STFT APPROXIMATION USING R&F NEURONS

This is an implementation of the first part of this paper: https://arxiv.org/pdf/2111.03746.pdf

Let's start out with Bamsumit's code for an approximation of an STFT

Link to original post here: https://github.com/lava-nc/lava-dl/discussions/198#discussioncomment-5971547

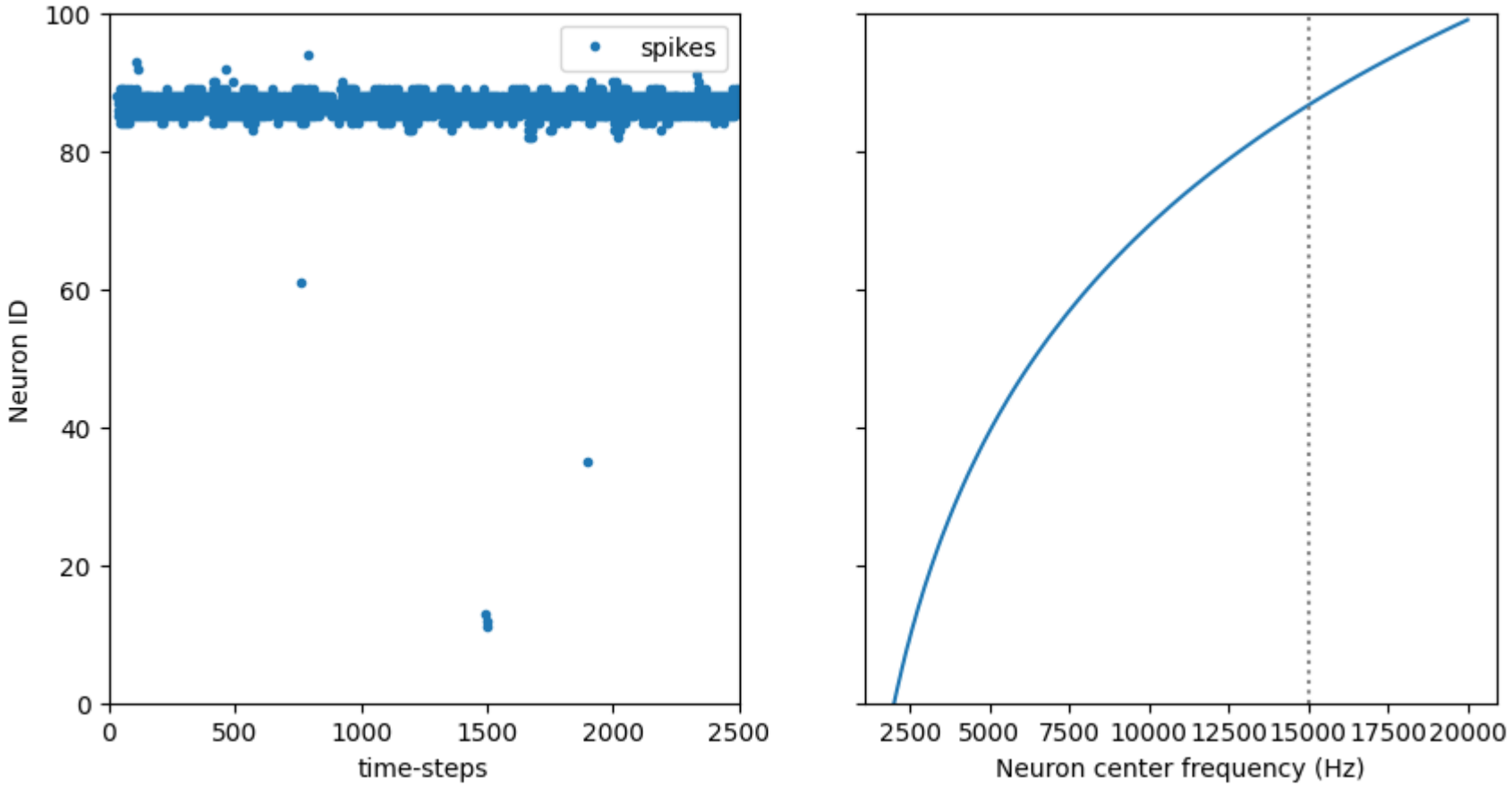
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from lava.lib.dl import slayer
import torch
from matplotlib import pyplot as plt

fs = 100e3  # 100 KHz of sampling frequency

fc = 15e3   # 15 KHz is the frequency of the actual sine wave

sinusoid = 2.2 * np.sin(2 * np.pi * fc / fs * np.arange(2500))
noise = np.random.randn(sinusoid.shape[0])

input = torch.FloatTensor(sinusoid + noise).reshape([1, 1, -1])
rf_params = dict(threshold=2,
                 period= [5, 50], # this is equivalent to a frequency between 1/(50 * 1/fs) and 1/(5 * 1/fs)
                 decay=0.01,
                 shared_param=False,
                 log_init=True)
block = slayer.block.rf.Dense(neuron_params=rf_params, in_neurons=1, out_neurons=100) # 100 determines the width of the
# Overwrite the weight initialization values
block.synapse.real.weight.data = 0.1 * torch.ones_like(block.synapse.real.weight.data)
block.synapse.imag.weight.data = 0.0 * torch.ones_like(block.synapse.imag.weight.data)
out = block(input)

# Plot spiking events of RF population
ev = np.argwhere(out[0].cpu().data.numpy() > 0)
fig, ax = plt.subplots(1, 2, figsize=(10, 5), sharey=True)
```

```
ax[0].plot(ev[:, 1], ev[:, 0], '.', label='spikes')
ax[0].axis([0, input.shape[-1], 0, out.shape[1]])
ax[1].plot(block.neuron.frequency * fs, np.arange(out.shape[1]))
ax[1].vlines(x=fc, ymin=0, ymax=1000, color='gray', linestyle='dotted')
ax[0].set_xlabel('time-steps')
ax[0].set_ylabel('Neuron ID')
ax[0].legend()
ax[1].set_xlabel('Neuron center frequency (Hz)')
```

```
C:\Users\mjurado3\workspace\lava_dev\lava-dl-fork\.venv\lib\site-packages\tqdm\auto.py:22: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Out[1]:

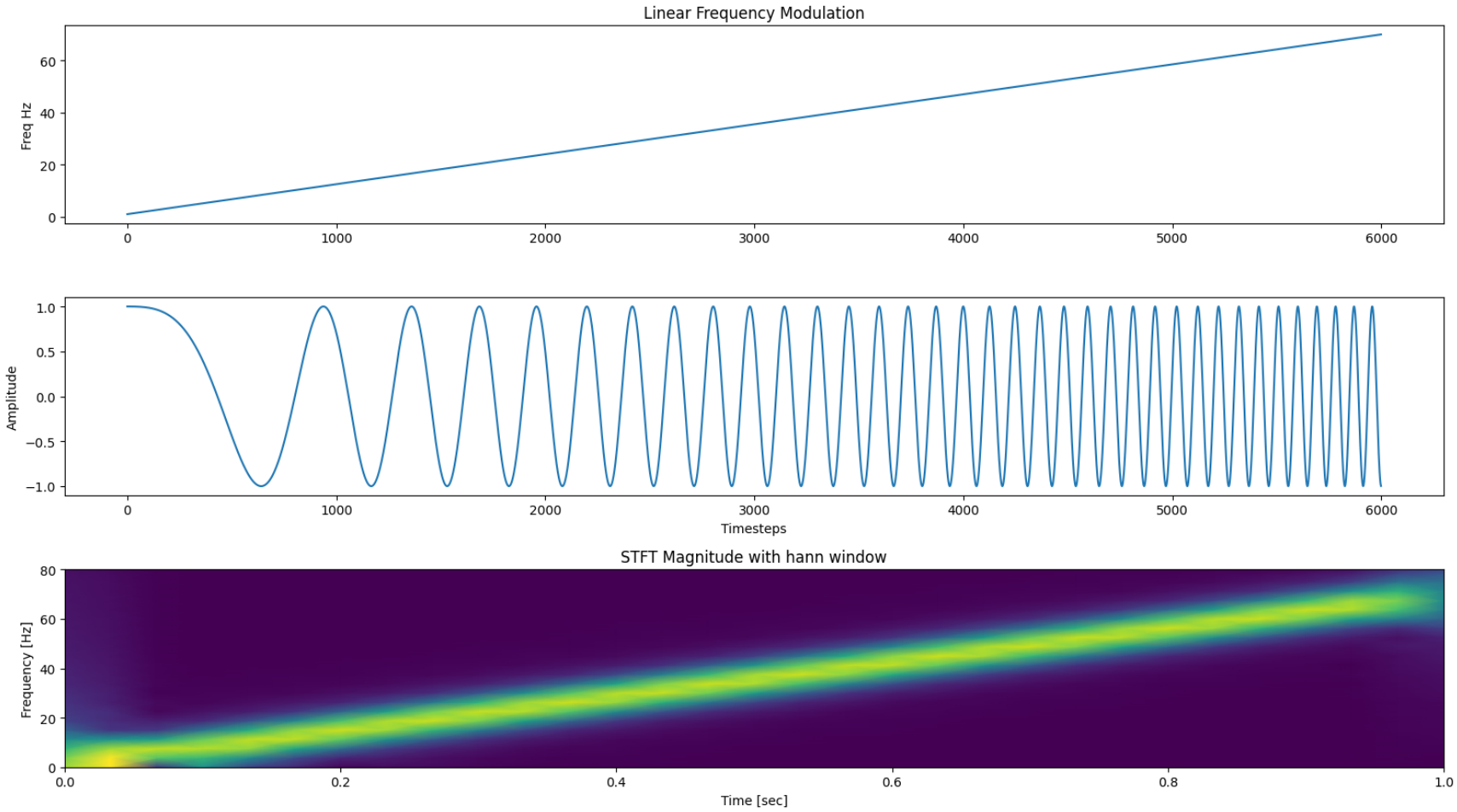
```
Text(0.5, 0, 'Neuron center frequency (Hz)')
```



Let's try a lfm signal and an STFT

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import chirp, stft, get_window
import numpy as np
import matplotlib.pyplot as plt

y = []
start_frequency = 1
end_frequency = 70
timesteps = np.arange(6000)
hz_frequencies = np.linspace(start_frequency, end_frequency, 6000)
fs = 6000
T = 1.0
t = np.arange(0, T, 1/fs)
waveform = chirp(t, f0=start_frequency, f1=end_frequency, t1=T, method='linear')
fig, (ax1, ax2, ax3) = plt.subplots(3, figsize=(16, 9))
ax1.plot(timesteps, hz_frequencies)
ax1.set_ylabel("Freq Hz")

ax2.plot(timesteps, waveform)
ax2.set_xlabel("Timesteps")
ax2.set_ylabel("Amplitude")
ax1.set_title("Linear Frequency Modulation")

frequencies, times, Zxx = stft(waveform, fs = 6000, nperseg=1000, nfft=1600, window="hann", noverlap=800)#,  nfft=3000,
magnitude = np.abs(Zxx)
ax3.pcolormesh(times, frequencies, magnitude, shading='gouraud')
ax3.set_title(f'STFT Magnitude with {"hann"} window')
ax3.set_ylabel('Frequency [Hz]')
ax3.set_xlabel('Time [sec]')
ax3.set_ylim(0, 80)
#ax3.set_xlim(0, .4)

plt.tight_layout()
plt.show()
```

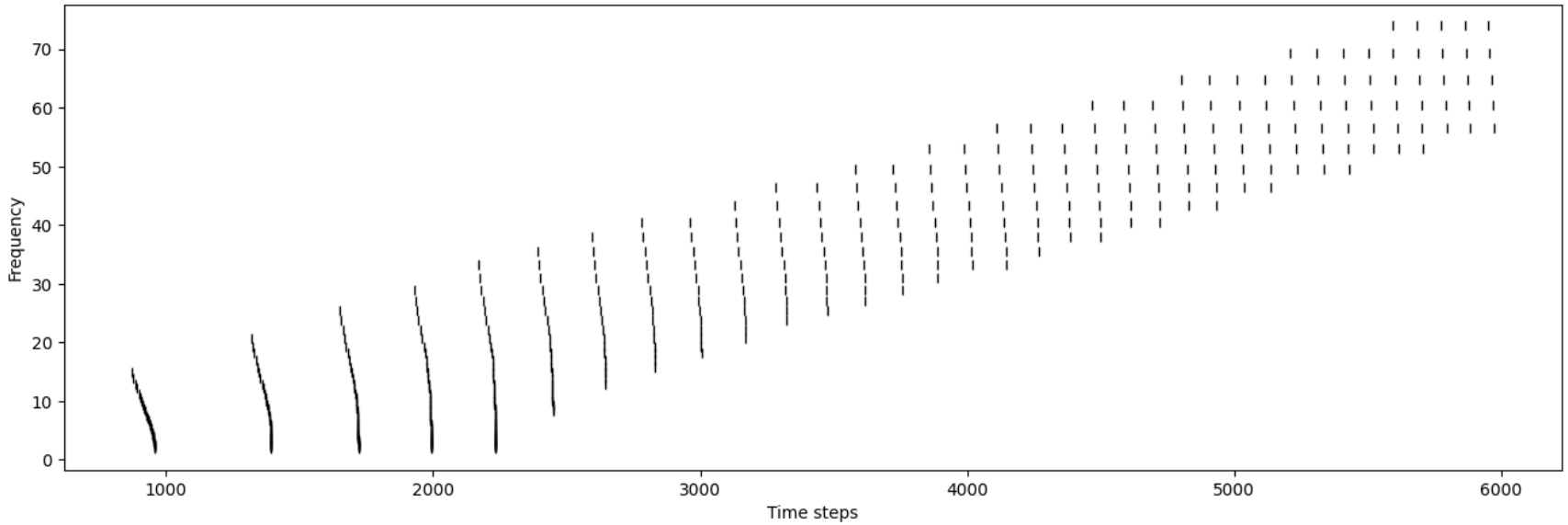


Let's write a resonate and fire spike encoder

In [7]:
```
def rf_encode(waveform, n_fft = 100, threshold = 2, decay = .01, weight_factor = .1, graded=False):
    waveform = waveform.unsqueeze(0).unsqueeze(0) # this makes the data 1, 1, timesteps
    rf_params = dict(threshold=threshold,
                     period= [4,3000],
                     decay=decay,
                     shared_param=False,
                     log_init=True,
                    )

    block = slayer.block.rf.Dense(neuron_params=rf_params, in_neurons=1, out_neurons=n_fft)
```

```
    # Overwrite the weight initialization values
    block.synapse.real.weight.data = weight_factor * torch.ones_like(block.synapse.real.weight.data)
    block.synapse.imag.weight.data = 0.0 * torch.ones_like(block.synapse.imag.weight.data)

    rf_spikes = block(waveform)
    return rf_spikes[0], block.neuron.frequency
```

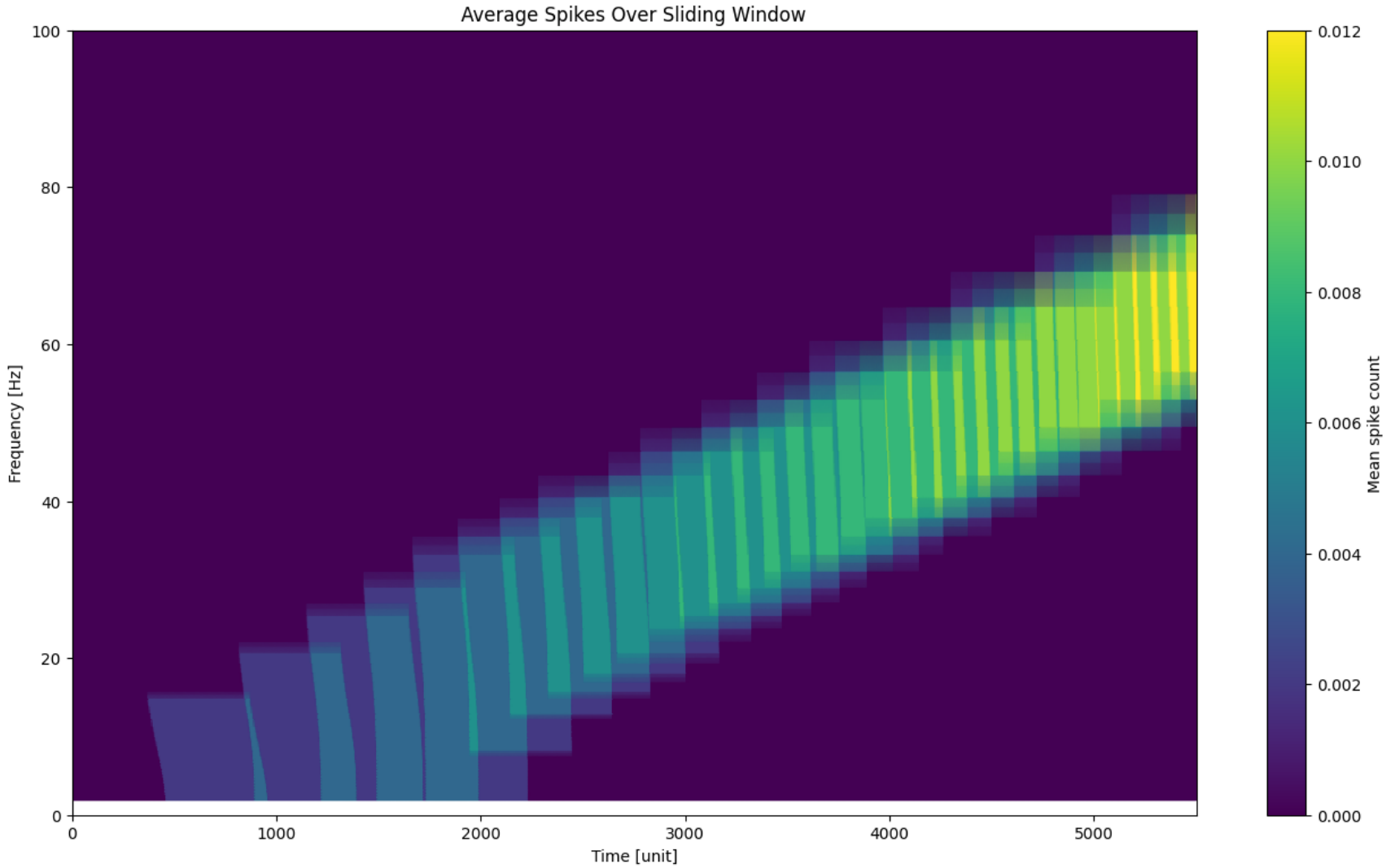
In [8]:
```
spikes, frequencies = rf_encode(torch.from_numpy(waveform).float(), threshold=3)
# Generate a raster plot
plt.figure(figsize=(16,5))

for idx, neuron in enumerate(spikes):
    spike_times = np.where(neuron > 0)[0]
    plt.plot(spike_times, frequencies[idx] * 6000 * np.ones_like(spike_times), '|', color='black')

plt.xlabel('Time steps')
plt.ylabel('Frequency')
plt.show()
```



Let's make an STFT like output from the spike raster

In [9]:
```
import matplotlib.pyplot as plt
import numpy as np
```

```
w = 500

window_view = np.lib.stride_tricks.sliding_window_view(spikes.detach().numpy(), (1, w))
mean_spikes = np.squeeze(window_view).mean(axis=-1)

# compute time (we'll just consider each time step as a unit of time)
times = np.arange(mean_spikes.shape[-1])

fig, ax = plt.subplots(figsize=(16, 9))

# generate a heatmap
c = ax.pcolormesh(times, frequencies*6000, mean_spikes, shading='gouraud', cmap='viridis')

ax.set_title("Average Spikes Over Sliding Window")
ax.set_ylabel('Frequency [Hz]')
ax.set_xlabel('Time [unit]')
ax.set_ylim(0, 100)

# set colorbar
fig.colorbar(c, ax=ax, label="Mean spike count")

plt.show()
```



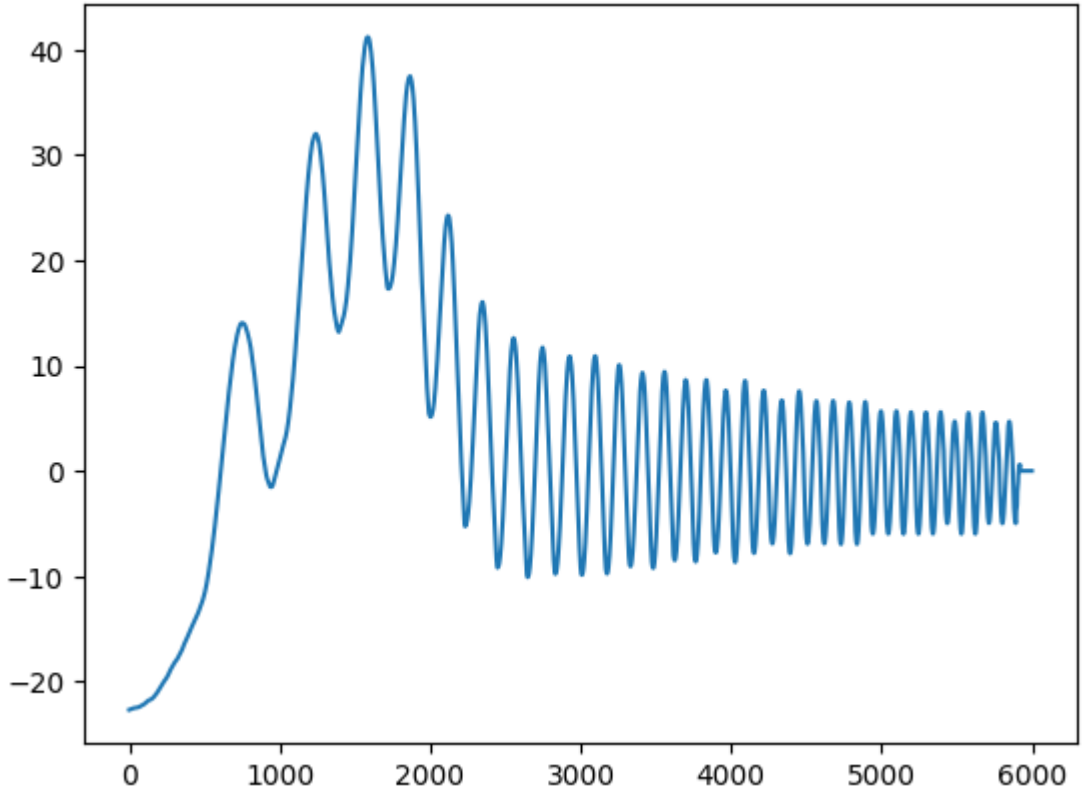
Let's attempt to 'decode' the spike raster

In [10]:
```
import torch
import numpy as np
print(spikes.sum())

# parameters
n_neurons = 100
```

```
timesteps =   6000
spike_data = spikes

kernel_size = 500


kernels = []
for freq in frequencies:
    # Generate a sine wave that lasts for one period
    kernel_size = int(1/freq)
    kernel = np.sin(2 * np.pi * freq * np.arange(kernel_size))
    kernels.append(torch.from_numpy(kernel).float())

output = torch.zeros(timesteps)
for i in range(n_neurons):
    # convolve the spikes with the neuron's oscillation kernel
    convolved = torch.conv1d(spike_data[i][None, None, ...], kernels[i][None, None, ...], padding="valid")
    output[:convolved.shape[-1]] += convolved[0][0]

plt.plot(output.detach().numpy())
```

Out[10]:

```
tensor(449., grad_fn=<SumBackward0>)
[<matplotlib.lines.Line2D at 0x25e8915ca90>]
```



In [ ]:

In [ ]: