QiShi Quiz 4

Xiangyu Zhang

June 2019

1 Math

Problem 1

Assume there are N people in total. We label all the people from 1 to N, and the seat is also labeled via its owner's number. We use P_n to denote the probability that the last 2 people sit on their own seats. We have $P_1 = 0, P_2 = \frac{1}{2}$. By conditioning on the label of seat the first sit on, we have

$$P_n = \frac{1}{n}(P_{n-1} + P_{n-2} + ... + P_2 + P_1)$$

so by induction we know

$$P_1 = 0, P_2 = \frac{1}{2}, P_n = \frac{1}{6} \text{ for } n \geq 3.$$

Problem 2

There are 5 people in total, and we are going to calculate the total number of different arrangements in 5 days. Assume we have already find 5 pairs , which satisfies the requirement. Then any permutation of these 5 pairs gives one valid arrangement.

So we only need to calculate the number of requirement satisfying pairs. We treat girl as node, and if two girl guard the store at the same day, we draw one edge between these two nodes. Then finding the number of requirement satisfying pairs is equivalent to find the number of all possible circle connecting these 5 nodes.

So the number of total arrangement is

$$\frac{1}{2}4! \cdot 5!$$

where the number of all possible circle is $\frac{1}{2}4!$, and the number of all possible permutation for requirement satisfying pairs is 5!.

Problem 3

There are 7 people in total, and we are going to calculate the total number of different arrangements in 7 days. Assume we have already find 7 pairs , which satisfies the requirement. Then any permutation of these 7 pairs gives one valid arrangement.

So we only need to calculate the number of requirement satisfying pairs. We treat girl as node, and if two girl guard the store at the same day, we draw one edge between these two nodes. Then finding the number of requirement satisfying pairs is equivalent to find the number of all possible circle connecting these 7 nodes.

To connect all nodes by circle, there are two possibility. One is a circle containing all 7 nodes, one is two circle containing 3 nodes and 4 nodes separately. So in total, the number of all possible circle connecting these 7 nodes is

$$\frac{1}{2}6! + \binom{7}{3} \cdot \frac{1}{2}2! \cdot \frac{1}{2}3!$$

So in total, the number of all possible arrangement is

$$7!(\frac{1}{2}6! + \binom{7}{3} \cdot \frac{1}{2}2! \cdot \frac{1}{2}3!)$$

Problem 4

Assume the stock is driven by a Geometric Brownian Motion,

$$\begin{aligned} S_{t_1} &= S_0 \exp((\mu - \frac{\sigma_1^2}{2})t_1 + \sigma_1 W_{t_1}) \\ S_{t_1} &= S_0 \exp((\mu - \frac{\sigma_2^2}{2})t_2 + \sigma_1 W_{t_2}) \end{aligned}$$

So we have

$$\begin{aligned} corr(S_{t_1}, S_{t_2}) &= corr(\exp(\sigma_1 W_{t_1}), \exp(\sigma_2 W_{t_2})) \\ &= \frac{cov(\exp(\sigma_1 W_{t_1}), \exp(\sigma_2 W_{t_2}))}{\sqrt{var(\exp(\sigma_1 W_{t_1}))}\sqrt{var(\exp(\sigma_2 W_{t_2}))}} \end{aligned}$$

Notice

$$\begin{aligned} var(\exp(\sigma W_t)) &= \mathbb{E}[\exp(2\sigma W_t)] - \mathbb{E}[\exp(\sigma W_t)]^2 \\ &= \exp(2\sigma^2 t) - \exp(\sigma^2 t) \end{aligned}$$

and

$$\begin{aligned}
cov(\exp(\sigma_1 W_{t_1}), \exp(\sigma_2 W_{t_2})) &= \mathbb{E}[\exp(\sigma_1 W_{t_1})\exp(\sigma_2 W_{t_2})] - \mathbb{E}[\exp(\sigma_1 W_{t_1})]\mathbb{E}[\exp(\sigma_2 W_{t_2})] \\
&= \mathbb{E}[\exp((\sigma_1+\sigma_2)W_{t_1})\exp(\sigma_2(W_{t_2} - W_{t_1})] - \mathbb{E}[\exp(\sigma_1 W_{t_1})]\mathbb{E}[\exp(\sigma_2 W_{t_2})] \\
&= \mathbb{E}[\exp((\sigma_1+\sigma_2)W_{t_1})]\mathbb{E}[\exp(\sigma_2(W_{t_2} - W_{t_1})] - \mathbb{E}[\exp(\sigma_1 W_{t_1})]\mathbb{E}[\exp(\sigma_2 W_{t_2})] \\
&= \exp(\frac{(\sigma_1+\sigma_2)^2}{2}t_1)\exp(\frac{\sigma_2^2}{2}(t_2-t_1)) - \exp(\frac{\sigma_1^2}{2}t_1 + \frac{\sigma_2^2}{2}t_2)
\end{aligned}$$

To conclude,

$$corr(S_{t_1}, S_{t_2}) = \frac{\exp(\sigma_1\sigma_2 t_1) - 1}{\sqrt{\exp(\sigma_1^2 t_1) - 1}\sqrt{\exp(\sigma_2^2 t_2) - 1}}$$

Problem 5

Only when $N = 0$ and $N = 1$, W_t^N is a martingale. When $N = 3$, we have

$$\begin{aligned}
\mathbb{E}[W_t^3 | W_s^3] &= \mathbb{E}[(W_s + W_t - W_s)^2 | W_s^3] \\
&= W_s^3 + 3W_s(t-s)
\end{aligned}$$

So $\{W_t^3\}_t$ is a not a martingale.

Problem 6

Under risk neutral measure Q, we have

$$\frac{dS_t}{S_t} = rdt + \sigma dW_t$$

So we have

$$S_t = S_0 \exp((r - \frac{\sigma^2}{2})t + \sigma W_t)$$

For the option price,

$$\begin{aligned}
\mathbb{E}[\frac{1}{S_T}] &= \mathbb{E}[\frac{1}{S_0}\exp(-(r - \frac{\sigma^2}{2})t - \sigma W_t)] \\
&= \frac{1}{S_0}\exp(\sigma^2 T - rT)
\end{aligned}$$

Problem 7

According to Wiki, `https://en.wikipedia.org/wiki/Numerical_stability`, we can define the absolute stability as following. Relative stability could be similarly defined.

Consider the problem to be solved by the numerical algorithm as a function f mapping the data x to the solution y. The result of the algorithm, say y^*, will usually deviate from the "true" solution y. The main causes of error are round-off error and truncation error. The forward error of the algorithm is the difference between the result and the solution; in this case, $\Delta y = y^* - y$. The backward error is the smallest Δx such that $f(x + \Delta x) = y^*$; in other words, the backward error tells us what problem the algorithm actually solved. The forward and backward error are related by the condition number: the forward error is at most as big in magnitude as the condition number multiplied by the magnitude of the backward error.

According to Wiki, `https://en.wikipedia.org/wiki/Explicit_and_implicit_methods`, the explicit and implicit scheme could be defined as following.

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. Mathematically, if $Y(t)$ is the current system state and $Y(t + \Delta t)$ is the state at the later time Δt is a small time step, then for an explicit method

$$Y(t + \Delta t) = F(Y(t))$$

while for an implicit method one solves an equation

$$G\Big(Y(t), Y(t + \Delta t)\Big) = 0$$

Problem 8

Yes, there exits a moment the success rate is exactly 0.5.

Problem 9

We should select the strategy with higher $P\&L$.

2 Programming

Problem 10

Python code:

```
// line function pass point x and y
def line(x, y):
  return lambda z: (x[0] - y[0]) * (z[1] - y[1]) - (x[1] - y[1]) *
    (z[0] - y[0])

def inside_triangle(x, y, z, w):
  f = line(x, y)
  t = f(z) * f(w)
  if t <= 0:
    return False

  f = line(x, z)
  t = f(y) * f(w)
  if t <= 0:
    return False

  f = line(y, z)
  t = f(x) * f(w)
  if t <= 0:
    return False

  return True
```

Problem 11

Python code:

```
def valid_palindrome(s):
  if not s:
    return True

  i, j = 0, len(s) - 1
  while i < j:
    if s[i] == s[j]:
      i += 1
      j -= 1
    else:
      return False

  return True
```

Problem 12

Python code:

```
def longest_decrease_subarray(l):
  if not l:
    return 0

  length = []
  for ind, val in enumerate(l):
    if ind == 0:
      length.append(1)
      continue

    if val <= l[ind - 1]:
      length.append(length[-1] + 1)
    else:
      length.append(0)

  return max(length)
```

Problem 13

Python code:

```
from collections import defaultdict
class Solution:
    """
    @param board: the sudoku puzzle
    @return: nothing
    """

    def __init__(self, n=9):
        self.n = n
        self.n_sqrt = int(n ** 0.5)
        self.rows = [defaultdict(int) for _ in range(self.n)]
        self.cols = [defaultdict(int) for _ in range(self.n)]
        self.boxes = [defaultdict(int) for _ in range(self.n)]
        self.success= False

    def _get_box_id(self, row, col):
        return int((row // self.n_sqrt) * self.n_sqrt + col // self
    .n_sqrt)

    def _fill(self, board, row, col, val):
        if val != 0:
            self.rows[row][val] = 1
            self.cols[col][val] = 1
            self.boxes[self._get_box_id(row, col)][val] = 1
        else:
            del self.rows[row][board[row][col]]
            del self.cols[col][board[row][col]]
            del self.boxes[self._get_box_id(row, col)][board[row][
    col]]
        board[row][col] = val

    def _valid(self, board, row, col, val):
        return (val not in self.rows[row]) and \
               (val not in self.cols[col]) and \
               (val not in self.boxes[self._get_box_id(row, col)])

    def _next_index(self, row, col):
        if col == self.n - 1:
            row += 1
            col = 0
        else:
            col += 1
        return row, col

    def _dfs(self, board, row, col):
        if row >= self.n:
            self.success = True
            return

        new_row, new_col = self._next_index(row, col)

        if board[row][col] == 0:
            for val in range(1, self.n + 1):
                if self._valid(board, row, col, val):
                    self._fill(board, row, col, val)
                    self._dfs(board, new_row, new_col)
                    if self.success:
                        return
                    self._fill(board, row, col, 0)
        else:
            self._dfs(board, new_row, new_col)

    def solveSudoku(self, board):
        for row in range(self.n):
            for col in range(self.n):
                if board[row][col] != 0:
                    self._fill(board, row, col, board[row][col])
        self._dfs(board, 0, 0)
```

Problem 14

Python code

```
def max_val_board(board):
  row, col = len(board), len(board[1])
  val = [[0]*col for _ in range(row)]

  for r in range(row - 1, -1, -1):
    for c in range(col - 1, -1, -1):
      current = board[r][c]
      move_right = 0 if c == col - 1 else val[r][c + 1]
      move_down = 0 if r == row - 1 else val[r + 1][c]
      val[r][c] = current + max(move_down, move_right)
      print(r, c, val[r][c])

  return val[0][0]
```

3 Reference

[1] http://faculty.wwu.edu/sarkara/dirac.pdf

[2] https://math.stackexchange.com/questions/13959/if-a-1-meter-rope-is-cut-at-two-uniformly-randomly-chosen-points-what-is-the-av