Julia for Applied Infectious Disease Modelling

Sam Abbott[1,*] Simon Frost[1,2] Sebastian Funk[1]

Abstract

Infectious disease modelling plays a critical role in public health decision-making, from outbreak response to long-term planning. The field faces mounting challenges as models grow increasingly complex, incorporate greater ranges of real-time data streams, and explore more intervention scenarios. Current approaches require either sacrificing performance for ease of use in high-level languages like R and Python, or accessibility for speed in low-level implementations. Domain-specific tools provide targeted solutions but lack integration with broader scientific computing advances, whilst general-purpose frameworks have steep learning curves, and often don't have all the features needed for applied infectious disease modelling. Julia addresses these challenges through its combination of multiple dispatch, native performance, and coherent scientific ecosystem. We examine Julia's key features and summarise its ecosystem for infectious disease modelling, spanning the SciML ecosystem, JuMP, AlgebraicJulia, Turing.jl, Gen.jl and RxInfer.jl, other model fitting options, neural networks, agent-based modeling, automatic differentiation, and data science tools. Through three case studies, we show how these components integrate to support complex epidemiological models. Julia offers a unique combination of performance, expressiveness, and composability that addresses limitations of both pipeline and monolithic modelling approaches.

[1] Centre for Mathematical Modelling of Infectious Diseases, London School of Hygiene & Tropical Medicine

[2] Microsoft Discovery & Quantum, Microsoft

[*] Correspondence: Sam Abbott <sam.abbott@lshtm.ac.uk>

1 Introduction

Infectious disease modelling (IDM) plays a critical role in public health decision-making, from outbreak response to long-term planning. The field faces mounting challenges as models grow increasingly complex, incorporate a greater range of real-time data streams, and explore more intervention scenarios. This complexity manifests in both computational demands and code maintenance challenges, making it difficult to produce impactful work within the timescales that matter when responding rapidly to emerging threats.

Modern outbreak response often requires rapid development and deployment of novel methods, frequently combining multiple modelling approaches to address emerging threats. This challenge is highlighted by tools like EpiNow2 [1,2] and EpiNowcast [3], which despite their success, have struggled to integrate novel features and methodological advances from the broader modelling community. The difficulty in incorporating external contributions and specialised backend tools, even for established projects, highlights a challenge in the field. While approaches like epidist [4], which builds on top of brms, demonstrate one path toward integration, such solutions are often limited in scope and only available for certain model types.

There are all so challenges in wider epidemiological research practice. The implementation of Bayesian workflows remains particularly challenging for many practitioners, requiring expertise in both statistical methodology and computational techniques. Furthermore, the current paradigm of research study development often requires teams to rebuild complex model frameworks from scratch, leading to duplicated effort, potential inconsistencies, and slower scientific progress. Even with advances in computing power, the persistent tension between computational performance and accessibility continues to impede both rapid outbreak response and longer-term research projects, where sophisticated models must be balanced against practical constraints of time and expertise.

Several solutions have been proposed to address these challenges, including domain-specific tools like odin and pomp, as well as general-purpose probabilistic programming languages like Stan [5] and JAX [6]. While these tools have made important contributions, they often face significant limitations. Domain-specific solutions, while powerful for their intended use cases, typically lack integration with broader scientific computing ecosystems, making it difficult to leverage advances in fields like differential equations or optimization. Meanwhile, general-purpose tools like Stan [5] and JAX [6], despite their mathematical sophistication, can be challenging to use effectively without substantial programming expertise. Furthermore, the development of efficient algorithms and implementation patterns often falls to domain experts rather than being handled by dedicated numerical computing specialists, leading to potential inefficiencies and missed opportunities for optimization.

This paper explores how Julia [7] addresses these challenges through its design philosophy and scientific ecosystem. We examine Julia's key features, including multiple dispatch, native performance, and metaprogramming, that enable efficient implementation of epidemiological models whilst maintaining code accessibility. We summarise Julia's ecosystem for infectious disease modelling, spanning the SciML ecosystem, JuMP, AlgebraicJulia, Turing.jl, Gen.jl and RxInfer.jl, other model fitting options, neural networks, agent-based modeling, automatic differentiation, and data science tools. Through three case studies, we show how these components integrate to support complex epidemiological models. Finally, we discussion of Julia's potential to address current challenges in infectious disease modelling and directions for future work.

2 Key features of the Julia language

- Multiple Dispatch and Type System
- Native Performance
- Metaprogramming
- Package Interoperability
- Interactive
- Auto-differentation

3 The Ecosystem

3.1 The Scientific Machine Learning Ecosystem (SciML)

3.2 Julia for Mathematical Programming (JuMP)

3.3 Applied Category Theory in Julia (AlgebraicJulia)

- EpiCats.jl

Applied epidemiological models can become complex, incorporating heterogeneities such as age, risk factors, and spatial location, which can make these models difficult to understand, and difficult to debug if the model does not behave as expected. One solution to this problem is to build complex models from simpler ones, with the idea that small, simple models can be understood. Applied category theory (ACT) has been proposed as one mechanism that can 'add' models together, e.g. adding a transition from susceptibles to immune to model vaccination, as well as 'multiply' models, for example combining an SIR model with a model of age classes to generate an age-structured model. The mathematics underlying ACT provide guarantees that the resulting models are correct by construction, while the ability to build up a model incrementally provides a clearer understanding of the model assumptions. ACT was originally applied to composing ordinary and delay differential equation models, as well as difference equation models.

A central data structure in AlgebraicJulia is the attributed C-set (`ACSet`, pronounced ‘ah-chet’), a joint generalization of a graph and a dataframe. ACSets have been used to represent Petri net models as well as stock-and-flow models, in addition to other areas where problems can be represented as wiring diagrams, including optimization, optimal control, databases, and planning.

3.4 Turing.jl

- Turing.jl [8]
- ecosystem
- PPL
- DynamicPPL
- JuliaBugs
- Inference
- Other cool stuff?

3.5 Gen.jl and RxInfer.jl

In addition to Turing.jl, there are other probabilistic programming languages in Julia that take a different approach to Turing.jl.

- Gen.jl [9]

3.6 Other Model Fitting Options

- Pigeons.jl parallel tempering - PPL agnostic
- Others?

3.7 Neural networks

- Lux.jl https://lux.csail.mit.edu/stable/introduction/overview

3.8 Agent-Based Modeling

Agents.jl [10] provides a comprehensive framework for agent-based modeling in Julia. The package offers high-performance simulations with minimal code complexity, supporting both continuous and discrete agent-based models. Its integration with the broader Julia ecosystem enables seamless incorporation of differential equations, optimisation, and statistical analysis within agent-based simulations, making it particularly valuable for epidemiological modeling where individual-level interactions drive population-level dynamics.

3.9 Automatic Differentiation

- Able to switch to different auto differentiation approaches
- Enables picking the one that works best for your problem
- Also allows for benefiting from ongoing research as can swap in new options as they become available.
- Forward differentiation and reverse differentiation
- Enzyme, Mooncake

3.10 Data Science Tools

Julia provides a comprehensive data science ecosystem that complements its scientific computing capabilities. DataFrames.jl and DataFramesMeta.jl offers powerful tabular data manipulation, while visualization libraries like Makie.jl. The AlgebraOfGraphics.jl (AoG) package provides a grammar of graphics system, and the Tidier.jl ecosystem brings familiar data wrangling patterns to Julia users. These tools integrate seamlessly with the modeling and inference capabilities discussed above, enabling end-to-end workflows from data preparation through analysis and visualization.

4 Case Studies

Julia has already been used successfully in several epidemiological modelling studies, demonstrating its growing adoption within the infectious disease modelling community. A notable example is the Epirecipes project [11], originally a multi-language collection of epidemiological models, which has increasingly focused on Julia implementations. The project serves as a “cookbook” of epidemiological models, showcasing Julia’s versatility through diverse approaches to classical models like SIR: from compositional modelling using AlgebraicJulia’s category theory framework (EpiCats), to Bayesian parameter estimation with Gen.jl and Turing.jl, to advanced stochastic processes and bifurcation analysis. This demonstrates how Julia’s ecosystem enables researchers to explore the same epidemiological concepts through multiple complementary computational paradigms whilst maintaining code reusability and mathematical rigour. The following case studies build upon this foundation to illustrate specific aspects of Julia’s capabilities for infectious disease modelling.

4.1 Case Study 1: Compartmental Susceptible-Exposed-Infectious-Recovered (SEIR) Model with time-varying transmission rate

1. ODE Model Specification
   - Define SEIR differential equations using SciML ecosystem
   - Implement core compartmental structure
   - Validate numerical stability and solver choice
2. Time-varying Transmission Rate
   - AR(1) process model for transmission rate (t)
   - Connect latent AR process to ODE system dynamics
   - Leverage existing parameter validation from previous case
3. Observation Process
   - Apply negative binomial observation model
   - Extend observation layer to capture:
     - Incidence data from E$\rightarrow$I transitions
     - Prevalence data from I compartment
4. Complete Model Integration
   - Combine deterministic ODE backbone with stochastic components
   - Link transmission process to compartmental dynamics
   - Connect compartmental states to observation layer
   - Implement efficient Bayesian inference pipeline
5. Model Validation
   - Simulate data from the model
   - Fit to the simulated data i.e. SBC
   - Simulate a known beta scenario
   - Fit and recover from it.

4.2 Case study 2: TBD

Potential options:

- Petri net composition of a SEIR model with age structure and variant dynamics. Simulate from and integrate into Turing for fitting. Something like: https://github.com/epirecipes/EpiCats/blob/main/pn_stratify_ageclasses/pn_stratify_ageclasses.md but extended.

- Agent.jl model to Gen.jl via Genify.jl (https://github.com/probcomp/Genify.jl)
- Neural ODEs: SEIR w/ AR + NN interaction terms: https://github.com/vboussange/HybridDynamicModels.jl
- UDEs SEIR with NN on multiple datasets: https://www.sciencedirect.com/science/article/pii/S240584402401394X

Some or all of the case studies likely need to include real data to convince applied modellers at the same time we want to keep the high level and schematic to make this achievable and to make sure we can communicate the key points.

4.3 Case study 3 : TBD

5 Discussion

This paper has examined Julia's capacity to address barriers in infectious disease modelling through its combination of performance, composability, and scientific ecosystem integration. We have outlined Julia's key features including multiple dispatch, native performance, and metaprogramming that enable efficient implementation of epidemiological models. We have summarised Julia's ecosystem spanning the SciML ecosystem, JuMP, AlgebraicJulia, Turing.jl, Gen.jl and RxInfer.jl, other model fitting options, neural networks, agent-based modeling, automatic differentiation, and data science tools that integrate to create a coherent environment for model development. Through our case studies, we have shown how these components support compartmental models with time-varying transmission, combining deterministic dynamics with stochastic processes and Bayesian inference. While challenges remain in compilation times and community size, Julia provides a coherent approach to infectious disease modelling that eliminates the traditional trade-off between accessibility and performance.

- Demonstrated Julia's unique position in scientific computing for IDM through multiple dispatch, native performance, and seamless package interoperability
- Showed how ecosystem components work together cohesively, enabling "two-language problem" elimination from prototyping to production
- Illustrated practical benefits through case studies spanning differential equations, Bayesian inference, and automatic differentiation
- True language-level composability between packages enables rapid model development and

component reuse, though requires learning curve for researchers familiar with monolithic approaches in R/Python

- Rich type system enabling both safety and performance facilitates systematic model validation, yet compilation times can impact interactive development workflow
- Growing, academically-focused ecosystem provides cutting-edge scientific computing tools, however IDM-specific tooling remains limited compared to established R/Python packages
- Native performance eliminates computational bottlenecks common in other high-level languages, but smaller community means fewer domain-specific examples and tutorials

While JAX [6] offers automatic differentiation and acceleration, it remains a framework rather than a language solution, leading to limited expressiveness due to Python's underlying type system, restricted functional programming patterns, and complex deployment challenges. Julia's multiple dispatch provides more natural problem expression for epidemiological models, particularly when implementing complex infectious disease models with feedback loops, integrating real-time data streams for forecasting, or mixing deterministic and stochastic model components.

Traditional probabilistic programming languages like Stan [5] suffer from limited flexibility in model specification, separation between modelling and analysis environments, and performance overhead from interface translations. Julia's native probabilistic programming through Turing.jl addresses these limitations whilst supporting stochastic differential equations, discrete event simulation, and agent-based modelling approaches that are essential for comprehensive infectious disease analysis.

Double censored delays are common in infectious disease modelling as most epidemiological delays have censored primary and secondary event [12,13]. Tools to model these delays are a good example of the potential benefits to the IDM ecosystem of adopting Julia. The R package primarycensored [14] addresses this by providing both R functions and separate Stan implementations, as well as tools for injecting the Stan code into custom models. CensoredDistributions.jl is a pure Julia version of the same solution [15] and uses multiple dispatch to automatically select between analytical solutions and numerical methods based on distribution type, without requiring separate implementations for different inference backends or custom dispatch methods. The package integrates natively with Turing.jl and other Julia PPLs, eliminating the need for language-specific implementations. It uses the SciML ecosystem for solving numerical integrals which works both when used as a standalone package and within a Turing.jl model. In contrast, the Stan and R implementations need to use different solvers with the Stan implementation having to be recast into an ODE in order to work within the Stan ecosystem. In addition to these scientific benefits the package can make use of the standard Julia tools for unit testing, documentation, and benchmarking all of which can be challenging in less full featured PPLs such as Stan. In addition to these scientific benefits, the package can make use of the standard Julia software development tools for unit testing, and documentation both of which can be challenging in less full-featured PPLs such as Stan.

Areas for future work include developing a cohesive epidemiology metapackage that provides domain-specific abstractions for common IDM patterns whilst connecting to the broader Julia ecosystem. Priority areas include standardised interfaces for model specification, parameter estimation, intervention modelling, and real-time forecasting. Technical improvements needed include further reducing compilation latency, expanding GPU support across the ecosystem, and developing IDM-specific visualisation recipes alongside standardised benchmarks for IDM applications.

Julia represents a significant advance in scientific computing for IDM, offering a unique combination of performance, expressiveness, and composability that addresses fundamental limitations of both pipeline and monolithic modelling approaches. Whilst some challenges remain, the foundation is set for a robust, maintainable, and efficient approach to infectious disease modelling that can adapt to future computational and methodological advances. Action must be taken to provide the necessary support for Julia adoption in the IDM community including composable, interoperable, and performant tools, educational materials, and community engagement.

6 Acknowledgements

References

1. Abbott S, Hellewell J, Thompson RN, Sherratt K, Gibbs HP, Bosse NI, et al. Estimating the time-varying reproduction number of SARS-CoV-2 using national and subnational case counts [version 2; peer review: 1 approved, 1 approved with reservations]. Wellcome Open Research. 2020;5. doi:10.12688/wellcomeopenres.16006.2

2. Abbott S, Hellewell J, Sherratt K, Gostic K, Hickson J, Badr HS, et al. EpiNow2: Estimate real-time case counts and time-varying epidemiological parameters. Zenodo; 2020. doi:10.5281/zenodo.3957489

3. Abbott S, Lison A, Funk S, Pearson C, Gruson H, Guenther F, et al. Epinowcast: Hierarchical nowcasting of right censored epidemiological counts. Zenodo; 2021. doi:10.5281/zenodo.5637165

4. Howes A, Park SW, Abbott S. Epidist: Estimate epidemiological delay distributions with brms. https://github.com/epiversehub/epidist; 2024.

5. Carpenter B, Gelman A, Hoffman MD, Lee D, Goodrich B, Betancourt M, et al. Stan: A probabilistic programming language. Journal of Statistical Software. 2017;76: 1–32. doi:10.18637/jss.v076.i01

6. Bradbury J, Frostig R, Hawkins P, Johnson MJ, Leary C, Maclaurin D, et al. JAX: Composable transformations of Python+NumPy programs. 2018. Available: http://github.com/google/jax

7. Bezanson J, Edelman A, Karpinski S, Shah VB. Julia: A fresh approach to numerical computing. SIAM review. 2017;59: 65–98.

8. Fjelde TE, Xu K, Widmann D, Tarek M, Pfiffer C, Trapp M, et al. Turing.jl: A general-purpose probabilistic programming language. ACM Transactions on Probabilistic Machine Learning. 2025. doi:10.1145/3711897

9. Cusumano-Towner MF, Saad FA, Lew AK, Mansinghka VK. Gen: A general-purpose probabilistic programming system with programmable inference. Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation. ACM; 2019. pp. 221–236. doi:10.1145/3314221.3314642

10. Datseris G, Vahdati AR, DuBois TC. Agents.jl: A performant and feature-full agent-based modeling software of minimal code complexity. SIMULATION. 2022;98: 451–467. doi:10.1177/00375497211068820

11. Epirecipes Contributors. Epirecipes: A cookbook of epidemiological models. https://github.com/epirecipes; 2024.

12. Park SW, Akhmetzhanov AR, Charniga K, Cori A, Davies NG, Dushoff J, et al. Estimating epidemiological delay distributions for infectious diseases. medRxiv. 2024. doi:10.1101/2024.01.12.24301247

13. Charniga K, Park SW, Akhmetzhanov AR, Cori A, Dushoff J, Funk S, et al. Best practices for estimating and reporting epidemiological delay distributions of infectious diseases. PLoS computational biology. 2024;20: e1012520.

14. Abbott S, Brand S, Pearson C, Funk S, Charniga K. Primarycensored: Primary event censored distributions. 2025. doi:10.5281/zenodo.13632839

15. Abbott S, Bayer D, Brand S, DeWitt M, Lemaitre J. CensoredDistributions.jl: Censored distributions for julia. 2025. Available: https://censoreddistributions.epiaware.org/