LabAuto — Laboratório de Automação Inteligente
CDSA/UFCG · Sumé–PB

Processamento de Sinais

Transformadas, Filtragem Digital e Análise Wavelet

CDSA/UFCG · Sumé–PB · 2025 · Engenharia de Biossistemas

Sumário

Capítulo 1 — Sinais e Sistemas

O estudo de sinais e sistemas constitui a base do processamento de sinais. Um sinal é uma grandeza física que varia no tempo, no espaço ou em ambos, e que carrega informação. Exemplos incluem sinais de áudio, tensão elétrica em sensores, imagens capturadas por drones e séries temporais de temperatura do solo. Um sistema é qualquer processo que transforma um sinal de entrada em um sinal de saída.

1.1 Classificação de sinais

Sinais podem ser classificados em diversas categorias de acordo com suas propriedades fundamentais. A compreensão dessas classificações é essencial para escolher as técnicas adequadas de análise e processamento.

Contínuos vs. Discretos

Um sinal de tempo contínuo x(t) é definido para todo instante t no eixo real. Um sinal de tempo discreto x[n] é definido apenas em instantes inteiros n = ..., -1, 0, 1, 2, ... A conversão de contínuo para discreto é feita pela amostragem, que coleta valores do sinal a intervalos regulares Ts (período de amostragem), resultando em x[n] = x(n Ts).

Na prática, sinais de sensores são originalmente contínuos (tensão elétrica, temperatura, pressão), mas são convertidos para discretos pelo conversor analógico-digital (A/D). A taxa de amostragem fs = 1/Ts determina a fidelidade da representação digital.

Determinísticos vs. Estocásticos

Sinais determinísticos podem ser descritos por uma expressão matemática exata: dado qualquer instante t, o valor do sinal é perfeitamente previsível. Exemplo: x(t) = A cos(2 pi f0 t). Sinais estocásticos (aleatórios) possuem componentes imprevisíveis e são caracterizados por propriedades estatísticas como média, variância e função de autocorrelação. Ruído em sensores é um exemplo clássico de sinal estocástico.

Na realidade, a maioria dos sinais é uma combinação de componente determinística (informação) e componente estocástica (ruído). O objetivo do processamento de sinais é, frequentemente, extrair a componente determinística e reduzir a estocástica.

Periódicos vs. Aperiódicos

Um sinal é periódico se existir um período T > 0 tal que x(t) = x(t + T) para todo t. O menor valor de T é o período fundamental e f0 = 1/T é a frequência fundamental. Sinais que não satisfazem essa condição são aperiódicos. Sinais biológicos como ECG são quasi-periódicos: possuem padrão repetitivo, mas com variações batimento a batimento.

Outras classificações

- Causal vs. Não-causal: Um sinal causal é nulo para t < 0 (existe apenas a partir de um instante inicial).
- Par vs. Ímpar: x(t) é par se x(t) = x(-t); ímpar se x(t) = -x(-t). Todo sinal pode ser decomposto em par + ímpar.

- De energia vs. De potência: Sinais de energia têm energia total finita; sinais de potência têm potência média finita (sinais periódicos).

1.2 Operações básicas com sinais

As operações elementares sobre sinais são ferramentas fundamentais para análise e manipulação:

- Deslocamento temporal: y[n] = x[n - n0]. Se n0 > 0, o sinal é atrasado; se n0 < 0, o sinal é adiantado.
- Escalonamento (reflexão): y[n] = x[-n] inverte o sinal no eixo temporal. No caso contínuo, y(t) = x(at) comprime (|a|>1) ou expande (|a|<1) o sinal.
- Soma: y[n] = x1[n] + x2[n]. Combina dois sinais ponto a ponto.
- Produto: y[n] = x1[n] . x2[n]. Usado em modulação, janelamento e técnicas de demodulação.
- Escalonamento em amplitude: y[n] = A x[n]. Amplifica (A>1) ou atenua (A<1) o sinal.

Essas operações formam a base para construção de sistemas mais complexos. Por exemplo, a modulação AM (amplitude) é realizada pelo produto de um sinal de informação com uma portadora senoidal.

1.3 Sistemas lineares e invariantes no tempo (LIT)

Um sistema LIT (ou LTI — Linear Time-Invariant) é aquele que satisfaz duas propriedades:

- Linearidade (superposição): Se x1[n] gera y1[n] e x2[n] gera y2[n], então a x1[n] + b x2[n] gera a y1[n] + b y2[n].
- Invariância no tempo: Se x[n] gera y[n], então x[n - n0] gera y[n - n0]. O comportamento do sistema não muda ao longo do tempo.

A grande vantagem dos sistemas LIT é que são completamente caracterizados pela sua resposta ao impulso h[n], que é a saída do sistema quando a entrada é o impulso unitário d[n]. A saída para qualquer entrada x[n] é obtida pela convolução: y[n] = x[n] * h[n].

Propriedades adicionais relevantes incluem causalidade (h[n] = 0 para n < 0, a saída depende apenas de entradas presentes e passadas) e estabilidade BIBO (entrada limitada produz saída limitada, garantida se a soma dos valores absolutos de h[n] converge).

No domínio da frequência, um sistema LIT é descrito pela sua função de transferência H(z), que é a transformada Z da resposta ao impulso. Para filtros digitais, H(z) é uma função racional em z, e seus polos e zeros determinam o comportamento do sistema.

1.4 Convolução discreta

A convolução discreta é a operação fundamental que relaciona entrada, resposta ao impulso e saída em sistemas LIT. Sua definição formal é:

```
y[n] = sum_k x[k] . h[n - k]
```

Para sinais de duração finita, a soma se reduz a um número finito de termos. Se x tem comprimento N e h tem comprimento M, a saída y terá comprimento N + M - 1.

Exemplo numérico:

Sejam x = [1, 2, 3] e h = [1, 1]:

- y[0] = x[0] h[0] = 1 . 1 = 1
- y[1] = x[0] h[1] + x[1] h[0] = 1 + 2 = 3
- y[2] = x[1] h[1] + x[2] h[0] = 2 + 3 = 5
- y[3] = x[2] h[1] = 3

Resultado: y = [1, 3, 5, 3]

A convolução pode ser calculada eficientemente no domínio da frequência usando a propriedade de que convolução no tempo equivale a multiplicação na frequência: Y(z) = X(z) . H(z). Para sinais longos, o método de sobreposição (overlap-add ou overlap-save) combina FFT com convolução para reduzir o custo computacional.

1.5 Representação espectral

A ideia de representação espectral é que qualquer sinal pode ser decomposto em uma soma (possivelmente infinita) de senoides de diferentes frequências, amplitudes e fases. Essa decomposição permite analisar o sinal no domínio da frequência em vez do domínio do tempo.

No domínio da frequência, operações como filtragem tornam-se intuitivas: um filtro passa-baixa simplesmente remove as componentes de alta frequência. Essa dualidade tempo-frequência é o conceito central do processamento de sinais e será formalizada no capítulo seguinte com a Transformada de Fourier.

O espectro de magnitude |X(f)| mostra a amplitude de cada frequência presente no sinal. O espectro de fase mostra a defasagem de cada componente. Juntos, eles constituem a representação completa do sinal na frequência.

> Conceito-chave
> O espectro de um sinal revela quais frequências estão presentes e com que intensidade. Um sinal senoidal puro tem apenas uma frequência; um sinal de onda quadrada possui infinitos harmônicos ímpares. Sinais reais contêm tipicamente uma faixa contínua de frequências.

Capítulo 2 — Série e Transformada de Fourier

A análise de Fourier é o pilar central do processamento de sinais. Desenvolvida por Jean-Baptiste Joseph Fourier no início do século XIX, a teoria estabelece que sinais periódicos podem ser decompostos em somas de senoides (séries de Fourier) e sinais aperiódicos podem ser representados por integrais de senoides (transformadas de Fourier).

2.1 Série de Fourier

Qualquer sinal periódico x(t) com período T pode ser expresso como uma soma infinita de funções senoidais harmonicamente relacionadas. Na forma exponencial complexa:

```
x(t) = sum_{k} c_k . exp(j 2 pi k f0 t)
```

onde f0 = 1/T é a frequência fundamental e os coeficientes são calculados por:

```
c_k = (1/T) . integral_0^T x(t) . exp(-j 2 pi k f0 t) dt
```

Na forma trigonométrica equivalente: x(t) = a0/2 + sum [a_k cos(2 pi k f0 t) + b_k sin(2 pi k f0 t)]. O coeficiente c0 = a0/2 representa o valor médio (componente DC) do sinal. Os harmônicos de ordem k representam componentes na frequência k f0.

A série de Fourier permite, por exemplo, analisar a composição harmônica de um sinal de ECG periódico, identificando as contribuições de cada frequência para a forma do complexo QRS.

Convergência: A série de Fourier converge para x(t) em pontos de continuidade (condições de Dirichlet). Em pontos de descontinuidade, converge para a média dos limites laterais. O fenômeno de Gibbs produz oscilações (~9%) próximo a descontinuidades, mesmo com muitos termos na soma.

2.2 Transformada de Fourier Contínua (TFC)

Para sinais aperiódicos de tempo contínuo, a Transformada de Fourier Contínua generaliza a série de Fourier, substituindo a soma discreta por uma integral:

```
X(f) = integral_{-inf}^{inf} x(t) . exp(-j 2 pi f t) dt
                    (Transformada)

x(t) = integral_{-inf}^{inf} X(f) . exp(j 2 pi f t) df (Inversa)
```

Propriedades principais:

- Linearidade: F{a x(t) + b y(t)} = a X(f) + b Y(f)
- Deslocamento temporal: F{x(t - t0)} = X(f) exp(-j 2 pi f t0)
- Convolução: F{x(t) * h(t)} = X(f) H(f). A convolução no tempo corresponde à multiplicação na frequência — esta é uma das propriedades mais úteis.
- Teorema de Parseval: A energia total no tempo é igual à energia total na frequência: integral $|x(t)|^2$ dt = integral $|X(f)|^2$ df.
- Dualidade: Se x(t) tem transformada X(f), então X(t) tem transformada x(-f).
- Modulação: Multiplicar por exp(j 2 pi f0 t) desloca o espectro por f0. Base de sistemas de comunicação.

2.3 Transformada de Fourier de Tempo Discreto (DTFT)

Para sinais de tempo discreto x[n], a DTFT é definida como:

$$X(e^{jw}) = \sum_{n} x[n] \cdot e^{-j w n}$$

onde w = 2 pi f / fs é a frequência angular normalizada. A DTFT produz um espectro contínuo e periódico em w com período 2 pi. Isso reflete o fato de que, para sinais discretos, frequências separadas por múltiplos da frequência de amostragem são indistinguíveis.

A DTFT é fundamental na teoria, mas, por produzir um espectro contínuo, não pode ser computada diretamente por algoritmos digitais — para isso, utiliza-se a DFT.

A relação entre DTFT e TFC é mediada pela amostragem: a DTFT de x[n] = x(n Ts) é a réplica periódica do espectro contínuo X(f), com período fs = 1/Ts.

2.4 Transformada Discreta de Fourier (DFT)

A DFT amostra o espectro da DTFT em N pontos igualmente espaçados, resultando em uma representação discreta tanto no tempo quanto na frequência:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j 2 \pi k n / N}, \quad k = 0, 1, \ldots, N-1$$

$$x[n] = (1/N) \cdot \sum_{k=0}^{N-1} X[k] \cdot e^{j 2 \pi k n / N} \quad \text{(IDFT)}$$

Cada amostra espectral X[k] corresponde à frequência fk = k fs / N, onde fs é a frequência de amostragem. A resolução em frequência é Df = fs / N. Para melhorar a resolução, aumenta-se N (mais amostras ou zero-padding).

O zero-padding (adicionar zeros ao final do sinal) interpola o espectro, dando uma curva mais suave, mas não melhora a resolução real — apenas a resolução aparente.

> Interpretação prática
> |X[k]| indica a amplitude da componente na frequência fk; o argumento de X[k] indica a fase. O espectro é simétrico para sinais reais: X[k] = X*[N-k]. Apenas as frequências de 0 a fs/2 contêm informação independente.

2.5 Algoritmo FFT (Fast Fourier Transform)

A FFT não é uma transformada diferente da DFT — é um algoritmo eficiente para calculá-la. O cálculo direto da DFT requer $O(N^2)$ multiplicações complexas. O algoritmo de Cooley-Tukey (1965) reduz para O(N log N) usando o princípio de decimação no tempo.

A ideia fundamental é dividir uma DFT de N pontos em duas DFTs de N/2 pontos — uma para as amostras pares e outra para as ímpares — e combinar os resultados usando as propriedades de simetria do fator twiddle W_N = exp(-j 2 pi / N):

$$X[k] = G[k] + W_N^k \cdot H[k] \quad (k = 0, \ldots, N/2 - 1)$$

$$X[k + N/2] = G[k] - W_N^k \cdot H[k]$$

onde G[k] é a DFT das amostras pares e H[k] das ímpares. Esse processo é aplicado recursivamente até DFTs de 2 pontos (operação butterfly). A restrição clássica é que N seja potência de 2 (versões mais gerais existem para N arbitrário).

Em termos práticos, para N = 1024 pontos, a DFT direta exige ~1 milhão de operações; a FFT precisa de apenas ~10.000 — uma melhoria de 100x. Para N = 1.000.000, a melhoria chega a

50.000x. Essa eficiência tornou a análise espectral viável em tempo real.

N	DFT (N^2)	FFT (N log N)	Ganho
64	4.096	384	~11x
256	65.536	2.048	~32x
1.024	1.048.576	10.240	~102x
65.536	4.294.967.296	1.048.576	~4.096x

Tabela 2.1 — Comparação de operações: DFT direta vs. FFT

2.6 Exemplo prático: espectro de frequências em Python

O exemplo a seguir gera um sinal simulando dados de um sensor de temperatura com ruído e aplica a FFT para identificar as componentes de frequência:

```
import numpy as np
import matplotlib.pyplot as plt

# Parâmetros do sinal
fs = 100            # Frequência de amostragem (Hz)
T = 2.0             # Duração (s)
N = int(fs * T)     # Número de amostras
t = np.linspace(0, T, N, endpoint=False)

# Sinal: variação lenta (0.5 Hz) + oscilação (10 Hz) + ruído
x = 25 + 3*np.sin(2*np.pi*0.5*t) + 0.5*np.sin(2*np.pi*10*t)
x += 0.3 * np.random.randn(N)  # Ruído gaussiano

# Cálculo da FFT
X = np.fft.fft(x)
freqs = np.fft.fftfreq(N, d=1/fs)

# Espectro de magnitude (lado positivo apenas)
mask = freqs >= 0
magnitude = 2.0/N * np.abs(X[mask])
magnitude[0] /= 2  # Componente DC não se duplica

# Visualização
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 6))
ax1.plot(t, x, linewidth=0.7)
ax1.set_xlabel('Tempo (s)')
ax1.set_ylabel('Temperatura (°C)')
ax1.set_title('Sinal de Sensor de Temperatura')

ax2.stem(freqs[mask], magnitude, markerfmt='.')
ax2.set_xlabel('Frequência (Hz)')
ax2.set_ylabel('Amplitude')
ax2.set_title('Espectro de Frequência (FFT)')
ax2.set_xlim([0, 20])

plt.tight_layout()
plt.savefig('espectro_temperatura.png', dpi=150)
plt.show()
```

Código 2.1 — Análise espectral de sinal de sensor de temperatura usando numpy.fft

Neste exemplo, os picos no espectro aparecerão em 0 Hz (valor DC = 25 C), 0,5 Hz (variação lenta de +/- 3 C) e 10 Hz (oscilação de +/- 0,5 C). O ruído distribuirá energia de forma uniforme pelo espectro (ruído branco).

Para aplicações práticas no LabAuto, esse mesmo procedimento pode ser aplicado a dados reais de sensores de temperatura, umidade ou vibração, permitindo identificar frequências de interesse (ciclo diurno, frequência de rotação, etc.) e componentes de ruído.

Capítulo 3 — Filtragem Digital

A filtragem digital é o processo de modificar um sinal para enfatizar ou atenuar determinadas componentes de frequência. Filtros digitais são implementados por algoritmos que operam sobre amostras discretas e são fundamentais em aplicações de sensores, comunicação, áudio e biomedicina.

3.1 Resposta em frequência

A resposta em frequência H(exp(jw)) de um sistema LIT discreto descreve como o sistema modifica cada componente de frequência do sinal de entrada. É a DTFT da resposta ao impulso h[n]:

```
H(exp(jw)) = sum_n h[n] . exp(-j w n)
```

O módulo |H(exp(jw))| indica o ganho (amplificação ou atenuação) para cada frequência; a fase indica o atraso de fase introduzido. O diagrama de Bode apresenta ambas as curvas: ganho em dB (20 log10 |H|) vs. frequência e fase em graus vs. frequência.

Para filtros digitais descritos por equações a diferenças, a resposta em frequência pode ser avaliada substituindo z = exp(jw) na função de transferência H(z). Em Python, scipy.signal.freqz() calcula a resposta em frequência a partir dos coeficientes do filtro.

3.2 Tipos de filtros

Os filtros são classificados pela região de frequências que permitem passar:

Tipo	Descrição	Aplicação típica
Passa-baixa	Permite f abaixo de fc	Suavização de sinais, anti-aliasing
Passa-alta	Permite f acima de fc	Remoção de offset DC, detecção de bordas
Passa-banda	Permite faixa [f1, f2]	Isolamento de QRS no ECG (5-15 Hz)
Rejeita-banda	Bloqueia faixa [f1, f2]	Remoção de ruído de 60 Hz (rede)

A frequência de corte fc é definida como a frequência na qual o ganho cai para -3 dB (aproximadamente 70,7% da amplitude máxima). A taxa de atenuação na banda de transição é medida em dB/oitava ou dB/década. Um filtro ideal tem transição abrupta; filtros reais têm uma banda de transição de largura finita.

A ordem do filtro determina a inclinação na banda de transição: maior ordem = transição mais abrupta, mas também maior atraso e complexidade computacional.

3.3 Filtros FIR (Resposta ao Impulso Finita)

Filtros FIR possuem resposta ao impulso de duração finita — h[n] = 0 para n < 0 e n >= M. A equação a diferenças é:

```
y[n] = b0 x[n] + b1 x[n-1] + ... + b_{M-1} x[n-M+1]
```

Os coeficientes bk são iguais aos valores da resposta ao impulso: bk = h[k]. A função de transferência é: H(z) = b0 + b1 z^{-1} + ... + b_{M-1} $z^{-(M-1)}$ (apenas zeros, sem polos além da origem).

Vantagens dos filtros FIR: sempre estáveis (não possuem polos fora da origem), podem ter fase linear exata (essencial quando a forma de onda deve ser preservada, como em ECG), e são simples de implementar em hardware (apenas multiplicações e somas).

Projeto por janelamento:

O método de janelamento parte do filtro ideal (resposta ao impulso infinita obtida pela IDTFT de H_ideal(exp(jw))) e o trunca multiplicando por uma janela de comprimento finito. A escolha da janela afeta o compromisso entre largura do lóbulo principal e atenuação dos lóbulos laterais:

Janela	Atten. lateral	Lóbulo principal	Uso
Retangular	-13 dB	Estreito	Pouca atenuação necessária
Hamming	-43 dB	Moderado	Uso geral, boa atenuação
Hanning	-31 dB	Moderado	Análise espectral
Blackman	-58 dB	Largo	Máxima atenuação de lóbulos
Kaiser	Variável	Variável	Flexível — ajuste por beta

Outros métodos de projeto FIR incluem amostragem em frequência (especificar H[k] desejado e usar IDFT) e Parks-McClellan (minimax, distribuição equirriple do erro — implementado em scipy.signal.remez()).

3.4 Filtros IIR (Resposta ao Impulso Infinita)

Filtros IIR utilizam realimentação — a saída depende de entradas e saídas passadas:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k]$$

A função de transferência é H(z) = B(z)/A(z), onde B(z) e A(z) são polinômios em z^{-1}. Os polos de A(z) devem estar dentro do círculo unitário para garantir estabilidade.

Filtros IIR atingem especificações mais rigorosas com ordem muito menor que FIR, mas não possuem fase linear e podem ser instáveis se mal projetados. São tipicamente projetados a partir de protótipos analógicos usando transformação bilinear ou invariância ao impulso.

Tipo	Banda passante	Transição	Característica
Butterworth	Maximamente plana	Moderada	Sem ripple, resposta suave
Chebyshev I	Ripple na passante	Rápida	Melhor seletividade
Chebyshev II	Plana	Rápida	Ripple na rejeição apenas
Bessel	Plana	Lenta	Fase linear (preserva forma)
Elíptico	Ripple em ambas	Mais rápida	Menor ordem possível

FIR vs. IIR — quando usar cada um?
Use FIR quando fase linear é essencial (ECG, áudio), quando estabilidade garantida é prioridade, ou em implementações em hardware (FPGA). Use IIR quando a eficiência computacional é crítica (mesma seletividade com menos coeficientes) e a distorção de fase é aceitável.

3.5 Filtro de média móvel e filtro de Kalman simplificado

O filtro de média móvel é o filtro FIR mais simples, com todos os coeficientes iguais a 1/M:

$$y[n] = (1/M) \sum_{k=0}^{M-1} x[n-k]$$

É um passa-baixa eficiente para suavizar dados de sensores. Quanto maior M, maior a suavização, mas maior o atraso e a perda de detalhes. A frequência de corte aproximada é fc = 0.443 fs / M.

Uma variante importante é a média móvel exponencial (EMA), que é na verdade um filtro IIR de 1a ordem: y[n] = alpha x[n] + (1 - alpha) y[n-1], onde alpha controla o peso das amostras recentes.

O filtro de Kalman é um estimador ótimo recursivo para sistemas dinâmicos lineares com ruído gaussiano. Na sua forma simplificada para sensores escalares:

Etapa de predição:

$$x_{pred} = x_{est\_anterior}$$

$$P_{pred} = P_{anterior} + Q$$

Etapa de atualização:

$$K = P_{pred} / (P_{pred} + R) \text{ (Ganho de Kalman)}$$

$$x_{est} = x_{pred} + K (z_{medido} - x_{pred})$$

$$P = (1 - K) P_{pred}$$

Onde R é a variância do ruído de medição, Q é a variância do processo, P é a covariância do erro de estimação e z_medido é a medição. O filtro de Kalman é particularmente útil para fusão de sensores em sistemas IoT do LabAuto, pois adapta automaticamente sua confiança entre modelo e medição.

3.6 Implementação em Python com scipy.signal

O módulo scipy.signal oferece ferramentas completas para projeto e aplicação de filtros digitais. O exemplo a seguir projeta um filtro Butterworth passa-baixa de 4a ordem com frequência de corte de 5 Hz:

```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt

# Parâmetros
fs = 100           # Frequência de amostragem (Hz)
fc = 5             # Frequência de corte (Hz)
ordem = 4          # Ordem do filtro

# Projeto do filtro Butterworth
# Wn normalizado: fc / (fs/2) = fc / Nyquist
b, a = signal.butter(ordem, fc / (fs / 2), btype='low')

# Gerar sinal de teste: 2 Hz + 30 Hz + ruído
t = np.linspace(0, 1, fs, endpoint=False)
x = np.sin(2*np.pi*2*t) + 0.5*np.sin(2*np.pi*30*t)
x += 0.2 * np.random.randn(len(t))

# Aplicar filtro (filtfilt = zero-phase, sem atraso)
y = signal.filtfilt(b, a, x)

# Resposta em frequência do filtro
w, h = signal.freqz(b, a, worN=1024, fs=fs)

fig, axes = plt.subplots(3, 1, figsize=(10, 8))

axes[0].plot(t, x, alpha=0.7, label='Original')
axes[0].plot(t, y, linewidth=2, label='Filtrado')
axes[0].set_title('Sinal original vs. filtrado')
axes[0].legend()

axes[1].plot(w, 20*np.log10(np.abs(h)))
axes[1].set_ylabel('Ganho (dB)')
axes[1].set_title('Resposta em frequência')
axes[1].axvline(fc, color='r', linestyle='--',
                label=f'fc={fc}Hz')
axes[1].legend()

axes[2].plot(w, np.unwrap(np.angle(h))*180/np.pi)
axes[2].set_ylabel('Fase (graus)')
axes[2].set_xlabel('Frequência (Hz)')
axes[2].set_title('Resposta de fase')

plt.tight_layout()
plt.savefig('filtro_butterworth.png', dpi=150)
```

Código 3.1 — Projeto e aplicação de filtro Butterworth passa-baixa com scipy.signal

Nota: A função filtfilt() aplica o filtro nos dois sentidos (forward-backward), eliminando o atraso de fase. Isso dobra a ordem efetiva do filtro (8a ordem no exemplo). Para processamento em tempo real, use lfilter() que opera causalmente.

3.7 Efeitos de aliasing

O Teorema de Nyquist-Shannon é fundamental na amostragem de sinais:

Teorema de Nyquist-Shannon

Um sinal de banda limitada com frequência máxima fmax pode ser perfeitamente reconstruído a partir de suas amostras se a frequência de amostragem satisfizer: fs > 2 fmax. A frequência fN = fs/2 é chamada de frequência de Nyquist.

Quando fs < 2 fmax, ocorre aliasing: componentes de alta frequência são erroneamente mapeadas para frequências mais baixas, criando artefatos impossíveis de remover após a amostragem. A frequência aliased é: f_alias = |f - k fs| para o inteiro k mais próximo.

Por isso, filtros anti-aliasing (passa-baixa analógicos) são aplicados antes do conversor A/D para eliminar componentes acima de fs/2.

Exemplo prático: para amostrar corretamente um sinal de ECG com componentes até 150 Hz, a frequência de amostragem deve ser no mínimo 300 Hz (na prática, usa-se 500 Hz ou mais para margem de segurança). No LabAuto, sensores LoRa com dados até 50 Hz requerem fs >= 100 Hz.

Aplicação	fmax	fs mínimo	fs recomendado
Áudio (voz)	4 kHz	8 kHz	16 kHz
Áudio (música)	20 kHz	40 kHz	44,1 / 48 kHz
ECG diagnóstico	150 Hz	300 Hz	500 Hz
Sensor LoRa	50 Hz	100 Hz	200 Hz
Vibração (motor)	5 kHz	10 kHz	25 kHz

Tabela 3.1 — Frequências de amostragem recomendadas por aplicação

Capítulo 4 — Análise Wavelet

A análise wavelet representa uma evolução significativa em relação à transformada de Fourier, permitindo a análise simultânea de um sinal nos domínios do tempo e da frequência. Desenvolvida principalmente a partir da década de 1980, tornou-se ferramenta indispensável em processamento de sinais biomédicos, análise de vibração e compressão de dados.

4.1 Limitações da FFT para sinais não estacionários

A Transformada de Fourier pressupõe que o conteúdo espectral do sinal é constante ao longo do tempo. Para sinais estacionários (cujas propriedades estatísticas não mudam), isso funciona bem. Entretanto, muitos sinais reais são não estacionários: um ECG apresenta eventos localizados (complexo QRS dura ~100 ms), sinais sísmicos têm componentes transitórias, e dados de sensores podem sofrer mudanças abruptas.

A FFT de um sinal não estacionário fornece quais frequências estão presentes, mas não informa quando cada frequência ocorreu. A STFT (Short-Time Fourier Transform) tenta resolver isso usando janelas deslizantes, mas enfrenta o princípio da incerteza: boa resolução temporal exige janela curta (ruim em frequência) e vice-versa.

Matematicamente: Dt Df >= 1/(4 pi), onde Dt é a resolução temporal e Df a resolução em frequência. A STFT usa janela fixa, resultando em resolução constante. Já as wavelets usam janelas de comprimento adaptativo: janelas longas para baixas frequências e janelas curtas para altas frequências — a chamada análise multirresolução.

4.2 Transformada Wavelet Contínua (CWT)

A CWT decompõe um sinal usando versões dilatadas e transladadas de uma função base chamada wavelet mãe psi(t):

```
W(a, b) = (1/sqrt(|a|)) integral x(t) psi*((t-b)/a) dt
```

onde a é o parâmetro de escala (inversamente proporcional à frequência: escalas grandes capturam baixas frequências) e b é o parâmetro de translação (posição temporal). O asterisco indica complexo conjugado.

O resultado W(a,b) é um mapa bidimensional tempo-escala (escalograma) que mostra a energia do sinal em cada combinação de tempo e escala. O escalograma é análogo a um espectrograma, mas com resolução variável.

Uma wavelet mãe deve satisfazer a condição de admissibilidade: integral nula (valor médio zero) e energia finita. Isso garante que a transformada é invertível e o sinal pode ser reconstruído.

4.3 Transformada Wavelet Discreta (DWT)

A DWT discretiza os parâmetros de escala e translação em potências de 2: $a = 2^j$, $b = k\,2^j$ (amostragem diádica). A implementação eficiente usa bancos de filtros (algoritmo de Mallat, 1989):

- O sinal é filtrado por um passa-baixa (filtro de escala) e um passa-alta (filtro wavelet)
- As saídas são dizimadas por 2 (downsampling)

- Os coeficientes de aproximação (cA) vêm do passa-baixa; os coeficientes de detalhe (cD) do passa-alta
- O processo se repete nos coeficientes de aproximação para o próximo nível de decomposição

Em J níveis de decomposição, obtemos: cD1, cD2, ..., cDJ (detalhes em cada escala) e cAJ (aproximação final). A reconstrução perfeita é possível usando os filtros de síntese (condições de QMF — Quadrature Mirror Filter).

> **Banco de filtros — visão intuitiva**
> A cada nível, o sinal é dividido em 'metade inferior' (aproximação) e 'metade superior' (detalhe) de frequência. É como um equalizador que separa graves dos agudos em múltiplas oitavas. Com 4 níveis e fs = 1000 Hz: cD1 = 250-500 Hz, cD2 = 125-250 Hz, cD3 = 62.5-125 Hz, cD4 = 31.25-62.5 Hz, cA4 = 0-31.25 Hz.

A complexidade computacional da DWT é O(N) — linear no comprimento do sinal — tornando-a mais eficiente que a FFT para decomposição multirresolução.

4.4 Família de wavelets

A escolha da wavelet mãe depende da aplicação:

Wavelet	Suporte	Simetria	Quando usar
Haar (db1)	2 pontos	Simétrica	Descontinuidades, bordas
Daubechies (dbN)	2N pontos	Assimétrica	Uso geral, boa localização
Morlet	Contínuo	Simétrica	CWT, sinais suaves
Mexican Hat	Contínuo	Simétrica	Detecção de picos, curvatura
Coiflets (coifN)	6N-1 pts	Quase simétrica	Momentos nulos importantes
Symlets (symN)	2N pontos	Quase simétrica	Alternativa simétrica às db

Para sinais biomédicos (ECG, EMG), Daubechies de ordem 4 a 8 (db4-db8) são frequentemente utilizadas por terem boa localização tempo-frequência e suporte compacto. Para análise de vibração, Morlet é popular por sua boa resolução em frequência. Haar é ideal para aplicações simples ou sinais com transições abruptas.

O número de momentos nulos de uma wavelet indica sua capacidade de suprimir polinômios de grau N-1, sendo importante para separar tendências de detalhes. Daubechies dbN tem N momentos nulos.

4.5 Aplicações

Detecção de arritmias (ECG)

A DWT com wavelet db4 ou db6 é usada para detectar o complexo QRS em sinais de ECG. Os coeficientes de detalhe nos níveis 3-5 (correspondentes à faixa 5-40 Hz em fs = 360 Hz) concentram a energia do QRS, permitindo detecção robusta mesmo na presença de ruído e variação da linha de base. A detecção é baseada em limiarização dos coeficientes wavelet.

Análise de vibração em motores

Sinais de vibração de motores contêm componentes transitórias (impactos de rolamentos defeituosos) sobrepostas a componentes periódicas (rotação normal). A CWT com wavelet Morlet produz um escalograma que permite identificar a localização temporal e a frequência dessas transientes, possibilitando diagnóstico precoce de falhas. Frequências características de defeito em rolamento: BPFO (pista externa), BPFI (pista interna), BSF (elemento rolante), FTF (gaiola).

Denoising de sinais de sensores agrícolas

O denoising por wavelet segue três passos: (1) decomposição DWT do sinal ruidoso, (2) aplicação de threshold nos coeficientes de detalhe (thresholding suave ou duro), (3) reconstrução do sinal limpo pela IDWT.

O limiar pode ser calculado pelo método VisuShrink (universal: T = sigma sqrt(2 ln(N)), onde sigma é estimado pela mediana dos coeficientes cD1) ou SureShrink (adaptativo por subbanda, minimiza o risco SURE).

O thresholding suave reduz os coeficientes pela magnitude do limiar (sign(c) max(|c|-T, 0)), produzindo sinal mais suave. O thresholding duro zera coeficientes abaixo do limiar e mantém os demais inalterados, preservando melhor a amplitude dos picos.

Capítulo 5 — Aplicações em Biossistemas e Sensores

Este capítulo apresenta aplicações práticas de processamento de sinais no contexto da Engenharia de Biossistemas e do LabAuto, focando em problemas reais do semiárido paraibano.

5.1 Processamento de sinais de ECG

O eletrocardiograma (ECG) é um sinal biomédico fundamental para monitoramento cardíaco. A cadeia de processamento típica envolve:

- Pré-processamento: Remoção da deriva de linha de base com filtro passa-alta (fc = 0,5 Hz). Remoção de interferência de 60 Hz com filtro notch (IIR de 2a ordem, Q = 30). Suavização com filtro passa-baixa (fc = 40-150 Hz, dependendo da aplicação).
- Detecção de picos R: O algoritmo de Pan-Tompkins (1985) é o padrão: aplica filtro passa-banda (5-15 Hz), derivada, elevação ao quadrado, integração por janela móvel (~150 ms), e detecção por limiar adaptativo.
- Cálculo de HRV (variabilidade da frequência cardíaca): A partir dos intervalos RR (tempo entre picos R consecutivos), calcula-se métricas no domínio do tempo e da frequência.

Métricas de HRV no domínio do tempo:

- SDNN: Desvio padrão dos intervalos NN (normal-to-normal)
- RMSSD: Raiz quadrada da média dos quadrados das diferenças sucessivas
- pNN50: Percentual de diferenças sucessivas > 50 ms

Métricas de HRV no domínio da frequência:

- Banda VLF: 0,003-0,04 Hz (regulação térmica, SRAA)
- Banda LF: 0,04-0,15 Hz (simpático + parassimpático)
- Banda HF: 0,15-0,4 Hz (parassimpático, arritmia sinusal respiratória)
- Razão LF/HF: indicador de balanço simpato-vagal

```
# Detecção simplificada de picos R com scipy
from scipy.signal import find_peaks, butter, filtfilt
import numpy as np

def detect_r_peaks(ecg, fs=360):
    # Filtro passa-banda 5-15 Hz
    b, a = butter(2, [5/(fs/2), 15/(fs/2)], 'band')
    filtered = filtfilt(b, a, ecg)

    # Derivada + quadrado
    diff_ecg = np.diff(filtered)
    squared = diff_ecg ** 2

    # Integração por janela móvel (150ms)
    win_size = int(0.15 * fs)
    integrated = np.convolve(squared,
                    np.ones(win_size)/win_size, 'same')

    # Detecção de picos com distância mínima
    min_dist = int(0.2 * fs)  # 200ms mínimo
    peaks, _ = find_peaks(integrated,
                    distance=min_dist,
                    height=0.5*np.max(integrated))
    return peaks

# Cálculo de HRV básico
def calc_hrv(r_peaks, fs=360):
    rr = np.diff(r_peaks) / fs * 1000  # ms
    sdnn = np.std(rr)
    rmssd = np.sqrt(np.mean(np.diff(rr)**2))
    bpm = 60000 / np.mean(rr)
    return {'SDNN_ms': sdnn,
            'RMSSD_ms': rmssd,
            'HR_mean_bpm': bpm}
```

Código 5.1 — Detecção de picos R e cálculo de HRV simplificados

5.2 Análise espectral de sinais de solo

Sensores de umidade e temperatura do solo geram séries temporais que contêm informações valiosas sobre padrões sazonais, ciclos diários e anomalias. A análise espectral dessas séries permite:

- Identificar o ciclo diurno (frequência de ~1/24h) na temperatura do solo
- Detectar padrões sazonais de umidade relacionados ao regime de chuvas do semiárido
- Identificar anomalias que podem indicar falha do sensor ou evento climático extremo
- Calcular a difusividade térmica do solo a partir da atenuação e defasagem da onda térmica em diferentes profundidades

A FFT é aplicada a janelas de dados (tipicamente 7 a 30 dias) para gerar periodogramas. A técnica de Welch (periodograma médio com janelas sobrepostas de 50-75%) é preferível por reduzir a variância da estimativa espectral. Em Python: scipy.signal.welch().

No semiárido, a identificação do ciclo anual de umidade é particularmente relevante para planejamento de irrigação. Tipicamente observa-se um pico espectral correspondente ao período de aproximadamente 365 dias, com harmônicos sub-anuais relacionados a eventos

de chuva.

5.3 Denoising de sinais LoRa ruidosos

Redes LoRa (Long Range) são amplamente usadas no LabAuto para transmissão de dados de sensores distribuídos no semiárido. Sinais LoRa podem ser corrompidos por ruído devido a longas distâncias de transmissão, interferências eletromagnéticas e condições atmosféricas adversas.

A abordagem de denoising por wavelet é particularmente eficaz nesse contexto por preservar transientes e bordas nos dados enquanto remove ruído gaussiano:

```
import pywt
import numpy as np

def wavelet_denoise(signal, wavelet='db4', level=4):
    """Denoising de sinal com wavelet threshold."""
    # Decomposição DWT
    coeffs = pywt.wavedec(signal, wavelet,
                          level=level)

    # Estimativa robusta do ruído (MAD)
    sigma = np.median(np.abs(coeffs[-1])) / 0.6745

    # Limiar universal (VisuShrink)
    threshold = sigma * np.sqrt(
        2 * np.log(len(signal)))

    # Thresholding suave nos coefs. de detalhe
    denoised_coeffs = [coeffs[0]]
    for c in coeffs[1:]:
        denoised_coeffs.append(
            pywt.threshold(c, threshold,
                           mode='soft'))

    # Reconstrução
    return pywt.waverec(denoised_coeffs, wavelet)

# Exemplo de uso com dados LoRa
# raw = load_lora_data('sensor_01.csv')
# cleaned = wavelet_denoise(raw)
# snr_improvement = 10*np.log10(
#     np.var(raw)/np.var(raw - cleaned))
```

Código 5.2 — Denoising de sinais LoRa com transformada wavelet

A escolha do nível de decomposição depende da frequência de amostragem e da faixa de frequência do ruído. Para dados LoRa amostrados a 100 Hz com ruído acima de 25 Hz, 2-3 níveis são suficientes.

5.4 Visão computacional básica

O processamento de imagens é uma extensão natural do processamento de sinais para 2D. No contexto do LabAuto, imagens aéreas capturadas por drones no semiárido são processadas para:

- Índices de vegetação: NDVI = (NIR - Red)/(NIR + Red), usando bandas multiespectrais para avaliar a saúde da vegetação em áreas de caatinga
- Filtragem espacial: Filtros de suavização (gaussiano, mediana) para remoção de ruído, e filtros de detecção de bordas (Sobel, Laplaciano) para identificação de limites de parcelas
- Segmentação: Separação de solo exposto, vegetação e água em imagens RGB usando técnicas como limiarização de Otsu ou k-means clustering
- Transformada de Fourier 2D: Análise espectral de texturas do solo e padrões de plantio, útil para identificação de culturas e estimativa de biomassa

A convolução 2D é a operação fundamental: y[m,n] = sum_{i,j} x[m-i, n-j] h[i,j], onde h é o kernel do filtro. Kernels 3x3 são comuns: suavização gaussiana, Sobel horizontal/vertical, Laplaciano. Em Python, OpenCV (cv2) e scikit-image são as bibliotecas padrão.

A FFT 2D é computada eficientemente por numpy.fft.fft2() e permite análise de frequências espaciais — padrões de plantio regulares aparecem como picos no domínio espacial de frequência, permitindo estimar espaçamento entre linhas e cobertura vegetal.

Exercícios Resolvidos

Exercício 1 — DFT manual de 4 pontos

Enunciado: Calcule a DFT de x = [1, 0, -1, 0] manualmente.

Solução:

Usando X[k] = sum_{n=0}^{3} x[n] exp(-j 2 pi k n / 4), com N = 4:

Os fatores twiddle são W4 = exp(-j pi/2) = -j. Logo $W4^0$ = 1, $W4^1$ = -j, $W4^2$ = -1, $W4^3$ = j.

- X[0] = 1(1) + 0(1) + (-1)(1) + 0(1) = 0
- X[1] = 1(1) + 0(-j) + (-1)(-1) + 0(j) = 1 + 1 = 2
- X[2] = 1(1) + 0(-1) + (-1)(1) + 0(-1) = 0
- X[3] = 1(1) + 0(j) + (-1)(-1) + 0(-j) = 1 + 1 = 2

Resultado: X = [0, 2, 0, 2]

Interpretação: O sinal x = [1, 0, -1, 0] contém apenas as frequências k=1 e k=3 (que são simétricas para sinais reais), sem componente DC.

Exercício 2 — Frequência de Nyquist

Enunciado: Um sensor de vibração captura sinais com componentes até 500 Hz. Qual a frequência mínima de amostragem? Se usarmos fs = 800 Hz, o que acontece?

Solução:

Pelo teorema de Nyquist: fs_min = 2 fmax = 2 x 500 = 1000 Hz.

Com fs = 800 Hz < 1000 Hz: ocorre aliasing. A frequência de Nyquist é fN = 800/2 = 400 Hz. Componentes entre 400 Hz e 500 Hz serão refletidas para a faixa 300-400 Hz (f_alias = fs - f = 800 - 500 = 300 Hz para a componente de 500 Hz). Isso corrompe irreversivelmente o sinal.

Na prática, recomenda-se fs >= 2,5 fmax = 1250 Hz, com filtro anti-aliasing a 500 Hz.

Exercício 3 — Projeto de filtro FIR

Enunciado: Projete um filtro FIR passa-baixa de ordem 20 com frequência de corte de 10 Hz para uma frequência de amostragem de 100 Hz, usando janela de Hamming.

Solução em Python:

```
from scipy.signal import firwin, freqz
import numpy as np

# Parâmetros
fs = 100        # Hz
fc = 10         # Hz
num_taps = 21   # ordem + 1

# Projeto do filtro
h = firwin(num_taps, fc, fs=fs, window='hamming')

# Verificação: resposta em frequência
w, H = freqz(h, worN=512, fs=fs)
# Em fc, o ganho deve ser aprox. -6 dB
idx = np.argmin(np.abs(w - fc))
print(f'Ganho em fc: {20*np.log10(np.abs(H[idx])):.1f} dB')
# Resultado esperado: aprox. -6.0 dB
```

Os 21 coeficientes do filtro h são simétricos (fase linear garantida). A janela de Hamming proporciona atenuação de lóbulos laterais de ~43 dB. Os coeficientes são: h[k] = h_ideal[k] . w_hamming[k], onde h_ideal é o filtro ideal truncado (sinc) e w_hamming[k] = 0,54 - 0,46 cos(2 pi k / 20).

Exercício 4 — Convolução e resposta de sistema LIT

Enunciado: Um sistema LIT tem resposta ao impulso h = [1, 2, 1]. Calcule a saída para a entrada x = [3, -1, 2, 4].

Solução: y = x * h, comprimento = 4 + 3 - 1 = 6.

- y[0] = 3 x 1 = 3
- y[1] = 3 x 2 + (-1) x 1 = 5
- y[2] = 3 x 1 + (-1) x 2 + 2 x 1 = 3
- y[3] = (-1) x 1 + 2 x 2 + 4 x 1 = 7
- y[4] = 2 x 1 + 4 x 2 = 10
- y[5] = 4 x 1 = 4

Resultado: y = [3, 5, 3, 7, 10, 4]

Verificação: sum(x) x sum(h) = 8 x 4 = 32 = sum(y). A soma se preserva na convolução.

Exercício 5 — Decomposição Wavelet Haar

Enunciado: Aplique um nível de decomposição wavelet Haar ao sinal x = [6, 4, 8, 2].

Solução:

A wavelet Haar usa filtros h0 = [1/sqrt(2), 1/sqrt(2)] (passa-baixa) e h1 = [1/sqrt(2), -1/sqrt(2)] (passa-alta), seguidos de dizimação por 2.

- cA = [(6+4)/sqrt(2), (8+2)/sqrt(2)] = [10/sqrt(2), 10/sqrt(2)] = [7.07, 7.07]
- cD = [(6-4)/sqrt(2), (8-2)/sqrt(2)] = [2/sqrt(2), 6/sqrt(2)] = [1.41, 4.24]

Os coeficientes de aproximação cA capturam a tendência geral (média local); os de detalhe cD capturam variações (diferença local).

Verificação da energia (Parseval): sum(x^2) = 36+16+64+4 = 120; sum(cA^2) + sum(cD^2) = 50+50+2+18 = 120.

Roteiro de Laboratório — Lab 01

Análise Espectral e Filtragem de Sinais com Python

Objetivo

Gerar sinais sintéticos com componentes de frequência conhecidas, analisar seu espectro usando a FFT, projetar e aplicar filtros digitais, e avaliar os resultados no domínio do tempo e da frequência. Ao final, o aluno deve ser capaz de identificar componentes espectrais, projetar filtros adequados e avaliar a qualidade da filtragem.

Materiais e Software

- Python 3.8+ com Anaconda ou ambiente virtual
- Bibliotecas: numpy, scipy, matplotlib, pywt (PyWavelets)
- Instalação: `pip install numpy scipy matplotlib PyWavelets`
- Editor: Jupyter Notebook ou VS Code

Fundamentação teórica

A FFT permite identificar as frequências presentes em um sinal discreto. A resolução espectral é Df = fs/N. Filtros digitais modificam o conteúdo espectral: passa-baixa remove componentes de alta frequência (útil para suavização), passa-alta remove componentes de baixa frequência (útil para remoção de tendências). A SNR (Signal-to-Noise Ratio) em dB quantifica a qualidade: SNR = 10 log10(P_sinal / P_ruido).

Procedimento

Parte 1 — Geração de sinal e análise FFT (30 min)

- 1. Gere um sinal composto: x(t) = 2 sin(2pi 5t) + sin(2pi 20t) + 0.5 sin(2pi 50t), com fs = 200 Hz e duração de 2 s.
- 2. Adicione ruído gaussiano com sigma = 0,3.
- 3. Calcule a FFT e plote o espectro de magnitude. Identifique os três picos.
- 4. Varie o número de pontos (N = 64, 256, 1024) e observe a mudança na resolução espectral.
- 5. Aplique zero-padding (N = 2048) e compare a suavidade do espectro.
- 6. Calcule e reporte a SNR do sinal ruidoso.

Parte 2 — Projeto e aplicação de filtro (30 min)

- 1. Projete um filtro Butterworth passa-baixa de 4a ordem com fc = 30 Hz.
- 2. Aplique o filtro ao sinal ruidoso usando scipy.signal.filtfilt().
- 3. Compare o sinal original, o ruidoso e o filtrado no mesmo gráfico.
- 4. Calcule e plote o espectro do sinal filtrado — confirme que a componente de 50 Hz foi atenuada.
- 5. Repita com filtro FIR (firwin, 51 taps) e compare o resultado.
- 6. Compare a SNR antes e depois da filtragem (IIR e FIR).

Parte 3 — Denoising wavelet (20 min)

- 1. Aplique decomposição DWT com wavelet db4, nível 4, ao sinal ruidoso.
- 2. Visualize os coeficientes de aproximação e detalhe em cada nível.
- 3. Aplique thresholding suave (VisuShrink) e reconstrua o sinal.
- 4. Compare SNR: sinal ruidoso vs. Butterworth vs. FIR vs. wavelet.

Questões para discussão

- 1. Qual método de filtragem obteve melhor SNR? Por quê?
- 2. Qual a diferença visual entre o sinal filtrado por Butterworth e o filtrado por wavelet?
- 3. O que acontece se aumentar a ordem do filtro Butterworth para 8?
- 4. Qual o efeito de usar wavelet Haar em vez de db4 para denoising?

Relatório

Entregue notebook Jupyter (.ipynb) contendo: código completo comentado, gráficos com títulos e legendas, tabela comparativa de SNR para todos os métodos, e discussão respondendo às questões acima. Prazo: 7 dias após a aula de laboratório.

Critérios de avaliação:

- Código funcional e bem comentado (30%)
- Gráficos corretos com formatação adequada (25%)
- Cálculos de SNR corretos (20%)
- Discussão e análise crítica (25%)

Referências

[1] HAYKIN, S.; VAN VEEN, B. Sinais e Sistemas. 2. ed. Porto Alegre: Bookman, 2001. 668 p.

[2] OPPENHEIM, A. V.; WILLSKY, A. S. Sinais e Sistemas. 2. ed. São Paulo: Pearson Prentice Hall, 2010. 592 p.

[3] OPPENHEIM, A. V.; SCHAFER, R. W. Discrete-Time Signal Processing. 3rd ed. Upper Saddle River: Pearson, 2010. 1108 p.

[4] PROAKIS, J. G.; MANOLAKIS, D. G. Digital Signal Processing: Principles, Algorithms, and Applications. 4th ed. Upper Saddle River: Pearson Prentice Hall, 2007. 1084 p.

[5] MALLAT, S. A Wavelet Tour of Signal Processing: The Sparse Way. 3rd ed. Burlington: Academic Press, 2009. 805 p.

[6] DAUBECHIES, I. Ten Lectures on Wavelets. Philadelphia: SIAM, 1992. 357 p. (CBMS-NSF Regional Conference Series in Applied Mathematics, 61).

[7] SMITH, S. W. The Scientist and Engineer's Guide to Digital Signal Processing. 2nd ed. San Diego: California Technical Publishing, 1999. 650 p. Disponível em: http://www.dspguide.com.

[8] PAN, J.; TOMPKINS, W. J. A real-time QRS detection algorithm. IEEE Transactions on Biomedical Engineering, v. BME-32, n. 3, p. 230-236, mar. 1985.

[9] VIRTANEN, P. et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods, v. 17, n. 3, p. 261-272, 2020. Disponível em: https://doi.org/10.1038/s41592-019-0686-2.

[10] HARRIS, C. R. et al. Array programming with NumPy. Nature, v. 585, n. 7825, p. 357-362, 2020. Disponível em: https://doi.org/10.1038/s41586-020-2649-2.