Qishi Quiz 4

Group F

Q1.

1/3

a. Consider seats 1/49/50. If the drunk man take seat 1, then 100%, the last two persons will have their seat. Likewise, if the drunk man take seat 49/50, probability is 0. So in total, the probability is 1/3
b. Consider the drunk man takes seat 2-48.
   Take 48: Then 2-47 people have their own seat. For the 48th people, he has 3 choices: 1/49/50, so the probability is 1/3 again.
   Take 47: 2-46 people have their own seat. For the 47th person. If he takes seat 48, then 48th person have the same situation as previous p=1/3. If he takes 1/49/50, then still 1/3.
   …
   So for any man between 2 and 48, the probability is still 1/3.

Q2.

Let's name the girls using number 1-5. Since there are no same combinations for the 5 days, we can figure out how many kind of combinations there are in a week and then assign the combinations to 5 days in the week.

Now without loss of generality, let's assume that 1 chooses to partner with girl 2 and 3. There are $C\binom{4}{2}$ =4*3/2 = 6 different kind of choices. It is easy to see that girl 2 and 3 cannot work as partners now, because then 4 and 5 would have to be partners for 2 days in a week, which is not allowed. So girl 2 only have 2 choices for her partners and girl 3 would have to choose the remaining girl.

There are $C\binom{4}{2} * 2 = 12$ combinations in total. And for each combination, we have 5! ways to assign the combination, so in total we have 12*120 = 1440 different ways.

Q3.

The method is similar as in Q2, now let's name the girls using number 1-7 and we can figure out how many kind of combinations there are in a week and then assign the combinations to 5 days in the week.

Now without loss of generality, let's assume that 1 chooses to partner with girl 2 and 3. There are $C\binom{6}{2} = 6 * \frac{5}{2} = 15$ different kind of choices.

Now for girl 2 there are two different scenarios:

1. She chooses to work with girl 3 and now girl 4,5,6,7 work with each other like a closed circle, given that girl 1,2,3 are fully occupied. It is trivial to see that there are only 3 kinds of combinations for girl 4,5,6,7 in a group.
2. She chooses to work with a girl from girl 4,5,6,7 and without a loss of generality let's say girl 4. Now there are also two different scenarios for girl 3:
    a. She chooses to work with girl 4, the same girl who was chosen by girl 2. Now there are only 1 combination for the remaining 3 girls (5,6,7).
    b. She chooses to work with one of the girl 5,6,7 say girl 5, now there are 2 ways for girl 4 to choose from, she could either choose 6 and 7 and then there is only one combination left for the remaining girls.

So there are 3+(4*(1+3*2)) = 31 combinations depends on girl 2's choice.

In total, there are 15*31*7! = 2343600 different ways.

Q4.

S(t1)=s(0)*exp[(r-sigma_1^2/2)*t1+sigma_1*W(t1)]

S(t2)=s(0)*exp[(r-sigma_2^2/2)*t2+sigma_2*W(t2)]

Cor(St1,St2)=Cov(St1,St2)/[sd(St1)*sd(St2)]

Cov(St1,St2)

=E(St1*St2)-E(St1)*E(St2)

=s(0)^2*exp[(r-sigma_1^2/2)*t1+(r-sigma_2^2/2)*t2+1/2*(sigma_1+sigma_2)^2*t1+1/2*(sigma_2)^2*(t2-t1)]-s(0)^2*exp[(r-sigma_1^2/2)*t1+[(r-sigma_2^2/2)*t2+1/2*sigma_1^2*t1+1/2*sigma_2^2*t2]

=s(0)^2*exp[(r-sigma_1^2/2)*t1+(r-sigma_2^2/2)*t2+1/2*sigma_2^2*t2+1/2*sigma_1^2*t1]

*{exp[2*sigma_1*sigma_2*t1-1]}

Sd(St1)=sqrt(E(St1*St1)-E(St1)^2)=S(0)*exp(r*t1)*sqrt[exp(sigma_1^2*t1)-1]

Sd(St2)=sqrt(E(St2*St2)-E(St2)^2)= S(0)*exp(r*t2)*sqrt[exp(sigma_2^2*t2)-1]

Cov(St1,St2)=[exp(sigma_1*sigma_2*t1)-1]/sqrt{ [exp(sigma_1^2*t1)-1]*[exp(sigma_2^2*t2)-1]}

Q5.

Since W_t is normal with mean 0 and variance t, its moment generating function is given by

$$\psi(s) = E\left[e^{sW_t}\right] = \int_{-\infty}^{\infty} e^{sw} \frac{1}{\sqrt{2\pi t}} e^{-\frac{w^2}{2t}} dw = e^{\frac{ts^2}{2}} \frac{1}{\sqrt{2\pi t}} \int_{-\infty}^{\infty} e^{-\frac{(w-ts)^2}{2t}} dw = e^{\frac{ts^2}{2}}$$

$$E[W_t^n] = \psi^{(n)}(s)|_{s=0}, E[W_t] = 0, E[W_t^2] = t, E[W_t^3] = 0 \ldots$$

So we can find, if n is odd, $E[W_t^n] = 0$. If n is even, $E[W_t^n] \neq 0$

$$E[W_t^n \mid W_u, 0 \le u \le s] = E[\{W_s + W_t - W_s\}^n \mid W_u, 0 \le u \le s]$$

If $n \ge 2, \{W_s + W_t - W_s\}^n$ will have at least one even item that

$$i = 2k \le n, E[(W_t - W_s)^i \mid W_u, 0 \le u \le s] = E[{W_{t-s}}^i \mid W_u, 0 \le u \le s] = E[{W_{t-s}}^i] \neq 0$$

So for $n \ge 2, E[W_t^n \mid W_u, 0 \le u \le s] = E[\{W_s + W_t - W_s\}^n \mid W_u, 0 \le u \le s] = W_s^n + M$

M is a non-zero item that is related to t and s.

Therefore, only when n=1, W_t is a martingale. When n=3,

$$E[W_t^3 \mid W_u, 0 \le u \le s] = E[\{W_s + W_t - W_s\}^3 \mid W_u, 0 \le u \le s]$$
$$= W_s^3 + 3W_s^2 E[W_t - W_s \mid W_u, 0 \le u \le s] + 3W_s E[(W_t - W_s)^2 \mid W_u, 0 \le u \le s]$$
$$+ E[(W_t - W_s)^3 \mid W_u, 0 \le u \le s]$$
$$= W_s^3 + 3W_s^2 E[W_{t-s}] + 3W_s E[{W_{t-s}}^2] + E[{W_{t-s}}^3]$$
$$= W_s^3 + 3W_s(t-s)$$

So W_t^3 is not a martingale. $E[W_t^3 - 3W_t t \mid W_u, 0 \le u \le s] = W_s^3 - 3W_s s$

Since W_t is a martingale, so $W_t^3 - 3W_t t$ is a martingale.

Q6.

Under traditional geometric Brownian motion assumption, $S_t = S_0 * e^{(r-q-\sigma^2/2)*T+\sigma*T*x}$, where x follows standard normal distribution. We can calculate the price of the option using the discount of the expected payoff:

$$V = \frac{e^{-rT}}{\sqrt{2\pi}} \times \int_{-\infty}^{\infty} e^{-x^2/2} \cdot \left(\frac{1}{S0} \cdot e^{-\left(r-q-\frac{\sigma^2}{2}\right)T-\sigma\sqrt{T}x}\right) dx$$

$$= \frac{e^{-(q-\sigma^2)T}}{S0 \cdot \sqrt{2\pi}} \times \int_{-\infty}^{\infty} e^{-(x+\sigma\sqrt{T})^2/2}\, dx$$

$$= \frac{e^{-(q-\sigma^2)T}}{S0}$$

Q7

Stability means that the error caused by a small perturbation in the numerical solution remains bound. A more formal definition is that:

Let T(V) = F be a numerical scheme. Let Vi be two solutions, i.e. T(Vi) = Fi, for i = 1, 2. We shall say that the scheme is stable if for each ε > 0 there exist δ(ε) such that |F1-F2| <δ implies |V1-V2| <ε. (An Introduction to Partial Differential Equations By Yehuda Pinchover, Jacob Rubinstein)

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+Δt) 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

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

to find $Y(t+\Delta t)$

Q8.

$$\text{Successful rate } = \frac{\#\text{ made shots}}{\#\text{ total shots}} = \frac{\#\text{ made shots}}{\#\text{ made shots} + \#\text{ missed shots}}$$

Successful rate < 0.5 ⇔ # made shots < # missed shots

Successful rate = 0.5 ⇔ # made shots = # missed shots

Successful rate > 0.5 ⇔ # made shots > # missed shots

Therefore, to change from Successful rate < 0.5 to Successful rate > 0.5 is equivalent to changing from # made shots < # missed shots to # made shots > # missed shots. Since # made shots and # missed shots can increase by 1 at a time, there must be a moment when # made shots = # missed shots, i.e. successful rate = 0.5

Q9.

If there is no other requirement and one want to see general risk and return, we would use Sharpe Ratio as measurement.

If the strategy is conservative and want to minimize risk, then Max Drawdown and Sortino Ratio could be used.

If the strategy only cares about return, then Total Return is the best way.

If the strategy is compared to benchmark, then information ratio will be a good measure.

If the strategy want to see how skillful the trader is, excluding the fact of the market, the alpha might be the measurement relative to beta.

Reference:

Sortino Ratio: https://en.wikipedia.org/wiki/Sortino_ratio

Information Ratio: https://en.wikipedia.org/wiki/Information_ratio

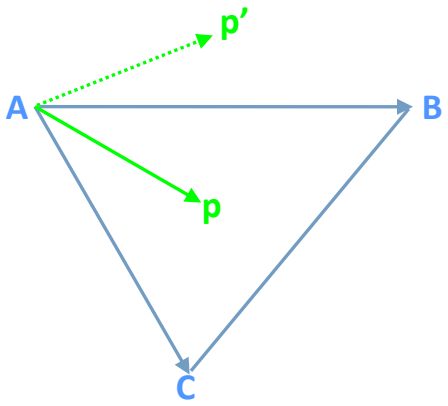
Q10

First we need the interface for vectors:

```
class VECTOR3D
{
 public:
long double x, y, z;

VECTOR3D(long double xx = 0.0, long double yy = 0.0, long double zz = 0.0) : x(xx), y(yy), z(zz)    // make a 3d vector
{
}
        VECTOR3D operator-(const VECTOR3D& vec)          // subtract two vectors
       {
        return VECTOR3D(x - vec.x, y - vec.y, z - vec.z);
        }
       VECTOR3D operator&(long double scalar)        // product of a vector and a scalar
       {
```

```
        return VECTOR3D(x*scalar, y*scalar, z*scalar);
       }
        long double operator*(const VECTOR3D& vec)        // dot product of two vectors
        {
        return x*vec.x + y*vec.y + z*vec.z;
       }
       VECTOR3D operator^(const VECTOR3D& vec)       // cross product of two vectors
       {
return VECTOR3D(y*vec.z - z*vec.y,  z*vec.x - x*vec.z, x*vec.y-y*vec.x);
        }
};
```



If point p is inside the triangle ABC, then the point is on the same side of AB as C and is also on the same side of BC as A and on the same side of CA as B, and vice versa.

To check that, we define vector AB, AC, and Ap(Ap'). If point p is on the same side of AB as C, then the cross product of AB and AC and the cross product of AB and Ap, should point to the same direction(out of plane here). f point p' is on the other side of AB as C, then the cross product of AB and AC and the cross product of AB and Ap', should point to the opposite direction. It can be judged by the dot product of the tow cross product vectors.

Above procedure should be repeated for the other two vertexes B and C. Here is the code:

```
bool SameSide( VECTOR3D& p1, VECTOR3D& p2, VECTOR3D& A, VECTOR3D& B)
{
VECTOR3D vec = B-A;
VECTOR3D vecp1 = p1-A;
VECTOR3D vecp2 = p2-A;

if ( ((vec^vecp1)*(vec^vecp2)) > = 0) return true;
else return false;
}
bool PointInTriangle ( VECTOR3D& p, VECTOR3D& A, VECTOR3D& B, VECTOR3D& C)
{
if (SameSide(p,A,B,C) && SameSide(p,B,A,C) && SameSide(p,C,A,B)) return true;
        else return false;
}
```

There is another method called Barycentric Technique.

The three vertex points of the triangle define a plane in space. Pick one of the points and we can consider all other locations on the plane as relative to that point. Let A be our origin on the plane. Now what we need are basis vectors so we can give coordinate values to all the locations on the plane. We'll pick the two edges of the triangle that touch A, AC and AB. Now we can get to any point on the plane just by starting at A and walking some distance along AC and then from there walking some more in the direction AB.

With that in mind we can now describe any point on the plane as

$P = A + u * AC + v * AB$

Notice that the condition that p is inside the triangle is that $u + v < 1$. Here is the code

```
bool PointInTriangle ( VECTOR3D& p, VECTOR3D& A, VECTOR3D& B, VECTOR3D& C)
```

```
{
VECTOR3D v0 = C-A;
VECTOR3D v1 = B-A;
VECTOR3D v2 = p-A;

// Compute barycentric coordinates
double invDenom = 1 / ((v0*v0) * (v1*v1) - (v0*v1) * (v0*v1))
double u = ((v1*v1) * (v0*v2) - (v0*v1) * (v1*v2)) * invDenom
double v = ((v0*v0) * (v1*v2) - (v0*v1) * (v0*v2)) * invDenom

// Check if point is in triangle
return (u >= 0) && (v >= 0) && (u + v < 1)

}
```

Q11.

```
// Use begin and end iterators to check if the sequence is palindrome
// C++ 11
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int main()
{
   vector<int> v = {1,2,3,4,5,4,3,2,1};
   int len = v.size()/2;
   vector<int>::iterator it1 = v.begin();
   vector<int>::iterator it2 = v.end()-1;
```

```
   for(int i=0; i<len; i++)
   {
      if(*it1 == *it2)
      {
         it1++;
         it2--;
      }
      else
      {
         cout << "Not palindrome sequence" << endl;
         break;
      }
   }
   cout << "Finished" << endl;
   return 0;
}
```

Q12.

Case 1: Find the longest (non-successive) decreasing sub-array

This problem can be solved by dynamic programming. Let's call the given array P, and let me define another vector L, where L[i] is the longest decreasing subsequence that ends with P[i] for $0 \le i \le n-1$. We know that L[0] = [P[0]] by this definition, and we can find L[i] from i=1 to i=n-1 step by step according to the following strategy:

For every L[i], look at all the L[j] that satisfy $j < i$ and $P[j] > P[i]$, find the longest such L[j] and append P[i] to the end of it. If there is no such L[j], then L[i] = [P[i]].

Repeat this process for i = 1, ……, n-1, and find all the L[i], the longest L[i] is the longest decreasing subsequence that we are looking for.

For example: P = [10, 3, 9, 7, 11], with n=5 here

L[0] = [10]

L[1] = [10, 3]

L[2] = [10, 9]

L[3] = [10, 9, 7]

L[4] = [11]

Looking at the 5 subsequences, we see that L[3] is the longest one that we are after.

There are two levels of loops, one from i = 1 to n, the other from 1 to i − 1, and finding the longest L[i] is also O(n), so the running time of this dynamic programming algorithm is $O(n^2) + O(n) = O(n^2)$

Python code:

```
import numpy as np
def LDS(P) :
   L = {0:[P[0]]}
   for i in range(1, len(P)):
      L[i] = []
      for j in range(i):
         if (P[i] < P[j]) & (len(L[i]) < len(L[j])):
            L[i] = L[j]
      L[i] = np.append(L[i], P[i])
   return max(L.values(), key=len)
```

Test case:

```
>>> LDS([10, 3, 9, 7, 11])
array([10,  9,  7])
>>> LDS([5.6, 3.2, 11.4, 8.6, 12.5, 6.1, 20.8, 0.9])
```

array([11.4, 8.6, 6.1, 0.9])

Case 2: Find the longest (successive) decreasing sub-array

```
// C++ 11
#include <iostream>
#include <vector>
using namespace std;

void getSub(vector<int>& ivec)
{
   int count = 1;
   int max = 0;
   int mark = 0;
   int len = (int)ivec.size();
   for (int i=1; i<len;i++)
   {
      if (ivec[i-1] > ivec[i]) { count++; }
      else
      {
         if (count>max)
         {
            max = count;
            mark = i;
         }
         count = 0;
      }
   }
   for (int j=mark-max; j<mark; j++)
   {
```

```
        cout << ivec[j] << " ";
    }
}

int main ()
{
    vector<int> ivec = {5,4,3,2,4,3,2,1};
    getSub(ivec);
    return 0;
}
```

Q13.

The most straight forward way is backtrace. Code:

```
bool check(vector<vector<char>> &board, int i, int j, char val)
{
    int row = i - i%3, column = j - j%3;
    for(int x=0; x<9; x++) if(board[x][j] == val) return false;
    for(int y=0; y<9; y++) if(board[i][y] == val) return false;
    for(int x=0; x<3; x++)
    for(int y=0; y<3; y++)
        if(board[row+x][column+y] == val) return false;
    return true;
}
bool solveSudoku(vector<vector<char>> &board, int i, int j)
{
    if(i==9) return true;
    if(j==9) return solveSudoku(board, i+1, 0);
    if(board[i][j] != '.') return solveSudoku(board, i, j+1);
```

```
    for(char c='1'; c<='9'; c++)
    {
        if(check(board, i, j, c))
        {
            board[i][j] = c;
            if(solveSudoku(board, i, j+1)) return true;
            board[i][j] = '.';
        }
    }

    return false;
}
```

Q14.

This problem can be solved by dynamic programming. Let's say the input is an n x n Table, for a chess board n=8. And let's define another n x n table OPT, where OPT_{ij} [0] stands for the maximized score from $Table_{11}$ to $Table_{nn}$, and OPT_{ij} [1] saves the path associated with the maximized score.

By definition, $OPT_{11}[0] = Table_{11}$, and $OPT_{i1}[0] = Table_{11} + Table_{21} + \ldots + Table_{i1}$, and $OPT_{1j}[0] = Table_{11} + Table_{12} + \ldots + Table_{1j}$. For the rest i, j, $OPT_{i1}[0] = \max(OPT_{i-1,\,j}[0] , OPT_{i,j-1}[0]) + Table_{ij}$. And $OPT_{nn}[0]$ is the largest value from up-left to down-right , and $OPT_{nn}[1]$ is the path that we are looking for.

The run time is the size of the table $O(n^2) = O(k)$, where $k=n^2$ is the total number of elements in the table. K=64 for a chess board.

Python code:

```
import numpy as np
def chess(table):
```

```
  if table.shape[0] != table.shape[1]:
      print ('Error: the input table should be a square matrix')
      pass
  else:
      n = table.shape[0]
      opt = {}
      opt[(0,0)] = (table[0,0], [table[0,0]])
      for i in range(1, n):
          opt[(0, i)] = (opt[(0, i-1)][0] + table[0, i], np.append(opt[(0, i-1)][1], table[0, i]))
          opt[(i, 0)] = (opt[(i-1, 0)][0] + table[i, 0], np.append(opt[(i-1, 0)][1], table[i, 0]))
      for i in range(1, n):
          for j in range(1, n):
              if opt[(i-1, j)][0] >= opt[i, j-1][0]:
                  opt[(i,j)] = (opt[(i-1, j)][0] + table[(i,j)], np.append(opt[(i-1, j)][1], table[i,j]))
              else:
                  opt[(i,j)] = (opt[(i, j-1)][0] + table[(i,j)], np.append(opt[(i, j-1)][1], table[i,j]))
      return opt[n-1, n-1]
```

Test case:

```
>>> x = np.arange(16).reshape((4,4))
>>> x
array([[ 0,  1,  2,  3],
           [ 4,  5,  6,  7],
           [ 8,  9, 10, 11],
           [12, 13, 14, 15]])
>>> chess(x)
(66, array([ 0,  4,  8, 12, 13, 14, 15]))
```

```
>>> y = np.diag(np.diag(x))
>>> y
array([[ 0,  0,  0,  0],
       [ 0,  5,  0,  0],
       [ 0,  0, 10,  0],
       [ 0,  0,  0, 15]])
>>> chess(y)
(30, array([ 0,  0,  5,  0, 10,  0, 15]))

>>> z = np.arange(12).reshape((3,4))
>>> z
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> chess(z)
Error: the input table should be a square matrix

>>> np.random.seed(123)
>>> table = np.random.rand(8,8)
>>> table
array([[ 0.69646919,  0.28613933,  0.22685145,  0.55131477,  0.71946897, 0.42310646,  0.9807642 ,  0.68482974],
       [ 0.4809319 ,  0.39211752,  0.34317802,  0.72904971,  0.43857224, 0.0596779 ,  0.39804426,  0.73799541],
       [ 0.18249173,  0.17545176,  0.53155137,  0.53182759,  0.63440096, 0.84943179,  0.72445532,  0.61102351],
       [ 0.72244338,  0.32295891,  0.36178866,  0.22826323,  0.29371405, 0.63097612,  0.09210494,  0.43370117],
       [ 0.43086276,  0.4936851 ,  0.42583029,  0.31226122,  0.42635131, 0.89338916,  0.94416002,  0.50183668],
       [ 0.62395295,  0.1156184 ,  0.31728548,  0.41482621,  0.86630916, 0.25045537,  0.48303426,  0.98555979],
       [ 0.51948512,  0.61289453,  0.12062867,  0.8263408 ,  0.60306013, 0.54506801,  0.34276383,  0.30412079],
       [ 0.41702221,  0.68130077,  0.87545684,  0.51042234,  0.66931378, 0.58593655,  0.6249035 ,  0.67468905]])
>>> chess(table)
```

(9.5921382752228137, array([0.69646919, 0.4809319, 0.39211752, 0.34317802,
0.72904971,0.53182759, 0.63440096, 0.84943179, 0.63097612, 0.89338916, 0.94416002, 0.50183668,
0.98555979, 0.30412079, 0.67468905]))