Lab4_NF_Preliminary_Group1

June 5, 2025

1 Preparation for Lab 4 from Group 1 (Wörner, Velez, Northe)

The source code and the binaries from this report can be found in our Github repository.

1.0.1 Import libs

[74]:
```
import numpy as np
from scipy.constants import Boltzmann
```

1.0.2 Helper constants and functions

[75]:
```
k_B = Boltzmann     # Boltzman constant

db2lin = lambda db: 10**(db / 10)

lin2db = lambda g: 10 * np.log10(g)

lin2dBm = lambda g: lin2db(g/1e-3)

def scientific_2_str(value: float, unit: str = "Ω") -> str:
    prefixes = [
        (1e9, 'G'),
        (1e6, 'M'),
        (1e3, 'k'),
        (1, ''),
        (1e-3, 'm'),
        (1e-6, 'µ'),
        (1e-9, 'n'),
        (1e-12, 'p')
    ]

    for factor, prefix in prefixes:
        if abs(value) >= factor:
            formatted = value / factor
            return f"{formatted:.3g} {prefix}{unit}"
    return f"{value:.3g} {unit}"  # fallback for very small values
```

1.1 3.1 Exercise 1

Given values are

[76]:
```
P_s_ref = 23    # [dBm] desired signal level
P_n     = 6     # [dbm] noise level
Att     = 15    # [dB] attenuation

G_amp   = 18    # [dB] Gain amplifier
NF_amp  = 3     # [db] Noise Figure amplifier

T_0     = 290   # [K] Room Temperature
```

[77]:
```
P_s_lin = db2lin(P_s_ref)
P_n_lin = db2lin(P_n)
Att_lin = db2lin(Att)

G_amp_lin = db2lin(G_amp)
NF_amp_lin = db2lin(NF_amp)
```

Amp before line

[78]:
```
SNR_i = P_s_ref - P_n
SNR_i_lin = db2lin(SNR_i)

F = NF_amp_lin + (Att_lin - 1) / G_amp_lin

SNR_o_lin = SNR_i_lin / F
SNR_o = lin2db(SNR_o_lin)

print(f"SNR at input: {SNR_i:.2f} dB")
print(f"SNR after LNA->Line output: {SNR_o:.2f} dB")
```

```
SNR at input: 17.00 dB
SNR after LNA->Line output: 13.05 dB
```

Amp after line

[79]:
```
SNR_i = P_s_ref - P_n
SNR_i_lin = db2lin(SNR_i)

T = T_0
F_line = 1 + (Att_lin - 1) * T / T_0

F = F_line + (NF_amp_lin - 1) / (1 / Att_lin)
NF = lin2db(F)

SNR_o_lin = SNR_i_lin / F
SNR_o = lin2db(SNR_o_lin)
```

```python
print(f"SNR at input: {SNR_i:.2f} dB")
print(f"SNR after Line->LNA output: {SNR_o:.2f} dB")
print(f"NF of Line {NF} dB")
```

```
SNR at input: 17.00 dB
SNR after Line->LNA output: -1.00 dB
NF of Line 18.0 dB
```

Justification: As we can see is the G_{LNA} in the second case unused, therefore we are basically just subtracting the NF_{Line} from our Signal.

1.2 3.2 Exercise 2

[80]:
```python
T_A          = 150    # [K] equivalent temp of antenna
BW_IF        = 10e6   # [Hz] BW of IF signal

# LNA
G_amp        = 10     # [dB]
F_a          = 2      # [dB]

# Bandpass Filter
L_bpf        = 1      # [dB]

# Mixer values
L_mix        = 3      # [dB]
F_mix        = 4      # [dB]

SNR_min      = 20      # [dB]
Z_0          = 50      # [Ohm]
```

[81]:
```python
G_amp_lin = db2lin(G_amp)
F_a_lin = db2lin(F_a)
L_bpf_lin = db2lin(L_bpf)
L_mix_lin = db2lin(L_mix)
F_mix_lin = db2lin(F_mix)
```

[82]:
```python
N_i = k_B * T_A * BW_IF
N_dbm = lin2dBm(N_i)
print(f"Noise power Antenna: {scientific_2_str(N_dbm, "dBm")}")

F_tot = F_a_lin + (L_bpf_lin - 1) / G_amp_lin + (F_mix_lin - 1) / (G_amp_lin *␣
 ↪1 / L_bpf_lin)
NF = lin2db(F_tot)

G_tot = G_amp_lin * 1 / L_bpf * 1 / L_mix
N_O =  N_i * F_tot
```

```
N_0_db = lin2dBm(N_0)
print(f"Output Noise Power: {scientific_2_str(N_0_db, "dBm")}")
```

```
Noise power Antenna: -107 dBm
Output Noise Power: -104 dBm
```

[83]:
```
SNR_min_lin = db2lin(SNR_min)

P_signal = SNR_min_lin * N_0
V = np.sqrt(P_signal * Z_0)

print(f"Min. Voltage required for {SNR_min} is {scientific_2_str(V, "V")}")
```

```
Min. Voltage required for 20 is 13.7 µV
```

1.3 3.3 Exercise 3

Insert solution here

[84]:
```
G_DUT       = 14        # [dB]

G_lna       = 40        # [dB]
F_lna       = 2         # [dB]

RBW         = 10        # [Hz]
P_1GHz      = -145      # [dBm]
N_1         = -82.92    # [dBm]
N_2         = -73.42    # [dBm]

T_0         = 290       # [K]

ENR         = 13.03     # [dB] @ 1GHz
f_meas      = 1e9       # [Hz] Frequency of meassurement
```

[85]:
```
# - in log is equivalent to / in lin
Y = N_2 - N_1
print(f"Y-Factor: {Y:.2f} dB")

Y_lin = db2lin(Y)
ENR_lin = db2lin(ENR)

N_a_lin = k_B * T_0 * RBW * G_lna * ((ENR_lin - 1) / (Y_lin - 1) - 1)
N_a = lin2dBm(N_a_lin)
print(f"Internal noise power of DUT @{scientific_2_str(f_meas, "Hz")}: {N_a:.
  ↪2f} dBm")
```

```
Y-Factor: 9.50 dB
Internal noise power of DUT @1 GHz: -146.45 dBm
```

[ ]:
```python
F_lna_lin = db2lin(F_lna)
G_DUT_lin = db2lin(G_DUT)

F_sys_lin = ENR_lin / (Y_lin - 1)
F_sys = lin2db(F_sys_lin)
print(f"Total  F of DUT + Meassurement-Equipment: {F_sys:.2f} dB")

F_DUT = F_sys_lin - (F_lna_lin - 1) / G_DUT_lin
NF_DUT = lin2db(F_DUT)
print(f"NF of DUT {NF_DUT:.2f} dB")

# G_1 = (N_2 - N_1) / (9)
```

```
Total  F of DUT + Meassurement-Equipment: 4.05 dB
F of DUT 4.01 dB
```

1.4 3.4 Exercise 4

Name four important points when building a noise source

- Amplification of noise or more specifiv Resulting output noise power
- Bandwidth/Wideband characteristic -> uniformity across the specified frequency bandwidth
- Matching characteristics across bandwidth for optimal power transfer
- Thermal stability -> stability of the noise and behaviour across a temperature band

1.- Component Selection: Choose components like noise diodes or zener diodes that generate broadband white noise to ensure uniform behavior across the desired frequency range.

2.- Thermal Stability: Use thermally stable components or incorporate temperature compensation to prevent output noise level changes due to temperature variations.

3.- Impedance Matching: Match the output impedance (typically $50\,\Omega$) between components of the noise source to minimize signal reflections and maximize power transfer.

4.- Shielding and Grounding: Proper shielding and grounding to prevent electromagnetic interference, ensuring a clean and stable noise signal output.