HW4

Haiming Wang

July 9, 2019

Problem Math1. Given a fifty-seat room where each seat is numbered, people entered the room in ordered. The first one is drunk and he will take a seat randomly. The rest people will take their own seat as long as it is not taken. Calculate the probability that last two people can take their own seats.

Solution 1. Consider seats #1and #49 and #50. There are two possible outcomes: E_1: Seat #1 is taken before #49 and #50. E_2: Seat #49 or #50 is taken before #1. If any passengers before 49 and 50 is taken seat #1, the last two people can take their own seats.

Consider any passenger n ($2 \leq n \leq 49$). If seat #1 is being taken before n, then all of the passenger after (and including) n will get their seats correctly. If seat #1 is not being taken before n, then passengers between 2 and n-1 already got their seats. Then n eventually becomes the new drunk man, who face the choice taken #1 or #49 or #50 with equal probability. So by mathematic induction: $P(E_1) = \frac{1}{3}$

Thus, the answer to this problem is $\frac{1}{3}$.

Problem Math2. There are 5 girls who need to guard a store from Mon-Fri. Calculate the number of possible arrangements given the constraint: (a) Everyone works twice a week. (b) Two girls guard the store in a single day and no two girls can work together twice.

Solution 2. For 5 girls there are $5! = 120$ kinds of permutation in total. In order to not get two girls work together twice. We can not have a circle that has length 2. As a result, there are $C_4^3 * 2 = 12$ number of combination. Consequently, we get $120 * 12 = 1440$ arragements.

Problem Math 3. Follow up, what if there are 7 girls and 7 days schedule to be set up, what is the number of possible arrangement?

Solution 3. We apply the same logic as in problem 2. There are 7! permutation for 7 people. The number of combination is $C_6^2 \times (3 + 4 + 4 * 3 * 2) = 15 * 31$. As a result, we have 2343600 arrangement.

Problem Math 4. (sellside) Given a stock, assume the implied volatility to time t1 is σ_1 and the implied volatility to time t2 is σ_2, calculate the correlation between St1 and St2.

Solution 4. The stock price should follow the geometric brownian motion. As a result, we can know the expression of S_{t1} and S_{t2}.

$$\begin{aligned} S_{t1} &= S_0 e^{(\mu-\frac{1}{2}\sigma_1^2)t+\sigma_1 W_{t1}} \\ S_{t2} &= S_0 e^{(\mu-\frac{1}{2}\sigma_2^2)t+\sigma_2 W_{t2}} \end{aligned}$$

Thus,

$$E[S_{t1}S_{t2}] = S_0^2 e^{\left(2\mu-\frac{1}{2}(\sigma_1^2+\sigma_2^2)\right)t} E\left[e^{\sigma_1 W_{t1}+\sigma_2 W_{t2}}\right]$$

where

$$E\left[e^{\sigma_1 W_{t1}+\sigma_2 W_{t2}}\right] \sim N\left[0, \sigma_1^2\, t_1 + \sigma_2^2\, t_2 + 2\sigma_1\sigma_2\, cov\left(W_{t1}, W_{t2}\right)\right].$$

$$E[S_{t1}]E[S_{t2}] = S_0^2\, e^{\left(2\mu-\frac{1}{2}(\sigma_1^2+\sigma_2^2)\right)t} E\left[e^{\sigma_1 W_{t1}}\right] E\left[e^{\sigma_2 W_{t2}}\right].$$

Then, we can calculate the correlation as

$$\rho = \frac{e^{\sigma_1\sigma_2 t_1 - 1}}{\sqrt{(e^{\sigma^2\, t_1}-1)(e^{\sigma^2\, t_2}-1)}}$$

Problem Math 5. (sellside) Given a Brownian motion W_t, when is W_t^N a martingale? Why W_t^3 is not a martingale?

Solution 5. (1) When is W_t^N a martingale
Apply Ito formula to W_t^N.

$$dW_t^N = \frac{1}{2}N(N-1)W_t^{N-2}\, dt + N\, W_t^{N-1}\, dW_t.$$

In order to be a martingale, we need the drift to be zero. That means N should be 0 or 1.

(2) Why W_t^3 is not a martingale
Plug $N = 3$ to the above diffusion process. We get

$$dW_t^3 = 3W_t\, dt + 3\, W_t^2\, dW_t.$$

We can see that, the drift term is not zero.

Problem Math 6. (sellside) Calculate the price of the option with payoff $\frac{1}{S_t}$ under traditional geometric Brownnian motion assumption.

Solution 6. ???

Problem Math 7. (sellside) What is the definition of stability in numerical PDE? What is the definition of implicit and explicit scheme?

Solution 7. Numerical stability
An algorithm for solving a linear evolutionary partial differential equation is stable if the total variation of the numerical solution at a fixed time remains bounded as the step size goes to zero. The Lax equivalence theorem states that an algorithm converges if it is consistent and stable (in this sense). Von Neumann stability analysis is a commonly used procedure for the stability analysis of finite difference schemes as applied to linear partial differential equations. These results do not hold for nonlinear PDEs, where a general, consistent definition of stability is complicated by many properties absent in linear equations.[**?**]

What is the definition of implicit and explicit scheme?
An implicit scheme need to solve equations, where explicit scheme can be calculated directly. The formal definition is explicit methods calculate the state of a system at a later time from the state of the system at the current time, while implicit methods find a solution by solving an equation involving both the current state of the system and the later one. It is clear that implicit methods require an extra computation (solving equation). [**?**]

Forward Euler is a common example of explicit scheme, while backward Euler is a common example of implicit scheme.

Problem Math 8. Consider the basketball shooting game, define success rate as number of successful shoots divided by number of total shoots. Assume the successful rate rising from below 0.5 to above 0.5, is there a moment which has exactly success rate 0.5.

Solution 8. Yes it must be. if jump across 0.5, then $r_t = T0/T1 < 0.5$, $r_{t+1} = (T0 + 1)/(T1 + 1) > 0.5$, No matter $T1$ is even or odd, we can see contradictions.

Problem Math 9. Given two strategies A and B, as well as the corresponding P&L of these strategies on each day. If one is going to be shut down, how to decide which one to shut?

Solution 9. On one hand we need to consider the investing goal (absolute return/ risk adjusted return) of the fund in order to select a fund that matches our goal to retain, if there is anyone missing our expectation then just close it. On another hand we need to consider exit cost, for some strategies, exiting positions is quite expensive, so we will try to retain those strategies if possible.

Problem Programming 10. Design an algorithm to check is a point is inside the triangle or not.

Solution 10. We can use the barycentric coordinate system to test whether a point is inside the triangle or not. We can express the new point p(x,y) using the vertices p1(x1,y1), p2(x2,y2), and p3(x3,y3).

$$\begin{aligned} x &= a * x1 + b * x2 + c * x3 \\ y &= a * y1 + b * y2 + c * y3 \\ a + b + c &= 1 \end{aligned}$$

Then, we can solve out a,b, and c and applying the following criteria:

p lies in T $\iff$ $0 \leq a, b, c \leq 1$.

Problem Programming 11. Design an algorithm to check palindrome sequence.

Solution 11. This is the algo to check palindrome sequence

```
def isPalindrome(s):
        """
        leetcode 125: valid palindrome
        """
        s = re.sub(r'[^a-zA-Z0-9]', "", s).lower()
        return s == s[::-1]
```

Problem Programming 12. Given an array, design an algorithm to return the longest decreasing sub-array.

Solution 12. Algo to return the longest decreasing sub-array

```
def longestDecreasingSubarray(arr):
    if not arr:
        return 0
    n = len(arr)
    length, maxLen = 1, 1
    for i in range(1, n):
        if arr[i] < arr[i-1]:
            length += 1
        else:
            length = 1
        maxLen = max(maxLen, length)
    return maxLen
```

Problem Programming 13. Implement the Sudoko algorithm.

Solution 13. The general idea to write a Suduko solver is to use backtracking. Like leetcode 37, we assume the empty cells are defined as ".".

```
class Solution:

    def solveSudoku(self, board):
       """
        :type board: List[List[str]]
        :rtype: void Do not return anything, modify board in-place instead.
        """
        self.board = board
        self.solve()

    def findUnassigned(self):
      """
   find the unassigned cell, return its column and row
```

```
"""
    for row in range(9):
        for col in range(9):
            if self.board[row][col] == ".":
                return row, col
    return -1, -1 # no unassigned

def solve(self):
    row, col = self.findUnassigned()
    #no unassigned position is found, puzzle solved
    if row == -1 and col == -1:
        return True
    for num in ["1","2","3","4","5","6","7","8","9"]:
        if self.isSafe(row, col, num): # check whether it is safe to place
            num in (row,col)
            self.board[row][col] = num
            if self.solve():
                return True
            self.board[row][col] = "."
    return False

def isSafe(self, row, col, num):
  # locate the sub-boxes of the grid
    boxrow = row - row%3
    boxcol = col - col%3
    if self.checkrow(row,num) and self.checkcol(col,num) and
        self.checkbox(boxrow, boxcol, num):
     # three criterion must meet
        return True
    return False

def checkrow(self, row, num):
    for col in range(9):
        if self.board[row][col] == num:
            return False
    return True

def checkcol(self, col, num):
    for row in range(9):
        if self.board[row][col] == num:
            return False
    return True

def checkbox(self, row, col, num):
    for r in range(row, row+3):
        for c in range(col, col+3):
            if self.board[r][c] == num:
```

```
                return False
        return True
```

Problem Programming 14. Consider a chess board with each cell has a value on it, you can only walk right or down, find the path from up-left to down-right which has largest value.

Solution 14.

```
// similar question can be found in Leetcode 64
def minPathSum( grid ):
    m, n = len(grid), len(grid[0])
    dp = [ [ grid[0][0] for j in range(n) ] for i in range(m) ]
    for i in range(1, m):
        dp[i][0] = dp[i-1][0] + grid[i][0]
    for j in range(1, n):
        dp[0][j] = dp[0][j-1] + grid[0][j]
    for i in range(1, m):
        for j in range(1, n):
            dp[i][j] = min(dp[i-1][j] + dp[i][j-1]) + grid[i][j]
    return dp[-1][-1]
```