expansion for two-variable case:

$$\begin{aligned}
f(x,y) = f(a,b) &+ (x-a)\frac{\partial f}{\partial x}(a,b) + (y-b)\frac{\partial f}{\partial y}(a,b) \\
&+ \frac{1}{2}\left[(x-a)^2\frac{\partial^2 f}{\partial x^2} + 2(x-a)(y-b)\frac{\partial^2 f}{\partial x \partial y}(a,b) + (y-b)^2\frac{\partial^2 f}{\partial y^2}(a,b)\right]
\end{aligned}$$

Set $(a,b) = (x_n, y_n)$ as before; Now we take the partial derivatives w.r.t. x, y and then set them$= 0$ as before; (*remember all the* ∂ *terms are constants w.r.t.* x)

if (x_{n+1}, y_{n+1}) is an improved estimate for the optimum, we can write this compactly as:

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{pmatrix} + \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x \partial x_2} \\ \frac{\partial^2 f}{\partial x_1 \partial y} & \frac{\partial^2 f}{\partial x_2^2} \end{pmatrix} \begin{pmatrix} x_{n+1} - x_n \\ y_{n+1} - y_n \end{pmatrix}$$

$$\text{i.e.} \quad \mathbf{0} = \nabla f + H(\mathbf{x}_{n+1} - \mathbf{x}_n)$$

$$\implies \mathbf{x}_{n+1} = \mathbf{x}_n - H^{-1}\nabla f$$

This gives us our iterative method.

Sufficient condition for global optima: S convex, objective function f_0 concave $\implies$ local max is optimum

Let $\mathbf{x}^*, \mathbf{x}' \in S$ both local max, with $f(\mathbf{x}^*) > f(\mathbf{x}')$:

- By concavity of f, $f(c\mathbf{x}' + (1-c)\mathbf{x}^* > f(\mathbf{x}')$ (plug in def of concavity) [1]
- $\mathbf{x}'$ is local max so $f(x') \geq f(\mathbf{x})$ for all $\mathbf{x} \in$ some neighbourhood N [2]
- Let $\mathbf{x} = (c\mathbf{x}' + (1-c)\mathbf{x}^*$ s.t. $\mathbf{x} \in N$ $(c \to 1)$
- (By [1]) $f(\mathbf{x}) > f(\mathbf{x}') \geq f(\mathbf{x})$ (by [2]). Contradiction!

Cor: minimisation, S convex, f_0 convex $\implies$ any local min is an optimal solution

---

Proofs & derivations

7 Adv-Disad-When

Big-M // 2-phase - Two-phase for testing feasibility
- Two-phase for implementing on computer
- Big-M is simpler though

Interior pt: - Large problems
- Don't need post-analysis

⇝ Combo?

Newton vs

✓ conceptually simple
✓ easy to implement
✓ can converge fast
✗ f ☆ must be twice differentiable ☆
✗ might converge to local but not global optimum
✗ badly behaved functions (could diverge or wander)

interval

✓ simple
✓ easy computation
✗ slow convergence
✗ f must be differentiable
✗ need single optimum in the interval

Multivar unconstrained exact

...
✓ simple
✓ can often find local optima

✗ often not applicable
✗ need differentiability
✗ can be hard/impossible to solve simultaneous eqn.

Multivar Newton NB Newton is *exact* for quadratic problems!

☆ Good starting point is crucial ☆
✓ Fast convergence in some cases.
✗ ☆ BOOM ☆ but "not a viable practical tool":
✗ If H has a singularity (between starting point and true optimum), ☆ BOOM ☆
✗ Can be badly behaved // not robust //
⇝ sensitive to starting point
⇝ can reach stationary iteration point
⇝ or get stuck in a cycle
✗ Convergence might not be to an optimum (local; saddle point...)
✗ Need:
  H invertible and well conditioned
  f twice differentiable with explicit analytic form of derivatives
✗ Remember this is an approximation method and we're discarding quadratic (& higher power) terms of a series ... but that can actually cause problems for convergence
✗ Needs a lot of computing power

Adv-Disad-When

Quasi-Newton ☆ Not generally recommended (except well conditioned problems) ☆

! But: many methods suffer from "tricky computing", local optima, need for differentiability

Steepest ascent:

✓ Simple idea
✓ Usually converges

✗ Often slowly (compared to other approaches)
✗ Tricky computing, including univariate search
✗ Could be a *local* optimum
✗ Require differentiability

Adapt steepest ascent to fix zigzagging & slow convergence? $\implies$ bring it in line with other methods?

⇝ change step size e.g. 0.9α

⇝ modify direction e.g. $\alpha(\frac{1}{2}(\nabla f(\mathbf{x}_n) + \nabla$

Eq. constrained: subst - non-obvious

Barrier/Penalty:

✓ normally converge; handle cusps & other anomalies well

✓ easier programming

(only unconstrained functions)

✗ working with more complex functions

✗ can be issues with slow convergence

B: even if not convergence, all solutions are feasible

B: typically require fewer function evaluations $\implies$ faster

P: good with equality constraints

(barrier methods are complicated)

P: barrier need feas. start point $\implies$ poss. hard to find

7.1 Maxes and mins

- LP: $\boxed{\text{max}}$ $\mathbf{c}^t\mathbf{x}$ s.t. $A\mathbf{x}\boxed{\leq}\mathbf{b}$
- Lemke: $\boxed{\text{min}}$ $\frac{1}{2}\mathbf{x}Q\mathbf{x} + \mathbf{c}^\mathbf{t}\mathbf{c}$ s.t. $A\mathbf{x}\boxed{\leq}\mathbf{b}$
- NLP $\boxed{\text{max}}$ $f_0(\mathbf{x})$ s.t. $f_i(\mathbf{x})\boxed{\leq}0$
- NLP for KKT: $\boxed{\text{min}}$ $f_0(\mathbf{x})$ s.t. $f_i(\mathbf{x})\boxed{\leq}0$
- Lagrange multipliers: also $\boxed{\textit{min}}$ f; work with $\boxed{\text{equality}}$ constraints
- Convex: $f(c\mathbf{x}_1 + (1-c)\mathbf{x}_2)\boxed{\{\leq, <\}}cf(\mathbf{x}_1) + (1-c)f(\mathbf{x}_2)$
- Steepest $\boxed{\text{descent}}$: choose α to $\boxed{\text{minimise}}$ $f(\mathbf{x}_n\boxed{+}\alpha_{n+1}\nabla f(\mathbf{x}_n))$
- Steepest $\boxed{\text{ascent}}$: choose α to $\boxed{\text{maximise}}$ $f(\mathbf{x}_n\boxed{+}\alpha_{n+1}\nabla f(\mathbf{x}_n))$ (+ in both cases)
- Second deriv: Convex $\iff$ $f''(x)$ $\boxed{\geq}$ 0 (Hessian positive semidef)
- Principal minors: $\boxed{\geq 0}$: Convex
- Leading p. minors / stationary point: $\boxed{> 0}$ local min.
- Maximisation, f.r. convex, f $\boxed{\text{concave}}$: $\implies$ local max is global max
- Sensitivity analysis
  - Forcing a non-basic var: subtract col from RHS (remove)
  - Forcing a basic var: subtract chosen change var from RHS (remove)
  - Change to constraint: add to RHS (don't remove)
  - Change to bottom line: subtract (+ve) change amount from b.r. entry (then pivot)