0.1 Importing Libraries

In [216]:
```
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

from scipy.stats import norm, describe
from scipy.optimize import curve_fit
```

executed in 5ms, finished 14:25:30 2023-10-15

1 Simulating fractional Brownian motion

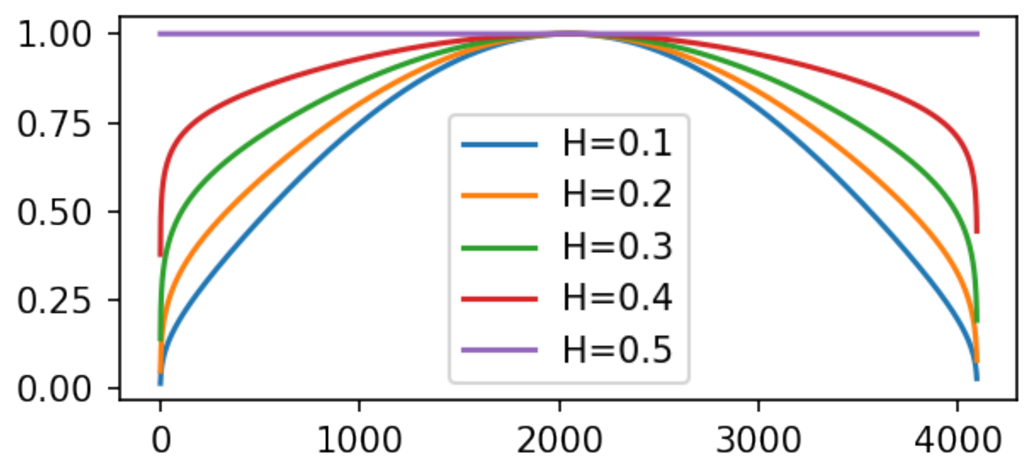
1.1 $(n+1)^G + \frac{(n-1)^G - 2nG}{2}$

In [9]:
```
# define functions
def fbc(n,G):
    return ((n+1)**G + np.abs(n-1)**G - 2*n**G)/2.

def lambda_func(H,N):
    M = 2*N - 2
    C = np.zeros(M)
    G = 2*H
    for i in np.arange(N):
        C[i] = fbc(i,G) # fill in first N out of M values (
    C[N:] = C[1:N-1][::-1]
    return np.real(np.fft.fft(C))**0.5
```
executed in 10ms, finished 12:17:57 2023-10-15

In [52]:
```
noise = np.random.normal(size=(M))
```

In [55]:
```
plt.figure(dpi = 150, figsize = (10,2))

for i in range(5):
    plt.subplot(121)
    H = i*0.1 + 0.1
    lambda_function = lambda_func(H, 2049)
    plt.plot(lambda_function/lambda_function.max(), ls = '-
    plt.legend()

    plt.subplot(122)
    H = i*0.1 + 0.6
    lambda_function = lambda_func(H, 2049)
    plt.plot(lambda_function/lambda_function.max(), ls = '-
    plt.legend()

plt.show()
```

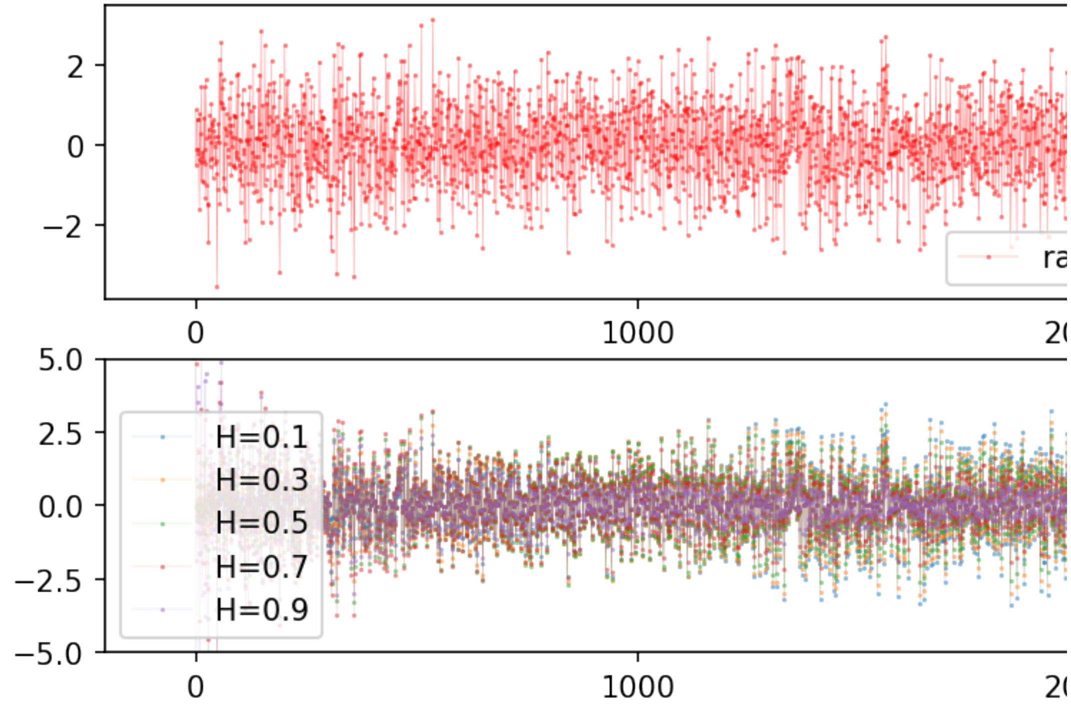
executed in 343ms, finished 12:44:22 2023-10-15



In [87]:

```
plt.figure(dpi = 150, figsize = (12,4))
plt.subplot(211)
plt.plot(noise, 'r.-', label = "random noise", lw = 0.2, a
plt.legend()

for i in range(5):
    plt.subplot(212)
    H = i*0.2 + 0.1
    lambda_function = lambda_func(H, 2049)*noise
    plt.plot(lambda_function, '.-', label = "H=%.1f" % H, 
    plt.ylim(-5,5)
    plt.legend()

plt.show()
```

executed in 443ms, finished 12:50:55 2023-10-15



In [111]:

```
# set process parameters
q = 11
N = 2**q + 1 #number of datapoints
M = 2*N - 2
delta = 0.002
print("q: %d, N: %d, M: %d" % (q, N, M))
```

executed in 7ms, finished 13:07:16 2023-10-15

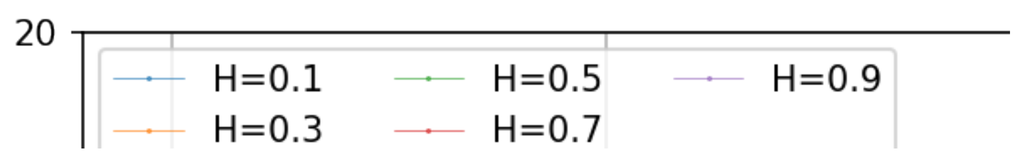
```
q: 11, N: 2049, M: 4096
```

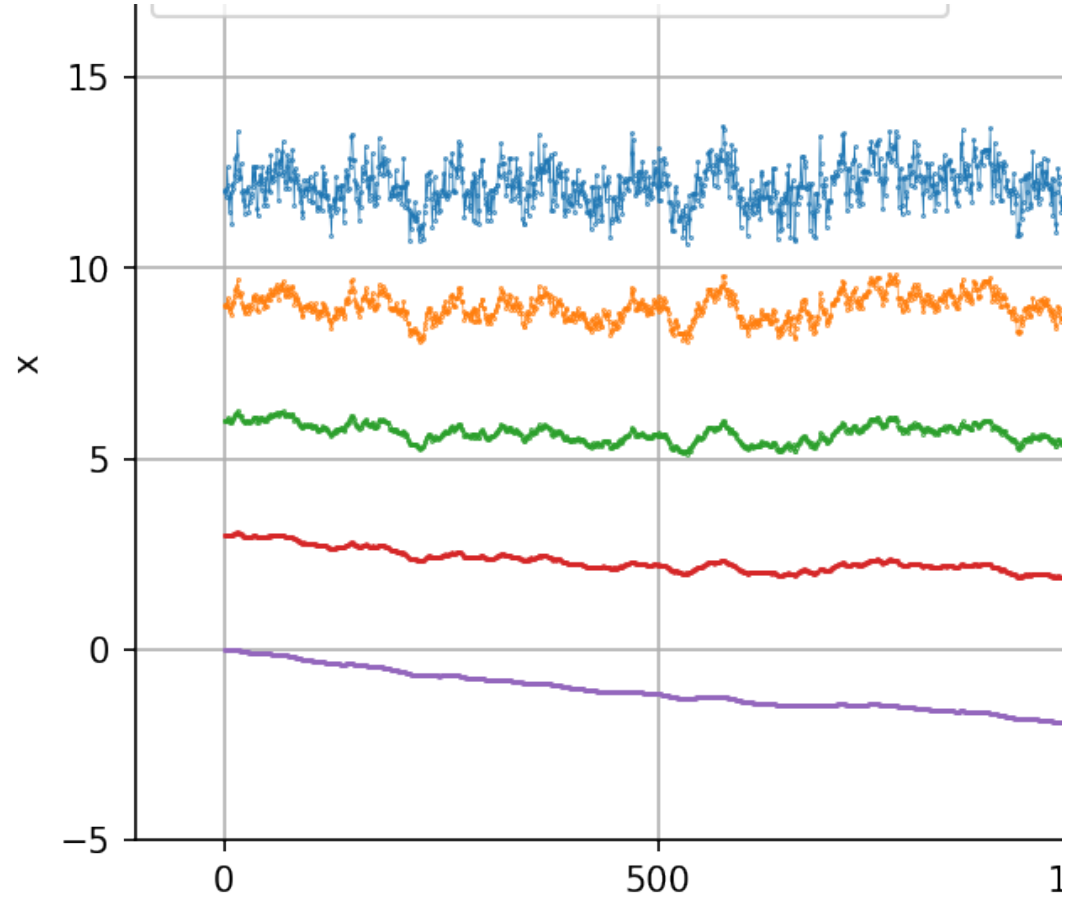
In [114]:

```
# initialize
fGnsamples = np.zeros((5,N))

# generate fractional Gaussian noise samples
for i in np.arange(5):
    H = 0.2*(i+1) - 0.1
    lambda_res = lambda_func(H,N)
    a = np.fft.ifft(noise)*lambda_res
    b = np.real(np.fft.fft(a))
    fGnsamples[i,0] = 15-3*(i+1)           # apply offset
    fGnsamples[i,1:N] = delta**H*b[0:N-1]

# take cumulative sums to get the fractional Brownian moti
fBmsamples = np.transpose(np.cumsum(fGnsamples,axis=1)) # 

# get values of H
Hval = (np.arange(5) + 1)*0.2 - 0.1
print(Hval)

# plot fractional Brownian motion samples
plt.figure(dpi = 150
           , figsize = (10,5))
plt.grid()
for i in np.arange(len(Hval)):
    plt.plot(fBmsamples[:,i], label="H=%.1f" % Hval[i], ma
    plt.ylim(-5, 20)
    plt.legend(loc="upper left", ncol=3)
    plt.xlabel("t")
    plt.ylabel("x")
```

executed in 193ms, finished 13:07:35 2023-10-15

```
[0.1 0.3 0.5 0.7 0.9]
```





1.2 Generating fBm samples

In [143]:
```
def fBm_generator(noise):
    # initialize
    fGnsamples = np.zeros((5,N))

    # generate fractional Gaussian noise samples
    for i in np.arange(5):
        H = 0.2*(i+1) - 0.1
        lambda_res = lambda_func(H,N)
        a = np.fft.ifft(noise)*lambda_res
        b = np.real(np.fft.fft(a))
        fGnsamples[i,0] = 15-3*(i+1)           # apply of
        fGnsamples[i,1:N] = delta**H*b[0:N-1]

    # take cumulative sums to get the fractional Brownian 
    fBmsamples = np.transpose(np.cumsum(fGnsamples,axis=1)
    return fBmsamples
```

executed in 15ms, finished 13:45:01 2023-10-15

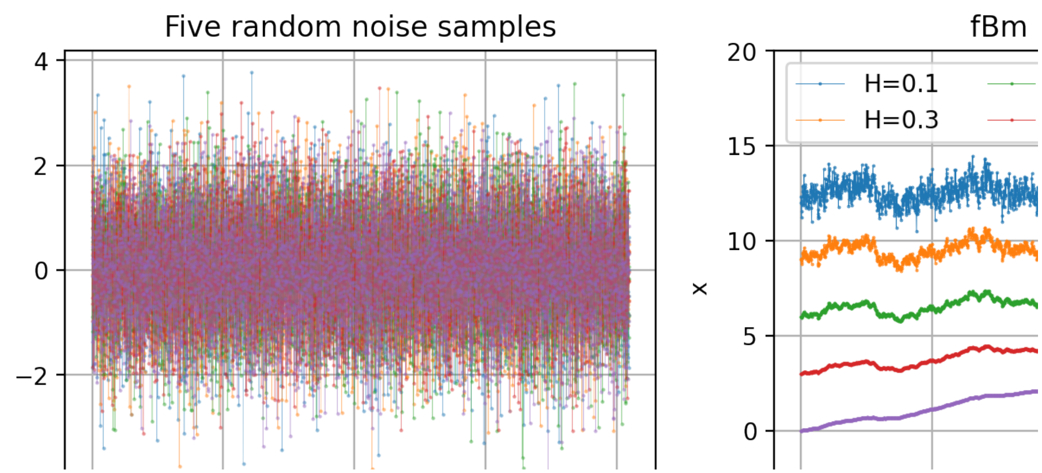
In [144]:
```
np.random.seed(seed=17)
n_samp = 5
# Initialize array to hold the noise samples
noise_samp = np.zeros((M, n_samp))
for i in range(n_samp):
    noise = np.random.normal(size=(M))
    noise_samp[:,i] = noise
```

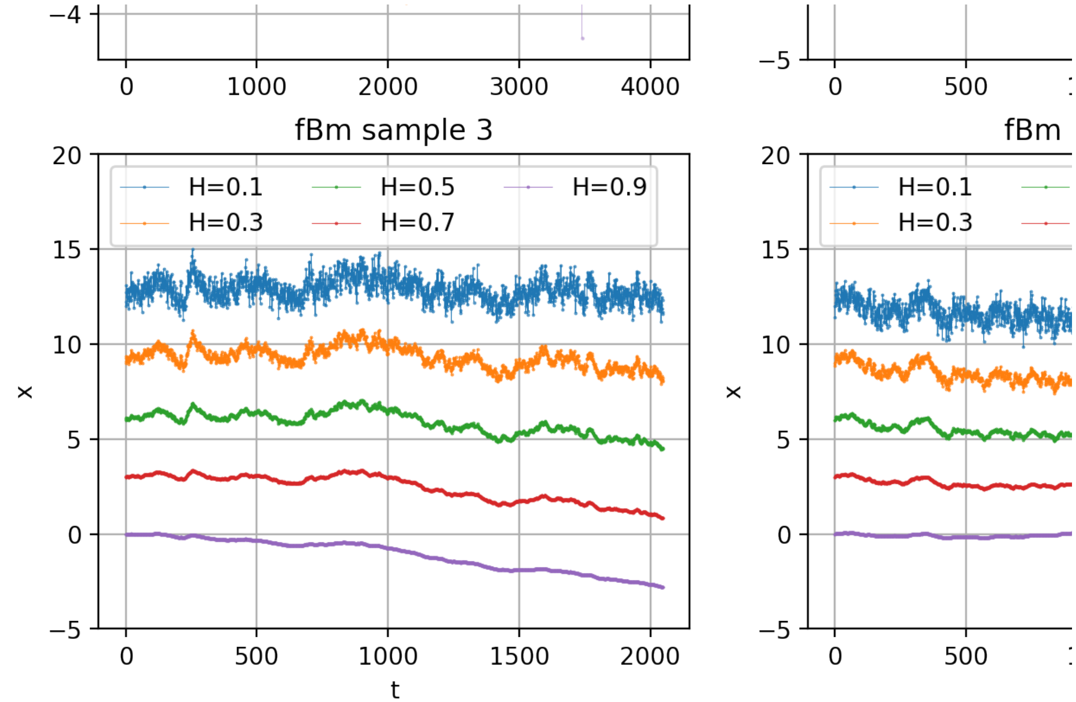
executed in 19ms, finished 13:45:01 2023-10-15

In [146]:

```python
plt.figure(dpi = 200, figsize = (15,8))

plt.subplot(2,3,1)
plt.title("Five random noise samples")
plt.plot(noise_samp, '.-', lw = 0.2, alpha = 0.5, ms = 1)

# Initialize list to hold the fBm samples
fBm_simulations = []

for i, noise in enumerate(noise_samp.T):
    fBmsamples = fBm_generator(noise)
    fBm_simulations.append(fBmsamples)
    plt.grid()
    plt.subplot(2,3,i+2)
    for j in np.arange(len(Hval)):
        plt.title("fBm sample " + str(i+1))
        plt.plot(fBmsamples[:,j], label="H=%.1f" % Hval[j]
        plt.ylim(-5, 20)
        plt.xlabel("t")
        plt.ylabel("x")
        plt.legend(loc="upper left", ncol=3)
plt.show()
```

executed in 919ms, finished 13:45:08 2023-10-15





1.3 Calculating PDFs

1.3.1 Pre-defined functions

In [285]:

```
def get_sample_dx(x, delta):
    x_trunc = x[:-1*delta]
    x_shift = x[delta:]
    dx = x_shift - x_trunc
    return dx

def get_pdf(x, delta, bin_edges, norm=True):
    dx = get_sample_dx(x, delta)
    pdf, junk = np.histogram(dx, bins = bin_edges, density=
    return pdf

def gaussian(x, sigma2, N):
    fac = N/(2.*np.pi*sigma2)**0.5
    return fac*np.exp(-1.*(x)**2/2./sigma2)

# define function that performs fit
def fit_pdf(bin_centers, y, yerr=np.array([]), initial=[1.
    if(len(yerr)==0): # no uncertainties given
        popt, pcov = curve_fit(gaussian, bin_centers, y, i
    else:
        popt, pcov = curve_fit(gaussian, bin_centers, y, i
    sigma2, N = popt[0], popt[1]
    err_sigma2, err_N = pcov[0,0]**0.5, pcov[1,1]**0.5
    return sigma2, err_sigma2, N, err_N
```

executed in 14ms, finished 14:48:22 2023-10-15

1.3.2 Sample 1, H = 0.1, τ = 30

In [286]:
```
sample = 0
H_n = 0
H_test = Hval[H_n]
test_data = fBm_simulations[H_n][:, sample]
tau_test = 30
```
executed in 5ms, finished 14:48:22 2023-10-15

1.3.3 Obtaining displacements

In [287]:
```
test_dx = get_sample_dx(test_data, tau_test)
```
executed in 5ms, finished 14:48:23 2023-10-15

1.3.4 Probability Distribution Functions

In [293]:
```
xlimit = 5.
n_bins = 30
bin_edges = np.linspace(xlimit*-1., xlimit, n_bins+1)
pdf = get_pdf(test_dx, tau_test, bin_edges, norm=False)
```
executed in 9ms, finished 14:48:48 2023-10-15

1.3.5 Gaussian Fit

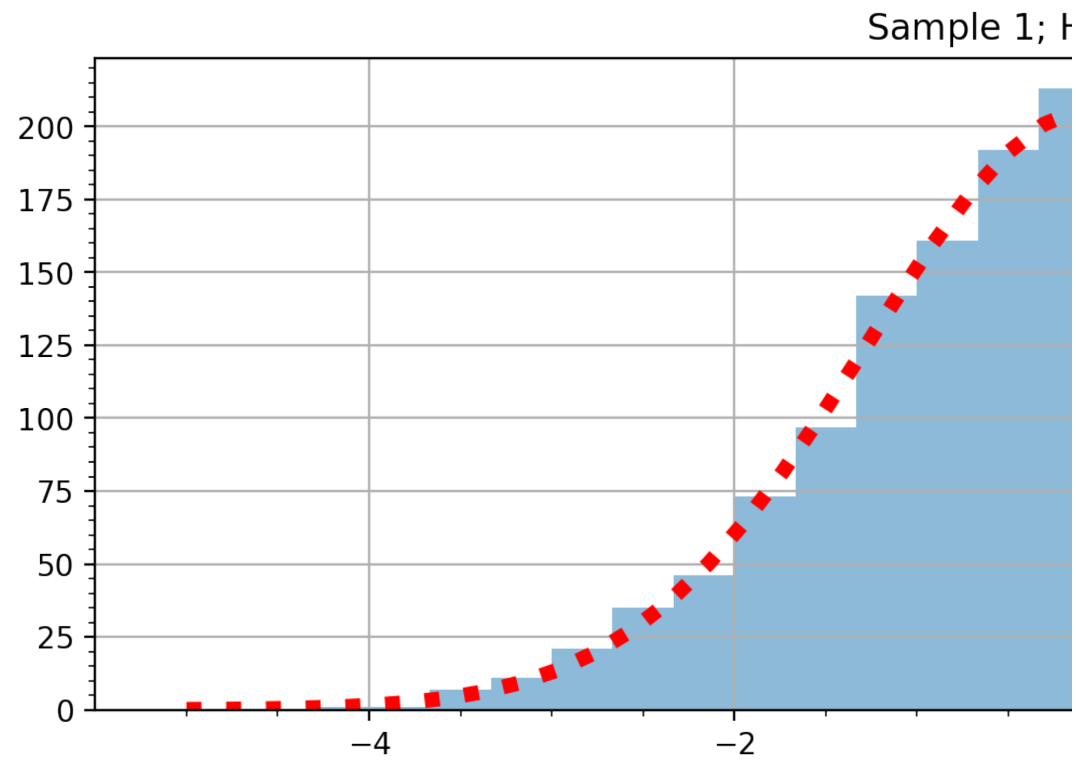
In [294]:
```
bin_centers = 0.5*(bin_edges[:-1] + bin_edges[1:])
sigma2, err_sigma2, N, err_N = fit_pdf(bin_centers, pdf)
xx = np.linspace(xlimit*-1., xlimit)
yy = gaussian(xx, sigma2, N)
```
executed in 10ms, finished 14:48:49 2023-10-15

1.3.6 Plotting the fBm PDF and Gaussian fit

In [296]:
```
plt.figure(dpi = 200, figsize = (12,4))
plt.grid()
plt.stairs(pdf, bin_edges, fill=True, alpha = 0.5)
plt.plot(xx,yy, 'r:', lw = 5, label = "Gaussian fit: \n" \
        + r"$\sigma^2$ = %.2f (%.2f)" % (sigma2, err_sigma2
         + "\n N = %.2f (%.2f)" % (N, err_N))
plt.minorticks_on()
plt.title(r"Sample %d; H=%.2f; $\Delta=%d$" % (sample+1, H_
plt.legend()
```

executed in 204ms, finished 14:49:05 2023-10-15

Out[296]: <matplotlib.legend.Legend at 0x7f9dc20df2e0>



1.3.7 Automate Plotting

In [322]:
```
def plot_PDF_fBm(sample, H_n, tau_test, xlimit, n_bins):
    H_test = Hval[H_n]
    test_data = fBm_simulations[H_n][:, sample]

    test_dx = get_sample_dx(test_data, tau_test)

    bin_edges = np.linspace(xlimit*-1., xlimit, n_bins+1)
    pdf = get_pdf(test_dx, tau_test, bin_edges, norm=True)

    bin_centers = 0.5*(bin_edges[:-1] + bin_edges[1:])
    sigma2, err_sigma2, N, err_N = fit_pdf(bin_centers, pd
    xx = np.linspace(xlimit*-1., xlimit)
    yy = gaussian(xx, sigma2, N)

    plt.grid()
    plt.stairs(pdf, bin_edges, fill=True, alpha = 0.5)
    plt.plot(xx,yy, 'r-', lw = 3, label = "Gaussian fit: \
            + r"$\sigma^2$ = %.2f (%.2f)" % (sigma2, err_s
             + "\n N = %.2f (%.2f)" % (N, err_N))
    plt.minorticks_on()
    plt.title(r"Sample %d; H=%.2f; $\Delta=%d$" % (sample+
    plt.legend()
```

executed in 16ms, finished 15:03:36 2023-10-15

In [*]:

```
# generate plot for 1 sample
i_samp = 0 # select sample to plot (by index)
plt.figure(dpi = 200, figsize = (15,6))

plt.subplot(2,3,1)
plt.plot(fBm_simulations[i_samp], label="H=%.1f" % Hval[H_
plt.xlabel("t")
plt.ylabel("x")
plt.title("fBm simulation sample " + str(i_samp+1))
plt.legend(loc="upper left", ncol=3)

for i in np.arange(5):
    plt.subplot(2,3,i+2)
    plot_PDF_fBm(sample = i_samp, H_n = i, tau_test = 10, 

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

execution queued 15:05:42 2023-10-15

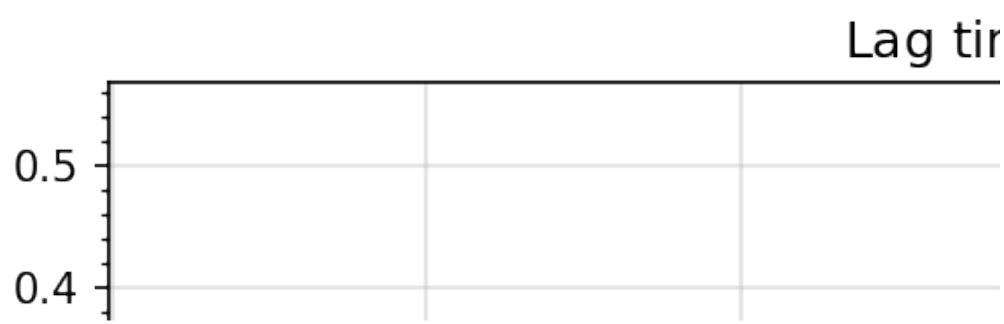
In [ ]:

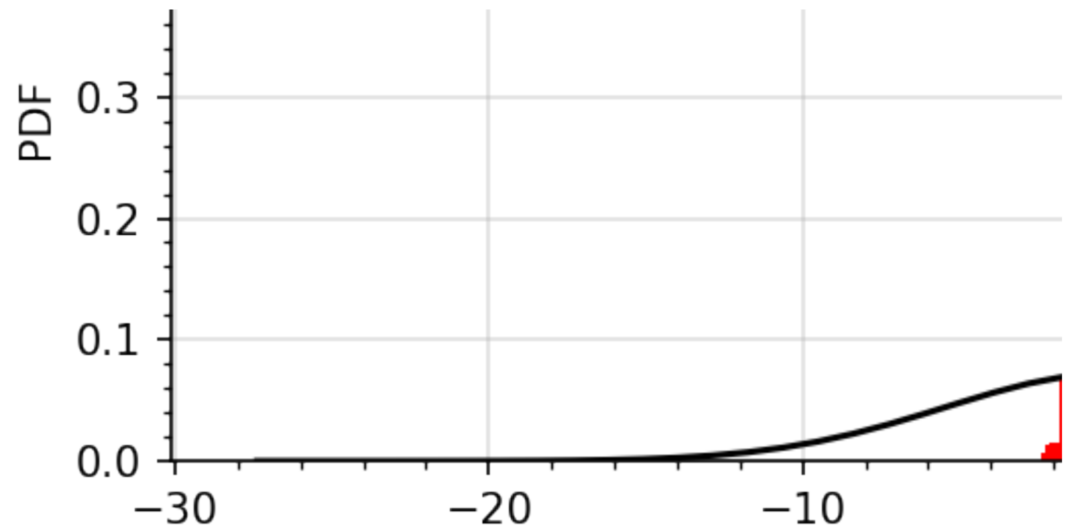
In [291]:

```
# Generate normal distribution
xx_sd = np.sqrt(tau)
xx = np.linspace(-5, 5)*xx_sd # gridded points from -5 to :
xx_mean = 0.
yy = norm.pdf(xx, xx_mean, xx_sd)

plt.figure(figsize=(8,3), dpi = 150)
plt.grid(alpha = 0.4)
# Plot PDF of displacements
plt.hist(test_dx, density=True, bins="auto", label="dx his
plt.xlabel("Delta x")
plt.ylabel("PDF")

# # Overlay normal distribution
plt.plot(xx, yy, 'k-', label="N(%.1f,%.2f)" % (xx_mean, xx_
plt.ylabel("PDF")
plt.xlabel(r"$\Delta x$")
plt.legend(loc="best")

plt.minorticks_on()
plt.xlabel(r"$\Delta x$")
plt.ylabel("PDF")
plt.title(r"Lag time: $\Delta=$%d " % (tau))
# plt.yscale('log')
```

executed in 172ms, finished 14:48:25 2023-10-15

Out[291]: Text(0.5, 1.0, 'Lag time: $\\Delta=$30 ')






```
In [ ]:
```