PFB inversion

Ningyuan Li

June 2022

1 Forward PFB's matrix form

We recap first the basic procedures of an r-tap PFB. Say, we want to have n_c channels after the transformation. In a naive FFT procedure, we would need a time stream of length $2n_c$, after taking account of the Nyquist criterion[1]. However, in the PFB:

1. We first take a time stream that is of length $2rn_c$.
2. We then apply a sinc function of appropriate shape to the original time stream. The sinc is also of length $2rn_c$.
3. The sinc-modified time stream is then split into r pieces of the same length (say, M[2]). The r pieces are then stacked together (term-wise added).
4. We finally take the M-point real-DFT on the stacked signal, which gives us the final answer.

We wish to enumerate some of these steps with linear algebra notation, which makes the inversion procedure more straightforward. Using the prof's notation, we have that:

$$\boldsymbol{D} = FSW\boldsymbol{d}, \tag{1}$$

where $\boldsymbol{D}$ is the transformed signal, $\boldsymbol{d}$ is the original time stream, and:

F: is the DFT matrix, of size $M \times M$. It is unitary and hence invertible, which is convenient. The matrix is used in step 4. (Question 1: we are taking the real-DFT instead of the full DFT, which changes the matrix from an invertible one to a non-invertible one (since it would not be square). Is it not a big issue, because the real-DFT is simply the DFT minus some complex conjugates, so we can easily reconstruct the original, and the maths will be the same?)

S: is the matrix that takes care of the split-and-stacking. It does what step 3 instructs.

W: is the matrix that applies the window. It takes care of step 2.

[1] In practice, we take the `rfft` function, which disregards the complex conjugates – and hence having half the number of channels than the original time stream.

[2] Notice that $M = 2n_c$.

Writing everything out explicitly, we have that[3]:

$$\underbrace{\begin{bmatrix} \boldsymbol{D}_1 \\ \vdots \\ \boldsymbol{D}_{n_c} \end{bmatrix}}_{\boldsymbol{D}} = \underbrace{\begin{bmatrix} F_{1,1} & \cdots & F_{1,2n_c} \\ \vdots & \ddots & \vdots \\ F_{n_c,1} & \dots & F_{n_c,2n_c} \end{bmatrix}}_{F} \underbrace{\begin{bmatrix} \underbrace{\mathrm{id}_{2n_c} \cdots \mathrm{id}_{2n_c}}_{r} \end{bmatrix}}_{S} \underbrace{\begin{bmatrix} W_{1,1} & 0 & \cdots & 0 \\ 0 & W_{2,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & W_{2rn_c,2rn_c} \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} \boldsymbol{d}_1 \\ \vdots \\ \boldsymbol{d}_{2rn_c} \end{bmatrix}}_{\boldsymbol{d}}$$

It is also of future interest to write the exact form of SW[4]. Notice that, since S is a size $2n_c \times 2rn_c$ matrix, the product would not actually be square, and hence not invertible. Some techniques shall be explored in the future to undo its effect.

$$SW = \begin{bmatrix} W_1 & \cdots & 0 & W_{2n_c+1} & \cdots & 0 & \cdots & W_{2(r-1)n_c+1} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ 0 & \cdots & W_{2n_c} & 0 & \cdots & W_{4n_c} & \cdots & 0 & \cdots & W_{2rn_c} \end{bmatrix}, \tag{2}$$

as we can see, SW is in the form of r size $2n_c$ diagonal matrices horizontally aligned.

It may also be interesting to observe that each row of SW only acts upon samples in $\boldsymbol{d}$ whose indices are $2n_c$ apart. This will become important later on.

(Question 2: in the first paragraph of section 3 in your notes, I'm not quite sure what the procedure is that gives a Toeplitz matrix in the end. Is it the result of taking the IDFT (so multiplying F^{-1} again on SW)? Or is it something else? Moreover, is the 'decoupled chunk' that you are referring to the horizontal, diagonal blocks in SW?)

2 PFB of long time stream

Say, we want to compute the PFB of a very long time stream, with the parameters r and n_c predetermined. A normal FFT would return n_c channels from a time stream of length $2n_c$, and the PFB simply returns the same number of channels from $2rn_c$ samples. As such, in order to be able to return the same amount of processed data as ordinary FFT (Question 3: is this the reason why we shift by $2n_c$?), we also shift forward in time by $2n_c$ samples after every instance of PFB.

To illustrate what this means more concretely, say, we have the original (very long time stream) $\boldsymbol{d} = [\boldsymbol{d}_1 \;\dots\; \boldsymbol{d}_{2n_c} \; \boldsymbol{d}_{2n_c+1} \;\dots\; \boldsymbol{d}_{4n_c} \;\dots]^T$, and we wish to compute an r-tap[5] PFB that outputs n_c channels per block. If we stack the outputs in a matrix, we would obtain:

$$\mathcal{D}^T = \begin{bmatrix} \boldsymbol{D}_1 & \cdots & \boldsymbol{D}_{n_c} \\ \boldsymbol{D}_{n_c+1} & \cdots & \boldsymbol{D}_{2n_c} \\ & \cdots & \end{bmatrix},$$

[3] id_s stands for identity matrix of size s.

[4] In below discussion, elements $W_{i,j} \in W$ will be denoted more compactly as W_i, since $i = j \; \forall i, j$.

[5] Note that the tap number does not impact the sizes of inputs and outputs.

where the first row consists of the outputs from a PFB of $\boldsymbol{d}_1$ to $\boldsymbol{d}_{2rn_c}$, the second row consists of the outputs from the PFB of $\boldsymbol{d}_{2n_c}$ to $\boldsymbol{d}_{2(r+1)n_c}$, and so on.

More ambitiously, we can try to condense the operation into one single matrix equation, $\mathcal{D} = \mathcal{F}\mathcal{S}\mathcal{W}\boldsymbol{d}$. $\mathcal{F}$ is easily interpreted as the operator that applies the matrix F (cf. Eq. (1)) to each block of the output, which leaves $\mathcal{S}\mathcal{W}$ to be the more complicated part. (Question 4: the matrix equation does not actually provide the output $\mathcal{D}$ as a stacked matrix, since the dimensions of matrix multiplication dictates otherwise. I don't think it's a huge problem since the output of $\mathcal{S}\mathcal{W}d$ can easily be sliced-and-stacked to produce a stacked matrix; am I wrong to consider it this way?)

To figure out the form that the matrix takes, we notice that the matrix is necessarily some stacked form of Eq. (2), since we are still foundationally doing PFB's to each block. Moreover, each SW block in the bigger $\mathcal{S}\mathcal{W}$ matrix must be stacked vertically, in order to make sure that the output in $\mathcal{D}$ is stacked block-by-block. The other key characteristic of $\mathcal{S}\mathcal{W}$ is that each SW block is aligned vertically in a staggered fashion, which is for the purpose of shifting forward by $2n_c$ samples in d for instance of PFB computations.

With those in mind, we can deduce the form of $\mathcal{S}\mathcal{W}$, with SW blocks as components:

$$\mathcal{S}\mathcal{W} = \begin{bmatrix} SW & 0 & \cdots & \cdots & 0 \\ \underset{2n_c}{\longleftrightarrow} & SW & 0 & \ddots & 0 \\ \underset{2n_c}{\longleftrightarrow} & \underset{2n_c}{\longleftrightarrow} & SW & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}, \tag{3}$$

note that contrary to this rather poor portrayal, the SW block are not vertically disjoint from each other; rather, they are staggered, with each block moving $2n_c$ entries to the right.

Example 2.1. *We now try to illustrate our method with a concrete example. Say, we have the input $\boldsymbol{d}$ with $|\boldsymbol{d}| = 12$, with each individual sample denoted as the corresponding letter of the English alphabet. We would also want to impose a PFB scheme that is 3-tap, with $2n_c = 3$*[6]*. As a result, our matrix equation looks something like this:*

$$\begin{bmatrix} W_1 & & & W_4 & & & W_7 & & & & & \\ & W_2 & & & W_5 & & & W_8 & & & & \\ & & W_3 & & & W_6 & & & W_9 & & & \\ & & & W_1 & & & W_4 & & & W_7 & & \\ & & & & W_2 & & & W_5 & & & W_8 & \\ & & & & & W_3 & & & W_6 & & & W_9 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ \vdots \\ k \\ l \end{bmatrix},$$

with the empty spaces in the matrix filled with zeros. Now, if we compute the output, we would

[6] Yes, this is not ideal. But for the purpose of illustration it should suffice, as we are not dealing with the final FFT step here.

obtain the following vector:

$$\boldsymbol{v} = \begin{bmatrix} aW_1 + dW_4 + gW_7 \\ bW_2 + eW_5 + hW_8 \\ cW_3 + fW_6 + iW_9 \\ dW_1 + gW_4 + jW_7 \\ eW_2 + hW_5 + kW_8 \\ fW_3 + iW_6 + lW_9 \end{bmatrix};$$

easily, we could do some basic manipulation to rearrange $\boldsymbol{v}$ into the form that we want. Recall that we want each row to be independently ready for the ensuing DFT operation. As a result, we simply have to arrange every three entries in $\boldsymbol{v}$ in a row, and we have our desired result. To spell it out, we want:

$$\mathcal{SW}\boldsymbol{d} = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \boldsymbol{v}_3 \\ \boldsymbol{v}_4 & \boldsymbol{v}_5 & \boldsymbol{v}_6 \end{bmatrix},$$

where the notation $\boldsymbol{v}_k$ means the k-th entry of $\boldsymbol{v}$. We would get our end result by simply computing the DFT of each row. ■

Now, if we inspect the form of Eq. (3) in combination with Eq. (2), we can see that entries that are $2n_c$ indices apart in the original time stream $\boldsymbol{d}$ can actually be seen as equivalent classes that each forms one decoupled problem. In other words, within $\boldsymbol{d}$, there are $2n_c$ separate problems, each is its own matrix equation, with the matrix being certain entries from Eq. (3).

With that in mind, we shift our focus to those said matrix operations. The principle observation to make is that each decoupled problem out of the $2n_c$ total problems is operated by a Toeplitz matrix, with r diagonals for an r-tap PFB. To demonstrate it more concretely, consider Exp. 2.1 again. In that particular problem, we have three decoupled problems out of the 12 samples in $\boldsymbol{d}$: $\boldsymbol{d}_1 = [a\ d\ g\ j]^T$, $\boldsymbol{d}_2 = [b\ e\ h\ k]^T$, $\boldsymbol{d}_3 = [c\ f\ i\ l]^T$. We take $\boldsymbol{d}_1$ as an example; by extracting the entries from the preceding $\mathcal{SW}$ that operates on this four-long vector, we arrive at the following matrix equation:

$$\text{output} = T_1\boldsymbol{d}_1 = \begin{bmatrix} W_1 & W_4 & W_7 & 0 \\ 0 & W_1 & W_4 & W_7 \end{bmatrix} \begin{bmatrix} a \\ d \\ g \\ j \end{bmatrix}. \tag{4}$$

The matrix T_1 clearly takes care of all the operations that is responsible for the subproblem.

As we have foreshadowed, every single instance of $\boldsymbol{d}_k : k \in \{1, \ldots, 2n_c\}$ is operated by a corresponding Toeplitz matrix T_k. This situation is interesting, in that the multiplication of a Toeplitz matrix with a vector actually is another way of computing convolutions. We can write out an example just to convince ourselves:

Example 2.2. *We wish to show that a convolution between two vectors can be represented easily using a Toeplitz matrix. For simplicity, we stick to vectors of small cardinality for the example.*

Consider vector $\boldsymbol{a}$ of length three and vector $\boldsymbol{b}$ of length four. We wish to compute their convolution $\boldsymbol{a} * \boldsymbol{b} = \boldsymbol{c}$, clearly of length $3 + 4 - 1 = 6$. Using the definition of the convolution operation, we can compute each entry of $\boldsymbol{c}$ rather easily:

$$\boldsymbol{c} = \begin{bmatrix} a_1 b_1 \\ a_1 b_2 + b_1 a_2 \\ a_1 b_3 + b_2 a_2 + a_3 b_1 \\ a_1 b_4 + a_2 b_3 + a_3 b_2 \\ a_2 b_4 + a_3 b_2 \\ a_3 b_4 \end{bmatrix}.$$

We can also do the same with a matrix, however:

$$\boldsymbol{c} = T_b \boldsymbol{a} = \begin{bmatrix} b_1 & & \\ b_2 & b_1 & \\ b_3 & b_2 & b_1 \\ b_4 & b_3 & b_2 \\ & b_4 & b_3 \\ & & b_4 \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix},$$

where empty entries are zeros. As we can see, T_b is indeed a Toeplitz matrix. ■

The Toeplitz-ness will come in handy when we try to invert the problem.

3 Inverting the PFB

The big problem with convolutions is that they are not always invertible (like what's happening in Exp. 2.2). As evidenced by Eq. (4), after manipulating the input data with the PFB, the output vector is smaller in dimension compared to the input. This is a problem, since we could clearly not construct a bijection – and hence no inversion – between pre- and post-PFB data. Nevertheless, we could still try and do some manipulations that allows is to 'invert' the PFB, despite always imperfectly.

(From now on everything I do is a wild guess at attempting to figure out what you meant in the memo. I'm guessing on paper for the moment to work out some maths since I'm stuck currently.)