Optimisation and Operations Research
Lecture 14: ILPs in Matlab and AMPL

Matthew Roughan
<matthew.roughan@adelaide.edu.au>

School of Mathematical Sciences,
University of Adelaide

September 12, 2019
Section 1

Integer Programming: Matlab
**Matlab intlinprog**

Similar to the linprog command in Matlab for linear programs that have variables which can take on real solutions, there exists a command intlinprog for those linear programs that are also constrained to have variables to be *integer*, i.e., which can only take on the values 0 or 1.

That is, intlinprog solves binary linear programming problems of the form

\[
\min_{x} f^T x, \quad \text{such that} \quad \begin{cases} 
A x & \leq b \\
A_{eq} x & = b_{eq} \\
\text{some } x_i \text{ integer}
\end{cases}
\]

where, \( f, b, \) and \( b_{eq} \) are vectors, \( A \) and \( A_{eq} \) are matrices, and some of the variables are required to be integers.
MATLAB intlinprog example

Commands: for Binary program below

```matlab
> f = [-9; -5; -6; -4];
> A = [6,3,5,2; 0,0,1,1; -1,0,1,0; 0,-1,0,1];
> b = [9; 1; 0; 0];
> Aeq = [];
> beq = [];
> intcon = [1,2,3,4];
> ub = ones(4,1);
> lb = zeros(4,1);
> x = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub)
```

Output:

```
x = 1
   1
   0
   0
```
When to use intlinprog

- Don’t cheat
  - use this to check solutions
  - but solve them the way required in your assignments

- Yes, you can use it in your project
  - but display understanding
  - show alternatives

- In general
  - you still need to construct matrices and vectors which is awkward
  - you need to write every constraint, even if they fit a pattern
  - you still need explicit (closed form) constraints and objective functions
  - Matlab doesn’t tell you much about how it does it, and what its limitations are
    - we know the problem might be NP-complete, so this could be an issue
Section 2

AMPL
Practical Optimisation

Matlab is all very well, but what do real optimisers do?
Mathematical Modelling Languages

- There are computer languages designed specifically for optimisation
  - they allow natural expression of LPs, etc.
  - they link to multiple backends to solve the problem
  - they separate the model from the data
  - they link to other tools, e.g., databases, spreadsheets, ...
  - support reuse

- Common examples:
  - AMPL
  - GAMS
  - AIMMS
  - MPS (not really a modelling language, but is used for standard input)
  - ...
Why AMPL?

- I like it
- it’s one of the most commonly used (Neos says 59%)
- lots of backends
- free student version

History

- designed 1985 by Robert Fourer, David Gay and Brian Kernighan
- 2003 AMPL Optimization LLC
- 2012 INFORMS Impact Prize
Resources


- Other tutorials
  - [www.ieor.berkeley.edu/~atamturk/ieor264/samples/ampl/ampldoc.pdf](http://www.ieor.berkeley.edu/~atamturk/ieor264/samples/ampl/ampldoc.pdf)

- Download your own (student) copy of AMPL from
  - backends from the same place or others
    - there are other backends (we are using lpsolve, and I notice they don’t have a direct link to this anymore).

- Online solver NEOS [http://www.neos-server.org/neos/](http://www.neos-server.org/neos/)
  - [https://neos-server.org/neos/solvers/lp:Gurobi/AMPL.html](https://neos-server.org/neos/solvers/lp:Gurobi/AMPL.html)
Example

LP

\[
\begin{align*}
\text{max z} & = 3x_1 + 2x_2 \\
\text{subject to} & \\
2x_1 + x_2 & \leq 5 \\
-x_1 + 4x_2 & \leq 3 \\
x_i & \geq 0
\end{align*}
\]

AMPL file example.mod

\[
\begin{align*}
\text{var x\{i in 1..2\} >= 0;} \\
\text{maximize z: 3*x[1] + 2*x[2];} \\
\text{subject to c1: 2*x[1]+1*x[2]<=5;} \\
\text{subject to c2: -x[1]+4*x[2]<=3;}
\end{align*}
\]
Start `ampl` and then type

```plaintext
# commands to run in AMPL
ampl: reset;          # get rid of old data
ampl: model example.mod;  # choose our model
ampl: option solver lpsolve;  # set the backend
ampl: solve;
ampl: display x;      # output the results
```

- You could just type these commands at the AMPL prompt
- Could also run them from a script
- Remember the semi-colons.
Programming Style

Mixes

- **Imperative:**
  - sequences of commands to execute
  - focus on *how* to perform the task

- **Declarative:**
  - describe the problem, not how to solve it
  - focus on *what* the task should achieve

- **Interpreted:**
  - executed in source code form
  - can interact with interpreter (like Matlab)
Models

**Variables**

```plaintext
var x{i in 1..2} >= 0;
```

- or could have named variables, e.g., amount_of_paint
- and we can have *sets*, and other constructs

**Objective**

```plaintext
maximize z: 3*x[1] + 2*x[2];
```

- we can put a wide range of mathematical expressions here

**Constraints**

```plaintext
subject to c1: 2*x[1]+1*x[2]<=5;
subject to c2: -x[1]+4*x[2]<=3;
```

- we can put a wide range of mathematical expressions here
- constraints have names, e.g., c1, which could use later
Another Example

- Index values can be from an arbitrary set
- Coefficients and variables can be specified as vectors or matrices

Example

```plaintext
set possibilities := {"A", "B", "C"};

param a{possibilities};
param b;
param c{possibilities};

var x{possibilities} integer;

maximize profit: sum{i in possibilities} c[i]*x[i];

subject to limit1: sum{i in possibilities} a[i]*x[i] <= b;
subject to limit2{i in possibilities}: 0<= x[i] <= 1;
```
Data and Model Separation

- Why separate data and model?
  - model might actually be very small when you remove repeated bits
    - \( x_i \geq 0 \) for all \( i \)
  - model might be static, but data changes
    - *e.g.*, prices change
  - data might be in another file
    - *e.g.*, spreadsheet or database
  - conceptually easier to understand

- What is separation
  - model shows mathematical structure
  - data fills in the coefficients, which are called *parameters*

- We use notation much like standard mathematical notation
Example: model

Example (Example model file)

```ampl
## Introduction to AMPL - A Tutorial, by Kaminsky and Rajan
## Example 2

param n;
param t;
param p{i in 1..n};
param r{i in 1..n};
param m{i in 1..n};

var paint{i in 1..n} >= 0 integer;

maximize profit: sum{i in 1..n} p[i]*paint[i];

subject to time: sum{i in 1..n} (1/r[i])*paint[i] <= t;
subject to capacity{i in 1..n}: 0 <= paint[i] <= m[i];
```
Example: data

Example (Example data file)

```plaintext
## Introduction to AMPL - A Tutorial, by Kaminsky and Rajan
## Example 2

param n := 2;
param t := 40;

param p := 1 10
      2 15;

param r := 1 40
      2 30;

param m := 1 1000
      2 860;
```

Example: data
Example

Example (Example commands)

```AMPL
# commands to run in AMPL
ampl: reset;              # get rid of old data
ampl: model test.mod;    # choose our model
ampl: data test.dat;     # specify data
ampl: option solver lpsolve; # set the backend
ampl: solve;
ampl: display paint;    # output the results
```

Note we input the model before the data, and we reset first, to clear any old definitions that might conflict.
Data and Model Separation

Note that

- $n$ is set in the data
  - we can change the number of types of paint easily
- The coefficients of the constraints, and the objective are set in the data
  - we can change these as the market changes
  - we could choose more meaningful names for everything
- The expression of all the constraints is done very concisely
  - makes it easier to get it right (less typos)
Data and Model Separation

Model

files

Data

interpreter

AMPL

solution

back end

Solver
Advanced AMPL

- We can go way beyond this
  - all sorts of constraints
  - all sorts of objective
  - general sets of objects
  - 2D arrays of parameters
  - data from files

- Solvers (backends)
  - Ipsoolve
  - CPLEX
  - MINOS
  - Gurobi

Each can handle different size problems, and different types of constraints (e.g., MINOS can’t do Integer problems).
Where to next?

- We will use AMPL in some practical questions
  - you’ll get some more help
  - you might need to read some of the other available resources to fill in gaps
  - I may have some more examples in lectures
- You can use it to solve some assignment questions or in your project
  - but read questions carefully – some expect you to use it, and other ask not
  - and make sure you understand the results
- There will be a question in the Exam
Takeaways

- Matlab has a solver for ILPs
  - it’s useful when we want an integrated environment to create, solve, and visualise our problem
- AMPL (or another modelling language) is the way most industrial mathematicians should approach big problems
  - it’s natural
  - multiple backends
  - separation of data and model is very valuable
- We’re going to spend some time now to understand how these might work
Further reading I