Paul Fischer

PHYS 522 Statistical Physics

Dr. Michael R. Peterson

01/31/2021

Homework 0 Problem 2

Problem 2: Consider flipping coins like in lecture. You flip the coin N times. Assign the ith flip a value l_i with $l_i = 1$ if it lands heads and $l_i = -1$ if it lands tails. Recall that there are 2^N values possible for $S_N = \sum_{i=1}^{N} l_i$ which, in general, range from $\{N^{(1)}, (N-2)^{(\alpha)}, \ldots, -(N-2)^{(\alpha)}, -N^{(1)}\}$ (for $N \geq 2$) where the superscript indicates how many different configurations of the $\{l_i\}$ yield the same value of S_N. We also showed in class that $\langle S_N \rangle = 0$ and $\langle S_N^2 \rangle = N$ and we defined the standard deviation from the mean to be $\sigma_N \equiv \sqrt{\langle S_N^2 \rangle} = \sqrt{N}$.

a) For $N = 4$, 5, 6, 10, and 20 calculate the fraction of the total values for S_N that are within $\pm\sigma_N$ of the mean $\mu \equiv \langle S_N \rangle = 0$ (for $N = 10$ and definitely $N = 20$ you may want to use some sort of computational approach (C, Fortran, Python, Excel, etc.)).

In [1]:
```
# Let us import the necessary modules here.

from math import sqrt, ceil, floor
from scipy.special import binom
import matplotlib.pyplot as plt
from IPython.display import display, Latex
import numpy as np
from scipy.signal import argrelextrema
```

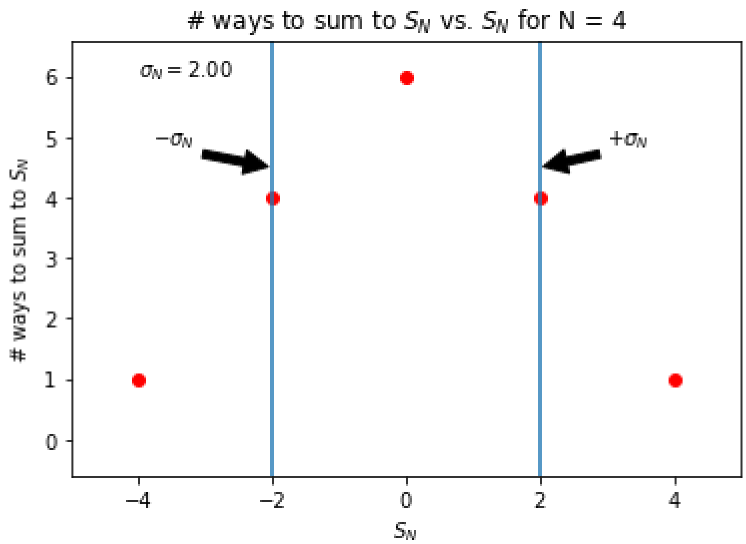
In [2]:

```python
# Let us define a function with which we can calculate the fraction of the total values for S_N
# that are within +/- sigma_N of the mean and can represent the result visually.

def totValsSN(n):
    """The function totValsSN takes as input a nonnegative integer n
    and outputs the total value for S_N."""
    return 2**n

def sigmaN(n):
    """The function sigmaN takes as input a nonnegative integer n
    and outputs the standard deviation from the mean."""
    return sqrt(n)

def fracValsWInSigma(n, plot=True):
    """The function fracValsWInSigma takes as input a nonnegative integer n
    and outputs the fraction of the total values for S_N
    that are within +/- sigma_N of the mean
    and plots a visual representation of the result if plot=True."""

    # define sigma_N, the S_N values, and the # of ways to get those values for the plot
    stDev = sigmaN(n)
    xAxis = list(range(n, -(n+1), -2))
    howManyWays = []
    for i in range(0, n+1):
        howManyWays.append(binom(n, abs(i)))

    # determine which x-values lie within +/- sigma_N so that the corresponding # of ways to get
    # that value of sigma_N can be summed to calculate the desired fraction
    xValsWInStDev = [i for i in xAxis if (i > -stDev) & (i < stDev)]
    yvalsWInStDev = [howManyWays[i] for i in range(floor((len(howManyWays)-len(xValsWInStDev))/2),\
                                                   floor((len(howManyWays)+len(xValsWInStDev))/2))]
    numerator = int(sum(yvalsWInStDev))
    denominator = totValsSN(n)

    if plot==False:
        return numerator/denominator
    else:
        frac = "{:.4f}".format(numerator/denominator)

        # display the desired fraction
        display(Latex(f"""The fraction of the total values for $S_N$ that are within $\pm\sigma_N$ of
        the mean for N = {n}: $\dfrac{{{numerator}}}{{{denominator}}}$ $\\approx$ {frac}"""))

        # represent the results visually via a plot
        plt.plot(xAxis,howManyWays,'ro')
        plt.axvline(x=-stDev)
```

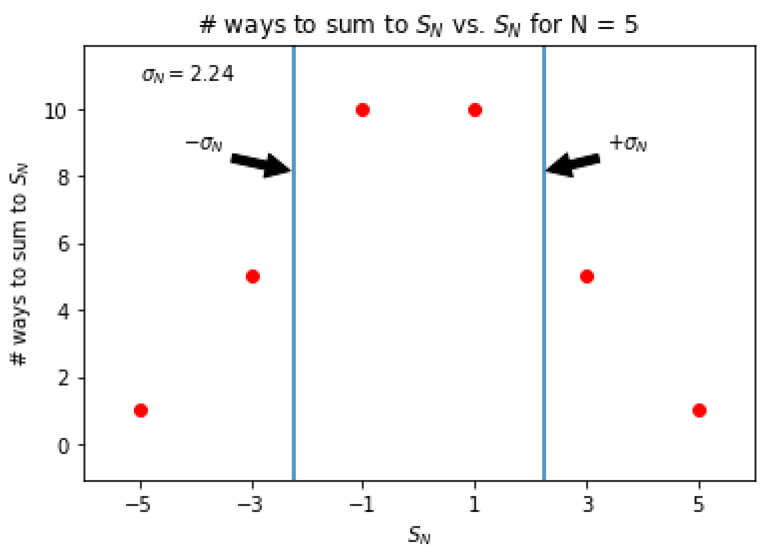
```
        plt.axvline(x=stDev)
        plt.axis([-(n+1), n+1, -.1*binom(n,n/2), 1.1*binom(n,n/2)])
        plt.xticks(xAxis)
        plt.xlabel('$S_N$')
        plt.ylabel('# ways to sum to $S_N$')
        plt.title('# ways to sum to $S_N$ vs. $S_N$ for N = %d' % n)
        plt.text(-n, binom(n,n/2), '$\sigma_N = $%.2f' % stDev)
        plt.annotate('$+\sigma_N$', xy=(stDev, 3/4*binom(n,n/2)), xytext
=(1.5*stDev, 13/16*binom(n,n/2)),
                    arrowprops=dict(facecolor='black', shrink=0.05))
        plt.annotate('$-\sigma_N$', xy=(-stDev, 3/4*binom(n,n/2)), xytex
t=(-1.9*stDev, 13/16*binom(n,n/2)),
                    arrowprops=dict(facecolor='black', shrink=0.05))
        plt.show()
```

In [3]:
```
# Let us calculate the desired fraction for each of the indicated values
# and represent each result visually.

for n in [4, 5, 6, 10, 20]:
    fracValsWInSigma(n, plot=True)
```

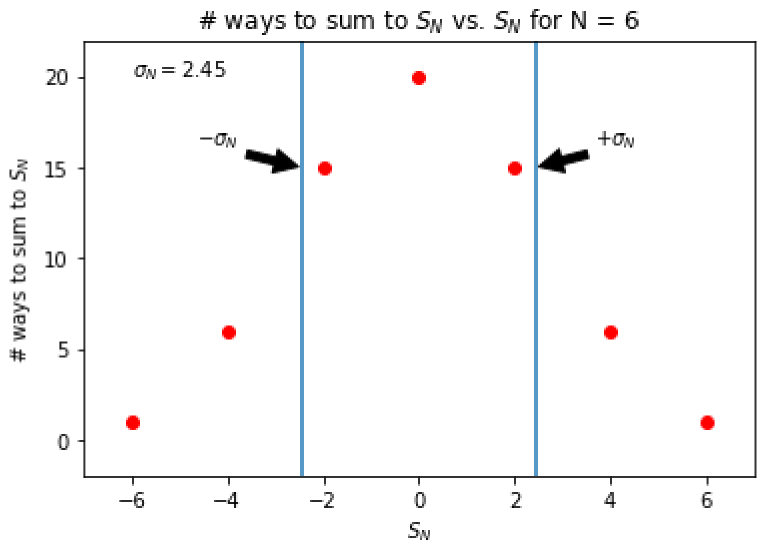
The fraction of the total values for S_N that are within $\pm\sigma_N$ of the mean for N = 4: $\frac{6}{16} \approx 0.3750$



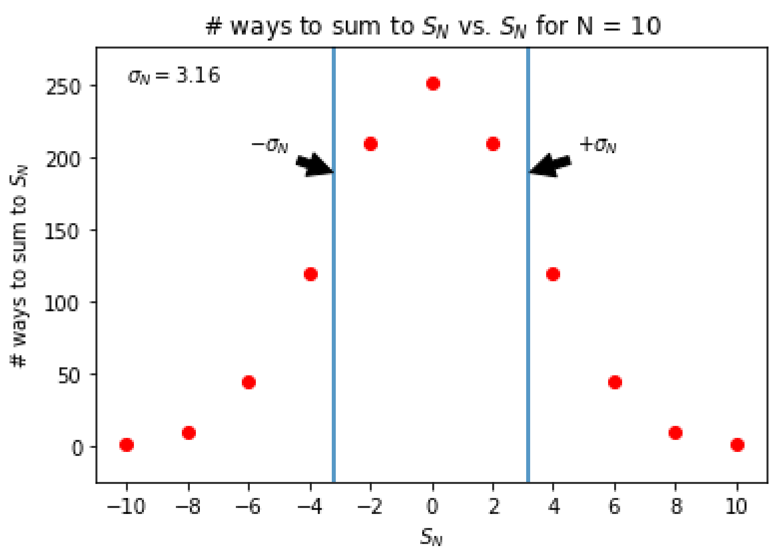
The fraction of the total values for S_N that are within $\pm\sigma_N$ of the mean for N = 5: $\frac{20}{32} \approx 0.6250$



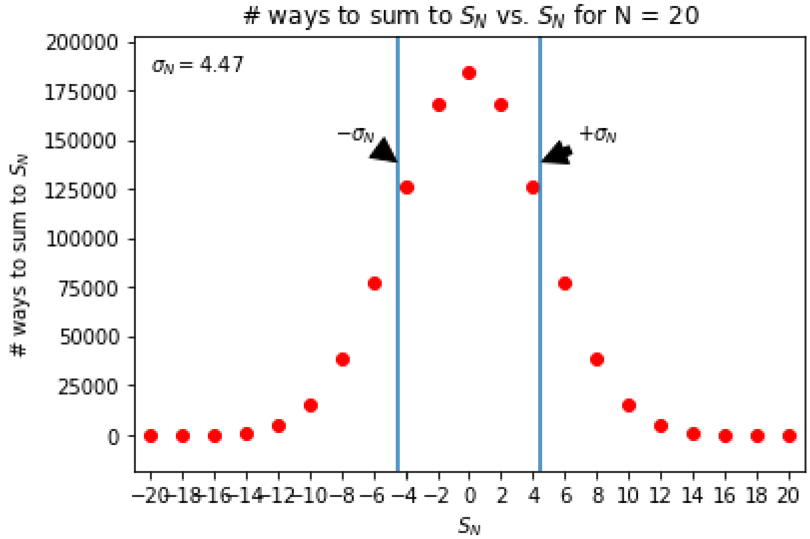
The fraction of the total values for S_N that are within $\pm\sigma_N$ of the mean for N = 6: $\frac{50}{64} \approx 0.7812$



The fraction of the total values for S_N that are within $\pm\sigma_N$ of the mean for N = 10: $\frac{672}{1024} \approx$ 0.6562



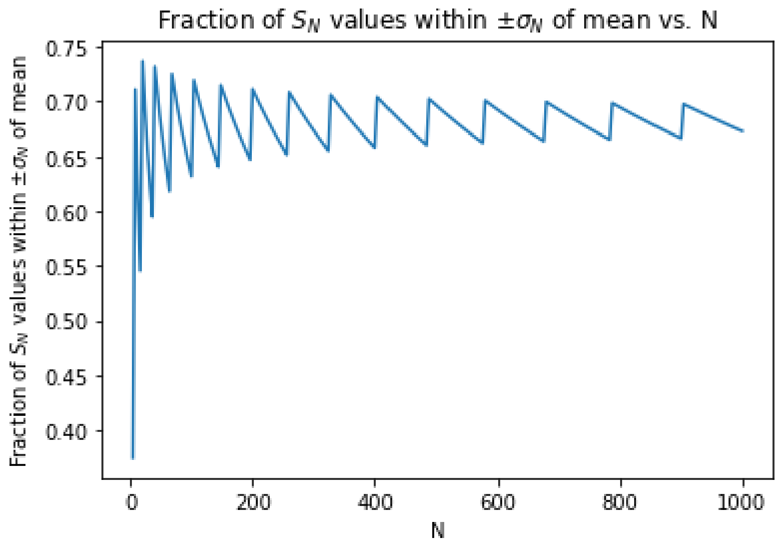
The fraction of the total values for S_N that are within $\pm\sigma_N$ of the mean for N = 20: $\frac{772616}{1048576} \approx$ 0.7368



b) Can you say anything about this fraction as $N \to \infty$?

In [17]:
```
# Let us define a function fracConv that we can use to observe how this
 fraction
# behaves as N goes to infinity.

def fracConv(nMax, convVal=True, plot=True):
    """The function fracConv takes as input a maximum nonnegative ineteg
er
    and outputs an approximated convergence value of the desired fractio
n
    for 4<=n<=nMax if conVal=True and a plot of the fraction vs. these n
values
    if plot=True."""

    n = range(4, nMax+4, 4)
    fracPlot = []
    for i in n:
        fracPlot.append(fracValsWInSigma(i, plot=False))

    if convVal==True:
        # take the average of the last local min and max values to calcu
late
        # the approximate convergence value
        convMin = fracPlot[argrelextrema(np.array(fracPlot), np.less)[0]
[-1]]
        convMax = fracPlot[argrelextrema(np.array(fracPlot), np.greater)
[0][-1]]
        conv = "{:.4f}".format(np.average([convMin, convMax]))
        return conv

    if plot==True:
        plt.plot(n,fracPlot)
        plt.title('Fraction of $S_N$ values within $\pm\sigma_N$ of mean
vs. N')
        plt.ylabel('Fraction of $S_N$ values within $\pm\sigma_N$ of mea
n')
        plt.xlabel('N')
        plt.show()
```

In [18]:
```
# Let us observe visually how the desired fraction converges as N goes t
o 1000.

fracConv(1000, convVal=False, plot=True)
```



In [22]:
```
# We can see that the function oscillates back and forth while convergin
g to a value.
# Let us take the average of the last local maximum and minimum value to
approximate
# where this function connverges to as N goes to infinity.

convVal = fracConv(1000, convVal=True, plot=False)
print(convVal)
```

```
0.6819
```

In [28]:
```
# We can compare our approximate value to the true value of 0.6827 and g
et a % error.

trueVal = 0.6827
pctErr = "{:.4f}".format(abs(trueVal-float(convVal))/trueVal*100)
print('Percent Error: ',pctErr,'%')
```

```
Percent Error:  0.1172 %
```

In [ ]: