Physics 305 Demo Notebook 3: Calculating the Mean Square Displacement (MSD) from a Time-Series - Brownian Motion

The mean square deviation or MSD is defined to be: $\langle x^2 \rangle - \langle x \rangle^2$. This is also known as the *structure function*.

Recall that for any two points x_i and x_j, we define the lag time τ to be $t_j - t_i$.

Then, for Brownian motion, MSD $= 2D\tau$ (Eq. 1), where D is the diffusion coefficient. Recall also that the PDF for a given lag time τ is a Gaussian with a standard deviation of MSD$^{1/2} = \sqrt{2D\tau}$ (as derived in class and illustrated in Demo Notebook #2).

Let us now also define Δ to be the number of time steps equivalent to the lag time τ, i.e., $\tau = \Delta * dt$, where dt is the size of the time step.

We introduce the empirical definition of the MSD as a function of Δ (Eq. 2):

$$\text{MSD}(\Delta) = \frac{1}{n-\Delta} \Sigma_{j=0}^{j=n-1-\Delta} \left(x_{j+\Delta} - x_j \right)^2$$

Here, we will calculate the MSD as a function of lag time (using Eq. 2) for standard Brownian motion samples and compare them with the analytical result. Recall that for standard Brownian motion, $D = 1/2$, so we expect (from Eq. 1):

$$\text{MSD}(\tau) = 2D\tau = \tau.$$

This is simply a linear relation with a slope of 1.

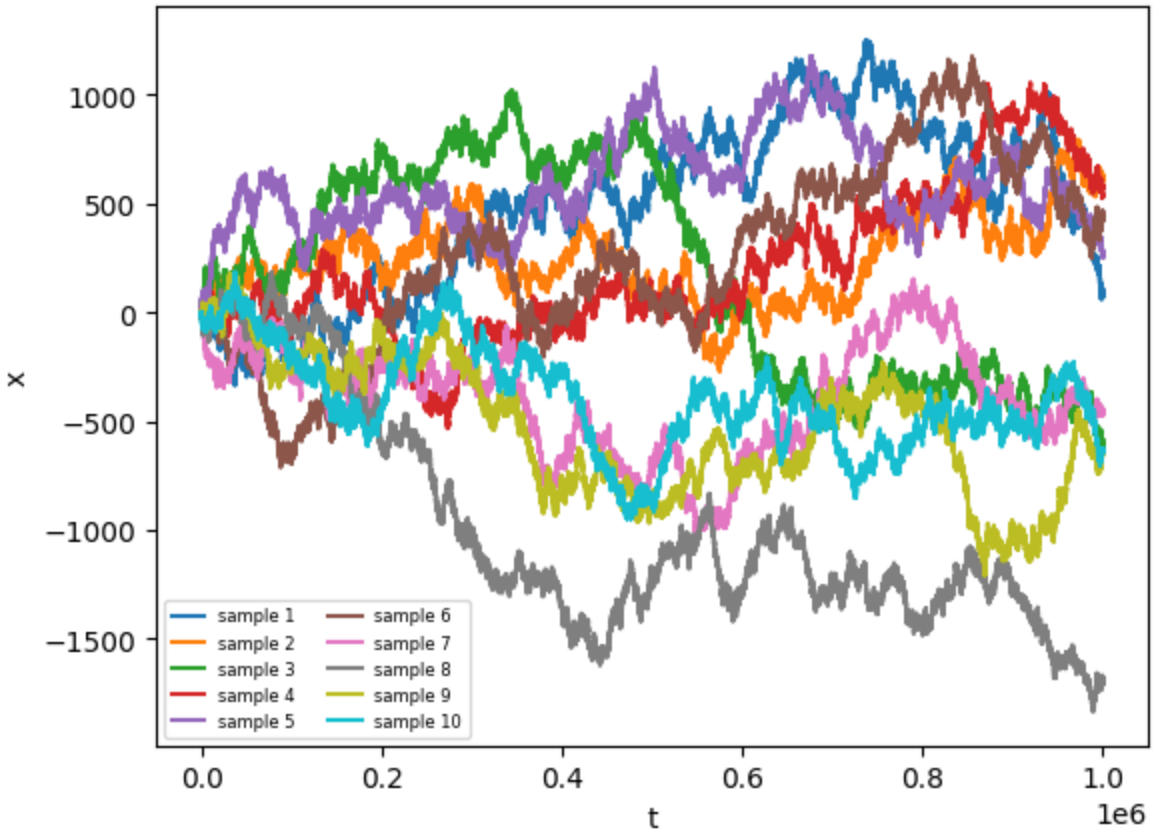
Step 1: Generate Brownian motion samples

In [2]:
```
# import libraries
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
# Set random seed
np.random.seed(seed=17)

# Set parameters
n_samp = 10 # no. of samples/realizations
n = int(1e6)  # no. of timesteps
dt = 1.  # size of time step
sd = np.sqrt(dt) # standard deviation
t = dt*np.arange(n) # time

# Initialize array to hold the BM samples
x_samp = np.zeros((n, n_samp))

# Loop over realizations
for i in np.arange(n_samp):
  # Generate random numbers from Gaussian distribution centered at 0 and
standard deviation sd
  rnd = norm.rvs(size = n, scale = sd)
  # Get cumulative sum of the elements of the array of random numbers
  x_samp[:, i] = np.cumsum(rnd)
```

In [4]:
```
# Plot BM samples
for i in np.arange(n_samp):
  plt.plot(t, x_samp[:, i],label="sample %d" % (i+1))
plt.xlabel("t")
plt.ylabel("x")
plt.legend(loc="lower left", ncol=2, fontsize=6)
```

Out[4]: <matplotlib.legend.Legend at 0x7974e8fe9f60>



Step 2: Generate the MSD

First, let us define the range of lag times to calculate the MSD for. Recall that when we calculated the displacement probability distributions (PDFs) in Demo Notebook #2, we selected $\tau = 10, 10^2, 10^3, 10^4, 10^5, 9 \times 10^5$.

We found that for $\tau = 10^3$ and lower, there is a very good match between the calculated and analytical results, even for a single realization (but this does not necessarily hold for larger lag times).

Here, let us calculate the MSD for $1 \leq \tau \leq 9 \times 10^5$ and indicate the regime $\tau < 10^3$ (to remind us that small-number statistics is affecting large lag times).

Computationally, we will do this for an grid of τ values *equally-spaced in log scale*. Let us also perform the calculation for all $n_{\text{samp}} = 10$ samples.

In [5]:
```
n_tau = 100
tau_vals = np.logspace(0, np.log10(9e5), n_tau)
delta_vals = np.round(tau_vals/dt).astype(int)

tau_vals
```

Out[5]:
```
array([1.00000000e+00, 1.14853402e+00, 1.31913040e+00, 1.51506614e+00,
       1.74010500e+00, 1.99856979e+00, 2.29542540e+00, 2.63637416e+00,
       3.02796541e+00, 3.47772129e+00, 3.99428121e+00, 4.58756786e+00,
       5.26897775e+00, 6.05160020e+00, 6.95046871e+00, 7.98284976e+00,
       9.16857453e+00, 1.05304198e+01, 1.20945453e+01, 1.38909968e+01,
       1.59542824e+01, 1.83240361e+01, 2.10457788e+01, 2.41717930e+01,
       2.77621265e+01, 3.18857468e+01, 3.66218650e+01, 4.20614578e+01,
       4.83090152e+01, 5.54845475e+01, 6.37258903e+01, 7.31913530e+01,
       8.40627589e+01, 9.65489385e+01, 1.10889740e+02, 1.27360639e+02,
       1.46278027e+02, 1.68005291e+02, 1.92959792e+02, 2.21620885e+02,
       2.54539126e+02, 2.92346846e+02, 3.35770298e+02, 3.85643611e+02,
       4.42924807e+02, 5.08714209e+02, 5.84275575e+02, 6.71060375e+02,
       7.70735671e+02, 8.85216138e+02, 1.01670085e+03, 1.16771551e+03,
       1.34116099e+03, 1.54036903e+03, 1.76916623e+03, 2.03194761e+03,
       2.33376095e+03, 2.68040385e+03, 3.07853501e+03, 3.53580219e+03,
       4.06098910e+03, 4.66418414e+03, 5.35697416e+03, 6.15266707e+03,
       7.06654744e+03, 8.11617014e+03, 9.32169752e+03, 1.07062867e+04,
       1.22965345e+04, 1.41229882e+04, 1.62207325e+04, 1.86300631e+04,
       2.13972612e+04, 2.45754825e+04, 2.82257777e+04, 3.24182659e+04,
       3.72334812e+04, 4.27639199e+04, 4.91158168e+04, 5.64111866e+04,
       6.47901669e+04, 7.44137108e+04, 8.54666784e+04, 9.81613878e+04,
       1.12741693e+05, 1.29487670e+05, 1.48720995e+05, 1.70811122e+05,
       1.96182384e+05, 2.25322143e+05, 2.58790146e+05, 2.97229287e+05,
       3.41377948e+05, 3.92084187e+05, 4.50322027e+05, 5.17210168e+05,
       5.94033474e+05, 6.82267654e+05, 7.83607611e+05, 9.00000000e+05])
```

Similar to what we did when calculating the PDFs, we get the displacements for all τ values and samples.

In [6]:
```
# initialize array to store displacements
# note that the size n is larger than the actual number of values to be
stored
# values are initialized to NaN (which are not included in the MSD calcu
lation)
dx_tau = np.empty((n, n_tau, n_samp))*np.nan

for i_samp in np.arange(n_samp):
  for i, tau in enumerate(tau_vals):
    delta = delta_vals[i]

    # get truncated copy of x, ending in initial data point of the last
pair
    x_trunc = x_samp[:-1*delta, i_samp]

    # get shifted copy of x, starting from end data point of the first p
air
    x_shift = x_samp[delta:, i_samp]

    # get displacements
    dx = x_shift - x_trunc

    # store in output array
    dx_tau[:len(dx), i, i_samp] = dx

    #print(i, tau, delta, len(dx))
```

In [7]:
```
np.shape(dx_tau)
```

Out[7]:
```
(1000000, 100, 10)
```

Next, we get the mean of the squared displacements for each value of τ:

In [18]:
```
# initialize array to store MSD values
msd_tau = np.empty((n_tau, n_samp))*np.nan

for i_samp in np.arange(n_samp):
  for i, tau in enumerate(tau_vals):
    dx2_sum = np.nansum(dx_tau[:, i, i_samp]**2) # returns sum treating
NaNs as zero
    denom = n-delta_vals[i]
    msd_tau[i, i_samp] = dx2_sum/denom
```

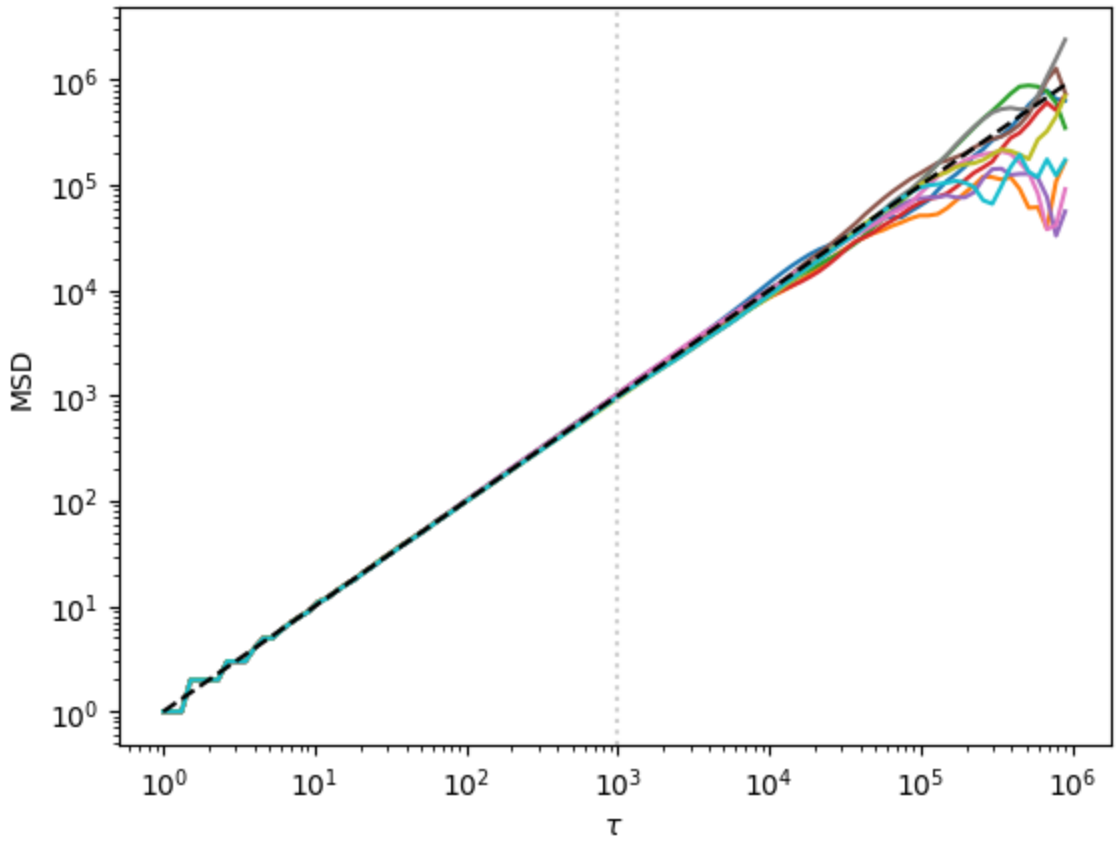
In [19]:
```
np.shape(msd_tau)
```

Out[19]:
```
(100, 10)
```

Finally, we plot the MSD as a function of τ for the different samples and overlay the analytical result MSD $(\tau) = \tau$.

In [24]:
```
# plot MSD for all realizations
for i_samp in np.arange(n_samp):
  plt.plot(tau_vals, msd_tau[:, i_samp])

# plot analytical result
msd_theo = tau_vals
plt.plot(tau_vals, msd_theo, 'k--')

# plot vertical line at tau=10^3
tau_ref = 1e3
plt.axvline(tau_ref, color='0.8', ls=':')

plt.xscale("log")
plt.yscale("log")
plt.xlabel(r"$\tau$")
plt.ylabel("MSD")
```

Out[24]: Text(0, 0.5, 'MSD')



We find deviations from the analytical result for large values of τ, as expected from the smaller number of data point pairs used in this regime.

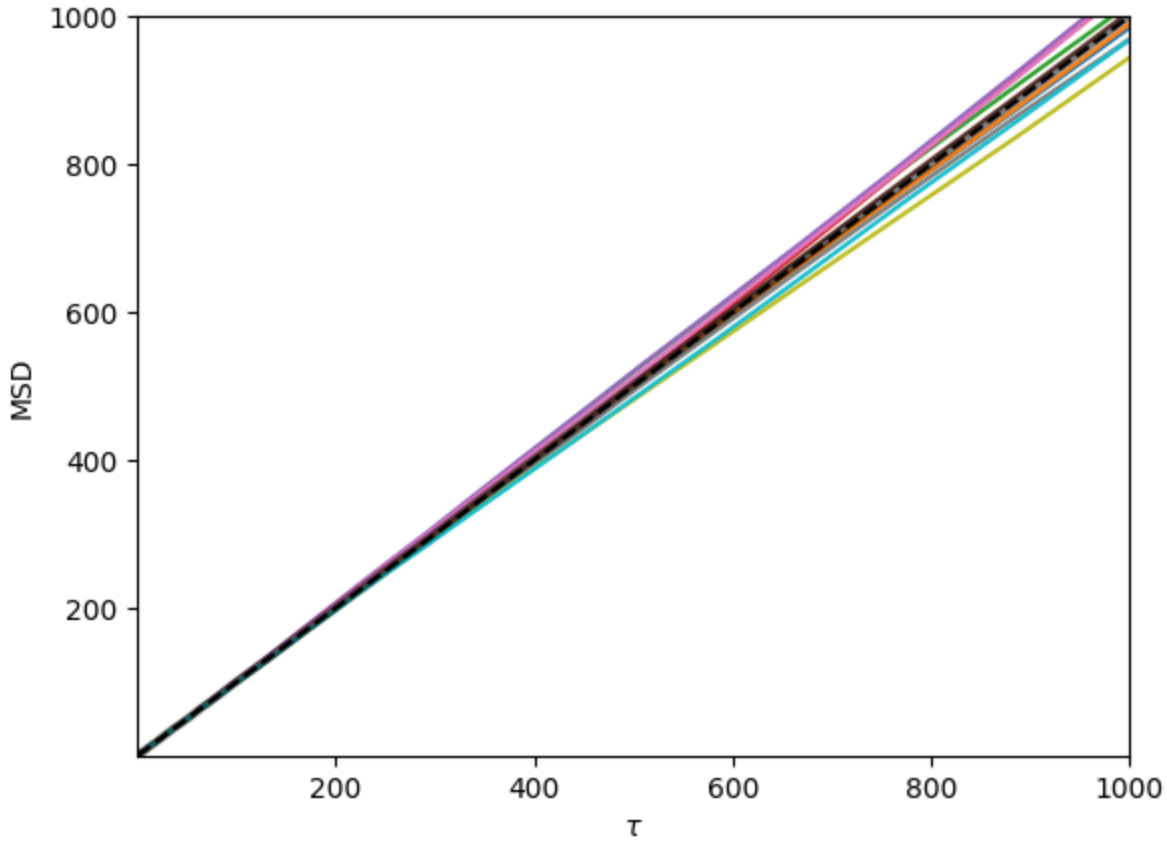
Zooming in on $\tau < 10^3$, we find deviations from the analytical result for individual simulations, as expected, but a good match to the average MSD curve (over all realizations).

In [31]:
```
# get the average MSD over all realizations
msd_mean = np.mean(msd_tau, axis=1)
np.shape(msd_mean)
```

Out[31]:
```
(100,)
```

In [32]:
```
# plot MSD for all realizations
for i_samp in np.arange(n_samp):
  plt.plot(tau_vals, msd_tau[:, i_samp])

# plot average MSD over all realizations
plt.plot(tau_vals, msd_mean, color='k', lw=3, alpha=0.5)

# plot analytical result
msd_theo = tau_vals
plt.plot(tau_vals, msd_theo, 'k--')

#plt.xscale("log")
#plt.yscale("log")
plt.xlabel(r"$\tau$")
plt.ylabel("MSD")

plt.xlim((1, 1e3))
plt.ylim((1, 1e3))
```

Out[32]:
```
(1.0, 1000.0)
```



In [42]:
```
# plot residuals from analytical result

# plot MSD for all realizations
for i_samp in np.arange(n_samp):
  plt.plot(tau_vals, msd_tau[:, i_samp]-msd_theo)

# plot average MSD over all realizations
plt.plot(tau_vals, msd_mean-msd_theo, color='k', lw=3, alpha=0.5)

# plot horizontal line at zero
plt.axhline(0., color='k', ls=':')

plt.xlabel(r"$\tau$")
plt.ylabel("MSD")

plt.xlim((1, 1e3))
plt.ylim((-55, 55))
```

Out[42]: (-55.0, 55.0)