A perspective: Julia for Biologists

Elisabeth Roesch, Adam MacLean, Joe Greener, Huda Nassar, Chris Rackauckas, Timothy Holy, Michael Stumpf.

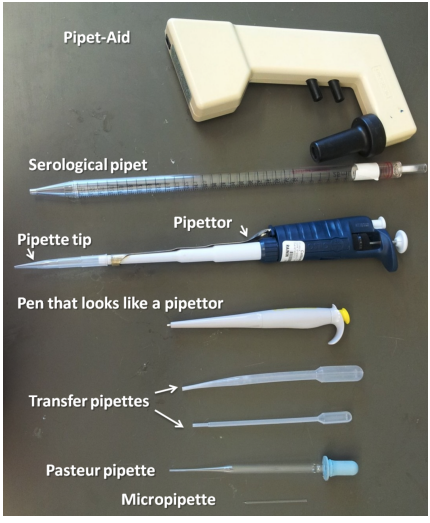
30th of July, 2021

Further resources and contact details

1. GitHub repository: `ElisabethRoesch/Perspective_Julia_for_Biologists`
   1.1 Examples
   1.2 Dynamic collection of links to further resources
   1.3 *Soon* link to corresponding publication
2. Contact
   - email: eroesch@student.unimelb.edu.au
   - twitter: Roesch_E

Content

Introduction: Tools of a biologist



- *1. Tool group: Conducting experiments and collecting data.*
- Optimised and specialised to perform a given task in the most efficient and effective way possible.

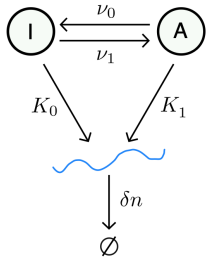
[1] Picture: http://cellularscale.blogspot.com/2013/01/the-cellular-guide-to-pipettes.html, 13/7/21.

Introduction: Tools of a biologist

- *2. Tool group: Computers.*
- Computers form also part of a biologist's toolkit.
- Again, different types exist.

[1] "A researcher uses a confocal microscope equipped for super-resolution and fluorescence imaging at the University of Bristol, UK" Picture: https://www.nature.com/articles/d41586-019-03650-w, 13/7/21.

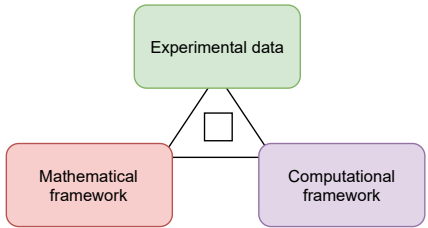
Introduction: Tools of a biologist



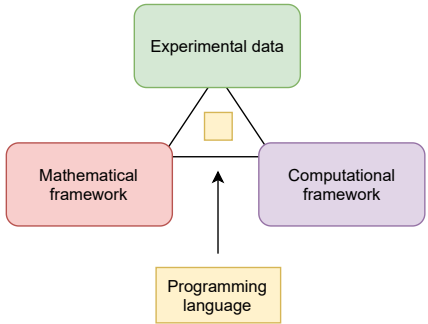
- *3. Tool group: Mathematical models.*
- Mathematical models allow us to integrate theoretical knowledge of a given system into our study.

[1] Picture: Ham, Lucy, et al., Extrinsic Noise and Heavy-Tailed Laws in Gene Expression, Phys. Rev. Lett., 2020.

Introduction: Tools of a biologist



Introduction: Tools of a biologist



- *4. Tool group: Programming language.*
- The programming language connects the three cornerstones of the experimental data, the mathematical framework and the computational framework.

Programming languages in biology

The two-language approach:

- First draft: R/Python ("easy to use" language)
- Performance needed: C/C++ ("harder to use" language)

This is a two-language problem:

- Inefficient
- Effective? Errors, misinterpretation, durability, etc.

So why are people doing this?

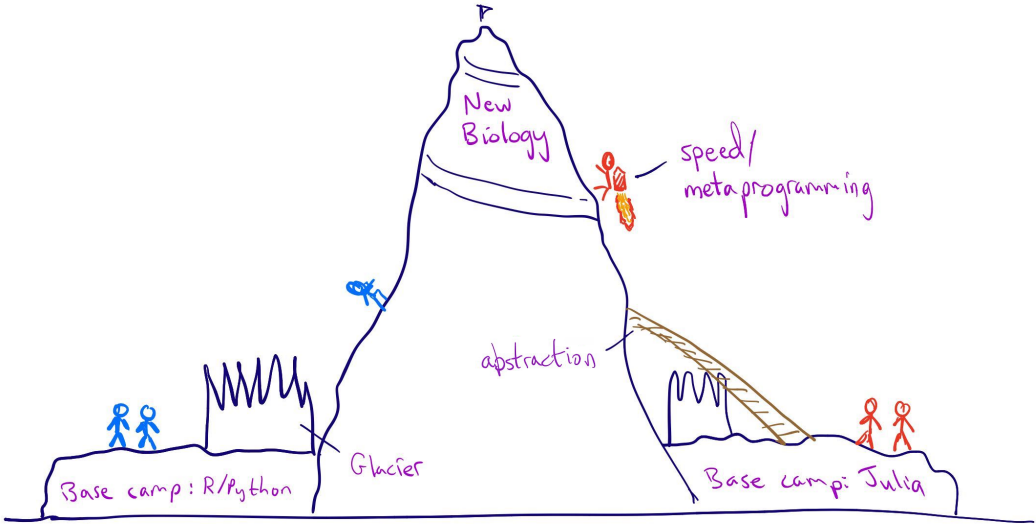
- We believe: Usability of high performance languages is bottleneck for biologists.

Julia challenges the two-language problem!

- Looks like Python, runs like C.
- One reason why Julia is the perfect programming language for biologists.

Figure: Word cloud of answers to the survey question: "Where do you see the biggest advantage(s) of Julia for a biologist?"

Julia enables new biology.

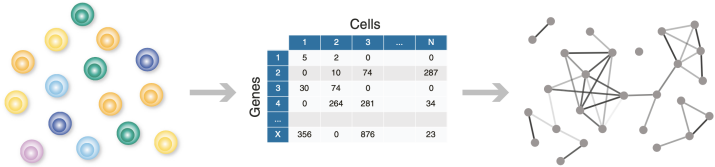


Why should a biologist care about speed?

Speed is important...

1. when analysing large data sets.
2. when modeling complex systems (larger models, more unknown/to be inferred parameters, etc.).
3. for the accuracy of a study when performing simulation-based approximations and optimisations.
4. for more reliable methodology as a fast programming language allows for more rigorous and extensive tests during development.
5. for clinical trials.

Speed: Example 1 – Single cell data and network inference

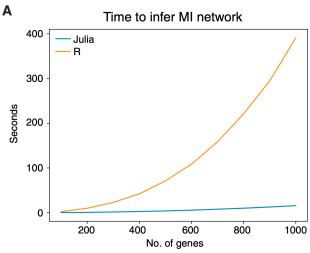


- Chan et al. [1] use Mutual Information (MI) to infer gene regulatory networks (GRNs) from transcriptomic single cell data of qPCR experiments.
- Compare times taken to calculate a matrix of pairwise MI values to the R package minet [2].

[1] Chan et al., Gene Regulatory Network Inference from Single-Cell Data Using Multivariate Information Measures, Cell Systems, 2017.

[2] Meyer et al., minet: A R/Bioconductor Package for Inferring Large Transcriptional Networks Using Mutual Information. BMC Bioinformatics, October 2008.

Speed: Example 1 – Single cell data and network inference



- Julia [1]: Data set with 100/1000 genes takes around 0.3 seconds/around 17 seconds.
- R [2]: Data set with 100/1000 genes takes around 1.5 seconds (5- fold difference)/390 seconds (>20-fold difference).
- Point of practical feasibility: Data set of 3500 genes and 600 cells. Julia takes 134 seconds, R runs for over 2.5 hours and finally reaches the hardware limitations of a regular working laptop.
- *More info on this: Paper [1] or JuliaCon 2017 talk by Thalia Chan on youtube.*

[1] Chan et al., Gene Regulatory Network Inference from Single-Cell Data Using Multivariate Information Measures, Cell Systems, 2017.

[2] Meyer et al., minet: A R/Bioconductor Package for Inferring Large Transcriptional Networks Using Mutual Information. BMC Bioinformatics, October 2008.

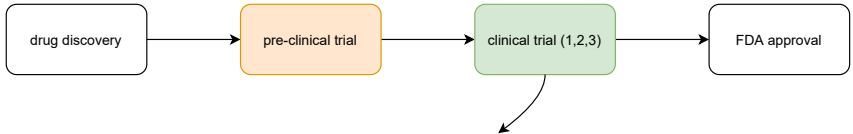
Speed: Example 1 – Intuition for speed difference

Example: Y=W*X+B *(Chain of vector expressions)*

1. NumPy *(calls optimized C code under the hood)*
   - Operations are computed sequentially:
     1.1 W*X calls a C code to produce temporary array with result
     1.2 tmp+Y calls a C code to produce desired Y
   - Allocating memory for tmp+Y and the result Y is O(n) and scales proportionally to compute cost → a major overhead.
2. Julia
   - Broadcast via . operator which signifies elementwise operations: Y.=W.*X.+B
   - Just-in-time (JIT) compilation + vectorized functions → Operator fusion.
     2.1 Fuses all nearby dots into a single function and JIT compiles this function in a loop.

→ NumPy makes two function calls + spends time generating 2 big arrays.
→ Julia makes one function call + smartly reuses existing memory.

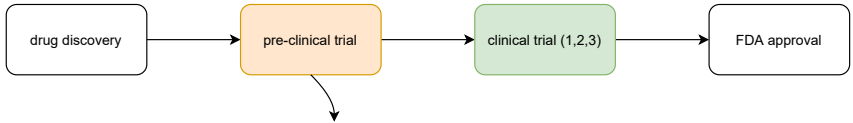
Speed: Example 2 – Pharmacology and dynamical systems



1. Modeling and simulation within clinical trial transformed the pharmacoeconomics of drug discovery:
   $\rightarrow$ lowering the risk of failed trials
   $\rightarrow$ increased equity in drug development
   2013: Estimated saving by model-informed drug development:
   - Pfizer Inc. an annual \$100 million
   - Merck & Co an annual \$500 million

[1] EFPIA MID3 Workgroup, SF Marshall, et al. Good practices in model-informed drug discovery and development: practice, application, and documentation. CPT: pharmacometrics & systems pharmacology, 2016.

Speed: Example 2 – Pharmacology and dynamical systems

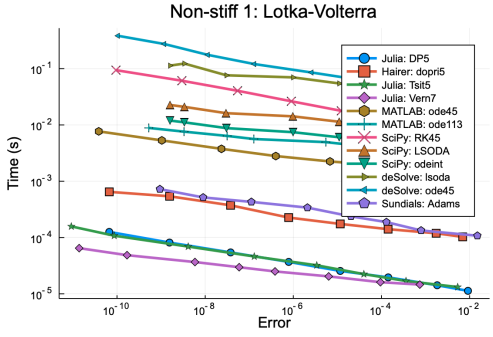


- Most trials do not undergo pre-clinical analysis:
  $\rightarrow$ Reason: Time! Any delay in the start of the clinical trial creates an increase in the overall cost.
- Pfizer's quantitative systems pharmacology team with MIT researchers:
  $\rightarrow$ **175x acceleration** to a production model analyses in the preclinical drug development pipeline.
  $\rightarrow$ by translating the model from a combination of C and MATLAB to pure Julia.

[1] Rackauckas et al. Accelerated predictive healthcare analytics with pumas, a high performance pharmaceutical modeling and simulation platform. bioRxiv, 2020.

Speed: Example 2 – Technical background

Model: Dynamical models such as (nonlinear) ordinary differential equations (ODEs) and Gillespie simulations.
ODE solvers benchmark: solver time and error in order to ensure correctness of interpretations.



- Julia → OrdinaryDiffEq.jl package
- FORTRAN → Hairer dopri5
- MATLAB → MATLAB ordinary differential equation solvers
- Python → SciPy solve_ivp module
- R → R deSolve package
- C → C librarySundials

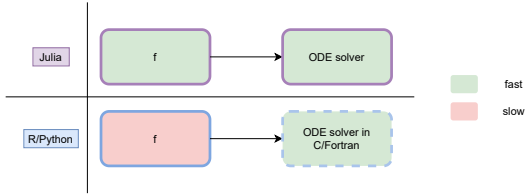
[1] https://benchmarks.sciml.ai/html/MultiLanguage/wrapper_packages.html

Speed: Example 2 – Intuition for speed difference

- ODE with state U, time t and function f describing the derivative of the system

$$\frac{dU}{dt} = f(U).$$

- User defines f which does contain scalar operations are limiting factor in overall performance.

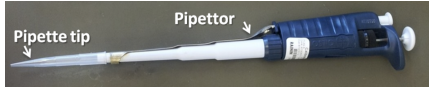


- In Python/R $\rightarrow$ High function call overhead (Python vs. Julia: 350ns and 5ns [1]).

[1] https://ilovesymposia.com/2015/12/10/the-cost-of-a-python-function-call/

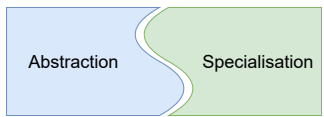
What is abstraction in context of a programming language?

- Analogy to Pipettor
  → Made with slightly different designs by each manufacturers but all perform the same task (same abstract interface).



- Julia: Human
  → picking up any specific pipettor and customizing every aspect of grip, pressure, and other movements
  → use new pipettor with optimal efficiency.
- Most other languages are more specialised to a specific type of pipettor.
- Python: 1990s-era robot
  → designed for Eppendorf pipettors and now given a Rainin → needs to recalibrate itself before every single movement.

What is abstraction in Julia?

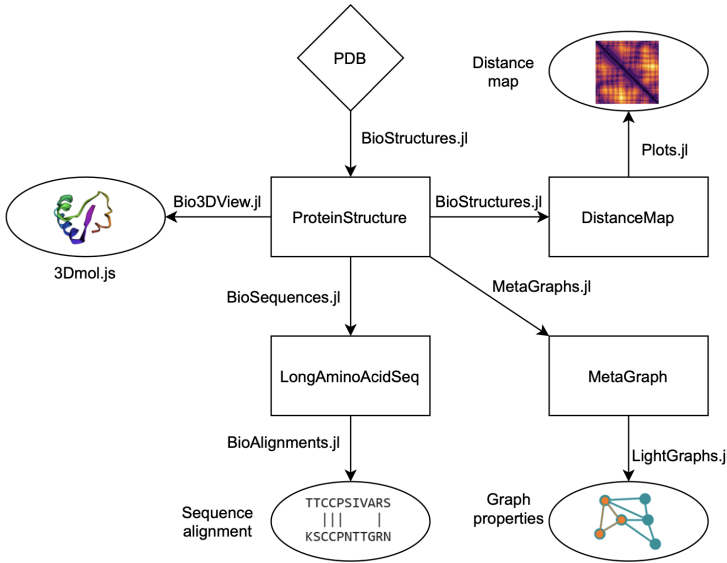


→ Allows Julia to exploit many implementations with different internal characteristics without a performance penalty.
→ Flexibility: biologists can implement models at an appropriate degree of complexity, without compromising on performance, and with the code being reusable in other contexts.

Why should a biologist care about abstraction?

1. **Heterogeneous** nature of biological data
   $\rightarrow$ Abstraction enables easy and elegant reuse of existing code.
   $\rightarrow$ Helpful to design, realise and implement biological studies faster and more robustly.
2. Abstraction allows for **intuitive representations** of biological problems (e.g. types).
   $\rightarrow$ Reduces the chance of misinterpretation and mistakes.
3. Enables and encourages easy **contributions and collaborations** between scientists.

Abstraction: Example 1 – Structural Bioinformatics and code reuse

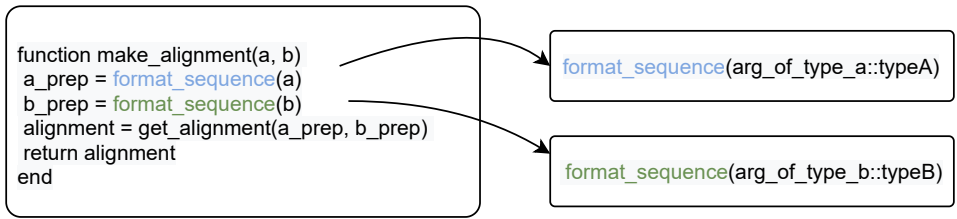


Abstraction: Technical background – Expression problem and multiple dispatch [1]

Expression problem:

- Can you extend you data model and operations nicely?
  - Can you easily integrate new types?
  - Can you easily integrate new operations?

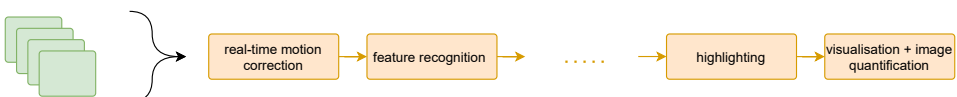
→ With multiple dispatch, yes!



[1] Talk Stefan Karpinski: https://www.youtube.com/watch?v=kc9HwsxE1OY&ab_channel=TheJuliaProgrammingLanguage

Abstraction: Example 2 – Image processing

- Fundamental to disciplines such as cell biology and neuroscience.
- Images from biology and their representation in Julia
  - Represented as arrays with high diversity and large size.
    → Several dimensions: Spatial and temporal dimensions, color channels, etc.
    → Up to multiple terabytes.



- Image processing pipeline to analyse data.
  - Pipelines often consist of many steps.
    → Nearly-endless series of altered versions of movie on disk?

→ Requires software that can handle this diversity and size in the data.

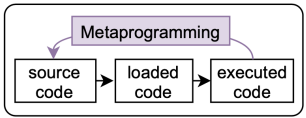
Abstraction: Example 2 – Intuition for lazy operations.

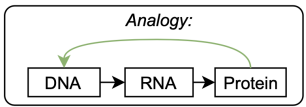
Example: Function f(img[x,y,t]) modifies the intensity of a pixel. (contrast)

```
#Eager:
f(img[x,y,t])
#Lazy:
img2 = mappedarray(f, img)
```

- **Eager** operations
  → gets run **immediately**
- **Lazy** operations
  → Creates img2 is instantaneous.
  → But computation is performed **when accessing** img2[x, y, t] returning f(img[x, y, t]) referencing the original f and img.
  → Creating img2 takes virtually no memory **regardless of the size of img**.

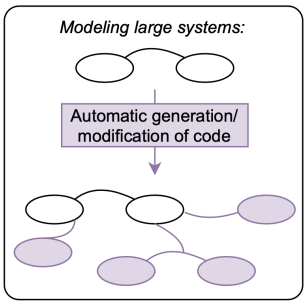
Julia's language design: Metaprogramming





What is metaprogramming?

- Code that modifies code.
- Computer code forms a part of the data.



Why should a biologist care about metaprogramming?
→ Anywhere where manual coding is impractical/infeasible.

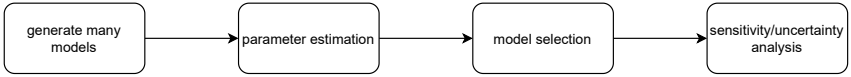
1. Generating many models (e.g. assumptions unclear)
2. Generating large models

Example 1: Metaprogramming in biological sciences – Biochemical reaction networks.

- Enzymatic process with enzyme E, substrate S, complex C, product P.

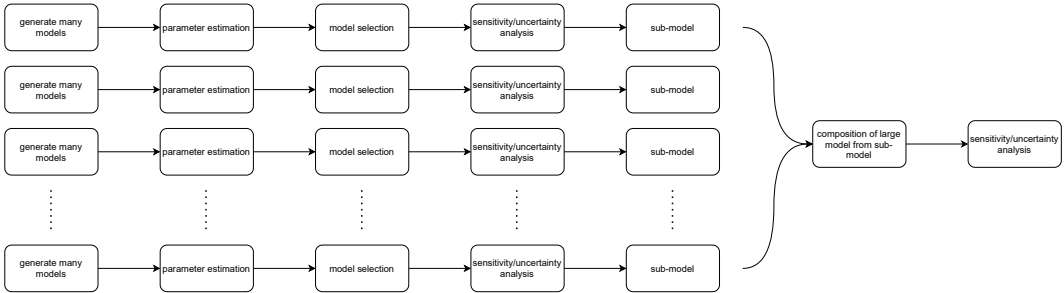
$$E + S \underset{k_2}{\overset{k_1}{\rightleftharpoons}} C \xrightarrow{k_3} E + P$$

- In a metaprogramming context:



[1] https://github.com/alavendelm/julia-dynsys-resources

Example 2: Metaprogramming in biological sciences – Whole cell modeling.



Summary: Why should a biologist use Julia?

The language design:

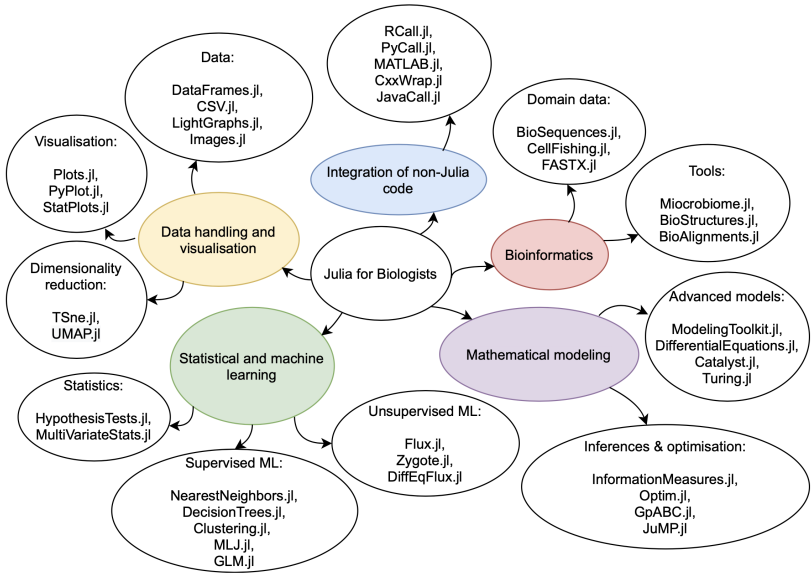
1. Julia is very user friendly. $\rightarrow$ It is easy to code.
2. Julia is a high performance language. $\rightarrow$ It is fast.
3. Julia offers a wide range of levels of abstraction. $\rightarrow$ It is flexible.
4. Julia can be used for metaprogramming. $\rightarrow$ It can code *automatically*.
5. Julia is not only good in one area but in many. $\rightarrow$ Enables "one language" projects.

Starting with/switching to Julia

The low barrier to entry:

1. Easy to learn due to intuitive semantics and easy to read syntax.
2. Accessible via various interfaces, REPL, IDE, or Jupyter notebook.
3. Existing non-Julia code can be easily integrated into new Julia projects via language specific packages.

The package ecosystem in Julia for biologists



More reasons why to use Julia as a biologist.

Additional reasons:

1. Julia's package ecosystem provides functionality for a wide range of oft-performed tasks in computational biology research.
2. Julia is free, open-source and hosted on GitHub.
3. Julia code is smoothly extendable which enables and encourages easy contributions and collaborations to/with existing projects, as well as writing, integrating and sharing new, user specific packages.
4. Julia offers (generally) excellent documentation, tutorials, and help available directly from active and welcoming community members via various communication channels such as Slack, discourse, twitter.

Acknowledgements

Authors: Elisabeth Roesch, Adam MacLean, Joe Greener, Huda Nassar, Chris Rackauckas, Timothy Holy, Michael Stumpf
GitHub repository: `ElisabethRoesch/Perspective_Julia_for_Biologists`

Tips from the community

Figure: Some survey answers to the questions "If you could give a biologist just starting Julia one tip, what would it be?"