# Problem 1: Optimization and Probability

1. The sequence is finite, so we differentiate to find the minimum value of θ:

If some of the *w* terms are negative in the series, this can create errors with optimization by creating local extrema, which will make the convergent solution of the optimization problem at least partially dependent on initial conditions.

1. We want to compare

Suppose that there is some x’ for which f(x) is maximized. Then, we have

Since the ranges of the summation are equivalent, we then have

However, because both x and the summation range are constant between f and g, it follows that

And so therefore

1. For a six-sided fair dice, the probability of rolling any given number is 1/6. Under the specified conditions, the expected number of points accumulated by the time a 1 is rolled is defined by the following recurrence, which can be solved for P:
2. To optimize L(p) at some p=pmax, we suppose that dL/dp = 0 for some value p = pmax. Taking the logarithm of this function, we see that the optimizing condition becomes

Therefore, the value of p which maximizes L(p) must also maximize L’(p). Therefore, we take the natural logarithm of both sides. We have

Differentiating both sides to find the optimizing value of p yields

This value of p represents the optimal “fairness” of the coin to provide the maximum total probability of obtaining the event sequence {H,H,T,H,T,T,H} in a series of 7 tosses. I.e. a coin which has a 4/7 chance of landing heads will maximize the probability of obtaining said sequence.

1. <REVISIT THIS> Differentiating, we have

The second term reduces to 0 for all components of the vector w that are not wk and 2wk otherwise, so

# Problem 2: Complexity

1. Since we have no constraints on size or position of six rectangles and assuming they are not disjoint, capturing all possible faces involves iteration across the entire *n x n* (2D) matrix, such that we have O(n2) complexity required to iterate across and draw all the faces.
2. We can only take steps down and right, therefore the cost function is equivalent to the Manhattan distance between the starting point and the destination, such that we have (assuming a starting point of (1,1) and that all movements incur the same unit cost) the following as the most efficient means of computing the cost of movement:

The above operation involves two absolute-value operations, two subtractions of one, and one addition of two integers of variable size. Ignoring runtime effects of the size of *i* and *j,* all of these are constant-time operations such that we have O(k) runtime for some constant k.

1. Whenever one is at step *n* in a stairwell, the total number of ways they can get to that stair is the sum of the total number of ways they could have gotten to that stair from every preceding stair in the stairwell (since steps can be any number of steps in length). Therefore, it follows that the total number of ways one can climb a stairwell of length *n* is given by
2. We have

Factoring, we have

Expanding the quadratic term gives

Which in turn gives