INSTITUT SUPÉRIEUR D'INFORMATIQUE DE MODÉLISATION ET DE LEURS APPLICATIONS

Stochastic Simulation

Jiarui XIE

Semester 2018/2019
Superviser: David Hill

Catalogue

Introduction

From the previous TP, we know for scientific applications default system random number generators are often very old and statistically weak, so we tried to use MT algorithm as the random number generator. Besides, we know the result of single experiment can’t represent the real state of the system, so we use the mean of N independent experiments as well as confidence intervals as substitutions. At last , we wrote a code simulating the rabbit population growth.

1 Compute π with the Monte Carlo Method

1.1 Monte Carlo Method

Drop a dot into a square, the possibility of the place where it located is equality (like Figure1.1). Inspired form this , why can't we use the ratio of dots location to representing the ratio of the area.

There is a equation : $\frac{Area\ of\ Circle}{Area\ of\ Square} = \frac{\pi}{4}$

Then how to judge whether the dot located in the circle or not? Suppose the location of dot is (x, y) , if $x^2 + y^2 \leq r^2$ then the dot in the circle.

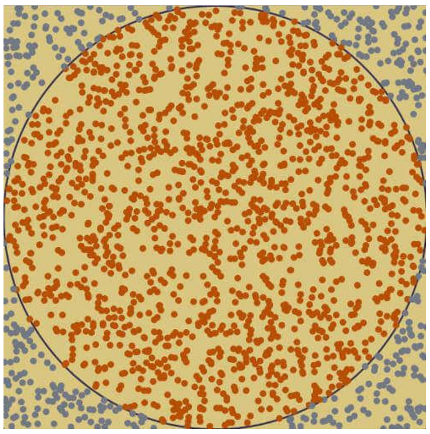
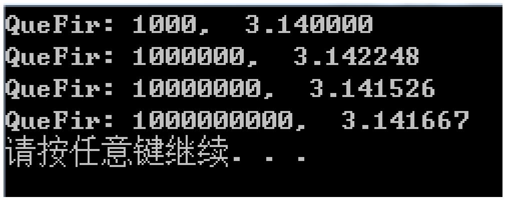


Figure1.1 Illustration of Monte Carlo Method

1.2 Simulation

Test my code with 1000 points、1000000 points and the 1 000 000 000 points, the results are 3.140000、3.142248 and 3.141667. But the last case cost too much time. When the number of points are 10000000 we got a precision of 10^{-3} as well as a precision of 10^{-4} .The result of simulation as following figure1.2.

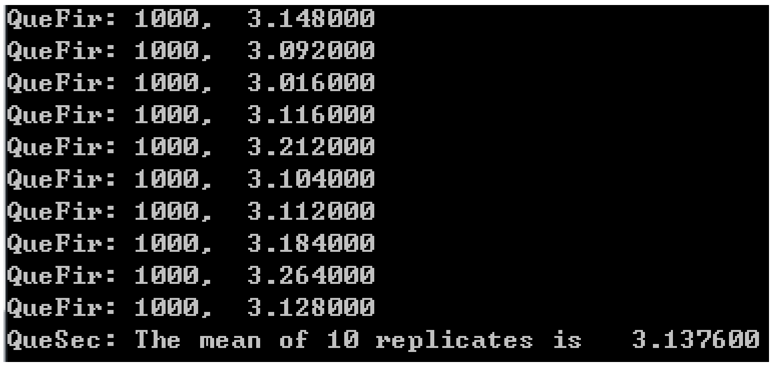


```
QueFir: 1000,  3.140000
QueFir: 1000000,  3.142248
QueFir: 10000000,  3.141526
QueFir: 1000000000,  3.141667
请按任意键继续. . .
```

Figure1.2 Results of Question1

1.3 Mean of N independent Experiments

We computed 10 independent experiments obtained result as figure1.3.And it seems better than anyone of the ten result.

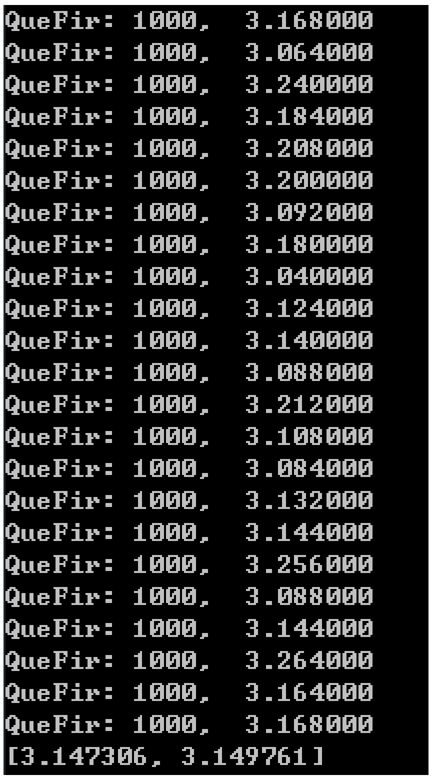


```
QueFir: 1000,  3.148000
QueFir: 1000,  3.092000
QueFir: 1000,  3.016000
QueFir: 1000,  3.116000
QueFir: 1000,  3.212000
QueFir: 1000,  3.104000
QueFir: 1000,  3.112000
QueFir: 1000,  3.184000
QueFir: 1000,  3.264000
QueFir: 1000,  3.128000
QueSec: The mean of 10 replicates is   3.137600
```

Figure1.3 Result of Question2

1.4 Confidence Intervals

As principle, the confidence radius at the 1-α level, is given by R = $t_{n-1,1-\alpha/2}$ $\times\sqrt{\frac{S(n)^2}{n}}$, in which the $S(n)^2 = \frac{\sum_{i=1}^{1}\left[x_i - \overline{X(n)}\right]^2}{n-1}$, in which $\overline{X(n)} = \sum_{i=1}^{n} X_i/n$. In my code I stored the table of $t_{n-1,1-\alpha/2}$ in the array, use the subscript as the value of n. We obtained the results as following figure1.4. The M_PI in math.h is 3.14159265358979323846. The number of drawings improves my results and decreases the confidence radius.



```
QueFir: 1000,  3.168000
QueFir: 1000,  3.064000
QueFir: 1000,  3.240000
QueFir: 1000,  3.184000
QueFir: 1000,  3.208000
QueFir: 1000,  3.200000
QueFir: 1000,  3.092000
QueFir: 1000,  3.180000
QueFir: 1000,  3.040000
QueFir: 1000,  3.124000
QueFir: 1000,  3.140000
QueFir: 1000,  3.088000
QueFir: 1000,  3.212000
QueFir: 1000,  3.108000
QueFir: 1000,  3.084000
QueFir: 1000,  3.132000
QueFir: 1000,  3.144000
QueFir: 1000,  3.256000
QueFir: 1000,  3.088000
QueFir: 1000,  3.144000
QueFir: 1000,  3.264000
QueFir: 1000,  3.164000
QueFir: 1000,  3.168000
[3.147306, 3.149761]
```

Figure1.4 Result of Question3

2 Rabbit Population Growth

2.1 Introduction about rabbit population

For this exercise, we will simulate a reproduction of rabbits. Indeed, we will consider rabbits as couples, which can be either young or adult. A couple of young rabbits need a month to become an adult then can start copulating. The gestation lasts one month after which they give birth to a new pair of young rabbits. In addition, we will consider that rabbits never die and reproduce to infinity. The simulation must then Following figure 2.1 is the illustration of rabbit population, table 2.1 is the number sequence of rabbit.

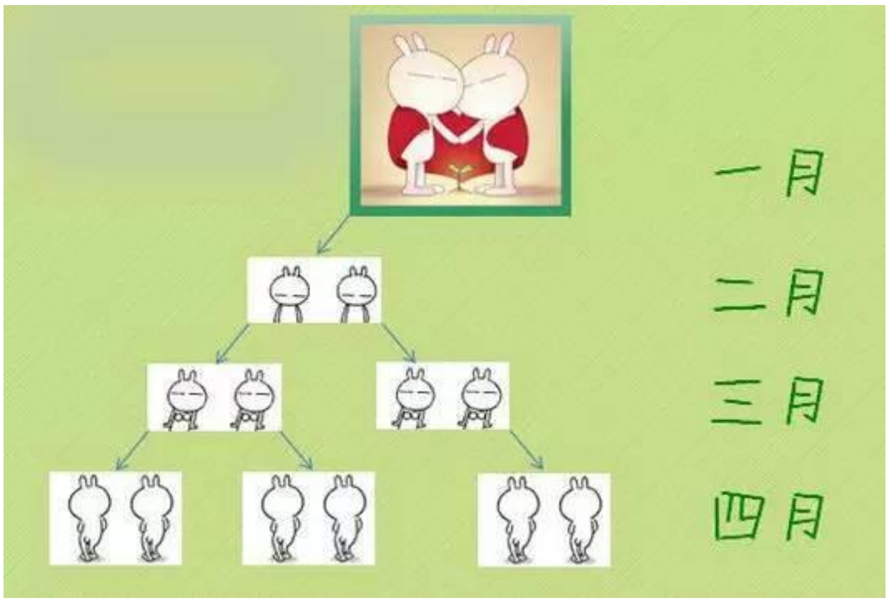


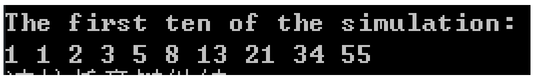
Figure2.1 Illustration of Rabbit Population

Table2.1 Values of Rabbits

Iteration	Number of Rabbits
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21
9	34

2.2 Simulation

We ran our program, using long int to simulate 10 month. In this way, we were able to obtain the following results as figure 2.2.



```
The first ten of the simulation:
1 1 2 3 5 8 13 21 34 55
```

Figure2.1 Results of Question4

Conclusion

Due to Monte Carlo Method we can calculate the π easily, and it let me realize the importance of Stochastic Simulation. For rabbit population the essence is Fibonacci Sequence.

Reference

[1] *https://blog.csdn.net/nomad2/article/details/6307824*.
[2] *http://www.math.sci.hiroshima-u.ac.jp/~m-mat/eindex.html*.

Appendix

Appendix 1 : The whole code.

```
/*
Complier:VS 2015
Auther:Jiarui XIE
*/

#include <stdio.h>
# include <stdlib.h>
#include <math.h>

#define ZERO 0.0000001
#define PI 3.141592
#define SIZE_MONTH 10000
#define SIZE_REPLICATE 1000

/* Period parameters */
#define N 624
#define M 397
#define MATRIX_A 0x9908b0dfUL   /* constant vector a */
#define UPPER_MASK 0x80000000UL /* most significant w-r bits */
#define LOWER_MASK 0x7fffffffUL /* least significant r bits */

static unsigned long mt[N]; /* the array for the state vector  */
static int mti = N + 1; /* mti==N+1 means mt[N] is not initialized */

double TableArr[] = {
               -1, 12.706, 4.303, 3.182, 2.776, 2.571, 2.474, 2.365, 2.308, 2.262, 2.228,
                   2.201, 2.179, 2.160, 2.145, 2.131, 2.120, 2.110, 2.101, 2.093, 2.086,
                    2.08, 2.074, 2.069, 2.064, 2.060, 2.056, 2.052, 2.048, 2.045, 2.042
                    };


                       /* initializes mt[N] with a seed */
void init_genrand(unsigned long s)
{
        mt[0] = s & 0xffffffffUL;
        for (mti = 1; mti<N; mti++)
        {
               mt[mti] =
                       (1812433253UL * (mt[mti - 1] ^ (mt[mti - 1] >> 30)) + mti);
               /* See Knuth TAOCP Vol2. 3rd Ed. P.106 for multiplier. */
               /* In the previous versions, MSBs of the seed affect   */
               /* only MSBs of the array mt[].                        */
               /* 2002/01/09 modified by Makoto Matsumoto             */
               mt[mti] &= 0xffffffffUL;
               /* for >32 bit machines */
        }
}
```

```
/* initialize by an array with array-length */
/* init_key is the array for initializing keys */
/* key_length is its length */
/* slight change for C++, 2004/2/26 */
void init_by_array(unsigned long init_key[], int key_length)
{
	int i, j, k;
	init_genrand(19650218UL);
	i = 1; j = 0;
	k = (N>key_length ? N : key_length);
	for (; k; k--)
	{
		mt[i] = (mt[i] ^ ((mt[i - 1] ^ (mt[i - 1] >> 30)) * 1664525UL))
			+ init_key[j] + j; /* non linear */
		mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
		i++; j++;
		if (i >= N) { mt[0] = mt[N - 1]; i = 1; }
		if (j >= key_length) j = 0;
	}
	for (k = N - 1; k; k--)
	{
		mt[i] = (mt[i] ^ ((mt[i - 1] ^ (mt[i - 1] >> 30)) * 1566083941UL))
			- i; /* non linear */
		mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
		i++;
		if (i >= N) { mt[0] = mt[N - 1]; i = 1; }
	}

	mt[0] = 0x80000000UL; /* MSB is 1; assuring non-zero initial array */
}

/* generates a random number on [0,0xffffffff]-interval */
unsigned long genrand_int32(void)
{
	unsigned long y;
	static unsigned long mag01[2] = { 0x0UL, MATRIX_A };
	/* mag01[x] = x * MATRIX_A  for x=0,1 */

	if (mti >= N)
	{ /* generate N words at one time */
		int kk;

		if (mti == N + 1)   /* if init_genrand() has not been called, */
			init_genrand(5489UL); /* a default initial seed is used */

		for (kk = 0; kk<N - M; kk++)
		{
			y = (mt[kk] & UPPER_MASK) | (mt[kk + 1] & LOWER_MASK);
			mt[kk] = mt[kk + M] ^ (y >> 1) ^ mag01[y & 0x1UL];
		}
		for (; kk<N - 1; kk++)
		{
			y = (mt[kk] & UPPER_MASK) | (mt[kk + 1] & LOWER_MASK);
			mt[kk] = mt[kk + (M - N)] ^ (y >> 1) ^ mag01[y & 0x1UL];
		}
		y = (mt[N - 1] & UPPER_MASK) | (mt[0] & LOWER_MASK);
```

```
                mt[N - 1] = mt[M - 1] ^ (y >> 1) ^ mag01[y & 0x1UL];

                mti = 0;
        }

        y = mt[mti++];

        /* Tempering */
        y ^= (y >> 11);
        y ^= (y << 7) & 0x9d2c5680UL;
        y ^= (y << 15) & 0xefc60000UL;
        y ^= (y >> 18);

        return y;
}

/* generates a random number on [0,0x7fffffff]-interval */
long genrand_int31(void)
{
        return (long)(genrand_int32() >> 1);
}

/* generates a random number on [0,1]-real-interval */
double genrand_real1(void)
{
        return genrand_int32()*(1.0 / 4294967295.0);
        /* divided by 2^32-1 */
}

/* generates a random number on [0,1)-real-interval */
double genrand_real2(void)
{
        return genrand_int32()*(1.0 / 4294967296.0);
        /* divided by 2^32 */
}

/* generates a random number on (0,1)-real-interval */
double genrand_real3(void)
{
        return (((double)genrand_int32()) + 0.5)*(1.0 / 4294967296.0);
        /* divided by 2^32 */
}

/* generates a random number on [0,1) with 53-bit resolution*/
double genrand_res53(void)
{
        unsigned long a = genrand_int32() >> 5, b = genrand_int32() >> 6;
        return(a*67108864.0 + b)*(1.0 / 9007199254740992.0);
}
/* These real versions are due to Isaku Wada, 2002/01/09 added */

double QueFir(int numPoint)
{
        int numInCircle = 0;
        double radius = 0.5;
        double x, y;
        for (int i = 0; i < numPoint; i++)
        {
                x = genrand_real1() - 0.5;
                y = genrand_real1() - 0.5;
                if (x * x + y *y - 0.5 * 0.5 <= ZERO)
                        numInCircle++;
        }
        double pi = numInCircle / (numPoint * 0.25);
        printf("QueFir: %d,%10.6f\n", numPoint, pi);
        return pi;
}

void QueSec(int n)
{
        double sum = 0;
        for (int i = 0; i < n; i++)
                sum += QueFir(1000);
        printf("QueSec: The mean of %d replicates is %10.6f\n", n, sum / n);
}

double GetMean(double arr[], int size)
{
        double sum = 0.0;
        for (int i = 0; i < size; i++)
                sum += arr[i];
        return sum / size;
}

double GetS2(double arr[], int size, double mean)
{
        double sum = 0.0;
        for (int i = 0; i < size; i++)
                sum += pow(arr[i] - mean, 2);
        return sum / (size - 1);
}

double GetTa(int n)
{
        if (n >= 1 && n <= 30)
                return TableArr[n];
        else if (n == 30)
                return 2.021;
        else if (n == 40)
                return 2.0;
        else if (n == 80)
                return 1.980;
        else if (n > 80)
                return 1.96;
        else
                return -1;
}

void QueThird(int n)
{
        double arr[SIZE_REPLICATE] = { 0 };
        int in = 0;
```

```
	if (n >= 0 && n <= SIZE_REPLICATE)
	{
		for (int i = 0; i < n; i++)
			arr[i] = QueFir(1000);

		double ave = GetMean(arr, n);
		//printf("ave=%10.6f \n", ave);
		double s2 = GetS2(arr, n, ave);
		//printf("s2=%10.6f \n", s2);
		double ta = GetTa(n);
		//printf("ta=%10.6f \n", ta);
		double r = ta * sqrt(pow(s2, 2) / n);
		printf("[%8.6f, %8.6f]\n", ave - r, ave + r);
		for (int i = 0; i < n; i++)
			if (arr[i] > (ave - r) && arr[i] < (ave + r))
				in++;
		//printf("%d\n", in);
		//printf("the percetage out of  is %.2f\n", (n-in) / (double)n);
		//printf("As 95\% the mean of pi is %f \n", r);
	}
	else
		printf("Please input right n\n");

}

void QueForth()
{
	long int a[SIZE_MONTH] = { 1,1 };
	for (int i = 2; i < SIZE_MONTH; i++)
		a[i] = a[i - 1] + a[i - 2];
	printf("The first ten of the simulation:\n");
	for (int i = 0; i < 10; i++)
		printf("%d ", a[i]);
	printf("\n");
}


int main(void)
{
	//QueFir(1000);
	//QueFir(1000000);
	//QueFir(10000000);
	//QueFir(1000000000);
	QueSec(10);
	//QueThird(30);
	//QueForth();

	system("pause");
	return 0;
}
```