Fast Convolution and Fast Fourier Transform under Interval and Fuzzy Uncertainty ⋆

Guoqing Liu

School of Sciences, Nanjing University of Technology, Nanjing, Jiangsu 210009, P.R. China

Vladik Kreinovich *

Department of Computer Science, University of Texas at El Paso, 500 W. University, El Paso, TX 79968, USA

Abstract

Convolution $y(t) = \int a(t-s) \cdot x(s)\, ds$ is one of the main techniques in digital signal processing. A straightforward computation of the convolution $y(t)$ requires $O(n^2)$ steps, where n is the number of observations $x(t_0), \ldots, x(t_{n-1})$. It is well known that by using the Fast Fourier Transform (FFT) algorithm, we can compute convolution much faster, with computation time $O(n \cdot \log(n))$.

In practice, we only know the signal $x(t)$ and the function $a(t)$ with uncertainty. Sometimes, we know them with interval uncertainty, i.e., we know intervals $[\underline{x}(t), \overline{x}(t)]$ and $[\underline{a}(t), \overline{a}(t)]$ that contain the actual (unknown) functions $x(t)$ and $a(t)$. In such situations, it is desirable, for every t, to compute the range $[\underline{y}(t), \overline{y}(t)]$ of possible values of $y(t)$. Of course, it is possible to use straightforward interval computations to compute this range, i.e., replace every computational step in FFT by the corresponding operations of interval arithmetic. However, the resulting en-

closure is too wide. In this paper, we show how to provide asymptotically accurate ranges for $y(t)$ in time $O(n \cdot \log(n))$.

We also explain how to use these new algorithms to compute the convolution (and the Fourier transform) under fuzzy uncertainty.

Key words: convolution, Fourier transform, interval uncertainty, fuzzy uncertainty

1 Formulation of the Problem

Convolution is important. The output $y(t)$ of an electronic device depends on its inputs $x(t)$. In many practical situations, the input signal $x(t)$ is reasonably small; as a result, we can safely ignore quadratic and higher order terms in the dependence of $y(t)$ on $x(t)$ and assume that the dependence is linear, i.e., that $y(t) = a_0(t) + \int a_1(t,s) \cdot x(s)\, ds$ for some coefficients $a_0(t)$ and $a_1(t,s)$.

In many interesting cases, the signal processing device does not change in time,

⋆ This work was done during Guoqing Liu's visit to the USA supported by the Jiangsu government scholarship. It was also supported in part by NSF grant HRD-0734825, by Grant 1 T36 GM078000-01 from the National Institutes of Health, and by the Japan Advanced Institute of Science and Technology (JAIST) International Joint Research Grant 2006-08. The authors are very thankful to Radim Belohlavek and Rudolf Kruse, the editors of the special issue, for their encouragement, to Andreas Griewank for his valuable advice, and to the anonymous referees for helpful suggestions.

* Corresponding author.
Email addresses: `guoqing@njut.edu.cn` (Guoqing Liu), `vladik@utep.edu` (Vladik Kreinovich).

in the sense that if we feed the same input again, we get the same output. In precise terms, this means that for every time shift t_0, if we input the signal $x(t+t_0)$, we should get $y(t+t_0)$, i.e., we should have

$$a_0(t) + \int a_1(t,s) \cdot x(s+t_0)\, ds = a_0(t+t_0) + \int a_1(t+t_0,s) \cdot x(s)\, ds$$

for all functions $x(t)$ and for all real numbers t and t_0. For $x(t) \equiv 0$, this requirement leads to $a_0(t) = a_0(t+t_0)$, i.e., to the conclusion that $a_0(t)$ is a constant. By subtracting this constant from $y(t)$, we get a simplified expression $y(t) = \int a_1(t,s) \cdot x(s)\, ds$ with

$$\int a_1(t,s) \cdot x(s+t_0)\, ds = \int a_1(t+t_0,s) \cdot x(s)\, ds.$$

By introducing a new variable $s+t_0$ in the first integral, we conclude that $\int a_1(t,s-t_0) \cdot x(s)\, ds = \int a_1(t+t_0,s) \cdot x(s)\, ds$ for all t and all $x(s)$. In particular, for a pulse signal, i.e., for function $x(t)$ which is non-zero only in a small neighborhood of a point s, we conclude that $a_1(t,s-t_0) = a_1(t+t_0,s)$ for all t, s, and t_0.

For every t' and s, we can take $t_0 = s$ and $t = t'-s$, then we get $s-t_0 = 0$ and $a_1(t',s) = a_1(t+t_0,s) = a_1(t'-s,0)$. Thus, for $a(t) \stackrel{\text{def}}{=} a_1(t,0)$, we conclude that

$$y(t) = \int a(t-s) \cdot x(s)\, ds.$$

In digital signal processing, this formula is called a *convolution*; see, e.g., [22].

Similarly, it is reasonable to consider optical filters for processing 2-D or 3-D images $x(\vec{t})$. If we require that the result of filtering does not depend on the exact spatial location of the image, then we can conclude that these filters can also described as convolutions: $y(\vec{t}) = \int a(\vec{t}-\vec{s}) \cdot x(\vec{s})\, d\vec{s}$ for some function $a(\vec{t})$.

Thus, if we want the computer to simulate the work of different electronic and/or optical devices such as filters, we must be able to compute one-dimensional and multi-dimensional convolutions.

It is important to compute convolutions fast. In computation, the input signal $x(t)$ is represented by its samples $x_0 = x(t_0), x_1 = x(t_1), \ldots, x_{n-1} = x(t_{n-1})$, usually measured at equally spaced moments of time $t_k = t_0 + k \cdot \Delta t$. Similarly, the input image $x(\vec{t})$ is usually represented by the intensity values at pixels forming a rectangular grid.

Based on this information, we can approximate the integral by the corresponding integral sum, and compute n values $y_i \stackrel{\text{def}}{=} y(t_i)$ as

$$y_i = \sum_{k=0}^{n-1} b_{i-k} \cdot x_k,$$

where $b_i \stackrel{\text{def}}{=} a(i \cdot \Delta t) \cdot \Delta t$. This discrete convolution operation that transforms two sequences $b = (b_0, \ldots, b_{n-1})$ and $x = (x_0, \ldots, x_{n-1})$ into a new sequence $y = (y_0, \ldots, y_{n-1})$ is usually denoted by $*$:

$$y = b * x.$$

If we use this formula to compute the outputs y_i, then we need n multiplications and $n - 1$ additions to compute each of n outputs, to the total of $O(n) \cdot O(n) = O(n^2)$ computational steps.

However, the number of samples n is usually in thousands and millions (an image is usually several Megabytes); for such large n, n^2 steps require too much time. For example, for $n \approx 10^6$, n^2 steps would require $\approx 10^{12}$ steps, i.e., about 15 minutes on a standard Gigahertz computer, i.e., on a computer which performs 10^9 computational steps per second.

Illustrative example. To illustrate the problem, let us run a simple example of the convolution. In this example, $n = 4$,

- the input signal has the form $x_0 = 1$, $x_1 = -1$, $x_2 = 1$, and $x_3 = 0$; and
- the weights b_i have the form $b_0 = 1$, $b_1 = -1$, and $b_2 = b_3 = 0$.

In this case,

$$y_0 = b_0 \cdot x_0 = 1 \cdot 1 = 1;$$

$$y_1 = b_1 \cdot x_0 + b_0 \cdot x_1 = (-1) \cdot 1 + 1 \cdot (-1) = -2;$$

$$y_2 = b_1 \cdot x_1 + b_0 \cdot x_2 = (-1) \cdot (-1) + 1 \cdot 1 = 2;$$

$$y_3 = b_1 \cdot x_2 = (-1) \cdot 1 = -1.$$

In this following text, we will use this example and its modifications to illustrate different formulas and algorithms.

Fast Fourier Transform leads to fast computation of convolution. It is well known that convolution can be computed much faster, in time

$$O(n \cdot \log(n)),$$

if we use the $O(n \cdot \log(n))$ Fast Fourier Transform (FFT) algorithm for computing Fourier transforms $\hat{x}(\omega) \stackrel{\text{def}}{=} \dfrac{1}{\sqrt{2\pi}} \cdot \displaystyle\int x(t) \cdot \exp(-\mathrm{i} \cdot \omega \cdot t)\, dt$; see, e.g., [1,22] (Please notice that in this paper, we use mathematical notation i for $\sqrt{-1}$; in signal processing, $\sqrt{-1}$ is usually denoted by j, to avoid confusion with the current i.)

Namely, it is known that the Fourier transform of the convolution is equal to the product of the Fourier transforms: $\hat{y}(\omega) = \hat{a}(\omega) \cdot \hat{x}(\omega)$. Thus, to compute the convolution $y(t)$, we do the following:

- first, we apply FFT to $x(t)$ and $a(t)$ and compute $\hat{x}(\omega)$ and $\hat{a}(\omega)$;

- then, we multiply $\hat{x}(\omega)$ and $\hat{a}(\omega)$, thus computing $\hat{y}(\omega) = \hat{a}(\omega) \cdot \hat{x}(\omega)$;
- finally, we apply the inverse FFT to $\hat{y}(\omega)$ and compute $y(t) = \dfrac{1}{\sqrt{2\pi}} \cdot \displaystyle\int \hat{y}(\omega) \cdot \exp(\mathrm{i} \cdot \omega \cdot t)\, d\omega$.

This algorithm can be visualized by the following diagram:

$$\begin{array}{ll} x \xrightarrow{\text{FFT}} \hat{x} & \searrow \\ & \hat{y} = \hat{a} \cdot \hat{x} \xrightarrow{\text{FFT}^{-1}} y \\ a \xrightarrow{\text{FFT}} \hat{a} & \nearrow \end{array}$$

To compute the convolution between the two given sequences, we use a discrete version of Fourier transform. Specifically, when we have two sequences x_i and b_i, we do the following:

- first, we apply discretized FFT to x_i and compute the corresponding Fourier coefficients as $X_j = \dfrac{1}{\sqrt{n}} \cdot \displaystyle\sum_{k=0}^{n-1} x_k \cdot \exp\left(-\mathrm{i} \cdot 2\pi \cdot \frac{j \cdot k}{n}\right)$; similarly, we compute the Fourier coefficients B_j of the sequence b_i;
- then, we multiply $\sqrt{n}$, B_j, and X_j, thus computing $Y_j = \sqrt{n} \cdot B_j \cdot X_j$;
- finally, we apply the inverse FFT to Y_j and compute

$$y_k = \frac{1}{\sqrt{n}} \cdot \sum_{k=0}^{n-1} Y_j \cdot \exp\left(\mathrm{i} \cdot 2\pi \cdot \frac{j \cdot k}{n}\right).$$

This algorithm can be visualized by the following diagram:

$$\begin{array}{ll} x_i \xrightarrow{\text{FFT}} X_j & \searrow \\ & Y_j = \sqrt{n} \cdot B_j \cdot X_j \xrightarrow{\text{FFT}^{-1}} y_i \\ b_i \xrightarrow{\text{FFT}} B_j & \nearrow \end{array}$$

Similarly, for every natural number d, convolution of d-dimensional functions $x(\vec{t})$ can be computed by using the discretized version of the multi-dimensional Fast Fourier Transform $\hat{x}(\vec{\omega}) \stackrel{\text{def}}{=} \dfrac{1}{(\sqrt{2\pi})^d} \cdot \displaystyle\int x(\vec{t}) \cdot \exp(-\mathrm{i} \cdot (\vec{\omega} \cdot \vec{t}))\, d\vec{t}$.

Example. Let us illustrate this algorithm on the above example. For $n = 4$,

we have $\sqrt{4} = 2$, and

$$\exp\left(-\mathrm{i} \cdot 2\pi \cdot \frac{1}{4}\right) = \exp\left(-\mathrm{i} \cdot \frac{\pi}{2}\right) = \cos\left(-\frac{\pi}{2}\right) + \mathrm{i} \cdot \sin\left(-\frac{\pi}{2}\right) = -\mathrm{i}.$$

The powers of this value are -1, i, 1, $-$i, etc.

Thus, the corresponding discrete Fourier transform takes the following form:

$$X_0 = \frac{1}{2} \cdot (x_0 + x_1 + x_2 + x_3);$$

$$X_1 = \frac{1}{2} \cdot (x_0 - \mathrm{i} \cdot x_1 - x_2 + \mathrm{i} \cdot x_3);$$

$$X_2 = \frac{1}{2} \cdot (x_0 - x_1 + x_2 - x_3);$$

$$X_3 = \frac{1}{2} \cdot (x_0 + \mathrm{i} \cdot x_1 - x_2 - \mathrm{i} \cdot x_3).$$

In particular, for the above input signal $x_0 = 1$, $x_1 = -1$, $x_2 = 1$, and $x_3 = 0$, we get

$$X_0 = \frac{1}{2} \cdot (1 + (-1) + 1 + 0) = \frac{1}{2};$$

$$X_1 = \frac{1}{2} \cdot (1 - \mathrm{i} \cdot (-1) - 1 + \mathrm{i} \cdot 0) = \frac{1}{2} \cdot \mathrm{i};$$

$$X_2 = \frac{1}{2} \cdot (1 - (-1) + 1 - 0) = \frac{3}{2};$$

$$X_3 = \frac{1}{2} \cdot (1 + \mathrm{i} \cdot (-1) - 1 - \mathrm{i} \cdot 0) = -\frac{1}{2} \cdot \mathrm{i}.$$

Comment. For simplicity, we do not describe, in detail, how the FFT algorithm computes the Fourier transform, we just present the results of the Fourier transform.

Similarly, for the weights $b_0 = 1$, $b_1 = -1$, $b_2 = b_3 = 0$, we have

$$B_0 = \frac{1}{2} \cdot (b_0 + b_1 + b_2 + b_3) = 0;$$

$$B_1 = \frac{1}{2} \cdot (b_0 - \mathrm{i} \cdot b_1 - b_2 + \mathrm{i} \cdot b_3) = \frac{1}{2} \cdot (1 - \mathrm{i});$$

$$B_2 = \frac{1}{2} \cdot (b_0 - b_1 + b_2 - b_3) = 1;$$
$$B_3 = \frac{1}{2} \cdot (b_0 + \mathrm{i} \cdot b_1 - b_2 - \mathrm{i} \cdot b_3) = \frac{1}{2} \cdot (1 + \mathrm{i}).$$

Based on the value B_j and X_j, we compute the value $Y_j = \sqrt{n} \cdot B_j \cdot X_j$:

$$Y_0 = 0; \quad Y_1 = \frac{1}{2} \cdot (-1 + \mathrm{i}); \quad Y_2 = 3; \quad Y_3 = \frac{1}{2} \cdot (-1 - \mathrm{i}).$$

By applying the inverse discrete Fourier transform

$$y_0 = \frac{1}{2} \cdot (Y_0 + Y_1 + Y_2 + Y_3);$$
$$y_1 = \frac{1}{2} \cdot (Y_0 + \mathrm{i} \cdot Y_1 - Y_2 - \mathrm{i} \cdot Y_3);$$
$$y_2 = \frac{1}{2} \cdot (Y_0 - Y_1 + Y_2 - Y_3);$$
$$y_3 = \frac{1}{2} \cdot (Y_0 - \mathrm{i} \cdot Y_1 - Y_2 + \mathrm{i} \cdot Y_3),$$

we get the desired values of y_i:

$$y_0 = \frac{1}{2} \cdot \left(0 + \frac{1}{2} \cdot (-1 + \mathrm{i}) + 3 + \frac{1}{2} \cdot (-1 - \mathrm{i})\right) = 1;$$
$$y_1 = \frac{1}{2} \cdot \left(0 + \mathrm{i} \cdot \frac{1}{2} \cdot (-1 + \mathrm{i}) - 3 - \mathrm{i} \cdot \frac{1}{2} \cdot (-1 - \mathrm{i})\right) = -2;$$
$$y_0 = \frac{1}{2} \cdot \left(0 - \frac{1}{2} \cdot (-1 + \mathrm{i}) + 3 - \frac{1}{2} \cdot (-1 - \mathrm{i})\right) = 2;$$
$$y_1 = \frac{1}{2} \cdot \left(0 - \mathrm{i} \cdot \frac{1}{2} \cdot (-1 + \mathrm{i}) - 3 + \mathrm{i} \cdot \frac{1}{2} \cdot (-1 - \mathrm{i})\right) = -1.$$

In practice, we often have interval uncertainty. In practice, we only know the signal $x(t)$ and the function $a(t)$ with uncertainty; in other words, we only know the approximate (measured) values $\widetilde{x}(t)$ and $\widetilde{a}(t)$ of these functions. In this case, the convolution $\widetilde{y}(t)$ of these approximate functions is also only an approximation to the desired convolution $y(t)$ of the (unknown) actual values $x(t)$ and $a(t)$.

Usually, we know the upper bounds $\Delta_x(t)$ and $\Delta_a(t)$ on the deviations $\Delta x(t) \stackrel{\text{def}}{=} x(t) - \widetilde{x}(t)$ and $\Delta a(t) \stackrel{\text{def}}{=} a(t) - \widetilde{a}(t)$. In some cases, we also know

the probability of different deviations, but often, these upper bounds is all we know. In such cases, we only know the intervals $[\widetilde{x}(t) - \Delta_x(t), \widetilde{x}(t) + \Delta_x(t)]$ and $[\widetilde{a}(t) - \Delta_a(t), \widetilde{a}(t) + \Delta_a(t)]$ that contain the actual (unknown) functions $x(t)$ and $a(t)$.

In terms of sample values, we know the intervals $\mathbf{x}_i = [\widetilde{x}_i - \Delta_{x,i}, \widetilde{x}_i + \Delta_{x,i}]$ and $\mathbf{b}_i = [\widetilde{b}_i - \Delta_{b,i}, \widetilde{b}_i + \Delta_{b,i}]$ that contain the actual (unknown) values of x_i and b_i.

In such situations, it is desirable, for every t, to compute the range of possible values of $y(t)$, i.e., to find the upper bounds $\Delta y(t)$ such that for every moment t, the actual value of the convolution $y(t)$ always lies within the interval

$$[\widetilde{y}(t) - \Delta_y(t), \widetilde{y}(t) + \Delta_y(t)].$$

In terms of sample values, we need to find intervals $\mathbf{y}_i = [\widetilde{y}_i - \Delta_{y,i}, \widetilde{y}_i + \Delta_{y,i}]$ that contain the actual (unknown) values y_i.

Case of expert uncertainty. In some practical situations, our information about the signal $x(t)$ and about the function $a(t)$ comes from expert estimations. Expert estimates are never absolutely accurate, they come with uncertainty. An expert usually describes his/her uncertainty by using words from the natural language, like "most probably, the value of the quantity is between 6 and 7, but it is somewhat possible to have values between 5 and 8". To formalize this knowledge, it is natural to use fuzzy set theory, a formalism specifically designed for describing this type of informal ("fuzzy") knowledge; see, e.g., [10,19].

As a result, for every value x_i, we have a fuzzy set $\mu_i(x_i)$ which describes the expert's prior knowledge about x_i: the number $\mu_i(x_i)$ describes the expert's

degree of certainty that x_i is a possible value of the i-th quantity.

An alternative user-friendly way to represent a fuzzy set is by using its α-cuts $\{x_i \,|\, \mu_i(x_i) > \alpha\}$ (or $\{x_i \,|\, \mu_i(x_i) \geq \alpha\}$). For example, the α-cut corresponding to $\alpha = 0$ is the set of all the values which are possible at all, the α-cut corresponding to $\alpha = 0.1$ is the set of all the values which are possible with degree of certainty at least 0.1, etc. In these terms, a fuzzy set can be viewed as a nested family of intervals $[\underline{x}_i(\alpha), \overline{x}_i(\alpha)]$ corresponding to different level α.

In general, we have fuzzy knowledge $\mu_i^x(x_i)$ and $\mu_j^b(b_j)$ about each value x_i and b_j; we want to find the fuzzy set corresponding to a each value $y_i = \sum\limits_{k=0}^{n-1} b_{i-k} \cdot x_k$. Intuitively, the value y_i is a reasonable value of the corresponding quantity if $y_i = \sum\limits_{k=0}^{n-1} b_{i-k} \cdot x_k$ for some reasonable values x_i and b_j, i.e., if for some values $x_1, \ldots, x_n$, $b_1, \ldots, b_n$, x_1 is reasonable, and x_2 is reasonable, $\ldots$, b_1 is reasonable, and b_2 is reasonable, $\ldots$, and $y_i = \sum\limits_{k=0}^{n-1} b_{i-k} \cdot x_k$. If we interpret "and" as min and "for some" ("or") as max, then we conclude that the corresponding degree of certainty $\mu_i(y_i)$ in y_i is equal to

$$\mu_i(y_i) = \max\left\{\min(\mu_1^x(x_1), \ldots, \mu_n^x(x_n), \mu_1^b(b_1), \ldots, \mu_n^b(b_n)) : y_i = \sum_{k=0}^{n-1} b_{i-k} \cdot x_k\right\}.$$

This formula is a particular case of the *extension principle*.

It is known that the extension principle can be reformulated as follows: for each α, the α-cut $\mathbf{y}_i(\alpha)$ of y_i is equal to the range of possible values of $y_i = \sum\limits_{k=0}^{n-1} b_{i-k} \cdot x_k$ when $x_i \in \mathbf{x}_i(\alpha)$ and $b_j \in \mathbf{b}_j(\alpha)$ for all i and j. Thus, from the computational viewpoint, the problem of computing the statistical characteristic under fuzzy uncertainty can be reduced to the problem of computing this characteristic under interval uncertainty; see, e.g., [2,10,17–19].

In view of this reduction, in the following text, we will consider the case of

interval uncertainty.

Interval computations, interval arithmetic, and straightforward interval computations: brief reminder. The problem of computing the range

$$[\underline{y}, \overline{y}] = f([\underline{x}_1, \overline{x}_1], \ldots, [\underline{x}_n, \overline{x}_n]) \stackrel{\text{def}}{=}$$

$$\{f(x_1, \ldots, x_n) : x_1 \in [\underline{x}_1, \overline{x}_1], \ldots, x_n \in [\underline{x}_n, \overline{x}_n]\}$$

of a given function $f(x_1, \ldots, x_n)$ under interval uncertainty $x_i \in [\underline{x}_i, \overline{x}_i]$ occurs in many practical situations; it is known as the problem of *interval computations*; see, e.g., [9].

In the simplest case when $n \le 2$ and $f(x_1, x_2)$ is an arithmetic operation (i.e., $f(x_1, x_2) = x_1 + x_2$, $f(x_1, x_2) = x_1 - x_2$, etc.), we can write down explicit expressions for the corresponding range; the operations for generating these expressions are called *interval arithmetic*. For $f(x_1, x_2) = x_1 + x_2$, the resulting interval

$$[\underline{y}, \overline{y}] = [\underline{a}, \overline{a}] + [\underline{b}, \overline{b}]$$

is equal to

$$[\underline{a}, \overline{a}] + [\underline{b}, \overline{b}] = [\underline{a} + \underline{b}, \overline{a} + \overline{b}].$$

Similarly, we have

$$[\underline{a}, \overline{a}] - [\underline{b}, \overline{b}] = [\underline{a} - \overline{b}, \overline{a} - \underline{b}];$$

$$[\underline{a}, \overline{a}] \cdot [\underline{b}, \overline{b}] = [\min(\underline{a} \cdot \underline{b}, \underline{a} \cdot \overline{b}, \overline{a} \cdot \underline{b}, \overline{a} \cdot \overline{b}), \max(\underline{a} \cdot \underline{b}, \underline{a} \cdot \overline{b}, \overline{a} \cdot \underline{b}, \overline{a} \cdot \overline{b})];$$

$$\frac{1}{[\underline{a}, \overline{a}]} = \left[\frac{1}{\overline{a}}, \frac{1}{\underline{a}}\right] \text{ if } 0 \notin [\underline{a}, \overline{a}];$$

and

$$\frac{[\underline{a}, \overline{a}]}{[\underline{b}, \overline{b}]} = [\underline{a}, \overline{a}] \cdot \frac{1}{[\underline{b}, \overline{b}]}.$$

In general, interval computation is NP-hard even for quadratic functions

$f(x_1, \ldots, x_n)$; see, e.g., [11]. Crudely speaking, this means that it is not possible to have an algorithm that always computes the exact range $\mathbf{y} = [\underline{y}, \overline{y}]$ in reasonable times (i.e., in time that does not exceed a polynomial of the size of the input). Since we cannot always compute the *exact* range in reasonable time, it is reasonable to try to compute an *enclosure* $\mathbf{Y} \supseteq \mathbf{y}$ for this range.

Historically the first method of computing such an enclosure is the method of *straightforward interval computations*. This method is based on the fact that inside the computer, the compiler presents every algorithm as a sequence of elementary operations (mostly arithmetic operations). In the straightforward interval computations technique, we replace each operation with numbers by the corresponding operation of interval arithmetic. It can be shown that the resulting interval $\mathbf{Y}$ indeed encloses the desired range $\mathbf{y}$; see, e.g., [9].

For some algorithms, this method leads to the exact range: e.g., for *single-use expressions* (SUE), i.e., arithmetic expressions in which each variable occurs only once; see, e.g., [5,9]. However, in general, the interval $\mathbf{Y}$ has excess width: $\mathbf{Y} \supset \mathbf{y}$ (and $\mathbf{Y} \neq \mathbf{y}$). For example, for a simple algorithm $y = 2x - x$ a routine compiler would first multiply x by 2, resulting in the intermediate result $r = 2x$, and then subtract x from r. For $\mathbf{x} = [0, 1]$, the resulting straightforward interval computations lead to $\mathbf{r} = 2 \cdot \mathbf{x} = [2, 2] \cdot [0, 1] = [0, 2]$ and then $\mathbf{Y} = \mathbf{r} - \mathbf{x} = [0, 2] - [0, 1] = [0 - 1, 2 - 0] = [-1, 2]$, while the actual range $\mathbf{y}$ of the expression $y = 2x - x = x$ is, of course, the original interval $[0, 1]$.

The reason why we got excess width is that straightforward interval computations ignore the dependence between intermediate results: in this case, the dependence between $r = 2x$ and x.

Comment. People who are vaguely familiar with interval computations sometimes erroneously assume that the above straightforward techniques is all there is in interval computations. In conference presentations (and even in published papers), one often encounters a statement: "I tried interval computations, and it did not work". What this statement usually means is that they tried the above straightforward approach and – not surprisingly – it did not work well.

In reality, interval computations is *not a single algorithm*, it is a *problem* for which many different techniques exist; see, e.g., [9]. Some of these techniques will be presented in the following text.

It is possible to compute convolution under interval uncertainty in time $O(n^2)$. In terms of sample values, computing convolution means computing the values $y_i = \sum\limits_{k=0}^{n-1} b_{i-k} \cdot x_k$. For each i, the expression for y_i is a single-use expression (SUE). Thus (see, e.g., [16]), we can compute the exact range $\mathbf{y}_i$ for y_i by replacing each arithmetic operation in this expression with the corresponding operation of interval arithmetic:

$$\mathbf{y}_i = \sum_{k=0}^{n-1} \mathbf{b}_{i-k} \cdot \mathbf{x}_k \qquad (1).$$

The problem with this computation is that it requires $O(n)$ computational steps for each of n values i, to the total of $O(n^2)$ steps – and we already know that this is too long. It is desirable to design faster algorithms for computing convolution under interval uncertainty.

Example. Let us supplement our illustrative example with interval uncertainty. Specifically, let us assume that the approximate values of the input signal and of the weights are the same as before: $\widetilde{x}_0 = 1$, $\widetilde{x}_1 = -1$, $\widetilde{x}_2 = 1$,

and $\widetilde{x}_3 = 0$, $\widetilde{b}_1 = 1$, $\widetilde{b}_2 = -1$, and $\widetilde{b}_2 = \widetilde{b}_3 = 0$. Let us also assume that the uncertainty with which we know (non-zero components of) the signal x_i does not exceed $\Delta_{x,i} = 0.2$, and the the uncertainty with which we know (non-zero) weights b_i does not exceed $\Delta_{b,i} = 0.1$. Under this assumption, the only information that we have about the actual (unknown) values x_i and b_i is that they belong to the corresponding interval:

$$\mathbf{x}_0 = [\widetilde{x}_0 - \Delta_{x,0}, \widetilde{x}_0 + \Delta_{x,0}] = [0.8, 1.2]$$

and similarly, $\mathbf{x}_1 = [-1.2, -0.8]$, $\mathbf{x}_2 = [0.8, 1.2]$, $\mathbf{x}_3 = [0, 0]$, $\mathbf{b}_0 = [0.9, 1.1]$, $\mathbf{b}_1 = [-1.1, -0.9]$, and $\mathbf{b}_2 = \mathbf{b}_3 = [0, 0]$. In this case, the corresponding intervals $\mathbf{y}_i$ of possible values of the output signal y_i take the form

$$\mathbf{y}_0 = \mathbf{b}_0 \cdot \mathbf{x}_0 = [0.9, 1.1] \cdot [0.8, 1.2] = [0.72, 1.32];$$

$$\mathbf{y}_1 = \mathbf{b}_1 \cdot \mathbf{x}_0 + \mathbf{b}_0 \cdot \mathbf{x}_1 = [-1.1, -0.9] \cdot [0.8, 1.2] + [0.9, 1.1] \cdot [-1.2, -0.8] = [-2.64, -1.44];$$

$$\mathbf{y}_2 = \mathbf{b}_1 \cdot \mathbf{x}_1 + \mathbf{b}_0 \cdot \mathbf{x}_2 = [-1.1, -0.9] \cdot [-1.2, -0.8] + [0.9, 1.1] \cdot [0.8, 1.2] = [1.44, 2.64];$$

$$\mathbf{y}_3 = \mathbf{b}_1 \cdot \mathbf{x}_2 = [-1.1, -0.9] \cdot [0.8, 1.2] = [-1.32, -0.72].$$

Straightforward interval version of FFT leads to too wide enclosures. In principle, it is possible to use straightforward interval computations to compute this range, i.e., replace every computational step in FFT by the corresponding operations of interval arithmetic. However, as noticed already in the pioneering paper [3], the resulting enclosure is too wide.

An even more simplified example. Let us first illustrate the phenomenon of wide enclosures on an even more simplified example of $n = 2$ data points

x_0 and x_1 known with interval uncertainty: $\mathbf{x}_0 = \mathbf{x}_1 = [-\Delta, \Delta]$. Let us assume that the values b_i are exactly known: $b_0 = 1$ and $b_1 = 0$. In this case, $y_i = \sum\limits_{k=1}^{n-1} b_{i-k} \cdot x_k = x_i$.

The FFT algorithm requires that we compute the FFT B_j of the sequence b_i, the FFT X_j of the signal x_i, then compute $Y_j = \sqrt{n} \cdot B_j \cdot X_j$ and apply the inverse FFT to the values Y_j.

For $n = 2$, discrete FFT takes the form $X_0 = \dfrac{x_0 + x_1}{\sqrt{2}}$ and $X_1 = \dfrac{x_0 - x_1}{\sqrt{2}}$, and the inverse FFT takes a similar form $x_0 = \dfrac{X_0 + X_1}{\sqrt{2}}$ and $x_1 = \dfrac{X_0 - X_1}{\sqrt{2}}$. In particular, for the above simple sequence b_i, the FFT leads to $B_0 = \dfrac{b_0 + b_1}{\sqrt{2}} = \dfrac{1}{\sqrt{2}}$ and $B_1 = \dfrac{b_0 - b_1}{\sqrt{2}} = \dfrac{1}{\sqrt{2}}$.

When we know the exact values x_0 and x_1, we compute $X_0 = \dfrac{x_0 + x_1}{\sqrt{2}}$ and $X_1 = \dfrac{x_0 - x_1}{\sqrt{2}}$, and then $Y_j = \sqrt{2} \cdot B_j \cdot X_j = \sqrt{2} \cdot \dfrac{1}{\sqrt{2}} \cdot X_j = X_j$. So, the inverse Fourier transform returns

$$y_0 = \frac{Y_0 + Y_1}{\sqrt{2}} = \frac{X_0 + X_1}{\sqrt{2}} = \frac{\dfrac{x_0 + x_1}{\sqrt{2}} + \dfrac{x_0 - x_1}{\sqrt{2}}}{\sqrt{2}} = x_0;$$

$$y_1 = \frac{Y_0 - Y_1}{\sqrt{2}} = \frac{X_0 - X_1}{\sqrt{2}} = \frac{\dfrac{x_0 + x_1}{\sqrt{2}} - \dfrac{x_0 - x_1}{\sqrt{2}}}{\sqrt{2}} = x_1.$$

However, for intervals, we get excess width. Indeed, for interval data, we thus get

$$\mathbf{X}_0 = \frac{\mathbf{x}_0 + \mathbf{x}_1}{\sqrt{2}} = \frac{[-\Delta, \Delta] + [-\Delta, \Delta]}{\sqrt{2}} = [-\sqrt{2} \cdot \Delta, \sqrt{2} \cdot \Delta]$$

and $\mathbf{X}_1 = \dfrac{\mathbf{x}_0 - \mathbf{x}_1}{\sqrt{2}} = [-\sqrt{2} \cdot \Delta, \sqrt{2} \cdot \Delta]$, after which we get $\mathbf{x}_0 = \dfrac{\mathbf{X}_0 + \mathbf{X}_1}{\sqrt{2}} = [-2 \cdot \Delta, 2 \cdot \Delta]$ and similarly, $\mathbf{x}_1 = [-2 \cdot \Delta, 2 \cdot \Delta]$.

As a result, we get intervals which are twice wider than the actual range. The reason for this excess width, as we have mentioned earlier, is that straightfor-

ward interval computations ignore the dependence between the intermediate results: in this case, the dependence between X_0 and X_1.

Illustrative example. On our illustrative example, the width is increased even more. Indeed, for the intervals $\mathbf{x}_i$, the discrete Fourier transform leads to

$$\mathbf{X}_0 = \frac{1}{2} \cdot (\mathbf{x}_0 + \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3) = [0.2, 0.8];$$

$$\mathbf{X}_1 = \frac{1}{2} \cdot (\mathbf{x}_0 - \mathrm{i} \cdot \mathbf{x}_1 - \mathbf{x}_2 + \mathrm{i} \cdot \mathbf{x}_3) = [-0.2, 0.2] + \mathrm{i} \cdot [0.4, 0.6];$$

$$\mathbf{X}_2 = \frac{1}{2} \cdot (\mathbf{x}_0 - \mathbf{x}_1 + \mathbf{x}_2 - \mathbf{x}_3) = [1.2, 1.8];$$

$$\mathbf{X}_3 = \frac{1}{2} \cdot (\mathbf{x}_0 + \mathrm{i} \cdot \mathbf{x}_1 - \mathbf{x}_2 - \mathrm{i} \cdot \mathbf{x}_3) = [-0.2, 0.2] + \mathrm{i} \cdot [-0.6, -0.4].$$

Similarly, for the weights, we get

$$\mathbf{B}_0 = \frac{1}{2} \cdot (\mathbf{b}_0 + \mathbf{b}_1 + \mathbf{b}_2 + \mathbf{b}_3) = [-0.1, 0.1];$$

$$\mathbf{B}_1 = \frac{1}{2} \cdot (\mathbf{b}_0 - \mathrm{i} \cdot \mathbf{b}_1 - \mathbf{b}_2 + \mathrm{i} \cdot \mathbf{b}_3) = [0.45, 0.55] + \mathrm{i} \cdot [0.45, 0.55];$$

$$\mathbf{B}_2 = \frac{1}{2} \cdot (\mathbf{b}_0 - \mathbf{b}_1 + \mathbf{b}_2 - \mathbf{b}_3) = [0.8, 1.2];$$

$$\mathbf{B}_3 = \frac{1}{2} \cdot (\mathbf{b}_0 + \mathrm{i} \cdot \mathbf{b}_1 - \mathbf{b}_2 - \mathrm{i} \cdot \mathbf{b}_3) = [0.45, 0.55] + \mathrm{i} \cdot [-0.55, 0.45].$$

Here, the intervals $\mathbf{Y}_j = 2 \cdot \mathbf{B}_j \cdot \mathbf{X}_j$ take the following form: $\mathbf{Y}_0 = [-0.16, 0.16]$,

$$\mathbf{Y}_1 = ([-0.2, 0.2] + \mathrm{i} \cdot [0.4, 0.6]) \cdot ([0.9, 1.1] + \mathrm{i} \cdot [0.9, 1.1]) =$$

$$([-0.2, 0.2] \cdot [0.9, 1.1] - [0.4, 0.6] \cdot [0.9, 1.1]) +$$

$$\mathrm{i} \cdot ([-0.2, 0.2] \cdot [0.9, 1.1] + [0.4, 0.6] \cdot [0.9, 1.1]) =$$

$$([-0.22, 0.22] - [0.36, 0.66]) + \mathrm{i} \cdot ([-0.22, 0.22] + [0.36, 0.66]) =$$

$$[-0.88, -0.14] + \mathrm{i} \cdot [0.14, 0.88];$$

and similarly, $\mathbf{Y}_2 = [1.92, 3.84]$, and $\mathbf{Y}_3 = [-0.88, -0.14] + \mathrm{i} \cdot [-0.88, -0.14]$. For y_0, the inverse Fourier transform leads to the following interval

$$\mathbf{y}_0 = \frac{1}{2} \cdot (\mathbf{Y}_0 + \mathbf{Y}_1 + \mathbf{Y}_2 + \mathbf{Y}_3) = [0.0, 1.36] + \mathrm{i} \cdot [-0.37, 0.37].$$

Already the real part of this interval is much wider than the actual range $[0.72, 1.32]$ of the value y_0: actually, more than twice wider. For other components y_i, we also get enclosures which are too wide.

What we do in this paper. Examples like the one above lead to an impression that doing convolution via FFT is impossible without really inflating the intervals.

In this paper, we show, that, contrary to this impression, it is possible to compute convolution via FFT (i.e., fast) without generating significant interval inflation.

2 Main Result: Fast Convolution under Interval Uncertainty

We will use Rump's circular arithmetic. We were able to come up with a fast algorithm only when we decided to use, instead of the standard interval arithmetic, Rump's circular arithmetic (see, e.g., [20]). This arithmetic provides exact range for addition and subtraction and asymptotically precise range for multiplication.

In Rump's circular arithmetic,

$$[\widetilde{a} - \Delta_a, \widetilde{a} + \Delta_a] + [\widetilde{b} - \Delta_b, \widetilde{b} + \Delta_b] = [(\widetilde{a} + \widetilde{b}) - (\Delta_a + \Delta_b), (\widetilde{a} + \widetilde{b}) + (\Delta_a + \Delta_b)]$$

and

$$[\widetilde{a} - \Delta_a, \widetilde{a} + \Delta_a] \cdot [\widetilde{b} - \Delta_b, \widetilde{b} + \Delta_b] = [\widetilde{a} \cdot \widetilde{b} - \Delta, \widetilde{a} \cdot \widetilde{b} + \Delta],$$

where

$$\Delta = |a| \cdot \Delta_b + \Delta_a \cdot |b| + \Delta_a \cdot \Delta_b.$$

Comments.

- In these two operations, the midpoint of each resulting interval is equal to the result of applying the same arithmetic operation to the corresponding midpoints. It is known that the same property holds for any operation (or any sequence of operations) of Rump's circular arithmetic.
- In the above formulas, we did not take rounding errors into account because in the signal processing applications, rounding errors are usually negligible in comparison with the measurement errors. If necessary, rounding errors can be taken into account by introducing appropriate roundings [20] – as it is usually done in interval computations (see, e.g., [9]).

Analysis of the problem. Applying the formulas for Rump's circular arithmetic to the expression (1), we conclude that $\mathbf{y}_i = [\widetilde{y}_i - \Delta_{y,i}, \widetilde{y}_i + \Delta_{y,i}]$, where

$$\Delta_{yi} = \sum_{k=0}^{n-1} |\widetilde{b}_{i-k}| \cdot \Delta_{x,k} + \sum_{k=0}^{n-1} \Delta_{b,i-k} \cdot |\widetilde{x}_k| + \sum_{k=0}^{n-1} \Delta_{b,i-k} \cdot \Delta_{x,k}.$$

Thus, once we have the sequences $|\widetilde{b}| = (|\widetilde{b}_0|, \ldots, |\widetilde{b}_{n-1}|)$, $|\widetilde{x}| = (|\widetilde{x}_0|, \ldots, |\widetilde{x}_{n-1}|)$ $\Delta_b = (\Delta_{b,0}, \ldots, \Delta_{b,n-1})$, and $\Delta_x = (\Delta_{x,0}, \ldots, \Delta_{x,n-1})$, we can compute the desired sequence $\Delta_y = (\Delta_{y,0}, \ldots, \Delta_{y,n-1})$ as the sum of three convolutions:

$$\Delta_y = |\widetilde{b}| * \Delta_x + \Delta_b * |\widetilde{x}| + \Delta_b * \Delta_x. \tag{2}$$

Since by using FFT, we can compute each of these three convolutions fast (in time $O(n \cdot \log(n))$), we thus arrive at the following fast algorithm for computing convolution under interval uncertainty.

Fast algorithm for computing convolution under interval uncer-

tainty. Suppose that we are given the intervals $[\widetilde{b}_i - \Delta_{b,i}, \widetilde{b}_i + \Delta_{b,i}]$ and $[\widetilde{x}_i - \Delta_{x,i}, \widetilde{x}_i + \Delta_{x,i}]$. Then, to compute the (asymptotically exact enclosure) $[\widetilde{y}_i - \Delta_{y,i}, \widetilde{y}_i + \Delta_{y,i}]$, we do the following:

- first, we use FFT-based convolution algorithm to compute $\widetilde{y} = \widetilde{b} * \widetilde{x}$;
- then, we use FFT-based convolution algorithm to compute three auxiliary convolutions $|\widetilde{b}| * \Delta_x$, $\Delta_b * |\widetilde{x}|$, and $\Delta_b * \Delta_x$;
- finally, we add the resulting three sequences and compute Δ_y by using the formula (2).

Since each FFT-based convolution requires $O(n \cdot \log(n))$ steps, we thus arrive at the $O(n \cdot \log(n))$ algorithm for computing convolution under interval uncertainty.

This algorithm can be visualized by the following diagram:

$$\begin{array}{ll} \widetilde{x}, \Delta_x \longrightarrow |\widetilde{x}| \searrow & \\ & \Delta_y = |\widetilde{b}| * \Delta_x + \Delta_b * |\widetilde{x}| + \Delta_b * \Delta_x \\ \widetilde{b}, \Delta_b \longrightarrow |\widetilde{b}| \nearrow & \end{array}$$

Comment. The same algorithm works in multi-dimensional case as well.

Example. Let us illustrate the accuracy of this algorithm on our example. Here, the values $\widetilde{y} = \widetilde{b} * \widetilde{x}$ have already been computed earlier: $\widetilde{y}_0 = 1$, $\widetilde{y}_1 = -2$, $\widetilde{y}_2 = 2$, and $\widetilde{y}_3 = -1$.

Our goal is to check how accurate are the resulting computations. For computing with numbers (not with intervals), FFT leads to exact convolution, so, for simplicity, we will use the regular formula to compute the corresponding auxiliary convolutions $|\widetilde{b}| * \Delta_x$, $\Delta_b * |\widetilde{x}|$, and $\Delta_b * \Delta_x$. Of course, the FFT-based convolution algorithm will lead to the same results.

We have $|\widetilde{b}_0| = |\widetilde{b}_1| = 1$, $|\widetilde{b}_2| = |\widetilde{b}_3| = 0$, $\Delta_{x,0} = \Delta_{x,1} = \Delta_{x,2} = 0.2$, and $\Delta_{x,3} = 0$, thus, for the first auxiliary sequence $f = |\widetilde{b}| * \Delta_x$, we get

$$f_0 = |\widetilde{b}_0| \cdot \Delta_{x,0} = 1 \cdot 0.2 = 0.2;$$

$$f_1 = |\widetilde{b}_1| \cdot \Delta_{x,0} + |\widetilde{b}_0| \cdot \Delta_{x,1} = 1 \cdot 0.2 + 1 \cdot 0.2 = 0.4;$$

$$f_2 = |\widetilde{b}_1| \cdot \Delta_{x,1} + |\widetilde{b}_0| \cdot \Delta_{x,2} = 1 \cdot 0.2 + 1 \cdot 0.2 = 0.4;$$

$$f_3 = |\widetilde{b}_0| \cdot \Delta_{x,2} = 1 \cdot 0.2 = 0.2.$$

Similarly, we have $\Delta_{b,0} = \Delta_{b,1} = 0.1$, $\Delta_{b,2} = \Delta_{b,3} = 0$, $|\widetilde{x}_0| = |\widetilde{x}_1| = |\widetilde{x}_2| = 1$, and $|\widetilde{x}_3| = 0$, thus, for the second auxiliary sequence $s = \Delta_b * |\widetilde{x}|$, we get

$$s_0 = \Delta_{b,0} \cdot |\widetilde{x}_0| = 0.1 \cdot 1 = 0.1;$$

$$s_1 = \Delta_{b,1} \cdot |\widetilde{x}_0| + \Delta_{b,0} \cdot |\widetilde{x}_1| = 0.1 \cdot 1 + 0.1 \cdot 1 = 0.2;$$

$$s_2 = \Delta_{b,1} \cdot |\widetilde{x}_1| + \Delta_{b,0} \cdot |\widetilde{x}_2| = 0.1 \cdot 1 + 0.1 \cdot 1 = 0.2;$$

$$s_3 = \Delta_{b,1} \cdot |\widetilde{x}_2| = 0.1 \cdot 1 = 0.1.$$

Finally, for the third auxiliary sequence $t = \Delta_b * \Delta_x$, we get

$$t_0 = \Delta_{b,0} \cdot \Delta_{x,0} = 0.1 \cdot 0.2 = 0.02;$$

$$t_1 = \Delta_{b,1} \cdot \Delta_{x,0} + \Delta_{b,0} \cdot \Delta_{x,1} = 0.1 \cdot 0.2 + 0.1 \cdot 0.2 = 0.04;$$

$$t_2 = \Delta_{b,1} \cdot \Delta_{x,1} + \Delta_{b,0} \cdot \Delta_{x,2} = 0.1 \cdot 0.2 + 0.1 \cdot 0.2 = 0.04;$$

$$t_2 = \Delta_{b,0} \cdot \Delta_{x,2} = 0.1 \cdot 0.2 = 0.02.$$

Adding these three auxiliary sequences, we get the sequence Δ_y, with components $\Delta_{y,0} = 0.2 + 0.1 + 0.02 = 0.32$, $\Delta_{y,1} = \Delta_{y,2} = 0.4 + 0.2 + 0.04 = 0.64$, and $\Delta_{y,2} = 0.2 + 0.1 + 0.02 = 0.32$. Thus, for y_i, we get the following ranges:

$$[\widetilde{y}_0 - \Delta_{y,0}, \widetilde{y}_0 + \Delta_{y,0}] = [1.0 - 0.32, 1.0 + 0.32] = [0.68, 1.32].$$

This range is slightly wider than the actual range $[0.72, 1.32]$ that we computed earlier, but the difference $0.72 - 0.68 = 0.04 = 0.2^2$ is, as expected, of second (quadratic) order in terms of the approximation errors $\Delta_{x,i} = 0.2$.

Similarly, for y_1, we get the range

$$[\widetilde{y}_1 - \Delta_{y,1}, \widetilde{y}_1 + \Delta_{y,1}] = [-2.0 - 0.64, -2.0 + 0.64] = [-2.64, -1.36],$$

which is slightly wider than the actual interval $[-2.64, -1.44]$. For y_2, we get the range

$$[\widetilde{y}_2 - \Delta_{y,2}, \widetilde{y}_2 + \Delta_{y,2}] = [2.0 - 0.64, 2.0 + 0.64] = [1.36, 2.64],$$

which is slightly wider than the actual interval $[1.44, 2.64]$. Finally, for y_3, we get the range

$$[\widetilde{y}_3 - \Delta_{y,3}, \widetilde{y}_3 + \Delta_{y,3}] = [-1.0 - 0.32, -1.0 + 0.32] = [-1.32, -0.68],$$

which is slightly wider than the actual interval $[-1.32, -0.72]$.

This algorithm can be made even faster. In the above algorithm, we need four convolutions of sequences of real numbers to compute a convolution of two interval-valued sequences. It turns out that we can further speed up this computation because it is possible to use only three convolutions instead of four.

Namely, since

$$(|\widetilde{b}| + \Delta_b) * (|\widetilde{x}| + \Delta_x) = |\widetilde{b}| * |\widetilde{x}| + |\widetilde{b}| * \Delta_x + \Delta_b * |\widetilde{x}| + \Delta_b * \Delta_x,$$

we can compute Δ_y as

$$\Delta_y = (|\widetilde{b}| + \Delta_b) * (|\widetilde{x}| + \Delta_x) - |\widetilde{b}| * |\widetilde{x}|. \tag{3}$$

Thus, we arrive at the following algorithm for computing convolution under interval uncertainty:

- first, we use FFT-based convolution algorithm to compute $\widetilde{y} = \widetilde{b} * \widetilde{x}$;
- then, we use FFT-based convolution algorithm to compute two auxiliary convolutions $(|\widetilde{b}| + \Delta_b) * (|\widetilde{x}| + \Delta_x)$ and $|\widetilde{b}| * |\widetilde{x}|$;
- finally, we subtract the resulting sequences and compute Δ_y by using the formula (3).

This algorithm can be visualized by the following diagram:

$$\begin{array}{lll} \widetilde{x}, \Delta_x \longrightarrow |\widetilde{x}| \longrightarrow |\widetilde{x}| + \Delta_x & \searrow & \\ & & \Delta_y = (|\widetilde{b}| + \Delta_b) * (|\widetilde{x}| + \Delta_x) - |\widetilde{b}| * |\widetilde{x}| \\ \widetilde{b}, \Delta_b \longrightarrow |\widetilde{b}| \longrightarrow |\widetilde{b}| + \Delta_b & \nearrow & \end{array}$$

Comment. The possibility to reduce the number of underlying numerical operations from four to three is similar to the situation with standard interval multiplication

$$[\underline{a}, \overline{a}] \cdot [\underline{b}, \overline{b}] = [\min(\underline{a} \cdot \underline{b}, \underline{a} \cdot \overline{b}, \overline{a} \cdot \underline{b}, \overline{a} \cdot \overline{b}), \max(\underline{a} \cdot \underline{b}, \underline{a} \cdot \overline{b}, \overline{a} \cdot \underline{b}, \overline{a} \cdot \overline{b})].$$

In this situation,

- we seem to need four multiplication of numbers to compute the product of two intervals, but
- in reality, three multiplications are sufficient [8] (see also [4]).

Example. In our illustrative example, we have

$$|\widetilde{b}_0| + \Delta_{b,0} = |\widetilde{b}_1| + \Delta_{b,1} = 1 + 0.1 = 1.1, \quad |\widetilde{b}_2| + \Delta_{b,2} = |\widetilde{b}_2| + \Delta_{b,2} = 0,$$

$$|\widetilde{x}_0| + \Delta_{x,0} = |\widetilde{x}_1| + \Delta_{x,1} = |\widetilde{x}_2| + \Delta_{x,2} = 1 + 0.2 = 1.2, \quad |\widetilde{x}_3| + \Delta_{x,3} = 0.$$

Thus, the first auxiliary sequence $F = (|\widetilde{b}| + \Delta_b) * (|\widetilde{x}| + \Delta_x)$ has the form

$$F_0 = (|\widetilde{b}_0| + \Delta_{b,0}) \cdot (|\widetilde{x}_0| + \Delta_{x,0}) = 1.1 \cdot 1.2 = 1.32;$$

$$F_1 = (|\widetilde{b}_1| + \Delta_{b,1}) \cdot (|\widetilde{x}_0| + \Delta_{x,0}) + (|\widetilde{b}_0| + \Delta_{b,0}) \cdot (|\widetilde{x}_1| + \Delta_{x,1}) =$$

$$1.1 \cdot 1.2 + 1.1 \cdot 1.2 = 2.64;$$

$$F_2 = (|\widetilde{b}_1| + \Delta_{b,1}) \cdot (|\widetilde{x}_1| + \Delta_{x,1}) + (|\widetilde{b}_0| + \Delta_{b,0}) \cdot (|\widetilde{x}_2| + \Delta_{x,2}) =$$

$$1.1 \cdot 1.2 + 1.1 \cdot 1.2 = 2.64;$$

$$F_3 = (|\widetilde{b}_1| + \Delta_{b,1}) \cdot (|\widetilde{x}_2| + \Delta_{x,2}) = 1.1 \cdot 1.2 = 1.32.$$

For the second auxiliary sequence $S = |\widetilde{b}| * |\widetilde{x}|$, we get

$$S_0 = |\widetilde{b}_0| \cdot |\widetilde{x}_0| = 1 \cdot 1 = 1;$$

$$S_1 = |\widetilde{b}_1| \cdot |\widetilde{x}_0| + |\widetilde{b}_0| \cdot |\widetilde{x}_1| = 1 \cdot 1 + 1 \cdot 1 = 2;$$

$$S_2 = |\widetilde{b}_1| \cdot |\widetilde{x}_1| + |\widetilde{b}_0| \cdot |\widetilde{x}_2| = 1 \cdot 1 + 1 \cdot 1 = 2;$$

$$S_3 = |\widetilde{b}_1| \cdot |\widetilde{x}_2| = 1 \cdot 1 = 1.$$

Thus, for the difference $\Delta_y = F - S$, we get exactly the same values as for the previous algorithm (with four convolutions):

$$\Delta_{y,0} = F_0 - S_0 = 1.32 - 1 = 0.32, \quad \Delta_{y,1} = \Delta_{y,2} = 2.64 - 1 = 0.64,$$

$$\Delta_{y,3} = F_3 - S_3 = 1.32 - 1 = 0.32.$$

3 Auxiliary Result: Fast Fourier Transform under Interval Uncertainty

Need for FFT under interval uncertainty. In the above text, we considered FFT as a technique to compute convolution fast. However, in many

practical problems, we actually need to compute the Fourier transform $\hat{X}(\vec{\omega})$ of a given signal $x(\vec{t})$.

For example, in image processing, it is known that in the far-field (Fraunhofer) approximation, the observed signal is actually equal to the Fourier transform of the desired image; see, e.g., [15,21,23,24]. So, if we want to reconstruct the original image, we must apply the inverse Fourier transform to the measurement results $x_{\rm re}(\vec{t}) + {\rm i} \cdot x_{\rm im}(\vec{t})$, and find the values

$$\hat{X}_{\rm re}(\vec{\omega}) = \frac{1}{(\sqrt{2\pi})^d} \cdot \left(\int x_{\rm re}(\vec{t}) \cdot \cos(\vec{\omega} \cdot \vec{t}\,)\, d\vec{t} - \int x_{\rm im}(\vec{t}) \cdot \sin(\vec{\omega} \cdot \vec{t}\,)\, d\vec{t} \right); \quad (4)$$

$$\hat{X}_{\rm im}(\vec{\omega}) = \frac{1}{(\sqrt{2\pi})^d} \cdot \left(\int x_{\rm re}(\vec{t}) \cdot \sin(\vec{\omega} \cdot \vec{t}\,)\, d\vec{t} + \int x_{\rm re}(\vec{t}) \cdot \cos(\vec{\omega} \cdot \vec{t}\,)\, d\vec{t} \right). \quad (5)$$

In real life, we may only know the real and imaginary part of the signal with interval uncertainty, i.e., we only know the intervals

$$[\widetilde{x}_{\rm re}(\vec{t}) - \Delta_{\rm re}(\vec{t}), \widetilde{x}_{\rm re}(\vec{t}) + \Delta_{\rm re}(\vec{t})] \text{ and } [\widetilde{x}_{\rm im}(\vec{t}) - \Delta_{\rm im}(\vec{t}), \widetilde{x}_{\rm im}(\vec{t}) + \Delta_{\rm im}(\vec{t})]$$

that contain the actual (unknown) values of $x_{\rm re}(\vec{t})$ and $x_{\rm im}(\vec{t})$. In such situations, it is desirable to find, for every $\vec{\omega}$, the intervals of possible values of $\hat{X}_{\rm re}(\vec{\omega})$ and $\hat{X}_{\rm im}(\vec{\omega})$.

We will show how this can be done under the assumption that discretization error is much smaller than the measurement error $\Delta_{\rm re}(\vec{t})$ and $\Delta_{\rm im}(\vec{t})$ and thus, we can safely assume that we know the values of the signal $\widetilde{x}_{\rm re}(\vec{t})$ and $\widetilde{x}_{\rm im}(\vec{t})$ for all $\vec{t}$.

What was known. The solution to this problem is known for the case when all the measurements have the same measurement error, i.e., when $\Delta_{\rm re}(\vec{t})$ and $\Delta_{\rm im}(\vec{t})$ do not depend on $\vec{t}$. For this case, the solution is given in [3]. In this paper, we extend this solution to the general case.

Analysis of the problem. Formulas (4) and (5) are linear in terms of the unknowns $x_{\rm re}(\vec{t})$ and $x_{\rm im}(\vec{t})$. Thus, to describe the range of the corresponding expressions, let us recall how to estimate the range of a general linear function under interval constraints.

In general, every value x_j from the interval $[\widetilde{x}_j - \Delta_j, \widetilde{x}_j + \Delta_j]$ can be described as $x_j = \widetilde{x}_j + \Delta x_j$, where $|\Delta x_j| \le \Delta_j$. In terms of $\widetilde{x}_j$ and Δx_j, the value of a (general) linear function $y = a_0 + \sum\limits_{j=1}^{N} a_j \cdot x_j$ takes the form $\widetilde{y} + \Delta y$, where

$$\widetilde{y} \stackrel{\rm def}{=} a_0 + \sum_{j=1}^{N} a_j \cdot \widetilde{x}_j \tag{6}$$

and $\Delta y = \sum\limits_{j=1}^{N} a_j \cdot \Delta x_j$. The largest possible value of Δy for $\Delta x_j \in [-\Delta_j, \Delta_j]$ is attained when $\Delta x_j = \Delta_j$ for $a_j \ge 0$ and when $\Delta x_j = -\Delta_j$ for $a_j \le 0$. In both cases, the largest value Δ of Δy is equal to

$$\Delta = \sum_{j=1}^{N} |a_j| \cdot \Delta_j. \tag{7}$$

Thus, the range of a linear function $y = a_0 + \sum\limits_{j=1}^{N} a_j \cdot x_j$ under interval uncertainty $x_j \in [\widetilde{x}_j - \Delta_j, \widetilde{x}_j + \Delta_j]$ is equal to $[\widetilde{y} - \Delta, \widetilde{y} + \Delta]$.

In particular, for $\hat{X}_{\rm re}(\vec{\omega})$, the desired range is equal to

$$[\widehat{\widetilde{X}}_{\rm re}(\vec{\omega}) - \delta_{\rm re}(\vec{\omega}), \widehat{\widetilde{X}}_{\rm re}(\vec{\omega}) + \delta_{\rm re}(\vec{\omega})],$$

where $\widehat{\widetilde{X}}_{\rm re}(\vec{\omega})$ is the real part of the (easy-to-compute) Fourier transform of the approximate function $\widetilde{x}_{\rm re}(\vec{t}) + {\rm i} \cdot \widetilde{x}_{\rm im}(\vec{t})$ and

$$\delta_{\rm re}(\vec{\omega}) = \frac{1}{(\sqrt{2\pi})^d} \cdot \left(\int \Delta_{\rm re}(\vec{t}) \cdot |\cos(\vec{\omega} \cdot \vec{t})| \, d\vec{t} + \int \Delta_{\rm im}(\vec{t}) \cdot |\sin(\vec{\omega} \cdot \vec{t})| \, d\vec{t} \right). \tag{8}$$

Similarly, for $\hat{X}_{\rm im}(\vec{\omega})$, the desired range is equal to

$$[\widehat{\widetilde{X}}_{\rm im}(\vec{\omega}) - \delta_{\rm im}(\vec{\omega}), \widehat{\widetilde{X}}_{\rm im}(\vec{\omega}) + \delta_{\rm im}(\vec{\omega})],$$

where $\widehat{\widetilde{X}}_{\rm im}(\vec{\omega})$ is the imaginary part of the (easy-to-compute) Fourier transform of the approximate function $\widetilde{x}_{\rm re}(\vec{t}\,) + {\rm i} \cdot \widetilde{x}_{\rm im}(\vec{t}\,)$ and

$$\delta_{\rm im}(\vec{\omega}) =$$

$$\frac{1}{(\sqrt{2\pi})^d} \cdot \left(\int \Delta_{\rm re}(\vec{t}\,) \cdot |\sin(\vec{\omega} \cdot \vec{t}\,)|\, d\vec{t} + \int \Delta_{\rm im}(\vec{t}\,) \cdot |\cos(\vec{\omega} \cdot \vec{t}\,)|\, d\vec{t} \right). \qquad (9)$$

Thus, to be able to find these ranges, we must be able to compute the integrals

$$I_{{\rm re},c} \stackrel{\rm def}{=} \int \Delta_{\rm re}(\vec{t}\,) \cdot |\cos(\vec{\omega} \cdot \vec{t}\,)|\, d\vec{t}, \quad I_{{\rm im},c} \stackrel{\rm def}{=} \int \Delta_{\rm im}(\vec{t}\,) \cdot |\cos(\vec{\omega} \cdot \vec{t}\,)|\, d\vec{t},$$

$$I_{{\rm re},s} \stackrel{\rm def}{=} \int \Delta_{\rm re}(\vec{t}\,) \cdot |\sin(\vec{\omega} \cdot \vec{t}\,)|\, d\vec{t}, \quad I_{{\rm im},s} \stackrel{\rm def}{=} \int \Delta_{\rm im}(\vec{t}\,) \cdot |\sin(\vec{\omega} \cdot \vec{t}\,)|\, d\vec{t}.$$

Let us consider the first of these integrals (the other three can be computed similarly). By definition, $\vec{\omega} \cdot \vec{t} = \omega_1 \cdot t_1 + \ldots + \omega_d \cdot t_d$; thus,

$$I_{{\rm re},c}(\omega_1, \ldots, \omega_d) = \int \Delta_{\rm re}(t_1, \ldots, t_d) \cdot |\cos(\omega_1 \cdot t_1 + \ldots + \omega_d \cdot t_d)|\, dt_1 \ldots dt_d.$$

How can we speed up the computation of this integral? We know how to speed up the computation of the convolution $y(t) = \int a(t-s) \cdot x(s)\, ds$. Let us try to use this knowledge here. For that, let us first describe the similarity and the differences between the convolution integral and the integral $I_{{\rm re},c}$ that we want to compute.

- In the convolution, we integrate the product of a function $x(s)$ of an auxiliary variable s and a function $a(t-s)$ depending on the *difference* between the variable t and the auxiliary variable s.
- In the expression for $I_{{\rm re},c}(\vec{\omega})$, we integrate the product of a function $\Delta_{\rm re}(\vec{t}\,)$ of the auxiliary variables $t_1, \ldots, t_d$, and a (cosine) function of the *products* $\omega_i \cdot t_i$ between the variables ω_i and the auxiliary variables t_i.

The only difference between our integral and the convolution is that in our integral, we have the product of the two variables, while in the convolution, we have the difference between the variables. Thus, to reduce our integral to convolution, we must reduce the product $\omega_i \cdot t_i$ to a difference. This reduction will be done in two natural steps.

First, we use the well-known fact that the logarithm of a product is equal to the sum of logarithms. We use this fact to reduce the product to the sum. Specifically, we introduce the new variables $W_i \stackrel{\text{def}}{=} \ln(\omega_i)$ and $S_i \stackrel{\text{def}}{=} \ln(t_i)$.

Comment. Strictly speaking, this idea works only for $\omega_i > 0$ and $t_i > 0$, since (real-valued) logarithms are only defined for positive values. To cover the entire integral, we must represent the integral as the sum of 2^d integrals over all 2^d (i.e., 4 or 8) orthants, and use the variables $W_i = \ln|\omega_i|$ and $S_i = \ln|t_i|$ in each orthant.

In terms of the new variables, $\omega_i = e^{W_i}$, $t_i = e^{S_i}$, and thus, $\omega_i \cdot t_i = e^{W_i} \cdot e^{S_i} = e^{W_i + S_i}$. Thus, the expression under the integral $I_{\text{re},c}$ takes the form

$$\Delta_{\text{re}}(e^{S_1}, \ldots, e^{S_d}) \cdot |\cos(e^{W_1+S_1} + \ldots + e^{W_d+S_d})|.$$

To describe the integral $I_{\text{re},c}$ itself in terms of the new variables, we must describe dt_i in terms of these new variables S_i. For $t_i = e^{S_i}$, we have $dt_i = e^{S_i} \cdot dS_i$, and therefore,

$$dt_1 \ldots dt_d = e^{S_1} \cdot \ldots \cdot e^{S_d}\, dS_1 \ldots dS_d = e^{S_1 + \ldots + S_d}\, dS_1 \ldots dS_d.$$

Thus, in terms of the new variables W_i and S_i, the desired integral

$I_{\mathrm{re},c}(\omega_1,\ldots,\omega_d)$ gets the form

$$I_{\mathrm{re},c}(\omega_1,\ldots,\omega_d) = F_{\mathrm{re},c}(\ln(\omega_1),\ldots,\ln(\omega_d)),$$

where

$$F_{\mathrm{re},c}(W_1,\ldots,W_d) =$$

$$\int \Delta_{\mathrm{re}}(e^{S_1},\ldots,e^{S_d})\cdot e^{(S_1+\ldots+S_d)}\cdot|\cos(e^{W_1+S_1}+\ldots+e^{W_d+S_d})|\,dS_1\ldots dS_d.$$

In the above expression, we integrate the product of a function

$$\Delta_{\mathrm{re}}(e^{S_1},\ldots,e^{S_d})\cdot e^{(S_1+\ldots+S_d)}$$

of the auxiliary variables $S_1,\ldots,S_d$, and a (cosine) function of the sums W_i+S_i between the variables W_i and the auxiliary variables S_i. This is *almost* convolution, the only difference is that in the convolution, we have a difference between the variables, and here we have the sum. So, to reduce our problem to the problem of computing of the convolution, it is sufficient to represent each sum W_i+S_i as the difference. This can be done by simply introducing a new variable $T_i \stackrel{\mathrm{def}}{=} -\ln(S_i)$ (i.e., $T_i=-\ln(t_i)$). In terms of this new variables T_i, we have $S_i=-T_i$ and therefore, $W_i+S_i=W_i-T_i$ (i.e., the sum indeed turns into the difference).

To complete the description of the desired integral in terms of the new variables, we must also describe dS_i in terms of the new variables. Since $S_i=-T_i$, we have $dS_i=-dT_i$, hence

$$dS_1\ldots dS_d=(-1)^d\,dT_1\ldots dT_d,$$

and the expression for the integral $F_{\mathrm{re},c}(W_1,\ldots,W_d)$ takes the form

$$F_{\mathrm{re},c}(W_1,\ldots,W_d) =$$

$$(-1)^d \cdot \int \Delta_{\rm re}(e^{-T_1}, \ldots, e^{-T_d}) \cdot e^{-(T_1+\ldots+T_d)} \cdot |\cos(e^{W_1-T_1}+\ldots+e^{W_d-T_d})|\, dT_1 \ldots dT_d.$$

This is already a convolution. Specifically,

$$I_{{\rm re},c}(\omega_1, \ldots, \omega_d) = (-1)^d \cdot F_{{\rm re},c}(\ln(\omega_1), \ldots, \ln(\omega_d)), \tag{10}$$

where $F_{{\rm re},c}(W_1, \ldots, W_d)$ is the convolution of the following two functions:

$$f_{\rm re}(T_1, \ldots, T_d) = \Delta_{\rm re}(e^{-T_1}, \ldots, e^{-T_d}) \cdot e^{-(T_1+\ldots+T_d)} \tag{11}$$

and

$$g_c(T_1, \ldots, T_d) = |\cos(e^{T_1} + \ldots + e^{T_d})|. \tag{12}$$

The three other integrals $I_{{\rm im},c}$, $I_{{\rm re},s}$, and $I_{{\rm im},s}$ can be similarly reduced to computing convolutions. For computing these convolutions, in addition to the above functions $f_{\rm re}$ and g_c, we also need to compute two similarly defined functions

$$f_{\rm im}(T_1, \ldots, T_d) = \Delta_{\rm im}(e^{-T_1}, \ldots, e^{-T_d}) \cdot e^{-(T_1+\ldots+T_d)} \tag{13}$$

and

$$g_s(T_1, \ldots, T_d) = |\sin(e^{T_1} + \ldots + e^{T_d})|. \tag{14}$$

We know that convolution can be computed fast. Thus, we arrive at the following fast algorithm for computing Fourier transform under interval uncertainty.

Resulting algorithm. Suppose that we are given the intervals

$$[\widetilde{x}_{\rm re}(\vec{t}) - \Delta_{\rm re}(\vec{t}), \widetilde{x}_{\rm re}(\vec{t}) + \Delta_{\rm re}(\vec{t})] \text{ and } [\widetilde{x}_{\rm im}(\vec{t}) - \Delta_{\rm im}(\vec{t}), \widetilde{x}_{\rm im}(\vec{t}) + \Delta_{\rm im}(\vec{t})].$$

Then, to compute the ranges $[\widehat{\widetilde{X}}_{\rm re}(\vec{\omega}) - \delta_{\rm re}(\vec{\omega}), \widehat{\widetilde{X}}_{\rm re}(\vec{\omega}) + \delta_{\rm re}(\vec{\omega})]$ of the real and imaginary parts of the Fourier transform, we do the following:

- first, we apply FFT to the approximate function $\widetilde{x}_{\rm re}(\vec{t}) + {\rm i} \cdot \widetilde{x}_{\rm im}(\vec{t})$ and compute $\hat{\widetilde{X}}_{\rm re}(\vec{\omega})$ and $\hat{\widetilde{X}}_{\rm re}(\vec{\omega})$;
- then, we use FFT-based convolution algorithm to compute the convolutions $F_{{\rm re},c} = f_{\rm re} * g_c$, $F_{{\rm re},s} = f_{\rm re} * g_s$, $F_{{\rm im},c} = f_{\rm im} * g_c$, and $F_{{\rm im},s} = f_{\rm im} * g_s$;
- re-scale these functions by computing

$$I_{{\rm re},c}(\omega_1, \ldots, \omega_d) = (-1)^d \cdot F_{{\rm re},c}(\ln(\omega_1), \ldots, \ln(\omega_d)),$$

$$I_{{\rm re},s}(\omega_1, \ldots, \omega_d) = (-1)^d \cdot F_{{\rm re},s}(\ln(\omega_1), \ldots, \ln(\omega_d)),$$

$$I_{{\rm im},c}(\omega_1, \ldots, \omega_d) = (-1)^d \cdot F_{{\rm im},c}(\ln(\omega_1), \ldots, \ln(\omega_d)),$$

$$I_{{\rm im},s}(\omega_1, \ldots, \omega_d) = (-1)^d \cdot F_{{\rm im},s}(\ln(\omega_1), \ldots, \ln(\omega_d));$$

- finally, we compute

$$\delta_{\rm re}(\vec{\omega}) = \frac{1}{(\sqrt{2\pi})^d} \cdot (I_{{\rm re},c}(\vec{\omega}) + I_{{\rm im},s}(\vec{\omega})); \quad \delta_i(\vec{\omega}) = \frac{1}{(\sqrt{2\pi})^d} \cdot (I_{{\rm re},s}(\vec{\omega}) + I_{{\rm im},c}(\vec{\omega})).$$

Since we are only using FFT or FFT-based convolution, we thus arrive at a fast algorithm for computing Fourier Transform under interval uncertainty.

References

[1] Th. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 2001.

[2] D. Dubois and H. Prade, "Operations on fuzzy numbers", *International Journal of Systems Science*, 1978, Vol. 9, pp. 613–626.

[3] J. Garloff, "Zur intervallmässigen Durchführung der schnellen Fourier-Transformation", *ZAMM*, 1980, Vol. 60, pp. T291–T292.

[4] C. Hamzo and V. Kreinovich, "On Average Bit Complexity of Interval Arithmetic", *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 1999, Vol. 68, pp. 153–156.

[5] E. Hansen, "Sharpness in interval computations", *Reliable Computing*, 1997, Vol. 3, pp. 7–29.

[6] M. Hauenschild, "Arithmetiken für komplexe Kreise", *Computing*, 1974, Vol. 13, No. 3–4, pp. 299–312.

[7] M. Hauenschild, "Extended Circular Arithmetic, Problems and Results", In: K. Nickel (ed.), *Interval Mathematics 1980*, Academic Press, New York, 1980, pp. 367–376.

[8] G. Heindl, *An improved algorithm for computing the product of two machine intervals*, Interner Bericht IAGMPI- 9304, Fachbereich Mathematik, Gesamthochschule Wuppertal, 1993.

[9] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics*, Springer-Verlag, London, 2001.

[10] G. Klir and B. Yuan, *Fuzzy sets and fuzzy logic: theory and applications.* Prentice Hall, Upper Saddle River, New Jersey, 1995.

[11] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational complexity and feasibility of data processing and interval computations*, Kluwer, Dordrecht, 1997.

[12] N. Krier, *Komplexe Kreisarithmetik*, Ph.D. thesis, Universität Karlsruhe, 1973.

[13] N. Krier, "Komplexe Kreisarithmetik", *Z. Angew. Math. Mech.*, 1974, Vol. 54, pp. T225–T226.

[14] N. Krier and P. Spellucci, "Einschliessungsmengen von Polynom-Nullstellen", in: K. Nickel, Ed., *Interval Mathematics*, Springer, Heidelberg, 1975, pp. 223–228.

[15] J. R. Meyer-Arendt, *Introduction to Classical and Modern Optics*, Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[16] I. Najfeld, *Discrete Fourier transform of vector sets and iterations with uncertain circulant matrices*, Report No. 81–19, Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, Virginia 23665, 1981.

[17] H. T. Nguyen, "A note on the extension principle for fuzzy sets", *J. Math. Anal. and Appl.*, 1978, Vol. 64, pp. 369–380.

[18] H. T. Nguyen and V. Kreinovich, "Nested intervals and sets: concepts, relations to fuzzy sets, and applications" In: R. B. Kearfott and V. Kreinovich (eds.), *Applications of Interval Computations*, Kluwer, Dordrecht, 1996, pp. 245–290.

[19] H. T. Nguyen and E. A. Walker, *A first course in fuzzy logic*, CRC Press, Boca Raton, Florida, 2006.

[20] S. M. Rump, "Fast and parallel interval arithmetic", *BIT*, 1999, Vol. 39, No. 3, pp. 534–554.

[21] R. A. Serway, *Physics for Scientists and Engineers*, Saunders Publ., Philadelphia, 1996.

[22] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, California Technical Publishing, Pasadena, California, 1997.

[23] A. R. Thompson, J. M. Moran, G. W. Swenson Jr., *Interferometry and Synthesis in Radio Astronomy*, Wiley, New York, 2001.

[24] G. Verschuur and K. I. Kellerman (eds.), *Galactic and extragalactic radio astronomy*, Springer Verlag, 1988.