The Journal of Open Source Software

Sigma: Uncertainty Propagation for C++

Jonathan M. Waldrop [1¶] and **Ryan M. Richard** [1,2]

1 Chemical and Biological Sciences, Ames National Laboratory, USA **2** Department of Chemistry, Iowa State University, USA ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Vissarion Fisikopoulos

Reviewers:

- @baxmittens
- @YehorYudinIPP

Submitted: 17 October 2024
Published: unpublished

Summary

Sigma is a header-only C++-17 library for uncertainty propagation, inspired by `uncertainties` (Lebigot, 2009) for Python and `Measurements.jl` (Giordano, 2016) for Julia. The library tracks the functional correlation between dependent and independent variables, ensuring that the uncertainty of the independent variables is properly considered in the calculation of the dependent variables' uncertainties. It is intended as a near drop-in replacement for the standard floating point types (aside from uncertainty specification), and aims to be easily interoperable with the existing standard types.

Statement of need

In scientific analysis, values are often paired with the degree of uncertainty in the accuracy of that value. This uncertainty (or error) could be derived from a number of sources, including the level of accuracy provided by a measuring instrument, the statistical nature of the value being measured, or approximations made in the determination of the value. Often, this uncertainty is represented as the standard deviation of the value. When using these values as function inputs, they convey an uncertainty on the new results. Propagating the uncertainty by hand can be tedious, possibly prohibitively so in the case of calculations that require machine computation to be feasible. As such, it has been found prudent to automate the propagation of error as an extension of the calculations themselves (Giordano, 2016; Lebigot, 2009). To the best of our knowledge, there is no currently maintained C++ library to facilitate this kind of uncertainty propagation. As C++ is an important language in the development of scientific software and high-performance computing, Sigma has been developed in an attempt to fill this gap.

Mathematics

Assume $F(A)$ is a function of A, where A is a set whose elements are some or all of the elements of the sequence of n variables $(a_i)_{i=1}^{n}$. These element are defined as $a_i = \bar{a}_i \pm \sigma_{a_i}$, where $\bar{a}_i$ is the mean value of the variable and σ_{a_i} is called the uncertainty and is assumed to represent an error measure closely related to the standard deviation of a random variable. The linear uncertainty of $F(A)$ can be determined as

$$\sigma_F \approx \sqrt{\sum_{i=1}^{n}\left(\left(\left.\frac{\partial F}{\partial a_i}\right|_{a_i=\bar{a}_i}\sigma_{a_i}\right)^2 + 2\sum_{j=i+1}^{n}\left(\left(\frac{\partial F}{\partial a_i}\right)_{a_i=\bar{a}_i}\left(\frac{\partial F}{\partial a_j}\right)_{a_j=\bar{a}_j}\sigma_{a_i a_j}\right)\right)}.$$

Note that for any element a_i that is not a member of A, $\frac{\partial F}{\partial a_i} = 0$ and those terms vanish in the summations. The term $\sigma_{a_i a_j}$ is the covariance of a_i and a_j, defined as

$$\sigma_{a_i a_j} = E[(a_i - E[a_i])\,(a_j - E[a_j])],$$

The Journal of Open Source Software

where $E[a_i]$ is the expectation value of a_i. The covariances can be eliminated from the above equation if the uncertainties of the variables are independent from one another, which is a requirement imposed here. As such, the uncertainty of $F(A)$ when the members of A are independent from one another is simply

$$\sigma_F \approx \sqrt{\sum_{a_i \in A} \left(\left. \frac{\partial F}{\partial a_i} \right|_{a_i = \bar{a}_i} \sigma_{a_i} \right)^2}.$$

Next, we consider a set $B = \{x, y\}$ where $x = x(a_i, a_j)$ and $y = y(a_j)$, i.e. the elements of B are functions of some number of independent variables. As the values of x and y are dependent on the values of a_i and a_j, they are said to be *functionally correlated* to the independent variables (Giordano, 2016) and their uncertainties are easily calculated from the previous equation. Given the function $G(B)$, the value of σ_G cannot be calculated from the previous equation as it does not account for the functional correlation of the elements of B. The uncertainty of G can be properly determined by application of the chain rule to relate the independent variables to G through their relationships with the dependent variables

$$\sigma_G \approx \sqrt{\left(\left(\frac{\partial G}{\partial x} \frac{\partial x}{\partial a_i} \right)_{a_i = \bar{a}_i} \sigma_{a_i} \right)^2 + \left(\left(\frac{\partial G}{\partial x} \frac{\partial x}{\partial a_j} + \frac{\partial G}{\partial y} \frac{\partial y}{\partial a_j} \right)_{a_j = \bar{a}_j} \sigma_{a_j} \right)^2}.$$

Usage

Sigma is header-only, so it only needs to be findable by the dependent project to be used. The library is buildable with CMake (*CMake*, 2024), and utilizes the CMaize (Crandall et al., 2024) extension to handle configuration, dependency management, and building the tests and/or documentation. To use the library in a project, simply add `#include <sigma/sigma.hpp>` in an appropriate location within the project's source.

The primary component of Sigma is the `Uncertain<T>` class, templated on the floating point type used to represent the mean and uncertainty of the variable. Simple construction of an uncertain floating point value can be accomplished by passing the mean and a value for the uncertainty (such as a standard deviation):

```
using numeric_t = double;
numeric_t a_mean{100.0};
numeric_t a_sd{1.0};
sigma::Uncertain<numeric_t> a{a_mean, a_sd};
std::cout << a << std::endl;     // Prints: 100+/-1
```

The same can be accomplished in a less verbose way as `sigma::Uncertain a{100.0, 1.0}`. Sigma also provides the typedefs `UFloat` and `UDouble` (uncertain `float` and `double`, respectively) for convenience.

Basic arithmetic with certain or uncertain values is accomplish trivially,

```
sigma::Uncertain a{1.0, 0.1};
sigma::Uncertain b{2.0, 0.2};
auto c = a + 2.0 // 3.0+/-0.1
auto d = a * 2.0 // 2.0+/-0.2
auto e = a + b   // 3.0+/-0.2236
auto f = a * b   // 2.0+/-0.2828
```

The resulting variables here are functionally correlated to `a` and/or `b`, meaning the operation `e - c` would return an instance with the value 0 ± 0.2 as the contributions from `a` would exactly negate each other.

Sigma also implements many of the most common math functions found in the C++ standard library, such as those for trigonometry and rounding:

```
sigma::Uncertain radians{0.785398, 0.1};
sigma::Uncertain degrees{45.0, 0.1};
auto to_degrees = sigma::degrees(radians); // 45.0000+/-5.7296
auto in_radians = sigma::radians(degrees); //  0.7854+/-0.0017
auto tangent    = sigma::tan(radians)      //  1.0000+/-0.2000
auto truncated  = sigma::trunc({1.2, 0.1}) //  1.0+/-0.0
```

Sigma also has a limited degree of compatibility with the Eigen library (*Eigen*, 2024), allowing for matrix operations and a number of linear solvers. Additional functionality is possible, though not currently ensured.

Acknowledgements

This work was supported by the Ames National Laboratory's Laboratory Directed Research and Development (LDRD) program. The Ames Laboratory is operated for the U.S. DOE by Iowa State University under contract # DE-AC02-07CH11358.

References

CMake. (2024). https://cmake.org/

Crandall, Z., Windus, T. L., & Richard, R. M. (2024). CMaize: Simplifying inter-package modularity from the build up. *The Journal of Chemical Physics*, *160*(9), 092502. https://doi.org/10.1063/5.0196384

Eigen. (2024). https://eigen.tuxfamily.org/

Giordano, M. (2016). Uncertainty propagation with functionally correlated quantities. *ArXiv e-Prints*. https://arxiv.org/abs/1610.08716

Lebigot, E. O. (2009). Uncertainties: A python package for calculations with uncertainties. In *GitHub repository*. GitHub. https://github.com/lmfit/uncertainties