POLITECNICO
MILANO 1863

052820 - SOUND ANALYSIS, SYNTHESIS AND PROCESSING

Politecnico di Milano

Music and Acoustic Engineering

Digital Audio Analysis and Processing Homework #1: Talking Instrument

Authors:
Gerardo Cicalese (ID: 10776504)
Umberto Derme (ID: 10662564)

Contents

1 Introduction

Cross-synthesis is a technique that involves blending the spectral characteristics of two or more sound signals to create a new synthesized sound.
Our approach is to use the spectral envelope of one signal, known as **modulator** and specifically human speech, to modulate the amplitude of the other signal, known as **carrier**, which should be spectrally richer as musical instruments.
Without introducing math, the process involves:

- Performing short-time analysis of the frequency content of each signal through Short-Time Fourier Transform (STFT)
- Obtaining the spectral envelope of each frame of the modulator with Linear Predictive Coding (LPC)
- Applying the spectral characteristics of the modulator to the carrier through filtering
- Re-synthesizing the resulting hybrid sound through iSTFT

2 Linear Prediction Coding

Given an input signal $s \in \mathbb{R}^N$, linear prediction aims to find a set of **prediction coefficients** $\{a\}_{k=1}^{p}$ such that $\forall n < N$, $s(n)$ can be accurately estimated as a linear combination of the previous p samples:

$$s(n) = e(n) - \sum_{k=1}^{p} a_k s(n-k).$$

Here p denotes the **order** of the linear prediction, and $e(n)$ denotes the **prediction error** at sample n. Computing these linear prediction coefficients requires solving an optimization problem: finding the optimal coefficients a_k which minimize the sum of squared errors $\|e\|^2$.
If we assume that the signal can be modelled as an autoregressive (AR) stochastic process, then $s(n)$ can be expressed as

$$s(n) = \sum_{k=1}^{p} a_k s(n-k) + Gu(n),$$

where G is a gain parameter and $u(n)$ is additive white gaussian noise (AWGN).
Taking the DTFT of both sides:

$$S(\omega) = S(\omega) \sum_{k=1}^{p} a_k e^{j\omega k} + GU(\omega),$$

which lead to the following frequency response function which we define **shaping filter**:

$$H(\omega) \triangleq \frac{S(\omega)}{GU(\omega)} = \frac{1}{1-\sum_{k=1}^{p} a_k e^{j\omega k}} \triangleq \frac{1}{1-P(\omega)} \triangleq \frac{1}{A(\omega)},$$

where we define the **prediction filter**

$$P(\omega) \triangleq \sum_{k=1}^{p} a_k e^{j\omega k},$$

and the **whitening filter**

$$A(\omega) \triangleq = 1 - P(\omega) = 1 - \sum_{k=1}^{p} a_k e^{j\omega k}.$$

We now define the linearly predicted estimate

$$\hat{s}(n) \triangleq \sum_{k=1}^{p} a_k s(n-k).$$

So that the prediction error can be expressed as

$$e(n) \triangleq s(n) - \hat{s}(n) = s(n) - \sum_{k=1}^{p} a_k s(n-k).$$

Taking the DTFT of both sides we obtain

$$E(\omega) = S(\omega) - P(\omega)S(\omega) = (1 - P(\omega))S(\omega) = A(\omega)S(\omega).$$

We can deduce that $e(n)$ is a scaled AWGN:

$$e(n) = Gu(n) \quad \longleftrightarrow \quad E(\omega) = GU(\omega).$$

As $U(\omega) = \sigma^2$ is constant, $E(\omega)$ is constant too, thus the whitening filter $A(\omega)$ has the property of whitening the input $S(\omega)$ (making its spectrum flat). Analogously, the shaping filter $H(\omega)$ has the property of approximating $S(\omega)$ if the input is AWGN:

$$S(\omega) = H(\omega)E(\omega) = \frac{G\sigma^2}{A(\omega)}.$$

The problem of finding the optimal prediction coefficients a_k (which corresponds to the coefficients of the prediction filter $P(\omega)$) can be set up as a Wiener Filtering problem, where $s(n)$ constitutes both the filter input and the desired response.
A simple scheme of a system performing the LPC analysis is shown in Fig. 1.

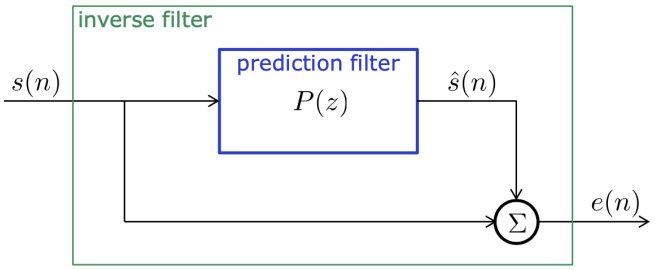


Figure 1: Scheme of a system performing LPC analysis.

3 Wiener-Hopf method

The goal of the Wiener-Hopf method is to compute the optimal linear prediction coefficients. Let $r(i)$ be the auto-correlation function of the input signal:

$$r(i) = \mathrm{E}\{s(n)s(n-i)\}.$$

The Wiener-Hopf equations for the LPC problem are given by

$$\sum_{k=1}^{p} a_k r(i-k) = r(i), \quad i = 1, 2, \ldots, p.$$

The optimal prediction coefficients are found as the solution of these equations, which we express in matrix form:

$$\underline{a} = [R]^{-1}\underline{r},$$

where $\underline{r}$ is the autocorrelation vector, $\underline{a}$ is the LPC coefficient vector, and $[R]$ is a Toeplitz matrix formed from the autocorrelation sequence:

$$[R] = \begin{bmatrix} r(0) & r(1) & \ldots & r(p-1) \\ r(1) & r(0) & \ldots & r(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ r(p-1) & r(p-2) & \ldots & r(0) \end{bmatrix},$$

$$\underline{r} = \begin{pmatrix} r(1) \\ r(2) \\ \vdots \\ r(p) \end{pmatrix},$$

$$\underline{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{pmatrix}.$$

Note that $[R]$ is a Toeplitz matrix, which means that each descending diagonal from left to right is constant. This property greatly simplifies the computation of its inverse and reduces the computational complexity of the Wiener filter algorithm ($\mathcal{O}(p^2)$), as we can use the Levinson-Durbin algorithm.

3.1 Visualizing the shaping filters obtained

We have developed a script, shaping_filter_visualizer.m, which reads an audio file and plots the FFT and the shaping filter of a frame, in order to compare them. Using $L = 1024, M = 512, w\_fun =$ bartlett, $R = L/2$ for the speech signal, the results for frame number 32 are shown in Fig. 2. The filter coefficients have been computed using the Wiener-Hopf equation.

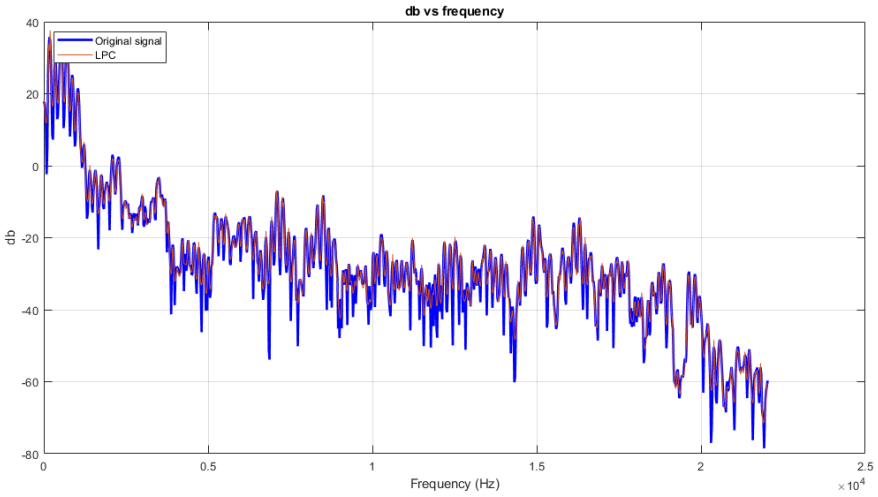


Figure 2: FFT of a frame of the speech signal and associated shaping filter.

4 Gradient descent method

The gradient descent method is a gradient-based adaptation technique defined by an iterative algorithm that, when both the input and the desired signals are stationary, converges to the optimal Wiener solution. Differently from the method described in section 3 the gradient descent gives us a time-varying optimal solution, improving the

tap-weight vector at each iteration. The general scheme for the implementation of this method can be seen in Fig. 3. The procedure can be summarised in a four step process:

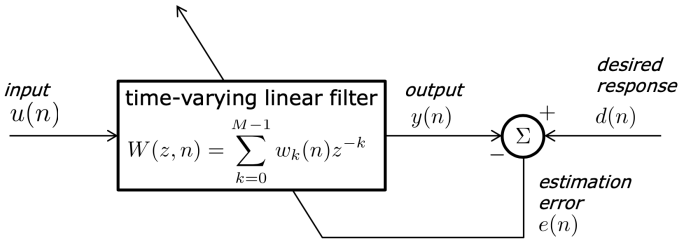


Figure 3: Scheme of a system performing the gradient descent method.

1) Define an initial guess for the tap-weight vector at time zero $\underline{w}(0)$ that provides a starting point where the minimum point of the error-performance surface may be located.

2) Compute the gradient of the cost function $\nabla J\left(\underline{w}(n)\right)$ (that in the Weiner-Hopf was the MSE) using the current guess of $\underline{w}(n)$:

$$\nabla J\left(\underline{w}(n)\right) = -2\underline{p} + 2[R]\underline{w}(n),$$

where the cross-correlation vector $\underline{p}$ and the auto-correlation matrix $[R]$ are defined as in Sec. 3, while the cost function $J(\underline{w})$ is defined as

$$J(\underline{w}(n)) = \sigma_d^2 - \underline{w}^H(n)\underline{p} - \underline{p}^H\underline{w}(n) + \underline{w}^H(n)[R]\underline{w}(n),$$

where σ_d^2 is the variance of the desired output signal.

3) Guess the tap-weight vector at the current time following the direction given by the gradient computed in the previous point:

$$\underline{w}(n+1) = \underline{w}(n) + \frac{1}{2}\mu\left[-\nabla J\left(\underline{w}(n)\right)\right] = \underline{w}(n) + \mu\left[\underline{p} - [R]\underline{w}(n)\right] \quad n = 0, 1, 2, \dots ,$$

where $\mu \in \mathbf{R}$ is the step-size parameter, also known as the learning rate or the step size, that controls the size of the update made to $\underline{w}(n)$ at each iteration of the optimization algorithm.

4) Repeat the process from the second point.

4.1 Stability and convergence

Since we're dealing with a feedback loop, special care must be taken in assuring the stability of the system, that depends on both μ and $[R]$. Since the latter one is usually given, we will focus on the choice of an appropriate step-size parameter. Being the cost function J a second order function the dependence of the cost function on the tap weights can be visualized as a bowl-shaped function as in Fig. 4 so, at every iteration, following the direction opposite to the gradient of the cost function itself the algorithm will converge to the unique minimum of this error-performance surface, provided that the stability is verified. The condition for the stability will be expressed in terms of

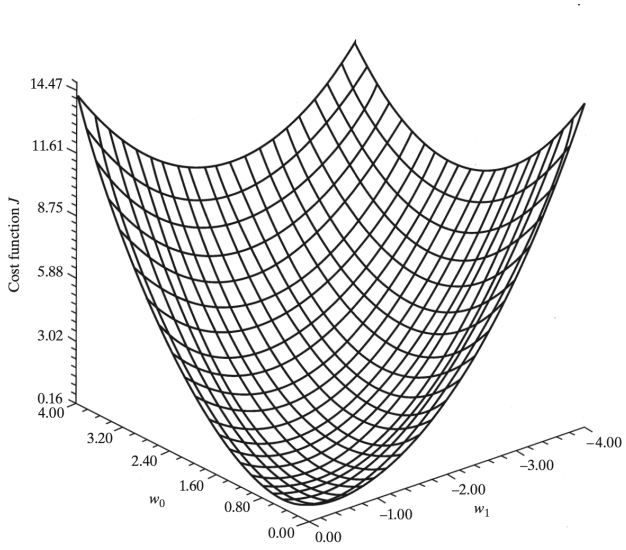


Figure 4: Example of an error performance surface.

natural modes of the algorithm. Firstly we need to use eigendecomposition to the auto-correlation matrix $[R]$

$$[R] = \underline{Q}[\Lambda]\underline{Q}^H,$$

where $\underline{Q}$ is the matrix whose columns are the eigenvectors associated to the eigenvalues that define the diagonal matrix $\underline{\Lambda}$. Furthermore we define a transformed version of the difference between the optimum solution $\underline{w}_o$ and the tap-weight vector $\underline{w}$ as

$$\underline{v} = \underline{Q}^H(\underline{w}(n) - \underline{w}_o),$$

that, after some mathematical passages, allow us to define a new updating rule:

$$\underline{v}(n+1) = ([I] - \mu[\Lambda])\underline{v}(n).$$

Trivially, the initial value of $\underline{v}(0)$ is given by

$$\underline{v}(0) = \underline{Q}^H(\underline{w}(0) - \underline{w}_o).$$

As we're dealing with a diagonal matrix, we can treat each mode separately in the update rule:

$$\underline{v}(n+1) = (1-\mu\lambda_k)\underline{v}_k(n) \quad k = 1, 2, \ldots, M,$$

that allows us easily, knowing $v_k(0)$, to compute the solution

$$v_k(n) = (1-\mu\lambda_k)^n v_k(0).$$

The geometric series in the last equation is convergent if

$$|1-\mu\lambda_k| < 1 \quad \forall k,$$

that represent also the stability condition for the gradient descent method.
The minimum value of the cost function J_{min} can be computed as

$$J_{min} = \sigma_d^2 - \underline{p}^H[R]\underline{p},$$

where σ_d^2 is the variance of the desired output which, in our case, coincides with the input, while the cost function associated to the coefficients vector $\underline{w}$, $J(\underline{w})$, can be computed as

$$J = J_{min} + (\underline{w} - \underline{w}_o)^H[R](\underline{w} - \underline{w}_o).$$

The normalized minimum value of the cost function is

$$\varepsilon \triangleq \frac{J_{min}}{\sigma_d^2}, \quad 0 \leq \varepsilon \leq 1, \tag{1}$$

which, for $\varepsilon = 0$ indicates complete agreement between desired output and obtained output, while for $\varepsilon = 1$ indicates the worst possible situation of no agreement.

4.2 Convergence test

The *gradient_descent_visualizer* script has been implemented to show an example of both the convergence path and the error $J - J_{min}$ (where J is the cost function obtained and J_{min} is its minimum) as a function of the number of iterations.
We have chosen to perform only 50 iterations to simplify the plots. As input signal, we have chosen to use a vector of random numbers uniformly distributed between 1 and -1 and with a random length between $[100, 200]$. In order to obtain 2D plots, we have chosen to perform LPC analysis with LPC order $M = 2$. The convergence path for the gradient descent method that has just been described, with a learning rate $\mu = 0.3\mu_{max}$ ($\mu_{max} = 2/\lambda_{max}$), can be seen in Fig. 5. Having chosen $\mu < \mu_{max}$, the stability condition is verified as seen in Fig. 5. We know that the natural modes λ_k of the steepest descend algorithm follow an exponential decay path as a function of the number of iterations, so an exponential envelope of time constant τ_J can be fitted to the geometric series

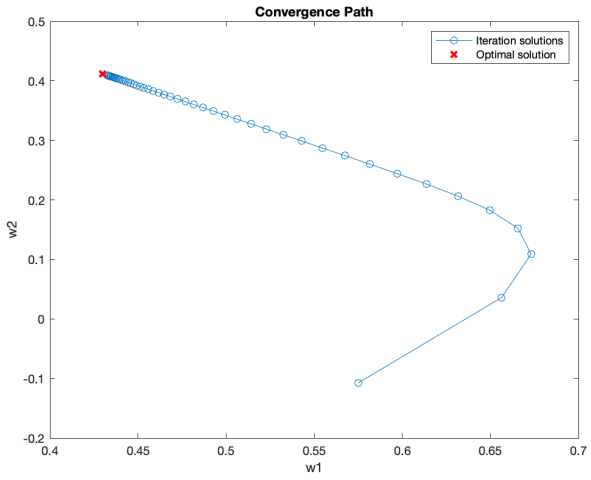


Figure 5: Convergence path of the gradient descent method with a learning rate $\mu = 0.3$.

described in the last equation of Sec. 4, which provides an upper limit for the decay curve:

$$\tau_J = \frac{-1}{2\ln(1-\mu\lambda_{min})}$$

which defines the number of iteration required for the slowest mode (the one with the smaller eigenvalue λ_{min}) to decay of $1/e$ with respect to its initial value. In Fig. 6, we show the experimental curve $J - J_{min}$ as a function of the number of iteration along with the upper limit given by the exponential of time constant τ_J. It's clearly visible how the upper is never overstepped by the exponential trend of the error. The values obtained in the specific case represented in Fig.6 are the following:

```
Jmin = 0.078427833059
J = 0.078427851851
ratio = 1.000000239613
```

Some important remarks must be made about the learning rate value and the ratio between the highest and lowest normal modes:

- if $\mu \ll \mu_{max}$, the transient behaviour of the gradient descent algorithm will be overdamped, requiring lots of iterations to reach the optimum solution; instead, if $\mu \approx \mu_{max}$, the transient behaviour will be underdamped, showing lots of oscillations in its trajectory;

- if the ratio between the maximum and minimum natural modes $\lambda_{max}/\lambda_{min}$ is high then, even with an increased number of iterations, the algorithm will not get a convergence rate as good as in the case of a lower ratio between the two modes.

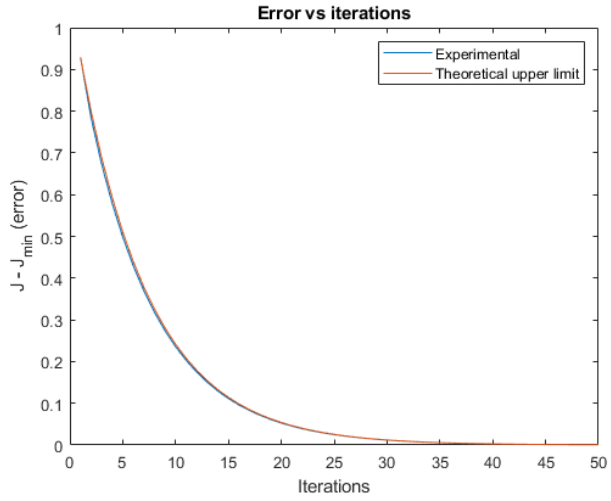


Figure 6: Exponential trend of the error as a function of the number of iterations.

4.3 Visualizing the shaping filters obtained

We decided to show how the gradient descent method to compute the shaping filters performs progressively worst at higher frequencies, particularly when the tolerance chosen for the *get_lpc_w_o_gd* function is coarse.
In Fig. 7 we've plotted the speech signal FFT and the shaping filters obtained with two different values of tolerance for frame number 32, using $L = 1024, M = 512, w\_fun =$ bartlett, $R = L/2$. We notice how the signal obtained with the finer tolerance will approximate closely the original signal up to 20000 Hz, while the one with the coarser tolerance will be a good approximation of the original signal only up to 2500 Hz.

5 Comparison between Wiener-Hopf and gradient descent method

The main difference between the two different methods described in Sec. 3 and Sec. 4 is that the former one provides a closed-form solution to the LPC problem by directly computing the coefficients of the autoregressive model solving a system of linear equations, while the latter one is an iterative optimization algorithm that iteratively updates the coefficients of the autoregressive model. As a result, the Wiener-Hopf equations provides a more accurate result of the LPC problem with respect to the gradient descent method which, on the other hand, should be computationally more efficient especially when dealing with large signals.
We have developed a script, get_lpc_w_o_tester.m, which compares the results obtained with the two different methods with the one obtained with the *lpc* MATLAB built in function in the *get_lpc_w_o_tester* script, with a tolerance of about 1%. The

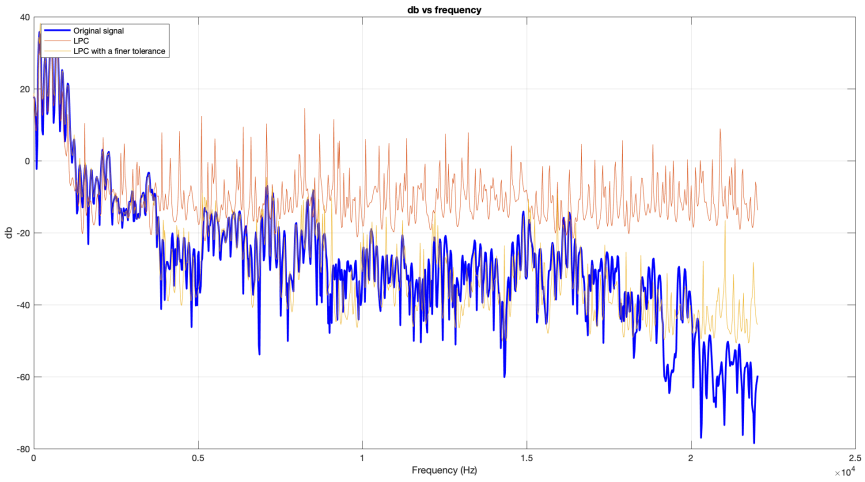


Figure 7: Plot of the speech signal FFT (blue curve) and the associated shaping filters obtained with the gradient descent method: the red curve with a tolerance of 10^{-2} (coarser), and the yellow curve with a tolerance of 10^{-10} (finer).

gradient descent is performed with tolerance $1e-4$ and maximum number of iterations $1e6$, with random initial guess. As input signal, we use a random length vector of random size in the range $[1000-2000]$, and with a filter order M that takes a random value in the range $[10-20]$. With $M = 16$ and input signal of length $N = 1653$, the computation took:

- Wiener-Hopf equation - 0.027997 seconds;
- Gradient descent method - 0.011293 seconds with 440 iterations.

It's clear that the gradient descent method is faster. Furthermore, the test confirms that all the methods yield the same result.
However, with a filter order $M = 191$ (chosen random in the range $100-200$), the gradient descent method seems to be more computationally expensive:

- Wiener-Hopf equation - 0.023579 seconds;
- Gradient descent method - 0.11483 seconds with 13638 iterations.

6 Cross-synthesis implementation

The flowchart of the application is shown in Fig. 8. The steps are:

1. Framing the signals: The piano and speech signals are framed using the window size, hop size, and window function provided as input parameters. The framing process splits the signals into smaller segments or frames.

2. Transforming to the discrete Fourier domain: The Short-Time Fourier Transform (STFT) is applied to the signals to transform them to the frequency domain. The STFT applies a Discrete Fourier Transform to each frame of the signal, and then concatenates the results into a 2D matrix.

3. Whitening the piano: LPC analysis is performed on the piano frames to obtain the shaping filters, which are used to compute the whitening filters (the inverse of the shaping filters). The whitening filters are applied to the STFT of the piano signal to obtain the piano prediction error.

4. Applying shaping filter to the whitened piano: LPC analysis is performed on the speech frames to obtain the shaping filters, which represent the spectral envelope of the speech signal. The piano signal is filtered through the shaping filter of the speech signal by multiplying each spectral frame of the piano signal by the spectral envelope of the speech signal.

5. Go back to time domain and return: The cross-synthesized signal is obtained by applying the inverse STFT to the cross-synthesized STFT signal, and is finally returned.

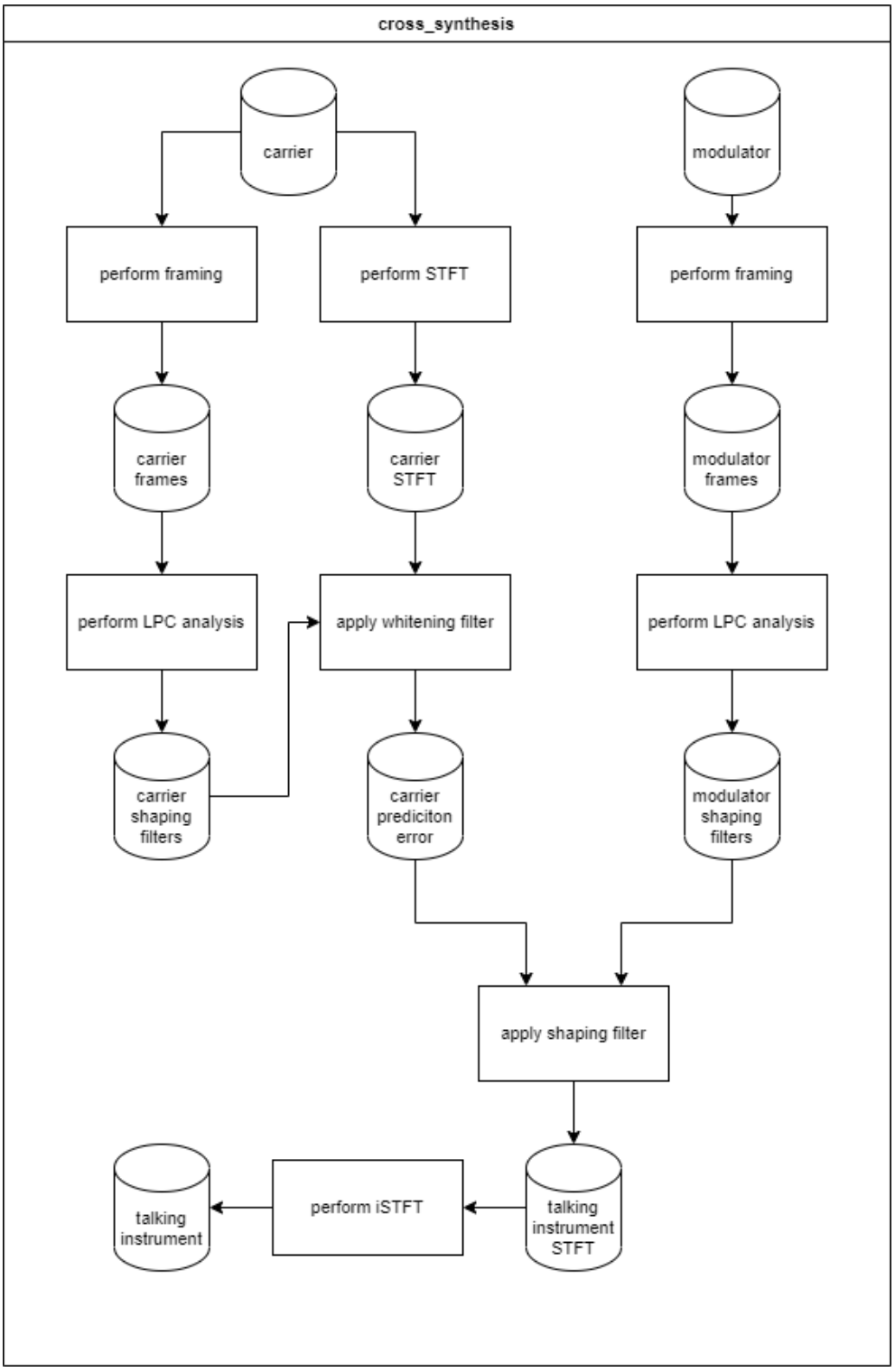


Figure 8: Flowchart of the cross-synthesis application.

6.1 Framing

We have implemented the function **get_signal_frames**(signal, L, R, w_fun, keep_extremes) which returns a $L \times N$ matrix, where L is the **window length**, N is the **number of frames**, signal is the **input signal**, w_fun is a **function handle to a windowing function**, and keep_extremes specifies whether to keep or not the first and last frames. It performs the following operations:

1. Instantiate a windowing function $\underline{w}$ of length L.
2. Zero-pad the signal with R zeros at the beginning and at the end, so that the **Constant OverLap and Add** (COLA) condition is satisfied.
3. Compute the expected number of frames, and initialize the output matrix.
4. Iterate over the number of frames and fill the output matrix. We refer to the input signal as x and to the output signals as y_n, where n is the frame index starting from 1:
$$\underline{y}_n = [x((i-1)R+1), x((i-1)R+2), \ldots, x((i-1)R+L)] \odot \underline{w},$$
where $\odot$ represents the Hadamard product (element-wise).
5. If keep_extremes is false, discard the first and last frames.

This function is tested in windowing_cola_tester.m.

6.2 LPC analysis

We have implemented the function **get_shaping_filters**(framed_signal, M, $NFFT$, gd, error_tolerance, max_num_iter, reuse), which takes as input a matrix framed_signal where each column is a windowed signal, the order M of a linear predictor, the FFT size $NFFT$, and a flag gd indicating whether to use **gradient descent** or not to find the optimal filter coefficients. The function returns a matrix of shaping filters, where the m-th column is the shaping filter for the m-th signal frame, and also returns the number of iterations performed if the chosen method is gradient descent. The implementation consists of the following steps:

1. Determine the number of frames in the input matrix and initialize the output matrix of spectral envelopes.
2. Iterate over the frames and compute the corresponding shaping filter. For each frame:
    a) Extract the m-th column of framed_signal and use it as input to the linear predictor with order M and using either gradient descent or the Wiener-Hopf equation (depending on the value of gd). If we use the gradient descent method, we use the parameters error_tolerance, max_num_iter, and

we have the possibility of using the optimal coefficients of the previous frame as initial guess: this is done when reuse is set to true. This results in the optimal filter coefficients $\underline{w}_o$.

b) Compute the shaping filter as the inverse of the absolute value of the FFT of the sequence $[1, -\underline{w}_o]$, using an FFT size of $NFFT$. This is also known as **spectral magnitude envelope**.

c) Store the shaping filter as the m-th column of the output matrix.

♣ ♣ ♣

The function **get_lpc_w_o**(x, M) computes the optimal coefficients $w_{o,0}, w_{o,1}, \ldots, w_{o,M}$ for a given input signal x and order M of the linear predictor. The steps performed by the function are:

1. Compute the autocorrelation function $\underline{p}$ of $\underline{x}$ considering only the non-negative lags (from zero lag up to lag M) using the built-in Matlab function xcorr.

2. Create a Toeplitz matrix $[R]$ of size $M \times M$ starting from the autocorrelation vector $\underline{p}$ skipping the last lag.

3. Solve the system $[R] \times \underline{w}_o = \underline{p}$ (skipping the zero lag) using the built-in Matlab function linsolve, where $\underline{w}_o$ is the vector of optimal coefficients. Since $[R]$ is a Toeplitz matrix, its inverse is also a Toeplitz matrix and linsolve computes it more efficiently using the Levinson-Durbin algorithm, which has a computational complexity of $\mathcal{O}(M^2)$ (as opposed to the standard $\mathcal{O}(M^3)$).

♣ ♣ ♣

We have implemented the function **get_lpc_w_o_gd**(x, M, error_tolerance, max_num_iter, rand_init, initial_guess) which returns the optimal coefficients $\underline{w}_o = [w_{o_0}, w_{o_1}, ..., w_{o_M}]$ for a signal x using **gradient descent** (GD) optimization. The input parameters are x which is the input signal, M which is the order of **Linear Prediction (LP)** coefficients, error_tolerance is the threshold under which the gradient is considered negligible, max_num_iter is the maximum number of iterations, rand_init is a boolean which specifies if the initial guess should be initialized as random or as the vector specified in initial_guess. The function performs the following operations:

1. Compute the length N of the input signal $\underline{x}$.

2. Calculate the autocorrelation $\underline{p}$ of the input signal using the built-in function **xcorr** keeping only the non-negative lags.

3. Construct the Toeplitz matrix $[R]$ of the autocorrelation, where the first row is constructed from $\underline{p}$ and each subsequent row is constructed by shifting the previous row to the right by one.

4. Calculate the eigenvalues of the submatrix of $[R]$ excluding the first row and column. Set the learning rate μ to 0.95 times the maximum learning rate $\mu_{max} = 2/\lambda_{max}$ (λ_{max} is the maximum eigenvalue).

5. Initialize the LP coefficients $\underline{w}_o$ to random values between −1 and 1 if rand_init is true, else initialize them to the initial_guess vector.

6. Perform the GD algorithm until the number of iterations is less than max_num_iter and the gradient $\|\underline{grad}\| >$ error_tolerance, where $\underline{grad}$ is defined as

$$\underline{grad} = \underline{r}_x(2:end) - \underline{\underline{R}}(2:end, 2:end) \times \underline{w}_o.$$

   Update the LP coefficients according to the formula

$$\underline{w}_o := \underline{w}_o + \mu \cdot \underline{grad},$$

   where $\underline{r}_x(2:end)$ is the subvector of rx excluding the first element, and $\underline{\underline{R}}(2:end, 2:end)$ is the submatrix of R excluding the first row and column.

6.3 Multi-resolution analysis

After we compute the whitened piano STFT and the speech shaping filters matrix, we need to make sure that the number of frames of the piano and the number of speech shaping filters is the same, and that the window length values are the same. This is because we want to apply the shaping filter to each whitened piano frame. That's why, if the two signals have different resolution, the piano prediction error STFT is anti-transformed to time-domain and, then, STFT is applied again using speech resolution. A different LPC order for each signal is supported too, and it doesn't require special attention.

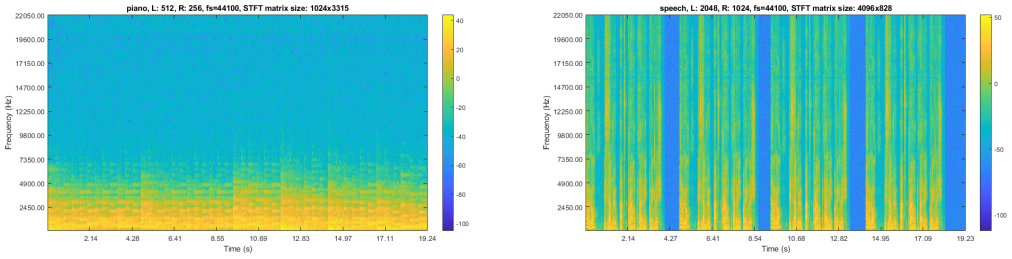
7 Results

Using the implementation described in Sec. 6, we have performed cross-synthesis on the piano and speech signals, the first used as carrier and the second as modulator, trying to match the expected result expected_result.wav. The chosen parameters are:

```
L_piano = 512;          % window length piano
M_piano = 128;            % lpc order piano

L_speech = 2048;          % window length speech
M_speech = 512;            % lpc order speech

w_fun = @bartlett;          % window type
R_piano = L_piano/2;          % hop size piano
```

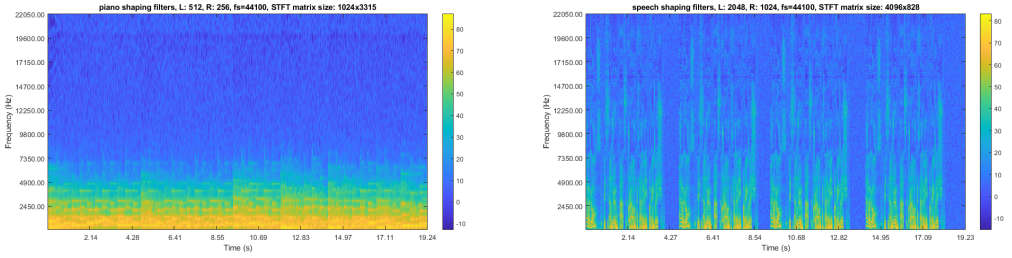
```
R_speech = L_speech/2;                % hop size speech

use_gradient_descent = false;
error_tolerance = 0; % only has effect for gradient descent
max_num_iter = 0; % only has effect for gradient descent
reuse = false; % only has effect for gradient descent
```

where we note that the last three parameters are unused, having chosen the Wiener-Hopf method.
In the next figures, we see some spectrograms relevant in the computation of the output. We appreciate the fact that the chosen parameters let us analyze the formants of voice in frequency while retaining a good time resolution. Furthermore, we notice that the shaping filters (Fig. 10) closely approximate the STFT of the input signals, having chosen a high LPC order for both signals. We notice how the spectrogram of the output (Fig. 12) corresponds to the blended spectrogram of two input signals (Fig. 9).
We qualitatively assess that the output of our application with the previously specified parameters closely resembles the ground truth.



(a) Piano. (b) Speech.

Figure 9: STFT of the input signals.



(a) Piano. (b) Speech.

Figure 10: Shaping filters.

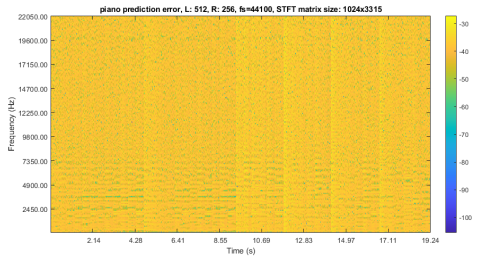


Figure 11: Piano signal prediction error.

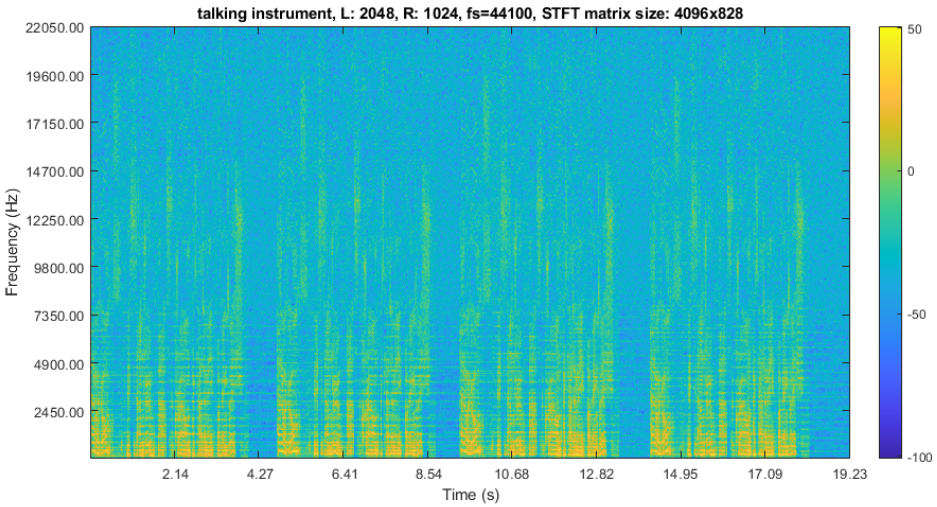


Figure 12: Cross-synthesized signal.

As a final remark, we note that the chosen LPC order values are really high. As a rule of thumb, a good LPC order for speech lies in the range

$$\frac{F_s}{1000} \leq M \leq \frac{F_s}{1000} + 4, \tag{2}$$

which yields, for $F_s = 44100Hz$, $44 \leq M \leq 48$. $M = 512$ exceeds it by a lot; however, as we want high reconstruction fidelity, we had no choices but choosing such a high order.
If we had implemented a frequency selective LPC, we would have been able to choose more appropriate values: a high LPC order for the voiced speech range $0-4kHz$ (where we need a more precise approximation), a lower LPC order for the range of fricatives $4-8kHz$ (where we want a smoother fit), and an even lower LPC order for the remaining range (where voice is less significant). This could be part of a future development of our application.