Reproducibility and replicability of computer simulations

Konrad Hinsen

Centre de Biophysique Moléculaire, Orléans, France
Synchrotron SOLEIL, Saint Aubin, France

ACM REP ’24
20 June 2024

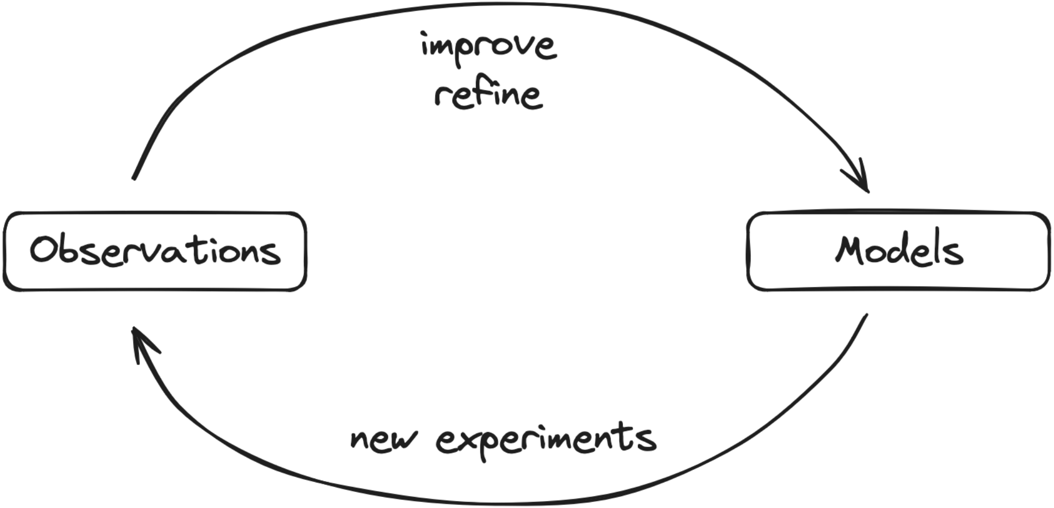
Science

Observations

→

Model

Observations and models



Computational models

Parameterized specifications for prediction algorithms

Computational models

Parameterized specifications for prediction algorithms

Specifications from theory

Computational models

Parameterized specifications for prediction algorithms

Specifications from theory

Parameters from observational data

Computational models

Parameterized specifications for prediction algorithms

Specifications from theory

Parameters from observational data

Physics, engineering: strong theory, few parameters
Deep learning: weak theory, many parameters

K. Hinsen, The Nature of Computational Models, Comp. Sci. Eng. **25**, 61-66 (2023)

Scenarios of computational science

Simulation: making predictions from models

Scenarios of computational science

Simulation: making predictions from models

Data analysis: interpret data using trusted models

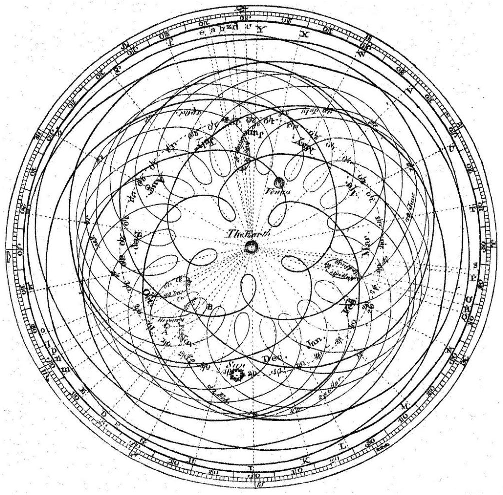
Scenarios of computational science

Simulation: making predictions from models

Data analysis: interpret data using trusted models

Data science: derive parameters for weak-theory models from data

Side note: data science isn't new



Source: Encyclopaedia Britannica (1st Edition, 1771)

Focus for today

Simulation

Focus for today

Simulation

- concentrate on rep[.*]bility issues related to **models**
- no distraction by **data** issues

R&R definitions

Reproducibility

is obtaining consistent results using the same input data, computational steps, methods, and code, and conditions of analysis.

Replicability

is obtaining consistent results across studies aimed at answering the same scientific question, each of which has obtained its own data.

Two studies may be considered to have replicated if they obtain consistent results given the level of uncertainty inherent in the system under study.

R&R definitions

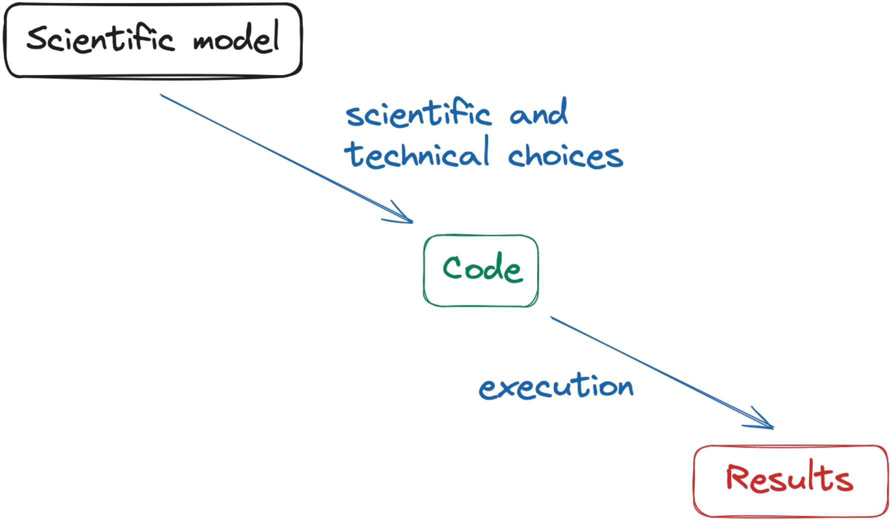
Reproducibility

is obtaining ~~consistent~~ **bit for bit identical** results using the same input data, computational steps, methods, and code, and conditions of analysis.

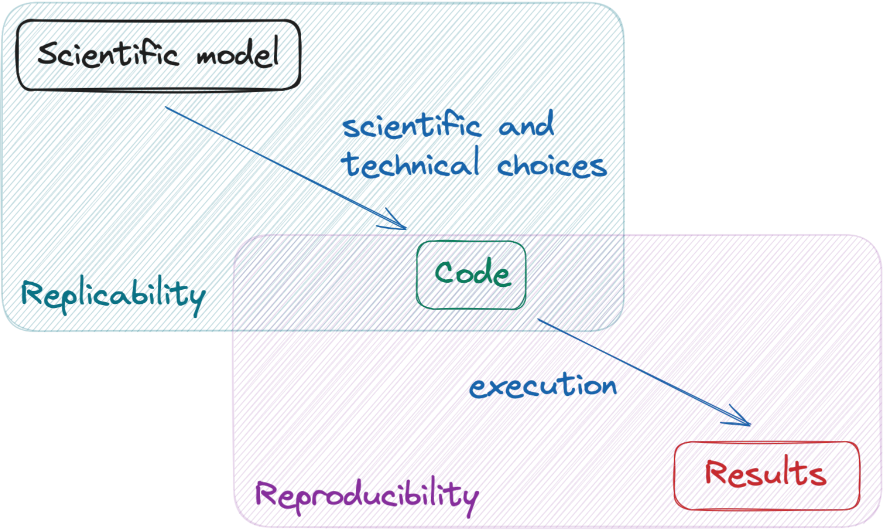
Replicability

is obtaining consistent results across studies aimed at answering the same scientific question, each of which has obtained its own data **and/or uses different code**.
Two studies may be considered to have replicated if they obtain consistent results given the level of uncertainty inherent in the system under study.

Model, code, results



Model, code, results



Scientific model

scientific and technical choices

Code

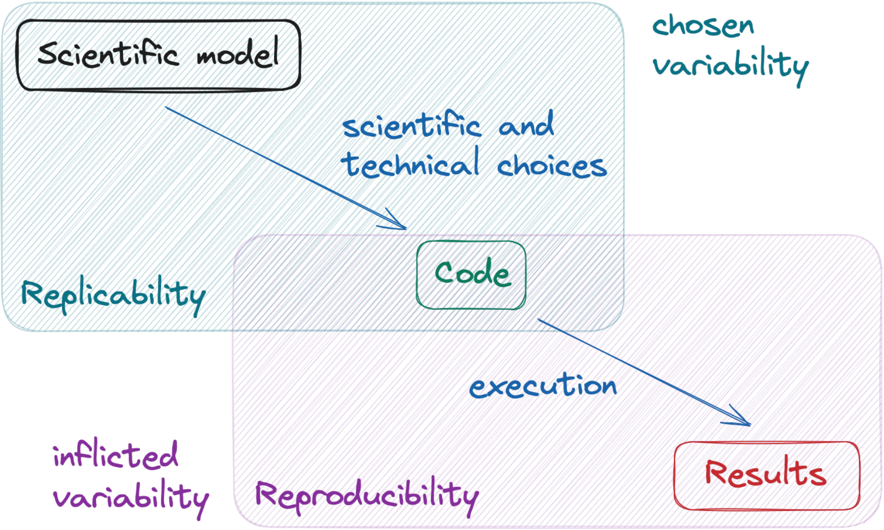
Replicability

execution

Results

Reproducibility

Model, code, results



chosen variability

inflicted variability

Why care about R&R in science?

Resolution of incompatible findings

Why care about R&R in science?

Resolution of incompatible findings

- A and B work in the same field

  collaborators, author/reviewer, competing teams, ...

Why care about R&R in science?

Resolution of incompatible findings

- A and B work in the same field

  collaborators, author/reviewer, competing teams, ...
- A finds X, B finds Y, X and Y are incompatible

  X, Y: observations, inferences, computed results, conclusions, ...

Why care about R&R in science?

Resolution of incompatible findings

- A and B work in the same field

  collaborators, author/reviewer, competing teams, ...
- A finds X, B finds Y, X and Y are incompatible

  X, Y: observations, inferences, computed results, conclusions, ...
- A and/or B want/need to resolve the conflict

  ideally: in collaboration, worst case: adversarial

Why care about R&R in science?

Resolution of incompatible findings

- A and B work in the same field

  collaborators, author/reviewer, competing teams, ...
- A finds X, B finds Y, X and Y are incompatible

  X, Y: observations, inferences, computed results, conclusions, ...
- A and/or B want/need to resolve the conflict

  ideally: in collaboration, worst case: adversarial

Involves both Rs.

Why care about R&R in science?

Resolution of incompatible findings

- A and B work in the same field

  collaborators, author/reviewer, competing teams, ...
- A finds X, B finds Y, X and Y are incompatible

  X, Y: observations, inferences, computed results, conclusions, ...
- A and/or B want/need to resolve the conflict

  ideally: in collaboration, worst case: adversarial

Involves both Rs.

Requires explorability down to the last details.

Why care about R&R in science?

Resolution of incompatible findings

- A and B work in the same field

  collaborators, author/reviewer, competing teams, ...
- A finds X, B finds Y, X and Y are incompatible

  X, Y: observations, inferences, computed results, conclusions, ...
- A and/or B want/need to resolve the conflict

  ideally: in collaboration, worst case: adversarial

Involves both Rs.
Requires explorability down to the last details.
Purely technical reproducibility is not sufficient.

Computation and its scientific context

What’s the result of this program?

`data_analysis.py`

```
from datalib import Dataset

points = [(1, 1), (-1, 1), (2, 4)]

data = Dataset()
for x, y in points:
    if x > 0:
        data.add_value(y)
print(data.average())
```

What’s the result of this program?

`data_analysis.py`

```python
from datalib import Dataset

points = [(1, 1), (-1, 1), (2, 4)]

data = Dataset()
for x, y in points:
    if x > 0:
        data.add_value(y)
print(data.average())
```

Expected answer: The average of y for the points with positive x → **2.5**.

What’s the result of this program?

data_analysis.py

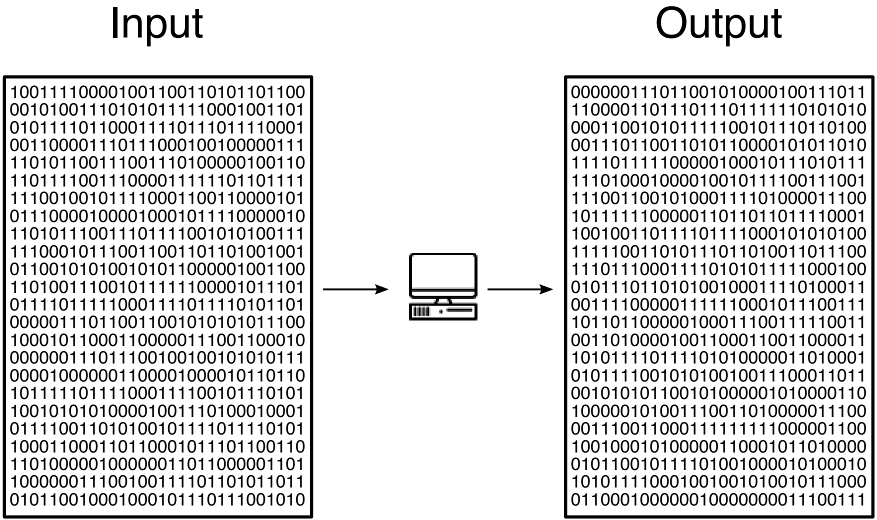
```
from datalib import Dataset

points = [(1, 1), (-1, 1), (2, 4)]

data = Dataset()
for x, y in points:
    if x > 0:
        data.add_value(y)
print(data.average())
```

Correct answer: **It depends on datalib**

What’s a computation?



Input

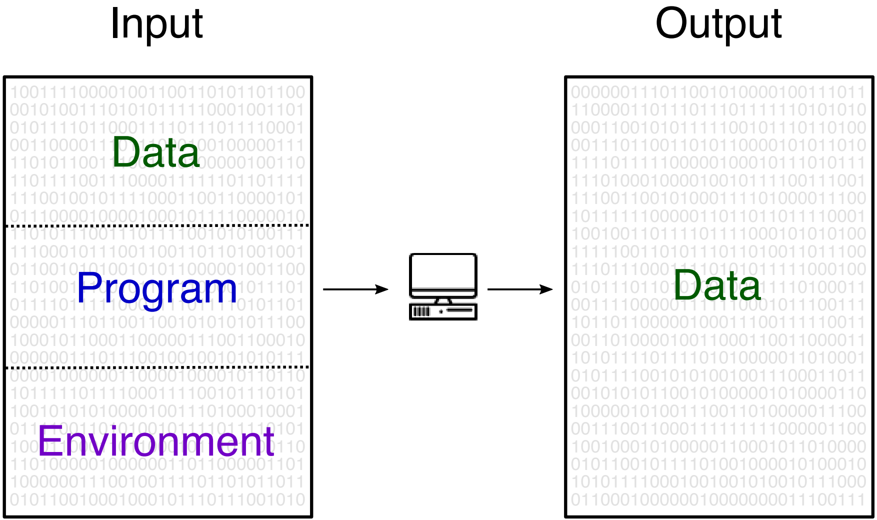
```
100111100001001100110101101100
001010011101010111110001001101
010111101100011110111011110001
001100001110111000100100000111
110101100111001110100000100110
110111100111000011111101101111
111001001011110001100110000101
011100001000010001011110000010
110101110011101111001010100111
111000101110011001101101001001
011001010100101011000001001100
110100111001011111100001011101
011110111110001111011110101101
000001110110011001010101011100
100010110001100000111001100010
000000111011100100100101010111
000010000001100001000010110110
101111101111000111100101110101
100101010100001001110100010001
011110011010100101111011110101
100011000110110001011101100110
110100000010000001101100001101
100000011100100111101101011011
010110010001000101110111001010
```

Output

```
000000111011001010000100111011
110000110111011101111110101010
000110010101111100101110110100
001110110011010110000101011010
111101111100000100010111010111
111010001000010010111100111001
111001100101000111101000011100
101111111000001101101101110001
100100110111101111000101010100
111110011010111011010011011100
111011100011110101011111000100
010111011010100100011110100011
001111000000111111000101110011
101101100000100011100111110011
001101000010011000110011000011
101011110111101010000011010001
010111100101010010011100011011
001010101100101000001010000110
100000101001110011010000011100
001110011000111111111000001100
100100010100000110001011010000
010110010111101001000010100010
101011110001001001010010111000
011000100000010000000011100111
```

Computer by Creative Stall from the Noun Project

What’s a computation?



Input

Data

Program

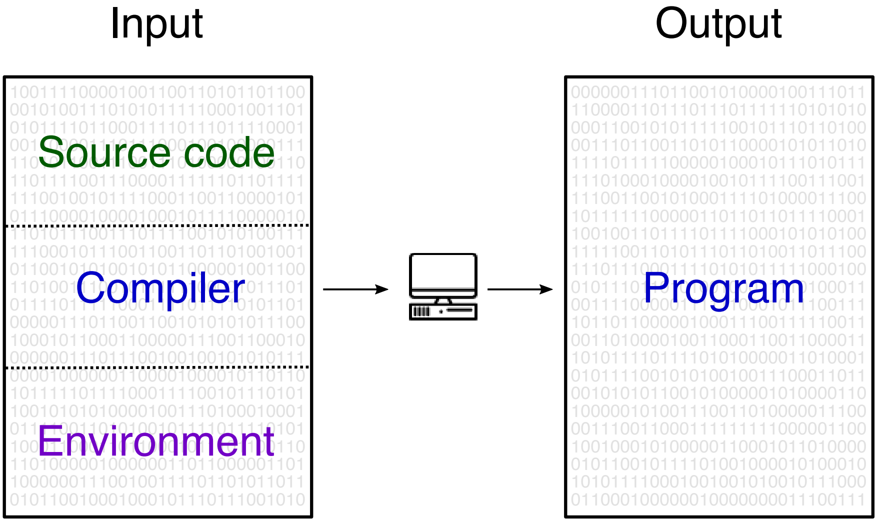
Environment

Output

Data

Computer by Creative Stall from the Noun Project

Programs are computed results



Input

Source code

Compiler

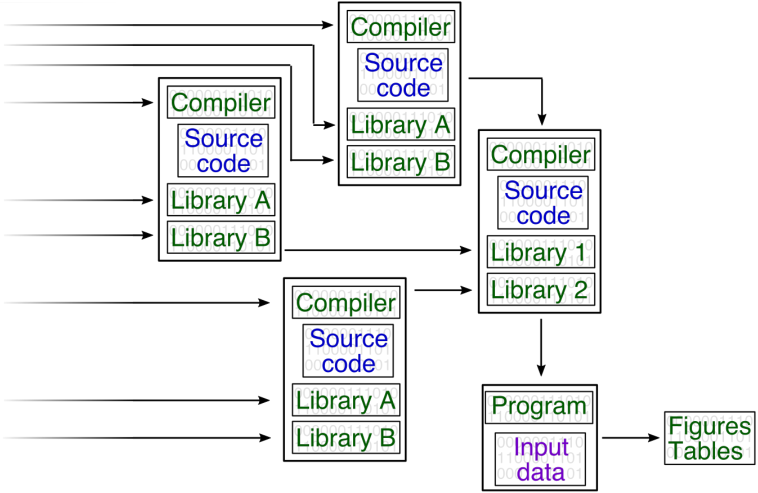
Environment

Output

Program

Computer by Creative Stall from the Noun Project

The provenance of computational results



Compiler
Source code
Library A
Library B

Compiler
Source code
Library A
Library B

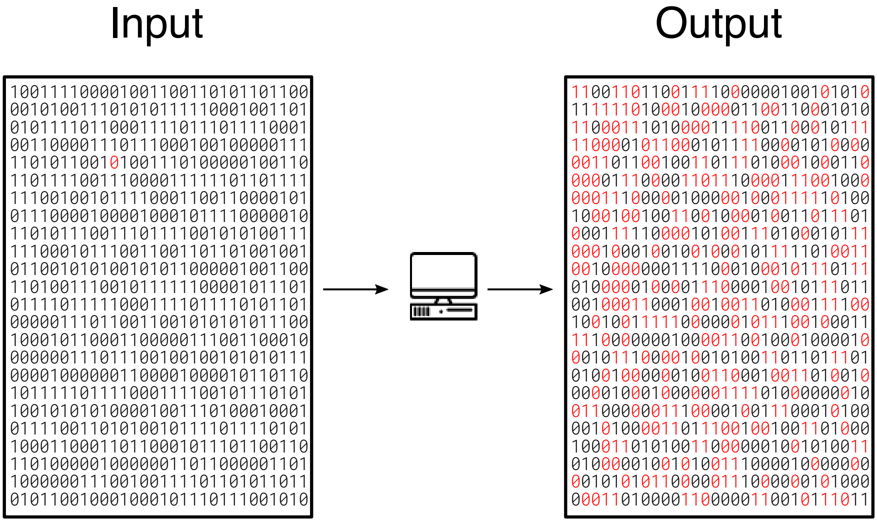
Compiler
Source code
Library 1
Library 2

Compiler
Source code
Library A
Library B

Program
Input data

Figures
Tables

Every bit matters



Computer by Creative Stall from the Noun Project

Are these bit patterns similar?

1100110011100111101001101110000 1100110011100111110000110000000

Are these bit patterns similar?

1100110011100111101001101110000 1100110011100111110000110000000

6673D370 6673E180

Are these bit patterns similar?

1100110011100111101001101110000	1100110011100111110000110000000
6673D370	6673E180
1718866800	1718870400

Are these bit patterns similar?

1100110011100111101001101110000	1100110011100111110000110000000
6673D370	6673E180
1718866800	1718870400
2.8785885e23	2.879237e23

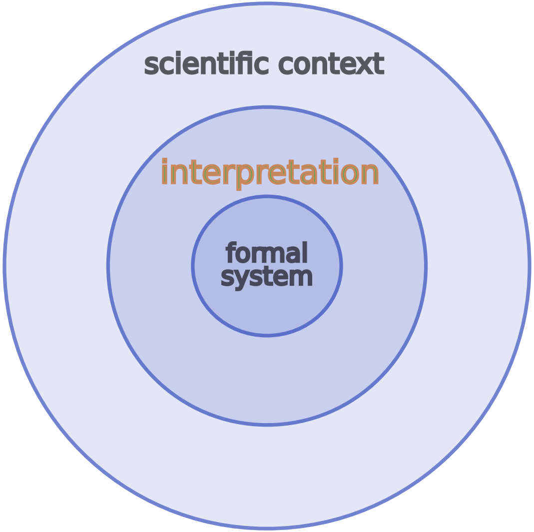
Are these bit patterns similar?

11001100111001111010011011110000	11001100111001111100001100000000
6673D370	6673E180
1718866800	1718870400
2.8785885e23	2.879237e23
2024-06-20T09:00:00.000000+02:00	2024-06-20T10:00:00.000000+02:00

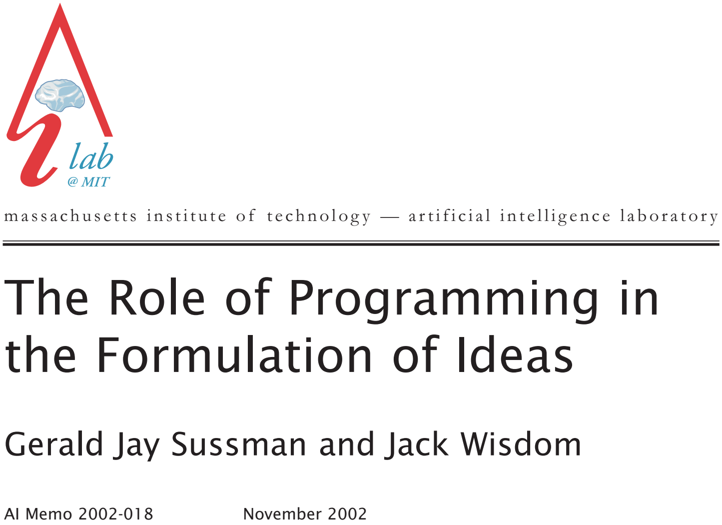
Formal systems

- Mechanical manipulation of **symbols** according to **fixed rules**
- Symbols in – symbols out: **no interpretation**

Formal systems in science



Formal systems in science



massachusetts institute of technology — artificial intelligence laboratory

The Role of Programming in the Formulation of Ideas

Gerald Jay Sussman and Jack Wisdom

AI Memo 2002-018 November 2002

Formal systems in science

> *It has often been said that a person does not really understand something until he teaches it to someone else. Actually a person does not really understand something until he can teach it to a computer, i.e., express it as an algorithm.*

Donald Knuth, Computer Science and its Relation to Mathematics, The American Mathematical Monthly 81, no. 4 (1974): 323–43

Bit-for-bit vs. “good enough”

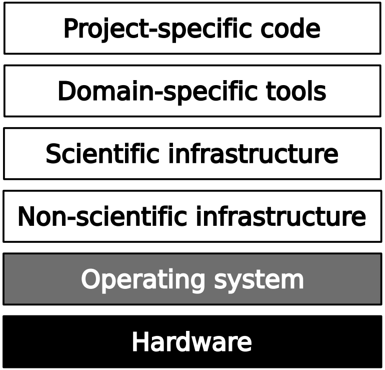
Bit-for-bit

- in the formal system
- yes-or-no answer
- automated tests
- can be ensured by infrastructure

Good enough

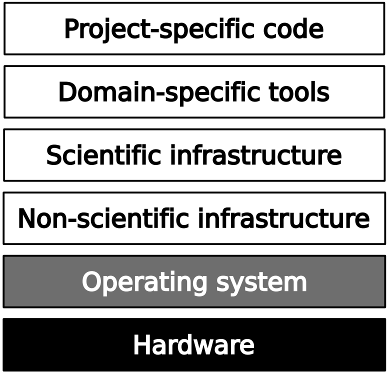
- interpretation
- depends on context
- expert judgement
- *a posteriori* verification

Reproducibility is an infrastructure problem



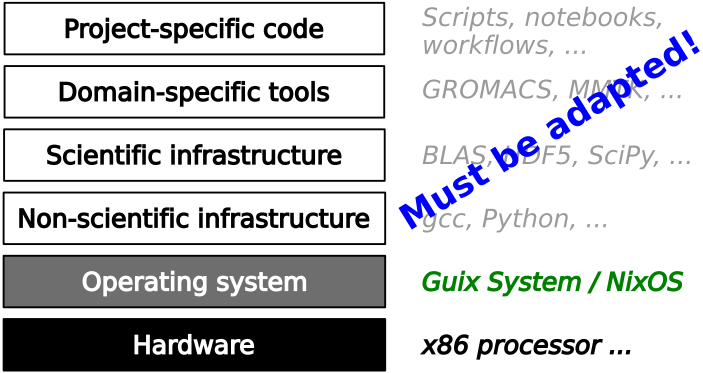
Project-specific code	***Scripts, notebooks, workflows, ...***
Domain-specific tools	***GROMACS, MMTK, ...***
Scientific infrastructure	***BLAS, HDF5, SciPy, ...***
Non-scientific infrastructure	***gcc, Python, ...***
Operating system	***GNU/Linux, ...***
Hardware	***x86 processor ...***

Reproducibility is an infrastructure problem



Project-specific code	*Scripts, notebooks, workflows, ...*
Domain-specific tools	*GROMACS, MMTK, ...*
Scientific infrastructure	*BLAS, HDF5, SciPy, ...*
Non-scientific infrastructure	*gcc, Python, ...*
Operating system	***Guix System / NixOS***
Hardware	***x86 processor ...***

Reproducibility is an infrastructure problem



Project-specific code	*Scripts, notebooks, workflows, ...*
Domain-specific tools	*GROMACS, MMTK, ...*
Scientific infrastructure	*BLAS, HDF5, SciPy, ...*
Non-scientific infrastructure	*gcc, Python, ...*
Operating system	***Guix System / NixOS***
Hardware	***x86 processor ...***

Must be adapted!

Reproducibility is a socio-economic problem

- Collective agreement
- Investments in infrastructure
- Institutional backing

Workarounds

- Freeze binaries of computational environments (containers, VMs)
  Container image / VM must also be reproducible
  for supporting **replicability**

Workarounds

- Freeze binaries of computational environments (containers, VMs)
  Container image / VM must also be reproducible
  for supporting **replicability**
- Cross-platform environment managers (conda, Spack, ...)
  Works for a few months
  Long-term stability requires controlling the full stack

Workarounds

- Freeze binaries of computational environments (containers, VMs)
  Container image / VM must also be reproducible
  for supporting **replicability**
- Cross-platform environment managers (conda, Spack, ...)
  Works for a few months
  Long-term stability requires controlling the full stack

Workarounds are necessary today, but harmful in the long run.

Questions on reproducibility

- Should computer simulations be made reproducible? Why?

  Yes. If I cannot reproduce your simulation, then I don't know what you have simulated.

Questions on reproducibility

- Should computer simulations be made reproducible? Why?
  Yes. If I cannot reproduce your simulation, then I don’t know what you have simulated.
- To the last bit, or on a "good enough" basis?
  Bit for bit, because it is cheaper and more useful.

Questions on reproducibility

- Should computer simulations be made reproducible? Why?

  Yes. If I cannot reproduce your simulation, then I don’t know what you have simulated.
- To the last bit, or on a "good enough" basis?

  Bit for bit, because it is cheaper and more useful.
- At what cost?

  Zero, once we have suitable infrastructure and adapted our code to it.

Questions on reproducibility

- Should computer simulations be made reproducible? Why?
  Yes. If I cannot reproduce your simulation, then I don't know what you have simulated.
- To the last bit, or on a "good enough" basis?
  Bit for bit, because it is cheaper and more useful.
- At what cost?
  Zero, once we have suitable infrastructure and adapted our code to it.
- Can we ensure reproducibility without repeating lengthy computations?
  Yes, it can be guaranteed by the infrastructure.

Questions on replicability

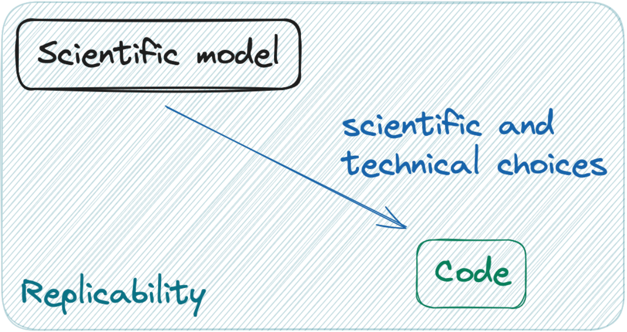
- Is replicability more or less important than reproducibility in scientific practice?
  Are apples better than oranges?

Questions on replicability

- Is replicability more or less important than reproducibility in scientific practice?
  Are apples better than oranges?

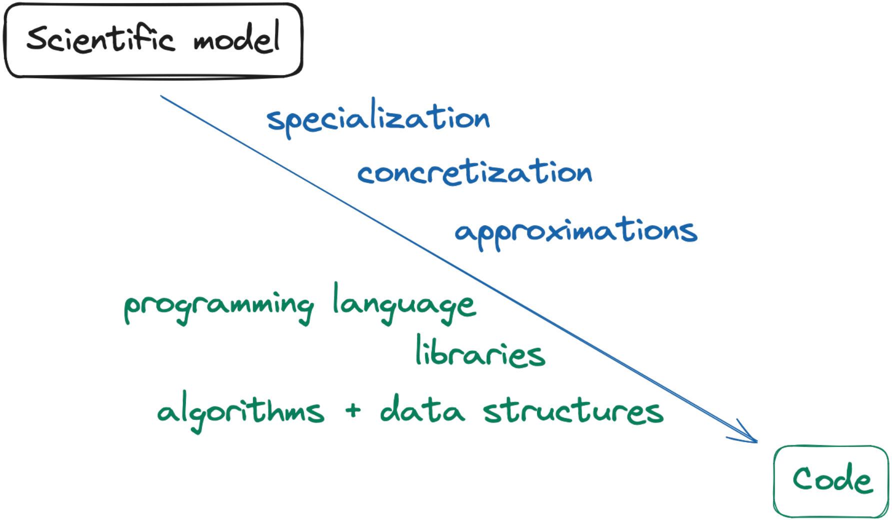
- How replicable are computer simulations today?
- What are the obstacles to better replicability?
  Stay tuned!

Choices



chosen variability

Choices


Scientific model
specialization
concretization
approximations
programming language
libraries
algorithms + data structures
Code

Chemical physics: how many phases for supercooled water?

PHYSICS TODAY

DOI:10.1063/PT.6.1.20180822a

22 Aug 2018 in Research & Technology

The war over supercooled water

How a hidden ~~coding error~~ methodological choice fueled a seven-year dispute between two of condensed matter's top theorists.

Ashley G. Smart

A.G. Smart, Physics Today, 2018

Chemical physics: how many phases for supercooled water?

As it turned out, the trouble stemmed from the algorithmic trick the Berkeley team had used to speed up its code. Both teams had performed their free-energy calculations using Monte Carlo simulations, which can be used to find the low-energy states of a molecular ensemble by randomly sampling—and systematically accepting or rejecting—various potential ensemble configurations. To do Monte Carlo, you need an efficient way to generate those sample configurations. The Berkeley team chose to generate them by running short molecular dynamics simulations in which molecules were initialized with random positions and velocities.

The procedure the Berkeley team used to initialize the molecular dynamics simulations was unorthodox — it involved randomly selecting a pair of molecules and then swapping the velocities of their constituent atoms. Palmer and company discovered that the technique produced sample configurations that seemed to flout basic laws of statistical mechanics: The energies deviated from the expected equilibrium values, governed by the Boltzmann distribution, and the molecules' rotational and translational temperatures didn't match up. Perhaps most important, the molecules behaved as if they were tens of degrees hotter than their assigned temperature.

The meaning of “replicable”

Robust under irrelevant changes

The meaning of “replicable”

Robust under irrelevant changes

Replication

- tests the relevance of specific choices
- exposes tacit assumptions
- explores the space of variability

The meaning of “replicable”

Robust under irrelevant changes

Replication

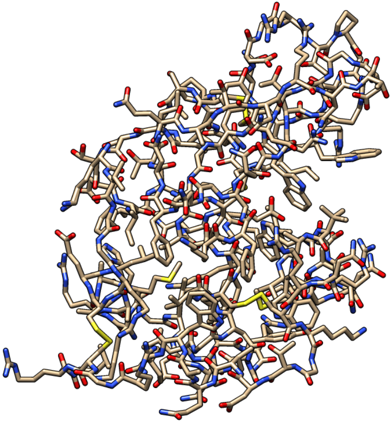
- tests the relevance of specific choices
- exposes tacit assumptions
- explores the space of variability

Requires precise documentation of all choices.

Molecular simulations of protein

A small protein: lysozyme

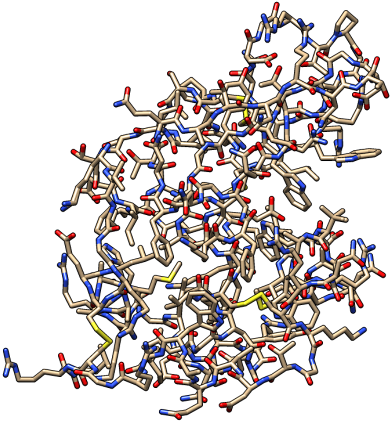
1960 atoms, 1001 shown



Molecular simulations of proteins

A small protein: lysozyme

1960 atoms, 1001 shown



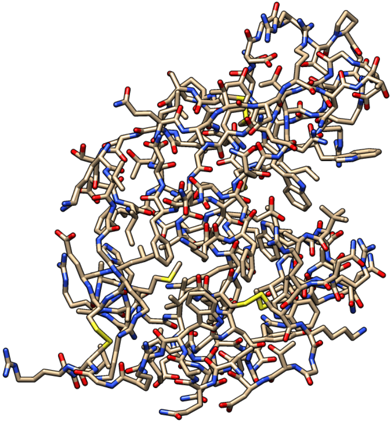
Molecular Mechanics Model

- Atoms are point masses
- Newtonian mechanics: $\mathbf{F} = m \cdot \mathbf{a}$
- Positions → forces → velocity update → position update

Molecular simulations of proteins

A small protein: lysozyme

1960 atoms, 1001 shown



Molecular Mechanics Model

- Atoms are point masses
- Newtonian mechanics: $\mathbf{F} = m \cdot \mathbf{a}$
- Positions → forces → velocity update → position update

Major scientific choice:

- *Force field* $U(\Gamma, \mathbf{r}_1, \ldots, \mathbf{r}_N)$
- Graph Γ: molecular structure
- Force on atom i: $\mathbf{F}_i = -\frac{\partial}{\partial \mathbf{r}_i} U$

Biomolecular force fields

$$
\begin{aligned}
U \quad = \quad & \sum_{\text{bonds } ij} k_{ij} \left(r_{ij} - r_{ij}^{(0)} \right)^2 \\
& + \sum_{\text{angles } ijk} k_{ijk} \left(\phi_{ijk} - \phi_{ijk}^{(0)} \right)^2 \\
& + \sum_{\text{dihedrals } ijkl} k_{ijkl} \cos \left(n_{ijkl} \theta_{ijkl} - \delta_{ijkl} \right) \\
& + \sum_{\text{all pairs } ij} 4\epsilon_{ij} \left(\frac{\sigma_{ij}^{12}}{r^{12}} - \frac{\sigma_{ij}^{6}}{r^{6}} \right) \\
& + \sum_{\text{all pairs } ij} \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}}
\end{aligned}
$$

Biomolecular force fields

$$
\begin{aligned}
U \;=\; & \sum_{\text{bonds } ij} k_{ij}\left(r_{ij} - r_{ij}^{(0)}\right)^2 \\
& + \sum_{\text{angles } ijk} k_{ijk}\left(\phi_{ijk} - \phi_{ijk}^{(0)}\right)^2 \\
& + \sum_{\text{dihedrals } ijkl} k_{ijkl}\cos\left(n_{ijkl}\theta_{ijkl} - \delta_{ijkl}\right) \\
& + \sum_{\text{all pairs } ij} 4\epsilon_{ij}\left(\frac{\sigma_{ij}^{12}}{r^{12}} - \frac{\sigma_{ij}^{6}}{r^{6}}\right) \\
& + \sum_{\text{all pairs } ij} \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}}
\end{aligned}
$$

missing details

not quite true

common approximations are glossed over

Biomolecular force fields

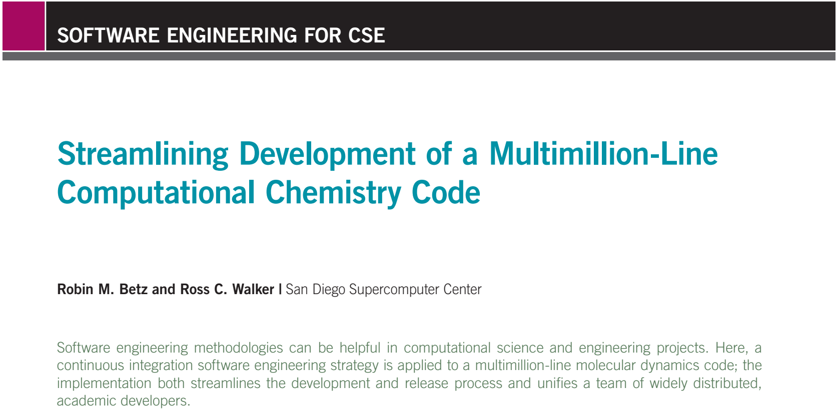
Protein parameter files for the 2012 edition of the AMBER force field, in somewhat documented formats:

lines	filename
984	`amino12.in`
814	`aminoct12.in`
782	`aminont12.in`
533	`frcmod.ff12SB`
744	`parm99.dat`

For the details... read the source code!

For the algorithms that select the parameters for a given Γ... read the source code!

Read the source code!



SOFTWARE ENGINEERING FOR CSE

Streamlining Development of a Multimillion-Line Computational Chemistry Code

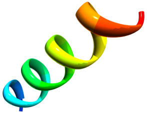
Robin M. Betz and Ross C. Walker | San Diego Supercomputer Center

Software engineering methodologies can be helpful in computational science and engineering projects. Here, a continuous integration software engineering strategy is applied to a multimillion-line molecular dynamics code; the implementation both streamlines the development and release process and unifies a team of widely distributed, academic developers.

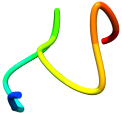
Betz & Walker, Comp. Sci. Eng. **16**(3), 10-17 (2014)

A personal story

Question: What is the structure of a certain family of peptides in gas phase?



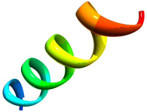
or



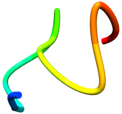
?

A personal story

Question: What is the structure of a certain family of peptides in gas phase?



or



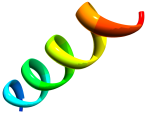
?

M.F. Jarrold, Phys. Chem. Chem. Phys. **9**, 1659 (2007):

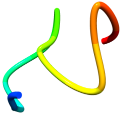
- $Ac\,A_{15}\,K + H^{+} \longrightarrow$ helicoidal
- $Ac\,K\,A_{15} + H^{+} \longrightarrow$ globular

A personal story

Question: What is the structure of a certain family of peptides in gas phase?



or



?

M.F. Jarrold, Phys. Chem. Chem. Phys. **9**, 1659 (2007):

- $Ac\,A_{15}\,K + H^{+} \longrightarrow$ helicoidal
- $Ac\,K\,A_{15} + H^{+} \longrightarrow$ globular

My simulations: globular structure for all sequences

Read the paper!

M.F. Jarrold, Phys. Chem. Chem. Phys. **9**, 1659 (2007):

> *Molecular Dynamics (MD) simulations were performed to help interpret the experimental results. The simulations were done with the MACSIMUS suite of programs [31] using the CHARMM21.3 parameter set. A dielectric constant of 1.0 was employed.*

Read the paper!

M.F. Jarrold, Phys. Chem. Chem. Phys. **9**, 1659 (2007):

> *Molecular Dynamics (MD) simulations were performed to help interpret the experimental results. The simulations were done with the MACSIMUS suite of programs [31] using the CHARMM21.3 parameter set. A dielectric constant of 1.0 was employed.*
>
> *A variety of starting structures were employed (such as helix, sheet, and extended linear chain) and a number of simulated annealing schedules were used in an effort to escape high energy local minima. Often, hundreds of simulations were performed to explore the energy landscape of a particular peptide. In some cases, MD with simulated annealing was unable to locate the lowest energy conformation and more sophisticated methods were used (see description of evolutionary based methods below).*

Read the paper!

M.F. Jarrold, Phys. Chem. Chem. Phys. **9**, 1659 (2007):

> *Molecular Dynamics (MD) simulations were performed to help interpret the experimental results. The simulations were done with the MACSIMUS suite of programs [31] using the CHARMM21.3 parameter set. A dielectric constant of 1.0 was employed.*

This is a typical level of description!

> *A variety of starting structures were employed (such as helix, sheet, and extended linear chain) and a number of simulated annealing schedules were used in an effort to escape high energy local minima. Often, hundreds of simulations were performed to explore the energy landscape of a particular peptide. In some cases, MD with simulated annealing was unable to locate the lowest energy conformation and more sophisticated methods were used (see description of evolutionary based methods below).*

Replicability is a documentation problem

Huge variability space

- Too many details to document in a paper

Replicability is a documentation problem

Huge variability space

- Too many details to document in a paper
- Code is too low-level and technical

Replicability is a human-computer interface problem

Huge variability space

Replicability is a human-computer interface problem

Huge variability space

Place it at the interface!

Replicability is a human-computer interface problem

Huge variability space

Place it at the interface!

- Make all dimensions of variability human-readable and machine-readable

Replicability is a human-computer interface problem

Huge variability space

Place it at the interface!

- Make all dimensions of variability human-readable and machine-readable
- Similar in many ways to knowledge graphs
- But: knowledge graphs are about established knowledge, not variability

Replicability is a human-computer interface problem

Huge variability space

Place it at the interface!

- Make all dimensions of variability human-readable and machine-readable
- Similar in many ways to knowledge graphs
- But: knowledge graphs are about established knowledge, not variability
- Human and machine interpretation must be compatible
- A task for formal methods?

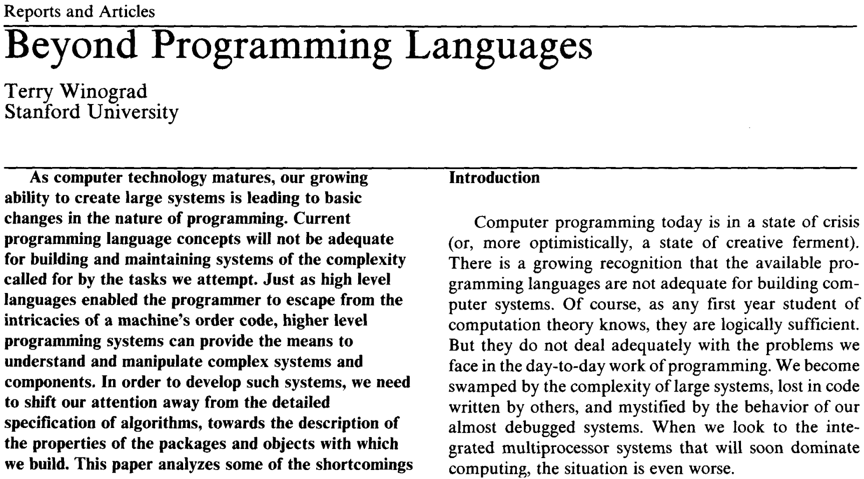
Programs as machines

Photo by Mehmet Turgut Kirkgoz

Computational media

Photo by Karolina Grabowska

An old idea



Reports and Articles

Beyond Programming Languages

Terry Winograd
Stanford University

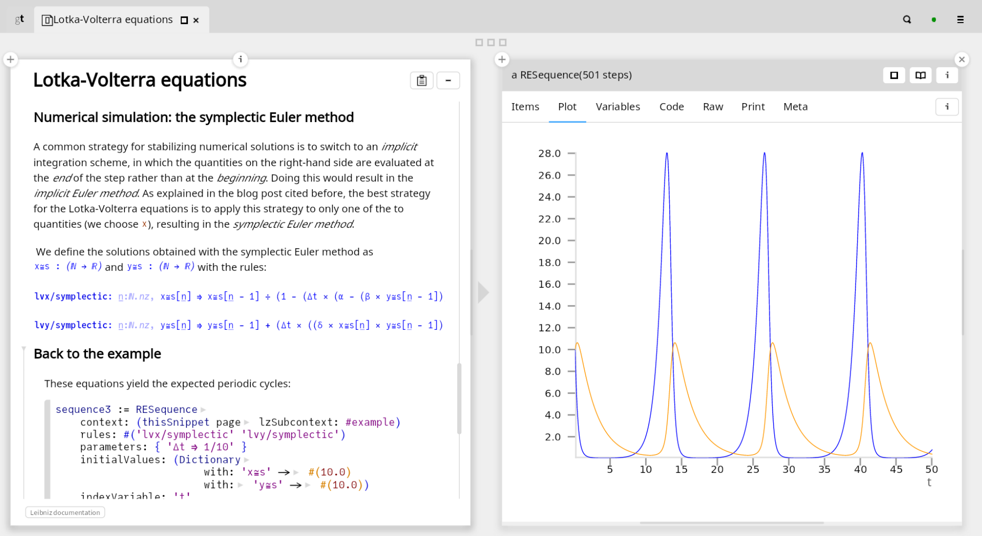
As computer technology matures, our growing ability to create large systems is leading to basic changes in the nature of programming. Current programming language concepts will not be adequate for building and maintaining systems of the complexity called for by the tasks we attempt. Just as high level languages enabled the programmer to escape from the intricacies of a machine's order code, higher level programming systems can provide the means to understand and manipulate complex systems and components. In order to develop such systems, we need to shift our attention away from the detailed specification of algorithms, towards the description of the properties of the packages and objects with which we build. This paper analyzes some of the shortcomings

Introduction

Computer programming today is in a state of crisis (or, more optimistically, a state of creative ferment). There is a growing recognition that the available programming languages are not adequate for building computer systems. Of course, as any first year student of computation theory knows, they are logically sufficient. But they do not deal adequately with the problems we face in the day-to-day work of programming. We become swamped by the complexity of large systems, lost in code written by others, and mystified by the behavior of our almost debugged systems. When we look to the integrated multiprocessor systems that will soon dominate computing, the situation is even worse.

T. Winograd, Communications of the ACM **22** (7), 391-401 (1979)

Models first, tools attached



Lotka-Volterra equations

Numerical simulation: the symplectic Euler method

A common strategy for stabilizing numerical solutions is to switch to an *implicit* integration scheme, in which the quantities on the right-hand side are evaluated at the *end* of the step rather than at the *beginning*. Doing this would result in the *implicit Euler method*. As explained in the blog post cited before, the best strategy for the Lotka-Volterra equations is to apply this strategy to only one of the to quantities (we choose `x`), resulting in the *symplectic Euler method*.

We define the solutions obtained with the symplectic Euler method as `x≅s : (ℕ → ℝ)` and `y≅s : (ℕ → ℝ)` with the rules:

```
lvx/symplectic: n:ℕ.nz, x≅s[n] ⇒ x≅s[n - 1] ÷ (1 - (Δt × (α - (β × y≅s[n - 1])
```

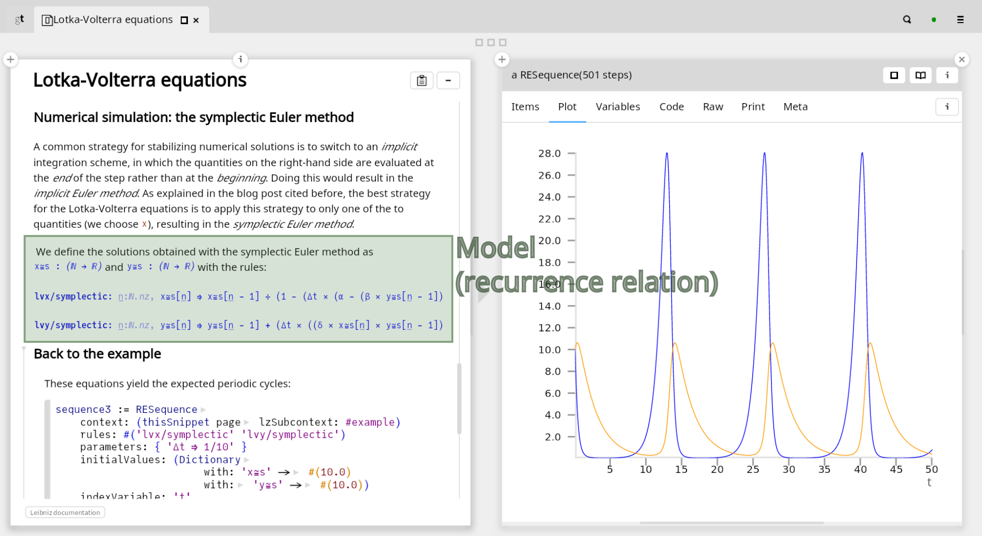
```
lvy/symplectic: n:ℕ.nz, y≅s[n] ⇒ y≅s[n - 1] + (Δt × ((δ × x≅s[n] × y≅s[n - 1])
```

Back to the example

These equations yield the expected periodic cycles:

```
sequence3 := RESequence
    context: (thisSnippet page  lzSubcontext: #example)
    rules: #('lvx/symplectic' 'lvy/symplectic')
    parameters: { 'Δt ⇒ 1/10' }
    initialValues: (Dictionary
                         with: 'x≅s' →  #(10.0)
                         with:  'y≅s' →  #(10.0))
    indexVariable: 't'
```

Models first, tools attached



Lotka-Volterra equations

Numerical simulation: the symplectic Euler method

A common strategy for stabilizing numerical solutions is to switch to an *implicit* integration scheme, in which the quantities on the right-hand side are evaluated at the *end* of the step rather than at the *beginning*. Doing this would result in the *implicit Euler method*. As explained in the blog post cited before, the best strategy for the Lotka-Volterra equations is to apply this strategy to only one of the to quantities (we choose `x`), resulting in the *symplectic Euler method*.

We define the solutions obtained with the symplectic Euler method as `x≅s : (ℕ → ℝ)` and `y≅s : (ℕ → ℝ)` with the rules:

```
lvx/symplectic: n:ℕ.nz, x≅s[n] ⇒ x≅s[n - 1] ÷ (1 - (Δt × (α - (β × y≅s[n - 1])

lvy/symplectic: n:ℕ.nz, y≅s[n] ⇒ y≅s[n - 1] + (Δt × ((δ × x≅s[n] × y≅s[n - 1])
```

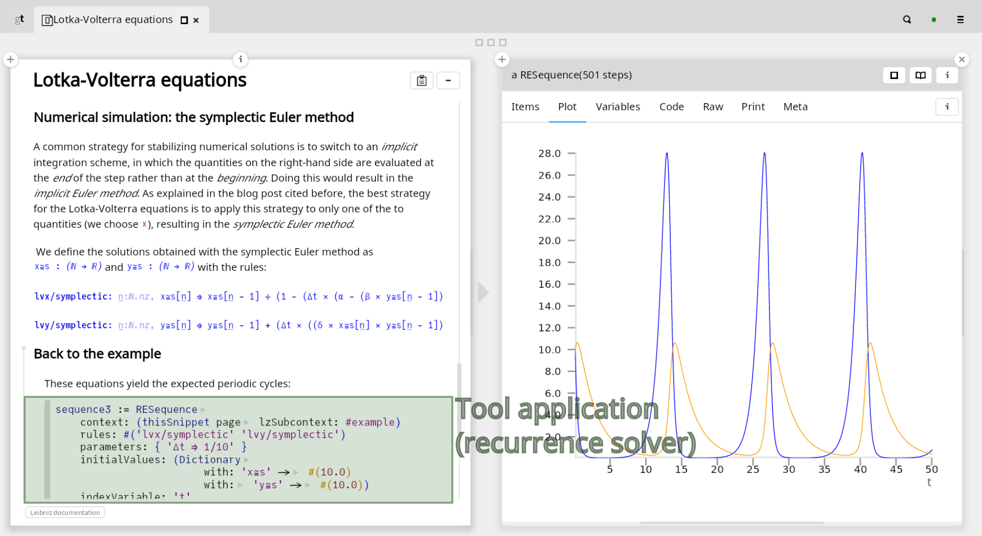
Model (recurrence relation)

Back to the example

These equations yield the expected periodic cycles:

```
sequence3 := RESequence
    context: (thisSnippet page  lzSubcontext: #example)
    rules: #('lvx/symplectic' 'lvy/symplectic')
    parameters: { 'Δt ⇒ 1/10' }
    initialValues: (Dictionary
                        with: 'x≅s' →  #(10.0)
                        with:  'y≅s' →  #(10.0))
    indexVariable: 't'
```

Models first, tools attached



Lotka-Volterra equations

Numerical simulation: the symplectic Euler method

A common strategy for stabilizing numerical solutions is to switch to an *implicit* integration scheme, in which the quantities on the right-hand side are evaluated at the *end* of the step rather than at the *beginning*. Doing this would result in the *implicit Euler method*. As explained in the blog post cited before, the best strategy for the Lotka-Volterra equations is to apply this strategy to only one of the to quantities (we choose `x`), resulting in the *symplectic Euler method*.

We define the solutions obtained with the symplectic Euler method as `x≅s : (ℕ → ℝ)` and `y≅s : (ℕ → ℝ)` with the rules:

```
lvx/symplectic: n:ℕ.nz, x≅s[n] ⇒ x≅s[n - 1] ÷ (1 - (Δt × (α - (β × y≅s[n - 1]))
```

```
lvy/symplectic: n:ℕ.nz, y≅s[n] ⇒ y≅s[n - 1] + (Δt × ((δ × x≅s[n] × y≅s[n - 1])
```

Back to the example

These equations yield the expected periodic cycles:

```
sequence3 := RESequence
    context: (thisSnippet page  lzSubcontext: #example)
    rules: #('lvx/symplectic' 'lvy/symplectic')
    parameters: { 'Δt ⇒ 1/10' }
    initialValues: (Dictionary
                        with: 'x≅s' →  #(10.0)
                        with:  'y≅s' →  #(10.0))
    indexVariable: 't'
```

Tool application (recurrence solver)

Digital Scientific Notations

- Formal languages → machine-readable, machine-searchable

Digital Scientific Notations

- Formal languages → machine-readable, machine-searchable
- **Not** a programming language
- Encode models, specializations, concretizations, approximations ...

Digital Scientific Notations

- Formal languages → machine-readable, machine-searchable
- **Not** a programming language
- Encode models, specializations, concretizations, approximations ...
- Data rather than code, but can express algorithms

Digital Scientific Notations

- Formal languages → machine-readable, machine-searchable
- **Not** a programming language
- Encode models, specializations, concretizations, approximations ...
- Data rather than code, but can express algorithms
- Technically: specification languages

K. Hinsen, PeerJ Computer Science **4** e158 (2018)

Application scenarios

Software documentation

- Embedded into prose written for humans
- Equivalence to papers/textbooks verified by human reviewers
- Equivalence to the code proven by formal methods

Application scenarios

Software documentation

- Embedded into prose written for humans
- Equivalence to papers/textbooks verified by human reviewers
- Equivalence to the code proven by formal methods

User interface

- Users write specializations etc. in a DSN
- Code generators translate to optimized code

Application scenarios

Software documentation

- Embedded into prose written for humans
- Equivalence to papers/textbooks verified by human reviewers
- Equivalence to the code proven by formal methods

User interface

- Users write specializations etc. in a DSN
- Code generators translate to optimized code

Computational medium

- Scientists do all their work in a Wiki-like environment
- Computational tools become plug-ins
- UI elements include DSN, visualizations, etc.

Take-home messages

Computation is about formal systems

The most important boundary in scientific computing is between a **formal system** (→ *reproducibility*) and its **interpretation** (→ *replicability*).

Take-home messages

Computation is about formal systems

The most important boundary in scientific computing is between a **formal system** (→ *reproducibility*) and its **interpretation** (→ *replicability*).

Reproducibility is a socio-economic problem

Challenge: transition towards a reproducibility-supporting infrastructure

Requires institutional backing!

Take-home messages

Computation is about formal systems

The most important boundary in scientific computing is between a **formal system** (→ *reproducibility*) and its **interpretation** (→ *replicability*).

Reproducibility is a socio-economic problem

Challenge: transition towards a reproducibility-supporting infrastructure

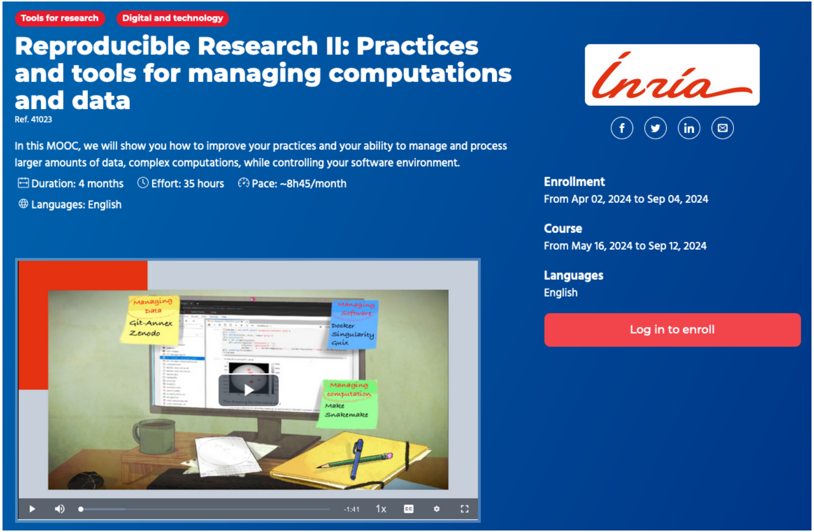
Requires institutional backing!

Replicability is a human-computer interface problem

Challenges:

- expose models, specializations, concretizations, approximations
- provide tools to examine, search, and transform them
- make technical choices inspectable and modifiable

Follow the MOOC!



Tools for research

Digital and technology

Reproducible Research II: Practices and tools for managing computations and data

Ref. 41023

In this MOOC, we will show you how to improve your practices and your ability to manage and process larger amounts of data, complex computations, while controlling your software environment.

Duration: 4 months Effort: 35 hours Pace: ~8h45/month

Languages: English

Enrollment
From Apr 02, 2024 to Sep 04, 2024

Course
From May 16, 2024 to Sep 12, 2024

Languages
English

Log in to enroll

The End

Slides, transcript, and more:

Questions?