Lecture 04 – Sensitivity Analysis (and Algorithms and Software)

Optimisation CT5141

James McDermott

University of Galway

Ollscoil na Gaillimhe
University of Galway

Overview

1. **Algorithms**
2. Software
3. Sensitivity: answering what-if questions

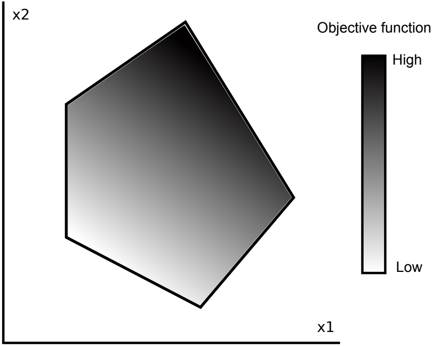
Solving LP problems

- An LP problem in 2 decision variables (DVs) can be solved with the graphical method
- Of course, these are toy problems!
- For larger ones and practical use, we need algorithms.
- We'll mention the main algorithms and two main software implementations.

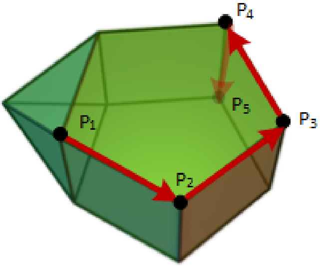
Simplex Algorithm for LP

The main algorithm for solving **LP** problems is the **Simplex Algorithm**. It is based on the fundamental theorem and Gaussian elimination (solving equations simultaneously).

A problem with 2 DVs gives a polytope in 2D:

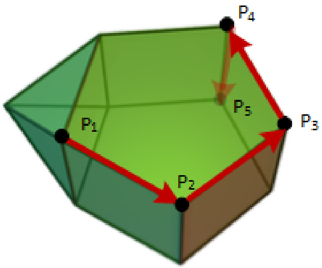


A problem with 3 DVs gives a polytope in 3D:



From `mathstools.com`

Simplex Algorithm for LP



From `mathstools.com`

We start at a feasible corner point and iteratively move along an edge to a better corner point. Because the feasible area is convex, an improvement is always possible (until we reach the optimum).
The edge is chosen by length and amount of improvement per unit length. This is deterministic and fast.

Simplex Algorithm for LP

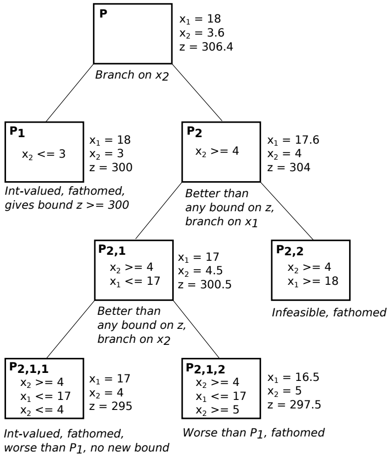
- The space that has to be explored is this feasible area
- Both number of constraints and number of DVs contribute to the number of edges of the feasible area, so both are relevant to problem difficulty.

Full description (not examinable) here.

Example video (not examinable) here.

Branch and Bound for IP

The main algorithm for solving **IP** problems is **Branch and Bound**. It relies on the fact that **the LP relaxation gives an upper bound on profit** (**lower bound on cost**). It is recursive, leading to a tree structure.



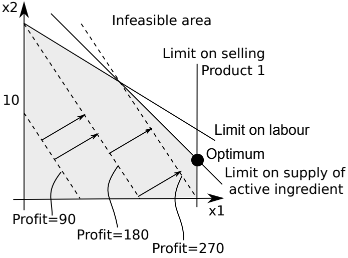
Branch and Bound for IP

Given a problem P, this is Branch-and-bound(P):

1. Solve the LP relaxation of problem P.
2. If the LP relaxation is **infeasible** or the objective value is **worse** than some integer solution already seen, then the true solution will not come from further branching here. Mark this branch as **fathomed** (do not expand it further).
3. Else if the solution has any non-integer DV e.g. $x_1 = 4.56$, then **branch**, i.e. construct new problems: P_1 adds $x_1 \leq 4$ and P_2 adds $x_1 \geq 5$. Call Branch-and-bound(P_i) for both i.
4. Else (the solution has no non-integer DV) this is an integer solution. It gives a **bound**: the true solution is not worse. Store the objective value and mark this branch as **fathomed**.
5. Return.

Eventually, this recursive process will **fathom** all branches and we return the best integer-valued solution we found.

Recall: Sanitizer problem



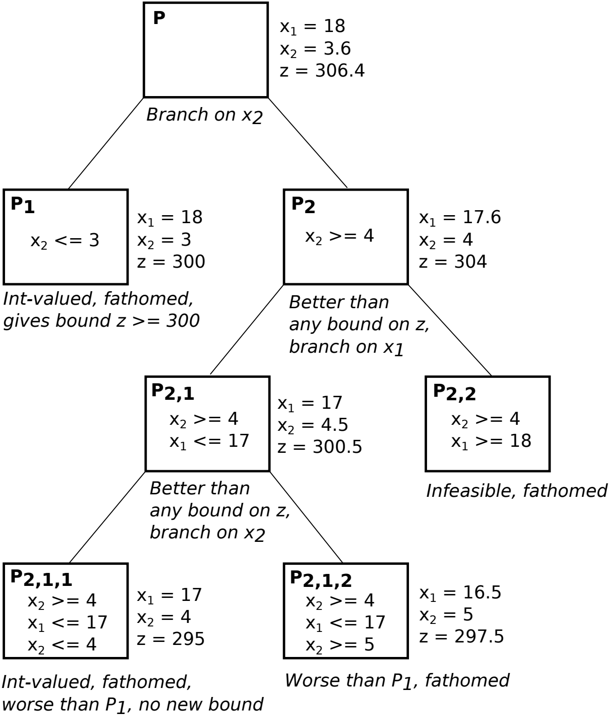
Maximise profit:

$$15x_1 + 10x_2$$

Subject to:

$$0.1x_1 + 0.1x_2 \leq 2.2 \text{ (raw materials)}$$
$$1.5x_1 + 2.5x_2 \leq 45 \text{ (labour)}$$
$$x_1 \leq 18 \text{ (demand for Product 1)}$$
$$x_2 \leq 30 \text{ (demand for Product 2)}$$
$$x_1, x_2 \geq 0 \text{ (non-negativity)}$$

Solution: $(18, 4)$ giving profit $z = 310$.


P
x1 = 18
x2 = 3.6
z = 306.4
Branch on x2
P1
x2 <= 3
x1 = 18
x2 = 3
z = 300
Int-valued, fathomed,
gives bound z >= 300
P2
x2 >= 4
x1 = 17.6
x2 = 4
z = 304
Better than
any bound on z,
branch on x1
P2,1
x2 >= 4
x1 <= 17
x1 = 17
x2 = 4.5
z = 300.5
Better than
any bound on z,
branch on x2
P2,2
x2 >= 4
x1 >= 18
Infeasible, fathomed
P2,1,1
x2 >= 4
x1 <= 17
x2 <= 4
x1 = 17
x2 = 4
z = 295
Int-valued, fathomed,
worse than P1, no new bound
P2,1,2
x2 >= 4
x1 <= 17
x2 >= 5
x1 = 16.5
x2 = 5
z = 297.5
Worse than P1, fathomed

Why is IP slower than LP?

As we have seen, our best algorithm for IP solves **many** LP problems while solving one IP problem!

Hungarian Algorithm

The Assignment problem has binary variables and a special structure which allows for a specialised algorithm, the **Hungarian algorithm**. This is more efficient than the branch-and-bound algorithm.

Explanation here (not examinable).

Algorithms

We won't be programming these algorithms by hand because (in contrast to heuristic/metaheuristic optimisation) it is very rare to need to program a custom variant.

Instead we'll move straight to considering software.

Overview

1. Algorithms
2. **Software**
3. Sensitivity: answering what-if questions

Software

- **Excel Solver**
- **OR-Tools**
- `scipy.optimize/linprog`
- PuLP
- COIN-OR
- Xpress-MP
- ...

We've seen Excel Solver in labs. Now we'll see a little of OR-Tools and its Python interface. We'll see how to model a problem and get the solution. Later we'll see how to do some sensitivity analysis.

OR-Tools

- Home: https://developers.google.com/optimization/
- LP: https://developers.google.com/optimization/lp/glop
- Guide to LP in OR-Tools: https://developers.google.com/optimization/lp/lp
- IP: https://developers.google.com/optimization/mip/integer_opt
- Guide to IP: https://developers.google.com/optimization/mip/mip_var_array

OR-Tools

```
# install in bash/PowerShell
$ pip install ortools
```

```
# check install ok
from ortools.linear_solver import pywraplp
```

OR-Tools modelling

(We'll use the Hand Sanitizer problem as an example)

- Instantiate a solver:

```
solver = pywraplp.Solver('Sanitizer',
     pywraplp.Solver.GLOP_LINEAR_PROGRAMMING)
```

OR-Tools modelling

- Create a continuous variable with bounds:

```
x1 = solver.NumVar(0, 18, 'x1')
```

OR-Tools modelling

- Create a continuous variable with bounds:

```
x1 = solver.NumVar(0, 18, 'x1')
```

Notice that here we are creating variables with special library-specific types and with strings as names. The variable `x1` doesn't have a numerical value like `4.56`. It is really a **placeholder**. A similar concept is used in neural network libraries like Tensorflow and symbolic maths libraries like Sympy (see CT5132/CT5148).

OR-Tools modelling

You can give any name you like, but don't confuse yourself!

```
x1 = solver.NumVar(0, 18, 'my favourite variable')
x2 = solver.NumVar(0, 30, 'x1')
```

OR-Tools modelling

- Create a constraint:

```
solver.Add(1.5*x1 + 2.5*x2 <= 45,
           name="limit on labour")
```

OR-Tools modelling

- Create a constraint:

```
solver.Add(1.5*x1 + 2.5*x2 <= 45,
           name="limit on labour")
```

Again, notice that this expression **looks** like it will have a Boolean value, but in fact variables `x1` and `x2` are of a type which **over-rides** $*$, `+` and `<=`. It just becomes a constraint object, stored inside the `solver` object.

OR-Tools modelling

- Create a constraint:

```
solver.Add(1.5*x1 + 2.5*x2 <= 45,
           name="limit on labour")
```

Again, notice that this expression **looks** like it will have a Boolean value, but in fact variables `x1` and `x2` are of a type which **over-rides** `*`, `+` and `<=`. It just becomes a constraint object, stored inside the `solver` object.

By the way, the Python builtin `sum()` runs `+` behind the scenes, so you can use things like `sum(c*x for c, x in zip(coefs, vars))` on the LHS of a constraint.

OR-Tools modelling

- Create objective:

```
objective = solver.Objective()
objective.SetCoefficient(x1, 15)
objective.SetCoefficient(x2, 10)
objective.SetMaximization()
```

OR-Tools modelling

- Solve:

```
result = solver.Solve()
```

- Print out solution:

```
for v in solver.variables():
    print(f"{v.name()} = {v.solution_value()}")
print(f"Obj val = {solver.Objective().Value()}")
```

OR-Tools outcomes

`result = solver.Solve()` will give an integer. We should check it.

```
pywraplp.Solver.OPTIMAL = 0
pywraplp.Solver.FEASIBLE = 1
pywraplp.Solver.INFEASIBLE = 2
pywraplp.Solver.UNBOUNDED = 3
pywraplp.Solver.ABNORMAL = 4
pywraplp.Solver.NOT_SOLVED = 6
```

OR-Tools outcomes

`result = solver.Solve()` will give an integer. We should check it.

```
pywraplp.Solver.OPTIMAL = 0
pywraplp.Solver.FEASIBLE = 1
pywraplp.Solver.INFEASIBLE = 2
pywraplp.Solver.UNBOUNDED = 3
pywraplp.Solver.ABNORMAL = 4
pywraplp.Solver.NOT_SOLVED = 6
```

I have not managed to see `UNBOUNDED` (it wrongly gives `INFEASIBLE`): https://github.com/google/or-tools/issues/2198

It has no way to detect **multiple equal optima** (it just gives `OPTIMAL`). (Excel is the same in this.)

OR-Tools Practicalities

Compared to Excel Solver, OR-Tools:

- Is harder to get started with
- Is more convenient for large problems:
  - Easier to type than to point-n-click
  - No need to type large matrix of constraint coefficients
- Has far more powerful solvers
- Amenable to version control.

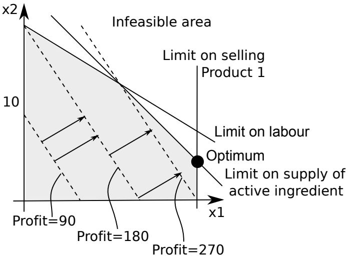
Overview

1. Algorithms
2. Software
3. **Sensitivity: answering what-if questions**

Asking what-if questions

- We should be able to **interpret** LP solutions
- Often we may be interested in follow-up "what if?" questions of high business importance
- The special assumptions of LP allow us to answer these questions
- This is called **sensitivity analysis** and sometimes **post-solution** analysis.

Recall: Sanitizer problem



Maximise profit:

$$15x_1 + 10x_2$$

Subject to:

$$0.1x_1 + 0.1x_2 \leq 2.2 \text{ (raw materials)}$$
$$1.5x_1 + 2.5x_2 \leq 45 \text{ (labour)}$$
$$x_1 \leq 18 \text{ (demand for Product 1)}$$
$$x_2 \leq 30 \text{ (demand for Product 2)}$$
$$x_1, x_2 \geq 0 \text{ (non-negativity)}$$

Interpreting solutions

The **solution** to this LP is:

$$(x_1, x_2) = (18, 4)$$

with objective value $f(x_1, x_2) = 310$.

Interpreting solutions

The **solution** to this LP is:

$$(x_1, x_2) = (18, 4)$$

with objective value $f(x_1, x_2) = 310$.

We should remember to interpret our solution fully in the real-world context. The solution means we should produce 18 units of Product 1 and 4 of Product 2, to achieve profits of EUR310.

Interpreting solutions

The **solution** to this LP is:

$$(x_1, x_2) = (18, 4)$$

with objective value $f(x_1, x_2) = 310$.

We should remember to interpret our solution fully in the real-world context. The solution means we should produce 18 units of Product 1 and 4 of Product 2, to achieve profits of EUR310.

Also, with this solution we use all 2.2L of active ingredient, and 37 hours of labour (leaving 8 hours of labour unused).

Sensitivity analysis

- How much of each resource have we used?
- How much of each resource is “left over”?
- Which constraints turned out to be important?
- How much would one more unit of a resource be worth to us? What if we had one unit less?
- How much more of that resource would be useful, before some other constraint prevented further increases in profit?
- How large a change could happen in the profit per unit of a product, before the location of the optimum would change?

Examples

- How many hours of labour will our plan use?
- Our value for raw materials availability is an estimate: should I go and find out the real value?
- I think we could increase the **selling price** on Product 1 with an advertising campaign – would it be worthwhile?
- I think we could increase the **demand** for Product 2 with an advertising campaign – would it be worthwhile?

Sensitivity

Is our outcome **sensitive** to the exact values of the data?

- That is: if some parameter changed slightly, would the outcome change **slightly**, or could it even change **a lot**?
- If a slight change in a parameter wouldn't change the outcome at all, we say the outcome is **insensitive** to changes in that parameter, otherwise we say it's **sensitive** to that parameter.

Sensitivity

Is our outcome **sensitive** to the exact values of the data?

- That is: if some parameter changed slightly, would the outcome change **slightly**, or could it even change **a lot**?
- If a slight change in a parameter wouldn't change the outcome at all, we say the outcome is **insensitive** to changes in that parameter, otherwise we say it's **sensitive** to that parameter.

(Sometimes **sensitivity analysis** is used to refer to all of these "what-if" questions.)

Sensitivity analysis

- **Activity**: How much of each resource have we used?
- **Slack**: How much of each resource is “left over”?
- **Binding**: Which constraints turned out to be important?
- **Shadow price**: How much would one more unit of a resource be worth to us? What if we had one unit less?
- **Allowable increase of a constraint**: How large an increase in the constraint RHS resource would be useful, before some other constraint prevented further increases in profit?
- **OFC sensitivity of a DV**: How large an increase in the objective function coefficient (OFC) of the DV could happen, before the location of the optimum would change?

Why don't we just…

Why don't we just re-run the problem, after adjusting the coefficients or RHS values, to see what happens?

Why don't we just...

Why don't we just re-run the problem, after adjusting the coefficients or RHS values, to see what happens?

- Actually, sometimes in IP we do that!
- But there are many possible adjustments, and larger problems take a long time
- The theory of LP (but not IP) gives us all the information we want "for free", as a **by-product** of running the simplex algorithm once.

Interactive/animated solution

Geogebra: https://www.geogebra.org/m/abjxez2u

Activity and Slack

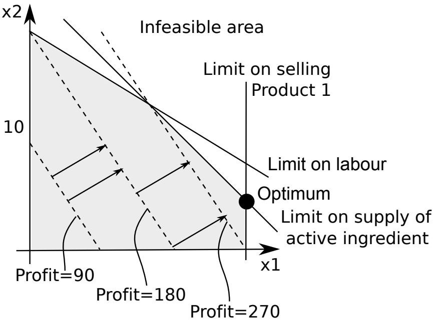
- The **activity** of a constraint is the value of its LHS at the optimum.
- E.g. the labour constraint is: $1.5x_1 + 2.5x_2 \leq 45$
- The optimum is (18, 4)
- The activity of the labour constraint is $1.5x_1 + 2.5x_2 = 1.5 \cdot 18 + 2.5 \cdot 4 = 37$, i.e. we are using 37 hours of labour.

Activity and Slack

- The **activity** of a constraint is the value of its LHS at the optimum.
- E.g. the labour constraint is: $1.5x_1 + 2.5x_2 \leq 45$
- The optimum is $(18, 4)$
- The activity of the labour constraint is $1.5x_1 + 2.5x_2 = 1.5 \cdot 18 + 2.5 \cdot 4 = 37$, i.e. we are using 37 hours of labour.

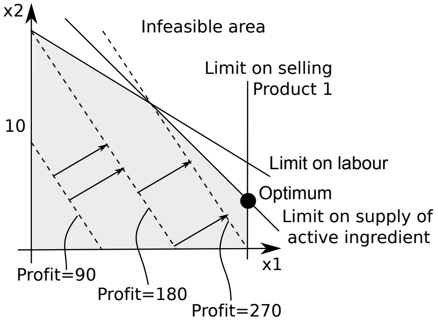
- The **slack** of a constraint is the amount “left over” on the RHS
- The slack of the labour constraint is 8 hours
- For any constraint: activity + slack = RHS value.

Binding



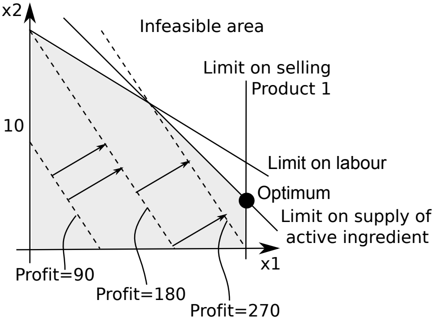
- A constraint is **binding** if its **slack is zero**: there is none of the RHS "left over"
- The constraint line **touches** the optimum
- Removing the constraint entirely would allow the optimum to be improved
- E.g., the labour constraint **is not binding**
- E.g., the active ingredient constraint **is binding**.

Shadow price



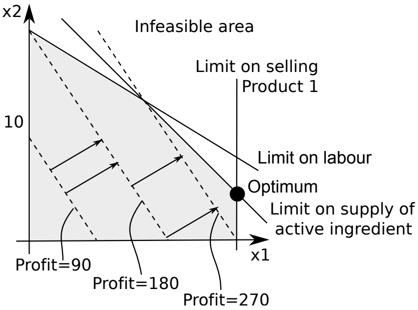
- The **shadow price** of a constraint is the improvement in the optimum we would achieve for unit increase in the RHS
- Shadow price = $\frac{\partial z}{\partial b_i}$ where z is profit and b_i is RHS.

Shadow price



- A **non-binding** constraint has **zero** shadow price
- E.g. the shadow price for labour is EUR0: it is non-binding, so extra available labour won't help
- E.g. the shadow price for Product 1 demand is EUR5: adding 1 extra unit of demand will allow profit EUR310 → EUR315.

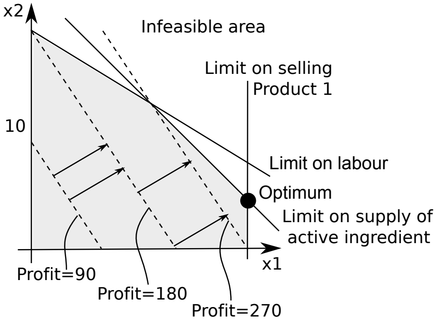
Allowable increase in a constraint



- E.g. the shadow price for Product 1 demand is EUR5: adding 1 extra unit of demand will allow profit EUR310 $\rightarrow$ EUR315.
- The location of the optimum would change to (19, 3).

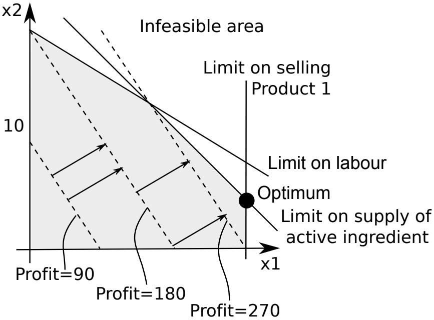
- Further increases in Product 1 demand up to 22 would be useful (giving (22, 0) and profit EUR330). But then Product 2 non-negativity becomes binding!
- The **allowable increase** in the Product 1 demand constraint was therefore 4 (from 18 up to 22).
- The shadow price of a constraint is **valid only up to the allowable increase**.

OFC sensitivity



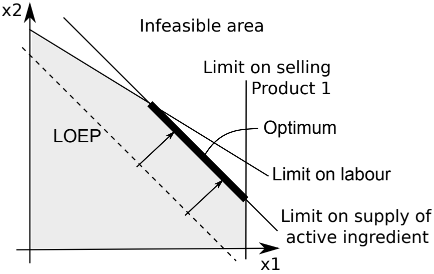
- The **sensitivity** of an objective function coefficient (OFC) is the range in which that OFC can change without changing the location of the optimum.
- Note the **profit** at the optimum will certainly change: we are concerned here with the **location**.

OFC sensitivity



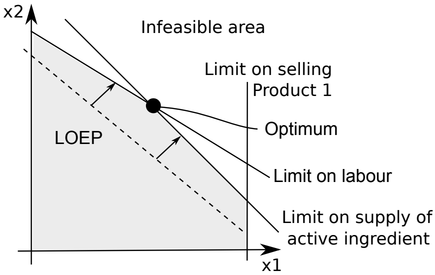
- The slope of the active ingredient constraint is -1: $0.1x_1 + 0.1x_2 \leq 2.2$
- Suppose Product 1 profit changes, so that the LOEP $15x_1 + 10x_2$ has slope -1 also
- That could happen if the 15 changes to 10: $10x_1 + 10x_2$

OFC sensitivity



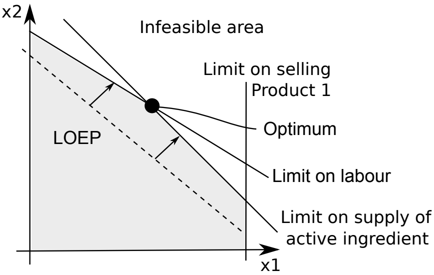
- That could happen if the 15 changes to 10: $10x_1 + 10x_2$
- Now the LOEP is **parallel** with the active ingredient constraint
- The whole line segment becomes **multiple equal optima**

OFC sensitivity



- If Product 1 profit decreased to < 10, the optimum would move to next corner as shown.

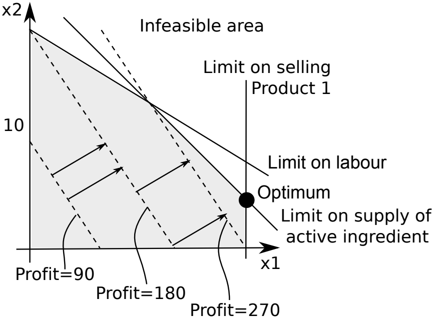
OFC sensitivity



- If Product 1 profit decreased to < 10, the optimum would move to next corner as shown.

Thus the location of the optimum is **insensitive to decreases** of the x_1 OFC from 15 down to 10.

OFC sensitivity



Other direction: any x_1 OFC **increases** will not change location.

Sensitivity is for one change only

All of this analysis is valid when we make a single hypothetical change at a time

With 2 or more changes, some analysis is still possible, but it's more complicated!

Sensitivity in minimisation problems

All of the above works in minimisation problems:

- Instead of a resource "left over" in a $\leq$ constraint, our solution provides "more than was required" of some RHS limit in a $\geq$ constraint, e.g. more than the daily minimum of vitamin C in a diet problem.
- Shadow prices must be interpreted carefully: a negative shadow price, in minimisation, means an improvement in the objective.

Sensitivity in Excel Solver

Excel Solver does a lot of sensitivity analysis (Google Docs and Libre-Office Solvers do not provide sensitivity analysis, unfortunately).

(See also Beasley's OR-notes: http://people.brunel.ac.uk/~mastjjb/jeb/or/lpsens_solver.html)

Sensitivity in Excel Solver

	A	B	C	D	E	F	G
1							
2							
3							
4		**Sanitizer**	**x1**	**x2**			
5		**values**	18	4			
6		**maximise**	15	10	310		
7		**subject to**	0.1	0.1	2.2	<=	2.2
8			1.5	2.5	37	<=	45
9			1	0	18	<=	18
10			0	1	4	<=	30
11							

Sensitivity in Excel Solver

Variable Cells

Cell	Name	Final Value	Reduced Cost	Objective Coefficient	Allowable Increase	Allowable Decrease
C5	values x1	18	0	15	1E+30	5
D5	values x2	4	0	10	5	10

Constraints

Cell	Name	Final Value	Shadow Price	Constraint R.H. Side	Allowable Increase	Allowable Decrease
E7	subject to	2.2	100	2.2	0.32	0.4
E8		37	0	45	1E+30	8
E9		18	5	18	4	8
E10		4	0	30	1E+30	26

Sensitivity in OR-Tools

We can access sensitivity information as follows:

- Activity: `solver.ComputeConstraintActivities()`
- Slack: `c.ub() - activity` or `activity - c.lb()`
- Binding: `abs(c.DualValue()) > epsilon`
- Dual Value: `c.DualValue()` (for constraint `c`)
- OFC Sensitivity and allowable increases/decreases: **not so easy**, we will not cover this (see https://groups.google.com/g/or-tools-discuss/c/0AXuU6AMfr0/m/0TLP2DQzBgAJ).

Summary

- Sensitivity is an important part of using LP in practice.
- Main concepts: activity, slack, binding, shadow price, allowable increase, OFC sensitivity.
- The language of sensitivity, shadow prices, etc., may not be accessible to our client or management. We should be able to provide simple explanations.

Overview

1. Algorithms
2. Software
3. Sensitivity: answering what-if questions