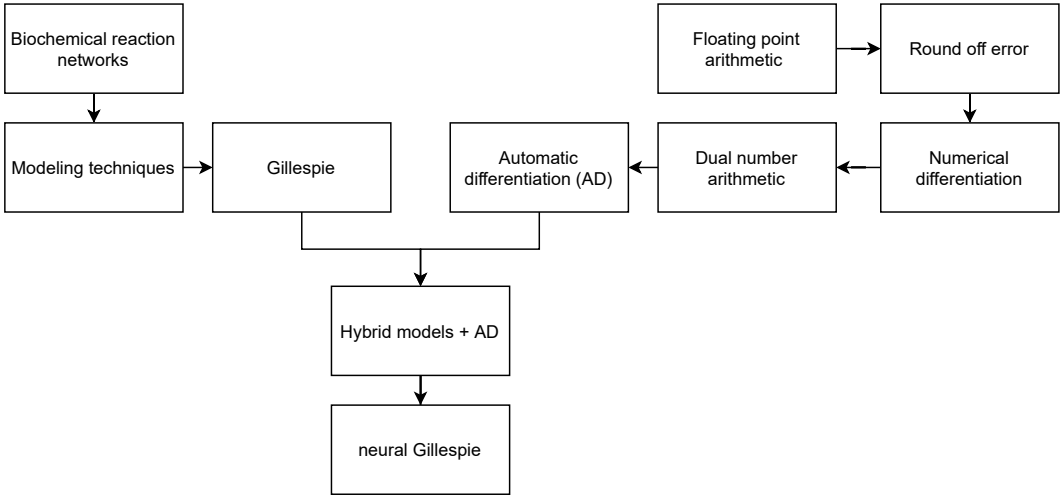
Project proposal

Accelerating Biochemical reaction models: A neural Gillespie Algorithm

Elisabeth Roesch, Chris Rackauckas, (Alex Lenail, Frank Schäfer, Yingbo Ma,) Michael Stumpf.

15th of August, 2021

Talk overview



Modeling biochemical reaction networks

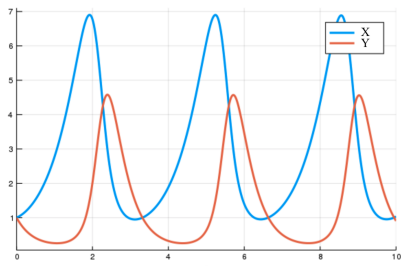
Example: Lotka -Volterra model

$$Y_1 \xrightarrow{k_1} 2Y_1,$$

$$Y_1 + Y_2 \xrightarrow{k_2} 2Y_2,$$

$$Y_2 \xrightarrow{k_3} \emptyset,$$

where Y_1, Y_2 are species and k_1, k_2, k_3 are rate constants.



Mass-action kinetics

$\rightarrow$ Use them to formulate **rates** based on the reactions and rate constants.
$\rightarrow$ These laws hold **only** under certain physical assumptions, such as "large molecule numbers for all species". [1]

Example: Second reaction of LV model:

- Reaction: $Y_1 + Y_2 \xrightarrow{k_2} 2Y_2$
- Rate based on mass action kinetics: $k_2[Y_1][Y_2]$

Deterministic, continuous case:

- We are not counting molecules but species are measured as e.g. moles/liter.
- Using mass-action kinetics, we can get ODE models.
- This *can* be good starting point as approximation even if we know the true process is discrete and stochastic.

[1] Details in e.g. D Schnoerr, G Sanguinetti, R Grima , "Approximation and inference methods for stochastic biochemical kinetics", Journal of Physics A: Mathematical and Theoretical, 2017.

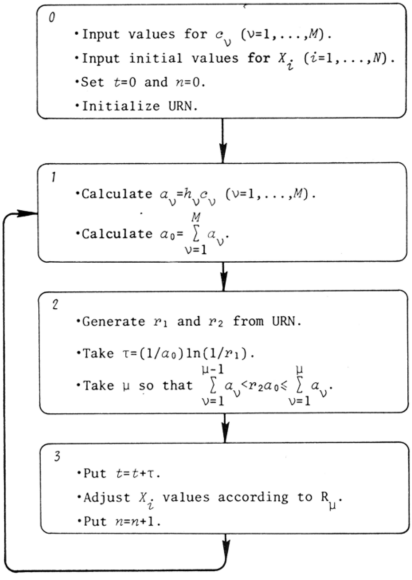
Stochasticity and discrete nature of true process

- Generally: Impact of stochasticity and discrete nature of the true process too significant to solely rely on "macroscopic" view via deterministic approximation.
- From molecular perspective: Time evolution of current state of the system is continuous time **Markov process with discrete state space**.
- Extension to stochastic mass-action kinetics:
  $\rightarrow$ Construct stochastic rates, based on reactions and rate constants.

Example: Second reaction of LV model:

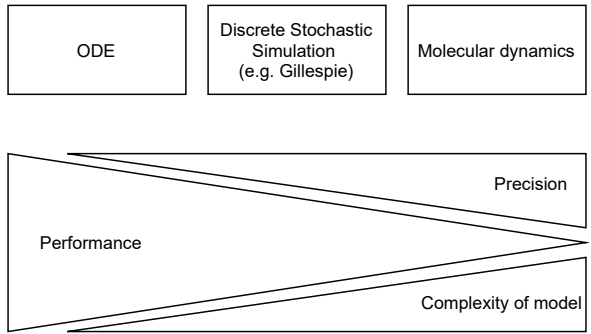
- Reaction: $Y_1 + Y_2 \xrightarrow{k_2} 2Y_2$
- Rate based on stochastic mass action kinetics: $k_2 y_1 y_2$

Gillespie algorithm (Gillespie, 1977)



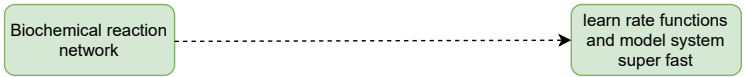
- Based on reactions and rates
  $\rightarrow$ Make exact simulation of true process.
- Most famous way to do this:
  $\rightarrow$ Gillespie algorithm.
- A single Gillespie simulation:
  $\rightarrow$ An **exact** sample from the probability mass function that is the solution of the master equation.

Context and limitations of Gillespie algorithm



- Models ever reaction event $\rightarrow$ Can be very slow.
- Biggest limitation: Speed.

Goal of project: Develop a new approach to make Gillespie faster.



How? Using a neural approach.

Differentiating the Gillespie algorithm

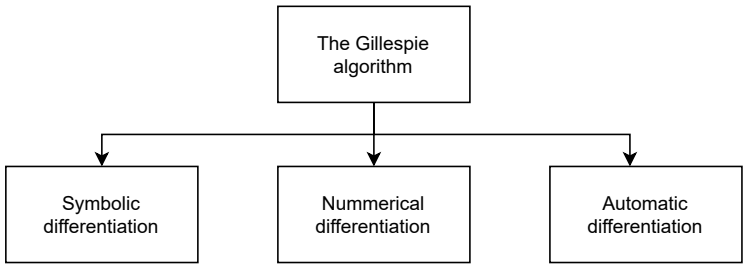
Question: What is the derivative with respect to p, where p is in the rate but not in the update function?

Example: Second reaction of LV model:

- Reaction: $Y_1 + Y_2 \xrightarrow{k_2} 2Y_2$
- Rate based on stochastic mass action kinetics: $k_2 y_1 y_2$
- p could be for example k_2.

$\rightarrow$ So p influences distribution of dt.
Reminder: $\Delta t = rand(Exp(sum(\textbf{rates})))$

How do we differentiate something like this?



Numerical differentiation

Example:

- For function $f : \mathbb{R} \to \mathbb{R}$, the numerical differentiation via **finite differences** is given by,

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

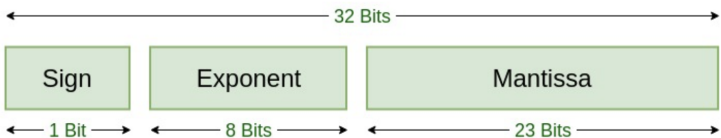
  where h represents a small change in x, $x \in \mathbb{R}$.
- Smaller step size h $\to$ better approximation.

The floating-point arithmetic

Floating-point arithmetic:

Representation of real numbers via floating-point arithmetic as an approximation (In Julia: e.g. Float32):

$$x = -32 * 10^{-5}$$



- Finite number of bits to save the real number x (trade off between range and precision).

Round off error

- One would assume: For $h \in \mathbb{R}$, it holds that (1+h)-1=h.
- Code example: Fix h and then do h2 = (1+h)-1. Question: Is h==h2?

```
julia> h = 1e-1rand()
0.09399514921175445

julia> h2 = (1+h)-1
0.09399514921175456
```

Make h smaller

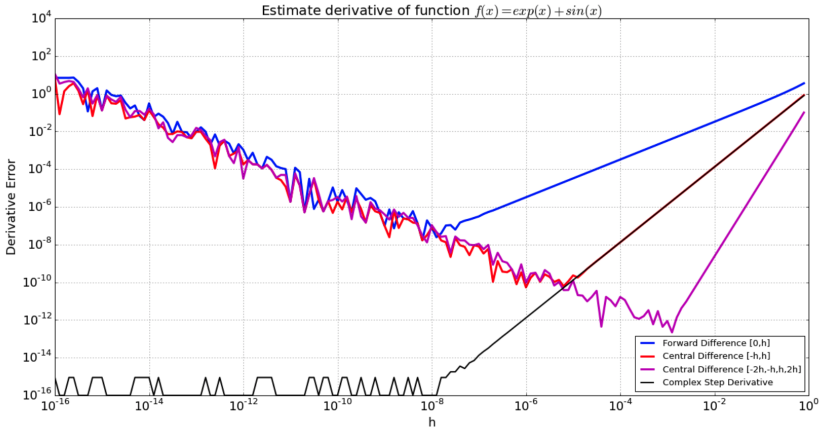
```
julia> h = 1e-10rand()
9.906183448919843e-11

julia> h2 = (1+h)-1
9.906186981822884e-11
```

- Number of lost digits increases as the two numbers become more different.

$$\begin{array}{r} 5000 \\ +\ \ 001.1 \\ \hline 5001 \end{array} \qquad \begin{array}{r} 5000 \\ +\ \ 001.1 \\ \hline 5011 \end{array}$$

- Large number + small number (such as a very small h) $\rightarrow$ Large round off error.

Round off error in numerical differentiation



- Floating point representation + numerical differentiation can be very dangerous.
- Focus on black line! $\rightarrow$ Here round off error not there. Why?

Complex step differentiation: Differencing in a different dimension

- Idea: Let $x \in \mathbb{R}$ and $f : \mathbb{R} \to \mathbb{R}$ analytic.
  $\to$ We want to calculate the *real* derivative wrt x.
- Taylor series expansion (in complex direction): $\to$ Add perturbation h to x, but in complex direction.

$$f(x + ih) = f(x) + f'(x)ih + \mathcal{O}(h^2) \iff if'(x) = \frac{f(x + ih) - f(x)}{h} + \mathcal{O}(h)$$

$$\iff f'(x) = \frac{Im(f(x + ih))}{h} + \mathcal{O}(h).$$

- Note h never directly interacts with x as in the **two dimensions of the complex numbers**. $\to$ Keeping small perturbation separate form large ones.
- In code: h and x – as the real and imaginary part of a complex number – are stored in two different floating point representations (e.g. **two** Float64).
- This method brings us to automatic differentiation: AD extends the idea of complex step differentiation beyond complex analytic functions.

New arithmetic: Dual numbers

```
julia> struct Dual{T}
           val::T   # value
           der::T  # derivative
       end
```

Idea: Derivative measures **sensitivity** of function: How much changes the function output when input changes by small ϵ

$$f(a+\epsilon) = f(a) + \epsilon f'(a) + o(\epsilon).$$

Formally set: $\epsilon^2 = 0$ and function f be will represented by value $f(a)$ and the derivative $f'(a)$, encoded as the coefficients of a degree-1 Taylor polynomial in ϵ:

$$\rightsquigarrow f(a+\epsilon) = f(a) + \epsilon f'(a).$$

$\rightarrow$ Set rules for functions on dual numbers[2].

[2] John L. Bell, "A Primer of Infinitesimal Analysis", Cambridge University Press, 2008.

Functions of dual objects

Given

$$f(a+\epsilon) = f(a) + \epsilon f'(a)$$

and

$$g(a+\epsilon) = g(a) + \epsilon g'(a)]$$

we manipulate the Taylor expansions to combine the functions.

Sum rule:

$$(f+g) = [f(a) + g(a)] + \epsilon[f'(a) + g'(a)]$$

Product rule:

```
julia> # Product Rule
       Base.:*(f::Dual, g::Dual) = Dual(f.val*g.val, f.der*g.val + f.val*g.der)
```

$\rightarrow$ Infer derivatives by taking the component of ϵ.

Example differentiation

→ Based on arithmetic of dual numbers, we can differentiate an arbitrary function h.
→ We present $a + \epsilon$ as Dual(a,1) which gives Dual(h(a),h'(a)) when applying h to it.

```
julia> h(x) = x^2 + 2
h (generic function with 1 method)

julia> a = 3
3

julia> xx = Dual(a, 1)
Dual{Int64}(3, 1)

julia> h(xx)
Dual{Int64}(11, 6)
```

Test: $h(a) = 11,\ h'(a) = 2a = 6 \rightarrow$ Correct.

Extension: Higher dimensional functions

- Differentiating functions where $gg : \mathbb{R}^n \to \mathbb{R}$ such as $gg(x, y) = x^2 * y + x + y$
- In AD, we add n independent partial derivative components to dual numbers:

Multidimensional dual number in julia:

```
julia> struct MultiDual{N,T}
           val::T
           derivs::SVector{N,T}
       end
```

Defining primitive functions on them (via multiple dispatch):

```
julia> function +(f::MultiDual{N,T}, g::MultiDual{N,T}) where {N,T}
           return MultiDual{N,T}(f.val + g.val, f.derivs + g.derivs)
       end
+ (generic function with 203 methods)
```

AD on higher dimensional functions

$$gg(x, y) = x^2 * y + x + y.$$

Example: $(x, y) = (a, b)$ where $a = 1, b = 2$.

- $gg(x, 2) = x^2 * 2 + x + 2 \iff gg(x, 2) = 2x^2 + x \xrightarrow{diff_x} gg'(x) = 4x + 1$
- $gg(1, y) = 1^2 * y + 1 + y \iff gg(1, y) = 2y + 1 \xrightarrow{diff_y} gg'(x) = 2$

```
julia> gg(x, y) = x**y + x + y
gg (generic function with 1 method)

julia> (a, b) = (1.0, 2.0)
(1.0, 2.0
)

julia> xx = MultiDual(a, SVector(1.0, 0.0))
MultiDual{2, Float64}(1.0, [1.0, 0.0])

julia> yy = MultiDual(b, SVector(0.0, 1.0))
MultiDual{2, Float64}(2.0, [0.0, 1.0])

julia> gg(xx, yy)
MultiDual{2, Float64}(5.0, [5.0, 2.0])
```

Test: $gg(1, 2) = 2 + 1 + 2 = 5$, $gg'(1, 2) = (4x + 1, 2) = (5, 2) \rightarrow$ Correct.

Differentiating functions with multiple outputs

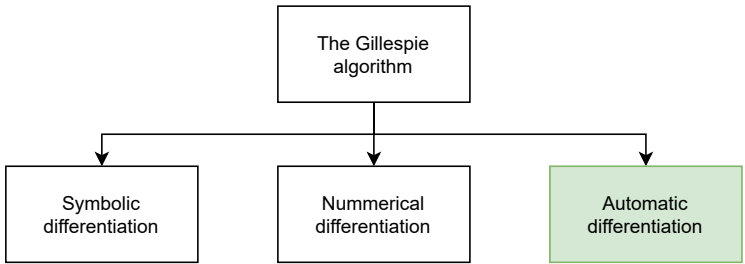
- Differentiating functions where $ff : \mathbb{R}^n \to \mathbb{R}^m$ such as $ff(x, y) = (x * x + y * y, x + y)$.
- In AD, we add calculate the mXn matrix of partial derivative components (e.i the Jacobian):

```
julia> ff(x, y) = SVector(x*x + y*y , x + y)
ff (generic function with 1 method)

julia> ff(xx, yy)
2-element SVector{2, MultiDual{2, Float64}} with indices SOneTo(2):
 MultiDual{2, Float64}(5.0, [2.0, 4.0])
 MultiDual{2, Float64}(3.0, [1.0, 1.0])
```

In red: Value and partial derivatives.

Summary: Dual numbers and AD + Back to Gillespie

Question: What is the derivative of the Gillespie algorithm with respect to p, where p is in the rate but not in the update function?
Answer: AD takes a program $\rightarrow$ recompiles it in this different arithmetic of dual number $\rightarrow$ Accurate and fast calculation of directional derivatives.



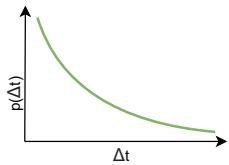
Problematic with AD and Gillespie

- Δt is a constant.
- not dual $\rightarrow$ derivative part of dual is zero.

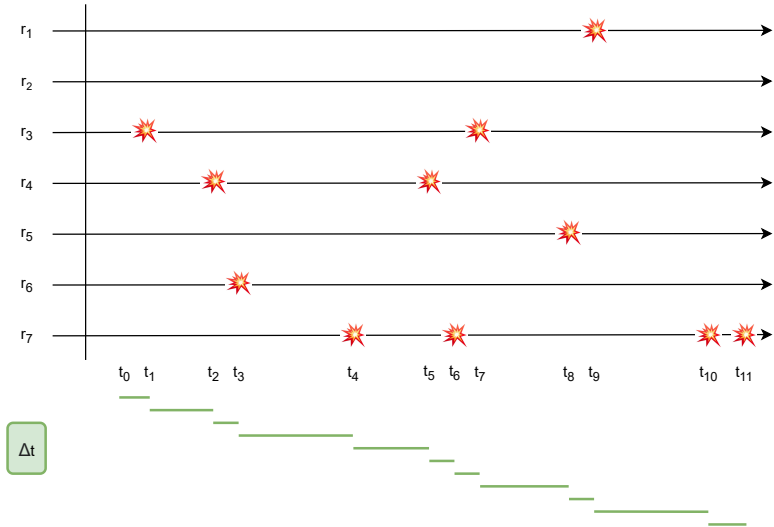
$\rightarrow$ this breaks the whole differentiation chain.

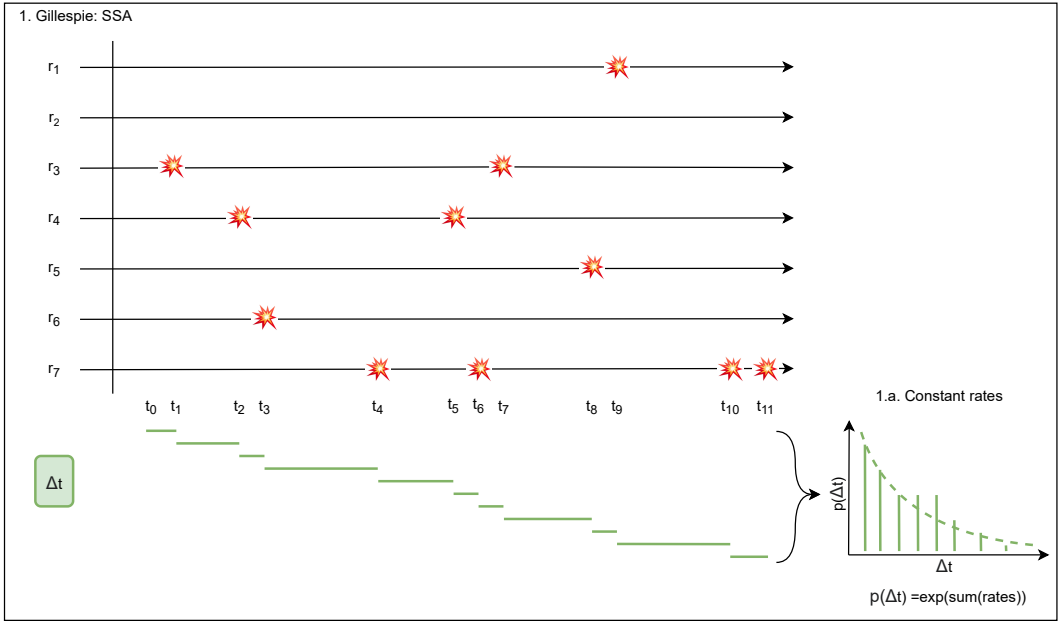
```
julia> h(x) = x^2 + 2
h (generic function with 1 method)

julia> h(3)
11

julia> xx=Dual(a,1)
Dual{Int64}(3, 1)

julia> h(xx)
Dual{Int64}(11, 6)

julia> xxx=Dual(a,0)
Dual{Int64}(3, 0)

julia> h(xxx)
Dual{Int64}(11, 0)
```

1. Gillespie: SSA

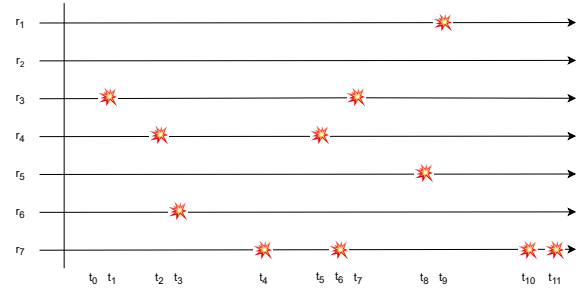


$p(\Delta t) = \exp(\text{sum}(\text{rates}))$




1. Gillespie: SSA
r1
r2
r3
r4
r5
r6
r7
t0 t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11
Δt
1.a. Constant rates
p(Δt)
Δt
p(Δt) =exp(sum(rates))

1. Gillespie: SSA

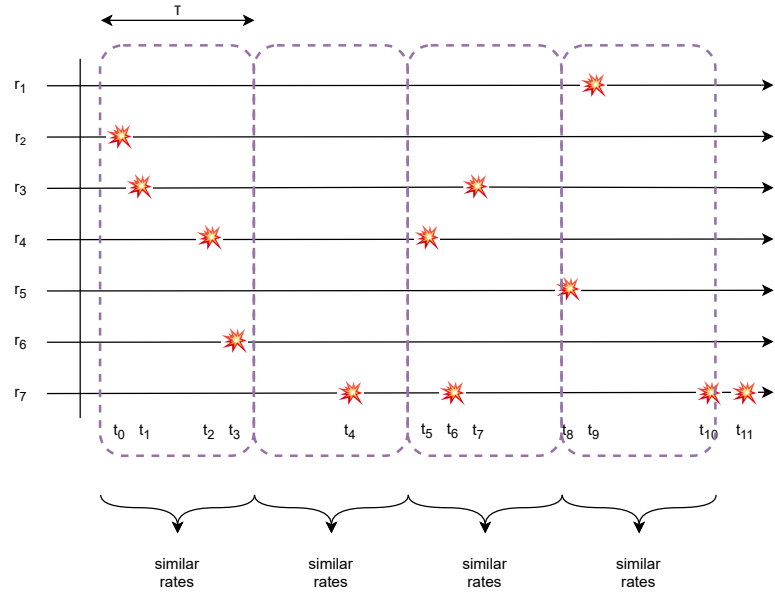


1.b. Non-constant rates

$p_i(\Delta t) = \exp(\text{sum}(\text{rates}_i))$

$p_j(\Delta t) = \exp(\text{sum}(\text{rates}_j))$

2. τ- leaping


τ
r1
r2
r3
r4
r5
r6
r7
t0 t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11
similar rates
similar rates
similar rates
similar rates

τ-leaping method: Approximating the SSA to be faster.

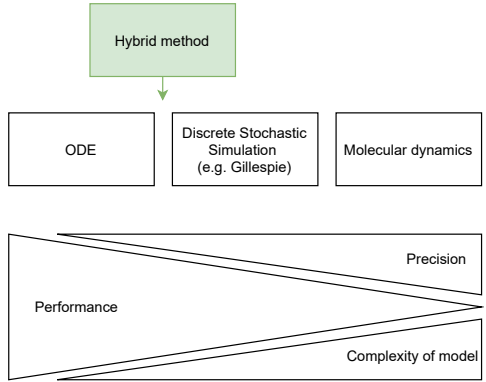
Leap Condition: Require τ to be small enough that the change in the state during $[t, t+\tau]$ will be so slight that no propensity function will suffer an appreciable(i.e., macroscopically noninfinitesimal) change in its value.

τ-leaping method: Select largest τ that fulfils the Leap Condition.

τ-leaping method

- Helpful for speeding up the SSA.
- Popular option for speeding up the SSA.
- But: I don't see how it is helpful for the differentiation problem we want to solve.
- We want to use a neural approach to speed up Gillespie, we are looking at other methods to find something that helps us with the differentiation problem, not speed up directly.

Hybrid methods



- Also method to speed up SSA.
- Slow reactions with species that occur in small numbers $\rightarrow$ SSA.
- Reactions where all species in large numbers $\rightarrow$ Reaction rate equations (ODEs).

Hybrid methods

- Linda Petzold: "We thought it's great, then we tried on our real problem and it failed." [3]
- Postulated reason: Cannot efficiently handle fast reactions involving species present in small numbers.
- Their published solution: Slow scale SSA (ssSSA) [4]
  $\rightarrow$ Stochastic partial equilibrium approximation for the not addressed ones.
- But: Hybrid models are differentiable!

[3] Talk by Petzold: "Discrete stochastic simulation of spatially inhomogeneous biochemical systems", published 10 Nov 2012: https://www.youtube.com/watch?v=vvE4U7o51cU&ab_channel=MathsStatsUNSW

[4] Cao, Petzold, Gillespie, "The slow-scale stochastic simulation algorithm", THE JOURNAL OF CHEMICAL PHYSICS, 2004.

Hybrid models

- In Julia, implemented as VariableRateJumps combinable with normal ODEs and **differentiable**.
- Use ODE formulation to derive sensitivity for strong solution u(t) wrt. p for set of random numbers.
- Sequential algorithm $\rightarrow$ we propagate through the sequence of events $\rightarrow$ we get sensitivity for each event $\rightarrow$ brings us back to discrete space.
- Average sensitivities to get the gradient.
- TO DO: Show that exchange of derivatives i.e. expected value of derivative is equal to the derivative of the expected value.

Project overview and proposal