SMPTE ST 2042-1 (VC-2) Quantisation Matrix Computation Routines

Release 0.1.2

BBC

Mar 19, 2021

CONTENTS

The *vc2_quantisation_matrices* package, provides both a standalone software tool and Python module for computing 'default' quantisation matrices for the SMPTE ST 2042-1 VC-2 professional video codec. Specifically, this software implements the procedure from section (D.3.2) for computing quantisation matrices which achieve noise-power normalisation.

You can find the source code for *vc2_quantisation_matrices* on GitHub.

Note: This documentation is also available to browse online in HTML format.

CHAPTER

ONE

VC2-MAKE-QUANTISATION-MATRIX STANDALONE LINE TOOL

The `vc2-make-quantisation-matrix` command line application computes noise power normalised quantisation matrices for arbitrary VC-2 filter configurations. This utility is installed alongside the *vc2_quantisation_matrices* module.

The utility expects the following arguments:

- `--wavelet-index`: The index of the wavelet transform used for vertical filtering (as enumerated by (Table 12.1) in the VC-2 specification).
- `--wavelet-index-ho`: The index of the wavelet transform used for horizontal filtering. (If not given, the same wavelet specified by `--wavelet-index` is assumed).
- `--dwt-depth`: The number of 2D transform levels applied.
- `--dwt-depth-ho`: The number of horizontal-only transform levels to apply. (If not given, defaults to 0.)

For example, the following invocation computes the quantisation matrix for a 4-level 2D LeGall transform:

```
$ vc2-make-quantisation-matrix --wavelet-index 1 --dwt-depth 4
Level 0: LL:  4
Level 1: HL:  2, LH:  2, HH:  0
Level 2: HL:  4, LH:  4, HH:  2
Level 3: HL:  5, LH:  5, HH:  3
Level 4: HL:  7, LH:  7, HH:  5
```

CHAPTER

TWO

VC2_QUANTISATION_MATRICES PYTHON MODULE USAGE

The *vc2_quantisation_matrices* may be used to create quantisation matrices which normalise noise power between transform bands. This may be done using the following function:

derive_quantisation_matrix(*wavelet_index*, *wavelet_index_ho*, *dwt_depth*, *dwt_depth_ho*)

Derive a noise-power normalising quantisation matrix for the specified wavelet transform. This quantisation matrix will seek to cause quantisation noise-power to be spread evenly over all wavelet levels and bands.

Parameters

wavelet_index, wavelet_index_ho [vc2_data_tables.WaveletFilters ([vc2_data_tables], page 7)] The vertical and horizontal wavelet filter indices respectively.

dwt_depth, dwt_depth_ho The wavelet transform depth (2D depth and horizontal only depth respectively).

Returns

{level: {band: int, …}, …} A quantisation matrix, as laid out by the quant_matrix VC-2 pseudocode function (12.4.5.3) and in vc2_data_tables.QUANTISATION_MATRICES ([vc2_data_tables], page 8).

As an example, the quantisation matrix for a 4-level 2D LeGall transform may be found as follows:

```
>>> from vc2_data_tables import WaveletFilters
>>> from vc2_quantisation_matrices import derive_quantisation_matrix

>>> matrix = derive_quantisation_matrix(
...     wavelet_index=WaveletFilters.le_gall_5_3,     # 1
...     wavelet_index_ho=WaveletFilters.le_gall_5_3,  # 1
...     dwt_depth=4,
...     dwt_depth_ho=0,
... )

>>> from pprint import pprint
>>> pprint(matrix)
{0: {'LL': 4},
 1: {'HH': 0, 'HL': 2, 'LH': 2},
 2: {'HH': 2, 'HL': 4, 'LH': 4},
 3: {'HH': 3, 'HL': 5, 'LH': 5},
 4: {'HH': 5, 'HL': 7, 'LH': 7}}
```

Tip: The vc2_data_tables ([vc2_data_tables], page 3) module, used here, provides named constants for all of the VC-2 filter types.

The returned nested dictionary structure matches the layout used by the quant_matrix() VC-2 pseudocode function (12.4.5.3). For each transform level, the outer dictionary contains an entry with an inner dictionary giving the quantisation index offset for each orientation. Orientations are labelled using the strings 'L', 'H', 'LL' 'LH', 'HL', and 'HH'.

CHAPTER

THREE

DERIVATION & IMPLEMENTATION

The (heavily annotated) `vc2_quantisation_matrices` module implements the procedure required to compute quantisation matrices for arbitrary combinations of VC-2 filters and transform depths.

If you're not interested in the details, you can skip directly to the convenience function for computing noise-power normalising quantisation matrices: `derive_quantisation_matrix()`.

3.1 Motivation/background

VC-2 achieves lossy compression by quantizing wavelet transform coefficients. This quantisation introduces errors (noise) into the transformed signal. When a picture is later synthesised from these transform values, this picture too will have added noise.

The transformed signal is broken up into several individual bands, approximately corresponding to different spatial frequency components. Noise added to each of these bands has a differing effect on the final picture. Depending on the specific filter in use, a certain amount of noise added in one band may have a greater impact on the final picture than the same noise added to a different band.

As a result, a uniform source of noise (e.g. quantisation) can result in a skewed distribution of noise in the resulting picture (e.g. excessive low- or high-frequency noise with little in other spatial frequencies). This is undesirable since the noise level at some spatial frequencies will become much higher than it otherwise would be.

VC-2's quantisation matrices allow the quantisation levels in different transform bands to be adjusted relative to each other. In this way, for example, bands which are very sensitive to noise can be assigned lower quantisation levels than bands which are relatively insensitive.

The default quantisation matrices provided with VC-2 are chosen such that quantisation noise results in noise which is evenly spread across the frequency spectrum in the synthesised picture. It is the calculation of these matrices which is the focus of this module.

It is worth emphasising that the default quantisation matrices are *not* intended to exploit psycho-visual phenomena (for example by preserving low-frequency components at the expense of higher frequencies). VC-2 users are free to define custom quantisation matrices which exploit these phenomena if required, but this will not be discussed further here.

3.2 Implementation

This module performs all of the necessary steps required to compute quantisation matrices for VC-2's wavelet filters. Internally the SymPy computer algebra system is used for all calculations. This means that all operations are carried out symbolically in much the same way they would be performed on paper.

3.3 Filter noise gain

The noise-gain of a FIR filter with coefficients $h_1, \ldots, h_m$ is:

$$\sqrt{\sum_M h_m^2}$$

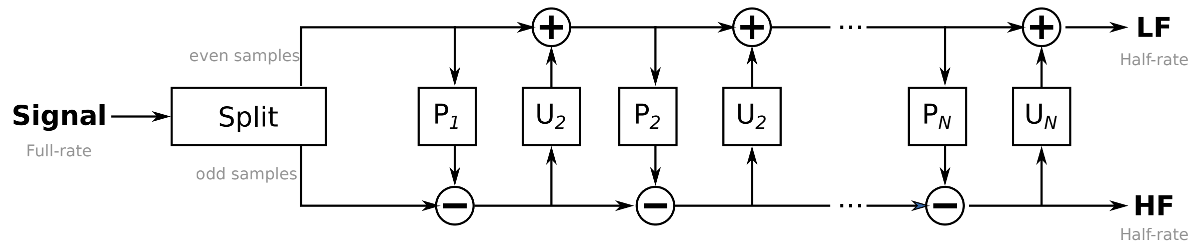
This figure indicates the gain the filter will introduce to a white-noise signal. This function is implemented as:

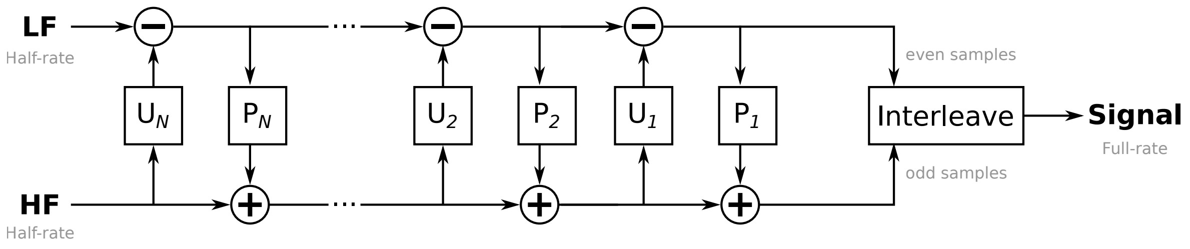
`fir_filter_noise_gain`(*coefficients*)
Compute the noise-gain of a FIR filter with the specified list of filter coefficients.

If we make the (mostly reasonable) assumption that quantisation introduces white noise, it is the filter noise gains (of the synthesis wavelet filters) which our quantisation matrix must attempt to even out. To be able to do this we need to find the FIR filter coefficients which are to be fed to `fir_filter_noise_gain()`.

3.4 From lifting to classical filters

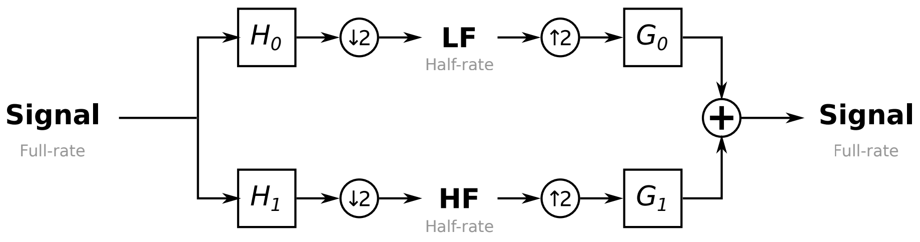
For reasons of efficiency and perfect reconstruction, the VC-2 wavelet filters are specified in terms of lifting operations:





This figure shows both the analysis (picture to transform coefficients) and synthesis (transform coefficients to picture) filtering processes. Both filters are defined by N update (U_n) and predict (P_n) stages which operate on sub-sampled signals. In most (but not all) of the VC-2 filters, only a single predict and update pair is used (i.e. $N = 1$).

By contrast, the `fir_filter_noise_gain()` function requires our filters to be defined as classical Finite Impulse Response (FIR) filters. That is, we must transform the picture above into the one below:

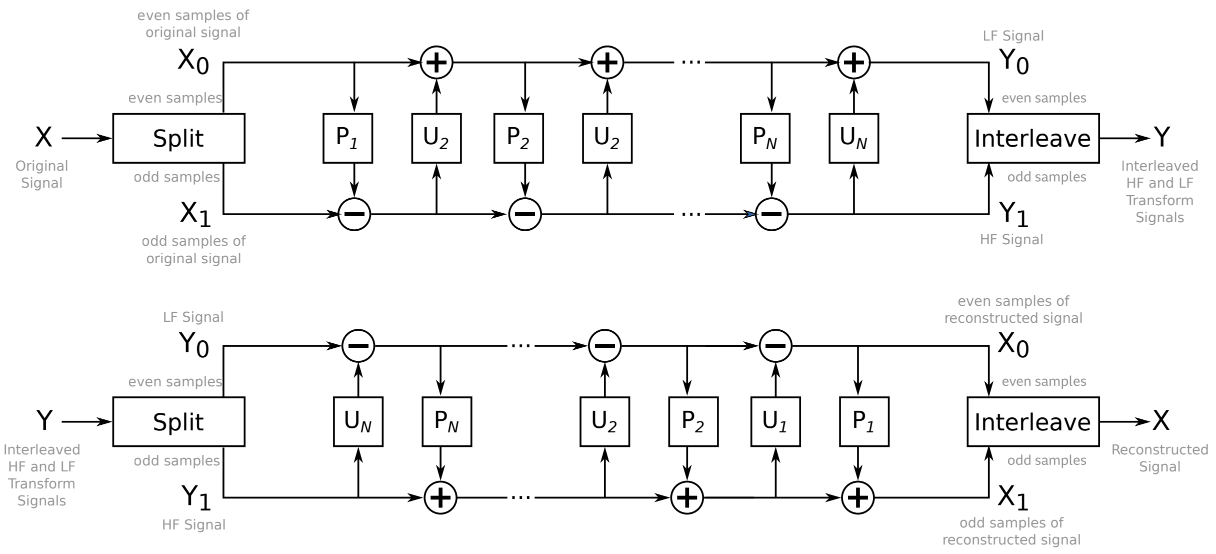


3.4.1 Matrix form

The first step in transforming the lifting filter representation into classical form is producing a matrix representation for the lifting procedure.

In this section we'll use the z-domain representation of all the filters and signals involved. (See section 7.1 in "Ripples in Mathematics" for a targeted, whirl-wind introduction.)

The figure below shows the lifting representation of the analysis (top) and synthesis (bottom) filters again, additionally labelled according to the convention used here:



Using a z-domain representation then our picture signal, $X(z)$, is split into even ($X_0(z)$) and odd ($X_1(z)$) samples:

$$\begin{array}{ll} X_0(z) = \sum_n X[2n]z^{-n} & \text{Even samples} \\ X_1(z) = \sum_n X[2n+1]z^{-n} & \text{Odd samples} \end{array}$$

Likewise the transform signal is made up of a Low Frequency (LF) component, $Y_0(z)$ and a High Frequency (HF) component $Y_1(z)$. We define $Y(z)$ to be an interleaving of these two signals where the LF component makes up the even samples and the HF component the odd samples:

$$\begin{array}{ll} Y_0(z) = \sum_n Y[2n]z^{-n} & \text{LF samples (even)} \\ Y_1(z) = \sum_n Y[2n+1]z^{-n} & \text{HF samples (odd)} \end{array}$$

The resulting z-domain matrix forms of the analysis and synthesis lifting processes respectively are:

$$\begin{bmatrix} Y_0(z) \\ Y_1(z) \end{bmatrix} = \begin{bmatrix} 1 & -U_N(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -P_N(z) & 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & -U_2(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -P_2(z) & 1 \end{bmatrix} \begin{bmatrix} 1 & -U_1(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -P_1(z) & 1 \end{bmatrix} \begin{bmatrix} X_0(z) \\ X_1(z) \end{bmatrix}$$

$$\begin{bmatrix} X_0(z) \\ X_1(z) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ P_1(z) & 1 \end{bmatrix} \begin{bmatrix} 1 & U_1(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ P_2(z) & 1 \end{bmatrix} \begin{bmatrix} 1 & U_2(z) \\ 0 & 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & 0 \\ P_N(z) & 1 \end{bmatrix} \begin{bmatrix} 1 & U_N(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} Y_0(z) \\ Y_1(z) \end{bmatrix}$$

Where $U_n(z)$ and $P_n(z)$ are the z-transform representations of the lifting step filters. These functions can be found for a given wavelet transform using:

`lifting_stage_to_z_transform`(*stage*)

Given a `vc2_data_tables.LiftingStage` ([vc2_data_tables], page 7) describing wavelet filter stage, return the type of lifting stage (either predict or update) and a z-domain representation of the filtering operation as used in the matrix filter representation.

Parameters

stage [`vc2_data_tables.LiftingStage` ([vc2_data_tables], page 7)]

Returns

stage_type [*`StageType`*]

z_transform

`class StageType`(*value*)
Lifting stage type specifier.

If the left-most parts of the above matrices are multiplied together into 2×2 matrices: $\mathbf{H}(z)$ (the analysis filter in matrix form) and $\mathbf{G}(z)$ (the synthesis filter in matrix form) yielding:

$$\begin{bmatrix} Y_0(z) \\ Y_1(z) \end{bmatrix} = \mathbf{H}(z) \begin{bmatrix} X_0(z) \\ X_1(z) \end{bmatrix}$$
$$\begin{bmatrix} X_0(z) \\ X_1(z) \end{bmatrix} = \mathbf{G}(z) \begin{bmatrix} Y_0(z) \\ Y_1(z) \end{bmatrix}$$

For the analysis filter and synthesis filters respectively.

The following function may be used to convert a `vc2_data_tables.LiftingFilterParameters` ([vc2_data_tables], page 7) into a 2×2 matrix.

`wavelet_filter_to_matrix_form`(*lifting_filter_parameters*)
Convert a `vc2_data_tables.LiftingFilterParameters` ([vc2_data_tables], page 7) filter specification into z-domain matrix form.

Note: All of the wavelet specifications in the VC-2 specification (and therefore in `vc2_data_tables.LIFTING_FILTERS` ([vc2_data_tables], page 7)) define *synthesis* filter lifting stages. As suggested by the figures above, these are easily converted into analysis filter specifications by reversing the order and changing the operation. The following function may be used to convert between analysis and synthesis lifting filters:

`convert_between_synthesis_and_analysis`(*lifting_filter_parameters*)
Given a synthesis wavelet filter specification, return the complementary analysis filter (or visa versa).

Parameters

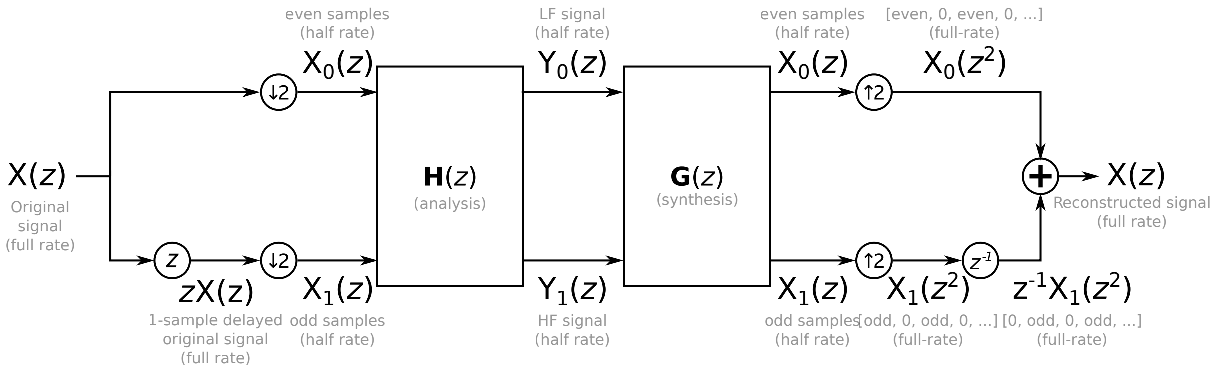
lifting_filter_parameters [`vc2_data_tables.LiftingFilterParameters` ([vc2_data_tables], page 7)]

Returns

lifting_filter_parameters [`vc2_data_tables.LiftingFilterParameters` ([vc2_data_tables], page 7)]

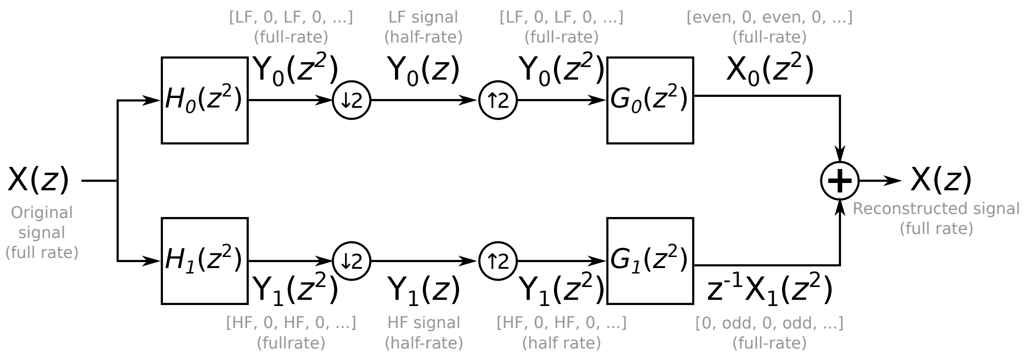
3.4.2 Matrix form to classical form

The matrix form representation achieved above implements the following (slightly more formally illustrated, this time) analysis/synthesis filtering processes:



In this new diagram, the 'split' and 'interleave' processes are shown in terms of their z-domain operations.

From the matrix based representation (where our filters are defined by the matrices $\mathbf{H}(z)$ (analysis) and $\mathbf{G}(z)$ (synthesis) we now wish to decompose this into the classical form below:

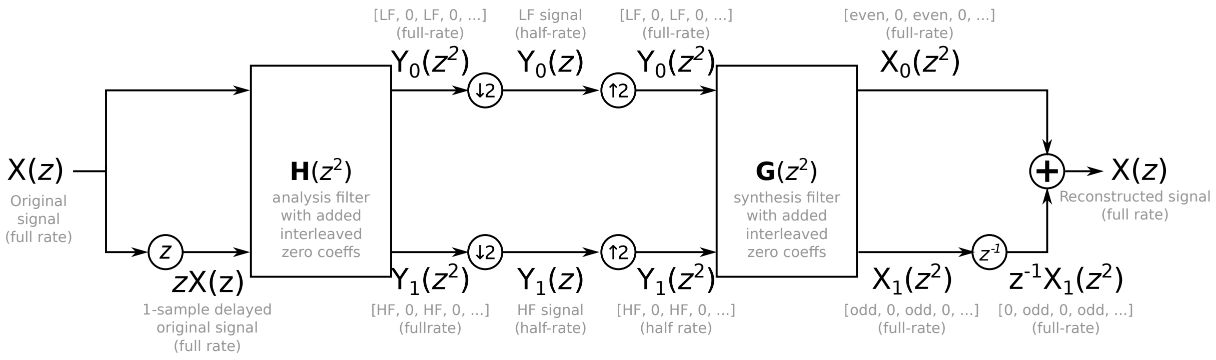

[LF, 0, LF, 0, ...] (full-rate)
LF signal (half-rate)
[LF, 0, LF, 0, ...] (full-rate)
[even, 0, even, 0, ...] (full-rate)
Y0(z2)
Y0(z)
Y0(z2)
X0(z2)
H0(z2)
↓2
↑2
G0(z2)
X(z)
Original signal (full rate)
X(z)
Reconstructed signal (full rate)
H1(z2)
↓2
↑2
G1(z2)
Y1(z2)
Y1(z)
Y1(z2)
z-1X1(z2)
[HF, 0, HF, 0, ...] (fullrate)
HF signal (half-rate)
[HF, 0, HF, 0, ...] (half rate)
[0, odd, 0, odd, ...] (full-rate)

In this representation, the analysis filter is defined by $H_0(z^2)$ and $H_1(z^2)$ and synthesis filter is defined by $G_0(z^2)$ and $G_1(z^2)$.

Note: For those new to the z-transform, for some signal $[a_0, a_1, a_2, \ldots]$, whose z-transform is $A(z) = a_0z^0 + a_1z^{-1} + a_2z^{-2} + \ldots$ then $A(z^2) = a_0z^0 + a_1z^{-2} + a_2z^{-4} + \ldots$ which is equivalent to a signal $[a_0, 0, a_1, 0, a_2, 0, \ldots]$.

Full-rate filter matrix

The first step is to modify the $\mathbf{H}(z)$ and $\mathbf{G}(z)$ filters to work on full-rate signals (i.e. to move the decimation step after analysis or before synthesis, as illustrated below:


[LF, 0, LF, 0, ...] (full-rate)
LF signal (half-rate)
[LF, 0, LF, 0, ...] (full-rate)
[even, 0, even, 0, ...] (full-rate)
Y0(z2)
Y0(z)
Y0(z2)
X0(z2)
X(z)
Original signal (full rate)
H(z2)
analysis filter with added interleaved zero coeffs
↓2
↑2
G(z2)
synthesis filter with added interleaved zero coeffs
X(z)
Reconstructed signal (full rate)
z
z-1
zX(z)
1-sample delayed original signal (full rate)
Y1(z2)
Y1(z)
Y1(z2)
X1(z2)
z-1X1(z2)
[HF, 0, HF, 0, ...] (fullrate)
HF signal (half-rate)
[HF, 0, HF, 0, ...] (half rate)
[odd, 0, odd, 0, ...] (full-rate)
[0, odd, 0, odd, ...] (full-rate)

The modification is straight-forward – the filter coefficients are interleaved with zeros; yielding the filters $\mathbf{H}(z^2)$ and $\mathbf{G}(z^2)$ for the analysis and synthesis stages respectively.

If we ignore the decimation and upsampling steps in the diagram above (which now directly cancel eachother out) we get the following matrix representation:

$$\begin{bmatrix} Y_0(z^2) \\ Y_1(z^2) \end{bmatrix} = \mathbf{H}(z^2) \begin{bmatrix} X(z) \\ zX(z) \end{bmatrix} \qquad \text{Analysis filter}$$

$$\begin{bmatrix} X_0(z^2) \\ X_1(z^2) \end{bmatrix} = \mathbf{G}(z^2) \begin{bmatrix} Y_0(z^2) \\ Y_1(z^2) \end{bmatrix} \qquad \text{Synthesis filter}$$

These can be written in expanded form like so:

$$\begin{bmatrix} Y_0(z^2) \\ Y_1(z^2) \end{bmatrix} = \begin{bmatrix} H_{00}(z^2) & H_{01}(z^2) \\ H_{10}(z^2) & H_{11}(z^2) \end{bmatrix} \begin{bmatrix} X(z) \\ zX(z) \end{bmatrix} \qquad \text{Analysis filter}$$

$$\begin{bmatrix} X_0(z^2) \\ X_1(z^2) \end{bmatrix} = \begin{bmatrix} G_{00}(z^2) & G_{01}(z^2) \\ G_{10}(z^2) & G_{11}(z^2) \end{bmatrix} \begin{bmatrix} Y_0(z^2) \\ Y_1(z^2) \end{bmatrix} \qquad \text{Synthesis filter}$$

Deriving the classical analysis filter

Rewriting the matrix form of the analysis filter as two equations and rearranging:

$$\begin{aligned} Y_0(z^2) &= H_{00}(z^2)X(z) + zH_{01}(z^2)X(z) \\ &= \big(H_{00}(z^2) + zH_{01}(z^2)\big)X(z) \end{aligned}$$

$$\begin{aligned} Y_1(z^2) &= H_{10}(z^2)X(z) + zH_{11}(z^2)X(z) \\ &= \big(H_{10}(z^2) + zH_{11}(z^2)\big)X(z) \end{aligned}$$

This leads us to the following expressions for the classical analysis filter representations:

$$\begin{aligned} H_0(z^2) &= H_{00}(z^2) + zH_{01}(z^2) \\ H_1(z^2) &= H_{10}(z^2) + zH_{11}(z^2) \end{aligned}$$

Deriving the classical synthesis filter

Next, we repeat the same process of producing a formulaic representation of the matrix equation:

$$X_{\text{even}}(z) = G_{00}(z^2)Y_0(z^2) + G_{01}(z^2)Y_1(z^2)$$

$$zX_{\text{odd}}(z) = G_{10}(z^2)Y_0(z^2) + G_{11}(z^2)Y_1(z^2)$$

In the diagrams we have defined $X(z) = X_0(z^2) + z^{-1}X_1(z^2)$. Substituting the formulae above into this expression and then rearranging we get:

$$\begin{aligned} X(z) &= X_0(z^1) + z^{-1}X_1(z^2) \\ &= G_{00}(z^2)Y_0(z^2) + G_{01}(z^2)Y_1(z^2) + z^{-1}\big(G_{10}(z^2)Y_0(z^2) + G_{11}(z^2)Y_1(z^2)\big) \\ &= G_{00}(z^2)Y_0(z^2) + G_{01}(z^2)Y_1(z^2) + z^{-1}G_{10}(z^2)Y_0(z^2) + z^{-1}G_{11}(z^2)Y_1(z^2) \\ &= \big(G_{00}(z^2) + z^{-1}G_{10}(z^2)\big)Y_0(z^2) + \big(G_{01}(z^2) + z^{-1}G_{11}(z^2)\big)Y_1(z^2) \end{aligned}$$

From this we get the following expressions for the classical filter bank representation.

$$\begin{aligned} G_0(z^2) &= G_{00}(z^2) + z^{-1}G_{10}(z^2) \\ G_1(z^2) &= G_{01}(z^2) + z^{-1}G_{11}(z^2) \end{aligned}$$

Implementation

The steps above which convert from the matrix representation of a filter to classical filters are implemented as:

`analysis_matrix_to_classical_form`(*H*)

Given an analysis filter matrix, $\mathbf{H}(z)$ as produced by, e.g. *`wavelet_filter_to_matrix_form()`*, return the equivalent pair of classical filters, $H_0(z^2)$ and $H_1(z^2)$.

`synthesis_matrix_to_classical_form`(*G*)

Given an synthesis filter matrix, $\mathbf{G}(z)$ as produced by, e.g. *`wavelet_filter_to_matrix_form()`*, return the equivalent pair of classical filters, $G_0(z^2)$ and $G_1(z^2)$.

The filter coefficients can then be extracted from the resulting algebraic expressions using

`z_to_coeffs`(*poly*)

Get a dictionary `{delay: coeff, ...}` from a z-transform expressed as a polynomial.

The returned dictionary will contain `int` `delay` values and SymPy expressions for the coefficients.

The resulting coefficients may then finally be passed to *`fir_filter_noise_gain()`* to determine the filter noise gain for that filter.

3.4.3 Convenience function

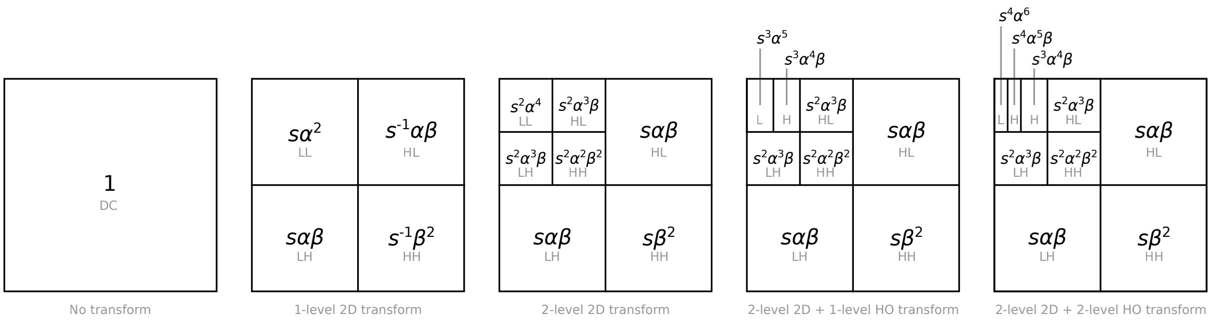
A convenience function is provided which carries out all of the above steps for **synthesis** filters, yielding the low-pass band synthesis filter noise gain (α) and high-pass band synthesis filter noise gain (β).

`wavelet_filter_to_alpha_beta`(*synthesis_lifting_filter_parameters*)
Given synthesis filter definition (in a `vc2_data_tables.LiftingFilterParameters` ([vc2_data_tables], page 7)) return the low-pass and high-pass filter noise gains (α and β).

3.5 Computing quantisation matrices

The α and β values found by `wavelet_filter_to_alpha_beta()` may now be used to create the quantisation matrices for a given transform.

During the VC-2 2D wavelet transform, the filtering process is applied recursively. The consequence of this is that the noise gains accumulate (multiplicatively). This is illustrated below:



The s term is the scaling factor due to the bit shift used by VC-2 between every transform layer. This scaling factor is simply:

$$s = 2^{-\text{bitshift}}$$

And is computed by:

`wavelet_filter_to_synthesis_bit_shift_scale`(*synthesis_lifting_filter_parameters*)
Given synthesis filter definition (in a `vc2_data_tables.LiftingFilterParameters` ([vc2_data_tables], page 7)) return the scaling factor, s, imposed after each 2D or horizontal-only transform level.

Note: When an asymmetric transform is used, the bit shift for the horizontal transform is used (see `filter_bit_shift` (15.4.2)).

The weighting of α, β and s for all bands and levels may be computed automatically using:

`accumulated_noise_gains`(*alpha_v, beta_v, alpha_h, beta_h, s, dwt_depth, dwt_depth_ho*)
Compute the total accumulated noise gain for all bands of a given wavelet transform.

Parameters

alpha_v, beta_v The LF and HF filter noise gains for the vertical wavelet synthesis filter (from, e.g. `wavelet_filter_to_alpha_beta()`).

alpha_h, beta_h The LF and HF filter noise gains for the horizontal wavelet synthesis filter (from, e.g. `wavelet_filter_to_alpha_beta()`).

s The scaling applied by the bit-shift of the horizontal wavelet synthesis filter.

dwt_depth, dwt_depth_ho The wavelet transform depth (and horizontal-only transform depth).

Returns

{level: {band: noise_gain, . . . }, . . . } A list with one dictionary per level in the same layout as the quantisation matrices in `vc2_data_tables.QUANTISATION_MATRICES` ([vc2_data_tables], page 8).

The objective of the quantisation matrix is for quantisation to have the same impact on every band. As such we only care about the relative noise gains. The noise gains computed by *`accumulated_noise_gains()`* can be normalised using:

`normalize_noise_gains`(*noise_gains*)

Normalize a set of noise gains such that the minimum gain is 1.

This operation will be performed symbolically and the resulting noise gains will be SymPy values.

In principle, the values returned by *`normalize_noise_gains()`* should be used to scale the quantisation factors used for each frequency band. In practice, VC-2 specifies quantisation factors via exponential quantisation indices:

$$\text{quantisation factor} = 2^{\text{quantisation index}}/_{4}$$

Therefore, the best approximation to the desired scaling factor is achieved by subtracting from the quantisation index:

$$\text{quantisation index adjustment} = \text{round}(4\log_2(\text{normalised noise gain}))$$

This conversion is performed by:

`normalized_noise_gains_to_quantisation_matrix`(*normalized_noise_gains*)

Given a set of normalised noise gains, returns the equivalent quantisation index adjustments.

All results will be `int`s.

3.6 Convenience function

A convenience function, *`derive_quantisation_matrix()`*, is provided which carries out the entire process described above.

BIBLIOGRAPHY

[vc2_data_tables] The vc2_data_tables manual.

PYTHON MODULE INDEX

INDEX