Probabilistic Numerical Approximation

Nicholas Krämer

Technical University
of Denmark

Probabilistic Numerical Approximation

Nicholas Krämer

Technical University of Denmark

Probabilistic **Numerical** Approximation

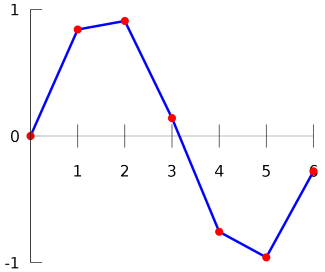
Nicholas Krämer

Technical University of Denmark

Surprised?

Example: Interpolation

- Unknown function $f : \Omega \to \mathbb{R}$
- Given data $(x_n, f(x_n))_{n=1}^{N}$, what is $f(\tilde{x})$?



Relevant for:

- Regression (e.g. statistical emulators)
- Some classification

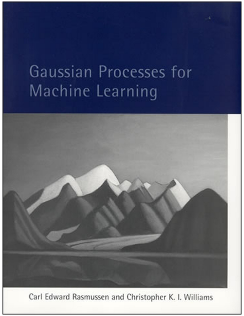
Solving the interpolation problem

Traditional approaches:

- ◇ Polynomials
- ◇ Polynomial splines
- ◇ Neural networks (maybe less traditional)

So why are we so keen on Gaussian processes?

- ◇ Very flexible. Work on all sorts of problems
- ◇ Easy to do fun statistics with ("uncertainty quantification")



We are already breathing probabilistic numerics. Let's dig deeper.

The numerics of Gaussian processes

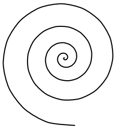
The posterior mean of a Gaussian process:

$$m(x) = k(x, \mathbf{X}) \underbrace{k(\mathbf{X}, \mathbf{X})^{-1}}_{\text{Large \& ill-cond.}} y$$

Feasible Gaussian processes depend on good numerics.

Solutions:

- As usual: ~~Cholesky decomposition~~ (no chance)
- Iterative solvers
- Low-rank approximations

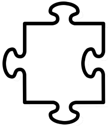


Real-life dynamical systems

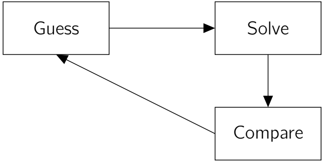
- We know: $y(t)$ solves $\dot{y}(t) = f(y(t), \theta)$, $y(0) = y_0$.
- We know: $y(1) + \epsilon = 4$, $\epsilon \sim N(0, 0.1^2)$
- What is θ?

Who cares about this kind of problem? E.g.

- Physics-informed ("scientific") machine learning
- Diffusion models, neural ODEs
- AI and the physical world
- Me. And therefore (today), you.



Common solution:

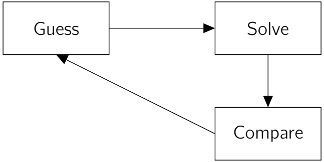


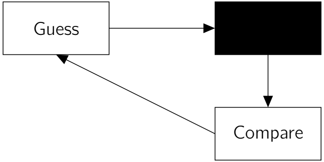
1. Guess θ (e.g. $\theta = 10$)
2. Compute $y(1)$ given $\dot{y}(t) = f(y(t), 10)$, $y(0) = y_0$
3. Compare $y(1)$ to $y(1) + \epsilon = 4$, $\epsilon \sim N(0, 0.1^2)$
4. Use the comparison to improve the guess

Nonlinear ODEs don't have closed-form solutions

- Use a numerical ODE solver.
- E.g. a Runge-Kutta method
- Well-understood. Performant software.

DifferentialEquations.jl





There must be a better way!

What is the problem?

History

- RK methods from $\sim$ 100 years ago.
- Not designed for use in a (statistical) context

Language barriers

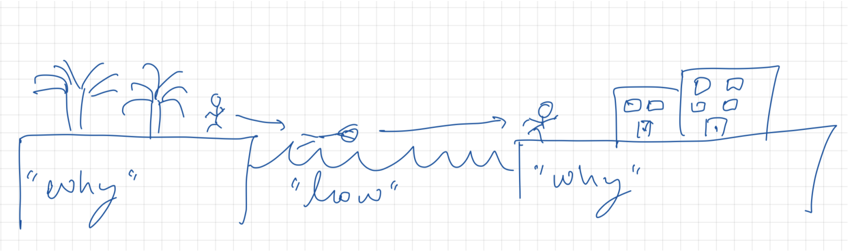
- Know stats/ML or know solvers
- When simulating, we must take solvers for granted?

What does this imply?

- Numerical methods were not designed to deal with *webs of algorithms*
- Numerical methods were not designed to deal with *multiple sources of information*
- Numerical methods were not designed to deal with *model discrepancies*

Probabilistic numerical approximation.

Outline for today



"How".

Introduction to numerical algorithms

	Course outline	*The right way* (my way)
1.	How to store a number	Probability distributions
2.	Matrices	Bayes' rule, manipulating Gaussians
3.	Interpolation & Least squares	Gaussian processes and the like
4.	Integration, differentiation	**Next**
5.	Krylov methods, optimisation, differential equations	**Afterwards**

Recap: Gaussian process interpolation

- Prior: $p(u) = \mathrm{GP}(0, k(\cdot, \cdot))$
- Information: $p(y \mid u(\mathbf{X})) = N(u(\mathbf{X}), \sigma^2 I)$
- Posterior: $p(u(\cdot) \mid y) = \mathrm{GP}(W(\cdot)y, E(\cdot, \cdot))$ with

$$W(z) = k(z, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma^2 I)^{-1},$$
$$E(z, z') = k(z, z') - k(z, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma^2 I)^{-1} k(\mathbf{X}, z')$$

Why do we like Gaussian processes?

- ◇ Closed-form marginals
- ◇ Closed-form conditionals
- ◇ Well-behaved under linear operations
- ◇ Learn from different communities

	Monday	Tuesday	Wednesday	Thursday	Friday
Dates	17 July 2023	18 July 2023	19 July 2023	20 July 2023	21 July 2023
Themes	Introduction to Probablistic Modeling	Probabilistic Models/Sequential Decision Making	Probabilistic Numerics	Implicit Models/Diffusion Models	Further Probabilistic Modeling
09:00	Carl Rasmussen- Gaussian processes	Mark van der Wilk	Nico Kramer- Probabilistic Numerical Approximation	Francisco Vargas	Rich Turner- Neural Processes for Environmental Research
09:30					
10:00					Break
10:30	Break	Break	Break	Break	Yingzhen Li- Approximate Inference: An Intro
11:00	Mike Tipping- Probability, Bayesian Inference & Parsimonious Models	Ian Osban	Jonathan Wenger- Computation-aware Gaussian Processes	José Miguel Hernández Lobato- Normalizing Flows for Molecular Modeling	
11:30					
12:00				Lunch	Lunch
12:30	Lunch	Lunch	Poster session & Lunch		
13:00				opportunities in accelerating materials design with geometric deep learning and	Neill Campbell
13:30	Tony O'Hagan- Gaussian Processes I have Known	Katja Hofmann-Towards human-like AI in video games			
14:00				Break	Break
14:30			Henry Moss	Marc Deisenroth	Neil Lawrence
15:00	Break	Break			
15:30	David Ginsbourger- On Gaussian Process Multiple-Fold Cross-Validation	Arno Solin- Sequential Inference and Learning			Poster Session & Farewell Reception
16:00					
16:30					
17:15		Science Tour- 90 min*			
19:00-22:00	Evening Dinner at Sidney Sussex College*				
19:30				Cambridge Shakespeare Festival*	

Probabilistic numerical integration

aka {Kernel, Bayesian(-Hermite), probabilistic} {quadrature, cubature, integration}

Problem

Compute

$$\mu = \int_\Omega f(x) p(x) \mathrm{d}x$$

from evaluations of f

Solution

Gaussian process! Generative model

$$\mu = \int_\Omega f(x) p(x) \mathrm{d}x \qquad p(f) = \mathrm{GP}(0, k(\cdot, \cdot)) \qquad y = f(\mathbf{X})$$

Probabilistic numerical integration (continued)

Solution

Gaussian process! Generative model

$$\mu = \int_\Omega f(x)p(x)\mathrm{d}x \qquad p(f) = \mathrm{GP}(0, k(\cdot,\cdot)) \qquad y = f(\mathbf{X})$$

Then, $p(f \mid y)$ is a Gaussian process. $p(\mu \mid y) = N(Wy, E)$ is a Gaussian random variable,

$$W = \left[\int_\Omega k(x, \mathbf{X})p(x)\mathrm{d}x\right] k(\mathbf{X}, \mathbf{X})^{-1}$$

$$E = \left[\int_\Omega \left[\int_\Omega k(x, y)p(x)\mathrm{d}x\right] p(y)\mathrm{d}y\right] - Wk(\mathbf{X}, \mathbf{X})W^\top$$

Why is probabilistic numerical integration so fantastic?

- ◇ Posterior mean replicates non-probabilistic numerical integration routines:

Rule	**Prior**	**Point set**
Trapezoidal rule	Wiener process	equispaced nodes
Gaussian quadrature	polynomial features	suitable point set

- ◇ Yet: choose $k(\cdot,\cdot)$ and $\mathbf{X}$ *as the problem dictates*, not as the solver requires
- ◇ Convergence guarantees
- ◇ Easy to modify: adaptive, multi-level, control-variates, etc.

Probabilistic numerical integration is a template for probabilistic numerical algorithms

(Bayesian) probabilistic numerical algorithms

Definition

A Bayesian probabilistic numerical algorithm requires

- ◇ A prior distribution — Gaussian process
- ◇ An information "operator" — e.g. point evaluations
- ◇ Conditioning — As usual
- ◇ A quantity of interest — e.g. an integral Integral

Cockayne, Oates, Sullivan, Girolami. Bayesian probabilistic numerical methods. SIAM Review. 2019.

Why do we need a definition?

Modifying probabilistic numerical integration

Generative model

$$s(\mathbf{Y}) := \frac{d^2}{dx^2} f(\mathbf{Y}) \quad \cancel{\mu = \int_\Omega f(x) p(x) \mathrm{d}x} \qquad p(f) = \mathrm{GP}(0, k(\cdot, \cdot)) \qquad y = f(\mathbf{X})$$

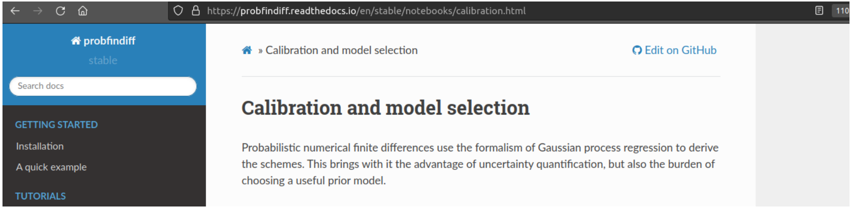
Then, $p(s(\mathbf{Y}) \mid y) = N(W(\mathbf{Y})y, E(\mathbf{Y}, \mathbf{Y}))$ is Gaussian,

$$W(\mathbf{Y}) = \frac{d^2}{dx^2} k(\mathbf{Y}, \mathbf{X}) k(\mathbf{X}, \mathbf{X})^{-1},$$
$$E(\mathbf{Y}, \mathbf{Y}) = \frac{d^4}{dx^2 dx'^2} k(\mathbf{Y}, \mathbf{X}) - W(\mathbf{Y}) k(\mathbf{X}, \mathbf{X}) W(\mathbf{Y})^\top$$

This is probabilistic numerical *differentiation*.

Why is probabilistic numerical **differentiation** so fantastic?

- Generalises numerical differentiation formulas (for certain choices of $k(\cdot,\cdot)$ and $\mathbf{X}$)
- Strong connections to radial basis function collocation & finite differences
- Choose $k(\cdot,\cdot)$ and $\mathbf{X}$ as the problem dictates, not as the solver requires
- Do statistics (model validation, etc) on a numerical algorithm



`pip install probfindiff`

Some more modification

Generative model

$$s(\mathbf{Y}) = \frac{d^2}{dx^2} f(\mathbf{Y}) \qquad p(f) = \mathrm{GP}(0, k(\cdot, \cdot)) \qquad y = f(\mathbf{X})$$

Then, $p(y \mid s(\mathbf{Y})) = N(W(\mathbf{X})y, E(\mathbf{X}, \mathbf{X}))$ is Gaussian,

$$W(\mathbf{X}) = k(\mathbf{X}, \mathbf{Y}) \left[\frac{d^4}{dx^2 dx'^2} k(\mathbf{Y}, \mathbf{Y})\right]^{-1},$$

$$E(\mathbf{X}, \mathbf{X}) = k(\mathbf{Y}, \mathbf{X}) - W(\mathbf{Y}) \left[\frac{d^4}{dx^2 dx'^2} k(\mathbf{X}, \mathbf{X})\right] W(\mathbf{Y})^\top$$

and we have a probabilistic numerical *solver for a partial differential equation*

Take-away message

- It seems that we can solve any problem.
- To do so:
  - Know your prior
  - Know your information
  - Know your quantity of interest
- Learn from traditional algorithms about stability, convergence, and so on
- But don't be afraid of modifications:

  Do what the problem dictates, not what the solver requires

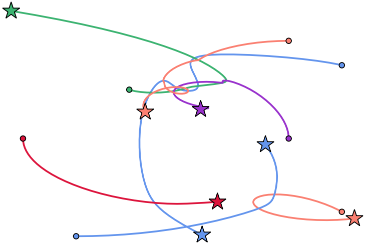
A more sophisticated example

Problem

Simulate $\dot{y}(t) = f(y(t))$ from $y(0) = y_0$ to $y(1)$.

More sophisticated because:

- Nonlinear derivative constraints
- Explicit temporal structure
- (I like to think about this kind of problem)



Solving ODEs

Problem

Simulate $\dot{y}(t) = f(y(t))$ from $y(0) = y_0$ to $y(1)$.

But we come well-equipped:

- ◇ Prior: $p(y) = \mathrm{GP}(0, k(\cdot, \cdot))$
- ◇ Information:

$$\begin{cases} \dot{y}(\mathbf{T}) = f(y(\mathbf{T})), \\ y(0) = y(t_0) = 0 \end{cases}$$

- ◇ Quantity of interest: $y(1)$

Goal

Estimate

$$p(y(1) \mid \dot{y}(\mathbf{T}) = f(y(\mathbf{T})), y(0) = y_0)$$

as fast as possible.

Prior

Choose $p(y) = \mathrm{GP}(0, k(\cdot, \cdot))$ such that it has a state-space representation: let $\mathbf{y} = (y, \dot{y}, \ldots)$

$$\mathrm{d}\mathbf{y}(t) = F\mathbf{y}(t)\mathrm{d}t + L\mathrm{d}w(t), \quad p(\mathbf{y}(0)) = N(m_0, C_0)$$

Once-integrated Wiener process

$$F = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad L = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Twice-integrated Wiener process

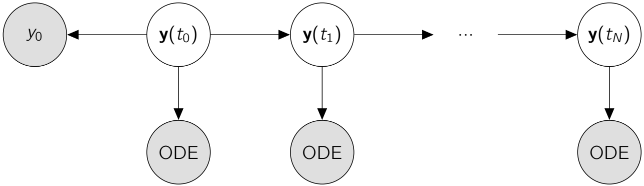
$$F = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \quad L = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Discretised prior

Time-grid $\mathbf{T} = (t_0, ..., t_N)$, then:

$$p(\mathbf{y}(t_{n+1}) \mid \mathbf{y}(t_n)) = N(\Phi(\Delta t_n)\mathbf{y}(t_n), \Sigma(\Delta t_n)), \quad p(\mathbf{y}(t_0)) = N(m_0, C_0)$$

with computable Φ and Σ.

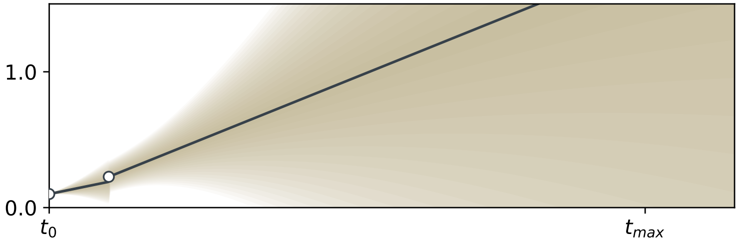


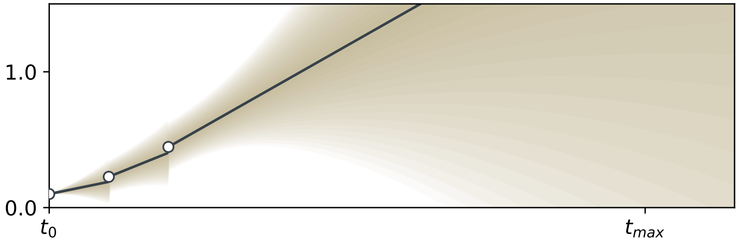
Algorithm

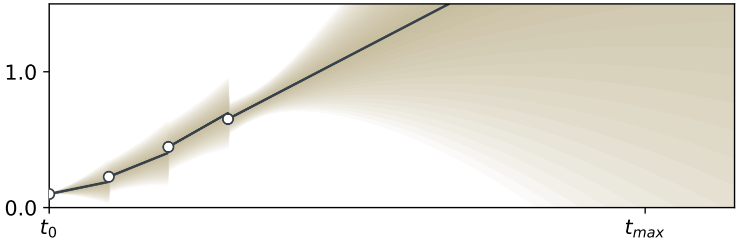
1. Initialise $p(\mathbf{y}(0) = N(m_0, C_0)$
2. Condition $p(\mathbf{y}(0) \mid y(0) = y_0)$
3. For $n = 1, ..., N$:
   3.1 Linearise $f(x) \approx A_n x + b_n$; ODE becomes $\dot{y}(t) \approx A_n y(t) + b_n$
   3.2 Correct

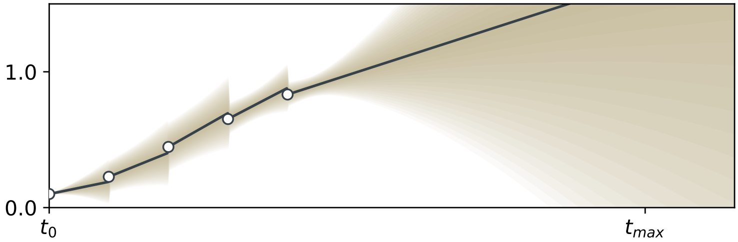
$$
\begin{aligned}
& p(\mathbf{y}(t_n) \mid \dot{y}(t_n) = A_n y(t_n) + b_n,\ \dot{y}(\mathbf{T}_{1:n-1}) = f(y(\mathbf{T}_{1:n-1}),\ y(0) = y_0) \\
\approx\ & p(\mathbf{y}(t_n) \mid \dot{y}(\mathbf{T}_{1:n}) = f(y(\mathbf{T}_{1:n}), y(0) = y_0))
\end{aligned}
$$

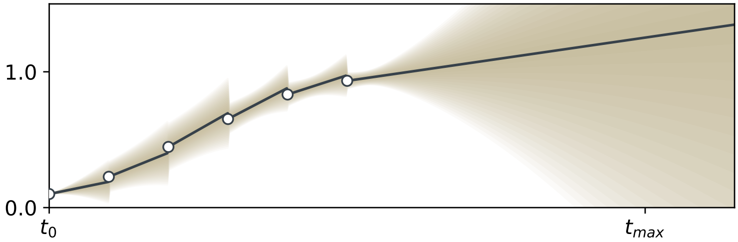
   3.3 Extrapolate $p(\mathbf{y}(t_{n+1}) \mid \dot{y}(\mathbf{T}_{1:n}) = f(y(\mathbf{T}_{1:n})), y(0) = y_0)$
4. Do something with the probabilistic numerical ODE solution

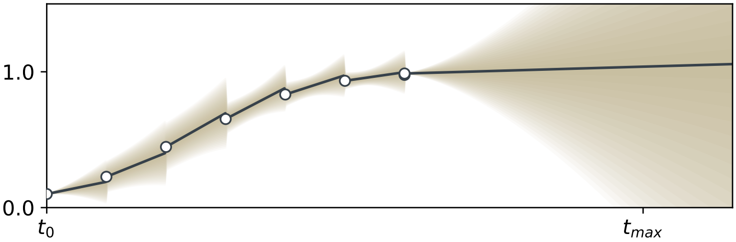

1.0
0.0
t_0
t_{max}

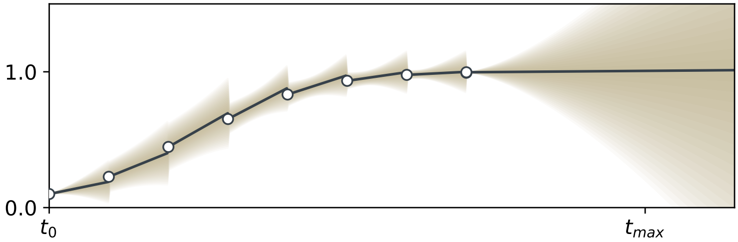

1.0
0.0
t_0
t_{max}

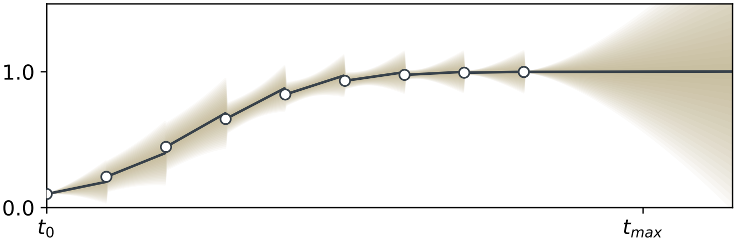

1.0
0.0
t_0
t_{max}


1.0
0.0
t_0
t_{max}

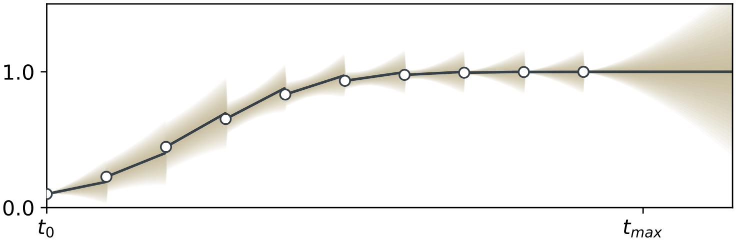

1.0
0.0
t_0
t_{max}

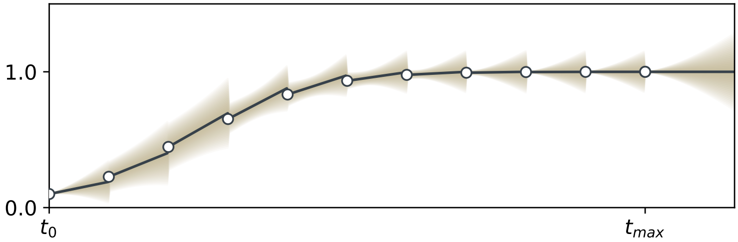

1.0
0.0
t_0
t_{max}

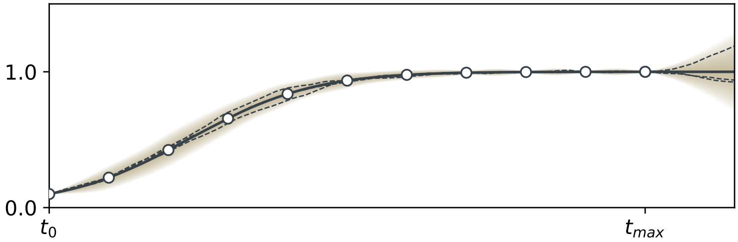

1.0
0.0
t_0
t_{max}



1.0
0.0
t_0
t_{max}


1.0
0.0
t_0
t_{max}

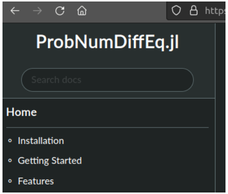

1.0
0.0
t_0
t_{max}


1.0
0.0
t_0
t_{max}

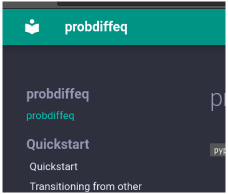
State of the art algorithm

1. Initialise whole state $p(\mathbf{y}(0))$ exactly
2. Guess an initial step-size Δt
3. While $t_n < 1$:
   - 3.1 Apply preconditioner
   - 3.2 Extrapolate in square-root form
   - 3.3 Compute smoothing gains (optional)
   - 3.4 Un-apply preconditioner
   - 3.5 Linearise $f \approx A_n x + b_n$
   - 3.6 Compute marginal likelihood
   - 3.7 Calibrate hyperparameters
   - 3.8 Estimate error
   - 3.9 Reject step if error too large
   - 3.10 Complete correction
   - 3.11 Propose new time-step
4. Do something with the probabilistic numerical ODE solution

```
] add ProbNumDiffEq
```




```
pip install probdiffeq
```



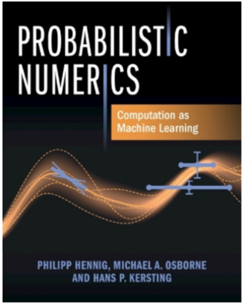
Conclusion

To build a (probabilistic) numerical algorithm

- ◇ Write down the prior belief
- ◇ Separate the information from the quantities of interest
- ◇ Modify the algorithm according to what the problem dictates
- ◇ Be clever about the implementation

More about the "how":

Hennig, Osborne, Kersting. Probabilistic Numerics. Cambridge University Press, 2022.



“Why”

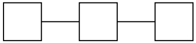
Back to why we are doing this

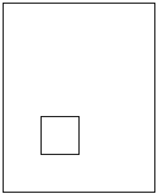
- Why explicit prior and posterior beliefs?
- Why separate the information from the quantity of interest?
- In other words: why take a probabilistic perspective?

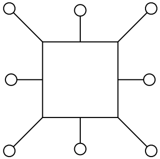
?

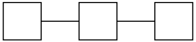
Traditional algorithms don't do that.

Here is why they should.









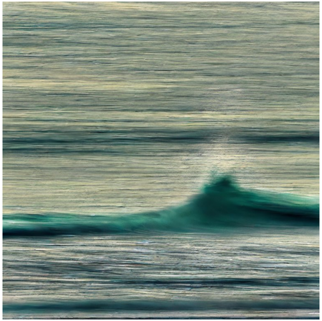


Image: Stable diffusion

Partial differential equations

Partial differential equation

$$\frac{\partial}{\partial t} u(t, x) = \frac{\partial^2}{\partial x^2} u(t, x), \quad u(0, x) = u_0(x)$$

Why?

- ◇ Spatiotemporal dynamics
- ◇ Climate, geophysics, airplanes, and so on
- ◇ Require large-scale computations

Solving PDEs as ODEs

Partial differential equation

$$\frac{\partial}{\partial t} u(t, x) = \frac{\partial^2}{\partial x^2} u(t, x), \quad u(0, x) = u_0(x)$$

Let $X := (x_0, ..., x_N)$ be some grid. Track only $U(t) = u(t, X) = [u(t, x_n)]_{n=0}^N$. Approximate

$$\frac{\partial^2}{\partial x^2} U(t) \approx \frac{1}{h^2} \begin{pmatrix} -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -1 & 2 & -1 \end{pmatrix} U(t) =: WU(t)$$

Solving PDEs as ODEs (continued)

Solve the PDE as an ODE: $\dot{U}(t) = AU(t)$, $U(0) = u_0(X)$

Advantages:

- Use any ODE solver
- Use any numerical differentiation method
- Move PDE-solving (unfamiliar territory) to ODE-solving (familiar territory)

What does this look like for probabilistic numerical solvers?

Posterior distribution

$$p(U \mid \dot{U}(\mathbf{T}) = AU(\mathbf{T}), U(0) = u_0(\mathbf{X}))$$

compute sequentially as usual.

Disadvantage: Numerical differentiation discards information

There must be a better way!

We know the way!

We know *probabilistic numerical differentiation*:

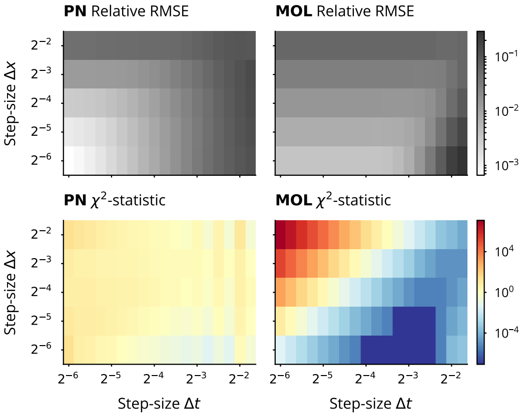
- Prior: $p(u) = \mathrm{GP}(0, k_t \otimes k_x)$, where $(k_t \otimes k_x)(t, t', x, x') = k_t(t, t')k_x(x, x')$
- Then, $p(\partial_x^2 u(\cdot, \mathbf{X}) \mid u(\cdot, \mathbf{X})) = \mathrm{GP}(Wu(\cdot, \mathbf{X}), k_t \otimes E) = Wu(\cdot, \mathbf{X}) + \xi(\cdot)$
- The PDE solution is

$$\begin{aligned} & p(u \mid \partial_t u(\mathbf{T}, \mathbf{X}) = \partial_x^2 u(\mathbf{T}, \mathbf{X}), u(0, \mathbf{X}) = u_0(\mathbf{X})) \\ = \; & p(u, \xi \mid \partial_t u(\mathbf{T}, \mathbf{X}) = Wu(\mathbf{T}, \mathbf{X}) + \xi(\mathbf{T}), u(0, \mathbf{X}) = u_0(\mathbf{X})) \end{aligned}$$

- Track differentiation error as model discrepancy

Calibrate the *PDE* solver

"PN" = "Probabilistic numerics"; "MOL"="Method of lines" (non-probabilistic).

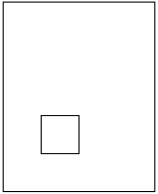


PDE solvers: pipelines of computation

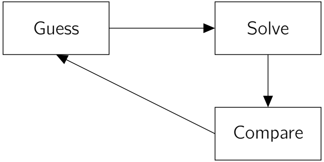
- ◇ Discretise spatial domain probabilistically.
- ◇ Compute spatiotemporal PDE solution without an unnecessary loss of information.

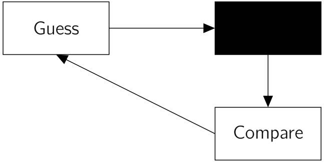
Krämer, Schmidt, Hennig.
Probabilistic Numerical Method of Lines for Time-Dependent Partial Differential Equations.
AISTATS 2021.

- ◇ *Don't throw information*
- ◇ *Especially not if future computations depend on it*



Real-life dynamical systems





- ◇ We know: $y(t)$ solves $\dot{y}(t) = f(y(t), \theta)$, $y(0) = y_0$.
- ◇ We know: $y_k = y(t_k) + \epsilon = 4$, $\epsilon \sim N(0, 0.1^2)$, $k = 1, ..., K$
- ◇ What is θ?

Parameter estimation

Abbreviate:

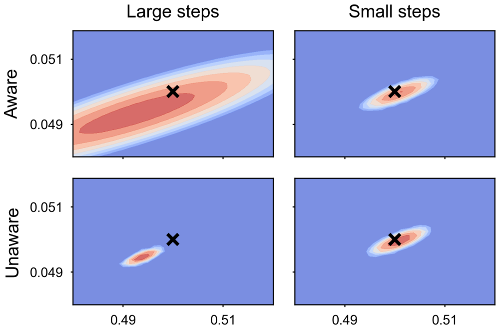
$$\pi(y \mid \theta) := p(y \mid \{\dot{y}(t_n) = f(y(t_n), \theta)\}_{n=0}^{N}, y(t_0) = y_0, \theta)$$

Marginalise ("average") likelihood of observations over all IVP solutions:

$$M(\theta) = p(\{y_k\}_{k=1}^{K} \mid \theta) \approx \int p(\{y_k\}_{k=1}^{K} \mid y)\pi(y \mid \theta)dy$$

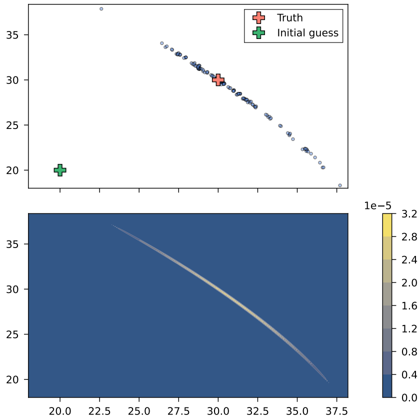
Run (gradient-based) MCMC or optimisation schemes.

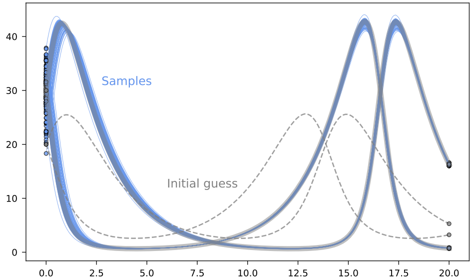
Averaging loss functions over probabilistic solutions



Kersting, Krämer, Schiegg, Daniel, Tiemann, Hennig.
Differentiable likelihoods for fast inversion of 'likelihood-free' dynamical systems.
ICML 2020.

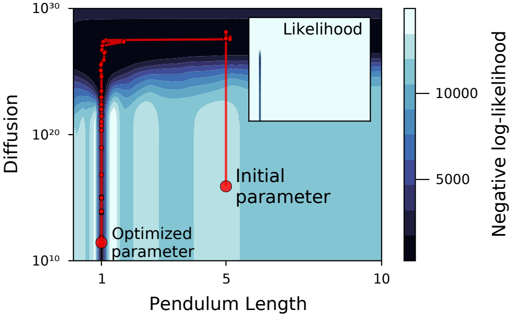
Probabilistic solvers & MCMC





High resolution images:

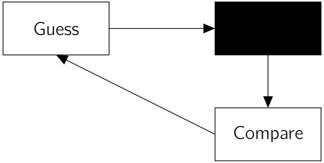
https://pnkraemer.github.io/probdiffeq/

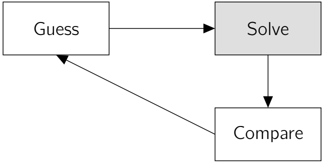


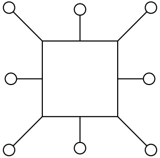
Tronarp, Bosch, Hennig.
Fenrir: Physics-enhanced regression for initial value problems.
ICML 2022.

Conclusion

- ◇ If you build a statistical model around a numerical algorithm:
  Use prior and posterior beliefs as much as you can
- ◇ Marginalise over probabilistic solutions







More parameter estimation

- We know: $\dot{y}(t) = f(y(t), \beta(t))$, $y(0) = y_0$
- We also know: $y_k = y(t_k) + \epsilon$, $p(\epsilon) = N(0, \sigma^2)$, $k = 1, ..., K$

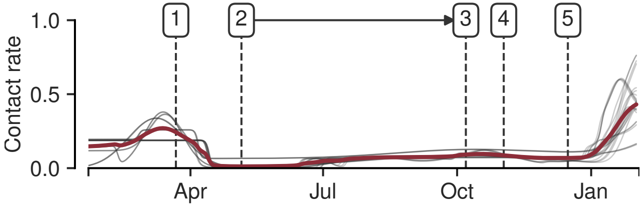
β is an unknown function!

How do people usually solve this?

- Assume finitely many features

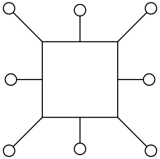
$$\beta(t) = \sum_{\ell=1}^{L} \beta_\ell \phi_\ell(t)$$

- Tune β by tuning L parameters $(\beta_\ell)_{\ell=1}^{L}$
- Use any optimiser or MCMC. E.g. in the SIRD model:

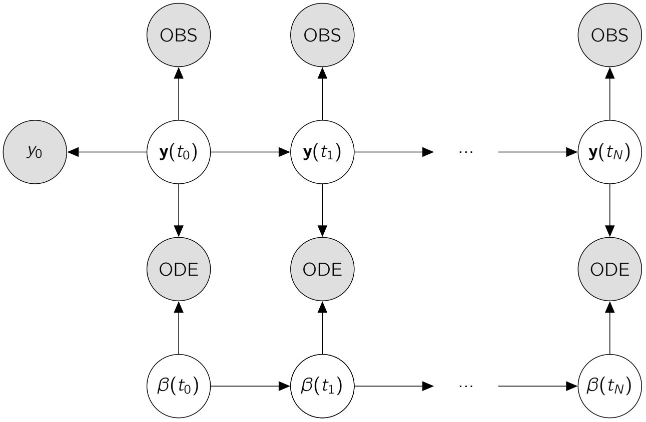


How do *we* solve this?

- Prior: $p(y) = \mathrm{GP}(0, k_1)$, $p(\beta) = \mathrm{GP}(0, k_2)$
- Information: $\dot{y}(\mathbf{T}) = f(y(\mathbf{T}), \beta(\mathbf{T}))$, $y(0) = y_0$, $y_k = y(t_k) + \epsilon$, $k = 1, ..., K$
- Conditioning as in the ODE solver setting

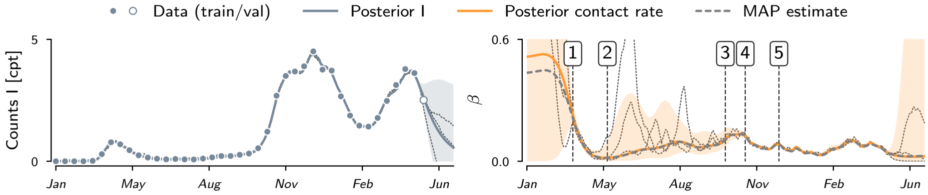


How do we solve this? (continued)



Posterior distribution

$$p(y, \beta \mid \dot{y}(\mathbf{T}) = f(y(\mathbf{T}), \beta(\mathbf{T})), y_k = y(t_k) + \epsilon)$$



Model discrepancy in the SIRD model

Problem

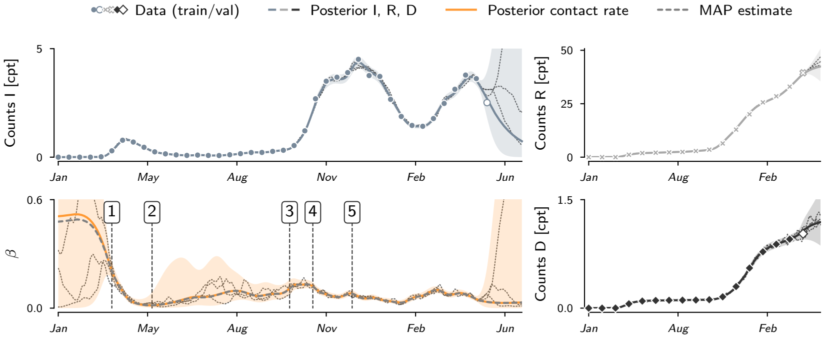
Positivity: $y(t) > 0$ guaranteed

SIRD model has issues

Solution

Assume $y = \exp(\tilde{y})$, $p(\tilde{y}) = \mathrm{GP}(0, k)$

Model the discrepancy $\dot{y}(\mathbf{T}) = f(y(\mathbf{T}), \beta(\mathbf{T})) + \xi(\mathbf{T})$



Conclusion

- ◇ Write down all sources of information
- ◇ Discretise and approximate as late as possible

Schmidt, Krämer, Hennig.
A Probabilistic State Space Model for Joint Inference from Differential Equations and Data.
Neurips 2021.

Epilogue

Numerical algorithms drive machine learning

But real-life starts when traditional treatments of numerical algorithms stop.

Dissect your algorithm:

- ◇ Prior distribution
- ◇ Information sources
- ◇ Conditioning methods
- ◇ Quantities of interest

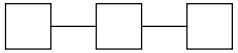
Results

- ◇ Numerical integration
- ◇ Numerical differentiation
- ◇ PDE solvers
- ◇ (Nonlinear) ODE solvers

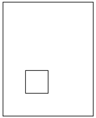
Do as the problem dictates.

Not as the solver requires.

PDEs:



Parameter estimation:



Multi-source: