ASSIGNMENT - II

CS 6381:
ADVANCED PROGRAMMING LABORATORY

SHILPA S
(220CS1256)

1 COMPUTE VALUE OF Π

Compute Π using randomized algorithm.

1.1 Algorithm:

Algorithm 1 Algorithm to compute value of Π

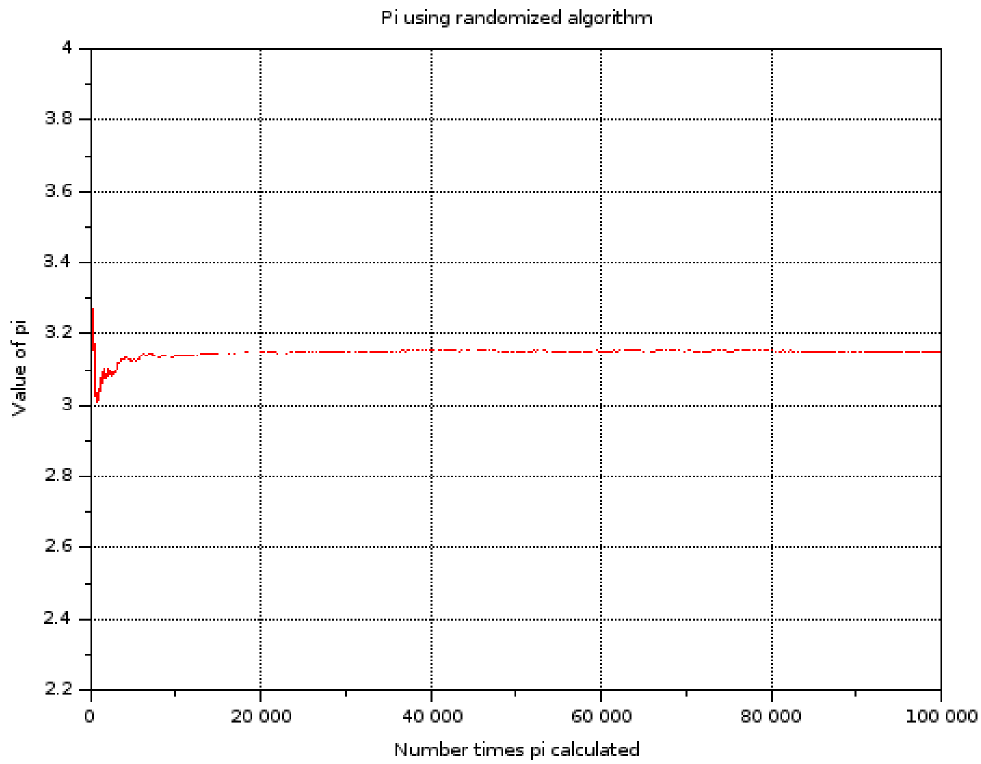
```
1. Initialize circle_points to 0.
2. Initialize numberOfIterations and piValue to zero
   matrix of 100000 elements.
3. For each i value from 1 to 100000
     3.1 Generate random point x.
     3.2 Generate random point y.
     3.3 Calculate d = x*x + y*y.
     3.4. If d <= 1, increment circle_points.
     3.5. Calculate piValue(i) = 4*(circle_points/i).
4. Print piValue(100000).
5. Plot numberOfIterations and piValue.
6. Give proper xlabel, ylabel and title.
```

1.2 Program code:

```
k=0;
n=zeros(1,100000);
pi=zeros(1,100000);
for i=1:100000
      n(i)=i;
      x= rand(1,1);
      y=rand(1,1);
      r = sqrt(x*x + y*y);
      if r<1
            k=k+1;
      end
      pi(i) = 4*k/i;
end
printf("Value of pi:\t");
printf("%.4f",pi(100000));
plot(n,pi,"r");

title("Pi using randomized algorithm");
xlabel("Number times pi calculated");
ylabel("Value of pi");
```

```
xgrid;
a=gca()  ;
a.box="on";
a.data_bounds=[0,2.2;100000,4];
```

1.3 Interpretation:



1.4 Result:

Value of pi: 3.1498

Number Of Iterations	**Value Of Π**
20000	3.1416
30000	3.1453
40000	3.1411
50000	3.1449
60000	3.1459
70000	3.1477
80000	3.1450
90000	3.1444
100000	3.1424

2 NUMERICAL INTEGRATION

Write a program that computes the value of the following integral using randomized algorithm.

$$\int_0^2 \sqrt{4-x^2}\,dx$$

Algorithm 2 Algorithm to compute value of given integral

```
1. Initialize k to 0.
2. Initialize numberOfIterations and Value to zero
   matrix of 100000 elements.
3. Assign values to a, b, c and d according to the question
4. For each i value from 1 to 100000
     4.1 Generate random number and multiply it with (b-a)
         and add b to the result, assign it to x.
     4.2 Generate random number and multiply it with (d-c)
         and add c to the result, assign it to y.
     4.3 Calculate d = square root of 4 - x*x.
     4.4. If d <= 1, increment k.
     4.5. Calculate Value(i) = (d-c)*(b-a)*k/i
4. Print Value(100000).
5. Plot numberOfIterations and Value.
6. Give proper xlabel, ylabel and title.
```

2.1 Program Code:

```
k=0;
loop = zeros(1,100000);
value = zeros(1,100000);
for i=1:100000
    loop(i) = i;
    x= rand()*2 ;
    y= rand()*2 ;

    if y<= sqrt(4-x*x)
        k=k+1;
    end

    value(i) = 2*(2*(k/i));

    if(pmodulo(i,10000) == 0 && i> 10000)
```

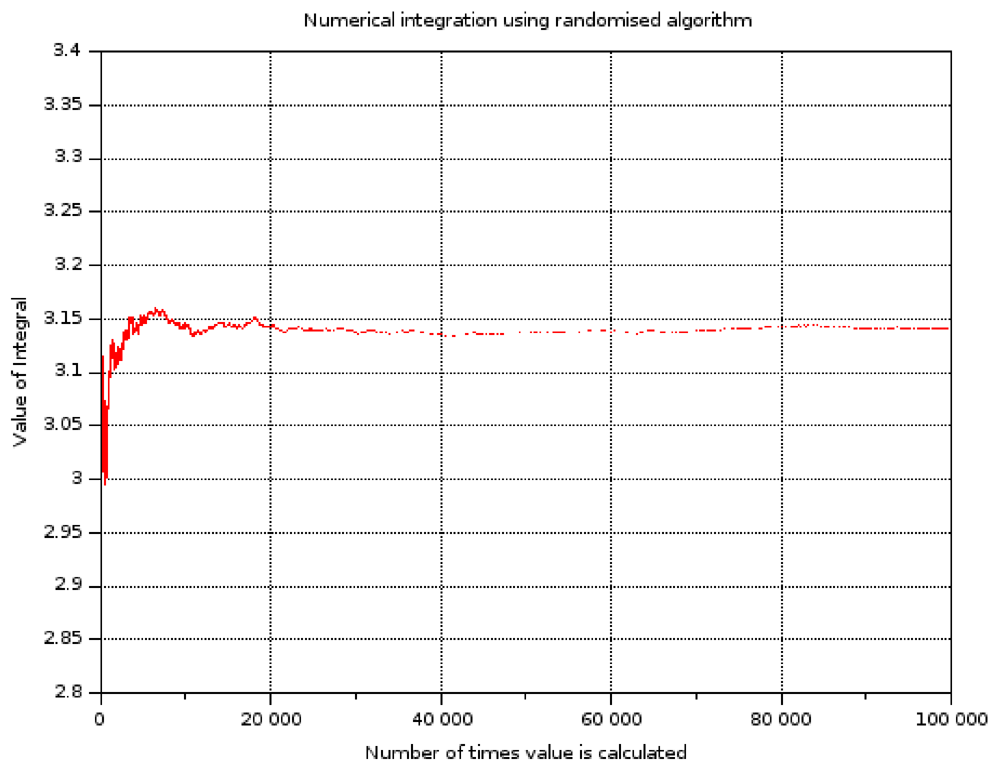
```
        printf("%d\t",i);
        printf("%.4f\n",pi(i));
    end
end

//disp(k);
disp(value(100000));
plot(loop,value,"r");
title("Numerical integration using randomised algorithm");
xlabel("Number of times value is calculated");
ylabel("Value of Integral");
xgrid;
a=gca() ;//get the current axes
a.box="on";
a.data_bounds=[0,2.9;100000,3.4];   //define the bounds
```

2.2 Result:

Value of Integral: 3.1502

Number Of Iterations	**Value Of Integral**
20000	3.1538
30000	3.1531
40000	3.1528
50000	3.1531
60000	3.1524
70000	3.1502
80000	3.1517
90000	3.1464
100000	3.1502

2.3 Interpretation:



3 PRIMALITY TESTING

Write a program that test a number to be prime or not. Perform an analysis to compute the correctness?

3.1 Algorithm:

Algorithm 3 FERMAT ALGORITHM

```
1  Repeat following k times:
      1.1 Pick a randomly in the range [2, n − 2]
      1.2 If gcd(a, n)     1, then return false
      1.3 If an−1 &nequiv; 1 (mod n), then return false
2 Return true [probably prime].
```

Algorithm 4 MILLER - RABIN ALGORITHM

```
1 Handle base cases for n < 3
2 If n is even, return false.
3 Find an odd number d such that n−1 can
  be written as d*2r. Note that since n is odd,
  (n−1) must be even and r must be
   greater than 0.
4 Do following k times
    4.1 if (millerTest(n, d) == false)
          return false
5 Return true.
```

Algorithm 5 MILLER TEST FUNCTION

```
1 Pick a random number 'a' in range [2, n−2]
2 Compute: x = pow(a, d) % n
3 If x == 1 or x == n−1, return true.

// Below loop mainly runs 'r−1' times.
4 Do following while d doesn't become n−1.
     4.1 x = (x*x) % n.
     4.2 If (x == 1) return false.
     4.3 If (x == n−1) return true
```

3.2 Program Code:

```
//Basic method to identify prime

 function[prime] = isPrime(n)
    prime=1;
    if n<=2 then
        prime = 0;
        return;
    else if pmodulo(n,2) == 0 then
            prime=0;
        end
    end
    t=sqrt(n);

    for i=3:2:t
        if pmodulo(n,i) == 0 then
            prime = 0
            return;
        end
    end
endfunction
//method to identify prime using fermant method

function [bool] = fermant(n,s)
    if n== 3 then
        bool=1;
        return;
    end
    bool = 1;
    for i=1:s
        a = round(2 + (n-3)* rand());
        if gcd(n,a) ~= 1 then
            bool=0;
            return;
        end
        t= a.^(n-1);
        p = pmodulo(t,n);
        if p ~= 1
             bool = 0;
             break;
        end
    end

endfunction
```

```
// This function is called for all k trials. It returns
// false if n is composite and returns false if n is
// probably prime.


function[prime]=millertest(n,d)
    flag=0;

    a = round(rand()*(n-4)+2);
    t= a.^d;
    x= pmodulo(t,n);
    if (x == 1) || (x == (n-1)) then
        prime=1;
        return;
    end
     while d ~= (n-1)
        p=x*x;
        d=d*2;
        x= pmodulo(p,n);
        if x== 1 then
            prime =0;
            return;
        end
        if x== (n-1) then
                prime =1;
                return;
         end

    end
    prime=0;
endfunction

/ It returns false if n is composite and returns true if n
// is probably prime.  k is an input parameter that determines
// accuracy level. Higher value of k indicates more accuracy.

function[prime] = millerRabin(n,k)
    prime=1;
    if n==4 then
        prime=0;
        return;
    end
    if n == 3 then
            prime=1;
            return;
```

```
    end
    d= n-1;
    while pmodulo(d,2) == 0
         d= d/2;
    end
    for i=1:k
        //prime = millertest(n,d);
        if ( millertest(n,d) == 0)

            prime = 0;
            return;
         end

    end
endfunction



fmiss =0;
mmiss =0;
fm = zeros(1,1000);
mm = zeros(1,1000);
noe = zeros(1,1000);
i=3;
k=2;
fm(1) =0;
fm(2)=0;
mm(1)=0
mm(2)=0

for j=3:1000
   noe(i)=j;
    //for j=1:i
        correct = isPrime(j);

        if millerRabin(j,k)~= correct
            mmiss = mmiss +1;


        end
      if fermant(j,k) ~=correct
          fmiss = fmiss +1;

        end
 fm(i) = fmiss ;
```

```
mm(i)= mmiss;
 i=i+1;


end

plot(noe,fm,"m");
plot(noe,mm,"k");

title("CORRECTNESS OF PRIMALITY TESTING");
xlabel("N");
ylabel("NUMBER OF NUMBERS <= N INCORRECTLY IDENTIFIED");
xgrid;
legend("FERMANT","MILLER-RABIN");
```

3.3 Conclusion:

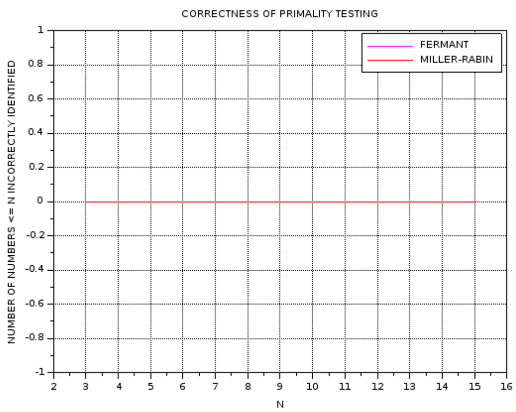
- From the experiment, It is observed that **Miller - Rabin** performs better than **Fermat** primality testing for small number ranges.
- For numbers less than 17, both algorithms identify prime numbers correctly.
- From the graph it is concluded that as number increases, accuracy of both algorithms remains same

FERMAT	**MILLER**
17	31
19	43
23	47
29	
31	
37	
41	
43	
47	

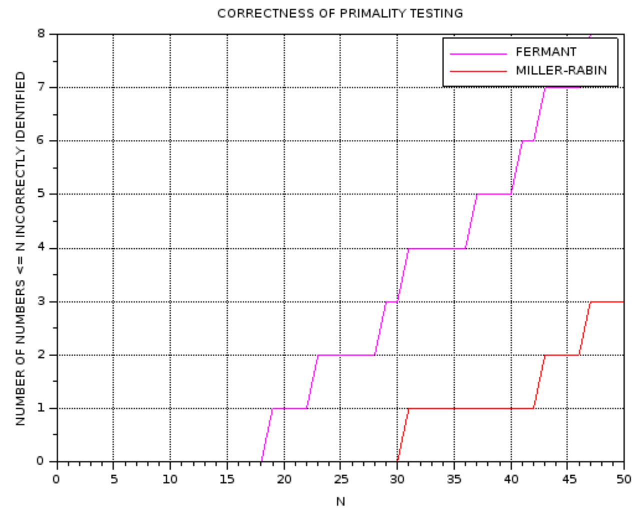
Table 1: upto 50

Numbers incorrectly identified (Prime numbers which are not correctly identified) by both algorithms

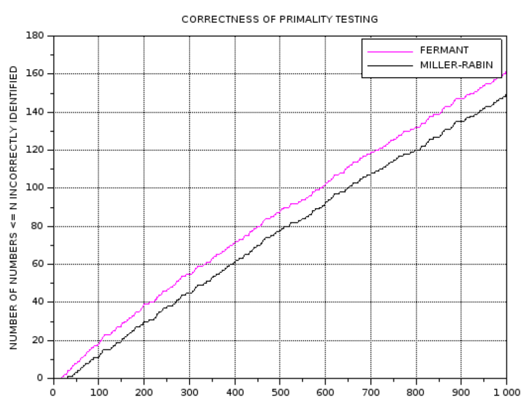
3.4 Intepretation:



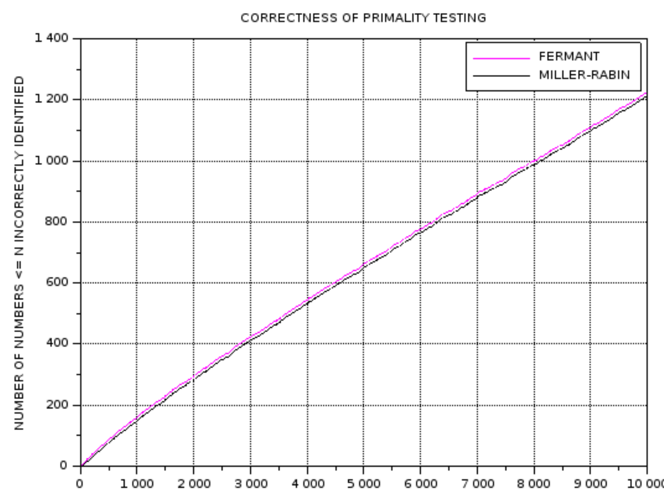
(a) N ranges from 3 to 15



(b) N ranges from 3 to 50



(a) N ranges from 1 to 1000



(b) N ranges from 1 to 10000

4 MAJORITY ELEMENT

Write a program that FINDS majority element from a linear array using randomized algorithm. Show that probability of missing majority element is 0.00097.

4.1 Algorithm:

Algorithm 6 Algorithm to find majority element

```
1. Generate a array containing majority element.
2. Checks whether a random number is the majority element
   2.1 if it is, then increment majP value.
   2.2 else increment majA value
3. Find the probability of occurrence of majA and majP
   in 100 outcomes.
4. Plot majA and majP
5. Give proper label, title and legend.
```

Algorithm 7 Algorithm to generate array containing majority element

```
1. Arraygenerate(M,N)
1.1   compute len = round(N/2) +1
1.2   For array indexes from 1 to len,
      assign M as value.
1.3   For array indexes from len+1 to n,
      assign values randomly.
1.4   Disorder array values by randomly swapping.
1.5   Return array.
```

Algorithm 8 Algorithm to search for majority element

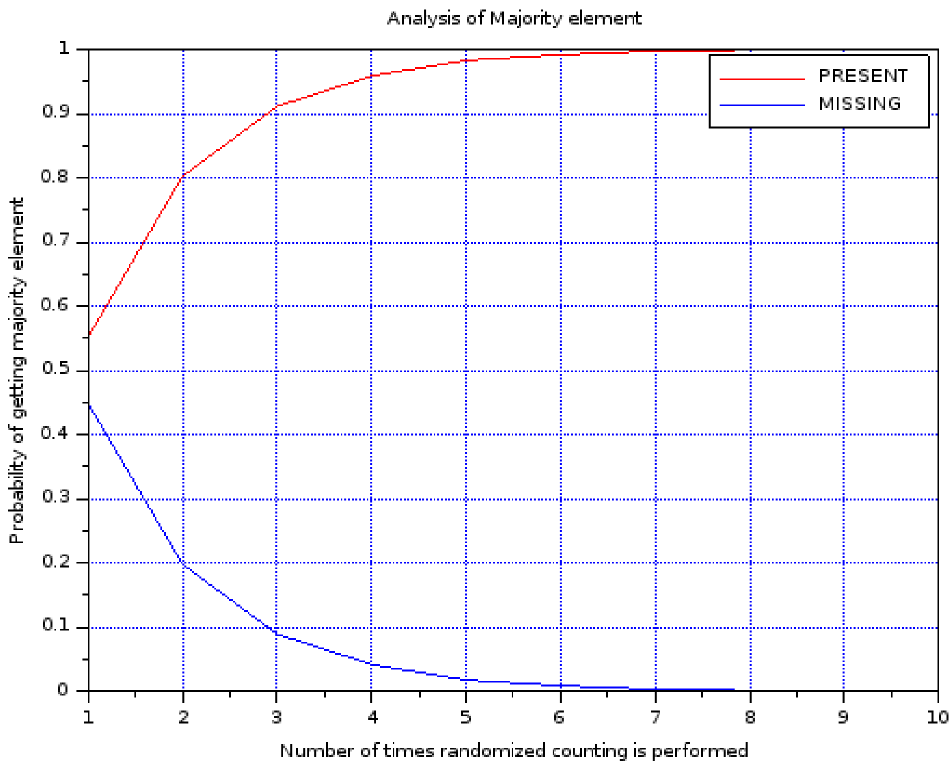
```
1. CheckMajority(arr,n)
1.2 Initialise j as a random number between 1 and n.
1.3 Initialise index as -1
1.4 for i value from 1 to n
    1.4.1 if arr(i) == arr(j) and i != j,
          then assign index as i
1.5 return index
```

4.2 Program Code:

```
//function checks for majority element k times
function [index] = SearchForMajorityElement(arr,k)
    for i=1:k
        index = IndexOfMajority(arr);
        if index ~= -1
            break;
        end
   end
endfunction

//function checks whether a random number is majority element or
not
//if it is majority element returns its index
function [index] = IndexOfMajority(arr)
    index = -1;
    n = length(arr);
    i = grand("uin",1,n)
    count = 0;
    for j=1:n
        if arr(j) == arr(i)
        count = count + 1;
        end
    end
    if count > n/2
        index = i;
    end
endfunction

//function generates array with majority element
function [arr] = ArrayGeneration(N,M)
    len = round(N/2) + 1;
    arr = zeros(1,N);
    for i=1:len
        arr(i) = M;
    end
    for i=(len+1):N
        arr(i) = round(rand()*10);
    end
    for i=1:N-len
        j = round(i + (N-i)*rand());
        temp = arr(i);
        arr(i) = arr(j);
        arr(j) = temp;
```

```
    end
endfunction

//main program
noi = zeros(1,10);
majP = zeros(1,10);
majA = zeros(1,10);
M= round(rand()*10);
N= round(rand()*1000)+1;

for j=1:10
for i=1:10000
    noi(j) =j
    a = ArrayGeneration(N,M);
    index = SearchForMajorityElement(a,j);
    if index ~= -1
        majP(j) = majP(j) + 1;
    else
        majA(j) = majA(j)+1;
    end
end
majP(j) = majP(j)/10000;
majA(j) = majA(j)/10000;
end

plot(noi,majP,"r");
plot(noi,majA);

title('Analysis of Majority element');
xlabel('Number of times randomized counting is performed');
ylabel('Probability of getting majority element');
xgrid(2);
legend('PRESENT','MISSING',1);
```

4.3 Intepretation:



4.4 Comments:

If a majority element is present in the array, then probability of missing majority element is less than 0.5.
If algorithm is repeated 10 times, then the probability of missing majority element, if there is a majority element is less than 0.5^{10} = **0.00097.**

Number Of Times Randomized Counting Performed	**Probability of Missing Majority element**
1	0.4478
2	0.1972
3	0.0887
4	0.0413
5	0.0171
6	0.0084
7	0.0028
8	0.0012
9	0.0007
10	0.0002