Why F-transform is more effective than a general convolution

Thi Minh Tam Pham[1], Irina Perfilieva[1], Olga Kosheleva[2], and Vladik Kreinovich[3]

[1] Institute for Research and Applications of Fuzzy Modeling
University of Ostrava, 30. dubna 22, 701 00 Ostrava 1,
Czech Republic, Thi.Pham@osu.cz, irina.perfilieva@osu.cz
[2] Department of Teacher Education, University of Texas at El Paso,
500 W. University, El Paso, Texas 79968, USA, olgak@utep.edu,
https://www.cs.utep.edu/vladik/olgavita.html
[3] Department of Computer Science, University of Texas at El Paso,
500 W. University, El Paso, Texas 79968, USA, vladik@utep.edu,
https://www.cs.utep.edu/vladik/

Abstract. In many applications, it is useful to use fuzzy-motivated techniques of F-transform, they often lead to better results that all previously known methods. However, this empirical success is somewhat puzzling. Indeed, from the purely mathematical viewpoint, F-transform is a particular case of convolution – so why is it more effective than a general convolution? In this paper, we explain this puzzle by showing that, in contrast to general convolution, F-transform requires much fewer computational steps. This is a big advantage in real-time situations when we need answers as soon as possible. Also, in situations when there is a finite time budget, this allow us to have more time for additional data processing – and thus, to get better results.

Keywords: F-transform, convolution, computation time

1 Formulation of the problem

F-transform is an empirically successful technique. Fuzzy logic (see, e.g., [1, 4–7, 11]) has many direct applications. It also has many indirect applications – i.e., applications of fuzzy-motivated techniques. One of such techniques is *F-transform*; see, e.g., [3, 8, 9, 12] and references therein.

Let us briefly explain, on a 1-D example, how it works. (At the end of this paper, we explain how our analysis can be generalized to the multi-dimensional case.) In the 1-D case, we start with a basic continuous non-negative function $A(t)$ that is different from 0 on an interval $[-h, h]$, for some $h > 0$, and reaches the value 1 for $t = 0$. We pick some starting point T_0. Let us denote, for all integers k, $T_k \stackrel{\text{def}}{=} T_0 + k \cdot h$. Then, when we have a continuous signal $x(t)$, we replace it with the values

$$F_k \stackrel{\text{def}}{=} \frac{\int x(t) \cdot A(T_k - t)\, dt}{\int A(t)\, dt}. \tag{1}$$

These values form what is called an *F-transform* of the original signal $x(t)$.

In most current applications – but not in all of them – the function $A(t)$ is selected in such a way that the sum of all shifted function is equal to 1:

$$\sum_k A(T_k - t) = 1.$$

This success is somewhat a puzzle. From the purely mathematical viewpoint, F-transform is a particular case of a general concept known as *convolution*. Namely, the expression (1) can be equivalently described as

$$F_k = \int x(t) \cdot a(T_k - t)\, dt, \tag{3}$$

where we denoted

$$a(t) \stackrel{\text{def}}{=} \frac{A(t)}{\int A(s)\, ds}. \tag{4}$$

In general, the expression

$$F(s) \stackrel{\text{def}}{=} \int x(t) \cdot a(s - t)\, dt, \tag{5}$$

is known as the *convolution* of the functions $x(t)$ and $a(t)$. In these terms, the values of F-transform are simply values $F_k = F(T_k)$ of the convolution function $F(s)$ at the points $s = T_k$.

And this is where the puzzle appears. Convolution is a well-known technique, with many efficient algorithms and many effective applications. So many researchers who apply F-transform are puzzled by the fact that in many applications, F-transform is more effective than the general convolution. (And some not-very-attentive anonymous reviewers of F-transform papers recommend rejection without looking at the successful results – based solely on the fact that since this method is a particular case of a convolution, there is nothing novel in this method.)

Comment. Of course, in practice, the signal $x(t)$ comes as a sequence of values $x_i \stackrel{\text{def}}{=} x(t_i)$, where usually $t_i = T_0 + i \cdot \Delta t$ for some small Δt. In this case, we use the corresponding integral sum to approximate the integrals (3) and (5):

$$F(t_i) = \sum_j x(t_j) \cdot a(t_i - t_j) \cdot \Delta t; \tag{6}$$

$$F_k = \sum_i x(t_i) \cdot a(T_k - t_i) \cdot \Delta t. \tag{7}$$

What we do in this paper. In this paper, we provide an answer to this puzzle: namely, we show that F-transform can be computed much faster than the general convolution – and this explains it effectiveness:

– in situations when we need the answer as soon as possible, computation speed is definitely a big advantage, and
– in situations when there is a limited time budget, this speed allow us to have more time for additional data processing – and thus, to get better results.

2 Analysis of the problem

How is F-transform different from the general convolution. To explain why F-transform is often empirically more efficient than the general convolution, let us first recall how is F-transform different from the general convolution. The main difference is as follows:

– Computing the convolution usually means computing the values $F(t)$ for all possible values t – i.e., in practice, for all the values t_i.
– On the other hand, F-transform only computes the values at points F_k – and these are much fewer such points, $h/\Delta t$ times fewer.

We will show that this difference actually leads to an explanation of F-transform's empirical effectiveness.

How is convolution usually computed? To explain why computing F-transform requires fewer computational steps that convolution, let us recall how convolution is usually computed.

One possible way to compute convolution is to directly implement the formula (6). Let us analyze how long it would take, in this case, to compute all the values of the convolution $F(t)$ – i.e., to compute all the values $F(t_i)$.

Let us denote the duration of the input signal by T. On the time interval of size T, we have $N \stackrel{\text{def}}{=} T/\Delta t$ values x_i. On this interval, we have $n = T/h$ values T_k. Each function $a(t_i)$ is different from 0 only on an interval of width $2h$. On this interval, there are $2h/\Delta$ different values t_i – i.e., taking into account the formulas for N and n, we conclude that there are $2N/n$ non-zero values $a(t_i)$ of the function $a(t)$.

To compute each value $F(t_i)$ by using the formula (6), we therefore need to perform $2N/n$ multiplications – and then, we need to add the resulting $2N/n$ terms. Thus, the overall number of arithmetic operations needed to compute each value $F(t_i)$ is equal to $2N/n + 2N/n = 4N/n$. Computing convolution means computing $F(t_i)$ for all N points t_i. So, the overall computation time needed to compute the convolution by directly implementing the formula (6) is equal to $N \cdot 4N/n = 4N^2/n$.

For large N, this number of steps is large. It is known that we can drastically speed up computations if we use Fast Fourier transform (FFT); see, e.g., [2]. This computation is based on the fact that the Fourier transform of a convolution is equal to the product of the Fouier transform of the convolved functions. Thus, to compute the convolution, we can do the following:

– First, we apply Fourier transform to the signal $x(t)$. This requires

$$1.5 \cdot N \cdot \log_2(N)$$

computational steps. We also need to compute the Fourier transform of the function $a(t)$, but this can be done before we start applying this F-transform to different signals, so we do not count this time as part of the algorithm's run-time.
- Then, we multiply both Fourier transforms element-wise. We have N elements, so we need N multiplications.
- Finally, we apply Inverse Fourier transform to the resulting product. This also requires $1.5 \cdot N \cdot \log_2(N)$ computational steps.

So, overall, this algorithm requires

$$1.5 \cdot N \cdot \log_2(N) + N + 1.5 \cdot N \cdot \log_2(N) = N \cdot (3 \cdot \log_2(N) + 1)$$

computational steps. For large N, this is much smaller that the direct computation whose time in, as we have shown, quadratic in N.

How fast can we compute F-transform? Let us now see how fast we can compute F-transform. Each moment t_i belongs to some interval $[T_k, T_{k+1}]$. According to the formula (7), the corresponding signal value $x(t_i)$ is only used in computing two values of the F-transform – the values F_k and F_{k+1}. In each of these computations, we multiply $x(t_i)$ by some pre-computed coefficient – $a(T_k - t_i) \cdot \Delta t$ or $a(T_{k+1} - t_i) \cdot \Delta t$ – and then add this product to the previous sum – we need 2 computational steps. Thus, for both computations, we use $2 \cdot 2 = 4$ computational steps that use the value $x(t_i)$. In general, there are N values $x(t_i)$, so overall, we need $4N$ computational steps.

Let us compare. For sufficiently large N, linear computation time $4N$ is definitely faster than quadratic time, and faster than FFT's $O(N \cdot \log_2(N))$.

We can find the threshold value N_0 starting with which F-transform is faster. This is the value for which the computation times are equal, i.e., for which $4N_0 = N_0 \cdot (3 \cdot \log_2(N_0) + 1)$. If we divide both sides by N_0 and then subtract 1 from both sides, we get $3 \cdot \log_2(N_0) = 3$, so $\log_2(N_0) = 1$ and $N_0 = 2$. So, already for $N \geq 3$, F-transform is faster.

How faster? For example, for $N = 1\,000$, we have $4N = 4\,000$ and

$$N \cdot (3 \cdot \log_2(N) + 1) \approx 1\,000 \cdot (3 \cdot 10 + 1) = 31\,000,$$

which is more almost 8 times faster.

OK, it is faster but why is it enough? We speed up computations by skipping some values $F(t)$, but maybe by doing this we lose some important information? A simple qualitative analysis shows that we do not lose anything. Indeed, one of the main reasons why convolution is used is that by averaging, it decreases the noise. The noise values are usually wildly oscillating – which means that they mostly consist of component with high frequency. When we apply convolution with a non-negative function $a(t)$ whose range is on the interval of size $2h$, we thus, in effect, filter out all the components of frequencies larger than or equal to $2\pi/2h = \pi/h$.

And it is known that a signal $F(t)$ that have no Fourier components of frequency $\geq \pi/h$ can be uniquely reconstructed based on its values $F(T_k)$ on a grid $T_k = T_0 + k \cdot h$ on a grid of step h; see, e.g., [10] and references therein. Thus, indeed, the F-transform values, in principle, enables us to uniquely reconstruct the whole function $F(t)$. So, by limiting ourselves to these values, we are not losing any information.

What about the multi-dimensional case? In the multi-dimensional case, FFT still requires $1.5 \cdot N \cdot \log_2(N)$ steps, when N is the overall number of values. So, similarly to the 1-dimensional case, we can conclude that the resulting number of steps in FFT-based computation of convolution is $N \cdot (3 \cdot \log_2(N) + 1)$.

However, in several dimensions, F-transform is more complicated. Usually, the functions for the multi-dimensional case are products of 1-dimensional functions, i.e., expressions of the type

$$A(T_{k_1} - t_1) \cdot A(T_{k_2} - t_2) \cdot \ldots \cdot A(T_{k_d} - t_d).$$

In this case, each point $t_i = (t_{i1}, \ldots, t_{id})$ is contained in one of the boxes $[T_{k_1}, T_{k_1+1}] \times \ldots \times [T_{k_d}, T_{k_d+1}]$ and, therefore, the value $x(t_i)$ is involved in 2^d computations of the values $F_{k_1+\varepsilon_1,\ldots,k_d+\varepsilon_d}$, where each ε_i is equal to 0 or 1. So, in this case, computation time of F-transform is equal to $2 \cdot 2^d \cdot N$.

So, in the d-dimensional case, the threshold N_0 at which F-transform is faster is determined by the equality

$$2 \cdot 2^d \cdot N_0 = N_0 \cdot (3 \cdot \log_2(N_0) + 1).$$

In this case, if we divide both sides by N_0 and subtract 1 from both sides, we get $2^{d+1} - 1 = 3 \cdot \log_2(N_0)$, so $\log_2(N_0) = (2^{d+1} - 1)/3$ and

$$N_0 = 2^{(2^{d+1}-1)/3}.$$

This number grows very fast with dimenion. For $d = 5$, we have $N_0 = 2^{21} \approx 2 \cdot 10^6$. For $d > 5$, the threshold is even higher. So, for such multi-D data, unless we have more than a million points, it is more effective to use FFT-based convolution techniques.

However, for most widely used cases of 2-D and 3-D data – e.g., for 2D and 3D images, F-transform is still more efficient:

- For $d = 2$, we have $N_0 = 2^{7/3} \approx 5.04$, so F-transform is better for $N \geq 6$.
- For $d = 3$, we have $N_0 = 2^5 = 32$, so F-tranform is better for $N \geq 33$.

And even for $d = 4$, we have $N_0 \approx 1290.2$, so F-transform is better for $N \geq 1291$. Thus, for $d = 4$, F-transform is better when we have more than a thousand signal values.

Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional

Practice in Computer Science), HRD-1834620 and HRD-2034030 (CAHSI Includes), EAR-2225395 (Center for Collective Impact in Earthquake Science C-CIES), and by the AT&T Fellowship in Information Technology.

It was also supported by a grant from the Hungarian National Research, Development and Innovation Office (NRDI), and by the Institute for Risk and Reliability, Leibniz Universitaet Hannover, Germany.

References

1. R. Belohlavek, J. W. Dauben, and G. J. Klir, *Fuzzy Logic and Mathematics: A Historical Perspective*, Oxford University Press, New York, 2017.
2. Th. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 2022.
3. F. Di Martino et al., "A color image reduction based on fuzzy transforms", *Information Sciences*, 2014, Vol. 266, pp. 101–111.
4. G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.
5. J. M. Mendel, *Explainable Uncertain Rule-Based Fuzzy Systems*, Springer, Cham, Switzerland, 2024.
6. H. T. Nguyen, C. L. Walker, and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2019.
7. V. Novák, I. Perfilieva, and J. Močkoř, *Mathematical Principles of Fuzzy Logic*, Kluwer, Boston, Dordrecht, 1999.
8. I. Perfilieva, "F-transform", In: *Springer Handbook of Computational Intelligence*, Springer, 2015, pp. 113–130.
9. T. M. T. Pham, J., Janeček, and I. Perfilieva, "Fuzzy transform on 1-D manifolds", In: *Biomedical and Other Applications of Soft Computing*, Springer, Cham, Switzerland, 2022, pp. 13–24.
10. N. Trefethen, "Unbounded growth of band-limited functions", *Notices of the American Mathematical Society*, 2025, Vol. 72, No. 6, pp. 666–669.
11. L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.
12. H. Zámečniková and I. Perfilieva, "F-transform on triangulated domain", In: M.-J. Lesot, M. Reformat, S. Vieira, J. P. Carvalho, F. Batista, B. Bouchon-Meunier, and R. R. Yager (eds.), *Information Processing and Management of Uncertainty in Knowledge-Based Systems, Abstracts of the 20th International Conference IPMU 2024*, Lisbon, Portugal, July 22–26, 2024.