2 Egg Drop Revisited

(a)

$$M(x,k) = M(x-1,k-1) + M(x-1,k) + 1$$

We will first deduce i, the optimal floor from which we will drop the first egg given we have x drops and k eggs. If the egg breaks after being dropped from floor i, we have reduced the problem to floors 0 through $i-1$, and the strategy can solve this subproblem using $x-1$ drops and $k-1$ eggs. The optimal strategy for $x-1$ drops and $k-1$ eggs can distinguish between $M(x-1,k-1)$ floors, so we should choose $i = M(x-1,k-1)+1$. On the other hand, if the egg doesn't break, we reduce the problem to floors $i+1$ to n with $x-1$ drops and k eggs. The maximum number of floors we can distinguish using this many drops and eggs is $M(x-1,k)$, so we can solve this subproblem for n as large as

$$i + M(x-1,k) = M(x-1,k-1) + M(x-1,k) + 1$$

(b) For base cases, we'll take $M(0,k) = 0$ for any k and $M(x,0) = 0$ for any x. Starting with $x = 1$, we compute $M(x,k)$ for all, $1 \le x \le k$, and do so again for increasing values of x, up until we compute $M(x,k)$ for all $1 \le x \le k$. We return $M(x,k)$. We compute xk subproblems, each of which takes constant time, so the overall runtime if $\Theta(xk)$.

(c) Again, starting with $x = 1$, we compute $M(x,k)$ for all, $1 \le x \le k$, and do so for increasing values of x. This time, we stop the first time we find that $M(x,k)$.

(d) Using the recurrence on $f(n,k)$ directly, the runtime of the algorithm is $O(n^2k)$. There are nk subproblems that each take $\mathcal{O}(n)$ time to solve.

(e) Because there are only n floors, the optimal number of drops, x will always be at most n. From part (b), we know the runtime is $\Theta(xk)$, so if $x \le n$, we know the runtime must be $\mathcal{O}(nk)$. (This is a very loose bound, but it is still much better than the naive $O(n^2k)$-time algorithm we'd get from using the recurrence on $f(n,k)$ directly.) While we're computing $M(x,k)$ for all $1 \le x \le k$, we only need to store $M(x1,k)$ and $M(x,k)$ for all x, i.e. we only ever need to store $\mathcal{O}(k)$ values. In particular, after computing $M(x,k)$ for all x, we can delete our stored values of $M(x1,k)$.

3 Knightmare

Solution. *We use length L bit strings to represent the configuration of rows of the chessboard (1 means there is a knight in the corresponding square and otherwise 0).*
The main idea of the algorithm is as follows: we solve the subproblem of the number of valid configurations of $(m-1) \times L$ chessboard and use it to solve the $m \times L$ case. Note that as we iteratively incrementing m, a knight in the m-th rows can only a ect configurations of rows $m+1$ and $m+2$. So we can denote $f(m,u,v)$ as the number of possible configurations of the first m rows with u being the $(m-1)$-th row and v being the m-th row, and then use dynamic programming to solve this problem. Let a list of bitstrings be valid if placing the knights in the first row according to the first bitstring, in the second row according to the second bitstring, etc. doesn't cause two knights to attack each other. Then we have $f(2,u,v) = 1$ if u, v are valid and 0 otherwise for all u, v pairs.

(a) **Subproblems:** $f(m,u,v)$ is the number of possible valid configurations $\mod 1337$ of the first m rows with u being the configuration of the $(m-1)$-th row and v being the configuration of the m-th row.

(b) **Recurrence and Base Cases:** For $m > 2$, $f(m,v,w) = \sum_{u:u,v,warevalid} f(m-1,u,v) \mod 1337$. For the base case, $f(2,u,v) = 1$ if u, v are valid and 0 otherwise for all u, v pairs. No other base cases are needed.

(c) **Proof of Correctness:** The only 2-row configuration of knights ending in row configurations u, v is the configuration u, v itself. So $f(2,v,w) = 1$ if u, v are valid. Otherwise, $f(2,v,w) = 0$. For $m > 2$, the above recurrence is correct because for any valid m-row configuration ending in v, w, the first $m-1$ rows must be a valid configuration ending in u, v for some u, and for this same u, the last three rows u, v, w must be also valid configuration. Moreover, this correspondence between m-row and $(m-1)$-row configurations is bijective.

(d) **Runtime and Space Complexity:**
See runtime to yield $\mathcal{O}(2^{3L}LM)$ as follows:
To bound the total runtime, first note that for each row, we iterate over all possible configurations of knights in the row and for each such configuration, we perform a sum over all possible valid configurations of the previous two rows. The time to check if a configuration is valid is $\mathcal{O}(L)$. Therefore, the time taken to compute the sub-problems for a single row is $\mathcal{O}(2^{3L}L)$ which gives us an overall runtime of $\mathcal{O}(2^{3L}LM)$ operations. Although there are a potentially exponential number of possible configurations (at most 2^{ML}), we are only interested in the answer mod 1337, which means that all of our numbers are constant size, and so arithmetic on these numbers is constant time. This gives us a final runtime of $\mathcal{O}(2^{3L}LM)$.
For space complexity, note that there are only $\mathcal{O}(M2^{2L})$ subproblems (M rows, $2L$ settings to the $(M-1)$-th row, and $2L$ settings to the M-th row). Each subproblem is a number of possible configurations $\mod 1337$, bounded above by 1337, so our total space is $\mathcal{O}(M2^{2L})$. Note that since each subproblem only depends on subproblems of the previous row, we could reduce this by a factor of M to $\mathcal{O}(2^{2L})$ by recycling space from earlier rows.

4 Standard Form LP

(a) $\min -\sum_i c_i x_i$

(b) $x_1 + s_1 = b_1, s_1 \geq 0$

(c) $-x_2 + s_2 = -b_2, s_2 \geq 0$

(d) Break it into two inequalities $x_3 \leq b_1$ and $x_3 \geq b_2$ and use the parts above

(e) $x_1 + x_2 + x_3 + s_1 = b_3, s_1 \geq 0$

(f) $\min t, y_1 \leq t, y_2 \leq t$

(g) Replace x_4 with $x^+ - x^-$ along with $x^+ \geq 0, x^- \geq 0$

5 Baker

(a) x = number of brownie batches

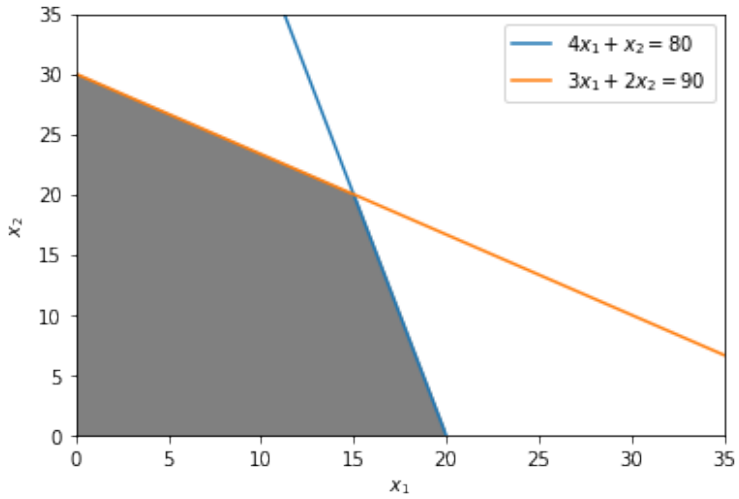
y = number of cookie batches

Maximize: $60x + 30y$

Linear Constraints:

$4x + y \leq 80$
$2x + 3y \leq 90$
$x \geq 0$
$y \geq 0$

The feasible region:



The vertices are $(x = 20, y = 0), (x = 15, y = 20), (x = 0, y = 30)$, and the objective is maximized at $(x = 15, y = 20)$, where $60x + 30y = 1500$. In other words, we can bake 15 batches of brownies and 20 batches of cookies to yield a maximum profit of 1500 dollars.

(b) There are lots of ways to solve this part. The most straightforward is to write and solve a system of inequalities checking when the objective of one vertex is at least as large as the objective of the other vertices. For example, for $(x = 15, y = 20)$ the system of inequalities would be

$$C \cdot 15 + 30 \cdot 20 \geq C \cdot 20$$

and

$$C \cdot 15 + 30 \cdot 20 \geq 30 \cdot 30$$

Doing this for each vertex gives the following solution:

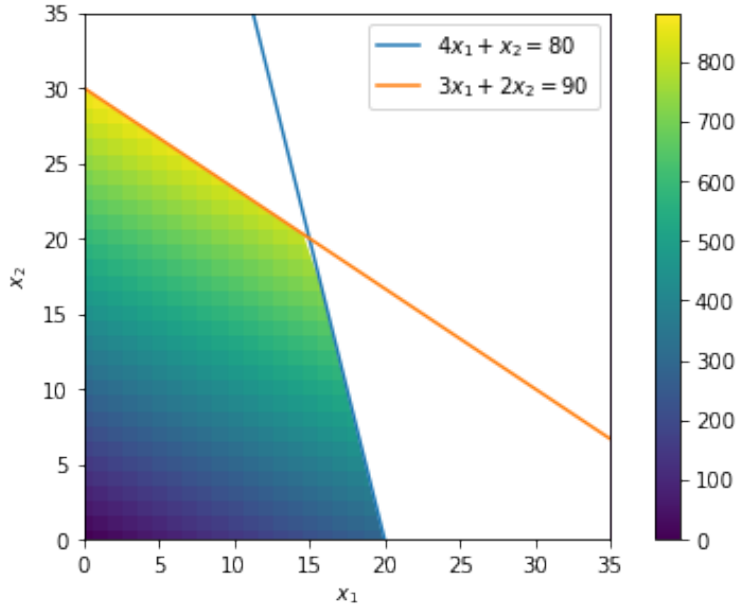
$(x = 0, y = 30) : C \leq 20$
$(x = 15, y = 20) : 20 \leq C \leq 120$

$(x = 20, y = 0) : 120 \leq C$

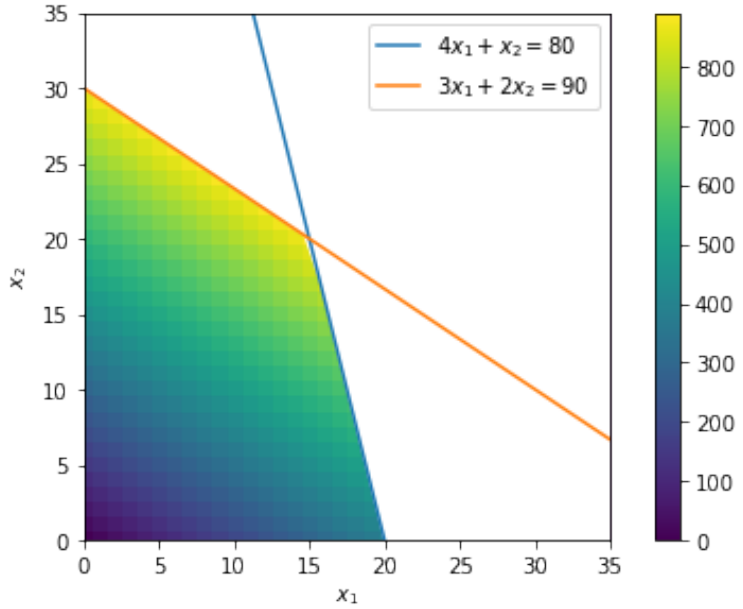
One should note that there is a nice geometric interpretation for this solution: Looking at the graph of the feasible region, as C increases, the vector $(C, 30)$ starts pointing closer to the x-axis. The objective says to find the point furthest in the direction of this vector, so the optimal solution also moves closer to the x-axis as C increases. When $C = 20$ or $C = 120$, the vector $(C, 30)$ is perpendicular to one of the constraints, and there are multiple optimal solutions all lying on that constraint, which are all equally far in the direction $(C, 30)$.

Below are some graphics to help build up your intuition about this problem.

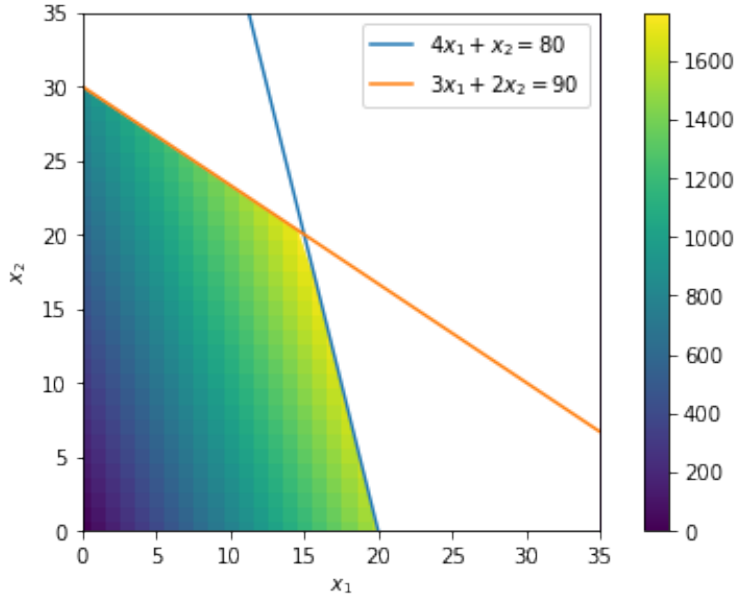
$C = 15 : (x = 0, y = 30)$ optimal:



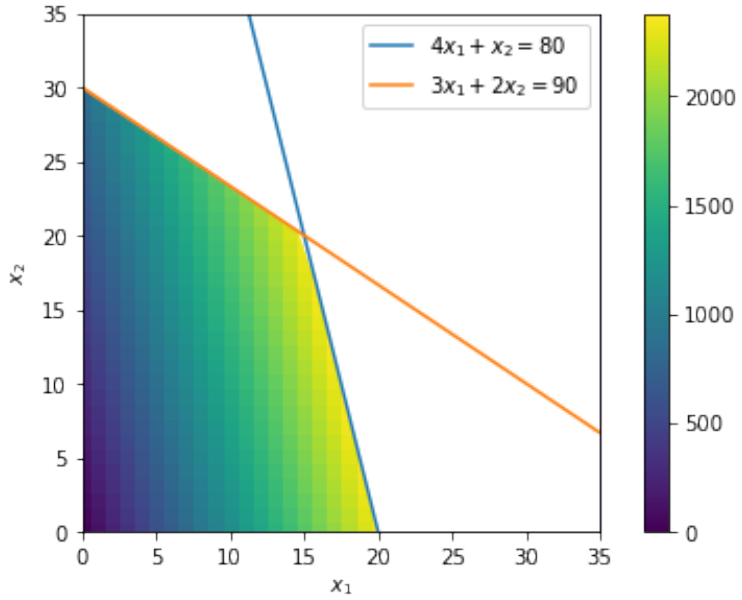
$C = 20 : (x = 0, y = 30)$ and $(x = 15, y = 20)$ optimal:



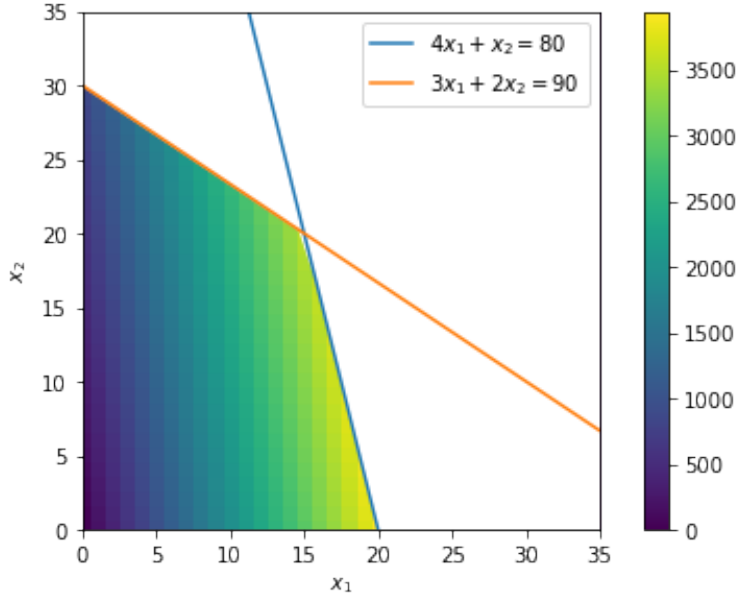
$C = 80 : (x = 0, y = 30)$ optimal:



$C = 120 : (x = 15, y = 20)$ and $(x = 20, y = 0)$ optimal:



$C = 200 : (x = 20, y = 0)$ optimal:



6 Meal Replacement

(a) Let a be the pounds of salmon we consume, b be the pounds of bread, and c be the pounds of squid. The objective is to minimize cost, and we have a constraint for each of protein/carbs/fat.

$$\begin{aligned} \min 5a + 2b + 4c \\ 500a + 50b + 300c &\geq 500 \\ 300b + 100c &\geq 800 \\ 500a + 25b + 200c &\geq 700 \\ a, b, c &\geq 0 \end{aligned}$$

(b) Let p, c, f be variables corresponding to the protein, carb, and fat constraints. The dual is:

$$\begin{aligned} \max 500p + 800c + 700f \\ 500p + 500f &\leq 5 \\ 50p + 300c + 25f &\leq 2 \\ 300p + 100c + 200f &\leq 4 \\ p, c, f &\geq 0 \end{aligned}$$

(c) The variables can be interpreted as the price per calorie for the protein, carb, and fat pills.
The objective says that the pharmacist wants to maximize the total revenue he gets from selling enough of these pills to us to meet our dietary needs.
The constraints say that no combination of pills should cost more than a pound of food providing the same dietary needs (otherwise, we would just buy that food instead of these pills).