Optimisation Theory and Applications

Dedicated to Amoebas

December 2018

1 Linear Programming

1.1 Introduction L1

- General form of an LP problem is max/min $\sum_i c_i x_i$ subject to constraints $\sum_j a_j x_i \leq, =, \geq b_n$ and $x_i \geq 0$. In matrix notation: max/min $\mathbf{c}^T\mathbf{x}$ subject to $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{x} \geq 0$
- **Feasible region** the set of all points that satisfy the constraints given by an optimisation problem. All feasible regions are 'convex polyhedra' (volume inside $\mathbf{Ax} \leq \mathbf{b}$).

1.2 Graphical Solution L2

- **METHOD 1**: Choose corners of the feasible polyhedra formed by the constraints and see which maximises/minimises the objective function (Theorem 2).
- Theorem 1: If an LP has a *set* of feasible solutions, i.e. multiple solutions (not just optimal), then this set is a *convex* set of points.
  - A convex set of points is such that one can form a line segment that is within the set between two points x_1 and x_2 that are also in the set. An arbitrary point on the line is given by $x_3 = \lambda x_1 + (1-\lambda)x_2), (0 \leq \lambda \leq 1)$.

$$f(\lambda \mathbf{x}_2 + (1-\lambda)\mathbf{x}_2) \leq f(\mathbf{x}_2) + (1-\lambda)f(\mathbf{x}_1), 0 \leq \lambda \leq 1$$

1.3 Standard Form L3

- Introduce slack variables $\rightarrow x_1 \leq 8 \rightarrow x_1 + s_1 = 8, s_1 \geq 0$ to turn $\mathbf{Ax} \leq \mathbf{b}$ into standard form i.e. $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{x} \geq 0$
  - Given $\mathbf{Ax} = \mathbf{b}$ one can see if some constraints are redundant or inconsistent. You can see if the system is inconsistent if the equations are not linear combinations of each other (use Gaussian elimination).
  - If the equations are consistent we can reduce the number of equations and solve the LP problem. If it is not consistent then the LP has no solution.

1.4 Basic Solutions L4

- Essentially these are solutions to a set of equations ($\mathbf{Ax} = \mathbf{b}$), where $\mathbf{x}$ contains the slack variables, in which one simply sets a particular determined number of variables to zero and generates the solutions for the non-zeroed variables One then iterates this with all possible combinations of 'basic solutions' to find the optimal solution. All it does is find the corners of the graphical solution feasible space analytically (one of which will be optimal, Theorem 3).
  - 'basic variables' are positive variables and 'non-basic variables' are zeroed variables (non-intuitively).
  - If there are m equations and n variables, i.e. $\mathbf{A}_{m\times n}$ and $\mathbf{x}_{n\times 1}$, then:
    - There are $(n-m)$ zeros (non-basic variables) in the basic solution.

* There exist ${}^nC_{n-m} = \frac{n!}{(n-m)!m!}$ basic solutions to the LP problem.

- Essentially, for an LP problem the method analytically is: Add slack variables → Find the number of variables to set to zero and how many posssible basic solutions there are → Enumerate the basic solutions with objective function to find the optimal solution.

1.5 Simplex Method L5

- **METHOD 2**: 1) Convert LP problem into standard form, 2) Convert it to *tableau form* (recall that objective row is negative for maximisation) 3) Choose the **pivot** element in the column that has the largest negative objective function value and in the row with the smallest ratio. 4) Use row reduction techniques to make that pivot point 1 with all other column entries being zero 5) Repeat steps 3) and 4) until all objective function entries in the tableau are positive.
- There must always be an 'identity matrix' visible in the tableau.
- Make sure you do questions with minimisation too.
- Beware of inverse relationships in the constraints.

1.6 Comments on Simplex Method L6

- Minimisation: Choose the most positive objective function value column to pivot and repeat METHOD 2 (Simplex) until the objective function row only contains zeros and negative numbers.
  - Use **Surplus variables** to do this. $x_1 \geq 5 \rightarrow x_1 - s_1 = 5, s_1 \geq 0$
  - If a variable is unconstrained one can use a difference between two positive variables $x = u - v, u, v \geq 0$. For multiple unconstrained variables keep v the same and just change u to u_i.
- The set of basic solutions that are identical to each other is referred to as a "degenerate set". This means the simplex method may stall, i.e. not improve for several steps or stay in a degenerate loop forever.

1.7 Big M Method L7

- Just like Simplex method only you add artificial variables a_i if an 'identity matrix' is not visible. One gives the artificial variables a high cost in the objective function to penalise them, use $-M(\sum_i a_i)$ for maximisation and $M(\sum_i a_i)$ for minimisation. Set M to a semi large round number.
- If there is a negative constraint, multiply by -1 and flip the inequality
- Add slack variable for $\leq$; subtract surplus variable (negative slack) and add artificial variable for $\geq$; add artificial variable for $=$. (For maximisation).
- Add $(\sum_i a_i)$ to objective function and move everything to left-hand side.
- Eliminate all M's from artificial variables columns in objective function row. Then start simplex.

1.8 Duality L8

- A LP minimisation problem can be reformulated as an LP maximisation problem using "duality".

min $S_x = \mathbf{c}^T\mathbf{x}$	max $S_y = \mathbf{b}^T\mathbf{y}$
given $\mathbf{Ax} \geq \mathbf{b}$	given $\mathbf{A}^T\mathbf{y} \leq \mathbf{c}$
and $\mathbf{x} \geq \mathbf{0}$	and $\mathbf{y} \geq 0$

Table 1: Duality

- If you are given either a min or a max LP then this is called the **Primal Problem** and the dual is called the **Dual Problem**.

- **Duality Theorem**: If an optimal solution exists for the primal problem then an optimal solution exists for the dual problem. (Weak Duality Theorem just says that if the primal problem has a feasible solution then so does the dual problem)
- **Complementary slackness theorem**: If $\mathbf{x}$ and $\mathbf{y}$ are feasible solutions to an primal-dual LP pair then both are optimal if and only if $\mathbf{y}^T(\mathbf{Ax} - \mathbf{b}) = 0$ and $\mathbf{y}^T(\mathbf{A}^T\mathbf{y} - \mathbf{c}) = 0$. Proof: For $S_x = S_y$ then $y^TAx = y^Tb = by^T = S_y = S_x = c^Tx = x^Tc = x^T(A^Ty)$ hence the above.
- The solutions to the slack variables in objective row using the Simplex method are the same solutions as that for the dual problem, i.e $s_1 = y_1, s_2 = y_2$ for example.

2 Integer Programming

2.1 Introduction to Integer Programming L9

- The LP solution to an IP problem gives a bound on the optimal value. LP solution "relaxes" IP constraints.
- Rounding the LP solution will not necessarily give the optimal or a feasible IP solution.

2.2 Branch and Bound Method L10

- The idea is to continually solve equivalent LP problems, until one reaches a feasible integer solution. One does so by "branching" on the largest non-integer and then "bounding" the new LP problem to a new constraint.
- When you see a float solution like $x_1 = 3.8$ then branch out on i) $x_1 \leq 3$ and ii) $x_1 \geq 4$

3 Nonlinear Programming

3.1 Introduction to Nonlinear Programming L11

- The quantity to be optimised is called the **objective function**. It's variables are called *control* or *decision variables*. The constraint functions and the objective function are refered to sometimes as the *problem functions*.
- Example of a nonlinear quadratic equation in matrix form:

$$\begin{aligned} f(\mathbf{x}) &= \underbrace{\mathbf{x}^T\mathbf{A}\mathbf{x}}_{quadratic} + \underbrace{\mathbf{b}^T\mathbf{x}}_{linear} + c \\ f'(\mathbf{x}) &= 2\mathbf{A}\mathbf{x} + \mathbf{b} \quad \{Derivative\} \\ f''(\mathbf{x}) &= \mathbf{A} \quad \{Hessian\} \end{aligned}$$

- **Optima**:
  - Strict global - the very best optimal in the function space, nothing is better
  - global - one of the best, there could be other optima that are just as good.
  - Strong local - the very best optima in a local neighbourhood.
  - Weak local - One of the best in the local neighbourhood.
- Stationary points are found at $\nabla f = \mathbf{0}$. If the Hessian is positive definite ($eig(H) > 0$, all principle minors are positive) or negative definite ($eig(H) < 0$, all principle minors are $(-1)^k$, i.e. of alternating sign, starting negative) then this is a strong local min and max respectively.
  - Principle minors go from top left corner and then expand outwards until you have the full matrix, always calculating the determinant along the way.
- **Convexity** determines whether local optima is the unique (strict) global optima.

- Again find the Hessian, but not at the optima point but as a function of all feasible $\mathbf{x}$ and if it is **positive definite** for all $\mathbf{x}$ then the function **convex**. If is *negative definite* for all $\mathbf{x}$ then it is *concave*. If it is neither then we cannot say that the optima is a global optima.

3.2 Downhill Simplex/Amoeba method/Nelder-Mead method) L12

- **Simplex** - A simplix is a generalisation of a triangle into arbitrary dimensions, i.e. dot - 0D, line- 1D, triangle - 2D, tetrahedron-3D. It has $n+1$ vertices, given n dimensions.
- **METHOD**
  - 1) Evaluate and order vertices: $f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \leq f(\mathbf{x}_{n+1}$
  - 2) Find centroid (vector $\mathbf{x}_0$) of all vertices of simplex except $\mathbf{x}_{n+1}$, so on a triangle this is just the midpoint of two points, but in 4D its the centroid of three points.
  - 3) Reflect away from worst point to form new simplex $\mathbf{x}_{new} = \mathbf{x}_0 + \phi(\mathbf{x}_0 - \mathbf{x}_{n+1})$.
  - 4)This new point can be *expanded* $\phi = \gamma$ if $f(\mathbf{x}_{new}) < f(\mathbf{x}_1)$ or *contracted* $\phi = \rho \in [0, 0.5]$ if $f(\mathbf{x}_{new}) \geq f(\mathbf{x}_1)$
  - 5) If nothing gets better then we can shrink the simplex, all point $\mathbf{x}_i$ except x_1 (the best point) are replaced with $\mathbf{x}_i = \mathbf{x}_1 + \sigma(x_i - x_1)$
  - Repeat 4-5 until you meet some specified criteria, like only repeat 3 times.
- This is a referred to as a function-call method, because it doesn't use gradients.

3.3 Steepest Descent (SD) Method L13

- Gradient based method.
- **METHOD**
  - Condition - choose a tolerance ϵ, such that when the modulus of the gradient $|\nabla f(\mathbf{x}_t)| < \epsilon$ we stop the following iterative method.
  - 1) Chose a random starting point $\mathbf{x}_t$
  - 2) Given objective function f find the amount (λ) by which we should move in the steepest direction by optimising max/min $f\left(\mathbf{x}_t + \lambda(\pm\nabla f(\mathbf{x}_t))\right)$ using $\frac{\partial}{\partial\lambda} f\left(\mathbf{x}_t + \lambda(\pm\nabla f(\mathbf{x}_t))\right) = 0$ (might have to work out second derivative too.)
  - 3) Find new point $\mathbf{x}_{t+1} = \mathbf{x}_t + \lambda_t(\pm\nabla f(\mathbf{x}_t))$
  - 4) Repeat 2)-3) until condition is met.
- Note, the gradient vectors are always orthogonal to each other, which is quite inefficient. Can show using chain rule that $\frac{\partial}{\partial\lambda} f\left(\mathbf{x}_t + \lambda(\pm\nabla f(\mathbf{x}_t))\right) = 0 = \nabla f(x_{t+1}) \cdot \nabla f(x_t)$

3.4 Conjugate Gradient (CG) Method L14 Review Proofs

- The idea is to optimise the gradient descent/ascent by not going in orthogonal directions like in the SD. However this only works if the Hessian of the objective function is *positive definite* and *symmetric*.
- Two vectors $\mathbf{u}$ and $\mathbf{v}$ are considered conjugate (a-orthogonal) (w.r.t to matrix A) if $\mathbf{u}^T\mathbf{A}\mathbf{v} = \mathbf{0}$. Its like they are orthogonal but in a different warped space defined by the matrix.
- **METHOD**
  - The first point $\mathbf{x}_1$ is going to be the same as that attained by the SD method, however λ_0 can be calculated using the following. For simplicity, let $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$ (the gradient) and $\mathbf{d}_0 = \pm\nabla f(\mathbf{x}_0)$ (the directional derivative $\rightarrow$ depending on whether this is a maximisation (+) or minimisation (-).

$$\lambda_0 = -\frac{\mathbf{d}_0^T\mathbf{g}_0}{\mathbf{d}_0^T A\mathbf{d}_0}$$

  but you'll literally get the same thing when you solve $\frac{\partial}{\partial\lambda} f\left(\mathbf{x}_t + \lambda(\pm\nabla f(\mathbf{x}_t))\right) = 0$

- Now the next point $\mathbf{x}_2$ is going to be different to the SD method, because we choose a different direction, namely $\boxed{\mathbf{d}_1 = -\mathbf{g}_1 + \beta \mathbf{d}_0}$ (so its a combination of the previous and next steepest direction.)
  - Where β is optimised by computing $\beta = \frac{\mathbf{g}_1^T A \mathbf{d}_0}{\mathbf{d_0}^T A \mathbf{d_0}}$ | Generally: $\beta_j = \frac{\mathbf{g}_{j+1}^T A \mathbf{d}_j}{\mathbf{d_j}^T A \mathbf{d_j}}$
- And then finally $\mathbf{x}_2$ is found by applying the formula for λ above to give $\mathbf{x}_2 = \mathbf{x}_1 + \lambda_1 \mathbf{d}_1$

- So really, the idea is just the same, you're only changing the directional derivative! Except you need to find two parameters instead of one.
- Note, that the optimal solution can also be found just solving $\mathbf{g}(\mathbf{x}) = 0$ (so do this on the side)

3.5 Newton's Method L15

- A 2nd order **gradient method** (like CG above) - not like SD (1st order, because it does not consider the hessian) or downhill simplex (0th order gradient method - no derivatives).
- Not to be confused with the other Newton Method for finding the zeros of functions.
- **METHOD**:
  - Iterative scheme: $\boxed{\mathbf{x}_{i+1} = \mathbf{x}_i - H(\mathbf{x}_i)^{-1} \nabla f(\mathbf{x}_i)}$ Notice that $\mathbf{x}_i \to \mathbf{x}_{i+1}$ as the gradient tends to zero. Review proof for formula in slides
  - Condition: Hessian H must be positive definite, otherwise it won't have an inverse.
    - If it isn't you can apply the **Levenberg-Marquard extension**: $x_{i+1} = x_i - [\epsilon \mathbf{I} + H(\mathbf{x}_i)]^{-1} \nabla f(\mathbf{x}_i)$, where $\epsilon > 0$

3.6 Gaussian Newton L16

- This is an extension to the Newton Method, but it only really applies to optimising residuals - so curve fitting, which also implies a minimisation problem.
- Iterative scheme: $\boxed{\mathbf{x}_{t+1} = \mathbf{x}_t - [J(\mathbf{x}_t)^T J(\mathbf{x}_t)]^{-1} J(\mathbf{x}_t) r(\mathbf{x}_t)}$ where J is the Jacobian Matrix of the residuals w.r.t the variables for each data point, so $r_1 = f(\mathbf{x}_1) - \mathbf{y_1} \to \frac{\partial r_1}{\partial x_1}$
- Again a **Levernberg-Marquard extension** can be added if $J(\mathbf{x}_t)^T J(\mathbf{x}_t)$ is not positive definite and therefore does not have inverse: $\mathbf{x}_{t+1} = \mathbf{x}_t - [J(\mathbf{x}_t)^T J(\mathbf{x}_t) + \lambda \mathbf{I}]^{-1} J(\mathbf{x}_t) r(\mathbf{x}_t)$ (start with λ being very small, e.g. $\lambda = 10^{-6}$
  - The value of λ in the Levenberg-Marquard equation above can be optimised using a so-called **Gain Ratio**: $GR = \frac{f(\mathbf{x}_t) - f(\mathbf{x}_{t+1})}{f(\mathbf{x}) - L(\mathbf{x}_{t+1})}$, where $L(\mathbf{x}_{t+1}) = somenasty formula$ is the error predicted in the model.
    - If the GR is large then reduce λ
    - If the GR is small increase λ

3.7 Constrained Optimisation using Penalty functions L17

- So far (SD,CG, N, GN) have looked at solving non-linear *unconstrained* optimisation problems. Here we look at **constrained optimisation**. The idea is to penalise an unconstrained optimisation if the constraints are not being met.
- External Penalty Functions (answers on route to solution can lie outside feasible region)
  - Given constraints $c_i(\mathbf{x}) = 0$ and obj. function $f(\mathbf{x})$, if the aim is to minimise, then a **quadratic penalty function** can be used $\to$ min $Z = f(\mathbf{x}) + \frac{1}{2\mu} \sum_i c_i(\mathbf{x})^2$. Start with $\mu = 1$ and then decrease μ towards zero at each iteration of an unconstrained optimisation technique like SD.
  - You can come up with a bunch of different penalty functions on your own too. The key thing is that they need to be differentiable so you can use techniques like SD and **that the penalty function becomes equivalent to the objective function if the constraints are met**.

- Interior/Internal Penalty functions - so called Barrier functions (remain in feasible region)
  - Given constraints $c_j(\mathbf{x}) \geq 0$, (notice the greater and equal to symbol) and obj. function $f(\mathbf{x})$ to be minimised, then we can use the so-called **SUMT** method (Sequential Unconstrained Minimisation Technique): min $\phi(\mathbf{x}, r) = f(\mathbf{x}) + r\sum_j \frac{1}{c_j(\mathbf{x})}$ and let r tend to zero as the iterations continue using unconstrained optimisation like SD.
  - Given same conditions as above, a **logarithmic barrier function** can be used too: min $\phi(\mathbf{x}, r) = f(\mathbf{x}) - \mu\sum_j \ln(c_j(\mathbf{x}))$ (notice minus sign)

3.8 Lagrangian Multipliers L18

- Let $f(\mathbf{x})$ be the objective function, $g(\mathbf{x}) = 0$ be the constraints. The idea is that they will share the same gradient vector i.e. $\nabla f = \lambda \nabla g$ at the optimal position because they'll be tangent to each-other.
- Given m constraints and n variables the lagrangian becomes: $L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_i^m \lambda_i \gamma(\mathbf{x})$. The 'necessary conditions' for an optimal solution, so-called **extremum** (not sure whether max or min) are that $\nabla L(\mathbf{x}, \boldsymbol{\lambda}) = 0$, naturally.
  - To determine whether it's a maximum or a minimum, the **bordered Hessian** can be applied to provide a 'sufficient condition' for max/min.
    * $H^B = \begin{bmatrix} 0 & P \\ P^T & Q \end{bmatrix} = \begin{bmatrix} 0 & \overbrace{\begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots \end{bmatrix}}^{n} \\ \overbrace{\begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_2}{\partial x_2} \\ \frac{\partial g_1}{\partial x_1} & \frac{\partial g_2}{\partial x_2} \\ \vdots & \vdots \end{bmatrix}}^{m} & \begin{bmatrix} \frac{\partial L}{\partial x_1 \partial x_1} & \frac{\partial L}{\partial x_2 \partial x_1} & \cdots \\ \frac{\partial L}{\partial x_1 \partial x_2} & \frac{\partial L}{\partial x_2 \partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \end{bmatrix}$
    * Q is essentially just the hessian.
    * $\mathbf{x}^*$ is a **maximum** if the last (n-m) principle minors of H^B are of alternating sign, starting with $(-1)^{m+1}$.
    * $\mathbf{x}^*$ is a **minimum** if the last (n-m) principle minors of H^B are of the same sign, starting with $(-1)^m$.
      · Don't get this mixed up with the definitions for a strong max and min, which only looks at the hessian matrix not this funky mess.

4 Dynamic Programming

- Dynamic Programming is a field of applied mathematics tasked with optimising discrete decision.

4.1 Bellman equation L20

- The Bellman equation is an optimal policy that can be used in various settings, teaching AI, route optimisation etc...
- The Bellman equation/ Value Iteration algorithm: $V(s) = max/min[R(s,a) + \gamma V(s+1, a+1)]$. There are alternative forms - one needs to change it slightly depending on the problem.
  - $V(s)$ is the value of being in the current state
  - $R(s,a)$ is the reward or cost of performing an action from the current state (there could be multiple)
  - $V(s+1, a+1)$ is the value associated with the previous or next state and the actions associated with it (depending on the question) - there will be multiple of these depending on the number of actions.

4.2 Formulating Dynamic Programming Problems

- Step 1) Formulate the problem without regard to DP
- Step 2) Generalise step 1) formulation by changing limits and constraints to variables
- Step 3) Identify state and stage variables
- Step 4) Define the Optimal value function (optimal value of objective function for DP)
- Step 5) Write down functional equation (Not sure about this step)
- Step 6) Solve functional equation using backward recursion and put into a table to read off answers easily.
- Not good enough just to solve problem, need to also try and write functional equation.
- Do lots of examples