Hamiltonian and Sequential Monte Carlo
An Ecosystem Example

Dominic Steinitz

27th September 2016

Outline

Outline

Outline

Outline

Outline

Some Application Areas

Suppose you have a model described by differential equations, e.g.

- Epidemiology: susceptible, infected, recovered (SIR)
- Pharmokinetics / pharmodynamics (PK / PD)
- Ecology: predator / prey

- How to fit parameters?
- How to capture second order effects?

Dynamical Systems

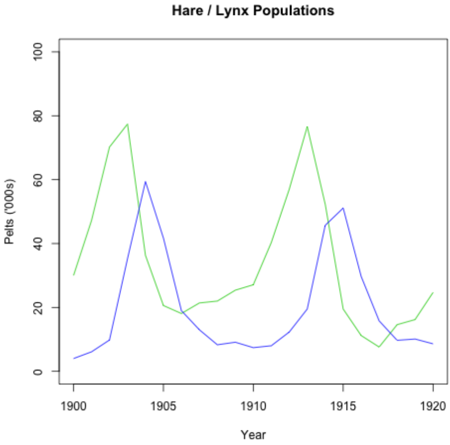
Let X be a set (usually taken to be a complete metric space), T a monoid and $S : T \times X \to X$ a family of operators, often written as $S_t(x) \triangleq S(t, x)$.
A dynamical system is a triple of such objects (X, S, T) such that the semi-group property holds.

$$S_0(x) = x$$
$$S_t(S_s(x)) = S_{t+s}(x)$$

S is called the evolutionary operator and sometimes the family $\{S_{t \in T}\}$ is called the flow of the dynamical system.

- Ordinary Differential Equation
- Delay Differential Equation
- Finite State Markov Chain
- Continuous State Markov Chain
- The Kalman Filter

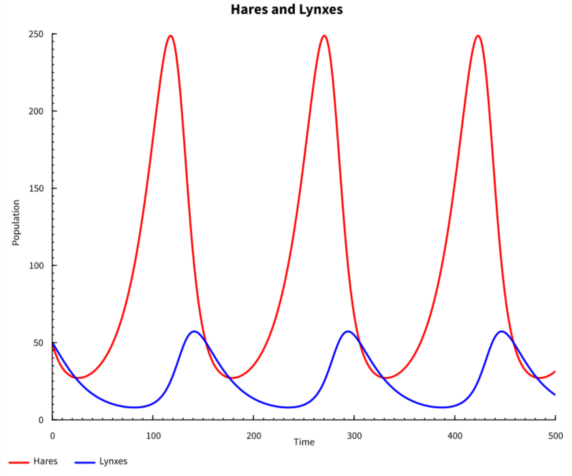
Hudson Bay Hares and Lynxes



Lotka and Volterra Original Model

$$
\begin{aligned}
\frac{\mathrm{d}N_1}{\mathrm{d}t} &= \rho_1 N_1 - c_1 N_1 N_2 \\
\frac{\mathrm{d}N_2}{\mathrm{d}t} &= c_2 N_1 N_2 - \rho_2 N2
\end{aligned}
$$

A Typical Path

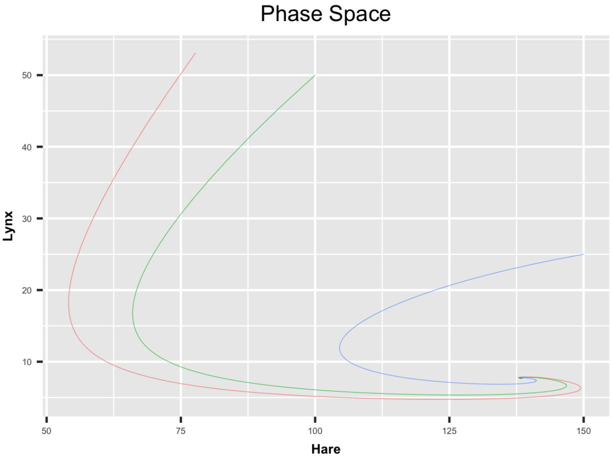


Structurally Unstable

- So it seems that this might be a good model.
- But can hare population really grow without a limit?
- Let us add carrying capacities.

$$
\begin{aligned}
\frac{\mathrm{d}N_1}{\mathrm{d}t} &= \rho_1 N_1 \left(1 - \frac{N_1}{K_1}\right) - c_1 N_1 N_2 \\
\frac{\mathrm{d}N_2}{\mathrm{d}t} &= -\rho_2 N_2 \left(1 + \frac{N_2}{K_2}\right) + c_2 N_1 N_2
\end{aligned}
$$

Typical Paths



Markov Transition Kernels

Recall that a **transition kernel** is a mapping $\mu : X \times \mathcal{Y} \to \overline{\mathbb{R}}_+$ where $(X, \mathcal{X})$ and $(Y, \mathcal{Y})$ are two measurable spaces such that $\mu(s, \cdot)$ is a probability measure on $\mathcal{Y}$ for all $s \in X$ and such that $\mu(\cdot, A)$ is a measurable function on X for all $A \in \mathcal{Y}$.
Recall further that a family of such transitions $\{\mu_t\}_{t \in T}$ where T is some index set satisfying

$$\mu_{t+s}(x, A) = \int_Y \mu_s(x, \mathrm{d}y)\mu_t(y, A) \tag{1}$$

gives rise to a Markov process.

Dynamical Systems as Markov Processes

A deterministic system can be formulated as a Markov process with a particularly simple transition kernel given by

$$\mu_t(x_s, A) = \delta(f_t(x_s), A) \triangleq \begin{cases} 1 & \text{if } f_t(x_s) \in A \\ 0 & \text{if } f_t(x_s) \notin A \end{cases} \tag{2}$$

where f_t is the deterministic state update function (the flow) and δ is the Dirac delta function.

Brownian Motion

- Now suppose determinstic system depends on some time-varying values.
- E.g. predator-prey model where the parameters cannot explain every aspect.
- $$\mu_t(\theta_s, \mathrm{d}\phi) = \mathcal{N}\left(\mathrm{d}\phi \,|\, \theta_s, \tau^2(t-s)\right) \tag{3}$$
- $\mathrm{d}\phi$ indicate probability densities.
- Parameter wiggles around like Brown's pollen particles rather than remaining absolutely fixed.

Ornstein-Uhlenbeck

- With Brownian motion, the parameters could drift off to $\pm\infty$.
- Parameters tend to stay close to some given value (mean-reverting).
- $$\mu_t(\theta_s, \mathrm{d}\phi) = \mathcal{N}\left(\mathrm{d}\phi \,|\, \alpha + (\theta_s - \alpha)e^{-\beta t}, \frac{\sigma^2}{2\beta}(1 - e^{-2\beta t})\right)$$
- β expresses how strongly parameter responds to perturbations.
- α is the mean to which the process wants to revert (aka the asymptotic mean).
- σ^2 expresses how noisy the process is.

Stochastic Differential Equations

It is sometimes easier to view these transition kernels in terms of stochastic differential equations. Brownian motion can be expressed as

$$\mathrm{d}X_t = \sigma \mathrm{d}W_t$$

and Ornstein-Uhlenbeck can be expressed as

$$\mathrm{d}X_t = -\beta(X_t - \alpha)\mathrm{d}t + \sigma \mathrm{d}W_t$$

where W_t is the Wiener process.

Capturing our Lack of Knowledge

Hare growth parameter

- Uncertainty grows over time
- Positive

$$
\begin{aligned}
\frac{\mathrm{d}N_1}{\mathrm{d}t} &= \rho_1 N_1\left(1 - \frac{N_1}{K_1}\right) - c_1 N_1 N_2 \\
\frac{\mathrm{d}N_2}{\mathrm{d}t} &= -\rho_2 N_2\left(1 + \frac{N_2}{K_2}\right) + c_2 N_1 N_2 \\
\mathrm{d}\rho_1 &= \rho_1 \sigma_{\rho_1} \mathrm{d}W_t
\end{aligned}
$$

W_t being a Wiener process.

Capturing our Lack of Knowledge II

By Itô we have

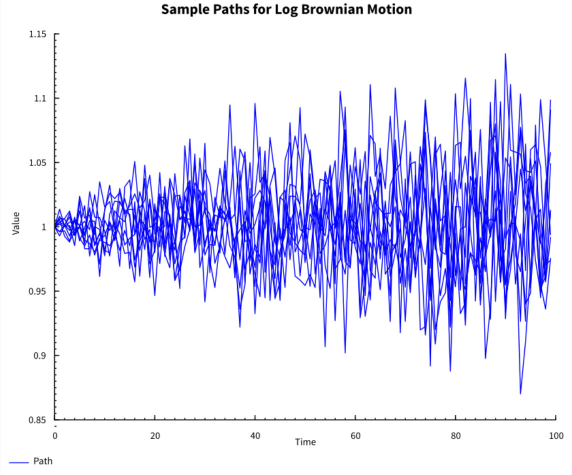
$$\mathrm{d}(\log \rho_1) = -\frac{\sigma_{\rho_1}^2}{2}\mathrm{d}t + \sigma_{\rho_1}\mathrm{d}W_t$$

We can use this to generate paths for ρ_1.

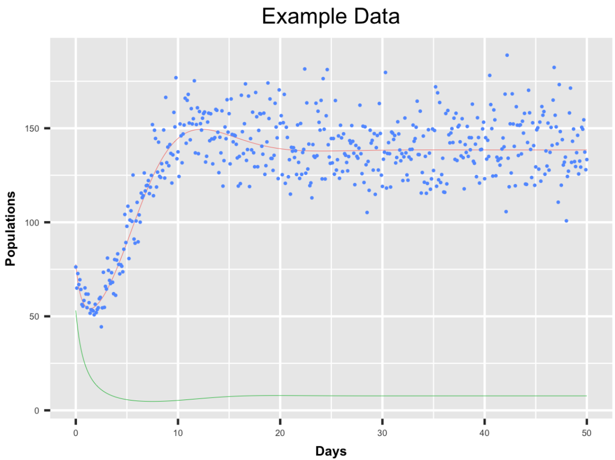
$$\rho_1(t+\Delta t) = \rho_1(t)\exp\left(-\frac{\sigma_{\rho_1}^2}{2}\Delta t + \sigma_{\rho_1}\sqrt{\Delta t}Z\right)$$

where $Z \sim \mathcal{N}(0,1)$.

Log Brownian Paths



A Typical Noisy System Path



LibBI

- LibBi http://libbi.org is used for state-space modelling and Bayesian inference on high-performance computer hardware, including multi-core CPUs, many-core GPUs (graphics processing units) and distributed-memory clusters.
- Default method: PMMH / PMCMC C. Andrieu, A.D. & R. Holenstein, Particle Markov chain Monte Carlo for Efficient Numerical Simulation

LibBI Constants

```
model PP {
  const h         = 0.1;     // time step
  const delta_abs = 1.0e-3; // absolute error tolerance
  const delta_rel = 1.0e-6; // relative error tolerance

  const a  = 5.0e-1 // Hare growth rate - superfluous
                    // for inference but a reminder
                    // of what we should expect
  const k1 = 2.0e2  // Hare carrying capacity
  const b  = 2.0e-2 // Hare death rate per lynx
  const d  = 4.0e-1 // Lynx death rate
  const k2 = 2.0e1  // Lynx carrying capacity
  const c  = 4.0e-3 // Lynx birth rate per hare
```

LibBI Parameters

```
state P, Z          // Hares and lynxes
state ln_alpha      // Hare growth rate (log of)
obs P_obs           // Observations of hares
param mu, sigma     // Mean and standard deviation of
                    // hare growth rate
noise w             // Noise
```

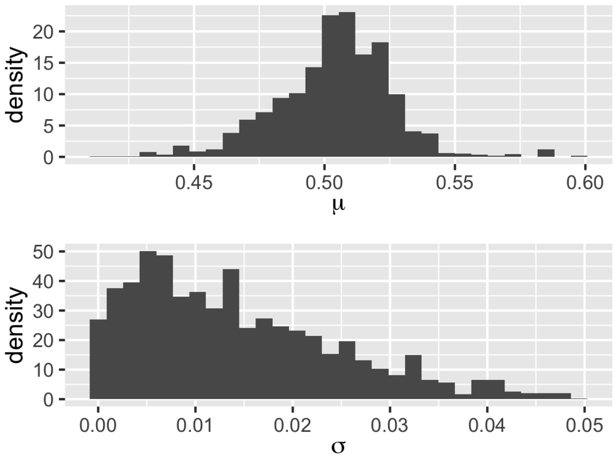
LibBI Initialisation

```
sub parameter {
  mu ~ uniform(0.0, 1.0)
  sigma ~ uniform(0.0, 0.5)
}

sub proposal_parameter {
   mu ~ truncated_gaussian(mu, 0.02, 0.0, 1.0);
   sigma ~ truncated_gaussian(sigma, 0.01, 0.0, 0.5);
 }

sub initial {
  P ~ log_normal(log(100.0), 0.2)
  Z ~ log_normal(log(50.0), 0.1)
  ln_alpha ~ gaussian(log(mu), sigma)
}
```

LibBI Model

```
  sub transition(delta = h) {
    w ~ normal(0.0, sqrt(h));
    ode(h = h, atoler = delta_abs, rtoler = delta_rel, alg =
        'RK4(3)') {
      dP/dt =  exp(ln_alpha) * P * (1 - P / k1) - b * P * Z
      dZ/dt = -d * Z * (1 + Z / k2) + c * P * Z
      dln_alpha/dt = -sigma * sigma / 2 - sigma * w / h
    }
  }

  sub observation {
    P_obs ~ log_normal(log(P), 0.1)
  }
}
```

Posteriors


density
20
15
10
5
0
0.45
0.50
0.55
0.60
μ
density
50
40
30
20
10
0
0.00
0.01
0.02
0.03
0.04
0.05
σ

Stan

Stan implements gradient-based Markov chain Monte Carlo (MCMC) algorithms for Bayesian inference, stochastic, gradient-based variational Bayesian methods for approximate Bayesian inference, and gradient-based optimization for penalized maximum likelihood estimation.

Stan implements reverse-mode automatic differentiation to calculate gradients of the model, which is required by HMC, NUTS, L-BFGS, BFGS, and variational inference. The automatic differentiation within Stan can be used outside of the probabilistic programming language.

Stan Constants

```
data {
  int<lower=1> T;    // Number of observations
  real y[T];         // Observed hares
  real k1;           // Hare carrying capacity
  real b;            // Hare death rate per lynx
  real d;            // Lynx death rate
  real k2;           // Lynx carrying capacity
  real c;            // Lynx birth rate per hare
  real deltaT;       // Time step
}

data=list(T  = length(rdata_PP$P_obs$value),
          y  = rdata_PP$P_obs$value,
          k1 = 2.0e2,
          b  = 2.0e-2,
          d  = 4.0e-1,
          k2 = 2.0e1,
          c  = 4.0e-3,
          deltaT = rdata_PP$P_obs$time[2] -
                   rdata_PP$P_obs$time[1]
          ),
```

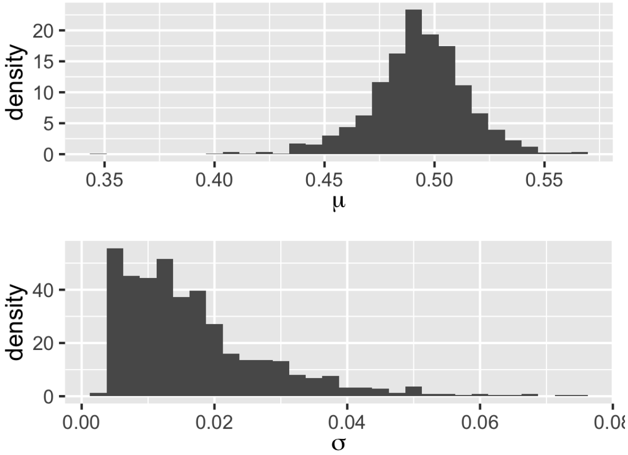
Stan Parameters

```
transformed parameters {
  real<lower=0> p[T]; real<lower=0> z[T]; real rho[T];
  p[1]    = p0; z[1]    = z0; rho[1] = rho0;
  for (t in 1:(T-1)){
    p[t+1] = p[t] +
             deltaT * f1 (exp(rho[t]), k1, b, p[t], z[t]);
    z[t+1] = z[t] + deltaT * f2 (d, k2, c, p[t], z[t]);
    rho[t+1] = rho[t] +
               sigma * w[t] - 0.5 * sigma * sigma * deltaT;
  }
}

functions {
  real f1 (real a, real k1, real b, real p, real z) {
    real q;
    q = a * p * (1 - p / k1) - b * p * z; return q;
  }
  real f2 (real d, real k2, real c, real p, real z) {
    real q;
    q = -d * z * (1 + z / k2) + c * p * z; return q;
  }
}
```

Stan Model

```
model {
  mu     ~ uniform(0.0, 1.0);
  sigma  ~ uniform(0.0, 0.5);
  p0     ~ lognormal(log(100.0), 0.2);
  z0     ~ lognormal(log(50.0), 0.1);
  rho0   ~ normal(log(mu), sigma);
  w      ~ normal(0.0,sqrt(deltaT));

  for (t in 1:T) {
    y[t] ~ lognormal(log(p[t]),0.1);
  }
}
```

Posteriors


density
20
15
10
5
0
0.35
0.40
0.45
0.50
0.55
μ
density
40
20
0
0.00
0.02
0.04
0.06
σ

Some Thoughts

- Designing programming languages especially probabilistic ones is hard.
- Stan and LibBI capture 30 years of advance in modelling but ignore 30 years of advance in programming languages.
- Both take time to compile and report errors.
- When things don't work, diagnosis is hard.
- Venture (née Church), Figaro, BLOG, JAGS, monad-bayes, PyMC(3)? and many more.