1. test can be used to test the goodness of fit.

The test-statistic is

Where n is the number of observation,

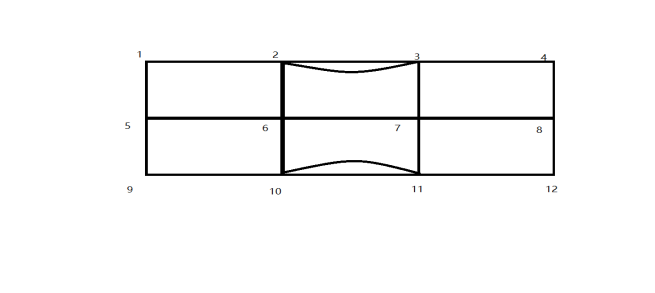
is the observed frequency of outcome i,

is the expected frequency of outcome i.

Once we got we can calculate the corresponding p value of distribution with freedom k=n-q-1, where q is the reduction of freedom typically equal the number of parameters for the fitting.

If p value smaller than certain conventional criteria say 0.05, namely the probability of getting this much difference between the observation and the theoretical expectation is too low if we believe the null hypothesis (no difference between the theory and experiment) then we need to reject the theory.

1. This is a Chinese postman problem. For a 2 by 3 grid we have 6 vertices with odd order which are 2,3,5,8,10,11. We can make it a semi Eulerian graph by connecting 2-3 and 10-11 directly. Now we can start from 5 and end at 8 to traverse all edges of the new graph. The length of the path will be 17+2=19.



1. X,Y are iid N(0,1).
2. DP
3. For the shortest one:

Assume we cut at point A and point B and A<B. Then the lengths of all pieces are A, B-A, 1-B. Let L be the length of the shortest one, then probability of L equal to x is the derivative of P(L<x)=1-P(L>x)=1-2P(A>x,B-A>x,1-B>x) where 2 comes from the equivalent case B<A.

To find this cumulative distribution function we need to integrate B from A+x to 1-x and A from x to 1-2x

Hence

Therefore

The shortest piece cannot be longer then 1/3 otherwise there must be another one shorter than it. This fixes the upper limit of integration.

For the longest one

Similarly, we still start from looking for

The problem is still how to fix the limit of integration.

What we have now are

We can find multiple lower and upper limit for A and B,

for A we have:

For B we have:

Therefore we need to consider different cases of x,

When 1-2x>0, or x<1/2, the integration limit of A is (1-2x,x) and for B it is (1-x,A+x)

When 1-2x<0, or x>1/2, the integration limit of A is (0,1-x) and for B it is (1-x, A+x), or A from (1-x,x) and B from (A,1).

Therefore the expected length of longest piece is

The expected value of all three pieces is 1

The expected value of midsize piece is 5/18.

1. Assign as the v.r. for ith person to get the right hat and

Therefore

By applying the linearity of expectation, represent the total number of person who got right hat, then

(1).

(2). As for the variance,

(3). Since the expected number of people who gets the right hat is 1, then the expected number of rounds R(N) should be N.

We know for N=1, E(R(N))=1, assume N=k E(R(k))=k is true. Then the expected number of rounds for N=k+1 is

X represents the number of person who got the right hat at this round.

Because we know E(R(K))=k

Hence we proved that E(R(N))=N by induction.

(4). 没想明白

(5). 没想明白

1. 不会
2. First we need to convert the 1/n into binary form up to certain accuracy, say D digits;

Then we start flipping the coin D times using the fair coin function. Record the Head as 1 and Tail as 0 for each flip. For example if the outcome is HHT we record 0.110;

Compare this number to 1/n in base two;

If the number we got from flipping is smaller than 1/n, return head, otherwise return tail.

The reasoning behind this is, to get a number smaller than 1/n, we need to successfully flip all the zeros before the first 1, say at ith digit, in the binary form of 1/n, the if we flip zero gain, we are guaranteed to have a number smaller than 1/n, this gives probability ; if we flip 1, then we need to continue to next 1 in binary form of 1/n, therefore the total probability of getting a number smaller than 1/n is :

Where Ci is the ith digit of 1/n in binary form.

We truncated the 1/n at Dth digit,

if after that there are only zeros in the binary expansion then we have the exact solution if there are all ones, we will have the maximum error .

1. Use to denote the expected number of bridges one need to cross from ith island to 10th island.

Then

The first term means there is ½ probability that one will make a cross from ith island to island i+1 by crossing one bridge and ½ probability of starting over from 1st island. And we know that .

We can use to express and use to express , so on so forth we can eventually express with as

Solve this we can get

1. This is a constant member function that takes parameter p, which is a reference to a constant pointer that points to a constant integer. This function returns a constant pointer which points to a constant integer.
2. This can be done by a recursive function

|  |
| --- |
| 1. **struct** LLNode\* removeElements(**struct** LLNode \* head, **int** val) { 2. **if**(head==NULL) **return** NULL; 3. head->next = removeElements(head->next,val); 4. **if**(head->val==val) 5. { 6. **struct** LLNode \* next; 7. next = head->next; 8. free(head); 9. **return** next; 10. } 11. **return** head; 12. } |

Namely we first go to the end of the list, then start check the values of the element, if it is equal the val, point its predecessor to its next, otherwise point to itself.

1. Matrix
2. There is no virtual constructor in C++ as far as I know.

But there is way to implement a effective virtual constructor

|  |
| --- |
| 1. **class** Shape { 2. **public**: 3. **virtual** ~Shape() { }                 // A virtual destructor 4. **virtual** **void** draw() = 0;             // A pure virtual function 5. **virtual** **void** move() = 0; 6. // ... 7. **virtual** Shape\* clone()  **const** = 0;   // Uses the copy constructor 8. **virtual** Shape\* create() **const** = 0;   // Uses the default constructor 9. }; 10. **class** Circle : **public** Shape { 11. **public**: 12. Circle\* clone()  **const**;   // Covariant Return Types; see below 13. Circle\* create() **const**;   // Covariant Return Types; see below 14. // ... 15. }; 16. Circle\* Circle::clone()  **const** { **return** **new** Circle(\***this**); } 17. Circle\* Circle::create() **const** { **return** **new** Circle();      } |

1. It sounds OK to me.
2. Any single character is a palindrome. If the character after this character is the same, then they form a two char palindrome. If the characters before and after this palindrome are the same then we got a four char palindrome.

We need a table to record whether the substring from i to j is a palindrome.

T[i,i] should be all true. Then check the consecutive char and see if T[i,i+1] is true.

After that we need to go over every substring T[i,j] and check if it is palindrome and if the chars before and after it are the same, if so we update the T[i-1,j+1]. Then we need to expand this to next possible length of the palindrome, Therefore this will have O(N^2) time complexity.

|  |
| --- |
| 1. string longestPalindrome(string s) 2. { 3. **int** n = s.length(); 4. **int** longestBegin = 0; 5. **int** maxLen = 1; 6. **bool** table[n][n] = {**false**}; 7. **for** (**int** i = 0; i < n; i++) 8. { 9. table[i][i] = **true**; 10. } 11. **for** (**int** i = 0; i < n-1; i++) 12. { 13. **if** (s[i] == s[i+1]) 14. { 15. table[i][i+1] = **true**; 16. longestBegin = i; 17. maxLen = 2; 18. } 19. } 20. **for** (**int** len = 3; len <= n; len++) 21. { 22. **for** (**int** i = 0; i < n-len+1; i++) 23. { 24. **int** j = i+len-1; 25. **if** (s[i] == s[j] && table[i+1][j-1]) 26. { 27. table[i][j] = **true**; 28. longestBegin = i; 29. maxLen = len; 30. } 31. } 32. } 33. **return** s.substr(longestBegin, maxLen); 34. } |