Interview Preparation

Songyu Ye

November 19, 2025

Abstract

These are notes for my final round interview for the Jane Street Quantitative Trading Internship 2025.

Contents

1 Practice Problems

We start with a bankroll of 100 dollars. We want to maximize our winnings from each game while strongly minimizing the risk of ruin. In particular, one should think very carefully about bet sizing.

Example 1.1. You and I are playing a game. We will draw four cards form a deck of standard playing cards at random. The payout of the game is equal to 5^n dollars, where n is the number of red cards.

Using a binomial distribution, one can roughly estimate the payout of the game to be 81 dollars. We decide to make a 10 wide market on the game. In particular, I will buy the game from you for 86 dollars and sell it to you for 76 dollars.

Let's suppose that I buy one contract. So now you have 186 dollars. You are potentially on the hook to lose everything if I hit 4 red cards. But luckily there is a side bet. Before the first card and each subsequent card is flipped over, you are allowed to take a 1:1 side bet on whether the card is red or black. Find an optimal strategy to minimize your risk of ruin.

At the beginning of the game, suppose the first card is red. Then 1/8 of the time, you will owe me 625 dollars at the end of the game. 3/8 of the time, you will owe me 125 dollars at the end of the game. 3/8 of the time, you will owe me 25 dollars at the end of the game. 1/8 of the time, you will owe me 5 dollars at the end of the game. Therefore, your expected loss is 135 dollars.

Conversely, if the first card is black, then 1/8 of the time, you will owe me 125 dollars at the end of the game. 3/8 of the time, you will owe me 25 dollars at the end of the game. 3/8 of the time, you will owe me 5 dollars at the end of the game. 1/8 of the time, you will owe me 1 dollar at the end of the game. Therefore, your expected loss is 27 dollars.

Therefore, you should take (135 - 27)/2 = 54 dollar sidebet on red for the first card to minimize your expected loss in the worst case scenario.

Now let's suppose the first card comes red. You now have 186 + 54 = 240 dollars. Then let's reason about how to hedge moving forward. If you hit red next, then you owe 25 dollars 1/4 of the time, 125 dollars 1/2 of the time, and 625 dollars 1/4 of the time. Therefore, your expected loss is 225 dollars. If you hit black next, then you owe 5 dollars 1/4 of the time, 25 dollars 1/2 of the time, and 125 dollars 1/4 of the time. Therefore, your expected loss is 45 dollars. Therefore, you should take the side bet of (225 - 45)/2 = 90 dollars on red to minimize your expected loss in the worst case scenario.

Say the next card comes black. You now have 240 - 90 = 150 dollars. Then let's reason about how to hedge moving forward. If I hit red next, then half the time you will owe me 25 dollars and the other half of the time you will owe me 125. If I hit black next, then half of the time I owe you 5 dollars and half the time you owe me 25 dollars. Therefore, if I hit red, your expected loss is 75 dollars, and if I hit black, your expected loss is 15 dollars. Therefore, you should take the side bet of 30 dollars on red to minimize your expected loss in the worst case scenario.

Say the third card comes black. At this point, you have 150 - 30 = 120 dollars. If the last card comes red, you owe me 25 dollars, and if the last card comes black, you owe me 5 dollars. Therefore, you should take a side bet of (25 - 5)/2 = 10 dollars on red to minimize your expected loss in the worst case scenario. No matter what happens, you will lose 15 dollars. So you end up with 105 dollars no matter what happens. This is the 5 dollars that comes from the bid-ask spread. So you have hedged away all of the risk of ruin while still making a profit.

Key takeaways from the game:

- Always think about worst case scenarios and try to minimize your expected loss in the worst case scenario.

- Don't just pick sizes blindly. Think about how your bet sizes affect your risk of ruin.
- Use side bets to hedge your risk.
- Rule of thumb: Don't risk more than 40% of your bankroll on a single bet, unless the odds are heavily in your favor.
- When keeping track of your bankroll, remember to include any credits from selling the contract, as well as any gains/losses from side bets. The point of hedging is to walk away with a guaranteed profit from the bid-ask spread.

Example 1.2. Suppose a fair coin is used to generate a sequence of H and Ts, what is the probability that HHT occurs before HTT

2. Suppose you can craft a deck with cards numbered 1 to 5. After you craft, I will check and pick a card. Then a random card is picked from the deck. If the value matches, you will gonna pay the value on the card. How to craft a deck to minimise my expected winning

Example 1.3. We are playing with a 40 card deck of cards numbered 1 to 10. A player will draw four cards from the deck. The value of the game is the sum of the values of the red cards.

One can estimate the expected value of the game to be 11 dollars. We decide to make a 10 wide market on the game. In particular, I will buy the option to play the game from you for 16 dollars and sell it to you for 6 dollars. There is a 1:1 sidebet where you can bet on whether the next card drawn is ≥ 6 or < 6. Find an optimal sidebet strategy. With this sidebet strategy, how many contracts should you sell?

Before any cards are drawn, we can consider four situations. Since you lose more on larger numbers, your sidebet should be biased towards larger numbers. Suppose you bet x dollars on the next card being ≥ 6.

- If the next card is black and ≥ 6, then you win x dollars from the sidebet.
- If the next card is black and < 6, then you lose x dollars from the sidebet.
- If the next card is red and ≥ 6, then you win $x - 8$ dollars from the sidebet. Note that 8 is the average value of the red cards ≥ 6.
- If the next card is red and < 6, then you win $-x - 3$ dollars from the sidebet.

We want to use the sidebet to minimize our expected loss in the worst case scenario. Therefore, we want to solve the following equation:

$$x - 8 = -x - 3.$$

Solving this equation gives $x = 2.5$. Therefore, we should bet 2.5 dollars on the next card being ≥ 6.

Then the two worst case scenarios for us are losing 5.5 dollars. Since each sidebet is independent (the removed cards are negligible), we do not have to dynamically adjust our sidebet sizes.

Now we calculate how many contracts we should sell. The worst case scenario is that I pick four red 10s or four red 5s. In this case, you will owe me 40 dollars from the main bet, win 10 dollars from the sidebets, and make 16 dollars from selling the contract. Therefore, your total loss is 14 dollars. Since you start with 100 dollars, you can sell at most 7 contracts. However you should only sell 3 contracts to be safe, abide by the rule of not risking more than 40% of your bankroll on a single bet.

Key takeaways from the game:

- Use expected values (like 8, 3) within branches to summarize future payoffs.
- Equalize losses between branches to flatten your worst-case profile.
- Minimizing expected loss in the worst case on average.
- Another viable hedge is to reason about the worst case scenario on both sides of the sidebet absolutely.
- Reason about the game before you decide how many contracts to sell.

Example 1.4. Payout = 5^n dollars where n = number of reds in four cards, but after any card you may cancel the game by paying \$50. Two players alternate control: A decides to continue or cancel on odd turns, B on even turns. Find equilibrium stopping thresholds. (Tests dynamic stopping games & expected value reasoning.)

Example 1.5. You roll a fair six-sided die once. The payout is 2^X dollars, where X is the number on the die. Before the roll, you may trade a 1:1 side bet on whether the result will be even or odd.

The fair value is $\frac{2^1+2^2+2^3+2^4+2^5+2^6}{6} \approx 21.33$ dollars. Suppose the contract is trading at 15 25 dollars.

One hedging strategy is to bet $\frac{(2^6+2^4+2^2)-(2^5+2^3+2^1)}{6} = 7$ dollars on even. Then if the opponent rolls even, on average you lose them $\frac{2^6+2^4+2^2}{3} - 7 = 21$ dollars; if they roll odd, on average you lose them $\frac{2^5+2^3+2^1}{3} + 7 = 21$ dollars. The loss in each case is equal and equal to 21.

In the worst case scenario, if you sell the contract for 25 dollars, you might lose 64 dollars, but make 25 + 7 = 32 dollars from selling and hedging, for a net loss of 32 dollars. This means you can sell 1 contract with this hedge before risking more than 40% of your bankroll.

The other hedging strategy would be to look at worst case scenario. You could lose 64 or 32 and you want those to be equal so you could hedge say 16 dollars by betting that amount on even. Therefore, if it came up 6 you would have $100 + 25 + 16 - 64 = 77$. If it came up 5 you would have

$100 + 25 - 16 - 32 = 77$ so the worst case scenario you would lose 23 dollars. With this hedging strategy, you could sell 2 contracts.

Example 1.6. You flip 3 fair coins. Payout equals $10(\#\text{ heads})^2$. Before each flip, you can bet 1:1 on the next coin's outcome.

The fair price is $10(0^2 \cdot \frac{1}{8} + 1^2 \cdot \frac{3}{8} + 2^2 \cdot \frac{3}{8} + 3^2 \cdot \frac{1}{8}) = 30$ dollars. However, if you sell a contract for 30 dollars, you can lose up to 90 dollars if all coins are heads. You can hedge dynamically as follows:

Before the first flip, if the first coin is heads, then you have a 1/4 chance of owing 90 dollars, a 1/2 chance of owing 40 dollars, and a 1/4 chance of owing 10 dollars. Expected loss is 45 dollars. If the first coin is tails, then you have a 1/4 chance of owing 40 dollars, a 1/2 chance of owing 10 dollars, and a 1/4 chance of owing 0 dollars. Expected loss is 15 dollars. Therefore, you should bet (45 - 15)/2 = 15 dollars on heads for the first flip.

So say the first coin is heads. You now have 115 dollars. Before the second flip, if the second coin is heads, then you have a 1/2 chance of owing 90 dollars and a 1/2 chance of owing 40 dollars. Expected loss is 65 dollars. If the second coin is tails, then you have a 1/2 chance of owing 40 dollars and a 1/2 chance of owing 10 dollars. Expected loss is 25 dollars. Therefore, you should bet (65 - 25)/2 = 20 dollars on heads for the second flip.

So say the second coin is tails. You now have 115 - 20 = 95 dollars. Before the third flip, if the third coin is heads, then you owe 40 dollars; if the third coin is tails, then you owe 10 dollars. Therefore, you should bet (40 - 10)/2 = 15 dollars on heads for the third flip.

Finally, say the third coin is tails. You now have 95 - 15 = 80 dollars. No matter what happens, you owe 10 dollars. So you end up with 70 dollars no matter what happens. However, you sold the contract for 30 dollars, so you broke exactly even.

Proceeding this way, you see that with optimal hedging, even if you get all heads, you will always break even. Therefore, if you sell at a spread, you can sell as many contracts as you want without risking ruin.

Example 1.7. Urn starts with 4 red and 4 black balls. We draw 3 balls without replacement. Payout equals 50 times the number of reds.

First we calculate the fair price. There are 4 ways to get 0 reds, 24 ways to get 1 red, 24 ways to get 2 reds, and 4 ways to get 3 reds.

There are $\binom{8}{3} = 56$ ways to draw 3 balls. Therefore, the fair price is

$$\frac{4}{56} \cdot 0 + \frac{24}{56} \cdot 50 + \frac{24}{56} \cdot 100 + \frac{4}{56} \cdot 150 = 75 \text{ dollars.}$$

Also note that expected number of reds in draws without replacement and so by linearity of expectation, the fair price is also $3 \cdot \frac{4}{8} \cdot 50 = 75$ dollars.

Now suppose that we make a 10 wide market on the game. In particular, I will buy the option to play the game from you for 80 dollars and sell it to you for 70 dollars. There is a 1:1 sidebet where you can bet on whether the next ball drawn is red or black. Find an optimal sidebet strategy. With this sidebet strategy, how many contracts should you sell?

Suppose the first ball drawn is red. Then there are 3 red balls and 4 black balls left. Then there is a $4/7 * 3/6 = 2/7$ chance of getting two blacks, $2 * 4/7 * 3/6 = 4/7$ chance of getting one black, and $3/7 * 2/6 = 1/7$ chance of getting zero blacks. In particular, there will be an expected loss of $2/7 * 50 + 4/7 * 100 + 1/7 * 150 = 650/7$ if the first ball comes red.

Suppose the first ball comes black. Then there is a $4/7 * 3/6 = 2/7$ chance of getting two reds, $2 * 4/7 * 3/6 = 4/7$ chance of getting one red, and $3/7 * 2/6 = 1/7$ chance of getting zero reds. In particular, there will be an expected loss of $2/7 * 100 + 4/7 * 50 + 1/7 * 0 = 400/7$ if the first ball comes black.

Thus in order to make the expected losses equal, we should bet $125/7$ dollars on red. Say it comes red. Then we currently have $100 + 80 + 125/7 = 1385/7$ dollars. If the second ball is red, then there are 2 red balls and 4 black balls left. There is a $2/6$ chance of losing 150 and a $4/6$ chance of losing 100. In this case, the expected loss is $2/6 * 150 + 4/6 * 100 = 700/6$.

If the second ball is black, then there are 3 black balls and 3 red balls left. There is a $3/6$ chance of losing 50 and a $3/6$ chance of losing 100. In this case, the expected loss is $3/6*50+3/6*100 = 450/6$.

Therefore, we should bet $(700/6 - 450/6)/2 = 125/6$ dollars on red for the second ball. However, since there are only 3 red balls and 4 black balls left, betting on red is an unfavorable bet. However, we still need to protect ourselves from the risk of ruin.

Say the second ball comes red. Then we currently have $1385/7+125/6 = (1385*6+125*7)/(7*6) = 9185/42$ dollars.

The third ball will come red with probability $2/6$ and black with probability $4/6$. The expected loss if the third ball is red is 150 dollars, and the expected loss if the third ball is black is 100 dollars. Weighing these by their probabilities, we see that we need to bet $(2/6 * 150 - 4/6 * 100)/2 = -50/6$ dollars on red for the third ball. In other words, we should bet $50/6$ dollars on black for the third ball. Say the third ball comes red. Then we currently have $9185/42 - 50/6 = (9185 - 350)/42 = 8835/42$ dollars. We also owe 150 dollars. Therefore, we end up with $8835/42 - 150 = (8835 - 6300)/42 = 2535/42$ dollars. This was very close to the worst case scenario, but we still managed to avoid ruin. However we could not survive being short more than 1 contract.

Example 1.8. There are two independent fair coins, A,B. Payout is 100 if both are heads, 0 otherwise. You can make 1:1 bets on each coin's outcome.

Say the first coin is heads. Then you owe 100 dollars with probability 1/2 and 0 dollars with probability

1/2. Expected loss is 50 dollars. If the first coin is tails, then you owe 0 dollars no matter what. Therefore, you should bet (50 - 0)/2 = 25 dollars on heads for the first coin.

So we flip the first coin. Say it comes up heads. You now have 125 dollars. Then for the second coin, if it is heads, you owe 100 dollars; if it is tails, you owe 0 dollars. Therefore, you should bet (100 - 0)/2 = 50 dollars on heads for the second coin.

Say the second coin comes up tails. You now have 125 - 50 = 75 dollars. So you end up with 75 dollars no matter what happens. If the second coin came up heads, you would make 50 dollars from the sidebet and pay 100 dollars from the main bet, for a net loss of 25 dollars. So you would end up with 75 dollars no matter what happens. So we see that hedging allows us to realize a sure final amount of 75 dollars. Therefore, if you sell the contract for more than 25 dollars, you can make a guaranteed profit.

Example 1.9. Biased coin with $P(\text{heads}) = 0.6$. We flip twice with payout $5^{\#\text{ heads}}$. The payout of the hedge is $(1 - .6)/.6 = 2/3$ dollars heads, -1 dollar tails.

The fair price is $0.6(0.6 \cdot 25 + 0.4 \cdot 5) + 0.4(0.6 \cdot 5 + 0.4 \cdot 1) = 11.56$ dollars. How many contracts would you sell for 12 dollars?

If you flip heads first, you owe 25 with probability 0.6 and 5 with probability 0.4. Expected loss is 17 dollars. If you flip tails first, you owe 5 with probability 0.6 and 1 with probability 0.4. Expected loss is 3.4 dollars. But since heads is going to happen 60% of the time, you should weight the two scenarios accordingly. Explicitly, you should solve the following equation:

$$17 - \frac{2}{3}x = 3.4 + x$$

because we want the expected loss to be equal in both scenarios. Solving this equation gives $x = 8.16$.

Say that the first flip comes heads. You now have $100 + 12 + 2/3 * 8.16 \approx 117.5$ dollars. Therefore you should solve

$$25 - 2/3x = 5 + x$$

which gives $x = 20/(5/3) = 12$. Suppose the second coin also comes heads. Then you have $117.5 + 8 - 25 = 100.5$.

Say the first flip comes tails. Then you have $112 - 8.18 \approx 104$. You now want to solve

$$5 - 2/3x = 1 + x$$

and so $x = 4/(5/3) = 12/5$. Say the second flip also comes tails. Then you have $104 - 1 - 2.4 = 100.6$. So we hedged away all the risk and the answer is you should sell all the contracts that you can at 12 dollars.

Example 1.10. Consider a game with 2 dice, where you cannot see the opponent's die. You can take your own die value, your opponent's die value, or defer by paying 20 cents. What is your optimal strategy?

Assume this model: each round you roll your die and see $x \in \{1, \ldots, 6\}$; the opponent rolls a hidden die. You may (1) take x, (2) take the opponent's hidden value (mean 3.5), or (3) pay $c = 0.2$ to defer and reroll both dice, with the same choices again. Repeat allowed.

Let V be the optimal expected value before seeing a roll. After paying to defer you return to the same state, so the "continue" value is $V - c$. Given a seen x, the stop value is $S(x) = \max(x, 3.5)$. Optimal choice at x is $\max\{S(x), V - c\}$. Therefore:

$$V = \mathbb{E}_x\big[\max\{S(x), V - c\}\big] = \frac{1}{6}\sum_{x=1}^{6} \max\{\max(x, 3.5), T\}$$

where $T := V - c$.

For any optimal solution we must have $T \geq 3.5$ (otherwise you would never defer). Work piecewise on T:

If $3.5 \leq T < 4$:

$$V = \frac{1}{6}(3T + 4 + 5 + 6) = 0.5T + 2.5$$

Set $T = V - 0.2$ gives $V = 4.8 \Rightarrow T = 4.6$, which contradicts $T < 4$. Discard.

If $4 \leq T < 5$:

$$V = \frac{1}{6}(3T + T + 5 + 6) = \frac{2}{3}T + \frac{11}{6}$$

Set $T = V - 0.2$:

$$V = \frac{2}{3}(V - 0.2) + \frac{11}{6} \Rightarrow V = 5.1, \quad T = V - 0.2 = 4.9$$

which is consistent with $4 \leq T < 5$.

Hence the fixed point is $V = 5.1$ and $T = 4.9$.

Decision rule: Defer (pay 0.2) if $S(x) = \max(x, 3.5) < 4.9$, i.e., for $x \in \{1, 2, 3, 4\}$. Stop if $x \in \{5, 6\}$. When stopping, take your own die (5 or 6 beats the hidden die's 3.5).

So the optimal strategy is: pay to defer when your die is 1-4; take your die when it is 5 or 6; never choose the hidden die directly. The optimal expected value from the start is $V = 5.1$ (in the same units as the die; if die are dollars, that's 5.10).

In particular, deferral is always worth 4.9 regardless of how many times you've deferred before, since the state resets. So it is never optimal to tkae the hidden die directly.

Example 1.11. Flip 4 fair coins and note the parity of each. You are allowed to reroll any instance of TT. You cannot flip one coin at a time; you must reroll both coins together. What is the expected number of Hs at the end?

Let t be the number of tails and a_t be the probability of ending with 0 tails starting from t tails. The base cases are $a_0 = 1$ (already all heads). Then $a_1 = 0$ and $a_2 = \frac{1}{4} + \frac{1}{4}a_2$ so $a_2 = 1/3$. Also $a_3 = 1/4a_1 + 1/4a_3 + 1/2a_2 = 1/4a_3 + 1/6$ so $a_3 = 2/9$. Finally, $a_4 = 1/4a_2 + 1/4a_4 + 1/2a_3 = 1/4a_4 + 1/12 + 1/9$ so $a_4 = 7/27$.

The starting number of tails is distributed binomially with $n = 4$ and $p = 1/2$. Therefore, the expected number of heads at the end is

$$a_0/16 + 4a_1/16 + 6a_2/16 + 4a_3/16 + a_4/16 = 7/27$$

The expected number of heads is $4 * 7/27 + 3 * (1 - 7/27) = 88/27 \approx 3.26$.

Example 1.12. Flip 4 fair coins and notes each outcome. After seeing them, he may flip any pair of tails again (i.e., reroll those two coins). He continues as long as he has fewer than 3 heads—once he has 3 or 4 heads, there's no TT pair, and the game stops. Each coin flip counts individually (so each pair reroll adds 2 to the flip count). Find the expected total number of flips, including the first 4.

Let $E[h]$ = expected number of individual flips remaining (not counting the initial 4) starting with h heads.

If you have $h \geq 3$, stop: $E[3] = E[4] = 0$.

Otherwise:

$$E[h] = 2 + \frac{1}{4}E[h] + \frac{1}{2}E[h+1] + \frac{1}{4}E[h+2]$$

Now we can solve the system:

$$\begin{aligned} E[2] &= 2 + \frac{1}{4}E[2] + \frac{1}{2} \cdot 0 + \frac{1}{4} \cdot 0 \Rightarrow E[2] = \frac{8}{3} \\ E[1] &= 2 + \frac{1}{4}E[1] + \frac{1}{2} \cdot \frac{8}{3} + \frac{1}{4} \cdot 0 \Rightarrow E[1] = \frac{40}{9} \\ E[0] &= 2 + \frac{1}{4}E[0] + \frac{1}{2} \cdot \frac{40}{9} + \frac{1}{4} \cdot \frac{8}{3} \Rightarrow E[0] = \frac{176}{27} \end{aligned}$$

and so the total expected number of flips is

$$\frac{1}{16}E[0] + \frac{4}{16}E[1] + \frac{6}{16}E[2] = 68/27$$

and the final answer is $4 + 68/27 = 176/27 \approx 6.52$ flips.

Example 1.13 (Coupon collector with 4 coupons). We start with no coupons. Let k be the number of distinct coupons we have collected so far. The probability of getting a new coupon is $\frac{4-k}{4}$. Therefore, the expected number of trials to get a new coupon is $\frac{4}{4-k}$. Therefore, the expected number of trials to collect all 4 coupons is

$$\frac{4}{4} + \frac{4}{3} + \frac{4}{2} + \frac{4}{1} = \frac{25}{3}$$

The same method can be used to solve the general coupon collector problem with n coupons. The expected number of trials to collect all n coupons is nH_n, where H_n is the n-th harmonic number.

Example 1.14. Take a deck of cards numbered 1 to 10, with 4 copies of each card (40 cards total). Three cards are dealt face-down on the table each round. The game presents a market for the sum of the cards face values (each card is 1-10 with equal probability). Say the market is 15 for 100, 100 at 25. How much would you pay to look at one card before deciding to buy/sell, assuming that you have to trade?

Contract is linear on the sum S of the three cards. Bid/ask is 15-25, so with no info you'd sell at 15. Baseline P/L: $15 - \mathbb{E}[S] = 15 - 16.5 = -3/2$.

If we peek one card $X = x$, the two unseen cards are drawn from the remaining 39 cards whose mean is $(220 - x)/39$. Therefore:

$$\mu_x := \mathbb{E}[S|X = x] = x + 2\frac{220 - x}{39} = \frac{440 + 37x}{39}$$

For optimal action given x: buy if $\mu_x > 20$ (mid), else sell. Solving $(440 + 37x)/39 > 20$ gives $x > 340/37$. So only $x = 10$ triggers a buy; for $x \in \{1, \ldots, 9\}$ you sell.

The expected P/L with the signal is:

$$\frac{1}{10}\left[\sum_{x=1}^{9}\big(15 - \mu_x\big) + \big(\mu_{10} - 25\big)\right] = -\frac{35}{26}$$

Therefore the value of the peek, which is the improvement over baseline, is:

$$-\frac{35}{26} - \left(-\frac{3}{2}\right) = \frac{2}{13} \approx 0.1538$$

For 100 contracts, the value is 15.38 units.

Example 1.15. Suppose you have a bankroll of 100 dollars. Take a deck of cards numbered 1 to 10, with 4 copies of each card (40 cards total). Three cards are dealt face-down on the table each round. The game presents a market for the sum of the cards face values (each card is 1-10 with equal probability). Say the market is 11.5 for 5, 5 at 21.5. How much would you pay to look at one card before deciding to buy/sell, assuming that you have to trade?

Per unit, the peek is worth $\frac{185}{78} \approx 2.372$ dollars.

Let S be the sum of the three cards. With one peek revealing $X = x$, we have

$$\mu_x := \mathbb{E}[S|X = x] = x + 2\frac{220 - x}{39} = \frac{440 + 37x}{39}$$

The market is 11.5-21.5 (center 16.5, half-width 5). If you must trade one unit, you buy if $\mu_x > 16.5$ and sell if $\mu_x < 16.5$. Your loss per unit after seeing x is the distance from μ_x to the nearer edge:

$$\text{loss}(x) = \min(\mu_x - 11.5,\ 21.5 - \mu_x) = 5 - |\mu_x - 16.5|$$

Without a peek the loss is always 5 per unit, so the value of the peek is

$$\mathbb{E}[|\mu_X - 16.5|] = \frac{1}{10}\sum_{x=1}^{10}\left|\frac{440+37x}{39} - \frac{33}{2}\right| = \frac{185}{78}$$

If you can trade multiple units subject to your 100bankroll and the posted sizes (5 at each side), the optimal sizing is: sell up to 5 units when $x \le 5$; buy up to 4 units when $x \ge 6$ (cannot afford 5×21.5). Then the peek's total value (vs. best no-peek choice, which is buying 4 to lose 20) is $\frac{555}{52} \approx 10.67$. If instead you must trade exactly one unit, use the per-unit value $\frac{185}{78}$.

Suppose the peek is mispriced and u have to pay 2 dollars per contract to peek. Suppose that the volume can be whatever you want. How many contracts would you trade subject to the condition you can't go bankrupt?

4 contracts.

The worst-case loss per contract equals market loss of 18.5 (since $S \in [3, 30]$ and you must trade at 11.5/21.5) plus peek fee of 2, totaling 20.5 per contract.

With a bankroll of 100, we must satisfy $Q \cdot 20.5 \le 100$ where Q is the number of contracts, which implies $Q \le 4.87$. Since we need an integer number of contracts, this means $Q = 4$.

While the peek reduces expected loss per unit by $\frac{2}{13} \approx 0.154$ compared to no peek, the net expected value per unit becomes negative after the 2 fee. Therefore, we don't scale beyond what the no-bankruptcy constraint allows.

Example 1.16. You have bankroll W. You bet a fraction f of it each round on an independent gamble with probability of win p, loss probability $q = 1 - p$, win payoff $+b \cdot fW$ (you gain b for each dollar risked) and loss payoff $-fW$. Assume the edge $p(b+1) - 1 > 0$ so the gamble has positive expected value and you can repeat it many times.

Your wealth after one round is $W(1 + bf)$ with probability p and $W(1 - f)$ with probability q. The quantity to maximize is expected log-wealth

$$G(f) = p\log(1 + bf) + q\log(1 - f).$$

Differentiate and set to zero:

$$G'(f) = \frac{pb}{1+bf} - \frac{q}{1-f} = 0,$$

so

$$\frac{pb}{1+bf} = \frac{q}{1-f}.$$

Cross-multiply and solve:

$$pb(1-f) = q(1+bf) \quad \Rightarrow \quad pb - q = bf(p+q) = bf,$$

hence the Kelly fraction is

$$f^* = \frac{pb-q}{b}.$$

In the special even-money case $b = 1$ this reduces to $f^* = 2p - 1$.

Example 1.17 (Kelly criterion with side bets)**.** You have a bankroll of 100 dollars. A fair coin will be flipped once. If it comes up heads, you win 200 dollars; if tails, you lose 100 dollars. Before the coin flip, you can place a side bet on tails at 1:1 odds. How much should you bet on tails in the side bet to maximize your expected logarithmic utility? How many contracts of the main bet should you take?

If you bet x dollars on tails in the side bet and take Q contracts of the main bet, your final wealth is:

- Heads: $100 + Q \cdot 200 - x$
- Tails: $100 - Q \cdot 100 + x$

We want to reduce the variance between the two outcomes, so we set them equal:

$$100 + 200Q - x = 100 - 100Q + x$$

Solving this gives $x = 150Q$. Therefore, if we take Q contracts of the main bet, we should bet $150Q$ dollars on tails in the side bet to minimize variance. We can spend at most $250Q$ dollars in total, so we must have $250Q \leq 100$, which implies $Q \leq 0.4$.

When $Q = 0.4$, our final wealth is always 120 dollars, regardless of the coin flip outcome.

2 ML and Programming Questions

The package we are most interested in is pandas. It provides data structures and functions needed to manipulate structured data seamlessly. Recall some basic operations that one can perform with pandas:

- Reading data: `pd.read_csv('file.csv')`
- Basic operations:

- Select columns: `df['column_name']` or `df.column_name`
- Filter rows: `df[df.column > value]`
- Sort: `df.sort_values('column')`
- Group by: `df.groupby('column').mean()`

• Data manipulation:

- Add column: `df['new_column'] = values`
- Drop missing values: `df.dropna()`
- Fill missing values: `df.fillna(value)`
- Merge dataframes: `pd.merge(df1, df2, on='key')`

• Aggregations:

- Summary statistics: `df.describe()`
- Mean/median: `df.mean()`, `df.median()`
- Count unique values: `df['column'].value_counts()`

2.1 Resume items

Cornell Quant Club — Machine Learning Subteam

- Deep learning trading algorithms: You designed and tuned models that make buy/sell decisions automatically. Implemented data pipelines and training logic in Python; optimized latency-critical components in C++ for execution speed.
- Neural networks for financial time series: Built RNNs (LSTMs, GRUs) and CNNs that learn temporal and local patterns in stock-price sequences.
- Frameworks: Used TensorFlow and PyTorch for model definition, training loops, and GPU acceleration.
- Latency optimization: Used C++ for order-book simulation and low-level numerical routines to minimize compute and communication delays in backtesting or live trading.
- Backtesting and evaluation: Ran simulations to compare strategy performance under different market scenarios.

Example 2.1 (Processing financial data with pandas and computing RSI).

```
        import pandas as pd, numpy as yfinance as yf

        data = yf.download("AAPL", period="60d", interval="1m")
        data["rsi"] = compute_rsi(data["Close"], 14)
        data["target"] = np.sign(data["Close"].shift(-5) - data["Close"])
   # 5-min horizon
```

Example 2.2 (RNN). For sequences $x_1, x_2, \ldots, x_T$, we often want to predict

$$y_t = f(x_1, x_2, \ldots, x_t)$$

where each new input depends on previous ones. For example, in language modeling, predicting the next word depends on all prior words, so the x_i are word embeddings and y_t is the next word prediction.

A feedforward network can't easily handle varying sequence lengths or reuse past information. Recall that a feedforward network is just:

$$y_t = W_2\sigma(W_1 x_t + b_1) + b_2$$

where σ is a nonlinearity like ReLU (Rectified Linear Unit, meaning $\sigma(x) = \max(0, x)$). This is used because the negative part does not make too much sense (in the general context of neural networks). This processes each x_t independently, losing context.

A single neuron computes a weighted linear combination followed by a nonlinearity:

$$y = \sigma(w^\top x + b),$$

where $x \in \mathbb{R}^d$, $w \in \mathbb{R}^d$, and σ is a nonlinear activation such as ReLU or $\tanh$.

To approximate more complex functions, we can stack several neurons together in a *layer*. Each neuron learns a different direction in input space, producing intermediate features:

$$z_i = \mathrm{ReLU}(x^\top w_i), \qquad i = 1, \ldots, 3,$$

and the layer output is a linear combination of them:

$$y = \sum_{i=1}^{3} a_i \, \mathrm{ReLU}(x^\top w_i) + b.$$

Each hidden neuron defines a half-space in $\mathbb{R}^d$. The sum of several ReLU activations yields a piecewise-linear approximation to any target function. A single wide layer can approximate any continuous function (universal approximation theorem), but may require exponentially many neurons. Stacking layers—adding *depth*—greatly reduces width requirements: deeper networks reuse intermediate features to represent complex functions more efficiently.

An RNN introduces a hidden state h_t that summarizes what has happened so far. At each time step:

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$$
$$y_t = W_y h_t + c$$

Here x_t is the input vector at time t, h_t is the hidden state (memory of the past), y_t is the output, W_x, W_h, W_y are learnable weight matrices, and $\tanh$ is a nonlinearity ensuring stability.

So h_t is the "memory" passed forward; each new step updates that memory using the new input. To train the network, compute losses at each step (e.g. difference between predicted and true y_t), sum them, and backpropagate through all unrolled steps. This multiplies many Jacobians:

$$\frac{\partial L}{\partial W_h} = \sum_t \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \cdots$$

The repeated multiplication can make gradients vanish (go to zero) or explode (blow up) — the core limitation of vanilla RNNs.

Example 2.3 (Stochastic gradient descent)**.** You have a model $h_\theta(x)$ with parameters θ (these are the weights). You have data (x_i, y_i). You want to make $h_\theta(x_i)$ close to y_i.

So you define a loss function

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(h_\theta(x_i), y_i)$$

(e.g. squared error $\ell = (\hat{y} - y)^2$).

The goal is to find

$$\theta^\star = \arg\min_\theta L(\theta).$$

The gradient $\nabla_\theta L(\theta)$ is a vector showing how the loss changes if you change each parameter slightly. If one coordinate of the gradient is positive, increasing that parameter increases the loss, so to reduce the loss, you move in the opposite direction.

Gradient descent means repeatedly applying:

$$\theta \leftarrow \theta - \eta \, \nabla_\theta L(\theta)$$

where $\eta > 0$ is the learning rate (step size). The minus sign moves you "downhill." This is a discrete version of moving along the negative gradient of the loss surface.

If $L(\theta)$ is smooth, then moving a small distance opposite the gradient direction always decreases L (to first order):

$$L(\theta - \eta \nabla L) \approx L(\theta) - \eta \|\nabla L\|^2 < L(\theta).$$

So each update reduces the loss until you reach a local minimum.

Computing ∇L over all data can be expensive. So you estimate it from a random subset (a mini-batch):

$$g = \frac{1}{B} \sum_{(x,y)\in\mathcal{B}} \nabla_\theta \ell(h_\theta(x), y)$$

and update

$$\theta \leftarrow \theta - \eta g.$$

The gradient estimate is noisy, but in expectation points the same way. Noise helps escape bad local minima.

Example 2.4 (C++ optimizations for low-latency trading)**.**

- C++ lets you decide when and where memory is allocated.
- Function calls require saving registers, jumping, returning. If the function body is small, that overhead dominates. Marking it inline lets the compiler paste it directly into the caller, removing the jump.
- constexpr tells the compiler to compute something at compile time rather than runtime.

C++ executes anything that must be fast, predictable, or close to hardware. For example, Feature Extraction for Streaming Market Data:

```
struct Tick { double bid, ask; int bid_sz, ask_sz; };
struct State { double mid, spread, imb; };

inline void update_features(const Tick& t, State& s) {
    s.mid    = 0.5 * (t.bid + t.ask);
    s.spread = t.ask - t.bid;
    s.imb    = double(t.bid_sz) / (t.bid_sz + t.ask_sz + 1e-9);
}
```

Python can then import it via pybind11 for high-level logic. Another example couple be to compute signal kernels, such as order book imbalance over time:

```
    inline double imbalance(double bid_sz, double ask_sz) {
    return (bid_sz - ask_sz) / (bid_sz + ask_sz + 1e-9);
}
```

Other examples of signals could be mid-price changes, spread changes, volume-weighted average price (VWAP), etc.

Example 2.5 (Interpositioning)**.** Function interpositioning is a Unix/Linux technique that lets you override or wrap existing library functions (like malloc, free, open, read, etc.) without changing the program's source code. Widely used for profiling, debugging, or performance instrumentation.

Interpositioning allows you to insert measurement or debugging code transparently.

For example, you can:

- Measure total bytes allocated.
- Time how long each call takes.
- Detect memory leaks (by logging unfreed pointers).
- Replace slow system calls with mocks during testing.

This works for any dynamically linked program — even proprietary binaries you can't recompile.

Bayesian Neural Networks for Time Series Forecasting 2024

1. Developed a Bayesian Neural Network (BNN) to predict stock prices with uncertainty quantification, using probabilistic forecasts to generate confidence intervals around price predictions.
2. Implemented variational inference and Monte Carlo dropout to approximate posterior weight distributions, allowing robust forecasts adaptable to dynamic market conditions.
3. Backtested BNN-based trading strategies, integrating a custom framework with transaction cost modeling and out-of-sample testing, using Sharpe ratio and max drawdown to evaluate performance.

Example 2.6 (Variational inference)**.** VI is a way of approximating a posterior distrubtion via a choice of a simpler family of distributions. In particular we choose a family of distributions $q(\theta; \phi)$ parameterized by ϕ (e.g. mean and variance of a Gaussian). We then optimize ϕ to make $q(\theta; \phi)$ close to the true posterior $p(\theta \mid \mathcal{D})$. We do so by minimizing the KL divergence (which is equivalent to maximizing the log likelihood) which becomes an optimizing problem.

Example 2.7 (Monte Carlo Dropout)**.** Traditional dropout randomly sets neurons to zero during training to prevent overfitting. Monte Carlo dropout extends this idea to inference time, allowing us to estimate uncertainty in predictions. In traditional dropout, dropout is turned off during inference, but in MC dropout, we keep it on and perform multiple forward passes through the network with different dropout masks. This results in a distribution of outputs for the same input, which we can use to estimate uncertainty. This replicates variational inference in a cheaper manner.

Example 2.8 (Bayesian Neural Networks)**.** A Bayesian neural network does not learn a single set of weights. It learns a distribution over weights, so that every forward pass draws slightly different weights, giving you a distribution over predictions. That predictive distribution directly gives you uncertainty, which you use to form confidence intervals or size trades.

In particular, Bayesian inference uses:

1. Prior: belief about weights before seeing data $p(\theta)$ Often a Gaussian prior.
2. Likelihood: how probable the data are under the model $p(y \mid x, \theta)$
3. Posterior: what we believe about the weights after seeing data $p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D}|\theta)\, p(\theta)}{p(\mathcal{D})}$

The posterior expresses "which weight configurations fit the data". In particular we assume a distributional form for the posterior approximation

The most common assumption is: $q(\theta) = \prod_i \mathcal{N}(\mu_i, \sigma_i^2)$ This is called a mean-field Gaussian posterior:

1. every weight has its own mean μ_i
2. every weight has its own variance σ_i^2
3. weights are assumed independent (this is not true, but it's easy to work with)

MAP estimation (maximum a posteriori) finds the most probable weights given the data. Define $\theta^\star = \arg\max_\theta p(\theta \mid \mathcal{D})$ and note

$$p(\theta \mid \mathcal{D}) \propto p(\mathcal{D} \mid \theta)\, p(\theta).$$

Taking the negative log (up to an additive constant) yields

$$-\log p(\theta \mid \mathcal{D}) = -\log p(\mathcal{D} \mid \theta) - \log p(\theta) + \text{const},$$

so MAP corresponds to minimizing the negative log-posterior, i.e. minimizing the negative log-likelihood plus the negative log-prior.

In practice, choosing a Gaussian prior $p(\theta) \propto \exp\big(-\frac{\lambda}{2}\|\theta\|^2\big)$ makes $-\log p(\theta)$ proportional to $\frac{\lambda}{2}\|\theta\|^2$, which is standard L_2 regularization (weight decay). Thus MAP training reduces to standard loss minimization with a regularizer.

MAP returns a single point estimate $\theta^\star$, the mode of the posterior, rather than a full posterior distribution.

Interactive Brokers API Integration for Algorithmic Trading 2024

- Integrated BNN model predictions with Interactive Brokers (IBKR) API for live/paper trading, using Python for data handling and C++ for low-latency order execution.

- Optimized trading system performance by implementing asynchronous data handling and error logging to ensure reliability in live market conditions.

Example 2.9 (C++ speedups). 1. Nonblocking I/O: Use asynchronous calls to avoid waiting for data, allowing other tasks to proceed.

2. Multithreading using synchronization techniques, i.e. bounded buffer, allows concurrent data processing and order execution.

3. Preallocated memory reduces overhead from head allocations

4. Cache friendly layout

Remark that this was largely pedagogical as we did not have sufficiently detailed market data to fully test latency improvements.

3 Interview day: Nov 19

3.1 Part 1

Standard market making games with dice and cards. Roll a 20 sided die 5 times and 24 sided die 5 times and make a market on the sum of the highest two rolls. How many contracts would you trade?

Ten cards arranged in a 2 by 5 grid, the game pays 4:1 on picking the column with the highest product. One row of cards is face up, the other is face down. Make a 6 wide market on the value of the game. How many contracts would you trade?

There were sidebets such as product is even, highest product is tied with second highest product, product is greater than 50, and you had to decide whether to buy or sell at the price given to you by the interviewer.

3.2 Part 2

Part 2 was programming with Jupyter notebook. There was a game in which you start with ten gold and there is a function which gives how many seeds one can buy with a certain amount of gold. It rewards buying seeds in bulk in the sense that you can get more seeds per gold if you buy seeds in larger quantities.

There are three types of seeds whose yields are random variables with unknown distributions. Every round, the game tells you how much yield you got from each seed type. Your objective is to get to 100 gold as fast as possible.

Your goal is to develop a trading bot that can minimize the average number of rounds to reach 100 gold over 1000 simulations.

The first thing I did was to visualize the distrubitons of the yields for each seed type. Then I did the naive thing of restricting to the seed with the highest expected yield and buying as many of those seeds as possible every round. Then I did a simple grid search, optimizing over how my starting strategy. In particular, my strategy was to start by buying n gold worth of B seed, and then scale up to spending the proportion m of my gold on B seeds every round. I optimized over n and m using grid search. This was quite a naive thing to do, as the interviewer later told me that the optimal strategy is to diversify between seed types. However, I did not have enough time to implement a more sophisticated strategy.

The interviewer remarked that someitmes the game is too complicated to solve in a short time, and that the best one can do is to implement a naive strategy that captures most of the value.

3.3 Part 3

This took place on a spreadsheet. You are doing an auction with 3 other people bidding on a treasure chest with gold coins in it. You always bid first. The other three people know the number of gold coins in the treasure chest within a margin of error of 5 coins. You can bid n times and if you refuse to bid, then you can not bid again. In some rounds, you can only bid once. In some rounds, there is a fee for bidding. The coins are worth 1:1 for the other people but they are worth 1.5:1 for you.

There are 6 different setups to the game, each time the interviewer shows you a different probability distribution of how the coins are distributed in the treasure chest. You have to decide how much to bid in each round.

I bombed this part as I didn't understand the subtlety of the fact that the other people know the number of coins within a margin of error.