Institut Mines-Télécom

Spectral and temporal modifications

Roland Badeau

TELECOM Paris

IP PARIS

In this practical work, you will implement the PSOLA method for the analysis/synthesis of speech signals. This method will be tested on signals that can be downloaded on the TSIA 206 Moodle. These signals are sampled at `Fs`. You can load them with Matlab e.g. by typing up `load aeiou`; they will then be stocked in variable `s`. To listen to them, you can type up `soundsc(s,Fs)`. In Python, you can use the provided notebook template `template-TP-modifications.ipynb` that you can download on the website.

1 Extraction of the analysis marks

Firstly, you will program the following function

```
function A = AnalysisPitchMarks(s,Fs)
```

which extracts the analysis marks. The arguments `s` and `Fs` respectively are the signal to be analyzed and the sampling frequency. The returned matrix `A` will contain the times and pitches corresponding to each analysis mark. More precisely, `A` will be formed of three rows, such that `A(1,n)` $= t_a(n)$ is the time corresponding to the n^{th} analysis mark ($t_a(n) \in \mathbb{N}$ is expressed in number of samples), `A(2,n)` = voiced(n) is a Boolean which indicates whether the signal is voiced or unvoiced in the neighborhood of this mark, and `A(3,n)` $= P_a(n) \in \mathbb{N}$ describes the pitch corresponding to the same mark (i.e. the period expressed in number of samples) in the voiced case, or equals 10ms $\times$ `Fs` in the unvoiced case.

To do so, you will need a pitch estimator. In order to spare time, you can use function `period.m`, whose Matlab code is provided with the example signals, and whose Python code is included in the notebook template `template-TP-HR.ipynb`. This function requires two arguments: a short term signal `x` extracted from `s`, and the sampling frequency `Fs` (the other arguments are optional), and returns a couple `[P, voiced]` where `voiced` is a Boolean which indicates whether `x` is voiced or non, and `P` $\in \mathbb{N}$ is the period expressed in number of samples in the voiced case, or equals 10ms $\times$ `Fs` in the unvoiced case.

Let us now detail how to determine the analysis marks. For the sake of simplicity, we will not try to align the mark $t_a(n)$ on the beginning of a glottal pulse. To compute $P_a(n)$ and $t_a(n)$, we proceed by recursion on $n \geq 1$:

- extraction of a sequence `x` that starts at time $t_a(n-1)$, and whose duration is equal to 2.5 $P_a(n-1)$;
- computation of $P_a(n)$ and voiced(n) by means of function `period`;
- computation of $t_a(n) = t_a(n-1) + P_a(n)$.

The algorithm will be initialized by setting $t_a(0) = 1$ (in Matlab) or $t_a(0) = 0$ (in Python) and $P_a(0) =$ 10ms $\times$ `Fs`.

2 Synthesis and modification of the temporal and spectral scales

To perform the synthesis of the signal, we must start by defining the synthesis marks. They will be stocked in a matrix `B` formed of two rows, such that `B(1,k)` $= t_s(k)$ is the time corresponding to the k^{th} synthesis mark, and `B(2,k)` $= n(k)$ is the index of the analysis mark corresponding to this same synthesis mark. To start, you can perform a synthesis without modification, by setting:

- `B(1,:)` = `A(1,:)`;
- `B(2,:)` = $[1, 2, 3, \ldots]$.

2.1 Signal synthesis

You will now program the following function

```
function y = Synthesis(s,Fs,A,B)
```

which computes the synthesis signal `y` from the original signal `s`, the sampling frequency `Fs`, the analysis marks stocked in matrix `A` and the synthesis marks stocked in matrix `B`. The synthesis is very simply performed by recursion on $k \geq 1$ (vector `y` being initialized to the zero vector of dimension $t_s(k_{\text{end}}) + P_a(n(k_{\text{end}}))$):

- extraction of a sequence `x` centered at $t_a(n(k))$ and of length $2\,P_a(n(k)) + 1$;
- windowing of `x` by a Hann window (Matlab function `hann` or Python function `scipy.signal.hanning`);
- overlap-add of the sequence `x` windowed on $y(t_s(k) - P_a(n(k)) : t_s(k) + P_a(n(k)))$.

2.2 Modification of the temporal scale

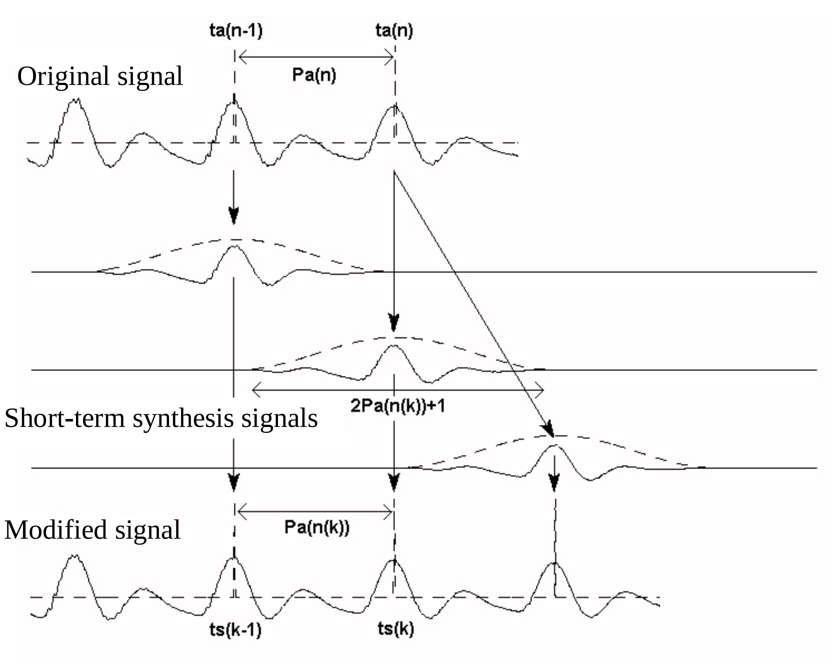

Figure 1: Modification of the temporal scale

We now want to determine the synthesis marks that will modify the temporal scale by a factor α, i.e. to determine a matrix `B` such that the duration of the signal synthesized by function `Synthesis` is equal to that of the original signal `s` multiplied by α. This operation will be performed by function

```
function B = ChangeTimeScale(alpha,A,Fs)
```

which computes matrix `B` from the factor α, the analysis marks stocked in `A`, and the sampling frequency `Fs`. You can proceed by recursion on $k \geq 1$, by using a non-integer index $n(k)$:

- $t_s(k) = t_s(k-1) + P_a(\lfloor n(k) \rfloor)$;
- $n(k+1) = n(k) + \frac{1}{\alpha}$.

The algorithm will be initialized by setting $t_s(0) = 1$ and $n(1) = 1$. You will take care of only stocking integer values in matrix `B`.

2.3 Modification of the spectral scale

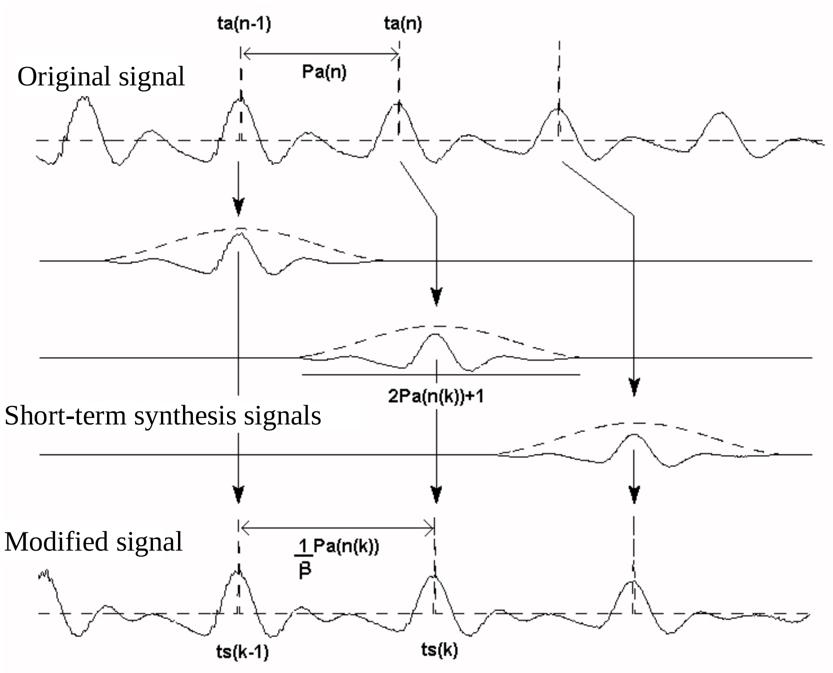

Figure 2: Modification of the spectral scale

You will now perform the dual operation of the previous one: determine the synthesis marks that will modify the spectral scale by a factor β, i.e. determine a matrix `B` such that the fundamental frequency of the signal synthesized by function `Synthesis` is equal to that of the original signal `s` multiplied by β. This operation will be performed by function

```
function B = ChangePitchScale(beta,A,Fs)
```

which computes matrix `B` from the factor β, the analysis marks stocked in `A`, and the sampling frequency `Fs`. As in the previous case, you can proceed by recursion on $k \geq 1$, by using a non-integer index $n(k)$ and non-integer synthesis times $t_s(k)$, and by making the difference between the voiced and unvoiced cases:

- if the analysis mark of index $\lfloor n(k) \rfloor$ is voiced, $\text{scale}(k) = \frac{1}{\beta}$, otherwise $\text{scale}(k) = 1$;
- $t_s(k) = t_s(k-1) + \text{scale}(k) \times P_a(\lfloor n(k) \rfloor)$;
- $n(k+1) = n(k) + \text{scale}(k)$.

Again, you will take care of only stocking integer values in matrix `B`.

2.4 Joint modification of the temporal and spectral scales

To finish, you will program a function that jointly modifies the two scales:

```
function B = ChangeBothScales(alpha,beta,A,Fs)
```

where the arguments are defined as previously. The content of this function will be almost identical to that of `ChangePitchScale`; you will just need to modify it properly.