UNIVERSIDADE
DO ESTADO DE MINAS GERAIS | UEMG
UNIDADE DIVINÓPOLIS

Pré-processamento, análise e filtragem de harmônicas em sinais

Marcos Vinícius / 16-95704

Nilton Rabelo / 16-24935

Prof.: Arismar Morais Gonçalves Júnior

Disciplina: TEEC – Fundamentos de *Machine Learning*

Turno: Noturno

Divinópolis

Maio de 2025

1. Introdução

A qualidade da energia elétrica é um fator crítico para o funcionamento adequado de dispositivos e sistemas em ambientes residenciais, comerciais e industriais. Sinais elétricos ideais apresentam uma forma de onda senoidal pura; no entanto, na prática, diversas não linearidades presentes nas cargas e nos sistemas de distribuição podem introduzir distorções nessa forma de onda. Entre as perturbações mais comuns e problemáticas estão as componentes harmônicas, que são frequências múltiplas da frequência fundamental do sistema e podem levar ao mau funcionamento de equipamentos, sobreaquecimento, perdas de energia e redução da vida útil de componentes. (Khan, M. U., Aziz, S. e Usman, A., 2023)

O processamento digital de sinais oferece um conjunto robusto de ferramentas para a análise e mitigação de tais distorções. Através de técnicas como a visualização de sinais no domínio do tempo, é possível observar alterações macroscópicas na forma de onda. Contudo, para uma caracterização precisa das componentes de frequência que constituem o sinal, a análise no domínio da frequência é indispensável. A Transformada Rápida de Fourier (FFT) destaca-se como um algoritmo eficiente para decompor um sinal em suas diversas frequências constituintes, permitindo a identificação clara da frequência fundamental e de suas harmônicas.

Uma vez identificadas, as componentes harmônicas indesejadas podem ser atenuadas ou eliminadas através da aplicação de filtros digitais projetados para selecionar ou rejeitar bandas de frequência específicas. Esse tipo de processamento é fundamental não apenas para restaurar a qualidade do sinal elétrico, mas também como uma etapa crucial de pré-processamento em sistemas mais complexos, como os baseados em Aprendizado de Máquina. Nesses sistemas, a extração de características relevantes a partir de sinais "limpos" ou bem caracterizados é essencial para o treinamento eficaz de modelos capazes de realizar diagnósticos, prognósticos ou classificações automáticas de distúrbios de qualidade de energia.

2. Metodologia

A análise comparativa dos sinais elétricos e a avaliação das técnicas de processamento seguiram uma sequência de procedimentos implementados na linguagem de programação Python. Foram utilizadas bibliotecas como Pandas para manipulação de dados, NumPy para operações numéricas, Matplotlib para visualização gráfica, e SciPy (módulos fft e signal) para a transformada de Fourier e projeto de filtros. Parâmetros globais foram definidos no início do script, incluindo os caminhos para os arquivos de dados (Pure_Sinusoidal.csv para sinais

normais e Harmonics.csv para sinais com harmônicas), o número de amostras por sinal (N_AMOSTRAS = 100), a frequência de amostragem (FREQ_AMOSTRAGEM = 5000 Hz), a frequência fundamental esperada (FREQ = 50 Hz), e o número de sinais de cada classe para visualização (N_SINAIS_PARA_PLOTAR = 3).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from scipy import signal as sig

# --- Parâmetros globais ---
ARQUIVO_NORMAL = 'Dataset_Power Quality Disturbance/Pure_Sinusoidal.csv'
ARQUIVO_ANORMAL = 'Dataset_Power Quality Disturbance/Harmonics.csv'
N_AMOSTRAS = 100            # Número de amostras por sinal
FREQ_AMOSTRAGEM = 5000      # Frequência de amostragem em Hz (assumindo
fundamental de 50Hz e 100 amostras/ciclo)
FREQ = 50                   # Hz
N_SINAIS_PARA_PLOTAR = 3    # Número de sinais de cada classe para plotar
```

O processamento iniciou-se com o carregamento e organização dos dados, realizado pela função carregar_e_organizar_dados. Esta função lê os dados dos arquivos CSV especificados, tratando exceções caso os arquivos não sejam encontrados. Os dataframes resultantes são transpostos para que cada coluna represente um sinal individual. Em seguida, são concatenados horizontalmente, com os sinais anormais precedendo os normais, e as colunas do dataframe combinado são renomeadas para identificação (ex: "Anormal_0", "Normal_0"). A função retorna este dataframe combinado e as contagens de sinais de cada classe.

```
# --- 1. Organização dos Dados ---
def carregar_e_organizar_dados(arquivo_normal, arquivo_anormal):
    # Organização dos sinais em dataframes de modo que as N primeiras
colunas/posições sejam ocupadas pela classe "Anormal" e as N seguintes pela
classe "Normal"
    # Carrega os dados dos arquivos CSV, transpõe e concatena os dataframes
    try:
        # Carregar dados da classe anormal (Harmonics)
        df_anormal = pd.read_csv(arquivo_anormal, header=None)
        # Carregar dados da classe normal (Pure_Sinusoidal)
        df_normal = pd.read_csv(arquivo_normal, header=None)
    except FileNotFoundError as e:
```

```
        print(f"Erro: arquivo não encontrado: {e}")
        return None, 0, 0

    # Transpor os dataframes (sinais como colunas, amostras como linhas)
    df_anormal_T = df_anormal.T
    df_normal_T = df_normal.T

    num_sinais_anormais = df_anormal_T.shape[1]
    num_sinais_normais = df_normal_T.shape[1]

    print(f"Número de sinais 'anormais' ({arquivo_anormal}):
{num_sinais_anormais}")
    print(f"Número de sinais 'normais' ({arquivo_normal}):
{num_sinais_normais}")
    print(f"Número de amostras por sinal: {df_anormal_T.shape}")

    # Renomear colunas para facilitar identificação
    df_anormal_T.columns = [f'Anormal_{i+1}' for i in
range(num_sinais_anormais)]
    df_normal_T.columns = [f'Normal_{i+1}' for i in range(num_sinais_normais)]

    # Concatenar: anormais primeiro, depois normais
    df_combinado = pd.concat([df_anormal_T, df_normal_T], axis=1)

    print(f"\nDimensões do dataframe anormal transposto:
{df_anormal_T.shape}")
    print(f"Dimensões do dataframe normal transposto: {df_normal_T.shape}")
    print(f"Dimensões do dataframe combinado: {df_combinado.shape}")

    return df_combinado, num_sinais_anormais, num_sinais_normais
```

Para a análise visual no domínio do tempo, a função visualizar_sinais_tempo seleciona aleatoriamente uma quantidade de sinais de cada classe do dataframe combinado, utilizando um estado aleatório fixo (random_state = 42) para reprodutibilidade. Estes são plotados em dois subplots verticais: o superior para os sinais anormais e o inferior para os normais. Títulos, rótulos dos eixos ("Amostra" vs. "Amplitude") e legendas são configurados antes de exibir a figura. Os dataframes dos sinais selecionados para plotagem são retornados por esta função.

```
# --- 2. Visualização no domínio do tempo ---
def visualizar_sinais_tempo(df, num_sinais_anormais, num_sinais_normais,
n_plot=3):
    # Escolha aleatória de três sinais de cada classe e visualização dos
mesmos no domínio do tempo (isto é, obtenção dos gráficos temporais dos
sinais)
```

```
    if df is None:
        return pd.DataFrame(), pd.DataFrame() # Retornar dataFrames vazios
caso haja algum problema

    # Verificar se há sinais suficientes para plotar
    n_plot_anormais = min(n_plot, num_sinais_anormais)
    n_plot_normais = min(n_plot, num_sinais_normais)

    if n_plot_anormais == 0 and n_plot_normais == 0:
        print("Não há sinais suficientes para plotar.")
        return pd.DataFrame(), pd.DataFrame()

    sinais_anormais_plot = pd.DataFrame()
    sinais_normais_plot = pd.DataFrame()

    if n_plot_anormais > 0:
        # Selecionar índices aleatórios para sinais anormais
        indices_anormais_plot = np.random.choice(num_sinais_anormais,
n_plot_anormais, replace=False)
        sinais_anormais_plot = df.iloc[:, indices_anormais_plot]

    if n_plot_normais > 0:
        # Selecionar índices aleatórios para sinais normais
        indices_normais_plot = np.random.choice(num_sinais_normais,
n_plot_normais, replace=False)
        sinais_normais_plot = df.iloc[:, num_sinais_anormais +
indices_normais_plot]

    tempo = np.arange(N_AMOSTRAS) # Vetor de amostras (0 a N_AMOSTRAS-1)

    # Ajustar o número de colunas do subplot dinamicamente
    num_cols_subplot = max(n_plot_anormais, n_plot_normais)
    if num_cols_subplot == 0: # Caso não haja sinais para plotar
        return sinais_anormais_plot, sinais_normais_plot

    plt.figure(figsize=(4 * num_cols_subplot, 8)) # Ajustar tamanho da figura
    plt.suptitle('Sinais selecionados no domínio do tempo', fontsize=16)

    for i in range(n_plot_anormais):
        # Plotar sinais anormais
        plt.subplot(2, num_cols_subplot, i + 1)
        plt.plot(tempo, sinais_anormais_plot.iloc[:, i])
        plt.title(f'Sinal anormal: {sinais_anormais_plot.columns[i]}')
        plt.xlabel('Número da amostra')
        plt.ylabel('Amplitude')
        plt.grid(True)

    for i in range(n_plot_normais):
        # Plotar sinais normais
```

```
        plt.subplot(2, num_cols_subplot, i + 1 + num_cols_subplot)
        plt.plot(tempo, sinais_normais_plot.iloc[:, i])
        plt.title(f'Sinal normal: {sinais_normais_plot.columns[i]}')
        plt.xlabel('Número da amostra')
        plt.ylabel('Amplitude')
        plt.grid(True)

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()

    return sinais_anormais_plot, sinais_normais_plot
```

A análise espectral no domínio da frequência é conduzida pela função analisar_sinais_frequencia. Ela recebe os sinais selecionados, a frequência de amostragem e o número de amostras. Uma função aninhada, plotar_fft, calcula a Transformada Rápida de Fourier (FFT) usando fft(), gera o vetor de frequências com fftfreq(), e calcula o espectro de amplitude unilateral (metade positiva, normalizada pelo número de amostras). Este espectro é plotado (Amplitude vs. Frequência em Hz) em subplots dedicados – um para os sinais anormais e outro para os normais. Títulos apropriados e rótulos são adicionados para clareza.

```
# --- 3. Emprego e visualização da Transformada de Fourier (FFT) ---
def analisar_sinais_frequencia(sinais_anormais, sinais_normais, fs, n_samples,
n_plot=3):
    # Emprego da Transformada de Fourier (FFT) para os sinais escolhidos na
etapa anterior e visualização do espectro de frequências do mesmo
    if sinais_anormais.empty and sinais_normais.empty:
        print("Nenhum sinal encontrado.")
        return

    n_plot_anormais = sinais_anormais.shape[1]
    n_plot_normais = sinais_normais.shape[1]
    num_cols_subplot = max(n_plot_anormais, n_plot_normais)

    if num_cols_subplot == 0:
        return

    plt.figure(figsize=(4 * num_cols_subplot, 8)) # Ajustar tamanho da figura
    plt.suptitle('Transformada de Fourier (FFT) dos sinais selecionados',
fontsize=16)

    # Frequências para o eixo X da FFT (apenas positivas)
    freq_axis = fftfreq(n_samples, 1/fs)[:n_samples//2]
```

```
    for i in range(n_plot_anormais):
        # FFT para sinais anormais
        sinal_anormal_atual = sinais_anormais.iloc[:, i]
        fft_anormal = fft(sinal_anormal_atual.values)
        magnitude_anormal = np.abs(fft_anormal)[:n_samples//2] # Pegar apenas
a metade positiva

        plt.subplot(2, num_cols_subplot, i + 1)
        plt.plot(freq_axis, magnitude_anormal)
        plt.title(f'FFT sinal anormal: {sinais_anormais.columns[i]}')
        plt.xlabel('Frequência (Hz)')
        plt.ylabel('Magnitude')
        plt.grid(True)
        plt.xlim(0, fs/10) # Limitar eixo X para melhor visualização (ex: até
500 Hz se FS=5000)

    for i in range(n_plot_normais):
        # FFT para sinais normais
        sinal_normal_atual = sinais_normais.iloc[:, i]
        fft_normal = fft(sinal_normal_atual.values)
        magnitude_normal = np.abs(fft_normal)[:n_samples//2] # Pegar apenas a
metade positiva

        plt.subplot(2, num_cols_subplot, i + 1 + num_cols_subplot)
        plt.plot(freq_axis, magnitude_normal)
        plt.title(f'FFT sinal normal: {sinais_normais.columns[i]}')
        plt.xlabel('Frequência (Hz)')
        plt.ylabel('Magnitude')
        plt.grid(True)
        plt.xlim(0, fs/10) # Limitar eixo X

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()
```

Com o objetivo de atenuar as componentes harmônicas, a função aplicar_e_visualizar_filtro é utilizada. Ela opera sobre um único sinal (neste caso, o primeiro sinal anormal selecionado). Primeiramente, define os parâmetros para um filtro Butterworth passa-baixas de 4ª ordem (ordem_filtro = 4). A frequência de corte é estabelecida em 75 Hz (1.5 vezes a frequência fundamental de 50 Hz) e normalizada em relação à frequência de Nyquist (freq_corte_normalizada = (FREQ * 1.5) / (FREQ_AMOSTRAGEM / 2)). O filtro é projetado com sig.butter() e aplicado ao sinal usando sig.filtfilt(), que realiza uma filtragem de passagem dupla para evitar distorção de fase. O resultado é um sinal_filtrado. Para avaliação, a função gera uma figura com dois subplots: o primeiro exibe o sinal original e o filtrado no domínio do

tempo; o segundo mostra os espectros de frequência de ambos os sinais (original e filtrado), permitindo a visualização do efeito do filtro na atenuação das harmônicas.

```
# --- 4. Filtragem digital (filtro IIR Butterworth Passa-Baixas) ---
def aplicar_e_visualizar_filtro(nome_sinal, sinais_originais, freq,
n_amostras):
    # Realização da filtragem digital dos sinais e visualização das respostas
em frequência dos mesmos
    sinal_original = sinais_originais.values

    # Parâmetros do filtro
    ordem_filtro = 4
    freq_corte = 300            # Hz (atenuar harmônicas acima da 5a para
50Hz)
    nyquist_freq = 0.5 * freq
    freq_corte_normalizada = freq_corte / nyquist_freq

    # Projetar o filtro Butterworth passa-baixas
    b, a = sig.butter(ordem_filtro, freq_corte_normalizada, btype='low',
analog=False)

    # Aplicar o filtro
    sinal_filtrado = sig.lfilter(b, a, sinal_original)

    # Visualizar resposta em frequência do filtro
    w, h = sig.freqz(b, a, worN=8000, fs=freq)
    plt.figure(figsize=(9.6, 8)) # Ajustar tamanho da figura
    plt.subplot(3, 1, 1)
    plt.plot(w, np.abs(h))
    plt.title(f'Resposta em frequência do filtro Butterworth (corte:
{freq_corte} Hz)')
    plt.xlabel('Frequência (Hz)')
    plt.ylabel('Ganho')
    plt.grid(True)
    plt.axvline(freq_corte, color='red', linestyle='--', label=f'Freq. Corte
({freq_corte} Hz)')
    plt.legend()

    # Visualizar sinal original vs. filtrado no domínio do tempo
    tempo = np.arange(n_amostras)
    plt.subplot(3, 1, 2)
    plt.plot(tempo, sinal_original, label='Sinal original')
    plt.plot(tempo, sinal_filtrado, label='Sinal filtrado', linestyle='--')
    plt.title(f'Sinal "{nome_sinal}" original vs. filtrado (Tempo)')
    plt.xlabel('Número da amostra')
    plt.ylabel('Amplitude')
```

```
    plt.legend()
    plt.grid(True)

    # Visualizar FFT do sinal original vs. filtrado
    freq_axis = fftfreq(n_amostras, 1/freq)[:n_amostras//2]

    fft_original = fft(sinal_original)
    magnitude_original = np.abs(fft_original)[:n_amostras//2]

    fft_filtrado = fft(sinal_filtrado)
    magnitude_filtrado = np.abs(fft_filtrado)[:n_amostras//2]

    plt.subplot(3, 1, 3)
    plt.plot(freq_axis, magnitude_original, label='FFT original')
    plt.plot(freq_axis, magnitude_filtrado, label='FFT filtrado', linestyle='-
-')
    plt.title(f'FFT Sinal "{nome_sinal}" original vs. filtrado')
    plt.xlabel('Frequência (Hz)')
    plt.ylabel('Magnitude')
    plt.legend()
    plt.grid(True)
    plt.xlim(0, freq/5) # Limitar eixo X para melhor visualização

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

A execução principal do script, contida no bloco if __name__ == "__main__":, orquestra estas etapas. Após uma mensagem inicial, chama carregar_e_organizar_dados(). Se bem-sucedido, prossegue para visualizar_sinais_tempo(). Com os sinais selecionados, invoca analisar_sinais_frequencia(). Finalmente, se sinais anormais foram selecionados, o primeiro deles é passado para aplicar_e_visualizar_filtro(). O script inclui verificações para tratar cenários onde os dados não são carregados ou não há sinais anormais para a etapa de filtragem, imprimindo mensagens informativas. Esta abordagem metodológica estruturada permite não apenas identificar e caracterizar as diferenças entre os sinais elétricos normais e os contaminados por harmônicas, mas também demonstrar a aplicação prática de uma técnica de mitigação por meio da filtragem digital, conforme detalhado na execução do código.

```
# --- Execução Principal ---
if __name__ == '__main__':
    print("--- Iniciando pré-processamento de Sinais ---")

    # 1. Organização dos Dados
```

```
    print("\n--- 1. Organização dos dados ---")
    df_combinado, num_anormais, num_normais =
carregar_e_organizar_dados(ARQUIVO_NORMAL, ARQUIVO_ANORMAL)

    if df_combinado is not None:
        # 2. Visualização no domínio do tempo
        print(f"\n--- 2. Visualização de até {N_SINAIS_PARA_PLOTAR} Sinais
Aleatórios no Domínio do Tempo ---")
        sinais_anormais_selecionados, sinais_normais_selecionados =
visualizar_sinais_tempo(
            df_combinado, num_anormais, num_normais,
n_plot=N_SINAIS_PARA_PLOTAR
        )

        # 3. Transformada de Fourier (FFT)
        if not sinais_anormais_selecionados.empty or not
sinais_normais_selecionados.empty:
            print(f"\n--- 3. Emprego e visualização da Transformada de Fourier
(FFT) ---")
            analisar_sinais_frequencia(
                sinais_anormais_selecionados, sinais_normais_selecionados,
                FREQ_AMOSTRAGEM, N_AMOSTRAS, n_plot=N_SINAIS_PARA_PLOTAR
            )

            # 4. Filtragem digital (IIR Butterworth Passa-Baixas)
            if not sinais_anormais_selecionados.empty:
                print("\n--- 4. Filtragem digital (IIR Butterworth Passa-
Baixas) ---")
                sinal_anormal_nome = sinais_anormais_selecionados.columns
                sinal_anormal_dados = sinais_anormais_selecionados.iloc[:, 0]
                aplicar_e_visualizar_filtro(sinal_anormal_nome,
sinal_anormal_dados, FREQ_AMOSTRAGEM, N_AMOSTRAS)
            else:
                print("\nNenhum sinal anormal selecionado para demonstrar a
filtragem.")
        else:
            print("\nNão foi possível selecionar sinais para análise de
frequência e filtragem.")
    else:
        print("\nProcessamento interrompido devido a erro no carregamento dos
dados.")

    print("\n--- Pré-processamento Concluído ---")
```

3. Resultados

Após obter e organizar os dados no dataframe e escolher 3 sinais normais e anormais, a análise inicial no domínio do tempo, conforme ilustrado na figura (1), revela as distintas morfologias dos sinais. Os três sinais normais selecionados, apresentados no subplot inferior, exibem a forma de onda senoidal pura esperada, caracterizada por sua oscilação suave e periódica, consistente com a frequência fundamental de 50 Hz. Em contrapartida, os três sinais anormais selecionados, exibidos no subplot superior, embora mantenham a periodicidade fundamental, apresentam uma acentuada distorção. Esta distorção manifesta-se visualmente por meio de picos achatados e ondulações adicionais na forma de onda, indicando a superposição de outras componentes de frequência à fundamental.

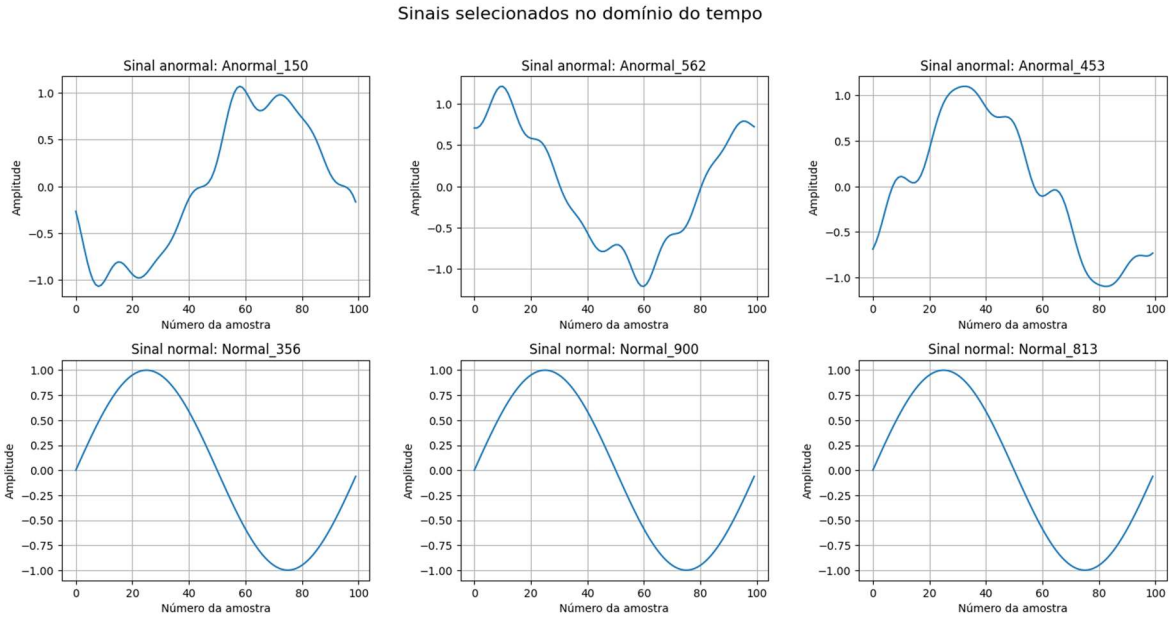

Figura 1: Sinais selecionados no domínio do tempo

A transição para o domínio da frequência, por meio da Transformada Rápida de Fourier (FFT), quantifica essas diferenças, como demonstrado na figura (2). Para os sinais normais (subplot inferior), o espectro de frequência exibe um único pico proeminente e bem definido em 50 Hz, correspondente à frequência fundamental. As demais componentes de frequência apresentam amplitudes virtualmente nulas, confirmando a natureza monofrequencial desses sinais. Em contraste acentuado, o espectro dos sinais anormais (subplot superior) mostra, além da componente fundamental em 50 Hz, uma série de picos significativos em frequências

múltiplas ímpares da fundamental. Observam-se claramente componentes harmônicas em 150 Hz (3ª harmônica), 250 Hz (5ª harmônica), 350 Hz (7ª harmônica) e em frequências superiores, com amplitudes que tendem a decrescer à medida que a ordem harmônica aumenta. Esta assinatura espectral é característica de sinais com distorção harmônica.

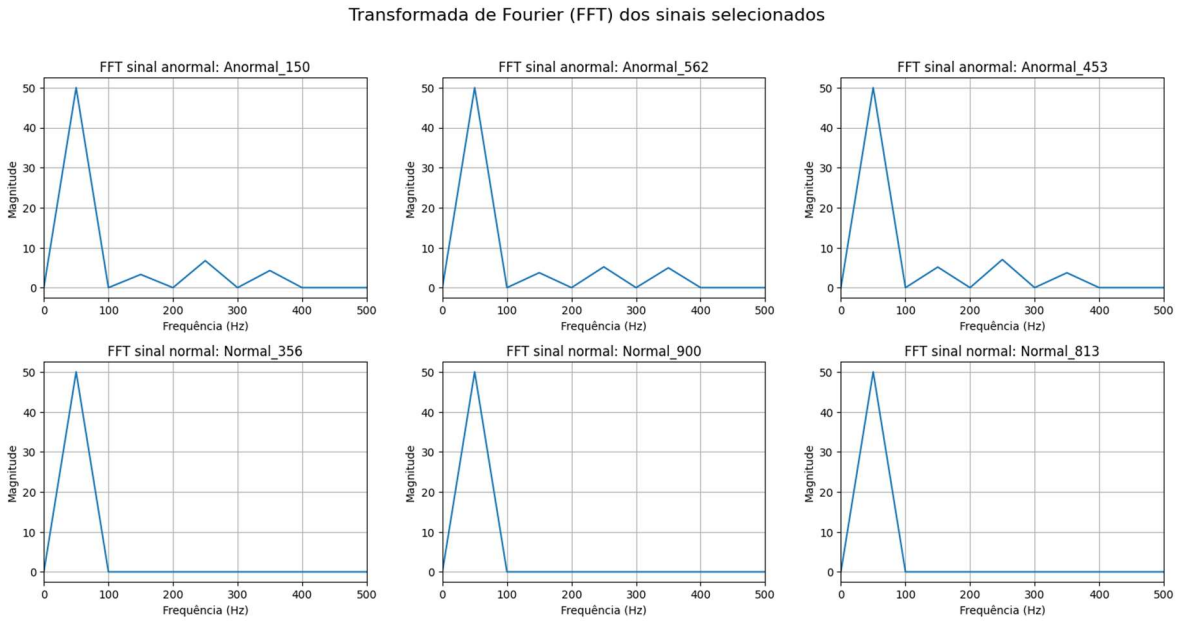

Figura 2: FFT dos sinais selecionados

Finalmente, a figura (3) demonstra o impacto da aplicação de um filtro Butterworth passa-baixas de 4ª ordem, com frequência de corte em 300 Hz, indicada pela linha tracejada vermelha. Observa-se que o ganho do filtro é unitário (ou próximo de 0 dB) para frequências significativamente abaixo de 300 Hz, e começa a atenuar as componentes de frequência à medida que se aproximam e ultrapassam este valor de corte, com uma atenuação progressivamente maior para frequências mais altas. Esta característica é consistente com o objetivo de atenuar harmônicas de ordem superior à 5ª (considerando uma fundamental de 50 Hz), enquanto preserva as componentes de frequência mais baixas.

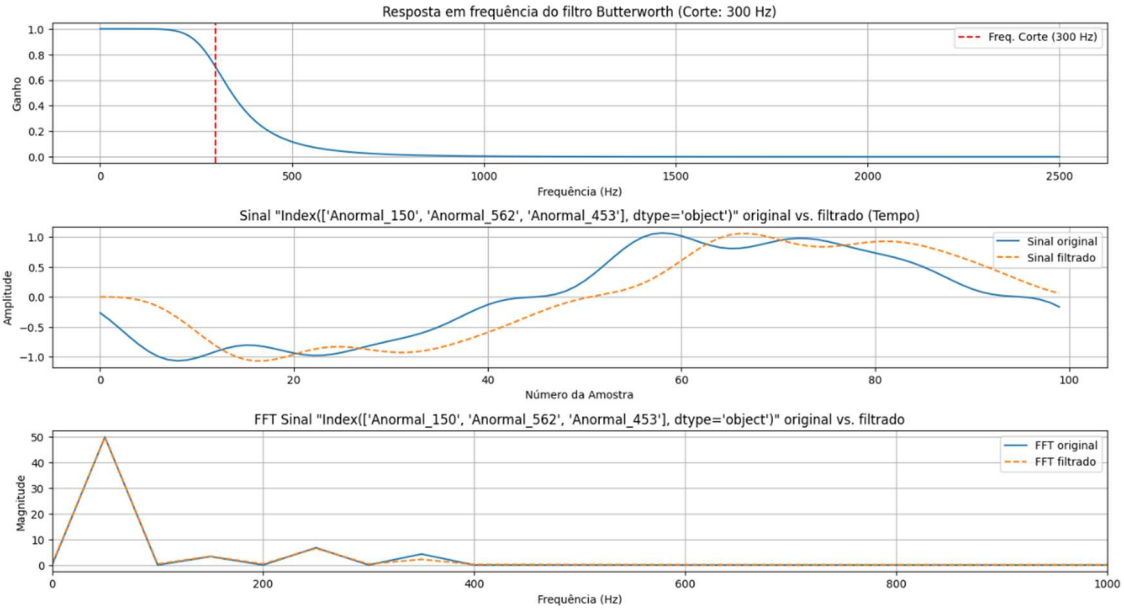

Figura 3: Aplicação do filtro digital

O segundo subplot da figura (3) apresenta a comparação do sinal anormal original e do sinal filtrado no domínio do tempo. O sinal original exibe a distorção característica causada pelas múltiplas harmônicas. Após a aplicação do filtro (que utiliza sig.lfilter, um filtro causal de passada única), o sinal filtrado demonstra uma forma de onda alterada. Espera-se que as ondulações de alta frequência, correspondentes às harmônicas de ordem mais elevada (como a 7ª harmônica em 350 Hz e superiores), sejam atenuadas, resultando em um sinal visualmente mais suave nessas componentes. No entanto, como a frequência de corte de 300 Hz permite a passagem da fundamental (50 Hz), da 3ª harmônica (150 Hz) e da 5ª harmônica (250 Hz), o sinal filtrado ainda reterá características não senoidais devido a essas harmônicas de ordem inferior. Pode-se também notar um pequeno deslocamento de fase (atraso) no sinal filtrado em relação ao original, uma característica da aplicação de filtros IIR com lfilter.

O impacto da filtragem no conteúdo espectral é claramente visualizado no terceiro subplot da Figura 3, que compara o espectro de frequência (FFT) do sinal anormal antes e depois da filtragem, com o eixo de frequência limitado a 1000 Hz para melhor detalhamento das componentes de interesse. No espectro do sinal original, a componente fundamental em 50 Hz é acompanhada por uma série de harmônicas (150 Hz, 250 Hz, 350 Hz, etc.). Após a filtragem, o espectro do sinal filtrado mostra que a componente fundamental (50 Hz), a 3ª harmônica (150 Hz) e a 5ª harmônica (250 Hz) são largamente preservadas, pois estão abaixo da frequência de corte de 300 Hz. Em contraste, as componentes harmônicas de ordem superior, como a 7ª

harmônica (350 Hz) e as subsequentes, apresentam uma atenuação significativa, conforme esperado pela ação do filtro passa-baixas.

4. Conclusões

A análise e caracterização de sinais elétricos senoidais puros e com distorção harmônica foram conduzidas utilizando técnicas fundamentais de processamento digital de sinais. A visualização no domínio do tempo, a aplicação da Transformada Rápida de Fourier (FFT) para análise espectral e a implementação de um filtro digital Butterworth passa-baixas permitiram identificar as diferenças entre os sinais e demonstrar um método para atenuar componentes indesejadas.

Os resultados obtidos mostraram que, enquanto a análise temporal oferece uma percepção visual da distorção da forma de onda, a FFT se revelou uma ferramenta quantitativa crucial para identificar a presença e a magnitude das diversas componentes harmônicas nos sinais anormais. Especificamente, foram observadas harmônicas ímpares como a 3ª (150 Hz) e a 5ª (250 Hz), entre outras, em contraste com o espectro limpo dos sinais senoidais puros, que continham apenas a componente fundamental de 50 Hz. A aplicação do filtro digital Butterworth de 4ª ordem, com frequência de corte em 300 Hz, demonstrou eficácia em atenuar seletivamente as harmônicas de ordem mais elevada (a partir da 7ª), preservando as componentes de frequência mais baixas, como a fundamental e as harmônicas de 3ª e 5ª ordem.

As técnicas de processamento de sinais aqui aplicadas possuem grande relevância e constituem uma etapa essencial de pré-processamento para sistemas de aprendizado de máquina. A capacidade de isolar, identificar e mitigar ruídos ou distorções, como as harmônicas, é fundamental para a extração de atributos significativos dos sinais. Esses atributos, por sua vez, alimentam algoritmos de machine learning destinados a tarefas como detecção automática de falhas, classificação de distúrbios de qualidade de energia ou monitoramento de condição de equipamentos.

Referências

KHAN, Muhammad Umar; AZIZ, Sumair; USMAN, Adil. XPQRS: Expert power quality recognition system for sensitive load applications. **Measurement**, v. 216, p. 112889, 2023.

Anexo I – código completo em Python

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from scipy import signal as sig

# --- Parâmetros globais ---
ARQUIVO_NORMAL = 'Dataset_Power Quality Disturbance/Pure_Sinusoidal.csv'
ARQUIVO_ANORMAL = 'Dataset_Power Quality Disturbance/Harmonics.csv'
N_AMOSTRAS = 100            # Número de amostras por sinal
FREQ_AMOSTRAGEM = 5000      # Frequência de amostragem em Hz (assumindo
fundamental de 50Hz e 100 amostras/ciclo)
FREQ = 50                   # Hz
N_SINAIS_PARA_PLOTAR = 3    # Número de sinais de cada classe para plotar

# --- 1. Organização dos Dados ---
def carregar_e_organizar_dados(arquivo_normal, arquivo_anormal):
    # Organização dos sinais em dataframes de modo que as N primeiras
colunas/posições sejam ocupadas pela classe "Anormal" e as N seguintes pela
classe "Normal"
    # Carrega os dados dos arquivos CSV, transpõe e concatena os dataframes
    try:
        # Carregar dados da classe anormal (Harmonics)
        df_anormal = pd.read_csv(arquivo_anormal, header=None)
        # Carregar dados da classe normal (Pure_Sinusoidal)
        df_normal = pd.read_csv(arquivo_normal, header=None)
    except FileNotFoundError as e:
        print(f"Erro: arquivo não encontrado: {e}")
        return None, 0, 0

    # Transpor os dataframes (sinais como colunas, amostras como linhas)
    df_anormal_T = df_anormal.T
    df_normal_T = df_normal.T

    num_sinais_anormais = df_anormal_T.shape[1]
    num_sinais_normais = df_normal_T.shape[1]

    print(f"Número de sinais 'anormais' ({arquivo_anormal}):
{num_sinais_anormais}")
    print(f"Número de sinais 'normais' ({arquivo_normal}):
{num_sinais_normais}")
    print(f"Número de amostras por sinal: {df_anormal_T.shape}")

    # Renomear colunas para facilitar identificação
```

```
    df_anormal_T.columns = [f'Anormal_{i+1}' for i in
range(num_sinais_anormais)]
    df_normal_T.columns = [f'Normal_{i+1}' for i in range(num_sinais_normais)]

    # Concatenar: anormais primeiro, depois normais
    df_combinado = pd.concat([df_anormal_T, df_normal_T], axis=1)

    print(f"\nDimensões do dataframe anormal transposto:
{df_anormal_T.shape}")
    print(f"Dimensões do dataframe normal transposto: {df_normal_T.shape}")
    print(f"Dimensões do dataframe combinado: {df_combinado.shape}")

    return df_combinado, num_sinais_anormais, num_sinais_normais

# --- 2. Visualização no domínio do tempo ---
def visualizar_sinais_tempo(df, num_sinais_anormais, num_sinais_normais,
n_plot=3):
    # Escolha aleatória de três sinais de cada classe e visualização dos
mesmos no domínio do tempo (isto é, obtenção dos gráficos temporais dos
sinais)
    if df is None:
        return pd.DataFrame(), pd.DataFrame() # Retornar dataFrames vazios
caso haja algum problema

    # Verificar se há sinais suficientes para plotar
    n_plot_anormais = min(n_plot, num_sinais_anormais)
    n_plot_normais = min(n_plot, num_sinais_normais)

    if n_plot_anormais == 0 and n_plot_normais == 0:
        print("Não há sinais suficientes para plotar.")
        return pd.DataFrame(), pd.DataFrame()

    sinais_anormais_plot = pd.DataFrame()
    sinais_normais_plot = pd.DataFrame()

    if n_plot_anormais > 0:
        # Selecionar índices aleatórios para sinais anormais
        indices_anormais_plot = np.random.choice(num_sinais_anormais,
n_plot_anormais, replace=False)
        sinais_anormais_plot = df.iloc[:, indices_anormais_plot]

    if n_plot_normais > 0:
        # Selecionar índices aleatórios para sinais normais
        indices_normais_plot = np.random.choice(num_sinais_normais,
n_plot_normais, replace=False)
        sinais_normais_plot = df.iloc[:, num_sinais_anormais +
indices_normais_plot]

    tempo = np.arange(N_AMOSTRAS) # Vetor de amostras (0 a N_AMOSTRAS-1)
```

```
    # Ajustar o número de colunas do subplot dinamicamente
    num_cols_subplot = max(n_plot_anormais, n_plot_normais)
    if num_cols_subplot == 0: # Caso não haja sinais para plotar
        return sinais_anormais_plot, sinais_normais_plot

    plt.figure(figsize=(4 * num_cols_subplot, 8)) # Ajustar tamanho da figura
    plt.suptitle('Sinais selecionados no domínio do tempo', fontsize=16)

    for i in range(n_plot_anormais):
        # Plotar sinais anormais
        plt.subplot(2, num_cols_subplot, i + 1)
        plt.plot(tempo, sinais_anormais_plot.iloc[:, i])
        plt.title(f'Sinal anormal: {sinais_anormais_plot.columns[i]}')
        plt.xlabel('Número da amostra')
        plt.ylabel('Amplitude')
        plt.grid(True)

    for i in range(n_plot_normais):
        # Plotar sinais normais
        plt.subplot(2, num_cols_subplot, i + 1 + num_cols_subplot)
        plt.plot(tempo, sinais_normais_plot.iloc[:, i])
        plt.title(f'Sinal normal: {sinais_normais_plot.columns[i]}')
        plt.xlabel('Número da amostra')
        plt.ylabel('Amplitude')
        plt.grid(True)

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()

    return sinais_anormais_plot, sinais_normais_plot

# --- 3. Emprego e visualização da Transformada de Fourier (FFT) ---
def analisar_sinais_frequencia(sinais_anormais, sinais_normais, fs, n_samples,
n_plot=3):
    # Emprego da Transformada de Fourier (FFT) para os sinais escolhidos na
etapa anterior e visualização do espectro de frequências do mesmo
    if sinais_anormais.empty and sinais_normais.empty:
        print("Nenhum sinal encontrado.")
        return

    n_plot_anormais = sinais_anormais.shape[1]
    n_plot_normais = sinais_normais.shape[1]
    num_cols_subplot = max(n_plot_anormais, n_plot_normais)

    if num_cols_subplot == 0:
        return

    plt.figure(figsize=(4 * num_cols_subplot, 8)) # Ajustar tamanho da figura
```

```
    plt.suptitle('Transformada de Fourier (FFT) dos sinais selecionados',
fontsize=16)

    # Frequências para o eixo X da FFT (apenas positivas)
    freq_axis = fftfreq(n_samples, 1/fs)[:n_samples//2]

    for i in range(n_plot_anormais):
        # FFT para sinais anormais
        sinal_anormal_atual = sinais_anormais.iloc[:, i]
        fft_anormal = fft(sinal_anormal_atual.values)
        magnitude_anormal = np.abs(fft_anormal)[:n_samples//2] # Pegar apenas
a metade positiva

        plt.subplot(2, num_cols_subplot, i + 1)
        plt.plot(freq_axis, magnitude_anormal)
        plt.title(f'FFT sinal anormal: {sinais_anormais.columns[i]}')
        plt.xlabel('Frequência (Hz)')
        plt.ylabel('Magnitude')
        plt.grid(True)
        plt.xlim(0, fs/10) # Limitar eixo X para melhor visualização (ex: até
500 Hz se FS=5000)

    for i in range(n_plot_normais):
        # FFT para sinais normais
        sinal_normal_atual = sinais_normais.iloc[:, i]
        fft_normal = fft(sinal_normal_atual.values)
        magnitude_normal = np.abs(fft_normal)[:n_samples//2] # Pegar apenas a
metade positiva

        plt.subplot(2, num_cols_subplot, i + 1 + num_cols_subplot)
        plt.plot(freq_axis, magnitude_normal)
        plt.title(f'FFT sinal normal: {sinais_normais.columns[i]}')
        plt.xlabel('Frequência (Hz)')
        plt.ylabel('Magnitude')
        plt.grid(True)
        plt.xlim(0, fs/10) # Limitar eixo X

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()

# --- 4. Filtragem digital (filtro IIR Butterworth Passa-Baixas) ---
def aplicar_e_visualizar_filtro(nome_sinal, sinais_originais, freq,
n_amostras):
    # Realização da filtragem digital dos sinais e visualização das respostas
em frequência dos mesmos
    sinal_original = sinais_originais.values

    # Parâmetros do filtro
    ordem_filtro = 4
```

```
    freq_corte = 300            # Hz (atenuar harmônicas acima da 5a para
50Hz)
    nyquist_freq = 0.5 * freq
    freq_corte_normalizada = freq_corte / nyquist_freq

    # Projetar o filtro Butterworth passa-baixas
    b, a = sig.butter(ordem_filtro, freq_corte_normalizada, btype='low',
analog=False)

    # Aplicar o filtro
    sinal_filtrado = sig.lfilter(b, a, sinal_original)

    # Visualizar resposta em frequência do filtro
    w, h = sig.freqz(b, a, worN=8000, fs=freq)
    plt.figure(figsize=(9.6, 8)) # Ajustar tamanho da figura
    plt.subplot(3, 1, 1)
    plt.plot(w, np.abs(h))
    plt.title(f'Resposta em frequência do filtro Butterworth (Corte:
{freq_corte} Hz)')
    plt.xlabel('Frequência (Hz)')
    plt.ylabel('Ganho')
    plt.grid(True)
    plt.axvline(freq_corte, color='red', linestyle='--', label=f'Freq. Corte
({freq_corte} Hz)')
    plt.legend()

    # Visualizar sinal original vs. filtrado no domínio do tempo
    tempo = np.arange(n_amostras)
    plt.subplot(3, 1, 2)
    plt.plot(tempo, sinal_original, label='Sinal original')
    plt.plot(tempo, sinal_filtrado, label='Sinal filtrado', linestyle='--')
    plt.title(f'Sinal "{nome_sinal}" original vs. filtrado (Tempo)')
    plt.xlabel('Número da Amostra')
    plt.ylabel('Amplitude')
    plt.legend()
    plt.grid(True)

    # Visualizar FFT do sinal original vs. filtrado
    freq_axis = fftfreq(n_amostras, 1/freq)[:n_amostras//2]

    fft_original = fft(sinal_original)
    magnitude_original = np.abs(fft_original)[:n_amostras//2]

    fft_filtrado = fft(sinal_filtrado)
    magnitude_filtrado = np.abs(fft_filtrado)[:n_amostras//2]

    plt.subplot(3, 1, 3)
    plt.plot(freq_axis, magnitude_original, label='FFT original')
```

```
    plt.plot(freq_axis, magnitude_filtrado, label='FFT filtrado', linestyle='-
-')
    plt.title(f'FFT Sinal "{nome_sinal}" original vs. filtrado')
    plt.xlabel('Frequência (Hz)')
    plt.ylabel('Magnitude')
    plt.legend()
    plt.grid(True)
    plt.xlim(0, freq/5) # Limitar eixo X para melhor visualização

    plt.tight_layout()
    plt.show()

# --- Execução Principal ---
if __name__ == '__main__':
    print("--- Iniciando pré-processamento de Sinais ---")

    # 1. Organização dos Dados
    print("\n--- 1. Organização dos dados ---")
    df_combinado, num_anormais, num_normais =
carregar_e_organizar_dados(ARQUIVO_NORMAL, ARQUIVO_ANORMAL)

    if df_combinado is not None:
        # 2. Visualização no domínio do tempo
        print(f"\n--- 2. Visualização de até {N_SINAIS_PARA_PLOTAR} Sinais
Aleatórios no Domínio do Tempo ---")
        sinais_anormais_selecionados, sinais_normais_selecionados =
visualizar_sinais_tempo(
            df_combinado, num_anormais, num_normais,
n_plot=N_SINAIS_PARA_PLOTAR
        )

        # 3. Transformada de Fourier (FFT)
        if not sinais_anormais_selecionados.empty or not
sinais_normais_selecionados.empty:
            print(f"\n--- 3. Emprego e visualização da Transformada de Fourier
(FFT) ---")
            analisar_sinais_frequencia(
                sinais_anormais_selecionados, sinais_normais_selecionados,
                FREQ_AMOSTRAGEM, N_AMOSTRAS, n_plot=N_SINAIS_PARA_PLOTAR
            )

            # 4. Filtragem digital (IIR Butterworth Passa-Baixas)
            if not sinais_anormais_selecionados.empty:
                print("\n--- 4. Filtragem digital (IIR Butterworth Passa-
Baixas) ---")
                sinal_anormal_nome = sinais_anormais_selecionados.columns
                sinal_anormal_dados = sinais_anormais_selecionados.iloc[:, 0]
                aplicar_e_visualizar_filtro(sinal_anormal_nome,
sinal_anormal_dados, FREQ_AMOSTRAGEM, N_AMOSTRAS)
```

```
            else:
                print("\nNenhum sinal anormal selecionado para demonstrar a
filtragem.")
        else:
            print("\nNão foi possível selecionar sinais para análise de
frequência e filtragem.")
    else:
        print("\nProcessamento interrompido devido a erro no carregamento dos
dados.")

    print("\n--- Pré-processamento Concluído ---")
```