Physics 305 Demo Notebook 6: Analyzing Sunspot Timeseries Data as a Stochastic Process

In this Demo Notebook, we apply stochastic process analysis to a dataset of sunspot numbers for 1 solar cycle (SC-24). The same analysis can be readily applied to the other solar cycles as well.

These are the steps:

1. Data preprocessing - includes interpolation, imputation, filtering to 1 solar cycle, fitting to the Hathaway function; we take our x(t) to be the residuals from the best-fit Hathaway function
2. Calculating the MSD and fitting the MSD curve with theoretical MSD model (Eq. 5 below)
3. Calculating the displacement PDFs and comparing with Gaussian model with width given by the theoretical MSD curve

This is based on data and scripts provided by Reynan Toledo (USC).

The probability density function (PDF) has the form,

$$P(x_1, t; x_0, 0) = \frac{1}{\sqrt{2\pi(MSD)}} \exp\left[\frac{-(x_1 - x_0)^2}{2(MSD)}\right] \quad , \tag{1}$$

where the mean square deviation (MSD) is given by:

$$MSD = g(t)^2 \int_0^t [f(t-\tau)\, h(\tau)]^2 d\tau \quad , \tag{2}$$

with t a constant final time in Eq. (2). Functions $g(t), f(t-\tau)$, and $h(\tau)$ determine the type or behavior of the stochastic process.

The following MSD's can be plugged-in to Eq. (1):

1) Ordinary Brownian motion (Wiener process):

$$MSD = 2Dt \quad (D \text{ is a constant diffusion coefficient}) \tag{3}$$

2) Fractional Brownian Motion:

$$MSD = \frac{t^{2H}}{2H\left[\Gamma\left(H+\frac{1}{2}\right)\right]^2} \tag{4}$$

The H is Hurst exponent, $0 \leq H \leq 1$, and $\Gamma(\alpha)$ is the Gamma function.

3) What we are using for Sunspots:

$$MSD = a\,\Gamma(\mu) t^{\mu-1} \beta^{-\mu} exp(-\beta/t) \quad . \tag{5}$$

N.B. The theoretical form of the MSD we have selected for the sunspots dataset is given by Eq. (5) above.

Step 1a: Read in sunspots data and apply pre-processing

In [213]:
```
# import libraries
import numpy as np
import random
import matplotlib.pyplot as plt
%matplotlib inline
import pickle
from scipy.optimize import curve_fit
import pandas as pd
import math

# Set random seed
np.random.seed(seed=17)
```

In [164]:
```
fn = "sunspot_data.csv"
df = pd.read_csv(fn)
```

In [165]:
```
n_init = len(df)
print("Size of original dataset: %d" % n_init)
```

```
Size of original dataset: 74906
```

In [166]:
```
df.head(2)
```

Out[166]:

	date	Year	Month	Day	Date In Fraction Of Year	Number of Sunspots	Standard Deviation	Observations	Indicator
0	0	1818	1	1	1818.001	-1	-1.0	0	1
1	1	1818	1	2	1818.004	-1	-1.0	0	1

In [167]:
```
df.columns
```

Out[167]:
```
Index(['date', 'Year', 'Month', 'Day', 'Date In Fraction Of Year',
       'Number of Sunspots', 'Standard Deviation', 'Observations',
       'Indicator'],
      dtype='object')
```

In [168]:
```
# rename columns
df.rename(columns={"Date In Fraction Of Year": "t", "Number of Sunspots": '
df.rename(columns={"Year": "year", "Month": "month", "Day": "day"}, inplace
```

In [169]:
```
# define datetime column
df["datetime"] = pd.to_datetime(df[["year","month","day"]])
df.head(2)
```

Out[169]:

	date	year	month	day	t	n	err	Observations	Indicator	datetime
0	0	1818	1	1	1818.001	-1	-1.0	0	1	1818-01-01
1	1	1818	1	2	1818.004	-1	-1.0	0	1	1818-01-02

In [170]:
```
df.tail(2)
```

Out[170]:

	date	year	month	day	t	n	err	Observations	Indicator	datetime
74904	74904	2023	1	30	2023.081	75	13.2	39	0	2023-01-30
74905	74905	2023	1	31	2023.084	77	14.0	24	0	2023-01-31

In [171]:
```
# check that there is no missing date
ndays = (df["datetime"].max() - df["datetime"].min()).days+1
ndays, n_init, df["datetime"].min(), df["datetime"].max()
```

Out[171]:
```
(74906,
 74906,
 Timestamp('1818-01-01 00:00:00'),
 Timestamp('2023-01-31 00:00:00'))
```

In [172]:
```
# replace null values with np.nan
df["n"].replace(-1, np.nan, inplace = True)
df["err"].replace(-1, np.nan, inplace = True)
```

In [173]:
```
# get range of dates with non-null data
df1 = df.loc[(df["n"].isnull()==False) & (df["err"].isnull()==False)]
len(df1), df1["datetime"].min(), df1["datetime"].max()
```

Out[173]:
```
(71659, Timestamp('1818-01-08 00:00:00'), Timestamp('2023-01-31 00:00:0
0'))
```

In [174]:
```
# divide the dataset according to the different solar cycles
# from SC-6 to SC-25 (note that there is only partial data for SC-6 and SC-
li=[1976,5813,9341,13880,17987,22280,26388,30711,34909,38591,42274,
    46082,49794,53627,57800,61632,65255,69763,73781]
i_sc=[0]+li  # append zero to be the first index
sc_start = 6 # solar cycle of the first data point

# add column to indicate solar cycle
df1["sc"] = 0
for i in np.arange(len(i_sc)):
  if(i<len(i_sc)-1):
    df.loc[i_sc[i]:i_sc[i+1], "sc"] = sc_start+i
  else:
    df.loc[i_sc[i]:, "sc"] = sc_start+i
```

```
<ipython-input-174-23cba9a782bf>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  df1["sc"] = 0
```

In [175]:
```
df.head(2)
```

Out[175]:

	date	year	month	day	t	n	err	Observations	Indicator	datetime	sc
0	0	1818	1	1	1818.001	NaN	NaN	0	1	1818-01-01	6.0
1	1	1818	1	2	1818.004	NaN	NaN	0	1	1818-01-02	6.0

In [176]:
```
df.tail(2)
```

Out[176]:

	date	year	month	day	t	n	err	Observations	Indicator	datetime	sc
74904	74904	2023	1	30	2023.081	75.0	13.2	39	0	2023-01-30	25.0
74905	74905	2023	1	31	2023.084	77.0	14.0	24	0	2023-01-31	25.0

In [177]:
```
df["sc"].value_counts().sort_index()
```

Out[177]:
```
6.0     1976
7.0     3837
8.0     3528
9.0     4539
10.0    4107
11.0    4293
12.0    4108
13.0    4323
14.0    4198
15.0    3682
16.0    3683
17.0    3808
18.0    3712
19.0    3833
20.0    4173
21.0    3832
22.0    3623
23.0    4508
24.0    4018
25.0    1125
Name: sc, dtype: int64
```

In [178]:
```
# drop dates before the starting date with non-null data
date_start = df1["datetime"].min()
df2 = df.loc[(df["datetime"] >= date_start)]
n2 = len(df2)

n2, n_init, n_init-n2
```

Out[178]:
```
(74899, 74906, 7)
```

In [179]:
```
# get count of null values in n
n2_null_n = len(df2[df2["n"].isnull()])
n2_null_n, n2, n2_null_n*1./n2
```

Out[179]:
```
(3240, 74899, 0.043258254449325094)
```

In [180]:
```
# get count of null values in err
n2_null_err = len(df[df["err"].isnull()])
n2_null_err, n2, n2_null_err*1./n2
```

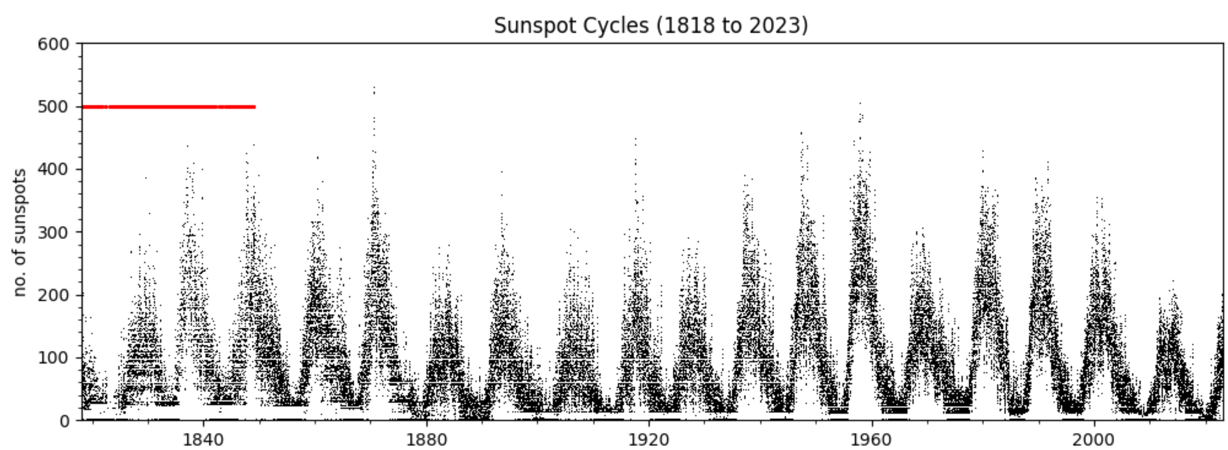
Out[180]:
```
(3247, 74899, 0.04335171364103666)
```

In [181]:
```
# define arrays to plot
date, t, x = df2["datetime"].values, df2["t"].values, df2["n"].values
n = len(t)

# get dates with null values for n
date_null = date[np.isnan(x)]
len(date_null), np.min(date_null), np.max(date_null)
```

Out[181]:
```
(3240,
 numpy.datetime64('1818-01-09T00:00:00.000000000'),
 numpy.datetime64('1848-12-22T00:00:00.000000000'))
```

In [182]:
```
# plot timeseries
plt.figure(figsize=(12,4))
plt.plot(date, x, marker=',', ls='', color='k')
#plt.errorbar(t, x, yerr=err, capsize=3, marker=',', ls='')

# plot red crosses where there is missing data
x_ref = 500
x_null_ref = np.zeros(len(date_null))+x_ref
plt.plot(date_null, x_null_ref, 'rx', ms=1)

plt.ylim((0, 600))
plt.xlim((np.min(date), np.max(date)))
plt.minorticks_on()
plt.ylabel("no. of sunspots")
plt.title("Sunspot Cycles (%d to %d)" % (1818, 2023))
```

Out[182]:
```
Text(0.5, 1.0, 'Sunspot Cycles (1818 to 2023)')
```

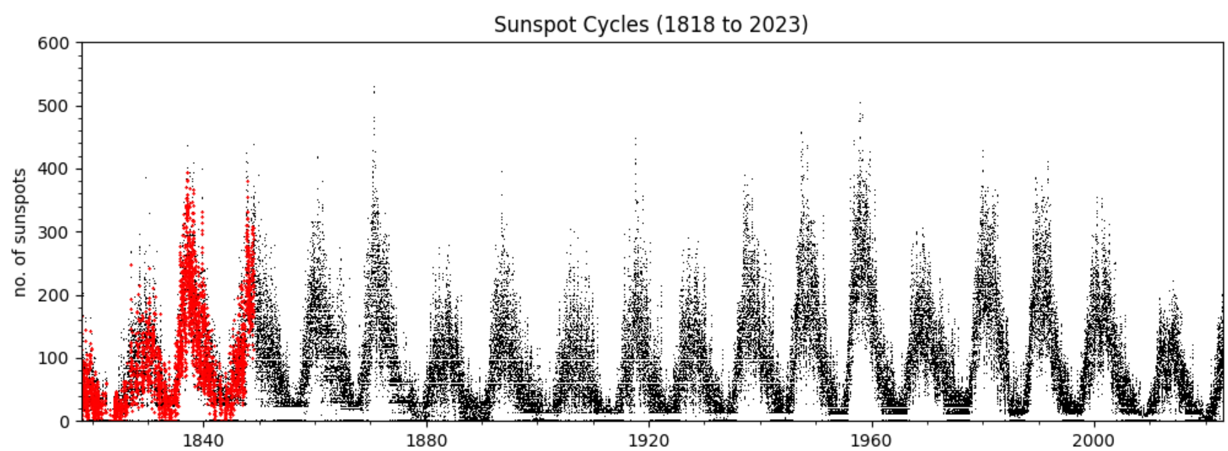


In [183]:
```
# interpolate to fill in missing n
df2["n_fill"] = df2["n"].interpolate(method="linear")
```

```
<ipython-input-183-b7f042d779d5>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  df2["n_fill"] = df2["n"].interpolate(method="linear")
```

In [184]:
```
# define arrays needed for plot
date, t, x0, x = df2["datetime"].values, df2["t"].values, df2["n"].values,
n = len(t)

# plot timeseries
plt.figure(figsize=(12,4))
plt.plot(date, x, marker=',', ls='', color='k')
#plt.errorbar(t, x, yerr=err, capsize=3, marker=',', ls='')

# plot red crosses for interpolated data
date_null = date[np.isnan(x0)]
x_null = x[np.isnan(x0)]
plt.plot(date_null, x_null, 'rx', ms=1)

plt.ylim((0, 600))
plt.xlim((np.min(date), np.max(date)))
plt.minorticks_on()
plt.ylabel("no. of sunspots")
plt.title("Sunspot Cycles (%d to %d)" % (1818, 2023))
```

Out[184]:
```
Text(0.5, 1.0, 'Sunspot Cycles (1818 to 2023)')
```



In [185]:
```
date_all, t_all, x_all, err_all = date, t, x, err # save full timeseries da
n_all = len(date_all)
```

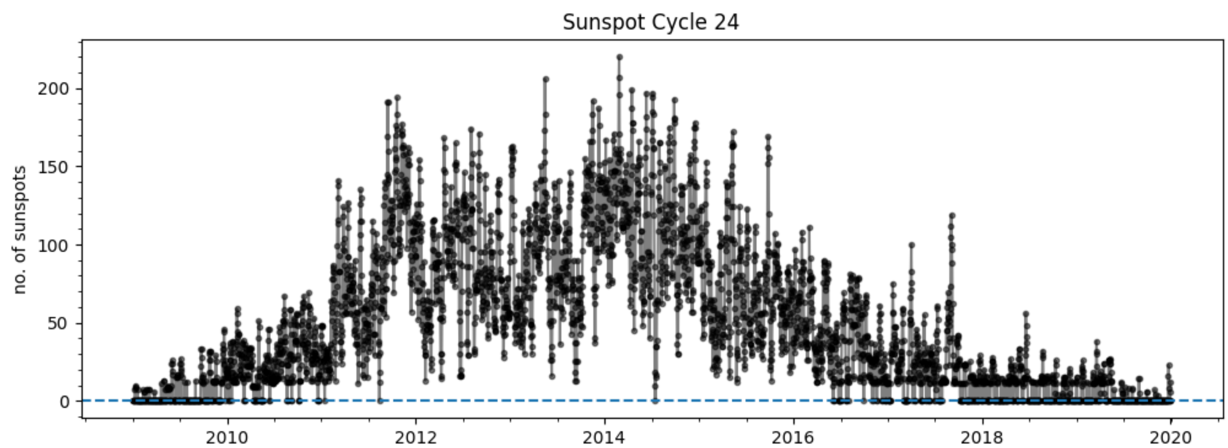
Let us focus on SC-24 and ignore errors for now

In [189]:
```
sc = 24
date = df2.loc[df2["sc"]==sc, "datetime"].values
t = df2.loc[df2["sc"]==sc, "t"].values
x = df2.loc[df2["sc"]==sc, "n_fill"].values
n_use = len(date)
print("SC-%d, sample size: %d" % (sc, n_use))
```

```
SC-24, sample size: 4018
```

In [190]:
```
# plot timeseries
plt.figure(figsize=(12,4))
plt.plot(date, x, marker='.', ls='-', color='k', alpha=0.5)
plt.axhline(0., ls='--')
plt.minorticks_on()
plt.ylabel("no. of sunspots")
plt.title("Sunspot Cycle 24")
```

Out[190]:
```
Text(0.5, 1.0, 'Sunspot Cycle 24')
```



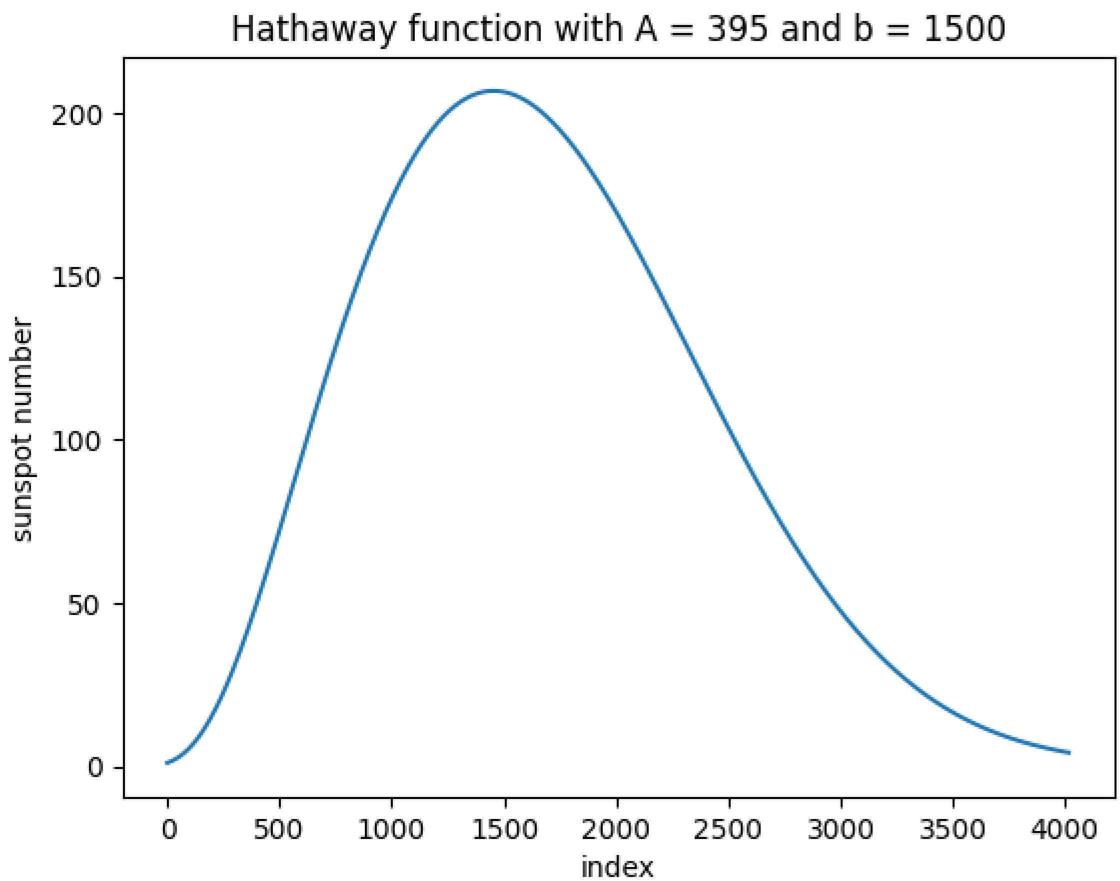
Step 1b: Fit Hathaway function and get residuals-- which will be our variable of interest, x(t)

First, we define and plot the Hathaway function.

In [191]:
```
def hathaway(x, A, b):
    return A*(((x+120)/b)**3)/(np.exp(((x+120)/b)**2)-0.8)
```

In [192]:
```
xx = np.arange(n_use)
A, b = [395, 1500]
yy = hathaway(xx, A, b)
plt.plot(xx, yy)
plt.xlabel("index")
plt.ylabel("sunspot number")
plt.title("Hathaway function with A = %d and b = %d" % (A, b))
```

Out[192]:
```
Text(0.5, 1.0, 'Hathaway function with A = 395 and b = 1500')
```



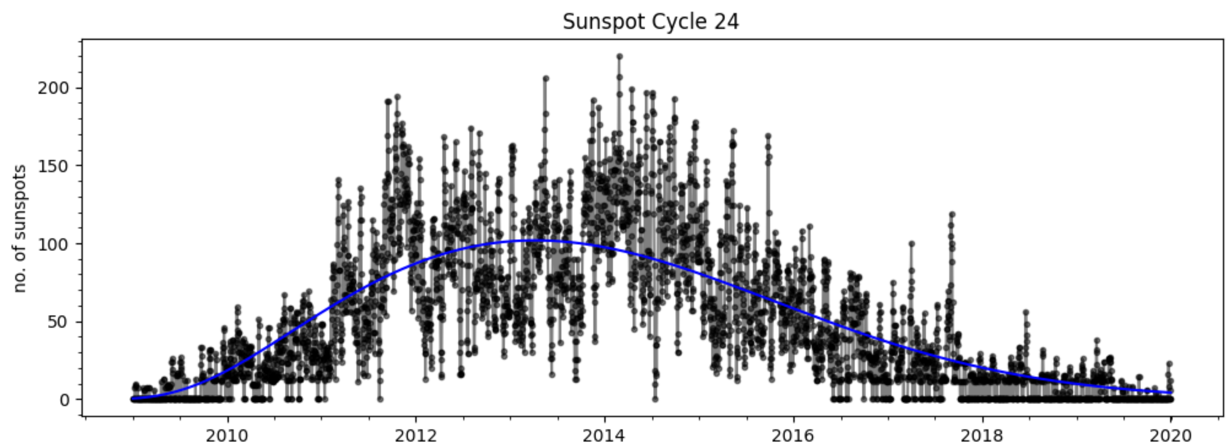
Let us fit the Hathaway function to our data.

In [196]:
```
x_fit = np.arange(n_use)
y_fit = x
err_fit = err # set error to a constant value
initial = [395, 1500]
maxfev = 5000

popt, pcov = curve_fit(hathaway, x_fit, y_fit, initial, maxfev=maxfev)

fit_A, fit_b = popt[0], popt[1]
err_A, err_b = pcov[0,0]**0.5, pcov[1,1]**0.5
print("A = %.2f (%.2f), b = %.2f (%.2f)" % (fit_A, err_A, fit_b, err_b))
```

```
A = 194.53 (1.48), b = 1597.76 (7.39)
```

In [201]:
```
# plot timeseries with error bars overlaid with fit
plt.figure(figsize=(12,4))
plt.plot(date, x, marker='.', ls='-', color='k', alpha=0.5)

xx = np.arange(n_use)
yy = hathaway(xx, fit_A, fit_b)
plt.plot(date, yy, 'b-')

plt.minorticks_on()
plt.ylabel("no. of sunspots")
plt.title("Sunspot Cycle 24")
```

Out[201]: Text(0.5, 1.0, 'Sunspot Cycle 24')

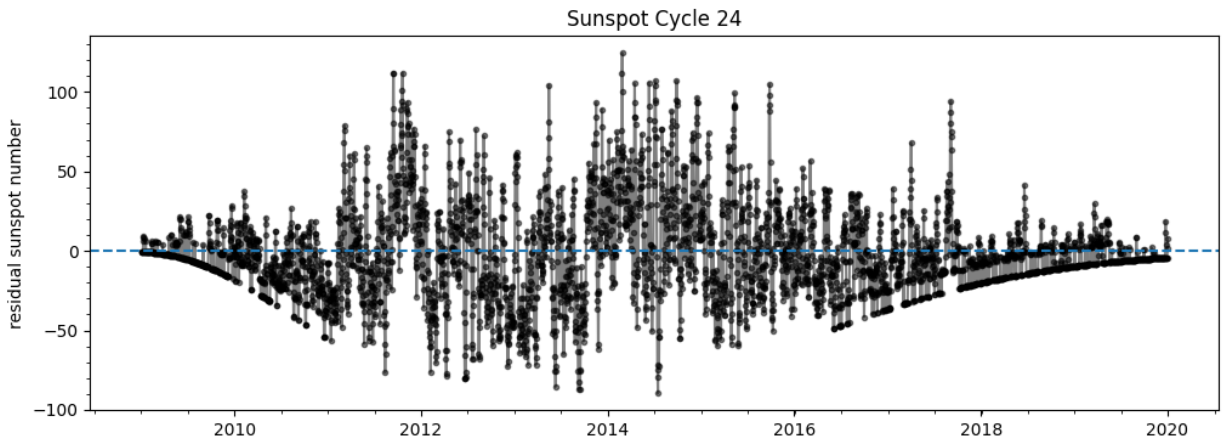


Let us now get the residuals.

In [199]:
```
xx = np.arange(n_use)
residuals = x - hathaway(xx, fit_A, fit_b)
```

In [202]:
```
# plot residuals
plt.figure(figsize=(12,4))
plt.plot(date, residuals, marker='.', ls='-', color='k', alpha=0.5)
plt.axhline(0., ls='--')
plt.minorticks_on()
plt.ylabel("residual sunspot number")
plt.title("Sunspot Cycle 24")
```

Out[202]: Text(0.5, 1.0, 'Sunspot Cycle 24')



Step 2a: Calculate empirical MSD

First, we calculate the empirical MSD values for an equally log-spaced grid in Δ values.

In [203]:
```
def get_msd(x, delta_vals):
  n = len(x)
  n_delta = len(delta_vals)
  msd = np.zeros(n_delta)*np.nan
  for i in np.arange(n_delta):
    this_delta = delta_vals[i]
    dx = get_sample_dx(x, this_delta)
    dx2_sum = np.nansum(dx**2)
    denom = n-this_delta
    msd[i] = dx2_sum/denom
  return msd
```

In [205]:
```
print("There are n = %d data points. log(n) = %.4f" % (n_use, np.log10(n_us
```

```
There are n = 4018 data points. log(n) = 3.6040
```

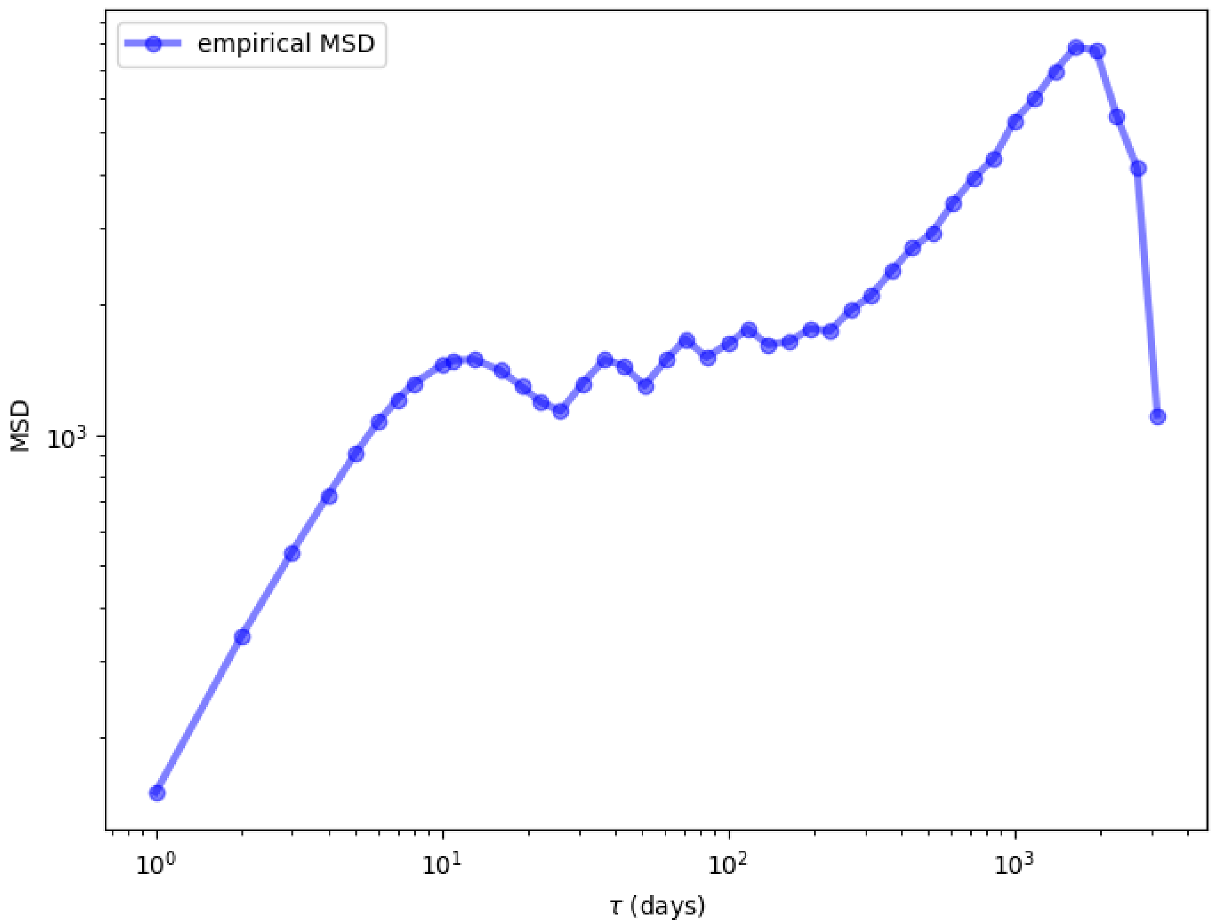
In [207]:
```
# define grid in Delta values
n_delta_init = 10 # later, can increase this
logdelta_min = 0
logdelta_max = 3.5
delta_vals = np.unique(np.floor(np.logspace(logdelta_min, logdelta_max, n_d
n_delta = len(delta_vals)
print("n_delta: %d, range in Delta: %d to %d" % (n_delta, np.min(delta_vals
print(delta_vals)
print(delta_vals/365)
```

```
n_delta: 10, range in Delta: 1 to 3162
[   1    2    5   14   35   87  215  527 1291 3162]
[2.73972603e-03 5.47945205e-03 1.36986301e-02 3.83561644e-02
 9.58904110e-02 2.38356164e-01 5.89041096e-01 1.44383562e+00
 3.53698630e+00 8.66301370e+00]
```

In [208]:
```
# get empirical msd
n_delta_emp = 50
delta_vals_emp = np.unique(np.floor(np.logspace(logdelta_min, logdelta_max,
msd_emp = get_msd(x, delta_vals_emp)
```

In [212]:
```
# plot the MSD curve
plt.figure(figsize=(8,6))
plt.plot(delta_vals_emp, msd_emp, 'bo-', alpha=0.5, lw=3, label="empirical

plt.legend(loc="best")
plt.xscale("log")
plt.yscale("log")
plt.xlabel(r"$\tau$ (days)")
plt.ylabel("MSD")
```

Out[212]: Text(0, 0.5, 'MSD')



Step 2b: Fit empirical MSD to theoretical MSD model

In [214]:
```
def msd_theo_ss(x,a,beta,mu):
  return a*math.gamma(mu)*((x)**(mu-1))*np.exp(-beta/(x))/(beta**mu)
```

In [221]:

```
# set range of fitting regime
tau_fit_min = 1
tau_fit_max = 200.

# set initial values for a, beta, mu
initial_ss = [100,1.22,0.03]
n_tau = len(delta_vals_emp)
i_fit = np.arange(n_tau)[(delta_vals_emp >= tau_fit_min) & (delta_vals_emp
x_fit = delta_vals_emp[i_fit]
y_fit = msd_emp[i_fit]
n_fit = len(i_fit)
popt, pcov = curve_fit(msd_theo_ss, x_fit, y_fit, initial_ss, maxfev=5000)

fit_a, fit_beta, fit_mu = popt[0], popt[1], popt[2]
err_a, err_beta, err_mu = pcov[0,0]**0.5, pcov[1,1]**0.5, pcov[2,2]**0.5

print("a = %.2f (%.2f), beta = %.2f (%.2f), mu = %.2f (%.2f)" % (fit_a, err
```

```
a = 3708.31 (1230.48), beta = 2.43 (0.56), mu = 1.02 (0.03)

<ipython-input-214-3532ef55ea9d>:2: RuntimeWarning: invalid value encount
ered in double_scalars
  return a*math.gamma(mu)*((x)**(mu-1))*np.exp(-beta/(x))/(beta**mu)
```

In [218]:

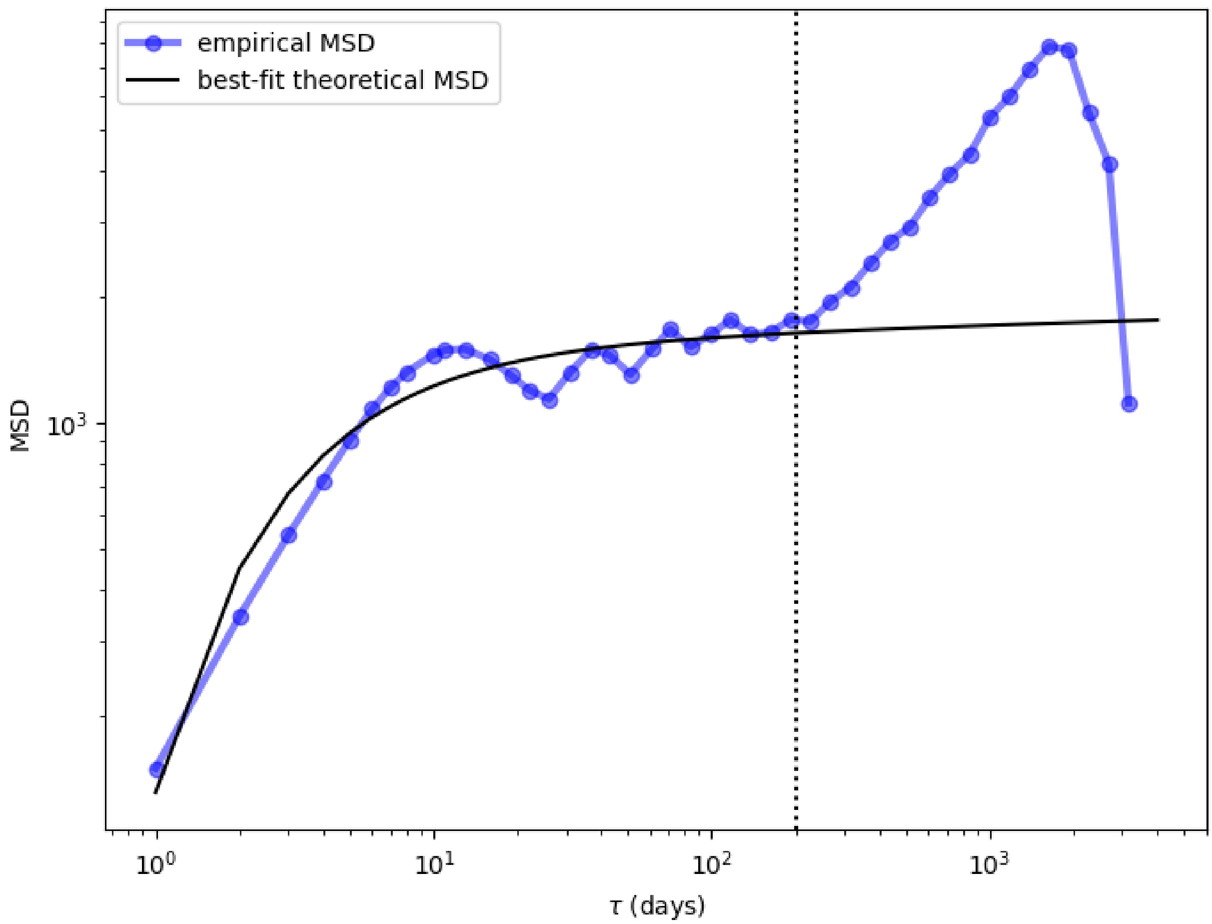
```
n_fit, np.min(x_fit), np.max(x_fit)
```

Out[218]:

```
(27, 1, 193)
```

In [225]:
```
# plot the MSD curve w/fit
plt.figure(figsize=(8,6))
plt.plot(delta_vals_emp, msd_emp, 'bo-', alpha=0.5, lw=3, label="empirical

xx = np.arange(n_use-1)+1
yy = msd_theo_ss(xx, fit_a, fit_beta, fit_mu)
plt.plot(xx, yy, 'k-', label="best-fit theoretical MSD")
plt.axvline(tau_fit_max, color='k', ls=':')

plt.legend(loc="best")
plt.xscale("log")
plt.yscale("log")
plt.xlabel(r"$\tau$ (days)")
plt.ylabel("MSD")
```

Out[225]: Text(0, 0.5, 'MSD')



Step 3: Calculate PDFs for $\Delta = 1, 3, 10, 30, 100, 200$

First, we define functions for getting the sample distribution of displacements and the PDF.

In [226]:
```
def get_sample_dx(x, delta):
  ### returns the sample of displacements dx, given lag in timesteps delta

  # 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
  return dx

def get_pdf(x, delta, bin_edges, norm=True):
  ### computes the displacement PDF given the ff:
  ### - data points x
  ### - lag in timesteps delta
  ### - bin edges
  ### can also be used to compute the raw counts (with norm=False)

  # get sample of displacements
  dx = get_sample_dx(x, delta)

  # get normalized histogram
  pdf, junk = np.histogram(dx, bins = bin_edges, density=norm)

  return pdf
```

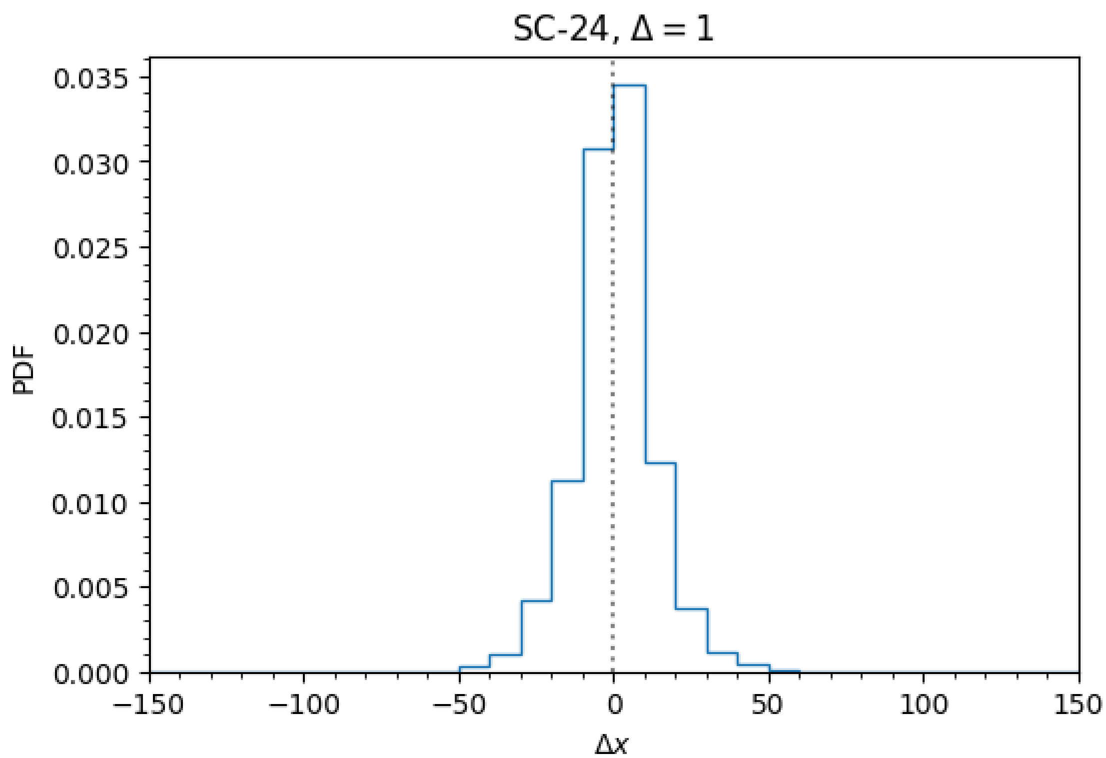
In [228]:
```
# define delta values to analyze
delta_vals = np.array([1, 3, 10, 30, 100, 200]).astype(int)
n_delta = len(delta_vals)
```

In [229]:
```
# set range of dx
xlimit = 150.
n_bins = 30
bin_edges = np.linspace(xlimit*-1., xlimit, n_bins+1)

print(bin_edges)
```

```
[-150. -140. -130. -120. -110. -100.  -90.  -80.  -70.  -60.  -50.  -40.
  -30.  -20.  -10.    0.   10.   20.   30.   40.   50.   60.   70.   80.
   90.  100.  110.  120.  130.  140.  150.]
```

In [230]:
```
# get the first pdf
delta_ref = delta_vals[0]
pdf = get_pdf(residuals, delta_ref, bin_edges, norm=True)

# plot the PDF
plt.figure(figsize=(6,4))
plt.stairs(pdf, bin_edges)
plt.axvline(0.0, color='k', alpha=0.5, ls=':')
plt.xlim((xlimit*-1., xlimit))
plt.minorticks_on()
plt.xlabel(r"$\Delta x$")
plt.ylabel("PDF")
plt.title(r"SC-%d, $\Delta=$%d" % (sc, delta_ref))
```

Out[230]:
```
Text(0.5, 1.0, 'SC-24, $\\Delta=$1')
```



In [231]:
```
# initialize array to store all pdfs
pdf_delta = np.zeros((n_delta, n_bins))

# get the pdfs
for i in np.arange(n_delta):
  this_delta = delta_vals[i]
  this_pdf = get_pdf(residuals, this_delta, bin_edges, norm=True)
  pdf_delta[i, :] = this_pdf
```

Overlay PDF plots with Gaussian model with width given by $\sqrt{\mathrm{MSD}}$.

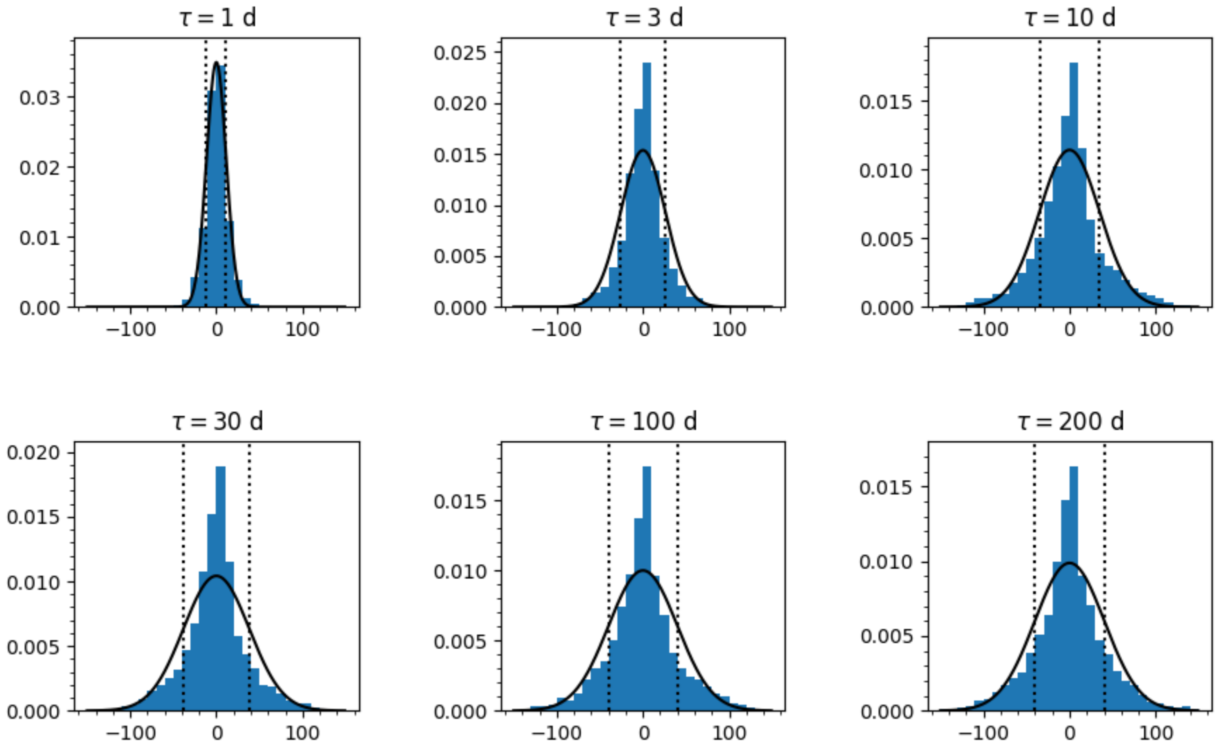
In [259]:
```
# define Gaussian function
def gaussian(x, sigma2, N):
  fac = N/(2.*np.pi*sigma2)**0.5
  return fac*np.exp(-1.*x**2/2./sigma2)
```

In [266]:
```
# calculate RMSE
rmse_delta = np.zeros(n_delta)
for i in np.arange(n_delta):
  this_delta = delta_vals[i]
  this_pdf = pdf_delta[i, :]
  this_msd = msd_theo_ss(this_delta, fit_a, fit_beta, fit_mu)
  this_yval = gaussian(bin_centers, this_msd, 1.0)
  this_rmse = np.sum((this_pdf-this_yval)**2)**0.5
  rmse_delta[i] = this_rmse
rmse_delta
```

Out[266]:
```
array([0.0099176 , 0.01370812, 0.01068445, 0.01320873, 0.01134187,
       0.01057397])
```

In [272]:
```
# plot the PDFs
nx = 2
ny = 3

print("delta msd_theo msd_theo**0.5 RMSE RMSE/mean")

plt.figure(figsize=(12,6))
for i in np.arange(n_delta):
  plt.subplot(nx,ny,i+1)
  this_delta = delta_vals[i]
  this_pdf = pdf_delta[i, :]
  plt.stairs(this_pdf, bin_edges, fill=True)

  # plot Gaussian with sigma = empirical MSD & N = 1
  xx = np.arange(-1.*xlimit, xlimit)
  this_msd = msd_theo_ss(this_delta, fit_a, fit_beta, fit_mu)
  this_N = 1.
  yy = gaussian(xx, this_msd, this_N)
  this_rmse_mean = rmse_delta[i]/np.mean(yy)
  print(this_delta, this_msd, this_msd**0.5, rmse_delta[i], this_rmse_mean)

  plt.plot(xx, yy, 'k-') #, label="RMSE/mean = %.4f" % this_rmse_mean) #, l
  plt.axvline(-1.*this_msd**0.5, color='k', ls=":")
  plt.axvline(this_msd**0.5, color='k', ls=":")

  plt.minorticks_on()
  #plt.xlim((-120., 120.))
  plt.ylim(0., np.max(np.concatenate((yy, this_pdf)))*1.1)
  plt.title(r"$\tau=$%d d" % this_delta)
  #plt.legend(loc="upper right")

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

```
delta msd_theo msd_theo**0.5 RMSE RMSE/mean
1 130.87491381212737 11.440057421714604 0.009917604522093287 2.9752813566
27986
3 675.198275685093 25.984577650696828 0.013708122823287743 4.112436879212
182
10 1219.0768356496653 34.91528083303449 0.01068445367744311 3.20539189273
2686
30 1465.4609667873492 38.28133966813791 0.013208727889421526 3.9629720087
727787
100 1589.1061228029075 39.863593952413616 0.011341873610300418 3.40313426
1473476
200 1631.2246824298431 40.38842262864252 0.010573967306850618 3.172838129
426031
```



We find that the empirical PDF for $\tau = 1$ day match best with the Gaussian model, but for the rest (longer lag times), the empirical PDFs are more densely concentrated in the middle (small displacements) compared to the Gaussian model.

In [291]:
```
# get the percentage of the sample that is within +/- 1-sigma (MSD**0.5)
sigma_vals = msd_theo_ss(delta_vals, fit_a, fit_beta, fit_mu)**0.5

print("delta sum(pdf_inc) sum(pdf_all) pct_inc")
for i in np.arange(n_delta):
  this_delta = delta_vals[i]
  this_sigma = sigma_vals[i]
  this_pdf = pdf_delta[i, :]
  i_inc = np.arange(n_bins+1)[(bin_edges > -1.*this_sigma) & (bin_edges < t
  #bin_edges_inc = bin_edges[i_inc]
  # bin_centers_inc = bin_centers[i_inc[:-1]]
  this_pdf_inc = this_pdf[i_inc[:-1]]
  sum_pdf_inc = np.sum(this_pdf_inc)
  sum_pdf_all = np.sum(this_pdf)
  pct_inc = sum_pdf_inc/sum_pdf_all
  print(this_delta, sum_pdf_inc, sum_pdf_all, pct_inc)
```

```
delta sum(pdf_inc) sum(pdf_all) pct_inc
1 0.06529748568583521 0.1 0.6529748568583521
3 0.06988792029887919 0.09999999999999998 0.698879202988792
10 0.06749937546839871 0.09999999999999999 0.6749937546839871
30 0.06889111891620672 0.09999999999999999 0.6889111891620673
100 0.0648710748021445 0.1 0.648710748021445
200 0.07273203985317253 0.09999999999999998 0.7273203985317255
```

We find that around 70% of the displacements are within $\pm 1\sigma$ of 0, where $\sigma = MSD_{\text{theo}}^{1/2}$. This tells us that the broadening of the empirical PDF with higher lag times is consistent with the trend seen in the empirical MSD curve.