Optimisation and Operations Research

Lecture 22: Linear Programming Revisited

Matthew Roughan
<matthew.roughan@adelaide.edu.au>
http://www.maths.adelaide.edu.au/matthew.roughan/notes/OORII/

School of Mathematical Sciences,
University of Adelaide

October 29, 2019

Section 1

Linear Programming Revisited

Other Issues for LPs

- Other approaches to solve LPs
  - We looked at Simplex
  - There are other approaches
- Preprocessing
  - Cleaning up the problem can make subsequent parts faster and more reliable
- What about non-linear, ...?

Section 2

Interior Point Methods

Boundary value and Interior point methods

There are two classes of methods for solving LPs

- Boundary value methods
  - searches around the vertices (boundary) of the feasible region
  - *e.g.*, Simplex
- Interior point method
  - find a feasible point inside region, and move towards the boundary
  - *e.g.*, Ellipsoid and Karmakar's method

These days Matlab has two algorithms, you can choose

- `dual-simplex`
- `interior-point`

`https://au.mathworks.com/help/optim/ug/linprog.html`
`https://au.mathworks.com/help/optim/ug/linear-programming-algorithms.html`

Simplex is exponential

In the practical we show that Simplex takes an exponential number of steps on the Klee-Minty example

- In the worst case instance, Simplex takes exponential time
- Interior point methods were developed in the pursuit of methods with polynomial worst-case time complexity.

Boundary value and Interior point methods

- Interior point methods were developed in the pursuit of methods with polynomial worst-case time complexity.
  Historically:

Date	**Method (Developer)**	**Worst case**	**Average case**
1947	Simplex (Dantzig)	exp	poly
1979	Ellipsoid (Khachiyan*)	poly	poly
1984	Interior point (Karmakar)	poly	poly
⋮			

- According to this table, you might expect the Ellipsoid Method to out-perform the Simplex Method. The ellipsoid method was a big theoretical breakthrough, but it turned out to be too slow for practical problems, despite its polynomial (in comparison to exponential) worst-case running time rating.

Boundary value and Interior point methods

Two points that are noteworthy

- Worst case analysis, and Big-O hide details
  - exponential might not be that slow for the problem you want to solve
  - worst cases may be rare for real problems
- Interior point methods have moved forward
  - it's not always obvious which will be best for your problem, but you should know they exist
  - they lead towards interesting algorithms for non-linear optimisation

Interior-point methods

- Follow a set of interior, feasible points
- Converge towards optimum, but don't exactly reach it
- Often work by transforming the problem at each step
- Use ideas like gradient projections
  - these can generalise to non-linear problems
  - you'll see more of these if you do Optimisation III

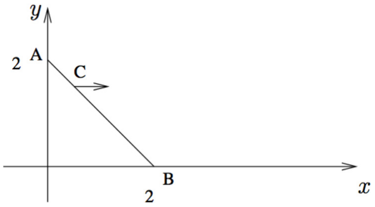
Karmarker's method

Example

$$\max \ z = f(x,y) = x \quad \text{s.t.} \quad x + y = 2 \quad \text{and} \quad x, y \geq 0.$$

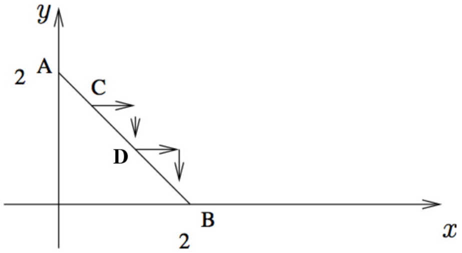
Obviously the answer is $z^* = 2$, at $x^* = 2, y^* = 0$.

The feasible region here is just a line segment AB from $(0,2)$ to $(2,0)$. Now suppose you pick any point $C \neq A$ or B, as a starting point on that line segment. Note that the direction of steepest ascent of z is the direction $\nabla f(x,y) = (1,0) = \mathbf{d}^T$, say (from Maths 1B). So from C, to maximise a function $z = f(x,y)$, we would normally head off in the direction $\mathbf{d} = (1,0)^T$.



In this case, we would move off horizontally from C and **out of the feasible region**!

Karmarker's method



If we move off from C in the direction $\mathbf{d}$, we would need to stop somewhere along that gradient $\mathbf{d} = (1, 0)$, and then project back onto the feasible region (our line segment) to a point D, then we will be closer to our maximiser. CD is called a projected gradient and has allowed us to improve on our value of z.

If we repeat the procedure over and over, with small enough steps, we should end up close to B.
Karmarkar's Method essentially does something like this.

Karmarkar's Interior Point Method – 1

For a problem in the form

$$\begin{array}{llcl} (P_1) & \min \quad z & = & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} & A\mathbf{x} & = & \underline{0} \\ & \mathbf{e}^T\mathbf{x} & = & 1 \\ & \mathbf{x} & \geq & \underline{0} \end{array}$$

where

$\mathbf{x} \in \mathbb{R}^n$,

$\min \ z = 0$,

$\text{rank}(A) = m$,

A is $m \times n$ and

$\left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right)^T$ is feasible.

We assume we will be satisfied with a feasible point having an optimal z-value $\in [0, \varepsilon)$, for some small $\varepsilon > 0$.

Karmarkar's Interior Point Method – 2

```
Initialise:
        Set $\mathbf{x}^{(0)} = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})^T$.
        Compute $r = \frac{1}{\sqrt{n(n-1)}}$ and set $\alpha = \frac{(n-1)}{3n}$.
        Compute $\mathbf{c}^T\mathbf{x}^{(0)}$ and set $k = 0$.

While   $\mathbf{c}^T\mathbf{x}^{(k)} \geq \varepsilon$          (for some given $\varepsilon$)
        Set $D_k = \text{diag}(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)})$ and $M = \begin{bmatrix} AD_k \\ \mathbf{e}^T \end{bmatrix}$.
        Compute $\mathbf{c}^{(k)} = \left(I - M^T\left(MM^T\right)^{-1}M\right)\left(-D_k\mathbf{c}\right)$.
        Compute $\mathbf{y}^{(k+1)} = \left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right)^T + \alpha r\,\frac{\mathbf{c}^{(k)}}{||\mathbf{c}^{(k)}||}$.
        Compute $\mathbf{x}^{(k+1)} = \frac{D_k\mathbf{y}^{(k+1)}}{\mathbf{e}^T D_k \mathbf{y}^{(k+1)}}$ (vector in the original space).
        Compute $\mathbf{c}^T\mathbf{x}^{(k+1)}$ and set $k = k + 1$.
End
```

Remarks About Karmarkar

- The direction $\mathbf{c}^{(k)}$ is the projection of the transformed "steepest descent" direction, $-D_k\mathbf{c}$ in the transformed problem.
- So finding $\mathbf{y}^{(k)}$ is making sure we find a feasible point in the transformed space and we have maximised the rate of decrease in the z-value.
- The αr term ensures $\mathbf{y}^{(k+1)}$ stays in the interior of the feasible region, *i.e.*, away from the boundary of the transformed unit simplex.
- The inverse transformation back from $\mathbf{y}^{(k+1)}$ to $\mathbf{x}^{(k+1)}$ implies $\mathbf{x}^{(k+1)}$ will be feasible for (P_1).

Karmarkar Example

Results to 4dp for $\varepsilon = 0.0005 = 5 \times 10^{-4}$ are given below.

k	$(\mathbf{x}^k)^T$	$(\mathbf{y}^k)^T$	z
0	$(0.3333, 0.3333, 0.3333)$	$(0.3333, 0.3333, 0.3333)$	-0.3333
1	$(0.3975, 0.3333, 0.2692)$	$(0.3975, 0.3333, 0.2692)$	-0.2692
2	$(0.4574, 0.3333, 0.2093)$	$(0.3930, 0.3415, 0.2655)$	-0.2093
3	$(0.5090, 0.3333, 0.1576)$	$(0.3883, 0.3489, 0.2628)$	-0.1576
4	$(0.5507, 0.3333, 0.1160)$	$(0.3839, 0.3549, 0.2612)$	-0.1160
5	$(0.5827, 0.3333, 0.0840)$	$(0.3803, 0.3594, 0.2602)$	-0.0840
$\vdots$	$\vdots$	$\vdots$	$\vdots$
19	$(0.6661, 0.3333, 0.0006)$	$(0.3704, 0.3703, 0.2593)$	-0.0006
20	$(0.6662, 0.3333, 0.0004)$	$(0.3704, 0.3703, 0.2593)$	-0.0004

For $\varepsilon = 0.00000005 = 5 \times 10^{-8}$, to 8dp, the procedure stops at $\mathbf{x}^{(46)} = (0.66666663, 0.33333333, 0.00000004)^T$ and $z = -0.00000004$.

Section 3

Preprocessing for optimisation

Preprocessing

Typical tasks

- put problem into standard form
- remove redundant constraints or variables
- rescale coefficients

Matlab: https://au.mathworks.com/help/optim/ug/linear-programming-algorithms.html

Preprocessing in Matlab's linprog before Simplex

- For row k, take the constraint $\mathbf{a}_k^T \mathbf{x} = b_k$
  - compute upper and a lower bounds of $\mathbf{a}_k^T \mathbf{x}$ (if finite)
  - if the bound $= b_k$, then the constraint is *forcing*
  - it fixes the values of x_i corresponding to non-zero values of $\mathbf{a}_k$, so they can be eliminated from the problem
- Check if any variables have equal upper and lower bounds. If so, check for feasibility, and then fix and remove the variables.
- Check if any inequality constraint involves just one variable. If so, check for feasibility, and change the linear constraint to a bound.
- Check if any equality constraint involves just one variable. If so, check for feasibility, and then fix and remove the variable.
- Check if any constraint matrix has a zero row. If so, check for feasibility, and delete the row.
- Check if the bounds and linear constraints are consistent.
- Check if any variables appear only as linear terms in the objective function and do not appear in any linear constraint. If so, check for feasibility and boundedness, and fix the variables at their appropriate bounds.

Section 4

Where to next?

Classes of optimisations

- Linear v non-linear
- Continuous v discrete (integer)
- Unconstrained v constrained
- Deterministic v Stochastic
- Problem structure
  - Low- v high-dimensional
  - Sparse v Dense
- Single- v Multi-objective
- Static v Dynamic
- Distributed v centralised

Taxonomy of optimisation problems
http://www.neos-guide.org/content/optimization-taxonomy

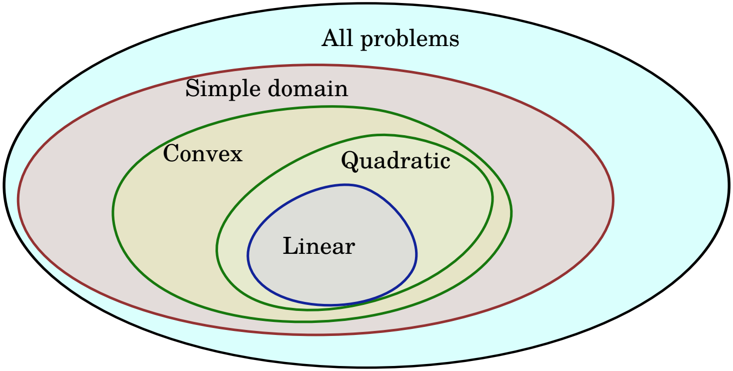
Matlab

Advice on which Matlab "solver" routine to use (from the Optimization Toolbox)

https://au.mathworks.com/help/optim/ug/optimization-decision-table.html

Non-linear programming

- Combinations of objective and constraint properties
  - ignoring for the moment integer v continous variables



- We saw how to tackle these by linear approximation in your project this year
- See *Optimisation III* next year.

Further reading I