OCaml Users in PariS: Epidemiological inference in OCaml

Benjamin Nguyen-Van-Yen

March 10, 2022

Who am I ?

- I work in the SED of INRIA Saclay for the Tropical team, on the organization of emergency call centers (15/17/18/112).

Who am I ?

- I work in the SED of INRIA Saclay for the Tropical team, on the organization of emergency call centers (15/17/18/112).
- I did a PhD at Institut Pasteur and ENS about Bayesian inference of infectious disease dynamics.

Who am I ?

- I work in the SED of INRIA Saclay for the Tropical team, on the organization of emergency call centers (15/17/18/112).
- I did a PhD at Institut Pasteur and ENS about Bayesian inference of infectious disease dynamics.
- And I decided to do it in OCaml...

Some of the things I did during my PhD

- Simulations of SIR-like ODE models

Some of the things I did during my PhD

- Simulations of SIR-like ODE models
- Simulations of SIR-like stochastic models

Some of the things I did during my PhD

- Simulations of SIR-like ODE models
- Simulations of SIR-like stochastic models
- Simulation of sequence evolution in a SIR-like model

Some of the things I did during my PhD

- Simulations of SIR-like ODE models
- Simulations of SIR-like stochastic models
- Simulation of sequence evolution in a SIR-like model
- Inference by MCMC

Some of the things I did during my PhD

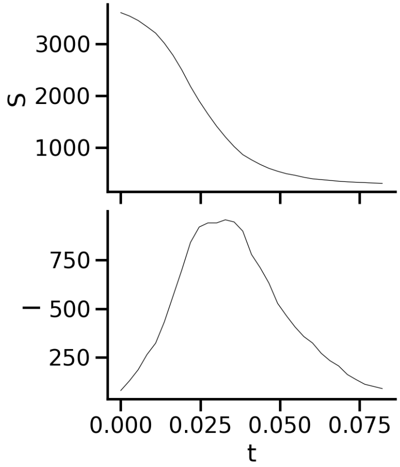
- Simulations of SIR-like ODE models
- Simulations of SIR-like stochastic models
- Simulation of sequence evolution in a SIR-like model
- Inference by MCMC
- Inference by MCMC for stochastic models

Some of the things I did during my PhD

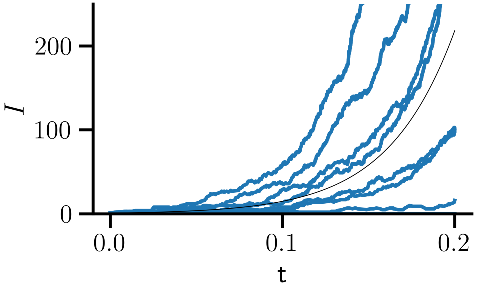
- Simulations of SIR-like ODE models
- Simulations of SIR-like stochastic models
- Simulation of sequence evolution in a SIR-like model
- Inference by MCMC
- Inference by MCMC for stochastic models
- Phylodynamic inference by MCMC

Some of the things I did during my PhD
Simulations of SIR-like ODE models

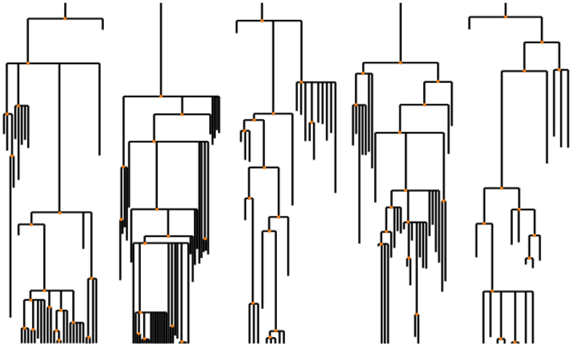
$$\begin{tikzcd} S \arrow[d, bend right, "\beta \frac{S}{N} I"'] \\ I \arrow[d, bend right, "\gamma I"'] \\ R \end{tikzcd}$$



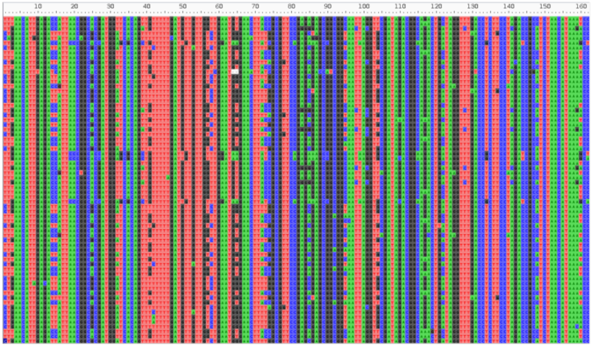
Some of the things I did during my PhD
Simulations of SIR-like stochastic models



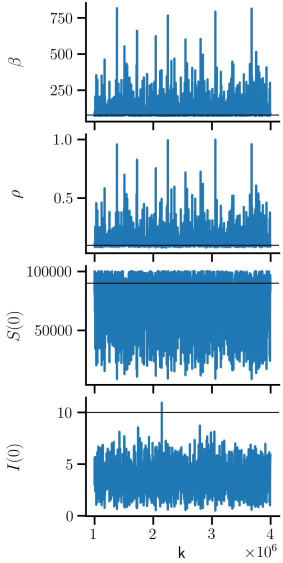
Some of the things I did during my PhD
Simulation of sequence evolution in a SIR-like model



Some of the things I did during my PhD
Simulation of sequence evolution in a SIR-like model

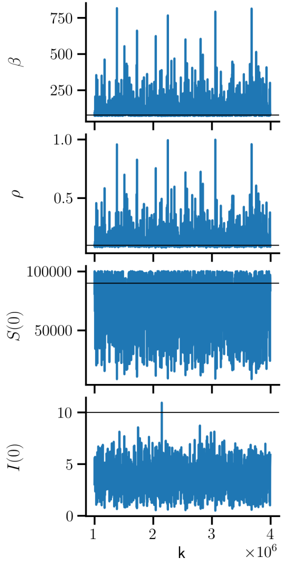


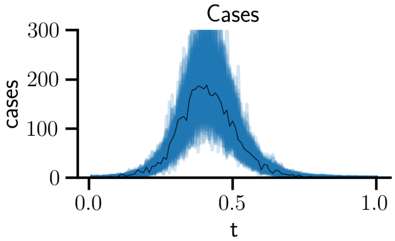
Some of the things I did during my PhD
Inference by MCMC



Some of the things I did during my PhD

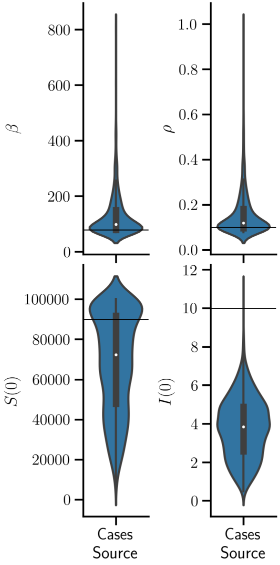
Inference by MCMC

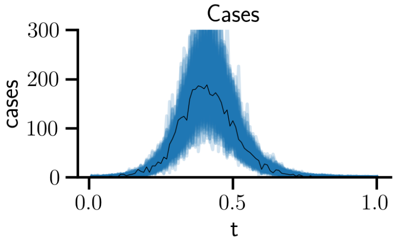




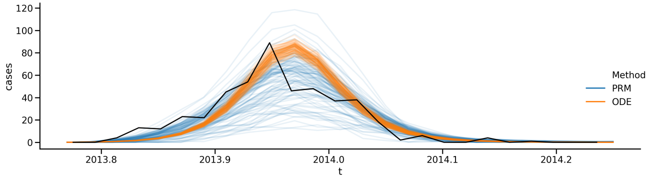
Some of the things I did during my PhD

Inference by MCMC

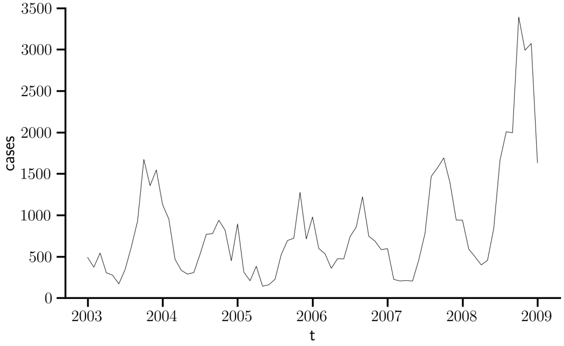


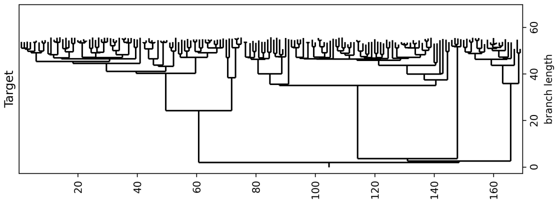


Some of the things I did during my PhD
Inference by MCMC for stochastic models



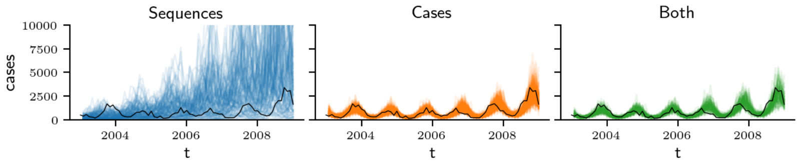
Some of the things I did during my PhD
Phylodynamic inference by MCMC





Some of the things I did during my PhD

Phylodynamic inference by MCMC



The problem

A range of models, of simulation algorithms, and of inference algorithms.

The problem

A range of models, of simulation algorithms, and of inference algorithms.

```
type params type model type traj
```

The problem

A range of models, of simulation algorithms, and of inference algorithms.

```
type params type model type traj

val model : params -> model
val simulate : model -> traj
```

The problem

A range of models, of simulation algorithms, and of inference algorithms.

```
type params type model type traj

val model : params -> model
val simulate : model -> traj

type observations

val infer : observations -> params
```

The problem

A range of models, of simulation algorithms, and of inference algorithms.

```
type params type model type traj
```

```
val model : params -> model
val simulate : model -> traj
```

```
type observations
```

```
val infer : rng -> observations -> params list
```

The problem

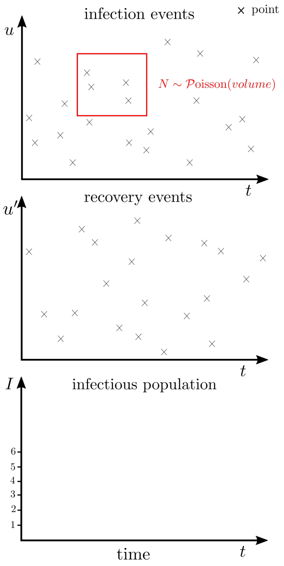
A range of models, of simulation algorithms, and of inference algorithms.

```
type params type model type traj

val model : params -> model
val simulate : model -> traj

type observations

val infer : rng -> observations -> params list
```

https://gitlab.com/bnguyenvanyen/ocamlecoevo

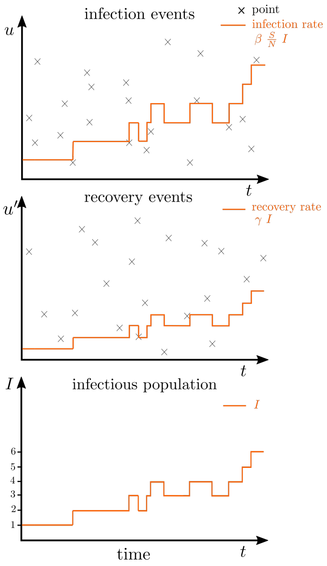
A few examples

Poisson Random measures

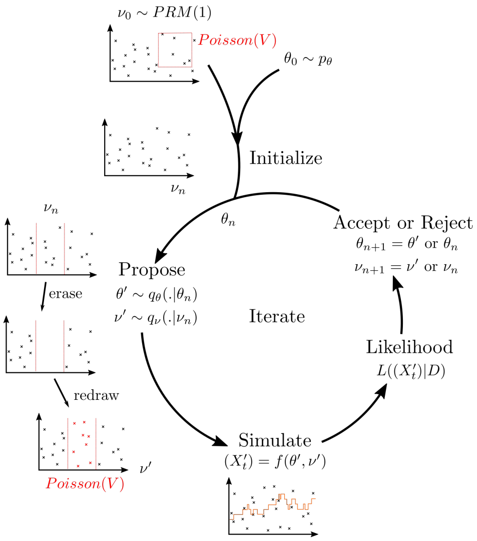


A few examples

Poisson Random measures



A few examples
Poisson Random measures



A few examples
Ecology-evolution dynamics: populations of individuals with traits

```
type id = [ `Id ]
type nonid = [ `Nonid ]

type 'a isid =
  | Isid of ('a, id) eq
  | Isnotid of ('a, nonid) eq

module type TRAIT = sig
  type 'a t

  type 'a group

  val isid : 'a group -> 'a isid

  val group_of : 'a t -> 'a group

  val of_group : nonid group -> nonid t
end
```

A few examples
Ecology-evolution dynamics: populations of individuals with traits

```
type id = [ `Id ]
type nonid = [ `Nonid ]

type 'a isid =
  | Isid of ('a, id) eq
  | Isnotid of ('a, nonid) eq

module type TRAIT = sig
  type 'a t

  type 'a group

  val isid : 'a group -> 'a isid

  val group_of : 'a t -> 'a group

  val of_group : nonid group -> nonid t
end
val count : 'a group -> t -> int
val choose : rng -> 'a group -> t -> 'a indiv
```

A few examples
SIR-like models

```
type idor
type 'a payload

type _ t =
  | S : nonid t
  | E : idor payload -> idor t
  | I : idor payload -> idor t
  | R : nonid t
  | C : nonid t
  | O : idor payload -> idor t

type _ group =
  | Sus : nonid group
  | Exp : idor group
  | Inf : idor group
  | Rem : nonid group
  | Cas : nonid group
  | Out : idor group

...
```

A few examples
SIR-like models

```
type idor
type 'a payload

type _ t =
  | S : nonid t
  | E : idor payload -> idor t
  | I : idor payload -> idor t
  | R : nonid t
  | C : nonid t
  | O : idor payload -> idor t

type _ group =
  | Sus : nonid group
  | Exp : idor group
  | Inf : idor group
  | Rem : nonid group
  | Cas : nonid group
  | Out : idor group

...
```

```
module Make (Get : GET) = struct
  type t = Get.t

  let leave_exposed par _ z =
    let e = Get.exp z in
    F.Pos.Op.((Param.sigma par) * e)

  let recovery par _ z =
    let i = Get.inf z in
    F.Pos.Op.((Param.nu par) * i)

  let immunity_loss par _ z =
    let r = Get.rem z in
    F.Pos.Op.((Param.gamma par) * r)

  ...
end
```

A few examples
Inference across simulation methods

```
type continuous_pop = ...
type discrete_pop = ...

type ode_sim = ...
type sde_sim = ...
type exact_sim = ...
type approx_sim = ...
type fast_sim = ...
```

A few examples
Inference across simulation methods

```
type continuous_pop = ...
type discrete_pop = ...

type ode_sim = ...
type sde_sim = ...
type exact_sim = ...
type approx_sim = ...
type fast_sim = ...
```

```
type _ pop =
  | Continuous : continuous_pop pop
  | Discrete : discrete_pop pop

type _ prm_sim_mode =
  | Exact : exact_sim prm_sim_mode
  | Approx : approx_sim prm_sim_mode
  | Fast : fast_sim prm_sim_mode

type _ sim_mode =
  | Ode : ode_sim sim_mode
  | Sde : sde_sim sim_mode
  | Prm : 'a prm_sim_mode -> 'a sim_mode

type ('pop, 'sim) settings =
  | Ode : (continuous_pop, unit) settings
  | Sde : (continuous_pop, Sim.Dbt.t) settings
  | Prm : 'a prm_sim_mode -> (discrete_pop, 'a)
```

Conclusion: OCaml for scientific programming

- Great language !

Conclusion: OCaml for scientific programming

- Great language !
  Rare difficult bugs mostly due to mutability or tricky algorithms.

Conclusion: OCaml for scientific programming

- Great language !
  Rare difficult bugs mostly due to mutability or tricky algorithms.
  Multiple refactorings have been done without too much pain.

Conclusion: OCaml for scientific programming

- Great language !
  Rare difficult bugs mostly due to mutability or tricky algorithms.
  Multiple refactorings have been done without too much pain.
- Limited ecosystem.

Conclusion: OCaml for scientific programming

- Great language !
  Rare difficult bugs mostly due to mutability or tricky algorithms.
  Multiple refactorings have been done without too much pain.
- Limited ecosystem.
  I used: Lacaml, odepack

Conclusion: OCaml for scientific programming

- Great language !
  Rare difficult bugs mostly due to mutability or tricky algorithms.
  Multiple refactorings have been done without too much pain.
- Limited ecosystem.
  I used: Lacaml, odepack
  There is also: owl, biocaml

Conclusion: OCaml for scientific programming

- Great language !
  Rare difficult bugs mostly due to mutability or tricky algorithms.
  Multiple refactorings have been done without too much pain.
- Limited ecosystem.
  I used: Lacaml, odepack
  There is also: owl, biocaml
- Limited scientific impact.

Notes de conclusion

Thank you !