lab8sol

October 27, 2014

1 Virtual Lab 8 Solution: Intro to Randomness

1.0.1 EECS 70: Discrete Mathematics and Probability Theory, Fall 2014

Due Date: Monday, October 27th, 2014 at 12pm **Name**:
Login: cs70-
Instructions:

- **Please fill out your name and login above.**
- **Please leave your answers in the Markdown cells, marked with `"YOUR COMMENTS HERE"`.**
- **Complete this lab by filling in all of the required functions, marked with `"YOUR CODE HERE"`.**
- If you plan to use Python, make sure to go over *Tutorial 1: Introduction to Python and IPython* and *Tutorial 2: Plotting in Python with Matplotlib* before attempting the lab.
- Make sure you run every code cell one after another, i.e. don't skip any cell. A shortcut for doing this in the notebook is Shift+Enter. When you finish, choose 'Cell > Run All' to test your code one last time all at once.
- Most of the solution requires no more than a few lines each.
- Please do not hardcode the result or change any function without the `"YOUR CODE HERE"` mark.
- Questions? Bring them to our Office Hour and/or ask on Piazza.
- Good luck, and have fun!

1.1 Table of Contents

The number inside parentheses is the number of functions or code blocks you are required to fill out for each question. Always make sure to double check before you submit.


```
In [1]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: from __future__ import division  # so that you don't have to worry about float division
        import random
        import math
```

Introduction

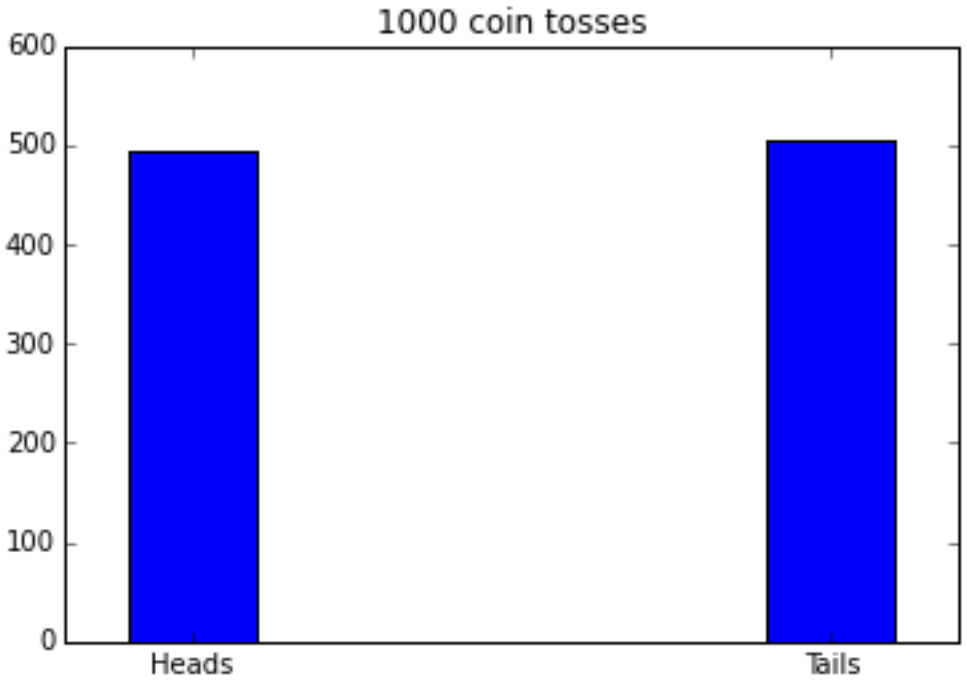
We now have all the tools to start diving into the world of randomness. The following questions are designed to give you some intuition about concepts in probability. Some of the powerful ideas behind the plots that we will make will be further explored later in the course.

Below is the sample solution for the last question in last week's lab. You might find it helpful for completing this week's lab.

```
In [3]: def count_heads(k):
            """
            Counts the number of heads in num_flips
            """

            return sum([round(random.random()) for _ in range(k)])
```

```
In [4]: heads = count_heads(1000)
        plt.bar([0, 1], [heads, 1000 - heads], width=0.2, align="center")
        plt.xticks([0, 1], ["Heads", "Tails"])
        plt.title("1000 coin tosses")
```

```
Out[4]: <matplotlib.text.Text at 0x978b7ec>
```



Part (a): N Heads

Last week, we did 1000 coin tosses and plotted a bar chart of how many heads we got v.s. how many tails. This week, we will do it again 1000 times. Plot a histogram of how many times you got N heads, where $0 \leq N \leq 1000$. What do you observe about N and its frequency?

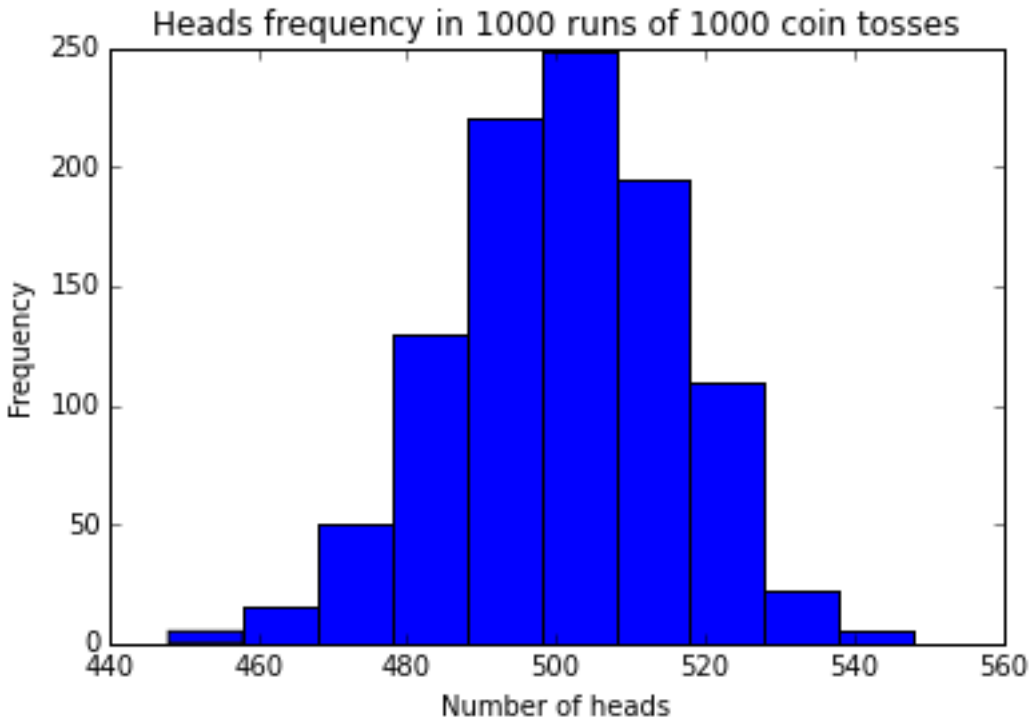
Hint: Implement the function `count_heads_in_runs(k, n)` in the skeleton, which returns a list of length n, where each element is the number of heads in k flips. In this case, $k = n = 1000$.

(In general, always try to parameterize the values instead of hard-coding it. This will make your code reusable for later parts.)

```
In [5]: def count_heads_in_runs(k, n):
            """
            Returns a list of length n, where each element is the number
            of heads in k flips.

            YOUR CODE HERE
            """

            return [int(count_heads(k)) for _ in range(n)]
```

```
In [6]: def partA():
            """
            YOUR CODE HERE
            """

            plt.hist(count_heads_in_runs(1000, 1000))
            plt.title("Heads frequency in 1000 runs of 1000 coin tosses")
            plt.xlabel("Number of heads")
            plt.ylabel("Frequency")
            plt.show()
```

```
In [7]: partA()
```



YOUR COMMENTS HERE:

Part (b): k Coins

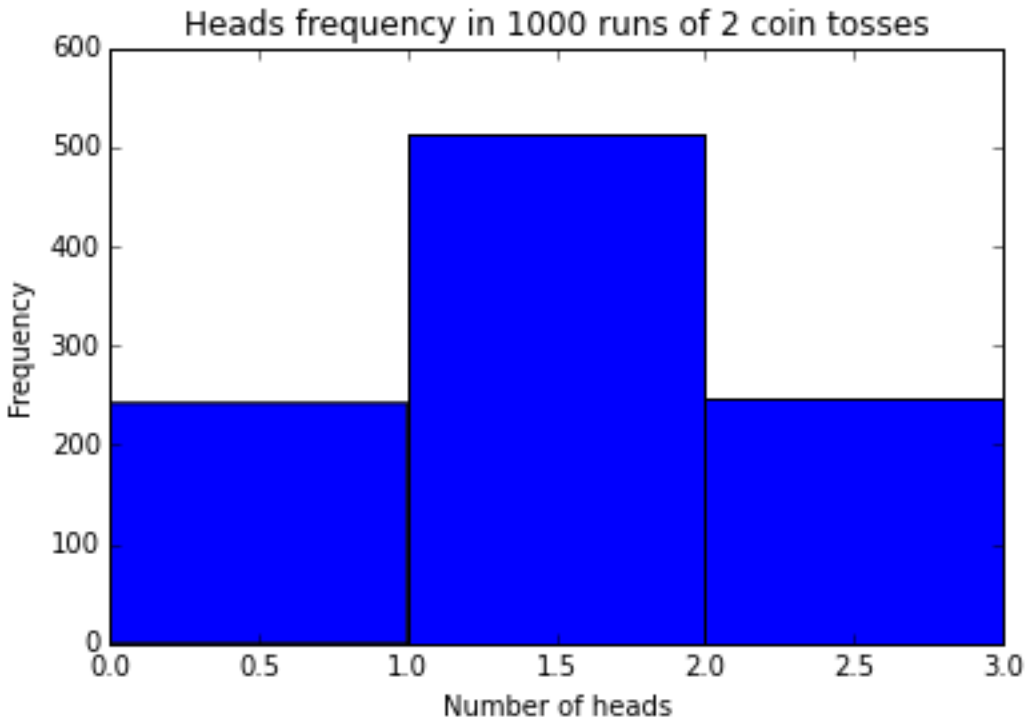
Let k be a parameter that tells how many coins you toss in one experiment.

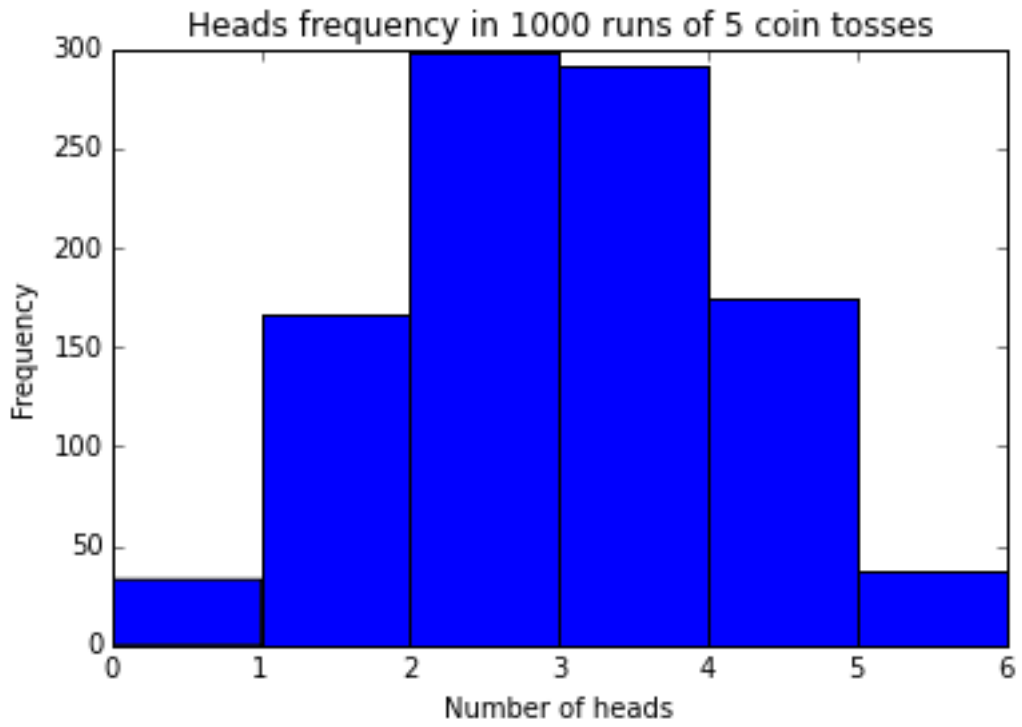
Do part (a) again for the following sequence of ks: 2, 5, 10, 100, 1000, 10000. What do you observe as k gets larger?

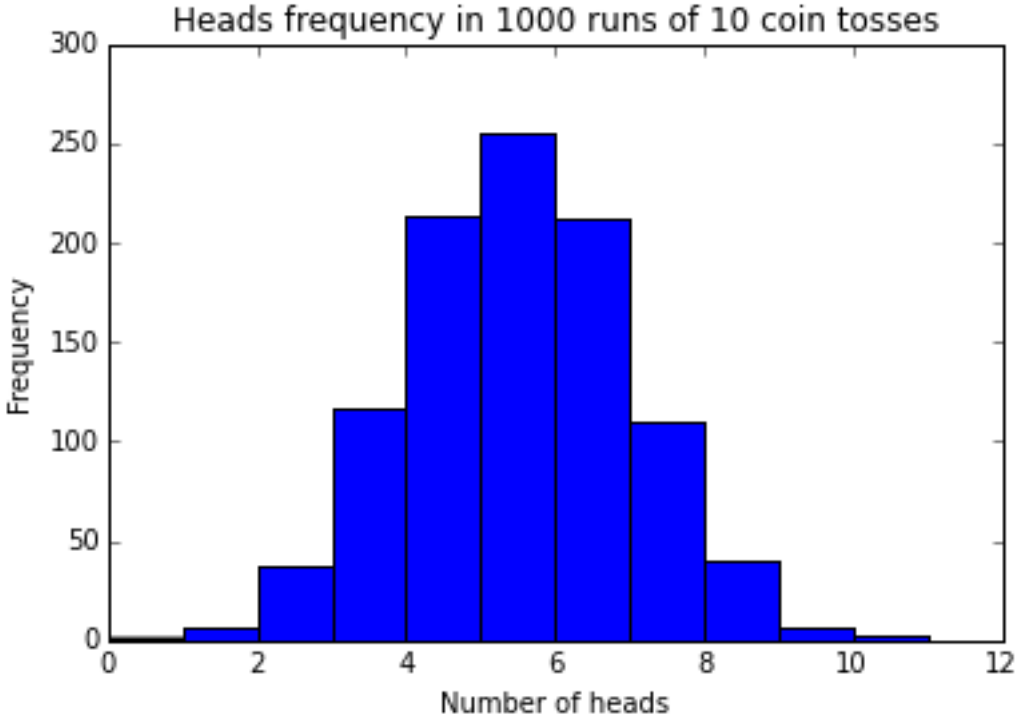
In [8]:
```
ks = [2, 5, 10, 100, 1000, 10000]
```

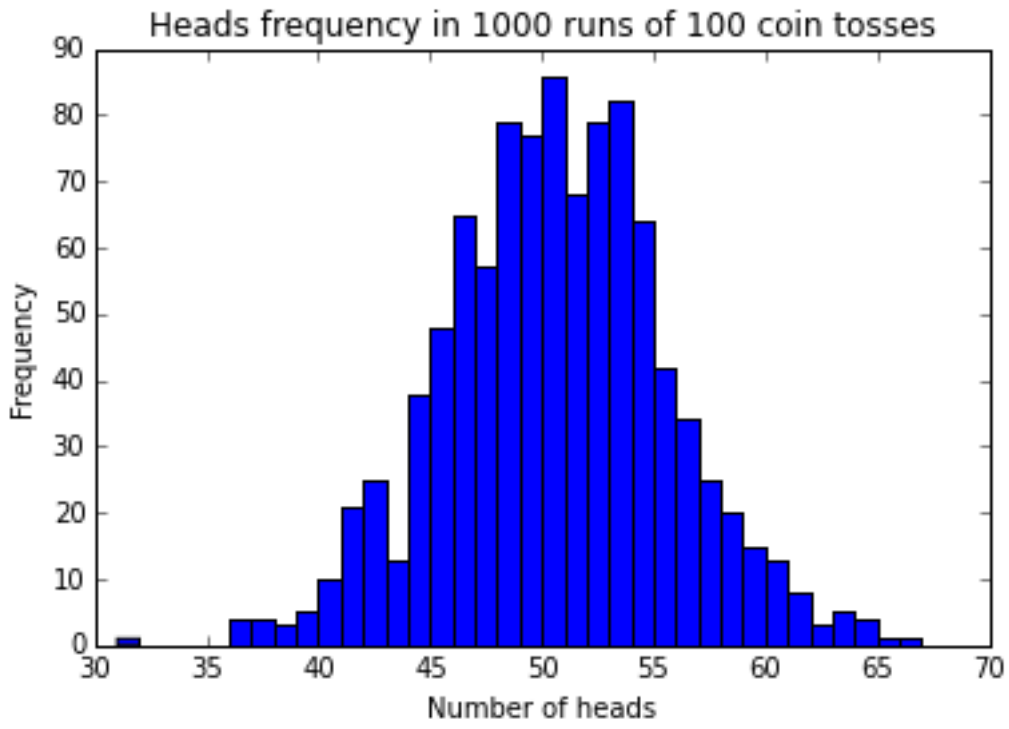
In [18]:
```
def partB():
    """
    YOUR CODE HERE
    """

    for k in ks:
        plt.figure()

        # YOUR CODE HERE
        data = count_heads_in_runs(k, 1000)
        bin_width = 1
        plt.hist(data, bins=xrange(min(data), max(data)+bin_width*2, bin_width))
        plt.title("Heads frequency in 1000 runs of " + str(k) + " coin tosses")
        plt.xlabel("Number of heads")
        plt.ylabel("Frequency")
        # END YOUR CODE HERE

        plt.show()
```

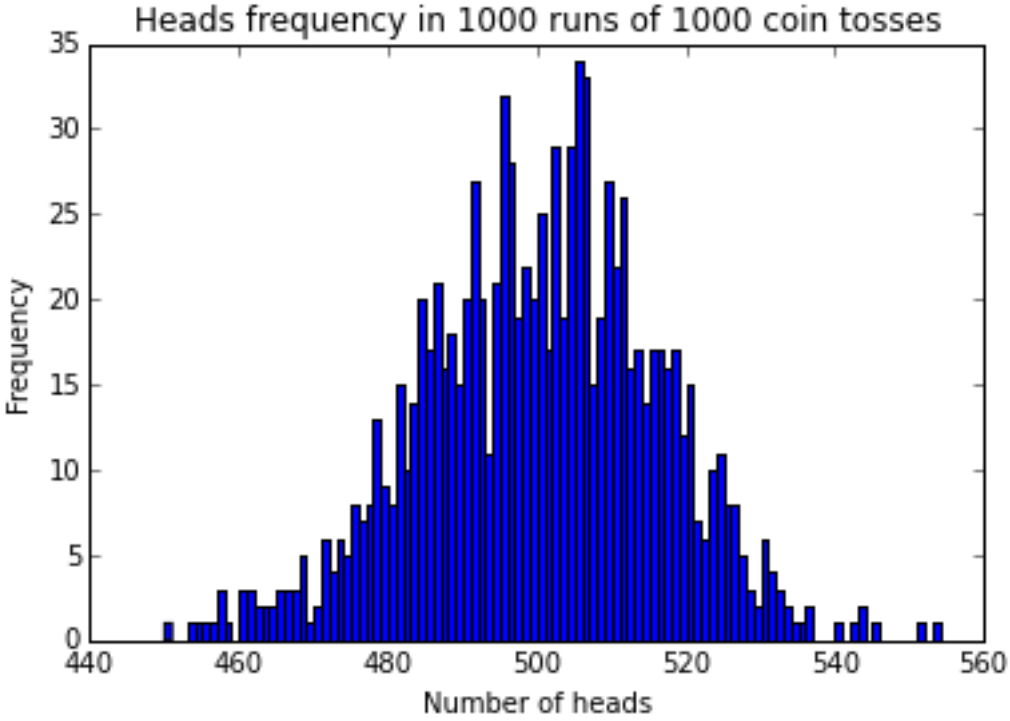
In [19]:
```
partB()
```

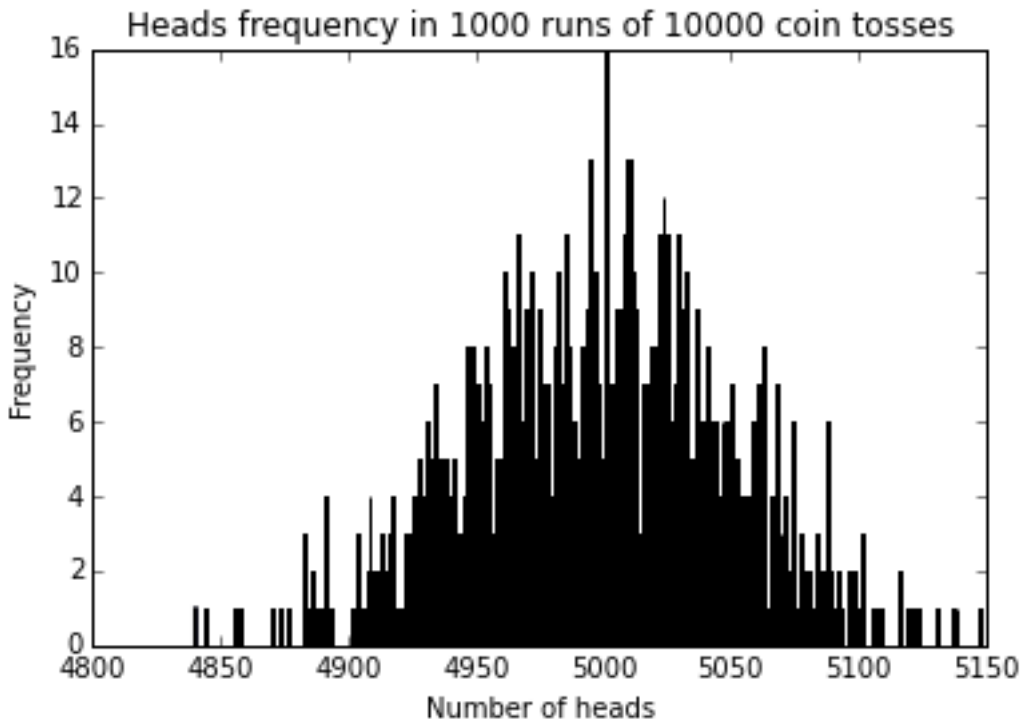



Heads frequency in 1000 runs of 5 coin tosses
Frequency
Number of heads
0
50
100
150
200
250
300
0
1
2
3
4
5
6


Heads frequency in 1000 runs of 10 coin tosses
Frequency
Number of heads
0
50
100
150
200
250
300
0
2
4
6
8
10
12


Heads frequency in 1000 runs of 100 coin tosses
Frequency
Number of heads
0
10
20
30
40
50
60
70
80
90
30
35
40
45
50
55
60
65
70


Heads frequency in 1000 runs of 1000 coin tosses
Frequency
Number of heads
0
5
10
15
20
25
30
35
440
460
480
500
520
540
560



YOUR COMMENTS HERE:

Part (c): Center the Histogram

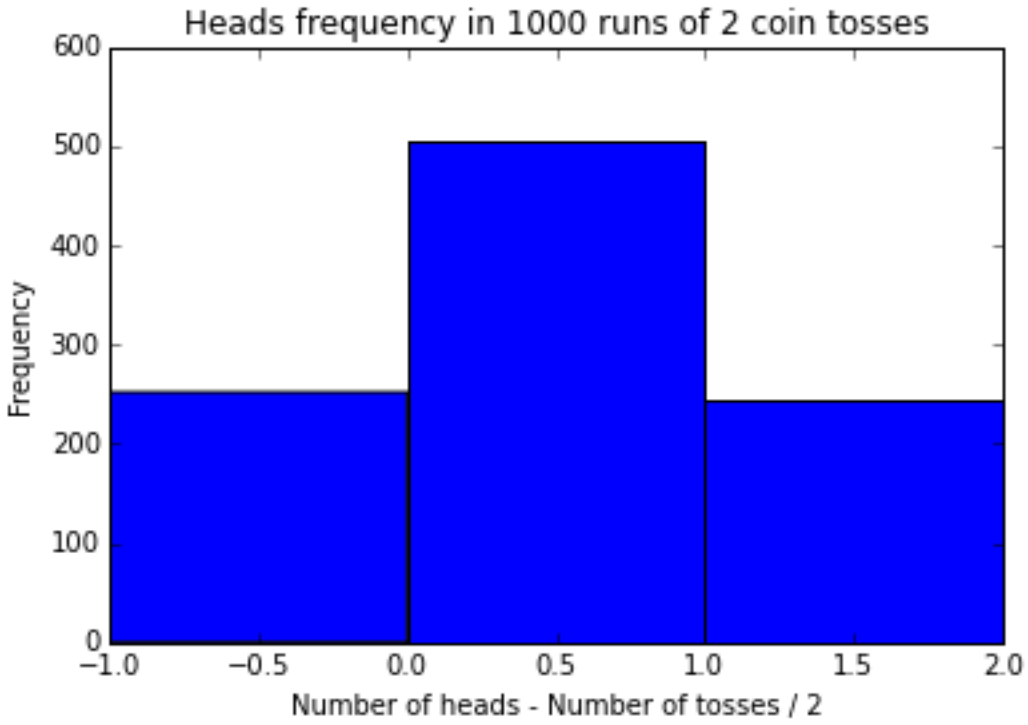
Notice that the horizontal axis has different scales as k varies. Suppose you wanted to "center" these histogram plots. How should you change your code for the above part to center the plots around the origin (0) as k varies? Plot the resulting histogram of $N - f(k)$ where N is the number of heads in a particular run of fair coin tosses and $f(k)$ is the shift that you have chosen.

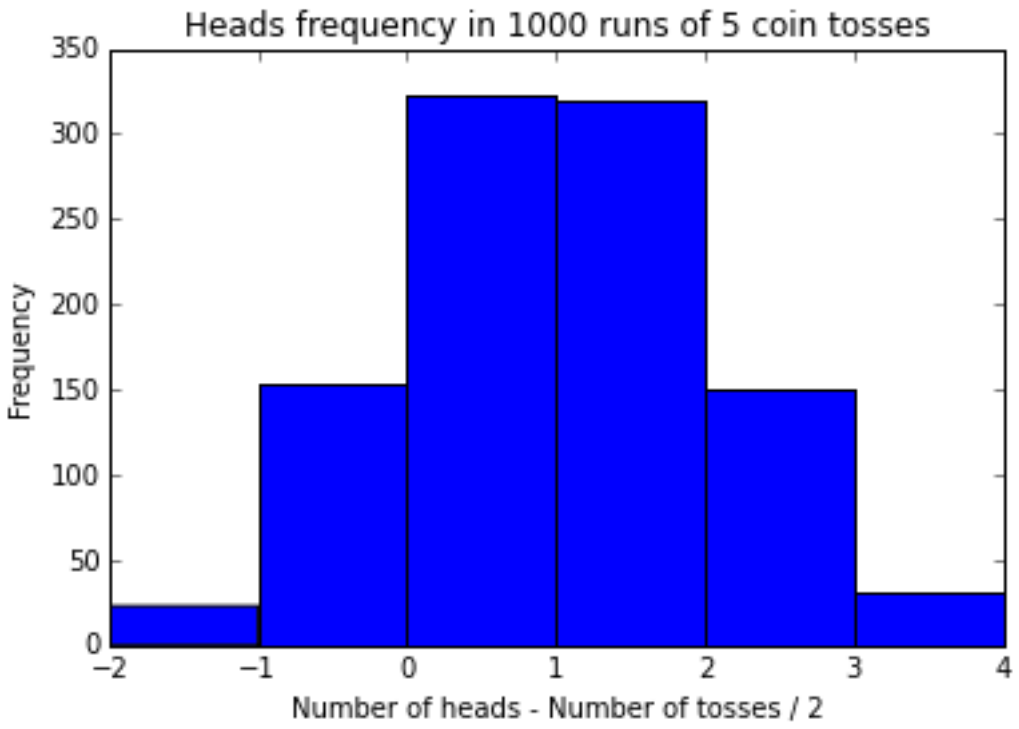
```
In [11]: def scaled_run(k, n):
             """
             Shifts the center of the horizontal axis to the origin by subtracting
             half the number of tosses from the number of heads

             YOUR CODE HERE
             """

             return [int(count_heads(k)) - k//2 for _ in range(n)]
```

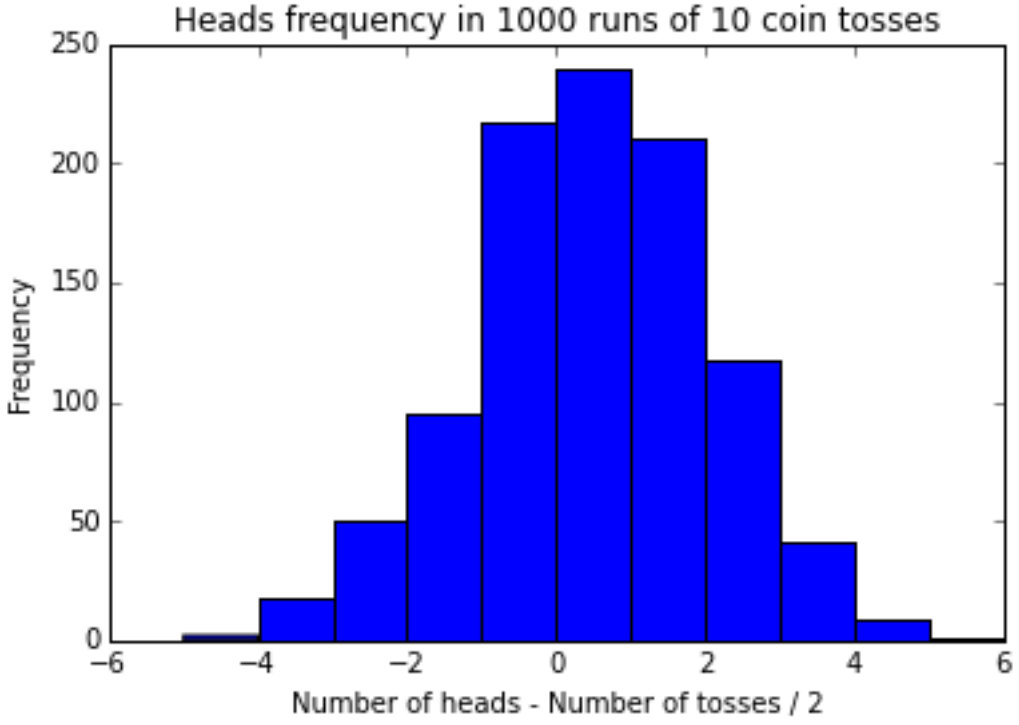
```
In [20]: # YOUR CODE HERE

         def partC():
             """
             YOUR CODE HERE
             """

             for k in ks:
                 plt.figure()

                 # YOUR CODE HERE
```

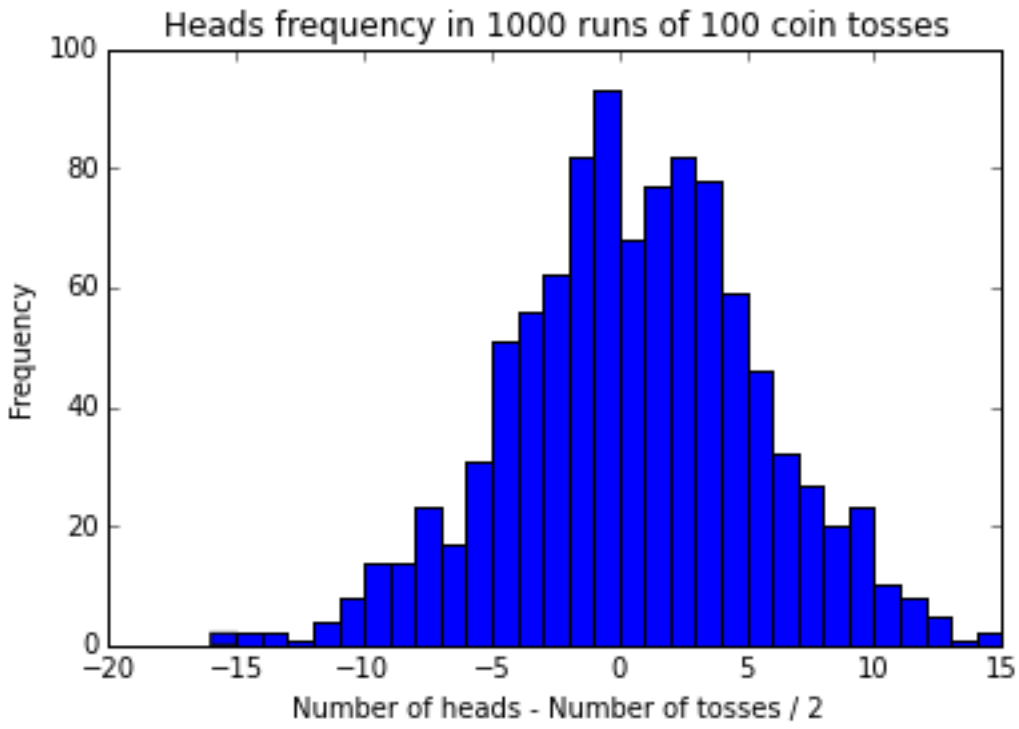
```
    data = scaled_run(k, 1000)
    bin_width = 1
    plt.hist(data, bins=xrange(min(data), max(data)+bin_width*2, bin_width))
    plt.title("Heads frequency in 1000 runs of " + str(k) + " coin tosses")
    plt.xlabel("Number of heads - Number of tosses / 2")
    plt.ylabel("Frequency")

    # END YOUR CODE HERE
    plt.show()
```

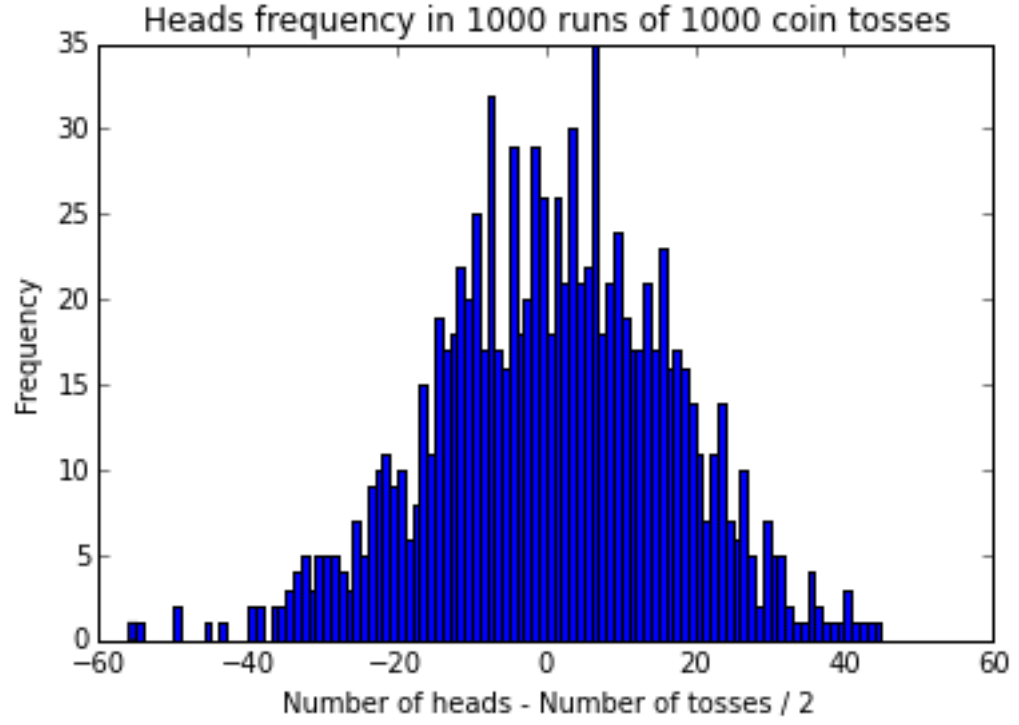
In [21]: `partC()`

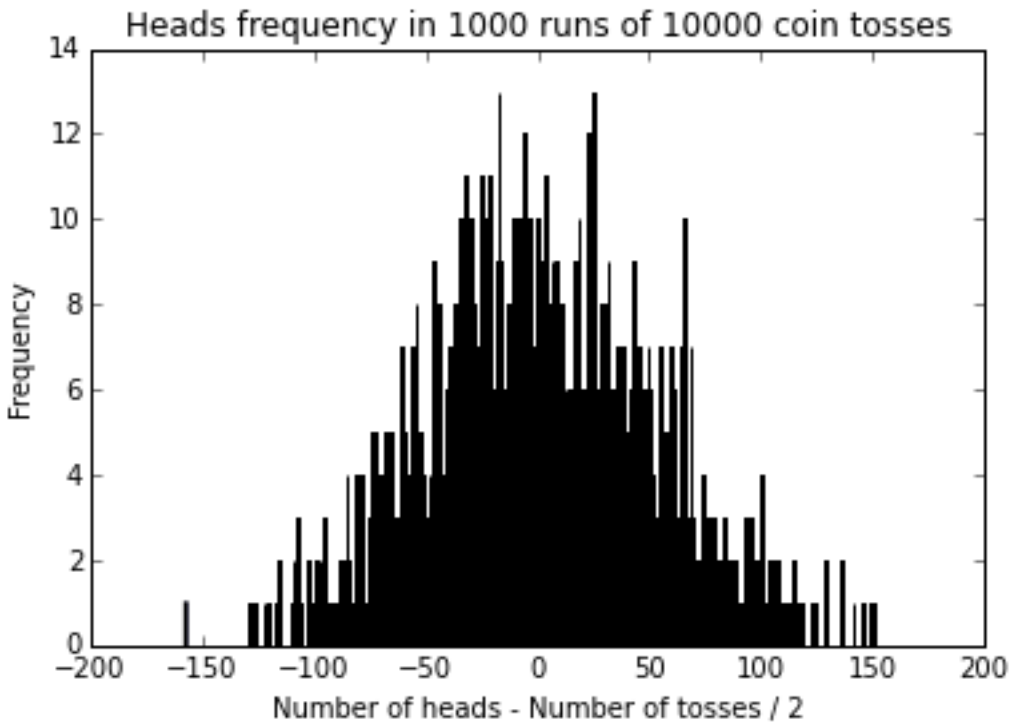



Heads frequency in 1000 runs of 5 coin tosses
Frequency
Number of heads - Number of tosses / 2
350
300
250
200
150
100
50
0
−2
−1
0
1
2
3
4


Heads frequency in 1000 runs of 10 coin tosses
Frequency
Number of heads - Number of tosses / 2
250
200
150
100
50
0
−6
−4
−2
0
2
4
6


Heads frequency in 1000 runs of 100 coin tosses
Frequency
Number of heads - Number of tosses / 2
100
80
60
40
20
0
−20
−15
−10
−5
0
5
10
15


Heads frequency in 1000 runs of 1000 coin tosses
Frequency
Number of heads - Number of tosses / 2
35
30
25
20
15
10
5
0
−60
−40
−20
0
20
40
60



YOUR COMMENTS HERE:

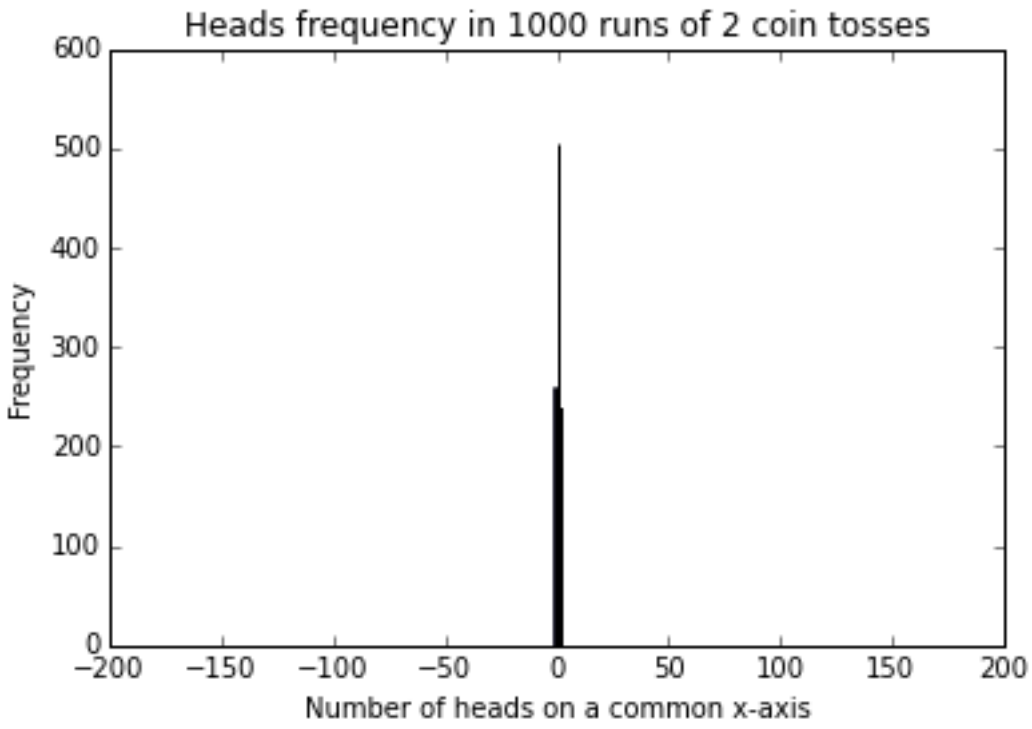
Part (d): Common x-axis

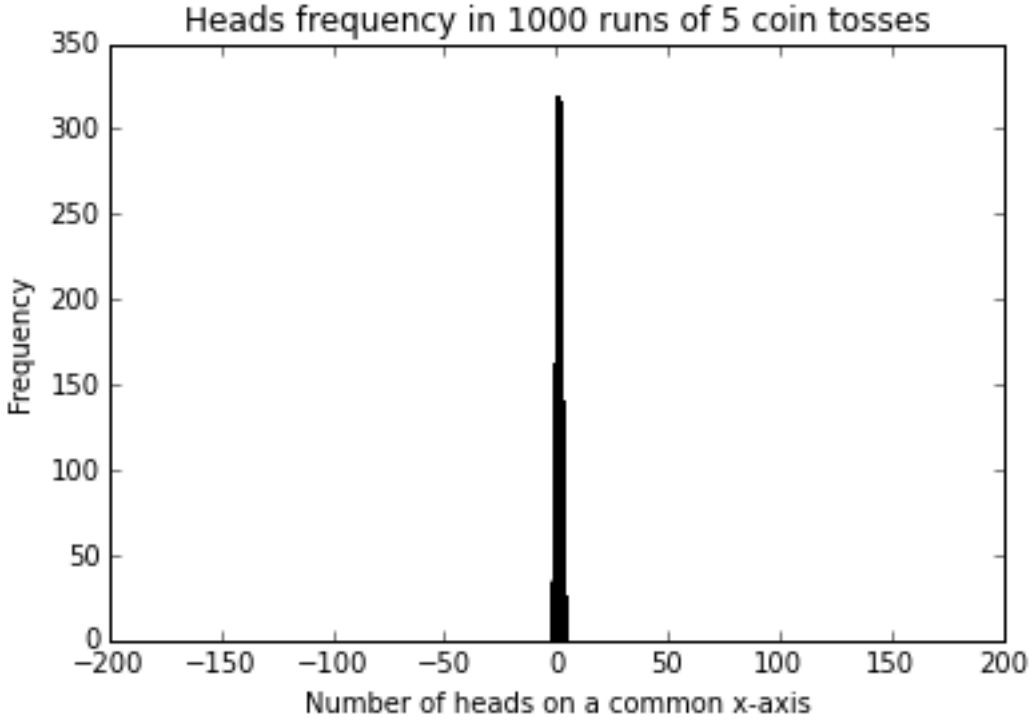
Repeat the plots of the previous part except this time, choose a common set of units for the x-axes (so the x-axes in all plots will have the same range). What range did you choose, and why?

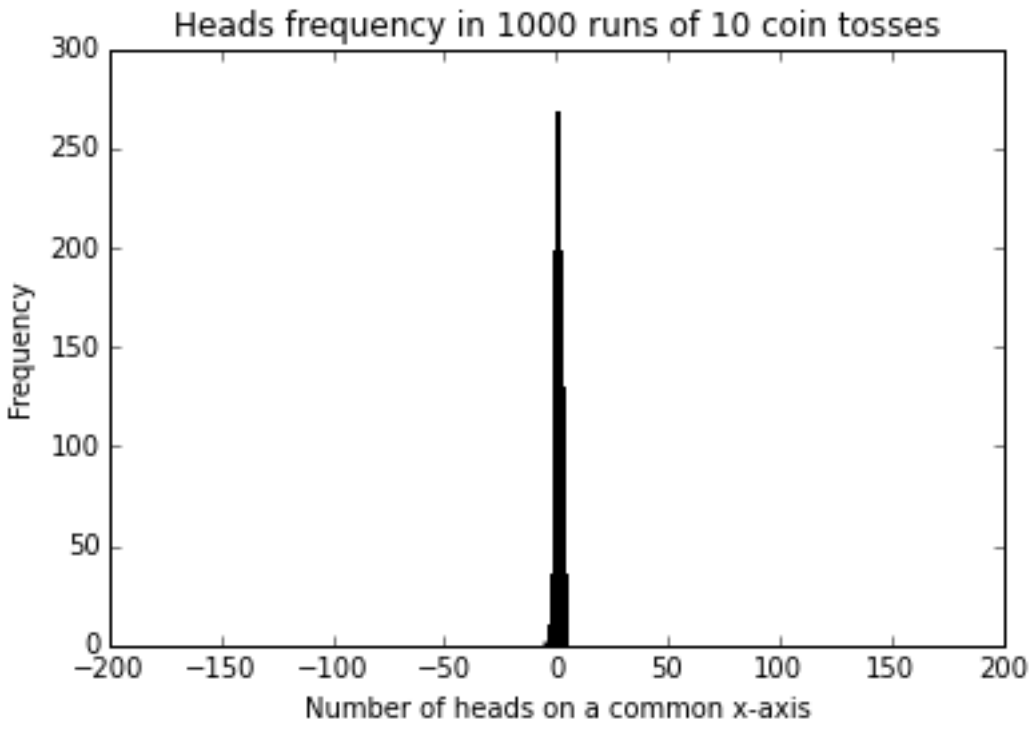
Hint: You can use `plt.xlim()` to set the x-axis's scale.

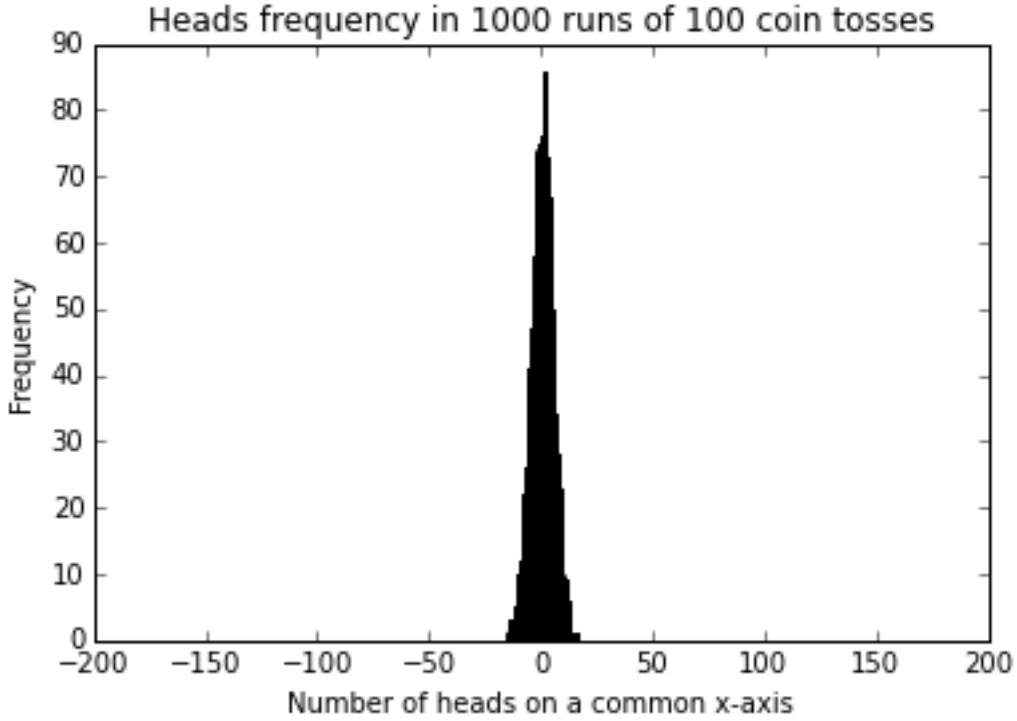
```
In [22]: def partD():
             """
             YOUR CODE HERE
             """

             for k in ks:
                 plt.figure()

                 # YOUR CODE HERE
                 data = scaled_run(k, 1000)
                 bin_width = 1
                 plt.hist(data, bins=xrange(min(data), max(data)+bin_width*2, bin_width))
                 plt.xlim(-200, 200)
                 plt.title("Heads frequency in 1000 runs of " + str(k) + " coin tosses")
                 plt.xlabel("Number of heads on a common x-axis")
                 plt.ylabel("Frequency")
                 # END YOUR CODE HERE

                 plt.show()
```

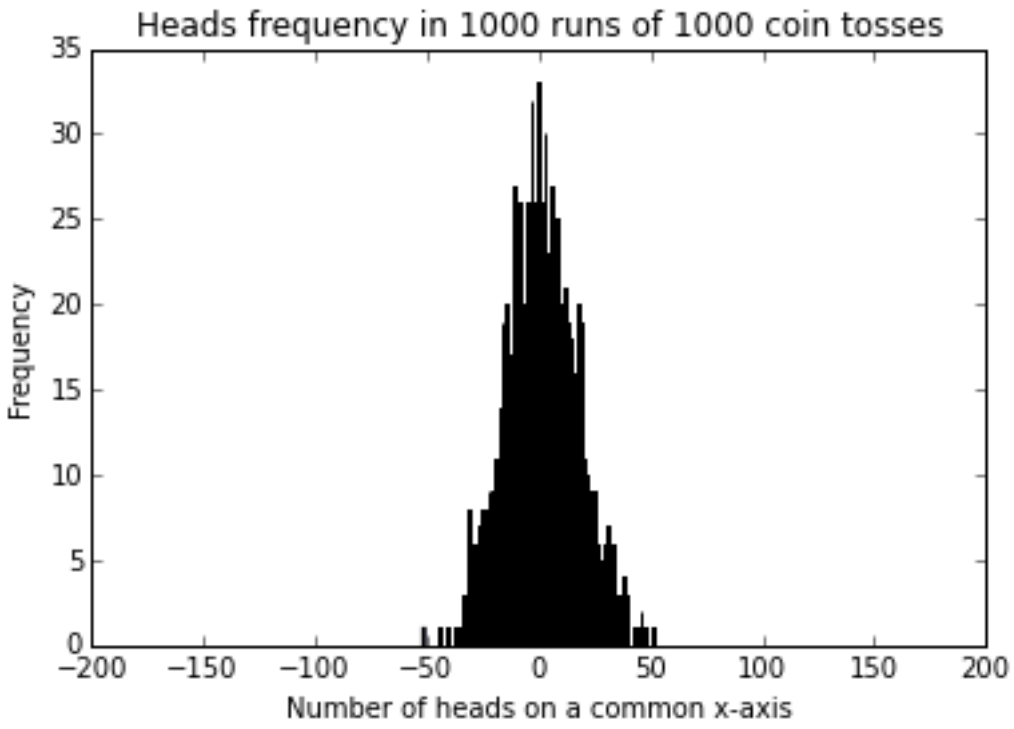
```
In [23]: partD()
```

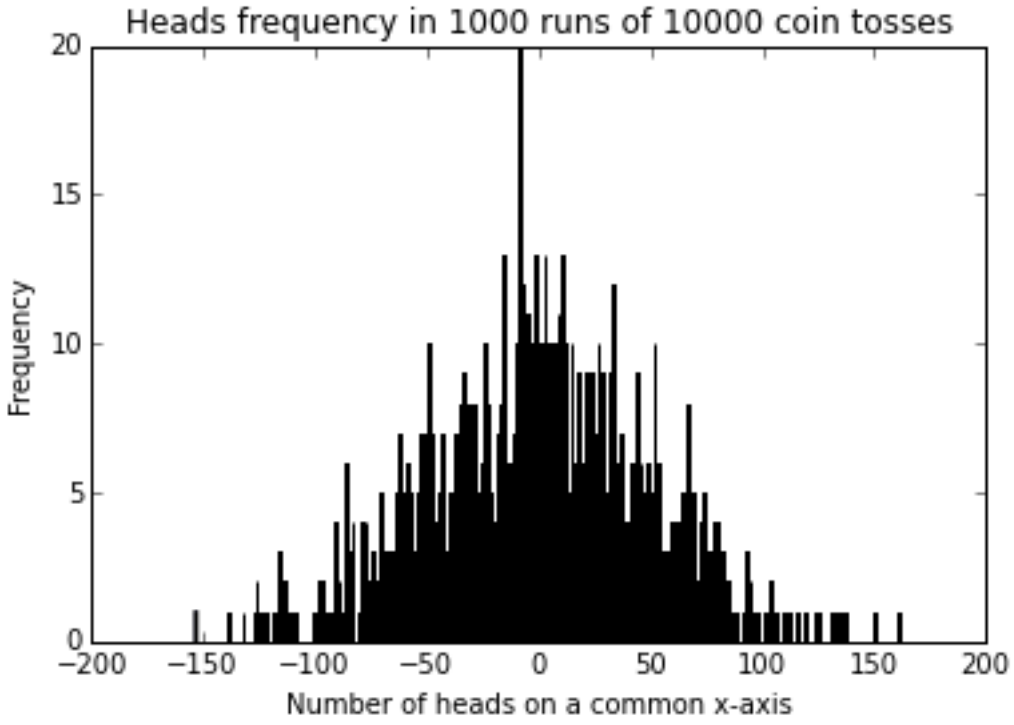

Heads frequency in 1000 runs of 2 coin tosses
Frequency
Number of heads on a common x-axis
600
500
400
300
200
100
0
−200
−150
−100
−50
0
50
100
150
200


Heads frequency in 1000 runs of 5 coin tosses
Frequency
Number of heads on a common x-axis
350
300
250
200
150
100
50
0
−200
−150
−100
−50
0
50
100
150
200


Heads frequency in 1000 runs of 10 coin tosses
Frequency
Number of heads on a common x-axis
300
250
200
150
100
50
0
−200
−150
−100
−50
0
50
100
150
200


Heads frequency in 1000 runs of 100 coin tosses
Frequency
Number of heads on a common x-axis
90
80
70
60
50
40
30
20
10
0
−200
−150
−100
−50
0
50
100
150
200





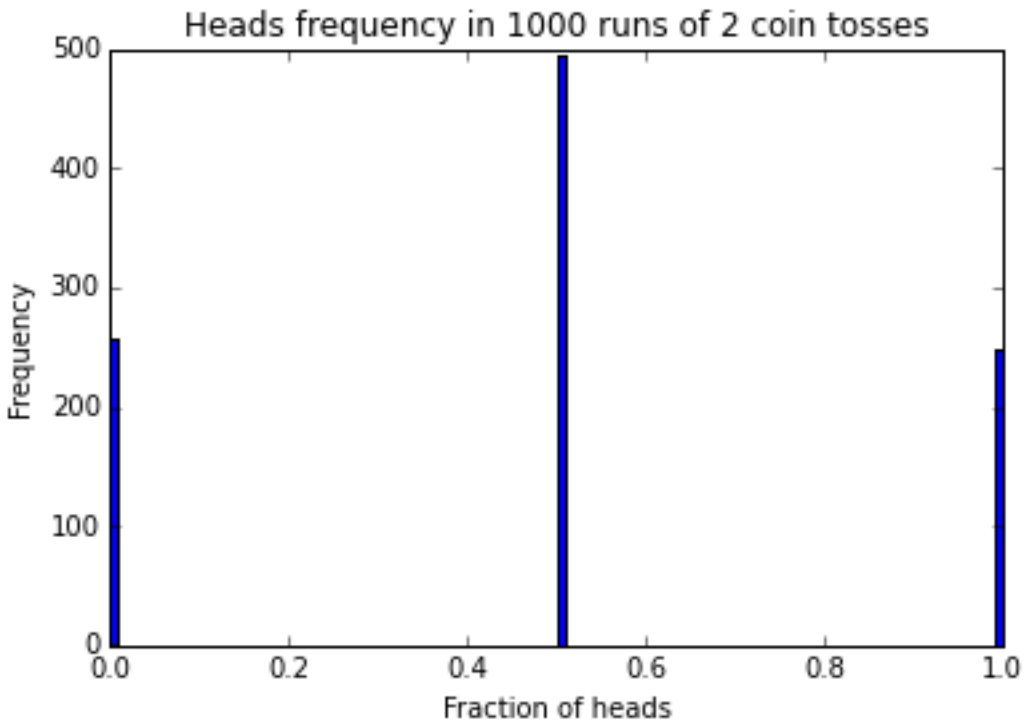
YOUR COMMENTS HERE:

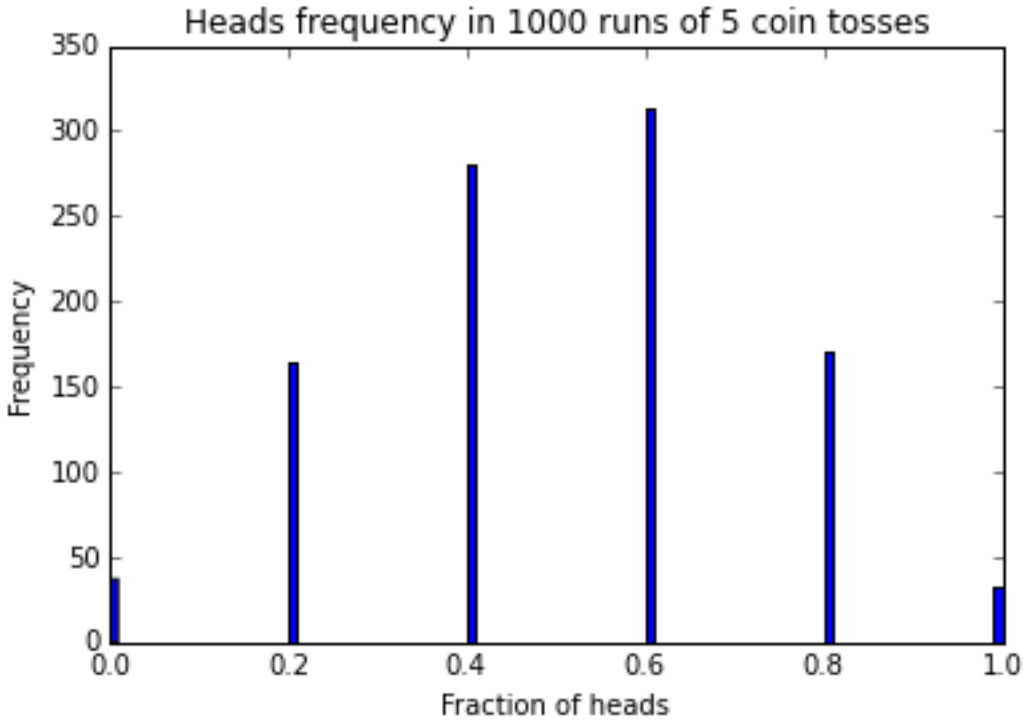
Part (e): Normalized x-axis

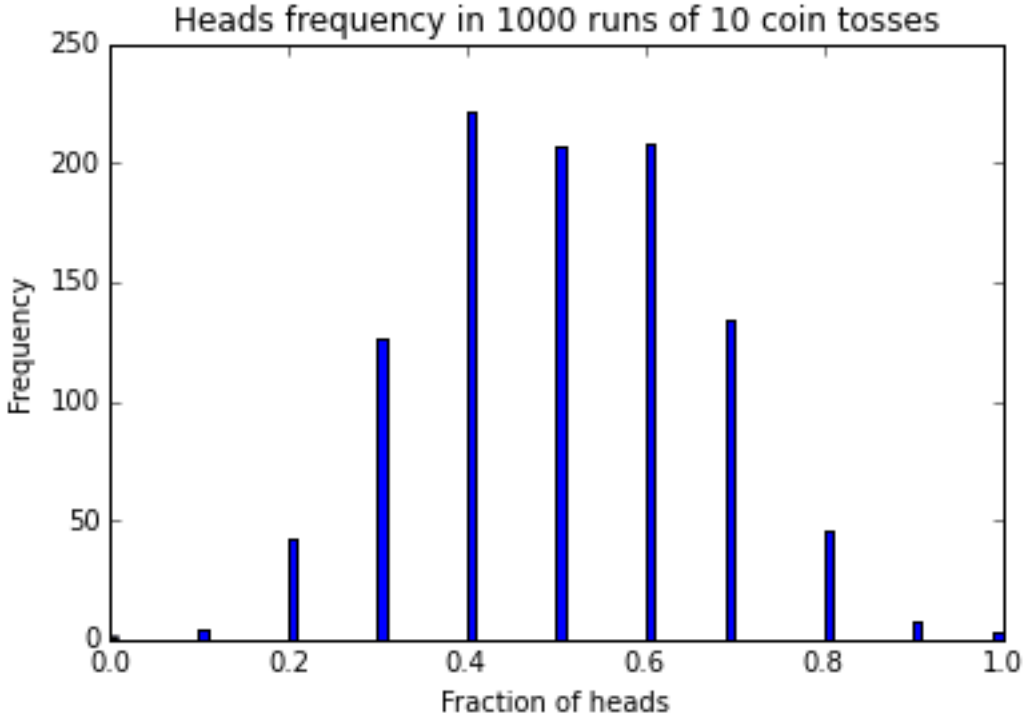
Repeat the plots of the previous part except this time, choose a normalized set of units, corresponding to the fraction of heads appearing (from 0 to 1). So the left-most point should correspond to the case of tossing all tails. And the right-most point should correspond to the case of tossing all heads. How is this set of plots differ from the previous ones?

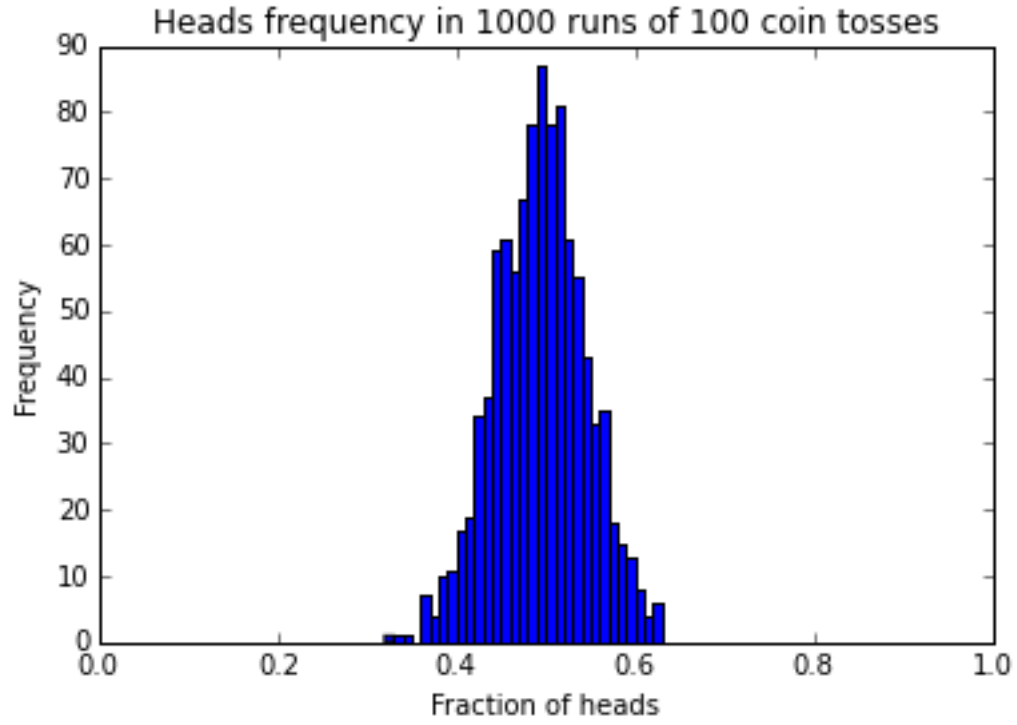
In [26]:
```
def partE():
    """
    YOUR CODE HERE
    """

    for k in ks:
        plt.figure()

        # YOUR CODE HERE
        data = [i / k for i in count_heads_in_runs(k, 1000)]
        bin_width = 0.01

        # Can't use xrange on float, so we have to use np.arange
        plt.hist(data, bins=np.arange(min(data), max(data)+bin_width, bin_width))
        plt.xlim(0, 1)
        plt.title("Heads frequency in 1000 runs of " + str(k) + " coin tosses")
        plt.xlabel("Fraction of heads")
        plt.ylabel("Frequency")
        # END YOUR CODE HERE

        plt.show()
```

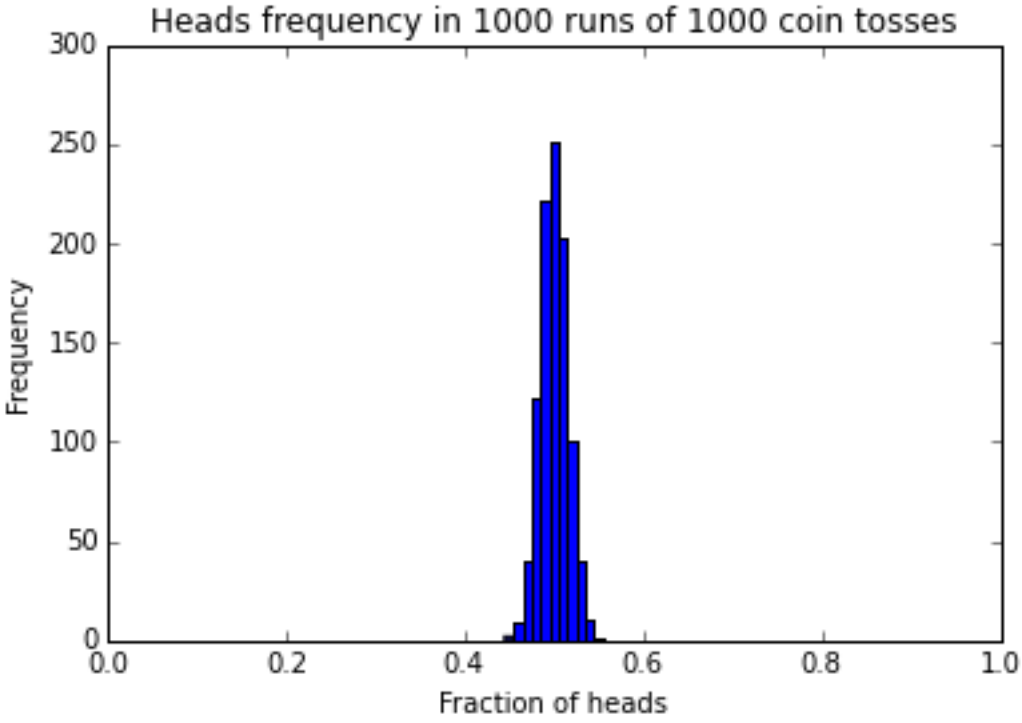
In [27]:
```
partE()
```

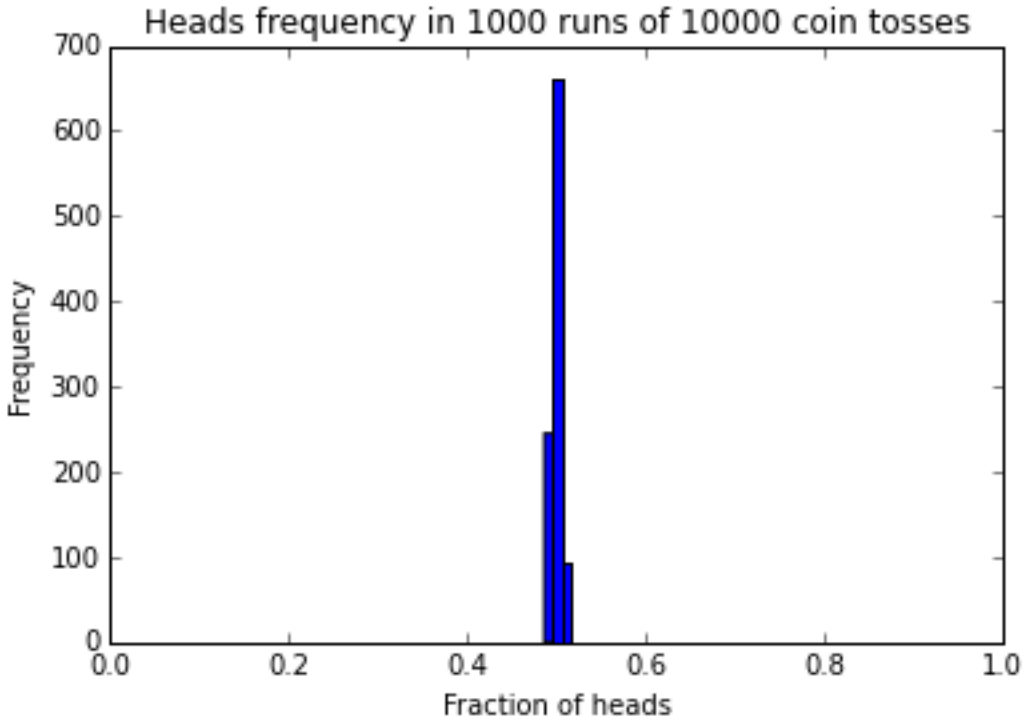



Heads frequency in 1000 runs of 5 coin tosses
Frequency
Fraction of heads
0
50
100
150
200
250
300
350
0.0
0.2
0.4
0.6
0.8
1.0


Heads frequency in 1000 runs of 10 coin tosses
Frequency
Fraction of heads
0
50
100
150
200
250
0.0
0.2
0.4
0.6
0.8
1.0


Heads frequency in 1000 runs of 100 coin tosses
Frequency
Fraction of heads
0
10
20
30
40
50
60
70
80
90
0.0
0.2
0.4
0.6
0.8
1.0


Heads frequency in 1000 runs of 1000 coin tosses
Frequency
Fraction of heads
0
50
100
150
200
250
300
0.0
0.2
0.4
0.6
0.8
1.0



YOUR COMMENTS HERE:

Part (f): Your comments

Comment on what you observed in the three sets of plots above.

YOUR COMMENTS HERE:

Congratulations! You are done with Virtual Lab 8.

Don't forget to convert this notebook to a pdf document, merge it with your written homework, and submit both the pdf and the code (as a zip file) on glookup.

Reminder: late submissions are NOT accepted. If you have any technical difficulty, resolve it early on or use the provided VM.