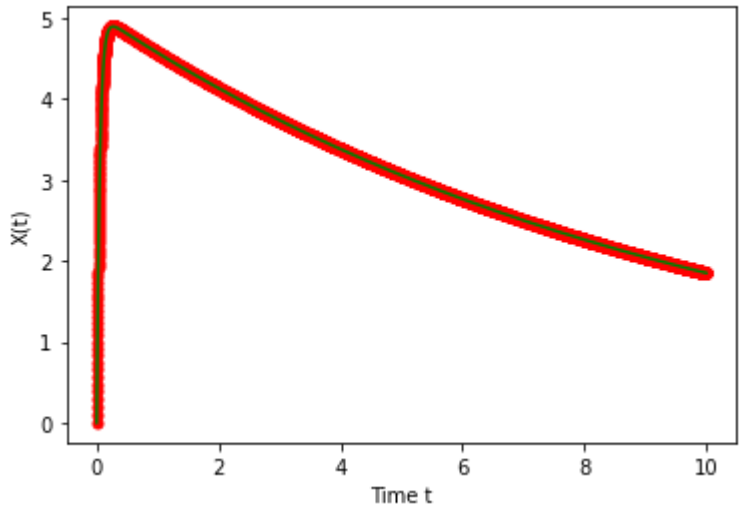
In [1]:

```
# Reference:
# https://blog.csdn.net/weixin_42376039/article/details/86485817
# https://www.epythonguru.com/2020/07/second-order-differential-equation.html
# https://apmonitor.com/pdc/index.php/Main/SolveDifferentialEquations

import matplotlib.pyplot as plt
from scipy.integrate import odeint
import numpy as np

# Initialized the parameters
h = 0.001 # time step
interval_len = 10 # the length of the interval
# total num of observed time stamps N = interval_len / h
# And I will chop off the decimal places after N in cases N is a non-integer

m = 1 # mass
K = 2 # spring constant
gamma = 20 # frictional coefficient

# x_n := x(t_n)
# dx_n := the first derivative of x(t_(n-1))
x_0 = 0 # initial position of the particle
p_0 = 100 # initial velocity of the particle

# initialize the values for iterations
x_n = x_0
p_n = p_0

N_ls = [0] # The i list, (which are the index of ti, the observed timestamp)
x_n_ls = [x_n] # contains the trajectory of X at different time ti, i.e. X(t)

# N = truncate_to_int(interval_len/h) + 1
for N in range(1, int(interval_len/h)+1):
    x_n = x_n + p_n*h
    p_n = p_n + h*(-gamma*p_n - K*x_n)/m
    N_ls.append(N)
    x_n_ls.append(x_n)

# change the plot's labels in i (the ith observation) to ti (observed timestamps)
time_axis = [i * h for i in N_ls]
plt.plot(time_axis, x_n_ls, color='red', marker = ".", linestyle='None', markersize = 10)

### We will plot the analytical ODE sol in a continous line to represent the ground truth #
# odeint() can define more than one 1st order differential equations
# And we can solve even higher order ODEs by using multiple 1st order ODEs.
def derivatives(initial_values, time_interval):
    x_0 = initial_values[0]
    dxdt_0 = initial_values[1]
    sec_dxdt_0 = -gamma/m*dxdt_0-K/m*x_0
    return(dxdt_0, sec_dxdt_0)
initial_values = [x_0, p_0]
N_boosted = N * 1000 # Boost up the total num of observed time stamps N to make a "continuo
time_interval = np.linspace(0, interval_len, N_boosted)
ODE_sol = odeint(derivatives, initial_values, time_interval)
ODE_sol = ODE_sol[:,0]
plt.plot(time_interval,ODE_sol, linestyle ='-', color='green' )

plt.xlabel("Time t")
plt.ylabel("X(t)")
```

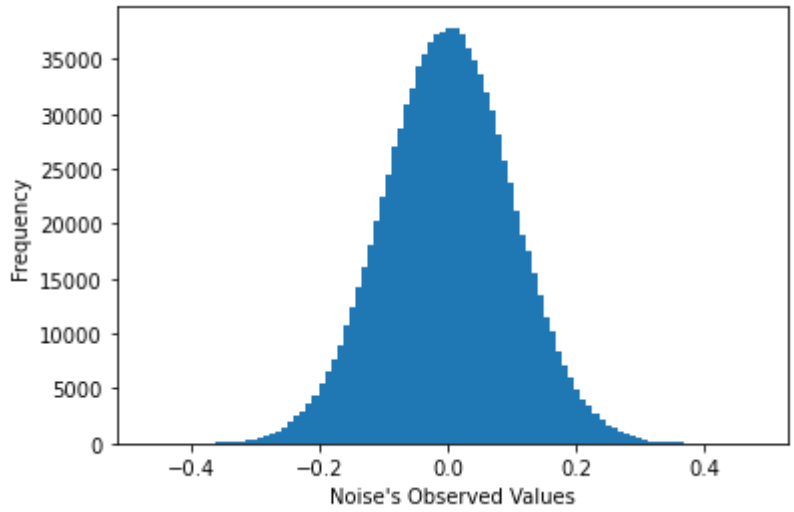
Out[1]:

```
Text(0, 0.5, 'X(t)')
```



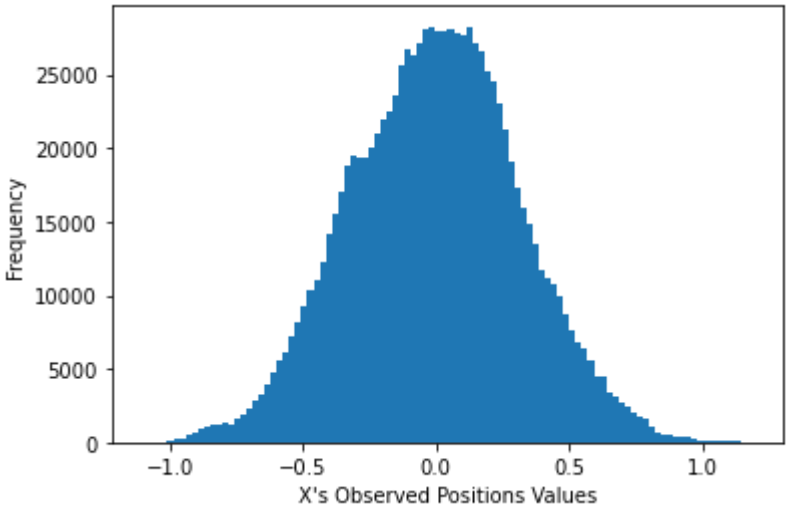
In [2]:

```
# Initialized the parameters
h = 0.001 # time step
interval_len = 1000 # the length of the interval

# mass = 0 and can be obmitted
K = 5 # spring constant
gamma = 10 # frictional coefficient

# N = truncate_to_int(interval_len/h) + 1
for N in range(1, int(interval_len/h)+1):
    x_n = x_n + p_n*h
    p_n = p_n + h*(-gamma*p_n - K*x_n)/m
    N_ls.append(N)
    x_n_ls.append(x_n)

mu, sigma = 0, 0.1 # mean and standard deviation
noise_sample = np.random.normal(mu, sigma, N)
plt.hist(noise_sample, bins = 100)
plt.xlabel("Noise's Observed Values")
plt.ylabel("Frequency")
plt.show()
```



In [3]:

```
x_noise_i = x_0
x_noise_ls = [x_noise_i]
for i in range(1, int(interval_len/h)+1):
    x_noise_i = x_noise_i - K/gamma*x_noise_i*h + noise_sample[i-1]/gamma
    x_noise_ls.append(x_noise_i)
plt.hist(x_noise_ls, bins = 100)
plt.xlabel("X's Observed Positions Values")
plt.ylabel("Frequency")
plt.show()
```



In [4]:

```
# Reference:
# https://stackoverflow.com/questions/20011122/fitting-a-normal-distribution-to-1d-data
import numpy as np
from scipy.stats import norm

## Find the best fitted normal distribution to X(t) ##
mu, std = norm.fit(x_noise_ls) # x_noise_ls: contains the trajectory of X(t)

## Plot the histogram of X(t) ##
# density: transfer the frequence on y-axis into probability density
# alpha: constrols the histogram's transparency
plt.hist(x_noise_ls, bins = 100, density=True, alpha=0.3, color='green')
xmin, xmax = plt.xlim() # set the boundary for x_axis
# larger N is, the higher the resolutions of (smoother) the fitted line of the PDF values c
selected_pts_on_x_axis = np.linspace(xmin, xmax, 100) # 100 is the number of equally spaced
corresponding_pdf_values = norm.pdf(selected_pts_on_x_axis , mu, std)
plt.plot(selected_pts_on_x_axis, corresponding_pdf_values, 'k', linewidth=2) # k:= (col = b
title = "Estimated Paramters: $\hat{\mu}$ = %.2f, $\hat{\sigma}$ = %.2f" % (mu, std)  # %.2
plt.title(title)
plt.xlabel("X's Observed Positions Values")
plt.ylabel("Probability Density")
plt.show()
```