Homework 1: ✓Checked on September 11.

Problem: Pick a small example from the AMPL book and write the corresponding LP in its original form, standard form and canonical form.
Solution: For this problem I used the very first LP presented in the book. It is presented as such:

$$\begin{aligned} \max \quad & 25X_B + 30X_c \\ \text{Subject To:} \quad & (1/200)X_B + (1/140)X_C \leq 40 \\ & 0 \leq X_B \leq 6000 \\ & 0 \leq X_C \leq 4000 \end{aligned}$$

Converting this to canonical form is easier than standard form so we'll start there.

Canonical Form

For this we want the following setup:

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

First we'll handle the conversion from max to min.

$$\max 25X_B + 30X_C = \min -25X_B - 30X_C$$

From here we need to account for something, we need all less than inequalities for our constraints however we have double inequalities. So we need to adjust those. Double inequalities aren't anything fancy really, they're just two sets of inequalities written in a more concise way.

On top of that, we need to capture all of the coefficients in these inequalities. Some of these constraints only have a single variable, but in a way they still contain both. The one that isn't present can be represented with a simple 0 coefficient. Capturing all of that information, let's begin.

First off,

$$\begin{aligned} 0 \leq X_B \leq 6000 &\iff 0 \leq X_B, X_B \leq 6000 \\ 0 \leq X_C \leq 4000 &\iff 0 \leq X_C, X_C \leq 4000 \end{aligned}$$

And,

$$\begin{aligned} 0 \leq X_B &\iff 0 \leq X_B + 0X_C \\ 0 \leq X_C &\iff 0 \leq X_C + 0X_B \end{aligned}$$

Now let's rewrite all of our constraints. I'll also be flipping these inequalities to ensure all of them are in the same direction.

$$\begin{aligned}
\frac{1}{200}X_B + \frac{1}{140}X_C &\leq 40 \\
X_B + 0X_C &\leq 6000 \\
0X_B + X_C &\leq 4000 \\
-X_B + 0X_C &\leq 0 \\
0X_B - X_C &\leq 0
\end{aligned}$$

Now we can rewrite all of what we have in vector/matrix notation and finish this up.

$$x = \begin{bmatrix} X_B \\ X_C \end{bmatrix}, c = \begin{bmatrix} -25 \\ -30 \end{bmatrix}$$

And now the constraints:

$$A = \begin{bmatrix} \frac{1}{200} & \frac{1}{140} \\ 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, b = \begin{bmatrix} 40 \\ 6000 \\ 4000 \\ 0 \\ 0 \end{bmatrix}$$

And so now in canonical form we have:

$$\begin{aligned}
\min \ & c^T x \\
\text{s.t.} \ & Ax \leq b
\end{aligned}$$

Standard Form

This modification isn't too bad going from canonical form now. We need slack variables to handle the inequalities but nothing too crazy.

Essentially, all we gotta do is create a slack variable s_i for all of the inequalities. These will be set up such that $s_i \geq 0$. These end up going with the non-negativity constraints on X_B, X_C, so we only need 3 slack variables in total. One for each of the main constraints.

For a simple example, the second inequality becomes $X_B + 0X_C + s_2 = 6000$.

$$\begin{aligned}
\frac{1}{200}X_B + \frac{1}{140}X_C + s_1 &= 40 \\
X_B + 0X_C + s_2 &= 6000 \\
0X_B + X_C + s_3 &= 4000 \\
X_B, X_C, s_1, s_2, s_3 &\geq 0
\end{aligned}$$

As these add new variables we adjust the matrices as such.

$$x = \begin{bmatrix} X_B & X_C & s_1 & s_2 & s_3 \end{bmatrix}^T$$

$$c = \begin{bmatrix} -25 & -30 & 0 & 0 & 0 \end{bmatrix}^T$$

$$A = \begin{bmatrix} \frac{1}{200} & \frac{1}{140} & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}, b = \begin{bmatrix} 40 \\ 6000 \\ 4000 \end{bmatrix}$$

So now we have everything. In standard form we have:

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

Homework 2

Given the system of equations, remove a set of 2 variables aside from $\{x_1, x_4\}$.

$$x_1 + 2x_2 + 3x_3 = 6$$
$$x_1 + x_2 + x_3 + x_4 = 4$$
$$x_1, x_2, x_3, x_4 \geq 0$$

For this homework we will remove $\{x_2, x_4\}$.
Step 1: Solve for x_2.

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 6 \\ 2x_2 &= 6 - 1x_1 - 3x_3 \\ x_2 &= 3 - \frac{1}{2}x_1 - \frac{3}{2}x_3 \end{aligned}$$

Step 2: Solve for x_4. Plug in x_2.

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 &= 4 \\ x_1 + 3 - \frac{1}{2}x_1 - \frac{3}{2}x_3 + x_3 + x_4 &= 4 \\ 3 + \frac{1}{2}x_1 - \frac{1}{2}x_3 + x_4 &= 4 \\ x_4 &= 1 + \frac{1}{2}x_3 - \frac{1}{2}x_1 \end{aligned}$$

Step 3: Handle non-negativity constraint. Simplify.

$$\begin{aligned} 3 - \frac{1}{2}x_1 - \frac{3}{2}x_3 &\geq 0 \\ -\frac{1}{2}x_1 - \frac{3}{2}x_3 &\geq -3 \\ \frac{1}{2}x_1 + \frac{3}{2}x_3 &\leq 3 \\ x_1 + 3x_3 &\leq 6 \end{aligned}$$

$$\begin{aligned} 1 + \frac{1}{2}x_3 - \frac{1}{2}x_1 &\geq 0 \\ x_1 - x_3 &\leq 2 \end{aligned}$$

So our new setup is:

$$\begin{aligned} x_1 + 3x_3 &\leq 6 \\ x_1 - x_3 &\leq 2 \\ x_1, x_3 &\geq 0 \end{aligned}$$

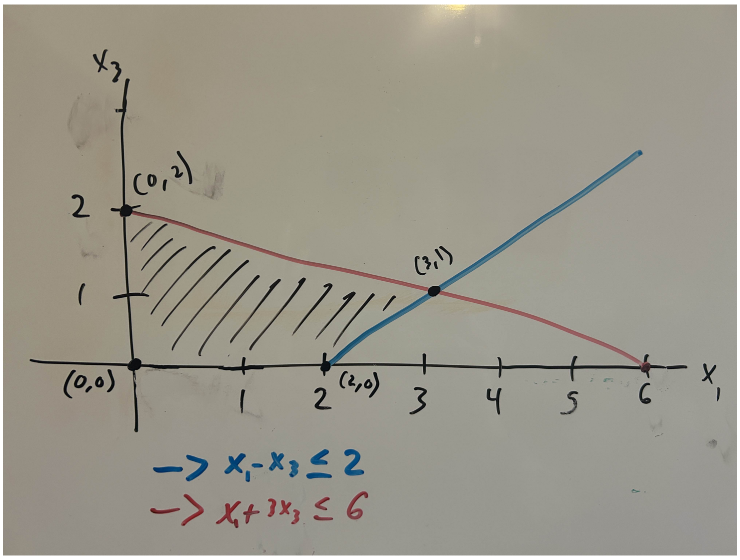


Figure 1: Feasible region of x_1 and x_3.

Homework 3

Problem Decide for our running example for which combinations of basic variables we get a basic feasible or infeasible solution via a computation. (From lecture 4).

I'll be using the same system of equations as in homework 2.

$$A = \begin{vmatrix} 1 & 2 & 3 & 0 \\ 1 & 1 & 1 & 1 \end{vmatrix}, \; b = \begin{pmatrix} 6 \\ 4 \end{pmatrix}$$

To get our basic solution we need to take A and choose 2 subsets of columns from it, B and N. The columns we choose for B (the basis) do not need to be consecutive. For each choice, the other two columns will go into N.

We then compute the basic solution as

$$x_B = B^{-1}b, \qquad x_N = 0.$$

A solution is feasible if all components of x are nonnegative. Below I will work through all six possible choices of bases.

—

Basis columns 1 and 2

$$B = \begin{vmatrix} 1 & 2 \\ 1 & 1 \end{vmatrix}$$

$$\begin{aligned} x_B &= B^{-1}b \\ &= \begin{vmatrix} -1 & 2 \\ 1 & -1 \end{vmatrix} \begin{pmatrix} 6 \\ 4 \end{pmatrix} \\ &= \begin{pmatrix} 2 \\ 2 \end{pmatrix} \end{aligned}$$

Thus the basic solution is

$$x = (2, 2, 0, 0).$$

All entries are nonnegative, so this solution is feasible.

—

Basis columns 1 and 3

$$B = \begin{vmatrix} 1 & 3 \\ 1 & 1 \end{vmatrix}$$

$$\begin{aligned} x_B &= B^{-1}b \\ &= \begin{vmatrix} -1 & 3 \\ 1 & -1 \end{vmatrix} \begin{pmatrix} 6 \\ 4 \end{pmatrix} \\ &= \begin{pmatrix} 3 \\ 1 \end{pmatrix} \end{aligned}$$

Thus the basic solution is

$$x = (3, 0, 1, 0).$$

All entries are nonnegative, so this solution is feasible.

—

Basis columns 1 and 4

$$B = \begin{vmatrix} 1 & 0 \\ 1 & 1 \end{vmatrix}$$

$$\begin{aligned} x_B &= B^{-1}b \\ &= \begin{vmatrix} 1 & 0 \\ -1 & 1 \end{vmatrix} \begin{pmatrix} 6 \\ 4 \end{pmatrix} \\ &= \begin{pmatrix} 6 \\ -2 \end{pmatrix} \end{aligned}$$

Thus the basic solution is

$$x = (6, 0, 0, -2).$$

Since $x_4 = -2 < 0$, this solution is infeasible.

—

Basis columns 2 and 3

$$B = \begin{vmatrix} 2 & 3 \\ 1 & 1 \end{vmatrix}$$

$$\begin{aligned} x_B &= B^{-1}b \\ &= \begin{vmatrix} -1 & 3 \\ 1 & -2 \end{vmatrix} \begin{pmatrix} 6 \\ 4 \end{pmatrix} \\ &= \begin{pmatrix} 6 \\ -2 \end{pmatrix} \end{aligned}$$

Thus the basic solution is

$$x = (0, 6, -2, 0).$$

Since $x_3 = -2 < 0$, this solution is infeasible.

—

Basis columns 2 and 4

$$B = \begin{vmatrix} 2 & 0 \\ 1 & 1 \end{vmatrix}$$

$$\begin{aligned} x_B &= B^{-1}b \\ &= \begin{vmatrix} 1 & 0 \\ -1 & 2 \end{vmatrix} \begin{pmatrix} 6 \\ 4 \end{pmatrix} \\ &= \begin{pmatrix} 3 \\ 1 \end{pmatrix} \end{aligned}$$

Thus the basic solution is

$$x = (0, 3, 0, 1).$$

All entries are nonnegative, so this solution is feasible.

—

Basis columns 3 and 4

$$B = \begin{vmatrix} 3 & 0 \\ 1 & 1 \end{vmatrix}$$

$$\begin{aligned} x_B &= B^{-1}b \\ &= \begin{vmatrix} 1 & 0 \\ -1 & 3 \end{vmatrix} \begin{pmatrix} 6 \\ 4 \end{pmatrix} \\ &= \begin{pmatrix} 2 \\ 2 \end{pmatrix} \end{aligned}$$

Thus the basic solution is

$$x = (0, 0, 2, 2).$$

All entries are nonnegative, so this solution is feasible.

—

Summary

Basis Columns	**Basic Solution x**	**Feasible?**
1, 2	(2, 2, 0, 0)	Yes
1, 3	(3, 0, 1, 0)	Yes
1, 4	(6, 0, 0, −2)	No
2, 3	(0, 6, −2, 0)	No
2, 4	(0, 3, 0, 1)	Yes
3, 4	(0, 0, 2, 2)	Yes

Homework 4: AMPL Book Exercise 1-2

Problem: The steel model for this chapter can be further modified to reflect various changes in production requirements. For each part below, explain the modifications to Figures 1-6a and 1-6b that would be required to achieve the desired changes. Make each change in isolation, not carrying modifications from part to part.

Reference Information

Before we begin, let's just keep the default info and solution up here as an easy reference.

Files: Figures 1-6a and 1-6b both use the **steel4** .dat and .mod files. So I'll be using them as a base and modifying them.

steel4.mod

```
set PROD;    # products
set STAGE;   # stages

param rate {PROD,STAGE} > 0; # tons per hour in each stage
param avail {STAGE} >= 0;     # hours available/week in each
    stage
param profit {PROD};          # profit per ton

param commit {PROD} >= 0;     # lower limit on tons sold in
    week
param market {PROD} >= 0;     # upper limit on tons sold in
    week

var Make {p in PROD} >= commit[p], <= market[p]; # tons
    produced

maximize Total_Profit: sum {p in PROD} profit[p] * Make[p];

               # Objective: total profits from all products

subject to Time {s in STAGE}:
   sum {p in PROD} (1/rate[p,s]) * Make[p] <= avail[s];

               # In each stage: total of hours used by all
               # products may not exceed hours available
```

steel4.dat

```
data;

set PROD := bands coils plate;
set STAGE := reheat roll;

param rate:  reheat  roll :=
  bands        200    200
  coils        200    140
  plate        200    160 ;

param:    profit  commit  market :=
  bands     25     1000    6000
  coils     30      500    4000
  plate     29      750    3500 ;

param avail :=  reheat 35   roll   40 ;
```

This provides the following solution:

$$\text{Total Profit} \approx 190071.43$$
$$\text{Bands} \approx 3357.14$$
$$\text{Coils} = 500$$
$$\text{Plates} \approx 3142.86$$

As for time used, we use 35 hours on the reheat stage and 40 hours on the roll stage.

A ✓Checked on September 11.

Problem: How would you change the constraints so that total hours used by all products must equal the total hours available for each stage? Solve the linear program and verify that you get the same results. Why is there no difference in solution?

Solution: All we need to do here is modify one line. Line 18 specifically. We change the $\leq$ to a strict $=$.

```
subject to Time: sum {p in PROD} (1/rate[p,s]) * Make[p] =
    avail[s];
```

The solution it gives is the exact same. This is because our goal is to produce as much as we can to maximize profit. So the original solution is already using up all of the available hours. We can check this programmatically and check the hours both solutions used. I don't have that included in here, but I personally verified that this was the case.

B

Problem: How would you add to the model to restrict the total weight of all products to be less than a new parameter, max_weight? Solve the linear program for a weight limit of 6500 tons, and explain how this extract restriction changes the results.

Solution: We need to make a few modifications here. We'll need to edit both the .dat and .mod files. In the data file we simply add a new `max_weight` parameter. Here it is next to `avail` for reference. Note that this parameter does not set specific weight limits for each product.

```
param avail := reheat 35 roll 40;
param max_weight := 6500;
```

In the model file we read in this parameter and set a constraint for it. Firstly, we want the max weight to be a non-negative value. This is just a data quality check.

After that, we create a new constraint using this parameter. This constraint ensures that the total weight across all products does not exceed `max_weight`.

```
param max_weight >= 0   # Total tons of weight allowed
    across all products

subject to Total_Weight:
  sum{p in PROD} Make[p] <= max_weight;
```

Below is the solution generated after these modifications.

$$\text{Total Profit} \approx 183791.67$$
$$\text{Bands} \approx 1541.67$$
$$\text{Coils} \approx 1458.33$$
$$\text{Plates} = 3500$$

What we see is a substantial shift from the original solution. Production of bands is nearly halved and production of coils is nearly tripled. The production of plates reaches its maximum. Total profit compared to the original solution drops by around \$6000.

This is caused by the new constraint on total weight. In the original problem, production was only limited by the products rates in each stage and their availability. Though there were hard minimum and maximums on production for each product, this was never directly involved in the objective function. As such, profit per ton played no real part in the optimization process and `rate` was the main parameter dictating which products were prioritized.

The new total weight constraint forces the model to consider the profit per ton of each product. Coils are the slowest to produce, but they have the highest profit per ton value (30) followed by plates (29). Bands, meanwhile, have the lowest profit per ton (25) and are produced far less as a result. This solution maxes out plates because they are just barely the second most profitable per ton and are the second fastest to produce. Then bands and coils fill in the remaining allocation. This is also why the total profit is lower now as production can't just focus production on the heaviest products.

C ✓Checked on September 11.

Problem: How would you change the objective function to maximize total tons? Does this make a difference to the solution?

Solution: This is the simplest to change. Just remove the profit from the objective function as it already factors in weight.

```
maximize Total_Weight: sum {p in PROD} Make[p];
```

Funnily enough this ends up with the exact same results as the original model!

$$\begin{aligned} \text{Total Weight} &= 7000 \\ \text{Bands} &\approx 3357.14 \\ \text{Coils} &= 500 \\ \text{Plates} &\approx 3142.86 \end{aligned}$$

We just don't get our profit shown in the objective function is all as it shows the total weight. The profit is the exact same as well, it just isn't shown here.

D

Problem: Suppose that instead of the lower bounds represented by `commit[p]` in our model, we want to require that each product represent a certain share of the total tons produced. In the algebraic notation of Figure 1-1, this new constraint might be represented as

$$X_j \geq s_j \sum_{k \in P} X_k, \text{ for each } j \in P$$

where s_j is the minimum share associated with project j. How would you change the AMPL model to use this constraint in place of the lower bounds `commit[p]`? If the minimum shares are 0.4 for bands and plate, and 0.1 for coils, what is the solution?

Verify that if you change the minimum shares to 0.5 for bands and plate, and 0.1 for coils, the linear program gives an optimal solution that produces nothing, at zero profit. Explain why this makes sense.

Solution: To start, we update the data file to remove the `commit` parameter and replace it with the new `share` parameter.

```
param:     profit  market  share :=
  bands      25     6000    0.4
  coils      30     4000    0.1
  plate      29     3500    0.4 ;
```

Next, in the model file, we remove the lower bound on `Make` set by `commit` and add a new constraint enforcing the minimum `share`.

```
param share {PROD} >= 0;      # minimum proportion of total
     tons for each product

var Make {p in PROD} >= 0, <= market[p]; # tons produced

subject to Share {p in PROD}:
    Make[p] >= share[p] * sum {k in PROD} Make[k];
```

With this our new solution is as follows:

$$\text{Total Profit} = 189700$$
$$\text{Bands} = 3500$$
$$\text{Coils} = 700$$
$$\text{Plates} = 2800$$

Firstly, this solution meets all minimum share requirements. The number of coils produced compared to the original solution increases by 200 as a result of this change.

Modifying the data file to shares of 0.5, 0.1, 0.5 results in no feasible solutions existing. As such, the optimizer produces nothing. This is because the shares are a proportion of the total production and the sum of these cannot exceed 1 which this group of shares does.

This shows some important behavior in AMPL. If the data and constraints provided result in no feasible solution then there will be no production. There has to be a feasible region for the solver to work with.

E

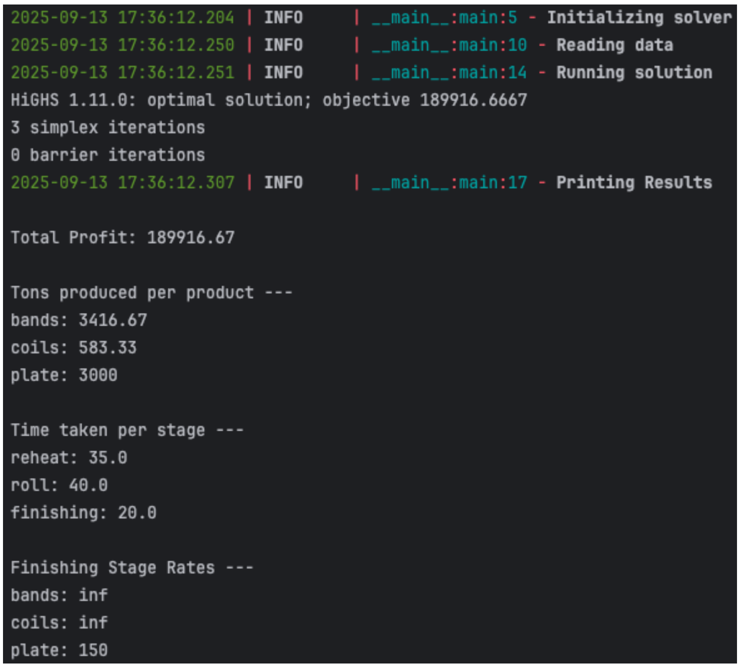
Problem: Suppose there is an additional finishing stage for plates only, with a capacity for 20 hours and a rate of 150 ton per hour. Explain how you could modify this data, without changing the model, to incorporate this new stage.
Solution: This is actually, thankfully, very easy to do! We can simply add a new stage to the data file and set arbitrarily large limits for bands and coils.

```
set STAGE := reheat roll finishing;

param rate:  reheat  roll  finishing:=
  bands         200   200  infinity
  coils         200   140  infinity
  plate         200   160  150;

param avail := reheat 35 roll 40 finishing 20;
```

This automatically gets worked in with 0 modifications to the model file! Below shows the run in Python.



```
2025-09-13 17:36:12.204 | INFO     | __main__:main:5 - Initializing solver
2025-09-13 17:36:12.250 | INFO     | __main__:main:10 - Reading data
2025-09-13 17:36:12.251 | INFO     | __main__:main:14 - Running solution
HiGHS 1.11.0: optimal solution; objective 189916.6667
3 simplex iterations
0 barrier iterations
2025-09-13 17:36:12.307 | INFO     | __main__:main:17 - Printing Results

Total Profit: 189916.67

Tons produced per product ---
bands: 3416.67
coils: 583.33
plate: 3000

Time taken per stage ---
reheat: 35.0
roll: 40.0
finishing: 20.0

Finishing Stage Rates ---
bands: inf
coils: inf
plate: 150
```

Figure 2: AMPL code running with finishing stage included.

Homework 4 Appendix

In this section I include python scripts that would be too distracting to include in the homework writeup itself.

The below script encompasses the bulk of the code used for HW4. Slight modifications may have been made here or there for specific purposes but this should provide most of the programming context needed.

```
from amplpy import AMPL
from loguru import logger

def main():
    logger.info("Initializing solver")
    ampl = AMPL()
    ampl.setOption("solver", "highs")

    file_name = "steel4"
    logger.info("Reading data")
    ampl.read(f"../models/{file_name}.mod")
    ampl.read_data(f"../data/{file_name}e.dat")

    logger.info("Running solution")
    ampl.solve()

    logger.info("Printing Results")

    total_profit = round(ampl.getObjective("Total_Profit").value(),2)
    print("\nTotal Profit:", total_profit)

    make_vals = {
        product: value
        for product, value in
        ↪ ampl.getVariable("Make").get_values().to_list()
    }

    print("\nTons produced per product ---")
    [print(f"{product}: {round(tons, 2)}") for product, tons in
    ↪ make_vals.items()]

    # Compute hours used per stage
    time_used = {}
    for stage in ampl.getSet("STAGE"):
        hours = sum(
```

```python
            make_vals[product] / ampl.getParameter("rate")[product,
            ↪  stage]
            for product in ampl.getSet("PROD")
        )
        time_used[stage] = hours

    print("\nTime taken per stage ---")
    [print(f"{stage}: {round(hours, 2)}") for stage, hours in
    ↪  time_used.items()]

    rate_param = ampl.getParameter("rate").get_values().to_list()
    print("\nFinishing Stage Rates ---")
    for x in rate_param:
        if x[1] == 'finishing':
            print(f"{x[0]}: {x[2]}")

if __name__ == "__main__":
    main()
```

Homework 5

Do exercise 1-3 from AMPL book

This exercise deals with some issues of sensitivity in the steel models.

A

Problem:

For the linear program of Figures 1-5a and 1-5b, display `Time` and `Make.rc`. What do these values tell you about the solution.

Relevant Files:

steel3.mod

```
set PROD;  # products

param rate {PROD} > 0;       # produced tons per hour
param avail >= 0;            # hours available in week
param profit {PROD};         # profit per ton

param commit {PROD} >= 0;   # lower limit on tons sold in week
param market {PROD} >= 0;   # upper limit on tons sold in week

var Make {p in PROD} >= commit[p], <= market[p]; # tons produced

maximize Total_Profit: sum {p in PROD} profit[p] * Make[p];

                # Objective: total profits from all products

subject to Time: sum {p in PROD} (1/rate[p]) * Make[p] <= avail;

                # Constraint: total of hours used by all
                # products may not exceed hours available
```

steel3.dat

```
data;

set PROD := bands coils plate;

param:     rate   profit   commit   market :=
  bands     200    25       1000     6000
```

```
  coils     140    30      500    4000
  plate     160    29      750    3500 ;

param avail := 40;
```

Solution:

NOTE: My basic python script for this part is provided in the HW5 appendix at the end of this section.

Running the code, `ampl.eval("Display Time, Make.rc")`, gives us the output below:

$$\begin{aligned} \text{time}_{dv} &= 4640 \\ \text{bands}_{rc} &= 1.80 \\ \text{coils}_{rc} &= -3.14 \\ \text{plates}_{rc} &\approx 0 \end{aligned}$$

The interpretation of these values is as follows.

First, time. If we were to add an extra unit of time (an hour) to availability we would see additional profit. To be precise we would see an increase of 4640 units of profit (dollar) per additional hour of availability.

For bands, we see the same kind of thing. Every unit (ton) increase in the upper bound of bands made would see an increase of 1.80 dollars of profit.

Coils has a negative coefficient. What this means is that every ton decrease in the lower bound of coils made would see an increase in profit by 3.14 dollars. This makes sense, we're maxing out the number of bands we can make and only producing the absolute minimum number of coils.

Plates have a coefficient of 0, meaning that increasing or decreasing the bounds on its production would have no impact on profit. This also makes sense intuitively as production of plates falls within the provided bounds already.

It is important to note that for these dual values and reduced costs, this relationship may not hold forever. These values are subject to change as constraints are modified.

B

Solution:

The figures mentioned in the problem statement refer to the steel4 data and model files. Those have been shown earlier in the homework so I will not be showing them again.

The table for this part show a massive increase in the number of plates produced and a massive decrease in the number of bands. Why is this?

	Steel3	**Steel4**
Bands	6000	≈ 3357.14
Coils	500	500
Plates	≈ 1028.57	≈ 3142.86
Total Profit	≈ 194828.57	≈ 190071.43

Table 1: Comparison of solution values and total profit for Steel3 and Steel4 models.

To explain what's going on here we need to understand the new rate constraint. `reheat` has the same rate across all 3 products. Previously bands had the highest rate of production which made up for it having the lowest profit coefficient of 2.5. `reheat`, as mentioned, levels the playing field a bit. Since, for that stage at least, all 3 products have the same rate, the profit coefficient matters a lot more.

What this means is that plates, with their very high profit coefficient of 2.9, outclass bands for this stage. Bands still come out on top for the rolling stage, which is why we still make so many bands, but this change is why we produce so much less of them. Coils still go completely ignored in this model, we still only produce the bare minimum required.

C

Solution

NOTE: For problems C and D the python script used is provided in the Homework 5 Appendix shown below this problem.

Using `amplpy` and saving the profit and reheat hours from each run gives us the following table:

reheat_hours	profit	time_dual_value
35	190071.43	1800.0
36	191871.43	1800.0
37	193671.43	1800.0
38	194828.57	0.0
39	194828.57	0.0
40	194828.57	0.0

This table verifies what the problem statement wanted us to check. We have a constant dual value for reheating up until we hit 38 reheat hours. From there, it has no influence on our profit whatsoever. This is likely due to the constraint on the rolling stage holding back any additional gains from a more generous reheating availability. These just become excess hours that go unused.

Next we check some other arbitrary values. We start with the provided $37\frac{9}{14}$. To test this we also solve for $36\frac{9}{14}$ so compare the dual values. We also test just beyond this value and do the exact same process for $37\frac{10}{14}$.

This gives us the following results:

reheat_hours	profit	time_dual_value
$36\frac{9}{14}$	193028.57	1800
$37\frac{9}{14}$	194828.57	0
$36\frac{10}{14}$	193157.14	1800
$37\frac{10}{14}$	194828.57	0

These tables verify the problem statement. Results below that $37\frac{9}{14}$ still show that 1800 profit increase. Right as soon as we surpass it even by a tiny amount, the dual value shows us that we would see no additional benefit.

One interesting observation is that $37\frac{10}{14}$ doesn't actually see an 1800 profit increase despite the dual value of $36\frac{10}{14}$. I wonder why that is. It must be that the slope at that point is still 1800, but the tiny part of $37\frac{10}{14}$ that exceeds our threshold means we don't quite capture 1800 in profit for a one unit increase. Essentially that the slope will flatten out before we see all 1800 dollars of profit realized. This means the dual value doesn't quite tell the whole picture.

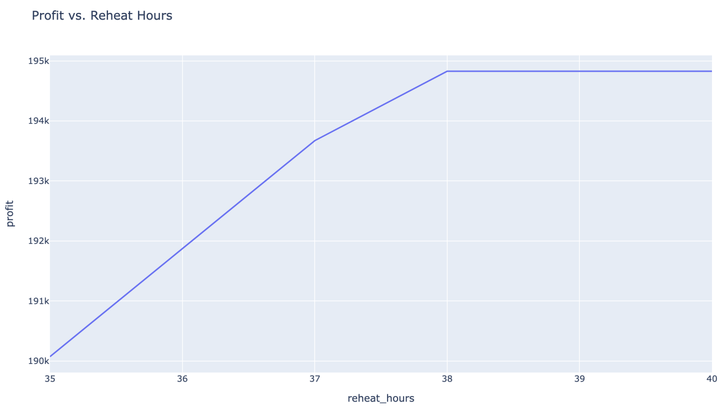


Figure 3: Plot created using the integer range of 35 through 40.

The plot here also provides further evidence of this thought. Though it is limited, only showing integer tests from 35-40, we still see that profit starts leveling off more going from 37 to 38. And then, of course, we see no additional gains to profit extending beyond 38 hours.

D

We extend the plot down to 25 reheating hours here.

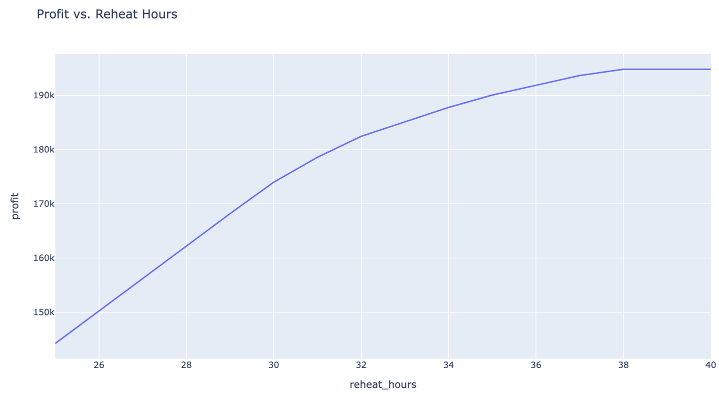


Figure 4: Plot created using the integer range of 25 through 40.

Interestingly enough we see the slope had changed well before we hit 35 hours. We see the first changes in the slope of this plot once we hit 30 available hours of reheat time. This isn't surprising. As an untested hunch, this is possibly due to increased access to unused `rolling` stage availability. With too little `reheat` availability, we can't properly utilize all of the availability of the other stage. So we see more steep growth early on.

We also extend our table to show all of the profit changes as we move from 10 to 25 available reheating hours.

reheat_hours	profit	time_dual_value
10	0.0	0.0
11	0.0	0.0
12	66250.0	6000.0
13	72250.0	6000.0
14	78250.0	6000.0
15	84250.0	6000.0
16	90250.0	6000.0
17	96250.0	6000.0
18	102250.0	6000.0
19	108250.0	6000.0
20	114250.0	6000.0
21	120250.0	6000.0
22	126250.0	6000.0
23	132250.0	6000.0
24	138250.0	6000.0
25	144250.0	6000.0

Here we have verified that from 12 available reheat hours to 25 hours we see a constant slope of 6000 dollars of profit per hour increase in availability. We also see 0s for 10 and 11 hours of availability. Why is that?

It's because of our minimum production constraints. In the steel4 data file, we have a commit parameter that controls our production lower bounds. We need a minimum of 1000 tons of bands, 500 tons of coils and 750 tons of plates.

Our time constraint is as follows:

```
subject to Time {s in STAGE}:
    sum {p in PROD} (1/rate[p,s]) * Make[p] <= avail[s];
```

Running through the math for all products in the reheating stage:

$$\frac{1}{200} \cdot (1000 + 500 + 750) = 11.25 > 11$$

Our minimum production requirements require more than 11 hours of reheating time, so anything below 11.25 results in no feasible solution being possible.

Homework 5 Appendix

In this section I include python scripts that would be too distracting to include in the homework writeup itself.

Below is the script used for 1-3 part A.

```
from amplpy import AMPL
from loguru import logger

def main():
    logger.info("Initializing solver")
    ampl = AMPL()
    ampl.setOption("solver", "highs")

    file_name = "steel3"
    logger.info("Reading data")
    ampl.read(f"../models/{file_name}.mod")
    ampl.read_data(f"../data/{file_name}.dat")

    logger.info("Running solution")
    ampl.solve()
    ampl.eval("display Time, Make.val, Make.rc;")

if __name__ == "__main__":
    main()
```

Below is the script used for 1-3 parts C and D.

```
from amplpy import AMPL
from loguru import logger
import polars as pl
import plotly.express as px

def main(reheat_hours_list):
    file_name = "steel4"
    profit_list = []
    duals = []

    # Goal here is to populate a dataframe we can examine through all of the necessary runs
    # We initiate AMLP() in this loop so thats its fully reset between runs.
    # Not doing this can result in weird results for edge case scenarios.
    for reheat_hours in reheat_hours_list:
```

```
        logger.info("Initializing solver")
        ampl = AMPL()
        ampl.setOption("solver", "highs")
        logger.info("Reading data")
        ampl.read(f"../models/{file_name}.mod")
        ampl.read_data(f"../data/{file_name}.dat")
        logger.info(f"Solving with reheat availability = {reheat_hours}")
        ampl.getParameter("avail")["reheat"] = reheat_hours
        ampl.solve()

        profit = ampl.getObjective("Total_Profit").value()
        profit_list.append(round(profit, 2))

        # Dual value (shadow price) for the reheat stage
        dual_value = ampl.getConstraint("Time")["reheat"].dual()
        duals.append(round(dual_value, 2))

    df = pl.DataFrame({
        "reheat_hours": reheat_hours_list,
        "profit": profit_list,
        "time_dual_value": duals
    }, strict=False)


    return df


if __name__ == "__main__":
    reheat_hours_list = list(range(10, 41))
    # test_value_1 = 37 + (9/14)
    # test_value_2 = 37 + (10/14)
    # reheat_hours_list = [test_value_1 - 1, test_value_1, test_value_2 -
    ↪   1, test_value_2]
    # reheat_hours_list = [test_value_1, test_value_2]

    df = main(reheat_hours_list)
    with pl.Config(tbl_rows=40):
        print(df)
    df.write_csv("../data/reheat-profit-table.csv")

    # fig = px.line(df, x="reheat_hours", y="profit, title='Profit vs.
    ↪   Reheat Hours')
    # fig.show()
```

Homework 6

Do exercise 2-6 from AMPL book.

The output of a paper mill consists of standard rolls 110 inches wide, which are cut into small rolls to meet orders. This week there are orders for rolls of the following widths:

Width	Orders
20"	48
45"	35
50"	24
55"	10
75"	8

The owner of the mill wants to know what cutting patterns to apply so as to fill the orders using the smallest number of 110" rolls.

Relevant Files:

The AMPL book provides sample model and data files for the cutting problem and I used them as a base. Here are those files with the knapsack part of the model removed.

cut.mod

```
param roll_width > 0;          # width of raw rolls

set WIDTHS;                    # set of widths to be cut
param orders {WIDTHS} > 0;     # number of each width to be cut

param nPAT integer >= 0;       # number of patterns
set PATTERNS = 1..nPAT;       # set of patterns

param nbr {WIDTHS,PATTERNS} integer >= 0;

   check {j in PATTERNS}:
      sum {i in WIDTHS} i * nbr[i,j] <= roll_width;

                             # defn of patterns: nbr[i,j] = number
                             # of rolls of width i in pattern j

var Cut {PATTERNS} integer >= 0;    # rolls cut using each pattern
```

```

minimize Number:                    # minimize total raw
    rolls cut
   sum {j in PATTERNS} Cut[j];

subject to Fill {i in WIDTHS}:
   sum {j in PATTERNS} nbr[i,j] * Cut[j] >= orders[i];
```

cut.dat

```
data;

param roll_width := 110 ;

param: WIDTHS: orders :=
          20      48
          45      35
          50      24
          55      10
          75       8   ;
```

It is worth noting that the cut.mod file is already set up for integer solutions. This file will need to be modified to acquire the non-integer solutions the textbook problems except for parts A-C.

A

Problem:

A cutting pattern consists of a certain number of rolls of each width, such as two of 45" and one of 20", or one of 50" and one of 55". Suppose, to start with, that we consider only the following six patterns.

Width	1	2	3	4	5	6
20"	3	1	0	2	1	3
45"	0	2	0	0	0	1
50"	1	0	1	0	0	0
55"	0	0	1	1	0	0
75"	0	0	0	0	1	0

Table 2: Number of rolls of given width created for each pattern.

How many rolls should be cut according to each pattern, to minimum the number of 110" rolls used? Formulate and solve this problem as a linear program, assuming that the number of smaller rolls produced need only be greater than or equal to the number ordered.

Solution:

For this part we first need to add these patterns to the data file. This requires an overall rework of the provided data file. I would normally only show the modified section for brevity but really I reworked the whole thing.

```
data;

param roll_width := 110 ;
set WIDTHS := 20 45 50 55 75;

param: orders :=
    20      48
    45      35
    50      24
    55      10
    75       8   ;

param nPAT := 6;

param nbr:
       1   2   3   4   5   6  :=
20      3    1    0    2    1    3
45      0    2    0    0    0    1
50      1    0    1    0    0    0
55      0    0    1    1    0    0
75      0    0    0    0    1    0  ;
```

The changes made are meant to accomodate the set up of the model file.

As we can see there is now a set of widths which we use to provide context to the orders and patterns. We now have a parameter `nPAT` which provides the number of patterns to the model. We also have all of the patterns and how they relate to the weights.

From there the provided model needs no modifications. All I do is remove the integer specification on `var CUT` so that we can get non-integer solutions. **NOTE:** As with previous problems all related python scripts are provided in the appendix for this problem.

Below are the results from running the AMPL code `ampl.eval(''display Number, Cut;'')`.

Total Rolls: 49.5

What we can note here are the fractional amounts for the cuts. Realistically for this kind of problem we can't do 7.5 pattern 1 cuts. So if we were to practically try to use this solution we'd need to round all of these up and end up using more rolls. This is why part d later calls for an integer solution to this

Cut	Times Used
1	7.5
2	17.5
3	16.5
4	0
5	8
6	0

Table 3: Results of `display Cut`

problem. It's because 49.5 rolls is ambiguous due to the fractional number of cuts being ambiguous. This solution is a start, but it's less helpful than we may want it to be.

B

Problem:

Re-solve the problem, with the restriction that the number of rolls produced in each size must be between 10% under and 40% over the number ordered.

Solution:

This problem is fairly straightforward. A simple modification to the model file is all that is necessary. We specifically alter the `Fill` constraint to provide a range of appropriate values. This is done by simply scaling the `orders[i]` value on either side of what is now a double inequality.

```
subject to Fill {i in WIDTHS};
0.9 * orders[i] <= sum {j in PATTERNS} nbr[i, j] * Cut[j] <=
    1.4 * orders[i]
```

This drastically decreases the overall number of cuts and total rolls we need.

Total Rolls: 44.55

Cut	Times Used
1	7.6
2	15.75
3	14
4	0
5	7.2
6	0

Table 4: Results of `display Cut`

We run into the same problem here. What is 7.2 cuts of pattern 5 even mean? Also of note here is that this change didn't result in an increase in any cuts outside of pattern 1. One thing that would have been interesting is if this relaxed upper bound resulted in some pattern now being overproduced to result the number of rolls but that doesn't seem to be happening here.

C

Problem:

Find another pattern that, when added to those above, improves the optimal solution.

Solution:

This one was interesting. Going back to the model and data in part A, we need to figure out where the inefficiences in our patterns are. We can do this by checking the `slack` values on our `Fill` constraint. The slack values here will tell us which widths of roll are being over or under-produced.

Width	Slack
20"	0
45"	0
50"	0
55"	6.5
75"	0

Table 5: Results of `display Fill.slack`

What we can see from this table is that we produce way more 55" rolls than we need to. We only need 10 55" rolls according the orders and we're overproducing that width by over half. Why is that? If we look at part A again and check the most used patterns, we make the most of patterns 2 and 3. Pattern 3 is what is important here. It creates 1 50" roll and 1 55" roll. Why is that important? It is one of the only patterns that produces 50" rolls. I'm not entirely sure why this pattern is used over pattern 1 for creating 50" rolls, but it is. What this hints at is that we could use a more efficient pattern for creating 50" rolls. To that end, I create a new pattern that only produces 2 50" rolls.

```
param nPAT := 7;

param nbr:
          1   2   3   4   5   6   7:=
20        3   1   0   2   1   3   0
45        0   2   0   0   0   1   0
50        1   0   1   0   0   0   2
55        0   0   1   1   0   0   0
75        0   0   0   0   1   0   0;
```

Re-running the original model with this new pattern gives us the following results:

Total Rolls Used: 46.25

Cut	Times Used
1	7.5
2	17.5
3	10
4	0
5	8
6	0
7	3.25
Width	**Slack**
20"	0
45"	0
50"	0
55"	0
75"	0

Table 6: Results of `display Cut, Fill.slack`

The number of rolls we need has gone down by 3, to 46.25 and now as we can see we are no longer overproducing on any of the widths.

D

Problem:

All of the above solutions use fractional number of rolls. Can you find solutions that also satisfy the constraints, but that cut a whole number of rolls for each pattern? How much does your whole-number solution cause the objective function value to go up in each case?

Solution:

For this we revert back to the original version of the model file.

```
var Cut {Patterns} integer >= 0;
```

we then just rerun the model and data files from the previous parts one last time.

Part	Rolls Float	Rolls Integer
A	49.5	50
B	44.55	46
C	46.25	47

Table 7: Objective value comparison for each homework part.

Across the board we see a slight increase in rolls produced. This makes sense. A integer solution is inherently more restrictive so we have to make some concessions to accomodate that requirement.

Homework 6 Appendix

```python
from amplpy import AMPL
from loguru import logger

def main():
    logger.info("Initializing solver")
    ampl = AMPL()
    ampl.setOption("solver", "highs")

    file_name = "cut_hw"
    logger.info("Reading data")
    ampl.read(f"../models/{file_name}.mod")
    ampl.read_data(f"../data/{file_name}.dat")

    logger.info("Running solution")
    ampl.solve()

    ampl.eval("display Number, Cut;")
    ampl.eval("display Fill.slack;")

if __name__ == "__main__":
    main()
```