ECE131 : Winter 2019 Project

Chris Baker

105.180.929

1. Probability Estimator

ece131fairdice.m | ece131unfairdice.m | Untitled3* | ece131probest.m

```
clc
clear
errcount = 0;
pwd
dpoints = dlmread("data.txt", "");
for array = [10, 200, 300, 500, 1000, 2000, 10000, 50000]
    subarr = dpoints(1:array);
    %below prints the character from index in sub array
    %fprintf("sa: %d\n", subarr);
    ecount = length(find(mod(subarr,2)==1));
    %the below piece gave the answer 2 for a subarr of 20 which is wrong.
    %errcount = double(errcount) + double(ecount);
    pn = ecount/array;
    fprintf("Cardinality of oneset is %d for N = %d\n", ecount, array);
    fprintf("P[error in %d array] = %f\n\n", array, pn);
end
x = [10 200, 300, 500, 1000, 2000, 10000, 50000];
y = [.1 .04 .043333 .056 .06 .0525 .0523 .04994];
plot(x,y);
%oneset (5,75,100, 107, 127 ....)
```

a.

ECE131 : Winter 2019 Project

Chris Baker

105.180.929

ece131fairdice.m | ece131unfairdice.m | Untitled3* | ece131probest.m

```
clc
clear
errcount = 0;
pwd
dpoints = dlmread("data.txt", "");
for array = [10, 200, 300, 500, 1000, 2000, 10000, 50000]
    subarr = dpoints(1:array);
```

Command Window

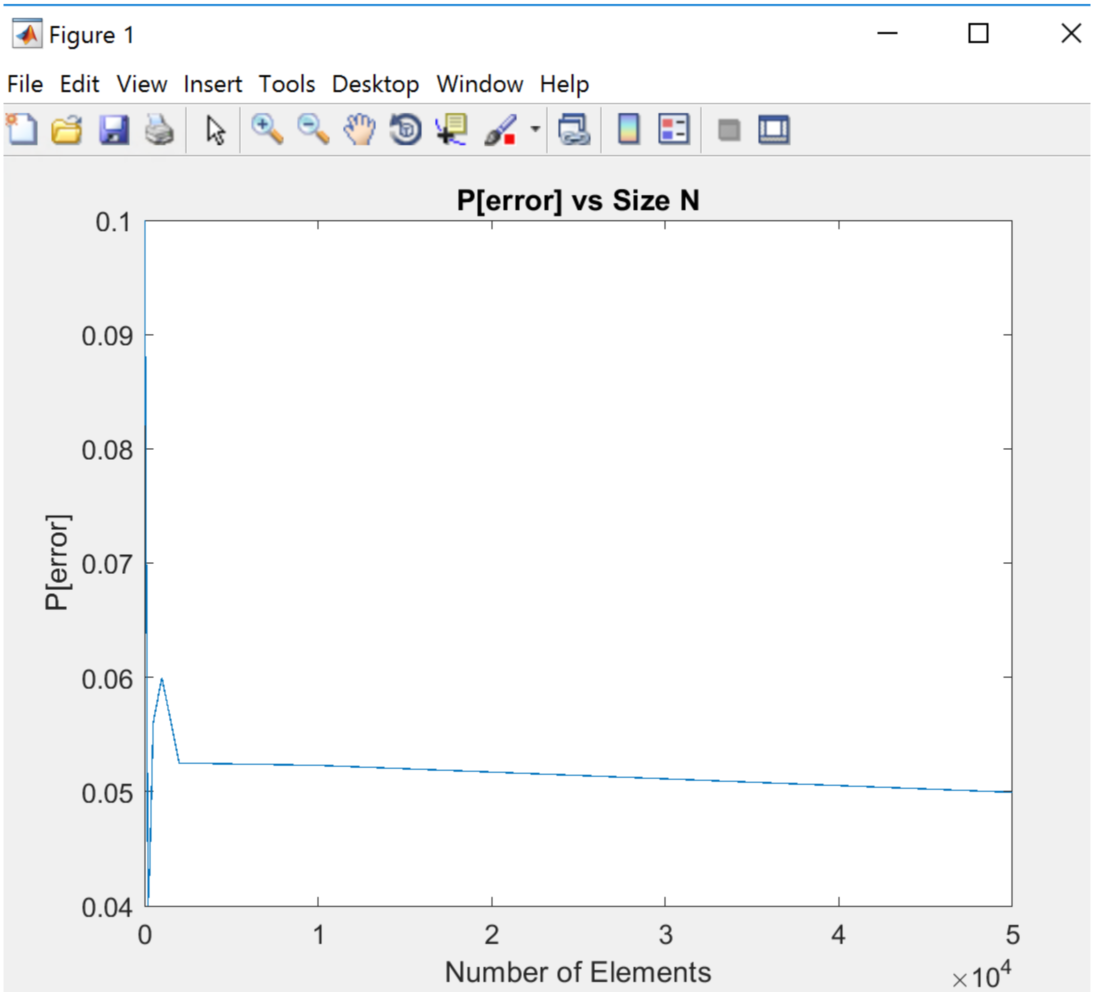
```
    'Z:\Documents'

Cardinality of oneset is 1 for N = 10
P[error in 10 array] = 0.100000

Cardinality of oneset is 8 for N = 200
P[error in 200 array] = 0.040000

Cardinality of oneset is 13 for N = 300
P[error in 300 array] = 0.043333

Cardinality of oneset is 28 for N = 500
P[error in 500 array] = 0.056000

Cardinality of oneset is 60 for N = 1000
P[error in 1000 array] = 0.060000

Cardinality of oneset is 105 for N = 2000
P[error in 2000 array] = 0.052500

Cardinality of oneset is 523 for N = 10000
P[error in 10000 array] = 0.052300

Cardinality of oneset is 2497 for N = 50000
P[error in 50000 array] = 0.049940
```

b.

ECE131 : Winter 2019 Project

Chris Baker

105.180.929



c.

d. This graph shows that the most error occurs within the smallest amount of sample size, but as we take a larger sample the probability of error tapers downward showing our system seems to be more reliable and possibly learning from previous errors to minimize the probability of error in the future. After the beginning portion which seems to jump up and down within the range of [0,300] the graph appears to have a somewhat linear error with a negative slope from [300, 50,000].

2. Tossing Fair and Unfair Dice

ECE131 : Winter 2019 Project

Chris Baker

105.180.929

Editor - Z:\Documents\ece131fairdice.m

ece131fairdice.m | ece131unfairdice.m

```
clc
for tosses = [10, 50, 100, 500, 1000] % sets up array of tosses

    %creates a matrix of 1xtosses to hold the values of random
    %numbers generated from 1 to 4 for the 4 sided die
    sides = randi([1 4],1,tosses);

    odds = mod(sides,2) == 1; % odds consists of sides 1 and 3

    %normally find() grabs nonzero entries in list
    %we are grabbing the entries that are odd, == 1 since mod2
    oddChance = find(odds);

    count = length(oddChance); %counting how many times we find

    %accumulate the probability: occurrences of desired event(co
    %divided over total occurences (tosses)
    p = count/tosses;

    fprintf("In %d tosses, P[odd] = %f\n", tosses, p);
end
```

Command Window

```
In 10 tosses, P[odd] = 0.700000
In 50 tosses, P[odd] = 0.460000
In 100 tosses, P[odd] = 0.370000
In 500 tosses, P[odd] = 0.466000
In 1000 tosses, P[odd] = 0.498000
fx
```

a.
b. If X is a random variable, then we can map X across the sides of the die 1 to 4.
    i. The die can land on either side 1, side 2, side 3, or side 4 each with the corresponding value being the number of the side.
    ii. Therefore, we have side 1 and side 3 being odd, whereas side 2 and side 4 are even. This means that for a fair die there are two chances that map to 1 (result in odd) and two chances that map to 0 (result in even).
    iii. Therefore, for RV X, there is 2/4 or ½ = 0.5 chance of being odd value

ECE131 : Winter 2019 Project

Chris Baker

105.180.929

c. We will use Percent Error Formula and the first set of values above in the photo
    i. **| (*measured* - *accepted*) / *accepted* | * 100 percent**
    ii. For 10 tosses: 40% error
    iii. For 50 tosses: 8% error
    iv. For 100 tosses: 26% error
    v. For 500 tosses: 6.8% error
    vi. For 1000 tosses: 0.4% error
    vii. As we would presume it makes sense that more error would correlate with less tosses since we are basing our results on a smaller sample. Even though the odds are truly 50% as sown in the theoretical application we can easily roll an odd number 7 times in one instance of 10 rolls of a 4 sided dice and receive a 70 percent probability of odd which is 40% error. However, as we roll the dice 1000 times or more the sample is larger and is able to average out the odds and evens better to more accurately depict the theoretical probability. In this case .498 which is almost .5.

d. For the last part we need to simulate a loaded die that is twice as likely for even numbers, however it is supposed to act like a 4-sided die. That means that we have sides 1 and 3 with the same probability, but 2 and 4 are repeated again. We attempt to simulate this by creating 4 groups which act like the sides, but we repeat two groups twice. This looks like a 6-sided die, but we map two groups to 1 side each, and the other two groups to two sides each, thus a 4-group or 4-side die.

ECE131 : Winter 2019 Project

Chris Baker

105.180.929

ece131fairdice.m | ece131unfairdice.m

```matlab
clc
fprintf("\n\n")
for tosses = [10, 50, 100, 500, 1000] % sets up array of tosses
    %creates a matrix of 1xtosses to hold the values of random
    %numbers are given with repeated value for 2 and 4.
    %technically this is a 6 sided dice now with repeated evens
    newsides = length([1,2,6,3,4,8]);
    %fprintf("%d\n", sides);
    % returns 6 which verifies 6 sided die
    %however we can group these as two sides twice likely. For 4 side die
    sides = randi(newsides,1,tosses); %randomize tosses, place in matrix
    %fprintf("%d\n", sides); % returns each integer result of roll (1 to 6)
    %we had a problem here, since mod2 == 1 was used on values
    % one through 6, so now we use mod3 and map (3,6 to 1,3) and
    %(1,2,4,5 to 2,4) so we have 4 groups with two groups twice likely
    odds = mod(sides,3) == 0;
    %fprintf("%d\n", odds);
    %normally find() grabs nonzero entries in list
    %we are grabbing the entries that are odd, == 0 since mod3 retuns 0 for
    %two choices 3 and 6
    oddChance = find(odds);
    count = length(oddChance); %counting how many times we find an ==1 odd
    %accumulate the probability: occurrences of desired event(count)
    %divided over total occurences (tosses)
    p = count/tosses;
    fprintf("In %d tosses with evens twice likely, P[odd] = %f\n", tosses, p);
end
```

Command Window

```
In 10 tosses with evens twice likely, P[odd] = 0.600000
In 50 tosses with evens twice likely, P[odd] = 0.360000
In 100 tosses with evens twice likely, P[odd] = 0.380000
In 500 tosses with evens twice likely, P[odd] = 0.296000
fx In 1000 tosses with evens twice likely, P[odd] = 0.323000
```

i.

ii. If X is a random variable for this loaded dice with evens twice likely, then we can visualize this mapping in two ways. Either by the groups I discussed or by the actual values. We can allow X to map to each side 1 through 4 with probabilities being 1/z for the odd values and 2/z for the even values. The sum of probabilities must be equal to 1 according to axioms of probability and using PMF properties. Therefore, 6/z = 1 which means z must be 6. The other approach is to consider that either way we have 4 groups but to represent each side of a 6-sided dice as probability 1/6. Then we would just have to group two sides twice to create our double probability (2/6) for a 4-sided dice otherwise known here as a 4-group dice. Therefore, the chances of an odd role are 2*(1/6) which is 2/6 = 1/3 = .333 repeated probability of odd role. Hence, we expect p(even) = 1-p(odd) = .666 repeated.

iii. We use the percent error formula again to compare the theoretical with observed values: theoretical being .333 and observed shown in the image

   1. **| (*measured - accepted*) / *accepted* | * 100 percent**

ECE131 : Winter 2019 Project

Chris Baker

105.180.929

2. For 10 tosses: 81.82% error
3. For 50 tosses: 9.09% error
4. For 100 tosses: 15.15% error
5. For 500 tosses: 11.111% error
6. For 1000 tosses: 3.003% error
7. Again, we see that the percent error decreases as we increase the number of tosses. We gather the depiction that the odds vs evens average out over larger numbers to yield a more accurate result to the theoretical value which we expect to be .333 for the chances of getting an odd number with this loaded dice. The value noticed at 50 tosses would not typically be expected to show up as 9 percent of, but this is plausible and with low sample sizes we can still expect outliers or less accurate data to occur, so this is understandable. The data set and results as whole makes sense.

3. Central Limit Theorem
   a.

Editor - Z:\Documents\ece131clta.m

ece131fairdice.m | ece131unfairdice.m | ece131probest.m | ece131clta.m | Untitled3* | Untitled4* | Untitled7*

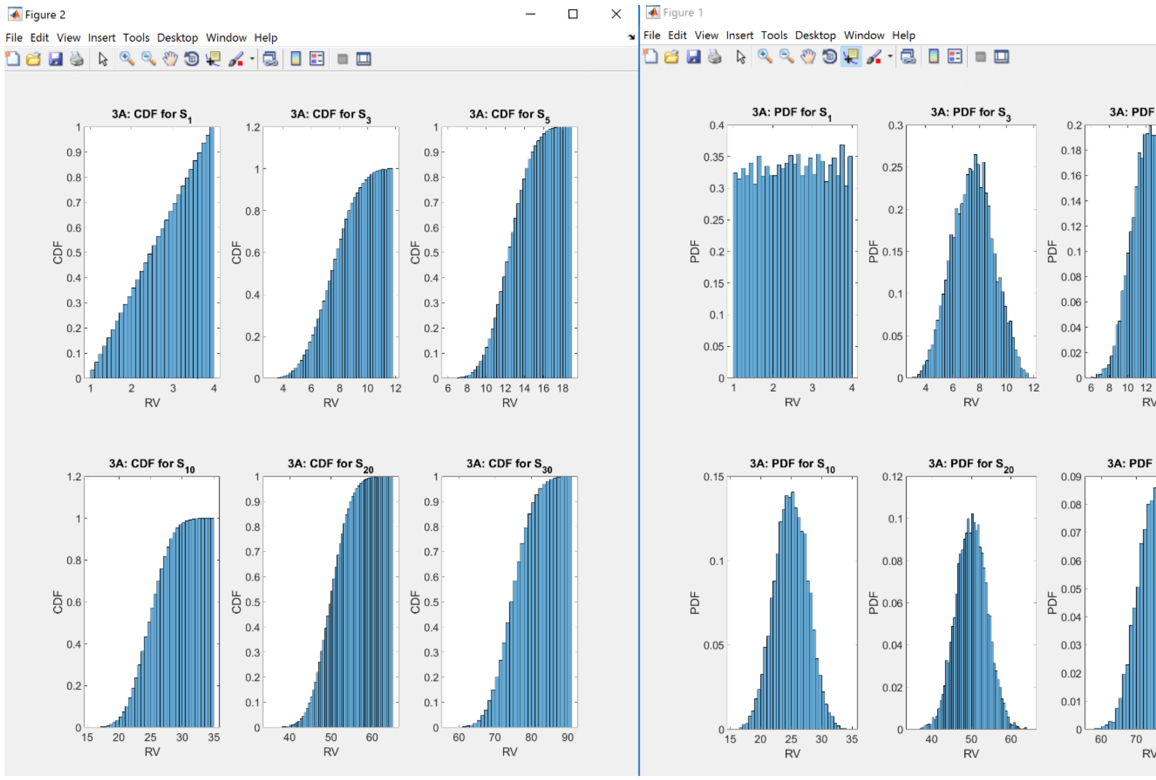
```
%3A
clc
clear
n = [1 3 5 10 20 30];
figure(1); figure(2);
for i=1:6
    Xi=unifrnd(1,4,10000,n(i)); %uniform Rand Var(interval [1,4], step 10k, for n(i))
    %fprintf("%d", n(i)); %testing correct n(i)
    Si=sum(Xi,2); % creates column vector with sum of rows
    fprintf("%d", Si); %testing correct n(i)
    figure(1),subplot(2,3,i)% 2x3 plots in figure 5
    figure(1),histogram(Si,'Normalization','pdf')
    xlabel("RV"); ylabel("PDF")
    title(sprintf("3A: PDF for S_{%d}", n(i)))

    figure(2),subplot(2,3,i)
    figure(2),histogram(Si,'Normalization','cdf')
    xlabel("RV"); ylabel("CDF")
    figure(2),title(sprintf("3A: CDF for S_{%d}", n(i)))
end
```

i.

ECE131 : Winter 2019 Project

Chris Baker

105.180.929

ii. 

b.

i. For the Mean: This can be done by taking the midpoint of each bar in the histogram and multiplying it by its frequency. Then we sum all the products. Finally, we divide by the sample size of t.

ii. For the Variance: We take the midpoint squared times the frequency in the histogram and subtract from this the value of the mean squared. Then we sum all these expressions and divide by the sample size of t.

iii. S1: 2.507477e+00 and 7.502984e-01
X1: 2.507477e+00 and 7.502984e-01

iv. S3: 7.489637e+00 and 2.317584e+00
X2: 2.496546e+00 and 7.507882e-01

v. S5: 1.251721e+01 and 3.689237e+00
X3: 2.503441e+00 and 7.517226e-01

vi. S10: 2.501862e+01 and 7.345563e+00
X4: 2.501862e+00 and 7.450720e-01

vii. S20: 5.002304e+01 and 1.467937e+01
X5: 2.501152e+00 and 7.485529e-01

viii. S30: 7.503072e+01 and 2.216641e+01
X6: 2.501024e+00 and 7.511728e-01

c. Using Values from S for mean and var

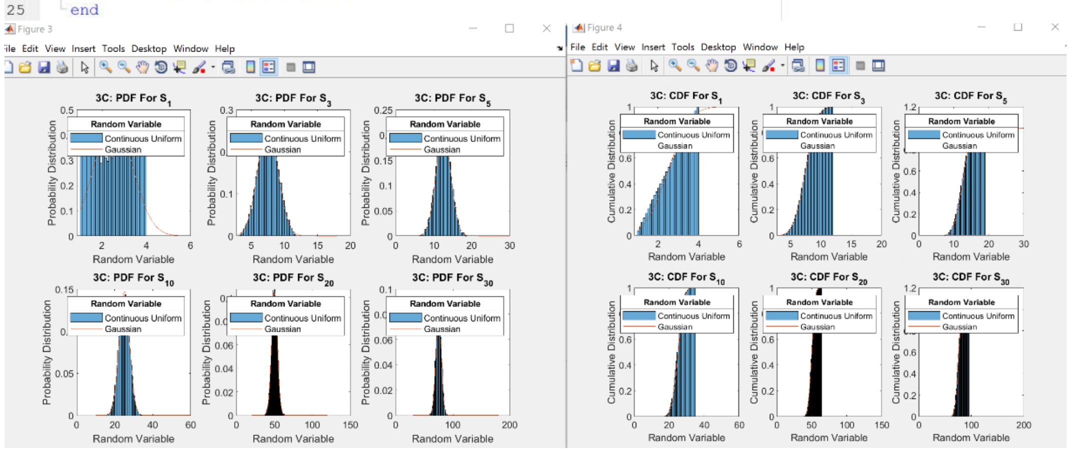
ECE131 : Winter 2019 Project

Chris Baker

105.180.929

```
n = [1 3 5 10 20 30];
figure(3); figure(4);
mu=[2.5 7.5 12.52 25.1 50.0 75.0];
sigmasqrd=[.75 2.31 3.68 7.35 14.68 22.17];
for i=1:6
    Xi=unifrnd(1,4,10000,n(i));
    Si=sum(Xi,2);
    figure(3),subplot(2,3,i)
    figure(3),histogram(Si,'Normalization','pdf')
    hold on
    figure(3),plot(1*n(i):0.01:6*n(i),normpdf(1*n(i):0.01:6*n(i),mu(i),sqrt(sigmasqrd(i))));
    xlabel("Random Variable"); ylabel("Probability Distribution");
    title(sprintf("3C: PDF For S_{%d}", n(i)));
    lgd=legend('Continuous Uniform','Gaussian');
    title(lgd,'Random Variable')

    figure(4),subplot(2,3,i)
    figure(4),histogram(Si,'Normalization','cdf')
    hold on
    figure(4),plot(1*n(i):0.01:6*n(i),normcdf(1*n(i):0.01:6*n(i),mu(i),sqrt(sigmasqrd(i))));
    xlabel("Random Variable"); ylabel("Cumulative Distribution");
    title(sprintf("3C: CDF For S_{%d}", n(i)))
    lgd2=legend('Continuous Uniform','Gaussian');
    title(lgd2,'Random Variable')
end
```

i.



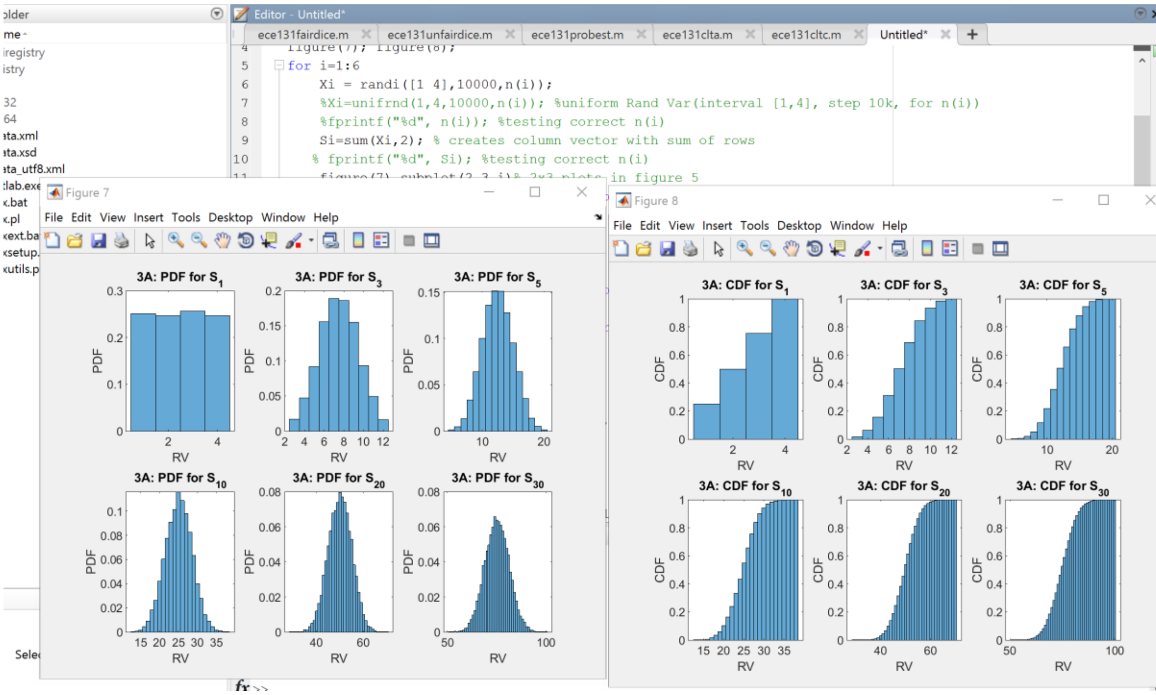
ii.

d. Fair die: The following code mirrors the earlier code, except for the noticeable substitutions at the top to represent the 4 side fair die

ECE131 : Winter 2019 Project

Chris Baker

105.180.929

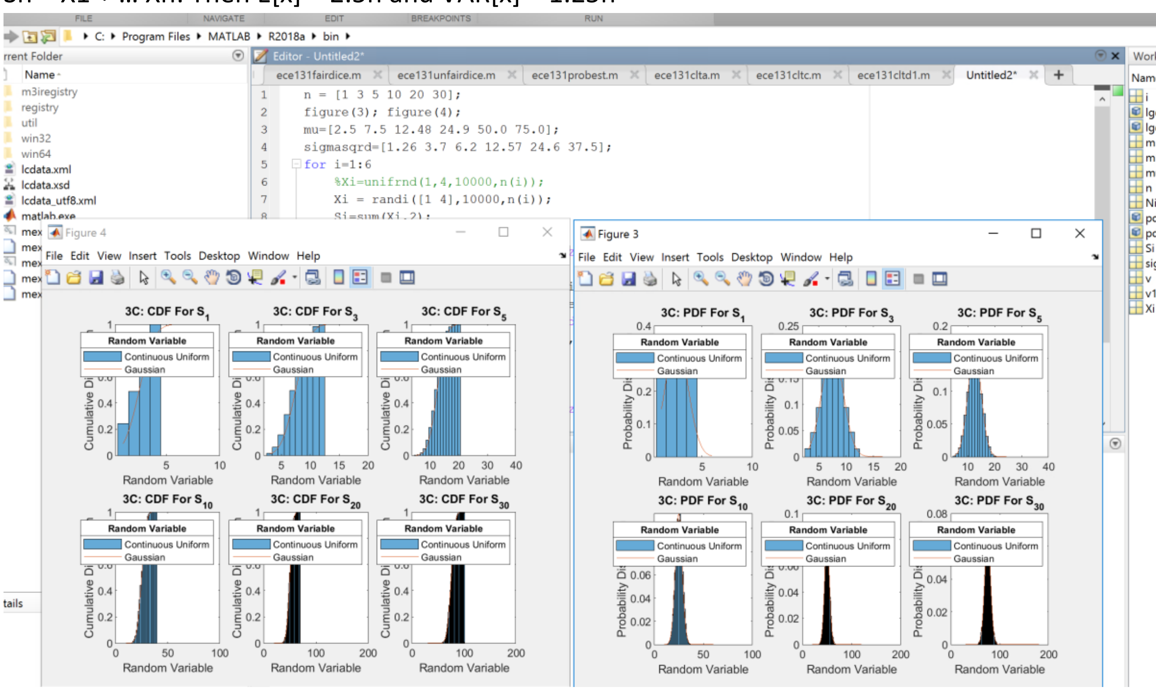


i.

ii. E[x] = .25(1 + 2 + 3 + 4) = 2.5

VAR[x] = .25[(1- 2.5)^2 + (2-2.5)^2 + (3-2.5)^2 + (4-2.5)^2] = 1.25

Sn = X1 + … Xn: Then E[x] = 2.5n and VAR[x] = 1.25n



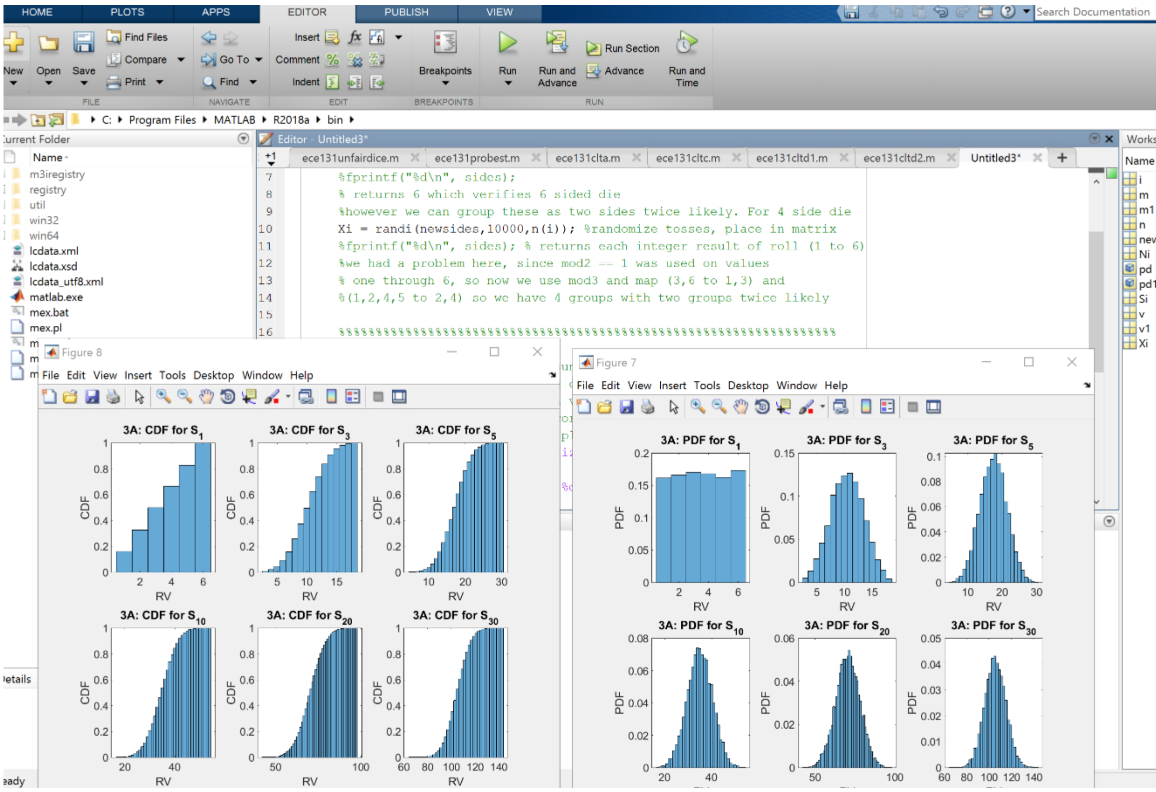
iii.

e. Unfair Die: Fair die: The following code mirrors the earlier code, except for the noticeable substitutions at the top to represent the 4 group biased die

ECE131 : Winter 2019 Project

Chris Baker

105.180.929



i.

ii. E[x] = (1/6)(1 + 2 + 2 + 3 + 4 + 4) = 2.67

VAR[x] = (1/6)[(1- 2.67)^2 + 2*(2-2.67)^2 + (3-2.67)^2 + 2*(4-2.67)^2] = 1.22

Sn = X1 + … Xn: Then E[x] = 2.67n and VAR[x] = 1.22n

ECE131 : Winter 2019 Project

Chris Baker

105.180.929

iii.