Implementation Notes

From-Scratch Digital Signal Processing for Audio Denoising

Pai Surya Darshan

Dec 2025

Abstract

This document provides a function-by-function breakdown of the codebase, detailing the purpose, structure, and operational role of each component. Where applicable, formal expressions are stated alongside their verbal interpretation to clarify how specific computational steps are realised. The document is intended as my technical reference for understanding when revisiting the implementation, rather than as a user guide or methodological claim.

1. Code walkthrough: function-level analysis

1.1. `run(cmd)`

Verbal meaning

Executes a shell command using Python's subprocess interface and raises an exception if the command fails.

1.2. `find_ffmpeg()`

Verbal meaning

Searches for the FFmpeg executable on the system PATH and raises an explicit error if it is unavailable.

Reasoning

FFmpeg is a hard external dependency; early failure avoids silent downstream errors.

1.3. `wav_read_pcm16(path)`

Verbal meaning

Reads a WAV audio file encoded as 16-bit PCM and converts it into a floating-point time-domain signal suitable for downstream digital signal processing. The function extracts the sampling rate, enforces a fixed bit-depth assumption, and ensures the output signal is mono.

The audio samples are returned as a one-dimensional NumPy array with amplitudes normalised to a unit range. Stereo inputs are downmixed to mono by channel averaging, while unsupported channel configurations are explicitly rejected.

Formal definition

$$x[n] = \frac{s[n]}{32768}$$

Where,

- $s[n]$ — signed 16-bit PCM integer sample, $s[n] \in [-32768, 32767]$
- $x[n]$ — normalised floating-point audio sample, $x[n] \in [-1, 1)$

Formal definition

$$x_{\text{mono}}[n] = \frac{x_{\text{left}}[n] + x_{\text{right}}[n]}{2}$$

Where,

- $x_{\text{left}}[n]$ — left-channel audio sample
- $x_{\text{right}}[n]$ — right-channel audio sample
- $x_{\text{mono}}[n]$ — mono downmixed audio sample

1.4. `wav_write_pcm16(path, x, sr)`

Verbal meaning

Writes a time-domain audio signal to disk as a standard 16-bit PCM WAV file at a specified sampling rate. The input signal is assumed to be a floating-point array where each sample represents the instantaneous amplitude of the sound.

Before writing the signal to disk, the function enforces an explicit amplitude range. Any sample values that exceed the representable range of the output audio format are corrected to prevent numerical errors during conversion. This step ensures that the resulting audio file is valid, free of overflow artefacts, and compatible with common audio tools and media containers.

The output file is written as a single-channel (mono) waveform with fixed bit depth and sampling rate, ensuring predictable behaviour in downstream processing and video remultiplexing.

Formal definition

$$x_{\text{clip}}[n] = \max(-1,\ \min(1,\ x[n]))$$

Where,

- $x[n]$ — input floating-point audio sample at time index n
- $x_{\text{clip}}[n]$ — amplitude-limited (clipped) audio sample

Verbal meaning

Amplitude clipping ensures that all audio samples lie within the valid dynamic range of the target representation. Any sample larger than $+1$ is replaced with $+1$, and any sample smaller than -1 is replaced with -1. This operation prevents overflow when converting from floating-point representation to fixed-width integer representation.

Formal definition

$$s[n] = 32767 \cdot x_{\text{clip}}[n]$$

Where,

- $x_{\text{clip}}[n]$ — clipped floating-point audio sample
- $s[n]$ — quantised 16-bit PCM integer sample

Verbal meaning

The clipped floating-point samples are scaled to the numerical range of a signed 16-bit integer. This step maps the continuous-valued audio signal into a discrete representation suitable for storage in a PCM WAV file. The scaling factor 32767 corresponds to the maximum positive value representable by a signed 16-bit integer.

1.5. `biquad_highpass(x, sr, fc, q)`

Verbal meaning

Applies a **high-pass filter** to an audio signal. A high-pass filter removes (or strongly reduces) **very low-frequency** components while keeping higher-frequency content mostly unchanged. In real recordings, these low frequencies are often not useful signal, but rather **rumble**, **handling noise**, **air-conditioning vibration**, **mic stand movement**, or **DC drift**.

This implementation uses a **biquad** filter, which is a common, efficient filter structure that uses the current input sample, the two previous input samples, and the two previous output samples to compute the next output sample. The filter is **second-order** (two-sample memory), which gives a stronger and smoother cutoff than a simple first-order filter, without being computationally expensive.

Verbal meaning

The inputs mean:

- x — the input audio signal (array of samples in the time domain)
- sr — sampling rate in Hz (samples per second)
- f_c — cutoff frequency in Hz (frequencies below this are reduced)
- q — **quality factor**, which controls how sharp / resonant the transition is near the cutoff (higher q means a narrower, more resonant transition)

Verbal meaning

This function has two conceptual parts:

- **Coefficient calculation:** compute the constants $b_0, b_1, b_2, a_0, a_1, a_2$ that define the filter.
- **Filtering loop:** apply the filter to every sample using a difference equation.

1.5.1. Line-by-line mapping to the implementation

Verbal meaning

1) Convert the cutoff frequency into a digital (normalised) angular frequency. The signal is sampled, so filter design uses a normalised frequency relative to the sampling rate:

$$\omega_0 = \frac{2\pi f_c}{sr}.$$

This is what the line `w0 = 2.0 * np.pi * fc / sr` computes.

Verbal meaning

2) Precompute sine and cosine of ω_0. The standard biquad formulas depend on $\sin(\omega_0)$ and $\cos(\omega_0)$, so the code computes: `cosw0 = np.cos(w0)` and `sinw0 = np.sin(w0)`.

Verbal meaning

3) Compute α, which controls the filter's bandwidth via q. The parameter α controls the "steepness / resonance behaviour" near the cutoff: `alpha = sinw0 / (2.0 * q)`.

Formal definition

$$\omega_0 = \frac{2\pi f_c}{sr}$$

Where,

- f_c — cutoff frequency (Hz)
- sr — sampling rate (Hz)

- ω_0 — normalised angular frequency (radians per sample)

Formal definition

$$\alpha = \frac{\sin(\omega_0)}{2q}$$

Where,

- α — bandwidth/resonance control term for the biquad design
- q — quality factor (dimensionless)
- $\sin(\omega_0)$ — sine of the normalised angular frequency

Verbal meaning

4) Compute the raw biquad coefficients for a high-pass filter. The code uses the standard "RBJ Audio EQ Cookbook" high-pass biquad form, producing:

$$b_0, b_1, b_2 \quad \text{(numerator terms)}, \qquad a_0, a_1, a_2 \quad \text{(denominator terms)}.$$

These coefficients define how the current and previous samples influence the output.

Formal definition

$$b_0 = \frac{1+\cos(\omega_0)}{2}, \quad b_1 = -(1+\cos(\omega_0)), \quad b_2 = \frac{1+\cos(\omega_0)}{2}$$

Where,

- b_0, b_1, b_2 — feedforward (numerator) coefficients for the high-pass filter
- $\cos(\omega_0)$ — cosine of the normalised angular frequency

Formal definition

$$a_0 = 1+\alpha, \quad a_1 = -2\cos(\omega_0), \quad a_2 = 1-\alpha$$

Where,

- a_0, a_1, a_2 — feedback (denominator) coefficients for the high-pass filter
- α — bandwidth/resonance control term
- $\cos(\omega_0)$ — cosine of the normalised angular frequency

Verbal meaning

5) Normalise coefficients by a_0. The biquad difference equation is typically written with the denominator leading coefficient set to 1. To enforce that, all coefficients are divided by a_0: `b0 /= a0; b1 /= a0; b2 /= a0; a1 /= a0; a2 /= a0`.

Formal rule

$$b_k \leftarrow \frac{b_k}{a_0} \text{ for } k \in \{0,1,2\}, \qquad a_k \leftarrow \frac{a_k}{a_0} \text{ for } k \in \{1,2\}$$

Where,

- a_0 — leading denominator coefficient
- b_k — numerator coefficients
- a_k — denominator coefficients (excluding a_0)

Verbal meaning

6) Apply the filter sample-by-sample using the biquad difference equation. The core filtering loop computes each output sample $y[n]$ from:

- the current input $x[n]$,
- the two previous inputs $x[n-1], x[n-2]$,
- and the two previous outputs $y[n-1], y[n-2]$.

This is why the code stores `x1, x2, y1, y2` and updates them each iteration.

Formal definition

$$y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] \; - \; a_1 y[n-1] \; - \; a_2 y[n-2]$$

Where,

- $x[n]$ — current input sample
- $x[n-1], x[n-2]$ — previous two input samples
- $y[n]$ — current output sample (filtered)
- $y[n-1], y[n-2]$ — previous two output samples
- b_0, b_1, b_2 — feedforward coefficients
- a_1, a_2 — feedback coefficients (with a_0 normalised to 1)

Verbal meaning

At the end, the function returns y, the high-pass filtered version of the input signal.

Reasoning

A biquad high-pass filter provides a strong and smooth suppression of low-frequency artefacts while remaining computationally lightweight. Compared to FFT-based filtering, it operates directly in the time domain with constant per-sample cost. Compared to a first-order high-pass, the second-order form provides a sharper and more controllable transition around the cutoff frequency for the same computational overhead class.

1.6. `stft(x, n_fft, hop)`

Verbal meaning

Computes the **Short-Time Fourier Transform (STFT)** of a one-dimensional audio signal. The STFT converts a time-domain signal into a **time–frequency representation** by analysing short, overlapping segments of the signal rather than the entire signal at once.

Instead of asking "what frequencies exist in the whole signal?", the STFT asks:

> "What frequencies exist *here*, and how does that change over time?"

This is essential for real-world audio, where sound content (speech, noise, transients) changes continuously.

The function outputs:

- X — a complex-valued matrix representing frequency content over time
- w — the window function used during analysis
- the effective signal length after padding

Verbal meaning

The STFT procedure implemented here consists of four conceptual steps:

- window generation,
- zero-padding (if required),
- frame extraction with overlap,
- frequency-domain transformation of each frame.

Windowing and the Hann window

Verbal meaning

Before transforming each signal segment into the frequency domain, the samples are multiplied by a **window function**. A window smoothly reduces the amplitude of samples near the edges of each frame. This avoids artificial discontinuities when frames are cut out of the continuous signal.

Without windowing, abrupt frame boundaries introduce spurious frequency components, a phenomenon known as **spectral leakage**.

The Hann window is a commonly used, smooth, cosine-based window that tapers the signal to zero at both ends of the frame.

Formal definition

$$w[n] = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right)$$

Where,

- $w[n]$ — Hann window value at sample index n
- n — sample index within the window, $0 \leq n < N$
- N — window length (equal to n_{fft})

Verbal meaning

The Hann window has three properties that make it well suited for STFT-based analysis:

- it smoothly tapers to zero at frame boundaries,
- it significantly reduces spectral leakage compared to a rectangular (no window) case,
- it supports stable overlap–add reconstruction when paired with common hop sizes.

It is not the only possible window (others include Hamming, Blackman, and Kaiser), but the Hann window offers a good balance between frequency resolution, leakage suppression, and simplicity.

Frame extraction and overlap

Verbal meaning

The input signal is divided into overlapping frames of fixed length. Each frame is offset from the previous one by a fixed number of samples known as the **hop size**. Overlapping frames ensure that short-lived signal features are not missed and that time resolution is preserved.

If the input signal is shorter than a single frame, it is zero-padded so that at least one complete frame can be analysed.

Formal definition

$$x_k[n] = x[n + kH] \cdot w[n]$$

Where,

- $x[n]$ — original time-domain signal
- $x_k[n]$ — windowed samples of frame k
- H — hop size (number of samples between frames)

- $w[n]$ — Hann window
- n — sample index within the frame

Verbal meaning

The total number of frames is computed so that frames slide across the entire signal with constant spacing and fixed length.

Frequency-domain transformation

Verbal meaning

Each windowed frame is transformed into the frequency domain using the **Discrete Fourier Transform (DFT)**. Because the input signal is real-valued, only the non-redundant positive-frequency terms are retained using the real FFT.

Formal definition

$$X_k[m] = \sum_{n=0}^{N-1} x_k[n]\ e^{-j2\pi mn/N}$$

Where,

- $X_k[m]$ — complex frequency coefficient at frequency bin m and frame k
- $x_k[n]$ — windowed time-domain samples
- N — FFT size (n_{fft})
- j — imaginary unit

Verbal meaning

The result is a two-dimensional representation where:

- rows correspond to time frames,
- columns correspond to frequency bins,
- magnitudes encode energy at each time–frequency location,
- phases encode time alignment information.

Reasoning

The STFT is preferred here over a single global Fourier transform because the denoising algorithm must reason about how noise and signal energy vary over time. The Hann window is chosen because it provides strong leakage suppression while remaining compatible with overlap–add reconstruction used later in the inverse STFT.

1.7. `istft(X, win, hop, out_len)`

Verbal meaning

Reconstructs a time-domain audio signal from its Short-Time Fourier Transform (STFT) representation. This function performs the **inverse Short-Time Fourier Transform (ISTFT)** using a standard **overlap–add** procedure.

Conceptually, the ISTFT answers the question:

> "Given many short frequency snapshots of a signal, how do we stitch them back together into one continuous waveform?"

The function takes the frequency-domain frames, converts each frame back into the time domain, aligns them according to the hop size, and adds them together. A normalization step is applied to correct for the energy scaling introduced by windowing and overlap.

The output is a single, continuous time-domain signal with the original signal length restored.

Verbal meaning

The inputs mean:

- X — complex-valued STFT matrix (time frames × frequency bins)
- w — window function used during the forward STFT
- H — hop size (number of samples between frames)
- out_len — desired output signal length

Verbal meaning

This function proceeds in four conceptual stages:

- determine signal and frame dimensions,
- inverse-transform each frequency frame,
- overlap and add windowed frames in time,
- normalize and trim the output signal.

Frame dimension reconstruction

Verbal meaning

The number of FFT points used during the forward transform is inferred from the number of frequency bins. For a real-valued FFT, only the non-redundant positive-frequency bins are stored. The original FFT size is therefore reconstructed from the STFT matrix shape.

Formal definition

$$N = 2\,(B - 1)$$

Where,

- N — FFT size used during the forward STFT
- B — number of frequency bins per frame

Verbal meaning

The length of the reconstructed time-domain signal prior to trimming is determined by the number of frames, the hop size, and the FFT size.

Formal definition

$$L = N + (K - 1)\,H$$

Where,

- L — length of the reconstructed signal before trimming
- K — number of STFT frames
- H — hop size
- N — FFT size

Inverse frequency transform

Verbal meaning

Each time–frequency frame is converted back into the time domain using the inverse real-valued Fast Fourier Transform (IRFFT). This reconstructs a real-valued time-domain segment from the stored frequency bins.

Formal definition

$$x_k[n] = \mathcal{F}^{-1}\{X_k[m]\}$$

Where,

- $X_k[m]$ — complex frequency-domain coefficients of frame k
- $x_k[n]$ — reconstructed time-domain samples of frame k
- $\mathcal{F}^{-1}\{\cdot\}$ — inverse Fourier transform

Verbal meaning

The reconstructed frames have the same length as the original FFT size and are real-valued, as expected for audio signals.

Overlap–add reconstruction

Verbal meaning

Each reconstructed frame is multiplied by the same window function used during analysis and then added into the output signal at the appropriate time offset. This process is known as **overlap–add** synthesis.

Formal definition

$$y[n] = \sum_k x_k[n - kH] \cdot w[n - kH]$$

Where,

- $y[n]$ — reconstructed output signal
- $x_k[\cdot]$ — time-domain samples of frame k
- $w[\cdot]$ — window function
- H — hop size

Verbal meaning

Because frames overlap in time, multiple windowed contributions are added together at many sample locations.

Energy normalization

Verbal meaning

Windowing and overlap cause some samples to be summed more times than others. To avoid artificial amplitude modulation, the signal is normalized by the sum of squared window values at each sample position.

Formal definition

$$y_{\text{norm}}[n] = \frac{y[n]}{\sum_k w^2[n - kH]}$$

Where,

- $y[n]$ — overlap–added signal
- $y_{\text{norm}}[n]$ — normalized output signal
- $w[\cdot]$ — window function

Verbal meaning

A small numerical constant is used in the implementation to prevent division by zero when the window sum is extremely small.

Output trimming

Verbal meaning

The final step trims the reconstructed signal to the original desired length. This removes any extra samples introduced by zero-padding during the forward STFT.

Reasoning

Explicit overlap–add reconstruction with normalization ensures that the inverse transform accurately inverts the forward STFT when paired with a compatible window and hop size. This approach preserves signal energy and avoids amplitude distortion, which is essential for fair comparison between original and denoised signals.

1.8. `denoise_dsp(x, sr, n_fft, hop, init_noise_seconds, noise_update, dd_alpha, gain_smooth, oversub, floor_base, hp_fc)`

Verbal meaning

Implements a **from-scratch, classical DSP** audio denoising pipeline based on **time–frequency analysis**. The core idea is:

> Convert the signal into short-time frequency frames (STFT), estimate what portion of each frequency bin is likely to be noise, suppress those bins using a gain function G, and reconstruct the time-domain signal using the inverse STFT (ISTFT).

Verbal meaning

This function is the main "DSP brain" of the project. It sits between the container I/O (FFmpeg and WAV reading/writing) and the final output video remuxing. The signal flow is:

- **Input:** time-domain audio x (mono float array) and sampling rate sr.
- (Optional) **High-pass filtering:** remove rumble / low-frequency drift via `biquad_highpass`.
- **STFT:** convert to time–frequency matrix $X[t, k]$ via `stft`.
- **Magnitude + phase split:** keep magnitude for denoising decisions, keep phase for reconstruction.
- **Noise PSD tracking:** estimate background noise power spectrum $N[k]$.
- **Gain computation:** compute suppression gain $G[t, k]$ for each time frame and frequency bin.
- **Apply gain:** create denoised spectrum $Y[t, k]$.
- **ISTFT:** reconstruct the denoised time-domain signal via `istft`.

Verbal meaning

Importantly, the denoising is **not** performed by modifying raw samples directly. Instead, it works on the **spectral power** of each short-time frame. This allows the algorithm to reduce noise that is concentrated in certain frequency regions while leaving other parts relatively unchanged.

Parameter meaning (beginner-friendly)

Verbal meaning

- n_{fft}: STFT frame size (how many samples are analysed at once).
- H: hop size (how far the frame moves each step; smaller hop = more overlap).

- `init_noise_seconds`: amount of early audio used to initialise the noise estimate.
- `noise_update`: controls how slowly or quickly the noise estimate adapts over time.
- `dd_alpha`: "decision-directed" smoothing factor used to stabilise SNR estimates across frames.
- `gain_smooth`: temporal smoothing applied directly to the gain G to avoid rapid fluctuations.
- `oversub`: exponent that increases suppression strength (more aggressive when > 1).
- `floor_base`: sets a minimum gain floor so the algorithm does not fully erase bins (reduces artefacts).
- `hp_fc`: high-pass cutoff frequency used to remove low-frequency rumble before STFT denoising.

Step 1: optional high-pass preprocessing

Verbal meaning

Code:

```
if hp_fc is not None and hp_fc > 0:  x = biquad_highpass(x, sr, fc=hp_fc)
```

Verbal meaning

This step removes very low-frequency energy (e.g., rumble) that can dominate energy statistics and interfere with noise estimation. The output of `biquad_highpass` is still a time-domain signal, but with low-frequency content reduced.

Step 2: STFT conversion + magnitude/phase separation

Verbal meaning

Code:

```
X, win, out_len = stft(x, n_fft=n_fft, hop=hop)
mag = np.abs(X)
phase = np.angle(X)
P = mag * mag
```

Verbal meaning

The STFT produces a complex matrix $X[t, k]$ where:

- t indexes the time frame,
- k indexes the frequency bin,
- the magnitude $|X[t, k]|$ represents "how much energy exists at that frequency at that time,"
- the phase $\angle X[t, k]$ represents timing/shift information needed for accurate reconstruction.

Verbal meaning

The denoising decisions are made in the **power domain**, not the raw magnitude domain. Power is computed as:

$$P[t, k] = |X[t, k]|^2$$

This makes ratios like "signal-to-noise" more natural to compute, because power ratios correspond directly to energy comparisons.

Formal definition

$$P[t, k] = |X[t, k]|^2$$

Where,

- $X[t, k]$ — complex STFT coefficient at frame t, frequency bin k
- $|X[t, k]|$ — magnitude of the complex coefficient

- $P[t, k]$ — power estimate at frame t, frequency bin k

Step 3: define frequency axis (for frequency-shaped floors)

Verbal meaning

Code:

```
freqs = np.linspace(0.0, sr/2.0, n_bins)
```

Verbal meaning

This creates a vector of actual frequency values (in Hz) for each frequency bin. The highest frequency represented is $sr/2$ (the Nyquist frequency). This frequency axis is later used to create a frequency-dependent minimum gain floor, because noise behaviour often differs across low vs high frequencies.

Step 4: initialise the noise power spectrum $N[k]$

Verbal meaning

Code:

```
init_frames = int(max(1, (init_noise_seconds * sr - n_fft) // hop))
init_frames = min(init_frames, n_frames)
Npsd = median(P[:init_frames, :], axis=0) + 1e-12
```

Verbal meaning

The algorithm needs an initial estimate of the **noise power** at each frequency bin. It assumes that the early portion of the recording can serve as a baseline noise reference (or at least contains representative noise).

It uses the **median across frames** for each frequency bin. Median is more robust than mean when there are occasional strong events (e.g., a brief speech burst), because a median is less affected by outliers.

A small constant 10^{-12} is added to prevent division by zero later.

Formal definition

$$N[k] = \mathrm{median}_{t\in\{0,\ldots,T_0-1\}}\left(P[t,k]\right) + \epsilon$$

Where,

- $N[k]$ — initial noise power estimate at frequency bin k
- $P[t, k]$ — observed power at frame t, bin k
- T_0 — number of initial frames used for noise estimation
- ϵ — small constant to prevent division by zero

Step 5: construct a frequency-shaped minimum gain floor

Verbal meaning

Code:

```
f_norm = freqs / (sr/2 + 1e-12)
floor_shape = 0.7 + 0.6 * sqrt(f_norm)
```

Verbal meaning

This creates a simple "shape" over frequency that will later scale the minimum gain floor. The floor is **not flat across frequency**. Instead, it increases gradually with frequency (via $\sqrt{f}$). This helps avoid excessively strong suppression at higher frequencies, where aggressive denoising can create unnatural artefacts.

Formal definition

$$f_{\text{norm}}[k] = \frac{f[k]}{f_{\text{Nyq}} + \epsilon}$$

Where,

- $f[k]$ — frequency (Hz) associated with bin k
- f_{Nyq} — Nyquist frequency ($sr/2$)
- $f_{\text{norm}}[k]$ — frequency normalised to $[0, 1]$
- ϵ — small constant to prevent division by zero

Formal definition

$$\text{floor\_shape}[k] = 0.7 + 0.6\sqrt{f_{\text{norm}}[k]}$$

Where,

- $\text{floor\_shape}[k]$ — frequency-dependent shaping term
- $f_{\text{norm}}[k]$ — normalised frequency value

Step 6: initialise smoothing state variables

Verbal meaning

Code:

```
    prev_G = ones(n_bins)
    prev_post_snr = ones(n_bins)
    prev_gain = ones(n_bins)
```

Verbal meaning

These arrays store information from the previous frame:

- `prev_G`: previous gain (used in decision-directed SNR estimation).
- `prev_post_snr`: previous posterior SNR estimate.
- `prev_gain`: previous smoothed gain used for temporal smoothing.

Verbal meaning

Without these stored states, the algorithm would make every decision independently per frame, which typically causes rapidly fluctuating gains and "musical noise" artefacts.

Step 7: per-frame denoising loop

Verbal meaning

The loop processes each time frame t and computes a gain value $G[t, k]$ for every frequency bin k.

7.1 Compute posterior SNR γ

Verbal meaning

Code:

```
    gamma = Pt / Npsd
```

Verbal meaning

Here Pt is the observed power spectrum in the current frame. Dividing by the current noise estimate produces a ratio:

"If this bin is 1.0, the bin matches noise. If it is much larger than 1.0, there is likely signal present."

Formal definition

$$\gamma[t,k] = \frac{P[t,k]}{N[k]}$$

Where,

- $P[t,k]$ — observed power at time frame t, bin k
- $N[k]$ — estimated noise power at bin k
- $\gamma[t,k]$ — posterior SNR estimate

7.2 Compute decision-directed prior SNR ξ

Verbal meaning

Code:

`xi = dd_alpha * (prev_G`$^2) * prev\_post\_snr + (1 - dd\_alpha) * max(gamma - 1, 0)$

Verbal meaning

The posterior SNR γ can change rapidly from frame to frame. To stabilise the estimate, the algorithm computes a **prior SNR** ξ using a **decision-directed** method:

- one part comes from the previous frame's behaviour (smoothed memory),
- one part comes from the current frame's instantaneous evidence.

Formal definition

$$\xi[t,k] = \alpha_{\mathrm{DD}}\; G[t-1,k]^2\, \gamma[t-1,k] + (1-\alpha_{\mathrm{DD}})\; \max(\gamma[t,k]-1, 0)$$

Where,

- $\xi[t,k]$ — prior SNR estimate at frame t, bin k
- α_{DD} — decision-directed smoothing parameter (`dd_alpha`)
- $G[t-1,k]$ — gain from the previous frame
- $\gamma[t-1,k]$ — posterior SNR from the previous frame
- $\gamma[t,k]$ — posterior SNR from the current frame

7.3 Convert ξ to a Wiener-style gain G

Verbal meaning

Code:

```
G = xi / (xi + 1)
```

Verbal meaning

This gain is close to 1 when ξ is large (signal dominates noise), and close to 0 when ξ is small (noise dominates signal). This is a classical Wiener-filter style suppression rule.

Formal definition

$$G[t,k] = \frac{\xi[t,k]}{\xi[t,k]+1}$$

Where,

- $G[t,k]$ — suppression gain (0 = remove, 1 = keep)
- $\xi[t,k]$ — prior SNR estimate

7.4 Oversubtraction and temporal smoothing

Verbal meaning

Code:

```
G = (G^{oversub})
G = gain_smooth * prev_gain + (1-gain_smooth)*G
```

Verbal meaning

Raising G to a power > 1 makes suppression stronger (more aggressive). Smoothing then reduces fast fluctuations over time.

Formal definition

$$G'[t,k] = (G[t,k])^p$$

Where,

- $G[t,k]$ — initial gain
- $G'[t,k]$ — oversubtracted gain
- p — oversubtraction exponent (`oversub`)

Formal definition

$$\tilde{G}[t,k] = \lambda\,\tilde{G}[t-1,k] + (1-\lambda)\,G'[t,k]$$

Where,

- $\tilde{G}[t,k]$ — temporally smoothed gain
- λ — gain smoothing factor (`gain_smooth`)
- $G'[t,k]$ — oversubtracted gain

7.5 Apply a minimum gain floor

Verbal meaning

Code:

```
floor = clip(floor_base * floor_shape, 0.02, 0.25)
G = max(G, floor)
```

Verbal meaning

The gain is never allowed to fall below a floor. This prevents “complete deletion” of bins, which can create unnatural artefacts. The floor is frequency-shaped, meaning different frequency regions have slightly different minimum allowed gains.

7.6 Update noise estimate $N[k]$

Verbal meaning

Code:

```
speech_prob = clip((gamma-1)/5, 0, 1)
beta = noise_update + (1-noise_update)*speech_prob
Npsd = beta*Npsd + (1-beta)*Pt
```

Verbal meaning

If a bin looks like it contains strong signal (large γ), the algorithm updates the noise estimate more slowly (so it does not accidentally “learn speech as noise”). If a bin looks like mostly noise, it updates more quickly.

7.7 Reconstruct complex spectrum using magnitude + phase

Verbal meaning

Code:

```
Ymag = sqrt(Pt) * G
Y[t] = Ymag * (cos(phase) + j sin(phase))
```

Verbal meaning

This is the step where the **cosine and sine appear**. The phase stored earlier is used to rebuild the complex number representing each frequency bin. A complex number can be written in polar form:

$$re^{j\theta} = r(\cos\theta + j\sin\theta)$$

So the code is reconstructing the complex STFT coefficient using:

- a new magnitude $Ymag$ (after suppression),
- the original phase $\theta = \text{phase}[t, k]$.

Formal definition

$$Y[t, k] = |Y[t, k]| \left(\cos(\theta[t, k]) + j\sin(\theta[t, k])\right)$$

Where,

- $Y[t, k]$ — denoised complex STFT coefficient
- $|Y[t, k]|$ — denoised magnitude (after applying gain)
- $\theta[t, k]$ — original phase from input STFT
- j — imaginary unit

Step 8: ISTFT reconstruction

Verbal meaning

Finally, the denoised STFT matrix Y is converted back to a time-domain waveform using `istft`. This returns a denoised signal with the same target length as the original.

Reasoning

Separating magnitude modification from phase preservation is a standard choice in classical spectral denoising. Modifying phase directly is difficult and often introduces audible artefacts, whereas retaining the original phase allows stable reconstruction while still achieving meaningful noise suppression through magnitude attenuation.

1.9. `denoise_video_mp4(input_mp4, output_dir, sample_rate)`

Verbal meaning

Implements the end-to-end wrapper that applies the DSP denoising algorithm to the audio track of a video file. The function performs three deterministic stages: (1) extract audio from the input MP4 into a temporary PCM16 WAV file, (2) run the from-scratch denoiser on the WAV audio, and (3) remux the cleaned audio back into the original video container while copying the original video stream unchanged.

Verbal meaning

The process is executed using a temporary working directory so that intermediate audio files are not left on disk after completion. The output video is written to `output_dir` with a fixed filename, enabling reproducible example generation and consistent downstream evaluation.

Author's Note

This document serves as a personal technical record intended to support future recall and reconstruction of the analytical tools, formal definitions, mathematical formulations, and methodological workflows adopted within this project. The preparation of this document was heavily assisted by ChatGPT (OpenAI) especially with respect to for drafting, formatting of LaTeXsupport (ensurning formula accuracy) and hyper condensation + thorough explaination of content **specifically in the way I understand best.**