THE LP PROBLEM IN STANDARD FORM

\[
\min_{x \in \mathbb{R}^n} c'x, \\
Ax = b, \quad x \geq 0.
\]

- \( x \geq 0 \) means \( x_i \geq 0, \ i = 1, \ldots, n \).

- \( A \) of size \( r \times n \) is supposed to have **full rank** \( r \).

- \( \Omega \) is a **polytope** (**polyhedron** if bounded).

- This is a **convex** optimization problem \( \Rightarrow \) KKT conditions sufficient for a global minimum.
GEOMETRY OF THE FEASIBLE SET

Definition: The point $x_e \in \partial \Omega (= \text{the boundary of } \Omega)$ is an extreme point if

$$x_e = \theta y + (1 - \theta) z , \quad y, z \in \Omega , \quad 0 < \theta < 1$$

implies that $y = z = x_e$.

Where are the extreme points for a line segment, for $\mathbb{R}$ and $\mathbb{R}^n_+$, a cube, and a sphere (all sets closed)?

The extreme points for $\Omega$ are the vertices.
**Definition:** A feasible point $x$ ($x \geq 0, Ax = b$) is called a *basic point* if there is an index set $\mathcal{B} = \{i_1, \cdots, i_r\}$, where the corresponding subset of columns of $A$,

$$\{a_{i_1}, \cdots, a_{i_r}\},$$

are linearly independent, and $x_i = 0$ for all $i \notin \mathcal{B}$.

If $x_i$ happens to be 0 also for some $i \in \mathcal{B}$, we say that the basic point is *degenerate*.

For a basic point, the corresponding $r \times r$ matrix

$$B = \begin{bmatrix} a_{i_1}, & \cdots, & a_{i_r} \end{bmatrix},$$

will be *non-singular*, and the equation $Bx_B = b$ has a unique solution.
The Fundamental Theorem for LP (N&W Theorem 13.2):

1. If $\Omega \neq \emptyset$, it contains basic points.

2. If there are optimal solutions, there are optimal basic points (basic solutions).

Theorem (N&W Theorem 13.3): The basic points are the extreme points of $\Omega$.

The number of basic points is between 1 (because of the first statement in the Fundamental Theorem) and $\binom{n}{r}$. 
THE SIMPLEX ALGORITHM

- The *Simplex Algorithm* is reported to have been discovered by G. B. Dantzig in 1947.

- The idea of the Simplex Algorithm is to search for the minimum by going from vertex to vertex (from basic point to basic point) in $\Omega$.

- Hand calculations are *never used* anymore!

The Simplex Iteration Step

We assume that the problem has the standard form, and that we are located in a basic point which, after a rearrangement of variables, has the form

$$x = \begin{bmatrix} x_B \\ 0 \end{bmatrix}.$$
The partition is therefore according to $A = [B \ N]$, where $B$ is non-singular, and

$$Ax = [B \ N] \begin{bmatrix} x_B \\ 0 \end{bmatrix} = Bx_B = b.$$ 

Split a general $x \in \Omega$ in the same way,

$$Ax = [B \ N] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = Bx_1 + Nx_2 = b.$$ 

Hence,

$$x_1 = B^{-1} (b - Nx_2) = x_B - B^{-1} Nx_2.$$ 

Note also that

$$f(x) = c' x = [c_1 \ c_2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$= c'_1 x_1 + c'_2 x_2$$

$$= c'_1 (x_B - B^{-1} Nx_2) + c'_2 x_2$$

$$= c'_1 x_B + (c'_2 - c'_1 B^{-1} N) x_2$$

Around $[x_B \ 0]'$, we may express both $x_1$ and $f(x)$ in terms of $x_2$. 
We are located at $x_1 = x_B$, $x_2 = 0$, and try to change one of the components $(x_2)_j$ of $x_2$ so that

$$f(x) = c_1' x_B + (c_2' - c_1' B^{-1}N) x_2$$

decreases.

- If $(c_2' - c_1' B^{-1}N) \geq 0 \Rightarrow$ FINISHED!

Assume that $(c_2' - c_1' B^{-1}N)_j < 0$:

- If all components of $x_1$ increase when $(x_2)_j$ increases, then
  $$\min c' x = -\infty.$$

$\Rightarrow$ FINISHED!

If not, we have the situation shown in Fig. 1.
The Simplex algorithm always converges if all basic points are non-degenerate.

Degenerate basic point: *Try a different component of $x_2$. (FINISHED if impossible!)*

It is straightforward to construct a generalized Simplex Algorithm for bounds of the form

$$l_i \leq x_i \leq u_i, \ i = 1, \ldots, n.$$
• If we $LU$-factorize $B$ once, we can update the factorization with the new column without making a complete new factorization (N&W, Sec. 13.4).

• It is often preferable to take the "steepest ridge" (fastest decrease in the objective) out from where we are (N&W, Sec. 13.5).
Starting the Simplex Method

The Simplex method consists of two phases:

- Phase 1: *Find a first basic point*

- Phase 2: *Solve the original problem*

The Phase 1 algorithm:

1. Turn the signs in $Ax = b$ so that $b \geq 0$.

2. Introduce additional variables $y \in \mathbb{R}^r$ and solve the extended problem

\[
\min (y_1 + \cdots + y_r),
\]

\[
[A ~ I] \begin{bmatrix} x \\ y \end{bmatrix} = b, \quad x, y \geq 0.
\]

(Note that $[0 ~ b]'$ already is a basic point for the extended problem!).
Assume that the solution of the extended problem is
\[
\begin{bmatrix}
  x_0 \\
y_0
\end{bmatrix}.
\]

- If \( y_0 \neq 0 \), then the original problem is infeasible \( (\Omega = \emptyset) \).

- If \( y_0 = 0 \), then \( x_0 \) is a basic point (= possible start for the original problem).

- This is not the only Phase 1 algorithm.

1 **EPILOGUE**

- Open Problem: *Are there LP algorithms of polynomial complexity?*
• The Simplex Method has exponential complexity in the worst case (Kree–Minty–Cheval counterexample)

• Interior Point Methods (Khachiyan, 1978): $\#Op \propto O\left(n^4 L\right)$

• Karmarkar (1984): $\#Op \propto O\left(n^{3.5} L\right)$

• Current record (?): Interior Barrier Primal–Dual methods, $\#Op \propto O\left(n^3 L\right)$. (We return to this method after discussing penalty and barrier methods)

• Solving large LP problems is BIG business!

• Entering data into the computer for large LP problems is a lot of work. Look up a description of the industry standard "MPS Data Format" on the internet.
LINEAR PROGRAMMING IN
MATLAB OPTIMIZATION TOOLBOX
(may be a little outdated!)

Basic function:  **linprog**

Solves the general LP-problem

\[
\begin{align*}
\min_x & \quad f'x, \\
\text{s.t.} & \quad Ax \leq b, \\
& \quad A_{eq} \cdot x = b_{eq}, \\
& \quad lb \leq x \leq ub
\end{align*}
\]

where \( f, x, b, b_{eq}, lb, \) and \( ub \) are vectors and \( A, A_{eq} \) are matrices (may be entered as \textit{sparse} matrices)

**Syntax:**

\[
\begin{align*}
x & = \text{linprog( } f, A, b, \text{ Aeq, beq) } \\
x & = \text{linprog( } f, A, b, \text{ Aeq, beq, lb, ub) } \\
x & = \text{linprog( } f, A, b, \text{ Aeq, beq, lb, ub, x0) } \\
x & = \text{linprog( } f, A, b, \text{ Aeq, beq, lb, ub, x0, options) }
\end{align*}
\]

\[
\begin{align*}
[x,fval] & = \text{linprog( } \ldots ) \\
[x,fval,exitflag] & = \text{linprog( } \ldots ) \\
[x,fval,exitflag,output] & = \text{linprog( } \ldots ) \\
[x,fval,exitflag,output,lambda] & = \text{linprog( } \ldots )
\end{align*}
\]
Example: The Standard form:

$$\min c'x,$$

$$Ax = b,$$

$$x \geq 0.$$ 

$$x = \text{linprog}(c,[ ],[ ],A,b,\text{zeros(size(c))},[ ])$$

- Note the Matlab convention with *placeholders*, "[ ]"

**INPUT:**

$$x_0$$: Starting point. Used only for medium problems (*Nelder-Mead amoeba*).

**Options**: Structure of parameters

- **LargeScale**: 'on'/'off'
- **Display**: 'off'/'iter'/'final' (large scale problems)
- **MaxIter**: Max number of iterations
- **Simplex**: 'on'/'off' ('on' ignores $$x_0$$)
- **TolFun**: Objective tolerance (large scale problems)
OUTPUT:

x,fval: Solution and objective

exitflag:
1 Iteration terminated OK
0 Number of iterations exceeded MaxIter
-2 No feasible point found
-3 Problem is unbounded
-4 NaN value encountered
-5 Both primal and dual are infeasible
-7 Search direction became too small

output: Structure of iteration information

iterations: Number of iterations
algorithm: Algorithm used
cgiterations: The number of PCG iterations (large-scale algorithm only)
message: Output message

lambda: Structure of Lagrange multipliers

ineqlin: for linear inequalities $Ax \leq b$, 
eqlin for linear equalities $A_{eq}x = b_{eq}$,
lower for lb,
upper for ub.

ALGORITHMS:

Small/Medium scale: SIMPLEX-like including Phase 1
Large scale: Primal-dual inner method
EXAMPLES FROM THE DOCUMENTATION

A. Small Problem

Find \( x \) that minimizes

\[
  f(x) = -5x_1 - 4x_2 - 6x_3
\]

subject to

\[
  x_1 - x_2 + x_3 \leq 20
\]
\[
  3x_1 + 2x_2 + 4x_3 \leq 42
\]
\[
  3x_1 + 2x_2 \leq 30
\]
\[
  0 \leq x_1, 0 \leq x_2, 0 \leq x_3
\]

First, enter the coefficients, then call LINPROG:

\[
  f = [-5 \ -4 \ -6]';
  A = [
    1 \ -1 \ 1 \\
    3 \ 2 \ 4 \\
    3 \ 2 \ 0 
  ];
  b = [20 \ 42 \ 30]';
  lb = zeros(3,1);
  [x,fval,exitflag,output,lambda] = ... linprog(f,A,b,[],[],lb);
\]

\[
  x = [0 \ 15 \ 3]
\]
\[
  fval = -78.0
\]

output:

- iterations: 6
- algorithm: 'large-scale: interior point' (!)
- cgiterations: 0
- message: 'Optimization terminated.'

\[
  \lambda.\text{ineqlin} = [0 \ 1.5 \ 0.5]
\]
\[
  \lambda.\text{lower} = [1 \ 0 \ 0]
\]
For solution by the Simplex method:

\[
f = [-5, -4, -6]';
A = [ 1 -1  1
     3  2  4
     3  2  0 ];
b = [20  42  30]';
lb = zeros(3,1);
\]

options = optimset('LargeScale','off','Simplex','on');
[x,fval,exitflag,output,lambda] = ... linprog(f,A,b,[],[],lb,[],[],options);

(NB! If you forget enough placeholders, [ ] , you get the error message "LINPROG only accepts inputs of data type double")

Now output gives:

\[
\begin{align*}
\text{iterations:} & \quad 3 \\
\text{algorithm:} & \quad \text{'medium scale: simplex'} \\
\text{cgiterations:} & \quad [] \\
\text{message:} & \quad \text{'Optimization terminated.'}
\end{align*}
\]

(same solution!)
B Medium Problem

This problem is stored as a Matlab MAT-file.

- 48 unknowns
- 30 inequality constraints
- 20 equality constraints
- \( x \geq 0 \)

Entered into Matlab simply by

```
load sc50b
```

\[
\begin{align*}
A & \quad 30 \times 48 \quad \text{(sparse)} \\
A_{\text{eq}} & \quad 20 \times 48 \quad \text{(sparse)} \\
b & \quad 30 \times 1 \\
beq & \quad 20 \times 1 \\
f & \quad 48 \times 1 \\
lb & \quad 48 \times 1
\end{align*}
\]

**Sparsity patterns:**

- For the matrix \( A \) (inequalities)
- For the matrix \( A_{\text{eq}} \) (equalities)
load sc50b
options = optimset('LargeScale','off','Simplex','on');
[x,fval,exitflag,output,lambda] = ...
    linprog(f,A,b,Aeq,beq,lb,[],[],options);

x = [ 30  28 42 ... 102.4870]

Only lambda.ineqlin(2) and lambda.ineqlin(3) equal to 0:
    only inequality 2 and 3 non-active.

max(lambda.lower)= 8.2808e-015 ⇒  x_i > 0 for i = 1,...,48.

output =
    iterations:  43
    algorithm:  'medium scale: simplex'
    cgiterations:  []
    message:  'Optimization terminated.'

Large scale option:

options = optimset('LargeScale','on');
[x,fval,exitflag,output,lambda] = ...
    linprog(f,A,b,Aeq,beq,lb,[],[],options);

output =
    iterations:  8
    algorithm:  'large-scale: interior point'
    cgiterations:  0
    message:  'Optimization terminated.'

Same solution!
With display of results for each iteration:

```matlab
options = optimset('LargeScale','on','Display','iter');
```

<table>
<thead>
<tr>
<th>Iter</th>
<th>Primal Infeas</th>
<th>Dual Infeas</th>
<th>Duality Gap</th>
<th>Total Rel Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A*x-b</td>
<td>A'*y+z-f</td>
<td>x'*z</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1.50e+03</td>
<td>2.19e+01</td>
<td>1.91e+04</td>
<td>1.00e+02</td>
</tr>
<tr>
<td>1</td>
<td>1.15e+02</td>
<td>3.18e-15</td>
<td>3.62e+03</td>
<td>9.90e-01</td>
</tr>
<tr>
<td>2</td>
<td>8.32e-13</td>
<td>1.96e-15</td>
<td>4.32e+02</td>
<td>9.48e-01</td>
</tr>
<tr>
<td>3</td>
<td>3.51e-12</td>
<td>1.87e-15</td>
<td>7.78e+01</td>
<td>6.88e-01</td>
</tr>
<tr>
<td>4</td>
<td>1.81e-11</td>
<td>3.50e-16</td>
<td>2.38e+01</td>
<td>2.69e-01</td>
</tr>
<tr>
<td>5</td>
<td>2.63e-10</td>
<td>1.23e-15</td>
<td>5.05e+00</td>
<td>6.89e-02</td>
</tr>
<tr>
<td>6</td>
<td>5.88e-11</td>
<td>2.72e-16</td>
<td>1.64e-01</td>
<td>2.34e-03</td>
</tr>
<tr>
<td>7</td>
<td>2.61e-12</td>
<td>2.59e-16</td>
<td>1.09e-05</td>
<td>1.55e-07</td>
</tr>
<tr>
<td>8</td>
<td>7.97e-14</td>
<td>5.67e-13</td>
<td>1.09e-11</td>
<td>3.82e-12</td>
</tr>
</tbody>
</table>

Optimization terminated.

FOR MORE INFO: Read documentation of `linprog`!
OPTIMIZATION SOFTWARE – 2010

(NEOS = Network-Enabled Optimization System)

- **AIMMS** modeling system
- **AMPL** modeling language.
- **ANALYZE** linear programming model analysis.
- **APOPT** - nonlinear programming.
- **APMonitor** modeling language.
- **ASA** - adaptive simulated annealing.
- **BPMPD** - linear programming.
- **BQPD** - quadratic programming.
- **BT** - minimization.
- **BTN** - block truncated Newton.
- **CBC** - mixed-integer linear programming.
- **CML** - constrained maximum likelihood.
- **CNM** - linear algebra and minimization.
- **CO** - constrained optimization.
- **COMPACT** - design optimization.
- **CONOPT** - nonlinear programming.
- **CONSOL-OPTCAD** - engineering system design.
- **CONTIN** - systems of nonlinear equations.
- **CLP** - linear programming.
- **CPLEX** - linear programming.
- **C-WHIZ** - linear programming models.
- **DATAFORM** - model management system.
- **DFNLP** - nonlinear data fitting.
- **DOC** - Design Optimization Control Program.
- **DONLP2** - nonlinear constrained optimization.
- **DOT** - Design Optimization Tools.
- **EASY FIT** - parameter estimation in dynamic systems.
- **Excel and Quattro Pro Solvers** - spreadsheet-based linear, integer and nonlinear programming
- **EZMOD** - modeling environment for decision support systems
- **FortMP** - linear and mixed integer quadratic programming.
- **FSQP** - nonlinear and minmax constrained optimization, with feasible iterates.
- **GAMS** - General Algebraic Modeling System.
- **GAUSS** - matrix programming language.
- **GENESIS** - structural optimization software.
- **GENOS 1.0** - nonlinear network optimization.
- **GINO** - nonlinear programming.
- **GRG2** - nonlinear programming.
- **GOM** - Global Optimization for Mathematica.
- **GUROBI** - linear programming.
- **HOMPACK** - nonlinear equations and polynomials.
- **HOPDM** - linear programming (interior-point).
- **HARWELL Library** - linear and nonlinear programming, nonlinear equations, data fitting.
- **HS/LP Linear Optimizer** - linear programming.
- **ILOG** - constraint-based programming and nonlinear optimization.
- **IMSL** - Fortran and C Library.
- **IPOPT** - nonlinear programming.
- **KNITRO** - nonlinear programming.
- **KORBX** - linear programming.
- **LAMPS** - linear and mixed-integer programming.
- **LANCELOT** - large-scale problems.
- **LBFGS** - unconstrained minimization.
- **LBFGS-B** - bound-constrained minimization.
- **LGO IDE** - continuous and Lipschitz global optimization.
• **LINDO** - linear, mixed-integer and quadratic programming.
• **LINGO** - modeling language.
• **LIPSOL** - linear programming.
• **LNOS** - linear programming/network flow problems.
• **LOQO** - Linear programming, unconstrained and constrained nonlinear optimization.
• **LP88 and BLP88** - linear programming.
• **LSGRG2** - nonlinear programming.
• **LSNNO** - large scale optimization.
• **LSSOL** - least squares problems.
• **M1QN3** - unconstrained optimization.
• **MATLAB** - optimization toolbox.
• **MAXLIK** - maximum likelihood estimation.
• **MCS** - global optimization.
• **MILP88** - mixed integer programming.
• **MINOS** - linear programming and nonlinear optimization.
• **MINTO** - mixed integer linear programming.
• **MINPACK-1** - nonlinear equations and least squares.
• **MIPIII** - mixed integer programming.
• **MODFIT** - parameter estimation in dynamic systems.
• **MODLER** - linear programming modeling language.
• **MODOLOPT** - unconstrained problems and simple bounds.
• **MOSEK** - linear programming and convex optimization.
• **MPL** - modeling system
• **MPSIII** - mathematical programming system.
• **NAG C Library** - nonlinear and quadratic programming, minimization
• **NAG Fortran Library** - nonlinear and quadratic programming, minimization
• **NETFLOW** - network optimization.
• **NITSOL** - systems of nonlinear equations.
• **NLopt** - local and global nonlinear optimization, including nonlinear constraints, with and without user-supplied gradients
• **NLPE** - minimization and least squares problems.
• **NLPJOB** - Mulicriteria optimization.
• **NLPQL** - nonlinear programming.
• **NLPQLB** - nonlinear programming with constraints.
• **NLSSOL** - constrained nonlinear least squares problems.
• **NLPSPR** - nonlinear programming.
• **NOVA** - nonlinear programming.
• **NPSOL** - nonlinear programming.
• **ODRPACK** - NLS and ODR problems.
• **OML** - linear and mixed-integer programming, model management.
• **OPL Studio** - optimization language and solver environment.
• **OPTDES** - design optimization tool.
• **OPTECH** - global optimization.
• **OptiA** - unconstrained, constrained, quadratic, minimax, nonsmooth, and global optimization.
• **OPTIMA Library** - optimization and sensitivity analysis.
• **OPTIMAX** - component software for optimization.
• **OPTMUM** - optimization.
• **OPTPACK** - constrained and unconstrained optimization.
• **OptQuest** - constrained and unconstrained optimization.
• **OSL** - linear optimization.
• **PLAM** - algebraic modeling language for mixed integer programming, constraint logic programming, etc.
• **PORT 3** - minimization, least squares, etc.
• **PROC LP** - linear and integer programming.
• **PROC NETFLOW** - network optimization.
• **PROC NLP** - various quadratic and nonlinear optimization problems.
• **PROPT** - optimal control software for MATLAB users.
• **Q01SUBS** - quadratic programming for matrices.
• **QAPP** - quadratic assignment problems.
• **QL** - quadratic programming.
• **QPOPT** - linear and quadratic problems.
• **RANDMOD** - linear programming model randomizer.
• **SCIP** - mixed-integer linear programming.
• **SIMUSOLV** - modeling software.
• **SPRNLNP** - sparse and dense nonlinear programming, sparse nonlinear least squares, including the **SCCS** package for optimal control.
• **SPEAKEASY** - numerical problems and operations research.
• **SNOPT** - large-scale quadratic and nonlinear programming problems.
• **SQOPT** - large-scale linear and convex quadratic programming problems.
• **SQP** - nonlinear programming.
• **SYMPHONY** - mixed-integer linear programming.
• **SYNAPS Pointer** - multidisciplinary design optimization software.
• **SYSFIT** - parameter estimation in systems of nonlinear equations.
• **TENMIN** - unconstrained optimization.
• **TENSOLVE** - nonlinear equations and least squares.
• **TN/TNBC** - minimization.
• **TNPACK** - nonlinear unconstrained minimization.
• **TSA88** - network linear programming.
• **TOMLAB** - Matlab Optimization.
• **UNCMIN** - unconstrained optimization.
• **VE08** - nonlinear optimization.
• **VE10** - nonlinear least squares.
• **VIG and VIMDA** - decision support system.
• **What'sBest** - linear and mixed integer programming.
• **WHIZARD** - linear programming, mixed-integer programming.
• **XLSOL** - Linear, integer and nonlinear programming for AMPL models.
• **XPRESS-MP** from Dash Associates - linear and integer programming.
Minimum Cost Network Flow Analysis Using LP

Harald E. Krogstad
March 2007
An arc is characterized by

- Prize pr. flow unit along arc
- Lower bound (for initiating transport)
- Upper bound (capacity)

Sources: (Production/providers)
- Capacity
- Cost pr. unit delivered to the network

Sinks (Consumers/receivers):
- Capacity
- Income to network from deliveries

Source: Production $b > 0$.
Sink: Absorption, $b < 0$. 

Source

Sink

弧是 Characterized by

- 奖金每流量单位沿弧
- 下限 (用于启动运输)
- 上限 (容量)

源：(生产/提供者)
- 容量
- 成本每单位交付给网络

接收者 (消费者/接收者)：
- 容量
- 收入给网络从递送

源：生产 $b > 0$。
接收者：吸收，$b < 0$。
Variables  \( x = \{ x_i \}, x_i \geq 0 \). (flow in the arcs)

NB! 2 variables for each arc: 2 directions

\[ \sum_{\text{inflow}} x_i = \sum_{\text{outflow}} x_i \]

Node:

\[ b_s = \sum_{\text{outflow}} x_i - \sum_{\text{inflow}} x_i \]

Source/Sink:

A balanced network:

\[ \sum_{\text{Sources/sinks}} b_s = 0 \]

Price for delivery:

\[ f(x) = \sum_{\text{arcs}} c_i x_i = c'x \]
Cost for one unit along arc “$i$”: $\{c_i\}$
Upper bound on capacity for arc “$i$”: $\{ub_i\}$
Lower bound on capacity for arc “$i$”: $\{lb_i\}$

The LP formulation:

$$\min \; c'x$$
$$\sum_{\text{outflow}} x_i - \sum_{\text{inflow}} x_i = b_n, \; n = 1, \ldots, \text{Nodes},$$
$$lb \leq x \leq ub.$$
Simsys_sparse

MATLAB Central
An open exchange for the MATLAB and Simulink user community

http://www.mathworks.com/matlabcentral/

Per Bergström
Luleå University of Technology
RANDOM NETWORK GENERATION

Prescribe:
- Numbers of sources and sinks
- Max number of arcs around one node
- Min number of arcs around one node
- Random upper bound
- Distribution of nodes
- Interactive network modification
- Random costs

The algorithm provides:
- Number of nodes
- Upper bound of capacity
- $A_{eq}$ matrix
- Balanced production/consumption at the sources and sinks

```matlab
[Aeq,beq,lb,ub,c]=simsys_sparse(100);
Solution in Matlab: x = linprog(c,[],[],Aeq,beq,lb,ub)
```
The LP-problem:

- Number of arcs: 304
- Lower bounds: 0
- Upper bounds: -
- Equality constraints: 48

$A_{eq}$-matrix:
Costs

Cost pr. flow unit vs Arc number

Costs
LP solution

Flow (x)

Arc

50 100 150 200 250 300
\[ \dim(x) = 3782 \]

\[ \dim(A_{eq}) = 506 \times 3782 \]
Practical Optimization: A Gentle Introduction
John W. Chinneck
Systems and Computer Engineering
Carleton University
Ottawa, Ontario K1S 5B6
Canada
http://www.sce.carleton.ca/faculty/chinneck/po.html

(very soft introduction 😊)