Physics 305 Demo Notebook 4: Calculating the Displacement Probability Distribution (PDF) from a Time-Series - Exoplanet Light Curve

We apply the stochastic process analysis to the light curve dataset of Kepler 1-b observed by TESS.

First, we read the light curve data and plot the timeseries.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:
```
fn = "tess_lc_kepler1b.txt"
df = pd.read_csv(fn, header=0)
df.head(2)
```

Out[4]:

	# Time (BTJD)	Normalized SAP_FLUX	Normalized SAP_BKG
0	1683.405757	0.999869	0.008537
1	1683.426590	0.998307	0.008400

In [5]:
```
df.columns = ["t", "f", "e"]
df.head(2)
```

Out[5]:

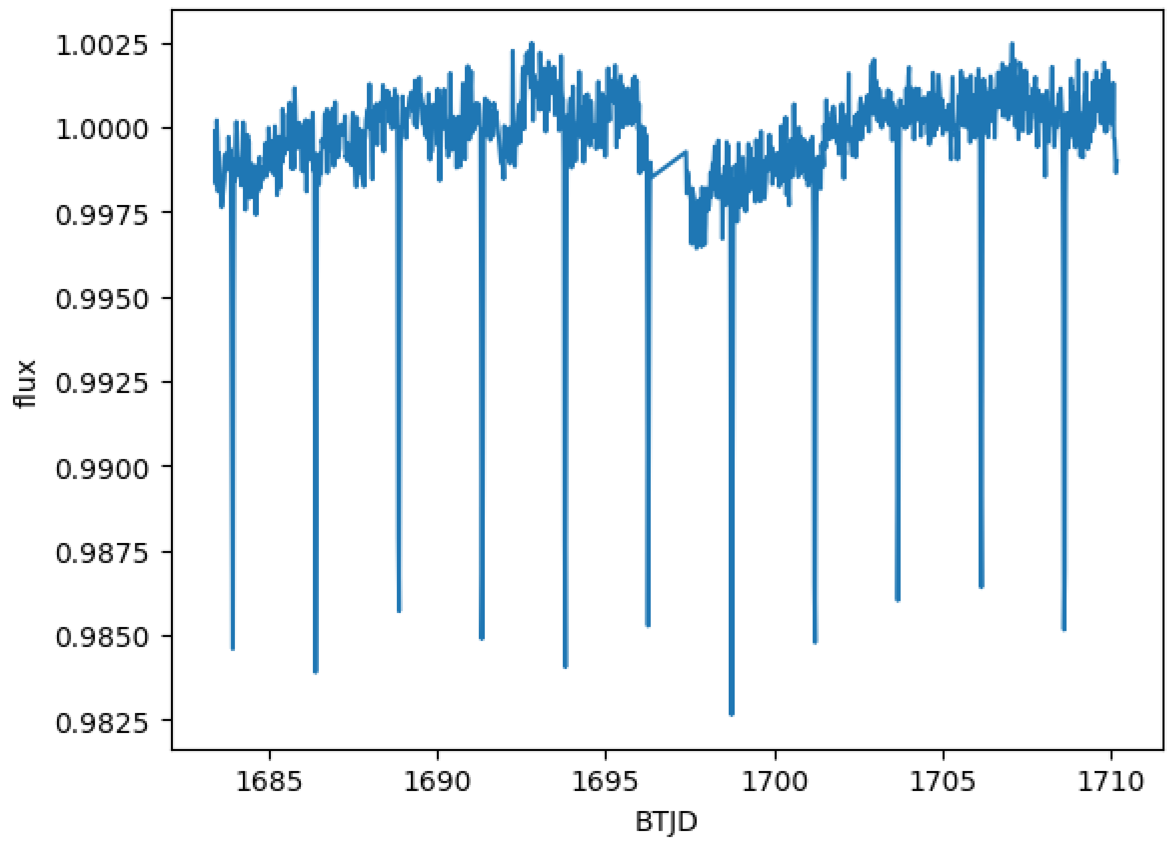
	t	f	e
0	1683.405757	0.999869	0.008537
1	1683.426590	0.998307	0.008400

In [6]:
```
len(df)
```

Out[6]: 1171

In [7]:
```
plt.plot(df['t'], df['f'])
plt.xlabel("BTJD")
plt.ylabel("flux")
```

Out[7]:
```
Text(0, 0.5, 'flux')
```



In [8]:
```
f_thresh = 0.9875
t_transit = df.loc[df['f']<=f_thresh, 't'].values
```

In [9]:
```
t_transit
```

Out[9]:
```
array([1683.92659508, 1686.38495208, 1688.86414229, 1691.32249884,
       1691.34333237, 1693.8016885 , 1696.2600432 , 1698.73924098,
       1698.76007459, 1701.19760673, 1701.21844034, 1703.67680558,
       1706.15600399, 1708.61436786, 1708.63520145])
```

In [10]:
```
deltat_transit_vals = t_transit[1:] - t_transit[:-1]
deltat_transit_vals
```

Out[10]:
```
array([2.458357  , 2.47919021, 2.45835655, 0.02083353, 2.45835613,
       2.4583547 , 2.47919778, 0.02083361, 2.43753214, 0.02083361,
       2.45836525, 2.47919841, 2.45836387, 0.02083358])
```

In [32]:
```
t_orbit = np.mean(deltat_transit_vals[deltat_transit_vals > 2.0])
t_orbit
```

Out[32]:
```
2.462527204276239
```

The time between transits is around 2.46 days-- this is an estimate of the orbital period of Kepler 1-b.

We note that the time interval between data points is not uniform-- and there is a gap in the dataset.

In [12]:
```
deltat = df["t"].values[1:] - df["t"].values[:-1]
```

In [13]:
```
pd.Series(deltat).describe()
```

Out[13]:
```
count    1170.000000
mean        0.022881
std         0.029862
min         0.020834
25%         0.020834
50%         0.020834
75%         0.020834
max         1.020845
dtype: float64
```

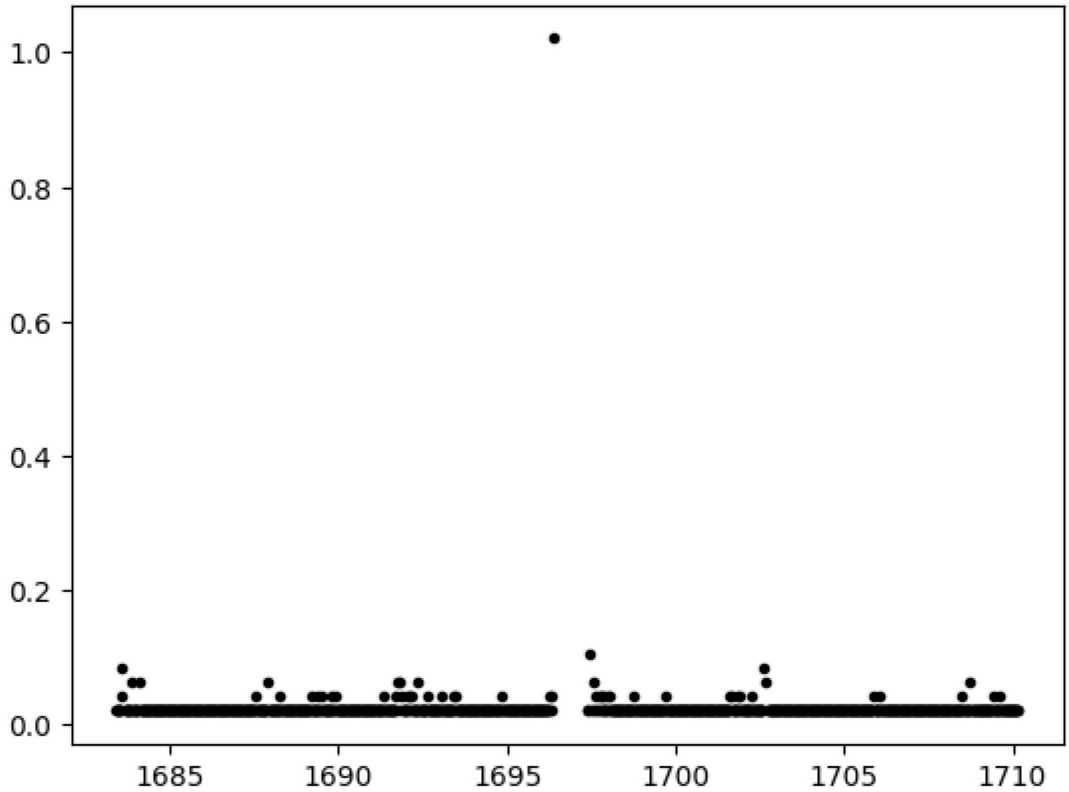
In [14]:
```
dt = np.mean(deltat)
dt
```

Out[14]:
```
0.022881307263408524
```

The mean time interval is 0.022881 days or around 32.9 minutes.

In [15]:
```
plt.plot(df.iloc[:-1]["t"].values, deltat, 'k.')
```

Out[15]: [<matplotlib.lines.Line2D at 0x7c7a8b0255a0>]



In [16]:
```
deltat_thresh = 0.8
t_gap = df.iloc[:-1].loc[deltat>deltat_thresh, "t"].values[0]
```

In [17]:
```
t_gap
```

Out[17]: 1696.364210804781

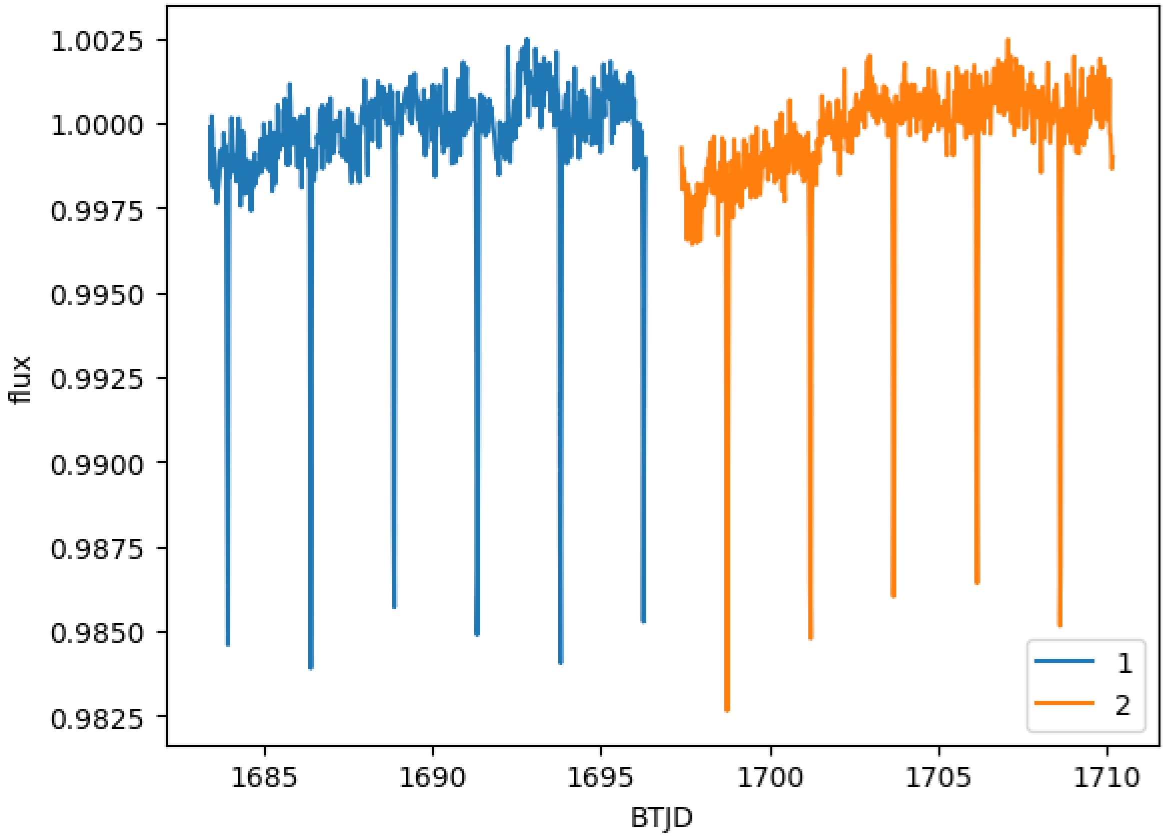
To get around the gap in the dataset, let us split the dataset into two-- before and after the gap.

In [18]:
```
df1 = df.loc[df["t"] < t_gap]
df2 = df.loc[df["t"] > t_gap]
n1, n2 = len(df1), len(df2)
n1, n2, n1+n2, len(df)
```

Out[18]: (586, 584, 1170, 1171)

In [19]:
```
plt.plot(df1['t'], df1['f'], label="1")
plt.plot(df2['t'], df2['f'], label="2")
plt.xlabel("BTJD")
plt.ylabel("flux")
plt.legend(loc="best")
```

Out[19]: <matplotlib.legend.Legend at 0x7c7a8ae8f4f0>

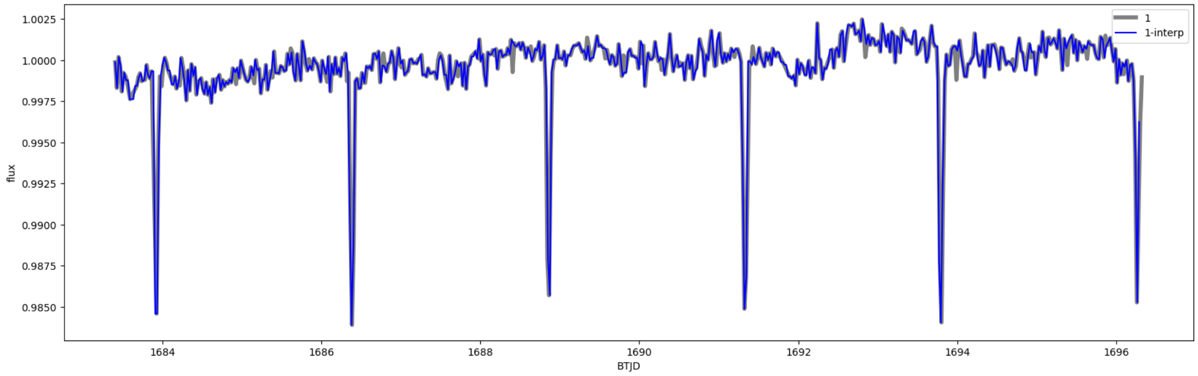


We perform interpolation to make the time variable uniform.

In [33]:
```
import scipy.interpolate as interp
interp_method = "nearest"

n1_interp = np.floor((df1["t"].max() - df1["t"].min())/dt)
f1 = interp.interp1d(df1["t"], df1["f"], kind=interp_method)
tt1 = np.arange(n1_interp)*dt + df1["t"].min()
ff1 = f1(tt1)

n2_interp = np.floor((df2["t"].max() - df2["t"].min())/dt)
f2 = interp.interp1d(df2["t"], df2["f"], kind=interp_method)
tt2 = np.arange(n2_interp)*dt + df2["t"].min()
ff2 = f2(tt2)
```

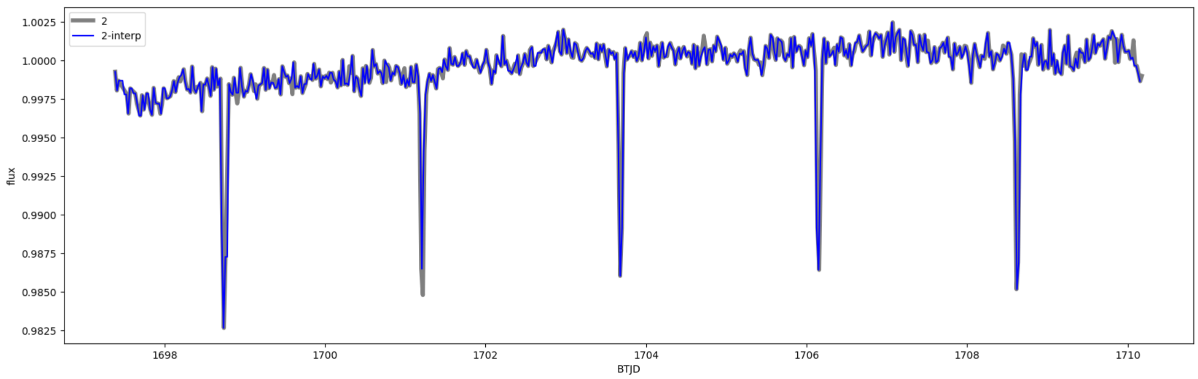
In [35]:
```
plt.figure(figsize=(20,6))
plt.plot(df1['t'], df1['f'], label="1", color='k', alpha=0.5, lw=4)
plt.plot(tt1, ff1, color='b', label="1-interp")
plt.xlabel("BTJD")
plt.ylabel("flux")
plt.legend(loc="best")
```

Out[35]: <matplotlib.legend.Legend at 0x7c7a8443aa40>



In [36]:
```
plt.figure(figsize=(20,6))
plt.plot(df2['t'], df2['f'], label="2", color='k', alpha=0.5, lw=4)
plt.plot(tt2, ff2, color='b', label="2-interp")
plt.xlabel("BTJD")
plt.ylabel("flux")
plt.legend(loc="best")
```

Out[36]: <matplotlib.legend.Legend at 0x7c7a81a5dfc0>



In [58]:
```
f_mean, f_min = df['f'].mean(), df['f'].min()
f_mean - f_min, f_mean, f_min
```

Out[58]: (0.016978163551521 82, 0.9996260365540742, 0.9826478730025524)

Let us calculate and plot the PDF for time lag τ = 1 for the first time series.

In [78]:
```
tau = dt
delta = int(np.round(tau/dt))
x = ff1
print(delta)
```

```
1
```

In [79]:
```
# get truncated copy of x, ending in initial data point of the last pair
x_trunc = x[:-1*delta]

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

# get displacements
dx = x_shift - x_trunc
```

In [80]:
```
np.sum(dx**2)/len(dx)
```

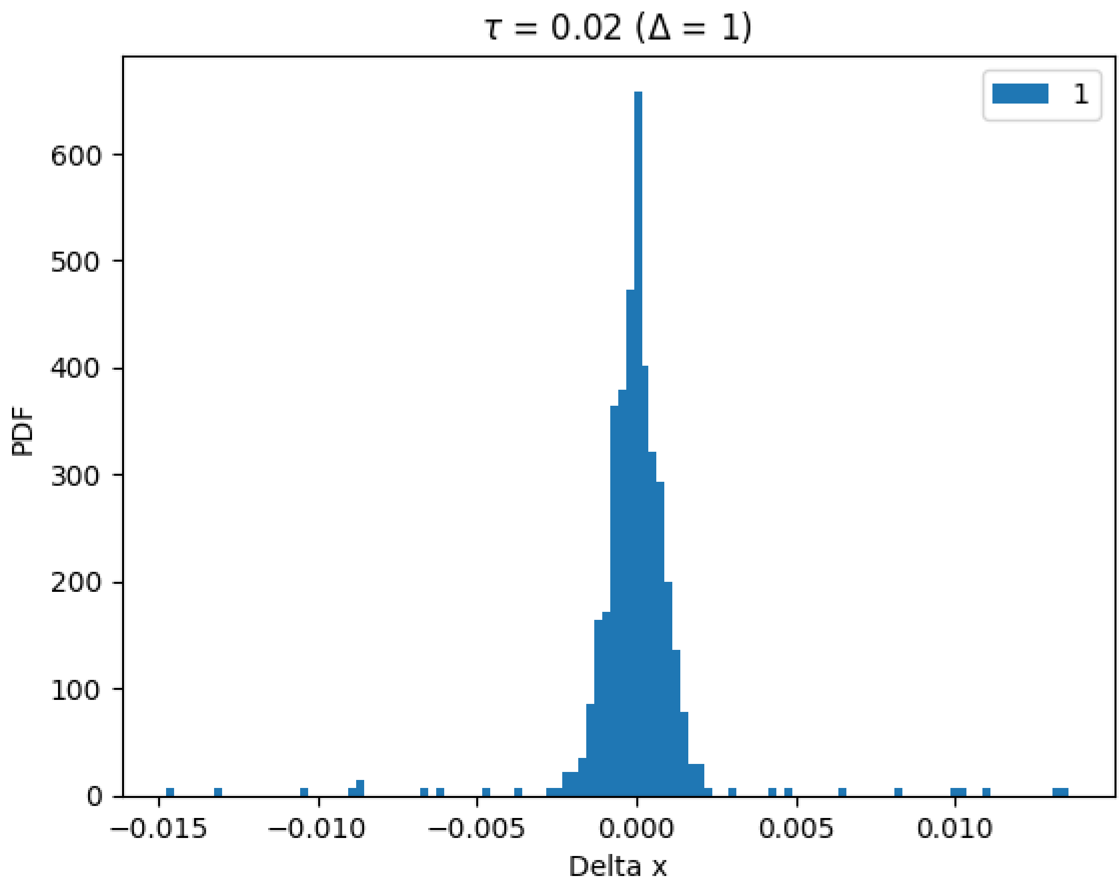
Out[80]:
```
3.585444038258923e-06
```

In [73]:
```
len(dx), len(df1)
```

Out[73]:
```
(563, 586)
```

In [76]:
```
# Plot PDF of displacements
plt.hist(dx, density=True, bins="auto", label="1")
plt.xlabel("Delta x")
plt.ylabel("PDF")
plt.legend(loc="best")
plt.title(r"$\tau$ = %.2f ($\Delta$ = %d)" % (tau, delta))
```

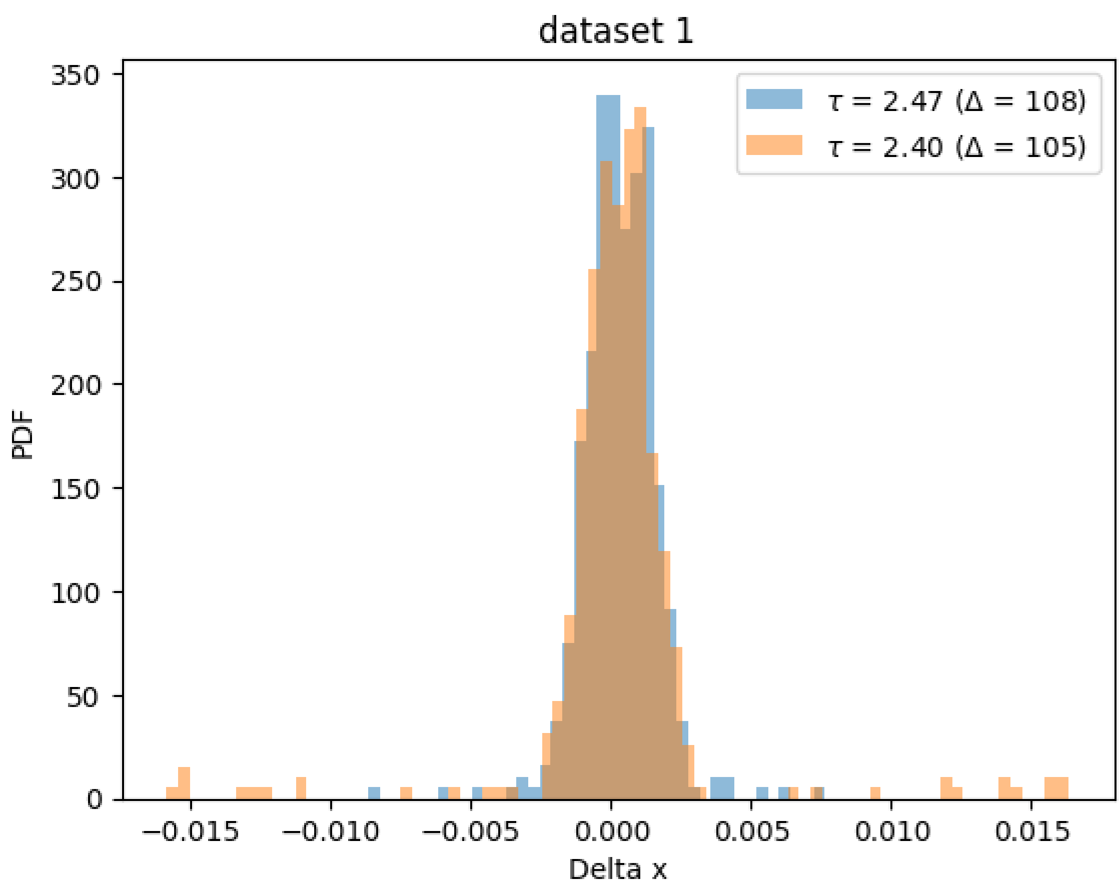
Out[76]:
```
Text(0.5, 1.0, '$\\tau$ = 0.02 ($\\Delta$ = 1)')
```



Let us calculate and plot the PDF for time lag corresponding to the orbital period τ = 2.47 (Δ = 108) and a smaller lag time τ = 2.40 (Δ = 105), for the first time series.

In [111]:
```
#tau = dt
#delta = int(np.round(tau/dt))
delta = 108 # corresponding to lag time equal to the orbital period
tau = dt*delta
x = ff1
deltaA, tauA = delta, tau

# get truncated copy of x, ending in initial data point of the last pair
x_trunc = x[:-1*delta]
# get shifted copy of x, starting from end data point of the first pair
x_shift = x[delta:]
# get displacements
dx = x_shift - x_trunc
dxA = dx

delta = 105
tau = dt*delta
deltaB, tauB = delta, tau

# get truncated copy of x, ending in initial data point of the last pair
x_trunc = x[:-1*delta]
# get shifted copy of x, starting from end data point of the first pair
x_shift = x[delta:]
# get displacements
dx = x_shift - x_trunc
dxB = dx

# Plot PDF of displacements
labelA = r"$\tau$ = %.2f ($\Delta$ = %d)" % (tauA, deltaA)
labelB = r"$\tau$ = %.2f ($\Delta$ = %d)" % (tauB, deltaB)
plt.hist(dxA, density=True, bins="auto", alpha=0.5, label=labelA)
plt.hist(dxB, density=True, bins="auto", alpha=0.5, label=labelB)
plt.xlabel("Delta x")
plt.ylabel("PDF")
plt.legend(loc="best")
plt.title("dataset 1")
```

Out[111]: Text(0.5, 1.0, 'dataset 1')



We notice that the outliers with large $|\Delta x|$ are not in the PDF for $\tau = 2.47$ because the data points corresponding to the transit times (dips in the light curve) are paired with one another and result in a smaller dx-- which is not the case for other values of lag times (that are not multiples of the orbital period).

Now, let us calculate and plot the PDF for different lag times $\Delta = 1, 10, 100, 108, 216, 432$, where we included values coinciding with the orbital period and multiples of the orbital period (2x and 3x).

In [122]:
```
delta_orbit = np.floor(t_orbit/dt)
delta_orbit, t_orbit

delta_vals = np.array([1, 10, 1e2, 108, 216, 432]).astype(int)
tau_vals = delta_vals*dt
n_tau = len(tau_vals)

print(delta_vals)
print(tau_vals)
```

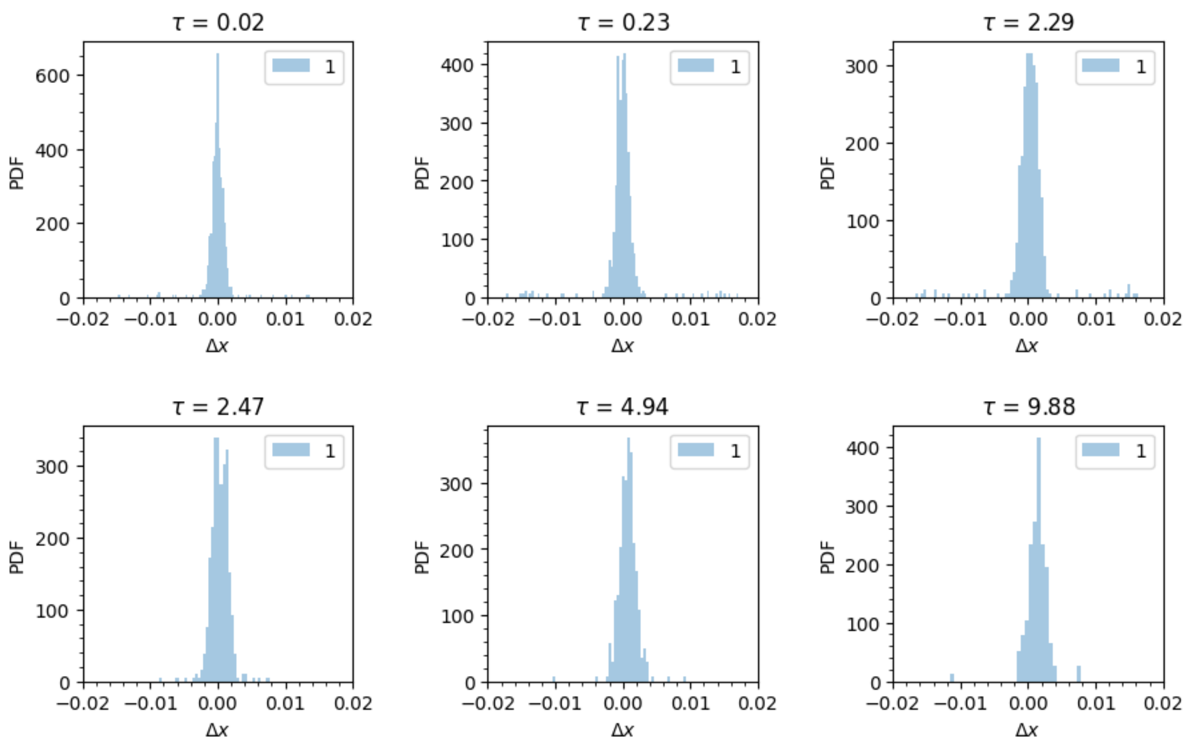
```
[  1  10 100 108 216 432]
[0.02288131 0.22881307 2.28813073 2.47118118 4.94236237 9.88472474]
```

In [123]:
```
n_samp = 2
n = np.max([n1, n2])
x_samp = np.zeros((n, n_samp))*np.nan
x_samp[:len(ff1), 0] = ff1
x_samp[:len(ff2), 1] = ff2
```

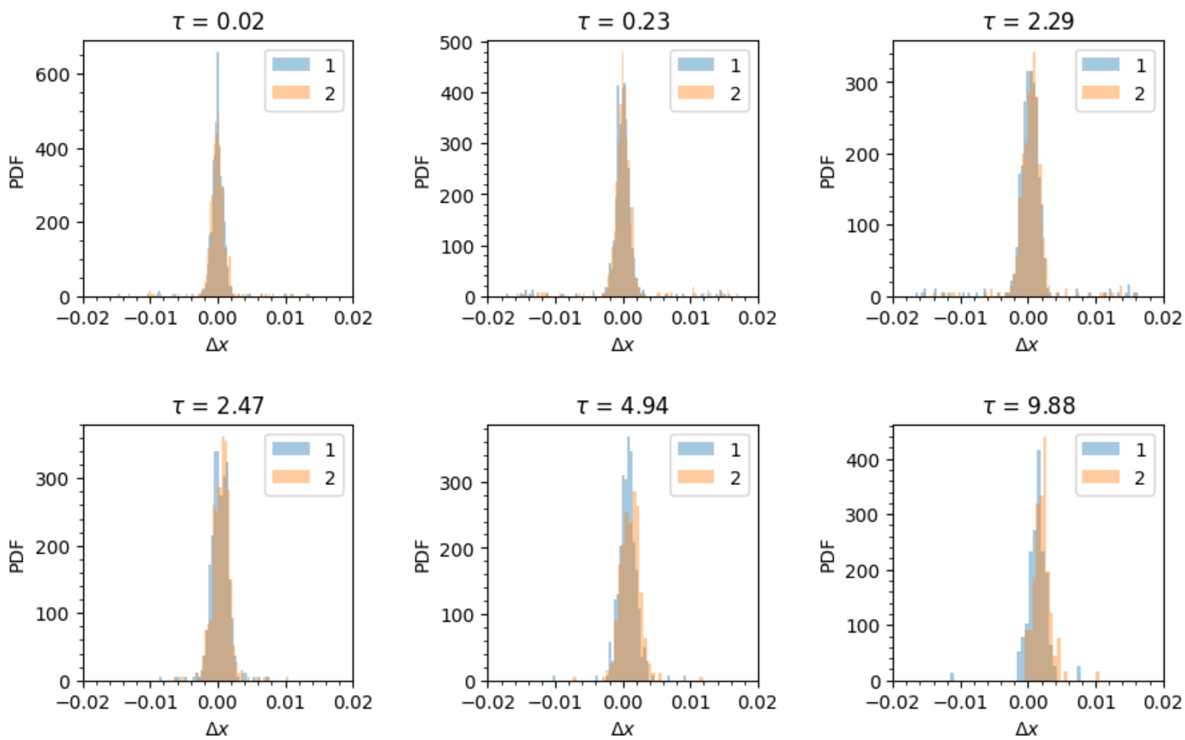
In [124]:
```
# initialize array to store displacements
# note that the size n is larger than the actual number of values to be
stored
# values are initialized to NaN (which are not included in the PDF calcu
lation)
dx_tau = np.empty((n, n_tau, n_samp))*np.nan

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 [125]:
```
# define plotting function
def plot_pdf(dx, tau, label=""):

  # Plot PDF of displacements
  plt.hist(dx, density=True, bins="auto", alpha=0.4, label=label)
  plt.xlabel("Delta x")
  plt.ylabel("PDF")

  plt.ylabel("PDF")
  plt.xlabel(r"$\Delta x$")
  #plt.legend(loc="best")
  plt.title(r"$\tau$ = %.2f" % tau)
```

In [126]:
```
# generate PDF plots for different values of tau
i_samp = 0
nx = 2
ny = 3

plt.figure(figsize=(12,6))
for i in np.arange(n_tau):
  plt.subplot(nx,ny,i+1)
  plot_pdf(dx_tau[:,i,i_samp], tau_vals[i], label="%d" % (i_samp+1))
  plt.xlim((-0.02, 0.02))
  plt.minorticks_on()
  plt.legend(loc="upper right")

plt.subplots_adjust(bottom=0.1, right=0.8, top=0.9, hspace=0.5, wspace=
0.5)
```



In [128]:
```
# generate plot showing PDF of both timeseries

plt.figure(figsize=(12,6))
for i in np.arange(n_tau):
  plt.subplot(nx,ny,i+1)
  for i_samp in np.arange(n_samp):
    plot_pdf(dx_tau[:,i,i_samp], tau_vals[i], label="%d" % (i_samp+1))
    plt.xlim((-0.02, 0.02))
    plt.minorticks_on()
  plt.legend(loc="upper right")
plt.subplots_adjust(bottom=0.1, right=0.8, top=0.9, hspace=0.5, wspace=
0.5)
```



Next, let us get the MSD curves.

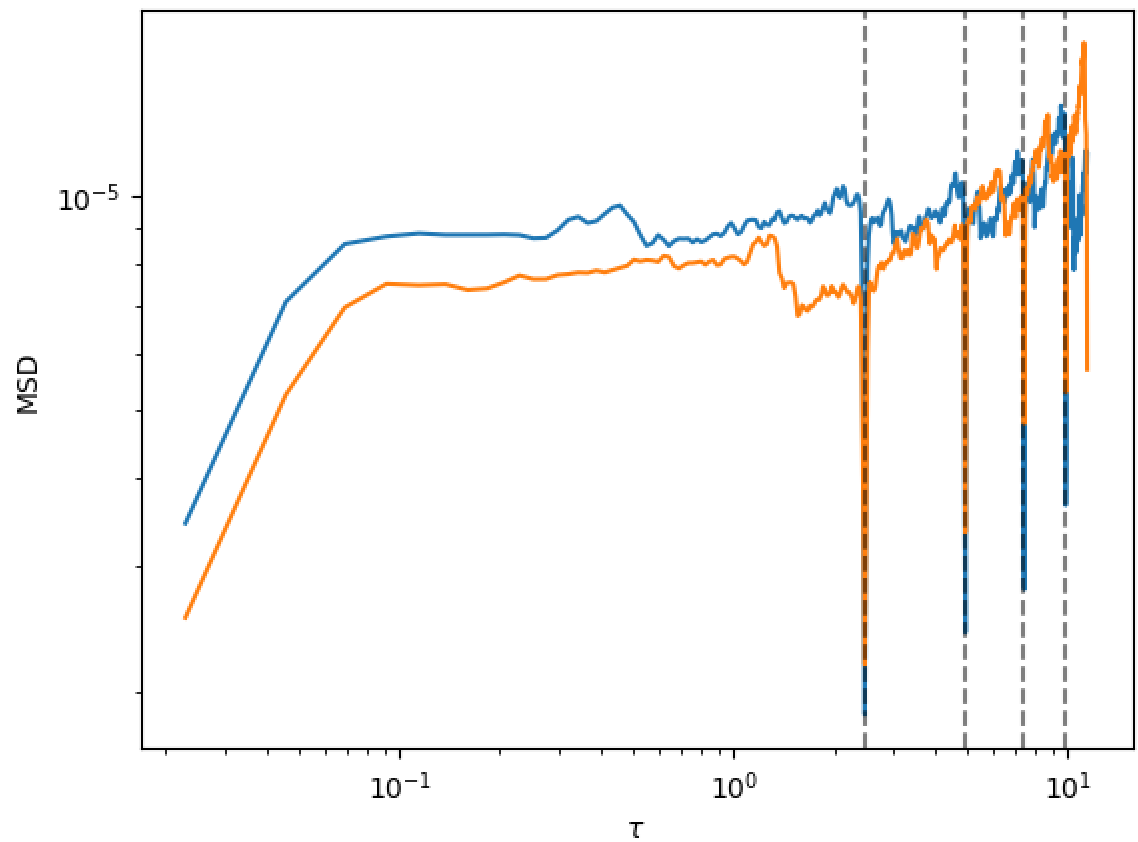
In [105]:
```
n_tau = 500
delta_vals = np.arange(n_tau).astype(int)+int(1)
#delta_vals = np.round(tau_vals/dt).astype(int)
tau_vals = delta_vals*dt
#tau_vals, delta_vals
```

In [101]:
```
# 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 [102]:
```
# 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 [103]:
```
# plot MSD curve
for i_samp in np.arange(n_samp):
  plt.plot(tau_vals, msd_tau[:, i_samp], label="%d" % (i_samp+1))

plt.axvline(t_orbit, color='k', alpha=0.5, ls="--")
plt.axvline(t_orbit*2, color='k', alpha=0.5, ls="--")
plt.axvline(t_orbit*3, color='k', alpha=0.5, ls="--")
plt.axvline(t_orbit*4, color='k', alpha=0.5, ls="--")

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

Out[103]: Text(0, 0.5, 'MSD')



We find features in the MSD-- sharp drops-- occuring at lag times equal to the orbital period and its multiples (2x, 3x, and 4x).