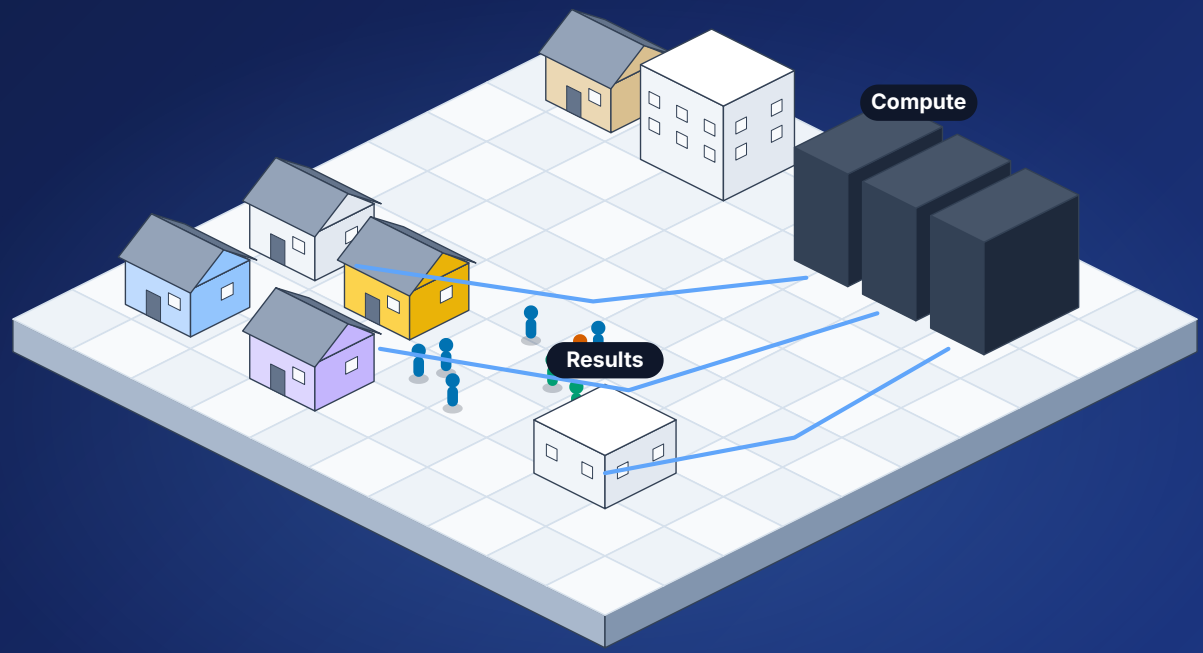
Applied Computational Mathematics Lab

EPIDEMIC MODELING · MODULE 12

Computational considerations

Software, numerical stability, speed and scale



4 EPISODES

1. Languages, libraries and architecture
2. Numerical stability
3. Making it fast
4. Calibration at scale

Based on Chapter 12 of T. Lazebnik, Epidemic Modeling: Principles, Algorithms, and Applications (Academic Press, 2026)

acml.teddylazebnik.com

BEFORE WE START

Warm-up: true or false?

CHECK YOURSELF

(a) Written correctly, the same model gives the same answer in any programming language.

(b) Halving the step of any ODE solver halves its error.

(c) If a model's infection curve looks right, the model has no bugs.

(d) Running a simulation on 64 cores makes it 64 times faster.

HOW TO USE THIS

Guess now; the module settles each one. Answers at the end.

WHY THIS MODULE

A model lives in code. This module is about getting the code right, keeping the numbers stable, making it fast, and fitting it when every run costs time.

BEFORE WE START

This module's map

PART 1

Software

Languages and libraries, how to arrange a model's parts, and how to test the model as well as the code.

PART 2

Numerical stability

Euler and Runge–Kutta, stiff models, daily-step models, and exact against fast stochastic simulation.

PART 3

Speed

Profiling, better algorithms, batching, memory, structure in the mixing, and parallel computing.

PART 4

Calibration at scale

What a run costs, exact gradients, emulators and history matching.

GOALS

By the end you should be able to choose tools and structure for a modeling project, test a model as well as its code, pick a stable and accurate solver and step, find and remove the real bottleneck in slow code, and plan a calibration the cost of a run can afford.

ONE RULE

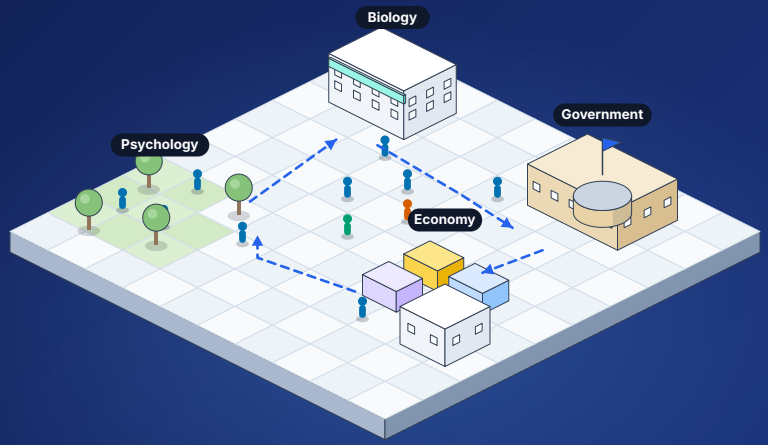
All numbers come from our own code and runs. Timings were measured on our shared two-core cloud machine (medians of repeats): your computer will give different times, but similar ratios.

Applied
Computational
Mathematics
Lab

PART 1

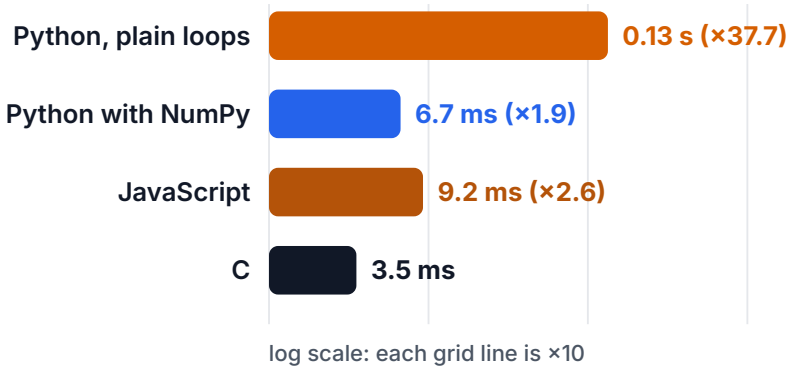
Software for models

Languages, libraries, architecture and tests.



PART 1 · LANGUAGES

One model, four languages



Time for one epidemic: agent-level SIR, 10,000 people, 200 daily steps, one random draw per person per day (our code: m12_bench).

LANGUAGE	ONE EPIDEMIC	EPIDEMICS PER HOUR	SHARE INFECTED
Python, plain loops	0.13 s	27,006	92.1%
Python with NumPy	6.7 ms	538,922	92.1%
JavaScript (Node)	9.2 ms	389,189	92.1%
C (gcc -O2)	3.5 ms	1,019,830	92.1%

Share infected: mean over runs; each language uses its own random numbers.

- Interpreted loops are slow; NumPy moves the loops into compiled code; JavaScript is compiled while it runs; C is compiled ahead of time.
- Speed is one side of the trade; the other is how fast you can write, read and change the code.
- The first rule of any faster version: it must give the same answer.

Stand on tested libraries

SOLVERS

ODE and SDE

Adaptive steps, stiff methods and error estimates, tested on thousands of problems.

RANDOM NUMBERS

Generators

Long periods, good statistics, independent streams and seeds you can record.

STATISTICS

Fitting

Likelihoods, optimizers, samplers and their diagnostics.

IN PLAIN WORDS

Writing your own solver or sampler is a great way to learn how it works, and a risky way to publish results.

REPRODUCIBLE

Anyone can rerun the analysis and get the same numbers: same code version, same library versions, same data, same random seeds, recorded with the results.

- Pin library versions in an environment file.
- Record the seed of every stochastic run; give each parallel worker its own random stream.
- Keep data versions next to the code that used them.

PART 1 · ARCHITECTURE

Keep the parts of a model apart



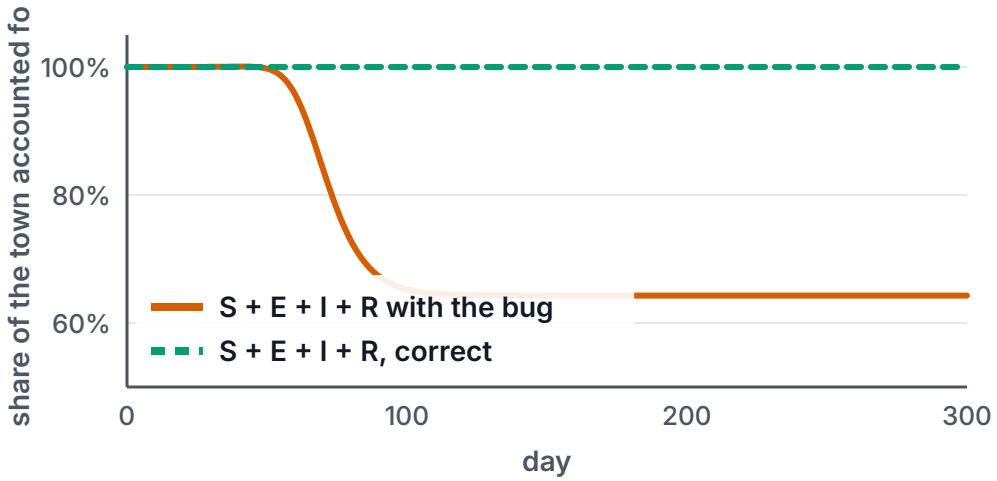
A modular layout: each part has one job and can be tested, swapped and reused on its own.

- **Mechanism**: the equations or rules of transmission, nothing else.
- **Parameters** and **settings** in files, not inside the code, so every run records what it used.
- **Observation**: a separate layer turns infections into reported cases (Module 9), so the same mechanism can meet different data.
- **Inference**: fitting code that only sees the model through an interface; changing the optimizer does not touch the model.
- **Outputs**: tables and figures made from saved results, never by rerunning by hand.

HABITS

Keep the code in version control, write down why each change was made, and tag the version used for every published figure.

Test the model, not just the code



A bug we planted in an SEIR model: recoveries taken from E instead of I. The infection curve is unchanged (peak 14.4% on the same day); the population leaks away.

CHECK	WHAT IT CATCHES	HERE
conservation: S + E + I + R = N	flows that lose or create people	fails at step 1
final size against the formula (Module 3)	wrong rates or missing terms	0.892645 vs 0.892645
limiting cases: no transmission, no recovery	sign errors, wrong branches	nothing spreads; nobody recovers
halve the step	solver error	see Part 2
fixed-seed regression run	unintended changes	same numbers as last time

IN PLAIN WORDS

Run these checks automatically after every change. They catch exactly the errors that look plausible.

Check yourself: Part 1

CHECK YOURSELF

1. A run takes 0.13 s in plain Python and 3.5 ms in C. How many runs per hour does each allow?

2. Name three things you must record so that someone else can reproduce a stochastic simulation exactly.

3. An SIR model with births and deaths keeps the population constant. Write one automatic test that would catch a missing death term.

TIP

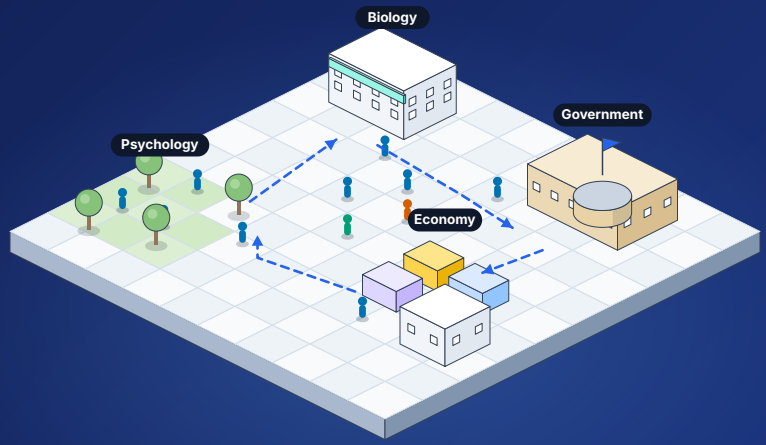
Answers are at the end of the deck. Try them before you look.

Applied Computational Mathematics Lab

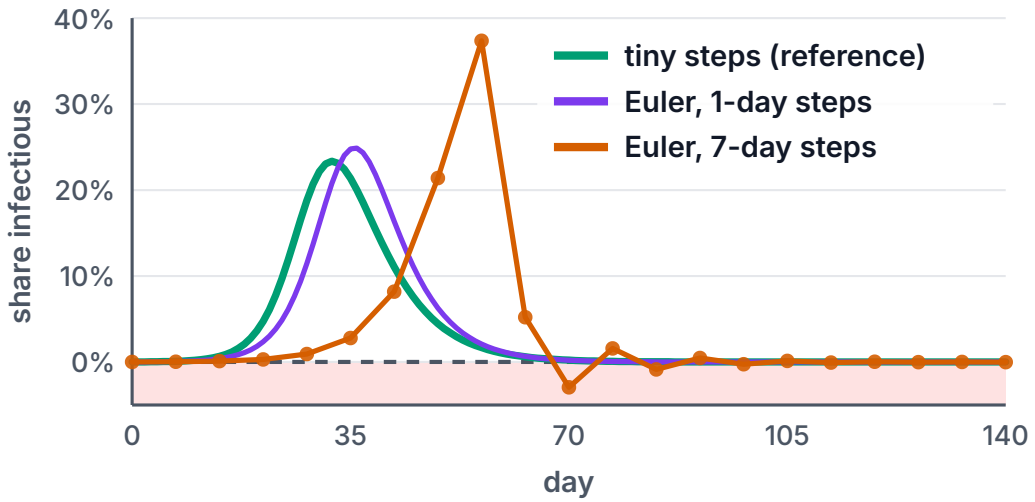
PART 2

Numerical stability

When the solver, not the model, decides the answer.



Follow one slope, or average four



Our SIR town: R0 2.5, infectious for 5 days, 1 in 10,000 infectious at the start; red band: below zero.

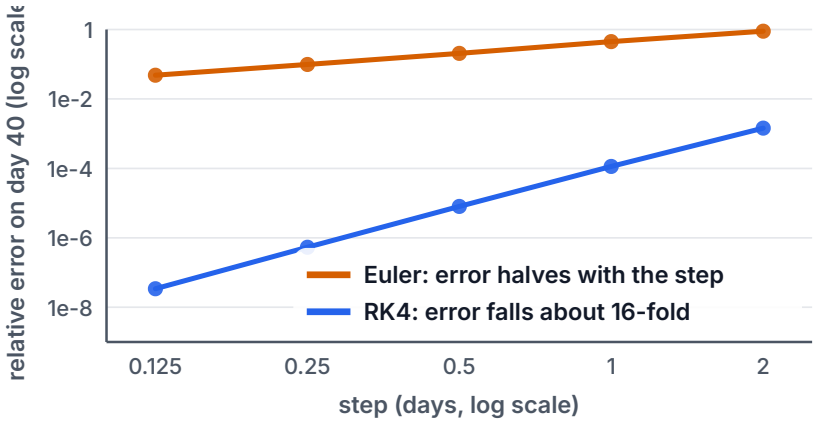
$$\text{Euler: } y_{n+1} = y_n + h\, f(y_n)$$

$$\text{RK4: } y_{n+1} = y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

STEP	EULER PEAK ERROR	RK4 PEAK ERROR
0.5 days	+3.2%	-0.00%
1 day	+6.5%	-0.00%
2 days	+12.9%	-0.01%
4 days	+22.9%	-0.06%
7 days	+60.0%	-4.74%

Reference (tiny steps): peak 23.4% on day 32. With 7-day steps Euler's lowest value was -2.9%.

How fast the error shrinks with the step



Relative error in the share infectious on day 40, against a solution with 0.001-day steps (our runs, computed for this slide).

$$\text{error} \approx C\,h^p: \qquad p = 1\ (\text{Euler}), \qquad p = 4\ (\text{RK4})$$

STEP	EULER	RK4
0.125	4.82e-02	3.43e-08
0.25	9.87e-02	5.33e-07
0.5	2.07e-01	8.04e-06
1	4.46e-01	1.14e-04
2	8.97e-01	1.45e-03

- Halving the step halves Euler's error but cuts RK4's about 16-fold.
- RK4 costs four slope evaluations per step, and still wins by far at any accuracy worth having.
- Measure order at a fixed time: a peak read off a coarse grid also carries the error of where the grid points fall.
- Adaptive solvers estimate the error as they go and choose the step for you; set their tolerances on purpose.

PART 2 · STABILITY

Why big steps go negative

$$I_{n+1} = I_n\left(1 + h(\beta S_n - \gamma)\right) \;\Rightarrow\; I_{n+1} < 0 \text{ when } h(\gamma - \beta S_n) > 1$$

- Late in the epidemic S is small, so Euler multiplies I by about $1 - h\gamma$ each step: negative once the step is longer than the infectious period (here 5 days).
- For a process that decays at rate λ, explicit Euler is stable only if $h < 2/\lambda$. The fastest process sets the step, even when it barely matters.
- Clipping negatives to zero hides the symptom, not the error: the step is still too long.

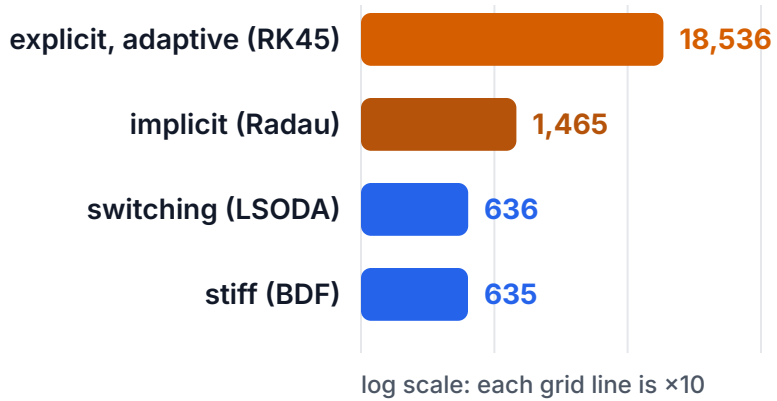
STIFF

A model is stiff when it mixes very fast and very slow processes. Explicit solvers must take tiny steps to stay stable; implicit solvers can take steps sized for the slow processes.

OUR RUNS

Adding a one-hour triage step (rate 24 a day) to the SEIR town: Euler is stable only for steps under 2 hours (4,380 steps a year). With 0.05-day steps it was fine; with 0.1-day steps it blew up.

Use a solver built for the problem



Model evaluations to solve one year of the SEIR town with triage (relative tolerance 10^{-6}, SciPy solve_ivp).

SOLVER	EVALUATIONS	STEPS	HOSPITAL PEAK
RK45	18,536	2,649	0.003939
Radau	1,465	206	0.003938
LSODA	636	354	0.003939
BDF	635	265	0.003939

- All four give the same answer; the stiff solvers need about 29 times fewer evaluations.
- Switching solvers (LSODA) detect stiffness and change method on the fly.
- Signs of stiffness: an explicit adaptive solver taking many tiny steps where nothing seems to happen.

PART 2 · DISCRETE TIME

A daily-step model is a different model

$$p = 1 - e^{-r\,\Delta t}, \qquad \text{mean days infectious} = \frac{1}{1 - e^{-\gamma}} = 5.52$$

MODEL OF THE SAME TOWN	PEAK SHARE INFECTIOUS	SHARE EVER INFECTED
the equations (tiny steps)	23.4%	89.3%
daily steps with rates (Euler, 1 day)	24.9%	90.6%
daily steps with probabilities	26.9%	92.1%

Daily probabilities raise R0 in effect from 2.5 to 2.76 (our runs).

- Turning a rate into a daily probability keeps it between 0 and 1, but counts every partial day as a whole one: the infectious time grows from 5 days to 5.5.
- The order of events inside a step (infect first or recover first) changes the result too.
- Nothing crashes; the numbers are just different. Report which model you ran, and fit its parameters in its own terms.

IN PLAIN WORDS

The agent model of Part 1 used daily probabilities: that is why it infected 92%, not the 89% of the equations.

Exact, or fast: Gillespie and tau-leaping

GILLESPIE (EXACT)

1. Rates: infection $a_1 = \beta SI/N$, recovery $a_2 = \gamma I$.
2. Wait an exponential time with rate $a_1 + a_2$.
3. Pick infection with probability $a_1/(a_1 + a_2)$, else recovery; update; repeat.

TAU-LEAPING (APPROXIMATE)

1. Fix a step τ.
2. Draw the numbers of infections and recoveries in the step from Poisson distributions with means $a_1\tau$ and $a_2\tau$.
3. Update all at once; guard against negative counts.

TOWN OF 10,000, R0 2.5	SHARE INFECTED	RUNS THAT WENT NEGATIVE
exact	89.2%	0 of 300
tau-leaping, 0.1-day steps	89.4%	4 of 300
tau-leaping, 1-day steps	90.6%	27 of 300
tau-leaping, 4-day steps	95.7%	220 of 300

Exact method: 17,836 events per epidemic on average; 90% of runs infected 88.2%–90.2%. The equations: 89.3%.

- Small steps reproduce the exact method closely at a fraction of the work.
- Large steps inflate the epidemic and try to remove people who are not there.

PART 2 · VALIDATION

Guards worth building into every model

BOUNDS

No negatives

Every compartment stays between 0 and the population; a violation is an error to report, not to clip silently.

CONSERVATION

Everyone counted

Totals match births, deaths and the starting population at every step.

CONVERGENCE

Halve the step

The answer barely moves when the step is halved, or the tolerance tightened.

KNOWN ANSWERS

Formulas and limits

Final size, threshold behavior, and cases with no transmission or no recovery.

TWO METHODS

Cross-check

An exact and a fast method, or two solvers, agree on the outputs that matter.

IN PLAIN WORDS

Cheap checks, run automatically, protect results that took hours to compute.

FROM PART 1

Our planted bug passed a visual check of the infection curve and failed the conservation check at the first step.

Check yourself: Part 2

CHECK YOURSELF

4. Euler's error in the peak is 6.5% with 1-day steps. Roughly what error do you expect with 0.25-day steps? And for RK4, if its error is 0.01% with 2-day steps, at 1-day steps?

5. A model has a process with rate 50 a day. What is the largest stable explicit Euler step?

6. A recovery rate of 0.5 a day is used as a daily probability of 0.5 in one model and $1 - e^{-0.5}$ in another. What are the mean infectious times in days?

TIP

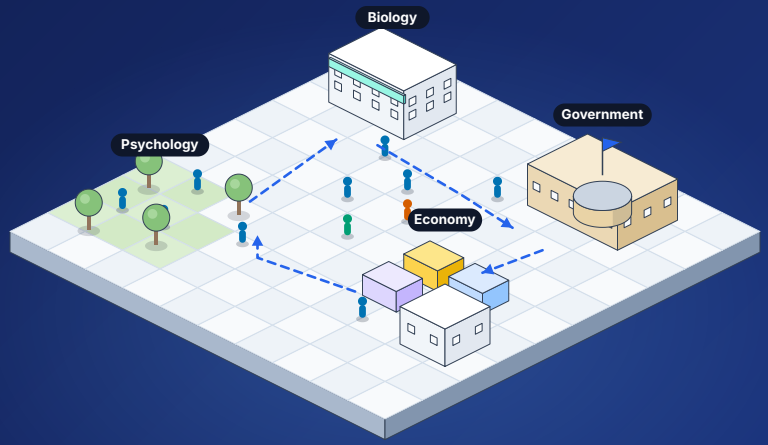
Answers are at the end of the deck.

Applied Computational Mathematics Lab

PART 3

Making it fast

Measure, then fix the algorithm, then the habits.



PART 3 · PROFILE FIRST

Find where the time actually goes

FUNCTION	SHARE OF RUN TIME	CALLS
the pair check (distance and chance)	38%	9,113,201
the loop over every pair that calls it	62%	–

Python's profiler on our spatial agent town (2,000 people, contact distance about 12 neighbors, one epidemic).

- The obvious version checks every person against every infectious person, every day: work grows with the square of the town.
- The profile points at one place. Speeding up anything else is wasted effort.
- Profile again after each change: the bottleneck moves.

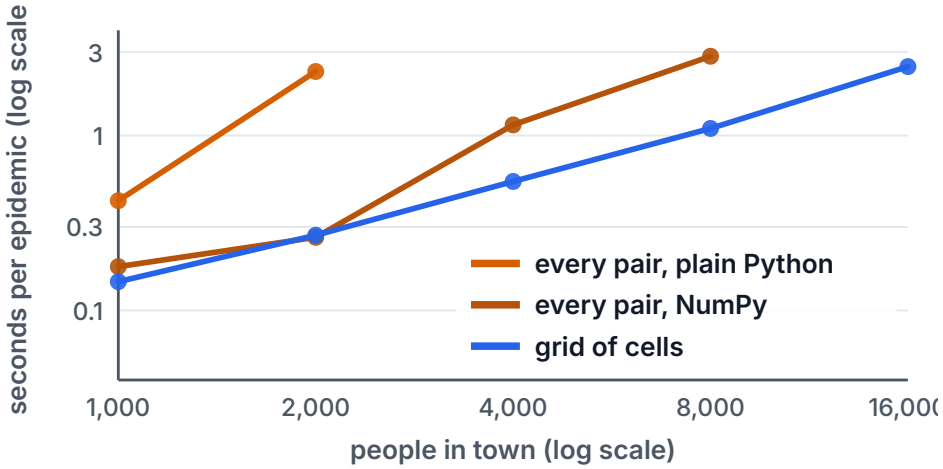
PROFILER

A tool that runs your program and reports how much time each function, or each line, takes and how often it is called.

A HABIT

Measure with realistic sizes. A loop that is harmless for 1,000 people can dominate for 100,000.

A better algorithm beats a faster language



Seconds per epidemic by town size (our runs; plain loops only up to 2,000, NumPy up to 8,000).

- A grid of cells as wide as the contact distance: each infectious person checks only the 9 cells around them.
- With every random draw fixed in advance, all three versions gave identical epidemics (900, 900, 900 infected of 1,000).
- At 2,000 people the grid was 8.6 times faster than plain loops; its work grows in proportion to the town, not with its square.

PEOPLE	EVERY PAIR (LOOPS)	EVERY PAIR (NUMPY)	GRID
1,000	0.42 s	0.18 s	0.15 s
2,000	2.33 s	0.26 s	0.27 s
4,000	–	1.15 s	0.55 s
8,000	–	2.84 s	1.10 s
16,000	–	–	2.48 s

PART 3 · HABITS

Work in bulk, and allocate once

TASK	SLOW WAY	FAST WAY	SPEED-UP
1,000 SIR parameter sets	one at a time: 2.58 s	one batch of arrays: 30.2 ms	×85
20,000 values, one at a time	grow an array: 58.6 ms	fill a ready array: 0.82 ms	×71

Our runs; the batched results were identical to the one-at-a-time results.

- Vectorize: replace loops over people or parameter sets with array operations.
- Batch: run many scenarios or parameter sets together, one time loop for all.
- Preallocate: growing an array copies it every time; make it once at full size.
- Cache results that do not change, such as contact lists or reference solutions.

EXAMPLE

A thousand parameter sets of the SIR model: the batch keeps one array per compartment, each with 1,000 entries, and steps them all forward together.

IN PLAIN WORDS

These habits rarely change the answer, so test that they do not.

PART 3 · STRUCTURE

Use the structure of the problem

PROBLEM	NAIVE	WITH STRUCTURE
contact network: 100,000 people, about 10 contacts each	full table: 80 GB	sparse list: 12.4 MB, one step 1.3 ms
network of 4,000: one step	dense: 7.8 ms	sparse: 0.047 ms (×166)
force of infection, 3,000 groups	full matrix: 3.94 ms	two vectors: 0.0024 ms (×1,655)

Our runs; the structured versions gave the same results (to 1 part in 10^{15} for the mixing).

$$C = a\,w^{\top} \;\Rightarrow\; \lambda = CI = a\,(w^{\top} I): \quad O(n^2) \to O(n)$$

- Store only what exists: most pairs of people never meet.
- Proportionate mixing makes the contact matrix rank one; a few groups of settings make it low rank.
- Reduce the state: age bands instead of single years, or identical agents grouped into counts, when the question allows.

SPARSE MATRIX

Most entries are zero, so only the nonzero ones are stored, with their positions.

More cores help only the parallel part

$$\text{speed-up}(c) = \frac{1}{(1-p) + p/c} \leq \frac{1}{1-p}$$

Amdahl's law: p is the share of the work that runs in parallel, c the number of cores.

SHARE IN PARALLEL	2 CORES	8 CORES	64 CORES	LIMIT
90%	×1.8	×4.7	×8.8	×10
95%	×1.9	×5.9	×15.4	×20
99%	×2.0	×7.5	×39.3	×100

- Ensembles of stochastic runs, scenarios and parameter sets are naturally parallel: no run waits for another.
- Give each worker its own random stream, or results become correlated or unrepeatable.
- Reading data, collecting results and the serial parts of a fit set the ceiling.

FROM THE TABLE

If a twentieth of the work is serial, 64 cores give about ×15, and no number of cores gives more than ×20.

Check yourself: Part 3

CHECK YOURSELF

7. A simulation spends 80% of its time in one function. If you make that function infinitely fast, what is the best possible speed-up?

8. How much memory does a full table of contacts need for 10 million people (8 bytes per entry)? And a sparse list with 15 contacts each (12 bytes per contact)?

9. Why should every parallel worker in a stochastic ensemble get its own random stream?

TIP

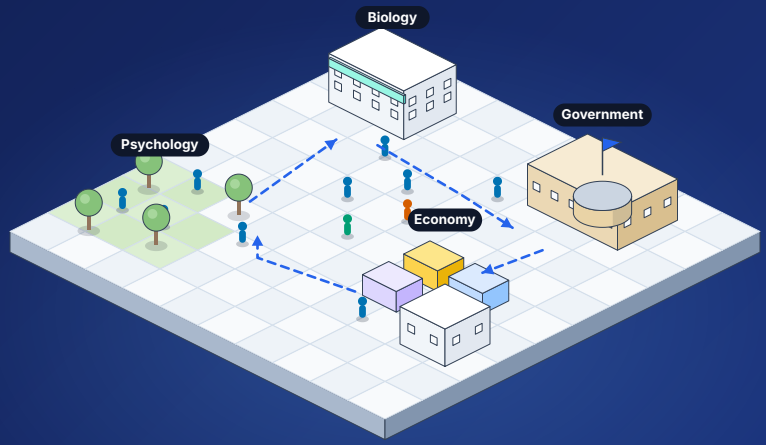
Answers are at the end of the deck.

Applied Computational Mathematics Lab

PART 4

Calibration at scale

When every run costs real time.



PART 4 · THE COST OF A RUN

What a run costs decides what you can afford

TASK	RUNS	SIR EQUATIONS (9 MS A RUN)	AGENT TOWN OF 16,000 (1.9 S A RUN)
one fit (about 300 runs)	300	3 s	10 min
MCMC, 20,000 steps	20,000	3 min	10.7 h
ABC, 1,000,000 runs	1,000,000	2.4 h	22 days

Our runs, on our shared machine; costs scale with the number of runs.

- A Bayesian sampler that is cheap for equations can take half a day for agents; approximate Bayesian computation, weeks.
- So: make each run cheaper (Part 3), need fewer runs (gradients), or replace most runs with a fast stand-in (emulators).

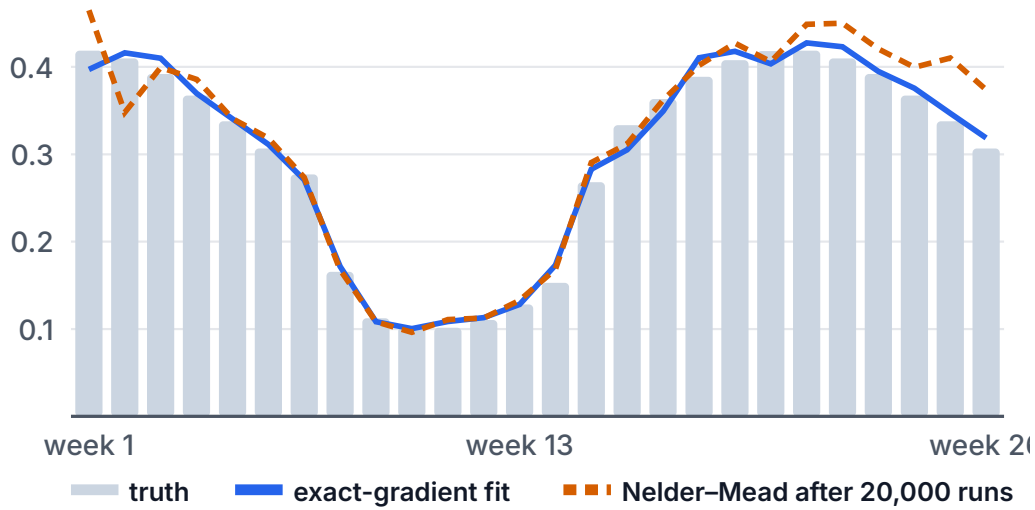
BAYESIAN SAMPLER

An algorithm that draws parameter values in proportion to how well they explain the data (Markov chain Monte Carlo); it needs thousands of model runs.

BACK OF THE envelope

A likelihood that takes one second and a sampler that needs 20,000 steps: about five and a half hours, before any repeats or checks.

Exact gradients need far fewer runs



Transmission rate in each of 26 weeks, fitted to half a year of noisy cases (negative binomial, with a smoothness penalty); our runs.

$$\frac{d}{dt}\frac{\partial x}{\partial \theta} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial \theta} + \frac{\partial f}{\partial \theta}$$

Sensitivity equations, solved alongside the model, give the exact derivative of every output.

METHOD	MODEL RUNS	TIME	LOSS ABOVE THE BEST
Nelder–Mead, no gradient	20,000 (unfinished)	53 s	21.2
quasi-Newton, finite differences	2,052	6.1 s	0.0
quasi-Newton, exact gradient	75 (with sensitivities)	2.5 s	0.0

One run with sensitivities cost about 19 times a plain run in our code.

Where exact derivatives come from

FINITE-DIFFERENCE STEP	DISAGREEMENT WITH THE EXACT GRADIENT
10^{-2}	8.1e-04
10^{-4}	8.1e-08
10^{-6}	1.3e-09
10^{-8}	2.0e-07
10^{-10}	2.1e-05

One component of the gradient, at the starting point of the fit (our runs).

- Too large a step: truncation error. Too small: rounding error. The best step here gave about one part in a billion.
- Finite differences need two runs per parameter for every gradient; with 26 parameters, 52 runs.

AUTOMATIC DIFFERENTIATION

Software that applies the chain rule to every operation in the model's code, so derivatives come out exactly, to rounding error. Forward mode carries derivatives along with the values; reverse mode (adjoint) works backwards and costs about one extra run, however many parameters there are.

TOOLS

Libraries such as JAX, PyTorch and Julia's differential-equation tools do this for models written in them (Griewank and Walther, 2008).

A fast stand-in for a slow simulator

$$f(x) \sim \mathcal{GP}\big(m(x),\, k(x, x')\big), \qquad k(x, x') = \sigma^2 \exp\Big(- \sum_j \frac{(x_j - x'_j)^2}{2\ell_j^2} \Big)$$

- We ran the agent town (8,000 people) 50 times at well-spread settings of contact chance and recovery chance (1.0 s a run), and fitted a Gaussian process to the share infected and the peak day.
- On 60 new runs it predicted the share infected with a typical error of 7 points; its 90% intervals held 88% of the truths (peak day: 17 days, 93%).
- The emulator also reports its own uncertainty, so we know where to trust it and where to run the simulator again.

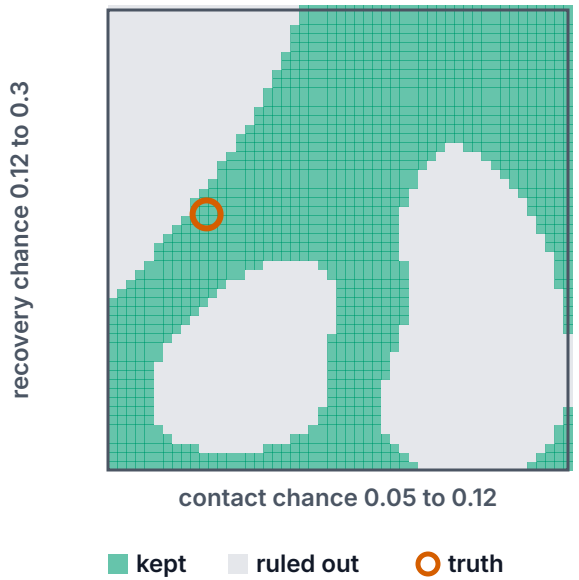
EMULATOR

A statistical model trained on a modest number of simulator runs that predicts the simulator's output, with error bars, in a fraction of the time (Kennedy and O'Hagan, 2001).

HABITS

Design the training runs to cover the space evenly (a Latin hypercube), and always check the emulator on runs it has not seen.

Rule out what cannot match, safely



Settings kept and ruled out on a 101 × 101 grid; ring: the setting that made the observed epidemic.

$$I(x) = \max_k \frac{|z_k - \mathbb{E}[f_k(x)]|}{\sqrt{\mathrm{Var}_{\mathrm{emulator}} + \mathrm{Var}_{\mathrm{observation}}}} > 3 \;\Rightarrow\; \text{ruled out}$$

RESULT	VALUE
settings ruled out	50%
time, emulator, all 10,201 settings	7 ms
time, simulator, same settings (estimated)	about 3 hours
kept settings that reproduced the epidemic	4 of 20
ruled-out settings that did	0 of 20

Observed: 78% infected, peak on day 75; reproduced means within 5 points and 15 days (our runs).

IN PLAIN WORDS

History matching does not finish the fit: it shrinks the search, so the expensive runs go where they can matter (Andrianakis et al., 2015).

Check yourself: Part 4

CHECK YOURSELF

10. A simulation takes 3 seconds. How long do 50,000 runs take? How many could you afford in one day?

11. A model has 40 parameters. How many runs does one central finite-difference gradient need? About how many does one adjoint gradient need?

12. An emulator says a setting cannot match the data, but its own uncertainty there is huge. Should you rule the setting out?

TIP

Answers are on the next slides.

ANSWERS

Answers: warm-up and Parts 1–2

- **(a)** True: all four languages gave 92.1%–92.1%, but C was 38 times faster than plain Python loops. **(b)** False: Euler's error halves; RK4's falls about 16-fold. **(c)** False: our planted bug left the curve perfect and lost a third of the town. **(d)** False: with 5% serial work, 64 cores give about ×15.
- **1.** 3,600 / 0.13 = about 27,700 runs an hour; 3,600 / 0.0035 = about 1,030,000.
- **2.** The code version, the library versions, the input data and every random seed (plus parameter settings).
- **3.** Check at every step that S + I + R stays equal to the starting population; a missing death term makes it grow.
- **4.** Euler: about 6.5% / 4 = 1.6%. RK4: about 0.01% / 16, under 0.001%.
- **5.** $h < 2/50 = 0.04$ days, about an hour.
- **6.** $1/0.5 = 2$ days and $1/(1 - e^{-0.5}) = 2.54$ days.

IN SHORT

A daily-step model is a model in its own right: fit and report its parameters in its own terms, or use small steps if you mean the equations.

ANSWERS

Answers: Parts 3–4

- **7.** $1/(1-0.8) = 5$: the other 20% remains.
- **8.** Full: $(10^7)^2 \times 8 = 8 \times 10^{14}$ bytes, 800 terabytes. Sparse: $10^7 \times 15 \times 12$ bytes, about 1.8 gigabytes.
- **9.** Shared or repeated streams make runs correlated (or identical), so the ensemble understates the spread and cannot be reproduced reliably.
- **10.** 150,000 s, about 42 hours; 86,400 / 3 = 28,800 runs a day.
- **11.** $2 \times 40 = 80$ runs; an adjoint gradient costs about one forward and one backward run, whatever the number of parameters.
- **12.** No: the implausibility divides by the emulator's uncertainty, so a very uncertain prediction cannot rule a setting out. Run the simulator there, or improve the emulator.

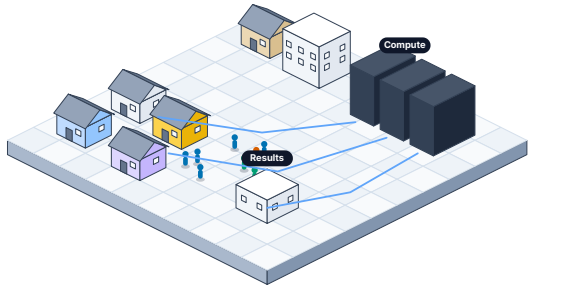
IN PLAIN WORDS

Correct first, then stable, then fast, and only then at scale.

CREDITS

Sources and credits

- This deck accompanies Chapter 12 of T. Lazebnik, Epidemic Modeling: Principles, Algorithms, and Applications (Academic Press, 2026), the course textbook. It follows the chapter's topics in our own words; no text, figures, tables, exercises or code are copied from the book.
- The text, figures, examples and simulations were written, drawn and run for ACML; nothing is copied from the book. Numbers labelled "our run" come from our own code; counts from agencies and papers are credited where they appear.
- All numbers come from our own code and runs (production/explains/sims/epi_m12_sims.py and m12_bench/), on made-up towns. Timings were measured on our shared two-core cloud machine; absolute times will differ on other computers.
- Named works, described in our own words: Gillespie, 1977 (exact stochastic simulation); Gillespie, 2001 (tau-leaping); Petzold, 1983 (LSODA); Hairer and Wanner, 1996 (stiff problems); Amdahl, 1967; Wilson et al., 2014 (practices for scientific computing); Griewank and Walther, 2008 (automatic differentiation); Kennedy and O'Hagan, 2001 (emulators and calibration); Andrianakis et al., 2015 (history matching of epidemic models).
- Companion videos: **Epidemic Modeling**, module 12, episodes 49–52. More from the lab: acml.teddylazebnik.com



Applied Computational Mathematics Lab

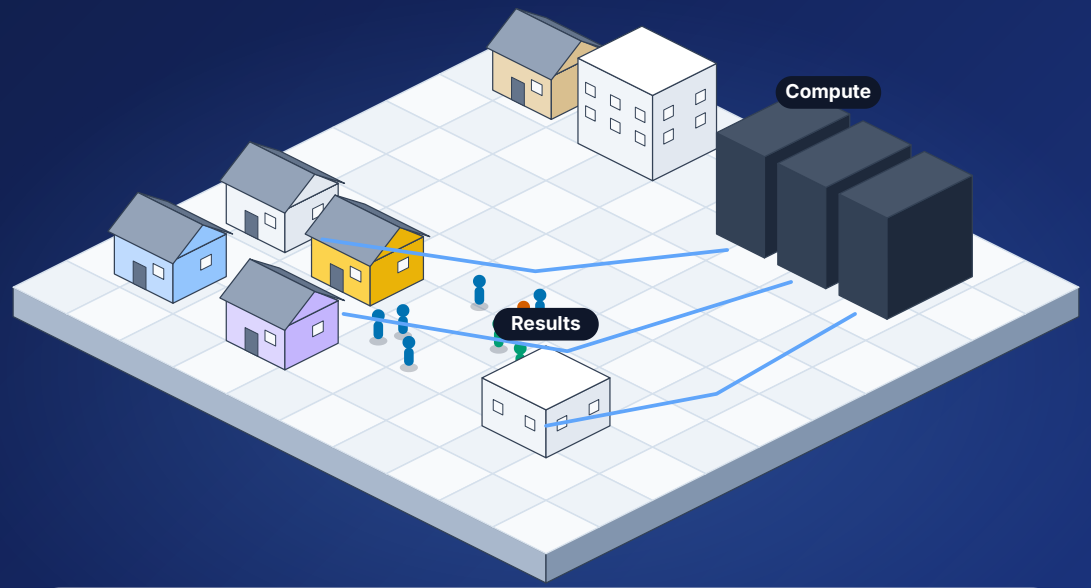
END OF MODULE 12

Computational considerations

YOU COVERED

- Languages, libraries and architecture
- Numerical stability
- Making it fast
- Calibration at scale

Module 12 complete



UP NEXT · MODULE 13

Closing remarks and future directions

Where epidemic modeling goes next

Based on Chapter 12 of T. Lazebnik, Epidemic Modeling: Principles, Algorithms, and Applications (Academic Press, 2026)