AI-assisted Design and Synthesis of Continuous-Time ADCs

Internal Report#5
AIR-CHIP Project (PID2022-138078OB-I00)

Author
Francisco Gómez Pulido

Distribution List
Gustavo Liñán-Cembrano
Jose M. de la Rosa

Date: March 4, 2025

Contents

1 Introduction

Date: March 04, 2025

The goal of this study is to analyze how different design parameters affect the Signal-to-Noise Ratio (SNR) of a $\Sigma\Delta$M.

This report presents a brute-force search approach to optimize modulator design and introduces a **visualization framework** to interpret the results. By automating simulations and employing interactive tools, we aim to gain insights into the influence of parameters such as modulator order, oversampling ratio (OSR), number of DAC levels, and feedback gain (Hinf).

The primary **objectives** of this project are:

- To explore how different modulator parameters impact the SNR.
- To store and analyze simulation results systematically.
- To visualize the data using static and interactive plots.
- To develop an interactive interface for better exploration of results.

To systematically analyze the effect of design parameters, a **brute-force search** was implemented, of course, knowing it's not efficient, as proved previously. The results from the simulations were structured into a pandas **DataFrame** and stored as a **CSV** file. This format allows for easy analysis and visualization.

To better understand the trends in the simulation results, various **plots** were generated:

- Box plots to show the impact of modulator order on SNR.
- Line plots illustrating the effect of OSR and feedback gain.
- Scatter plots to examine correlations between parameters.

To enhance the user experience, a **Streamlit-based web application** was developed. This allows users to:

- Select a modulator order.
- Filter and explore specific configurations.
- Visualize parameter dependencies dynamically.

A new notebook will be run in this section, called `visualization.ipynb`. The notebook is highly based on the previous ones, so I strongly recommend looking at the previous work first.

2 Analysis and interactive visualization of ΣΔM design parameters

Date: March 03, 2025

2.1 Brute-force search and storing results

In this section, we perform a brute-force search over several design parameters to optimize the performance of a ΣΔM. The key goal is to evaluate various configurations and store the results for later analysis.

```
import numpy as np
import pandas as pd
import cbadc as cb
import matplotlib.pyplot as plt
import seaborn as sns
from itertools import product
from numpy.linalg import LinAlgError
```

```
# target specifications:
target_SNR = 90
Bw = 20e3
```

```
# parameter search space:
orders = [2, 3, 4, 5] # modulator order
osr_values = [32, 64, 128, 256] # oversampling ratio
nlev_values = [2, 3] # number of DAC levels
Hinf_values = [1.0, 1.5, 2.0] # feedback gain
forms = ["CRFB", "CRFF", "CIFB", "CIFF", "CRFBD", "CRFFD"] # modulator architectures
tdac = [0, 1] # DAC sampling times
```

We create an empty list to store the results of each configuration tested during the brute-force search.

```
# store resutls:
results = []
```

```
# brute-force search:
for order, osr, nlev, Hinf, form in product(orders, osr_values, nlev_values, Hinf_values, forms):
    # compute the sampling frequency:
    fs = Bw * osr * 2
    print(f"Sampling frequency: {fs / 1e6} MHz")

    print("####################################")
    print(f"Testing: order={order}, osr={osr}, nlev={nlev}, Hinf={Hinf}, form={form}")
    print("####################################")

    # synthesize the Noise Transfer Function (NTF):
    ntf = cb.delsig.synthesizeNTF(order, osr, 2, Hinf, 0.0)

    # realize the NTF in state-space representation:
    a, g, b, c = cb.delsig.realizeNTF(ntf, form)
    ABCD = cb.delsig.stuffABCD(a, g, b, c)


    # create discrete-time analog frontend model:
```

```
    dt_analog_frontend = cb.AnalogFrontend.dtsdm(ABCD, nlev)

    try:
        # run an SNR simulation:
        snr, amp, _ = dt_analog_frontend.simulateSNR(osr)
        max_snr = max(snr)

        # store result:
        results.append({"Order": order, "OSR": osr, "Levels": nlev, "Hinf": Hinf, "Form": form,
            "SNR": max_snr})

    except (LinAlgError, ValueError) as e:
        print(f"Error encountered with order={order}, osr={osr}, nlev={nlev}, Hinf={Hinf},
            form={form}. Skipping this configuration")
```

```
INFO:root:Simulating discrete-time analog frontend
Sampling frequency: 2.56 MHz
###################################
Testing: order=2, osr=64, nlev=2, Hinf=1.0, form=CRFB
###################################
WARNING:cbadc.digital_backend:Discrete time Wiener filter not properly implemented. Results
    may be incorrect.
INFO:root:Simulating discrete-time analog frontend
Sampling frequency: 2.56 MHz
###################################
Testing: order=2, osr=64, nlev=2, Hinf=1.0, form=CRFF
###################################
WARNING:cbadc.digital_backend:Discrete time Wiener filter not properly implemented. Results
    may be incorrect.
...
```

In this part of the code, we perform the brute-force search by iterating over all combinations of the parameters defined in the search space. If an exception occurs during the SNR simulation, it is caught by the try-except block, and a message is printed. The algorithm then skips the problematic configuration and continues with the next one.

Finally, after all configurations have been tested, we convert the results into a DataFrame using pandas, which makes it easier to analyze and manipulate the data. We then save the results to a CSV file for later use or further analysis.

```
# convert to DataFrame:
df_results = pd.DataFrame(results)

# convert to CSV:
df_results.to_csv("sigma_delta_results.csv", index=False)
```

2.2 Visualizing desing parameters impact on SNR

In this section we obtain some plots of varying design parameters, in order to see how affect the SNR.

```
df = pd.read_csv("sigma_delta_results.csv")
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(x="Order", y="SNR", data=df, palette="viridis")
plt.title("Impact of the modulator order on SNR", fontsize=14)
plt.xlabel("Modulator order", fontsize=12)
```

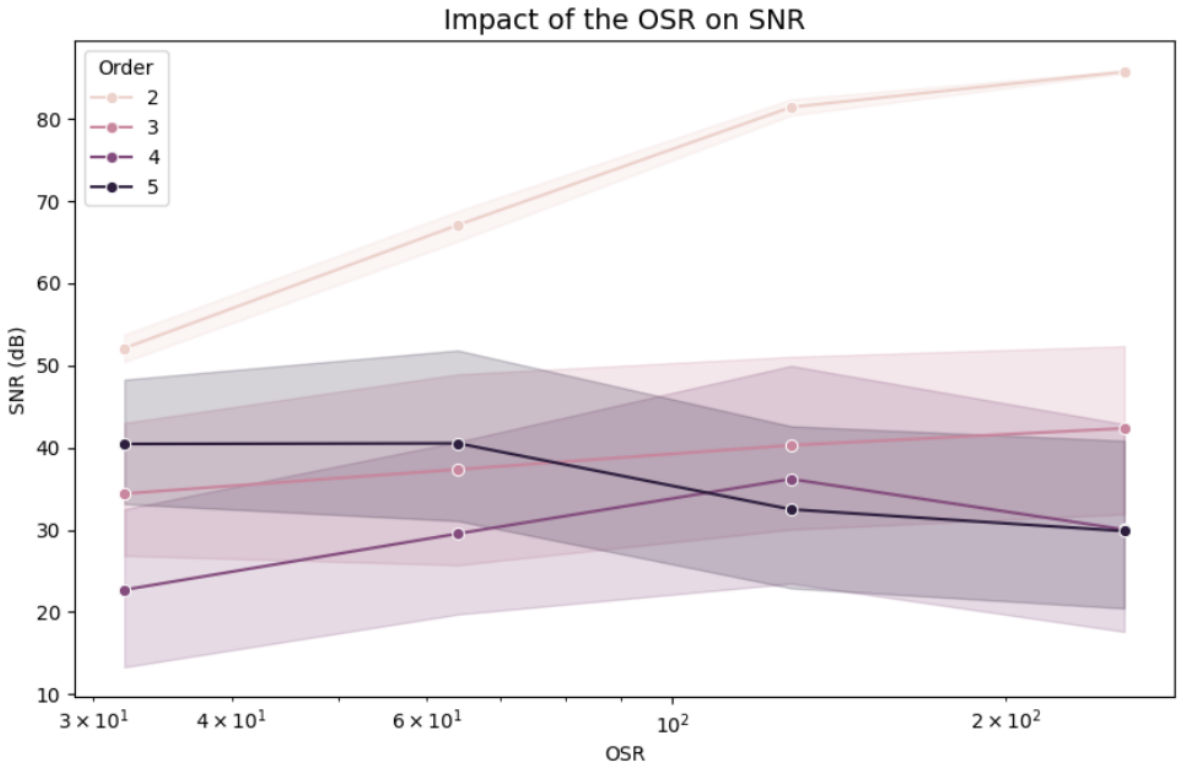
```
plt.ylabel("SNR (dB)", fontsize=12)
plt.grid(True)
plt.show()
```

This first visualization is a **boxplot** that illustrates how the modulator order influences the SNR. Different modulator orders are represented on the x-axis, while the corresponding SNR values (in dB) are plotted on the y-axis.

- Second-order ΣΔM: exhibits high SNR values with a relatively small variance. The median SNR is above 80 dB, suggesting it provides consistent and robust performance.
- Third-order ΣΔM: shows a huge variance in SNR values, ranging from close to 0 dB to above 80 dB. The median SNR is significantly lower than that of the second-order modulator, indicating that while it can achieve high SNR in some cases, it is also prone to instability or poor performance in others.
- Fourth-order ΣΔM: the SNR distribution is more concentrated, with values mostly ranging between 10 and 40 dB. The median is significantly lower than that of the second-order modulator, suggesting that increasing the order does not necessarily improve SNR.
- Fifth-order ΣΔM: exhibits the lowest SNR values on average and has a very narrow distribution. This suggests that increasing the modulator order beyond a certain point may lead to diminishing returns and potential instability.

This result suggests that simply increasing the modulator order does not guarantee improved SNR and may introduce instability.

```
plt.figure(figsize=(10, 6))
sns.lineplot(data=df, x="OSR", y="SNR", hue="Order", marker="o")
plt.title("Impact of the OSR on SNR", fontsize=14)
plt.xlabel("OSR")
plt.ylabel("SNR (dB)")
plt.legend(title="Order")
plt.xscale("log") # logarithmic scale (better visualization)
plt.show()
```

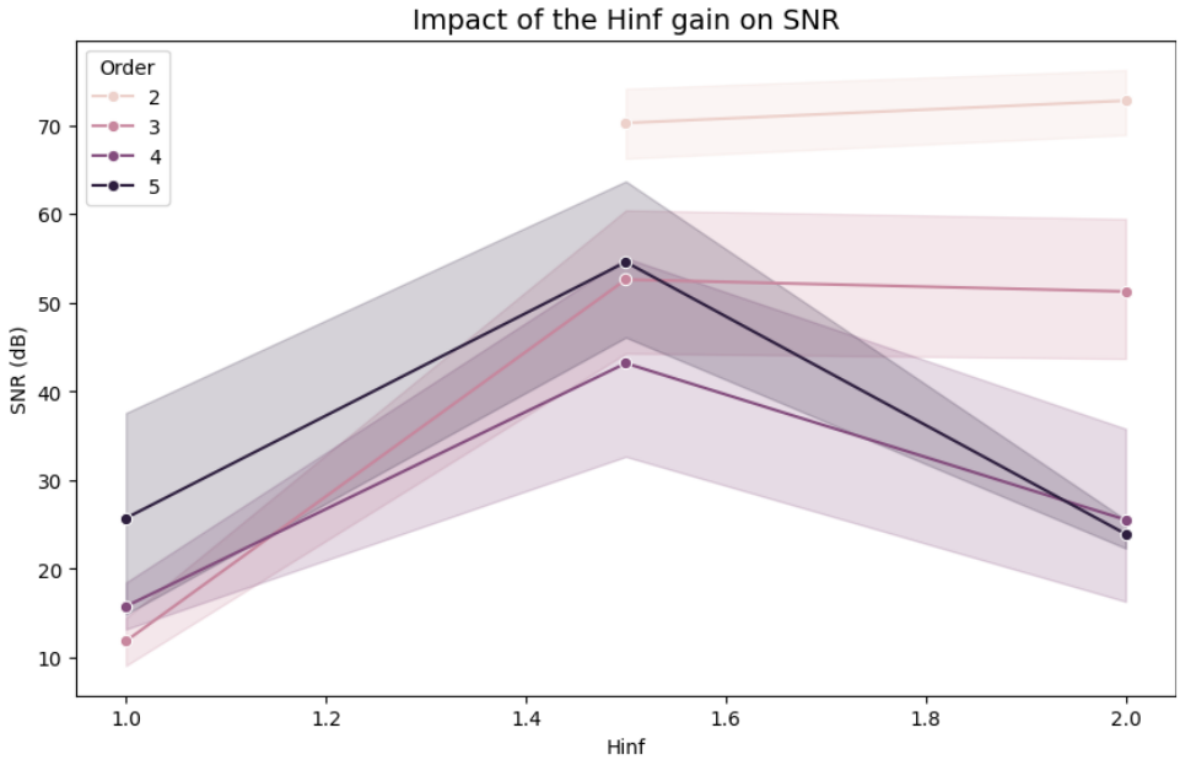


This plot is a line graph displaying the relationship between the OSR and the SNR for different modulator orders. A logarithmic scale is used on the x-axis for better visualization. The different modulator orders are color-coded and include shaded confidence intervals to indicate variability.

- Second-order ΣΔM: shows a clear and consistent increase in SNR as OSR increases. This follows the expected behavior, where higher OSR results in improved noise shaping and better performance.
- Third-order ΣΔM: initially, its SNR is comparable to the fourth-order modulator, but it shows an increasing trend with OSR. However, its performance remains lower than the second-order modulator, reinforcing the idea that increasing the order does not always lead to better SNR.
- Fourth-order ΣΔM: exhibits fluctuations in SNR rather than a clear increasing trend. At higher OSR values, its performance stabilizes but does not show significant improvement.
- Fifth-order ΣΔM: shows decreasing SNR with increasing OSR, which is counterintuitive and suggests instability or implementation issues. This further confirms that high-order modulators can suffer from performance degradation due to design limitations or increased quantization noise.

Higher-order modulators (fourth and fifth order) do not show clear improvements with OSR and may even degrade in performance.

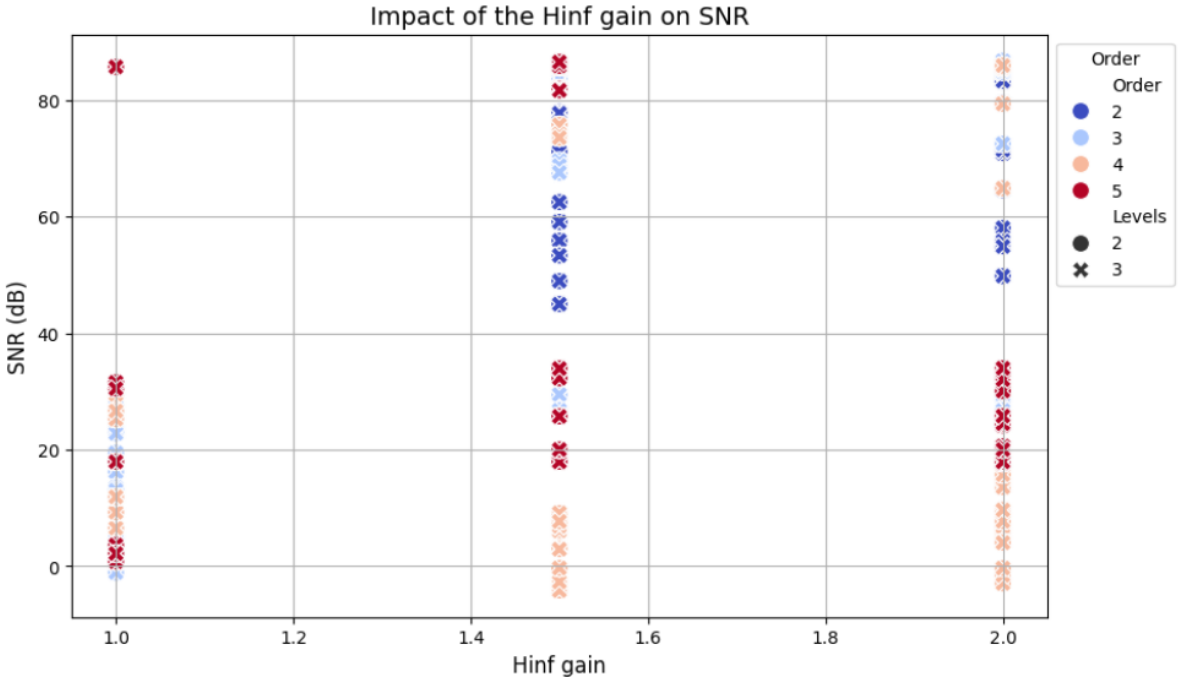
```
plt.figure(figsize=(10, 6))
sns.lineplot(data=df, x="Hinf", y="SNR", hue="Order", marker="o")
plt.title("Impact of the Hinf gain on SNR", fontsize=14)
plt.xlabel("Hinf")
plt.ylabel("SNR (dB)")
plt.legend(title="Order")
plt.show()
```



This line plot shows the relationship between the Hinf gain and the SNR for different modulator orders. The shaded areas represent confidence intervals, indicating variability in the data.

Conclusions: moderate values of Hinf gain improve SNR, especially for lower-order modulators. Excessive Hinf gain leads to performance degradation, particularly for higher-order modulators. Higher-order modulators tend to be more sensitive to Hinf variations, making them harder to optimize.

```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x="Hinf", y="SNR", hue="Order", style="Levels", palette="coolwarm", s=100)
plt.title("Impact of the Hinf gain on SNR", fontsize=14)
plt.xlabel("Hinf gain", fontsize=12)
plt.ylabel("SNR (dB)", fontsize=12)
plt.legend(title="Order", bbox_to_anchor=(1, 1), loc="upper left")
plt.grid(True)
plt.show()
```



This scatter plot further analyzes the effect of Hinf gain on SNR, incorporating modulator levels as an additional variable. The color indicates different modulator orders, while the shape/style indicates whether the modulator has 2 o 3 levels.

Conclusions: Hinf tuning is critical: too low results in poor performance, and too high may introduce instability. Second-order modulators perform consistently well across different Hinf values. Higher-order modulators exhibit more variability, making their behavior harder to predict and optimize.

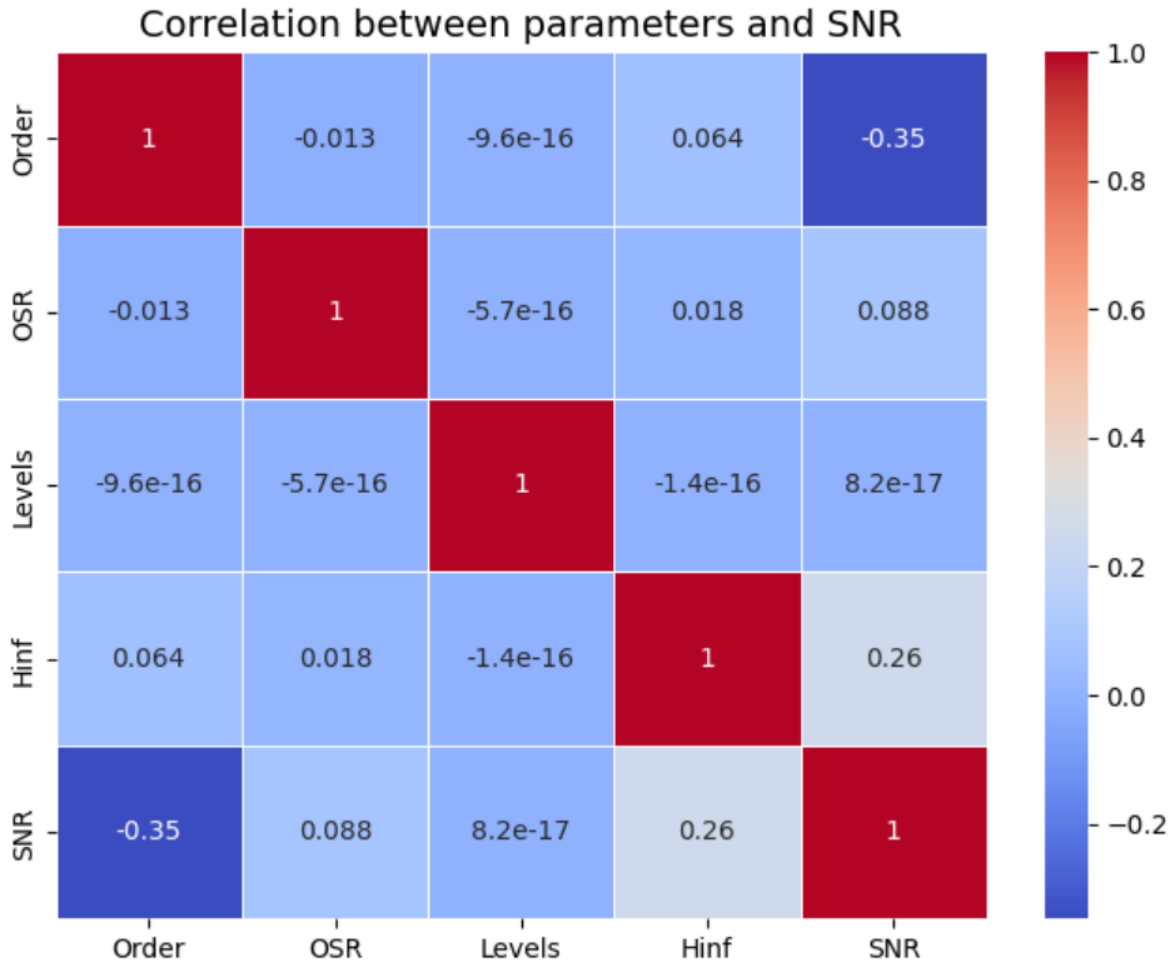
```
# convert cathegoric parameters to number:
df_corr = df.copy()
df_corr["Order"] = df_corr["Order"].astype(float)
df_corr["OSR"] = df_corr["OSR"].astype(float)
df_corr["Levels"] = df_corr["Levels"].astype(float)
df_corr["Hinf"] = df_corr["Hinf"].astype(float)

# correlation matrix:
corr_matrix = df_corr.corr()

# heatmap:
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Correlation between parameters and SNR", fontsize=14)
plt.show()
```

```
ValueError                                Traceback (most recent call last)
Cell In[30], line 9
      6 df_corr["Hinf"] = df_corr["Hinf"].astype(float)
      8 # correlation matrix:
----> 9 corr_matrix = df_corr.corr()
     11 # heatmap:
```

```
     12 plt.figure(figsize=(8, 6))
...
ValueError: could not convert string to float: 'CRFB'
```

```python
# first solution: not including the modulator architecture

# select only number columns:
df_corr_numeric = df.select_dtypes(include=["number"])

# correlation matrix:
corr_matrix = df_corr_numeric.corr()

# heatmap:
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Correlation between parameters and SNR", fontsize=14)
plt.show()
```

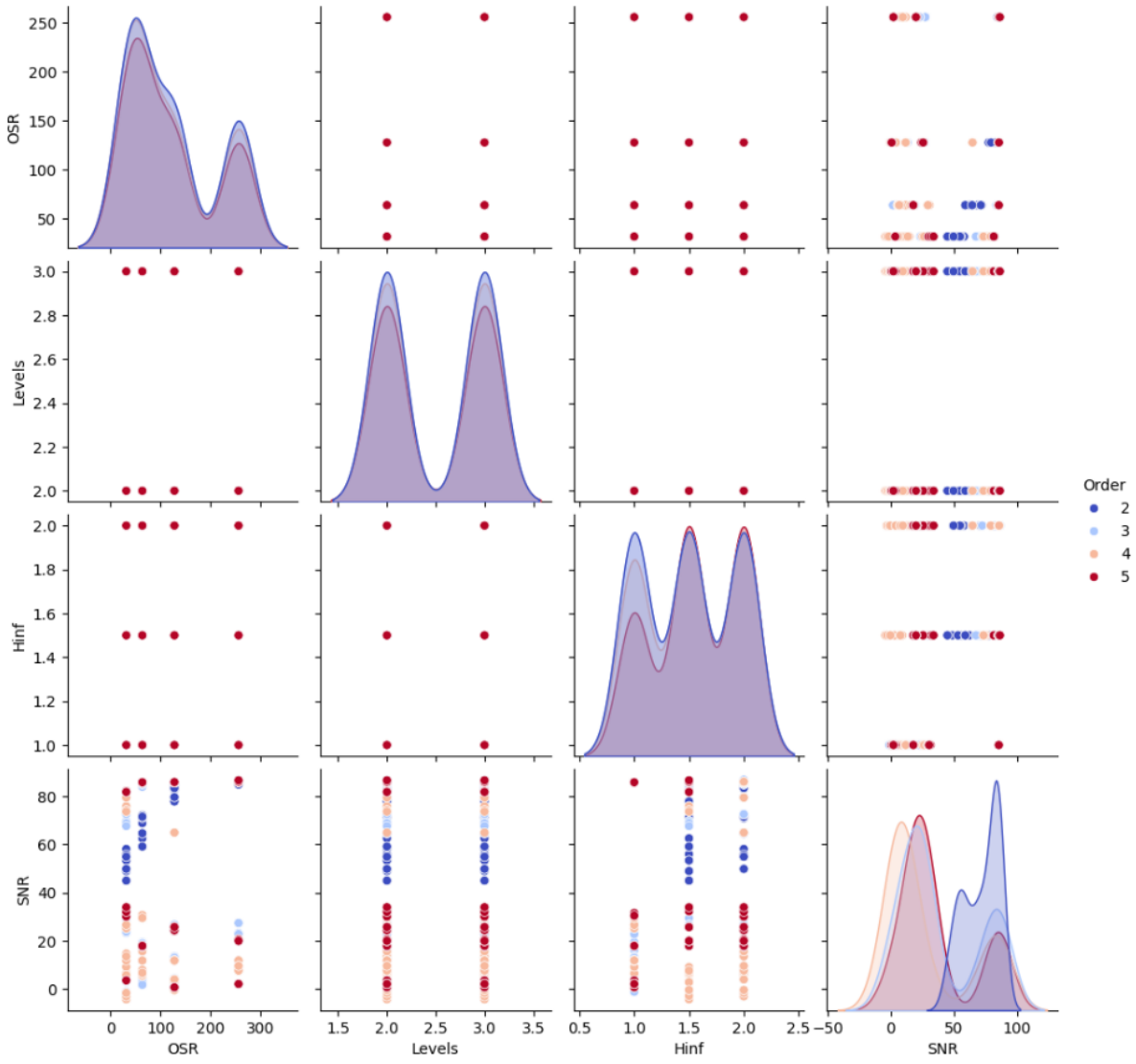
```python
# second solution: codify cathegorical values

# codifying the modulator architecture:
df_corr["Form"] = df_corr["Form"].astype("category").cat.codes

# select only number columns:
df_corr_numeric = df.select_dtypes(include=["number"])

# correlation matrix:
corr_matrix = df_corr_numeric.corr()

# heatmap:
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Correlation between parameters and SNR", fontsize=14)
plt.show()
```



It computes and visualizes the correlation between different numerical parameters of the ΣΔM, including the modulator order, the OSR, the number of quantization levels, the high-frequency gain and the SNR.

The diagonal of the heatmap has values of 1, of course, as each variable is perfectly correlated with itself. The SNR exhibits a negative correlation with the Order of the modulator (-0.35). This suggests that increasing the modulator order tends to reduce the SNR. The Hinf parameter shows a moderate positive correlation with SNR (0.26), implying that higher Hinf values could lead to an improvement in SNR. The OSR and SNR have a very weak correlation (0.088), indicating that within the analyzed range, the oversampling ratio does not significantly influence the SNR. The Levels parameter has almost no correlation with any other variable, suggesting that variations in the number of quantization levels do not significantly impact SNR or other parameters.

```
sns.pairplot(df, hue="Order", palette="coolwarm")
plt.show()
```



This plot allows for a detailed visualization of relationships between all numerical variables, color-coded by the modulator order.

The diagonal elements show Kernel Density Estimation (KDE) plots, which illustrate the distribution of each variable. The OSR and SNR distributions exhibit multiple peaks, indicating the presence of distinct operating regimes in the dataset. The Levels parameter shows a very discrete distribution, reinforcing its low correlation with other parameters.

The relationship between OSR and SNR appears to be highly dispersed, supporting the heatmap's finding that OSR does not strongly correlate with SNR. The Hinf vs SNR plot reveals a moderate positive trend, aligning with the heatmap correlation value (0.26). The Order vs SNR scatter plot shows that higher-order modulators (red dots) tend to have lower SNR values, consistent with the negative correlation (-0.35).

2.3 Creating an interactive interface with Streamlit

In this section, we aim to develop an interactive web-based application that allows users to dynamically adjust parameters and visualize the results in real time. For this purpose, we will use Streamlit, a Python package designed for building interactive web applications with minimal effort.

The first version of our interactive application is implemented in the script `streamlit_app_v1.py`, which we detail below.

```python
# streamlit_app_v1.py

import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# load dataset:
df = pd.read_csv("sigma_delta_results.csv")

# Streamlit page configuration:
st.set_page_config(page_title="SDM analysis", layout="wide")

# title:
st.title("Sigma-Delta Modulator analysis")

# sidebar filters:
st.sidebar.header("Filter options")

# select modulator order:
order = st.sidebar.selectbox("Select modulator order:", sorted(df["Order"].unique()))

# select architecture:
if "Form" in df.columns:
    architecture = st.sidebar.selectbox("Select architecture:", sorted(df["Form"].unique()))
    df = df[df["Form"] == architecture]

# select OSR range:
osr_min, osr_max = float(df["OSR"].min()), float(df["OSR"].max())
osr_range = st.sidebar.slider("Select OSR range:", osr_min, osr_max, (osr_min, osr_max))

# select Hinf gain:
if "Hinf" in df.columns:
    hinf_range = st.sidebar.slider("Select Hinf gain range:", float(df["Hinf"].min()),
                                   float(df["Hinf"].max()),
                                (float(df["Hinf"].min()), float(df["Hinf"].max())))

# select quantization levels:
if "Levels" in df.columns:
    levels = st.sidebar.multiselect("Select quantization levels:",
        sorted(df["Levels"].unique()), default=df["Levels"].unique())

# apply filters:
df_filtered = df[(df["Order"] == order) & (df["OSR"].between(*osr_range))]

if "Hinf" in df_filtered.columns:
    df_filtered = df_filtered[df_filtered["Hinf"].between(*hinf_range)]

```

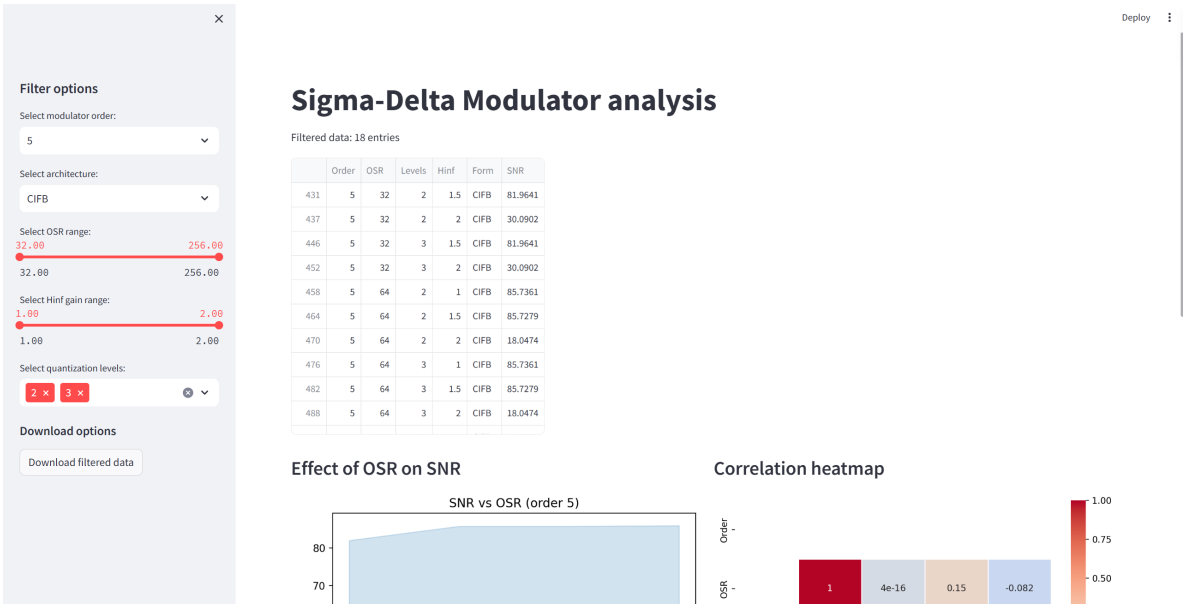
```
if "Levels" in df_filtered.columns:
    df_filtered = df_filtered[df_filtered["Levels"].isin(levels)]

# display filtered data:
st.write(f"Filtered data: {len(df_filtered)} entries")
st.dataframe(df_filtered)

# charts:
col1, col2 = st.columns(2)

# SNR vs OSR:
with col1:
    st.subheader("Effect of OSR on SNR")
    fig, ax = plt.subplots()
    sns.lineplot(data=df_filtered, x="OSR", y="SNR", marker="o", ax=ax)
    plt.xscale("log")
    plt.xlabel("OSR")
    plt.ylabel("SNR (dB)")
    plt.title(f"SNR vs OSR (order {order})")
    st.pyplot(fig)

# correlation heatmap:
with col2:
    st.subheader("Correlation heatmap")
    numeric_df = df_filtered.select_dtypes(include=["number"])
    corr_matrix = numeric_df.corr()
    fig, ax = plt.subplots(figsize=(8, 6))
    sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", linewidths=0.5, ax=ax)
    st.pyplot(fig)

# additional plots:
col3, col4 = st.columns(2)

# scatter plot: OSR vs SNR:
with col3:
    st.subheader("Scatter plot: OSR vs SNR")
    fig, ax = plt.subplots()
    sns.scatterplot(data=df_filtered, x="OSR", y="SNR", hue="Levels", palette="viridis", ax=ax)
    plt.xscale("log")
    plt.xlabel("OSR")
    plt.ylabel("SNR (dB)")
    plt.title(f"SNR vs OSR (order {order})")
    st.pyplot(fig)

# density plot: SNR distribution:
with col4:
    st.subheader("SNR density plot")
    fig, ax = plt.subplots()
    sns.kdeplot(df_filtered["SNR"], fill=True, color="blue", ax=ax)
    plt.xlabel("SNR (dB)")
    plt.ylabel("Density")
    plt.title("SNR distribution")
    st.pyplot(fig)

# download filtered data:
st.sidebar.subheader("Download options")
```

```
105 st.sidebar.download_button("Download filtered data", df_filtered.to_csv(index=False),
106     "filtered_data.csv", "text/csv")
```

Once we have created this script, we can go to a terminal and run:

```
streamlit run streamlit_app_v1.py
```

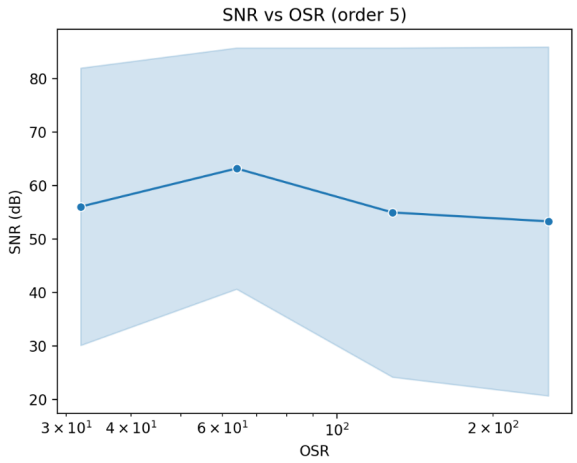
This command opens a new browser window displaying the interactive application:



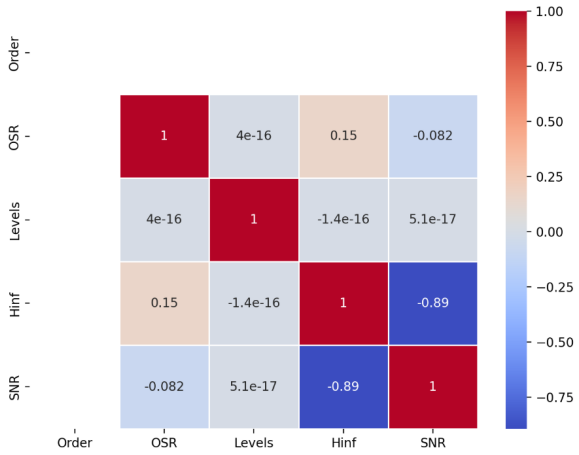
This is the main page of the application. In the left side, we can modify the parameters, such as the order of the modulator or the oversampling ratio. Then, the table shows which data have the specifications required (from the dataset previously obtained). We can download it as a CSV file if we want.

Also, it automatically shows some plots about the chosen configuration.

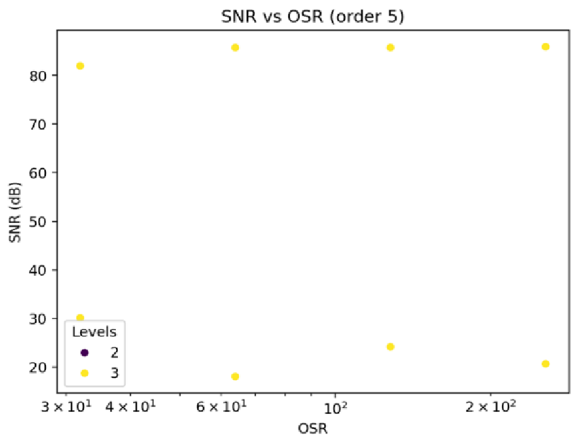
Effect of OSR on SNR



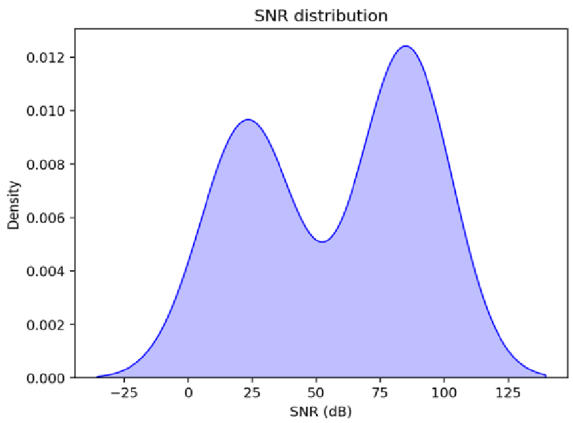
Correlation heatmap



Scatter plot: OSR vs SNR



SNR density plot



1. **Effect of OSR on SNR**: this plot illustrates how the SNR improves as the OSR increases. It is particularly useful for understanding the trade-offs in oversampling and how much SNR improvement can be expected for different modulator orders. The blue line with dots represents the mean SNR for a given OSR, and the shaded region around it likely represents confidence intervals or standard deviations.

2. **Correlation heatmap**: this heatmap visualizes the correlation between key parameters in the $\Sigma\Delta$M. It helps to understand how different factors influence each other. Each cell shows the correlation coefficient between two variables: 1 (red) indicates perfect postive correlation, -1 (blue) indicates perfect negative correlation, values near 0 (neutral colors) indicate weak or no correlation.

3. **Scatter plot: OSR vs SNR**: this scatter plot provides an alternative view of the OSR vs SNR relationship, while also differentiating results based on quantization levels.

4. **SNR density plot**: this plot shows the distribution of SNR values across the dataset. It helps analyze the range and frequency of different SNR values.

2.4 Improving the interactive interface

Date: March 04, 2025

In this section, we introduce several improvements to the interactive application developed in the previous section. The goal is to enhance user experience, provide additional visualization options, and improve data filtering capabilities for a more intuitive analysis of Sigma-Delta Modulator (SDM) parameters. To achieve this, we implement a refined version of the Streamlit application, named `streamlit_app_v2.py`. The code is as follow:

```python
# streamlit_app_v2.py

import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import numpy as np

# load dataset:
df = pd.read_csv("sigma_delta_results.csv")

# remove infinite values from SNR:
df = df.replace([np.inf, -np.inf], np.nan).dropna(subset=["SNR"])

# Streamlit page configuration:
st.set_page_config(page_title="SDM analysis", layout="wide")

# title:
st.title("Sigma-Delta Modulator analysis")

# sidebar filters:
st.sidebar.header("Filter options")

# select modulator order:
if "Order" in df.columns:
    orders = st.sidebar.multiselect("Select orders:", sorted(df["Order"].unique()),
        default=df["Order"].unique())

# select OSR range:
osr_min, osr_max = float(df["OSR"].min()), float(df["OSR"].max())
osr_range = st.sidebar.slider("Select OSR range:", osr_min, osr_max, (osr_min, osr_max))

# select quantization levels:
if "Levels" in df.columns:
    levels = st.sidebar.multiselect("Select quantization levels:",
        sorted(df["Levels"].unique()), default=df["Levels"].unique())

# select Hinf gain:
if "Hinf" in df.columns:
    hinf_range = st.sidebar.slider("Select Hinf gain range:", float(df["Hinf"].min()),
                                   float(df["Hinf"].max()),
                                (float(df["Hinf"].min()), float(df["Hinf"].max())))

# select architecture:
if "Form" in df.columns:
```

```
    architectures = st.sidebar.multiselect("Select architectures:",
        sorted(df["Form"].unique()), default=df["Form"].unique())

# select minimum SNR:
if "SNR" in df.columns:
    snr_min, snr_max = float(df["SNR"].min()), float(df["SNR"].max())
    snr_threshold = st.sidebar.slider("Select minimum SNR:", snr_min, snr_max, snr_min)

# apply filters:
df_filtered = df[df["Order"].isin(orders) & df["OSR"].between(*osr_range)]

if "Form" in df_filtered.columns:
    df_filtered = df_filtered[df_filtered["Form"].isin(architectures)]

if "Hinf" in df_filtered.columns:
    df_filtered = df_filtered[df_filtered["Hinf"].between(*hinf_range)]

if "Levels" in df_filtered.columns:
    df_filtered = df_filtered[df_filtered["Levels"].isin(levels)]

if "SNR" in df_filtered.columns:
    df_filtered = df_filtered[df_filtered["SNR"] >= snr_threshold]

# display filtered data:
st.write(f"Filtered data: {len(df_filtered)} entries")
st.dataframe(df_filtered)

# charts:
col1, col2 = st.columns(2)

# SNR vs OSR:
with col1:
    st.subheader("SNR vs OSR")
    fig = px.scatter(df_filtered, x="OSR", y="SNR", color="Levels"
        if "Levels" in df_filtered.columns else None,
                     log_x=True, title="SNR vs OSR", labels={"OSR": "OSR", "SNR": "SNR (dB)"})
    st.plotly_chart(fig, use_container_width=True)

# correlation heatmap:
with col2:
    st.subheader("Correlation heatmap")
    numeric_df = df_filtered.select_dtypes(include=["number"])
    corr_matrix = numeric_df.corr()
    fig, ax = plt.subplots(figsize=(8, 6))
    sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", linewidths=0.5, ax=ax)
    st.pyplot(fig)

# boxplots:
col3, col4 = st.columns(2)

# boxplot: SNR vs order:
with col3:
    st.subheader("SNR vs Modulator order")
    fig, ax = plt.subplots(figsize=(8, 6))
    sns.boxplot(data=df_filtered, x="Order", y="SNR", palette="viridis", ax=ax)
    plt.xlabel("Modulator order")
```

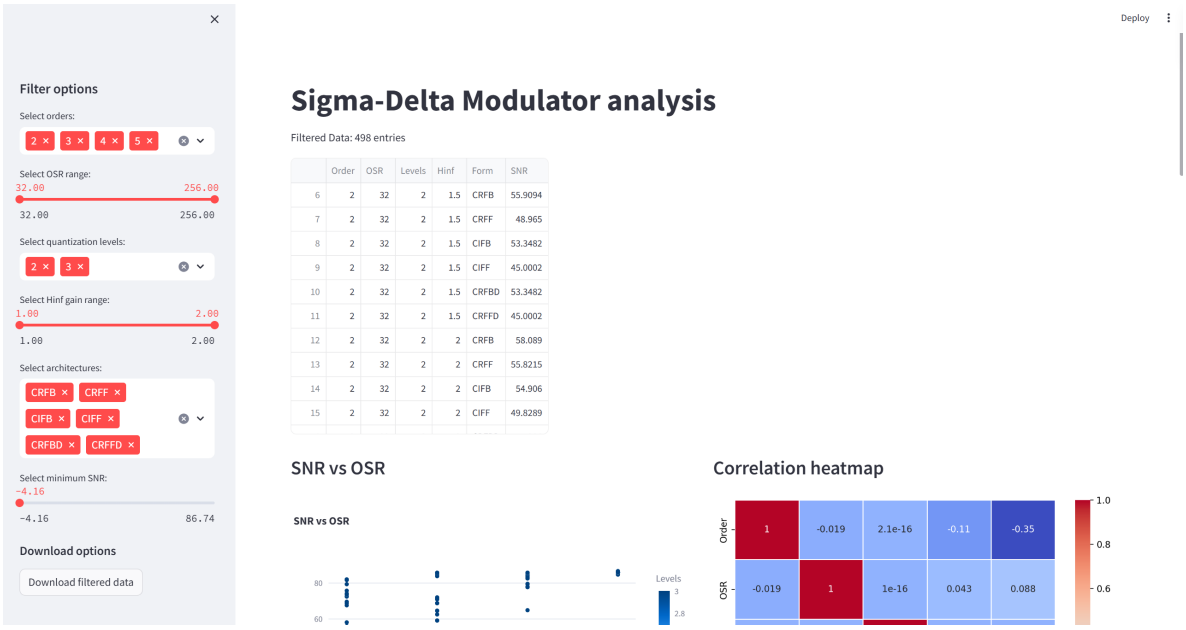
```
    plt.ylabel("SNR (dB)")
    plt.title("SNR distribution by modulator order")
    st.pyplot(fig)

# boxplot: SNR vs OSR:
with col4:
    st.subheader("SNR vs OSR")
    fig, ax = plt.subplots(figsize=(8, 6))
    sns.boxplot(data=df_filtered, x="OSR", y="SNR", palette="viridis", ax=ax)
    plt.xlabel("OSR")
    plt.ylabel("SNR (dB)")
    plt.title("SNR distribution by OSR")
    plt.xticks(rotation=45)
    st.pyplot(fig)

# SNR distribution: density plot and histogram:
col5, col6 = st.columns(2)

# density plot:
with col5:
    st.subheader("SNR distribution (density)")
    fig, ax = plt.subplots()
    sns.kdeplot(df_filtered["SNR"], fill=True, color="blue", ax=ax)
    plt.xlabel("SNR (dB)")
    plt.ylabel("Density")
    plt.title("SNR density plot")
    st.pyplot(fig)

# histogram:
with col6:
    st.subheader("SNR distribution (histogram)")
    fig = px.histogram(df_filtered, x="SNR", nbins=30, title="SNR Histogram",
                       labels={"SNR": "SNR (dB)"}, opacity=0.7, color="Levels"
                            if "Levels" in df_filtered.columns else None)
    st.plotly_chart(fig, use_container_width=True)

# architecture comparison:
if "Form" in df_filtered.columns:
    st.subheader("Architecture comparison: SNR vs OSR")
    fig = go.Figure()
    for arch in df_filtered["Form"].unique():
        arch_data = df_filtered[df_filtered["Form"] == arch]
        fig.add_trace(go.Scatter(x=arch_data["OSR"], y=arch_data["SNR"],
            mode="lines+markers", name=arch))
    fig.update_layout(xaxis_title="OSR", yaxis_title="SNR (dB)", xaxis_type="log")
    st.plotly_chart(fig, use_container_width=True)

# SNR vs Hinf gain:
if "Hinf" in df_filtered.columns:
    st.subheader("SNR vs Hinf gain")
    fig = px.scatter(df_filtered, x="Hinf", y="SNR", color="Form"
        if "Form" in df_filtered.columns else None,
                     labels={"Hinf": "Hinf gain", "SNR": "SNR (dB)"})
    st.plotly_chart(fig, use_container_width=True)

# download filtered data:
```

```
st.sidebar.subheader("Download options")
st.sidebar.download_button("Download filtered data", df_filtered.to_csv(index=False),
    "filtered_data.csv", "text/csv")
```

To run the improved interactive application, execute the following command in the terminal:

```
streamlit run streamlit_app_v2.py
```

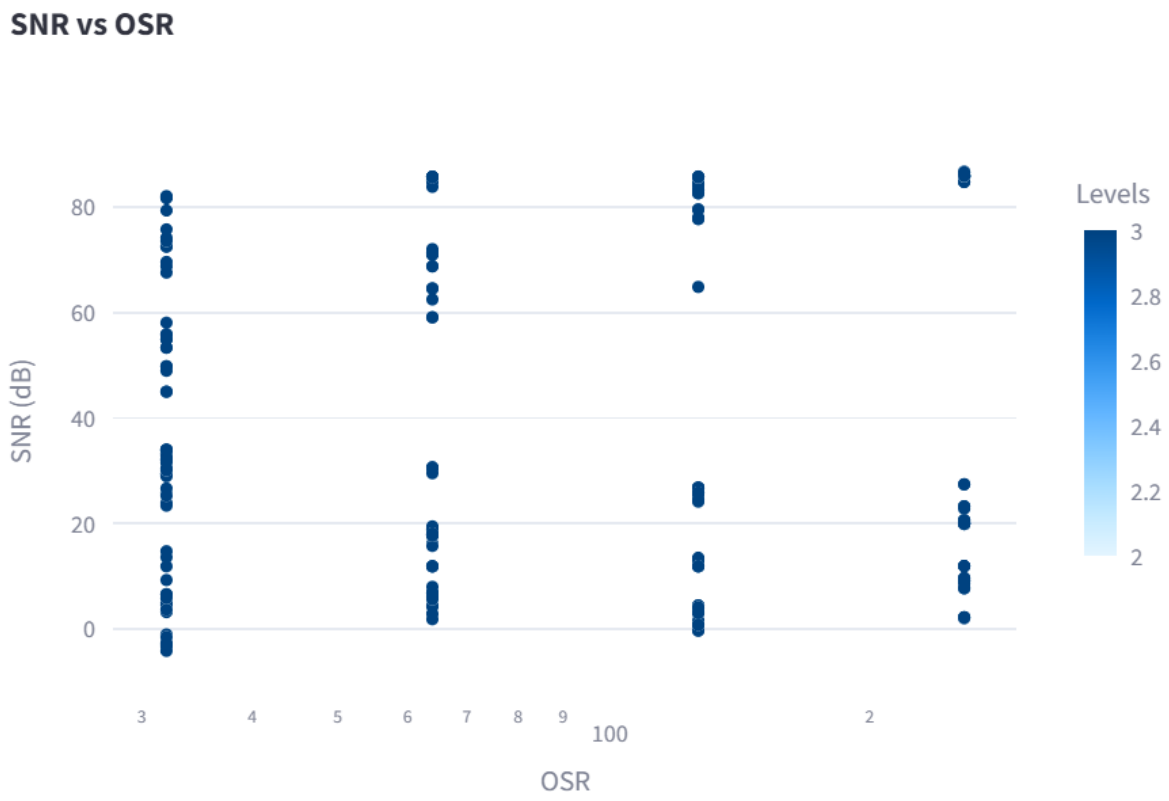
Once executed, the application launches in a browser, providing an interactive interface with real-time filtering and visualization updates. Users can manipulate the design parameters and instantly observe their impact on modulator performance.



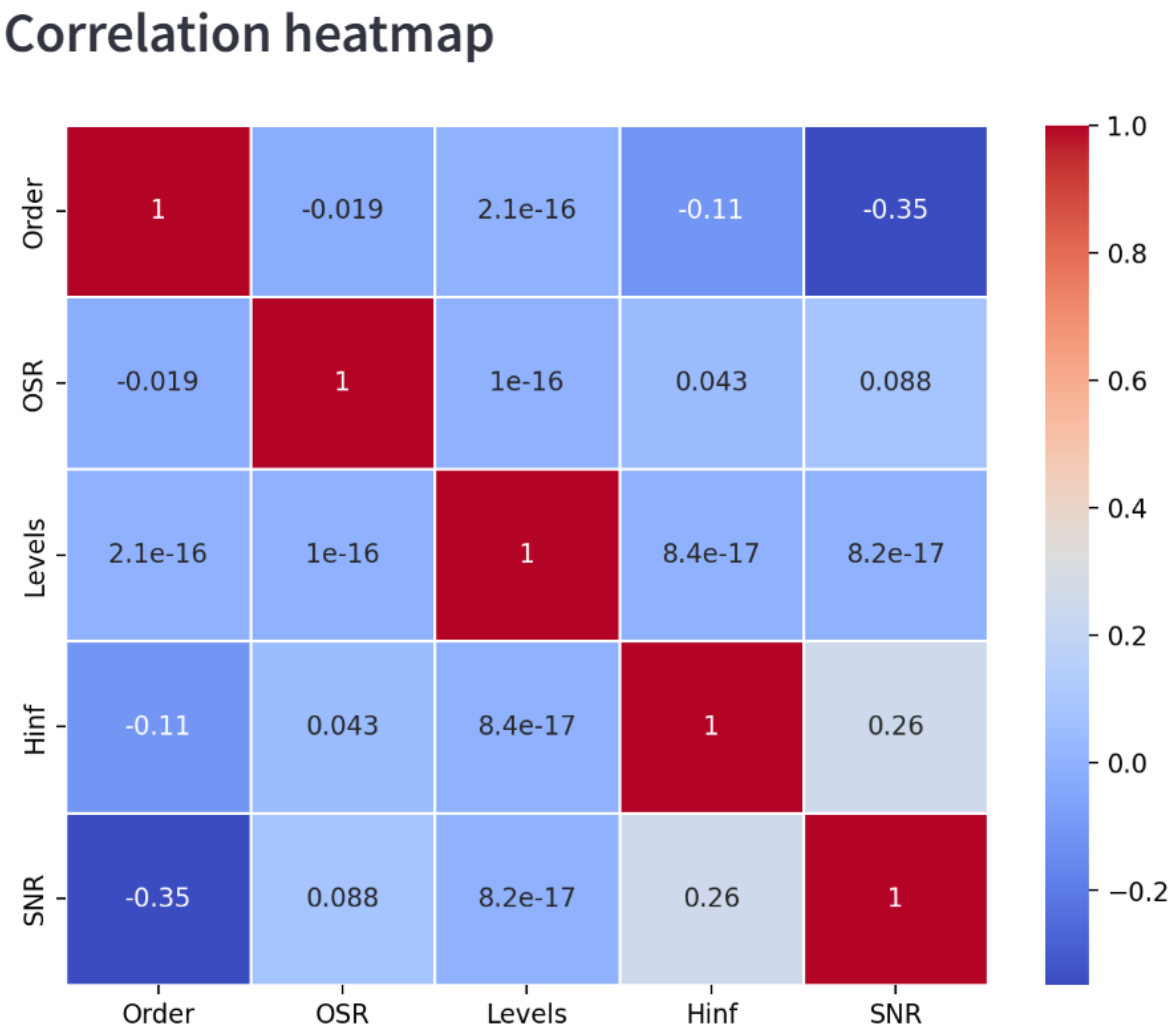
The sidebar now offers more customization filters, allowing you to select more possible configurations and displaying a larger number of results. In addition, we can also filter by SNR, so that we only keep those configurations with an SNR that exceeds a certain threshold.

We also have new graphics, which we will detail below one by one. Many of them are now interactive and we can play with them, enlarge certain areas, download them and more.

SNR vs OSR

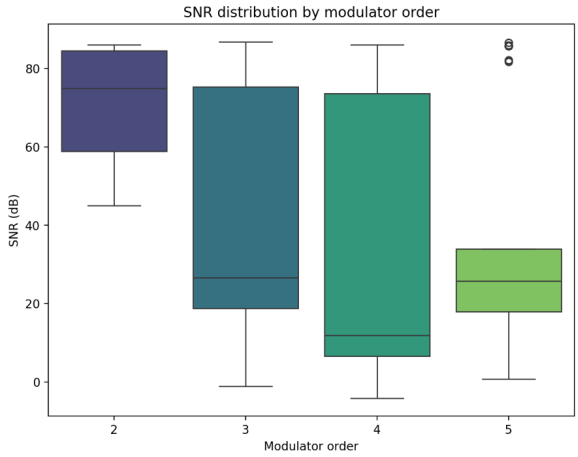


SNR vs OSR: in this plot we can see the relationship between SNR and OSR, taking into consideration the number of quantization levels. For instance, we can observe a trend: as OSR increases, SNR also increases (not a lot).

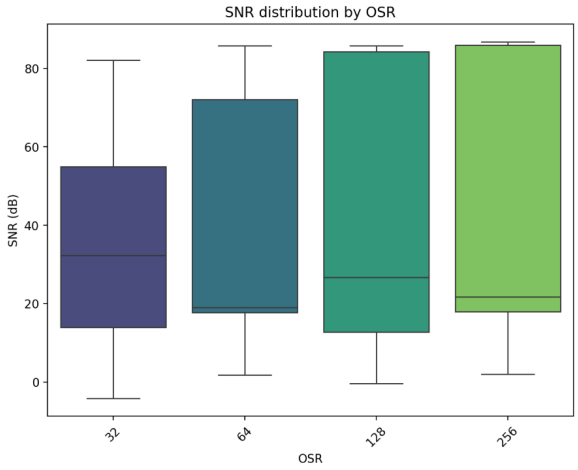


Correlation heatmap: this is one of the most interesting plots to analyze. This shows the correlation matrix between the design parameters. For the complete dataset generated previously, we can see (looking to the last row) that the order and the Hinf gain are the most correlated parameters with the SNR.

SNR vs Modulator order

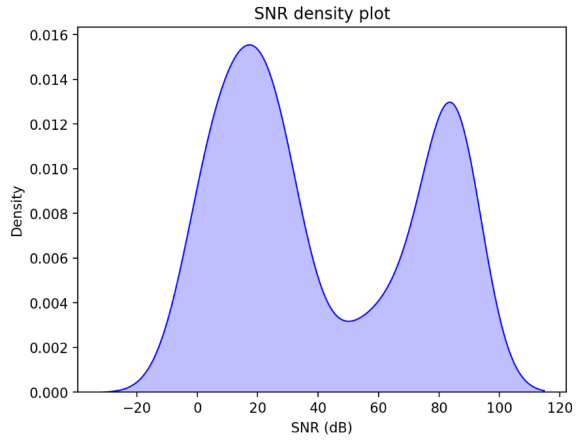


SNR vs OSR

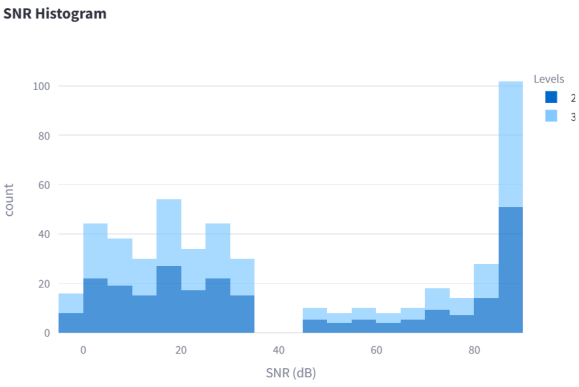


Boxplots: these two boxplots analyze the relationship between the SNR and different design parameters. The first one is versus the modulator order. We can see that as the modulator order increases, the SNR trends to decrease, although there are some higher outliers. The second boxplot shows the SNR distribution by OSR, illustrating that higher values of oversampling ratio means higher SNR, in general.

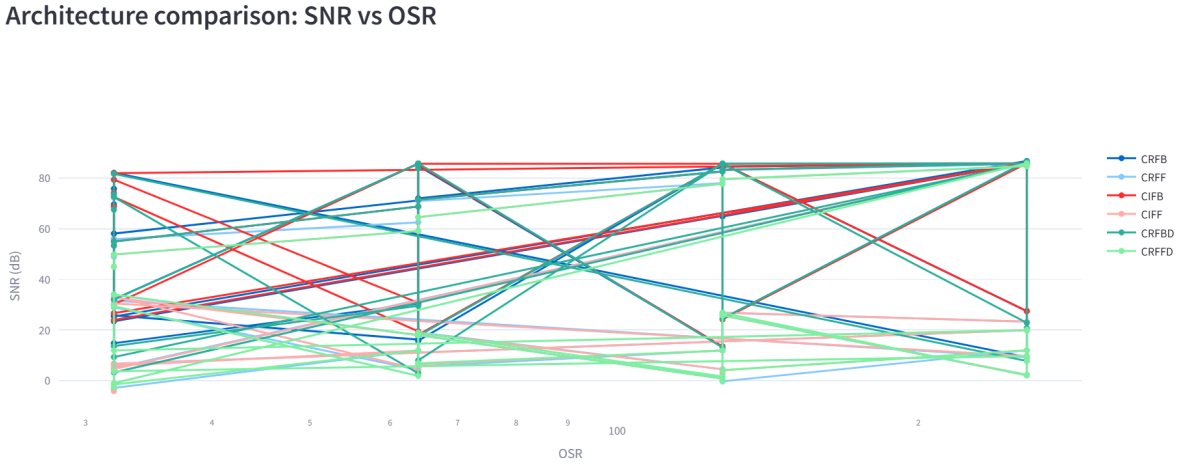
SNR distribution (density)



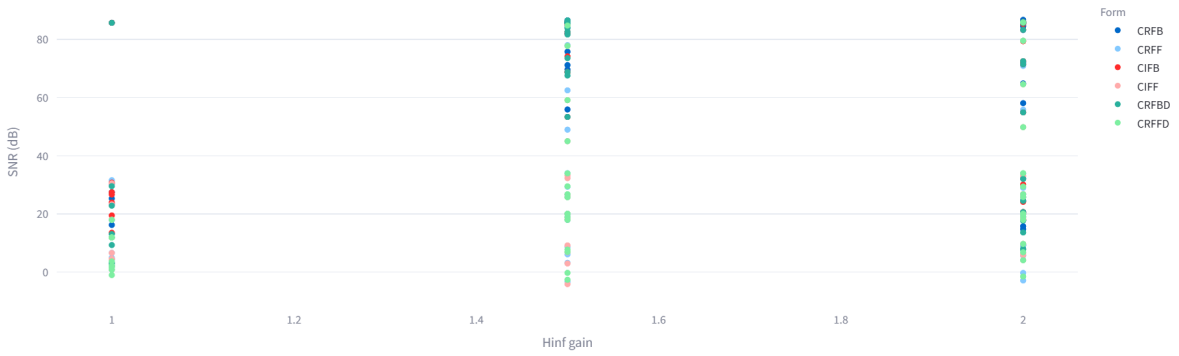
SNR distribution (histogram)



SNR distributions: these plots show the SNR distribution. The first one as a density function, and the second one as an histogram. They are also interesting because we can see where are mostly located SNR values.



Architecture comparison: SNR vs OSR: this plot shows the SNR values for different architectures and values of OSR.



SNR vs Hinf gain: this last plot shows the relationship between Hinf gain and SNR for different architecture. For instance, we can see that, in general, 1.0 is worse than 1.5 or 2.0.

3 Conclusions and future steps

Date: March 04, 2025

This report has detailed the development and refinement of an **interactive tool for analyzing ΣΔM design parameters**. By implementing brute-force search techniques, storing results in a structured format, and visualizing the data through interactive plots, we have built a comprehensive system for ΣΔM optimization. The final version of the interactive interface allows users to explore the impact of key design parameters such as order, OSR, quantization levels, and feedback gain on SNR.

The application is particularly useful for comparative analysis of different modulator architectures and parameters settings, rapid visualization of trends and correlations with large datasets and decision-making support when selecting optimal design parameters for specific applications.

The next phase of this project will focus on **generating datasets** to train artificial intelligence models for ΣΔM optimization. The goal is to move beyond brute-force search and develop machine learning algorithms capable of predicting optimal modulator configurations with greater efficiency. Also, we will be able to analyze these datasets thanks to the designed application (which needs to be improved).