# Data analysis in Astronomy

Homework 5 due 11/1 11:59 pm

## Name:

# 1. (Monte Carlo Exercise) The variance of the sample variance

A sample of N data points drawn from a normal distribution has a variance. This variance also has a variance.

Based on theoretical calculation, the variance of variance with N data points is $= \frac{2*\sigma^4}{N}$ where $\sigma$ is the standard deviation of the normal distribution.

## To do:

## 1. write a code and use Monte Carlo method to validate the theoretical expection. (15 points)

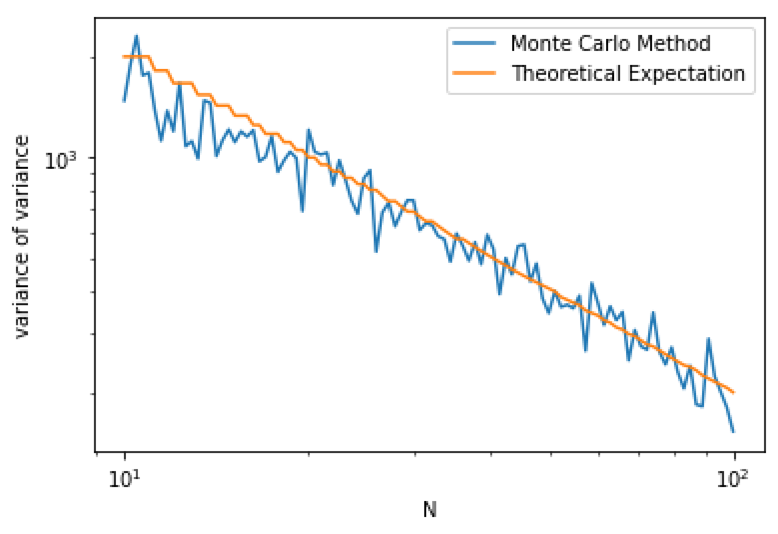
## 2. Produce a plot with x-axis (N data point) and y-axis (variance of variance) with two curves showing a. your Monte Carlo simulation and b. theoretical calculation. (10 points)

(You can just use a normal distribution with mean=0 and sigma=10)

In [1]:
```
# 1.1 write a code and use Monte Carlo method to validate the theoretical expection.
import numpy as np
import matplotlib.pyplot as plt
mu = 0
sigma = 10

def variance_of_variance(N):
    def f(N):
        rand = np.random.normal(mu,sigma,N)#np.zeros(N)
        return np.var(rand)
    n = 100
    er = []
    for i in range(n):
        er.append(f(N))
    err_sim = np.var(er)
    err_std = 2*sigma**4/N
    return err_sim, err_std

# Here is an example for the case of N=1000
err_sim, err_std = variance_of_variance(1000)
print("Monte Carlo simulation:"+str(err_sim)+", Theoretical Expectation:"+str(err_std))
```

```
Monte Carlo simulation:19.791670943568075, Theoretical Expectation:20.0
```

In [2]:
```
# 1.2 Produce a plot with x-axis (N data point) and y-axis (variance of variance)
# with two curves showing a. your Monte Carlo simulation and b. theoretical calculation.
# Here we take N = 100~1000
x = np.logspace(1,2,100)
y = []
y2 = []
for t in x:
    a,b =variance_of_variance(int(t))
    y.append(a)
    y2.append(b)

plt.plot(x,y,label='Monte Carlo Method')
plt.plot(x,y2,label='Theoretical Expectation')
plt.xlabel('N')
plt.ylabel('variance of variance')
plt.xscale('log')
plt.yscale('log')
plt.legend(loc='best')
plt.show()
```




---

# 2. When the central limit theorem breaks down

The central limit theorem states that given a sample with N data points drawn from some distributions with mean $\mu$ and standard deviation $\sigma$, the precision of the mean of the sample will scale with sigma/sqrt(N).

In other words, if you have a larger sample, you can obtain a more precise estimation of the mean value.

However, this is not always true.

## To do:

## a. Find a distribution that will break the central limit theorem and write a code using Monte Carlo method to demonstrate that. (10 points)

## b. Make a plot showing your result with that distribution and the expected trend based on the central limit theorem. (10 points)

(a)

For exponential law random walk, we can describe the sum of all steps as followed:

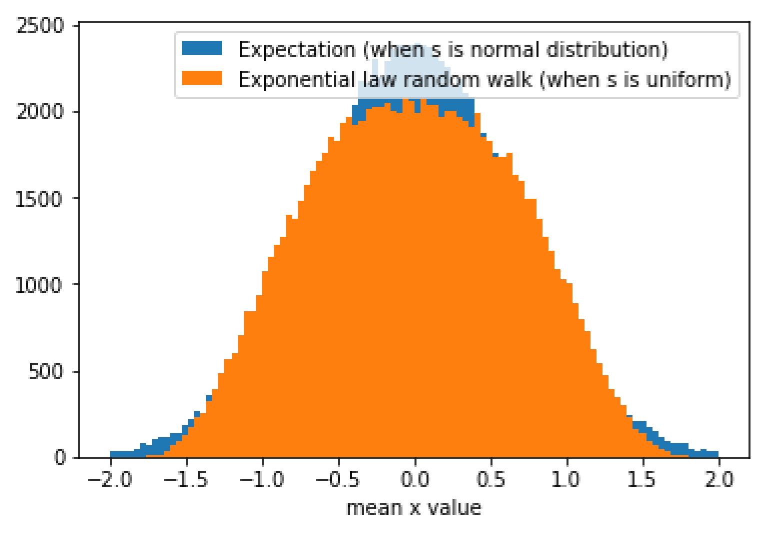
$X = \Sigma a_n s_n = \Sigma a^n s_n$

Here $X$ is the total length, $a_n$ is the exponential term, and $s_n$ is some other random distribution term.

We can choose $s_n$ to be normal distribution with average to be 0 and $\sigma$ to be 1, and then the distribution of $X$ will conform to CLT and converge to Gaussian distribution.

But if you choose $s_n$ to be other distributions (like uniform distribution), then you will get other distributions> of $X$

In [55]:
```
#(b)
# Here we take exponential law random walk as an example
# If we choose s to be normal distribution, then the distribution will converge to Gaussian as CLT expected.
# But if we choose s to be other distributions (like here we choose uniform distribution), then the distributio
# will not converge to Gaussian distribution.
import numpy as np
def exponential_law_random_walk_uniform(alpha,N):
    # This is the exponential law random walk when picking s to be uniform distribution.
    s = np.random.uniform(-1,1,N)
    a = (np.ones(N)*alpha)**np.linspace(0,N,N)
    return np.sum(s*a)
def exponential_law_random_walk_normal(alpha,N):
    # This is the exponential law random walk when picking s to be normal distribution.
    # Here we set its variance to be the same as that of the uniform distribution case.
    s = np.random.normal(0, 3**(-0.5), N)
    a = (np.ones(N)*alpha)**np.linspace(0,N,N)
    return np.sum(s*a)


simulation = []
expectation = []
N = 100000
for i in range(N):
    simulation.append(exponential_law_random_walk_uniform(0.5,100))
    expectation.append(exponential_law_random_walk_normal(0.5,100))

plt.hist(expectation, bins=100,label='Expectation (when s is normal distribution)',range=(-2,2))
plt.hist(simulation, bins=100,label='Exponential law random walk (when s is uniform)',range=(-2,2))
plt.xlabel('mean x value')
plt.legend(loc='best')
plt.show()
```




---

# 3. Finishing your Pearson and Spearman correlation coefficients calculation

In class, you have learned how to calculate Pearson and Spearman correlation coefficientts.

## https://www.dropbox.com/s/h7545q0vzcqhi38/sky_maps_new_64_v6.fits?dl=0

## To do:

## a. Write a code to estimate the uncertainty of the Pearson and Spearman correlation coefficients for (EBV, HI) and (EBV, H2) (15 points)

In [4]:
```
# Data loading
import astropy.io.fits as pf
ISM =pf.open('sky_maps_new_64_v6.fits')[1].data
EBV = ISM['SFD']
HI = ISM['HI']/1e21

conversion_factor = 2*1e20/1e21
H2 = ISM['CO10']*conversion_factor
```

In [5]:
```
# order(x) will return an array showing the order of each element of x
# Pearson_CC and Spearman_CC give Pearson's and Spearman's correlation coefficients
def order(x):
    y = np.argsort(x)
    z = x
    for i in range(len(y)):
        z[y[i]] = i
    return z
def Pearson_CC(x,y):
    c = np.array([x,y])
    return np.corrcoef(c)[0][1]
def Spearman_CC(x,y):
    c = np.array([order(x),order(y)])
    return np.corrcoef(c)[0][1]
```

In [7]:
```
def Rand_Idx(x):
    # Pick a random sample in x
    return int(np.random.uniform(0,len(x)))

def Uncertainty_from_BootStrap(x,y,method,N):
    # N is the number we do bootstrapping
    def f(x,y):
        x2 = []
        y2 = []
        for i in range(len(x)):
            n = Rand_Idx(x)
            x2.append(x[n])
            y2.append(y[n])
        if method=='Pearson':
            return Pearson_CC(x2,y2)
        elif method=='Spearman':
            return Spearman_CC(x2,y2)
    mean = []
    for i in range(N):
        mean.append(f(x,y))
    return np.std(mean)

# Here we take N=200
print('HI\' s Pearson\'s Correlation Coefficient\'s Uncertainty:'+str(Uncertainty_from_BootStrap(EBV,HI,'Pearso
print('HI\' s Spearman\'s Correlation Coefficient\'s Uncertainty:'+str(Uncertainty_from_BootStrap(EBV,HI,'Spear

print('H2\' s Pearson\'s Correlation Coefficient\'s Uncertainty:'+str(Uncertainty_from_BootStrap(EBV,H2,'Pearso
print('H2\' s Spearman\'s Correlation Coefficient\'s Uncertainty:'+str(Uncertainty_from_BootStrap(EBV,H2,'Spear
```

```
HI' s Pearson's Correlation Coefficient's Uncertainty:0.016913722170927462
HI' s Spearman's Correlation Coefficient's Uncertainty:0.00024970776295245783
H2' s Pearson's Correlation Coefficient's Uncertainty:0.015660664390919034
H2' s Spearman's Correlation Coefficient's Uncertainty:0.004526432456028419
```