# Homework Collection 3

## Table of Contents

# 1 Homework 15 (PRECHECKED)

## Problem

Provide an example where both the primal and its corresponding dual problem are both infeasible.

## Solution

**Primal**

$$\max -5x_1 + 7x_2$$

such that

$$\begin{aligned} -x_1 + x_2 &\leq 1 \\ x_1 - x_2 &\leq -9 \\ x_1, x_2 &\geq 0 \end{aligned}$$

This is a problem as $x_1 - x_2 \leq 9 \Rightarrow -x_1 + x_2 \geq 9$ which contradicts the first constraint. So the primal problem here is infeasible. Below I simply have the matrix/vector versions of our problem.

$$A = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}, b = \begin{pmatrix} 3 \\ -9 \end{pmatrix}, c = \begin{pmatrix} -5 \\ 7 \end{pmatrix}$$

with

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

**Dual**

First we convert it over.

$$\begin{aligned} &\min b^T y \\ &y \geq 0 \\ &A^T y \geq c \end{aligned}$$

Looking at the constraintss directly gives us:

$$\begin{aligned} \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}^T \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} &\geq \begin{pmatrix} -5 \\ 7 \end{pmatrix} \\ -y_1 + y_2 &\geq -5 \\ y_1 - y_2 \geq 7 \Rightarrow -y_1 + y_2 &\leq -7 \end{aligned}$$

This is a contradiction just like in the primal. So the dual problem is also infeasible.

# 2 Homework 16 (PRECHECKED)

## Problem

Create a linear program to solve a sudoku problem.

## Context

A sudoku puzzle is traditionally a 9x9 grid with some numbers provided in seemingly random cells. Solving a sudoku problems involves filling out all of the remaining cells with the given conditions:

1. Every column in the grid must contain every integer from 1 to 9.
2. Every row in the grid must contain every integer from 1 to 9.
3. The grid is split into 9 smaller 3x3 sections, and each of those subgrids must contain every integer from 1 to 9.
4. No duplicate values may exist in any column, row or subgrid.

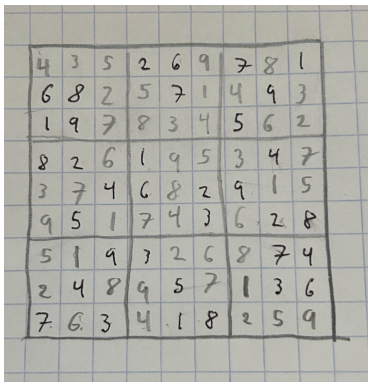
## Solution

I leveraged two different solutions to this problem. The first involved using a one-hot encoding strategy for handling the sudoku grid. The second leveraged the `alldiff` function that some solvers have access to. I will be covering both here.

Below is the sudoku problem that I used as a base for this homework.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 2 | 6 | | 7 | | 1 |
| 6 | 8 | | | 7 | | | 9 | |
| 1 | 9 | | | | 4 | 5 | | |
| 8 | 2 | | 1 | | | | 4 | |
| | | 4 | 6 | | 2 | 9 | | |
| | 5 | | | | 3 | | 2 | 8 |
| | | 9 | 3 | | | | 7 | 4 |
| | 4 | | | 5 | | | 3 | 6 |
| 7 | | 3 | | 1 | 8 | | | |

And here is my solution for the problem done by hand.



### The Data File

Here is how I set up the above sudoku problem. I will not include the full file here, the snippet here should suffice.

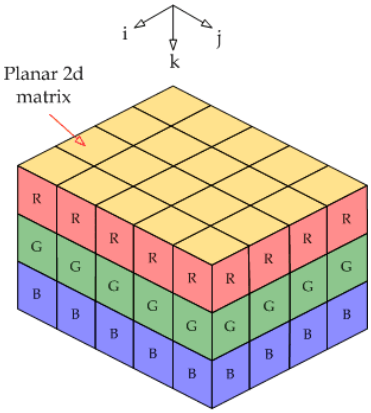
```
data;

param gridsize := 9;

param grid_data :=
1 4 2
1 5 6
1 7 7
1 9 1
2 1 6
2 2 8
2 5 7
...
```

I organize the `grid_data` as "row, column, value". So in this case row 1 column 4 has the value 2. I manually filled in all given values this way.

### Solution A: One-hot Encoding

I had originally wanted to think of this as a knapsack problem but got overwhelmed trying to track everything. This solution, instead of limiting itself to a 2d grid, expands out to a cube, with a matrix for each possible value.

We can think of this much like an image with different channels for RGB. Each pixel in an image can have values for each color. Below is a visualization of this.




Graphical presentation of RGB 3d matrix

So in our case we have 9 9x9 grids. One for each possible value. It's the same sudoku grid stacked on top of each other, and each layer simply asks "does this cell have value $x$ in it"? This seems bizarre at first but when we add constraints into the mix we get a very convenient structure for finding a solution. For starters, we can say "this cell can only contain red, green or blue". So if we'd put 2 in a cell, the 2 layer gets a 1, or a "True", in the corresponding cell. Every other layer would get a 0 in that cell.

This expands out in a very convenient way. This allows us to easily use sums for ALL checks which is a mess when you're working with the values themselves. Let's walk through the model now.

**MODEL CODE**

First we set up our grid in 2D. We want to read in the data and track everything that's been given. We want to make sure none of those can be modified later.

```
param gridsize > 0;
param grid_data {1..gridsize, 1..gridsize} integer >= 0, < 10, default 0;

# differentiates between given cells and cells we can modify
set FIXED := {i in 1..gridsize, j in 1..gridsize: grid_data[i,j] != 0};
set MODIFIABLE := {i in 1..gridsize, j in 1..gridsize: grid_data[i,j] = 0};
```

Now here is where we set up our cube. The cube itself is our variable for the linear program.

```
# possible values
set VALS := 1..9;

# Like a binary RGB grid for images. Asks "Does this cell have red?"
var X {i in 1..gridsize, j in 1..gridsize, v in VALS} binary;
```

Next we get to the constraints. How do we want the cube to behave?

```
# Restrict the models ability to change any given values
subject to prefilled { (i,j) in FIXED }:
    X[i,j, grid_data[i,j]] = 1;

# forces every cell to HAVE a value
# One of those value windows has to have something
subject to one_value_per_cell { (i,j) in MODIFIABLE }:
    sum {v in VALS} X[i,j,v] = 1;
```

The above constraint is really important, without it we can satisfy the other conditions and end up with a bunch of 0 cells. So every cell must have one and only one active layer. Next up, the classic constratints.

```
subject to row_unique {i in 1..gridsize, v in VALS}:
    sum {j in 1..gridsize} X[i,j,v] = 1;

subject to col_unique {j in 1..gridsize, v in VALS}:
    sum {i in 1..gridsize} X[i,j,v] = 1;

subject to block_unique {bi in 0..2, bj in 0..2, v in VALS}:
    sum {i in 3*bi+1..3*bi+3, j in 3*bj+1..3*bj+3} X[i,j,v] = 1;
```

These enforce the typical sudoku rules.

Now the objective function in this case is totally arbitrary. I just choose to sum everything up.

```
# Totally arbitrary objective function, just sum up everything
maximize dummy: sum {i in 1..gridsize, j in 1..gridsize, v in VALS} X[i,j,v];
```

**OUTPUT**

The output is huge here so I'll only show the relevant snippets and overall solution.

Here is our given problem

```
grid_data [*,*]
:   1   2   3   4   5   6   7   8   9    :=
1   0   0   0   2   6   0   7   0   1
2   6   8   0   0   7   0   0   9   0
3   1   9   0   0   0   4   5   0   0
4   8   2   0   1   0   0   0   4   0
5   0   0   4   6   0   2   9   0   0
6   0   5   0   0   0   3   0   2   8
7   0   0   9   3   0   0   0   7   4
8   0   4   0   0   5   0   0   3   6
9   7   0   3   0   1   8   0   0   0
```

Here is the solution (cleaned up with python)

```
[4, 3, 5, 2, 6, 9, 7, 8, 1]
[6, 8, 2, 5, 7, 1, 4, 9, 3]
[1, 9, 7, 8, 3, 4, 5, 6, 2]
[8, 2, 6, 1, 9, 5, 3, 4, 7]
[3, 7, 4, 6, 8, 2, 9, 1, 5]
[9, 5, 1, 7, 4, 3, 6, 2, 8]
[5, 1, 9, 3, 2, 6, 8, 7, 4]
[2, 4, 8, 9, 5, 7, 1, 3, 6]
[7, 6, 3, 4, 1, 8, 2, 5, 9]
```

And to show an example of the makeup of this solution under the hood, here is the layer for the value 1. This will show all the locations there the cells is 1.

```
X[i,j,1] [*,*]
:   1   2   3   4   5   6   7   8   9    :=
1   0   0   0   0   0   0   0   0   1
2   0   0   0   0   0   1   0   0   0
3   1   0   0   0   0   0   0   0   0
4   0   0   0   1   0   0   0   0   0
5   0   0   0   0   0   0   0   1   0
6   0   0   1   0   0   0   0   0   0
7   0   1   0   0   0   0   0   0   0
8   0   0   0   0   0   0   1   0   0
9   0   0   0   0   1   0   0   0   0
```

### Solution B - alldiff

I'll keep this section brief as it is far simpler. Here we use the power of alternative solvers like `scip-cpx` that allows for a powerful constraint tool, `alldiff`. This forces a set of integer variables to have distinct values. Using this with a set of possible values, say `1..9` makes building a model extremely easy.

**MODEL CODE**

For this solution we read in the data and track our fixed and modifiable indices in the exact same way. The differences are shown here.

```
var X {i in 1..gridsize, j in 1..gridsize} integer >= 1, <= 9;

# Restrict the models ability to change any given values
subject to prefilled { (i,j) in FIXED }:
    X[i,j] = grid_data[i,j];

subject to row_unique {i in 1..gridsize}:
    alldiff {j in 1..gridsize} X[i,j];

subject to col_unique {j in 1..gridsize}:
    alldiff {i in 1..gridsize} X[i,j];

subject to block_unique {bi in 0..2, bj in 0..2}:
    alldiff {i in 3*bi+1..3*bi+3, j in 3*bj+1..3*bj+3} X[i,j];
```

We use the exact same constraints as before but instead of summing across all the layers we just slap on the `alldiff` constraint. This allows us to stay with just a 2d matrix the entire time.

**OUTPUT**

```
X [*,*]
:   1   2   3   4   5   6   7   8   9    :=
1   4   3   5   2   6   9   7   8   1
2   6   8   2   5   7   1   4   9   3
3   1   9   7   8   3   4   5   6   2
4   8   2   6   1   9   5   3   4   7
5   3   7   4   6   8   2   9   1   5
6   9   5   1   7   4   3   6   2   8
7   5   1   9   3   2   6   8   7   4
8   2   4   8   9   5   7   1   3   6
9   7   6   3   4   1   8   2   5   9
```

As we cam see it reaches the same solution as both myself and the previous model.

# 3 Homework 17 (PRECHECKED)

## Problem

Prove the following theorem:

> **Theorem 3.1**
>
> Let $P = \{x : Ax \leq b\}$. Let $P$ have a vertex. Let $\bar{x} \in P$. Then $\bar{x}$ is a vertex if and only if an inclusion-maximal set of constraints is active at $\bar{x}$.
>
> In other words, $\bar{x}$ is a vertex if and only if there does not exist a $y \in P$ such that $A(\bar{x}) \subsetneq A(y)$.

## Solution

Before we start the proof, let's think about what this means. $A(\bar{x})$ is the index set of active constraints at $\bar{x}$. This theorem is saying that if theres any other feasible point that has an index set that is the superset of $\bar{x}$, that $\bar{x}$ can't be a vertex. This intuitively makes sense as it would imply there was some constraint that was inactive at $\bar{x}$ which would mean its not a vertex.

**Proof:**

$(A \Rightarrow B)$

Let $\bar{x}$ be a vertex. Because $\bar{x}$ is a vertex, we know that by definition: $\operatorname{rank} A_{\bar{x}} = n$. From this, we also know that $A_{\bar{x}}x = b_{\bar{x}}$ has a unique solution.

Suppose then that that there exists a $y \in P$ such that $A(\bar{x}) \subsetneq A(y)$. As this is a strict subset, $A(y)$ contains all of the same active constraint indices as $A(\bar{x})$. Therefore, $y$ satisfies every single equality in $A_{\bar{x}}x = b_{\bar{x}}$.

However, as previously stated, that system has only the unique solution $\bar{x}$. Therefore, $\bar{x} = y$ which contradicts the strict inclusion that was set up.

From this, by contradiction we have shown that there cannot exist a $y \in P$ such that $A(\bar{x}) \subsetneq A(y)$.

$(B \Rightarrow A)$

Assume that there does not exist a $y \in P$ such that $A(\bar{x}) \subsetneq A(y)$.

Suppose that $\bar{x}$ is not a vertex. Because of this, we know that $\operatorname{rank} A_{\bar{x}} < n$. Therefore, there must be some $d \neq 0$ where $A_{\bar{x}}d = 0$.

This is important, as it gives us a direction we are free to move along without deactivating any currently active constraints. From this, choosing either $+d$ or $-d$ and moving in that direction will eventually cause a new constraint to become active without deactivating any constraints. So this adds a constraint onto the previous set of active constraints. Let us call the point where this happens $y$.

Now, we have a new active constraint alongside all of the ones previously active for $\bar{x}$. As such, the set of active constraints at $y$ fully contains the set of active constraints at $\bar{x}$. In other words, $A(\bar{x}) \subsetneq A(y)$. This contradicts the assumption made at the beginning.

Therefore, by contradiction we have shown that $\bar{x}$ must be a vertex.

This completes the proof.

# 4 Homework 18

## Problem

Dualize the first LP (page 3) in the AMPL book and give an economic interpretation of the dual.

## Solution

The LP in the book is provided as such:

$$\begin{aligned} &\max 25x_B + 30x_c \\ \text{s.t. } &\frac{1}{200}x_b + \frac{1}{140}x_c \leq 40 \\ &0 \leq x_B \leq 6000 \\ &0 \leq x_C \leq 4000 \end{aligned}$$

The objective function values represent the "Profit per ton". The first constraint represents the hours constraint, where the values show the rate of production for bands and coils (hours per ton). The last two constraints are production weight limits on bands and coils respectively in tons.

Some basic cleanup gets us the following set of constraints:

$$\begin{aligned} \text{s.t. } &\frac{1}{200}x_b + \frac{1}{140}x_c \leq 40 \\ &x_B \leq 6000 \\ &x_C \leq 4000 \\ &x_B, x_C \geq 0 \end{aligned}$$

And we can rewrite all of this as:

$$A = \begin{pmatrix} \frac{1}{200} & \frac{1}{140} \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, b = \begin{pmatrix} 40 \\ 6000 \\ 4000 \end{pmatrix}, c = \begin{pmatrix} 25 \\ 30 \end{pmatrix}, x = \begin{pmatrix} x_B \\ x_C \end{pmatrix}$$

So the primal can now be written as:

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

### Dual Problem

Converting to the dual gives us the following form:

$$\begin{aligned} &\min b^T y \\ &A^T y \geq c \\ &y \geq 0 \end{aligned}$$

With $A^T = \begin{pmatrix} \frac{1}{200} & 1 & 0 \\ \frac{1}{140} & 0 & 1 \end{pmatrix}$.

Substituting in all of the values from the primal gives us:

$$\begin{aligned} \min\ & 40y_H + 6000y_B + 4000y_C \\ \text{s.t. } & \frac{1}{200}y_H + y_B + 0y_C \geq 25 \\ & \frac{1}{140}y_H + 0y_B + y_C \geq 30 \\ & y_H, y_B, y_C \geq 0 \end{aligned}$$

**Interpretation**

Let's say we have a guy named Ted and he's the one buying all the resources the factory needs. He wants to spend as little as possible on resources while still guaranteeing enough production so that the profits in the primal are realized.

Ted also wants to make sure the factory is getting the most out of its production. As an example, if the value of $y_C$ is extremely large, that would indicate to Ted that the production constraints on coils is causing some profit to be left on the table. Ted would be able to show this value to stakeholders to show them how much more profit would be realized if an additional ton of coils was allowed to be produced.

On the other hand; if $y_H = 0$, that would tell Ted that the number of hours of production isn't a limiting factor on profit and increasing it any further wouldn't help.

**REWRITING NOTES**

Discuss that the person buying resources and setting resource constraints is benevolent and has the factories best interest at mind. They are at an adversarial role to the maximization guy though, why is that? Also go into more detail on what the exact coefficients and objective function mean. More detail, not enough.

# 5 Homework 19

## Problem

Show that the below example cycles at 0 through a dictionary computation.

Note: Keep formatting clean and feel free to skip over tedious intermediate computations to save on space.

$$\begin{aligned}
\max \frac{3}{4}x_1 - 20x_2 + \frac{1}{2}x_3 - 6x_4 \\
\text{subject to } \frac{1}{4}x_1 - 8x_2 - x_3 + 9x_4 \leq 0 \\
\frac{1}{2}x_1 - 12x_2 - \frac{1}{2}x_3 + 3x_4 \leq 0 \\
x_3 \leq 1 \\
x_1, x_2, x_3, x_4 \geq 0
\end{aligned}$$

Note that the origin here is redundant with 6 active constraints. In a run of the simplex method with LCR & MRT, the algorithm cycles at 0. The same basis is repeated after 6 pivots.

## Solution

First we rewrite.

$$\begin{aligned}
J = \frac{3}{4}x_1 - 20x_2 + \frac{1}{2}x_3 - 6x_4 \\
\text{(subject to) } w_1 = 0 - \frac{1}{4}x_1 + 8x_2 + x_3 - 9x_4 \\
w_2 = 0 - \frac{1}{2}x_1 + 12x_2 + \frac{1}{2}x_3 - 3x_4 \\
w_3 = 1 - x_3
\end{aligned}$$

Initial solution, set all $x$ variables to 0. This gives us:

$$w_1 = 0, w_2 = 0, w_3 = 1, J = 0$$

### Pivot 1

Pick $x_1$ as it has the largest positive coefficient in the $J$ row at $\frac{3}{4}$. The two minimum ratios we get for this are 0 and 0. We'll pick $w_1$. Working through the pivot gives us the dictionary:

$$\begin{aligned}
J = -3w_1 + 4x_2 + \frac{7}{2}x_3 - 33x_4 \\
x_1 = -4w_1 + 32x_2 + 4x_3 - 36x_4 \\
w_2 = 2w_1 - 4x_2 - \frac{3}{2}x_3 + 15x_4 \\
w_3 = 1 - x_3
\end{aligned}$$

New basic feasible solution: $x_1 = 0, w_2 = 0, w_3 = 1, J = 0$.

### Pivot 2

Now we pick $x_2$ to enter as it has the largest coefficient at 4. Our only negative coefficient for $x_2$ is in the $w_2$ row. $w_2 = 0$ from our basic feasible solution so our ratio is $\frac{0}{4} = 0$.

New dictionary:

$$\begin{aligned} J &= -w_1 - w_2 + 2x_3 - 18x_4 \\ x_1 &= -8w_2 + 12w_1 - 8x_3 + 84x_4 \\ x_2 &= -\frac{1}{4}w_2 + \frac{1}{2}w_1 - \frac{3}{8}x_3 + \frac{15}{4}x_4 \\ w_3 &= 1 - x_3 \end{aligned}$$

New basic feasible solution: $x_1 = 0, x_2 = 0, w_3 = 1, J = 0$.

### Pivot 3

$x_3$ enters as it now has the largest coefficient at 2. The ratios are as follows.

$$x_1 : 0, x_2 : 0, w_3 : 1$$

We pick $x_1$ as its the top row. Solving that row for $x_3$ and pivoting gives us the new dictionary:

$$\begin{aligned} J &= -\frac{1}{4}x_1 + 2w_1 - 3w_2 + 3x_4 \\ x_3 &= -\frac{1}{8}x_1 - w_2 + \frac{3}{2}w_1 + 10.5x_4 \\ x_2 &= \frac{3}{64}x_1 - \frac{1}{16}w_1 + \frac{1}{8}w_2 - \frac{3}{16}x_4 \\ w_3 &= 1 + \frac{1}{8}x_1 + w_2 - \frac{3}{2}w_1 - 10.5x_4 \end{aligned}$$

New basic solution: $x_3 = 0, x_2 = 0, w_3 = 1, J = 0$

### Pivot 4

Same song and dance.

$$\begin{aligned} J &= -\frac{1}{4}x_1 + 2w_1 - 3w_2 + 0x_3 \\ x_4 &= \frac{1}{10.5}x_1 + 0x_2 - \frac{1}{10.5}w_1 + \frac{1}{10.5}w_2 + 0x_3 \\ x_2 &= \frac{3}{64}x_1 - \frac{1}{16}w_1 + \frac{1}{8}w_2 - \frac{3}{16}x_4 \\ w_3 &= 1 + \frac{1}{8}x_1 - \frac{3}{2}w_1 + w_2 - 10.5x_4 \end{aligned}$$

New basic solution: $x_4 = 0, x_2 = 0, w_3 = 1, J = 0$

### Pivot 5

$$
\begin{aligned}
J &= -\frac{1}{4}x_1 + 2w_1 + 0x_3 + 0x_4 \\
w_2 &= 0 + 0x_1 + 0x_2 + 0x_3 + 0x_4 \\
x_2 &= 0 \\
w_3 &= 1 - \frac{3}{2}w_1 - 10.5x_4 + \frac{1}{8}x_1
\end{aligned}
$$

New basic solution: $w_2 = 0, x_2 = 0, w_3 = 1, J = 0$

### Pivot 6

$$
\begin{aligned}
J &= -\frac{1}{4}x_1 + 2w_1 + 0x_4 \\
x_3 &= 0 \\
x_2 &= 0 \\
w_3 &= 1 + \frac{1}{8}x_1 - \frac{3}{2}w_1 - 10.5x_4
\end{aligned}
$$

New basic solution: $x_3 = 0, x_2 = 0, w_3 = 1, J = 0$

What we can see here is this is the same basic solution as Pivot 3. Thus showing the full cycle.

# 6 Homework 20

## Problem

Design a Phase I LP for a problem:

$$\begin{gathered}\max c^T x \\ Ax = b \\ x \geq 0\end{gathered}$$

Explain briefly (less than 100 words).

## Solution

$$\begin{gathered}\min \sum y_i \\ Ax + y = b \\ y \geq 0\end{gathered}$$

For this LP we use artifical variables $y$ to force initial feasibility using $x = 0, y = b$. The objective function allows us to then reduce the values of $y$ until they get as small as possible. This has two possible results. One, that $\min \sum y_i = 0$. This means we have found a feasible solution where $Ax = b$. We no longer depend on $y$ for feasibility and can move on. The other result is $\min \sum y_i > 0$. This means even at the optimal phase 1 value we still rely on $y$, meaning the original LP was infeasible.

# 7 Homework 21

## Project goal

This project seeks to explore the statistical technique known as $L_1$, or, LASSO regression from a linear programming perspective. This will be done by utilizing a simple toy dataset and performing LASSO regression using both the `statsmodels` library in python and by writing an AMPL model using the `amplpy` api. The goal will be to show the equivalence of the two perspectives. By doing that we can verify the correctness of our implementation and confidently explore the differences required for the two approaches.

We're pursuing this as the topic for our project as it feels like the perfect overlap between statistics and linear programming. Being able to look at tools we have used for years from a fresh perspective is exciting and we hope this project may help shed some light on regression for others in the class.

## Regression and LASSO background info

Regression at its core takes a set of points in $n$ dimensional space and attempts to fit a line, plane or hyperplane through that space in a way that minimizes its overall distance from those points. That is usually referred to as minimizing the residual sum of squares, or RSS. LASSO, or "Least Absolute Shrinkage and Selection Operation", is a special type of regression that includes a penalty for model complexity. It does this by shrinking some variables down to 0 while still minimizing the RSS. So this method also functions as a variable selection tool! Below is a comparison of the two using typical statistical notation.

$$\text{Linear Regression: } \min \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\text{LASSO Regression: } \underbrace{\min \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}_{\text{RSS}} + \underbrace{\lambda \sum_{j=1}^{p} |b_j|}_{\text{Penalty}}$$

$$\hat{y} = \sum_{j=1}^{p} b_j x_{ij}$$

Here $y$ are the observed values, $\hat{y}$ are the estimated values, $b$ are the regression coefficients and $\lambda$ represents the penalty scalar. We can see the two parts clearly here. The first compares the observed values to the models estimated values and the second part has a weighted total of the model coefficients.

The penalty scalar dictates to what degree large coefficients are penalized and its value is decided by the user. Techniques such as cross validation are often used to find a value of $\lambda$ that minimizes the objective function.

## LASSO as a quadratic program (QP)

Before we convert from statistical notation, it is worth pointing out that the goal of regression we are discussing is minimizing the "root sum of squares". We have a square in what would be the objective function, so in reality we will be building this as a "quadratic program", or QP for short.

As for the conversion, there are some modifications that must be made. For starters, the objective function of this QP is simply the RSS portion from above. The penalty portion becomes the constraint, with some handling for the absolute value as it is not a valid operation for linear programs.

Removing that operator requires breaking up each coefficient into two parts $b_j = b_j^+ - b_j^-$ where $b_j^+, b_j^- \geq 0$. The absolute value then is just $|b_j| = b_j^+ + b_j^-$.

Therefore, the QP for LASSO regression can be written as:

$$\begin{aligned} \min & \sum (y_i - \hat{y_i})^2 \\ \text{subject to:} & \sum_{j=1}^{p} \left(b_j^+ + b_j^-\right) \leq t \\ & \hat{y} = \sum_j b_j^+ x_{ij} - \sum_j b_j^- x_{ij} \\ & b_j^+, b_j^- \geq 0 \end{aligned}$$

This is the structure we will be working with for this project.

## The dataset

For this project we'll be using the classic `mtcars` toy dataset. Below is a description of the table from the R documentation.

> The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).
>
> R Documentation

The model for this dataset will be using the vehicles "miles per gallon" as the response variable and all others as predictor variables. So there will be 9 numeric predictors for just 32 rows. This is very important to note.

This table was chosen due to the small number of rows relative to the large number of variables. This is useful for this project as using too many variables on smaller datasets can cause coefficients to vary a lot and can result in many useless predictor variables. Each coefficient tries to estimate its own relationship with the response, but with so few observations, the model doesn't have enough information to estimate all coefficients reliably. This leads to high variability and unstable coefficient estimates.

LASSO is a great option for this kind of situation as it will, by design, drag any of the unnecessary predictor coefficients to 0 and remove them from the model. We anticipate that this dataset will allow us to showcase this functionality and how that penalty will change for different values of $\lambda$ or $t$.