Qishi Quiz 4

Yongjie Xin

July 1, 2019

Abstract

Results and answers.

Contents

1 Math

1.1 Q1

The events are as following:
E_1 : The first seat is taken before the 50th seat.
E_2 : The first seat is taken after the 50th seat.
Based on symmetry, the probability of $P(E_1) = P(E_2)$, which indicates, there are same number of combinations of E_1 and E_2.

In E_1, all the 50th seat will seat the 50th person. What we are interested actually is a sub category of E_1.
E_3: Given E_1, the first seat is taken before the 49th seat.
E_4: Given E_1, the first seat is taken after the 49th seat.
Again symmetry. So the probability that the last two could take their own seats are actually 1/4.

1.2 Q2

Say we have n persons, they are $A_i, i = 1, 2, ..., n$. Use $f(n)$ to indicate all the combinations of pairs that satisfies the requirements. Then the number of schedules are just $f(n) * n!$. The factorial of n is saying the order of the arrangement also matters. Here we focus on solving $f(n)$, the number of unique combinations.

The procedure starts like this: A_1 could first choose her two coworkers first;she has $(n-1)!/(2!(n-3)!)$ ways to choose. Now without loss of generality, assume A_2 and A_3 are chosen. We have $n-1$ girls left, to fill in (n-2) days. However, of all the $n-1$ girls, A_2 and A_3 are unique that :

$\boldsymbol{E}_1$. If they work on the same day, then the procedure becomes $n-3$ girls to fill in a $n-3$ days of schedule. The result is f(n-3).

$\boldsymbol{E}_2$. If they don't work on the same, they are only allowed to work once. In this case, they are like one person. So we could combine A_2 and A_3 into one person as X. The procedure becomes, $n-2$ girls to fill in $n-2$ schedule. The result is $f(n-2)$. However, since internally A_2 and A_3 are different. We have to multiply the result by 2. So the final result is $2f(n-2)$.

The events E_1 and E_2 are mutually exclusive. So we get :

$$f(n) = \binom{n-1}{2} * \big(f(n-3) + 2f(n-2)\big) \tag{1}$$

in which, $f(1) = f(2) = 0$, $f(3) = 1$. $f(4) = 3$.
So $f(5) = \binom{4}{2} * \big(f(2) + 2 * f(3)\big) = 12$
The total number of schedules are $f(5) * 5!$.
$f(7) = \binom{6}{2} * \big(f(4) + 2 * f(5)\big) = 15 * 27 = 405$.
The total number of schedules are $f(7) * 7!$.

1.3 Q4

It makes more sense to evaluate the correlation between the return of stocks. Correlation(lnS_1, lnS_2) = Correlation($I_1 = \int_0^{t_1} \sigma_1 dW_s$, $I_2 = \int_0^{t_2} \sigma_2 dW_s$). We have neglected the deterministic terms, since they have zero contribution in calculating the correlation. Ito's Isometry.

$$\rho = \frac{E[I_1 I_2] - E[I_1][I_2]}{\sqrt{Var[I-1]Var[I_2]}} \tag{2}$$

$$= \frac{\sigma_1 \sigma_2 t_1}{\sqrt{\sigma_1^2 t_1 \sigma_2^2 t_2}} \tag{3}$$

$$= \sqrt{\frac{t_1}{t_2}} \tag{4}$$

in which, for Ito integral, the $E[I] = 0$, and $Var[I] = \int_0^t \sigma^2 ds = \sigma^2 t$

1.4 Q5

The fundamental assumption of using Brownian motion, is $dW_t \approx \sqrt{t}$. Ito's lemma on any function $f(W_t)$ shows that

$$df(W_t) = f' dW_t + \frac{1}{2} f'' dW_t dW_t + \text{High order terms} \tag{5}$$

To have a martingale process, the coefficient of $dW_t dW_t = dt$ term has to be zero, which is basically to say no drift along time. We can see that the coefficient that $f'' = N(N-1)W_t^{N-2}$ is only zero, when N=0 or N=1.

1.5 Q6

Using BS, it is not hard to see that $S_t = S_0 e^{\int_0^t \mu_s ds - \frac{1}{2}\sigma^2 t + \int_0^t \sigma dW_s}$. S_t follows log-normal distribution and lnS_t is Gaussian, assuming $N(a, b^2)$. and $E[S_t] = e^{a+b^2/2}$.

Assume $Y_t = 1/S_t$, so $lnY_t = -lnS_t$ is Gaussian, $N(-a, b^2)$. So $E[Y_t] = e^{-a+b^2/2} = \frac{1}{E[S_t]}e^{b^2}$. Plug in the variance, we have $E[Y_t] = \frac{1}{E[S_t]}e^{\sigma^2 t}$

It also shows the Jensen's inequality, $1/x$ is a convex function, when $x > 0$.

1.6 Q7

For PDE, numerical stability is in general to say, the numerical error is decaying or at least stable during the time evolution. What is very useful theorem is Lax equivalence theorem. Because the convergence of PDE is very hard to prove. However, numerical stability is relatively easier to prove. For most cases, Von Neumann Stability analysis is enough, but not sufficient.

1.7 Q8

Say we start of a rate of m success out of n total throws, $m < 2n$ to make the success rate below 0.5. If we continue to throw in consecutively $n - 2m$ successful throws, we will have $n - m$ success out of $2(n - m)$ total throws, which is exactly 0.5 success rate.

1.8 Q9

Either Sharpe ratio or Sortino ratio. One focuses on the return with the risk taken. The other doesn't penalize the "good" volatility, whic focuses just the down side volatilities.

2 Programming

2.1 Q10

I have found a way to use the area of triangle should equal to sum of three triangle if the point faills within the triangle. My reference triangle. I don't like this approach. The reason is that it has irreducible numerical error. The "'equal' sign only works if happen to be no round off error.

The other approach is more intuitive, which is to decide whether the point is on the same side as the third point. The algo could be like

```
// whether p and A are on the same side of line B->C
bool sameside(p, A, B, C)
{
        // arrow here indicates vector, math vector. not c++ vector.
        side1 = CrossProduct( B->p, B->C );
        side2 = CrossProdcut( B->A, B->C);

        if side1*side2 >= 0
                return true;
        else
                false;
}
if(sameside(p, A, B, C) && sameside(p, B, A, C) && sameside(p, C, B, A))
        return true;
```

2.2 Q11

A palindrome is a word, number, phrase, or other sequence of characters which reads the same backward as forward, such as madam or racecar or the number 10801 (Quoted from Wikipedia).

So we could use two indices; index i starts from begining, each time increasing 1 step. The other index j starts from the end, each time decreasing 1 step. Then we check the values matches or not. If any of the steps does not match, return false. When $i \geq j$, and return true.

2.3 Q12

Two index, i and j.

```
int i = 0;
int j = 0;
int minLeft =MAX_INTEGER;
int maxRight = 0;
int maxLength = 0;
while( i < end && j < end )
{
    j++;
    if( A[i] > A[j] )
    {
        if(  j-i > maxLength )
        {
            minLeft = min( minLeft, i );
           maxRight = max( maxRight, j);
        }
        continue;
    }
    else
        i = j;
}

return (minLeft, maxRight);
```

2.4 Q13

This is a backtracking problem. A template I quite like about backtracking is this one. Backtracking. The sodoku problem also falls into the same pattern.

```
bool SolveSudoku(int grid[N][N])
{
    int row, col;

    // If there is no unassigned location,
    // we are done
    if (!FindUnassignedLocation(grid, row, col))
    return true; // success!

    // consider digits 1 to 9
    for (int num = 1; num <= 9; num++)
    {
        // if looks promising
        if (isSafe(grid, row, col, num))
        {
            // make tentative assignment
            grid[row][col] = num;

            // return, if success, yay!
            if (SolveSudoku(grid))
                return true;

            // failure, unmake & try again
            grid[row][col] = UNASSIGNED;
        }
    }
    return false; // this triggers backtracking
}

// findUnassignedLocation is to assign value to row and col when the cell is empty
// isSafe is to tell whether the filled in number is valid.
```

My Reference SudokuGeekforGeeks

2.5 Q14

Dynamic programming. The a[i][j] += max(a[i-1][j], a[i][j-1]) // also don't forget corners.