EE3006* Experiment-1 Lab Report

Putta Shravya - EE22B032
Polamuri Hasmitha - EE22B075
Vasireddy Yasaswi - EE22B040

September 2024

Study the sample Scilab code 'gauss1.sce' and plot the histogram of random numbers produced using a gaussian distribution. How will you modify the generated r(n) to get an arbitrary mean xand standard deviation

Questions

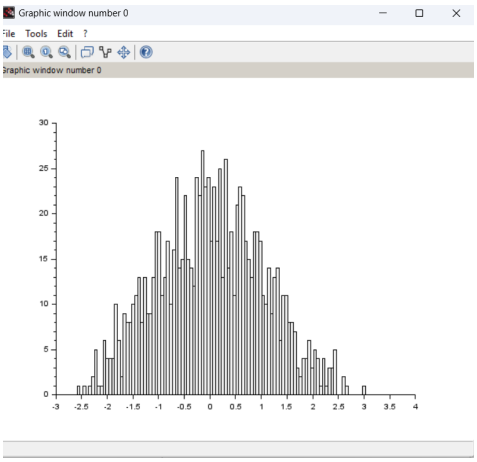
1) Modify the sample code and calculate the average and standard deviation of r(n) using two 'for' loops. (Set normalization to false and remove the gaussian comparison plot for simplicity). Try for N = 1000 and 100000 events. Use built-in help in Scilab or check the web. Show the results to your TA for credits. Check your answers using the built-in mean() and stdev() functions.

Code

```
N = 1000;
r=rand(N,1,"normal");

scf();

sum_r = 0;
for i = 1 :N
    sum_r = sum_r + r(i);
end

mean_r = sum_r / N;
diffr = 0;
for i=1:N
   diffr = diffr + (r(i)- mean_r)^2;
end
std_dev = sqrt(diffr/N);

printf("Calculated average of r(n): %f\n", mean_r);
printf("Calculated standard deviation of r(n): %f\n", std_dev);

printf("Mean using mean(): %f\n", mean(r));
printf("Standard deviation using stdev(): %f\n", stdev(r));


histplot(100,r,normalization=%f);
```

Output

```
--> exec('C:\Users\user\AppData\Local\Temp\02ddf197-0c28-44cc-a06e-9f07537a0326_GROUP_SHY.z
Calculated average of r(n): 0.017272
Calculated standard deviation of r(n): 1.036757
Mean using mean(): 0.017272
Standard deviation using stdev(): 1.037276

-->
```



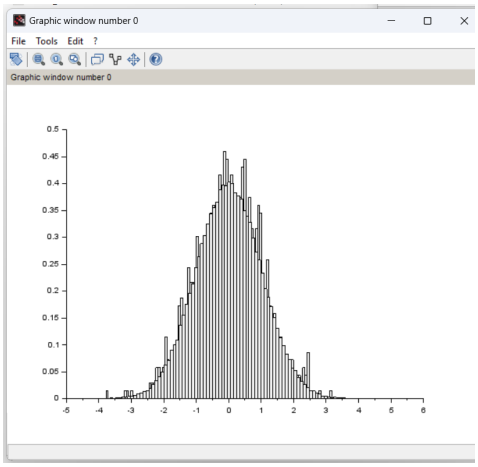
2) Using a 'for' loop and 'if' statements, count the fraction of events that lie outside range x ¿ 1, x ¿ 2 and x ¿ 3 and compare with the expected numbers. What do you think are the errors on these actual measured counts? Try for N =1000 and 100000 events.

Code

```
N_vals = [1000, 100000];

for N = N_vals
    r = rand(N, 1, "normal");

    count_1 = sum(abs(r) > stdev(r));
    count_2 = sum(abs(r) > 2 * stdev(r));
    count_3 = sum(abs(r) > 3 * stdev(r));

    avg_r = mean(r);
    std_dev_r = stdev(r);

    printf("For N = %d\n", N);
    printf("Calculated average of r(n): %f\n", avg_r);
    printf("Calculated standard deviation of r(n): %f\n", std_dev_r);

    printf("Fraction of events outside 1*sigma: %f\n", count_1 / N);
    printf("Fraction of events outside 2*sigma: %f\n", count_2 / N);
    printf("Fraction of events outside 3*sigma: %f\n", count_3 / N);

    histplot(100, r);
end
```

Output

```
--> exec('C:\Users\user\AppData\Local\Temp\6fad2c98-30f6-42f8-80fb-3de98b5f5bfa_GROUP_SHY.:
For N = 1000
Calculated average of r(n): 0.000540
Calculated standard deviation of r(n): 0.996234
Fraction of events outside 1*sigma: 0.306000
Fraction of events outside 2*sigma: 0.051000
Fraction of events outside 3*sigma: 0.004000
For N = 100000
Calculated average of r(n): 0.001977
Calculated standard deviation of r(n): 1.000561
Fraction of events outside 1*sigma: 0.317860
Fraction of events outside 2*sigma: 0.044830
Fraction of events outside 3*sigma: 0.002750

-->
```



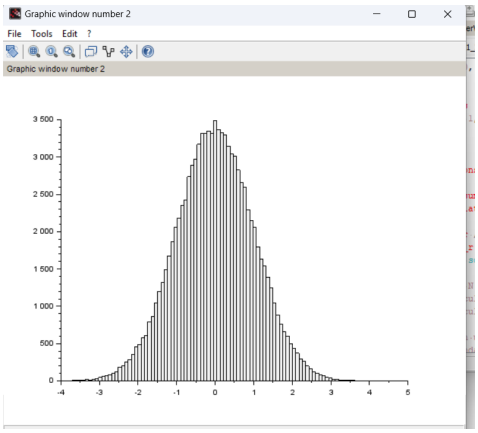
3) Note that in part 1, the calculation of standard deviation requires another separate 'for' loop, after calculating the average in the first 'for' loop. This problem can be avoided by finding sum of deviations around an arbitrary number k, which need not be the average x, as given: Write a Scilab code to verify this. Take the arbitrary k = 1. Note when k = x, these two expressions take the usual standard forms. Check your answers using the built-in mean() and stdev() functions.

Code

```
N_values = [1000, 100000];
k = 1;

for N = N_values
    r = rand(N, 1, "normal");
    scf();

    sumr = 0;
    sum_deviations = 0;
    for i = 1:N
        sumr = sumr + r(i) - k;
        sum_deviations = sum_deviations + (r(i) - k)^2;
    end
    avg_r = sumr / N;
    avg_r = avg_r + k;
    std_dev_r = sqrt(sum_deviations / N - (avg_r - k)^2);

    printf("For N = %d\n", N);
    printf("Calculated average of r(n): %f\n", avg_r);
    printf("Calculated standard deviation of r(n): %f\n", std_dev_r);

    printf("Mean using mean(): %f\n", mean(r));
    printf("Standard deviation using stdev(): %f\n", stdev(r));

    histplot(100,r,normalization=%f);
end
```

Output

```
--> exec('C:\Users\user\AppData\Local\Temp\e1a430ca-49c4-41a3-9982-30867d100bfe_GROUP_SHY.z
For N = 1000
Calculated average of r(n): -0.023321
Calculated standard deviation of r(n): 0.999598
Mean using mean(): -0.023321
Standard deviation using stdev(): 1.000098
For N = 100000
Calculated average of r(n): 0.001703
Calculated standard deviation of r(n): 1.000738
Mean using mean(): 0.001703
Standard deviation using stdev(): 1.000743
```



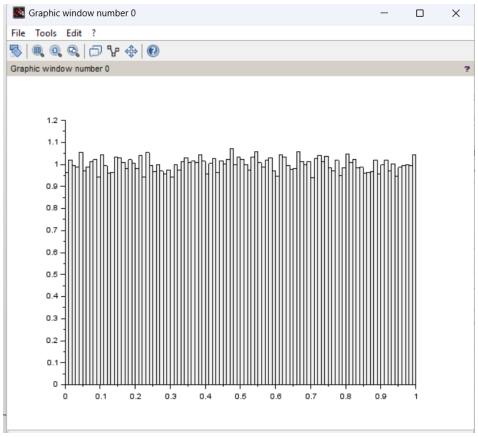
4) The Scilab 'rand' function can also produce a uniformly random distribution between 0 and 1. Show the histogram of this distribution and find and compare xand n with the expected values. Set histogram normalization to true. From now on, for simplicity, just use the mean() and stdev() functions.

Code

```
N = 100000;

r = rand(1, N);

histplot(100, r, normalisation = %t);

mean_r = mean(r);
std_dev_r = stdev(r);

expected_mean = 0.5;
expected_std_dev = 1 / sqrt(12);

printf("Calculated mean of the distribution: %f\n", mean_r);
printf("Expected mean of the uniform distribution [0, 1]: %f\n", expected_mean);

printf("Calculated standard deviation of the distribution: %f\n", std_dev_r);
printf("Expected standard deviation of the uniform distribution [0, 1]: %f\n", expected_std_dev);
```

Output

```
--> exec('C:\Users\user\AppData\Local\Temp\d2786971-9936-43ee-8f6e-eecb38d2be55_GROUP_SHY.z
Calculated mean of the distribution: 0.499811
Expected mean of the uniform distribution [0, 1]: 0.500000
Calculated standard deviation of the distribution: 0.288078
Expected standard deviation of the uniform distribution [0, 1]: 0.288675

-->
```



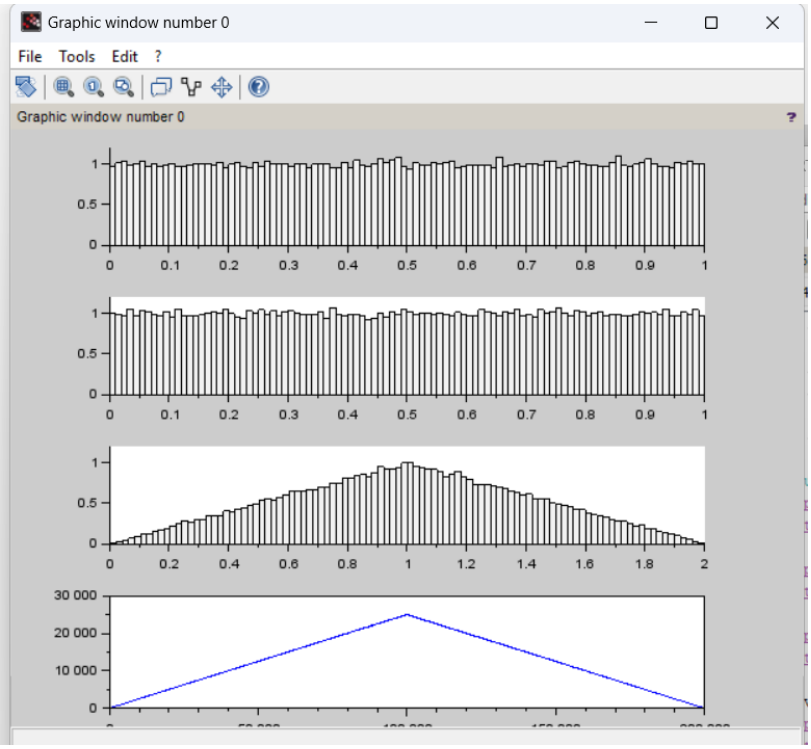
5) When you add two random variables z = x + y, show, as homework, that the pdf of the result z is given by the convolution: pz(z) = px(x)py(z-x)dx. Check this by adding and histogramming two

uniformly distributed random variables. Set normalization to true to see the pdf.

Code

```
N = 100000;

x = rand(1, N);
y = rand(1, N);

z = x + y;

figure();
subplot(4,1,1);
histplot(100, x, normalisation = %t);

subplot(4,1,2);
histplot(100, y, normalisation = %t);

subplot(4,1,3);
histplot(100, z, normalisation = %f);

convolution_pdf = convol(x, y);
subplot(4,1,4);
plot(convolution_pdf);
```

Output



6) Central Limit Theorem: when you add and average a very large number of n random variables, each with a different type of 'bell-shaped' pdfs of arbitrary xand n, the result tends to a gaussian distribution. Check this by producing and adding five different uniform and five different gaussian distributions. Note that if the pdfs are all gaussian with the same xand n, the new becomes smaller as n increases. Also the poisson and binomial distributions became gaussian for large N. For these

reasons, the gaussian distribution is very common in nature.

Code

```
N = 100000;

a = 1*rand(1, N);
b = 2*rand(1, N);
c = 3*rand(1, N);
d = 4*rand(1, N);
e = 5*rand(1, N);

z = a + b + c + d + e
figure();
subplot(3,1,1);
histplot(100, z, normalisation = %t);

f = grand(1, N, "nor", 0, 1);
g = grand(1, N, "nor", 0, 1);
h = grand(1, N, "nor", 0, 1);
i = grand(1, N, "nor", 0, 1);
j = grand(1, N, "nor", 0, 1);

y = f + g + h + i + j
printf("%f/n", stdev(y));
subplot(3,1,2);
histplot(100, y, normalisation = %t);

m=(y+z)/10;
subplot(3,1,3);
histplot(100, m, normalisation = %t);
```

Output