

Analytics Software Tools

Optimization Module

Chris McCord

August 29, 2018

MIT Operations Research Center

*Thanks to Jack Dunn and Daisy Zhuo for their help preparing the slides.

1. Introduction to Optimization
 - General Optimization
 - Linear Optimization
2. Overview of Optimization in Practice
 - Optimization Solvers
 - Algebraic Modeling Languages
3. Intro to Julia
4. Guide to MIO modeling and solvers

Introduction to Optimization

What is Optimization?

Mathematically, optimization can be defined as the following:

Maximize (or minimize) the **objective function** $f(\mathbf{x})$
by choosing the **decision variables** \mathbf{x} ,
subject to some **constraints** $g(\mathbf{x}) \leq \mathbf{b}$.

What is Optimization?

A typical formulation looks like this:

$$\begin{array}{ll}\max & f(\mathbf{x}) \\ \text{s.t.} & g(\mathbf{x}) \leq \mathbf{b} \\ & x_i \in \mathbb{R} \quad \forall i = 1, 2, \dots, n\end{array}$$

What is Optimization?

Or could look like this:

$$\begin{aligned} \min_{r,s,\theta,\mathbf{y}_1,\mathbf{y}_2,\mathbf{Z}} \quad & r + s & (D.13) \\ s.t. \quad & \begin{pmatrix} r + \mathbf{y}_1^{+T} \hat{\mathbf{u}}^{(0)} - \mathbf{y}_1^{-T} \hat{\mathbf{u}}^{(N+1)} & \frac{1}{2}(\mathbf{q} - \mathbf{y}_1)^T, \\ \frac{1}{2}(\mathbf{q} - \mathbf{y}_1) & \mathbf{Z} \end{pmatrix} \succeq \mathbf{0}, \\ & \begin{pmatrix} r + \mathbf{y}_2^{+T} \hat{\mathbf{u}}^{(0)} - \mathbf{y}_2^{-T} \hat{\mathbf{u}}^{(N+1)} + \theta t - 1 & \frac{1}{2}(\mathbf{q} - \mathbf{y}_2 - \theta \mathbf{v})^T, \\ \frac{1}{2}(\mathbf{q} - \mathbf{y}_2 - \theta \mathbf{v}) & \mathbf{Z} \end{pmatrix} \succeq \mathbf{0}, \\ & s \geq (\gamma_2^B \hat{\Sigma} + \hat{\mu} \hat{\mu}^T) \circ \mathbf{Z} + \hat{\mu}^T \mathbf{q} + \sqrt{\gamma_1^B} \|\mathbf{q} + 2\mathbf{Z} \hat{\mu}\|_{\hat{\Sigma}^{-1}}, \\ & \mathbf{y}_1 = \mathbf{y}_1^+ - \mathbf{y}_1^-, \quad \mathbf{y}_2 = \mathbf{y}_2^+ - \mathbf{y}_2^-, \quad \mathbf{y}_1^+, \mathbf{y}_1^-, \mathbf{y}_2^+, \mathbf{y}_2^- \theta \geq \mathbf{0}. \end{aligned}$$

Reference: Nathan Kallus (Operations Research Center Ph.D. '15) thesis

What is Optimization?

Or could look like this:

$$\begin{aligned}
 \min \quad & \frac{1}{L} \sum_{t \in \mathcal{T}_L} L_t + \alpha \cdot \sum_{t \in \mathcal{T}_B} \sum_{j=1}^p s_{jt} & (28) \\
 \text{s.t.} \quad & L_t \geq N_t - N_{kt} - n(1 - c_{kt}), & k = 1, \dots, K, \forall t \in \mathcal{T}_L, \\
 & L_t \leq N_t - N_{kt} + nc_{kt}, & k = 1, \dots, K, \forall t \in \mathcal{T}_L, \\
 & L_t \geq 0, & \forall t \in \mathcal{T}_L, \\
 & N_{kt} = \frac{1}{2} \sum_{i=1}^n (1 + Y_{ik}) z_{it}, & k = 1, \dots, K, \forall t \in \mathcal{T}_L, \\
 & N_t = \sum_{i=1}^n z_{it}, & \forall t \in \mathcal{T}_L, \\
 & \sum_{k=1}^K c_{kt} = l_t, & \forall t \in \mathcal{T}_L, \\
 & a_m^T x_t + \mu \leq b_m + (2 + \mu)(1 - z_{it}), & i = 1, \dots, n, \forall t \in \mathcal{T}_B, \forall m \in A_L(t), \\
 & a_m^T x_t \geq b_m - 2(1 - z_{it}), & i = 1, \dots, n, \forall t \in \mathcal{T}_B, \forall m \in A_R(t), \\
 & \sum_{t \in \mathcal{T}_L} z_{it} = 1, & i = 1, \dots, n, \\
 & z_{it} \leq l_t, & \forall t \in \mathcal{T}_L, \\
 & \sum_{i=1}^n z_{it} \geq N_{\min} l_t, & \forall t \in \mathcal{T}_L, \\
 & \sum_{j=1}^p \hat{a}_{jt} \leq d_t, & \forall t \in \mathcal{T}_B, \\
 & \hat{a}_{jt} \geq a_{jt}, & j = 1, \dots, p, \forall t \in \mathcal{T}_B, \\
 & \hat{a}_{jt} \geq -a_{jt}, & j = 1, \dots, p, \forall t \in \mathcal{T}_B, \\
 & -s_{jt} \leq a_{jt} \leq s_{jt}, & j = 1, \dots, p, \forall t \in \mathcal{T}_B, \\
 & s_{jt} \leq d_t, & j = 1, \dots, p, \forall t \in \mathcal{T}_B, \\
 & \sum_{j=1}^p s_{jt} \geq d_t, & \forall t \in \mathcal{T}_B, \\
 & -d_t \leq b_t \leq d_t, & \forall t \in \mathcal{T}_B, \\
 & d_t \leq d_{p(t)}, & \forall t \in \mathcal{T}_B \setminus \{1\}, \\
 & z_{it}, l_t \in \{0, 1\}, & i = 1, \dots, n, \forall t \in \mathcal{T}_L, \\
 & d_t \in \{0, 1\}, & j = 1, \dots, p, \forall t \in \mathcal{T}_B.
 \end{aligned}$$

Reference: Dimitris Bertsimas and Jack Dunn. *Optimal classification trees*.
Machine Learning (2017): 1-44.

Why Optimization?

It is widely used in these application areas because:

- a natural way to model the world
- extensive flexibility in modeling choices
- tractability and scalability of modern solvers
- Growing interests in the application of ML and AI
- Industry shift of focus from predictive to prescriptive

Linear Optimization

Linear optimization in particular is a simple yet still powerful type. The objective function and the constraints are *linear* in the decision variables.

Here is a general formulation:

$$\begin{aligned} \min_{\mathbf{x}} \quad & c_1x_1 + c_2x_2 + \dots + c_nx_n \\ \text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \geq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \geq b_2 \\ & \dots \\ & a_{k1}x_1 + a_{k2}x_2 + \dots + a_{kn}x_n \geq b_k \end{aligned}$$

Or in matrix notation,

$$\begin{aligned} \min \quad & \mathbf{c}'\mathbf{x} \\ \text{s.t.} \quad & \mathbf{Ax} \geq \mathbf{b} \end{aligned}$$

Linear Optimization

Example: minimizing absolute sum of residuals (similar to least square regression). Given data \mathbf{x} and \mathbf{y} :

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^n |y_i - \boldsymbol{\beta}^T \mathbf{x}_i|$$

We can use a trick to linearize the absolute terms:

$$\begin{aligned} \min_{\boldsymbol{\beta}} \quad & \sum_{i=1}^n z_i \\ \text{s.t.} \quad & y_i - \boldsymbol{\beta}^T \mathbf{x}_i \leq z_i \quad \forall i = 1, \dots, n \\ & -y_i + \boldsymbol{\beta}^T \mathbf{x}_i \leq z_i \quad \forall i = 1, \dots, n \end{aligned}$$

Now you just need to solve it!

Algorithm for Linear Optimization

Many algorithms, with different performance, exist to date:

1. the *simplex method* developed in 1947
2. the *ellipsoid method* by Shor (1970), Yudin and Nemirovskii (1977)
3. the *interior point method* by Karmarkar (1984)
4. ...

1. Start with a question
2. Formulate the optimization problem
3. Collect data
4. **Solve it using an algorithm**
5. Evaluate the solutions and iterate

Topic of next section!

Overview of Optimization in Practice

Solving an optimization problem

We've analyzed a real-world problem, collected data, and formulated an optimization model that describes the scenario

How do we solve the problem? Should we just code up the simplex method from a textbook and pass in our problem? **X**

Real-world optimization is fraught with difficulties that don't exist in a textbook:

- Floating-point errors
- Ill-conditioned matrices and numerical instability
- Modifications to algorithms to speed things up or work around problems (e.g. degeneracy cycling)

Leave writing the optimization algorithms to the experts!

Use an off-the-shelf **optimization solver**

Types of optimization problem

- **Linear:** everything is linear/affine
- **Quadratic:** quadratic objective and constraints
- **Conic:** variables and constraints lie in cones
- **Convex:** constraints and objective are convex
- **Nonlinear:** everything else...
- **Integer:** some variables restricted to integer values

Optimization solvers

www.juliaopt.org

Solver	Linear / Quadratic	Convex		Nonconvex	Integer	License
		Conic	Smooth			
Clp	✓					Open
Cbc	✓				✓	Open
GLPK	✓				✓ ^{cb}	Open
ECOS	✓	✓				Open
SCS	✓	✓				Open
CPLEX	✓	✓			✓ ^{cb}	Comm. ^a
Gurobi	✓	✓			✓ ^{cb}	Comm. ^a
FICO Xpress	✓	✓			✓	Comm. ^a
Mosek	✓	✓	✓		✓	Comm. ^a
NLopt			✓	✓		Open
Ipopt	✓		✓	✓		Open
Bonmin	✓		✓	✓	✓	Open
Couenne	✓		✓	✓	✓	Open
Artelys Knitro	✓		✓	✓	✓	Comm.

Limits of optimization

Some very rough limits of what problem sizes can be handled by state-of-the-art optimization solvers:

- **Linear:** 1,000,000s of variables
- **Quadratic:** 100,000s
- **Nonlinear:** 1000s*
- **Integer Linear:** 100,000s**

* (or 10s)

** (or 100s)

Using an optimization solver

How we see the problem:

$$\begin{aligned} \min \quad & \sum_{(i,j) \in E} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{(i,j) \in E} x_{ij} = \sum_{(j,k) \in E} x_{jk} \quad j = 2, \dots, n-1 \\ & \sum_{(i,n) \in E} x_{in} = 1 \\ & 0 \leq x_{ij} \leq c_{ij} \end{aligned}$$

What the solver wants:

Inputs: (A, b, c)

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

To use the solver directly, we would need to create the A matrix and b/c vectors from our problem

- Slow (both to implement and when running)

Forming A/b/c ourselves?

$$\text{Minimize } y = -x_1 - 2x_2$$

Subject to

$$2x_1 + x_2 + S_1 = 10$$

$$x_1 + x_2 + S_2 = 6$$

$$-x_1 + x_2 + S_3 = 2$$

$$-2x_1 + x_2 + S_4 = 1$$

$$x_1, x_2, S_1, S_2, S_3, \text{ and } S_4 \geq 0$$

The arguments for the Matlab command $X=\text{LINPROG}(f, A, b, Aeq, beq, LB, UB)$ are given as

$$f = \begin{bmatrix} -1 \\ -2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, A = 0, b = 0, Aeq = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 1 & 0 \\ -2 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, beq = \begin{bmatrix} 10 \\ 6 \\ 2 \\ 2 \\ 0 \\ 0 \end{bmatrix}, LB = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, UB = \begin{bmatrix} \infty \\ \infty \\ \infty \\ \infty \\ \infty \\ \infty \end{bmatrix}$$

The following Matlab statements are used to solve this linear programming problem.

Matlab Example -----

% Example 8.3-2

```
f=[-1;-2;0;0;0;0];
```

```
A=zeros(6,6);
```

```
b=zeros(6,1);
```

```
Aeq=[2 1 1 0 0 0; 1 1 0 1 0 0; -1 1 0 0 1 0; -2 1 0 0 0 1; 0 0 0 0 0 0; 0 0 0 0 0 0];
```

```
beq=[10;6;2;1;0;0];
```

```
LB=[0;0;0;0;0;0]; UB=[inf;inf;inf;inf;inf;inf];
```

```
x=linprog(f,A,b,Aeq,beq,LB,UB)
```

Talk directly to a solver?

We can talk directly to solvers using a normal programming language (that the solver supports!)

- Tied to that solver — have to start over to try another solver
- Not easy to understand — made for coders not optimizers

```
import ilog.concert.*;
import ilog.cplex.*;
public class Example {
    public static void main(String[] args) {
        try {
            IloCplex cplex = new IloCplex();
            double[] lb = {0.0, 0.0, 0.0};
            double[] ub = {40.0, Double.MAX_VALUE, Double.MAX_VALUE}; IloNumVar[] x =
            cplex.numVarArray(3, lb, ub);
            double[] objvals = {1.0, 2.0, 3.0};
        }
        6
        cplex.addMaximize(cplex.scalProd(x, objvals));
        cplex.addLe(cplex.sum(cplex.prod(-1.0, x[0]),
            cplex.prod( 1.0, x[1]),
            cplex.prod( 1.0, x[2])), 20.0);
        cplex.addLe(cplex.sum(cplex.prod( 1.0, x[0]),
            cplex.prod(-3.0, x[1]),
            cplex.prod( 1.0, x[2])), 30.0);
        if ( cplex.solve() ) {
            cplex.out().println("Solution status = " + cplex.getStatus()); cplex.out().println("Solution
            value = " + cplex.getObjValue());
            double[] val = cplex.getValues(x);
            int ncols = cplex.getNcols();
            for (int j = 0; j < ncols; ++j)
                cplex.out().println("Column: " + j + " Value = " + val[j]);
        }
        cplex.end();
    }
    catch (IloException e) {
        System.err.println("Concert exception '" + e + "' caught");
    }
}
```

Excel Solver?

Excel Solver gives a point-and-click interface for solving optimization problems in Excel

- Slow, hard to scale to large problems

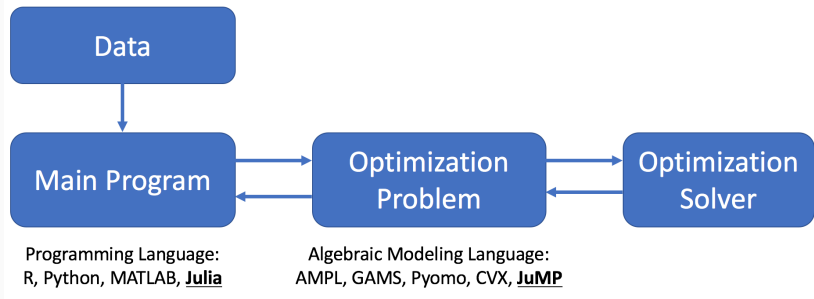
The screenshot displays the Microsoft Excel interface with a spreadsheet titled "Panel Manufacturing.xlsx". The spreadsheet contains data for a panel manufacturing optimization problem. The Solver Parameters dialog box is open, showing the following settings:

- Set Objective:** \$F\$5
- To:** ☒ Max ☐ Min ☐ Value Of: 0
- By Changing Variable Cells:** \$B\$4:\$E\$4
- Subject to the Constraints:** (Empty list)
- ☐ Make Unconstrained Variables Non-Negative
- Select a Solving Method:** Simplex LP
- Solving Method:** Select the GRG Nonlinear engine for Solver Problems that are smooth nonlinear. Select the LP Simplex engine for linear Solver Problems, and select the Evolutionary engine for Solver problems that are non-smooth.

The spreadsheet data is as follows:

	Panel Type					
	Tahoe	Pacific	Savannah	Aspen		
Pallets	0	0	0	0	Total Profit	
Profit	\$450	\$1,150	\$800	\$400	\$0	
Resources Required per Pallet Type						
Glue	50	50	100	50	Used	Available
Pressing	5	15	10	5	0	730
Pine chips	500	400	300	200	0	29,200
Oak chips	500	750	250	500	0	60,500

Solving optimization problems in practice



In reality, we use an **Algebraic Modeling Language** to communicate with the solver

- We write down our model in code as we would on paper
- AML handles converting the model into solver-format and getting results from solver

Our problem in the JuMP AML

```
m = Model()
@variable(m, 0 <= x[e in E] <= e.capacity)
@constraint(m, flowcon[j=2:n-1],
    sum(x[e], e in E, e.to == j)
    ==
    sum(x[e], e in E, e.from == j)
)
@constraint(m, sum(x[e], e in edges, e.to == n) == 1)
@objective(m, Min, sum(e.cost * x[e], e in edges))
```

The code for our model reads almost the same as the mathematical version!

The Julia Programming Language



www.julialang.org

- High-level, high-performance, open-source dynamic language for technical computing
- Allows use to keep the productivity of dynamic languages without giving up any speed
- Developed here at MIT, relatively new (released in 2012)

How fast is Julia?





www.juliaopt.org

- Open-source, free, fast algebraic modeling language for Julia
- Created by MIT ORC students (Iain, Miles and Joey plus others)
- Easy to learn, use and read
- As fast as comparable commercial software

JuMP

(open-source, Julia, 2013)

```
mcf = Model()
@variable(mcf, 0 <= flow[e in edges] <= e.capacity)
@constraint(mcf, sum{flow[e], e in edges; e.to==5} == 1)
@constraint(mcf, flowcon[n=2:4], sum{flow[e], e in edges; e.to==node}
            == sum{flow[e], e in edges; e.from==node})
@objective(mcf, Min, sum{e.cost * flow[e], e in edges})
```

Pyomo

(open-source, Python, 2008)

```
mcf = ConcreteModel()
mcf.flow = Var(edges, bounds=lambda m,i,j: (0,capacity[(i,j)]))
mcf.uf = Constraint(expr=sum(mcf.flow[e] for e in edges if e[1]==5) == 1)
def con_rule(mcf,n): return sum(mcf.flow[e] for e in edges if e[1]==n) ==
                             sum(mcf.flow[e] for e in edges if e[0]==n)
mcf.flowcon = Constraint([2,3,4],rule=con_rule)
mcf.flowcost = Objective(expr=sum(cost[e]*mcf.flow[e] for e in edges))
```

JuMP vs. others—Syntax

JuMP

(open-source, Julia, 2013)

```
mcf = Model()
@variable(mcf, 0 <= flow[e in edges] <= e.capacity)
@constraint(mcf, sum{flow[e], e in edges; e.to==5} == 1)
@constraint(mcf, flowcon[n=2:4], sum{flow[e], e in edges; e.to==node}
            == sum{flow[e], e in edges; e.from==node})
@objective(mcf, Min, sum{e.cost * flow[e], e in edges})
```

GAMS

(commercial, custom language, 1976)

```
POSITIVE VARIABLE flow(nodefrom,nodeto); flow.UP(edges) = capacity(edges);
EQUATION unitflow;
unitflow.. sum{edges(nodefrom,lastnode), flow(nodefrom,lastnode)} =e= 1;
EQUATION flowcon(nodes);
flowcon(midnodes(n)).. sum{edges(nodefrom,n), flow(nodefrom,n)} =e=
                        sum{edges(n,nodeto), flow(n,nodeto)};
FREE VARIABLE obj;
EQUATION flowcost; flowcost.. obj =e= sum{edges, cost(edges)*flow(edges)};
MODEL mincostflow /all/; SOLVE mincostflow USING lp MINIMIZING obj;
```

JuMP

(open-source, Julia, 2013)

```
mcf = Model()
@variable(mcf, 0 <= flow[e in edges] <= e.capacity)
@constraint(mcf, sum{flow[e], e in edges; e.to==5} == 1)
@constraint(mcf, flowcon[n=2:4], sum{flow[e], e in edges; e.to==node}
            == sum{flow[e], e in edges; e.from==node})
@objective(mcf, Min, sum{e.cost * flow[e], e in edges})
```

AMPL

(commercial, custom language, 1985)

```
var flow{(i,j) in edges} >= 0.0, <= capacity[i,j];
subject to unitflow: sum{(i,5) in edges} flow[i,5] == 1;
subject to flowconserve {n in 2..4}:
    sum{(i,n) in edges} flow[i,n] == sum{(n,j) in edges} flow[n,j];
minimize flowcost:    sum{(i,j) in edges} cost[i,j] * flow[i,j];
```

Why do we use JuMP?

JuMP has become the de-facto AML of choice here at the ORC

- We use it in our research
- We teach it in our classes

Why?

- It's free and open-source—easy to add features, no contention for licenses
- It's embedded in an existing programming language—no need to switch to custom language just for optimization
- It's about as fast as the best commercial AMLs out there!

Intro to Julia

Basic arithmetic

Simple arithmetic:

$$3 + 7$$

$$4.8 / 9.3$$

$$11 ^ 5$$

Comparisons:

$$12 <= 5$$

$$!(5 == 8)$$

$$5 != 8$$

$$(2014 \% 4) == 0$$

Variables

Variables:

```
x = 5
y = 12
z = x - y
z
w
```

Everything has a type:

```
typeof(z)          # Int64
typeof(z + 1.5)    # Float64
typeof(z >= 0)     # Bool
```

Arrays, Vectors and Matrices

1D arrays (vectors) are for sequences of values, indexed from 1 to n:

```
v = [1, 2, 3]
```

```
v[1]      # access elements using [ ]; starts from 1
```

```
v[0]      # gives an error!
```

```
v[end]    # last element
```

We can also make 2D arrays (matrices):

```
A = [ 1  1  1;
```

```
     -1 -1 -2;
```

```
     0  0  0]
```

```
B = [0.1 0.5 0.4;
```

```
     0.2 0.3 0.5;
```

```
     0.8 0.1 0.1]
```

```
B[2, 3]
```

```
A + B
```

```
B ^ 100
```

```
B[1, :]
```

More arrays

Modifying arrays:

```
a = [1, 2, 3]
a[end + 1] = 4 # won't work!
```

Push/pop commands:

```
push!(a, 4)      # append 4 to end of a
pop!(a)          # remove last element of a; 4
a               # a is back to being [1, 2, 3]
z = []
push!(z, 5)      # what happens?
z = Int64[]
push!(z, 5)      # how about now?
push!(z, 1.5)    # and this?
```

Exercise 1

Create the arrays A , x and b as follows:

```
A = rand(10, 3); x = rand(3); b = rand(10)
```

Write an expression that evaluates to

- `true` if the inner product of the fourth row of A and x is greater than the fourth element of b ; and
- `false` otherwise.

What do you find?

Hint: you might want to use the `dot` function to compute the inner product. Type `?dot` to access the Julia help for this function

Lessons from Exercise 1

```
dot(A[4,:], x)                # 1.67087
dot(A[4,:], x) > b[4]         # true
```

How about if we didn't use dot?

```
A[4,:] * x                    # error, wrong dim
A[4,:]' * x                   # works now
A[4,:]' * x > b[4]            # error, num vs matrix
(A[4,:]' * x)[1] > b[4]       # works now
```

Creating arrays

```
ones(3)
```

```
zeros(4,5)
```

List comprehensions:

```
oddsnumbers = [2 * i - 1 for i in 1:10]
```

```
evennumbers = [j + 1 for j in oddsnumbers]
```

```
A = [i + j for i in 1:5, j in 1:5]
```

Printing things

```
a = 123.0
```

```
# printing without line break
```

```
# use comma to separate terms
```

```
print("The value of a = ", a); print(".")
```

```
# printing with line break
```

```
# use dollar sign within string for Julia expression
```

```
println("a is $a, a-10 is $(a-10)."); println("That's it.")
```


Conditionals

If-then-else:

```
x = 2
if x > 6
  x = x + 20
elseif (x < 5)
  x = factorial(x)
else
  println("Not changing x")
end
```

Loops

for loops:

```
for k = 1:5
    println(k ^ 2)
end
```

while loops:

```
x = 1
while x < 2000
    print("*")
    x = 2 * x
end
```

Functions

Functions are defined using the keyword function:

```
function convexcomb(x, y, theta)
    (1 - theta) * x + theta * y
end
```

Functions by default return the last statement, but you can also use return:

```
function convexcomb2(x, y, theta)
    if (theta < 0 || theta > 1)
        println("Bad theta!!!")
    else
        return (1 - theta) * x + theta * y
    end
end
```

Exercise 2

Write a function that takes a matrix A , vectors x and b , and returns a vector containing the row indices for which $Ax \geq b$ does not hold.

There are many ways to do this!

Hints:

- `.<` or `.<=` performs element-wise comparisons
- `find` function returns indices of non-zero elements for `Int64` or `Float64` arrays or true elements for `BitArrays`
- Use `?` to access the help

Example Solutions to Exercise 2

```
function findinds(x, A, b)
    find(A * x .< b)
end
```

```
function findinds(x, A, b)
    inds = collect(1:length(b))
    falseinds = A * x .< b
    inds[falseinds]
end
```

Example Solutions to Exercise 2

```
function findinds(x, A, b)
    inds = []
    for i in 1:length(b)
        diff = (dot(A[i,:], x) - b[i])[1,1]
        if (diff < 0)
            push!(inds, i)
        end
    end
    return inds
end
```

Another common way of interacting with Julia is using IJulia notebooks (similar to IPython)

To open IJulia:

```
using IJulia  
notebook()
```

Navigate to the folder with the iris.csv file. We are going to do some data work in an IJulia notebook.

Guide to MIO modeling and solvers

What is Integer Optimization?

Integer linear optimization is a surprisingly expressive class of problems of the form

$$\begin{array}{ll}\min_{\mathbf{x}} & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{R}^m \times \mathbb{Z}^n\end{array}$$

Essentially, this is linear optimization with a new primitive: integrality conditions on some of the variables.

Often we further restrict integer variables to be binary variables, $x \in \{0, 1\}$. This is a powerful way of modeling logical outcomes.

Practical Integer Optimization

Most real-world problems inevitably end up having to incorporate integer variables

- Most common: binary variables used to model yes/no decisions

Unfortunately, optimization with integer variables is much more difficult, and getting it right is **more of an art than a science**

To have the most success with MIO, it's important to know both how to formulate problems in a good way, and how to work effectively with mixed-integer optimization solvers

Progress of MIO in the past 25 years

- Speed up between CPLEX 1.2 (1991) and CPLEX 11 (2007): **29,000 times**
- Gurobi 1.0 (2009) comparable to CPLEX 11
- Speed up between Gurobi 1.0 and Gurobi 6.5 (2015): **73 times**
- Total software speedup: **2,100,000 times**
- A MIO problem that would have taken a year to solve 25 years ago can now be solved on the **same 25-year-old computer** in around 15 seconds.
- Hardware speedup: $10^{6.2} =$ **1,560,000 times**
- Total Speedup: **3.3 trillion times!**

Guide to linear MIO solvers

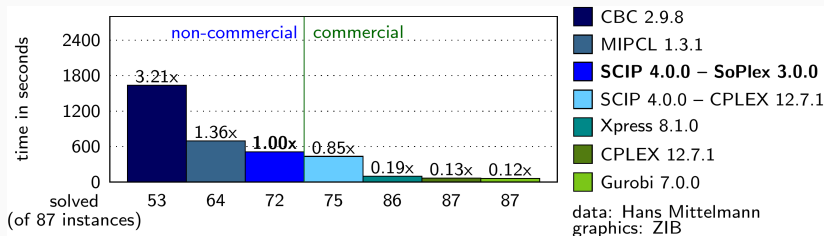
The commercial solvers are significantly faster than open-source alternatives:

- **Gurobi** is typically seen as the fastest, followed closely by
- **CPLEX** (owned by IBM), and then
- **Xpress** (owned by FICO)

Open-source MIO solvers:

- **CBC** is the best free and open-source solver
- **GLPK** is a much slower alternative
- **SCIP** is open-source, but only free for academic use

MIO solver speeds



Commercial MIO solvers are about 25x faster than CBC

We're going to focus on **Gurobi**:

- Fastest MIO solver
- Easy to get academic license
- Most research-friendly of the commercial solvers

Most of what we cover applies to the other solvers, with minor differences

Go to IJulia notebook: “6 — Working with MIO Solvers”

MIO modeling techniques

Now that we're familiar with Gurobi, we'll take a look at just a few of the things we can model using MIO

- Binary choices and boolean logic
- Logical implications using big-M constraints

This is by no means going to be exhaustive! A great reference for MIO modeling techniques is **MIP Formulations and Linearizations** by FICO:

<http://www.fico.com/en/node/8140?file=5125>

Logical Constraints — Boolean Logic

There is a natural mapping from the boolean TRUE and FALSE to the binary variables 1 and 0

It is also quite common to want to express logical constraints and implications in our optimization models, e.g.

- if this factory is built, we can build no other factories in this year
- either factory A or factory B can be built, but not both

The “trick” is expressing these logical constraints and implications as **linear inequalities**, and sometimes we need to introduce **auxiliary variables**

We'll start with modeling simple logical statements: AND & OR.

Boolean Logic — AND

z is TRUE if and only if x and y are both TRUE ($z = x \text{ AND } y$)

- $0 \text{ AND } 0 = 0$
- $1 \text{ AND } 0 = 0$
- $0 \text{ AND } 1 = 0$
- $1 \text{ AND } 1 = 1$

We can write this as

$$z \geq x + y - 1, \quad z \leq x, \quad z \leq y, \quad 0 \leq z \leq 1$$

which we can check by putting the values in for each case

Boolean Logic — OR

z is TRUE if and only if either x or y or both are TRUE ($z = x \text{ OR } y$)

- $0 \text{ OR } 0 = 0$
- $1 \text{ OR } 0 = 1$
- $0 \text{ OR } 1 = 1$
- $1 \text{ OR } 1 = 1$

This can also be expressed by linear inequalities, with no auxiliaries required:

$$z \geq x, \quad z \geq y, \quad z \leq x + y, \quad 0 \leq z \leq 1$$

Boolean Logic — XOR

z is TRUE if and only if either x or y is TRUE but not both
($z = x \text{ XOR } y$)

- $0 \text{ XOR } 0 = 0$
- $1 \text{ XOR } 0 = 1$
- $0 \text{ XOR } 1 = 1$
- $1 \text{ XOR } 1 = 0$

This one is a bit trickier. One way to do it is

$z = x \text{ XOR } y = (x \text{ OR } y) - (x \text{ AND } y)$, but then we would need two auxiliary variables, e.g. $z_1 = x \text{ OR } y$, $z_2 = x \text{ AND } y$, and then finally $z = z_1 - z_2$

Exercise: Express $z = x \text{ XOR } y$ with using ONE auxiliary variable.
Can you do it with NO auxiliary variables?

Logical Implications

We often need to model constraints that should only hold if a logical condition is satisfied. For example:

- We have a network with a 'source' and a 'sink', and want to maximize flow between them across the network. We have a set of arcs we can build, at a cost, so our objective is to maximize profit from the flow less the cost of building these arcs.

If we say that $x_{i,j}$ is the flow from i to j , and $z_{i,j}$ is a binary decision for whether we build the arc or not, we will have the constraint

$$x_{i,j} > 0 \implies z_{i,j} = 1$$

That is, if the flow on the arc is non-zero, we must build the arc

Big-M constraints for logical implications

Our logical constraint is:

$$x_{i,j} > 0 \implies z_{i,j} = 1$$

One way to express this as a linear equality is to write

$$0 \leq x_{i,j} \leq Mz_{i,j}$$

where M is a **sufficiently large constant**. That means that M is greater than the largest value $x_{i,j}$ would take in an optimal solution.

- if $z_{i,j} = 0$, we have $0 \leq x_{i,j} \leq 0$, so $x_{i,j} = 0$
- if $z_{i,j} = 1$, we have $0 \leq x_{i,j} \leq M$, so $x_{i,j}$ is unrestricted

Solution to XOR question

$$z \geq x - y$$

$$z \geq y - x$$

$$z \leq x + y$$

$$z \leq 2 - x - y$$

Check

- $x = 0, y = 0$: third constraint implies $z \leq 0$
- $x = 1, y = 0$: first constraint implies $z \geq 1$
- $x = 0, y = 1$: second constraint implies $z \geq 1$
- $x = 1, y = 1$: fourth constraint implies $z \leq 2 - 1 - 1 = 0$

Take-aways from Today

- **Julia** is a fast technical computing language with tools for data wrangling similar to R
- **Algebraic modeling languages** like **JuMP** make it easy to solve optimization problems
 - Why JuMP? It's fast, free, and embedded in a real programming language
- LO and MIO are powerful and flexible modeling frameworks, but...
- **Not all formulations are created equally!**
 - Different ways of expressing the same model can give very different behaviors
 - Understanding what the solver is doing is crucial for improving performance