Q1:  
3/4  
Two points will always in a semicircle.   
Nearest: If two points lie on the same, then the p=1  
Farthest: If two points have 180 angle, then p=0.5  
Along all the points, the probability are linearly decreasing, so the expected value shoud be (1+1/2)/2=3/4.  
  
  
Q2:  
10  
Set the vertex to be A, and the opposite one to be B, set three vertex that directly connet to A to be A1, the rest to be B1.  
Use E(A) as notation for starting from A, the expected steps taken to arrive at B.   
  
E(A)=1+E(A1)  
E(A1)=1/3\*(1+E(A))+2/3(1+E(B1))  
E(B1)=1/3+2/3\*(E(A1)+1)  
  
==> E(A1)=9 E(A)=10 E(B1)=7

Q3.

Assuming there are both 26 black and red cards, obviously we would play this game, because the worst outcome would be 0 given we can always choose to stop at the end.

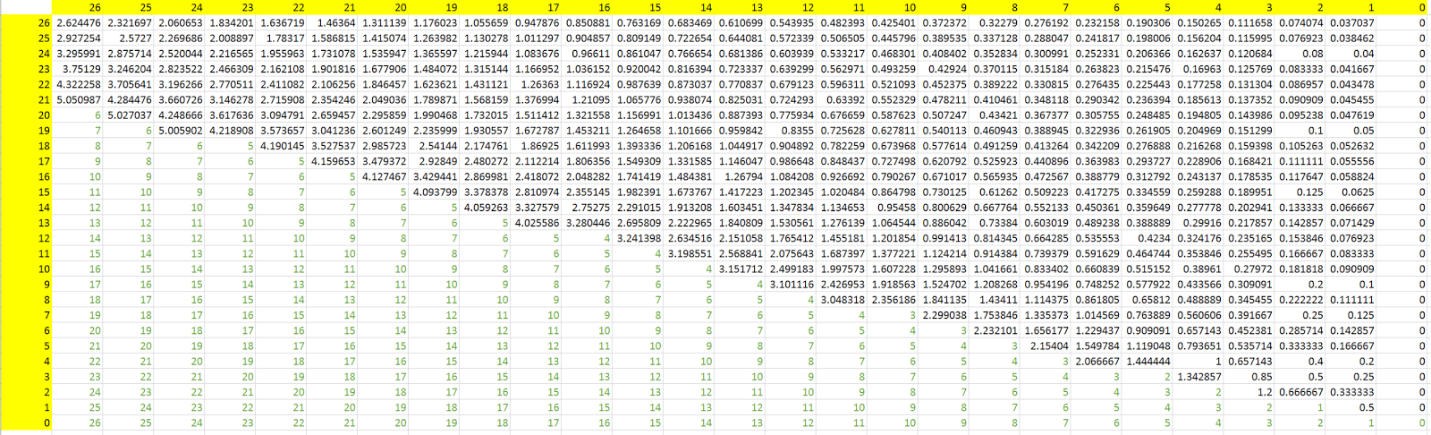
To calculate the expected payoff of this game, we can treat it like American Option pricing and price the game backwards:

We first start off at only one card left. If the card left is Black, we will have keep playing for sure and the expected payoff is 0 since all the cards is revealed. If the card left is Red, we will stop and keep $1 as the payoff.

Now let’s get to the position where there are two cards left. Using same logic we will keep playing if both left are black and get payoff 0, or stop and keep $2 as the payoff when both left are red cards. Now, if there is one each. then we need to consider two scenarios: 1. draw black, get $1 and stop. 2. draw red, get $-1 and keep playing until the end, which end up with $0 additional payoff. So the expected payoff is 0.5\*1+0.5\*0 = 0.5, you will keep play.

Using this recursive method we can calculate all the scenarios all the way back to the start. We finally get the expected payoff of the game: 2.62

Please see the excel table attached for the calculation details:



Q4. Given a coin with probability p of landing on heads after a flip, what is the probability that the number of heads will ever equal the number of tails assuming an infinite number of flips?

Answer:

This is a standard random walk problem. Consider the starting point to be 0, getting a head to be moving one step to the right, and getting a tail to be moving one step to the left. Then the probably to the right is p, and the probability of moving to the left is 1-p, and the probability of the number of heads ever equals the number of tails is essentially the probability that moving back to the origin after the first step

(a) The direction of positive and negative is arbitrary here, so the probability of p and 1-p is symmetric. Let ‘s consider the case when p >= 0.5, and the situation when p < 0.5 would be similar.

Let’s call:

A = probability of ever moving one step to the right of where you are

B = probability of moving back to the current position from its left

For A, to ever move one step to the right, one can move one step to its right directly, or move to the left and then moving back, or even back and forth like this many times, so

A = p + Bp + B2p + B3p + ……

When we flip the coin for infinite times,

A = p / (1-B)

Note that A is a probability distribution, so p/(1-B) <= 1, so we know B <= 1-p here

Also for B, the probability of moving back to the current position from the left is equal to the probability of moving one step to the left, times the probability of moving back to the current position from either direction is

B = (1-p) \* A

Solve for B, and we have:

B/(1-p) = p/(1-B), B = p or 1-p

When p > 0.5, B <= 1-p < 0.5, so B = p

When p =0.5, B = 0.5, also B = p

So if we start from origin, the probability of moving back to origin from its left is p. Because of the symmetry, we know that the probability of moving back to the origin from its right is also B = p. So the probability of moving back to the origin is 2p.

(b) When p < 0.5, repeat the above process by replacing p with 1-p. So the probability of moving back to the origin is 2(1-p).

Combine the results of the two cases, and the number of heads will ever equal the total number of tails p = 2min(p, (1-p))

Q5.

The probability density function of the shorter lived life bulbs is f1(x)=1/100 \* e-x/100.

The probability density function of the longer lived life bulbs is f2(x)=1/200 \* e-x/200.

The survival function is 1 minus the integral of the density function (CDF). S1(x)=e-x/100, and S2(x)=e-x/200.

We can get the survival function for the entire group: (e-x/100)5 \* (e-x/200)5.

The density function for the group is the derivative of 1 minus the survival function (CDF)  e-3x/40.

The expectation for the group to have first failure is the integral from 0 to infinity of x times the density function for the group = 40/3 hours.

Q6:  
Correct.  
100 year= 60\*60\*24\*365\*100 =>10^10 (seconds)  
consecutive heads=(1/2)^100=(2^10)^10=1024^10 =>10^-30  
  
P(tossing 100 consecutive heads)=P(  (100 consecutive heads start from 1st second) n (100 consecutive heads start from 2nd second) n ....)<=P (100 consecutive heads) \* total time=>10^-30\*10^10=10^-20<0.01%  
  
So the statement is correct.

Q7.

Cutting a stick into N pieces with N-1 cut points is equivalent to cutting a same-length circle into N pieces with N cut points, since the first cut is arbitrary. To form a polygon, we need the length of any piece should be less than sum of other pieces. Assume the length of the stick and the circumference of the circle is 1, if one piece is more than 0.5, then they cannot form a polygon.

Define event Ei as from first cut at point i the other points are in the clockwise semicircle.

P(Ei) = , for i = 1...N.

P(cannot form a polygon) = 

P(can form a polygon) = 

Ref: <http://www.zhihu.com/question/25408010> and Math.pdf

Q8:  
Moving average.  
Reason:  
1. Outliers, moving average catches outliers. Outliers are signal that worth tracing.  
2. Sensitiveness. Moving average is more sensative to instant trading change, comparing to moving medium.

Q9.

We can calculate the expectation of sum of local maximas by calculating the sum of expectation of local maximas of each position.

It is not hard to see that the expectation of being maxima for the first position and last position is 0.5, since there is only one number next to it and the chance of either one been bigger is equal. We can also get that the expectation of being maxima for all the other positions is 1/3, since

there are two numbers next to it instead of one and the chance of each one been the biggest is equal.

So the answer is 0.5\*2+⅓\*(n-2)=1+(n-2)/3

Q10.

A number of power of 2 has form of 10...0 and the number minus 1 has form of 01...0. If we and them bit-wise, it will return 0. We also want to exclude 0. C++ Code:

bool isPower2(int x)

{

return x && !(x & (x - 1));

}

Ref: <http://stackoverflow.com/questions/3638431/determine-if-an-int-is-a-power-of-2-or-not-in-a-single-line>

Q11.

Please see Q11\_smartpointer.cpp

Q12.

Please see Q12\_ReverseLinkedList.cpp

Q13.

Python Code:

import numpy as np

import heapq

def mnp(m, n, prices, p):

return np.mean(heapq.nlargest(m, prices[-n:])) <= p

Some test case:

>>> np.random.seed(123)

>>> prices = 5 \* np.random.randn(100) + 100

>>> mnp(5, 50, prices, 100)

False

>>> mnp(5, 50, prices, 105)

False

>>> mnp(5, 50, prices, 110)

True

>>> mnp(25, 50, prices, 105)

True

Better version:

Sometimes it is better to test if the input satisfy requirements first, i.e. if n < len(prices), m <= n, and if p and prices are numeric, m and n are integer, and then provide proper error message when the input is not valid. Here is an example:

import numpy as np

import heapq

def mnp(m, n, prices, p):

try:

m, n, p = int(m), int(n), float(p)

prices = [float(x) for x in prices]

if m > n or n > len(prices):

raise NotImplementedError('m should be no more than n, n should be no more than the length of prices')

else:

return np.mean(heapq.nlargest(m, prices[-n:])) <= p

except ValueError:

print ('Error: m, n should be integers, prices and p should be numeric')

Some test case:

>>> np.random.seed(123)

>>> prices = 5 \* np.random.randn(100) + 100

>>> mnp(5, 50, prices, 100)

False

>>> mnp(150, 50, prices, 100)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "<stdin>", line 6, in mnp

NotImplementedError: m should be no more than n, n should be no more than the length of prices

>>> mnp(5, 150, prices, 100)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "<stdin>", line 6, in mnp

NotImplementedError: m should be no more than n, n should be no more than the length of prices

>>> mnp('5.3', 50, prices, 100)

Error: m, n should be integers, prices and p should be numeric

Q14.

We use the straightforward way to compare the substring with the whole string. If the first character of substring appears, it starts to count until finished the comparison. If the comparison is interrupted, we will move forward to later comparison. At last, it returns the index list of all matched positions. C++ Code:

#include <iostream>

#include <string>

#include <vector>

using namespace std;

vector<int> indexOf(string& str, string& substr)

{

vector<int> ivec; // use vector to contain match index

int len = str.size();

int sublen = substr.size();

int limit = len - sublen + 1;

for (int i = 0; i < limit; i++)

{

int count = 0;

for (int j = 0; j < sublen; j++)

{

if (str[j+i] != substr[j])

{

i += j; // increase i by j if not match

break;

}

count++;

}

if (count == sublen)

{ ivec.push\_back(i);}

}

return ivec;

}

int main()

{

string s = "qishibuysidequantshixinhang";

string p = "shi";

vector<int> ivec = indexOf(s,p); // match at 2 and 17

for (int i = 0; i < ivec.size(); i++)

{

cout << ivec[i] << ' ';

}

return 0;

}

Q15.

We use Taylor expansions to expand the exponential function. Since the expansion terms will be smaller and smaller, we stop the approximation with the term is smaller than the value we defined. C++ Code:

#include <iostream>

using namespace std;

// use Taylor expansion to approximate the e^x

double expo(double x)

{

double ret = 0;

double t = 1;

// stop approximation if the item is small enough

for (int i = 1;t > 1.0E-20;i++)

{

ret += t;

t = t \* x / i;

}

return ret;

}

int main()

{

double j = expo(20);

cout << j << endl;

return 0;

}

Q16.

Please see Q16\_median.cpp and Q16.docx

Q17.

If we can both long and short stocks, the long short profit (lsp) will be the sum of absolute difference of any two consecutive numbers. If we can only long stocks, the long profit (lp) will be the sum of positive difference of any two consecutive numbers. We also attach the code to find the buy and sell day numbers when only long is permitted.

Python Code to find max profit:

plist = [10,11,12,13,14,12,12,11,12,14,13,11]

lp = 0 # max profit for only long

lsp = 0 # max profit for both long and short

for i in range(len(plist)-1):

if plist[i] < plist[i+1]:

lp += plist[i+1] - plist[i];

lsp += plist[i+1] - plist[i];

else:

lsp += plist[i] - plist[i+1]

print lp,lsp

Python Code to find buy and sell point:

plist = [10,11,12,13,14,12,12,11,12,14,13,11]

blist = []

slist = []

pos = False

if plist[0] < plist[1]:

pos = True

blist.append(1)

for i in range(1,len(plist)-1):

if plist[i-1] <= plist[i] and plist[i] > plist[i+1] and pos:

slist.append(i+1)

pos = False

if plist[i-1] >= plist[i] and plist[i] < plist[i+1] and not pos:

blist.append(i+1)

pos = True

if pos:

slist.append(len(plist)+1)

print 'Buy at ',blist,'\nSell at ',slist