moving-average

December 6, 2022

[ ]:
```
# Packages and adjustments to the figures
from scipy import signal
import matplotlib.pyplot as plt
#plt.rcParams.update(plt.rcParamsDefault)
import numpy as np
#plt.rcParams["figure.figsize"] = 10,5
#plt.rcParams["font.size"] = 12
#plt.rcParams.update({"text.usetex": True,"font.family": "sans-serif","font.
 ↪sans-serif": ["Helvetica"]})
```

https://www.dafx.de/paper-archive/2013/papers/06.dafx2013_submission_46.pdf

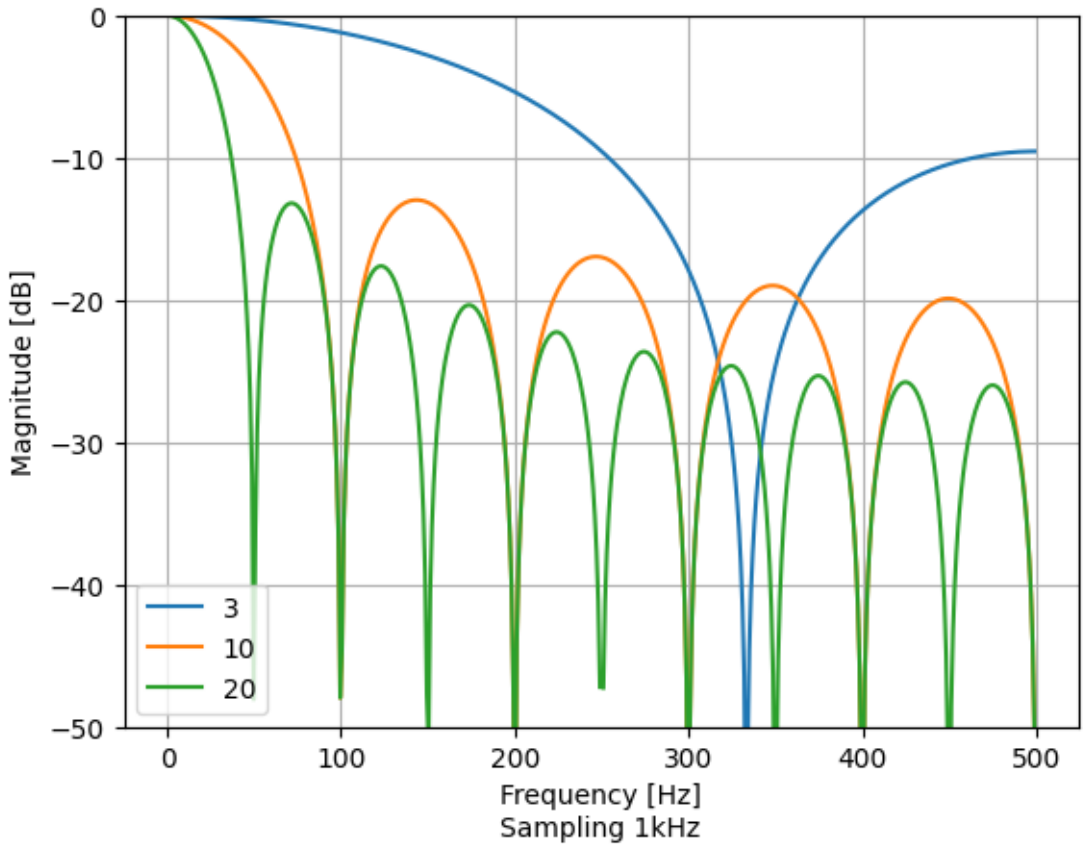
https://www.isprs.org/PROCEEDINGS/XXV/congress/part7/163_XXV-part7.pdf

Moving median is a complex non linear filter

[ ]:
```
import warnings
warnings.filterwarnings('ignore')

fs = 1000  # Hz
w = np.linspace(0, np.pi, 1024)
fig, ax = plt.subplots()
for samples in [3, 10, 20]:
    # numerator co-effs of filter transfer function
    b = (np.ones(samples))/samples
    a = np.ones(1)  # denominator co-effs of filter transfer function
    w, h = signal.freqz(b, a)
    f = w * fs / (2*np.pi)
    ax.plot(f, 20 * np.log10(abs(h)), label="" + str(samples))


plt.ylabel('Magnitude [dB]')
plt.xlabel('Frequency [Hz]\nSampling 1kHz')
plt.grid(True)
plt.ylim(-50, 0)
plt.legend()
plt.show()
```



IIR filter below, lookback of 1. Where n is a weigh for previous output

$Y_n = \frac{1}{n} * X_n + \frac{n-1}{n} * Y_{n-1}$

https://www.electronicdesign.com/technologies/analog/article/21778422/use-software-filters-to-reduce-adc-noise

[ ]:
```
def ma(x, n=5):
    b = np.repeat(1.0/n, n)  # Create impulse response
    xf = signal.lfilter(b, 1, x)  # Filter the signal
    return (xf)

def binomcoeffs(n):
    return (np.poly1d([0.5, 0.5])**n).coeffs


# This is IIR filter, all others here are FIRs
# Theory https://brianmcfee.net/dstbook-site/content/ch11-iir/IIRFilters.html
# This is is rather computationally simple https://www.electronicdesign.com/
 ↪technologies/analog/article/21778422/use-software-filters-to-reduce-adc-noise
```

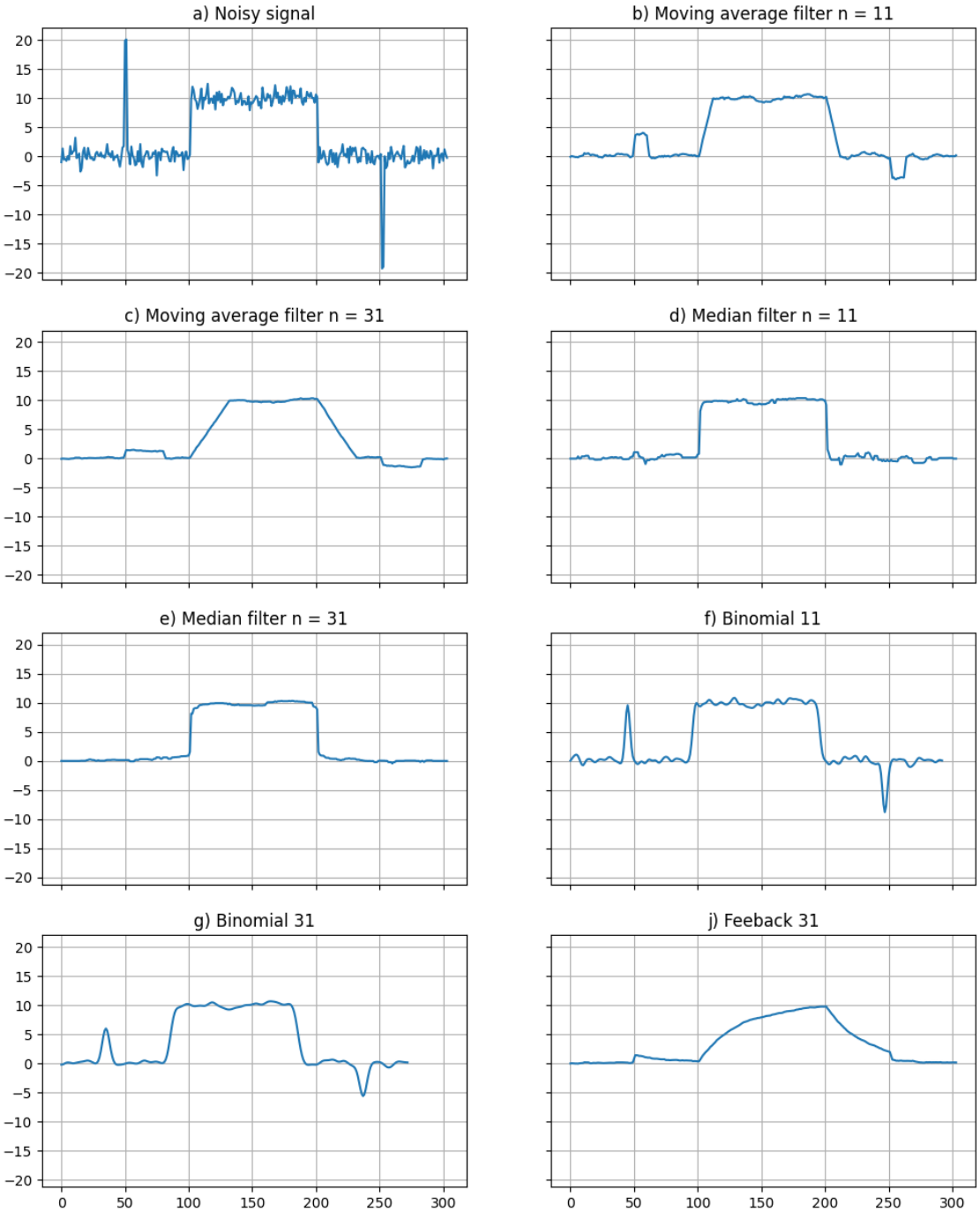
```
def feedback(x, n=5):
    result=[]
    y_prev = 0;
    for item in x:
        y_prev = 1.0/n*item+(n-1)/n*y_prev
        result.append(y_prev)

    return np.array(result);

def plotResponses(noisy):
    yBinomial11 = np.convolve(noisy, binomcoeffs(11), mode='valid')
    yBinomial31 = np.convolve(noisy, binomcoeffs(31), mode='valid')
    yFeedback = feedback(noisy, 31);
    # Moving average filters
    ym11 = ma(noisy, 11)
    ym31 = ma(noisy, 31)
    # Median filter from scipy.signal
    ymed11 = signal.medfilt(noisy, 11)
    ymed31 = signal.medfilt(noisy, 31)
    # Plots with shared x and y-axis
    f, ((ax1, ax2), (ax3, ax4), (ax5, ax6), (ax7, ax8)) = plt.subplots(
        nrows=4, ncols=2, sharex=True, sharey=True)
    ax1.plot(noisy)
    ax1.set_title("a) Noisy signal")
    ax2.plot(ym11)
    ax2.set_title("b) Moving average filter n = 11")
    ax3.plot(ym31)
    ax3.set_title("c) Moving average filter n = 31")
    ax4.plot(ymed11)
    ax4.set_title("d) Median filter n = 11")
    ax5.plot(ymed31)
    ax5.set_title("e) Median filter n = 31")
    ax6.plot(yBinomial11)
    ax6.set_title("f) Binomial 11")
    ax7.plot(yBinomial31)
    ax7.set_title("g) Binomial 31")
    ax8.plot(yFeedback)
    ax8.set_title("j) Feeback 31")
    # plt.ylim(-30,30);
    for ax in f.axes:
        ax.grid()
    f.set_size_inches(12, 15)
    plt.show()
```

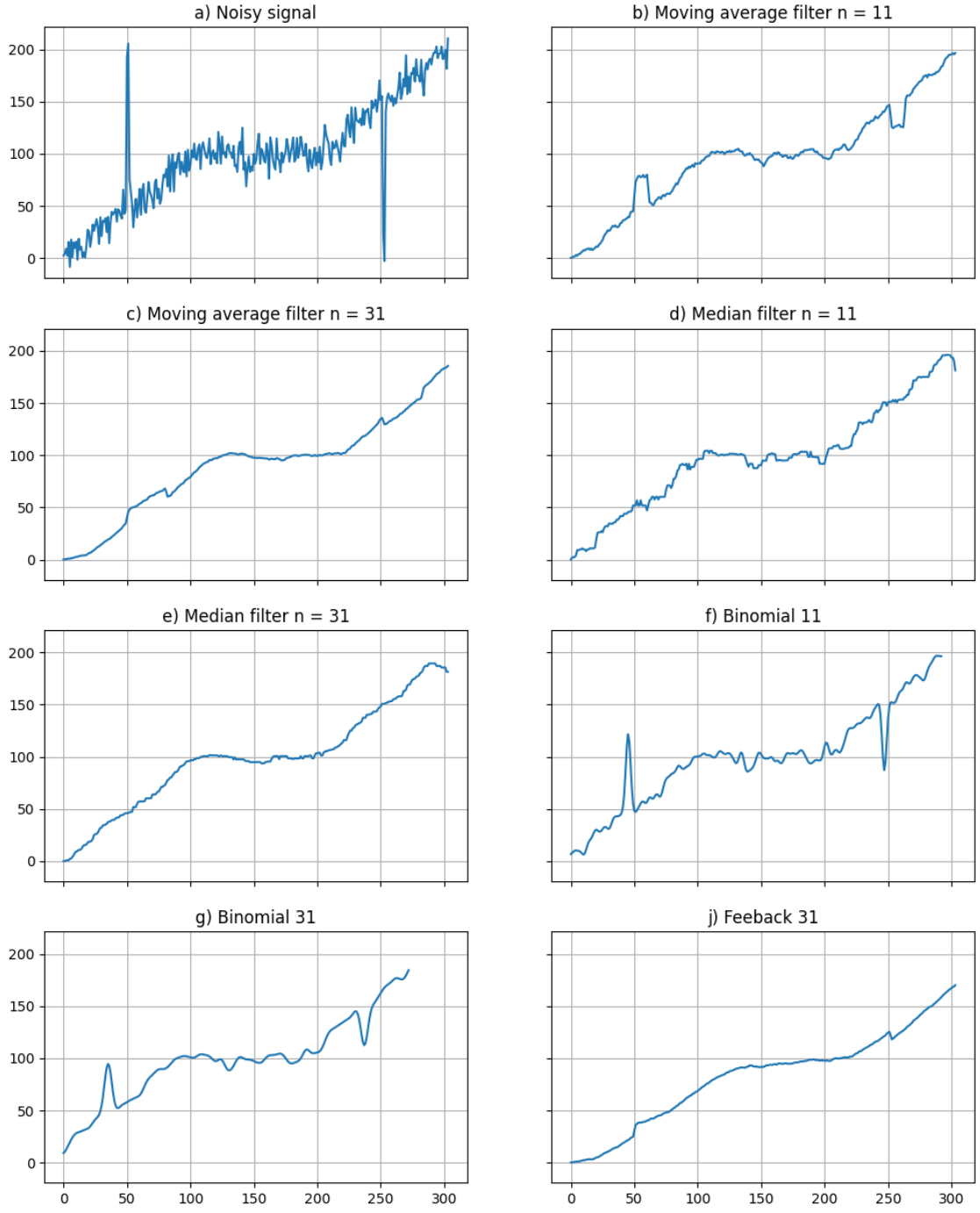
[ ]:
```
# simple random noise
ym = np.hstack([np.repeat(0, 50), np.repeat(20, 2), np.repeat(0, 50), np.repeat(
    10, 100), np.repeat(0, 50), np.repeat(-20, 2), np.repeat(0, 50)])
```

```
noisy = ym+np.random.randn(304)
plotResponses(noisy)
```



[ ]:
```
# simple random noise with spikes
```

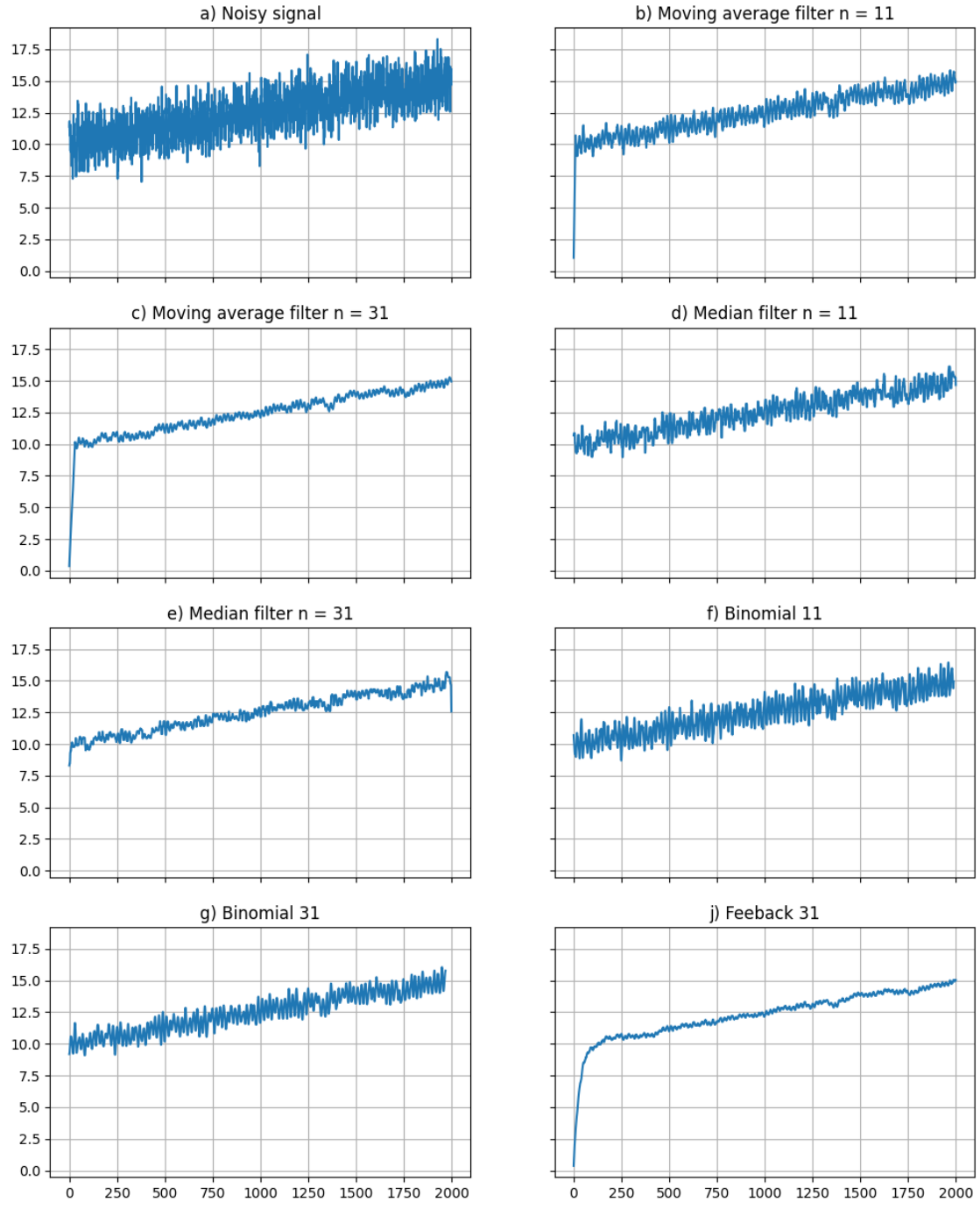
```
ym = np.hstack([np.arange(0, 50), np.repeat(200, 2), np.arange(50, 100), np.
  ↪repeat(
    100, 100), np.arange(100, 150), np.repeat(20, 2), np.arange(150, 200)])
noisy = ym+np.random.randn(304)*10
plotResponses(noisy)
```



[ ]:
```
# complex noise 50Hz hum super imposed with random noise

samplingFreq = 1000
dcBias = np.linspace(10, 15, 2*samplingFreq)
tlims = [0, 2]         # in seconds
signalFreq = [50, 150]  # Cycles / second
signalMag = [1, 0.3]  # magnitude of each sine
t = np.linspace(tlims[0], tlims[1], (tlims[1]-tlims[0])*samplingFreq)
y = np.random.randn(t.size)
y = dcBias + (np.random.randn(t.size)) + signalMag[0]*np.sin(
    2*np.pi*signalFreq[0]*t) + signalMag[1]*np.sin(2*np.pi*signalFreq[1]*t)

plotResponses(y)
```

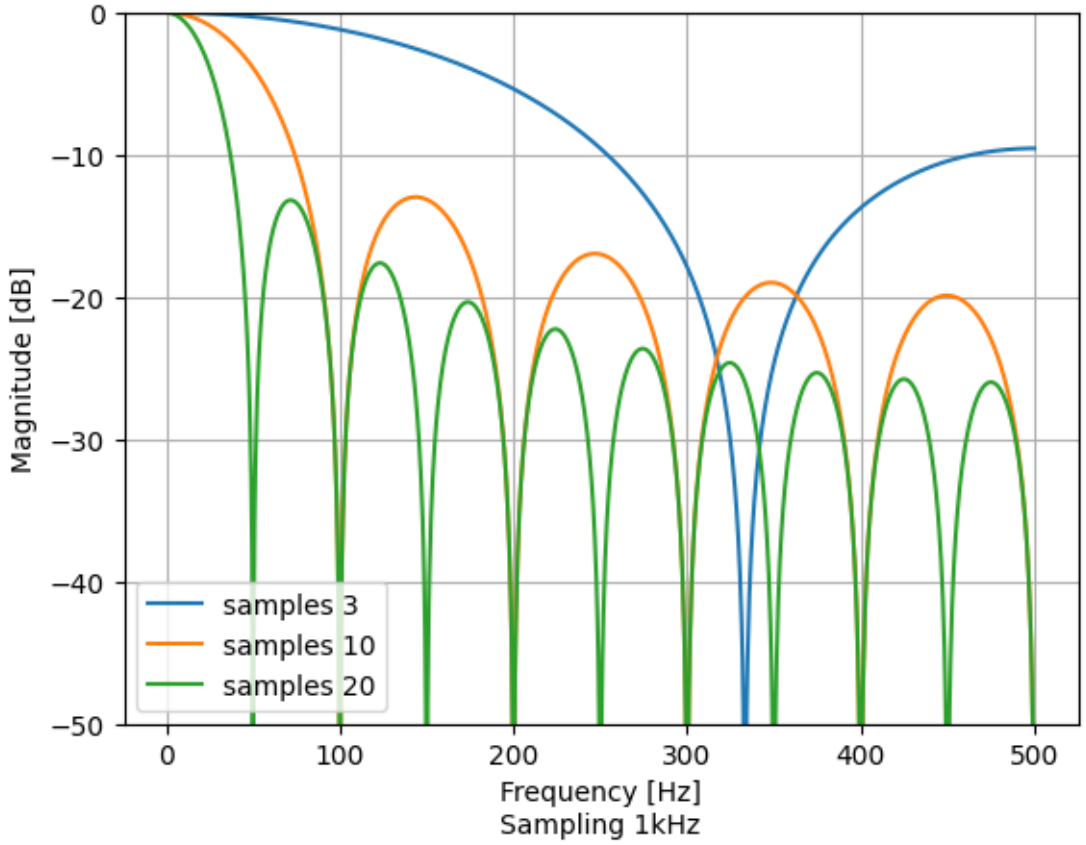


[ ]:
```
# Moving avergae filters
fs = 1000  # Hz
w = np.linspace(0, np.pi, 1024)

fig, ax = plt.subplots()
```

```
for samples in [3, 10, 20]:
    f = w * fs / (2*np.pi)
    H = (1/samples)*(1-np.exp(-1j*w*samples))/(1-np.exp(-1j*w))
    ax.plot(f, 20 * np.log10(abs(H)), label="samples " + str(samples))

plt.ylabel('Magnitude [dB]')
plt.xlabel('Frequency [Hz]\nSampling 1kHz')
plt.grid(True)
plt.legend()
plt.ylim(-50, 0)
plt.show()
```



[ ]:
```
# simple binomial filter
fs = 1000  # Hz
w = np.linspace(0, np.pi, 1024)
fig, ax = plt.subplots()
# numerator co-effs of filter transfer function
filt = np.array([0.25, 0.5, 0.25])
a = np.ones(1)  # denominator co-effs of filter transfer function
```

```
w, h = signal.freqz(filt, a)
f = w * fs / (2*np.pi)
ax.plot(f, 20 * np.log10(abs(h)), label="" + str(samples))


plt.ylabel('Magnitude [dB]')
plt.xlabel('Frequency [Hz]\nSampling 1kHz')
plt.grid(True)
plt.ylim(-50, 0)
plt.legend()
plt.show()
```