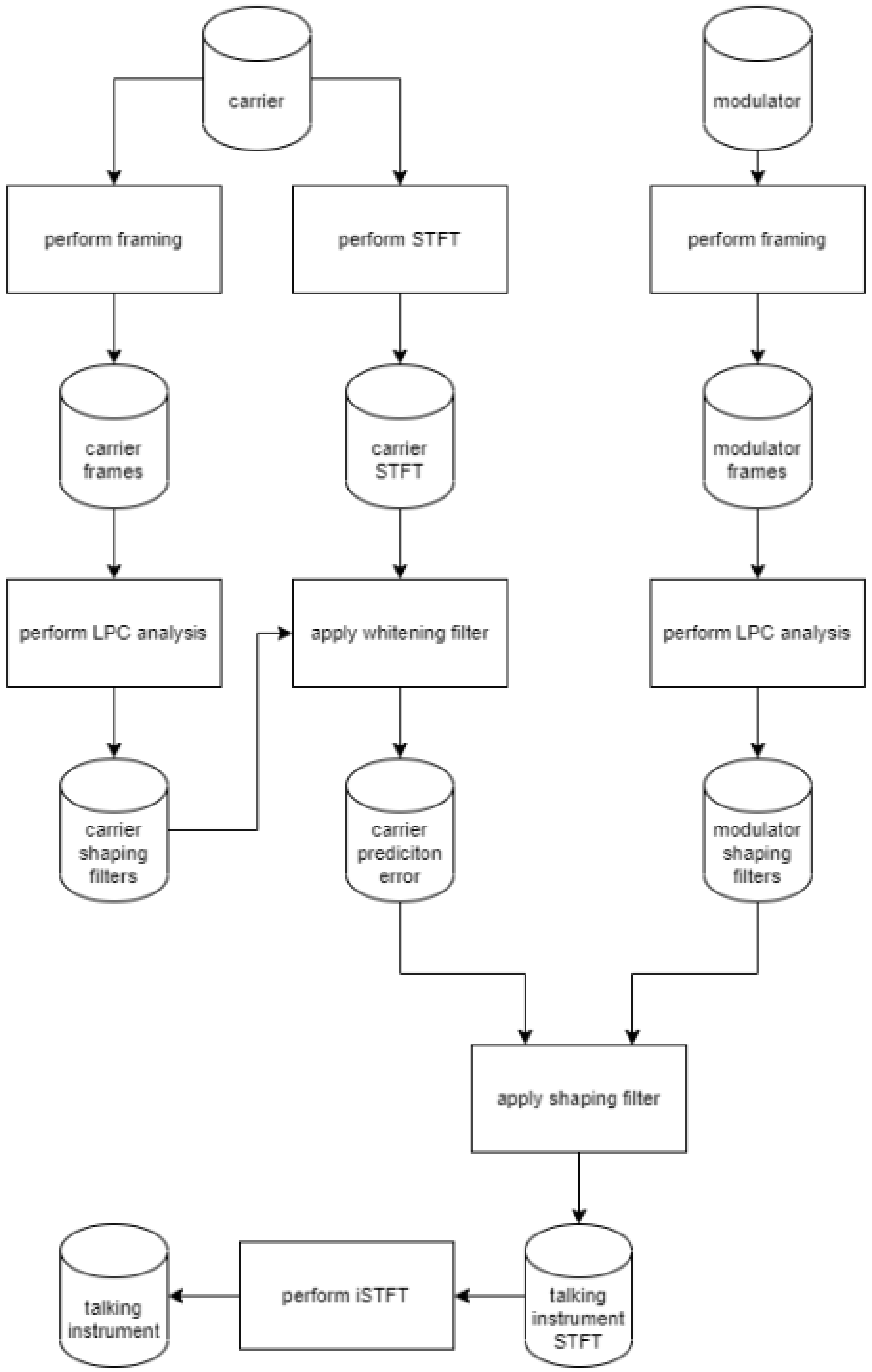
POLITECNICO
MILANO 1863

DAAP Homework 1: talking instrument

Umberto Derme 10662564
Gerardo Cicalese 10776504

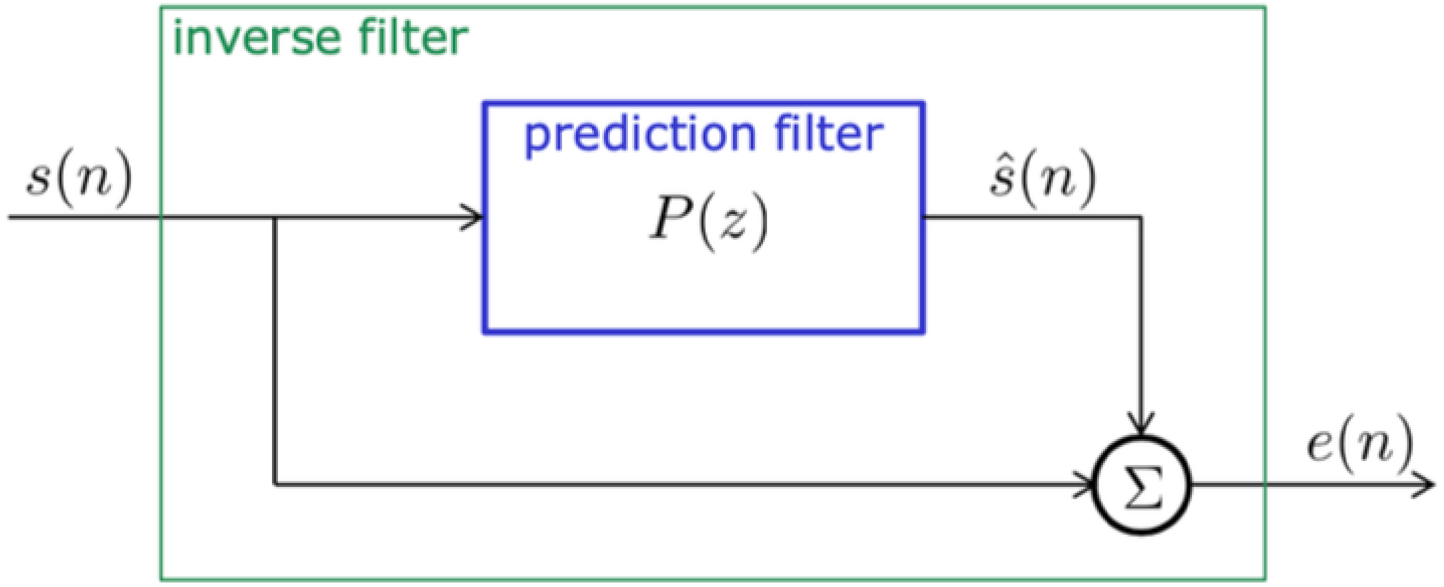
Cross-synthesis



- The goal of cross-synthesis is to imprint the spectral envelope of the modulator (voice) on the carrier (piano).
- The spectral envelopes are approximated through Linear Predictive Coding (LPC).
- LPC needs the assumption of stationariety, thus we exploit short time analysis (OLA).
- Filtering is performed in frequency domain.

Linear Prediction Coding (LPC)

We assume that the signal can be modelled as an AR stochastic process: $s(n) = \sum_{k=1}^{p} a_k s(n-k) + Gu(n),$



We can identify three different filters:

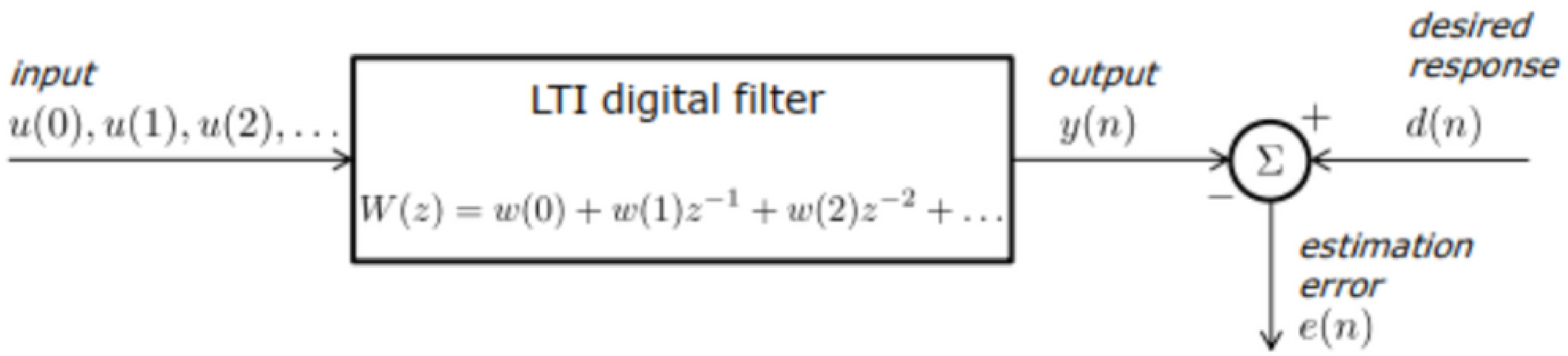
- Whitening filter and shaping filter

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

- Prediction filter

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

The purpose is to find the optimal coefficients of the Prediction filter so that the MSE of the prediction error is minimized: this can be set up as a Wiener filtering problem.



Wiener-Hopf Equations

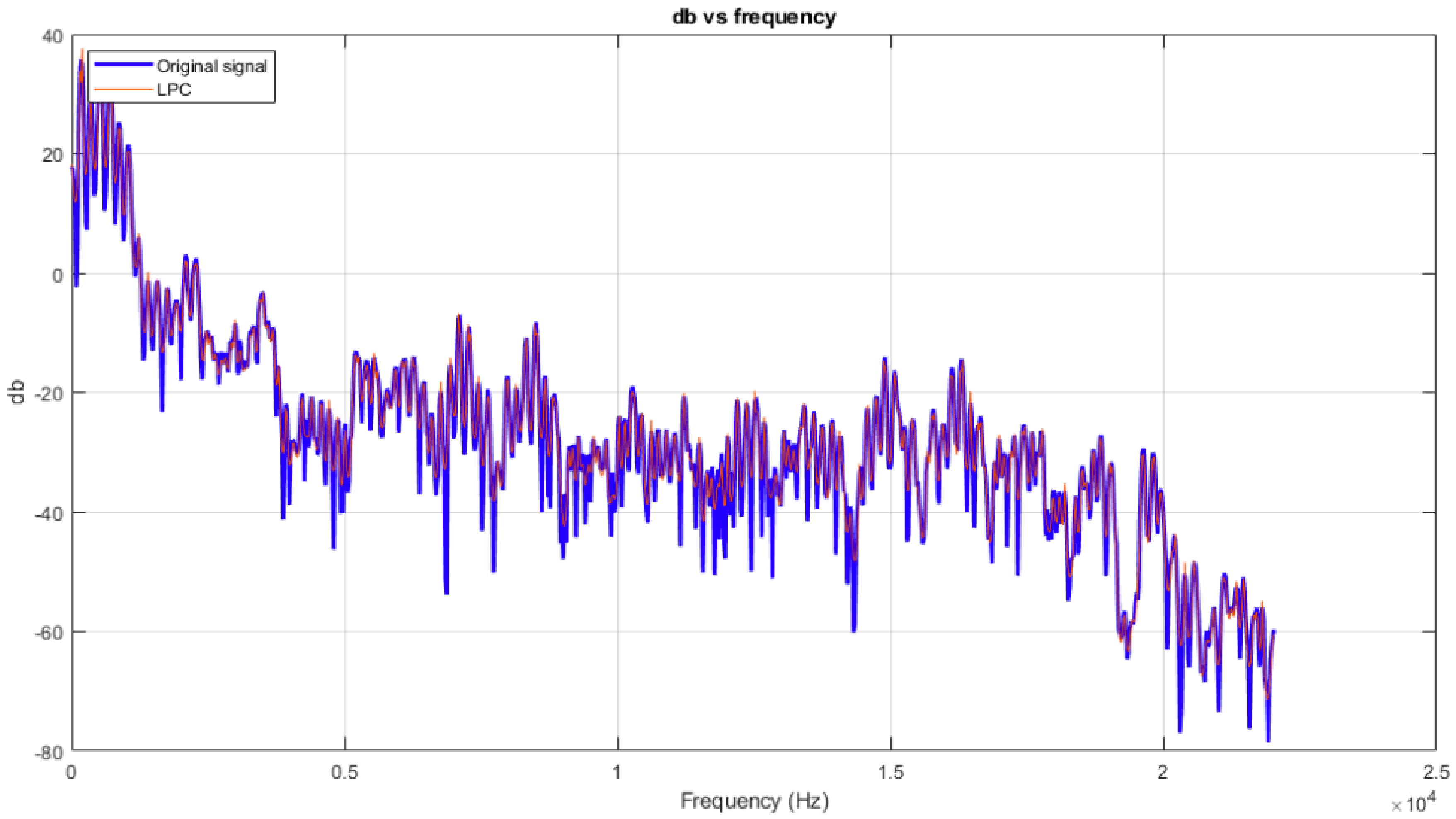
The **Wiener-Hopf equations** provide a closed-form solution to the LPC problem by directly computing the coefficients of the autoregressive model solving a system of linear equations resulting in a more accurate result of the LPC problem.

The optimal coefficients are computed as:

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

Computational complexity:

- O(p^3) through matrix inversion
- O(p^2) with the Levinson-Durbin algorithm

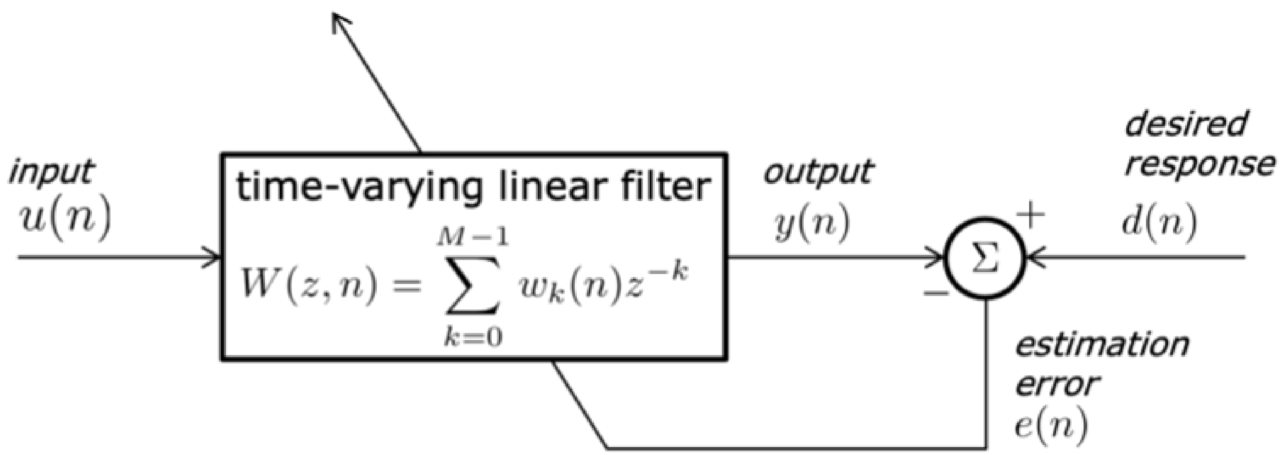


shaping_filter_visualizer.m

Gradient Descent

The **Gradient Descent** is a gradient-based adaptation technique defined by an iterative algorithm: the tap-weight vector is updated following the inverse direction given by the gradient at each iteration.

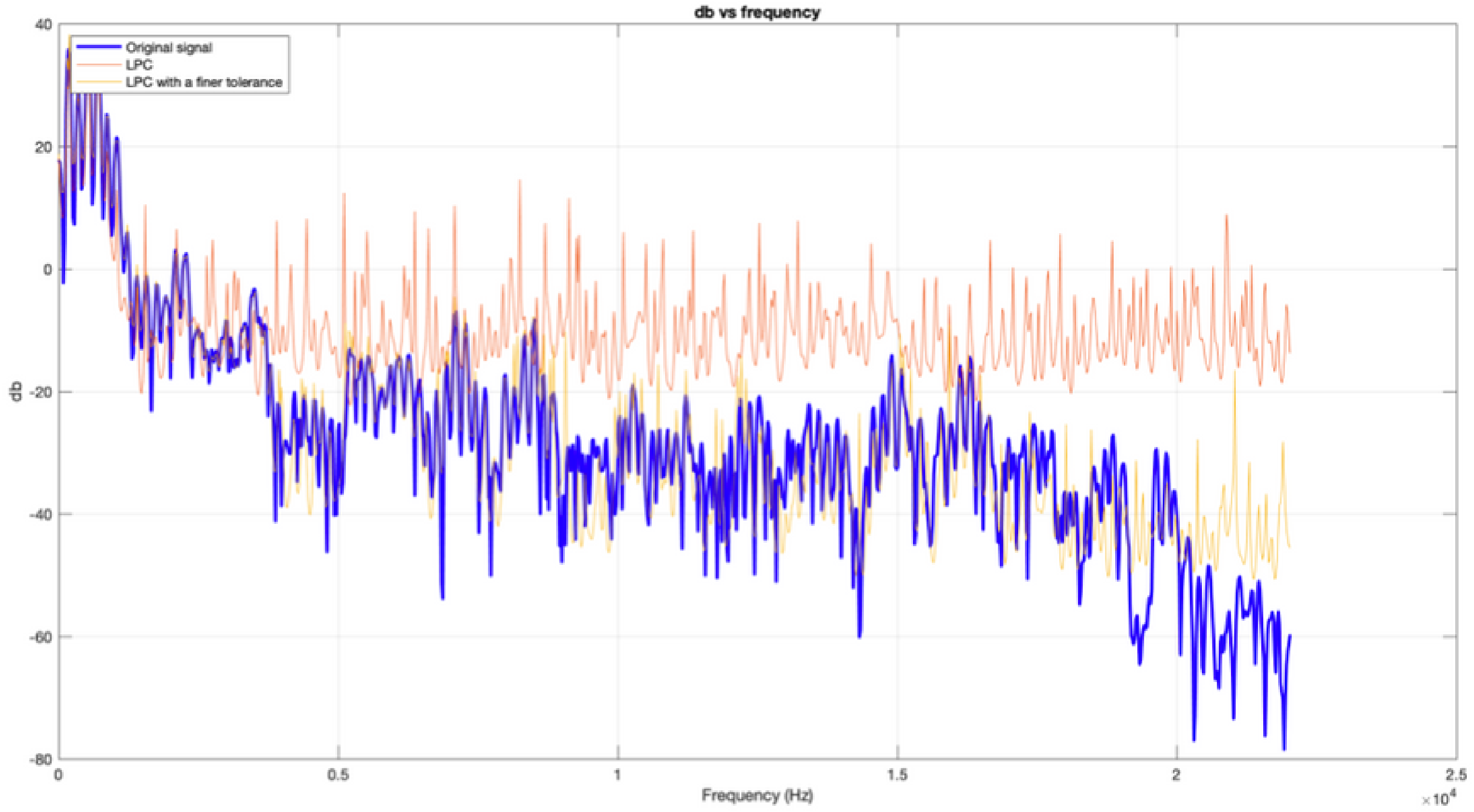
$$\underline{w}(n+1)=\underline{w}(n)+\frac{1}{2}\mu\left[-\nabla J\left(\underline{w}(n)\right)\right]=\underline{w}(n)+\mu\left[\underline{p}-[R]\underline{w}(n)\right]$$

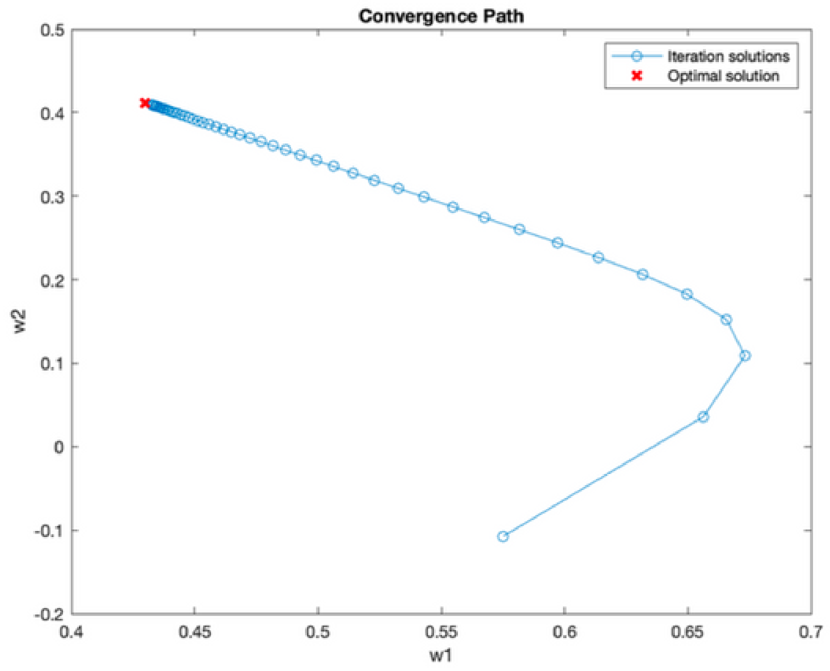


The necessary and sufficient condition for the stability of the algorithm is:

$$0<\mu<\frac{2}{\lambda_{max}}$$

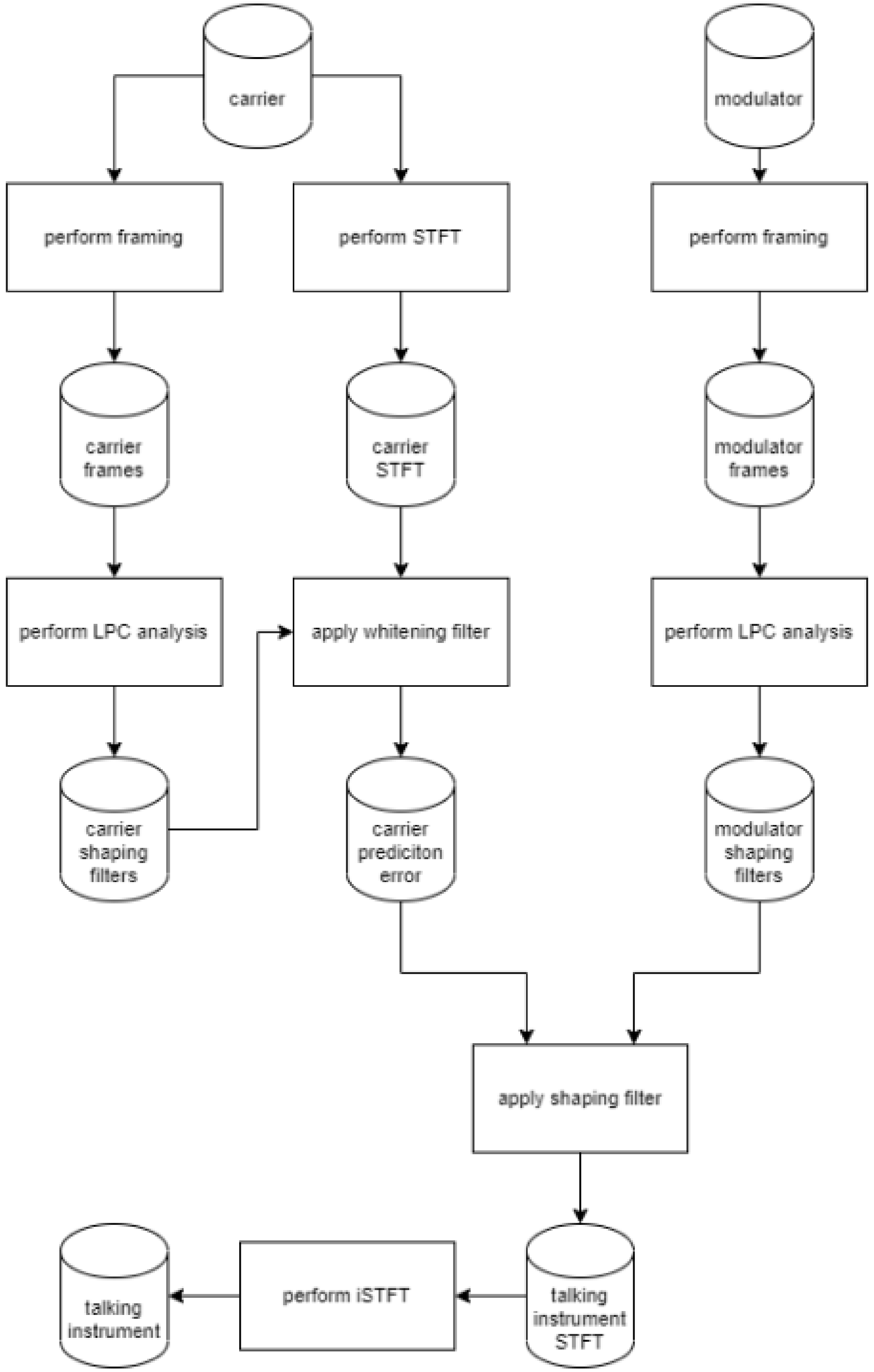
We see that the solution improves as the number of iterations increases:





gradient_descent_visualizer.m

Implementation: flowchart



1. The piano and speech signals are framed using tapered windows.
2. The Short-Time Fourier Transform (STFT) is applied to the carrier.
3. LPC analysis is performed on the piano frames to obtain the piano whitening filters.
4. The whitening filters are applied to the STFT of the piano signal to obtain the piano prediction error.
5. LPC analysis is performed on the speech frames to obtain the speech shaping filters.
6. The piano signal is filtered in frequency domain through the shaping filter of the speech signal, obtaining the cross-synthesized STFT.
7. The cross-synthesized signal is obtained by computing the inverse STFT of the cross-synthesized STFT signal.

Implementation: OverLap-and-Add (OLA)

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 $\underline{y}_n$, where n is the frame index starting from 1:
   $$\underline{y}_n = [x((i-1)R+1), x((i-1)R+2), \dots, x((i-1)R+L)] \odot \underline{w},$$
   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.

Implementation: 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.

Implementation: Wiener-Hopf equations

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 $\mathcal{O}(M^3)$).

Implementation: Gradient Descent

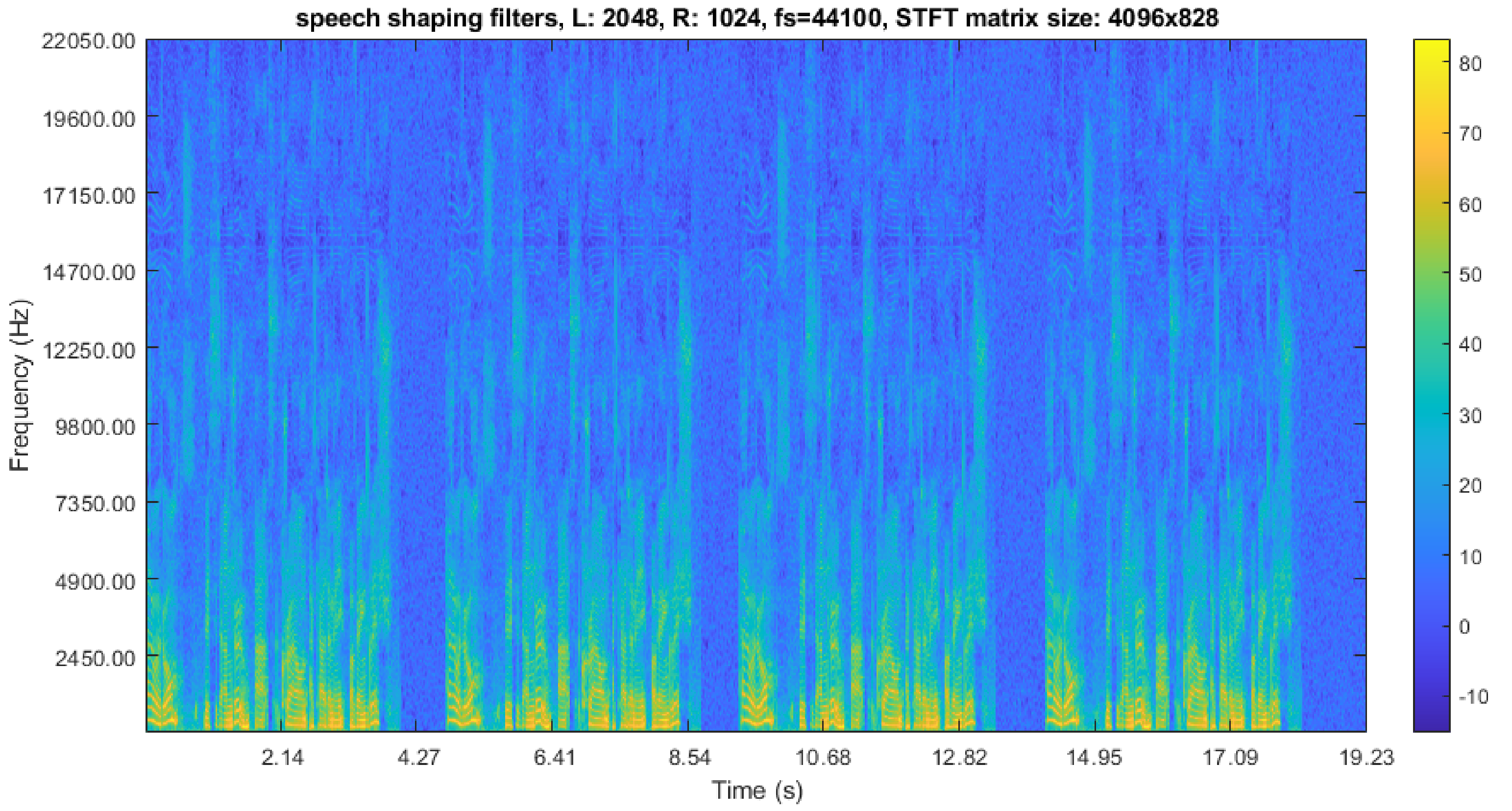
We have implemented the function **get_lpc_w_o_gd**(x, M, error_tolerance, max_num_iter, rand_init, initial_guess) which returns the optimal coefficients $\underline{w}_o = [w_{o_0}, w_{o_1}, ..., w_{o_M}]$ for a signal x using **gradient descent** (GD) optimization. The input parameters are x which 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:

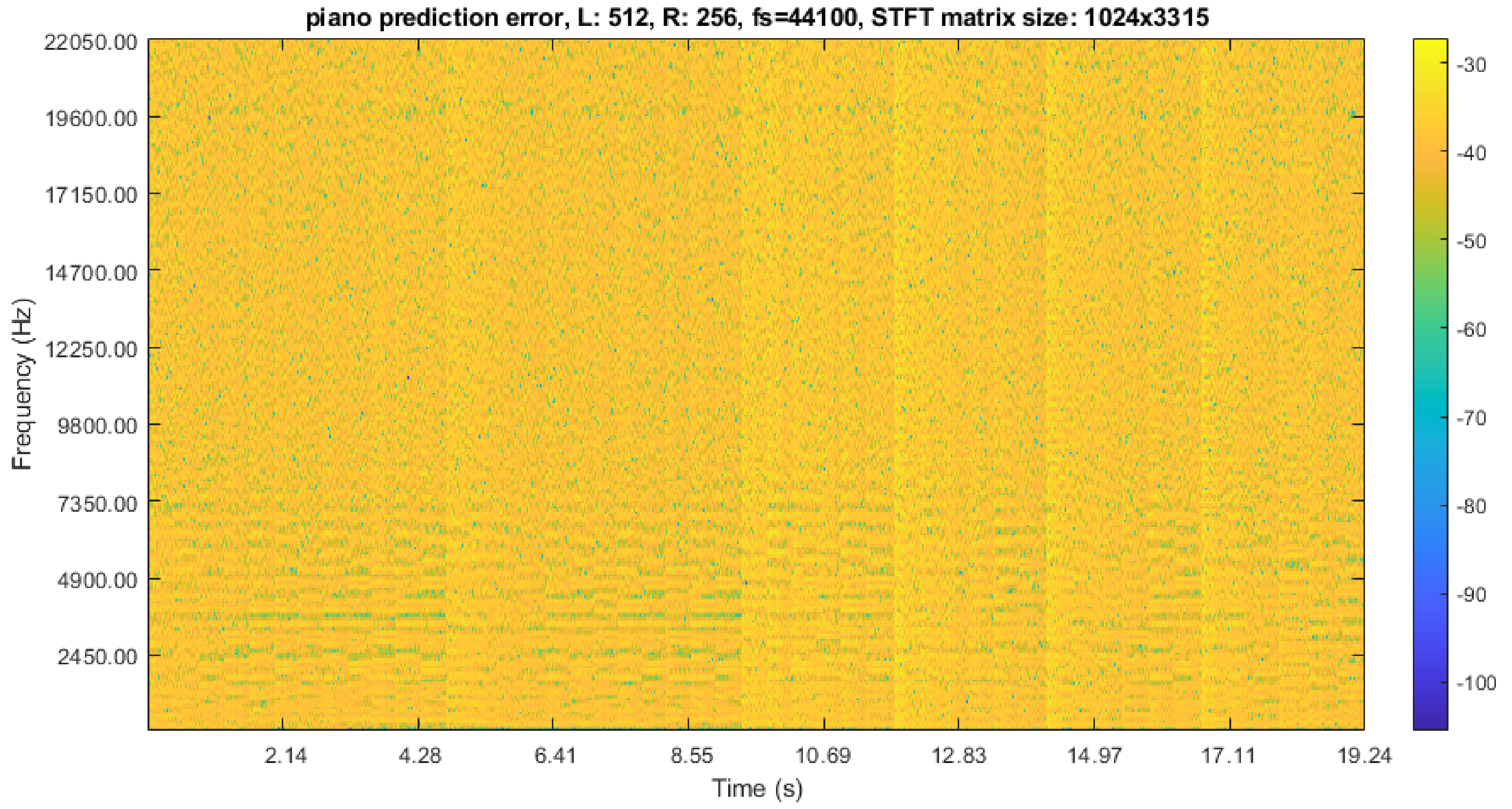
1. Compute the length N of the input signal $\underline{x}$.
2. Calculate the autocorrelation $\underline{p}$ of the input signal using the built-in function **xcorr** keeping only the non-negative lags.
3. Construct the Toeplitz matrix $[R]$ of the autocorrelation, where the first row is constructed 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
$$\underline{grad} = \underline{r}_x(2:end) - \underline{R}(2:end, 2:end) \times \underline{w}_o.$$
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 $\underline{R}(2:end, 2:end)$ is the submatrix of R excluding the first row and column.

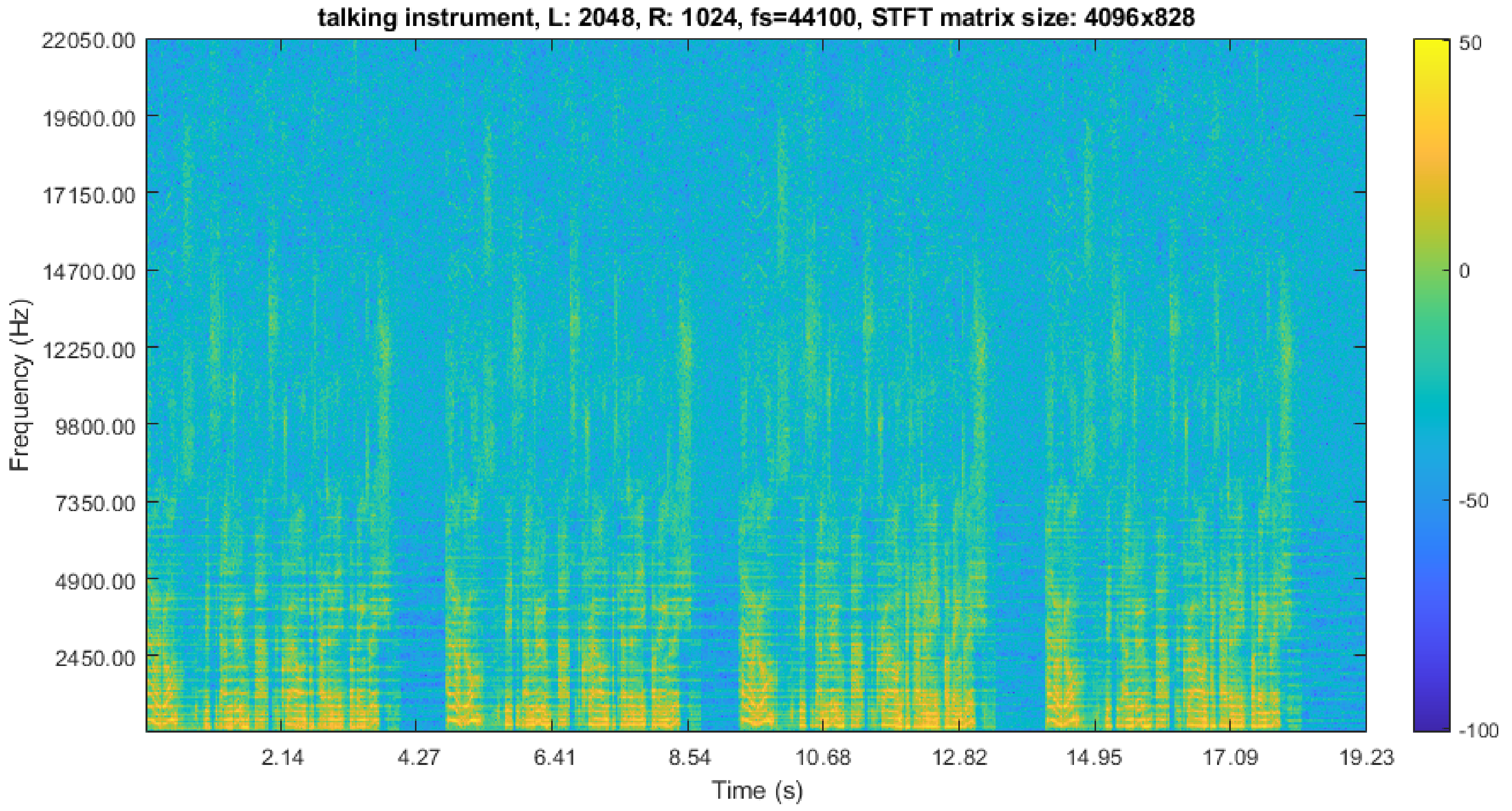
Results and conclusions

Chosen parameters:

```
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;
```







POLITECNICO
MILANO 1863

Thanks for the attention!

A more detailed description of the theory and implementation of the cross-synthesis can be found in the report.