URSI AP-RASC, Sydney, Australia, 17 – 22 August 2025

A Julia Package for the End-To-End Modeling of Correlator-Beamformers in Modern Radio Telescopes

Thushara K. Gunaratne[(1)]
(1) Herzberg Astronomy and Astrophysics Research Centre, National Research Council Canada, 717 White Lake Road, Kaleden, British Columbia, CANADA, V0H 1K0.

Abstract

A Julia package is currently being developed to model the digital signal processing operations in correlator-beamformers (CBFs) used in modern radio telescopes. This package includes a library of configurable functions that simulate the signal processing operations of key CBF components, such as channelizers, tunable filters, resamplers, beamformers and cross-correlators. Additionally, it is to include test signal generators and auxiliary functions to detect radio frequency interference (RFI) and to flag or excise contaminated data. This package supports multiple numerical formats, including several low-precision floating-point formats. The key signal processing functions in this package are being optimized to exploit the available hardware resources to achieve faster processing.

1 Introduction

The development of the signal processing systems, the correlator-beamformers (CBFs) for modern radio telescopes, e.g. the Square Kilometre Array (SKA), the Atacama Large Millimeter/sub-millimeter Array (ALMA) observatory and the next-generation Very Large Array (ngVLA), rely heavily on the interoperability of multiple sophisticated signal processing systems [1]. With budgets in the range of hundreds of millions of dollars, it is important to identify the potential shortcomings of the design of the CBFs as early as possible. Hence, the end-to-end models of CBFs can play a pivotal role in ensuring the success of the next-generation radio telescopes.

One of the most important aspects of the design of the correlator-beamformers is to make sure the desired sensitivities can be achieved with the planned system. By simulating the entire signal chain, these models help ensure that the system can achieve its desired sensitivity and resolution, which is critical for detecting faint celestial signals and resolving fine details in astronomical observations [2, 3]. End-to-end models also enable the optimization of signal processing algorithms and the identification of potential issues. This is particularly important for mitigating the effects of Radio Frequency Interference (RFI) and ensuring system reliability [2]. By simulating various scenarios, including different observational modes and environmental conditions, designers can refine their systems to perform optimally under a range of conditions. Modern radio telescopes are highly complex systems involving up to thousands of receivers, extensive data networks, and sophisticated software. The end-to-end models can help manage this complexity by providing a comprehensive framework for understanding how different modules interact and affect overall system performance. This is essential for diagnosing and resolving issues during the development and deployment phases.

End-to-end models for the SKA Mid.CBF and Low.CBF has been developed using MATLAB [2, 3] and has been used in evaluating the performance of the proposed signal processing architecture [4] and robustness against RFI [5, 6]. It has been found sharing these MATLAB models is not that convenient due to stringent licensing requirements. This hinders collaboration among developers.

The Julia programming language aims to solve the 'two-language' problem by achieving both productivity and performance within a single implementation for most scientific computation projects [7]. It utilizes dynamic compilation and multiple-dispatch functions to optimize performance. Since its inception in 2012, the open-source programming language Julia has facilitated a robust, built-in package management system that allows easy collaboration among developers. Over the years, it has gained many devoted developers in various branches of science, mathematics, engineering, and finance. Julia's package ecosystem continues to grow, providing a wide range of tools for the public use and enhancement [8]. Further, Julia's "multiple dispatch" feature is one of the main reasons for its effectiveness in scientific computing enabling functions to adapt their behavior based on the types of all their arguments, fostering highly expressive and flexible code that naturally handles diverse scientific data that facilitates straightforward extension of existing functions to new data types without altering the original code.

2 Test Signal Generation

One of the most important aspects of verifying any signal processing system is the generation of test signals that accurately reflect the characteristics being evaluated. For instance, a simple test signal, such as a single tone (i.e., a

sinusoid), can be used to verify the functionality of a channelizer. In this case, the output power of each channel is recorded to ensure that the expected levels are achieved. Conversely, when testing how the end-to-end signal chain of a CBF responds to strong, time-varying RFI, the test signals can become significantly more complex. An example of this is the train of Distance Measuring Equipment (DME) pulses emitted by an aircraft flying over a radio astronomy receiver [9]. Generating test signals for this scenario requires careful selection of several parameters, including the antenna pattern, observation frequency range, position and velocity of the aircraft, and specific DME parameters such as emitted power, operation frequency, pulse profile, and pulse repetition period.

In terms of the end-to-end modeling of a CBF facilitating aperture synthesis imaging, it requires the generation of "coherent" wideband test signals with time-varying delay that corresponds to the rotation of the Earth [2]. In [10] it has been shown that an efficient method based on the *chirp-z transform (CZT)* is used to generate wideband *coherent pseudo-random* test signals with approximated piece-wise linear delay for verifying the CBF signal chains. Moreover, efficient implementations of "approximations" to *non-uniform discrete Fourier transform* at arbitrary precision, for example, "FINUFFT" [11], offers a more straightforward way to generate coherent wideband test signals with time-varying delays. Note that this technique also allows the modeling of non-idealities such as sample-time jitter. The "NFFT.jl" [12] package offers a native Julia implementation for FINUFFT whereas the two Julia packages "FINUFFT.jl" [13] and "NonuniformFFTs.jl" [14], provide the functionality through wrappers.

The performance and the accuracy of non-uniform FFT evaluations carried out using functions of the above packages have been estimated and listed in Table 1. The input $X_k = e^{j2\pi r_k}$ contains $N_k = 2^{16}$ spectral components at uniform intervals $\omega_k \in (-\pi : 2\pi/N_k : \pi - 2\pi/N_k)$ having Uniform-distributed r_k. The signal values are evaluated at random points $t_n \in (0 : 1 : N_k - 1) + 0.02rand(N_k)$. For each package, the average time for the computation has been evaluated using the macros provided in the "BenchmarkTools.jl" Julia package. The accuracy of the evaluations is also calculated compared to the direct evaluation of $x_n = \sum_k X_k e^{j\omega_k t_n}$. The normalized rms error between each of the non-uniform FFT evaluations and the direct evaluation are listed in Table 1. Note that the direct evaluation takes around 198 seconds.

3 Signal Processing in Channelizers

As given in [2, 3], the key components for CBFs include channelizers, tunable filters, resamplers, beamformers and cross-correlators. The following outlines the interface and verification of the functionality of oversampled polyphase filter banks (OSPFBs) as a representation of the signal processing modules used in CBFs.

Table 1. The bench marked evaluation times for 2^{16} samples and rms error compared to the direct evaluation.

Package	Eval-Time (s)	Norm-Error
NFFT.jl	0.017644	1.9675×10^{-9}
FINUFFT.jl	0.033506	1.2333×10^{-8}
NonuniformFFTs.jl	0.014122	1.9201×10^{-7}
FINUFFT*	0.561047	1.3823×10^{-9}

*MATLAB implementation

Channelizers segment the input spectrum into multiple channels of equal bandwidths [15]-Ch06, -Ch09. In Mid.CBF [2] and Low.CBF [3], two main types of channelizers are employed. The first type employs OSPFBs, which leverage oversampling to achieve aliasing suppression at arbitrary levels at the expense of higher computational complexity [15]-Ch09. The second type is based on the critically-sampled polyphase filter banks (CSPFBs) [15]-Ch06. In the Julia package, both these types have been implemented as configurable Julia functions.

The function that evaluates the OSPFB outputs requires the following configuration parameters; 1) the number of channels (N_c), 2) the commutator length (CL), 3) the rotation step (RSt) and 4) the filter coefficients ($\{h(n)\}$). In Julia, it is convenient to arrange all these parameters into an immutable structure and pass it to the function. Conversely, the function that evaluates the CSPFB outputs only two parameters; 1) the number of channels (N_c) and 2) the filter coefficients ($\{h(n)\}$). Also, Julia's multiple dispatch feature has been exploited to compose four optimized implementations of the OSPFB function that process either real-valued or complex-valued inputs with polyphase filters with either real-valued or complex-valued filter coefficients.

The OSPFB function has been verified with numerical simulations that employ test signals synthesized with collections of sinusoids with known magnitudes and phases. The objective is to compare the magnitude and the phase of the input sinusoid for a given frequency with the magnitude and phase of the corresponding sinusoid in the channel into which the input frequency falls. Note that the magnitude variation due to the passband ripple of the frequency response has to be compensated before the comparison. For demonstration, the spectra of the test signals are comprised of uniformly spaced sinusoids each having unit magnitude, with their phases distributed uniformly, as in Section 2. The real-valued test signals are generated using the Fast Fourier Transform (FFT) method. These test signals are processed with the OSPFB function configured with $N_c = 20, CL = 18$ and $RSt = 2$. In the top-row of Figure 1, the magnitude and phase spectra of the input that is segmented into 30 channels are indicated in red-solid lines, along with the resulting channels of the OSPFB function are indicated in blue-dashed lines.

The bottom-row of Figure 1 shows the difference between the input and output magnitude and phase spectra after cor-

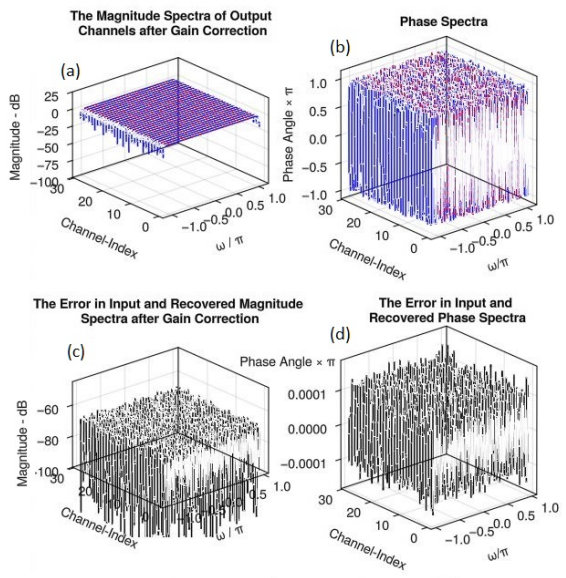

Figure 1. Top Row : The magnitude (a) and phase (b) spectra of the 30 output channels, after gain corrections (Blue : Dashed) and the corresponding input magnitude and phase spectra (Red : Solid). **Bottom Row** : Differences between segmented input and output magnitude (c) and phase (d) spectra.

recting for passband ripple. Note that to align the input and output phase spectra, the output sample sequences must be shifted to account for the propagation delay through the polyphase filters [15]-Ch06. The close agreements between input and output spectra for both magnitude and phase indicate the OSPFB function is correctly implemented.

Low-precision floating-point formats, i.e.Float16, BFloat16 and TensorFloat32, are being supported by hardware manufacturers such as Altera, AMD-Xilinx and NVIDIA in the latest FPGAs and GPUs [16]. Also, the novel floating-point number formats; 1) Positis [17] and 2) Takums [18], claim to 'beat' the performance floating-point with a fewer number of bits. Julia allows straight forward comparison of the accuracy achieved by the OSPFB function where both data and coefficients represented these floating-point formats. In Figure 2, the errors in magnitude and phase for the input and output signal spectra of the OSPFB function achieved with the Float16 (First Row), the Posit16 Second Row, the LinearTakum16 Third Row and the TensorFloat32 Forth Row, of Figure 2, respectively. Note that the variation in the magnitude and phase error for different channels reveals the *non-uniform* contribution of rounding errors in the FFT operation within the OSPFB function.

4 Discussion

It is realized that the Julia programming language is a good platform to develop and validate advanced signal processing systems for CBFs in modern radio telescopes. A comprehensive Julia package is being developed that consists of

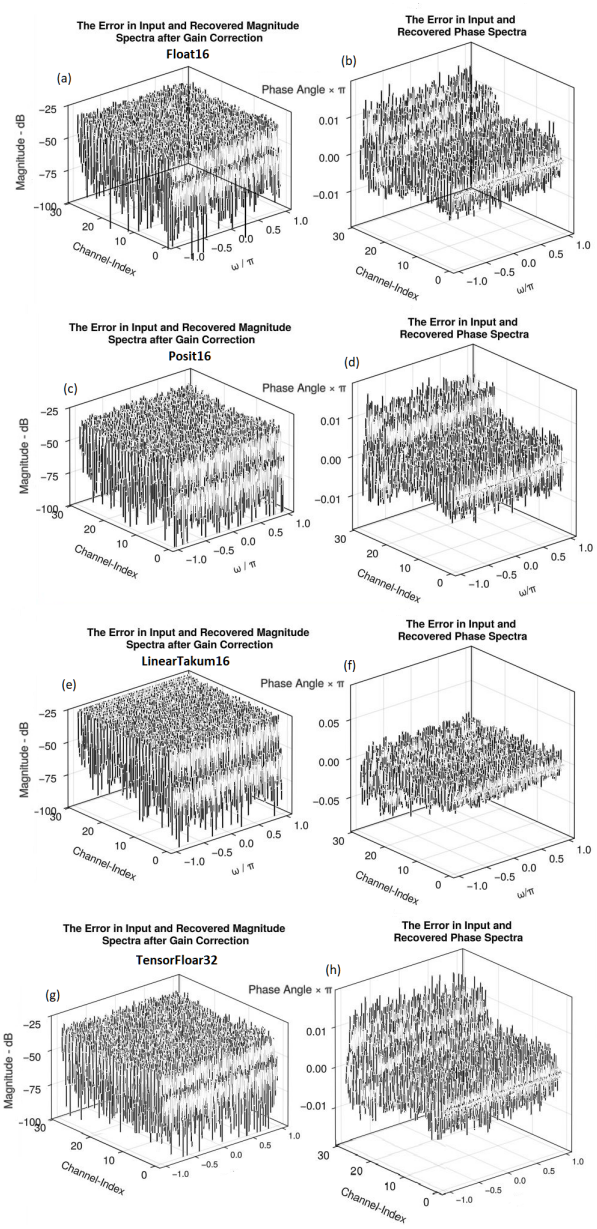

Figure 2. Differences between segmented input and output magnitude and phase spectra for OSPFB function evaluated with the Float16 **First Row**, the Posit16 [17] **Second Row**, the LinearTakum16 [18] **Third Row** and the TensorFloat32 **Forth Row**, respectively.

various Julia functions to efficiently perform key signal processing operations, including channelization, tunable filtering, resampling, beamforming, and cross-correlation. This Julia package also features specialized functions to generate test signals with time-varying delays that simulate the effects of Earth's rotation. Further, functions to detect RFI and flag affected data are being developed.

The increasing number of Julia packages enriches the tools available for data processing and visualization, providing the users with many options to optimize. For instance, at least three robust packages provide efficient im-

plementations of the non-uniform discrete Fourier transform (NUFFT), optimized for both CPU and GPU implementations while meeting different levels of accuracy and achieving different throughput. Julia's modularity enables the integration of functions that handle multiple data types with minimal changes, facilitating a straightforward way to evaluate performances. However, there is still an opportunity to enhance the speed and quality of Julia's plotting features.

References

[1] A. Richard Thompson, James M. Moran, and George W. Swenson Jr. *Interferometry and Synthesis in Radio Astronomy*. 3rd edition. New York, NY: Springer/Sci-Tech/Trade, Mar. 3, 2017.

[2] Thushara Gunaratne et al. "An end-to-end model for the correlator and beamformer of the Square Kilometer Array Mid Telescope". In: *Modeling, Systems Engineering, and Project Management for Astronomy IX*. Vol. 11450. SPIE, 2020. DOI: 10.1117/12.2559754.

[3] Simone Chiarucci and Giovanni Comoretto. "An end-to-end model for the correlator and beamformer of the Square Kilometer Array Low Frequency Aperture Array". In: *Modeling, Systems Engineering, and Project Management for Astronomy IX*. Vol. 11450. SPIE, 2020. DOI: 10.1117/12.2560838.

[4] Thushara Gunaratne and Mitchell Mickaliger. "An End-to-End Verification of Pulsar Search Signal Chain for Square Kilometre Array Mid Central Signal Processor". In: *2021 XXXIVth General Assembly and Scientific Symposium of the International Union of Radio Science (URSI GASS)*. DOI: 10.23919/URSIGASS51995.2021.9560600.

[5] Thushara Gunaratne, Brent Carlson, and Gianni Comoretto. "Novel RFI Mitigation Methods in the Square Kilometre Array 1 Mid Correlator Beamformer". In: *J. Astron. Instrum.* 08.1 (Mar. 2019). DOI: 10.1142/S2251171719400117.

[6] Thushara Gunaratne and Adriaan Peens-Hough. "Modeling of RFI and RFI Mitigation Techniques in SKA1 Mid Telescope". In: 2019 RFI Workshop - Coexisting with Radio Frequency Interference (RFI). Toulouse, France, Sept. 23, 2019. DOI: 10.23919/RFI48793.2019.9111712.

[7] Matthijs Cox. *How to solve the two language problem?* The Scientific Coder. May 8, 2023. URL: https://scientificcoder.com/how-to-solve-the-two-language-problem (visited on 02/11/2025).

[8] Jeff Bezanson et al. "Julia: A Fresh Approach to Numerical Computing". In: *SIAM Review* 59.1 (2017), pp. 65–98. DOI: 10.1137/141000671.

[9] Gary J. Hovey and Federico Di Vruno. "A Framework for RFI Simulation and Performance Verification". In: 2019 RFI Workshop - Coexisting with Radio Frequency Interference (RFI). Toulouse, France, Sept. 23, 2019. DOI: 10.23919/RFI48793.2019.9111822.

[10] Thushara Kanchana Gunaratne. "Generation of Coherent Signals for the Verification of Signal Processing Algorithms in Radio Astronomy". In: *2019 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. 2019. DOI: 10.1109/PACRIM47961.2019.8985071.

[11] Alexander H. Barnett, Jeremy Magland, and Ludvig af Klinteberg. "A Parallel Nonuniform Fast Fourier Transform Library Based on an "Exponential of Semicircle" Kernel". In: *SIAM Journal on Scientific Computing* 41.5 (2019), pp. C479–C504. DOI: 10.1137/18M120885X.

[12] Tobias Knopp, Marija Boberg, and Mirco Grosser. "NFFT.jl: Generic and Fast Julia Implementation of the Nonequidistant Fast Fourier Transform". In: *SIAM Journal on Scientific Computing* 45.3 (2023), pp. C179–C205. DOI: 10.1137/22M1510935.

[13] Ludvig af Klinteberg. *FINUFFT.jl*. https://github.com/ludvigak/FINUFFT.jl. Version 3.3.1. Dec. 2, 2024.

[14] Juan Ignacio Polanco. *NonuniformFFTs.jl*. https://github.com/jipolanco/NonuniformFFTs.jl. Version 0.7.1. Feb. 18, 2025.

[15] fredric j. harris. *Multirate Signal Processing for Communication Systems*. Ed-2. River Publishers, Mar. 2021.

[16] Thushara Kanchana Gunaratne. "Evaluation of the Use of Low Precision Floating-Point Arithmetic for Applications in Radio Astronomy". In: *Next Generation Arithmetic - CoNGA 2023*. Springer Nature Switzerland, 2023, pp. 155–170. DOI: 10.1007/978-3-031-32180-1_10.

[17] Gustafson, J. L. and Yonemoto, I. "Beating Floating Point at its Own Game: Posit Arithmetic". In: *Supercomputing Frontiers and Innovations* 4.2 (June 2017), pp. 71–86. DOI: 10.14529/jsfi170206.

[18] Laslo Hunhold. "Beating Posits at Their Own Game: Takum Arithmetic". In: *Next Generation Arithmetic CoNGA 2024*. Springer Nature Switzerland, pp. 1–51. DOI: 10.1007/978-3-031-72709-2_1.