# HW 1

Due: Friday 10/03/2025 @ 11:59pm EST

## Disclaimer

I encourage you to work together, I am a firm believer that we are at our best (and learn better) when we communicate with our peers. Perspective is incredibly important when it comes to solving problems, and sometimes it takes talking to other humans (or rubber ducks in the case of programmers) to gain a perspective we normally would not be able to achieve on our own. The only thing I ask is that you report who you work with: this is **not** to punish anyone, but instead will help me figure out what topics I need to spend extra time on/who to help. When you turn in your solution (please use some form of typesetting: do **NOT** turn in handwritten solutions), please note who you worked with.

### Question 1: Monte-Carlo Algorithms (better runtime at the cost of being always correct) (20 points)

Consider being a participant on family feud where Steve Harvey has you compete against a person from the rival family. In this setting, Steve Harvey brings you and your opponent to a table with two buzzers and reads out a question. The first person to ring the buzzer gets to answer: if they get the question right, then they earn the opportunity to win points in the game. If they get the question wrong, their opponent earns the opportunity to win points in the game. Suffice to say that having a quick reaction time is important.

We, being tired and overworked and whatnot, need help in this game. Our reaction time is not what it would be if we were rested and unstressed, so if Steve Harvey finishes the question we are sure to lose the race to hit the buzzer. Being rather clever mathematicians, we devise a rather devious plot: we'll hit the buzzer **before** the question is fully read. Of course, we run the risk of getting the question wrong because we didn't wait to hear the whole thing, but we are rather confident that if we wait for the entire question to be read, our chance of winning is near zero. Surely its better to take our chances here right?

In such a scheme, the normal running/processing time is too slow for us: we need something faster. While modeling the correct answer as a function of what words are spoken (and have yet to be spoken) is really complicated, we can instead try to solve a somewhat simpler problem that has similar behaviour.

Now, rather than playing family feud, let us pretend that you are given $n$ numbers and want to pick one of those numbers. The condition here is that you want to pick a number **larger** than a median of all $n$ numbers. We could solve this in a few ways, the most efficient being to find the median of all $n$ numbers in linear time, and then perform a linear scan until we find a number larger than the median, which we pick. However, like our family feud scenario, we don't have time to perform a linear scan to calculate the median. Instead, we want to examine only $k$ of these numbers (where $k$ is a constant). After examining all $k$ numbers, we want to pick one of those $k$ numbers as our choice. Let us choose the largest of the $k$ numbers we have examined as our answer. What is the probability that the number we pick is correct (your answer should be a function of $k$)?

(hint: try to give an upper bound on the probability that our algorithm is wrong first).

**Question 2: Bounding Probabilistic Events (20 points)**

You are given a coin with bias $p$. Let us flip the coin $n$ times and observe the outcomes $X_1, X_2, \cdots, X_n$. Let us count the average number of successes $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$. Since we are constructing $\bar{X}$ as an empirical average, we know that $0 \leq \bar{X} \leq 1$. What we are interested in is how likely the value of $\bar{X}$ differs from the bias of the coin $p$. Compute an upper bound for the quantity $Pr[|\bar{X} - p| \geq \frac{p}{10}]$ using the Chebyshev inequality. Your answer should be a function of $n$: the number of coin tosses.

**Question 3: Combining Probability Distributions (20 points)**

Consider two probability distributions $f$ and $g$, both defined over the same continuous domain. A *convex* combination of two function $f_1$ and $f_2$ creates a third function $f_3$ which is defined as follows:

$$f_3 := \lambda f_1 + (1 - \lambda) f_2$$

for some value of $0 \leq \lambda \leq 1$. Please show that the convex combination of two probability distributions $f$ and $g$ (mentioned earlier) produces a function that is also a valid probability distribution.

**Question 4: Expectation Maximization MLE (20 points)**

Let us say you are clustering $n$ 1-dimensional positive-valued points, and you claim that there are $k = 2$ clusters these points are drawn from. Cluster one behaves according to a Geometric distribution $Pr[x_i; p] = (1-p)^{x_i - 1}p$ and cluster two behaves according to a Borel distribution $Pr[x_i; \mu] = \frac{e^{-\mu x_i}(\mu x_i)^{x_i - 1}}{x_i!}$. So, you choose to use soft-EM which has the machinery to deal with clusters that have pdfs inside them. The function we want to optimize for a our mixture is:

$$Q = \sum_{i=1}^{n}\sum_{j=1}^{k} \gamma_{ij}\Big(\log(\pi_j) + \log(Pr[x_i;\vec{\theta}_j])\Big) \qquad \texttt{s.t.} \sum_{j=1}^{k} \pi_j = 1$$

where $\vec{\theta}_j$ is the parameters for cluster $j$, $\gamma_{i,j=1}, \gamma_{i,j=2}, \cdots, \gamma_{i,j=k}$ forms a pmf for the *segmentation* of point $x_i$ into the $k = 2$ clusters, and $\pi_{j=1}, \pi_{j=2}, \cdots, \pi_{j=k}$ form a prior pmf over the clusters. Derive the MLE estimates for the parameters of each cluster (i.e. derive the MLE estimates for $p^*$ and $\mu^*$)

**Question 5: MLE Estimates (20 points)**

In lecture we found the MLE estimate for the Binomial distribution for the event when we know we have $k$ heads. In this question, consider being presented with a dataset $X = \{x_1, x_2, \cdots, x_n\}$ where each $x_i$ is a specific outcome of flipping a coin (with bias $p$) $m$ times. So, each $x_i$ is a string containing $m$ characters where each character either denotes a success or not a success. What is the MLE estimate for $p$ given this dataset?

**Extra Credit: Turning a Monte-Carlo Algorithm into a Las-Vegas Algorithm (20 points)**

IN question 1, we talked about a monte-carlo algorithm for picking a number larger than the median. In this extra credit question we want to repeat this experiment until it succeeds. For instance, lets call our algorithm in question 1 `pick_only_see_k(pts, k)` where `pts` is a sequence of $n$ numbers, and $k$ is the number of elements in `pts` we can examine before picking the largest of the ones we examined. An algorithm is called a monte-carlo algorithm if it can *fail* (i.e. produce the wrong answer). By being wrong sometimes, we can often design algorithms that are much more efficient than if we were to always produce a correct answer. An algorithm that always produces a correct answer is called a las-vegas algorithm. You might think las-vegas algorithms are always better, but consider that when running a las-vegas algorithm, rather than gambling with correctness (like a monte-carlo algorithm does), you are instead gambling with how long it will take (which is why we are interested in big-O notation as it is a worst case estimate).

In this question, we will design a las-vegas algorithm by using a monte-carlo algorithm as a building block (this is always possible). We will simply run our `pick_only_see_k(pts, k)` algorithm until it succeeds (there is a little more work to do here but for now let us ignore it). How many times should I expect to have to repeat `pick_only_see_k(pts, k)` until I get a correct answer? Your answer should be a functin of $k$.