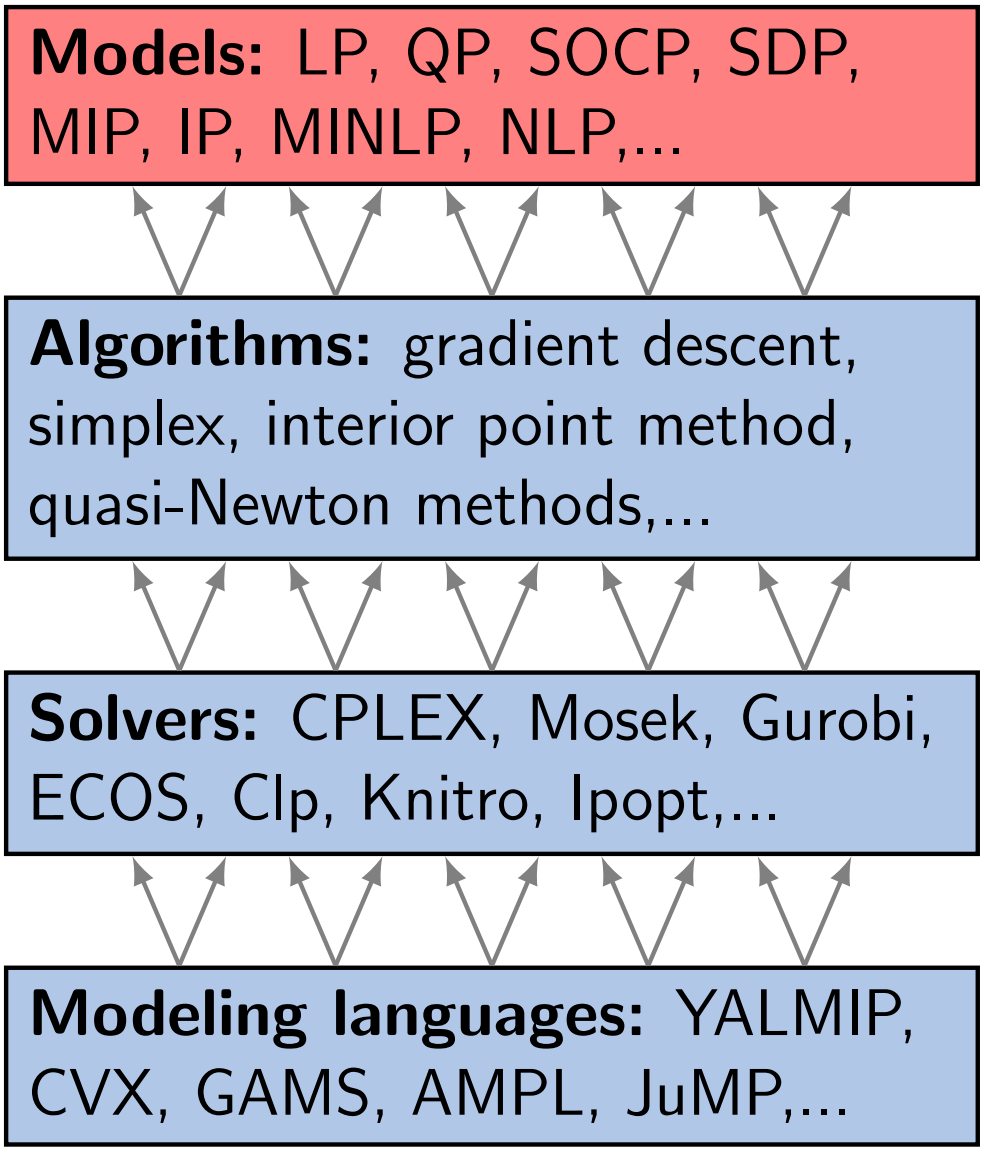
2. Introduction, part two

- Optimization hierarchy
- Available solvers in JuMP
- Writing modular code
- Geometrical intuition

Optimization hierarchy



Optimization models can be categorized based on:

- types of variables
- types of constraints
- type of objective

Example: every linear program (LP) has:

- continuous variables
- linear constraints
- a linear objective

We will learn about many other types of models.

Optimization hierarchy

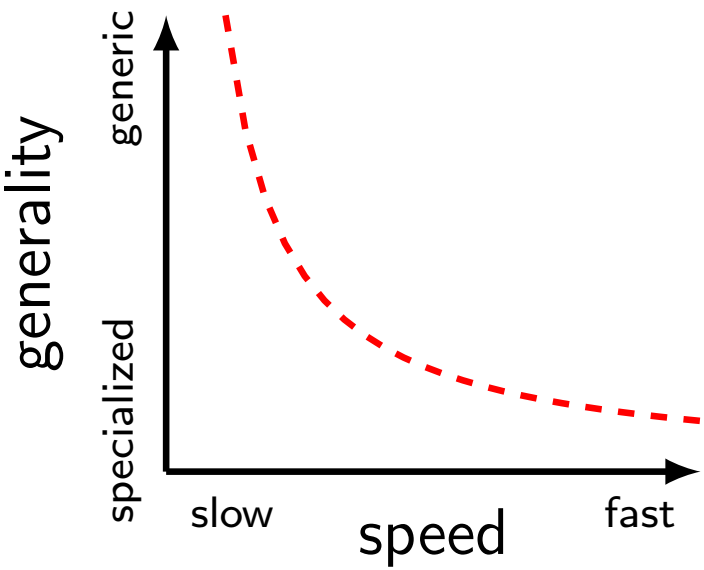
Models: LP, QP, SOCP, SDP, MIP, IP, MINLP, NLP,...

Algorithms: gradient descent, simplex, interior point method, quasi-Newton methods,...

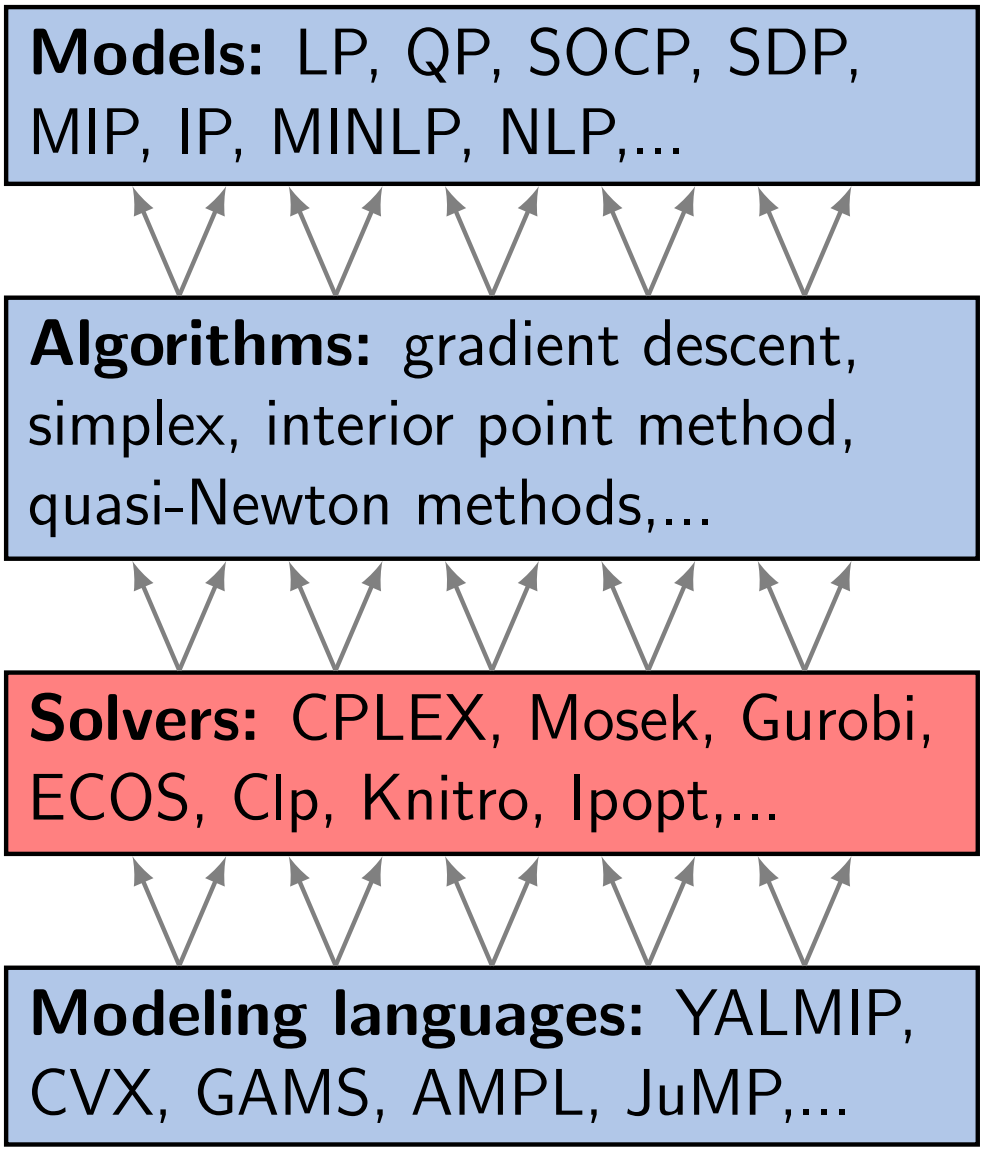
Solvers: CPLEX, Mosek, Gurobi, ECOS, Clp, Knitro, Ipopt,...

Modeling languages: YALMIP, CVX, GAMS, AMPL, JuMP,...

Numerical (usually iterative) procedures that can solve instances of optimization models. More specialized algorithms are usually faster.



Optimization hierarchy



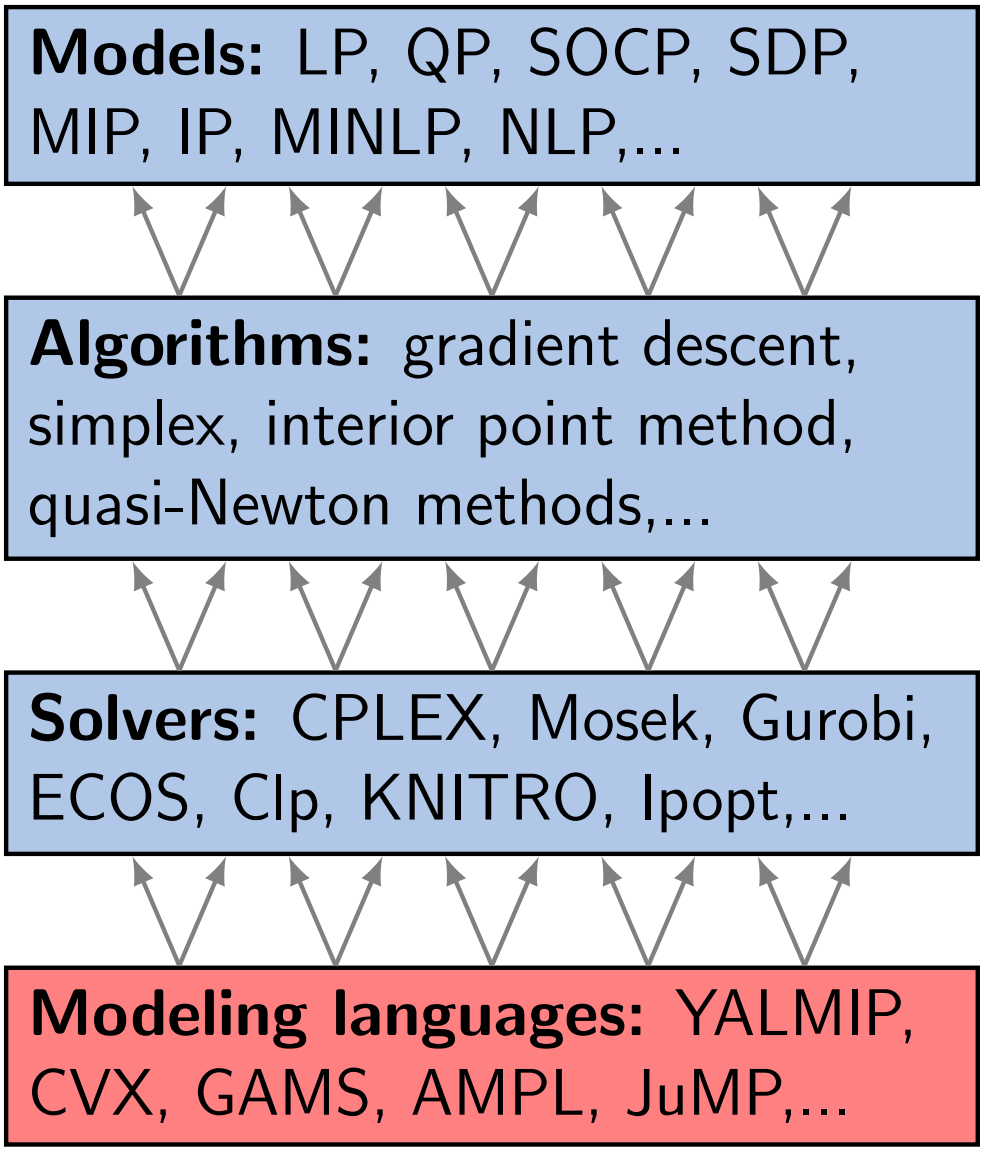
Solvers are *implementations* of algorithms. Sometimes they can be quite clever!

- typically implemented in C/C++ or Fortran
- may use sophisticated error-checking, complex heuristics etc.

Availability varies:

- some are open-source
- some are commercial
- some have .edu versions

Optimization hierarchy



Modeling languages provide a way to interface with many different solvers using a common language.

- Can be a self-contained language (GAMS, AMPL)
- Some are implemented in other languages (JuMP in Julia, CVX in Matlab)

Again, availability varies:

- some are open-source
- some are commercial
- some have .edu versions

Model classification

- The nature of the decision variables:
  - Default: continuous, e.g., $\mathbb{R}^n$.
  - Integer (**I**), e.g., binary or Integer
  - Mixed-integer (**MI**), e.g., a mix of both types
- The nature of the objective and constraint functions:
  - Linear (**L**)
  - Quadratic (**Q**)
  - Nonlinear (**NL**)
- Put a **P** at the end for "program".

Examples:

- QP: quadratic program
- MILP: mixed-integer linear program

Solvers in JuMP

About 50 solvers available in JuMP. They are open-source, or available with academic or commercial license. A selection:

- **LP:** HiGHS, GLPK, Clp, CPLEX, Gurobi
- **QP:** DAQP, HiGHS, OSQP, Gurobi
- **SOCP:** ECOS, Mosek
- **SDP:** SCS, COSMO, Mosek
- **NLP:** Ipopt, NLopt, Knitro
- **MILP:** HiGHS, GLPK, Cbc, Mosek, Gurobi
- **MICP:** HiGHS, Mosek, Gurobi
- **MINLP:** SCIP, NLopt, Baron, Knitro

Ordering is: (best free), . . . , (alternatives), . . . , (best academic/commercial)
Source: https://jump.dev/JuMP.jl/stable/installation/#Supported-solvers

Solvers in JuMP

Before solving a model, you must specify a solver.
You can do this when you declare the model:

```
using JuMP, Clp, ECOS, SCS
model = Model(Clp.Optimizer)
model = Model(ECOS.Optimizer)
model = Model(SCS.Optimizer)
```

You can change the solver later if you want:

```
model = Model()
set_optimizer(model, Clp.Optimizer)
optimize!(model)
set_optimizer!(model, ECOS.Optimizer)
optimize!(model)
```

Solver parameters

Solvers have settings that you can customize

```
m = Model()
set_optimizer(SCS.Optimizer)
# set relative tolerance
set_optimizer_attribute(m, "eps", 1e-5)
# set absolute tolerance
set_optimizer_attribute(m, "eps_abs", 1e-5)
```

Attribute names vary from solver to solver (annoying!)
one useful attribute that works across all solvers:

```
set_optimizer_attribute(m, MOI.Silent(),true)
```

(for suppressing solver output)

Solvers in JuMP

Before using a solver, you must "use" the appropriate packagess: `using JuMP, Clp`

Every solver must be installed before it can be used:

```
using Pkg; Pkg.add("Clp")
```

Some things to know:

- Installing a package may take a couple minutes, but it only has to be done once.
- All functions in Julia are slow the first time you use them. But they speed up greatly afterwards.
- Keep all your packages up-to-date using `Pkg.update()`

Solvers in JuMP

Top Brass.ipynb

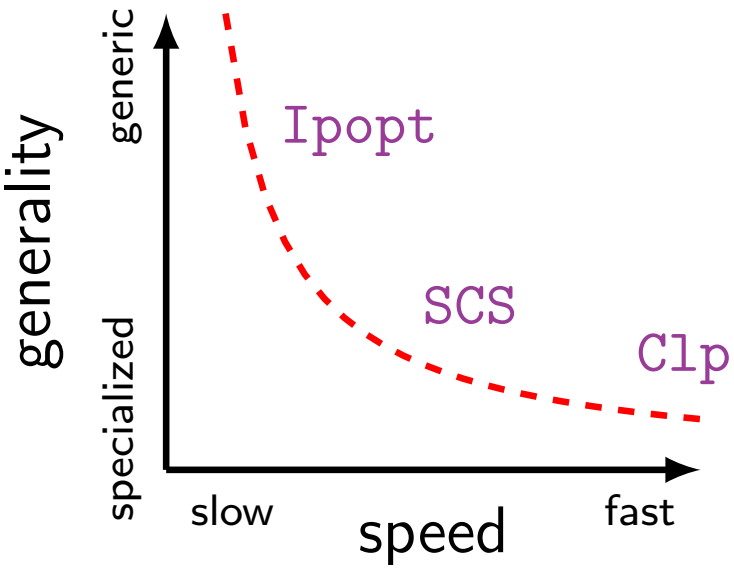
- Try `Clp`, `SCS`, `Ipopt` solvers. Is the answer the same?
- Measuring how long code takes to run:
  - `@time`: prints basic diagnostic info
  - `@timev`: prints verbose (extensive) diagnostic info
  - `@elapsed`: returns the time taken
- For one line: `@time ...` or `@time(...)`
  For a block of code: `@time begin ... end`
- What happens if an unsuitable solver is used?

Speed vs Generality

We will see later in the class that these models are nested:

$$\text{LP} \subseteq \text{SDP} \subseteq \text{NLP}$$

`SCS` (an SDP solver) is relatively slow at solving LPs because it solves them by first converting them to an SDP!



Writing modular code

It is good practice to separate the *data* from the *model*.

Top Brass 2.ipynb, Top Brass 3.ipynb

- Use *dictionaries* to make the code more modular
- Use *expressions* to make the code more readable
- Use `NamedArrays` for indexing over sets
- Try adding a new type of trophy!

Modeling languages

All modeling languages contain:

- A way of specifying the **variables**
- A way of specifying the **constraints**
- A way of specifying the **objective**
- A way of selecting a **solver**
- A solve command
- A way of displaying the results

Learning a modeling language amounts to learning the syntax for each of the parts above.

Comparison: GAMS (1)

JuMP and GAMS are structurally very similar

```
* TOP BRASS PROBLEM
set I/football, soccer/;
free variable profit "total profit";
positive variables x(I) "trophies";
* DATA section
parameters
         profit(I)   / "football" 12 , "soccer" 9 /
         wood(I)     / "football"  4 , "soccer" 2 /
         plaques(I)  / "football"  1 , "soccer" 1 /;
scalar
         quant_plaques  /1750/
         quant_wood     /4800/
         quant_football /1000/
         quant_soccer   /1500/;
* MODEL section
equations
obj    "max total profit"
foot   "bound on the number of brass footballs used"
socc   "bound on the number of brass soccer balls used",
plaq   "bound on the number of plaques to be used",
wdeq   "bound on the amount of wood to be used";
```

Comparison: GAMS (2)

```
* CONSTRAINTS
obj..
total_profit =e= sum(I, profit(I)*x(I));

foot..
I("football") =l= quant_football;

socc..
I("soccer") =l= quant_soccer;

plaq..
sum(I,plaques(I)*x(I)) =l= quant_plaques;

wdeq..
sum(I,wood(I)*x(I)) =l= quant_wood;

model topbrass /all/;

* SOLVE
solve topbrass using lp maximizing profit;
```

JuMP and GAMS are structurally very similar

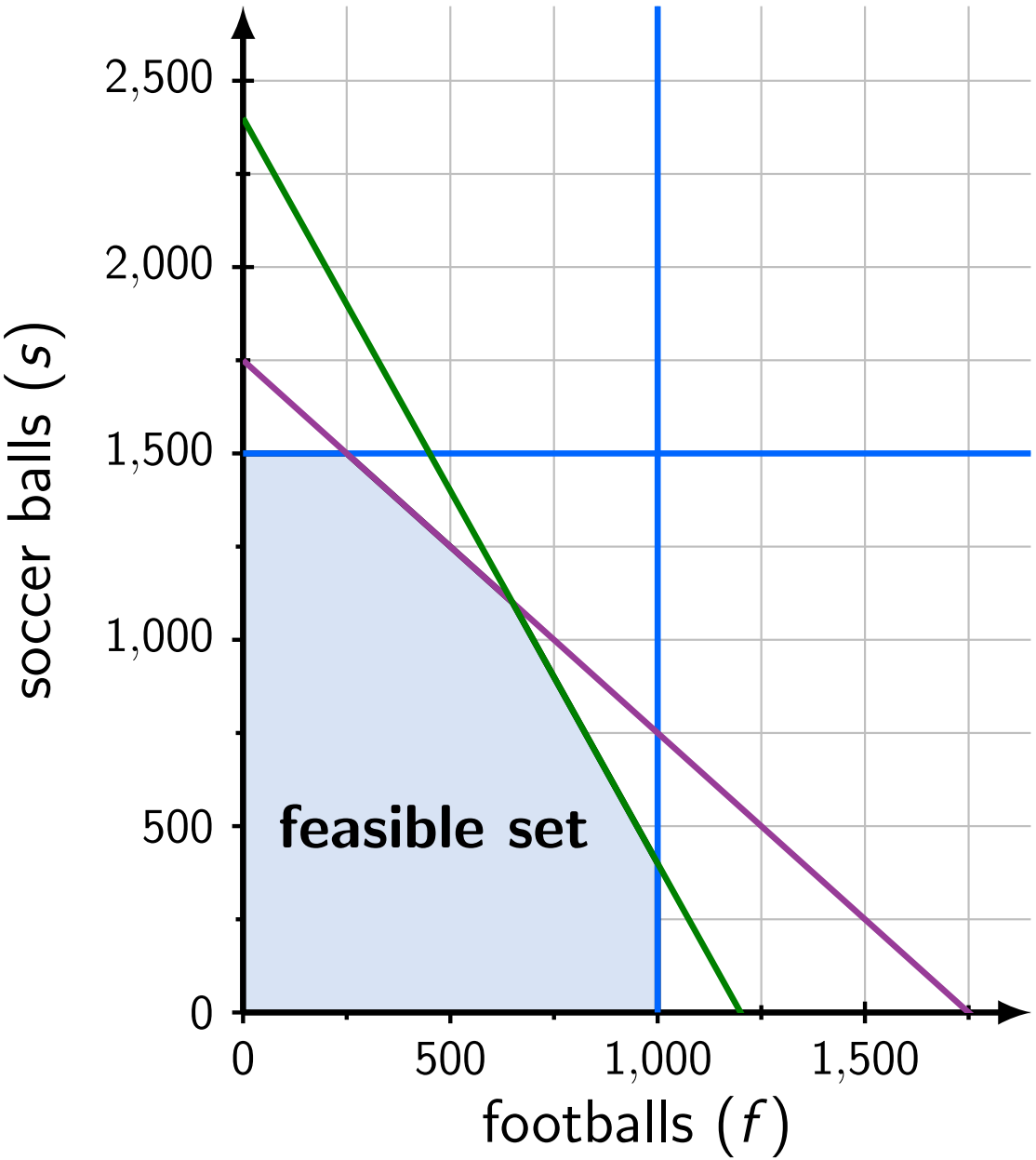
```
# Decision Variables
var f >= 0 integer;    # football trophies
var s >= 0 integer;    # soccer trophies

# Parameters
param f_profit := 12; # football trophy profit
param s_profit := 9;  # soccer trophy profit
param wood_f := 4;    # wood per football trophy
param wood_s := 2;    # wood per soccer trophy
param max_f := 1000;  # available brass footballs
param max_s := 1500;  # available brass soccer balls
param max_p := 1750;  # available plaques
param max_w := 4800;  # available wood

# Objective Function
maximize profit: f_profit * f + s_profit * s;

# Constraints
subject to w_cons: wood_f * f + wood_s * s <= max_w;
subject to f_cons: f <= max_f;
subject to s_cons: s <= max_s;
subject to p_cons: f + s <= max_p;

# Solve and display results
solve;
display f, s, profit;
```

same with AMPL...

...and with
CVX (Matlab)

```
cvx_begin
    % Define the variables
    variable f nonnegative
    variable s nonnegative

    % Parameters
    profit_football = 12; % Profit per football trophy
    profit_soccer = 9;    % Profit per soccer trophy
    wood_football = 4;    % Wood needed per football trophy
    wood_soccer = 2;      % Wood needed per soccer trophy
    max_wood = 4800;      % available wood
    max_football = 1000;  % available brass footballs
    max_soccer = 1500;    % available brass soccer balls
    max_plaques = 1750;   % available plaques

    % Objective Function: Maximize Total Profit
    maximize(profit_football * f + profit_soccer * s)

    % Constraints
    subject to
        wood_football * f + wood_soccer * s <= max_wood
        f <= max_football
        s <= max_soccer
        f + s <= max_plaques
cvx_end

% Display the results
fprintf('Maximum profit: $%.2f\n', cvx_optval);
```

Geometry of Top Brass



$$\max_{f,s} \quad 12f + 9s$$

$$\text{s.t.} \quad 4f + 2s \leq 4800$$

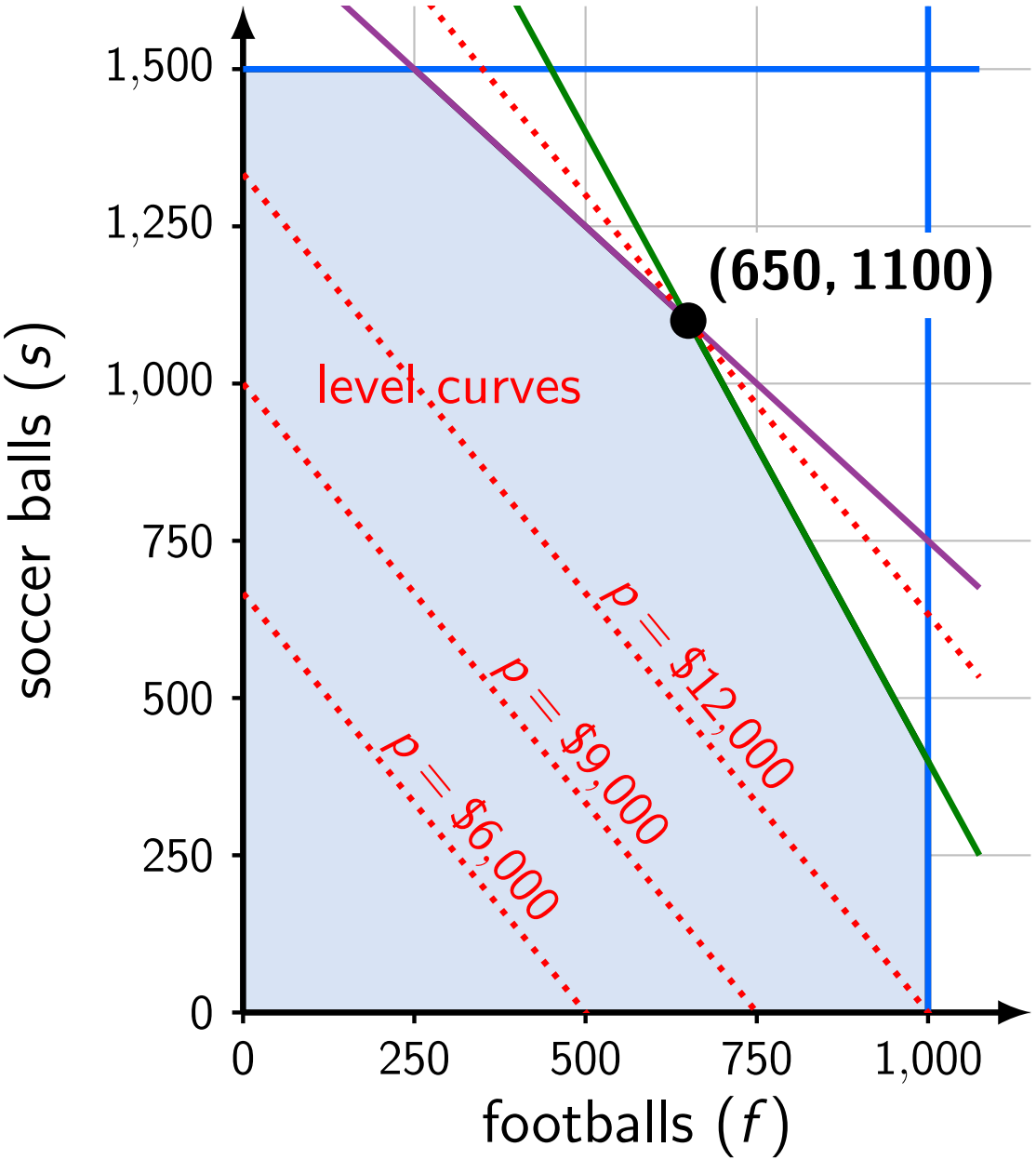
$$f + s \leq 1750$$

$$0 \leq f \leq 1000$$

$$0 \leq s \leq 1500$$

Each point (f, s) is a possible decision.

Geometry of Top Brass



$$\max_{f,s} \quad 12f + 9s$$

$$\text{s.t.} \quad 4f + 2s \leq 4800$$

$$f + s \leq 1750$$

$$0 \leq f \leq 1000$$

$$0 \leq s \leq 1500$$

Which feasible point has the max profit?

$$p = 12f + 9s$$

Exercise

Write an optimization model and solve graphically:

A calculator company produces a scientific calculator and a graphing calculator. Long-term projections indicate an expected demand of at least 100 scientific and 80 graphing calculators each day. Because of limitations on production capacity, no more than 200 scientific and 170 graphing calculators can be made daily. To satisfy a shipping contract, a total of at least 200 calculators much be shipped each day.

If each scientific calculator sold results in a \$2 loss, but each graphing calculator produces a \$5 profit, how many of each type should be made daily to maximize net profits?

Exercise

$$\begin{aligned} \underset{s,\,g}{\text{maximize}} \quad & 5g - 2s \\ \text{such that:} \quad & 100 \leq s \leq 200 \\ & 80 \leq g \leq 170 \\ & s + g \geq 200 \end{aligned}$$