

Simulation (spring 2005) – Solutions to assignment #2

Problem #1 - A random variable X is defined as follows:

$$\Pr[X = 1] = 0.25$$

$$\Pr[X = 2] = 0.15$$

$$\Pr[X = 3] = 0.05$$

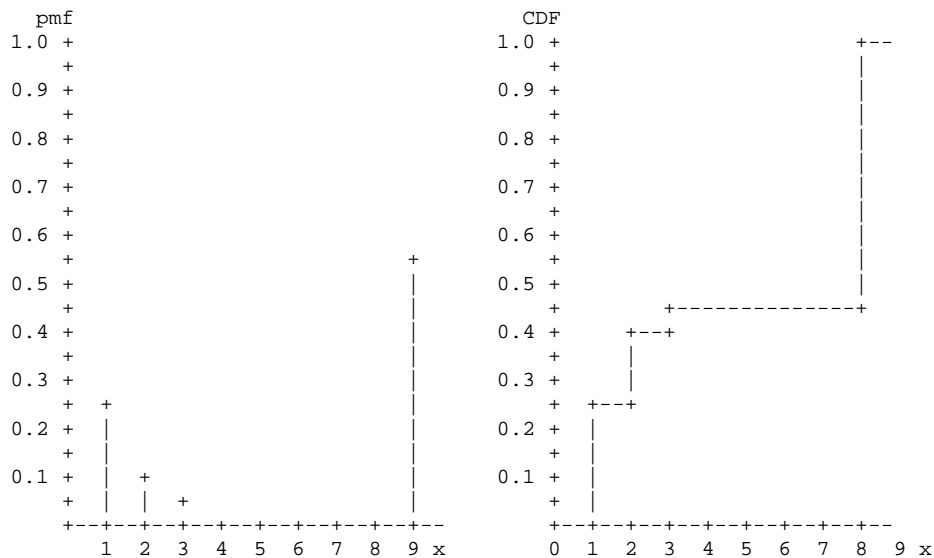
$$\Pr[X = 9] = 0.55$$

Find the mean, variance, and standard deviation of X . Plot the probability distribution function and cumulative distribution function. Show your work.

$$\text{mean} = 1 \cdot 0.25 + 2 \cdot 0.15 + 3 \cdot 0.05 + 9 \cdot 0.55 = 5.65$$

$$\text{variance} = (1^2 \cdot 0.25 + 2^2 \cdot 0.15 + 3^2 \cdot 0.05 + 9^2 \cdot 0.55) - 5.65^2 = 13.9275$$

$$\text{standard deviation} = \sqrt{\text{variance}} = \sqrt{13.9275} = 3.732$$



Problem #2 - In class you learned about the inversion method to generate random variables from various distributions. The program here generates and plots a histogram for exponentially distributed random variables. Change the program to generate $\text{unif}(1.0, 10.0)$ (i.e., uniformly distributed continuous-valued random variables between 1.0 and 10.0). Submit the program and an output from an execution.

The program, which includes the output from an execution in the header block, is here below. **The key new and changed code is highlighted in yellow.**

```

//===== file = as3_2sol_1.c =====
//= Program to demonstrate inversion method of generating an RV =
//=====
//= Notes: =
//= 1) This program is for assignment #2 for CIS 4930 (Simulation) for =
//= Spring 2005 =
//=====
//= Build: bcc32 as3_2sol_1.c =
//=====
//= Execute: as3_2sol_1 =

```

```

//=====
//= Sample execution (NUM_BINS 12, NUM_SAMPLES 50000, MAX_BAR 250) =
//=
//= 0 < x <= 1 --> =
//= 1 < x <= 2 --> ***** =
//= 2 < x <= 3 --> ***** =
//= 3 < x <= 4 --> ***** =
//= 4 < x <= 5 --> ***** =
//= 5 < x <= 6 --> ***** =
//= 6 < x <= 7 --> ***** =
//= 7 < x <= 8 --> ***** =
//= 8 < x <= 9 --> ***** =
//= 9 < x <= 10 --> ***** =
//= 10 < x <= 11 --> =
//= 11 < x <= 12 --> =
//=====
//= History: KJC (01/04/05) - Genesis (from as3_2.c) =
//=====
//----- Include files -----
#include <stdio.h> // Needed for printf()

//----- Constants -----
#define NUM_BINS 12 // Number of measurement "bins"
#define NUM_SAMPLES 50000 // Number of samples to simulate
#define MAX_BAR 250 // Maximum bar size for histogram print-out

//----- Function prototypes -----
double rand_val(void); // Returns a unif(0,1) random variable
double unif(double a, double b); // Returns a unif(a, b) random variable

//===== Main program =====
void main(void)
{
    double a, b; // Min and max of uniform distribution
    double unif_rv; // Uniform random variable
    int count[NUM_BINS]; // Counter for each bin
    int bar_size; // Bar size for print-out of histogram
    int i, j; // Loop counters

    // Clear the count vector
    for (i=0; i<NUM_BINS; i++)
        count[i] = 0;

    // Set min and max values for this run
    a = 1.0;
    b = 10.0;

    // Main loop to generate samples and update count
    for (i=0; i<NUM_SAMPLES; i++)
    {
        unif_rv = unif(a, b);
        for (j=0; j<(NUM_BINS - 1); j++)
            if (((double) j < unif_rv) && (unif_rv <= ((double) (j + 1))))
                count[j]++;
    }

    // Output a histogram of the count vector
    for (i=0; i<NUM_BINS; i++)
    {
        printf("%2ld < x <= %2ld --> ", i, i+1);
        bar_size = ((double) count[i] / NUM_SAMPLES) * (double) MAX_BAR;
        for (j=0; j<bar_size; j++)
            printf("*");
        printf("\n");
    }
}

//=====
//= Function to generate uniformly distributed RVs =
//= - Input: a, b (min and max values, respectively) =
//= - Output: Returns with uniform RV =
//=====

```

```

double unif(double a, double b)
{
    double z; // Uniform random number from 0 to 1

    // Pull a uniform RV (0 < z < 1)
    z = rand_val();

    // Return unif(a, b) RV
    return(z * (b - a) + a);
}

//=====
//= Multiplicative LCG for generating uniform(0.0, 1.0) random numbers =
//= - x_n = 7^5*x_(n-1)mod(2^31 - 1) =
//= - With x seeded to 1 the 10000th x value should be 1043618065 =
//= - From R. Jain, "The Art of Computer Systems Performance Analysis," =
//= John Wiley & Sons, 1991. (Page 443, Figure 26.2) =
//=====
double rand_val(void)
{
    const long a = 16807; // Multiplier
    const long m = 2147483647; // Modulus
    const long q = 127773; // m div a
    const long r = 2836; // m mod a
    static long x = 1; // Random int value (seed is set to 1)
    long x_div_q; // x divided by q
    long x_mod_q; // x modulo q
    long x_new; // New x value

    // RNG using integer arithmetic
    x_div_q = x / q;
    x_mod_q = x % q;
    x_new = (a * x_mod_q) - (r * x_div_q);
    if (x_new > 0)
        x = x_new;
    else
        x = x_new + m;

    // Return a random value between 0.0 and 1.0
    return((double) x / m);
}

```

Problem #3 - You are give the following population data:

1, 2, 1, 4, 5, 1, 3, 3

Find the mean, variance, and standard deviation of the data. Plot a histogram of the data.

$$\text{mean} = \frac{1}{8} \cdot (1 + 2 + 1 + 4 + 5 + 1 + 3 + 3) = 2.5$$

$$\text{variance} = \frac{1}{8} \left((1 - 2.5)^2 + (2 - 2.5)^2 + (1 - 2.5)^2 + (4 - 2.5)^2 + (5 - 2.5)^2 + (1 - 2.5)^2 + (3 - 2.5)^2 + (3 - 2.5)^2 \right) = 2$$

$$\text{standard deviation} = \sqrt{\text{variance}} = \sqrt{2} = 1.4142$$

```

Pr[occurrence]
  1/1 +
    +
  3/4 +
    +
  1/2 +
    + +
  1/4 + | + | + +
    + | + | + +
    +---+---+---+---+---+
        1  2  3  4  5

```

Problem #4 - Being able to characterize measurement data is very important as a first step to building a workload model for a simulation. A small data set is found [here](#). You are to find the population mean and variance, plot the histogram, and plot the autocorrelation for up to 100 lags. The tools on the Christensen tools page may be useful (hint: look at [summary1.c](#), [hist.c](#), and [autoc.c](#)).

The mean and variance we can get from summary1.c. Here is the program output:

```
----- summary1.c -----
Total of 1000000 values
  Minimum = 0.000000 (position = 551246)
  Maximum = 0.999999 (position = 407083)
  Sum      = 544140.922954
  Mean     = 0.544141
  Variance = 0.092607
  Std Dev  = 0.304314
  CoV      = 0.559256
-----
```

This histogram we can get from hist.c. The key here is what bucket size (and how many buckets) to use. The values range between 0 and 1 (see the summary1.c output above). It should be reasonable to plot 100 buckets, so here below (Figure 1) is a plot for BUCKET_SIZE 0.01 and NUM_BUCKETS 100 in hist.c. The autocorrelation we can get from autoc.c. A plot of the autocorrelation is also here below (Figure 2).

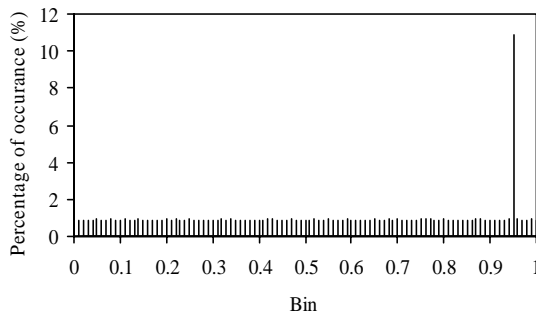


Figure 1 – Histogram of data1.txt

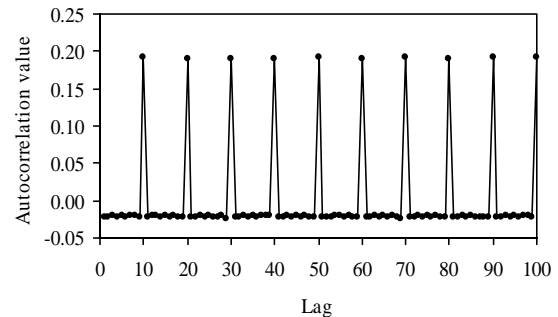


Figure 2 – Autocorrelation of data1.txt

Problem #5 - Being able to generate empirically distributed random variables is a very powerful "trick". Write your own C program or use the existing genemp.c on the Christensen tools page to generate empirically distributed random variables that match the data set from problem #2. Using your program generate 1 million values in a file. This is a "synthetic" data set. Now, repeat problem #2 (i.e., find the mean, variance, histogram, and autocorrelation) for the synthetic data set. Explain any differences in the characterization results for the actual and synthetic data sets

>>> NOTE THAT THIS PROBLEM SHOULD SAY "... data set from problem #4.". There is no data set in problem #2. I expect that all the student caught this since I received no emails or questions.

To use genemp.c an input file named dist.dat must be prepared. This input file is the pdf of the empirical distribution. To match an existing data set, this empirical distribution is simply the histogram from the data set. The program hist.c can be used to generate this empirical distribution input. Using 100 buckets, the first few lines of dist.dat are:

```
0.009060    0.010000
0.008927    0.020000
0.008873    0.030000
```

```

0.008927    0.040000
0.009080    0.050000
0.009031    0.060000
0.009063    0.070000
0.009114    0.080000
0.008976    0.090000
0.008945    0.100000
0.009073    0.110000
0.008968    0.120000
0.008957    0.130000
etc.

```

The program hist.c can easily be modified to give exactly the output format needed for dist.dat. The single printf() line in the main() of hist.c was changed to:

```

// Output probability and bin value
printf("%f    %f  \n", ((double) count / N), ((i + 1) * (double) BUCKET_SIZE));

```

Then, the characterization of problem #2 is repeated using summary1.c, hist.c, and autoc.c. The output from summary1.c is:

```

----- summary1.c -----
Total of 1000000 values
Minimum = 0.010000 (position = 999946)
Maximum = 1.000000 (position = 999990)
Sum      = 549407.920001
Mean     = 0.549408
Variance = 0.092686
Std Dev  = 0.304443
CoV      = 0.554130
-----

```

Note that the summary statistics match within a fraction of a percent between the actual and synthetic data sets. This is as expected. Figures 3 and 4 show the histogram and autocorrelation. The histogram shows the same spike at about 0.94, as expected. The smaller spikes cannot easily be explained. Also as expected, the autocorrelation is approximately zero for all lags. Generating random variates from a distribution cannot capture autocorrelation – i.e., dependence – properties due to the underlying rand_val() being (by definition of a good random number generator) independent.

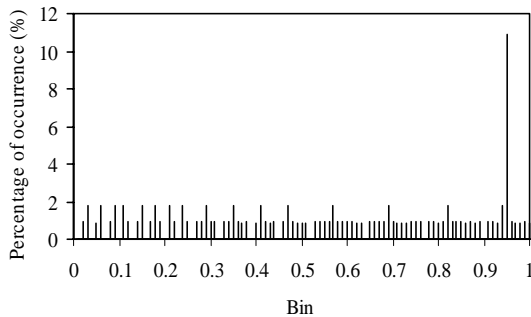


Figure 3 – Histogram of synthetic

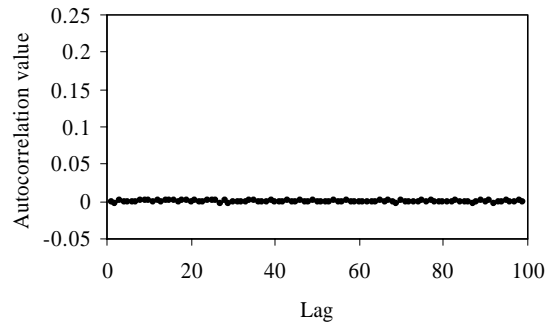


Figure 4 – Autocorrelation of synthetic
