Spectral Methods

Sébastien Boisgérault, Mines ParisTech

1 June, 2015

Contents

In the context of audio signal processing, spectral methods refer to algorithms that rely on the representation of signals as superposition of sinusoids. Such a decomposition – the spectrum of the signal – is obtained with the Fourier

transform ; efficient computations of the spectrum are possible with the fast Fourier transform algorithms.

Spectral methods are crucial in the study of filters such as the finite impulse response filters or autoregressive filters and more generally to understand any transformation based on a convolution ; they are also key in multirate systems that achieve compression through data rate reduction.

Signal, Spectrum, Filters

A **discrete-time signal** x with sample time Δt is a function defined on

$$\mathbb{Z}\Delta t = \{k\Delta t,\ k \in \mathbb{Z}\}.$$

This definition is nothing but a convenient packaging of the sequence of values (x_n), $n \in \mathbb{Z}$, with the sample time Δt into a unique mathematical object.

We investigate in this section the representation of a discrete-time signal as a superposition of sinusoids. Let's start the search for such a spectral representation with a real-valued discrete-time signal x. Given a non-negative frequency f, a sinusoid is determined uniquely by its amplitude $a(f) \geq 0$ and – provided that $a(f) \neq 0$ – its phase $\phi(f) \in [-\pi, \pi)$. We therefore search for a pair of functions a and ϕ – subject to the above constraints – such that

$$\forall\, t \in \mathbb{Z}\Delta t,\ x(t) = \int_0^{+\infty} a(f)\cos(2\pi f t + \phi(f))\, df$$

Alternatively, we may use complex exponentials instead of sinusoids: decompose the cos in the previous equation and set

$$x(f) = \left| \begin{array}{ll} 1/2 \times a(f)e^{i\phi(f)} & \text{if } f \geq 0 \\ 1/2 \times a(-f)e^{-i\phi(-f)} & \text{otherwise.} \end{array} \right. , \text{ or equivalently } \begin{array}{l} a(f) = 2|x(f)| \\ \phi(f) = \angle x(f) \end{array}$$

The equation () becomes

$$x(t) = \int_{-\infty}^{+\infty} x(f)\exp(i2\pi f t)\, df$$

and the only constraint that holds on the complex-valued function $x(f)$, defined for any real frequency f, is the symmetry constraint

$$x(-f) = \overline{x(f)}$$

This relation specifically ensures that the complex exponentials in () always combine to produce a real-valued signal $x(t)$, so that the values of $x(f)$ for negative frequency hold not extra information and are merely an artifact of the complex exponential representation.

However we can drop this symmetry constraint if we allow complex-valued signals $x(t)$ in the first place and then these negative frequencies values are no longer redundant. At the same time, we notice that () still makes sense if we consider vector-valued signals $x(t) \in \mathbb{C}^p$, so given this higher generality, and also the better mathematical tractability of (), we will stick to this formulation of the problem.

At this stage we clearly search for summable functions – that is $x(f) \in L^1(\mathbb{R}, \mathbb{C}^p)$ – so that the right-hand side of the equation makes sense. Still, the problem of finding a solution $x(f)$ to () is not well posed: let Δf be the signal sampling frequency, defined by

$$\Delta f \times \Delta t = 1.$$

If $x(f)$ is a solution to the equation, so is $f \mapsto x(f - k\Delta f)$ for any $k \in \mathbb{Z}$[1]: the spectral content of $x(t)$ is only determined up to frequency shifts that are multiples of the sampling frequency Δf.

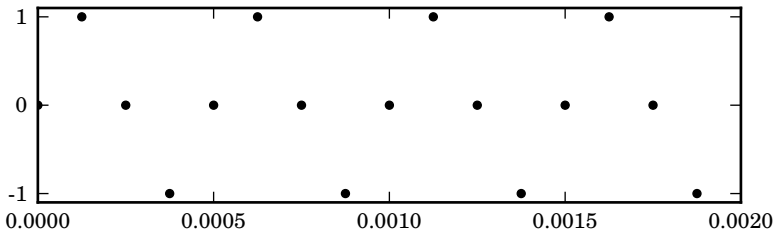

Figure 1:

A way to remove this ambiguity in $x(f)$ is to reassign to any spectral component at the frequency f the smallest frequency $f - k\Delta t$ that is the nearest from 0 among any possible values of $k \in \mathbb{Z}$ – that frequency has to be in $[-\Delta f/2, +\Delta f/2]$. In other words, for any spectral component
of the signal, we make a low-frequency interpretation. Mathematically, that

[1]Indeed, we have

$$\begin{aligned} \int_{-\infty}^{+\infty} x(f + k\Delta f) \exp(i2\pi ft)\, df &= \int_{-\infty}^{+\infty} x(f) \exp(i2\pi(f - k\Delta f)t)\, df \\ &= \int_{-\infty}^{+\infty} x(f) \exp(i2\pi ft) \exp(-i2\pi kt\Delta f)\, df \end{aligned}$$

As $t = n\Delta t$ for a $n \in \mathbb{Z}$, $\exp(-i2\pi kt\Delta f) = \exp(-i2\pi kn) = 1$ and consequently

$$\int_{-\infty}^{+\infty} x(f + k\Delta f) \exp(i2\pi ft)\, df = \int_{-\infty}^{+\infty} x(f) \exp(i2\pi ft)\, df$$

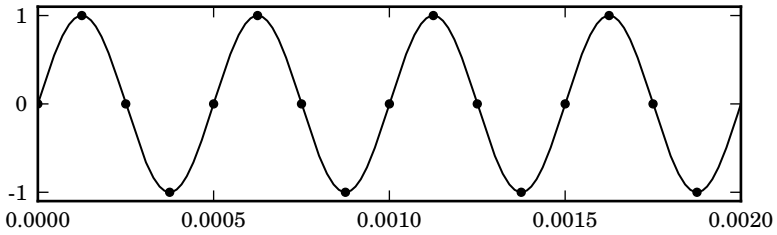

Figure 2:

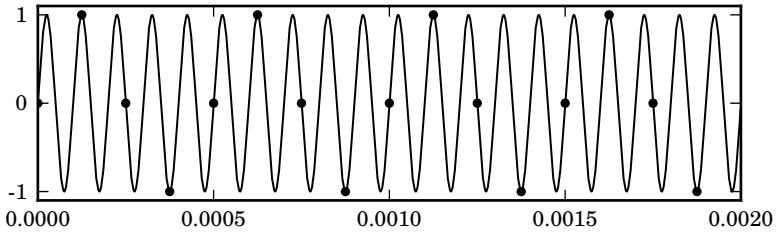

Figure 3: A signal sampled at 8khz (top) and two valid interpretation of its spectral content : either a pure 2 kHz sine (mid) or 10 kHz sine (bottom).

means that we replace $x(f)$ with[2]

$$x(f) \to x'(f) = \left| \begin{array}{ll} \displaystyle\sum_{k\in\mathbb{Z}} x(f - k/\Delta t) & \text{if } f \in [-\Delta f/2, +\Delta f/2], \\ 0 & \text{otherwise.} \end{array} \right.$$

and therefore if we rename $x(f)$ this particular solution $x'(f)$, we end up with the search for an integrable function $x(f) : [-\Delta f/2, +\Delta f/2] \to \mathbb{C}^p$ solution of the equation

$$\forall\, t \in \mathbb{Z}\Delta t,\ x(t) = \int_{-\Delta f/2}^{+\Delta f/2} x(f) \exp(i2\pi f t)\, df$$

The highest frequency value in the integration interval, $\Delta f/2$ is called the **Nyquist frequency** associated to the sample time Δt.

We notice at this stage that equation () defines $x(n\Delta t)$ as the n-th Fourier coefficient of the Fourier serie associated to $x(f)$. As a consequence, if we make

[2]this is legitimate because for any $t \in \mathbb{Z}\Delta t$

$$\int_{-\infty}^{+\infty} x(f) \exp(i2\pi f t)\, df = \int_{-\Delta f/2}^{+\Delta f/2} \left[\sum_{k\in\mathbb{Z}} x(f - k/\Delta t) \right] \exp(i2\pi f t)\, df$$

the assumption that the signal $x(t)$ is of finite energy, that is mathematically $x(t) \in L^2(\mathbb{Z}\Delta t, \mathbb{C}^p)$ or

$$\sum_{n \in \mathbb{Z}\Delta t} |x(t)|^2 < +\infty$$

then the function $x(f)$ is uniquely defined (almost everywhere). It also belongs to $L^2([-\Delta f/2, +\Delta f/2], \mathbb{C}^n)$

$$\int_{-\Delta f/2}^{\Delta f/2} |x(f)|^2 \, df < +\infty$$

and satisfies[3]

$$x(f) = \Delta t \sum_{n \in \mathbb{Z}\Delta t} x(t) \exp(-2i\pi t f)$$

The transform – denoted $\mathcal{F}$ – that maps a signal time-domain representation $x(t)$ to its frequency-domain representation or **spectrum** $x(f)$ is **(discrete-time) Fourier transform (DTFT)**.

Parseval's theorem also yields

$$\int_{-\Delta f/2}^{+\Delta f/2} |x(f)|^2 \, df = \Delta t \sum_{t \in \mathbb{Z}\Delta t} |x(t)|^2$$

which means that we may measure the energy of the signal by summing either the energy of each sample in the time domain, or the energy density of all signal spectral components.

The category of finite energy signals is sufficient most of the time but still does not encompass every signal we'd like to consider … and to begin with, pure tones! To perform the spectral decomposition of signals that have an infinite energy, we need to go beyond Δf-periodic (locally) integrable functions of the frequency f and consider instead Δf-periodic (vector-valued complex) measures. For such a measure $x(f)$, $x(t)$ is represented by the integral

$$\forall\, t \in \mathbb{Z}\Delta t,\ x(t) = \int_{(-\Delta f/2)^-}^{(+\Delta f/2)^-} \exp(i2\pi f t) \, dx(f)$$

In practice, we don't need measures with singular parts which means that every measure spectra we need to consider has on the interval $[-\Delta f/2, +\Delta f/2)$ the form

$$x(f) = x_1(f) + \sum_i a_i \delta(f - f_i) \quad \text{where} \quad \left| \begin{array}{l} x_1(f) \in L^1([-\Delta f/2, \Delta f/2), \mathbb{C}^n) \\ \sum_i |a_i| < +\infty \end{array} \right.$$

[3] as a limit in $L^2([-\Delta f/2, +\Delta f/2], \mathbb{C}^n)$ or pointwise but only almost anywhere (Carleson). Stronger convergence (such as uniform convergence) may be obtained under the assumption that $x(f)$ is continuously differentiable.

And then, every dirac component in the frequency domain represents a pure tone as

$$\int_{(-\Delta f/2)^-}^{(+\Delta f/2)^-} \exp(i2\pi ft)\, d\delta(f - f_i) = \exp(i2\pi f_i t)$$

Therefore the equation (**??**) reduces to

$$x(t) = x_1(t) + \sum_i a_i \exp(i2\pi f_i t)$$

Convolution and Filters

Convolution

Consider two scalar discrete-time signals x and y with a common sample time Δt. We assume for convenience that there is a $t_0 \in \mathbb{Z}\Delta t$ such that $x(t) = y(t) = 0$ for any $t \leq t_0$. We define the **convolution** between x and y as the discrete-time signal $x * y$ with sample time Δt such that

$$(x * y)(t) = \Delta t \sum_{t' \in \mathbb{Z}\Delta t} x(t')y(t - t')$$

The assumptions made on the signals x and y ensure that for every value of t, the sum in the right-hand side of () has only a finite number of non-zero values. These assumptions may be relaxed in several ways ; for example we may assume that x and y belong to $L^2(\mathbb{Z}\Delta t, \mathbb{C})$ and define $x * y$ as a bounded signal.

We notice that the convolution is an associative and commutative operation. Moreover, the spectrum of the convolution between two signals is the product of the signal spectra:

$$(x * y)(f) = x(f)y(f)$$

Proof. The discrete-time Fourier transform of $x * y$ satisfies

$$(x * y)(f) = \Delta t \sum_{t \in \mathbb{Z}\Delta t} \left[\Delta t \sum_{t' \in \mathbb{Z}\Delta t} x(t')y(t - t') \right] \exp(-2i\pi ft)$$

Notice that $\exp(-2i\pi ft) = \exp(-2i\pi t'f) \exp(-2i\pi f(t - t'))$, set $\tau = t - t'$ and conclude with

$$(x * y)(f) = \left[\Delta t \sum_{t' \in \mathbb{Z}\Delta t} x(t') \exp(-2i\pi ft') \right] \left[\Delta t \sum_{\tau \in \mathbb{Z}\Delta t} y(\tau) \exp(-2i\pi f\tau) \right]$$

■

Filters

A **filter** is a convolution operator $u \mapsto y$ with kernel h:

$$u \mapsto y = h * u$$

or equivalently in the frequency domain:

$$y(f) = h(f)u(f)$$

The function $h(f)$ is the **frequency response** of the filter. We define the **(unit) impulse** as the signal $\delta : \mathbb{Z}\Delta t \to \mathbb{C}$:

$$\delta(t) = \left| \begin{array}{rl} 1/\Delta t & \text{if } t = 0 \\ 0 & \text{otherwise} \end{array} \right.$$

The (unit) impulse is a unit – in the algebraic sense – for the convolution operator : for any signal x, $x * \delta = \delta * x = x$. For this reason, when $u = \delta$, the output of the filter () is $y = h$ and h is called the filter **impulse response**.

Convolution operators are a very general class of signal transformations. Consider an operator L that maps any finite discrete-time signal u with sample time Δt to a signal with the same sample time and such that for any finite input signals u and v:

$$\forall\, \lambda,\, \mu \in \mathbb{C},\; L(\lambda u + \mu v) = \lambda L(u) + \mu L(v)$$

$$\forall\, T \in \mathbb{Z}\Delta t,\; L(t \mapsto u(t - T)) = t \mapsto L(u)(t - T)$$

Such a **linear and time-invariant(LTI)** operator is a convolution operator. Indeed, we have:

$$L(u) = L\left(\sum_{t' \in \Delta t\mathbb{Z}} u(t')\delta(t - t')\right) = \sum_{t' \in \Delta t\mathbb{Z}} u(t')L(\delta)(t - t') = u * L(\delta)$$

As a consequence, a finite response impulse filter (FIR) is a convolution operator: the definition equation

$$y(t) = \sum_{n=0}^{N-1} a_n u(t - n\Delta t)$$

corresponds to $y = h * u$ with

$$h(t) = \left| \begin{array}{rl} a_{t/\Delta t}/\Delta t & \text{if } t \in \{0, \cdots (N-1)\Delta t\}, \\ 0 & \text{otherwise.} \end{array} \right.$$

Similarly, an autoregressive system whose evolution is given by

$$y(t) = \sum_{n=0}^{N-1} a_n y(t - (n-1)\Delta t) + u(t)$$

is a convolution operator but whose impulse response is not finite.

Finite Signals

Concrete digital signals are finite because only a finite number of samples may be stored in a finite memory. We usually represent a finite sequence of values x_0, ..., x_{N-1}) and a reference step time Δt, with a finite causal signal $x : \mathbb{Z}\Delta t \mapsto \mathbb{C}$ where the missing values are replaced with 0:

$$x(t) = \left| \begin{array}{cc} x_{t/\Delta t} & \text{if } t \in \{0, \Delta t, \ldots, (n-1)\Delta t\}, \\ 0 & \text{otherwise.} \end{array} \right.$$

This signal is said to be **causal** because $x(t) = 0$ whenever $t < 0$ and **finite** because it has only a finite number of non-zero values.

If the two finite causal signals x and y correspond to the finite sequences $x_0, \ldots, x_{N-1}$ and $y_0, \ldots, y_{M-1}$ their convolution $z = x * y$ is also a finite causal signal and corresponds to the sequence $(z_0, \ldots, z_{M+N-2})$ where

$$z_k = \Delta t \sum_{(i,j) \in S_k} x_i y_j \quad \text{with} \quad S_k = \{(i,j) \in \{0, \ldots, m-1\} \times \{0, \ldots, n-1\}, i+j = k\}$$

The NumPy implementation of the operation is the function `convolve` and it assumes that $\Delta t = 1$. For example

```
>>> x = array([0.5, 0.5])
>>> y = array([0.0, 1.0, 2.0, 3.0, 4.0])
>>> z = convolve(x, y)
>>> z
array([ 0. ,  0.5,  1.5,  2.5,  3.5,  2. ])
```

Now, this approach gives us a practical method to implement filters as long as their impulse response h is finite and causal – that is when filters have a **finite impulse response (FIR)**. If h corresponds to the finite sequence $h_0, \ldots, h_{M-1}$ and the filter is to be applied to the finite signal u, then the output y corresponds to

```
>>> y = dt * convolve(h, u)
```

Design of Low-Pass Filters

Let $f_c \in (0, \Delta f/2)$ be the **cutoff frequency** of our lowpass filter. What it means is that we want is a filter that generates from a signal u an output signal y such that

$$y(f) = \left| \begin{array}{cc} x(f) & \text{if } f \in (0, f_c) \\ 0 & \text{if } f \in (f_c, \Delta f/2) \end{array} \right.$$

As the filter operation $y = h * u$ translates into $y(f) = h(f)u(f)$ in the Fourier domain (see equation ()), the frequency response of the filter shall satisfy

$$h(f) = \left| \begin{array}{cc} 1 & \text{if } f \in (0, f_c) \\ 0 & \text{if } f \in (f_c, \Delta f/2) \end{array} \right.$$

and because h is a real signal, $h(f) = \overline{h(-f)} = h(-f)$ if $f \in (-\Delta f/2, 0)$. As a consequence, the inverse DTFT formula () provides

$$h(t) = \int_{-\Delta f/2}^{+\Delta f/2} h(f) \exp(2i\pi ft)\, df = \int_{-f_c}^{f_c} \exp(2i\pi ft)\, df,\ t \in \mathbb{Z}\Delta t$$

and after straightforward computations, with the sine cardinal sinc defined as

$$\operatorname{sinc} x = \frac{\sin \pi x}{\pi x} \text{ if } x \neq 0 \text{ and } \operatorname{sinc} 0 = 1$$

we end up with

$$h(t) = 2f_c \operatorname{sinc} 2f_c t,\ t \in \mathbb{Z}\Delta t.$$

An concrete implementation of such a filter has to overcome several issues. First of all, an implementation as a FIR requires a finite number of non-zero values of $h(t)$ only. We therefore typically replace $h(t)$ with an impulse response that is equal to $h(t)$ for $|t| \leq N$ and 0 for $|t| > N$ and end up with a $2N+1$-tap filter. Then, the implementation has to be causal: the $2N+1$ coefficients are shifted to correspond to the indices 0, 1, $\cdots$, $2N$ which effectively induces a delay of N samples during the filtering (see fig. **??**, bottom figure). The generation of such low-pass filters may be implemented as

```
def low_pass(fc, dt=1.0, window=ones):
    def h(n):
        t = arange(-0.5 * (n-1), 0.5 * (n-1) + 1) * dt
        return 2 * fc * mathrm{sinc} \,(2 * fc * t) * window(n)
    return h
```

and used as follows to perform for example a 31-tap low-pass filtering of a 44100 Hz at the cutoff frequency of 8000 Hz:

```
>>> N = 15
>>> h = low_pass(fc=8000.0, dt=1.0/44100.0)(2*N+1)
>>> y = dt * convolve(h, u)
```

Note that `len(y)` is equal to `len(u) + 2*N`. A restriction of the output that compensates for the induced delay and has the same size as the original signal is obtained as `y[N:-N]`.

The optional `window` argument (that defaults to a rectangular window) is useful to reduce the **Gibbs phenomenon** that we may observe in the frequency response of the filter (see fig. **??**): an oscillation of the frequency response that may result in overshoots in the filter outputs. Windows such as `hanning`, `bartlett`, `blackman`, etc. are available in NumPy.

Spectrum Computation

Given a finite causal signal x with sample time Δt and possibly non-zero values $x_0 = x(0)$, $x_1 = x(\Delta t)$, ..., $x_{N-1} = x((N-1)\Delta t)$, the spectrum $\mathcal{F}x$ of x is given

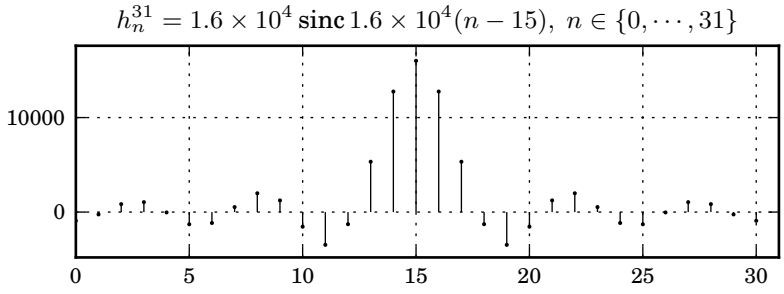

Figure 4:

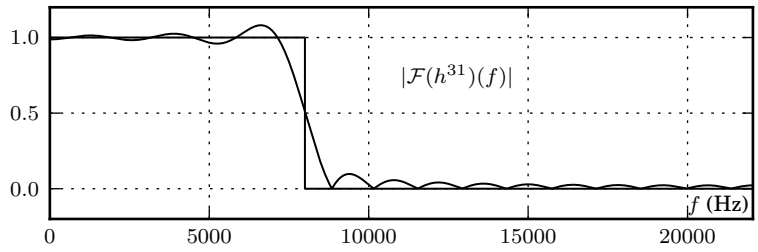

Figure 5:

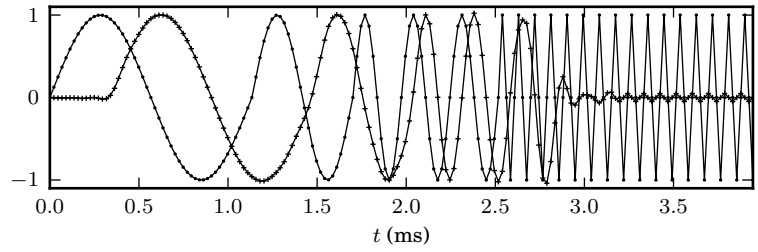

Figure 6: top : impulse response of a 31-tap low-pass filter with a cutoff frequency of $f_c = 8000$ Hz. Middle : frequency response (gain) of this filter. Bottom : a signal before (dots) and after (plus signs) low-pass filtering.

by the formula:

$$\mathcal{F}x(f) = \Delta t \sum_{n=0}^{N-1} x_n \exp(-i2\pi f n \Delta t)$$

The signal x being represented as the NumPy array `x` and the sample time Δt as a the float `dt`, a simple representation `Fx` of the spectrum $\mathcal{F}x$ – as a function taking arrays of frequencies as arguments – is given by:

```
nx = len(x)
n = reshape(arange(nx), (nx, 1))
def Fx(f):
    f = ravel(f)
    f = reshape(f, (1, len(f)))
    return dt * dot(x, exp(-1j * 2 * pi * dt * n * f))
```

The high-order programming support in Python actually allow us to automate the definition of this function and to represent the Fourier transform itself as a function `F`, that takes `x` and `dt` as arguments and returns the spectrum function `Fx`.

```
def F(x, dt=1.0):
    nx = len(x)
    n = reshape(arange(nx), (nx, 1))
    def Fx(f):
        f = ravel(f)
        f = reshape(f, (1, len(f)))
        return dt * dot(x, exp(-1j * 2 * pi * dt * n * f))
    return Fx
```

The main issue with this computation of the spectrum is performance: assume that you intend to compute N values of the spectrum, that is, as many values as there are in the signal. Then the number of sums and product needed to compute `F(x)(f)` is $\mathcal{O}(N^2)$.

An alternate idea is to compute enough spectrum values and then to use interpolation to build an approximation of the spectrum anywhere. If we decide to use N distinct spectrum values, it makes sense to compute regularly sampled values of $\mathcal{F}x(f)$ on the interval $[0, \Delta f)$ – the spectrum being $\Delta f-$periodic, there is no point going beyond this interval. We are therefore interested only in the frequencies

$$f_k = \frac{k}{N}\Delta f, \quad k = 0, \cdots, N-1$$

and in the values $\hat{x}_k = \mathcal{F}x(f_k)$ given by:

$$\hat{x}_k = \sum_{n=0}^{N-1} x_n \exp\left(-i2\pi \frac{kn}{N}\right), \quad k = 0, \cdots, N-1.$$

The transformation from the vector $(x_0, \cdots, x_{N-1})$ to the vector $(\hat{x}_0, \cdots, \hat{x}_{N-1})$ is called the **discrete Fourier transform (DFT)**:

$$\mathrm{DFT}\left[\begin{array}{ccc} \mathbb{C}^N & \to & \mathbb{C}^N \\ (x_0, \cdots, x_{N-1}) & \mapsto & (\hat{x}, \cdots, \hat{x}_{N-1}) \end{array}\right]$$

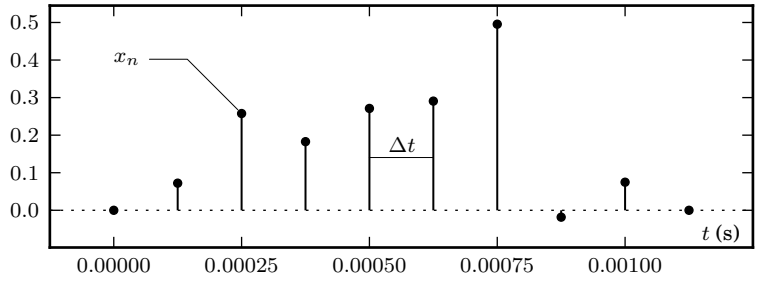

Figure 7:

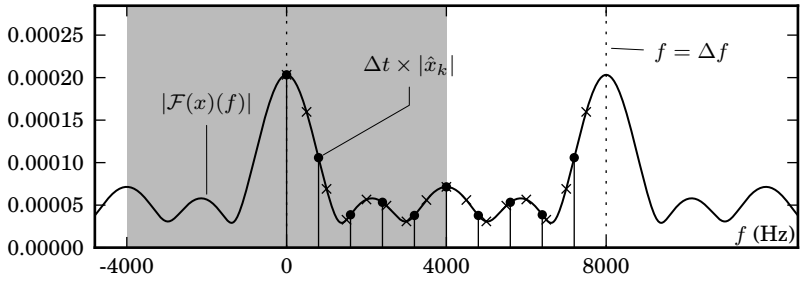

Figure 8: temporal and spectral representation of a 10-sample 8kHz signal $x(t)$. The 10-point DFT data are displayed as dots, whereas the 16-point DFT from the zero-padedd signal are displayed as crosses.

As we noted before, the straightforward implementation of the DFT has a $\mathcal{O}(N^2)$ complexity. Fortunately there is a family of algorithms called **fast Fourier transforms (FFT)** that achieve $\mathcal{O}(N \log N)$ performance instead. In NumPy, a fast Fourier transform is available as the `fft` function in the module `numpy.fft`[4]. With the help of this function, we may implement an alternative Fourier transform `F`, based on the discrete Fourier transform data and 0-order interpolation.

```
def F(x, dt=1.0):
    nx = len(x)
```

[4]The NumPy implementation is a transcription in C of the Fortran-77 'fftpack' (<http://www.netlib.org/fftpack/>)

```
fft_x = fft(x)
def Fx(f):
    k = (round_((ravel(f) * nx * dt)) % nx).astype(uint64)
    return dt * fft_x[k]
return Fx
```

We can actually compare the performance of the two approaches by measuring the time needed to compute the values $x(f_k)$, $k = 0, \cdots, N-1$, for a signal x of length N. The results are displayed in figure **??** in a log-log scale.

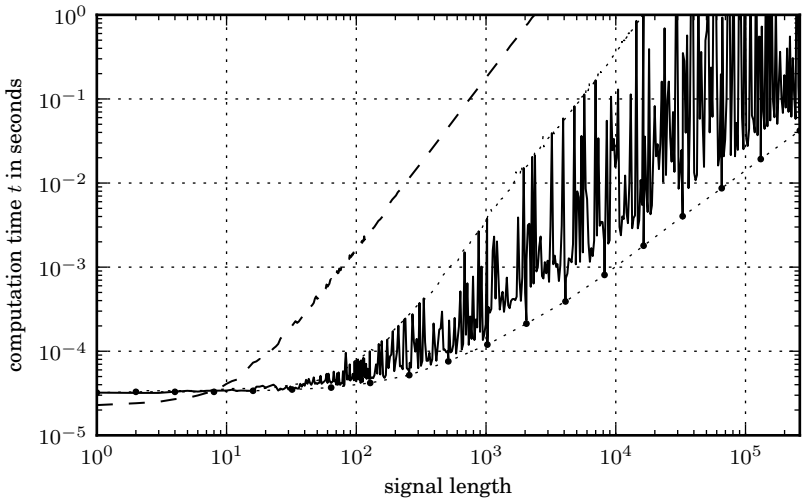

Figure 9: **spectrum computation performance:** computation time of `F(x)(f)` as a function of `nx = len(x)` with `f = arange(0, dt, dt/nx)` for a straightforward implementation (dashed curve) and a FFT-based one (solid curve) ; dots correspond to power-of-two signal length data. Reference platform: Intel i7 Q820 1.73GHz CPU, 6GiB memory.

The results for the straightforward computation method (dashed curve) are consistent with the $\mathcal{O}(N^2)$ bound as the curve exhibit an asymptotic slope of 2. For the FFT-based computation, the situation is more complex as the computation times varies strongly with respect to the signal length. The lower envelope of the curve is given by data points that correspond to signals whose length is a power of two (dotted data). For those signals, the asymptotic slope is 1, consistent with the $\mathcal{O}(N \log N)$ estimate. However, the performance may be far worse for arbitrary length, the upper enveloppe being $\mathcal{O}(N^2)$ again and is obtained for signal whose length is a prime number. This is a common artifact of many FFT algorithms: they behave well when the signal length has an integer decomposition that consists of many small primes numbers, the best case being

a power of two and the worst, a large prime number[5].

To cope with this fact, we may introduce zero-padding of the original signal: we append as many 0 values as necessary to the original vector so that its length is a power of two. We still compute data on the original signal spectrum as the signals values were 0 anyway. Note that NumPy `fft` implements 0-padding when it is given as a second argument a desired length for the fft vector larger than the signal length. As a consequence, we can support a power-of-two version of the spectrum computation with the following code.

```
def F(x, dt=1.0):
    nx = int(2**ceil(log2(len(x))))
    fft_x = fft(x, nx)
    def Fx(f):
        k = (round_((ravel(f) * nx * dt)) % nx).astype(uint64)
        return dt * fft_x[k]
    return Fx
```

Obviously, zero-padding may also be used to obtain a larger power of 2 in order to get more spectrum data. An additional parameter `n` may be given to define the minimum length of the DFT.

```
def F(x, dt=1.0, n=0):
    nx = len(x)
    nx = max(n, nx)
    nx = int(2**ceil(log2(nx)))
    fft_x = fft(x, nx)
    def Fx(f):
        k = (round_((ravel(f) * nx * dt)) % nx).astype(uint64)
        return dt * fft_x[k]
    return Fx
```

The signal we want to analyze has often more values than the ones contained in `x`. It may for example be – at least conceptually – be infinite, for example if it is a pure tone. The FFT-based spectral analysis is therefore based on a window of the original signal ; the most common choice is a rectangular window where we select some of the valued of the signal (multiply by 1) and implicitly consider that all other values are 0 (multiply by 0). Using a multiplication by a window function whose behavior is smoother on the window boundary is a classical method to improve the resolution of harmonics in the spectrum. Refer for example to [**?**] for a discussion on this subject and a comparison of the usual windows (such as `bartlett`, `hamming`, `hanning`, etc.). A version of the spectrum that support windows is given by

```
def F(x, dt=1.0, n=0, window=ones):
```

[5]This is explained by the fact that those algorithms use a divide-and-conquer approach to solve the problem. However, there are algorithms that may achieve an asymptotic $\mathcal{O}(N \log N)$ performance, even for signals of length a prime number, see for example [**?**].

```
    nx = len(x)
    x  = window(nx) * x
    nx = max(n, nx)
    nx = int(2**ceil(log2(nx)))
    fft_x = fft(x, nx)
    def Fx(f):
        k = (round_((ravel(f) * nx * dt)) % nx).astype(uint64)
        return dt * fft_x[k]
    return Fx
```

Multirate Signal Processing

Signal processing systems are **multirate** when they manage signals with different sample times. Filters, introduced in the previous sections, do not alter the sample time of the signals they are applied to, but **decimators** and **expanders** do; they are the new building blocks that allows us to **downsample** – decrease of the data rate – and **upsample** – increase in the data rate, while controlling the impact of these operations on the signal spectral content. A downsampling of a factor 5 for example may be used to get a 44.1 kHz signal down to a 8.82 kHz rate – which is still satisfactory for voice signals – and upsampling can be used to go back to the original data rate.

Decimation and Expansion

Decimation

The **decimation** of a factor M of a discrete signal x with a sampling time of Δt is the signal with a sampling time of $M\Delta t$ denoted $x \downarrow M$ and defined by:

$$x \downarrow M)(t) = x(t),\ t \in \mathbb{Z}M\Delta t.$$

If x is a finite causal signal represented by the NumPy array `x`, the implementation of decimation of a factor `M` is straightforward with the slicing mechanism of arrays:

```
def decimate(x, M=2):
    return x[::M].copy()
```

Note that the length of `decimate(x,M)` is `len(x)/M` – the quotient of the integer `len(x)` by the integer `M` – if `len(x)` is a multiple of `M` but `len(x)/M + 1` otherwise. The copy may be necessary in some use cases and is therefore included to be safe. Indeed, the slicing operation has a pass-by-reference semantics in

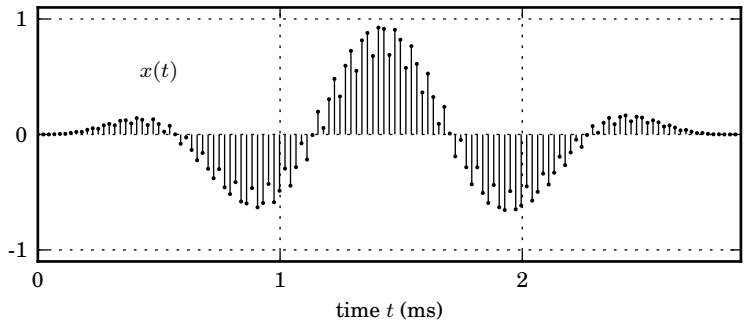

Figure 10:

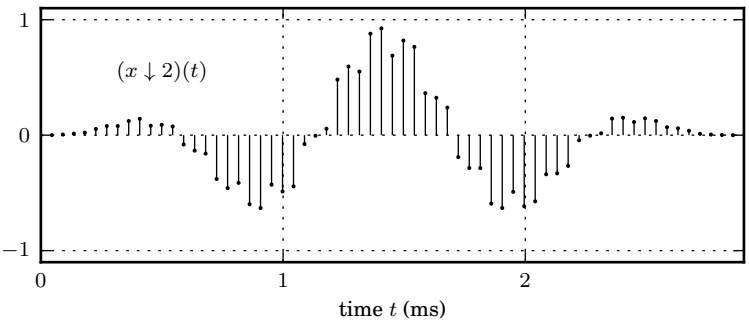

Figure 11: **Decimation, Temporal View:** a 44.1 kHz sampled signal $x(t)$ and its decimated version (by a factor of 2).

NumPy: `x[::M]` is not a copy of the content of `x` but merely a view into it, therefore a change in the values of `x` would also change the sliced data[6]

Decimation has a arguably strange – but well-defined – effect on the spectrum of a signal. Consider for example the decimation of factor 2 on the signal x with a sampling time of Δt:

$$\begin{aligned}(x \downarrow 2)(f) &= 2\Delta t \sum_{t \in \mathbb{Z}2\Delta t} (x \downarrow 2)(t) \exp(-2i\pi ft) \\ &= 2\Delta t \sum_{t \in \mathbb{Z}2\Delta t} x(t) \exp(-2i\pi ft) \\ &= \Delta t \sum_{t \in \mathbb{Z}\Delta t} x(t) \exp(-2i\pi ft) + \Delta t \sum_{t \in \mathbb{Z}\Delta t} (-1)^{t/\Delta t} x(t) \exp(-2i\pi ft)\end{aligned}$$

As we have

$$(-1)^{t/\Delta t} = \exp(-i\pi)^{t/\Delta t} = \exp(-2i\pi t/2\Delta t)$$

we end up with

$$(x \downarrow 2)(f) = \Delta t \sum_{t \in \mathbb{Z}\Delta t} x(t) \exp(-2i\pi ft) + \Delta t \sum_{t \in \mathbb{Z}\Delta t} x(t) \exp(-2i\pi (f + 1/2\Delta t)t)$$

and therefore

$$(x \downarrow 2)(f) = x(f) + x(f + \Delta f/2)$$

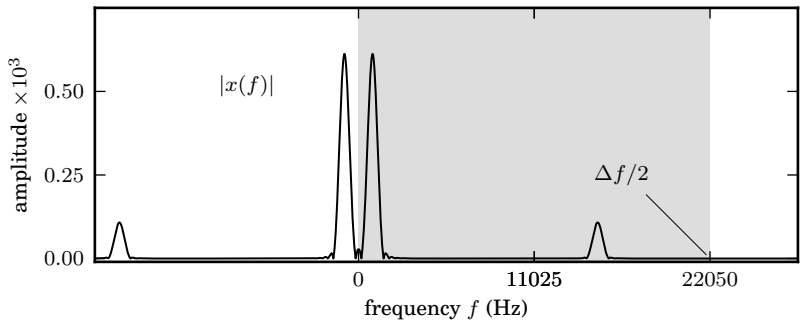

Figure 12:

For a decimation of factor M, we obtain by similar computations

$$(x \downarrow M)(f) = \sum_{k=0}^{M-1} x(f + k\Delta f/M)$$

[6] see `http://www.scipy.org/NumPy_for_Matlab_Users`.

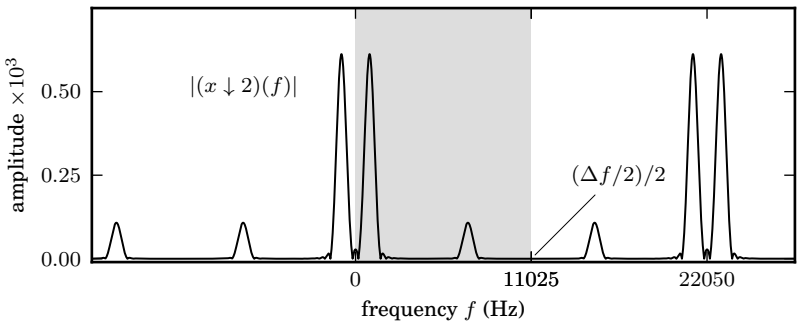

Figure 13: **Decimation and Spectrum:** spectra of a 44.1 kHz sampled signal, before and after a decimation of factor 2 (temporal view: fig. **??**). Doubling the sample rate Δt effectively halves Δf and therefore halves the frequency range where the spectral content is significant.

What this formula means is that after decimation of a factor M, the spectral content of the signal at frequency $f \in [0, \Delta f/M)$ is a mix of the spectral content of the original signal at the frequencies

$$f,\ f + \Delta f/M,\ f + 2\Delta f/M,\ \cdots\ nf + (M-1)\Delta f/M.$$

This phenomenon is called **(spectral) folding**. As every frequency in the original signals has generated copies of itself at new frequencies, the phenomenon is also called **(spectral) aliasing**.

Expansion

The **expansion** of factor M applies to a discrete signal x with a time step Δt and creates a signal with a time step $\Delta t/M$ denoted $x \uparrow M$ and defined by:

$$(x \uparrow M)(t) = \left| \begin{array}{cl} x(t) & \text{if } \ t \in \mathbb{Z}\Delta t \\ 0 & \text{otherwise.} \end{array} \right.$$

Again, the implementation in NumPy for finite causal signals is straightforward:

```
def expand(x, M=2):
    output = zeros(M * len(x))
    output[::M] = x
    return output
```

The length of `expand(x, M)` is `M * len(x)`. It could be reduced to {`(M - 1) * len(x) + 1`} without any information loss as the last `M - 1` values of `output` are zeros, but it is often convenient to obtain a signal whose length is a multiple

of the expansion factor. This operation does not alter the shape of the spectral content of the signal:

$$\begin{aligned}(x \uparrow M)(f) &= (\Delta t/M) \sum_{t \in \mathbb{Z}\Delta t/M} (x \uparrow M)(t) \exp(-2i\pi t f) \\ &= (\Delta t/M) \sum_{t \in \mathbb{Z}\Delta t} x(t) \exp(-2i\pi t f)\end{aligned}$$

and therefore

$$(x \uparrow M)(f) = \frac{1}{M} x(f)$$

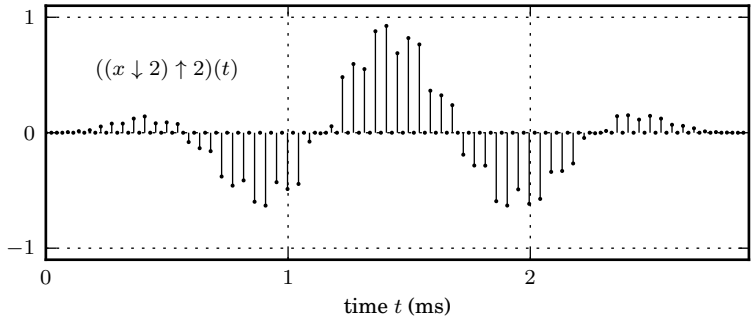

Figure 14:

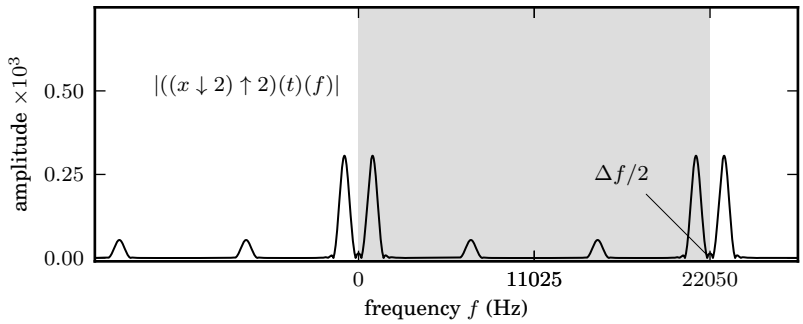

Figure 15: **Expansion :** temporal and spectral views of the signal x of figure **??** after downsampling then upsampling, both of a factor 2.

Downsampling and Upsampling

Decimation is the basic operation to reduce the data rate of a signal and therefore compress it. However, this operation creates aliases in the spectral content of the signal where high and low-frequency are mixed and cannot be separated one from the other anymore. We can however decide to get rid in a controlled manner of some spectral content of the signal to keep the rest intact.

Note that if the spectral content of a signal before decimation of factor M is entirely into the $(-\Delta f/2M, \Delta f/2M)$ band, aliasing does not happen as we have

$$(x \downarrow M)(f) = x(f) \text{ if } f \in (-\Delta f/2M, \Delta f/2M)$$

This can be achieved if we filter the original signal with a perfect low-pass filter of cutoff frequency $f_c = \Delta f/2M$. We then lose signal information in all frequency bands but the one of lowest frequency, but at least, this one is perfectly preserved by decimation. We call this combination of low-pass filtering and decimation **downsampling**.

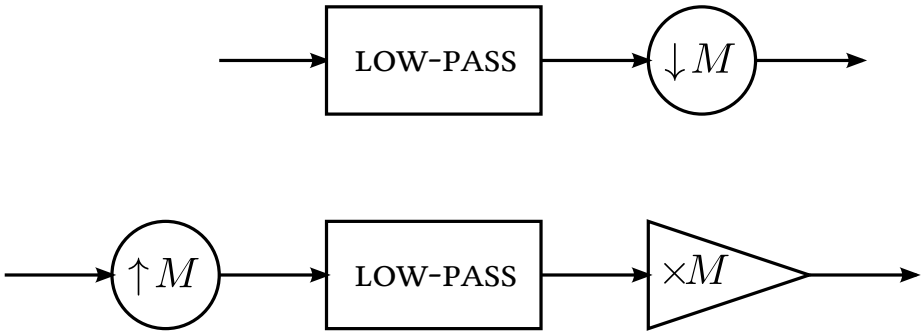

Figure 16: downsampling and upsampling diagrams

Reconstruction of the (low-frequency content of) original signal is then just a matter of getting back the the original rate, by expansion and apply a gain of M to the result. That leads to exactly the right spectrum in the band $(-\Delta f/2M, \Delta f/2M)$ but not in the rest of $(-\Delta f/2, \Delta f/2)$ as the spectrum is $\Delta f/2M$-perodic. To get rid of the high frequency content, we simply apply the perfect low-pass filter with cutoff frequency $f_c = \Delta f/2M$. The combination of (zero-)expansion, gain and filtering is called *upsampling*.

Let's summarize this: a downsampling of order M allows to reduce the data rate by a factor of M and keeps information one M-th of the spectral range – the lowest frequency part. Upsampling may be used to reconstruct a signal at the original rate whose content is the low-frequency content of the original one and has no higher spectral components.

Ideal Filter Banks and Perfect Reconstruction

In the previous section we have explained how we could divide the signal rate by a factor of M by keeping only one M-th of its spectral content and throwing away everything else. We now consider the steps leading to a more flexible approach: we split the data signal into M frequency bands and we will later design methods to allocate bits to such or such a band depending on the spectral content of the signal.

In order to split the signal into M uniformly spaced spectral bands, we introduce an **analysis filter bank**: a set of M filters a^i, $i = 0, \cdots, M-1$ with

$$a_n^i = a^i(n\Delta t),\ n \in \mathbb{Z}$$

that we all apply to the original signal. All the filters all band-pass, with low frequency $i\Delta f/M$ and high-frequency limit $(i+1)\Delta f/M$. We then decimate the signal on all branches, so that the original data rate can be kept. We know what is the spectral content of the signal after decimation on the branch $i = 0$, but what is going on with the other branches ? Let x be the original signal and x^i is the signal filtered by the i-th filter. The content of x^i is entirely in the i-th frequency band, that is $(i\Delta f/M, (i+1)\Delta f/M)$ (and the corresponding negative frequency band), so after decimation, the spectral content is

$$\sum_{k=0}^{M-1} x(f - k\Delta f/M) = x^i(f + i\Delta f/M) \ \text{ if } \ f \in (-\Delta f/2M, \Delta f/2M)$$

So again, in each branch, decimation has kept the relevant information. Given those M spectral components, are we able to reconstruct the original signal ? In order to get the contribution from the i-th band in the right place, we can first expand the signal and multiply by M: that shifts the subband content to build a $\Delta f/M$-periodic spectral content. To get this content only in the i th band, we apply a perfect pass-band that corresponds to the i-th subband. Then we sum all these contributions.

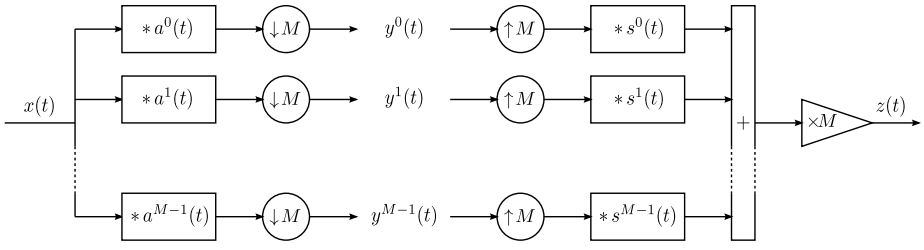

Figure 17: Analysis and synthesis filter banks diagrams

Filter Banks and Perfect Reconstruction

One issue with the previous scheme is that perfect band-pass filters cannot be implemented. We'd like as to replace them by some finite impulse response filters approximations, study if perfect reconstruction is still possible and if it can't be achieved, measure the error we are introducingg.

Consider the diagram in figure (**??**) where the a^i and s^i are the impulse response of the analysis and synthesis filter banks. The formulas {(**??**) and (**??**)} yield the following expression for the output z of the analysis + synthesis process from the input signal x:

$$z(f) = \sum_{k=0}^{M-1} \left[\sum_{i=0}^{M-1} s_i(f) a_i(f + k\Delta f/M) \right] x(f + k\Delta f/M)$$

which means that the diagram will achieve **perfect reconstruction** if we have

$$\sum_{i=0}^{M-1} s_i(f) a_i(f + k\Delta f/M) = \left| \begin{array}{ll} 1 & \text{if } k = 0, \\ 0 & \text{if } k = 1, \cdots M-1. \end{array} \right. ,$$

or in other words, if all the **distorsion functions** $D_k(f)$, $k = 0, \cdot, M-1$, defined by

$$D_0(f) = \sum_{i=0}^{M-1} s_i(f) a_i(f) - 1,$$
$$D_k(f) = \sum_{i=0}^{M-1} s_i(f) a_i(f + k\Delta f/M),\ k = 1, \cdots M-1$$

are identically zero.

Cosine Modulated Filter Banks

We build in this section a family of pass-band filters with impulse responses $a^i(t)$, $i = 0, \cdots, M-1$, whose pass-band is $(i\Delta f/M, (i+1)\Delta f/M)$, and based on a single prototype filter. The prototype is selected as a low-pass filter with cutoff frequency $f_c = \Delta f/4M$; the perfect prototype filter impulse response is (see (**??**)):

$$h(n\Delta t) = \frac{\Delta f}{2M} \text{sinc}\, \frac{\Delta f}{2M} n\Delta t = \frac{\Delta f}{2M} \text{sinc}\, \frac{n}{2M}$$

To generate the i-th pass-band filter, all we have to do it to shift the spectrum by $(i + 0.5)\Delta f/2M$ to the right, that is, multiply $h(n\Delta t)$ by

$$\exp(i2\pi(i + 0.5)(\Delta f/2M)(n\Delta t)) = \exp(i\pi(i + 0.5)n/M).$$

But then the filter impulse response would no longer be real, so we also perform the the opposite frequency shift : we multiply $h(n\Delta t)$ by $\exp(-i\pi(i+0.5)n/M)$ and add up both contributions ; we end up with

$$a^i(n\Delta t) = 2h(n\Delta t) \times \cos\left(\pi(i+0.5)n/M\right)$$

that is, a **cosine modulated filter bank**. The figure (**??**) displays the filters frequency responses where the prototype filter has been approximated by a FIR.

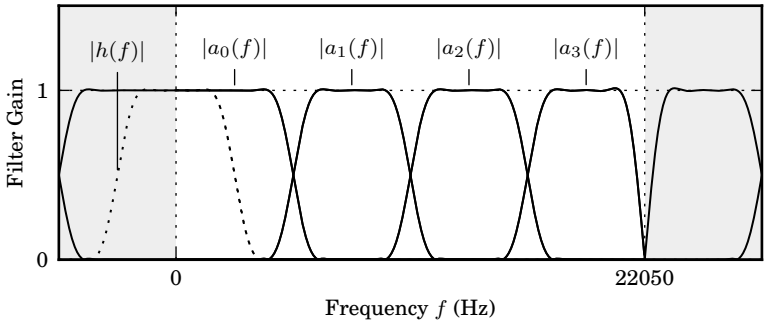

Figure 18: **Cosine Modulated Filter Banks:** gain of the components of a cosine modulated filter bank as a function of the frequency f. The sampling time Δt is 44.1 kHz, there are $M = 4$ filters and the prototype filter h – whose spectrum gain is dotted – has been truncated by the application of a Hanning window if length $N = 64$. The phase of such filter is not displayed as it is flat because the filters impulses responses are symmetrical – or linear as a function of f in a causal implementation.

If we selecting as synthesis filters the same pass-band filters used for the analysis – $s^i(t) = a^i(t)$ – we maye compute the distorsions induced by the analysis-synthesis process ; the results, displayed in figure (**??**), clearly points out that the basic approach we have adopted so far does not provide a good approximation to a perfect reconstruction.

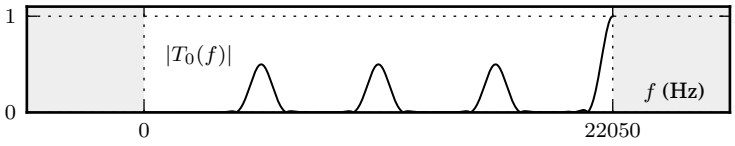

Figure 19:

Pseudo-QMF (for **pseudo - quadrature mirror filters**) may be introduced to obtain sufficiently small distorsion functions ; they are successfully used in layer I and II of MPEG-Audio for example. Their design relied on two modifications with respect to our approach so far. First, we introduce phase factors

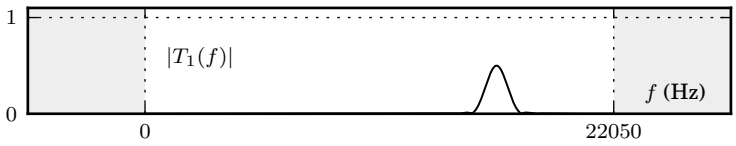

Figure 20:

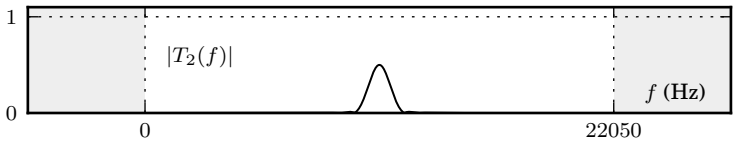

Figure 21:

ϕ_i in the definition of a^i and s^i

$$a^i(n\Delta t) = 2h(n\Delta t) \times \cos\left(\pi(i+0.5)n/M + \phi_i\right)$$
$$s^i(n\Delta t) = 2h(n\Delta t) \times \cos\left(\pi(i+0.5)n/M - \phi_i\right)$$

in order to cancel significant aliasing terms and ensure a relatively flat overall magnitude distorsion (see [**?**, **?**]). Among several options, we select

$$\phi_i = \frac{\pi}{2}\left(\frac{N-1}{M} - 1\right)(i+0.5)$$

where N is the filter length and M the number of sub-bands.

Then the selection of the prototype filter does not rely on the expression of the perfect low-pass filter but is optimized to reduce distorsion. The MPEG-Audio standard selection for this filter is displayed in figure **??**.

Polyphase Representation of Filters Banks

Polyphase representation is an alternate description of filter banks that is suited to a real-time implementation. Unlike convolution-based implementation that require the full input values to be available to produce output values, polyphase

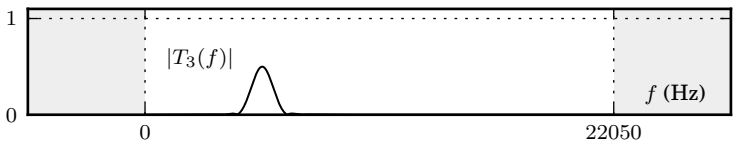

Figure 22: {cosine modulated filter banks: distortions}

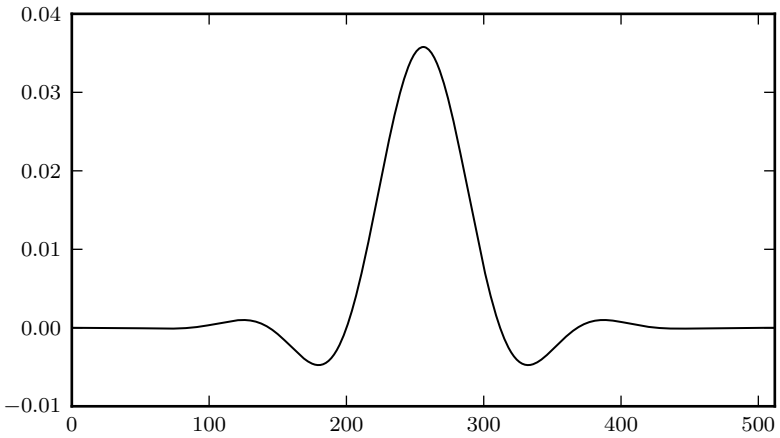

Figure 23: MPEG-Audio Layer I and II prototype 513-tap prototype filter

representation of analysis and synthesis filter banks are amenable to matrix implementation that work frame by frame: they consume chunks of M samples to produce the same amount of output values.

Analysis Filter Bank

Consider the analysis filter bank depicted on the left of figure **??**. Gather the output $y^i(t)$ of the i-th subbands into the vector signal $y(t) \in \mathbb{R}^M$, $t \in \mathbb{Z}M\Delta t$. This output vector is related to the input $x(t) \in \mathbb{R}$, $t \in \mathbb{Z}\Delta t$ by the formula:

$$y^i(t) = \Delta t \sum_{t' \in \mathbb{Z}\Delta t} a^i(t') x(t - t'),\ t \in \mathbb{Z}M\Delta t$$

Let $y^i_n = y^i(nM\Delta t)$, $x^i_n = x^i(n\Delta t)$, $a^i_n = a^i(n\Delta t)$. This relationship takes the form

$$y^i_j = \Delta t \sum_{n=0}^{N-1} a^i_n x_{Mj-n},$$

which can be considered as a simple matrix multiplication:

$$y_n = \mathcal{A} \begin{bmatrix} x_{Mn} \\ x_{Mn-1} \\ \vdots \\ x_{Mn-N+1} \end{bmatrix} \quad \text{with} \quad \mathcal{A} \in \mathbb{R}^{M \times N},\ \mathcal{A}_{ij} = \Delta t \cdot a^i_j.$$

Alternatively, this form may be turned into an alternate block-diagram displayed in figure **??** that is the **polyphase** representation of the analysis filter bank:

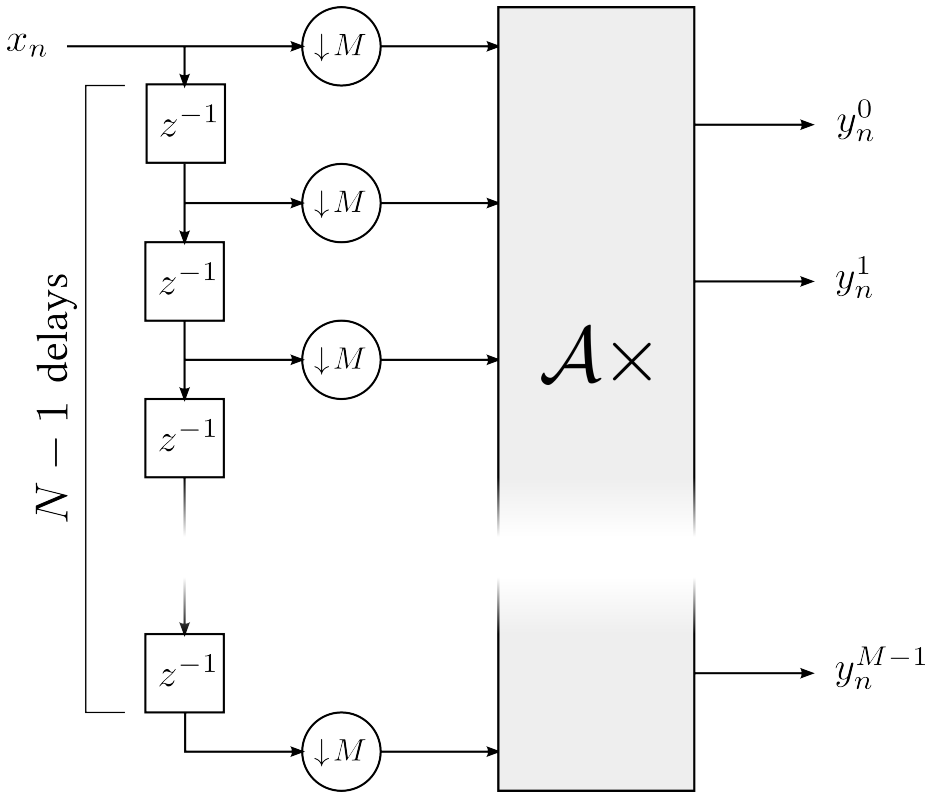

Figure 24: analysis filter bank: polyphase representation

assume that N is a multiple of M and that we intend to apply the analysis filter bank to a finite causal signal x represented by the NumPy array `x`. We notice that the vector in right-hand side of the equation () acts as a buffer: every new value of n shifts the oldest values of x towards the bottom of the vector by M slots – effectively forgetting M of the oldest values – and introduces M new values of x at the top, so the signal x is used in frames of M samples. We also notice that y_0 does not depend of a whole x frame, only of x_0. To simplify this matter, we assume that $x_0 = 0$ and won't compute y_0. Effectively, we implement a process that with respect to the theoretical one delays the input by one step – so that x_1 is the first non-zero value, not x_0 – and advances the output by one step – the first output we'll effectively compute is truly y_1, not y_0.

These computations may be carried by an instance of the `Analysis` class:

```
class Analysis(object):
```

```
    def __init__(self, a, dt=1.0):
        self.M, self.N = shape(a)
        self.A = a * dt
        self.buffer = zeros(self.N)
    def __call__(self, frame):
        frame = array(frame, copy=False)
        assert shape(frame) == (self.M,)
        self.buffer[self.M:] = self.buffer[:-self.M]
        self.buffer[:self.M] = frame[::-1]
        return dot(self.A, self.buffer)
```

The argument `a` in the `Analysis` constructor is meant to be a the 2-dim. array such that `a[i,:]` represent the i-th analysis filter impulse response. In order to use the instance `analysis = Analysis(a, dt)`, the array `x` has to be split in frames of length `M`.

In the implementation of the analysis filter banks for the MPEG PQMF, the pass-band filters are implemented as causal filters, introducing an extra delay of `MPEG.N / 2 = 256` samples. Given the the implementation delays already considered, the total delay induced by the implementation with respect to the original filter banks is `MPEG.N / 2 + 1 - MPEG.M`.

To make sure that the analysis filter banks has produced all its non zero-values, we feed the system extra zero frames. If the input data is available from the start in the array `x`, the corresponding output `y` may therefore be obtained as:

```
from filters import MPEG
from frames import split

x = r_[x, zeros(MPEG.N)]
frames = split(x, MPEG.M, zero_pad=True)
y = []
for frame in frames:
    y.extend(analysis(frame))
y = array(y)
```

Synthesis Filter Bank

Consider the synthesis filter bank depicted on the right of the diagram **??**. The output vector $z(t) \in \mathbb{R}$, $t \in \mathbb{Z}\Delta t$, is related to the input $y(t) \in \mathbb{R}^M$, $t \in \mathbb{Z}M\Delta t$, by the formula:

$$z(t) = M \sum_{i=0}^{M-1} \Delta t \sum_{t' \in \mathbb{Z}\Delta t} s^i(t')(y^i \uparrow M)(t - t')$$

or – using integer indices – by

$$z_n = M \sum_{i=0}^{M-1} \Delta t \sum_{j=0}^{N-1} s_j^i (y^i \uparrow M)_{n-j}.$$

With

$$\mathcal{S} = \left[M \Delta t s_j^i \right]_{i,j}$$

we may turn this equation into

$$z_n = \sum_{j=0}^{N-1} [\mathcal{S}^t (y \uparrow M)_{n-j}]_j.$$

Now consider the polyphase synthesis diagram **??**, dual of the analysis diagram **??**, where $\mathcal{P}$ is an unknown $N \times M$ matrix. Its output is related to its input by

$$z_n = \sum_{j=0}^{N-1} [\mathcal{P} (y \uparrow M)_{n-j}]_{N-1-j}.$$

So if we set $\mathcal{P} = J\mathcal{S}^t$, where J is the $M \times M$ matrix such as $J_{i,j} = 1$ if $i - j = M - 1$ and 0 otherwise, the diagram outputs the same thing as (), only delayed by $N - 1$ samples.

Let $w_n = \mathcal{S}^t y_n$. A careful examination of the polyphase representation of the synthesis filter banks show that the computation may be performed in frames of M values. Indeed, the output z_n is given by $z_0 = w_0^{N-1}$, $\cdots$, $z_{M-1} = w_0^{N-M}$, then $z_M = w_0^{N-M-1} + w_1^{N-1}$, $z_{M+1} = w_0^{N-M-2} + w_1^{N-2}$, etc.

Here is a possible implementation:

```
class Synthesis(object):
    def __init__(self, s, dt=1.0):
        self.M, self.N = shape(s)
        self.P = transpose(self.M * dt * s)[::-1,:]
        self.buffer = zeros(self.N)
    def __call__(self, frame):
        frame = array(frame, copy=False)
        assert shape(frame) == (self.M,)
        self.buffer += dot(self.P, frame)
        output = self.buffer[-self.M:][::-1].copy()
        self.buffer[self.M:] = self.buffer[:-self.M]
        self.buffer[:self.M] = zeros(self.M)
        return output
```

Again, the input vector `y` can be extended with as many zeros as necessary to get all non-zero output values extracted from the buffer.

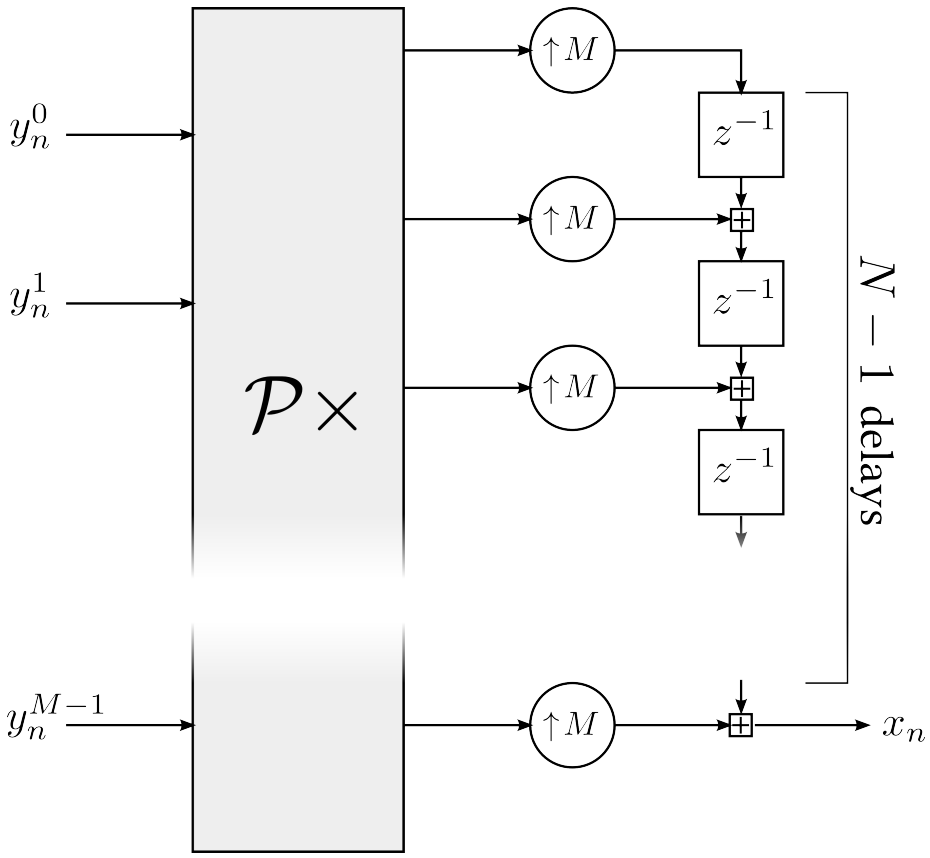

Figure 25: Synthesis filter bank: polyphase representation

Psychoacoustics - Perceptual Models

Acoustics - Physical Values

Audio signal values represent a variation of the sound pressure with respect to the atmospheric pressure. The standard unit used to measure such pressure is the pascal (Pa), a unit that corresponds to $\mathrm{N/m^2}$; while atmospheric pressure is around 100 Pa – the standard atmosphere (atm), an alternative unit, is equivalent to 101.325 Pa – variations of the pressure in the audio context typically range from 10^{-5} Pa (absolute threshold of hearing) to 10^2 Pa (threshold of pain).

In order to measure the **sound pressure level (SPL)** denoted L, we focus on the difference $p(t)$ between the actual pressure and the atmospheric pressure and compute its quadratic mean:

$$P^2 = \langle p^2 \rangle$$

where $\langle \cdot \rangle$ denotes, depending on the context, either a temporal mean or a probabilistic one. We then normalize this value with respect to $P_0 = 20$ μPa and measure the ratio in a logarithmic scale

$$L = 20 \log_{10} \frac{P}{P_0}$$

The sound pressure level unit is the **decibel (dB)**. When the sound is a plane travelling wave, the normalized **sound intensity** I is related to p by

$$\frac{I}{I_0} = \frac{P^2}{P_0^2} \quad \text{with} \quad I_0 = 10^{-12}\,\mathrm{N/m^2}$$

so that

$$L = 10 \log_{10} \frac{I}{I_0}.$$

Now, sound pressure level may be computed according to the spectral content of the signal: if $p(f)$ denotes the spectrum of the signal $p(t)$, a finite causal signal of length N with $T = N\Delta t$, we have by Parseval's formula (cf. formula ())

$$P = \frac{1}{T}\Delta t \sum_{n=0}^{N-1} p(t)^2 = \frac{1}{T} \int_{-\Delta f/2}^{\Delta f/2} |p(f)|^2 \, df$$

as we end up with

$$L = 10 \log_{10} \frac{2}{T} \int_0^{\Delta f/2} \frac{|p(f)|^2}{P_0^2} \, df$$

This result is usually presented in terms of **sound (intensity) density** also commonly called **sound power density**, a value measured in dB that we denote $\ell(f)$ and define as

$$\ell(f) = 10 \log_{10} \frac{2}{T} \frac{|p(f)|^2}{P_0^2}$$

so that

$$L = 10 \log_{10} \int_0^{\Delta f/2} 10^{\ell(f)/10}\, df.$$

For example, if a sound has a constant power density ℓ in a frequency range of width ΔF and no power outside this range, its sound pressure level is

$$L = \ell + 10 \log_{10} \Delta F$$

Digital audio signal being scaled to fit the range of their quantizer, we need to normalize somehow the signal before getting into the SPL computation. For 16-bit linearly quantized signals normalization takes place in the following way: first, scale to fit into $[-1.0, 1.0]$, then a quadratic mean of N signal values

$$X^2 = \langle x^2 \rangle = \frac{1}{T} \Delta t \sum_{n=0}^{(N-1)} |x(n\Delta t)|^2$$

is mapped to SPL with the formula

$$L = 10 \log_{10} \langle x^2 \rangle + 96\,\mathrm{dB}$$

or equivalently

$$P = 10^{4.8} P_0 \times X$$

or even

$$\ell(f) = 10 \log_{10} \left(\frac{2}{T} |x(f)|^2 \right) + 96$$

and

$$L = 10 \log_{10} \left(\frac{2}{T} 10^{9.6} \int_0^{\Delta f/2} |x(f)|^2\, df \right)$$

Threshold in Quiet

To understand why the normalization in the previous section actually makes sense, we need to know more about the human hearing range. The **absolute threshold of hearing (ATH)** or **threshold in quiet** is a function of the frequency f such that a pure tone with a frequency f will be noticed if and only if its SPL is above the ATH at this frequency. An approximate analytical model for the threshold of hearing – as a function of the frequency f in kHz – is:

$$T_a(f) = 3.64 f^{-0.8} - 6.5 \exp(-0.6\,(f - 3.3)^2) + 10^{-3} f^4$$

Now consider the values of a uniform 16-bit quantizer on $[-1.0, 1.0]$. Given that the maximum possible value of $\langle x^2 \rangle$ is 1.0, the maximum value of the right-hand side of the equation () is 96 dB. For a pure tone, before normalization, the

maximum value of $\langle x^2 \rangle$ is $1/\sqrt{2}$, and therefore the maximum sound pressure level is ≈ 93 dB.

Now, even if there is no minimum value *per se*, a good reference is the quantization noise energy. For a 16-bit linear quantization on $(-1.0, 1.0)$, the step size Δ is uniform with a value of $2.0/2^{16} = 2^{-15}$. In the context of the high resolution hypothesis, the equation (**??**) provides the estimate $\mathbb{E}[b^2] = \Delta^2/12$ for the noise power and therefore the normalized noise pressure level is

$$10 \log_{10} \left(2^{-30}/12\right) + 96 \approx -5.1 \,\text{dB}$$

So with the normalization of the previous section, the practical range in terms of sound pressure level of the 16-bit linear quantizer is $[-5.1, 93]$ dB, that is approximately a 100 dB range. Compared to the reference curve for the absolute threshold of hearing (see fig. **??**), we notice that this region covers essentially all of the frequency range from 20 Hz to 20 kHz and that the low bound closely matches the minimal value of the ATH. So this scaling by 96 dB would correspond to a kind of optimal amplification configuration of the loudspeakers, one that would allow to get into large audible value of the SPL without saturation of the signal and also to allow proper perception of the lowest sounds that the ear can actually detect.

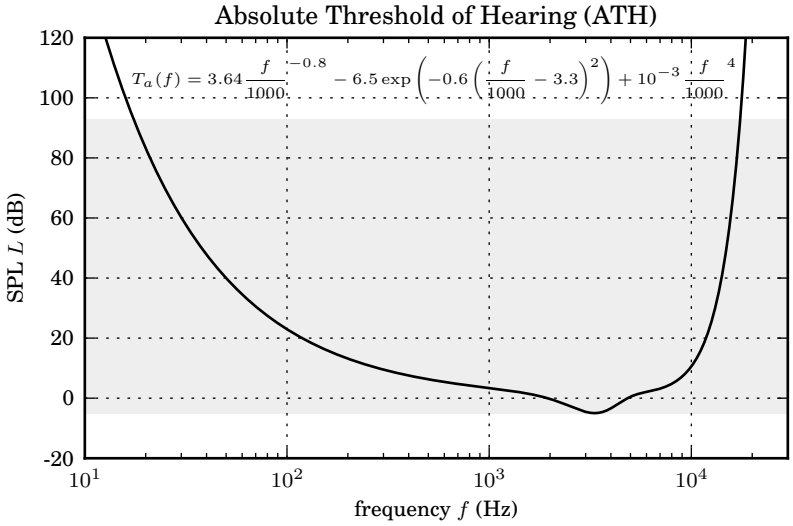

Figure 26: **Absolute Threshold of Hearing.** The grey region denotes the SPL range covered by a 16-bit linear quantization.

Simultaneous Masking

Beyond the model of the absolute threshold of hearing, the main characteristic of the psychoacoustic system that perceptual model used in technologies such as MP3, Ogg/Vorbis, AAC, etc. rely on is {**simultaneous masking**}. **Basically, a loud sound whose energy is located in a given narrow frequency range is going to make every other signal located in the same frequency neighbouhood harder to detect.**

A simple first computational model for this type of masking relies on Fletcher's critical band concept. Fletcher considers the possible masking of a pure tone with frequency f by a signal whose energy is located in the $[f - \Delta F/2, f + \Delta F/2]$ range and makes the assumption that there exist a critical bandwidth ΔF_c – or a critical band $[f - \Delta F_c/2, f + \Delta F_c/2]$ – such that:

1. **the distribution of the intensity of the masker within the critical band does not influence the outcome of the masking experiment, only the total SPL for the critical band matters,**
2. **no amount of intensity outside of the critical band may change the outcome of the experiment,**
3. **masking occurs when the intensity of the masker in the critical band exceeds the intensity of the test tone.**

In a few words, the critical band is the largest region around the test tone where the power density of masker signals consistently increases the masking effect. This set of assumption has a number of shortcomings: the distribution of intensity nearby the test tone *does* matter, but not so much as long as the distribution is quite uniform (say a band-limited noise or a combination of 5 pure tones uniformly gathered won't make much of a difference), the influence of the distribution of energy does not have a drop from 100% to 0% at a limit but is smoother and finally, the masker needs from 2 to 4 more intensity than the test tone to properly mask it.

Despite all these shortcomings, Fletcher's model of critical bands is important and estimates of the critical bandwidth as a function of the frequency center frequency f may be derived from simple experimental protocols. An approximate analytical formula for the critical bandwith in Hz – as a function of the center frequency f in Hz is:

$$\Delta F_c = 25 + 75(1 + 1.4(f/1000.0)^2)^{0.69}$$

it also accepts the following piecewise linear approximation:

$$\Delta F_c = 100 \text{ Hz} \quad \text{if } f \leq 500Hz \quad \text{and} \quad \Delta F_c = 0.2 \times f \text{ beyond.}$$

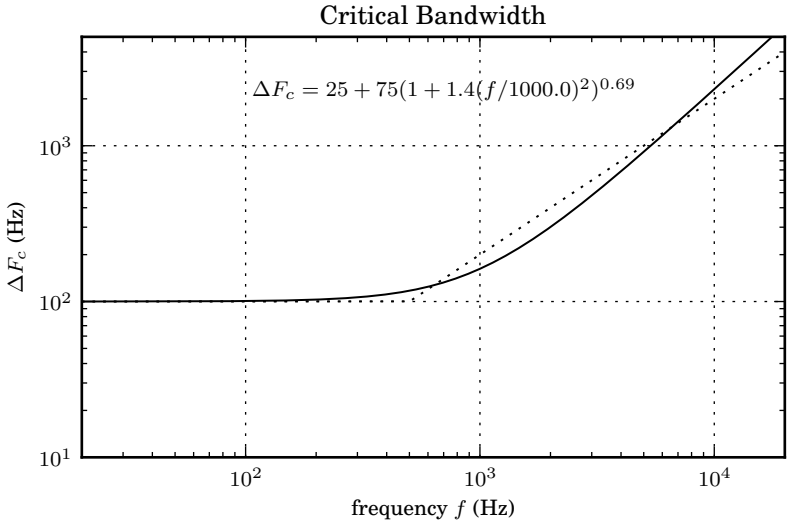

Figure 27: **Critical Bandwidth.** The dashed lined represent the simpler piecewise linear estimate.

The critical band concept is used in many model beyond masking ; for convenience, a unit is introduced to measure frequencies in the critical band rate scale : it is named the Bark. By convention, $f = 0$ **Bark corresponds to** $f = 0$ **Hz. Then the right end of the critical band that starts at** 0 **Bark corresponds to** 1 **Bark, the right end of the critical band that starts at** 1 **Bark corresponds to** 2 **Bark, and so on and so forth. A convenient analytical approximation of the Hz to Bark conversion is given by:**

$$f\,[\textbf{Bark}] = 13.0 \arctan(0.76 f/1000.0) + 3.5 \arctan(f/1000.0/7.5)^2$$

Spreading Functions

Let's consider for a moment the masks yielded by Fletcher's set of assumption, that is the audibility threshold of pure tones as a function of their frequency f**. We may consider as maskers either pure tones or band-limited white noises. The graphs in figure ?? is an example of the masks levels that can be derived from Fletcher's model of masking.**

If we assume that elementary maskers combine into a global one by addition of intensity – and take into account the absolute threshold of hearing as yet another mask, we end up with the kind of masking curves displayed in figure ??.

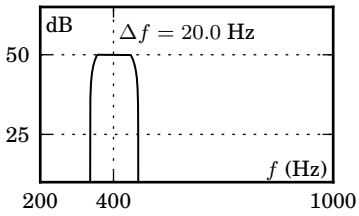

Figure 28:

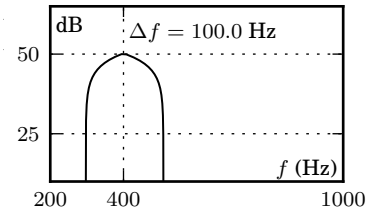

Figure 29:

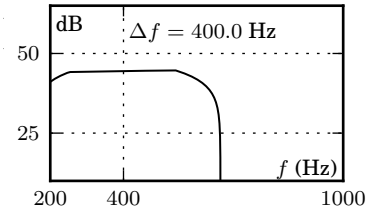

Figure 30:

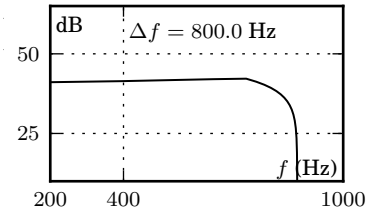

Figure 31: **Fletcher's Model – Band-Limited Noise Masking Curves.** The four maskers share a common SPL of $L = 50$ dB and center frequency of $f_c = 400$ Hz while their bandwidth increases from $\Delta f = 20.0$ Hz to 800 Hz.}

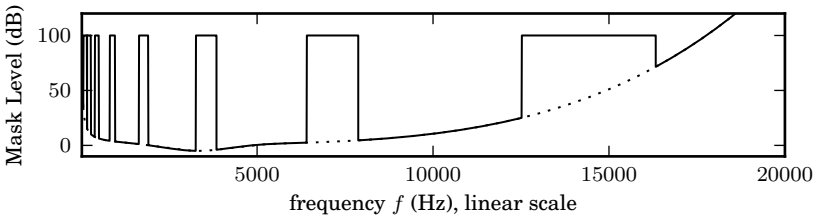

Figure 32:

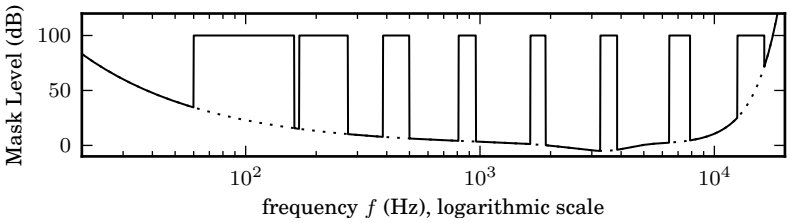

Figure 33:

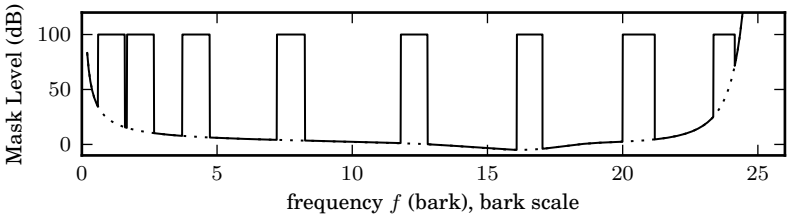

Figure 34: **Fletcher's Model – Pure Tones Masking Curves.** The masker is a combination of 8 pure tones of level 100 dB whose frequencies start with 110 Hz and double for each new tone. Frequencies are displayed in linear, logarithmic and bark scales.

Implementation - Bit Allocation Strategies

Consider a random signal X split into M subband signals X_k. Assume that in every subband an estimate $P_m(k)$ of the masking level intensity is available. Given a selection of quantizers $[\cdot]_k$, if we have

$$\forall\, k \in \{0, \cdots, M-1\},\ \mathbb{E}[(X_k - [X_k]_k)^2] \leq P_m(k)$$

then in every channel, the quantization noise is masked by the signal itself. These conditions () may be satisfied by a variable bitrate algorithm, but it is more likely that we have a total budget of bits to allocate per frame and that we are merely trying to spread the quantization noise above the masking levels among all subbands. We may achieve this by solving

$$\min \sum_{k=0}^{M-1} \frac{\mathbb{E}[(X_k - [X_k]_k)^2]}{P_m(k)}$$

or, under the high resolution assumption, if Δ_{f_k} denotes the quantizer step size of $[\cdot]_k$, as we have $\mathbb{E}[(X_k - [X_k]_k)^2] = (1/12)\mathbb{E}[\Delta_{f_k}(X_k)^2]$, by solving

$$\min \sum_{k=0}^{M-1} \frac{\mathbb{E}[\Delta_{f_k}(X_k)^2]}{P_m(k)}$$

Uniform Quantizers

Assume that every quantizer $[\cdot]_k$ is a uniform quantizer on $(-1, 1)$ with step size Δ_k. In every subband, the number of bits b_k is related to the step size by $\Delta_k = 2/2^{b_k}$. The availability of a constant number of bits per frame therefore leads to

$$\sum_{k=0}^{M-1} \log \Delta_k = \textbf{const.}$$

The constrained optimization problem () + () may be solved by lagrangian methods: we introduce $\Delta = (\Delta_0, \cdots, \Delta_{M-1})$,

$$L(\lambda, \Delta) = \sum_{k=0}^{M-1} \frac{\Delta_k^2}{P_m(k)} + \lambda \sum_{k=0}^{M-1} \log \Delta_k$$

and solve the equation $\nabla_\Delta L(\lambda, \Delta) = 0$. As for any k,

$$\frac{\partial L}{\partial \Delta_k} = 2\frac{\Delta_k}{P_m(k)} + \frac{\lambda}{\Delta_k},$$

the optimal set of step size satisfies

$$\Delta_k^2 \propto P_m(k)$$

the proportionaly constant being adjusted to match the bit budget.

Optimal Quantizers.

Instead of using uniform quantizers in every subbands, we may attempt to minimize every quantizer signal-to-noise ratio for a yet unknow number of bits, then consider the optimal allocation of bts. We assume that the characteristic function f_k used to implement $[\cdot]_k$ maps the real numbers into $[-1.0, +1.0]$ and hence that

$$\int_{-\infty}^{+\infty} f_k'(x)\,dx = 2$$

Note that if a uniform quantization applied on $[-1, 1]$ and has a budget of b_k bits,the corresponding constant step size is $\Delta_k = 2/2^{b_k}$. As we have $\Delta_{f_k}(x) = \Delta_k / f_k'(x)$, the equation () yields

$$\int_{-\infty}^{+\infty} \frac{1}{\Delta_{f_k}(x)}\,dx = \frac{2}{\Delta_k}$$

If the signal X_k has a density p_k, the step size Δ_{f_k} that is optimal with respect to the quantization signal-to-noise ratio satisfies $\Delta_{f_k}(x) \propto p_k^{-1/3}(x)$. Combined with (), this equation yields

$$\Delta_{f_k}(x) = \frac{\Delta_k}{2}\left[\int_{-\infty}^{+\infty} p_k^{1/3}(y)\,dy\right] p_k^{-1/3}(x)$$

and therefore

$$\mathbb{E}[\Delta_{f_k}(X_k)^2] = \frac{\Delta_k^2}{4}\left[\int_{-\infty}^{+\infty} p_k^{1/3}(x)\,dx\right]^3$$

Now, for common probability distributions p_k such as normal distributions or Laplace distributions, we have

$$\int_{-\infty}^{+\infty} p_k^{1/3}(x)\,dx \propto \mathbb{E}[X_k^2]^{1/3}$$

and hence

$$\mathbb{E}[\Delta_{f_k}(X_k)^2] \propto \Delta_k^2 \times \mathbb{E}[X_k^2]$$

The new minimization problem is therefore

$$\min \sum_{k=0}^{M-1} \Delta_k^2 \frac{\mathbb{E}[X_k^2]}{P_m(k)}$$

under

$$\sum_{k=0}^{M-1} \log \Delta_k = \textbf{const.}$$

If we introduce the signal-to-mask (SMR) ratio in the band k, defined by

$$\mathbf{SMR}_k^2 = \frac{\mathbb{E}[X_k^2]}{P_m(k)}$$

the solution to this optimization problem is given by

$$\Delta_k \propto \mathbf{SMR}_k^{-1}$$