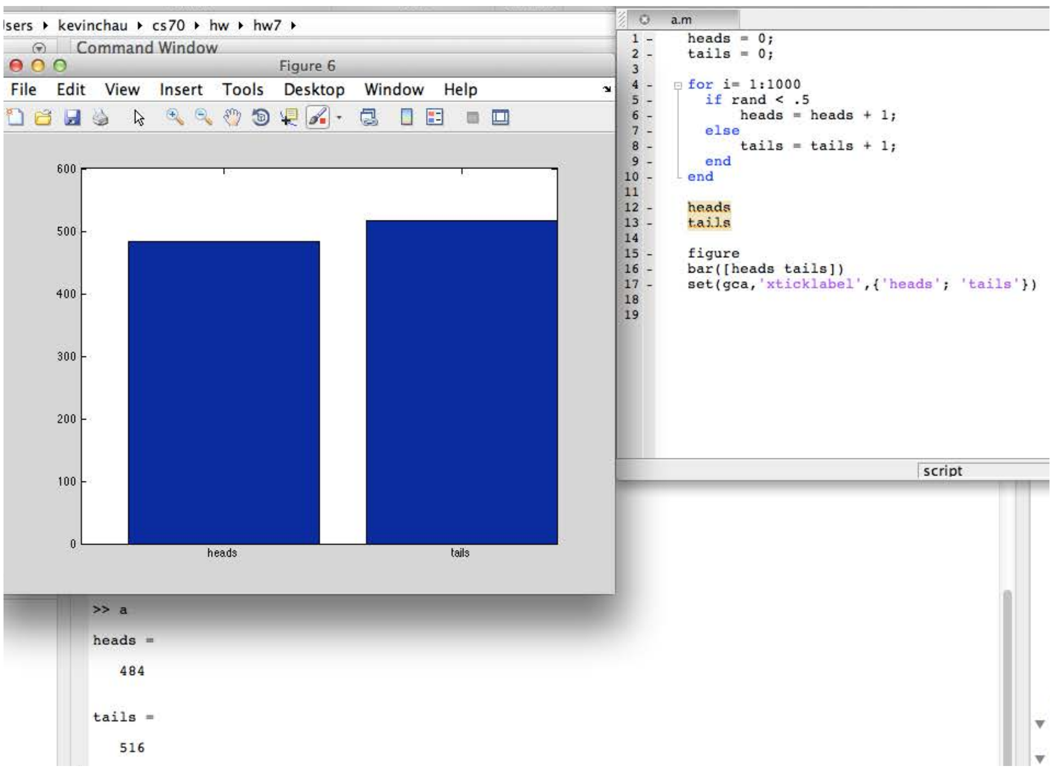
Kevin Chau
23816929
CS70 Homework 7
Partners: Samuel Drake, Ahdil Hameed, Unis Barakat

Problem 1

Code Instructions:

The code for each part is in a separate .m matlab file named after the part letter name. For example, part a is in a matlab file called a.m (all the way up to p.m). Just open each script in matlab and run to see any plots or data generated. The code from each file as well as any plots/figures that were generated when I ran it are included in the writeup below. In particular, I was using Matlab r2013a.

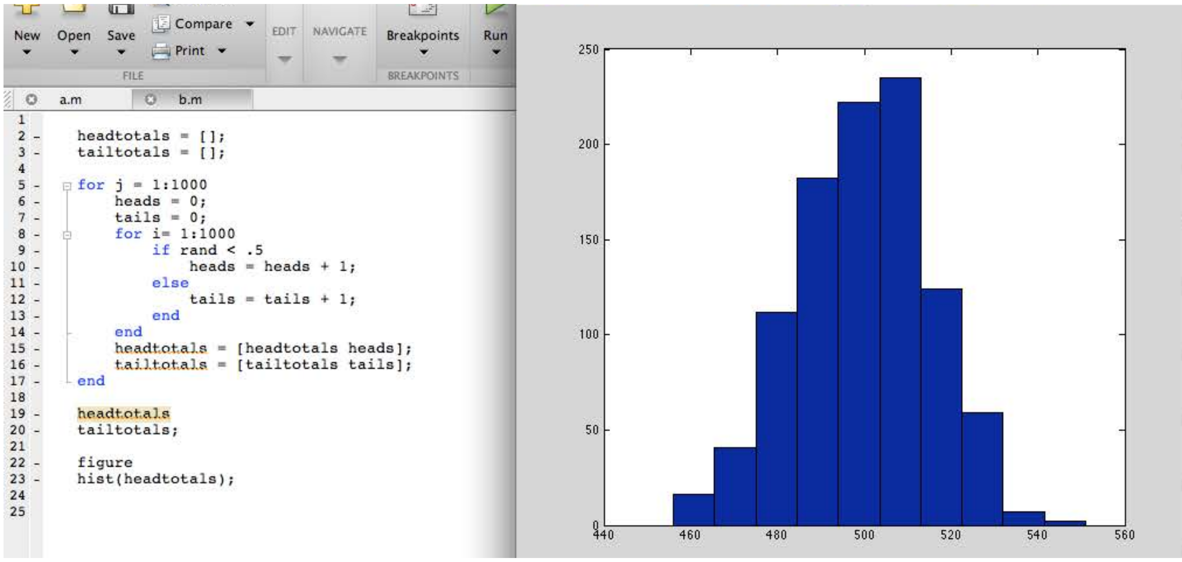
a) Use a random number generator to simulate a sequence of fair coin tosses. Do 1000



coin tosses. Plot a histogram of how many heads you got vs how many tails.

The rand function in Matlab returns a number in the interval (0,1) with an even distribution. I used the fact that theres a 50% chance that rand generates a number less than 1/2 and a 50% chance it will generate one greater than 1/2 in order to simulate a coin toss, where heads and tails have 50/50 chance. The histogram is just plotted with a bar graph, labeled heads and tails. In order to do 1000 tosses, I just used a for loop over the index “i”.

b) Do the previous part 1000 times. Plot a histogram of how many times you got N heads.

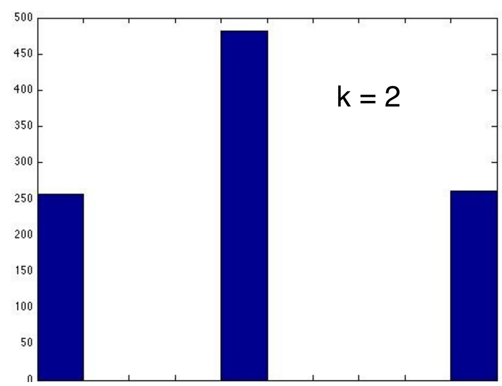


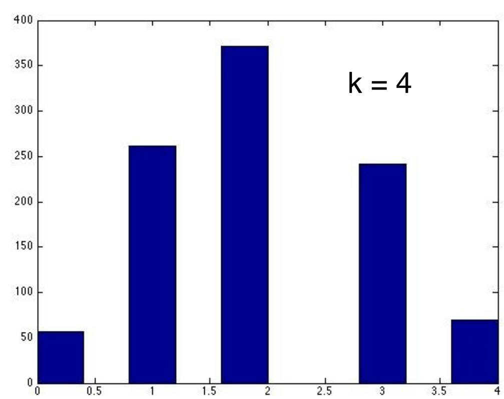
The histogram shows how many times I got N heads when I tossed 1000 coins 1000 times.The distribution is divided into intervals of 10 heads. The median number of heads was a little over 500 out of 1000, so even though a coin has a 50/50 chance of landing on either side, the result of tossing 1000 coins does not exactly match up with the probability (provided my simulation is a good model for a coin toss). 1000 is still too small of a number to get an even distribution.

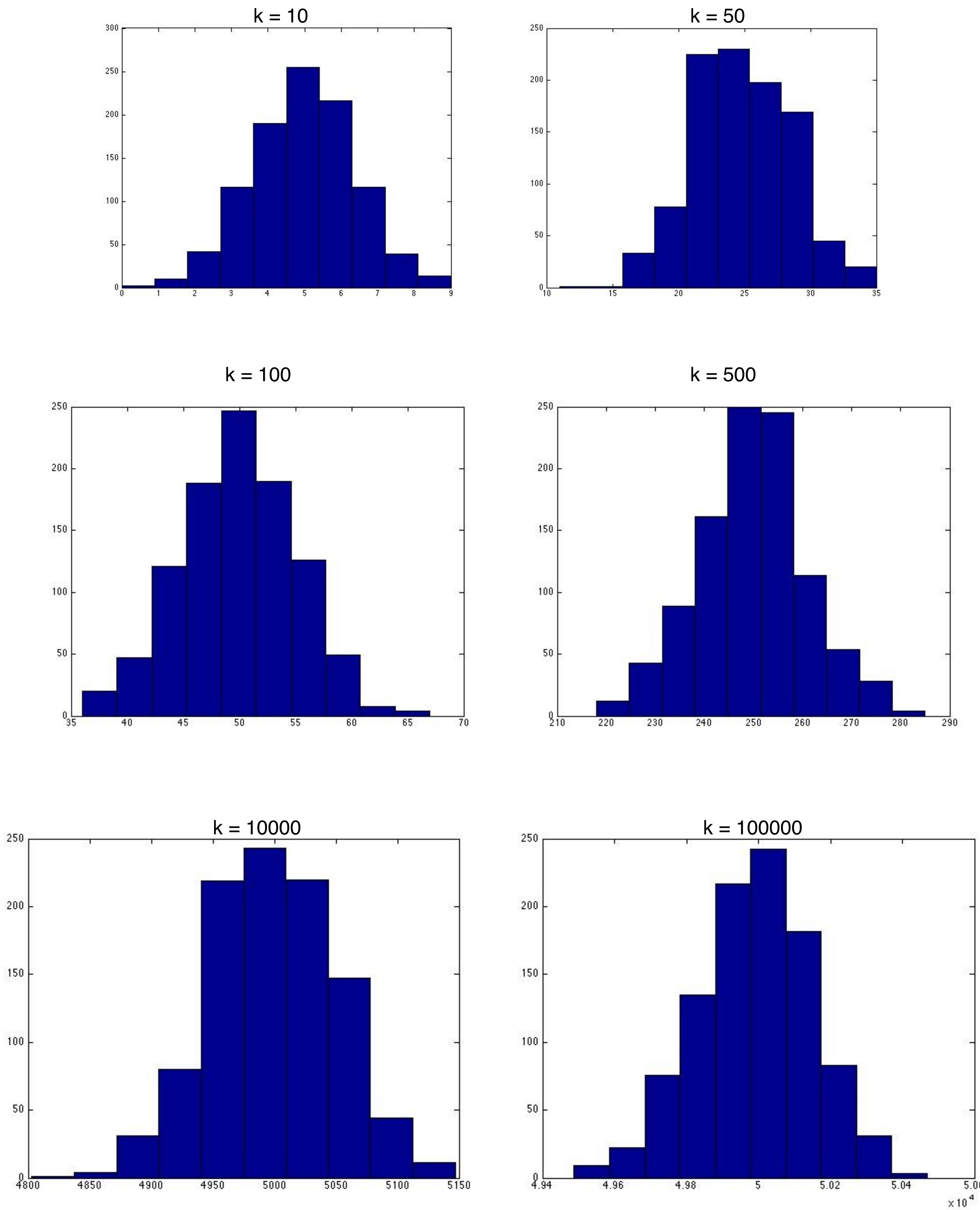
c) Consider the 1000 in part (a). Now, let that be a parameter k that tells how many coins you toss in one experiment. Do part (b) again for the following sequence of ks: 2, 4, 10, 50, 100, 500, 10000, 100000.

Code:

```
for k = [2 4 10 50 100 500 10000 100000];
 figure
 headtotals = [];
 tailtotals = [];
 for j = 1:1000
    heads = 0;
    tails = 0;
    for i= 1:k;
       if rand < .5
          heads = heads + 1;
       else
          tails = tails + 1;
       end
    end
    headtotals = [headtotals heads];
    tailtotals = [tailtotals tails];
 end
 headtotals;
 tailtotals;
 hist(headtotals);
 hold; end;
```







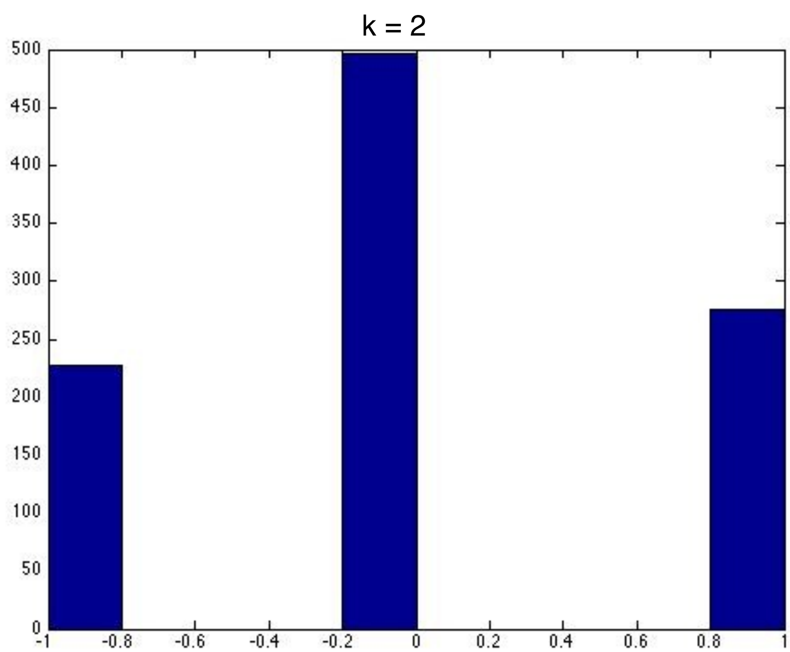
k = 10
k = 50
k = 100
k = 500
k = 10000
k = 100000
x 10^4

From the graphs in this section, we see that on an absolute x-scale, the higher the k value is, the wider the spread in N heads per run is. For example, k = 100000 has a spread of about 400 heads between the median and the rightmost side of the distribution, while k = 10000 only has a spread of about 150.

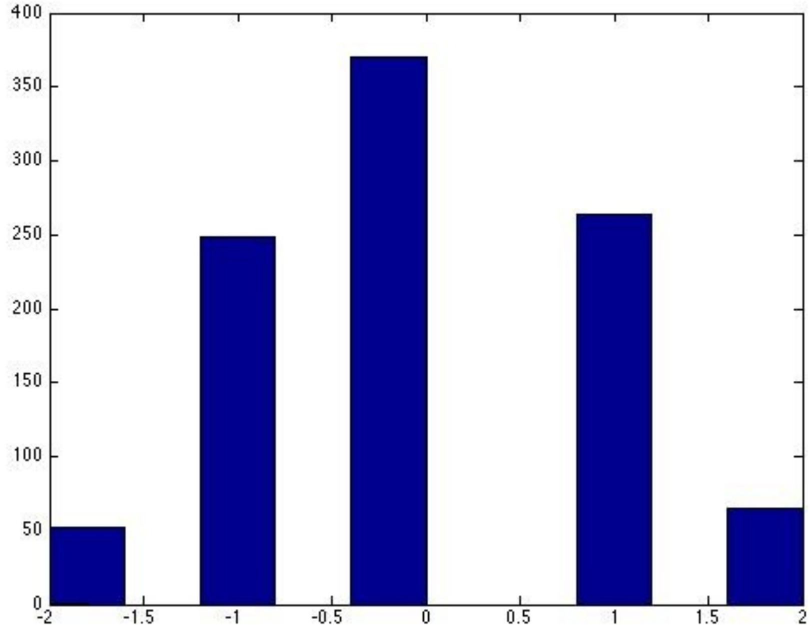
d) Notice that the horizontal axis has different scales as k varies. Suppose you wanted to "center" these histogram plots. To where would you move the origin as k varies? What is f(k) such that each N heads gets a shift to N-f(k). Place the histograms one above the other.

CODE:

```
for k = [2 4 10 50 100 500 10000 100000];
  figure
  headtotals = [];
  tailtotals = [];
  for j = 1:1000
      heads = 0;
      tails = 0;
      for i= 1:k;
          if rand < .5
              heads = heads + 1;
          else
              tails = tails + 1;
          end
      end
      headtotals = [headtotals heads - k/2];
      tailtotals = [tailtotals tails - k/2];
  end
  headtotals;
  tailtotals;
  hist(headtotals);
  hold;
```

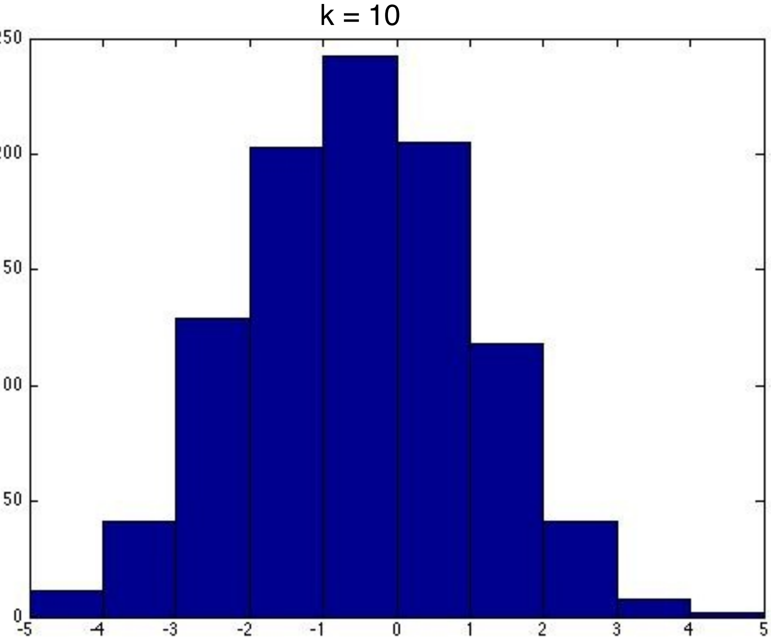


k = 4



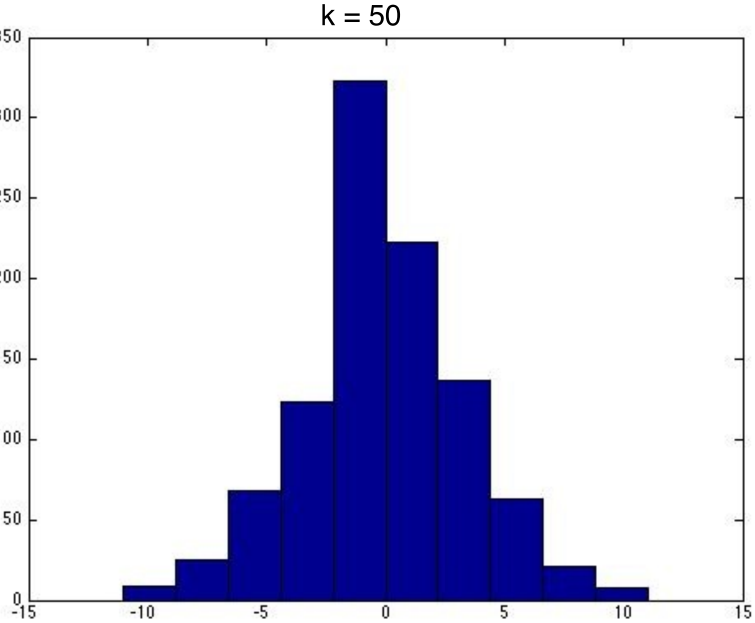
400
350
300
250
200
150
100
50
0
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

k = 10

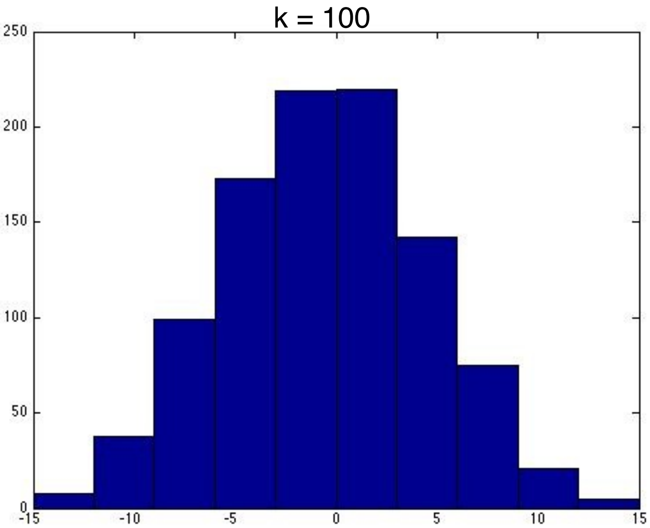


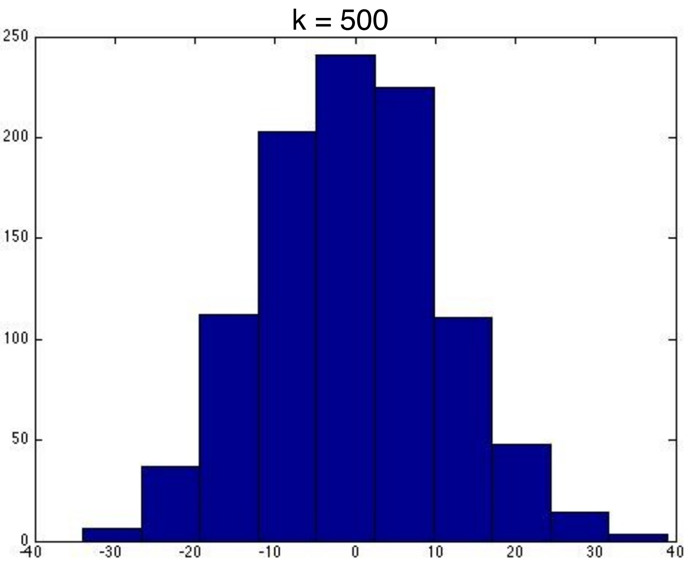
250
200
150
100
50
0
-5
-4
-3
-2
-1
0
1
2
3
4
5

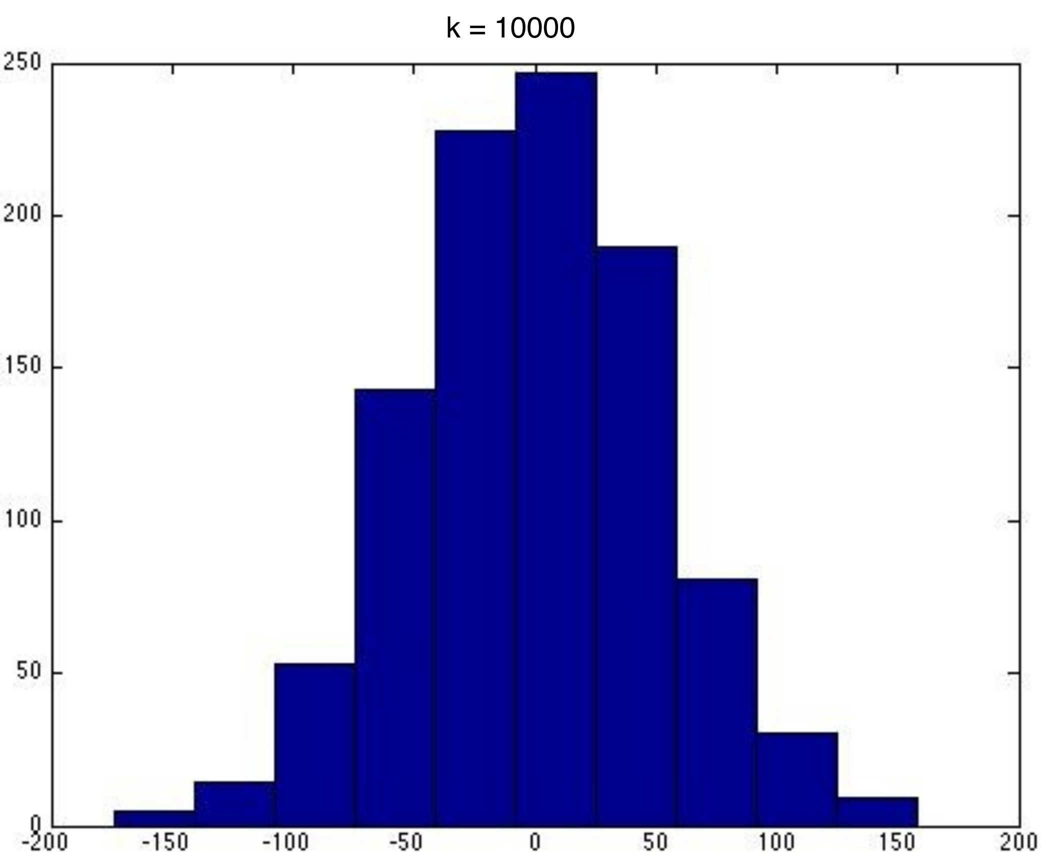
k = 50



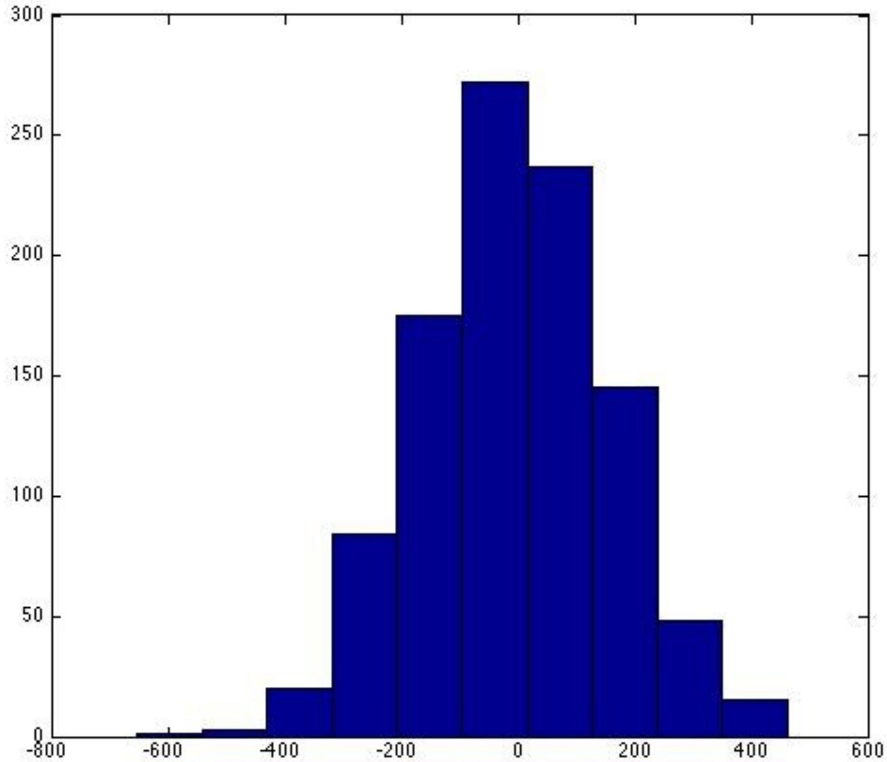
350
300
250
200
150
100
50
0
-15
-10
-5
0
5
10
15


k = 100
0
50
100
150
200
250
-15
-10
-5
0
5
10
15


k = 500
0
50
100
150
200
250
-40
-30
-20
-10
0
10
20
30
40


k = 10000
0
50
100
150
200
250
-200
-150
-100
-50
0
50
100
150
200

k = 100000



end

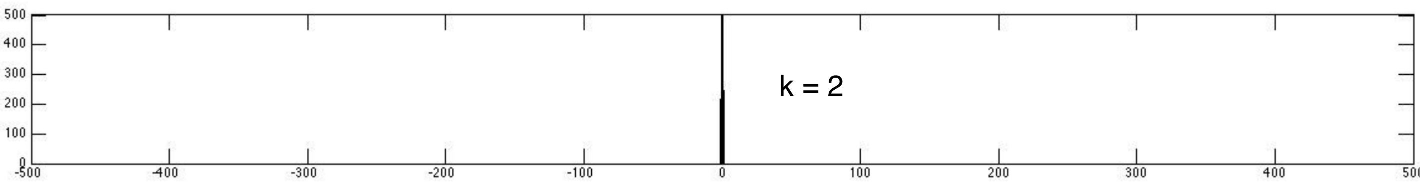
I shifted each histogram so that they were centered around the origin. So 0 corresponds to getting roughly the same amount of heads and tails, while being on the right means getting more heads, and being on the left means getting more tails. The shifting function f(k) that i chose to do this was f(k) = k/2, since we expect our origin to correspond to where we get heads half of k times.

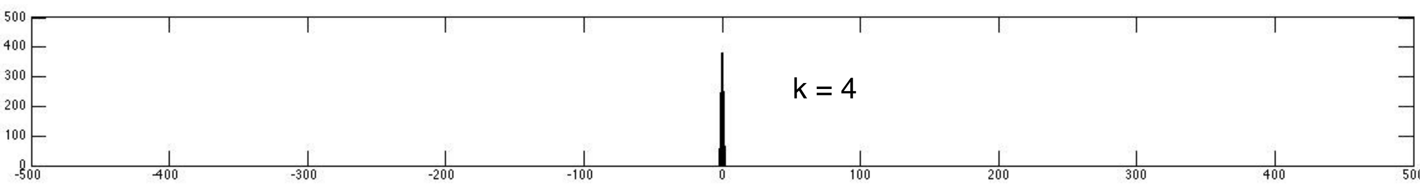
e) Repeat the plots of the previous part except this time, choose a common set of units — so one inch on the paper
should correspond to the same number of "ticks" of N. (e.g. one inch could correspond to 100. So the total
potential range for k = 100 would just be 1 inch while the potential range for k = 1000 would be 10 inches.)
Code:

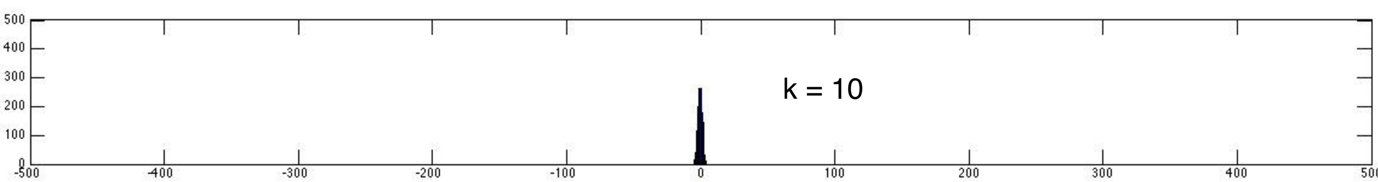
```
for k = [2 4 10 50 100 500 10000 100000];
  figure
  headtotals = [];
  tailtotals = [];
  for j = 1:1000
      heads = 0;
      tails = 0;
      for i= 1:k;
          if rand < .5
              heads = heads + 1;
          else
              tails = tails + 1;
          end
```

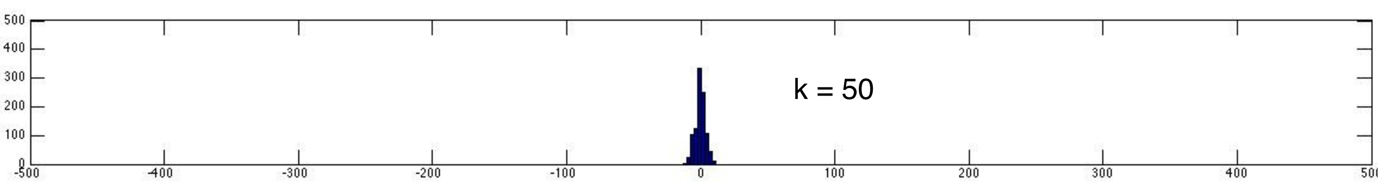
```
      end
      headtotals = [headtotals heads-k/2];
      tailtotals = [tailtotals tails-k/2];
  end
  headtotals;
  tailtotals;
  hist(headtotals,9);
  axis([-500 500 0 500]);
  hold;
end
```

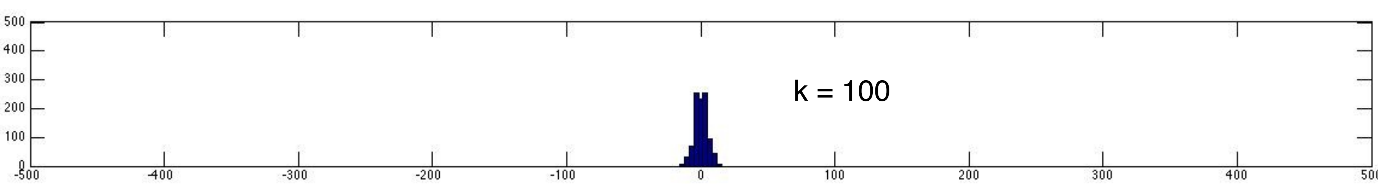
To do this part, I just used the axis function in matlab to make sure every histogram has the same relative units. So each graph is in the range of -500 to 500 for N—the smaller values of k will of course have much narrower distributions compared to the large values of k.

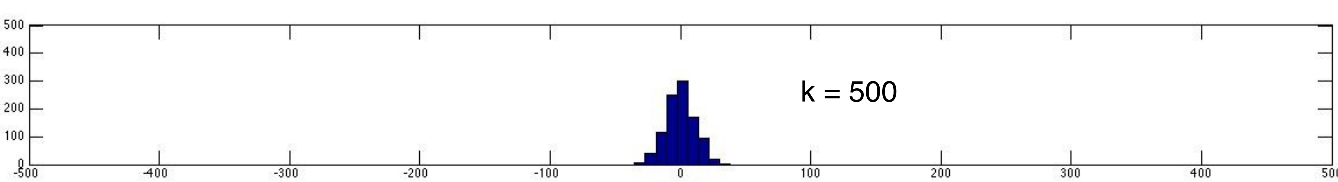


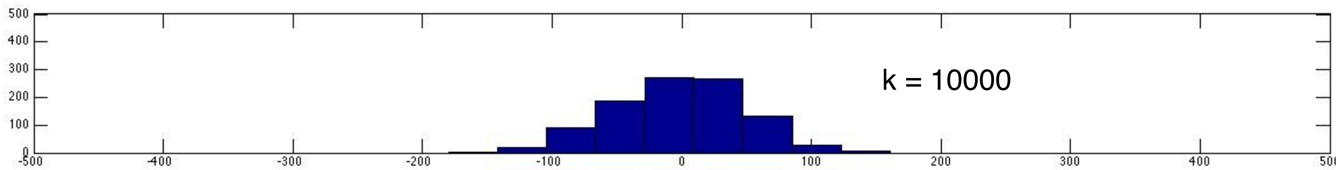


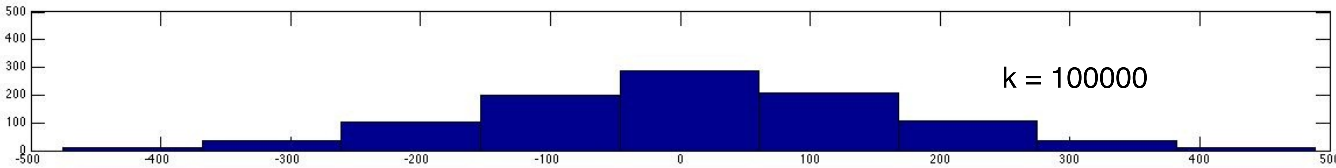










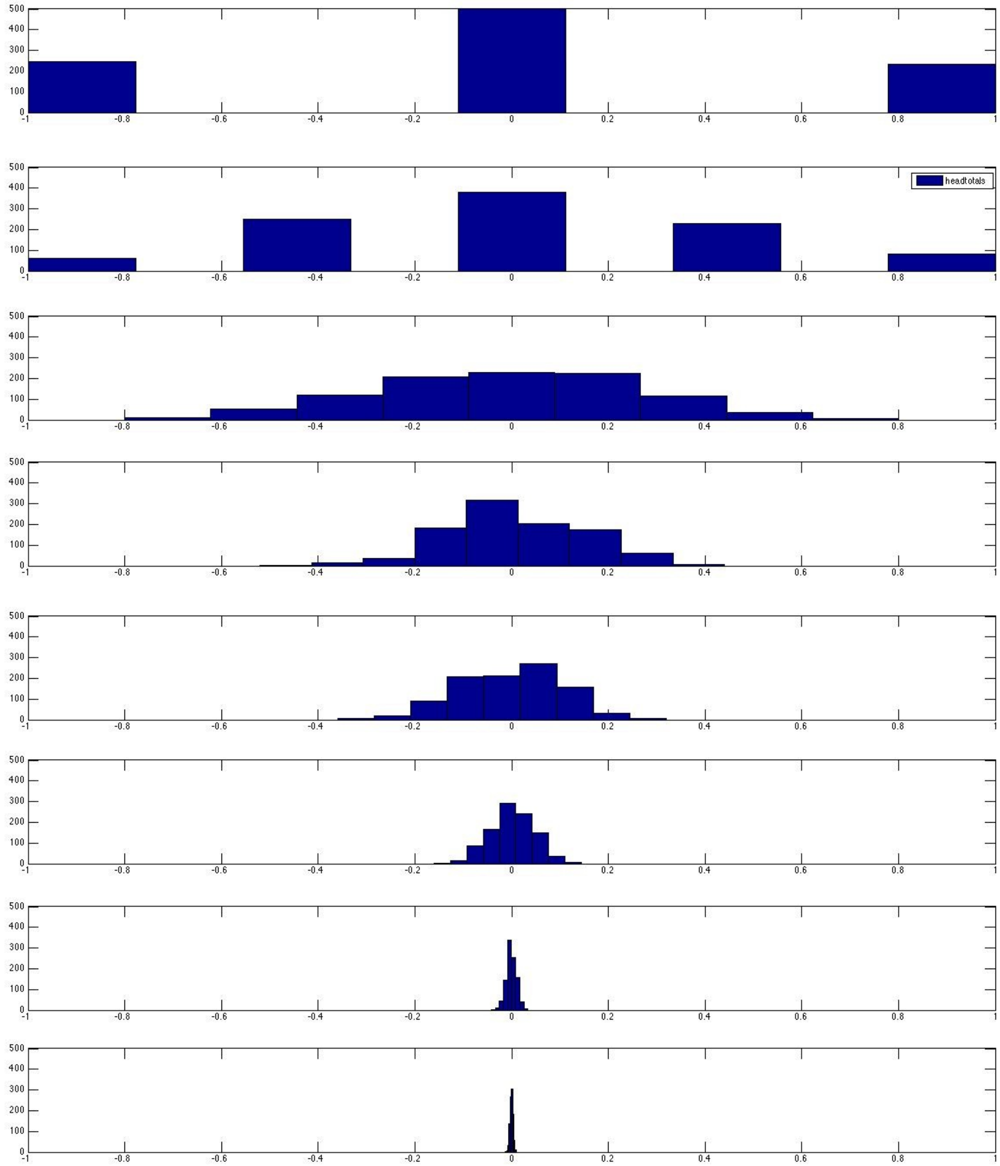


f) Repeat the plots of the previous part except this time, choose a normalized set of units. So the left-most point should correspond to the case of tossing all tails. And the right-most point should correspond to the case of tossing all heads. (this might never happen in your 1000 runs)

CODE:

```
for k = [2 4 10 50 100 500 10000 100000];
  figure
  headtotals = [];
  tailtotals = [];
  for j = 1:1000
      heads = 0;
      tails = 0;
      for i= 1:k;
          if rand < .5
              heads = heads + 1;
          else
              tails = tails + 1;
          end
      end
      headtotals = [headtotals (heads-k/2)/(k/2)];
      tailtotals = [tailtotals (tails-k/2)/(k/2)];
  end
  headtotals;
  tailtotals;
  hist(headtotals,9);
  axis([-1 1 0 500]);
  hold;
end
```

For this part, I normalized the x axis of the histogram by dividing the value of N after shifting the origin by k/2. This made it so that the very left side corresponding to -1 means all tails were flipped, and +1 means all heads were flipped.



The k values here are in the same order as the previous part.

g) Comment on what you observed in the three sets of plots you have seen above.

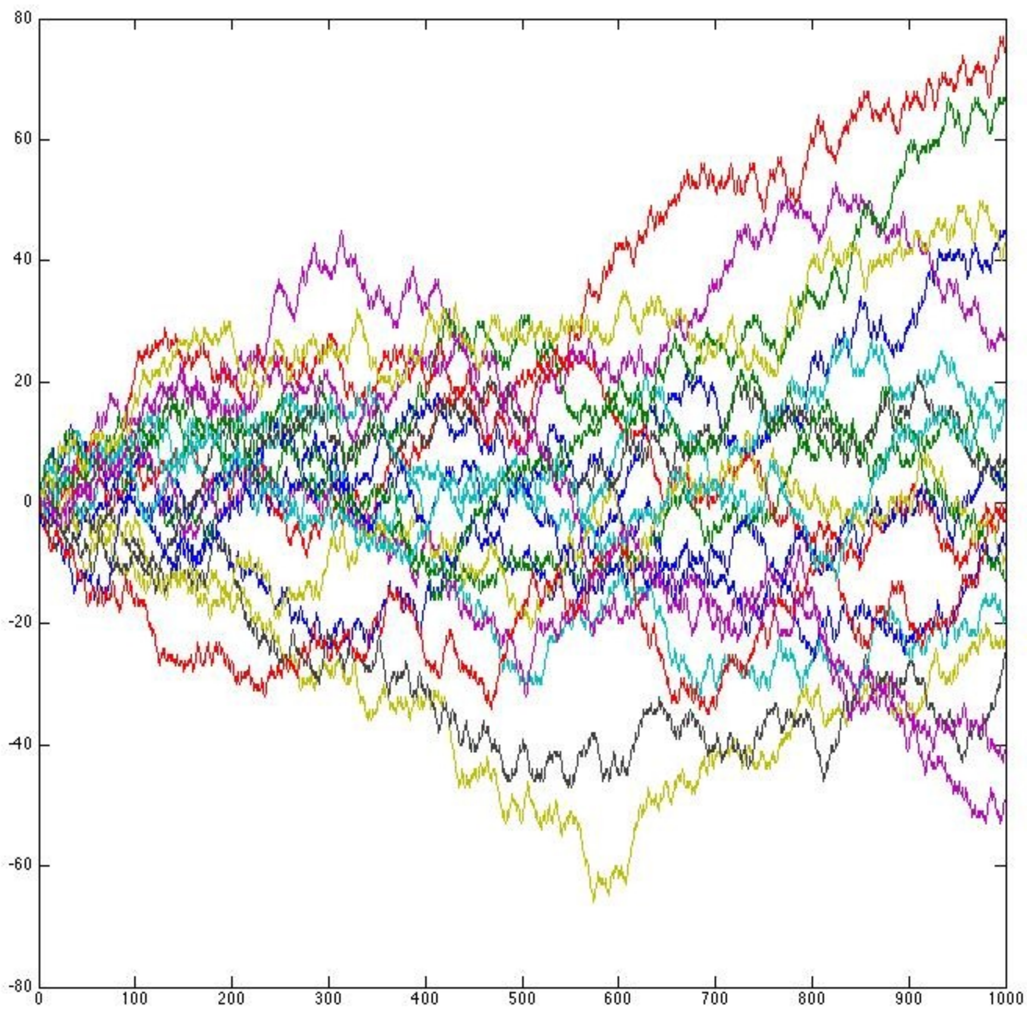
Notice that on the relative scale, larger k's seem to have less deviation from the origin, as indicated by the narrower peaks, while smaller k's tend to have -1 and +1 which much greater prevalence. This is opposed to the absolute scale, which shows that smaller k's have smaller deviations from the origin. However, the relative scale is actually much more informative because it shows that when you use large enough k's, we won't ever get any weird outcomes such as flipping all heads or flipping all tails. In other words, larger k means that the distribution starts matching up with the probability, despite the absolute value of the deviations growing bigger.

h) Now, we will change gears a little bit. Consider the following visualization of a sequence of coin flips. We start
at zero. For every head we get, we add one. For every tail we get, we subtract one. So a sequence of 1000 coin
tosses would be a path that starts at (0,0), and then goes to either (1,1) or (1,-1), and continues wandering till

(1000,y) somewhere. Plot 20 such paths on the same plot based on randomly flipped coins. Each sample path should have 1000 coin tosses.

CODE:

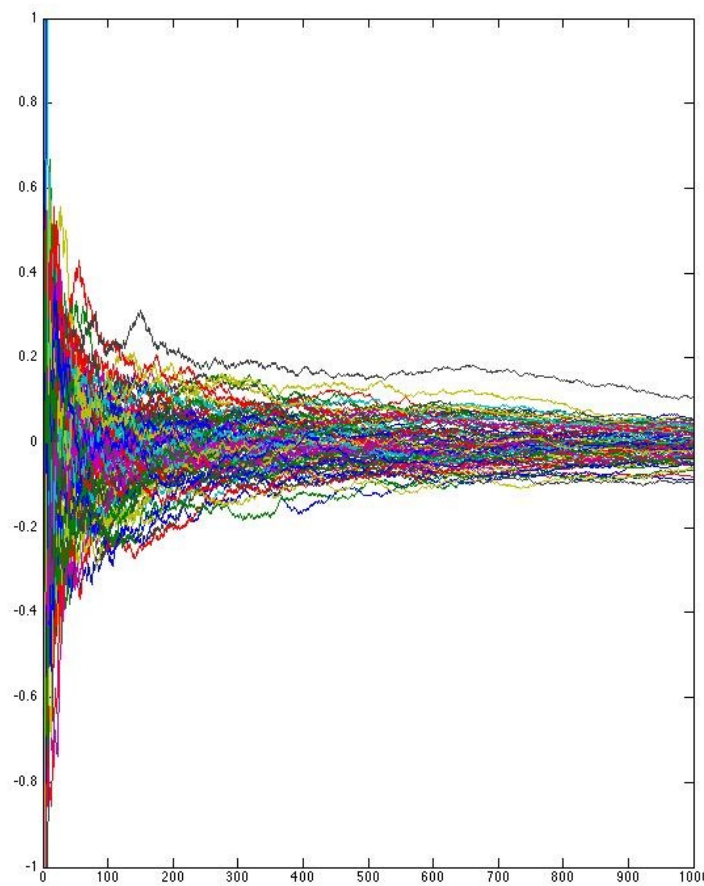
```
figure;
for j = 1:20;
cointotal = 0;
coinarray = [];
for i = 1:1000
   if rand < .5        %%heads
      cointotal = cointotal + 1;
      coinarray = [coinarray cointotal];
   else
      cointotal = cointotal - 1;
      coinarray = [coinarray cointotal];
   end
end
plot(1:1000,coinarray);
hold all;
end;
```



The plot shows 20 such runs of 1000 coin tosses. Notice that as we do more and more tosses, the likelihood that we end up far from the origin increases. In other words, if we do fewer tosses, we are more likely to have an even amount of heads and tails. However, at 1000, the difference between heads and tails isn't any worse than about 80, which isn't too terrible out of 1000 tosses.

Part i) Code:

```
figure;
for j = 1:100;
tosses = 0;
cointotal = 0;
coinarray = [];
for i = 1:1000;
   tosses = tosses + 1;
   if rand < .5        %%heads
      cointotal = cointotal + 1;
      coinarray = [coinarray cointotal/tosses];
   else
      cointotal = cointotal - 1;
      coinarray = [coinarray cointotal/tosses];
   end
end
coinarray
plot(1:1000,coinarray);
hold all;
end;
```



After normalizing part h, our graph reflects the observations we made from normalizing the histogram in part f. We see that when we toss less coins (smaller k), we are more likely to get outlier results such as all heads or all tails. When we toss a very large number of coins, we are closer to the origin, although the absolute size of the standard deviation increases.

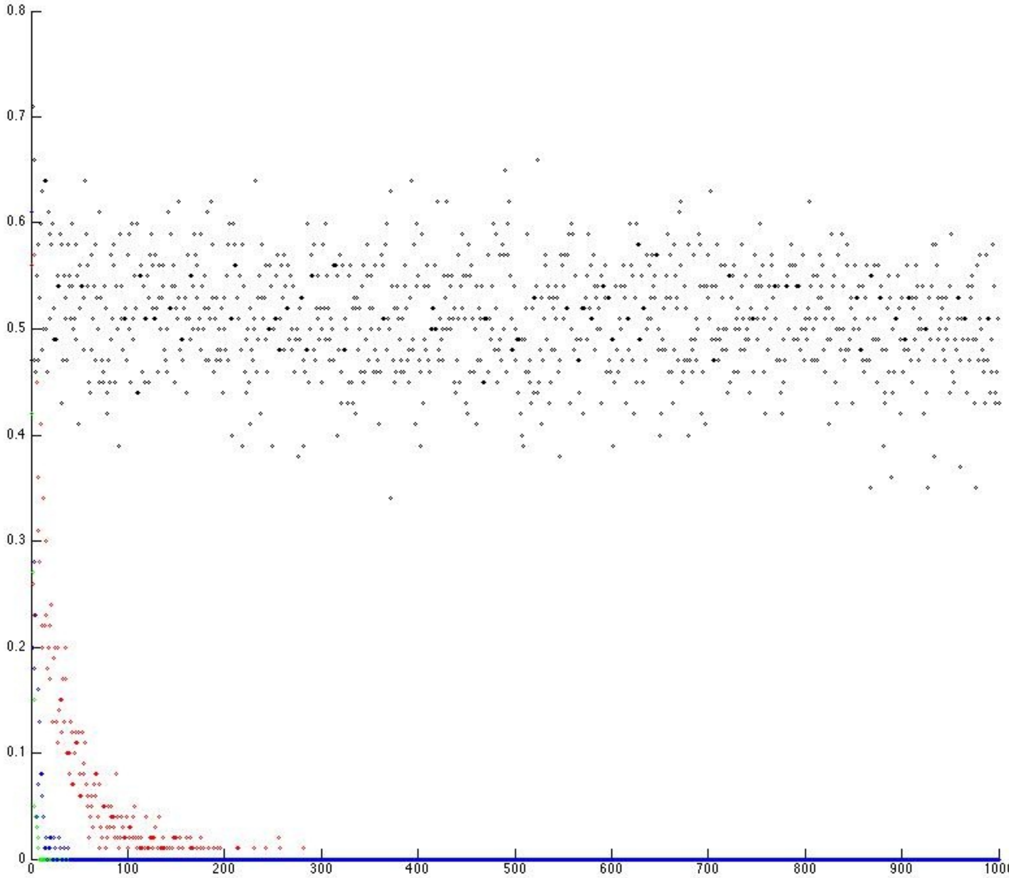
Part j) CODE:

```
figure;
q = .4;
m = 100;
for k = 1:1000;
frequency =0;
   for j = 1:m;
      heads = 0;
      headsfraction = 0;
      for i = 1:k;
         if rand < .5       %%heads
            heads = heads + 1;
         end
      end
      headsfraction = heads/k;
      if headsfraction <= q
         frequency = frequency + 1;
      end
   end
scatter(k,frequency/m,5,[1 0 0]);       %%red
hold all;
end

q = .1;
for k = 1:1000;
frequency =0;
   for j = 1:m;
      heads = 0;
      headsfraction = 0;
      for i = 1:k;
         if rand < .5       %%heads
            heads = heads + 1;
         end
      end
      headsfraction = heads/k;
      if headsfraction <= q
         frequency = frequency + 1;
      end
   end
scatter(k,frequency/m,5,[0 1 0]);   %%green
hold all;
end
```

```
q = .25;
for k = 1:1000;
frequency =0;
   for j = 1:m;
      heads = 0;
      headsfraction = 0;
      for i = 1:k;
         if rand < .5        %%heads
            heads = heads + 1;
         end
      end
      headsfraction = heads/k;
      if headsfraction <= q
         frequency = frequency + 1;
      end
   end
scatter(k,frequency/m,5,[0 0 1]);  %%blue
hold all;
end

q = .5;
for k = 1:1000;
frequency =0;
   for j = 1:m;
      heads = 0;
      headsfraction = 0;
      for i = 1:k;
         if rand < .5        %%heads
            heads = heads + 1;
         end
      end
      headsfraction = heads/k;
      if headsfraction <= q
         frequency = frequency + 1;
      end
   end
scatter(k,frequency/m,5,[0 0 0]);     %%black
hold all;
end
```

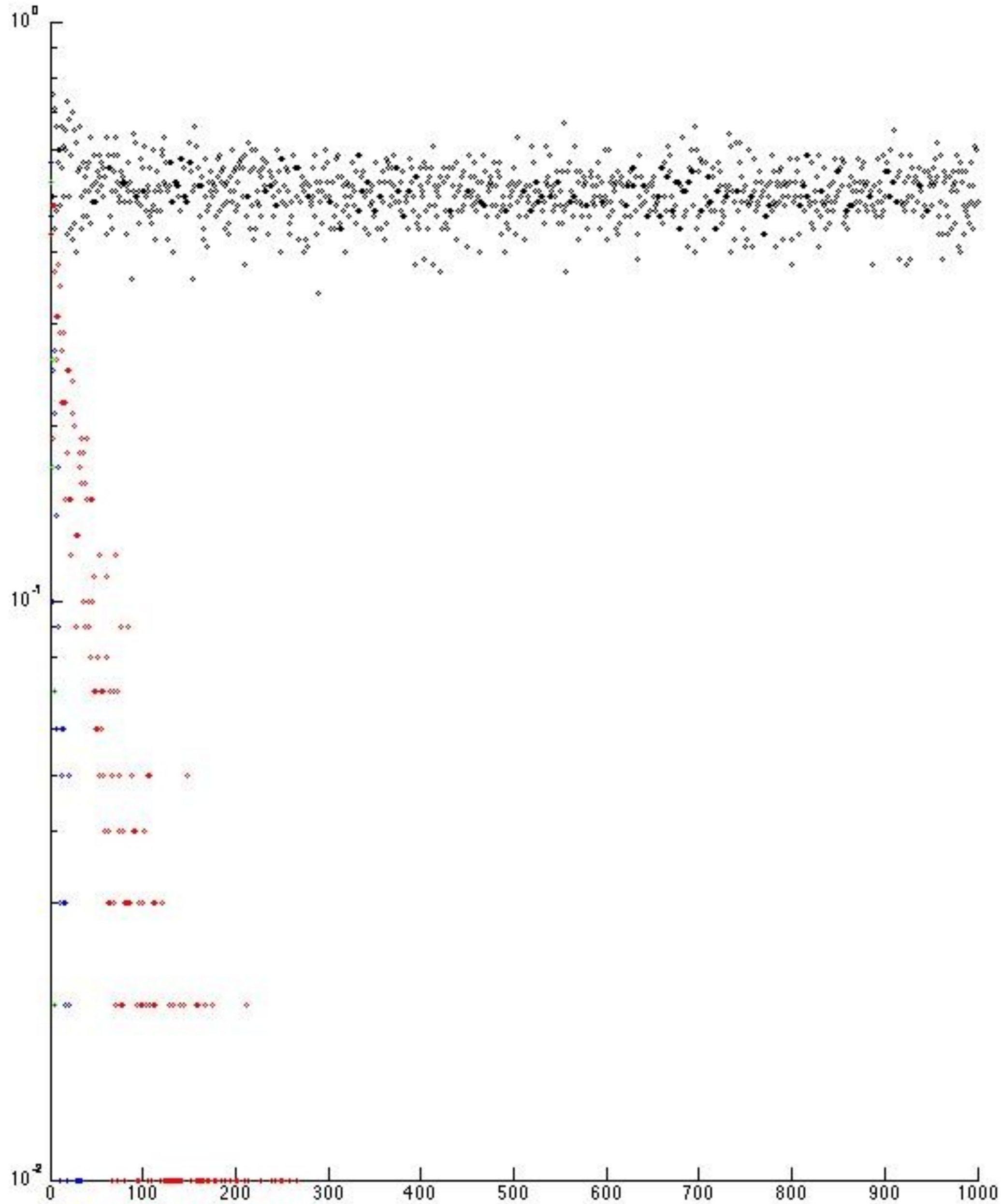


Note: I could only get the plot to generate when I made m a small number such as 100. Even with m = 1000, the computation time in matlab was so long that it didn't generate the plot. In theory, my code should work for m =10000 and will produce a plot; however given my time constraints i was unable to figure out whether there was a bug in my code or if there's a more efficient way to compute this part.

Part k) CODE:

```
figure;
q = .4;
m = 10000;
for k = 1:1000;
frequency =0;
   for j = 1:m;
      heads = 0;
      headsfraction = 0;
      for i = 1:k;
         if rand < .5        %%heads
            heads = heads + 1;
         end
      end
      headsfraction = heads/k;
      if headsfraction <= q
         frequency = frequency + 1;
```

```
      end
   end
scatter(k,frequency/m,5,[1 0 0]);       %%red
hold all;
end
set(gca,'YScale','log');
```



The only difference between this part and the previous part was that I used the set() function to change the Y scale to logarithmic. We see that the scatter plot spreads out more in the Y direction, and against a logarithmic axis the plots seems suggest straight lines (while on the linear axis they looked like decaying exponentials).

Problem 2

Claim: In $GF(p)$, there exists a polynomial P and Q (of any degree - from the last homework, any polynomial is equivalent to one of degree at most $p-1$) such that $PQ = 0$ over $GF(p)$.

Proof: We need to show there exists two polynomials P & Q such that when multiplied together, the resulting polynomial is zero at all points; however P and Q must not be zero at every point themselves. Working in $GF(p)$, I propose the following two polynomials work:

$$P = x^{p-1} + (p-1) \qquad Q = x$$

$$\text{so} \quad PQ = (x^{p-1} + p - 1)(x) = x^p + (p-1)x$$

By Fermat's little theorem, $x^p \equiv x \bmod p$, so

$$PQ \equiv x + px - x = px \equiv 0 \bmod p$$

since any number times p is a multiple of p, and hence congruent $0 \bmod p$. Therefore PQ is zero in $GF(p)$. We now need to show that P and Q are non zero. By Fermat's little theorem, $P = 1 + p - 1 = p \equiv 0 \bmod p$ for any $x \neq 0$ (which is why PQ is 0), but for $x = 0$, $P = 0 + p - 1 \equiv p - 1 \bmod p$, so $P(x)$ is actually non zero. $Q(x)$ is nonzero on all points $x \neq 0$, but at $x = 0$ $Q(x)$ is zero (which is why PQ is zero at $x=0$ despite $P(0)$ being non zero), hence $Q(x)$ is also non zero. Since P and Q are nonzero, but their product is zero in $GF(p)$, we have proven the original claim.

Problem 3

GF(11), K=2 errors (at most)

Assume the message is encoded into the values of the polynomial, not coefficients

a) (3, 0, 2, 0, 1, 1, 10)

For at most k=2 errors, there are 2k=4 reduncies => n=3

$E(x) = (x-e_1)(x-e_2) = x^2 - e_1x - e_2x + e_1e_2 = x^2 + b_1x + b_0$

$Q(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = P(x)E(x)$

Consider $Q(i) = r_iE(i)$ for $1 \le i \le n+2k$

$$\left.\begin{aligned}
Q(1) &= a_4 + a_3 + a_2 + a_1 + a_0 = 3(1+b_1+b_0)\\
Q(2) &= 16a_4 + 8a_3 + 4a_2 + 2a_1 + a_0 = 0\\
Q(3) &= 81a_4 + 27a_3 + 9a_2 + 3a_1 + a_0 = 2(9+3b_1+b_0)\\
Q(4) &= 256a_4 + 64a_3 + 16a_2 + 4a_1 + a_0 = 0\\
Q(5) &= 625a_4 + 125a_3 + 25a_2 + 5a_1 + a_0 = 25 + 5b_1 + b_0\\
Q(6) &= 1296a_4 + 216a_3 + 36a_2 + 6a_1 + a_0 = 36 + 6b_1 + b_0\\
Q(7) &= 2401a_4 + 343a_3 + 49a_2 + 7a_1 + a_0 = 10(49 + 7b_1 + b_0)
\end{aligned}\right] \bmod 11$$

$$\Rightarrow \left[\begin{array}{ccccccc|c}
1 & 1 & 1 & 1 & 1 & 8 & 8 & 3\\
5 & 8 & 4 & 2 & 1 & 0 & 0 & 0\\
4 & 5 & 9 & 3 & 1 & 5 & 9 & 7\\
3 & 9 & 5 & 4 & 1 & 0 & 0 & 0\\
9 & 4 & 3 & 5 & 1 & 6 & 10 & 3\\
9 & 7 & 3 & 6 & 1 & 5 & 10 & 3\\
3 & 2 & 5 & 7 & 1 & 7 & 1 & 6
\end{array}\right]$$

Using wolfram alpha to solve the system:

$a_4 = \frac{4891}{14623} \equiv \frac{7}{4} \equiv 3\cdot 7 \equiv 21 \equiv 10 \mod 11$

$a_3 = \frac{12238}{14623} \equiv \frac{6}{4} \equiv 3\cdot 6 \equiv 18 \equiv 7 \mod 11$

$a_2 = \frac{12970}{14623} \equiv \frac{1}{4} \equiv 3\cdot 1 \equiv 3 \mod 11$

$a_1 = \frac{-7713}{14623} \equiv 3\cdot 9 \equiv 27 \equiv 5 \mod 11$

$a_0 = -\frac{158813}{14623} \equiv 3\cdot 5 \equiv 15 \equiv 4 \mod 11$

$b_1 = \frac{4143}{2089} \equiv \frac{7}{10} = 70 \equiv 4 \mod 11$

$b_0 = -\frac{6464}{14623} = 3\cdot 4 \equiv 1 \mod 11$

$$\begin{array}{r}
10x^2 + 4 \\
x^2+4x+1 \overline{\smash{)}\,10x^4 + 7x^3 + 3x^2 + 5x + 4} \\
\underline{10x^4 + 40x^3 + 10x^2} \\
0 \quad 4x^2 + 5x + 4 \\
\underline{4x^2 + 16x + 4} \\
\underline{-11x} \\
0
\end{array}$$

$\Rightarrow Q(x) = 10x^4 + 7x^3 + 3x^2 + 5x + 4 \qquad E(x) = x^2 + 4x + 1$

From long division $P(x) = \frac{Q(x)}{E(x)} = 10x^2 + 4.$

$P(1) \equiv 3 \quad P(2) \equiv 0 \quad P(3) \equiv 6 \quad P(4) \equiv 10 \quad P(5) \equiv 1 \quad P(6) \equiv 1 \quad P(7) \equiv 10$

Original message: $\boxed{(3, 0, 6, 10, 1, 1, 10)}$ Errors at $x=3$ and $x=4$.

b) $(6,2,9,4,1,8) \longrightarrow k=2 \longrightarrow n=2 \longrightarrow n+2k=6$

$Q(x)=a_3x^3+a_2x^2+a_1x+a_0$

$E(x)=x^2+b_1x+b_0$

$Q(1)=a_3+a_2+a_1+a_0=6(1+b_1+b_0)$

$Q(2)=8a_3+4a_2+2a_1+a_0=2(4+2b_1+b_0)$

$Q(3)=5a_3+9a_2+3a_1+a_0=9(9+3b_1+b_0)$

$Q(4)=9a_3+5a_2+4a_1+a_0=4(5+4b_1+b_0)$

$Q(5)=4a_3+3a_2+5a_1+a_0=3+5b_1+b_0$

$Q(6)=7a_3+3a_2+6a_1+a_0=8(3+6b_1+b_0)$

$$=\left[\begin{array}{cccccc|c} 1 & 1 & 1 & 1 & 5 & 5 & 6 \\ 8 & 4 & 2 & 1 & 7 & 9 & 8 \\ 5 & 9 & 3 & 1 & 6 & 2 & 4 \\ 9 & 5 & 4 & 1 & 6 & 7 & 9 \\ 4 & 3 & 5 & 1 & 6 & 10 & 3 \\ 7 & 3 & 6 & 1 & 7 & 3 & 2 \end{array}\right] \Rightarrow \begin{array}{l} a_3=\frac{233}{203}\equiv\frac{2}{5}\equiv 9\cdot 2\equiv 7 \\ a_2=\frac{-41}{203}\equiv 9\cdot 3\equiv 5 \\ a_1=\frac{-216}{203}\equiv 9\cdot 4\equiv 3 \\ a_0=\frac{3592}{203}\equiv 9\cdot 6\equiv 10 \\ b_1=\frac{-71}{29}\equiv 8\cdot 6\equiv 4 \\ b_0=\frac{27}{203}\equiv 9\cdot 5\equiv 1 \end{array}$$

$\Rightarrow Q(x)=7x^3+5x^2+3x+10 \qquad E(x)=x^2+4x+1$

$$\begin{array}{r} 7x+10 \\ x^2+4x+1 \,\big)\, \overline{7x^3+5x^2+3x+10} \\ \underline{7x^3+28x^2+7x} \\ 10x^2+7x+10 \\ \underline{10x^2+40x+10} \\ 0 \end{array}$$

$P(x)=\frac{Q(x)}{E(x)}=7x+10$

$P(1)\equiv 6 \quad P(2)\equiv 2 \quad P(3)\equiv 9$

$P(4)\equiv 5 \quad P(5)\equiv 1 \quad P(6)\equiv 8$

Original message: $(6,2,9,5,1,8)$

Only $x=4$ had an error, although we protected against $k=2$ errors. $P(4)=5$ got corrupted to 4.

c) $(3,5,0,4,1,7,6)$ $k=2 \rightarrow n=3$

$Q(x)=a_4x^4+a_3x^3+a_2x^2+a_1x+a_0$ $E(x)=x^2+b_1x+b_0$

$Q(1)=a_4+a_3+a_2+a_1+a_0=3(1+b_1+b_0)$

$Q(2)=5a_4+8a_3+4a_2+2a_1+a_0=5(4+2b_1+b_0)$

$Q(3)=4a_4+5a_3+9a_2+3a_1+a_0=0$

$Q(4)=3a_4+9a_3+5a_2+4a_1+a_0=4(16+4b_1+b_0)$

$Q(5)=8a_4+4a_3+3a_2+5a_1+a_0=1(25+5b_1+b_0)$

$Q(6)=9a_4+7a_3+3a_2+6a_1+a_0=7(36+6b_1+b_0)$

$Q(7)=3a_4+2a_3+5a_2+7a_1+a_0=6(49+7b_1+b_0)$

$$=\left[\begin{array}{ccccccc|c} 1 & 1 & 1 & 1 & 1 & 8 & 8 & 3 \\ 5 & 8 & 4 & 2 & 1 & 1 & 6 & 9 \\ 4 & 5 & 9 & 3 & 1 & 0 & 0 & 0 \\ 3 & 9 & 5 & 4 & 1 & 6 & 7 & 9 \\ 9 & 4 & 3 & 5 & 1 & 6 & 10 & 3 \\ 9 & 7 & 3 & 6 & 1 & 2 & 4 & 10 \\ 3 & 2 & 5 & 7 & 1 & 2 & 5 & 8 \end{array}\right]$$

$a_4=\frac{-7031}{7987}\equiv\frac{9}{1}\equiv 1\cdot 9\equiv 9$

$a_3=\frac{8805}{7987}\equiv 5$

$a_2=\frac{-50481}{31948}\equiv\frac{9}{4}\equiv 3\cdot 9\equiv 5$

$a_1=\frac{10134}{7987}\equiv 3$

$a_0=\frac{269119}{31948}\equiv 3\cdot 2\equiv 6$

$b_1=\frac{-25189}{31948}\equiv 3\cdot 1\equiv 3$

$b_0=\frac{1943}{159.74}\equiv\frac{7}{2}\equiv 6\cdot 7\equiv 9$

$Q(x)=9x^4+5x^3+5x^2+3x+6$ $E(x)=x^2+3x+9$

$$\begin{array}{r|l} & 9x^2 \quad +1 \\ \hline x^2+3x+9 & 9x^4+5x^3+5x^2+3x+6 \\ & 9x^4+5x^3+4x^2 \\ \hline & \qquad\quad x^2+3x+6 \\ & \qquad\quad x^2+3x+9 \\ \hline & \qquad\qquad 8 \end{array}$$

remainder

By construction $Q(x)$ is divisible by $E(x)$ as long as $E(x)$ is really a degree $k=2$ polynomial. However, we received a message of more than 2 errors since we could not actually find a $P(x)=\frac{Q(x)}{E(x)}$. The number of errors is more than we can account for using the Berlekamp Welch algorithm.

Problem 4

a) message of n packets

Channel corrupts $0 \le f < \frac{1}{2}$ of the packets

consider $f = \frac{1}{4}$.

$n=4 \longrightarrow 1$ corrupted. Must send $4+2(1)=6$ packets

$n=8 \longrightarrow 2$ corrupted. Must send $8+2(2)=12$ packets

$n=12 \longrightarrow 3$ corrupted. $12+2(3)=18$ packets

$\frac{3}{2}n = n + \frac{n}{2} = n + 2(n)(\frac{1}{4})$ packets sent

consider $f = 1/3$

$$\left.\begin{array}{l} n=3 \rightarrow 1 \text{ corrupted. } 3+2(1)=5 \\ n=6 \rightarrow 2 \text{ corrupted. } 6+2(2)=10 \\ n=9 \rightarrow 3 \text{ corrupted. } 9+2(3)=15 \end{array}\right] n+2(n)(1/3)$$

For any f, there will be $fn=k$ errors. For k errors, we need $2k$ redundant packets, so we send $n+2fn$ packets total. The minimum number of additional packets is $\boxed{2fn}$.

b) $GF(p)$, with k transmission errors.

$a=(a_1,\dots,a_n)$ $b=(b_1,\dots,b_n)$ have n packets

$a+b=(a_1+b_1,\dots,a_n+b_n)$

Since a is length n, it gets encoded into a polynomial P_a of degree $n-1$, where $a_i = P_a(i)$. Message b is encoded into P_b with the same constraints. Now lets consider the message $a+b$ where the message is just made of the element wise sum of $a_i+b_i=(a+b)_i$. We know that this message is still length n, so it needs a $n-1$ degree polynomial to encode it; this polynomial P_{a+b} is such that $P_{a+b}(i)=(a+b)_i=a_i+b_i$. We can construct such a polynomial of degree $n-1$ by adding P_a+P_b. So at $x=i$, $P_{a+b}(i)=P_a(i)+P_b(i)$, which equals a_i+b_i. Hence we can construct a Reed Solomon encoding of the element wise sum message by just adding both polynomials together, and evaluating at the first n points.

We can go the other way by adding each a_i and b_i together, and making a point (i, a_i+b_i). We will have n such points, and they uniquely interpolate an $n-1$ degree polynomial, which is exactly just P_{a+b} at all points.

Assume the machine has error when inputing list of pairs

c) $\max(1, \lfloor n/4 \rfloor)$ outputs have errors.
Show that Reed-Solomon encoding can get rid of the errors. First, we split up the ordered pairs (x_i, y_i) for $1 \le i \le n$ input pairs into one long message of length $2n$ in the form of $[x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n] = Z$. So if we need to correct for $\lfloor n/4 \rfloor$ output errors, we really need to find and correct for $\lfloor n/4 \rfloor \times 2$ errors in the message Z. According to the Berlekamp Welch scheme, for k errors we need $2k$ redundant Reed-Solomon codeletters, so we need to add $2(2)\lfloor n/4 \rfloor = 4\lfloor n/4 \rfloor$ extra packets to message Z, for a total of $2n + 4\lfloor n/4 \rfloor$ packets that are sent to the machine. The Berlekamp Welch algorithm lets us find all x_i and y_i provided that the machine makes only $\max(1, \lfloor n/4 \rfloor)$ errors. We encode message Z in a $2n-1$ degree polynomial called P and take the first evaluations to make our Reed Solomon code letters, including the extra $4\lfloor n/4 \rfloor$ values. We can find the coefficients to $Q(x)$ and $E(x)$, and hence find $P(x)$ to rediscover the correct x_i and y_i values in message Z. We tell the machine to remake the ordered pairs by using the letters in Z: (Z_i, Z_{i+n}), where Z_i is the polynomial P evaluated at $P(i)$. So even if the machine has an error prone input channel, it can still error correct for all values of x_i and y_i and thus correctly compute the sums $x_i + y_i$ for $1 \le i \le n$.

Problem 5

secret number s between n parties.
Using only modular addition and subtraction, we can devise a simple secret sharing scheme like this: Choose a very large modulus number m (m can be any integer, but larger is better, and it does not have to be prime). Give the first person any number x_1 that is in the range $[0, m-1]$. Now give person 2 the number $x_2 \in [0, m-1]$ such that $x_1 + x_2 > m$, or $x_2 > m - x_1$. If person 1 and 2 add their numbers x_1 & x_2 together the number will be greater than m and be congruent to some number x_{12} that's in the range $[0, m-1]$. Now give person 3 a number x_3 such that $x_{12} + x_3 > m$. We can repeat this for all n people by giving person i a number x_i such that
$x_{(i-1)(i-2)} + x_i > m$ where $x_{(i-1)(i-2)}$ is congruent to $(x_{i-1} + x_{i-2}) \bmod m$. So each x_i can be any number in the range $[m - x_{(i-1)(i-2)}, m-1]$.
The secret number s is the result of adding all numbers x_i for $1 \le i \le n$, so $s \equiv \left(\sum_{i=1}^{n} x_i\right) \bmod m$.

s can only be recovered when all n people get together, and no partial gathering will be able to find s. Suppose only n-1 of the secret holders get together to find the secret by adding up their numbers to get some number z_{n-1} that is congruent to some number in $[0, m-1]$. So $z_{n-1} = \sum_{i=1}^{n-1} x_i$.
Now all n-1 people can add their numbers in any order, because modular addition is commutative and associative.
But in order to find s, the n-1 people

must know what number x_n to add to z_{n-1} because from the scheme $x_n + z_{n-1} \equiv S \bmod m$. Once the n+1 people know their number z_{n-1}, all they know is that x_n is some number that is in the range $[m - x_{n-1}, m-1]$ such that adding x_n to x_{n-1} wraps around the modulus number. So S really could be any number in $[0, m-1]$. Hence guessing what x_n is for the n-1 people is equivalent to guessing any number in $[0, m-1]$, and so n-1 people don't know any more about S than 1 person alone (1 person has the same chance guessing S as n-1 people guessing x_n). Hence only when n people are present can they precisely locate S. The larger the modulus m is, the harder it is for someone to brute force guess & check for the secret S. Of course, this scheme can be extended for subtraction (or a combination of subtraction and addition), as long as the subtracting number has an absolute value greater than the sum of all previous secret shares, so that the wrap around nature of working mod m is used.

Problem 6

1=k Erasure errors $\Longrightarrow$ Alice sends $n+1$ packets.

a) Suppose $n=1$, so Alice sends 2 packets to Bob. Consider the modulo number $N=12$. Since her message is determined by $n=1$ point, it is a $n-1=0$ degree polynomial — that is, it is just a constant horizontal line whose value is the message word at all points. So the polynomial is just $P(x)=m_1$, where m_1 is the message word. Alice can send any 2 packet list where both packets are the same and equal to the message word m_1 (she can send lists of the form (m_1, m_1), and there are 12 such lists). That way, if either is erased, Bob still knows what m_1 is. Therefore, the erasure correcting scheme will always work with $N=12$ and $n=1$.

b) Suppose $n=2$, so Alice sends 3 packets to Bob. There still wouldn't be any problems for our erasure correcting algorithm with $N=12$. Provided the channel only erases 1 packet, Bob can reconstruct the 1-degree polynomial / line with the two other packets.

c) $n=3 \Rightarrow$ Alice sends 4 packets. $N=12$

Show she can't send $(11, 6, 2)$.

$\Delta_1(x) = \dfrac{(x-2)(x-3)}{(1-2)(1-3)} = \dfrac{x^2-5x+6}{2} \equiv 2^{-1}(x^2+7x+6) \equiv ?$

If Bob receives all $n=3$ packets of Alice's message unerased and tries to interpolate the polynomial, he will be unable to, since 2 has no inverse in mod 12 — $\gcd(2,12)=2\neq 1$.

If $x=1$ gets erased but $x=2,3,4$ are preserved then we get the same division error for $\Delta_2(x)$:

$\Delta_2(x) = \dfrac{(x-3)(x-4)}{(2-3)(2-4)} = \dfrac{(x-3)(x-4)}{2}$ but 2 has no inverse

If the $x=2$ packet is lossed, Bob only gets $x=1,3,4$, and runs into a division error for $\Delta(x)$:

$$\Delta_1(x) = \frac{(x-3)(x-4)}{(1-3)(1-4)} = \frac{(x-3)(x-4)}{6}$$

but 6 has no inverse in mod 12.

Finally, if Bob loses $x=3$, he gets $x=1,2,4$, and there is an issue with $\Delta_1(x)$

$$\Delta_1(x) = \frac{(x-2)(x-4)}{(1-2)(1-4)} = \frac{(x-2)(x-4)}{3}$$

but 3 is not coprime with 12 and hence has no inverse.

So for $n=3$, there is no message Alice can send such that Bob can interpolate the message with 1 erasure error. Therefore she can't send (11, 6, 2), or (1, 2, 3), or (3, 2, 1), or any message for that matter.

d) Lagrange interpolation fails in part c because several $\Delta_i(x)$ functions have denominators that are not coprime with the modulus number N, so they aren't actually polynomials in the finite field $GF(N)$. Since the denomenators aren't coprime, there is no multiplicative inverse and so we can't actually divide. Lagrange interpolation does not fail for part a because no $\Delta_i(x)$ function for $n=1$ has a non coprime denominator — in fact, the only function is $\Delta_1(x) = x$ which has no denominator at all. Lagrange interpolation doesn't fail for part b for the same reason. With $n=2$, $\Delta_1(x) = \frac{(x-2)}{(1-2)} = \frac{x-2}{-1} \equiv \frac{x-2}{11}$

Since 11 is coprime with 12, Bob can always interpolate.

e) For any n that Alice chooses, Bob will not always be to recover the message because the denominators of the Δ_i Functions for lagrange interpolation are not guaranteed to be coprime with $N=12$, depending on where the erasure error is. A counterexample to the claim in this part is for $n=3$. When $n=3$, Alice can't send any message to Bob such that Bob can interpolate the polynomial

Problem 7

Each person inputs n commands. 1000 players
$A=1$ $B=2$ $L=3$ $R=4$ $U=5$ $D=6$ $\text{Start}=7$ $\text{Select}=9$

a) In a given time interval, each person inputs n commands. There are 1000 people, so every time step There are $n \times 1000 = \boxed{1000n}$ commands sent to the server.

b) Since the troll can <u>corrupt</u> (not erase) up to $k<n$ errors, the players must protect for up to k general errors. Using Reed-Solomon codes, this means they need to send $2k$ redundant packets after the original n commands. So each person sends $n+2k$ commands per turn. For a total of:

$$1000(n+2k) = \boxed{1000n + 2000k}$$

commands sent to the server each time step.

c) The troll can make up to k errors, but if 1 error invalidates a chain of n commands, each player only really needs to protect against 1 error. Even if the troll corrupts greater than 1 packet while the players only budget for 1, the moderators can detect that the troll corrupted at least 1 packet because they will be unable to solve for $p(x)$ after finding the coefficients for $Q(x)$ and $E(x)$. The mods assume $Q(x)$ is n degree while $E(x)$ is a <u>1</u> degree. If the troll didn't make any errors in the packets <u>at all</u>, then the moderators will be able to find $P(x)$ and evaluate at the first n points and see that no corruptions were made. If the troll made exactly 1 error, the moderaters will still be able to find $P(x)$ (since they can correct for $k=1$ error) and find the error location exactly (so they can simply detect it as well). Hence each player needs to protect against 1 error in order for the mods to detect at least 1 corruption from the troll. From the Berlekamp Welch algorithm, the players need to send $2k$ additional packets. with $k=1$, each player sends $n+2k=n+2$ packets. Therefore each time interval there are at most $\boxed{1000(n+2)}$ commands sent for the error dectection moderator scheme. We proved by cases that $k=1$ lets our moderators detect when the troll makes at least 1 error.

Note: In lecture, we proved that provided there are no more than k errors, the Berlekamp Welch algorithm guarantees that $E(x)$ will divide $Q(x)$ so we can find $P(x)$. If there are more errors than k, $Q(x)/E(x)$ will have a remainder.

Problem 8

Problem: Extend problem 4c by considering an improved computation machine that takes in a list of 1D arrays of length ℓ and outputs a list of the products of the elements in each array. However, the machine only tends to make erasure errors in a very particular way: it turns all the erased array values into 1. If we want to protect against k output errors, how can we use Reed Solomon codes to do so?

$\{(1,2,3),(3,4,5)\}$
$\Downarrow$
$\{6, 60\}$

Solution: We can unpack each array and make one long message of length $\ell \cdot n$ (where n is the output list length) and gaurd against ℓk erasure errors. For erasure errors, we only need to send $2\times$#errors $= 2\ell k$ extra packets for our machine to be able to recover all $\ell \cdot n$ elements of the long message. The machine can then recover all $l \cdot n$ packets even with k output errors, and therefore can regroup them into arrays of length l. Then the machine can remultiply the arrays, and check its output with errors against the corrected output.

It is worth taking note of the fact that this machine automatically tells you when it has made an input channel erasure error because the output list will contain a 0 anywhere an array got corrupted, provided all input arrays don't contain zero already. The machine can use this fact to increase efficiency by only using the error correction comparison when its output is ~~[illegible]~~ has a zero in the list — otherwise it does not have to run the correction algorithm at all.