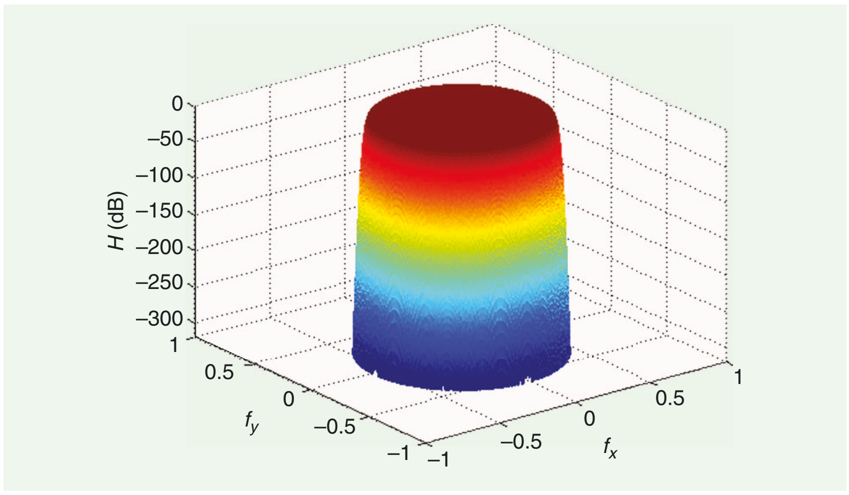


FIGURE 5. A circular 2-D filter.

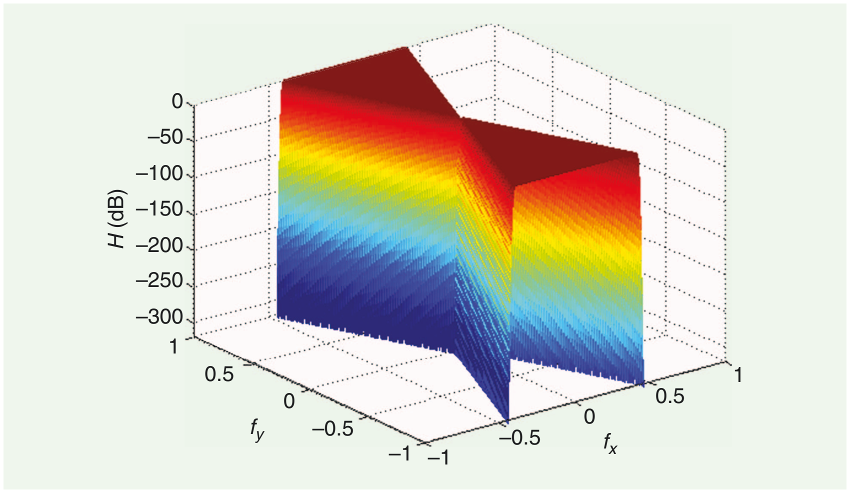


FIGURE 6. A fan 2-D filter.

mathematics and natural sciences faculty of South-West University, Blagoevgrad, Bulgaria. He is the author of two books and more than 60 articles, conference papers, and research works in mathematical logics, Hausdorff's approximations, and bioinformatics.

Alexey K. Stefanov (astef@abv.bg) has been an associate professor with the technical faculty of South-West University, Blagoevgrad, Bulgaria, since 2012. He was a previously deputy director of the Telecommunication Department, the Telecommunication Department of the Ministry of Interior, Sofia, Bulgaria. His research interests include digital audio and video processing and devices. He is the author of two books and more than 40 articles, as well as the chief designer of 10 constructive developments.

References

[1] W. S. Lu and A. Antoniou, *Two-Dimensional Digital Filters*. New York: Marcel Dekker, 1992.

[2] J. S. Lim and A. Antoniou, *Two-Dimensional Signal and Image Processing*. Englewood Cliffs, NJ: Prentice Hall, 1990.

[3] S. Kockanat and N. Karaboga, *The Design Approaches of Two-Dimensional Digital Filters Based on Metaheuristic Optimization Algorithms: A Review of the Literature*. New York: Springer-Verlag, 2015, pp. 265–287.

[4] X. Y. Hong, X. P. Lai, and R. J. Zhao, "Matrix-based algorithms for constrained least-squares and minimax designs of 2-d FIR filters," *IEEE Trans. Signal Process.*, vol. 64, no. 14, pp. 3620–3631, July 2013.

[5] L. Andrews, *Special Functions of Mathematics for Engineers*. London, U.K.: Oxford Univ. Press, 1998.

Logan L. Grado, Matthew D. Johnson, and Theoden I. Netoff

The Sliding Windowed Infinite Fourier Transform

The discrete Fourier transform (DFT) is the standard tool for spectral analysis in digital signal processing, typically computed using the fast Fourier transform (FFT). However, for real-time applications that require recalculating the DFT at each sample or over only a subset of the N center frequencies of the DFT, the FFT is far from optimal.

The sliding DFT (SDFT), first developed by Springer in 1988 [1] and then improved and popularized by Jacobsen and Lyons in 2003 [2], [3], is an algorithm that computes individual DFT bins recursively, allowing for efficient computation of the DFT on a sample-by-sample basis. The SDFT is efficient; however, it is limited in that it is only marginally stable and requires storing N previous inputs. Furthermore, the SDFT's rectangular window causes spectral leakage and is limited to computing the N center frequencies of the DFT.

Here, we present a novel sliding discrete-time Fourier transform (DTFT), which we call the *sliding windowed infinite Fourier transform* (*SWIFT*), that

Digital Object Identifier 10.1109/MSP.2017.2718039
Date of publication: 6 September 2017

has several advantages over the SDFT. The SWIFT is guaranteed stable, reduces spectral leakage without increasing computational complexity, improves frequency-domain sampling, and gives greater weight to more recent samples, allowing for improved real-time spectral and phase analysis. Additionally, we present a modified version of the SWIFT algorithm, called the *αSWIFT*, which further reduces spectral leakage. We conclude by comparing all three algorithms with a brief numerical simulation.

The sliding DFT

The SDFT performs an N-point DFT on samples within a sliding rectangular window. The DFT is initially computed on the first N samples. The time window is then advanced one sample, and a new N-point DFT is calculated directly from the results of the previous DFT. The SDFT can be expressed compactly as

$$X_n[k] = X_{n-1}[k]e^{j2\pi k/N} - x[n-N] + x[n]. \quad (1)$$

The SDFT's output is discrete in the frequency domain and limited to normalized frequencies of $2\pi k/N$, $k \in \mathbb{Z}$. Using this method, the DFT can be efficiently recalculated at each sample using only a few operations.

The single-bin SDFT algorithm can be implemented as an infinite-impulse response (IIR) filter with a comb filter followed by a complex resonator. The recursive nature of the SDFT dictates the requirement of some initialization method; the output $X_n[k]$ is only valid if $X_{n-1}[k]$ is valid. There are two methods for initializing the algorithm:

1) Reset all $X_{n-1}[k]$s to zero, and then begin cycling data; after N samples have cycled, the output will be valid.
2) Initialize all $X_{n-1}[k]$ with an FFT of the previous N samples.

For a full description of the SDFT, see Jacobsen and Lyons [2], [3].

While the SDFT is an efficient algorithm, the use of a rectangular window results in spectral leakage. To address this, Jacobsen and Lyons described how to implement time-domain windowing via frequency-domain convolution. This can be performed with almost any finite window, but it significantly increases the computational complexity and compromises the simplicity of the SDFT.

The sliding windowed infinite Fourier transform

The SWIFT is a type of sliding DTFT that is windowed with an infinite-length, causal exponential function

$$w[m] = \begin{cases} e^{m/\tau} & m \leq 0 \\ 0 & m > 0' \end{cases} \quad (2)$$

where $w[m]$ is the window function, $m = 0$ is the current sample, and $\tau > 0$ is the time constant of the window, with units of samples. The exponential window gives greater weight to more recent samples, allowing the SWIFT to be more sensitive to transient changes in signal power than the rectangular window. The exponential windowed DTFT is

$$X_n(\omega) = \sum_{m=-\infty}^{0} e^{m/\tau} x[n+m] e^{-j\omega m}, \quad (3)$$

where ω has normalized units of radians/sample ($\omega = 2\pi f/f_s$) and is continuous in the frequency domain. We can derive a recursive formula for (3) by relating $X_{n+1}(\omega)$ back to $X_n(\omega)$ [see (4) in the box at the bottom of the page]. Finally, we decrement the result of (4) one sample to yield the recursive SWIFT formulation

$$X_n(\omega) = e^{-1/\tau} e^{j\omega} X_{n-1}(\omega) + x[n]. \quad (5)$$

The SWIFT operates by rotating the phase of the previous DTFT by ω, decaying the amplitude by $e^{-1/\tau}$, and adding in the new data sample. Figure 1(a) demonstrates how the SWIFT's window advances one sample at a time, picking up the new data sample and updating the previous samples. (The incremental advance and infinite nature of the time window are what led us to the name *sliding windowed infinite Fourier transform.*)

Derivation and equivalence

The SWIFT is derived directly from, and shows exact equivalence to, the windowed DTFT; therefore there is no loss of information or distortion tradeoff with the SWIFT as compared to other means of calculating the DTFT. The SWIFT algorithm calculates $X_n(\omega)$ by phase shifting and decaying the previous $X_{n-1}(\omega)$ and adding the current $x[n]$ sample; thus, the SWIFT requires only one complex multiply and one real add per sample per bin.

Initialization

Like the SDFT, the SWIFT can be initialized by sliding onto the data or by

$$\begin{aligned}
X_{n+1}(\omega) &= \sum_{m=-\infty}^{0} e^{m/\tau} x[n+m+1] e^{-j\omega m} \\
&= \sum_{m=-\infty+1}^{1} e^{(m-1)/\tau} x[n+m] e^{-j\omega(m-1)} \\
&= \sum_{m=-\infty}^{0} e^{(m-1)/\tau} x[n+m] e^{-j\omega(m-1)} + \underbrace{e^{(1-1)/\tau} x[n+1] e^{-j\omega(1-1)}}_{x[n+1]} - \underbrace{e^{(-\infty-1)/\tau} x[n-\infty] e^{-j\omega(-\infty-1)}}_{0} \\
&= e^{-1/\tau} e^{j\omega} \underbrace{\sum_{m=-\infty}^{0} e^{m/\tau} x[n+m] e^{-j\omega m}}_{X_n(\omega)} + x[n+1] \\
X_{n+1}(\omega) &= e^{-1/\tau} e^{j\omega} X_n(\omega) + x[n+1] \qquad (4)
\end{aligned}$$

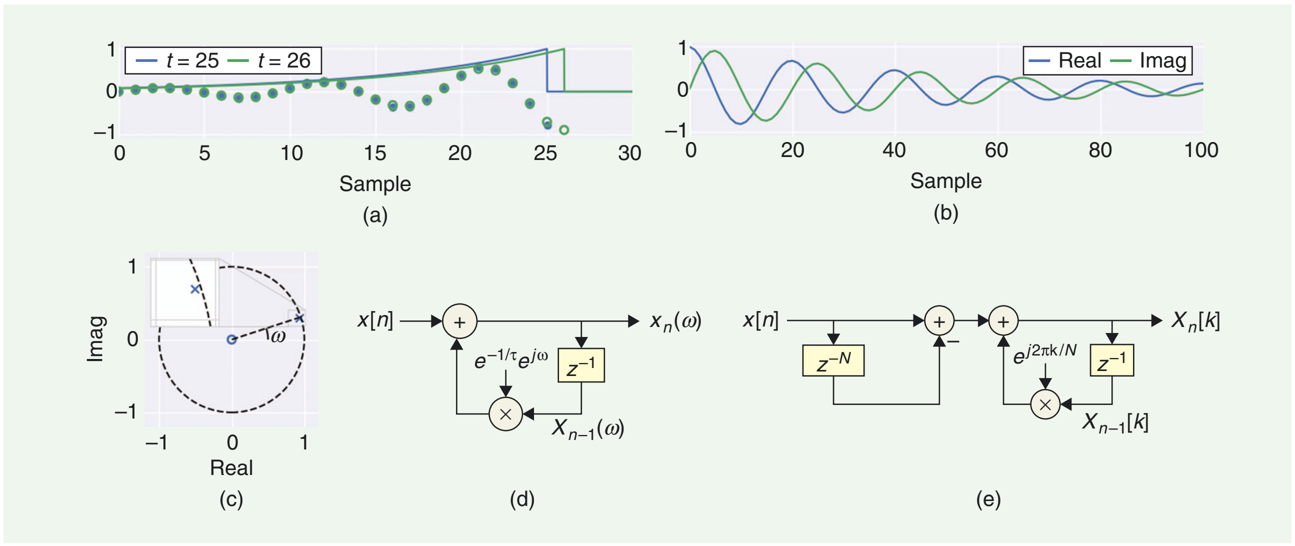


FIGURE 1. (a) The signal windowing for the SWIFT algorithm. The data samples and window used for the first computation (blue) and second computation (green). (b) The impulse response and (c) the pole/zero map for a single bin SWIFT with $\tau = 50$ samples and $\omega = \pi/10$ radians/sample. (d) The single-bin SWIFT filter structure and (e) the single-bin SDFT filter structure.

calculating the DTFT with an exponential window on all previous data. However, because the window is infinite in length, the output will never truly become valid but will instead asymptote to the true value with a time constant of τ. In practice, however, if τ is short enough, this is not an issue.

Transfer function and impulse response

The z-domain transfer function of the SWIFT filter with normalized angular frequency ω is given by

$$H_{\text{SWIFT}}(z) = \frac{1}{1 - e^{-1/\tau}e^{j\omega}z^{-1}}. \quad (6)$$

The SWIFT IIR filter has one zero at the origin and a single pole lying inside the unit circle at $e^{-1/\tau}e^{j\omega}$. The SWIFT's impulse response and pole/zero map are shown in Figure 1(b) and (c), with $\tau = 50$ samples and $\omega = \pi/10$ radians/sample.

IIR filter implementation

Like the SDFT algorithm, the SWIFT algorithm can be implemented as an IIR filter with a complex resonator, as shown in Figure 1(d). The major difference between the SWIFT filter and an SDFT filter [Figure 1(e)] is that the SWIFT filter does not require a comb filter. Any arbitrary number of frequency bins can be calculated by adding more complex resonators at the desired frequencies.

SWIFT versus SDFT

The SWIFT has several advantages over the SDFT, as discussed in the following.

Computational efficiency

The SWIFT is more efficient than the SDFT. To objectively compare the algorithms, we will consider only the costs of computing a single bin for each.

Both the SWIFT and SDFT share the property that the number of computations required to calculate $X_n(\omega)$ from $X_{n-1}(\omega)$ (or $X_n[k]$ from $X_{n-1}[k]$) is fixed and independent of the window length. However, the SWIFT requires one complex multiply and one real add to compute the next output, whereas the SDFT requires one complex multiply and two real adds. In addition to increased computational efficiency, the SWIFT has drastically reduced memory requirements.

To facilitate comparison, we have converted complex operations into real operations, assuming that one complex multiply requires two real adds and four real multiplies (although it is possible to compute with three real multiplies and five additions [4]). Both the SWIFT and SDFT require storing one previous complex output and one complex constant (four floating points). However, the SDFT must store N previous input samples, while the SWIFT does not require storage of any previous input samples. The storage and retrieval of N previous samples may be a significant limitation for small sensors and embedded devices. Table 1 compares the computational efficiency and memory requirements of the SDFT and SWIFT algorithms.

Frequency-domain sampling

As a type of DFT, the SDFT's output is limited to normalized frequencies of $2\pi k/N$, $k \in \mathbb{Z}$. To achieve finer frequency-domain sampling, the SDFT requires a larger N, reducing temporal resolution and thus producing a tradeoff

Table 1. A single-bin comparison of the computational cost and memory requirements of computing the next $X_n[k]/X_n(\omega)$ using the DFT, SDFT, and SWIFT.

Method	Real multiplies	Real adds	Memory (floats)
DFT	Two N	Two N	N
SDFT	Four	Four	N + Four
SWIFT	Four	Three	Four

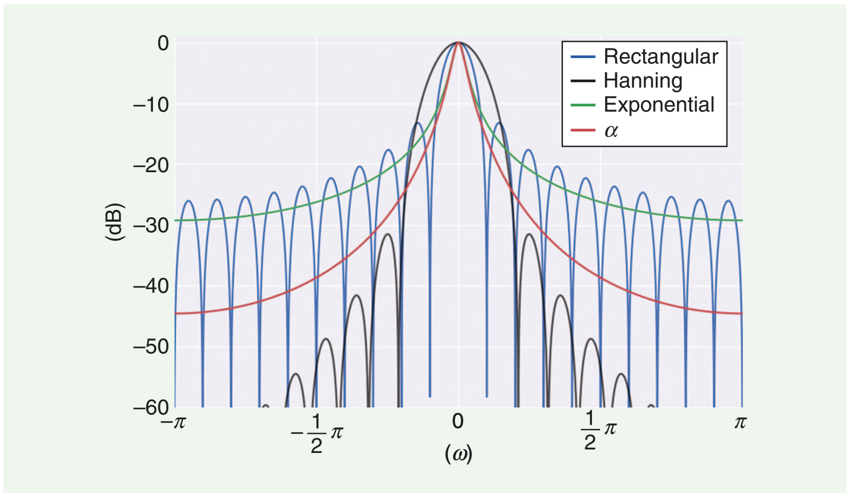


FIGURE 2. The normalized Fourier transform of four windows [rectangular (blue, $N = 20$), Hanning (black, $N = 20$), exponential (green, $\tau = 14.4$)] and of α (red, $\tau_{\text{slow}} = 14.4$, $\tau_{\text{fast}} = 2.89$).

between time and frequency resolution. Conversely, the SWIFT's output, as a type of DTFT, is continuous in the frequency domain, providing the SWIFT with great flexibility in tuning the frequencies of interest.

Time-frequency tradeoff

When operating multiple SWIFTs in parallel, each time constant can be tuned to the frequency bin of interest without increasing computational complexity, e.g., τ can be set as a multiple of the period such that $\tau = c/f$, where c is a unitless constant and f is the center frequency. Conversely, to achieve a similar effect with parallel SDFTs, one must add additional comb filters for each SDFT bin, further increasing computational complexity and memory requirements. This allows the SWIFT algorithm to be implemented with a multiresolution property, similar to a wavelet transform, which provides better time resolution at higher frequencies and better frequency resolution at lower frequencies.

Stability

The SWIFT is guaranteed stable, whereas the SDFT is only marginally stable. The SWIFT is guaranteed stable because its pole resides *within* the z-domain's unit circle. In contrast, the SDFT's pole resides *on* the z-domain's unit circle, which can lead to instabilities if numerical rounding causes the pole to move outside the unit circle.

To guarantee stability, the SDFT must add a damping factor, but this causes the SDFT's output no longer to be exactly equivalent to the N-point DFT. Other SDFTs have been developed that are both accurate and guaranteed stable, but at the cost of increased computational complexity [5].

Spectral leakage

The SWIFT's exponential window reduces spectral leakage compared to the SDFT's rectangular window, as shown in Figure 2. It is difficult to compare the leakage of finite-length windows to infinite-length windows directly; therefore, instead of requiring that each window have the same length, we required that each window have the same halfmass, i.e., the length of the window in which half the area is contained. For instance, a rectangular window of length $N = 20$ and an exponential window with $\tau = 14.43$ both have a halfmass of ten. The exponential window has a narrower main lobe and smoother falloff compared to the rectangular window. We can further reduce the SWIFT's spectral leakage with another window, which we will introduce in the αSWIFT algorithm.

Despite these advantages, there may be situations in which the traditional SDFT is called for. For instance, the sharpness of the SWIFT/αSWIFT's window may be too narrow for some applications that require tracking a broad oscillation. Additionally, any window can be implemented with the SDFT (at the cost of increased complexity), while the SWIFT is limited to the exponential window. The SDFT is also more directly comparable to the FFT.

The αSWIFT

The spectral leakage of the SWIFT can be further mitigated by removing the exponential window's discontinuity at $m = 0$. The discontinuity can be removed by modifying the window function to be the difference of two exponentials:

$$w_\alpha[m] = \begin{cases} e^{m/\tau_{\text{slow}}} - e^{m/\tau_{\text{fast}}} & m \le 0 \\ 0 & m > 0 \end{cases}, \tag{7}$$

where $\tau_{\text{slow}} > \tau_{\text{fast}} > 0$. We will refer to this as the α function, which goes smoothly to zero at $m = 0$.

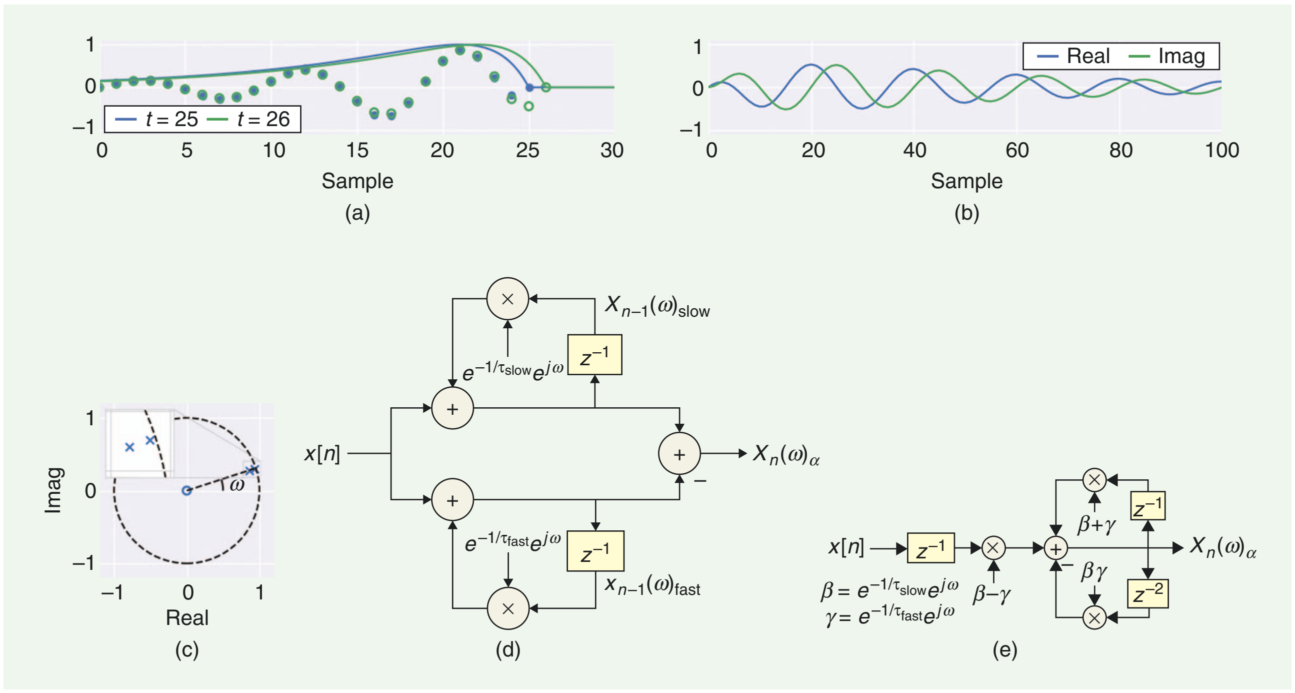
Figure 2 compares the spectral leakage of the α window with the exponential, rectangular, and Hanning windows. We have chosen to compare the αSWIFT to the Hanning SDFT, which is among the simplest windowed SDFTs and was presented by Jacobsen and Lyons in 2003 [2]. As compared to the exponential window, the α window has a similarly narrow main lobe but significantly faster fall off at surrounding frequencies. On the other hand, the Hanning window has a significantly wider main lobe, but its side lobes fall off faster than the α function's.

Derivation

The αSWIFT cannot be derived using the same method as the SWIFT because $w_\alpha[0] = 0$, and so the αSWIFT cannot be written as a difference equation in the form of $X_n(\omega)_\alpha = aX_{n-1}(\omega)_\alpha + x[n]$. However, the αSWIFT can be solved as the difference between two SWIFTs with different time constants through the linearity property of the Fourier transform:

$$X_n(\omega)_\alpha = X_n(\omega)_{\text{slow}} - X_n(\omega)_{\text{fast}}, \tag{8}$$

where $X_n(\omega)_\alpha$ is the αSWIFT and $X_n(\omega)_{\text{slow}}$ and $X_n(\omega)_{\text{fast}}$ are individual SWIFTs with τ's equal to the slow and fast time constants, respectively. We call this form of the αSWIFT the *parallel*



t = 25
t = 26
Sample
(a)
Real
Imag
Sample
(b)
Imag
Real
ω
(c)
x[n]
$X_{n-1}(\omega)_{\text{slow}}$
$e^{-1/\tau_{\text{slow}}}e^{j\omega}$
z^{-1}
$X_n(\omega)_\alpha$
$e^{-1/\tau_{\text{fast}}}e^{j\omega}$
$X_{n-1}(\omega)_{\text{fast}}$
(d)
$\beta+\gamma$
$\beta\gamma$
$\beta-\gamma$
z^{-2}
$\beta = e^{-1/\tau_{\text{slow}}}e^{j\omega}$
$\gamma = e^{-1/\tau_{\text{fast}}}e^{j\omega}$
(e)

FIGURE 3. (a) The signal windowing for the αSWIFT algorithm: the data samples and window used for the first computation (blue) and second computation (green). (b) The impulse response and (c) the pole/zero map for a single-bin αSWIFT with $\tau_{\text{slow}} = 50$ samples, $\tau_{\text{fast}} = 10$ samples, and $\omega = \pi/10$ radians/sample. (d) The parallel αSWIFT filter structure and (e) the direct αSWIFT filter structure.

form. The αSWIFT can be seen operating on an example signal in Figure 3(a).

Transfer function and direct form

We can solve for the z-domain transfer function of (8) by substituting in (6) one for each of the slow and fast SWIFTs, to yield

$$H_{\alpha\text{SWIFT}}(z) = \frac{(\beta-\gamma)z^{-1}}{1-(\beta+\gamma)z^{-1}+\beta\gamma z^{-2}},$$

where

$$\beta = e^{-1/\tau_{\text{slow}}}e^{j\omega}$$
$$\gamma = e^{-1/\tau_{\text{fast}}}e^{j\omega}. \qquad (9)$$

From this form, we can easily analyze the poles/zeros of the system. We can then derive the discrete difference form of the αSWIFT from the inverse z-transform of (9):

$$X_n(\omega)_\alpha = (\beta+\gamma)X_{n-1}(\omega)_\alpha - \beta\gamma X_{n-2}(\omega)_\alpha + (\beta-\gamma)x[n-1], \qquad (10)$$

which we call the *direct form*. The αSWIFT's impulse response and pole/zero map are shown in Figure 3(b) and (c), with $\tau_{\text{slow}} = 50$ samples, $\tau_{\text{fast}} = 10$ samples, and $\omega = \pi/10$ rad/sample.

IIR filter implementation

The αSWIFT can also be implemented as an IIR filter in either the parallel or direct form, as shown in Figure 3(d) and (e). Both filters produce identical impulse responses and pole/zero maps. However, the parallel form is more efficient than the direct form, requiring three fewer memory locations and two fewer real multiplies to compute the next $X_n(\omega)_\alpha$.

Computational efficiency

Like windowed SDFTs, the αSWIFT compromises computational efficiency to reduce spectral leakage. However, the αSWIFT is far more efficient than comparable windowed SDFTs. Table 2 compares the computational costs and memory requirements of the αSWIFT and the Hanning-windowed SDFT.

Numerical simulation

To demonstrate the differences between the three types of SFTs, Figure 4 depicts each transform operating on a chirp signal. Each transform's center frequency is 50 Hz, which the chirp crosses 5 s into the simulation (denoted by the dashed black line). To facilitate comparison, each window is set to have the same halfmass. Both the SDFT's rectangular window ($N = 100$) and the SWIFT's exponential window ($\tau = 72.1$) have a halfmass of 50 samples. As compared to the SDFT, both the SWIFT and αSWIFT have narrower peaks and lower spectral leakage. In addition, both the SWIFT and SDFT have noise in their outputs, which is reduced in the αSWIFT.

Each transform peaks at slightly different times, as well. The SDFT, with a rectangular window, peaks 0.05 s (or 50 samples) after the chirp passes 50 Hz.

Table 2. A single-bin comparison of the computational cost and memory requirements of computing the next $X_n[k]/X_n(\omega)$ using the Hanning-windowed SDFT and αSWIFT.

Method	Real multiplies	Real adds	Memory (floats)
Hanning SDFT	18	14	$N+15$
αSWIFT	Eight	Eight	Eight

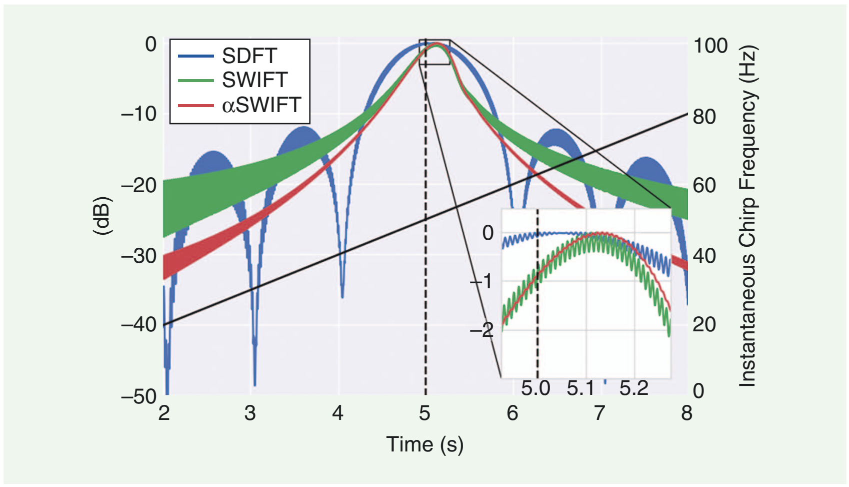


FIGURE 4. A comparison of an SDFT ($N = 100$), SWIFT ($\tau = 72.1$), and α SWIFT ($\tau_{\text{slow}} = 72.1$, $\tau_{\text{fast}} = 14.2$), with center frequencies at 50 Hz and comparable window lengths, operating on a chirp signal ($f_s = 1$ kHz).

This corresponds well with the halfmass of the window. The SWIFT and αSWIFT behave differently, however, peaking 0.121 s and 0.134 s after the chirp passes 50 Hz, despite also having halfmasses of 0.05 s.

Summary

The SWIFT algorithm for spectral analysis has been presented and shown to have several advantages over the SDFT algorithm, especially for applications that require successive calculations and real-time analysis. The SWIFT provides improved stability and frequency resolution while reducing computational complexity, memory requirements, and spectral leakage. Additionally, we presented the αSWIFT, which further reduces spectral leakage and reduces noise.

Authors

Logan L. Grado (grado@umn.edu) is a Ph.D. degree candidate in biomedical engineering at the University of Minnesota. He is broadly interested in the field of neural engineering, specifically deep brain stimulation technologies. He is also working to develop and apply machine-learning techniques to neurological disorders, such as Parkinson's disease and essential tremor.

Matthew D. Johnson (john5101@umn.edu) is an associate professor of biomedical engineering at the University of Minnesota. He received his S.B. degree in engineering sciences from Harvard University, Cambridge, Massachusetts, in 2002 and his M.S. and Ph.D. degrees in biomedical engineering from the University of Michigan in 2003 and 2007, respectively. Between 2007 and 2009, he completed a postdoctoral fellowship at the Cleveland Clinic. His primary research interests are the application of signal processing, control engineering, and neural interface technology to neuromodulation therapies.

Theoden I. Netoff (tnetoff@umn.edu) is an associate professor of biomedical engineering at the University of Minnesota. He received his bachelor's degree in psychology from the University of California, Berkeley, and his Ph.D degree in neuroscience from George Washington University. His research focuses on closed-loop therapies to optimize electrical stimulation as applied to the brain for treatment of Parkinson's disease and epilepsy.

References

[1] T. Springer, "Sliding FFT computes frequency-spectra in real-time," *Electr. Design News Mag.*, vol. 33, no. 20, pp. 161, 1988.

[2] E. Jacobsen and R. Lyons, "The sliding DFT," *IEEE Signal Process. Mag.*, vol. 20, no. 2, pp. 74–80, 2003.

[3] E. Jacobsen and R. Lyons, "An update to the sliding DFT," *IEEE Signal Process. Mag.*, vol. 21, no. 1, pp. 110–111, 2004.

[4] R. G. Lyons, *Understanding Digital Signal Procesing.* Englewood Cliffs, NJ: Prentice Hall, 2004.

[5] K. Duda, "Accurate, guaranteed stable, sliding discrete fourier transform," *IEEE Signal Process. Mag.*, vol. 27, no. 6, pp. 124–127, 2010.

SP

ERRATA

In the July 2017 issue of *IEEE Signal Processing Magazine*, an error was introduced in the title of a feature article during the production process. The title of the article by Z. Zhang, N. Cummins, and B.W. Schuller printed incorrectly [1]. The correct title is "Advanced Data Exploitation in Speech Analysis." We sincerely apologize for this error and any confusion it may have caused.

Digital Object Identifier 10.1109/MSP.2017.2727778
Date of publication: 6 September 2017

Reference

[1] Z. Zhang, N. Cummins, and B. W. Schuller, "Advanced data expoitation in speech analysis," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 107–129, July 2017.

SP