In [ ]:
```
!pip install -q numpy matplotlib
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams["figure.dpi"] = 100
```

0. Provided Functions

In [ ]:
```
def plot_signal_and_subsampled_signal(x, x_delta, t):
    """
    Plot the original signal and the subsampled signal

    Args:
        x: Original (time) signal of length N
        x_delta: Subsampled (time) signal of length N (zero padded)
        t: time instants [0, T_s, ... , NT_s=T]
    """

    fig, axs = plt.subplots(2, 1)
    fig.suptitle('Original signal and subsampled signal' )
    fig.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=Non
    axs[0].plot(t, x)
    axs[0].set_xlabel('Time (s)')
    axs[0].set_ylabel('Signal')
    axs[1].plot(t, x_delta)
    axs[1].set_xlabel('Time (s)')
    axs[1].set_ylabel('Subsampled signal')
    axs[0].grid()
    axs[1].grid()


def plot_dft_of_signal_and_subsampled_signal(X, X_delta, f):
    """
    Plot the original signal and the subsampled signal

    Args:
        X: DFT of the original signal
        x_delta: DFT of the subsampled (zero-padded) signal
        f: DFT frequencies [-fs/2, fs/2]
    """

    fig, axs = plt.subplots(2, 1)
    fig.suptitle('DFT of the original signal and subsampled signal')
    fig.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=Non
    axs[0].plot(f, np.abs(X))
    axs[0].set_xlabel('Frequency (Hz)')
    axs[0].set_ylabel('Signal')
    axs[1].plot(f, np.abs(X_delta))
    axs[1].set_xlabel('Frequency (Hz)')
    axs[1].set_ylabel('Subsampled signal')
    axs[0].grid()
    axs[1].grid()

    return fig, axs
```

```python
def compute_fft(x, fs: int, center_frequencies=True):
    """
    Compute the DFT of x.

    Args:
        x: Signal of length N.
        fs: Sampling frequency.
        center_frequencies: If true then returns frequencies on [-f/2,f/2]. If f
    Returns:
        X: DFT of x.
        f: Frequencies
    """
    N = len(x)
    X = np.fft.fft(x, norm="ortho")
    f = np.fft.fftfreq(N, 1/fs)
    if center_frequencies:
        X = np.fft.fftshift(X)
        f = np.fft.fftshift(f)
    return X, f


def compute_ifft(X, fs: int, center_frequencies=True):
    """
    Compute the iDFT of X.

    Args:
        X: Spectrum  of length N.
        fs: Sampling frequency.
        center_frequencies: If true then X is defined over the frequency range [
    Returns:
        x: iDFT of X and it should be real.
        t: real time instants in the range [0, T]
    """
    N = len(X)
    if center_frequencies:
      X = np.fft.ifftshift(X)
    x = np.fft.ifft(X, norm="ortho")
    t = np.arange(N)/fs
    return x.real, t
```

1.1 Sampling of bandlimited signals

For bandlimited signals, there will be no loss of information with we sample with a frequency fs greater than or equal to the bandwidth, so not only are there no frequencies outside [−W/2,W/2], there will also be none outside of [kfs-fs/2, kfs+fs/2]. So the copies of the spectrum will be spaced far enough apart in the frequency domain to prevent aliasing. This allows us to reconstruct the signal by using a low pass filter on the dirac train to get xδ(t)*[fs·sinc(pifst)]that ignores frequences outside the -fs/2 to +fs/2 range

1.2 Avoiding aliasing

We can avoid aliasing using a low pass filter by processing the signal with a convolution in the time domain x * sinc(pifst) so that the signal turns into one with spectrum with a bandwidth of fs which can be sampled without aliasing.

1.3 Reconstruction with arbitrary pulse trains

$$X_\delta(f) = \sum_{k=-\infty}^{\infty} X(f - kf_s)$$

$$\tilde{X}_\delta(f) = \Pi_{f_s}(f) \sum_{k=-\infty}^{\infty} X(f - kf_s) = \Pi_{f_s}(f) X(f)$$

$$X_p(f) = P(f) X_\delta(f) = P(f) \sum_{k=-\infty}^{\infty} X(f - kf_s)$$

$$\tilde{X}_p(f) = \Pi_{f_s}(f) P(f) \sum_{k=-\infty}^{\infty} X(f - kf_s) = P(f) \Pi_{f_s}(f) X(f)$$

$\Rightarrow$ condition for no distortion if its spectrum is always $= 1$

$p(t) = f_s \,\mathrm{sinc}(\pi f_s t)$ is a pulse that satisfies the condition

Insert your proof here

2. Subsampling

2.1 Subsampling theorem

2.1 we start with $x_\delta(n) = \sum_{m=-\infty}^{\infty} x_d\left(m\frac{\tau}{T_s}\right)\delta\left(n-m\frac{\tau}{T_s}\right)$

this equivalent to $x_\delta(n) = x_d(n)\sum_{m}\delta\left(n-m\frac{\tau}{T_s}\right)$

DTFT $x_d(n) = x(t)T_s\sum_{n}\delta(t-nT_s) \overset{F}{\Rightarrow} X_d(F) = X(F-KF_s)$

$x_\delta(n) = X_d(F) * F\left(\sum_{m}\delta\left(n-m\frac{\tau}{T_s}\right)\right)$

$= \frac{T_s}{\tau}\sum_{n}\sum_{m}\delta\left(n-m\frac{\tau}{T_s}\right)e^{-j2\pi Fn} = \sum_{m}\exp\left(-j2\pi Fm\frac{\tau}{T_s}\right)$

$= \frac{T_s}{\tau}\sum_{m}\delta\left(F-K\frac{1}{\tau}\right) = \frac{T_s}{\tau}\sum_{m=-\infty}^{\infty}\delta\left(F-K\frac{\tau}{T_s}F_s\right)$

$x_\delta(n) = X(F) * \sum_{m=-\infty}^{\infty}\delta\left(F-K\frac{\tau}{T_s}F_s\right) = X\left(F-K\frac{1}{\tau}\right)$

Q2.2 Subsampling function

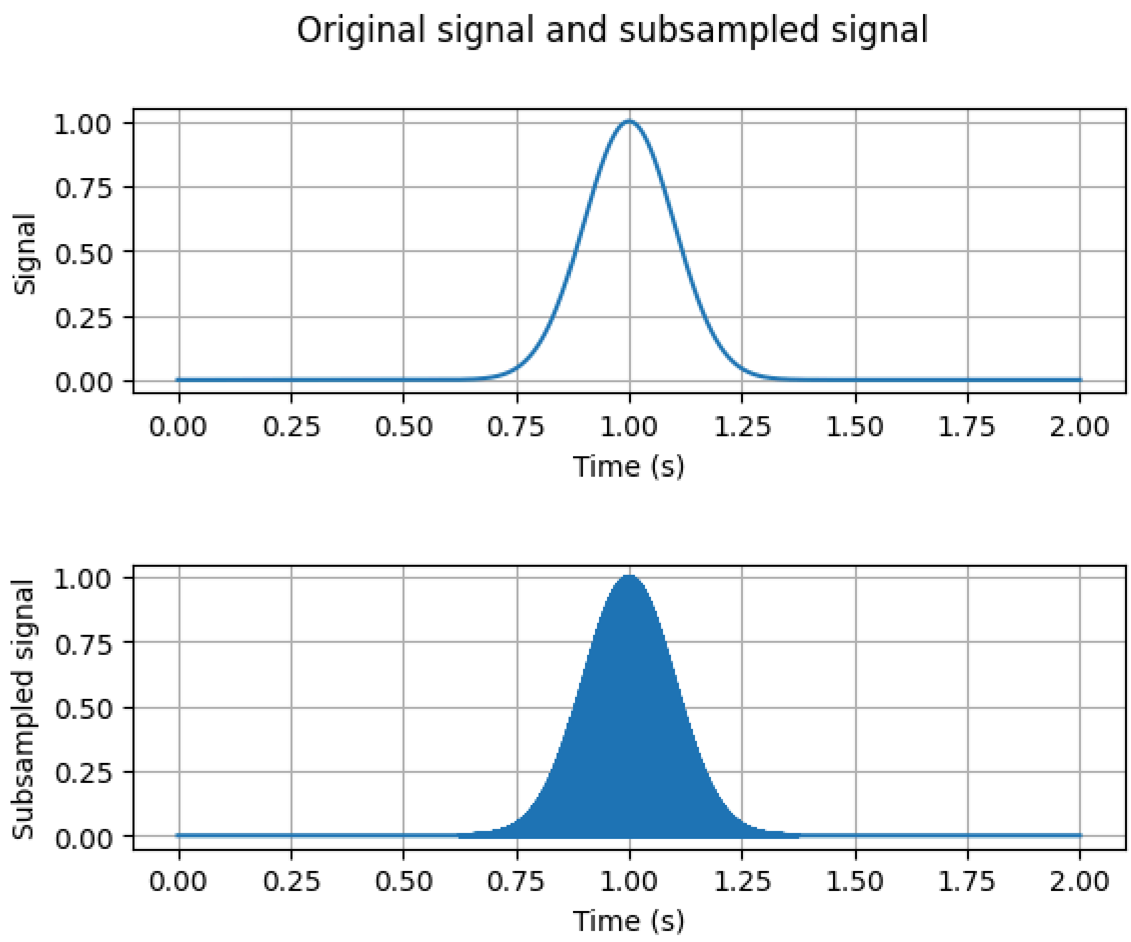
In [ ]:

```
def gaussian_pulse(mu, sigma, T, fs):
    """
    Generate a gaussian pulse with mean mu and std sigma

    Args:
        mu: mean
        sigma: standard deviation
        T: duration of the pulse
        fs: sampling frequency
    Returns:
        x: signal
        t: real time instants t = [0, T_s, ... , NT_s=T]
    """

    # TO DO: IMPLEMENT
    t = np.linspace(0, T, int(fs * T), endpoint=False)
    x = np.exp(-0.5 * ((t - mu) / sigma) ** 2)
    return x, t


def subsample(x, T_s, tau, prefilter = False):
    """
    Subsample the discrete signal x

    Args:
        x: discrete signal to be subsampled
        T_s: sampling time of x
        tau: sampling time of the subsampled signal
```

```python
        prefilter: anti-aliasing filtering (ignore until Q2.4)
    Returns:
        x_s: sampled signal
        x_delta: x_s padded with zeros, i.e., has the same support as x (same nu
    """

    # TO DO: IMPLEMENT
    factor = int(tau / T_s)

    if prefilter:
      fs_new = 1 / tau
      fs_original = 1 / T_s
      nu = fs_new / 2
      X, f = compute_fft(x, fs_original)
      mask = np.abs(f) >= nu
      X_filtered = X.copy()
      X_filtered[mask] = 0
      x_filtered, t = compute_ifft(X_filtered, fs_original)
      x_s = x_filtered[::factor]

    else:
      x_s = x[::factor]


    x_delta = np.zeros_like(x)
    x_delta[::factor] = x_s

    return x_s, x_delta
```

In [ ]:

```python
mu = 1
sigma = 0.1
f_s = 40000
f_ss = 4000
T = 2
N = int(T*f_s)

# Create a Gaussian pulse
x, t = gaussian_pulse(mu, sigma, T, f_s)

# Subsample the Gaussian pulse
x_s, x_delta = subsample(x, 1/f_s, 1/f_ss)

# Plot the original and subsampled signal
plot_signal_and_subsampled_signal(x, x_delta, t)
```



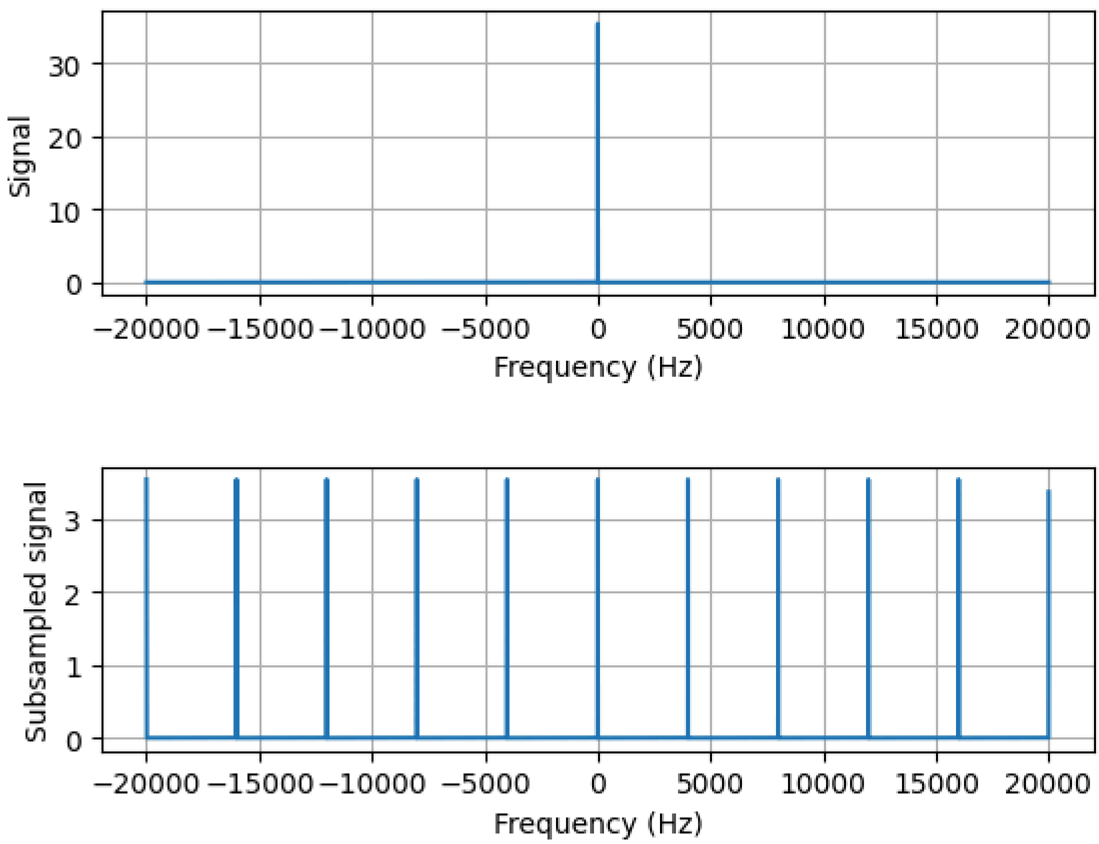
This plot function is bad since there is zeros in the X_delta function which result in a black body for the graph because it draws a line back and forth from up to down. We can see the effect of subsampling if we make F_ss = 40 so that we subsamplling is visible.

Q2.3 Spectrum Periodization

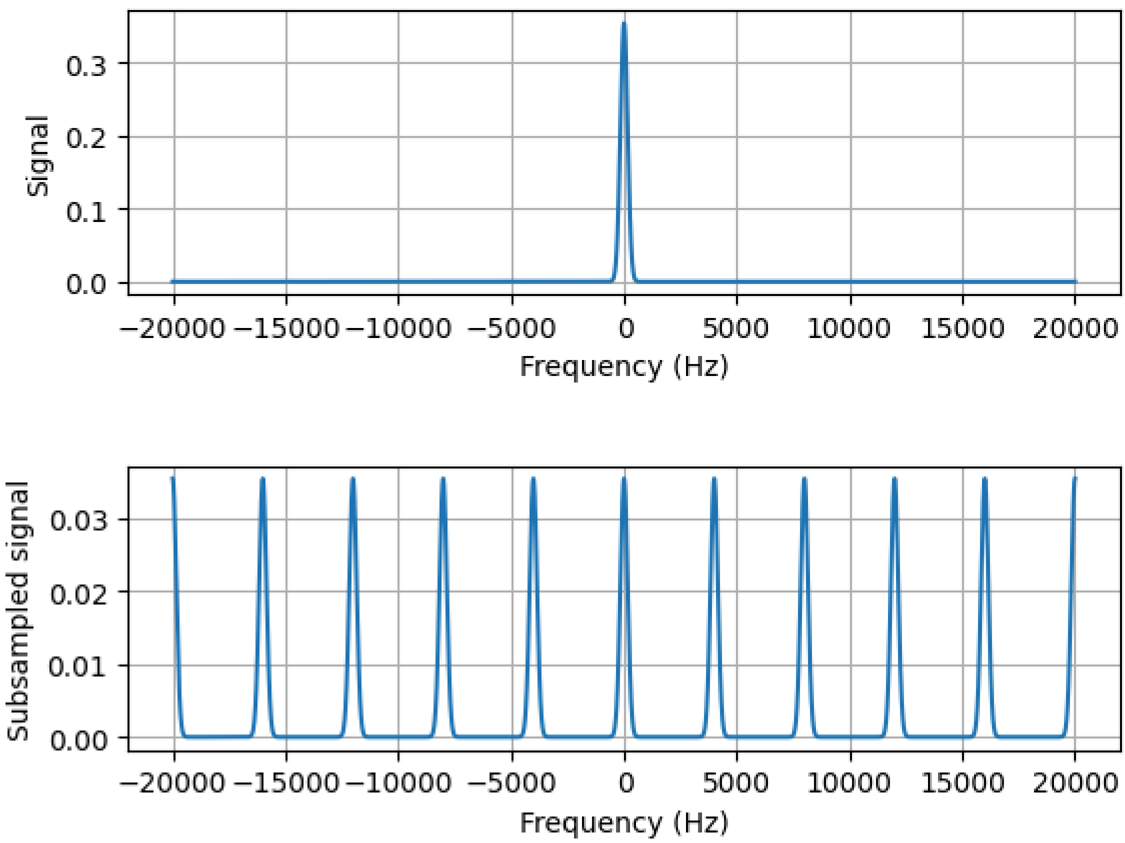
In [ ]:

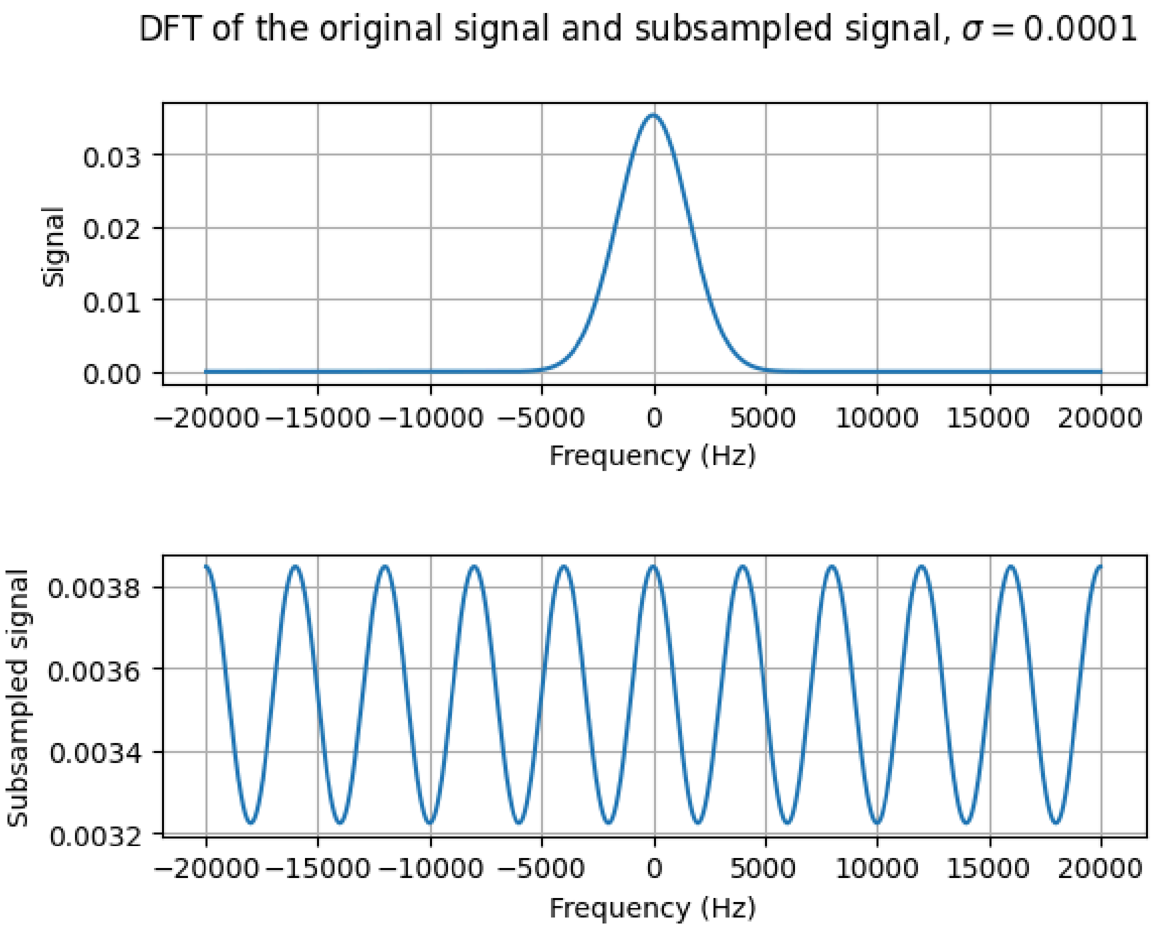
```
sigmas = [0.1, 1e-3, 1e-4, 1e-5]

for sigma in sigmas:
  x, t = gaussian_pulse(mu, sigma, T, f_s)
  x_s, x_delta = subsample(x, 1/f_s, 1/f_ss)

  X, f = compute_fft(x, f_s)
  X_delta, f_delta = compute_fft(x_delta, f_s)

  fig, _ = plot_dft_of_signal_and_subsampled_signal(X, X_delta, f)
  fig.suptitle('DFT of the original signal and subsampled signal, $\sigma = {}$'
```

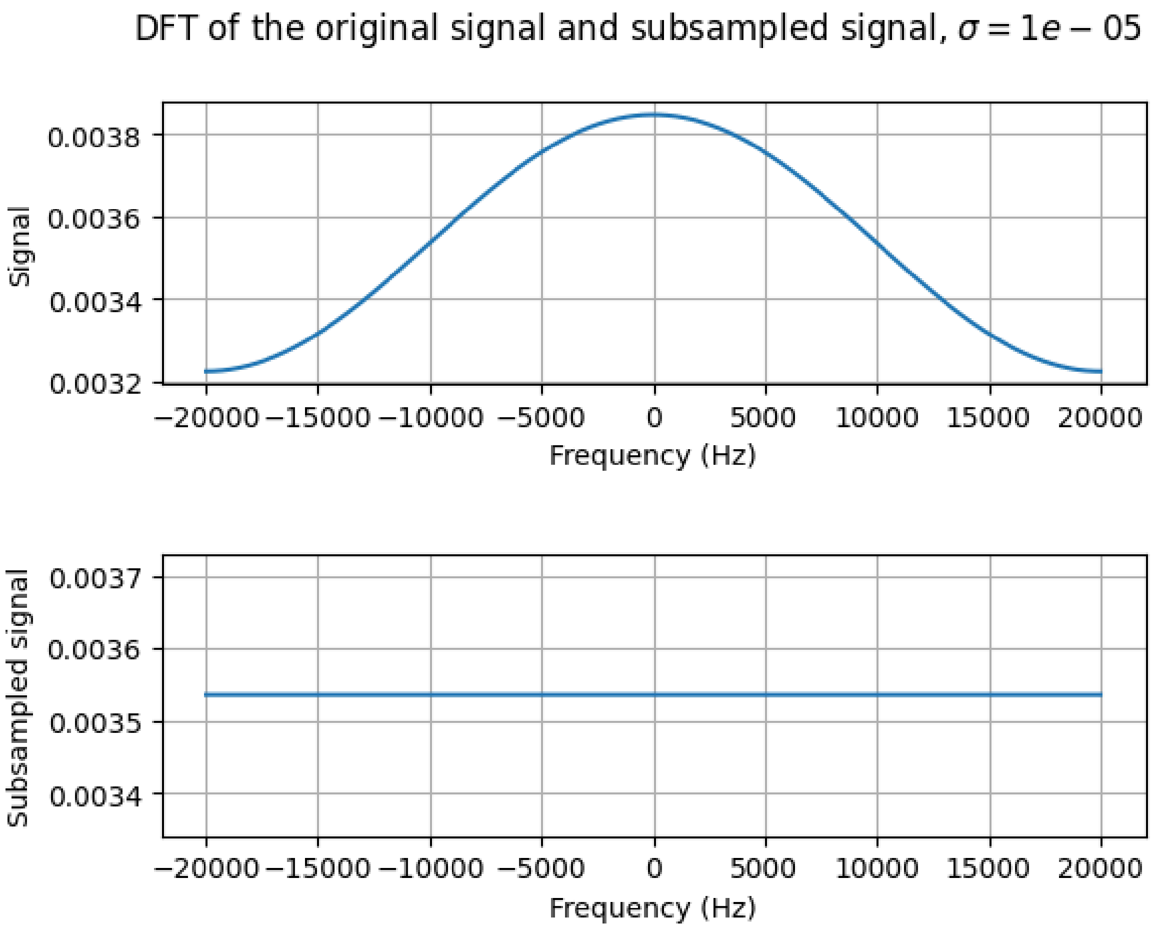
DFT of the original signal and subsampled signal, $\sigma = 0.1$



DFT of the original signal and subsampled signal, $\sigma = 0.001$





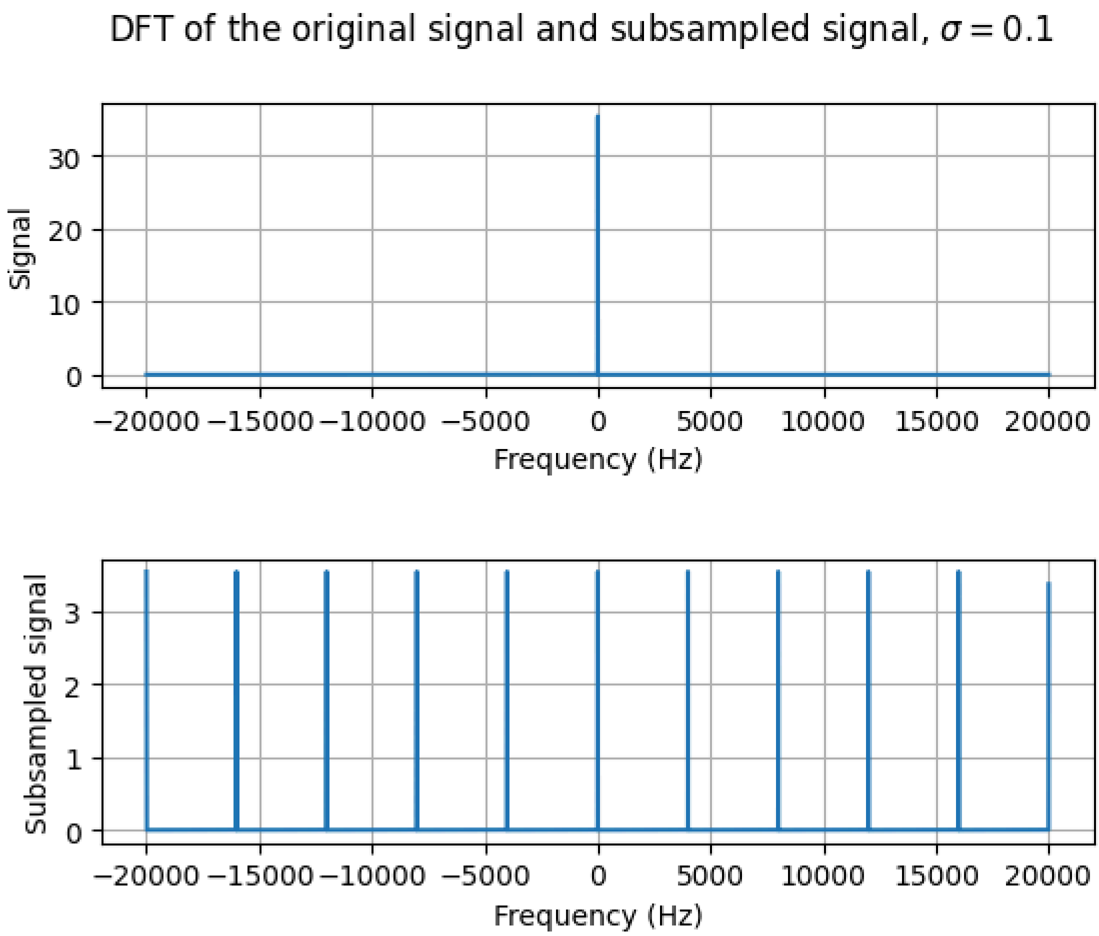


Q2.4 Prefiltering

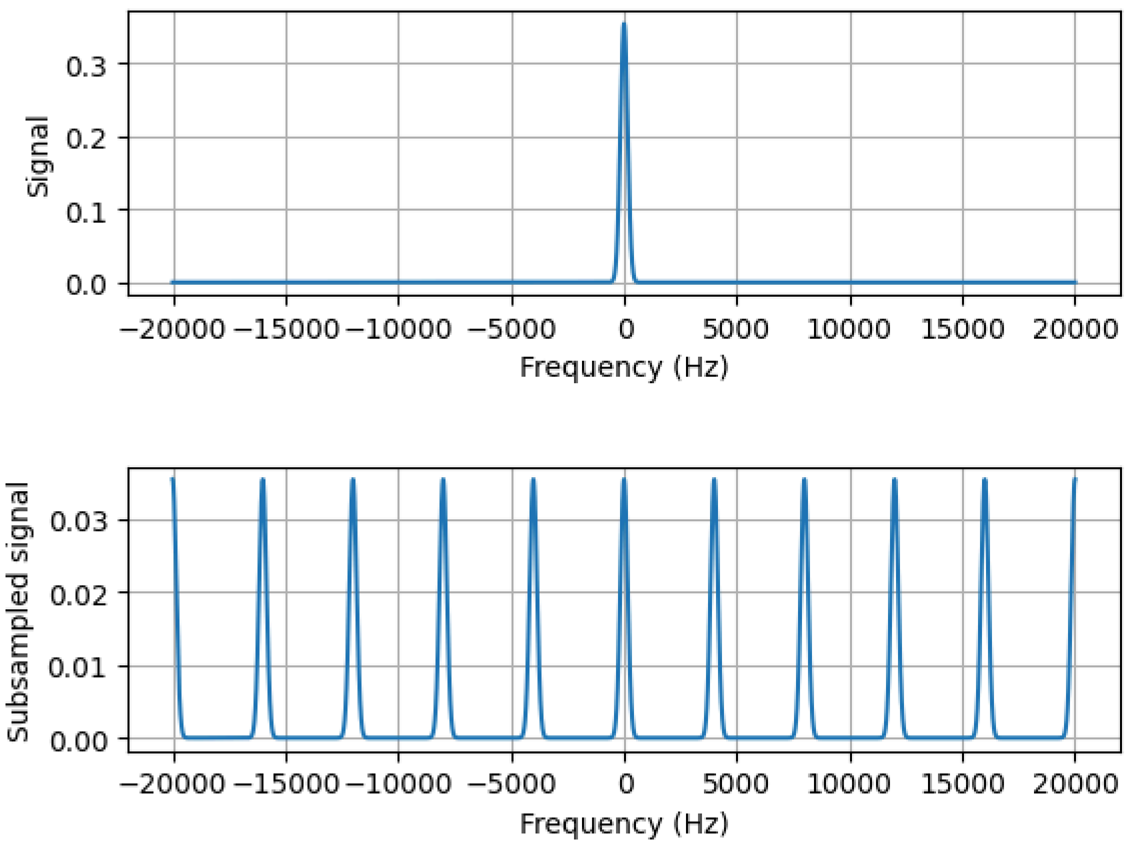
Go back and modify the `subsample` function to add prefiltering.

Q2.5 Spectrum Periodization with Prefiltering

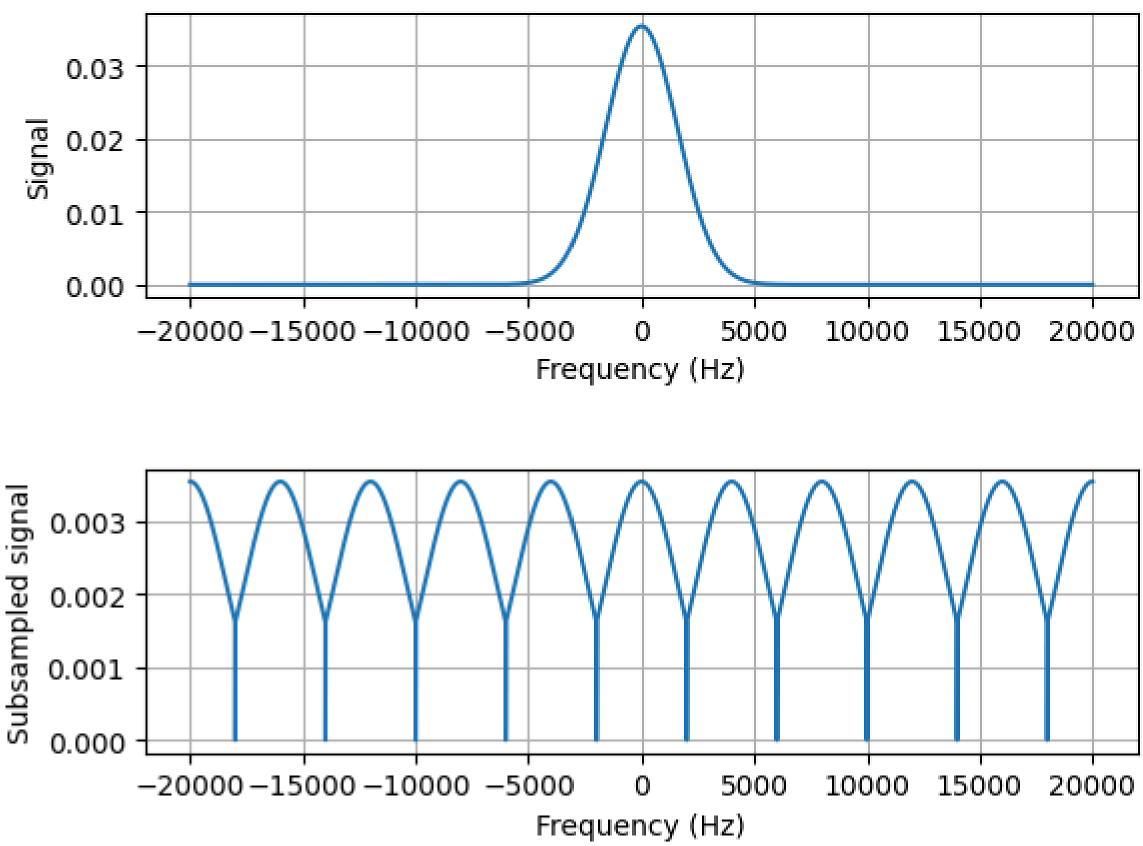
In [ ]:
```
sigmas = [0.1, 1e-3, 1e-4, 1e-5]

for sigma in sigmas:
  x, t = gaussian_pulse(mu, sigma, T, f_s)
  x_s, x_delta = subsample(x, 1/f_s, 1/f_ss, prefilter=True)

  X, f = compute_fft(x, f_s)
  X_delta, f_delta = compute_fft(x_delta, f_s)

  fig, _ = plot_dft_of_signal_and_subsampled_signal(X, X_delta, f)
  fig.suptitle('DFT of the original signal and subsampled signal, $\sigma = {}$'
```

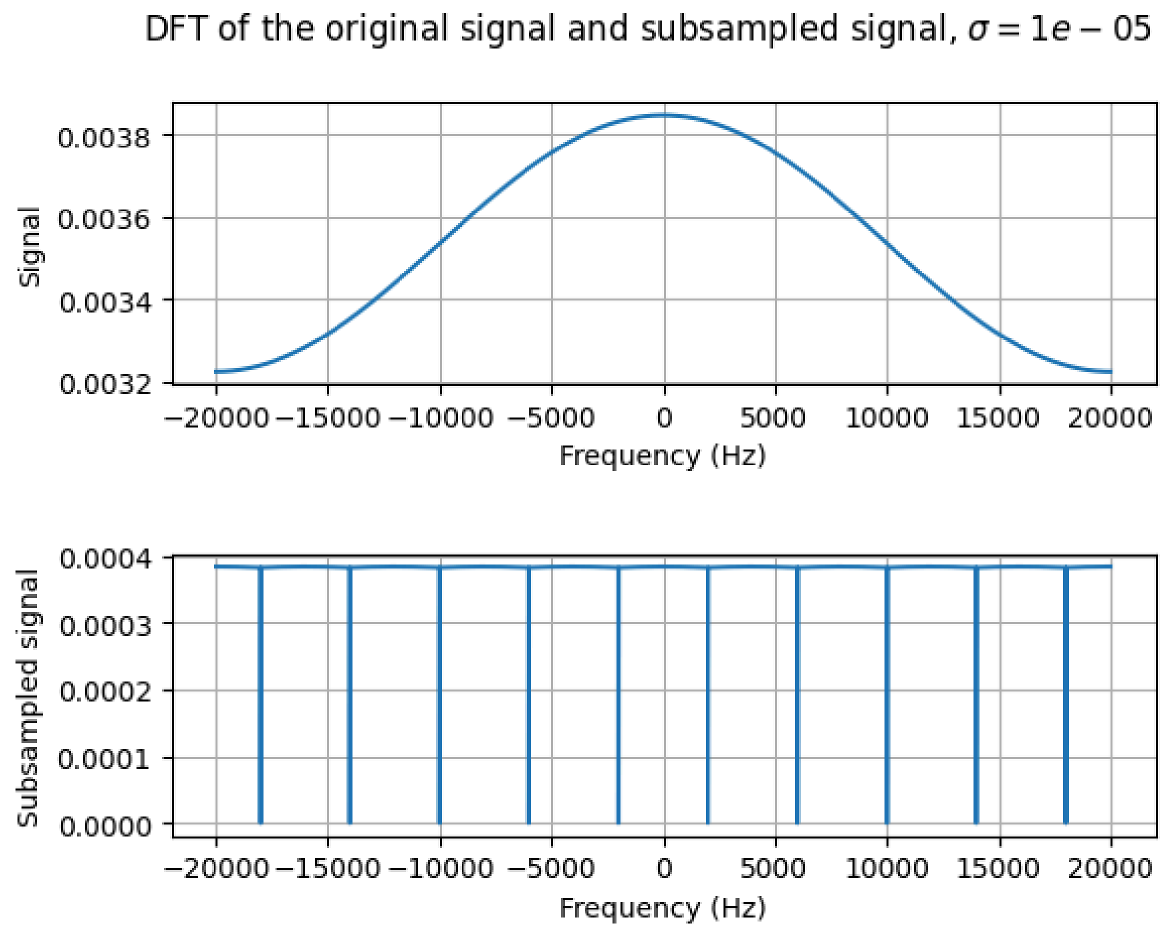


DFT of the original signal and subsampled signal, $\sigma = 0.001$



DFT of the original signal and subsampled signal, $\sigma = 0.0001$





In the previous part, aliasing happened when σ = 1e-4 or σ = 1e-5. Those cases had their frequency spectrum exceed [-fss/2, fss/2] which is [-2000, 2000]. (Where fss is the subsampling frequency.) Now, with prefiltering, we see that the subsampled signal's spectrum no longer contains distortion effects. For both cases, we see that X_delta(f) = 0 at every regular fss interval (every 4000) and at other points, there is a repeated pattern of just the slice of the original sample from [-2000, 2000] repeated across all other intervals. This repetition is a perfect copy of the original siganl's spectra without distortion. There is no more aliasing from overlapping repeated spectra distorting the non-bandlimitted signal.

Q2.6 Reconstruction Function

We leave this implementation to you.

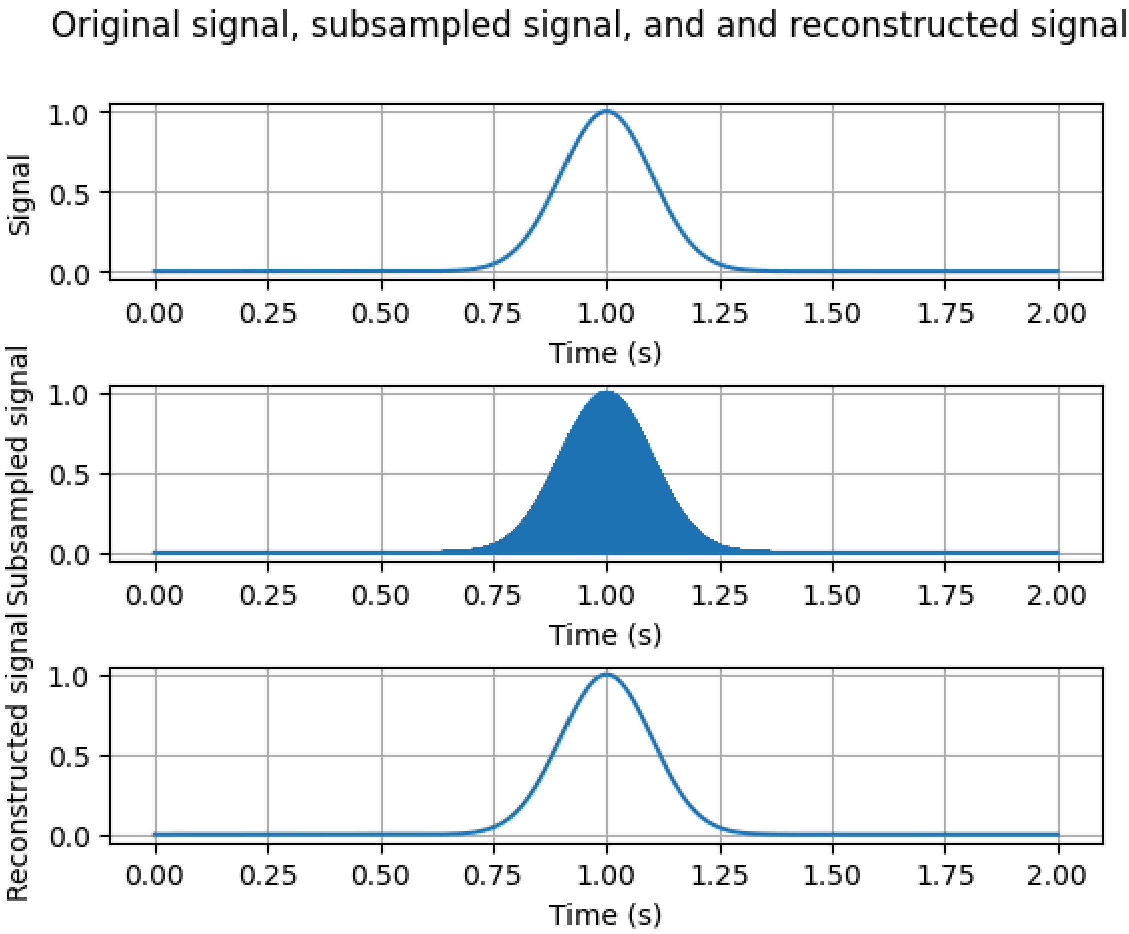
In [ ]:

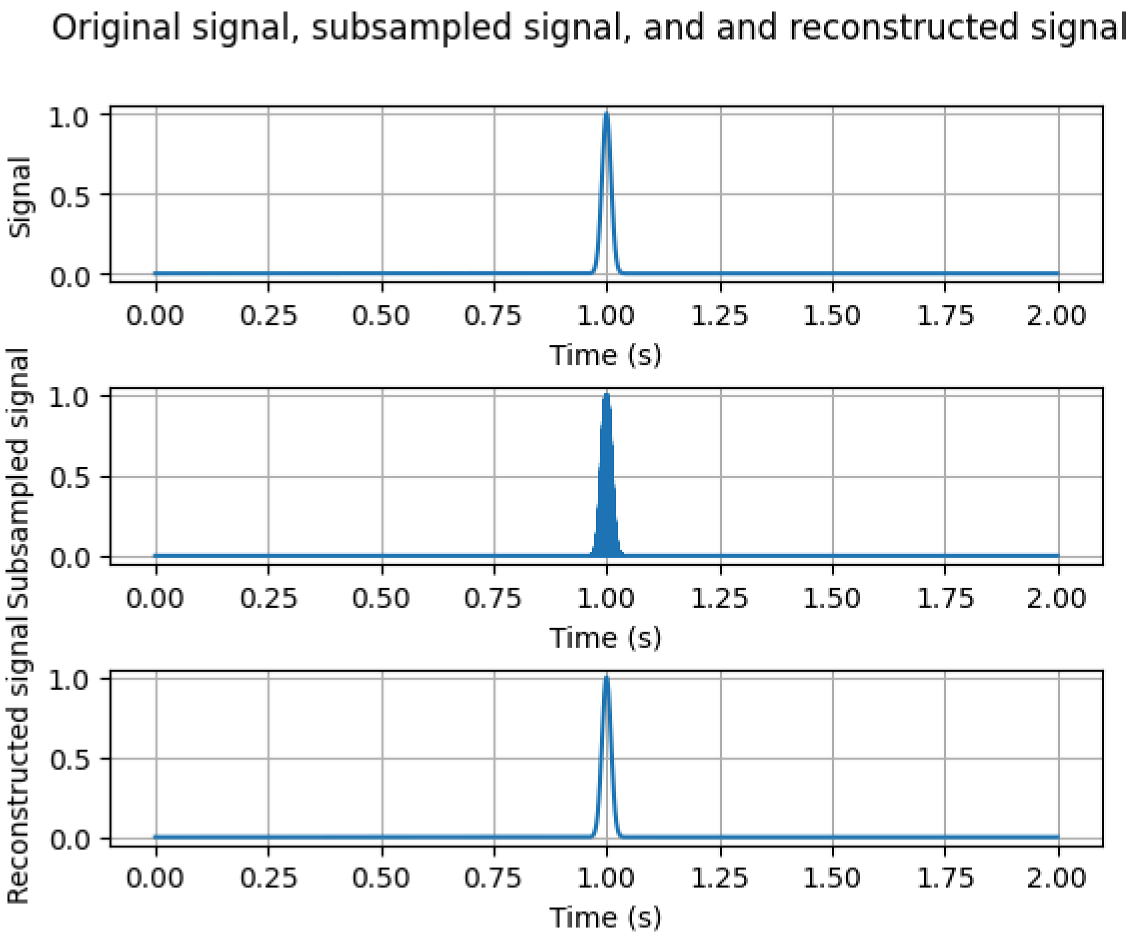
```
def reconstruct_subsampled_signal(xs, Tss, Ts):
    """
    xs: subsampled signal
    Tss: subsampling time
    Ts: sampling time

    returns: xd: reconstructed signal

    assume xs is bandlimited and has no aliasing.
    """
    # so I'm guessing we do this with FFT.
```

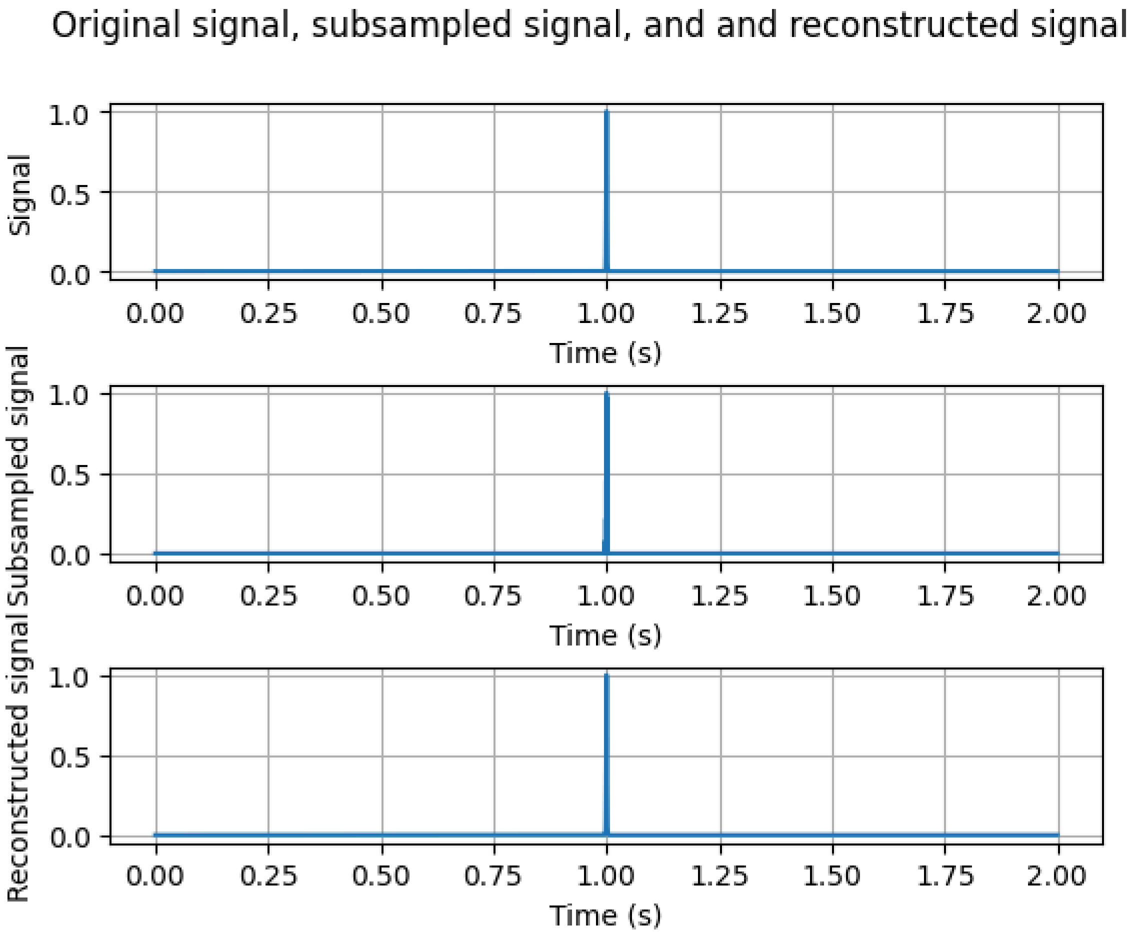
```
  N = len(xs)
  factor = int(Tss/Ts)
  xd = np.zeros(N*factor)
  xd[::factor] = xs
  X, f = compute_fft(x, f_s)
  # now we bandlimit X to [-fss/2, fss/2]
  fss = 1/Tss
  nu = fss/2
  mask = np.abs(f) >= nu
  X_filtered = X.copy()
  X_filtered[mask] = 0
  xd_filtered, t = compute_ifft(X_filtered, f_s)

  return xd_filtered
```

In [ ]:
```
def plot_signal_and_subsampled_and_reconstructed_signal(x, x_s, x_delta, t):
    """
    Plot the original signal and the reconstructed signal

    Args:
        x: Original (time) signal of length N
        x_delta: Subsampled (time) signal of length N (zero padded)
        t: time instants [0, T_s, ... , NT_s=T]
    """

    fig, axs = plt.subplots(3, 1)
    fig.suptitle('Original signal, subsampled signal, and and reconstructed sign
    fig.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=Non
    axs[0].plot(t, x)
    axs[0].set_xlabel('Time (s)')
    axs[0].set_ylabel('Signal')
    axs[1].plot(t, x_s)
    axs[1].set_xlabel('Time (s)')
    axs[1].set_ylabel('Subsampled signal')
    axs[2].plot(t, x_delta)
    axs[2].set_xlabel('Time (s)')
    axs[2].set_ylabel('Reconstructed signal')
    axs[0].grid()
    axs[1].grid()
    axs[2].grid()
```

In [ ]:
```
# Let's test on the gaussian pulses that did not result in aliasing.
sigmas = [0.1, 1e-2, 1e-3]

for sigma in sigmas:
  x, t = gaussian_pulse(mu, sigma, T, f_s)
  x_s, x_delta = subsample(x, 1/f_s, 1/f_ss)

  xd = reconstruct_subsampled_signal(x_s, 1/f_ss, 1/f_s)

  plot_signal_and_subsampled_and_reconstructed_signal(x, x_delta, xd, t)
```


Original signal, subsampled signal, and and reconstructed signal
Signal
Subsampled signal
Reconstructed signal
Time (s)
0.0
0.5
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Original signal, subsampled signal, and and reconstructed signal
Signal
Subsampled signal
Reconstructed signal
Time (s)
0.0
0.5
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Original signal, subsampled signal, and and reconstructed signal
Signal
Subsampled signal
Reconstructed signal
1.0
0.5
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Time (s)