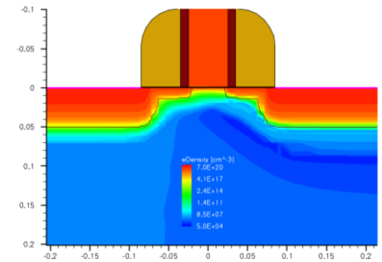
Assignment – Simulating an amplifier

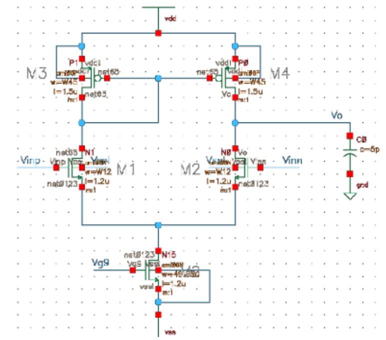
- We will simulate an amplifier at *behavioral level*
- Simulations in Matlab
  - You can run everything on your laptop
  - Basic explanations during rest of the class
  - Example code in the following slides
- Assignment
  - Discussed at the end of the class
  - You will have to deliver a Matlab model/simulation

Simulation – Abstraction levels

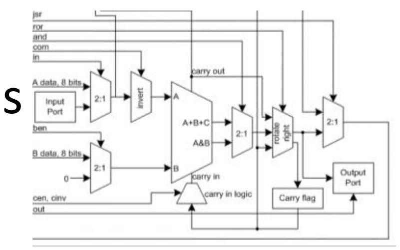
- Device simulation
  - Electrical, thermal and quantum effects
  - TCAD: Sentaurus, Medici, Atlas,…



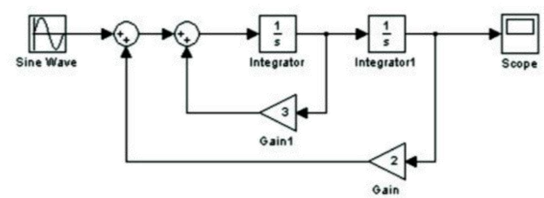
- Circuit simulation
  - Based on (non-linear) device models
  - Solve Kirchhoff’s laws
  - Simulators: Spice, Eldo, Spectre, …

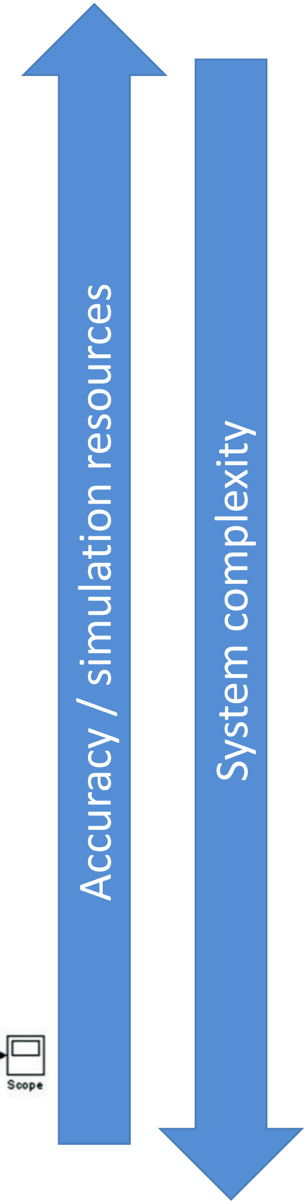


- Behavioral simulation
  - Use abstractions,not only physical quantities
  - HDL: VHDL, Verilog, VerilogA, …



- System simulation
  - Assume models for sub-blocks
  - Tools: **Matlab**, Simulink…





Matlab Simulation

Very simple scripting language:

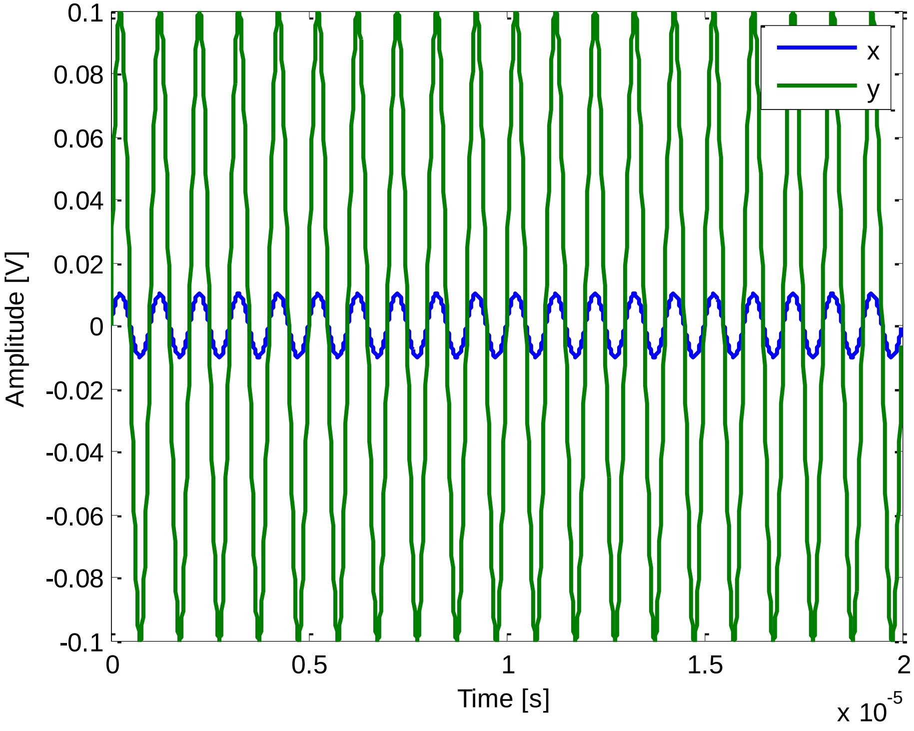
```
N=100;
t=0:N;          % t = 0,1,2, N

x=sin(2*pi*t);

plot(t,x,'k-');          % plot "x" with black lines

y=10*x;

hold on;          % allows the current plot to be overwritten
plot(t,x,'r-');
plot(t,y,'b-');
hold off;          % good habit!


plot(t,x,'k-',t,x,'r-',t,y,'b-'); % alternative
```

NOTE: For help on the plot function type: **help plot**

Time-domain simulation

On Brightspace:
amplifier.m

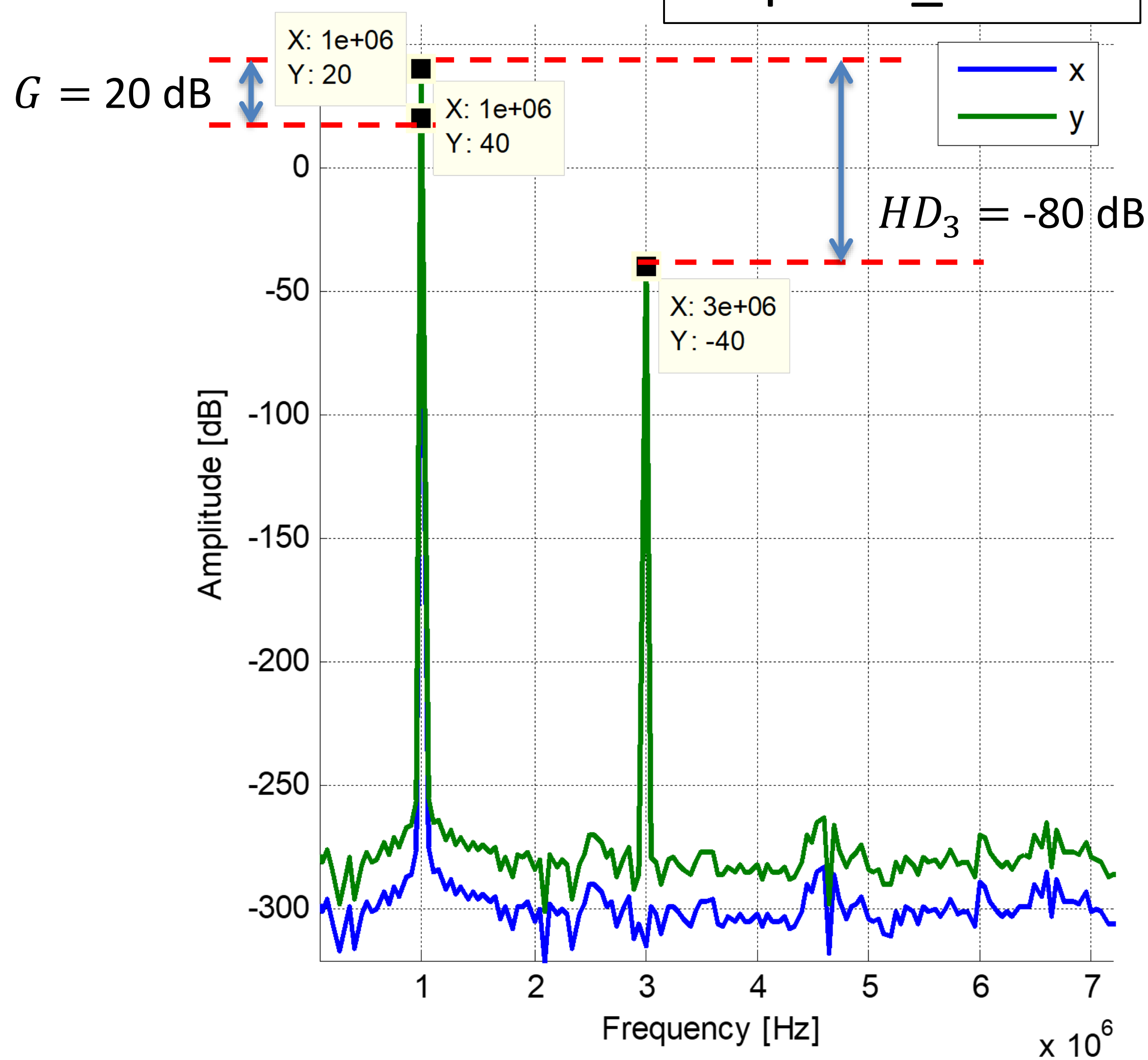
```
% define input signal
A=10e-3; %[V]
% input frequency
fin=1e6; %[Hz]

T=20/fin; %[s]
dt=1/fin/100; %[s]
t=0:dt:T-dt; %[s]

% input signal
x=A*sin(2*pi*fin*t); %[V]

% compute output signal
a1=10; %[-]
y=a1*x; %[V]

%plot
plot(t,x,t,y)
```



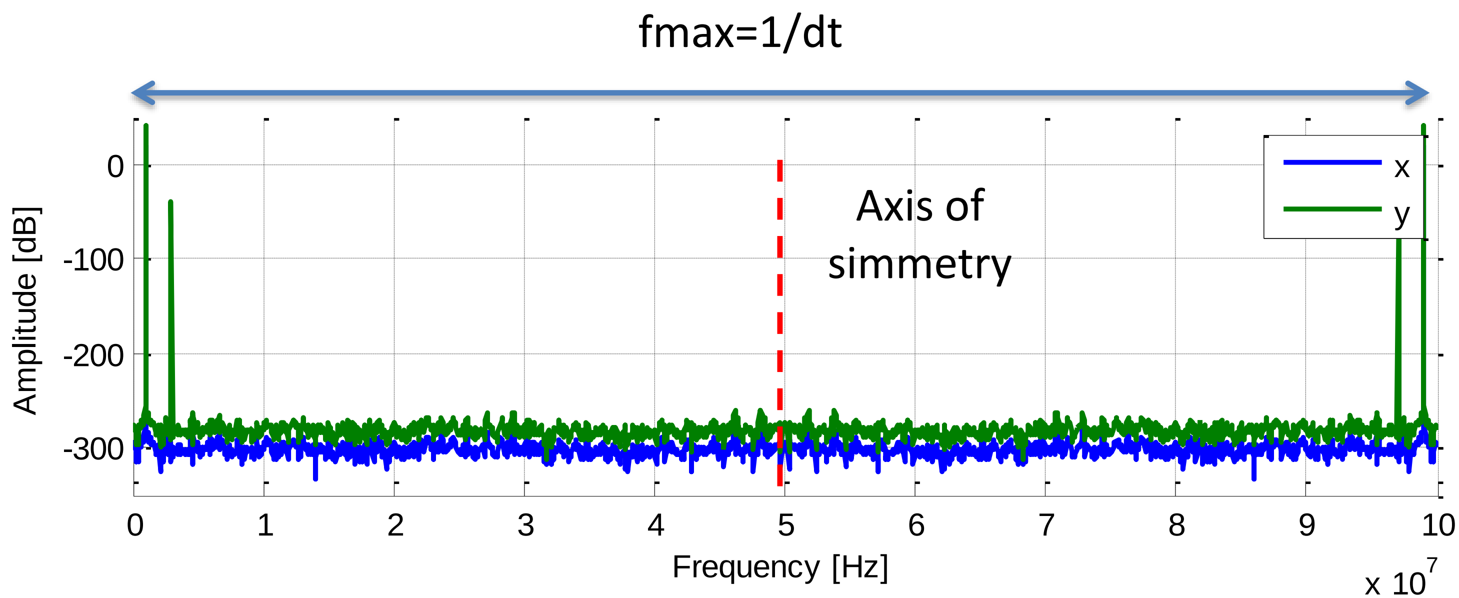
Frequency domain

On Blackboard:
amplifier_FFT.m

```
a1=10;  %[-]
a3=-40; %[V^-2]
y=a1*x+a3*x.^3; %[V]

%compute fft
X=abs(fft(x));
XdB=20*log10(X); %[dB]

Y=abs(fft(y));
YdB=20*log10(Y); %[dB]

% FFT resolution
fres=1/T; %[Hz]
% maximum FFT frequency
fmax=1/dt; %[Hz]
% frequency vector
f=0:fres:(fmax-fres); %[Hz]

%plot FFT
plot(f,XdB,f,YdB)
```



FFT details

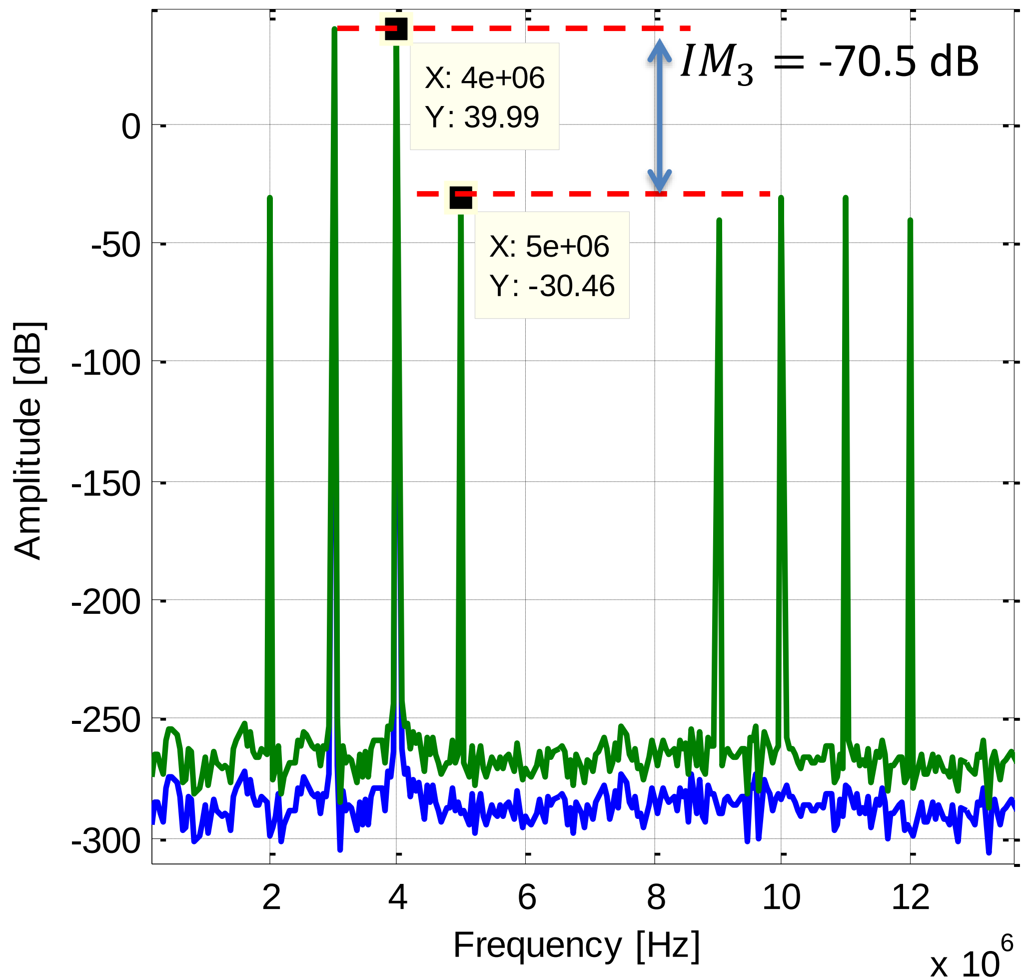


- Time step = Δt, total simulation time = T
- Max observable frequency = $\frac{1}{2\Delta t}$
- Frequency resolution = $\frac{1}{T}$
- Make sure that carrier frequency falls on a bin: $f_{in} = n\frac{1}{T}$

Two-tone test

On Brightspace:
amplifier_2_tones.m

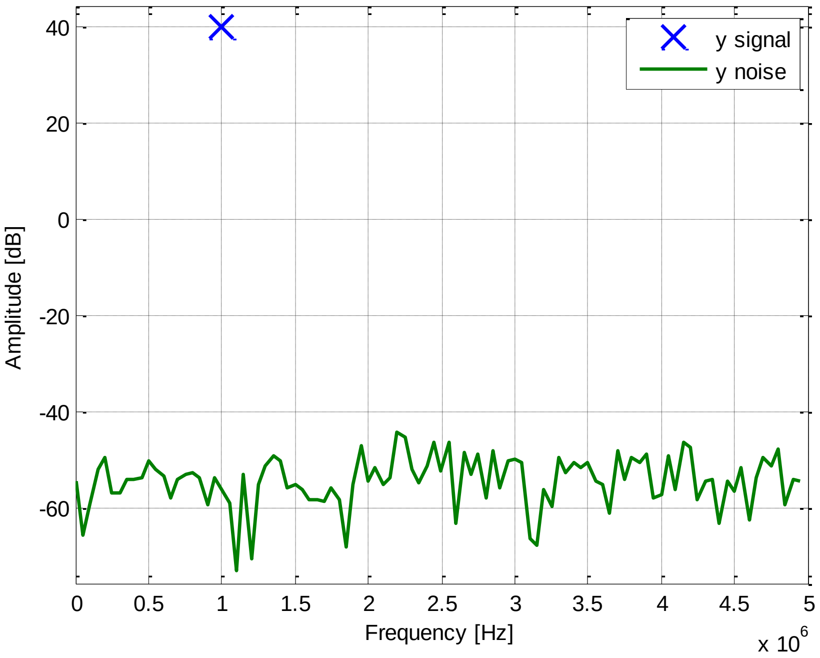
```
fin1=3e6; %[Hz]
fin2=4e6; %[Hz]

% input signal
x1=A*sin(2*pi*fin1*t); %[V]
x2=A*sin(2*pi*fin2*t); %[V]
x=x1+x2; %[V]

% compute output signal
a1=10; %[-]
a3=-40;%[V^-2]
y=a1*x+a3*x.^3; %[V]
```



Noise

On Brightspace: amplifier_SNR.m

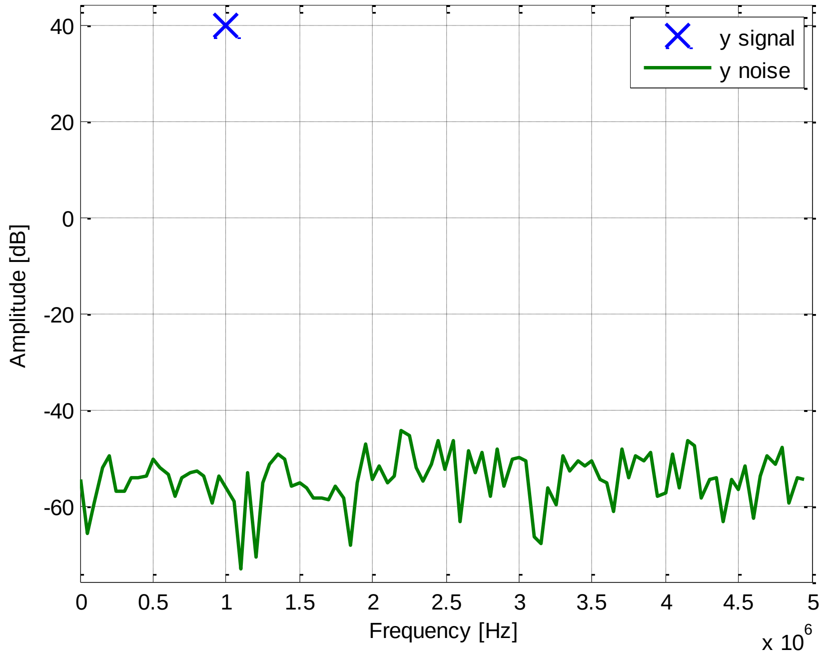
```
Te=300; %[K]
R0=50; %[ohm]
k=1.38e-23; %[m^2kgs^-2K^-1]
Sn=4*k*Te*R0; %[V^2/Hz]
noise_rms=sqrt(Sn/dt/2); %[V]
noise=noise_rms*randn(size(t));

% compute output signal
x_noise=x+noise;
y=a1*x_noise+a3*x_noise.^3; %[V]

signal_bins = round(fin/fres)+1;
BW=5e6; %[Hz]
inband_bins=1:round(BW/fres);
noise_bins=setdiff(inband_bins,signal_bins);

plot(f(signal_bins),YdB(signal_bins),'X',...
    f(noise_bins),YdB(noise_bins)
```



SNR

On Brightspace:
amplifier_SNR.m

```
signal_bins = round(fin/fres)+1;
BW=5e6; %[Hz]
inband_bins=1:round(BW/fres);
noise_bins=setdiff(inband_bins,signal_bins);

%signal power
S=sum(Y(signal_bins).^2);

%noise power
N=sum(Y(noise_bins).^2);

%SNR
SNR=10*log10(S/N);
```

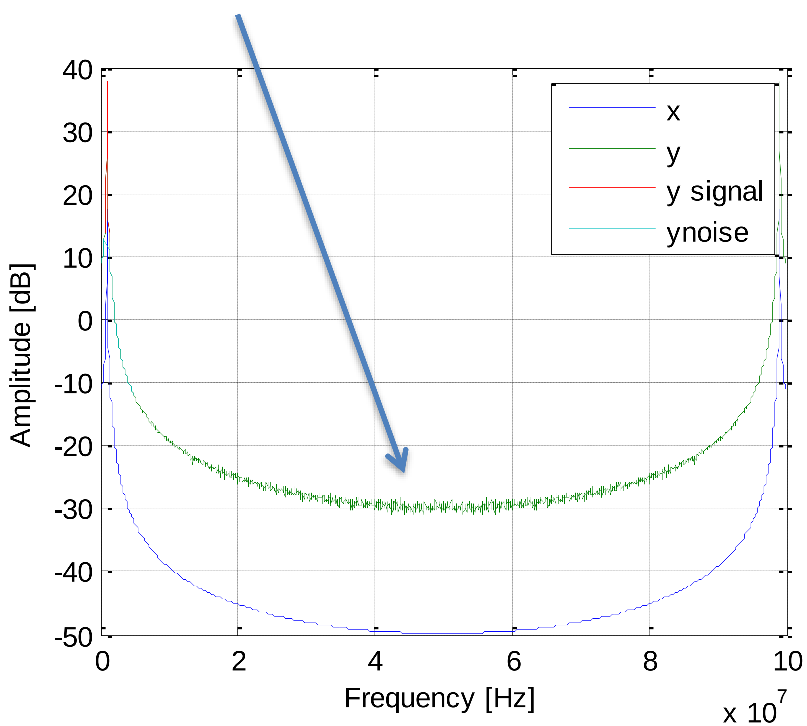


Windowing

On Brightspace:
amplifier_window.m

```
T=20/fin; %[s]
dt=1/fin/100; %[s]
t=0:dt:T-dt; %[s]

% input signal
x=A*sin(2*pi*1.02e6*t); %[V]

Y=abs(fft(y));
YdB=20*log10(Y); %[dB]

% FFT resolution
fres=1/T; %[Hz]
% maximum FFT frequency
fmax=1/dt; %[Hz]
% frequency vector
f=0:fres:(fmax-fres); %[Hz]
```

Spectral leakage



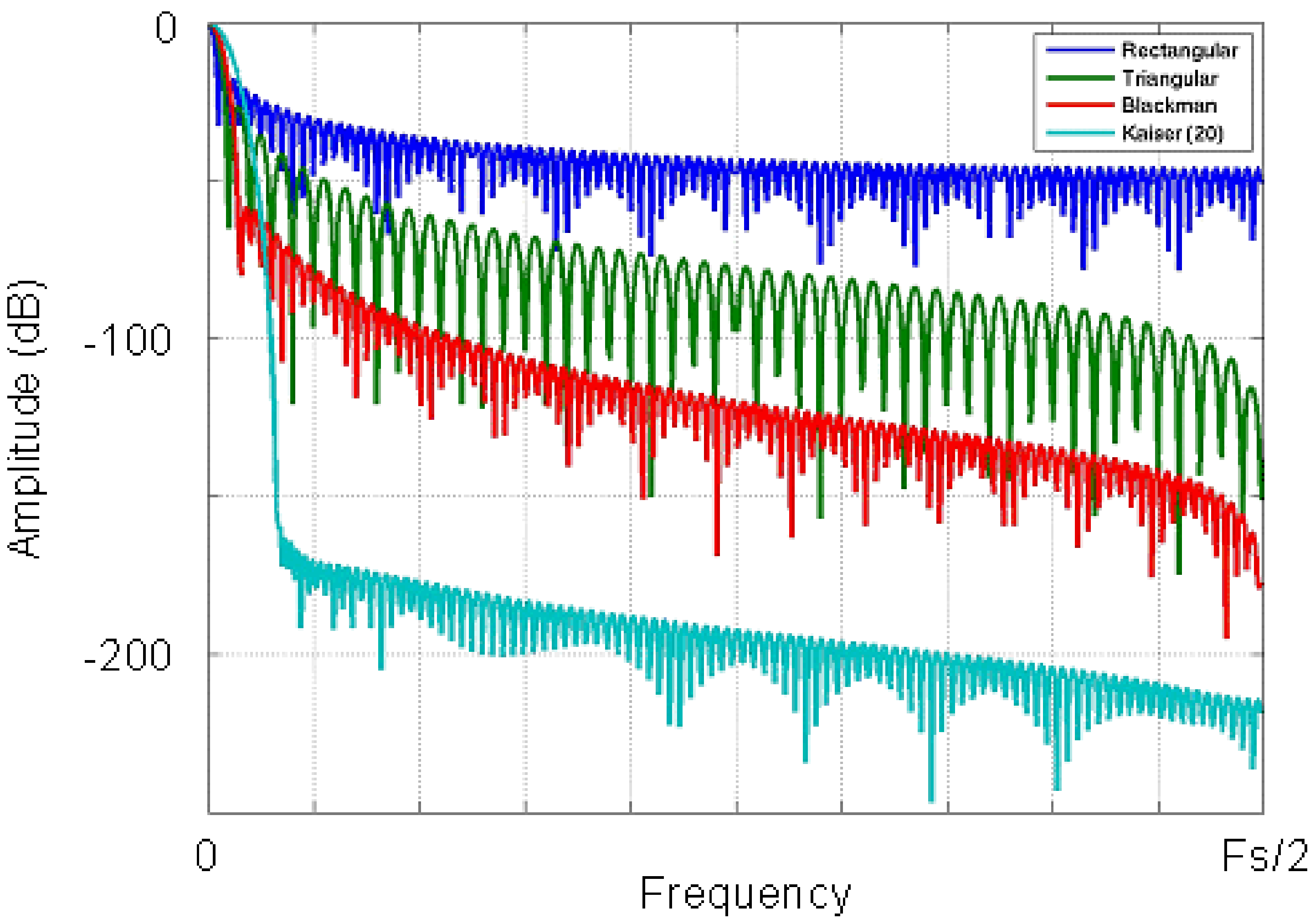
Windowing

- If f_{in} does not fall exactly into a FFT bin ⇒ some of its energy will "leak" into neighboring bins

- Minimizing leakage ⇒ multiply input by a window function
- Note: doing nothing ⇒ rectangular window ⇒ sinc(f) response

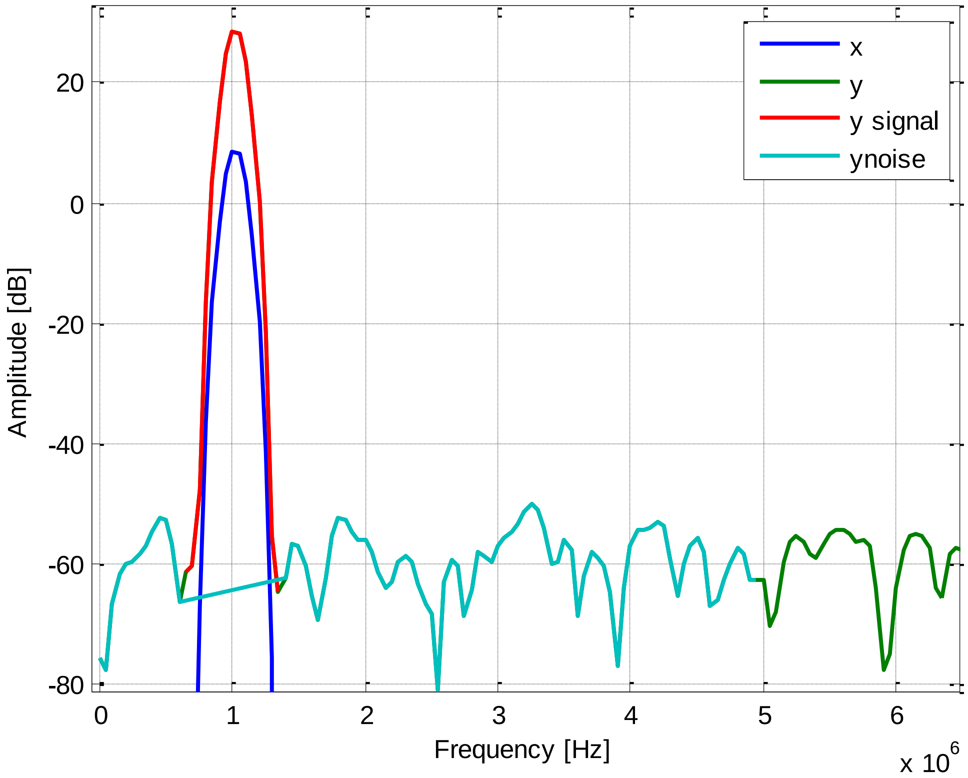
- FFT = convolution of the window frequency response with the actual input spectrum
- Ideal window function should have a narrow bandwidth and suppress high frequencies

Spectral response of common windows



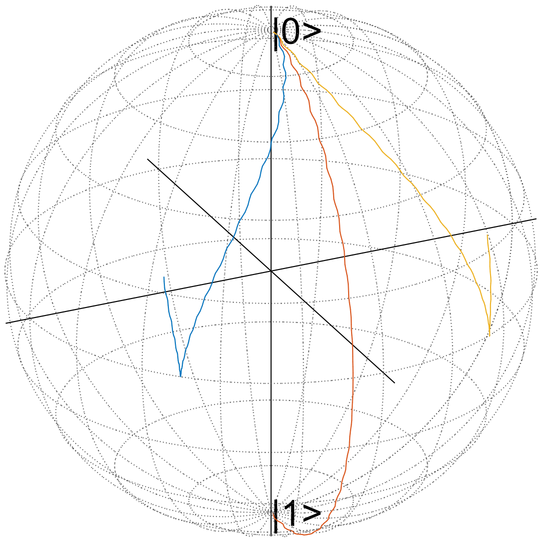
Windowing

```
%compute fft
w=kaiser(length(t),20)';
Y=abs(fft(y.*w));
YdB=20*log10(Y); %[dB]

%compute SNR
% signal bins
fres=1/T; %[Hz]
maintone = round(fin/fres)+1;
signal_bins = [maintone-7:maintone+7];
BW=5e6; %[Hz]
inband_bins=1:round(BW/fres);
noise_bins=setdiff(inband_bins,signal_bins);

%SNR
S=sum(Y(signal_bins).^2);
N=sum(Y(noise_bins).^2);
SNR=10*log10(S/N);
```



SPIN Emulator (SPINE)

```
% Set the Larmor frequencies of the qubits (3 qubits)
f0 = [0.99e9, 1e9, 1.01e9];

% Set the Rabi frequencies
fR = [10e6, 10e6, 10e6];

% Generate the signal, driving the 2nd qubit (pi-rotation)
dt = 0.4e-10;
tpi = 0.5 / fR(2);
Npi = tpi / dt;
t = (1:Npi)*dt;
signal = cos(2*pi*f0(2)*t);

% Simulate and plot
[U, probabilities] = spine(fR, f0, dt, signal, 10);
```



See live demonstration in Matlab

SPINE

```
% Print the probability of ending up in the spin up state
disp('Spin up probability for the 3 qubits:');
disp(squeeze(probabilities(3, end, :)));

% Print the fidelity of an identity operation/X-rotation
F = zeros(3, 1);
for n=1:3
   if (n == 2)
      % X-rotation
      Uideal = [0 1;
                1 0];
      F(n) = fidelity(U(:, :, n), Uideal);
   else
      % Identity
      Uideal = eye(2);
      F(n) = fidelity(U(:, :, n), Uideal);
   end
end
disp('Infidelity (I/X/I-gates):');
disp(1-F);
```

Reference material here (also on Brightspace):
- arXiv:1803.06176
- J. van Dijk et al., DATE 2018

Note on SPINE

- Signal to be applied
  - If applying a microwave pulse at the qubit Larmor frequency and with constant amplitude, an amplitude of 1 V generate a Rabi oscillation at frequency fR
- We will focus on simple operation that is π-rotation around X or Y starting from state |0>.
- For driven qubits, we will use the **fidelity.m** function to determine the operation fidelity.
- For fidelity on idle qubits:
  - Typically, a spurious Z-rotation appears due to leakage from driving other qubit.
  - A Z-rotation can be easily compensated via software but this is not currently implemented.
  - Since the qubits are all initialized in |0>, the fidelity after the Z-rotation compensation can be well approximated as the probability of measuring along the Z-axis **only for ideal qubits.**