Journal of Engineering Research and Education
Volume 18, (Special Issue) 2026 [97-107]

Exploring Julia for Statistical and Numerical Techniques in Electrical Engineering: Case Studies Aligned with EAC Standard 2024

Teddy Surya Gunawan

Electrical and Computer Engineering Department, International Islamic University Malaysia

ABSTRACT

In response to the Engineering Accreditation Council (EAC) Standard 2024, which mandates the integration of numerical and statistical techniques into engineering education, this paper advocates the adoption of the Julia programming language as a modern computational platform. Julia combines the speed of low-level languages with the intuitive syntax of high-level mathematical tools, making it ideal for electrical engineering curricula. This study presents practical case studies that illustrate Julia's application in solving circuit equations, performing frequency-domain signal analysis, and conducting data-driven modeling, key skills that directly map to Programme Outcomes (PO1, PO2, PO5). Julia's robust ecosystem, including packages such as DifferentialEquations.jl, Plots.jl, and Makie.jl support simulation, analysis, and high-quality visualization, enabling students to translate mathematical models into computational solutions effectively. Early pilot feedback suggests enhanced student engagement, deeper conceptual understanding, and stronger computational literacy. Rather than prescribing Julia, this paper offers a framework and encouragement for its integration wherever appropriate within engineering programs. Embedding Julia in labs, numerical courses, and final-year projects not only aligns with OBE and EAC standards but also equips graduates with industry-relevant skills for solving complex, data-driven engineering problems in a sustainable and accessible way.

Keywords: Julia Programming, Electrical Engineering Education, Numerical Methods, Statistical Analysis, EAC Standard 2024

1. INTRODUCTION

The Washington Accord, established in 1989 under the International Engineering Alliance (IEA), is a multilateral agreement that ensures the mutual recognition of engineering degrees across signatory countries. A core tenet of the Accord is the implementation of Outcome-Based Education (OBE), which emphasizes the attainment of defined learning outcomes aligned with graduate attributes, including engineering knowledge, problem analysis, and professional practice [1, 2]. These outcomes are articulated through the IEA's evolving Graduate Attributes and Professional Competencies framework, with the most recent Version 4 reinforcing themes such as sustainability, ethics, and lifelong learning alongside technical competencies [3, 4]. Malaysia became a provisional member in 2003 and a full signatory of the Accord in 2009, committing to the development of globally competent engineering graduates through OBE-focused accreditation. The establishment of the Engineering Accreditation Council (EAC) around 2000 further solidified this commitment, transitioning from content-based instruction to continuous quality improvement and outcomes-based assessment across engineering programs [5].

To align with these standards, the Malaysian engineering education ecosystem has evolved over decades, with strategic documents such as the Engineering Programme Accreditation Manual (2001–2017) transitioning into the more outcome-centric Engineering Programme Accreditation Standards (2020, 2024). These documents emphasized the integration of mathematics, statistics, and computing within engineering curricula to solve complex problems [6, 7]. Both the 2020 and

*tsgunawan@iium.edu.my

2024 standards consistently list relevant courses across disciplines, including Statistical and Numerical Techniques and Computer Applications for Electrical Engineering [8]. This focus acknowledges the growing need for engineering graduates to develop computational thinking, numerical literacy, and the ability to utilize modern software tools in various real-world engineering scenarios [9]. The EAC 2024 Standard refines this alignment with the Washington Accord by emphasizing applied learning, interdisciplinary integration, and the depth of mathematical and statistical content in engineering curricula.

In the context of the Bachelor of Electrical and Electronics Engineering (BEEE) program at IIUM, this paper proposes integrating the Julia programming language to address the EAC 2024 emphasis on statistical and numerical competencies. Julia is increasingly recognized for its high-performance execution, intuitive mathematical syntax, and robust ecosystem for data analysis, simulation, and modelling [10-12]. It offers a modern alternative to traditional tools like C/C++, MATLAB, and Python, particularly in courses such as Circuit Analysis, Signals and Systems, and Control Systems, as well as project-based learning contexts like IDP and FYP. This paper presents a structured framework for adopting Julia within the BEEE curriculum, supported by practical case studies and mapped alignment to Programme Outcomes (PO1, PO2, PO5). It aims to demonstrate how Julia can modernize computational pedagogy while fulfilling both national accreditation requirements and global engineering education standards.

2. BACKGROUND AND RELATED WORKS

Mathematics serves as the foundation of the Bachelor of Electrical and Electronics Engineering (BEEE) curriculum at the International Islamic University Malaysia (IIUM), underpinning students' analytical capabilities and problem-solving skills in alignment with the EAC Standard 2024 and the Washington Accord. As outlined in Table 1, the curriculum is structured to progressively build mathematical competency through courses such as Engineering Mathematics 1 and 2, Differential Equations, and Computational Methods and Statistics. These courses provide essential knowledge in complex numbers, multivariable calculus, differential equations, and numerical/statistical techniques, ensuring students are well-prepared to engage in system modeling, signal analysis, and simulation tasks in core engineering courses such as Circuit Analysis, Signals and Systems, and Instrumentation and Control Systems. This systematic progression supports the development of Programme Outcomes PO1 (Engineering Knowledge) and PO2 (Problem Analysis), enabling students to bridge the gap between mathematical theory and practical engineering applications [7].

Table 1 Mathematics Courses in BEEE-IIUM Curriculum 2021

Course Code	Course Title	Core Topics	PO Mapping
MATH 1310	Engineering Mathematics 1	Complex numbers, matrices and determinants, vectors, single-variable calculus	PO1, PO2
MATH 1321	Engineering Mathematics 2	Power series, multivariable calculus, multiple integrals, vector calculus, line & surface integrals	PO1, PO2
MATH 2311	Differential Equations	ODEs, PDEs, Laplace and Fourier transforms, system modelling	PO1, PO2
MATH 2330	Computational Methods and Statistics	Probability, distributions, hypothesis testing, regression, numerical methods, numerical solutions of ODEs and PDEs	PO1, PO2

* PO1 (Engineering Knowledge), PO2 (Problem Analysis)

Programming languages complement this mathematical foundation by serving as the computational medium through which students explore numerical methods, data analysis, and simulation. Traditionally, languages like C, C++, and Fortran have offered strong execution speed and control but often lack intuitive syntax for mathematical formulations, posing a steep learning curve and cognitive barrier for engineering students [10, 13]. On the other hand, MATLAB and Python have gained popularity in engineering education due to their ease of use, built-in libraries, and community support; however, each presents drawbacks, such as licensing costs and performance limitations in intensive simulations [14, 15]. Julia addresses these limitations by combining the computational efficiency of C with a syntax that closely resembles mathematical notation, making it particularly suitable for courses that require high-performance numerical and statistical analysis. As summarized in Table 2, Julia offers a compelling alternative with its rich package ecosystem, such as DifferentialEquations.jl, Statistics.jl, and Plots.jl, and native support for symbolic computation and advanced visualization [11, 16].

Table 2 Comparison of Programming Languages for Statistical and Numerical Techniques in Engineering Education

Programming Language	Cost	Installation Size & Required Packages	Description
C	Free/Open-Source	Small; needs GSL, LAPACK for numerical/statistics	Extremely fast, low-level control, steep learning curve, lacks high-level math syntax, widely used in embedded/real-time systems
C++	Free/Open-Source	Small; needs Eigen, Armadillo for numerical/statistics	Fast, object-oriented, moderate learning curve, widely used in engineering software and simulations
Fortran	Free/Open-Source	Small; built-in numerical computing capabilities	Excellent speed for numerical tasks, dated syntax, limited modern tooling, strong in scientific computing
Java	Free/Open-Source	Medium; needs Apache Commons Math for numerical tasks	Moderate speed, object-oriented, better for simulations, not optimized for numerically-intensive computing
MATLAB	Commercial (Expensive)	Large; built-in toolboxes for numerical/statistical analysis	User-friendly, strong plotting and matrix handling, widely used in academia/industry, costly licensing
Python	Free/Open-Source	Medium-Large; needs NumPy, SciPy, Matplotlib, Pandas	Easy to learn, readable syntax, slower unless optimized with C extensions, widely adopted for engineering education
R	Free/Open-Source	Medium; base R with tidyverse for data handling and ggplot2 for visualization	Excellent for statistical analysis and visualization, slower for extensive simulations, less suitable for complex system modelling
Rust	Free/Open-Source	Small; requires crates for numerical/statistical tasks (ndarray, nalgebra)	Fast, memory-safe, steep learning curve, less adopted in engineering education, but promising for safe high-performance computing
Julia	Free/Open-Source	Small-Medium; core with DifferentialEquations.jl, Statistics.jl, Plots.jl	Very fast (near C/Fortran), intuitive mathematical syntax, rich ecosystem for numerical/statistical analysis, growing adoption in engineering education

Within the BEEE curriculum, programming is introduced in EECE 1313 (Programming for Engineers), which is mapped to PO2 (Problem Analysis) and PO3 (Design/Development of Solutions). While C++ serves as the entry point for algorithmic thinking and problem-solving, Julia offers a more accessible and powerful platform for advanced computational tasks introduced in MATH 2330 (Computational Methods and Statistics). Julia's compatibility with C++ and Python, combined with its mathematical expressiveness and performance, makes it an ideal candidate for integrating numerical methods, simulations, and statistical modelling throughout the engineering program (Geike, 2025). This paper advocates for the phased adoption of Julia, starting from MATH 2330 and expanding into courses such as Circuit Analysis, Signals and Systems, Instrumentation and Control Systems, and project-based components, including Integrated Design Projects (IDP) and Final Year Projects (FYP). Such integration aligns with EAC Programme Outcomes PO1, PO2, and PO5 (Modern Tool Usage), reinforcing Julia's role not merely as a programming language but as a pedagogical enabler for high-quality engineering education within the OBE framework [8, 17].

3. JULIA FOR STATISTICAL AND NUMERICAL TECHNIQUES IN ENGINEERING EDUCATION

As engineering curricula evolve to meet the demands of OBE and the EAC Standard 2024, the integration of modern, high-performance computational tools becomes increasingly critical. Julia, particularly the recent long-term support (LTS) release version 1.11.2 (April 2025), is well-positioned for sustained academic deployment due to its stability, intuitive syntax, and execution speeds comparable to C/Fortran. It features a Just-In-Time (JIT) compiler and supports enhanced multithreading, GPU acceleration, and reduced latency for initial compilations, making it highly efficient for numerically intensive tasks. Julia's lightweight core (~300 MB) and fully functional scientific environment (under 1 GB) make it suitable for both institutional labs and student laptops without requiring extensive computing infrastructure. As detailed in Table 3, Julia can be strategically integrated across various BEEE courses. These integrations enable students to connect theoretical concepts with computational models directly, fulfilling EAC Programme Outcomes PO1 (Engineering Knowledge), PO2 (Problem Analysis), and PO5 (Modern Tool Usage).

Table 3 Julia Applications in Selected BEEE Courses

Course	Possible Topics and Required Packages
Computational Methods and Statistics	Linear & nonlinear systems (LinearAlgebra, NLsolve.jl), numerical integration & differentiation (QuadGK.jl, Calculus.jl), ODE/PDE solving (DifferentialEquations.jl), statistical analysis (Statistics.jl, GLM.jl), Monte Carlo methods (Distributions.jl, MonteCarloMeasurements.jl).
Circuit Analysis	Mesh/nodal analysis (LinearAlgebra), transient analysis of circuits (DifferentialEquations.jl), AC steady-state analysis, waveform visualization (Plots.jl, Makie.jl).
Signals and Systems	Convolution & correlation (DSP.jl), FFT and frequency analysis (FFT.jl), system response simulations (DifferentialEquations.jl), signal filtering (DSP.jl, SignalAnalysis.jl).
Instrumentation and Control Systems	Modelling & simulation of control systems (ControlSystems.jl), Bode/Root Locus/Nyquist plots (ControlSystems.jl, Plots.jl), PID tuning, system response analysis.
AI and Machine Learning Courses	Data preprocessing (DataFrames.jl), regression/classification/clustering (MLJ.jl, Flux.jl), neural networks (Flux.jl), reinforcement learning (ReinforcementLearning.jl).
Laboratory Courses	Data acquisition and filtering (DSP.jl), experimental data analysis (Statistics.jl), live plotting and analysis (Plots.jl, Makie.jl), modelling of lab experiments (DifferentialEquations.jl).

Integrated Design Project (IDP)	Simulation of proposed systems (DifferentialEquations.jl, ControlSystems.jl), optimization of system parameters (Optim.jl), data visualization and reporting (Plots.jl, DataFrames.jl).
Final Year Project (FYP)	Advanced system modelling, simulation, and analysis (DifferentialEquations.jl, ControlSystems.jl), large dataset analysis (DataFrames.jl, Statistics.jl), AI model implementation (Flux.jl, MLJ.jl), visualization for reporting (Plots.jl, Makie.jl).

Beyond numerical methods, Julia provides substantial advantages for system modelling, AI, and advanced data analytics across control systems, laboratory work, Integrated Design Projects, and Final Year Projects. In Instrumentation and Control Systems, ControlSystems.jl supports controller design, PID tuning, and frequency-domain analysis. In AI and machine learning courses, MLJ.jl and Flux.jl allow students to implement supervised learning, neural networks, and reinforcement learning workflows. Julia's utility extends to laboratory courses where tools like DSP.jl, Plots.jl, and Makie.jl facilitate real-time data acquisition and analysis. Moreover, project-based learning in IDP and FYP is enriched through system simulation (DifferentialEquations.jl), parameter optimization (Optim.jl), and publication-ready visualizations (PGFPlotsX.jl). Table 4 lists a broad range of Julia packages categorized by function, highlighting the platform's adaptability across engineering subfields. With a simple package installation mechanism (] add PackageName), students can quickly build custom environments tailored to their project or course needs. This modularity, scalability, and alignment with modern engineering practices reinforce Julia's role as a transformative educational tool that promotes computational literacy, interdisciplinary thinking, and lifelong learning, all while aligning with the EAC's complex problem-solving criteria and the Washington Accord's global benchmarks.

Table 4 Samples of Julia Packages for Engineering Applications

Package	Application Area
LinearAlgebra (StdLib)	Matrix computations, eigenvalues, linear systems
DifferentialEquations.jl	ODE/PDE/DAE solving, system simulation
NLsolve.jl	Nonlinear systems solver
QuadGK.jl, Calculus.jl	Numerical integration, differentiation
GLM.jl, Statistics.jl	Statistical analysis, regression models
Distributions.jl, MonteCarloMeasurements.jl	Probability, Monte Carlo simulations
Plots.jl, Makie.jl	Data and system visualization
ControlSystems.jl	Control system design, analysis, and frequency plots
DSP.jl, FFTW.jl	Signal processing, FFT
SignalAnalysis.jl	Advanced signal analysis tools
DataFrames.jl	Data manipulation and analysis
Optim.jl	Optimization and parameter tuning
Flux.jl, MLJ.jl	Machine learning, neural networks
ReinforcementLearning.jl	Reinforcement learning algorithms
Unitful.jl	Handling physical units in computations
JuMP.jl	Mathematical optimization, linear and nonlinear programming
Plots.jl + PGFPlotsX.jl	Publication-quality plots for reports
ControlSystemIdentification.jl	System identification and modelling
CSV.jl	Data import/export for analysis

Measurements.jl	Error propagation and measurement analysis
Gadfly.jl	Alternative visualization with a grammar of graphics approach

4. JULIA CASE STUDIES AND PRACTICAL EXAMPLES IN ELECTRICAL ENGINEERING COURSES

This section presents a series of case studies and practical examples that demonstrate how Julia can be effectively integrated into core Electrical Engineering courses to enhance students' numerical analysis, simulation, and problem-solving skills, aligning with EAC Programme Outcomes.

4.1. Signals Frequency Analysis using the Fast Fourier Transform

The Fast Fourier Transform (FFT) is fundamental in Signals and Systems for analyzing the frequency components of electrical signals, enabling engineers to understand system behavior in the frequency domain, design filters, and evaluate spectral content in practical scenarios such as vibration analysis, power system harmonics, and audio processing. FFT is also a cornerstone in digital signal processing, transforming theoretical continuous Fourier concepts into discrete, computationally efficient tools for engineering practice. Using Julia's DSP.jl and FFTW.jl packages, students can directly connect the mathematical foundation of Fourier analysis taught in class with practical computation, gaining intuition on sampling, spectral leakage, and resolution while leveraging Julia's high-performance computing capabilities.

Given a discrete-time signal $x[n]$ of length N, the Discrete Fourier Transform is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi kn}{N}}, \quad k = 0,1, \dots, N-1. \tag{1}$$

The FFT is an efficient algorithm to compute $X[k]$ with $O(N \log N)$ complexity. For illustration, let us consider a multi-frequency signal as follows:

$$x[n] = \sin(2\pi f_1 n T_s) + 0.5 \sin(r\pi f_2 n Ts) \tag{2}$$

where f_1 is 50 Hz, f_2 is 120 Hz, and $T_s = \frac{1}{1000}$ s (sampling at 1000 Hz).

This will produce a signal with two clear frequency components for spectral analysis. The short and clear working Julia code is listed in Figure 1.

Using Julia for FFT analysis enables high-speed, high-precision spectral analysis without complex syntax, making it accessible for undergraduate labs and projects. Its syntax mirrors that of mathematical equations, enhancing students' comprehension of the Fourier Transform's discrete implementation and its practical limitations (e.g., spectral leakage, resolution). Julia's plotting tools allow immediate visualization, reinforcing the connection between time-domain signals and their frequency-domain representations, while aligning with EAC PO1, PO2, and PO5 for practical engineering education.

prog1.jl

```
1  using FFTW, Plots, LaTeXStrings
2
3  # Parameters
4  fs = 1000          # Sampling frequency (Hz)
5  T = 1/fs           # Sampling period
6  t = 0:T:1          # Time vector, 1 second
7  N = length(t)      # Number of samples
8
9  # Signal: sum of 50 Hz and 120 Hz sine waves
10 f1 = 50
11 f2 = 120
12 x = sin.(2π * f1 * t) .+ 0.5 * sin.(2π * f2 * t)
13
14 # FFT computation
15 X = fft(x)
16 f = (0:N-1) * fs / N   # Frequency vector
17
18 # Magnitude spectrum (normalized)
19 magnitude = abs.(X) / N
20
21 # Plot
22 plot(f[1:div(N,2)], magnitude[1:div(N,2)],
23     xlabel = L"f\;(Hz)",
24     ylabel = L"|X(f)|",
25     title = "FFT of Signal with 50 Hz and 120 Hz Components",
26     legend = false, dpi = 300)
27 savefig("pprog2.png")
```

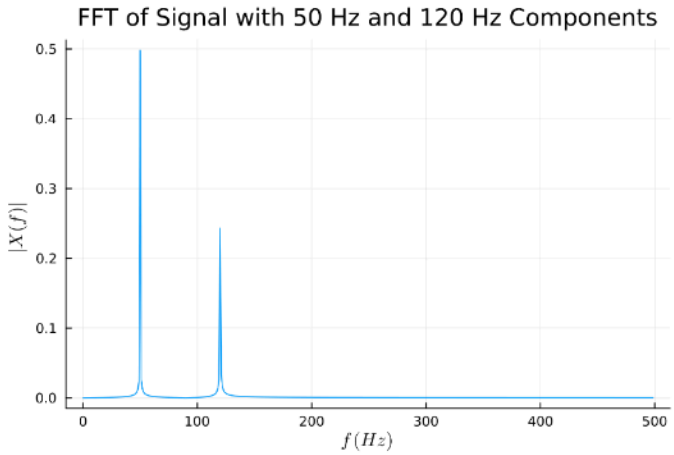

Figure 1. Julia code and output for Case 1.

4.2. Exploratory Data Analysis of Open Power System Data

Engineers must analyze real-world electricity generation and consumption data as power systems transition toward renewable integration and smart grid management. The Open Power System Data (OPSD) dataset provides clean, structured data on electricity consumption, wind, and solar generation across countries. It suits Exploratory Data Analysis (EDA) in power system analytics courses, lab projects, and FYP.

Given a dataset $X = \{x_{ij}\}$ where $i = 1, \dots, n$ (time samples), $j = 1, \dots, p$ (features), the descriptive statistics, such as mean, standard deviation, and correlation analysis can be calculated.

$$\begin{aligned} \bar{x}_j &= \frac{1}{n}\sum_{i=1}^{n} x_{ij} \\ s_j &= \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(x_{ij} - \bar{x}_j\right)^2} \\ r_{jk} &= \frac{\sum_{i=1}^{n}(x_{ij}-\bar{x}_j)(x_{ik}-\bar{x}_k)}{(n-1)s_j s_k} \end{aligned} \tag{3}$$

Correlation analysis could be performed to evaluate relationships between demand, wind, and solar generation. Several graphs can be plotted for time-series visualization, such as line plots to observe trends and seasonality, histograms for distribution analysis, and box plots to detect outliers and variability. Figure 2 shows a short, clear Julia code that performs EDA on the open power system data, printing statistical summaries and generating high-quality plots for publication-ready analysis.

prog2.jl

```
1  using CSV, HTTP, DataFrames, Statistics, Plots, StatsPlots
2
3  # Load OPSD dataset directly
4  url = "https://data.open-power-system-data.org/time_series/2019-06-
   05/time_series_60min_singleindex.csv"
5  df = CSV.read(HTTP.get(url).body, DataFrame)
6
7  # Select relevant columns
8  select_cols = [:utc_timestamp,
9          Symbol("DE_load_actual_entsoe_transparency"),
10         Symbol("DE_solar_generation_actual"),
11         Symbol("DE_wind_generation_actual")]
12
13 df = select(df, select_cols)
14
15 # Drop missing rows for clean EDA
16 dropmissing!(df)
17
18 # Rename for easier plotting
19 rename!(df, Dict(Symbol("DE_load_actual_entsoe_transparency") => :Load,
20          Symbol("DE_solar_generation_actual") => :Solar,
21          Symbol("DE_wind_generation_actual") => :Wind))
22
23 # Descriptive statistics
24 println("Summary Statistics:")
25 describe(df[:, [:Load, :Solar, :Wind]]) |> println
26
27 # Correlation matrix
28 corr_matrix = cor(Matrix(df[:, [:Load, :Solar, :Wind]]))
29 println("\nCorrelation Matrix:")
30 println(corr_matrix)
31
32 # Time-series plot (subsampled for clarity)
33 plot(df.utc_timestamp[1:5000], df.Load[1:5000], label="Load", xlabel="Time", ylabel="MW",
   dpi=300)
34 plot!(df.utc_timestamp[1:5000], df.Solar[1:5000], label="Solar")
35 plot!(df.utc_timestamp[1:5000], df.Wind[1:5000], label="Wind", title="Power System Time
   Series (Subsample)")
36 savefig("prog4_timeseries.png")
```

```
37
38  # Histogram of Load
39  histogram(df.Load, bins=50, xlabel="Load (MW)", ylabel="Frequency", title="Histogram of
    Load", dpi=300)
40  savefig("prog4_histo.png")
41
42  # Boxplot of Load, Solar, Wind
43  boxplot(["Load", "Solar", "Wind"],
44        [df.Load, df.Solar, df.Wind],
45        ylabel="MW", title="Boxplot of Load, Solar, Wind", dpi=300)
46  savefig("prog4_boxplot.png")
```

```
Summary Statistics:
3×7 DataFrame
 Row │ variable  mean     min      median   max      nmissing  eltype
     │ Symbol    Float64  Float64  Float64  Float64  Int64     DataType
─────┼──────────────────────────────────────────────────────────────────
   1 │ Load      56119.1  31455.0  55851.5  79063.0         0  Float64
   2 │ Solar     4156.17      0.0    106.0  30028.0         0  Float64
   3 │ Wind      10974.3    135.0   8498.0  45085.0         0  Float64

Correlation Matrix:
[1.0 0.32910361581196856 0.1458553486804314; 0.32910361581196856 1.0 -
0.18069618801617351; 0.1458553486804314 -0.18069618801617351 1.0]
```

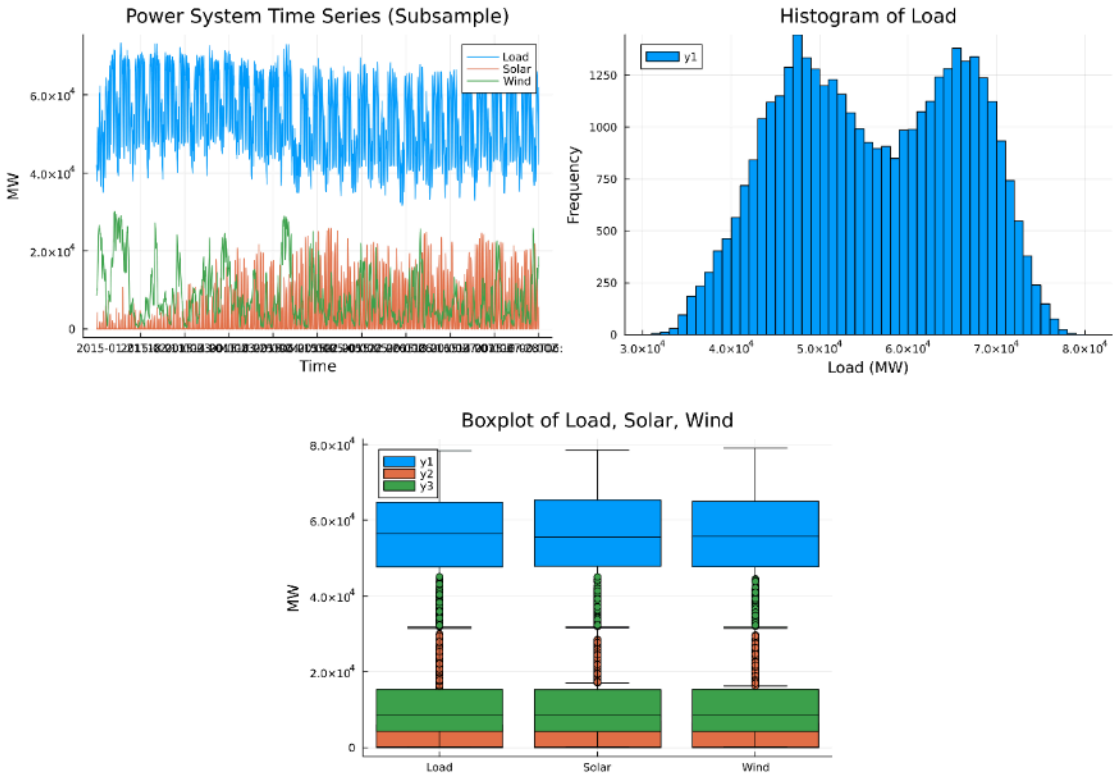

Figure 2. Julia code and output for Case 2.

5. CONCLUSIONS

This paper presents a structured framework and a series of case studies to demonstrate the integration of the Julia programming language in engineering education, specifically to enhance numerical and statistical competencies aligned with the EAC Standard 2024 and Washington Accord. By embedding Julia into core and supporting courses, such as MATH2330, Circuit Analysis, Signals and Systems, Instrumentation and Control, AI, and Final Year Projects (FYP), students are empowered to bridge theoretical knowledge with practical simulation through a high-performance, intuitive, and open-source platform. Julia's syntax mirrors mathematical formulations, facilitating deeper cognitive understanding, while its capacity to generate 300 dpi publication-quality plots cultivates professional reporting and documentation skills. The paper proposes rubric-based assessment strategies for PO1 (engineering knowledge), PO2 (problem analysis), and PO5 (modern tool usage), ensuring clear mapping to Programme Outcomes. Julia's capabilities in symbolic computation, real-time visualization, and system modeling promote critical thinking and experimentation, fostering computational literacy in both laboratory and project-based learning. Its compatibility with sustainable, low-cost educational infrastructure and its alignment with modern industry workflows further strengthen its relevance. While a learning curve exists, a phased integration, starting with lab modules and projects, offers a manageable path forward. Future work will include formal implementation and evaluation via assessment rubrics and student surveys to empirically validate Julia's role in modernizing engineering education under the Outcome-Based Education framework.

ACKNOWLEDGEMENTS

We would like to extend our gratitude to the Ministry of Higher Education for the FRGS grant (FRGS/1/2023/SSI07/UIAM/01/2), which provided financial support for this study.

REFERENCES

[1] Qadir, J., et al., 2020. Outcome-based engineering education: A global report of international OBE accreditation and assessment practices, in: Proceedings of the ASEE Virtual Annual Conference, ASEE, 2020.

[2] Kamran, M., Nisa, B. U., Fazal, M. R., Abid, M. I., Abid, I., 2020. Implementation of the outcome-based education system in engineering programs for Pakistan Engineering Council accreditation under Washington Accord signatory. Sci. Int. (Lahore) 32(2), 197–206.

[3] International Engineering Alliance, 2021. Graduate Attributes and Professional Competencies: Version 4 (2021.1), International Engineering Alliance.

[4] Liew, C. P., Kiew, P. L., 2022. Sustainable assessment: The inevitable future of engineering curriculum. ASEAN J. Eng. Educ. 6(1), 23–32.

[5] Yan, K. Y., Long, C. Y., Sin, M. C., 2019. Engineering programme accreditation: A comparison study between the accreditation criteria of academic curriculum in Malaysia and Singapore, in: Proceedings of the 19th Annual SEAAIR Conference, Taiwan, 2019.

[6] Engineering Accreditation Council Malaysia, 2020. Engineering Programme Accreditation Standard 2020, EAC Malaysia.

[7] Engineering Accreditation Council Malaysia, 2024. Engineering Programme Accreditation Standard 2024, EAC Malaysia.

[8] Liew, C. P., Puteh, M., Mohammad, S., Kiew, P. L., Tan, K. G., 2024. Sustainable assessment practices for engineering programme outcomes: Challenges and recommendations in Malaysian higher learning institutions. Int. J. Eng. Educ. 40(2), 216–230.

[9] Ling, J. H., 2024. Implementation of complex engineering problem solving projects in a Malaysian engineering programme. Indones. J. Educ. Soc. Sci. 3(2), 133–150.

[10] Rackauckas, C., 2018. A comparison between differential equation solver suites in Matlab, R, Julia, Python, C, Mathematica, Maple, and Fortran. Winnower, 1–14.
[11] Gao, K., Mei, G., Piccialli, F., Cuomo, S., Tu, J., Huo, Z., 2020. Julia language in machine learning: Algorithms, applications, and open issues. Comput. Sci. Rev. 37, 100254.
[12] Geike, T., 2025. Teaching tribology and engineering mechanics with the Julia programming language, in: Tribology Across Scales: Theory, Simulation and Experiment, Springer, 2025, pp. 413–439.
[13] Srpak, I., Havaš, L., Horvat, T., Tomičić, E., 2023. Aspects and roles of different programming languages and their use in STEM education, in: EDULEARN23 Proceedings, IATED, 2023, pp. 2944–2953.
[14] Brieda, L., Wang, J., Martin, R., 2024. Introduction to Modern Scientific Programming and Numerical Methods, CRC Press, 2024.
[15] Johns, A. N., Hesketh, R. P., Stuber, M. D., Ford Versypt, A. N., 2023. Numerical problem solving across the curriculum with Python and MATLAB using interactive coding templates, in: Proceedings of the ASEE Annual Conference, ASEE, 2023.
[16] Montoya, O. D., Ramírez-Vanegas, C. A., González-Granada, J. R., 2024. Dynamic active and reactive power compensation in distribution networks using PV-STATCOMs: A tutorial using the Julia software. Results Eng. 21, 101876.
[17] Sherrington, M., 2024. Mastering Julia: Enhance Your Analytical and Programming Skills for Data Modeling and Processing with Julia, Packt Publishing, 2024.