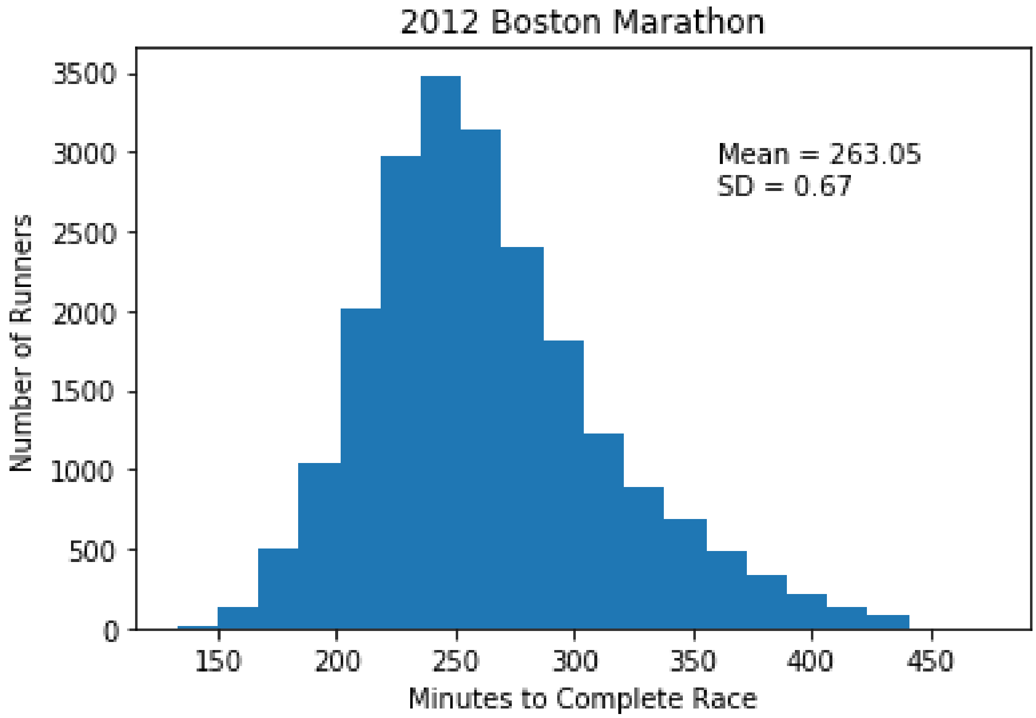
Boston Marathon

August 12, 2018

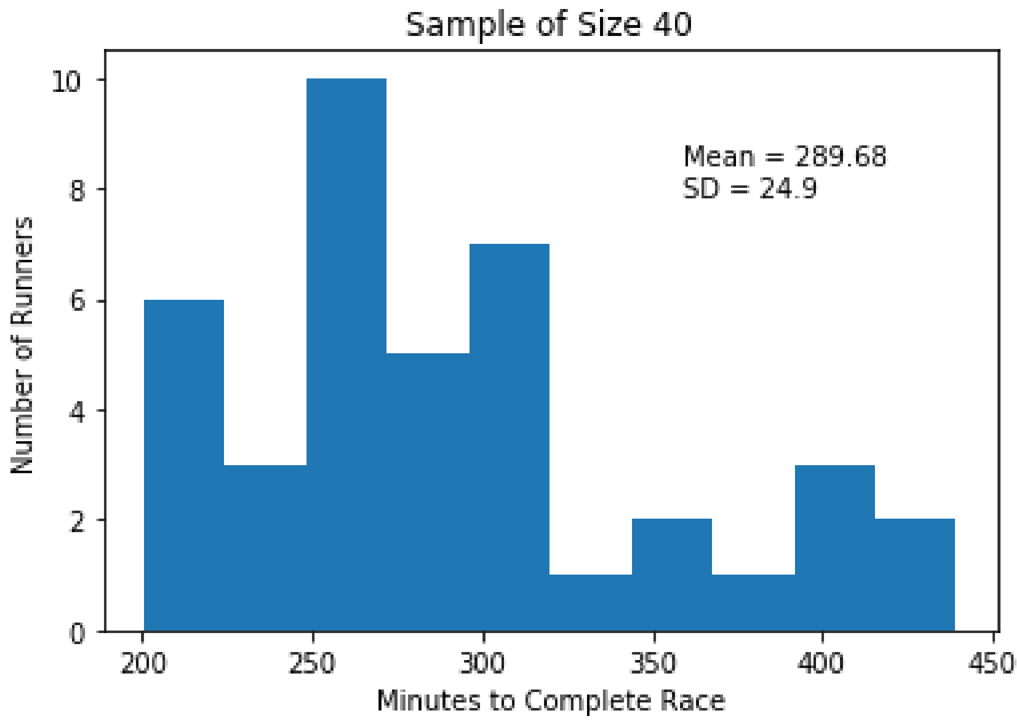
In [11]:
```
# Boston Marathon data
# Assignment 1, Alla Topp
```

In [28]:
```
import pylab
import random

def getBMData(filename):
    """Plot of Boston marathon based on given data """

    data = {}
    f = open('bm_results2012.txt', 'r')
    line = f.readline()
    data['name'], data['gender'], data['age'] = [], [], []
    data['division'], data['country'], data['time'] = [], [], []
    while line != '':
        split = line.split(',')
        data['name'].append(split[0])
        data['gender'].append(split[1])
        data['age'].append(split[2])
        data['division'].append(split[3])
        data['country'].append(split[4])
        data['time'].append(float(split[5][:-1]))
        line = f.readline()
    f.close()
    return data

def variance(data):
    """ data is from Boston marathon text file
        Returns the standart deviation of data."""
    mean = sum(data)/len(data)
    tot = 0.0
    for x in data:
        tot += (x - mean)**2
        return tot/len(data)
def stdDev(data):
    return variance(data)

def makeHist(data, bins, title, xLabel, yLabel):
```

```
    pylab.hist(data, bins)
    pylab.title(title)
    pylab.xlabel(xLabel)
    pylab.ylabel(yLabel)
    mean = sum(data)/len(data)
    std = stdDev(data)
    pylab.annotate('Mean = ' + str(round(mean, 2)) +\
                   '\nSD = ' + str(round(std,2)), fontsize = 10,
                  xy = (0.65, 0.75), xycoords = 'axes fraction')

times = getBMData('bm_results2012.txt')['time']
makeHist(times, 20, '2012 Boston Marathon', 'Minutes to Complete Race', 'Number of Ru
```




```
In [30]: def sampleTimes(times, numExamples):
             """Assumes times a list of floats representing finishing
                 times """
             sample = random.sample(times, numExamples)
             makeHist(sample, 10, 'Sample of Size ' + str(numExamples),
                      'Minutes to Complete Race', 'Number of Runners')
         sampleSize = 40
         sampleTimes(times, sampleSize)
```



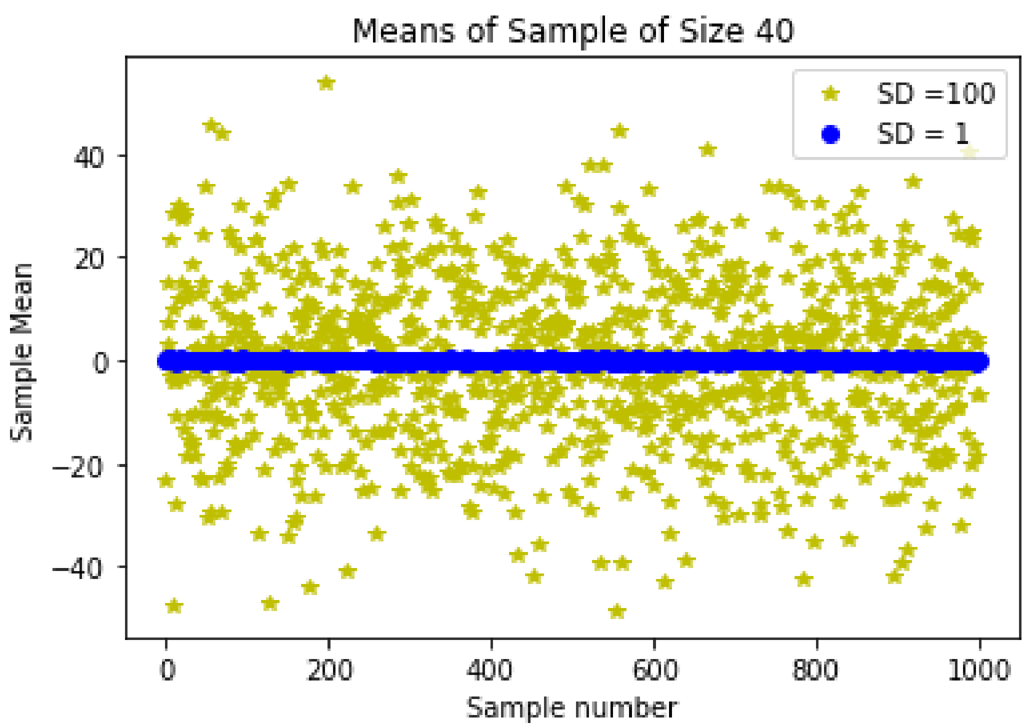

```
In [32]: import scipy.integrate

         def gaussian(x, mu, sigma):
             factor1 = (1/(sigma*((2*pylab.pi)**0.5)))
             factor2 = pylab.e**-(((x-mu)**2)/(2*sigma**2))
             return factor1*factor2
         area = round(scipy.integrate.quad(gaussian, -3, 3, (0,1))[0],4)
         print('Probability of being within 3', 'of true mean of tight dist. =', area)
         area = round(scipy.integrate.quad(gaussian, -3, 3, (0,100))[0],4)
         print('Probability of being within 3', 'of true mean of wide dist. =', area)
```

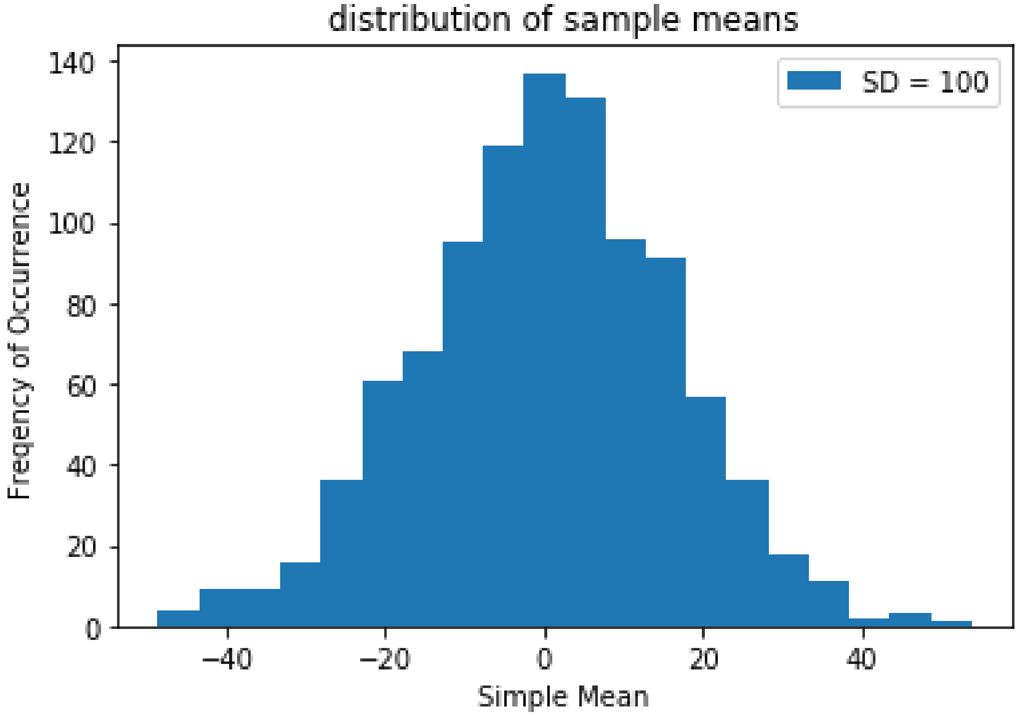
```
Probability of being within 3 of true mean of tight dist. = 0.9973
Probability of being within 3 of true mean of wide dist. = 0.0239
```

```
In [33]: def testSamples(numTrials,sampleSize):
             tightMeans, wideMeans = [], []
             for t in range(numTrials):
                 sampleTight, sampleWide = [], []
                 for i in range(sampleSize):
                     sampleTight.append(random.gauss(0,1))
                     sampleWide.append(random.gauss(0,100))
                 tightMeans.append(sum(sampleTight)/len(sampleTight))
                 wideMeans.append(sum(sampleWide)/len(sampleWide))
```

```
    return tightMeans,wideMeans
tightMeans, wideMeans = testSamples(1000,40)
pylab.plot(wideMeans, 'y*', label = 'SD =100')
pylab.plot(tightMeans, 'bo', label = 'SD = 1')
pylab.xlabel('Sample number')
pylab.ylabel('Sample Mean')
pylab.title('Means of Sample of Size ' + str(40))
pylab.legend()

pylab.figure()
pylab.hist(wideMeans, bins = 20, label = 'SD = 100')
pylab.title('distribution of sample means')
pylab.xlabel('Simple Mean')
pylab.ylabel('Freqency of Occurrence')
pylab.legend()
```

Out[33]: <matplotlib.legend.Legend at 0x2141ded6f98>






```
In [ ]: # Results:
        # The first graph shows the reprasentation of the finishing times,
        # we can see that 3500 runners finished within 230-250 minutes
        # the second graph represents the finishing time of part of
        # randomly chosen competitors based on statistical methods.
        # every time we run it - numbers change because they are random,
        # We can see that choosing the small sample size (40 out of 21000),
        # the estimated mean differs from the population mean by less than 2%.
        # Third plot show the mean of each 1000 sample of size 40 from two
        # normal distributions when the probability density function between -3 and 3(minutes)
        # and the last one shows the mean of each sample.
```

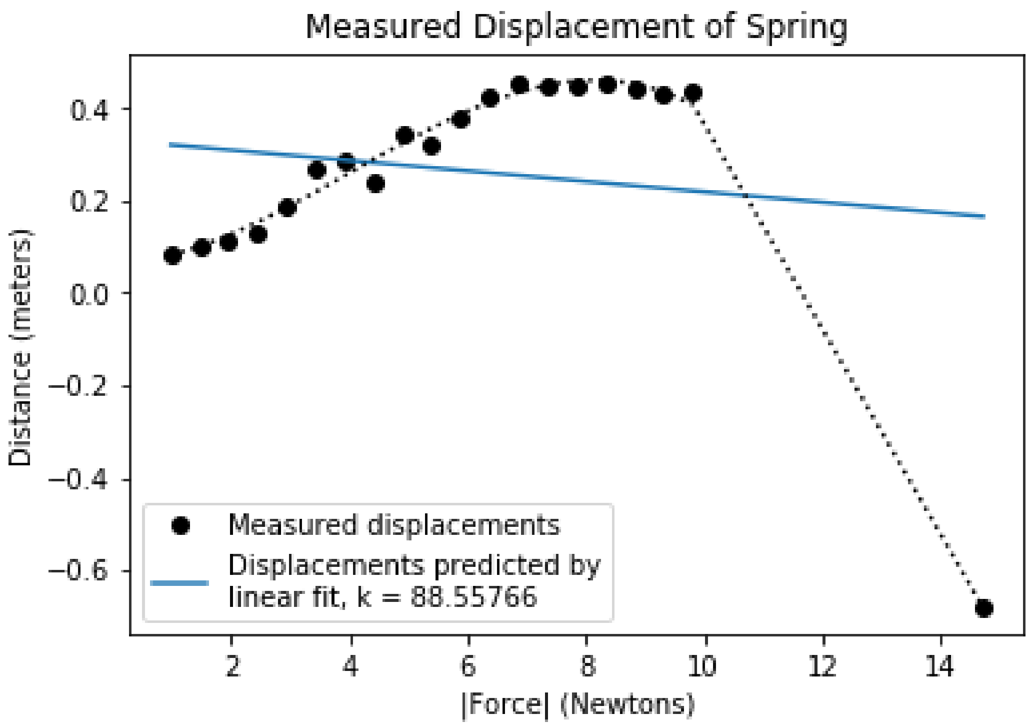
Experimental data

August 12, 2018

In [9]:
```
# finger exercise, p.313 textbook
# modify the code to get the plot in the figure 18.8
# Assignment 2, Alla Topp
```

In [66]:
```
import pylab
import random

def getData(fileName):
    dataFile = open('springData.txt', 'r')
    distances = []
    masses = []
    discardHeader = dataFile.readline()
    for line in dataFile:
        d, m = line.split(' ')
        distances.append(float(d))
        masses.append(float(m))
    dataFile.close()
    return (masses, distances)

def fitData(inputFile):
    masses, distances = getData(inputFile)
    distances = pylab.array(distances)
    masses = pylab.array(masses)
    forces = masses*9.81
    pylab.plot(forces, distances, 'ko',
               label = 'Measured displacements')
    pylab.title('Measured Displacement of Spring')
    pylab.xlabel('|Force| (Newtons)')
    pylab.ylabel('Distance (meters)')

    #find linear fit
    a,b = pylab.polyfit(forces, distances, 1)
    predictedDistances = a*pylab.array(forces) + b
    k = -1.0/a
    pylab.plot(forces, predictedDistances,
               label = 'Displacements predicted by\nlinear fit, k = '
               + str(round(k, 5)))
```

```
    pylab.legend(loc = 'best')

    #find cubic fit
    fit = pylab.polyfit(forces, distances, 3)
    predictedDistances = pylab.polyval(fit, forces)
    pylab.plot(forces, predictedDistances, 'k:', label = 'cubic fit')

fitData('springData.txt')
```




```
In [ ]: #Results:
        # when we hang 1.5 kg of weight and k = 21.53686,
        # the distant should be equal to -.6832
        # the k line is not placed properly because of the dot in the corner of the graph
```

```
In [ ]:
```