High-Performance Multi-Precision Tool for Floating-Point Computations

Laszlo LEITOLD [1], Monther ALRWASHDEH [2], Zsolt KOLLAR [2]

[1] Dept. of Mechatronics, Optics and Mechanical Engineering Informatics, Faculty of Mechanical Engineering
[2] Dept. of Artificial Intelligence and Systems Engineering, Faculty of Electrical Engineering and Informatics
Budapest University of Technology and Economics, Műegyetem rkp. 3., H-1111, Budapest, Hungary

leitold.laszlo@mogi.bme.hu

Submitted July 8, 2025 / Accepted September 22, 2025 / Online first October 24, 2025

Abstract. *This paper presents a MATLAB toolbox for multiple-precision arithmetic, enabling high-precision numerical computations beyond the standard double-precision format. The development step and functionality of the toolbox are described, and how it integrates seamlessly with the MATLAB computational ecosystem, offering a user-friendly interface. The performance enhancement and usability of the toolbox are demonstrated through four use cases where floating-point arithmetic becomes an issue. These benchmark results illustrate the toolbox's computational accuracy and performance, highlighting its ability to mitigate numerical instabilities and roundoff errors in sensitive computations. The toolbox empowers researchers and engineers to implement more reliable models, test advanced algorithms, and validate system designs and a diverse set of operations for applications that require enhanced precision, such as numerical analysis, scientific computing, and applied mathematics.*

Keywords

MATLAB, floating-point, GMP, MPFR, multiple-precision, quantization, QNP

1. Introduction

Digital computers represent analog values as numbers that approximate the original value, with range and accuracy limited by architectural constraints. Fixed-Point (FI) formats offer simplicity and predictable precision, while Floating-Point (FL) formats provide extended range and flexibility at the cost of greater complexity and potential rounding errors.

While the standard FL formats (e.g. single and double) are sufficient for many computational needs, they are prone to fail in applications requiring precision beyond the capabilities of standard number representations. These limitations become particularly problematic in numerical analysis, scientific computing, and applied mathematics, where even minor inaccuracies can propagate and lead to significant errors. Multiple-precision arithmetic offers a compelling alternative by enabling computations with arbitrary precision, addressing numerical instabilities, and minimizing roundoff errors.

To name examples, the calculation precision for the Fast Fourier Transform (FFT), especially in a Field-Programmable Gate Array (FPGA) environment, needs a thorough analysis and adjustment so that the lowest precision can be chosen that still meets the system requirements regarding quantization noise [1]. Higher precision is required in thermal analysis [2] where the time constants that need to be identified may vary on a large scale. Furthermore, in neural networks, reducing the FL precision towards smaller representations is a desired feature [3], which requires extensive simulations and careful analysis.

1.1 State-of-the-art Techniques

If a researcher has to use a precision other than the built-in FL precision, the following possibilities are currently available as options:

- Ball/interval arithmetic (rigorous bounds): Represent numbers as midpoint ± radius (balls), provable error bounds – powerful for verified numeric and difficult problems (roots, ODEs). FLINT [4] is the leading, widely used C library for ball arithmetic with rich special-function support.
- Fixed extended formats (double-double, quad-double): Libraries like QD [5] provide fast software emulation of 2× or 4× double precision balancing between the extra overhead and the gained precision. [6]
- Arbitrary precision libraries like GNU Multiple Precision Arithmetic Library (GMP) [7] and the Multiple Precision FL Reliable Library (MPFR) [8] are the standard solutions for FL representation with even higher precision.

Furthermore, currently the MATLAB programming language does not have a built-in format for higher precision than the IEEE 754 binary64. Although with the Symbolic

DOI: 10.13164/re.2025.0583

Math Toolbox [9], it is possible to use arbitrary precision. This toolbox has drawbacks, including high cost and slower performance, alongside limited support for specific functions with the variable-precision arithmetic (`vpa`) class.

The Advanpix Toolbox [10] is a comprehensive, fast library designed for multi-precision computing in MATLAB. This toolbox offers a robust suite of features that enable users to perform computations with accuracy beyond MATLAB's standard double precision. However, despite its technical advantages and performance improvements, the premium price can be a significant barrier, especially for smaller institutions or projects operating under tight budgets.

The Multiple Precision Toolbox for MATLAB [11] offers an open-source solution based on the MPFR library, allowing MATLAB users to perform high-precision arithmetic without licensing fees. However, it has a notable drawback: calculations are slow, which limits its effectiveness in large-scale or time-sensitive applications. Our solution builds on this library with the primary goal of improving execution speed.

The presented MATLAB toolbox is designed for multiple-precision arithmetic to bridge the gap between high-precision needs and user-friendly computational workflows. The toolbox integrates seamlessly with MATLAB's ecosystem, allowing researchers and practitioners to extend their numerical capabilities without departing from MATLAB's familiar environment. It supports arbitrary precision for real and complex numbers and matrix operations while offering precision control and automatic precision propagation features. The toolbox can be downloaded from GitHub [12]).

This paper, which is an extended version of the conference paper [13], explores the toolbox's design principles, implementation, and benchmarking results, highlighting its computational accuracy and performance. By demonstrating its effectiveness in mitigating numerical instabilities and enabling extreme numerical accuracy, this toolbox can be an invaluable resource for the scientific and engineering communities. Multiple-precision arithmetic is increasingly important in numerical simulations and algorithm development for signal processing, communications, neural networks, and electromagnetic field modeling, where high numerical accuracy and stability are critical. The contribution of this paper compared to the State-Of-The-Art (SOTA) can be summarized as follows:

- The proposed architecture takes full advantage of the MATLAB - C interface, leading to fast calculations.
- Full functionality with different basic functions of MATLAB has been extended and overloaded to be used with MP objects.
- Demonstration of the MP Toolbox on practical use cases, including FIR filtering and FFT computations, validating accuracy against theoretical predictions.
- Significant improvements in computational efficiency compared to previous implementations.

The paper is organized as follows: after introducing the current FL standard, in Sec. 2, the structure of the Multiple Precision (MP) Toolbox is presented, detailing the used libraries and custom classes. A brief description of the FL quantization noise is presented in Sec. 3. Section 4 presents four examples, namely Hilbert Matrix, Quantization Noise in the FFT, Roundoff Noise in Finite Impulse Response (FIR) Digital Filter, and Foster-Cauer Resistor-Capacitor (RC) network conversion, which demonstrate the merits of the MP Toolbox. Finally, the paper is summarized in Sec. 5.

1.2 Floating-Point Standards

The IEEE 754 standard [14–16] defines a widely adopted set of representations and operations for FL arithmetic, serving as the foundational model for numerical computation in modern hardware and software systems. It specifies multiple binary formats—namely binary16 (half precision), binary32 (single precision), binary64 (double precision), binary128 (quadruple precision), and binary256 (octuple precision)—as well as decimal formats such as decimal32, decimal64, and decimal128. These formats provide standardized representations for real numbers, enabling consistent behavior across architectures, compilers, and programming environments.

First introduced in 1985 and subsequently revised in 2008 and 2019, the IEEE 754 standard encompasses a comprehensive definition of FL behavior. It specifies how to encode finite numbers, infinities, signed zeros, and special values like Not-a-Number (NaN). The standard also defines arithmetic operations, including addition, subtraction, multiplication, division, Fused Multiply-Add (FMA), and square root, along with well-defined rounding modes and exception handling mechanisms. These elements are crucial for ensuring that FL computations are reliable, reproducible, and portable across different platforms.

A central feature of the standard is its normalized representation of real numbers, expressed in the form $(-1)^s \times 1p \times 2^e$, where s is the sign bit, p is the fraction (or significand), and e is the biased exponent. This format allows for compact and efficient encoding of a wide dynamic range. For example, binary32 described in Fig. 1 uses 1 sign bit, 8 exponent bits, and 23 fraction bits, while binary64 extends this to 1 sign bit, 11 exponent bits, and 52 fraction bits. The most recent revision, IEEE 754-2019 [16], introduced several important refinements, including more precise rules for directed rounding, better-defined exception signaling, and support for extended and hybrid formats to improve interoperability across heterogeneous computing platforms. Table 1 summarizes the key parameters of each standard format, including exponent width, significand precision, and encoded range.

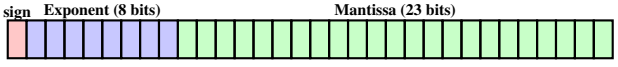

Fig. 1. Bit-Level representation of IEEE 754 single-precision FL number (binary32).

Format	Total bits*	Exponent bits	Mantissa bits	Exponent range	Range
binary16 (half)	16	5	10	−14 to +15	6.10×10^{-5} to 6.55×10^{4}
binary32 (single)	32	8	23	−126 to +127	1.18×10^{-38} to 3.40×10^{38}
binary64 (double)	64	11	52	−1022 to +1023	2.23×10^{-308} to 1.80×10^{308}
binary128 (quadruple)	128	15	112	−16382 to +16383	3.36×10^{-4932} to 1.19×10^{4932}
binary256 (octuple)	256	19	236	−262142 to +262143	$\approx 10^{-78899}$ to $\approx 10^{78899}$
decimal32	32	8	23	−95 to +96	$\approx 10^{-95}$ to $\approx 10^{96}$
decimal64	64	11	52	−383 to +384	$\approx 10^{-383}$ to $\approx 10^{384}$
decimal128	128	15	112	−6143 to +6144	$\approx 10^{-6143}$ to $\approx 10^{6144}$

Tab. 1. IEEE 754 FL Formats. (* Sign bit not shown explicitly)

The influence of IEEE 754 extends well beyond general-purpose computing. It is implemented in hardware such as central processing units (CPUs), graphics processing units (GPUs), and digital signal processors (DSPs), providing high-performance support for standardized FL operations. Programming languages such as C, C++, Java, Python (via NumPy), and Fortran offer IEEE-compliant numerical types and libraries, further reinforcing the portability and consistency of FL computations. From computational fluid dynamics and quantum chemistry to machine learning and financial modeling, IEEE 754 remains an essential standard that underpins the correctness, reproducibility, and interoperability of scientific and engineering computations.

While IEEE 754 provides a rigorous and portable framework, it does not cover all practical scenarios. For example, domain-specific applications, particularly in machine learning and signal processing, often employ non-standard formats optimized for speed and memory efficiency. One prominent example is the Brain Floating-point (BFloat16) format [17], which retains the 8-bit exponent of IEEE binary32 but reduces the significand to 7 bits. This design considerably lowers hardware complexity while preserving the dynamic range necessary for deep learning workloads, albeit at the cost of reduced precision.

Other custom formats, such as Google's TensorFloat-32 (TF32) and NVIDIA's Floating-Point 8 (FP8) variants, are gaining traction for similar reasons. These variants often deviate from strict IEEE compliance but are tailored for high-throughput, approximate computation environments where exact numerical fidelity is secondary to performance.

On the other end of the spectrum, with the extended dynamic range and precision offered by the binary256 format, specialized domains exist where even 256-bit FL arithmetic proves inadequate. One such area is arbitrary-precision arithmetic used in computational number theory, cryptography, and symbolic mathematics, where exact results or hundreds to thousands of digits of precision are often required. For example, in verifying mathematical constants (such as π or e) to billions of digits or primality testing of extremely large numbers, the limitations of fixed-width formats like binary256 become apparent. Similarly, high-precision simulations in theoretical physics, such as lattice Quantum ChromoDynamics (QCD), gravitational wave modeling, and perturbative quantum field theory, may require precision far beyond what binary256 can reliably support, particularly when dealing with ill-conditioned matrices or subtraction-sensitive operations. In such contexts, software-based arbitrary-precision libraries (e.g., GNU Multiple Precision Arithmetic Library (GMP), Multiple Precision FL Reliable Library (MPFR), or Arbitrary-Precision Ball Arithmetic Library (ARB)) are typically employed, offering user-defined precision at the cost of significantly reduced computational performance.

Taken together, IEEE 754 and its alternatives illustrate the spectrum of FL representations from rigorously standardized to task-optimized, underpinning contemporary computational science. The coexistence of these approaches reflects the evolving demands of scientific and engineering applications in both theory and practice.

1.3 Development of the MP Toolbox

Our research group first used and amended the Multiple Precision Toolbox for MATLAB for a thermal characterization application. Later, the communication layer between the MATLAB and MATLAB Executable (MEX) functions was replaced with a pointer-based solution instead of the original string-based one to increase performance. In this structure, the MATLAB class only stores the pointers for the multiple precision numbers. The numbers are stored in the C/C++ low-level layer alongside the data manipulation/arithmetic functions. Thus, the original String $\leftrightarrow$ MPFR conversation at every function call becomes unnecessary, speeding up the execution. As the latest improvement, we started using the MPLAPACK library [18] to take advantage of the computation efficiency of the Linear Algebra PACKage (LAPACK) functions.

2. Structure

2.1 Original C / C++ Libraries

- GMP Library [7]: It is an open-source, C library for high-performance arbitrary-precision arithmetic on integers, rational numbers, and FL numbers. GMP has evolved through decades of optimizations to support modular arithmetic, cryptographic computations, and large-scale numerical simulations, offering precision limited only by available memory and compatibility across various platforms. It is integrated into many other languages and libraries, making it an essential foundational tool for numerical computations.

- MPFR Library [8]: It is an open-source library, designed for precise FL computations with well-defined, correct rounding in compliance with the IEEE 754 standard. It ensures deterministic results across platforms, supporting rigorous error bounds, and includes special values (NaN, Inf) and directed rounding modes.

- MPLAPACK: The open-source multiple-precision linear algebra library [18] is designed to mirror the functionality of the LAPACK Fortran package while extending support for arbitrary precision arithmetic through libraries like GMP and MPFR. Its implementation enables highly accurate numerical computations for dense matrix operations, such as solving linear equations and eigenvalue problems, catering to scientific and engineering applications requiring precision beyond standard double-precision formats.

2.2 MATLAB Custom Classes

Custom MATLAB classes provide a structured and object-oriented framework for encapsulating data and defining bespoke behaviors beyond the default capabilities offered by MathWorks. This feature allows developers to define new data types, overload standard operators, control access to internal fields, and implement specialized methods, all while maintaining syntactic consistency with native MATLAB types. These classes are beneficial for extending MATLAB's functionality in contexts where specialized numerical behavior, data abstraction, or interoperability is required.

We leveraged MATLAB's class system to implement a custom numerical type supporting multiple-precision FL arithmetic. The custom class wraps arbitrary-precision numerical libraries and provides overloaded arithmetic, relational, and utility operations. The interface was designed to mimic native MATLAB behavior as closely as possible, enabling users to write scripts seamlessly using extended precision with minimal syntactic overhead. The custom class ensures compatibility with scalar and matrix expressions by overloading appropriate methods – member functions of the class – such as `subsref`, `subsasgn`, `plus`, `times`, and `disp`.

2.3 MATLAB MEX Interface

The MATLAB MEX interface for C++ allows developers to compile source code into dynamically loadable shared libraries, linking with the MEX API to invoke the resulting functions directly from MATLAB through the standardized mexFunction entry point. By exposing MATLAB's internal data structures (e.g. mxArray), memory management, and runtime environment, the MEX interface makes it possible to create high-performance, low-level implementations for specialized functionality while seamlessly passing data to and from MATLAB. The structure is illustrated in Fig. 2.

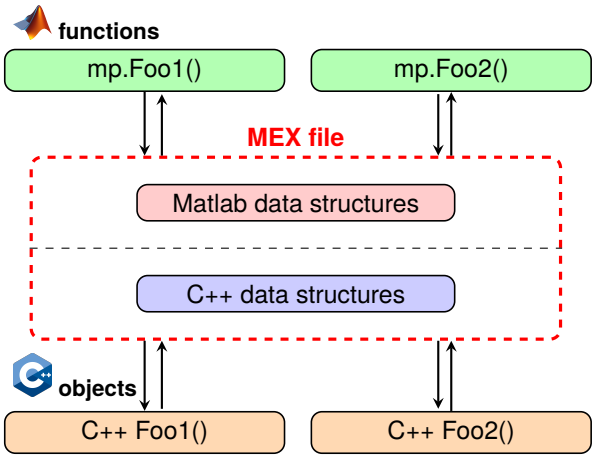

Fig. 2. General MATLAB External Language Interface for C/C++.

3. Floating-point Quantization Noise

Certain mathematical and technical details must be presented to explain the toolbox's use cases more effectively. In this part, the theory for FL quantization is briefly described.

The FL quantizer can be modeled exactly as a cascade of a nonlinear function (compressor) followed by a uniform quantizer (FI quantizer, $\mathrm{Q_{FI}}$) and an inverse nonlinear function (expander) as explained in Fig. 3.

The input-output characteristic for the compressor and expander can be found in [1]. The quantization noise of the FI and FL quantizers is calculated, respectively, by [1]

$$v = y - y_\mathrm{q}, \tag{1}$$

$$e_\mathrm{q} = x - x_\mathrm{q}. \tag{2}$$

Assuming that the noise v of the FI quantizer is small compared to y. Then v is a small increment to y. This causes an increment e_q at the expander output. Accordingly [1],

$$e_\mathrm{q} = v \left(\frac{\mathrm{d}x_\mathrm{q}}{\mathrm{d}y_\mathrm{q}} \right)_y. \tag{3}$$

The mean square of the quantization noise is given by [1] as

$$E\{e_\mathrm{q}^2\} \approx 0.180 \cdot 2^{-2p} \cdot E\{x^2\}. \tag{4}$$

The variance of the quantization noise is equal to [1]

$$\sigma_\mathrm{FL}^2 = E\{e_\mathrm{q}^2\} - (\mu_\mathrm{e})^2 \approx 0.180 \cdot 2^{-2p} \cdot E\{x^2\} \tag{5}$$

where μ_e is the expected value and p is the number of mantissa bits in the FL quantizer.

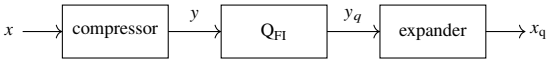

Fig. 3. A model for the FL quantizer [1].

4. Use Cases

This section presents four cases to demonstrate the toolbox's effectiveness and verify the results. The simulations were carried out in MATLAB R2022b with the Roundoff Toolbox [1] and the showcased MP Toolbox on a workstation equipped with an AMD Ryzen 9 7950X processor and 64 GB of RAM. First, the Hilbert Matrix is analyzed, then the FL quantization in FFT and FIR filter algorithms is verified using the toolbox, and finally, the effectiveness of the toolbox is demonstrated through Foster-Cauer network conversion.

4.1 Hilbert Matrix

The Hilbert matrix [19] is a classical example in numerical linear algebra, defined by

$$\mathbf{H}_{ij} = \frac{1}{i+j-1} \quad \text{with } 1 \leq i, j \leq n. \tag{6}$$

Despite its simple and well-structured appearance, the Hilbert matrix is notoriously ill-conditioned, particularly for large n. This makes it a canonical example when analyzing the stability and accuracy of numerical algorithms for matrix inversion.

One way to illustrate the effects of ill-conditioning is to examine the deviation of the product $\mathbf{H}^{-1}\mathbf{H}$ from the identity matrix $\mathbf{I}$. In exact arithmetic, this product would yield $\mathbf{I}$; however, due to rounding errors and the high condition number of the Hilbert matrix, the computed inverse $\widetilde{\mathbf{H}}^{-1}$ typically results in a product $\widetilde{\mathbf{H}}^{-1}\mathbf{H}$ that significantly deviates from $\mathbf{I}$, especially in the off-diagonal entries. This deviation grows rapidly with the matrix dimension and serves as a compelling demonstration of the limitations of finite-precision arithmetic.

Figure 4 shows a representative example of $\widetilde{\mathbf{H}}^{-1}\mathbf{H} - \mathbf{I}$ ($n = 14$) for various bit length. The plot illustrates how the deviation decreases for higher bit length, highlighting the sensitivity of matrix inversion to numerical precision. Such examples are commonly used in numerical analysis coursework and software validation to emphasize the importance of algorithmic robustness and conditioning in practical computations.

4.2 FFT

The Discrete Fourier Transform (DFT) of a sequence $x(n)$, where $n = 0, \ldots, N-1$ is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n)\exp(-\mathrm{j}2\pi kn/N) \tag{7}$$

where $k = 0, \ldots, N-1$. FFT is an algorithm used to calculate the DFT efficiently by reducing the computational complexity from $O(N^2)$ to $O(N \log_2 N)$. One commonly used way to calculate the DFT is the Decimation-In-Time FFT (DIT FFT). This algorithm's calculations are based on the butterfly structure [20–22] shown in Fig. 5.

Implementing the FFT using hardware with a restricted number of bits and finite precision arithmetical units introduces roundoff error (quantization noise) at the output of the FFT. In the case of FL quantization, quantization is needed after each multiplication or addition. This quantization noise can be found using (5).

According to Fig. 5, after the multiplication of $x(j)$ and W_8^k, quantization is needed. Also, quantization is required after the sum of $x(i)$ and $x(j)W_8^k$. The same procedure is applied to the lower part of the butterfly ($X(j)$). Furthermore, when the value of W_N^k equals to ± 1 or $\pm j$, then no quantization is needed after the multiplication of W_N^k and $x(j)$.

The Quantization Noise Power (QNP) at the N-FFT output was derived in [1]. The result is given by:

$$\mathrm{QNP}_{\mathrm{FFT,FL}} = 0.42 \cdot \log_2(N) \cdot \sigma_x^2 \cdot 2^{-2p} \tag{8}$$

where σ_x^2 is the variance of the signal at the FFT output. It is clear from (8) that $\mathrm{QNP}_{\mathrm{FFT,FL}}$ depends on the power of the input signal. For the case $x(n)$ is uniformally distributed in $[-1, 1]$, the value of $\sigma_x^2 = N/3$.

To validate the results in (8), the simulation was performed 200 times, and the results were averaged. Figure 6 shows that the theoretical and simulation results are approximately the same. The performance of the new MP Toolbox compared to the toolbox in [1] is shown in Fig. 7. It can be seen that as N increases, the speed gain becomes substantial, reaching more than 9 times faster execution speed.

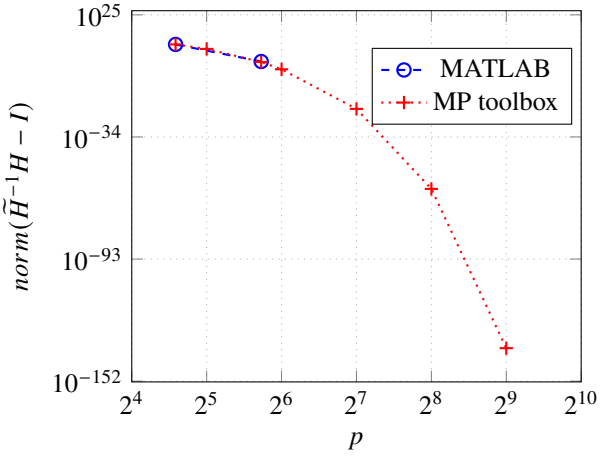

Fig. 4. Normed difference of the Hilbert matrix ($n = 14$) and its inverse $\widetilde{\mathbf{H}}^{-1}\mathbf{H}$ compared to the identity matrix $\mathbf{I}$ for different values of p.

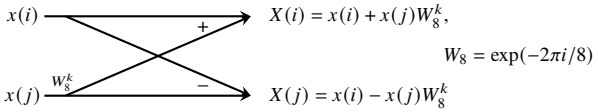

Fig. 5. General form of the DIT butterfly. In this case, $N = 8$.

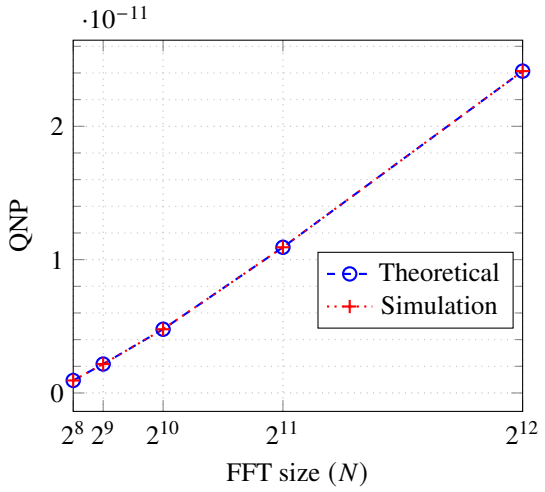

Fig. 6. Theoretical and simulated QNP at the output of the FFT for different values of N, $p = 24$ bits.

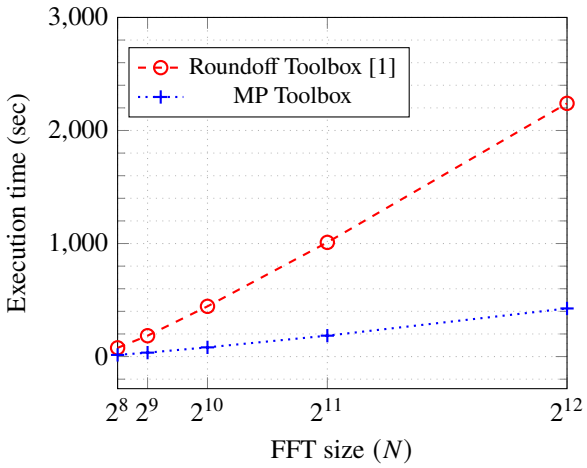

Fig. 7. Performance comparison of DIT-FFT implementations across different MATLAB toolboxes. Execution time is measured for varying input sizes N.

4.3 Roundoff Noise in FIR Digital Filter

The block diagram of the FIR filter with order M is shown in Fig. 8. If roundoff is performed after each multiplication and after the final sum, then total QNP at the output of the filter is given by [1]

$$\mathrm{QNP}_{\mathrm{FIR}} = 0.36 \cdot 2^{-2p} \cdot S_{\mathrm{m}} \cdot \sigma_{\mathrm{x}}^2 \tag{9}$$

where $S_{\mathrm{m}} = \sum_{i=0}^{M-1} |h_i|^2$ is the sum of squares of the filter coefficients and σ_{x}^2 is the power of the input signal. To validate the result in (9), a uniformly distributed signal between $[-1, 1]$ is used as the input signal. The FIR lowpass filter is designed in MATLAB using the command `fir1(100, 0.35)`. The theoretical and simulation results using the MP toolbox are shown in Fig. 9 for different values of p. The results show excellent agreement between the theoretical formula in (9) and the simulation results, confirming the accuracy of the MP Toolbox in modeling quantization effects.

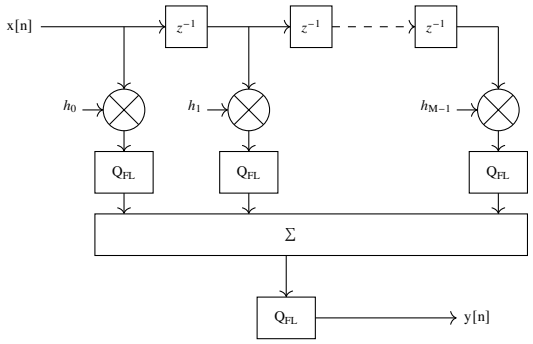

Fig. 8. Block diagram of the FIR filter with roundoff after each multiplication and roundoff after the final sum.

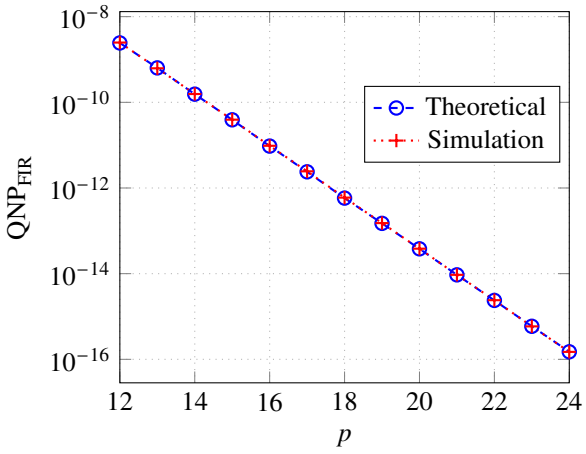

Fig. 9. Theoretical and simulated QNP at the output of the FIR for different values of p.

4.4 Foster - Cauer Network Conversion

A canonical Foster RC-network contains the serial connection of n parallel RC-elements; hence, its impedance in the frequency domain can be written as [23]

$$Z(s) = \sum_{i=1}^{n} \frac{R_{i\mathrm{F}}}{1 + sR_{i\mathrm{F}}C_{i\mathrm{F}}} \tag{10}$$

while an equivalent Cauer RC-network is a ladder whose impedance is the continued fraction [23]

$$Z(s) = \cfrac{1}{sC_{1\mathrm{C}} + \cfrac{1}{R_{1\mathrm{C}} + \cfrac{1}{sC_{2\mathrm{C}} + \ldots}}} \tag{11}$$

where $R_{i\mathrm{F}}, C_{i\mathrm{F}}, R_{i\mathrm{C}}, C_{i\mathrm{C}}$ denote the i^{th} resistance or capacitance in case of a Foster- or Cauer-network, respectively.

Conversion between different forms of canonical linear circuits has well-known uses, e.g. in filter design. However, specifically the Foster-Cauer conversion of RC-networks is an important part of creating compact thermal models of electronic systems, the reason being that based on measurement results, the Foster-form is obtainable, but the Cauer-ladder is physically more descriptive and allows for the plotting of cumulative structure functions [2], [24].

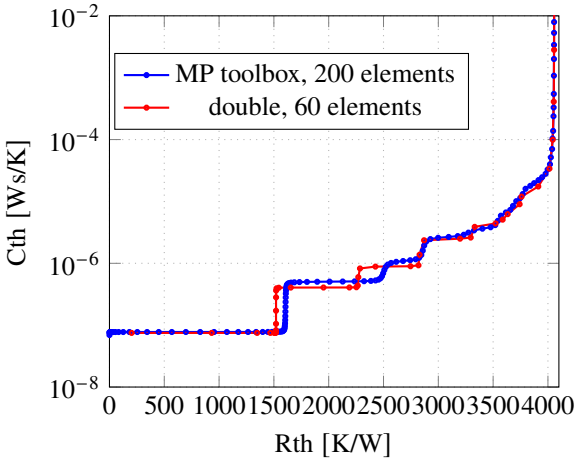

Fig. 10. Cumulative structure function of a system using the MP toolbox and double representation in case of the golden reference.

In these cases, the continuous thermal flow is modeled by a high-order system, whose time constants can span even 8 orders of magnitude; for precise modeling, numerous points are needed per each decade, and the Foster-Cauer conversion requires as many polynomial divisions as the order of the obtained Foster- and required Cauer-network.

With the large number of polynomial divisions, conventional standards, e.g., binary double, cannot produce satisfying results due to the span of the values and rounding errors. However, the MP Toolbox overcomes this problem, as shown in Fig. 10, which shows the performance of double precision vs. the MP Toolbox results for the cumulative R-C structure function of a so-called golden reference, physically containing a fourth-order RC circuit used for benchmarking thermal measurement systems. Using a double representation, a system with more than 60 time constants is not obtainable because the polynomial divisions cannot be computed successfully. At the same time, the MP Toolbox tackles the calculation of a 200^{th}-order system, and the difference of the two curves shows that a higher order model provides a smoother structure function.

5. Conclusion

This paper presented a new, open-source multiple-precision toolbox (available for download on GitHub [12]) that enhances MATLAB's capabilities beyond double precision. It illustrates the feasibility of integrating high-precision arithmetic into MATLAB and is a foundation for developing computational tools for research with different precision requirements. The development roadmap includes implementing all built-in MATLAB arithmetic and trigonometric functions, ensuring that the full potential of the toolbox is realized. This advancement promises to offer MATLAB users a robust, high-precision tool for addressing complex numerical challenges in various research and engineering applications. The capabilities of the toolbox have been demonstrated and verified across different use cases, including four practical examples. These showcases highlight the toolbox's effectiveness, flexibility, and broad applicability in handling real-world numerical problems with high-precision demands. In real-world scenarios, the MP Toolbox can be employed in applications such as signal processing, wireless communications, and electromagnetic field modeling, where maintaining high numerical accuracy and stability is essential.

Acknowledgments

The research presented in this paper is partially funded by the National Research Development and Innovation Office (NKFIH) through the OTKA Grant (K 142845), the János Bolyai Research Grant (BO/00042/23/6) of the Hungarian Academy of Sciences.

References

[1] WIDROW, B., KOLLÁR, I. *Quantization Noise: Roundoff Error in Digital Computation, Signal processing, Control, and Communications.* 1st ed. Cambridge (UK): Cambridge University Press, 2008. ISBN: 9780511754661

[2] JANICKI, M., NAPIERALSKI, A. Considerations on electronic system compact thermal models in the form of RC ladders. In *Proceedings of the IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*. Polyana (Ukraine). 2019, p. 1–4. DOI: 10.1109/CADSM.2019.8779274

[3] AL-RIKABI, H., RENCZES, B. Floating-point quantization analysis of multi-layer perceptron artificial neural networks. *Journal of Signal Processing Systems*, 2024, vol. 96, no. 4–5, p. 301–312. DOI: 10.1007/s11265-024-01911-0

[4] *FLINT: Fast Library for Number Theory.* [Online] Cited 2025-09-19. Available at: https://flintlib.org/

[5] HIDA, Y., LI, X. S., BAILEY, D. H. *Library for Double-Double and Quad-Double Arithmetic.* [Online] Cited 2008-05-08, p. 1–24. Available at: https://www.davidhbailey.com/dhbpapers/qd.pdf

[6] AHLSTRÖM, Å. *High-Performance Computing with Quad-Precision on Two GPUs.* Akademi University, Master's Thesis, 2024. [Online] Cited 2025-09-19. Available at: https://www.doria.fi/bitstream/handle/10024/190456/ahlstrom_anders.pdf

[7] FREE SOFTWARE FOUNDATION. *The GNU Multiple Precision Arithmetic Library*. [Online] Cited 2025-04-14. Available at: https://gmplib.org/

[8] FOUSSE, L., HANROT, G., LEFÉVRE, V., et al. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software*, 2007, vol. 33, no. 2, p. 1–13. DOI: 10.1145/1236463.1236468

[9] THE MATHWORKS, INC. *Symbolic Math Toolbox MATLAB Help Center*. [Online] Cited 2025-04-14. Available at: https://www.mathworks.com/help/symbolic/index.html

[10] ADVANPIX LLC. *Multiprecision Computing Toolbox for MATLAB*. [Online] Cited 2025-04-14. Available at: https://www.advanpix.com/

[11] BARROWES, B. *Multiple Precision Toolbox for MATLAB v2.0.* [Online] Cited 2025-04-14. Available at: https://www.mathworks.com/matlabcentral/fileexchange/6446-multiple-precision-toolbox-for-matlab

[12] LEITOLD, L., CSEPPENTŐ, B., KOLLÁR, Z. *MP Toolbox*. [Online] Cited 2025-04-14. Available at: https://github.com/LLeitold/MP-Toolbox

[13] LEITOLD, L., CSEPPENTŐ, B., KOLLÁR, Z. Development of an efficient and fast computational tool for multi-precision floating point numbers. In *Proceedings of the 35th International Conference Radioelektronika (RADIOELEKTRONIKA)*. Brno (Czech Republic). 2025, p. 1–5. DOI: 10.1109/RADIOELEKTRONIKA65656.2025.11008412

[14] ANSI/IEEE Std 754-1985. *IEEE Standard for Binary Floating-Point Arithmetic*. 1985, p. 1–20. DOI: 10.1109/IEEESTD.1985.82928

[15] IEEE Std 754-2008. *IEEE Standard for Floating-Point Arithmetic*. 2008, p. 1–70. DOI: 10.1109/IEEESTD.2008.4610935

[16] IEEE Std 754-2019 (Revision of IEEE 754-2008). *IEEE Standard for Floating-Point Arithmetic*. 2019, p. 1–84. DOI: 10.1109/IEEESTD.2019.8766229

[17] WANG, S., KANWAR, P. *BFloat16: The Secret to High Performance on Cloud TPUs*. [Online] Cited 2025-04-14. Available at: https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus

[18] NAKATA, M. *MPLAPACK version 2.0.1 (User Manual)*. 136 pages. [Online] Cited 2025-04-14. Available at: https://arxiv.org/pdf/2109.13406. DOI: 10.48550/arXiv.2109.13406

[19] TODD, J. Computational problems concerning the Hilbert matrix. *Journal of Research of the National Bureau of Standards*, 1961, vol. 65, no. 1, p. 19–22. DOI: 10.6028/jres.065B.005

[20] NIHARIKA, R., IMTIYAZ AHMED, B., NIRANJAN, L., et al. Custom precision method of floating-point operations of FFT processing for optimized area and delay performance. In *Proceedings of the International Conference on Intelligent Innovations in Engineering and Technology (ICIIET)*. Coimbatore (India), 2022, p. 171–177. DOI: 10.1109/ICIIET55458.2022.9967519

[21] ZHAO, X., YAN, C., WANG, C., et al. Design of approximate floating-point FFT with mantissa bit-width adjustment algorithm. In *Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. Shenzhen (China), 2022, p. 265–269. DOI: 10.1109/APCCAS55924.2022.10090295

[22] TEYMOURZADEH, R., ABIGO, M., HONG, M. Characteristic analysis of 1024-point quantized Radix-2 FFT/IFFT processor. In *Proceedings of the 10th IEEE International Conference on Semiconductor Electronics (ICSE)*. Kuala Lumpur (Malaysia), 2012, p. 664–668. DOI: 10.1109/SMElec.2012.6417231

[23] WING, O. *Classical Circuit Theory*. 1st ed. New York (USA): Springer, 2008. DOI: 10.1007/978-0-387-09740-4

[24] CODECASA, L. Canonical forms of one-port passive distributed thermal networks. *IEEE Transactions on Components and Packaging Technologies*, 2005, vol. 28, no. 1, p. 5–13. DOI: 10.1109/TCAPT.2004.843182

About the Authors ...

László LEITOLD obtained his B.Sc. degree in Mechatronics in 2013 from the Budapest University of Technology and Economics (BME), Hungary. He obtained his M.Sc. degree in Mechanical Engineering in 2019 from the Karlsruhe Institute of Technology (KIT), Germany. He is currently pursuing a Ph.D. degree at the BME, Hungary, under the supervision of András Czmerk and Zsolt Kollár. His research interests include digital signal processing, image processing and system identification.

Monther ALRWASHDEH obtained his B.Sc. in Electrical Engineering from Jordan University of Science and Technology in Jordan in 2004 and his M.Sc. in Wireless Communication from Yarmouk University in Jordan in 2015. He received his Ph.D. degree in Electrical Engineering from BME in 2024. His research interests include quantization effects, digital signal processing, and wireless communications.

Zsolt KOLLÁR obtained his diploma in Electrical Engineering in 2008 from the Budapest University of Technology and Economics (BME). He received his Ph.D. degree in Electrical Engineering from BME in 2013. He is currently an Associate Professor at the Department of Artificial Intelligence and Systems Engineering. Since 2016, he has been the Head of the MATLAB Laboratory. His research interests include digital signal processing, system identification, quantization effects, and wireless communication.