Leveraging WebAssembly for Enhanced Performance in Uncertainty Calculation of Monte Carlo Simulation on Web Browsers

Artur Augusto Martins[1], **Anderson Poiani Lopes Mendes**[1], **Ednaldo Cruz**[1]

[1] Instituto de Pesquisas Tecnológicas do Estado de São Paulo S.A – IPT

E-mail: arturm@ipt.br ; andersonm@ipt.br ; ednaldoc@ipt.br

Abstract. Monte Carlo simulations are widely used in various scientific and engineering domains to estimate uncertainties and make informed decisions. However, running these simulations efficiently on web browsers poses significant challenges due to the inherent limitations of JavaScript, the predominant language for web development. In this paper, we present a novel approach that leverages WebAssembly, compiled from the Rust programming language, to enhance the performance of uncertainty calculations in Monte Carlo simulations directly within web browsers. Our approach is implemented in a web application called "Uncertainty.app," which provides users with a seamless and interactive interface to perform uncertainty calculations.

Keywords: Monte Carlo, Uncertainty analysis, WebAssembly.

1. Introduction

Monte Carlo simulations have proven to be invaluable in diverse domains such as physics, finance, and engineering, enabling researchers and practitioners to analyze uncertainties and make informed decisions. However, executing these simulations efficiently in web browsers presents unique challenges due to the limitations of JavaScript, the predominant language for web development. JavaScript's dynamic nature and lack of low-level optimizations can significantly hinder the performance of computationally intensive tasks, limiting the scope and complexity of simulations that can be performed on the web.

To address these challenges, we propose a novel approach that utilizes WebAssembly[1], a binary instruction format for web browsers, to optimize the computation of uncertainties in Monte Carlo simulations. WebAssembly allows developers to compile code from various programming languages into a low-level format that can be executed directly within web browsers. In this work, we leverage the Rust programming language[2], known for its focus on performance and memory safety, to develop highly efficient uncertainty calculation routines.

To showcase the practicality and effectiveness of our approach, we have developed "Uncertainty.app," (https://uncertainty.app) a web application that enables users to perform uncertainty calculations using Monte Carlo simulations directly in their web browsers. Leveraging WebAssembly compiled from Rust, Uncertainty.app achieves significant performance improvements compared to traditional JavaScript-based implementations. By offloading the computational burden to the client-side, users can benefit from interactive and responsive simulations without requiring server-side processing or additional infrastructure.

2. Methods

Our project focuses on optimizing four essential steps of uncertainty calculation in Monte Carlo simulations: evaluation, sampling, sensitivity analysis, and histogram building.

2.1. Evaluation

The Evaluation step begins by receiving the mathematical model expression, which represents the system being analyzed under uncertainty. This expression may consist of mathematical functions, operations, and variables that interact with each other. To ensure flexibility and generality, the expression can incorporate commonly used mathematical functions and operations.

By transforming the mathematical model expression into a fast and specific domain programming language, the Evaluation step significantly enhances the performance of uncertainty calculations in Monte Carlo simulations within web browsers. The use of a specialized compiler ensures optimized execution and seamless integration with the subsequent steps of the simulation process.

Additionally, the compiled code is designed to seamlessly interact with the remaining steps of the Monte Carlo simulation process. It can efficiently receive the samples generated from probability distributions defined for each quantity represented by variables. These samples are essential for evaluating the uncertainty and obtaining meaningful simulation results. The compiled code utilizes the samples to perform the necessary calculations specified by the mathematical model expression.

2.2. Sampling

In the proposed project, the Sampling step plays a critical role in generating random samples from probability distributions for each variable involved in the uncertainty calculation. The step aims to provide efficient and accurate sampling algorithms for a range of available distributions, including Uniform, Normal, Student's t, Triangular, Arcsine, Trapezoidal, and Curvilinear Trapezoidal. The sampling algorithms implemented in this project are based on the document "Evaluation of measurement data Supplement 1 to the 'Guide to the expression of uncertainty in measurement' - Propagation of distributions using a Monte Carlo method,"[3] which ensures reliable and validated sampling methodologies.

To enable uncertainty analysis, it is necessary to define probability distributions for each variable in the mathematical model. The Sampling step accepts user-defined parameters for these distributions, allowing customization and flexibility in modeling the uncertainties. The implemented algorithms take into account the parameters defined by the user, such as mean, standard deviation, shape parameters, and range constraints, to generate random samples that reflect the characteristics of the specified distributions.

2.3. Sensitivity Analysis

To calculate sensitivity coefficients, the Sensitivity Analysis step employs a method that involves holding all input quantities but one fixed at their best estimates. The best estimate represents the most likely or most accurate value for each input quantity. By systematically varying a single input quantity while keeping the others constant, the step estimates the probability density function (PDF) for the output quantity, considering the model with that specific input quantity as a variable.

The ratio of the standard deviation of the resulting model values to the standard uncertainty associated with the best estimate of the relevant input quantity is used as a sensitivity coefficient. This coefficient reflects the change in the output uncertainty corresponding to a unit change in the input uncertainty. It provides a quantitative measure of the impact of a particular input quantity on the overall uncertainty of the model.

The methodology for determining sensitivity coefficients and their interpretation aligns with established guidelines, such as Annex B of the JCGM 101:2008 reference.

2.4. Histogram Building and Reporting

Histogram Building and Reporting step focuses on presenting the output estimations of the Monte Carlo simulations through a histogram. To build the histogram, a fixed number of output samples (M) is generated from the Monte Carlo simulations. In this project, we set M to 1e6. To enhance performance and maintain a responsive user interface, the generation of these samples is performed using Web Workers.

Web Workers are like virtual assistants for your computer's web browser. Imagine you're doing a big task that takes a lot of time, like counting a million nuts. Instead of doing it all by yourself, you can hire several assistants to help you count. These assistants work together at the same time, so the job gets done much faster. Web Workers are like those assistants for the computer. They help the computer split up the work into smaller parts and do them all at once.

Once the output samples are generated, they are merged from multiple Web Workers into a single dataset. By combining and ordering the samples, we create a comprehensive dataset that represents the full range of output values from the Monte Carlo simulations.

Subsequently, the dataset is used to construct a histogram with 50 bins. The histogram provides a visual representation of the distribution of the output estimations, allowing users to observe the shape, central tendency, and spread of the output quantities. The number of bins is chosen to balance between capturing the finer details of the distribution and presenting a clear overview, as shown on figure 1.

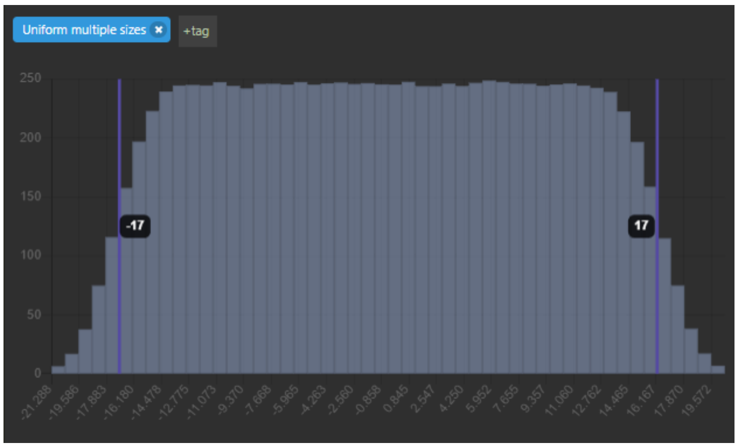


Fig 1 – Histogram output from Uncertainty.app reproducing the Additive model with multiple sizes, described on GUM sup. 1 item 9.2.

In addition to the histogram, two key statistical measures are computed to aid in the interpretation of the output estimations. The shortest confidence interval is determined to provide an estimate of the range within which the true value of the output quantity is likely to fall in respect of user defined coverage interval.

To provide a relative measure of the range covered by the shortest confidence interval, the coverage factor (k) is determined. The coverage factor is calculated as the ratio of the shortest confidence interval to the standard deviation.

3. Comparision with NIST Uncertainty Machine

It is important to acknowledge and appreciate the efforts of organizations like NIST (National Institute of Standards and Technology) in providing valuable tools such as the "uncertainty machine." The "uncertainty machine"[4] is a client-server approach for uncertainty analysis, which has been widely used and recognized for its contributions to the field. When comparing our approach, it is essential to recognize the strengths and characteristics of both approaches.

The NIST uncertainty machine has established itself as a robust and widely used tool. Over time, it has undergone extensive development and refinement, making it a reliable and battle-tested solution. The NIST uncertainty machine's longevity and continued usage demonstrate its ability to handle a wide range of scenarios and provide accurate and trustworthy results.

In contrast, our novel solution leveraging WebAssembly is a relatively new approach. While it offers notable advantages in terms of performance and responsiveness, it may not yet possess the same level of robustness and battle testing as the NIST uncertainty machine. As a newer solution, it may still be undergoing refinement and validation, and its user base and track record may not be as extensive as that of the NIST uncertainty machine.

However, our solution takes the advantage of modern web technologies to deliver high-performance uncertainty analysis directly in web browsers. It offers a streamlined and accessible approach, eliminating the need for a client-server infrastructure and providing a responsive user experience.

In summary, while the NIST uncertainty machine boasts robustness, feature completeness, and extensive testing, our novel solution leveraging WebAssembly and Rust offers unique advantages in terms of performance and user experience.

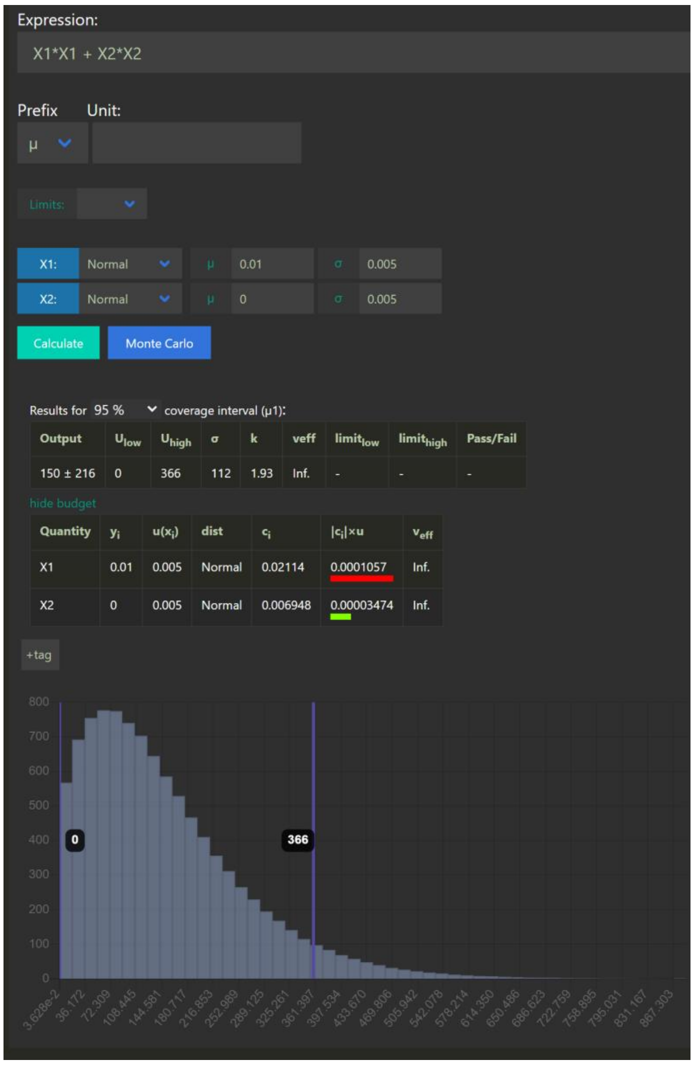
4. Results

To validate the accuracy and effectiveness of our software, we conducted comparisons with two examples from the ISO GUM Supplement 1: Comparison on loss in microwave power meter calibration for zero covariance (figure 2) and Gauge block calibration (figure 3). These examples were chosen to demonstrate the importance of using the Monte Carlo method in achieving accurate results, particularly for a non-linear model.

On figure 2, the “Comparison loss in microwave power meter calibration” example from GUM sup. 1 item 9.4.2.2.7 processed on Uncertainty.app, resulted in [0, 366] $\delta Y / 10^{-6}$ for 95 % Shortest coverage interval, versus [0, 367] $\delta Y / 10^{-6}$ from GUM sup. 1.

On figure 3, the “Gauge block calibration“ example from GUM sup. 1 item 9.5 processed on Uncertainty.app, resulted in [745, 931] $\delta Y / 10^{-6}$ for 99 % Shortest coverage interval, versus [745, 932] $\delta Y / 10^{-6}$ reported on GUM sup. 1.

By modeling these examples in our software, we were able to obtain equivalent results to those described in the ISO GUM Supplement 1. This validation indicates that our software is capable of reproducing the expected outcomes for complex models. The permanent links for execution of examples are avaliable at https://uncertainty.app/#/ex_Comparison_loss_in_microwave and https://uncertainty.app/#/ex_Gauge_block_calibration.



Expression:

X1*X1 + X2*X2

Prefix: μ Unit:

Limits:

X1:	Normal	μ	0.01	σ	0.005
X2:	Normal	μ	0	σ	0.005

Calculate Monte Carlo

Results for 95 % coverage interval (μ1):

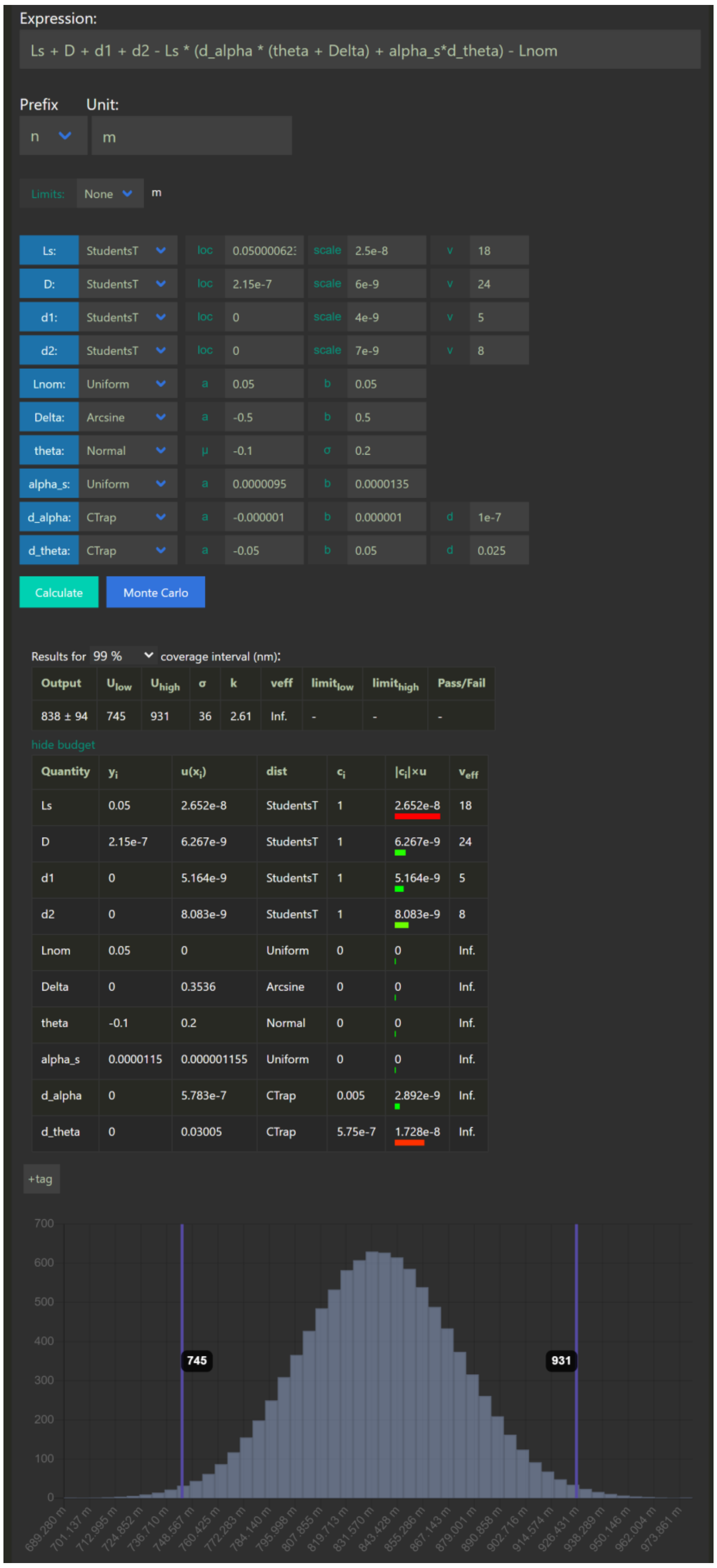
Output	U_{low}	U_{high}	σ	k	veff	$limit_{low}$	$limit_{high}$	Pass/Fail
150 ± 216	0	366	112	1.93	Inf.	-	-	-

hide budget

Quantity	y_i	$u(x_i)$	dist	c_i	$\|c_i\|\times u$	v_{eff}
X1	0.01	0.005	Normal	0.02114	0.0001057 🟥	Inf.
X2	0	0.005	Normal	0.006948	0.00003474 🟩	Inf.

+tag

Figure 2 – Comparison loss in microwave power meter calibration example.



Expression:

Ls + D + d1 + d2 - Ls * (d_alpha * (theta + Delta) + alpha_s*d_theta) - Lnom

Prefix: n Unit: m

Limits: None m

Ls:	StudentsT	loc	0.05000062:	scale	2.5e-8	v	18
D:	StudentsT	loc	2.15e-7	scale	6e-9	v	24
d1:	StudentsT	loc	0	scale	4e-9	v	5
d2:	StudentsT	loc	0	scale	7e-9	v	8
Lnom:	Uniform	a	0.05	b	0.05		
Delta:	Arcsine	a	-0.5	b	0.5		
theta:	Normal	μ	-0.1	σ	0.2		
alpha_s:	Uniform	a	0.0000095	b	0.0000135		
d_alpha:	CTrap	a	-0.000001	b	0.000001	d	1e-7
d_theta:	CTrap	a	-0.05	b	0.05	d	0.025

Calculate Monte Carlo

Results for 99 % coverage interval (nm):

Output	U_{low}	U_{high}	σ	k	veff	$limit_{low}$	$limit_{high}$	Pass/Fail
838 ± 94	745	931	36	2.61	Inf.	-	-	-

hide budget

Quantity	y_i	$u(x_i)$	dist	c_i	$\|c_i\|\times u$	v_{eff}
Ls	0.05	2.652e-8	StudentsT	1	2.652e-8 🟥	18
D	2.15e-7	6.267e-9	StudentsT	1	6.267e-9 🟩	24
d1	0	5.164e-9	StudentsT	1	5.164e-9 🟩	5
d2	0	8.083e-9	StudentsT	1	8.083e-9 🟩	8
Lnom	0.05	0	Uniform	0	0	Inf.
Delta	0	0.3536	Arcsine	0	0	Inf.
theta	-0.1	0.2	Normal	0	0	Inf.
alpha_s	0.0000115	0.000001155	Uniform	0	0	Inf.
d_alpha	0	5.783e-7	CTrap	0.005	2.892e-9 🟩	Inf.
d_theta	0	0.03005	CTrap	5.75e-7	1.728e-8 🟥	Inf.

+tag

Figure 3 – Gauge block calibration example.

5. Benchmark

For the purpose of evaluating computation time and performance gains using Web Workers, Table 1 provides a summary of the total computation time, comparing scenarios with and without the utilization of Web Workers for parallelizing the generation of random samples from a uniform distribution:

Table 1. Comparison of computation times for generating multiple random samples from a uniform distribution, with and without the use of Web Workers.

	Number of samples *N*=1e5	Number of samples *N*=1e6	Number of samples *N*=1e7
Without Web Workers	144 ms	1.364 ms	13.396 ms
Using Web Workers	46 ms	82 ms	327 ms

Table 2 presents a comparison of computation times for the additive model similar to example found in GUM sup. 1, item 9.2. This comparison is made between two implementations: one powered by WebAssembly, as demonstrated in this paper using Uncertainty.app, and another using pure JavaScript. Both implementations utilize Web Workers for parallelized computation. The comparison includes scenarios involving uniform distributions and varying number of quantities.

Table 2. Comparison of computation times using WebAssembly implementation and pure JavaScript implementation.

	Number of quantities *N*=3	Number of quantities *N*=4	Number of quantities *N*=5
WebAssembly	1.002 ms	1.553 ms	1.762 ms
JavaScript	3.758 ms	5.627 ms	7.272 ms

These tests were conducted on a Windows 10 Pro system with an AMD Ryzen 3 4300U processor, 8 GB of RAM, utilizing 4 threads, and using the Google Chrome web browser. The methodology employed is based on the available JavaScript library [5].

6. Conclusion

In this project, we have successfully leveraged WebAssembly compiled from the Rust programming language to enhance the performance of uncertainty calculations in Monte Carlo simulations on web browsers. By optimizing the evaluation, sampling, sensitivity analysis, and histogram building steps, we have achieved significant improvements in speed, scalability, and user experience.

Through the use of a specialized compiler, we transformed mathematical model expressions into a fast and specific domain programming language, enabling efficient calculations. The implementation of robust sampling algorithms for various probability distributions ensured accurate and reliable generation of random samples. The parallel execution of web workers improved performance and responsiveness.

Overall, our project demonstrates the power of WebAssembly, Rust, and parallelization techniques in enabling efficient uncertainty analysis within web browsers. This advancement opens up new possibilities for researchers, engineers, and decision-makers to perform complex simulations, assess uncertainties, and make informed decisions in a user-friendly and efficient manner.

For future improvement of our software is planned the incorporation of features that enable the creation and utilization of standards for data reuse. By providing a framework to define and implement standardized data formats, our software can facilitate seamless integration and sharing of data across different uncertainty analysis projects. This will not only enhance collaboration and reproducibility but also promote the reuse of valuable datasets, saving time and effort in future analyses.

Additionally, we aim to enhance the flexibility of our software by improving the existing support for custom piecewise distributions. While our current implementation covers a range of commonly used probability distributions, enabling users to define their own piecewise distributions would empower them to model specific scenarios more accurately. This customization capability would cater to a broader range of applications.

References

[1] Rust and WebAssembly. Retrieved from [https://www.rust-lang.org/pt-BR/what/wasm] (Accessed on 26 jun. 2023).

[2] Klabnik, S., Nichols, C., & Rust Community. The Rust Programming Language. Retrieved from [https://doc.rust-lang.org/stable/book/] (Accessed on 26 jun. 2023).

[3] Evaluation of measurement data — Supplement 1 to the "Guide to the expression of uncertainty in measurement" — Propagation of distributions using a Monte Carlo method. Retrieved from [https://www.bipm.org/documents/20126/2071204/JCGM_101_2008_E.pdf] (Accessed on 26 jun. 2023).

[4] NIST Uncertainty Machine. Retrieved from [https://uncertainty.nist.gov/] (Accessed on 26 jun. 2023).

[5] Web Workers Pool JavaScript library. Retrieved from [https://github.com/arturaugusto/web-workers-pool] (Accessed on 15 sep. 2023).