SDE Examples

AR(1) SDE model

In these stochastic differential equation models, the general notation is

$$\frac{dx(t)}{dt} = -\theta x(t) + \epsilon(t)$$

where $x(t)$ is the variable of interest (response), θ is the decay parameter, and $\epsilon(t)$ represents some white noise or forcing variable. In some papers, θ is expressed as $\tau = \frac{1}{\theta}$, where τ represents the the timescale over which fluctuations are dampened.

In many of the applications of this approach in the oceanography literature, θ or τ are fixed at known values. An additional consideration is the inclusion (or not) of observation or non-process errors; few applications explicitly include this term. Another consideration is the approximation used to solve the SDE. Some have used trapezoidal approaches (e.g. Cummins and Lagerloef 2002) but most have relied on Euler schemes (Di Lorenzo and Ohman 2014). This is also described in the Wikipedia implementation.

Existing approaches

A number of packages exist for solving these models in R (there's also a large range of other code out there, e.g. Matlab or Python). These include Sim.DiffProc or sde. The sde package is particularly useful because it includes maximum likelihood estimation see blog post here or here.

MLE example

We'll start with the MLE example, using the functions from sde. So that all of the examples run faster, we'll use just time series from 2006-2012.

```
d = read.csv("data/example slp sst data.csv",
  stringsAsFactors = FALSE)

# filter out NAs
d = dplyr::filter(d, !is.na(sst.anom), year>=2006)

# scale
d$sst.anom = scale(d$sst.anom)
d$slp.anom.lag2 = scale(d$slp.anom.lag2)
```

```
# simplified from https://renkun.me/2014/04/15/fit-an-ornstein-uhlenbeck-process-with-discrete-time-ser
ou.lik <- function(x) {
  function(theta2,theta3) {
    n <- length(x)
    dt <- deltat(x)
    -sum(dcOU(x=x[2:n], Dt=dt, x0=x[1:(n-1)],
      theta=c(0,theta2,theta3), log=TRUE))
  }
}
```

```
x=d$sst.anom
ou.fit <- mle(ou.lik(x),
  start=list(theta2=0.5,theta3=0.2),
  method="L-BFGS-B",lower=c(1e-5,1e-3), upper=c(1,1))
ou.coe <- coef(ou.fit)
names(ou.coe) = c("theta-mle","sigma-mle")
ou.coe
```

```
## theta-mle sigma-mle
## 0.4044537 0.8841380
```

And then we can get the states out as

```
# setOU() taken from SDE package
setOU <- function(Dt, x0, theta){
  Ex <- theta[1]/theta[2]+(x0-theta[1]/theta[2])*exp(-theta[2]*Dt)
  Vx <- theta[3]^2*(1-exp(-2*theta[2]*Dt))/(2*theta[2])
  return(list(Ex=Ex,Vx=Vx))
}

states =  setOU(Dt = rep(1,nrow(d)-1), x0=x[1:(nrow(d)-1)], theta = c(0, ou.coe))
states = data.frame(Ex = states$Ex, Vx = states$Vx)
states$low = states$Ex - 1.96*sqrt(states$Vx)
states$hi = states$Ex + 1.96*sqrt(states$Vx)
states$obs = d$sst.anom[-1]
states$time = seq(1,nrow(d)-1)

ggplot(states, aes(time,Ex)) +
  geom_ribbon(aes(ymin=low,ymax=hi), alpha=0.3,fill="blue") +
  geom_line(col="blue",size=0.3) +
  xlab("Time") + ylab("E[x]") +
  geom_point(aes(time,obs),col="red",alpha=0.3,size=1)
```

Just to largely confirm that the Bayesian analytical version is equivalent, we can run the model in Stan,

```
data_list = list(
  N = nrow(d),
  obs_y = c(d$sst.anom)
)

fit_1 = stan("code/ar1_analytical.stan",
  data = data_list,
  pars = c("sigma","pred_y","tau"),
  iter=5000,
  chains=1,
  control=list(adapt_delta=0.99, max_treedepth=20))
pars = rstan::extract(fit_1)
states$bayes_exact = apply(pars$pred_y,2,mean)[-1]
states$bayes_exact[1] = NA
```

And of course this gives perfectly correlated values,

As a second Bayesian model, we'll move away from the analytical solution and try to use the Euler approximation. This is considerably slower (the step size here is $1/M$, where $M = 2$ for speed).

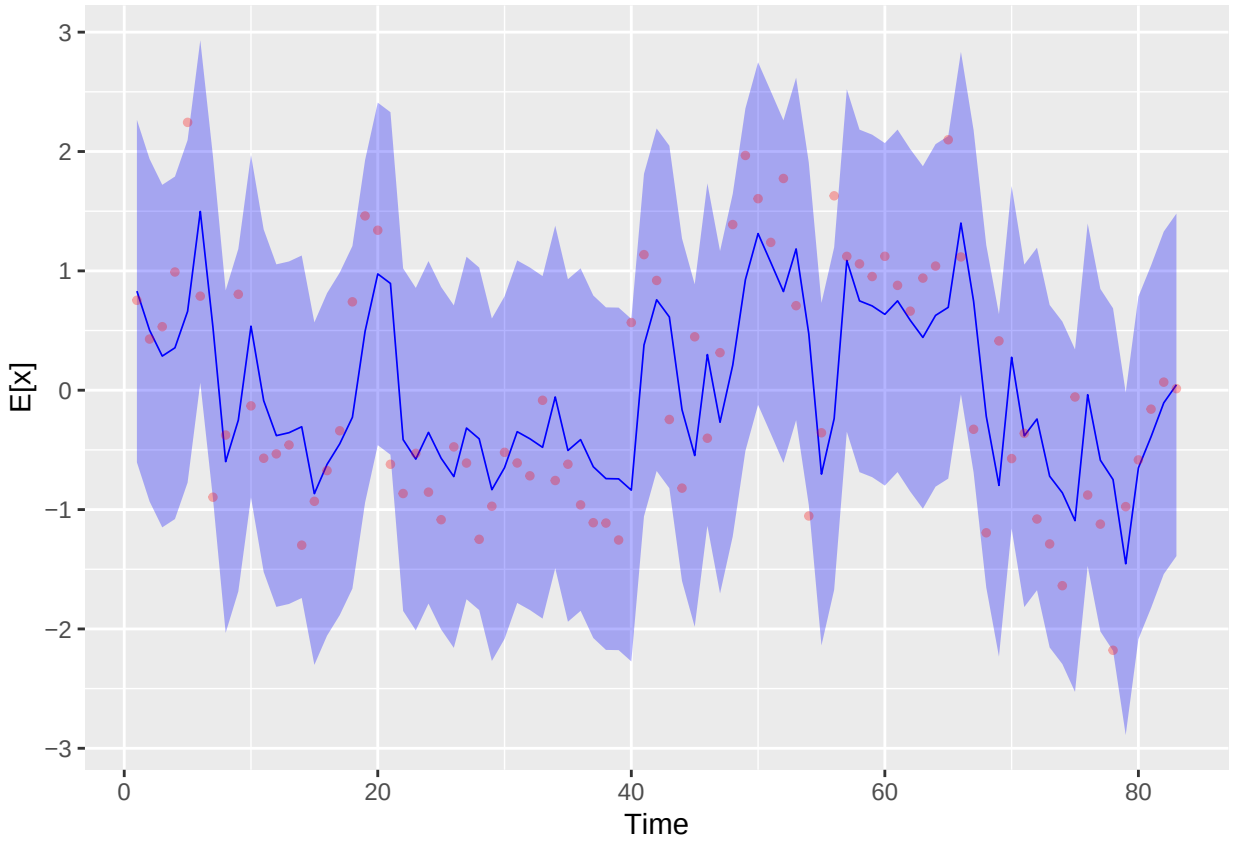


Figure 1: Maximum likelihood estimates of SDE fit to SST anomalies

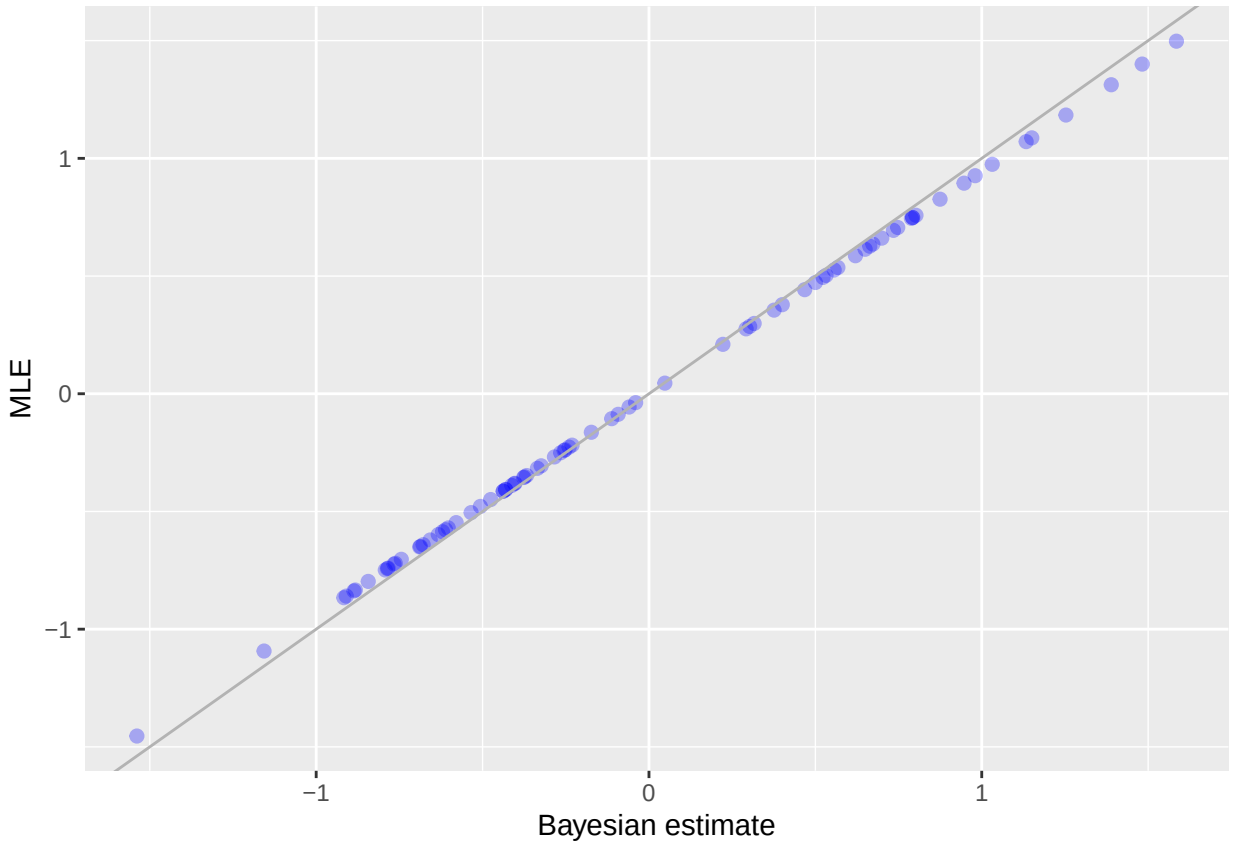


Figure 2: Bayesian analytical estimates vs MLE analytical estimates

```
data_list = list(
  N = nrow(d),
  M = 2, # resolution, should be higher than 10 (more like 100 or 1000)
  obs_y = c(d$sst.anom)
)

fit_2 = stan("code/ar1.stan",
  data = data_list,
  pars = c("sigma","pred_y","tau"),
  iter=5000,
  chains=1,
  control=list(adapt_delta=0.99, max_treedepth=20))
pars = rstan::extract(fit_2)
states$bayes_euler = apply(pars$pred_y,2,mean)[-1]
```

In some ways, this approach is too flexible – the euler approximation gives better estimates (closer to data) than the analytical solutions do.

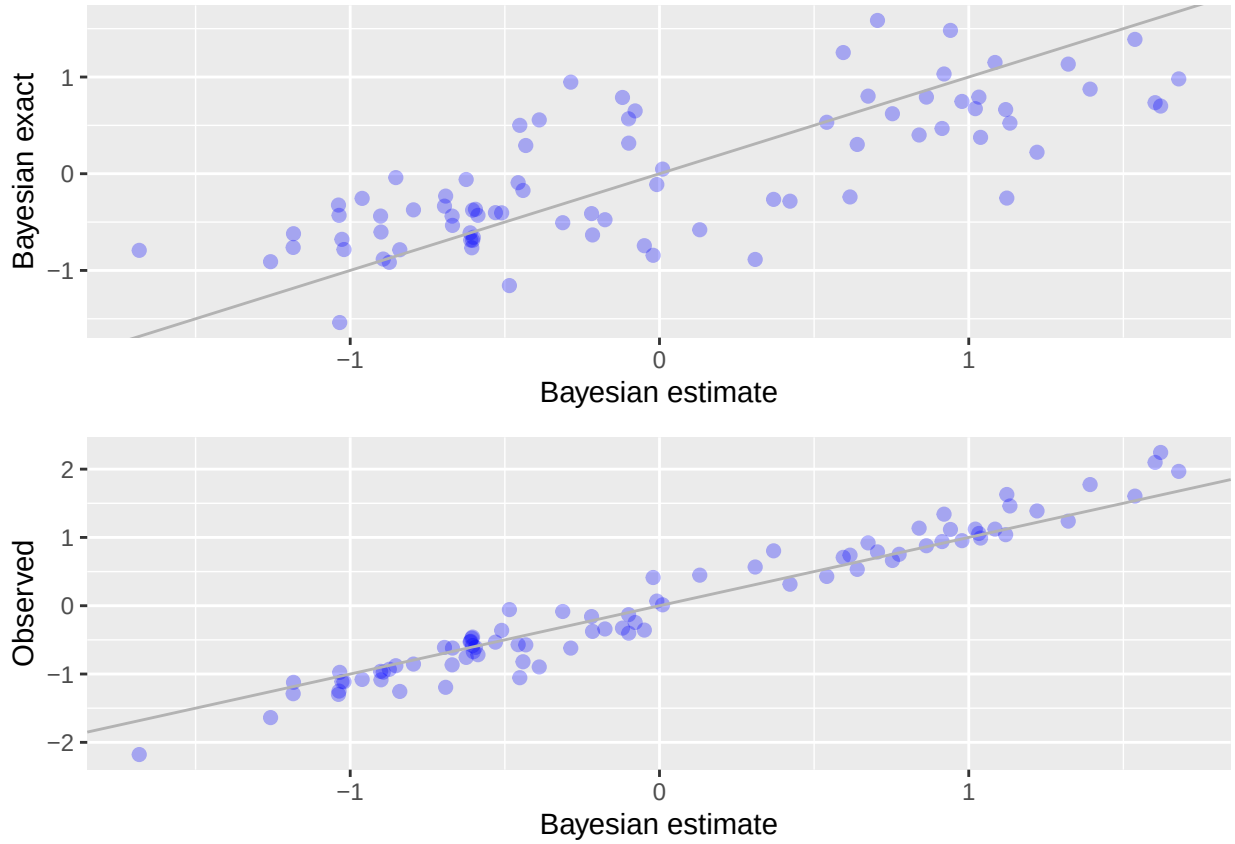


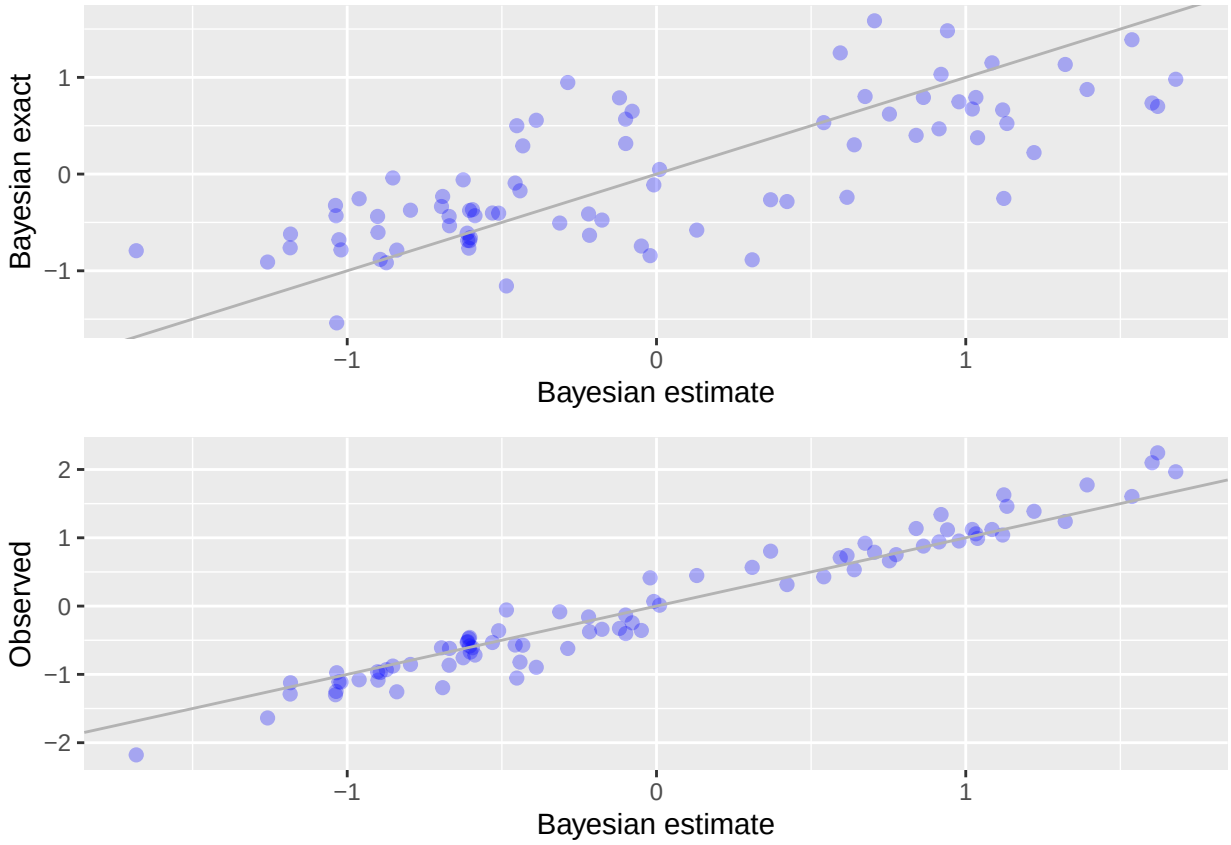
Figure 3: Bayesian analytical estimates vs Bayesian approximation

Finally, we can add the state-space component to the Euler implementation, where the data include a measurement / observation error.

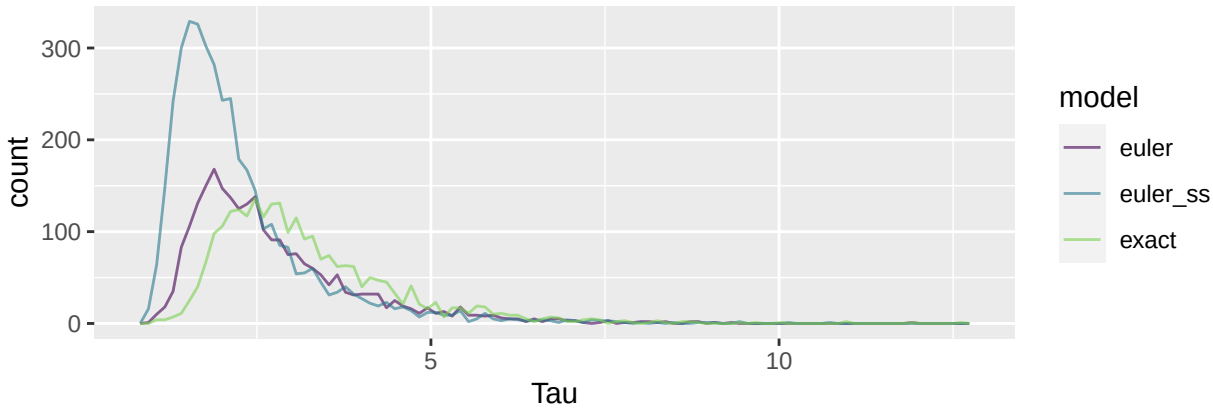
```
data_list = list(
  N = nrow(d),
  M = 2, # resolution, should be higher than 10 (more like 100 or 1000)
  obs_y = c(d$sst.anom)
```

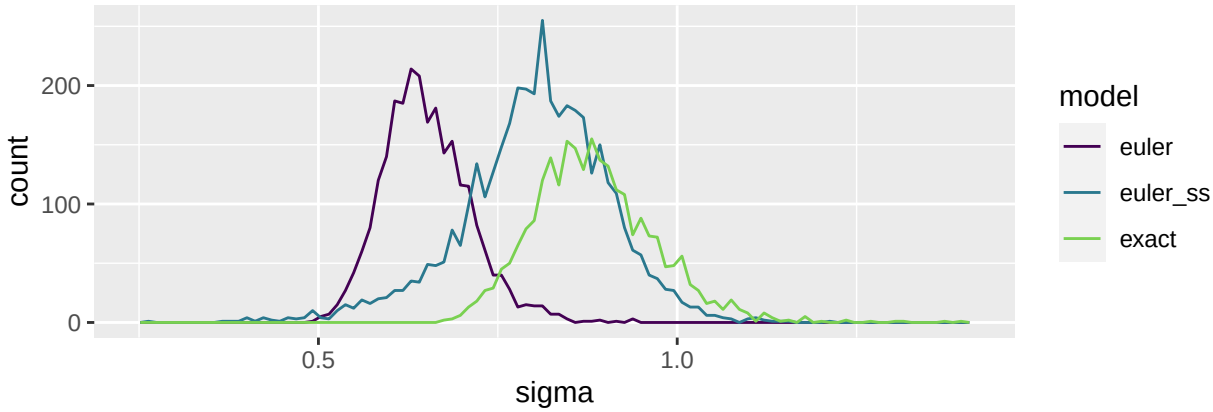
```
)

fit_3 = stan("code/ar1_ss.stan",
  data = data_list,
  pars = c("sigma","pred_y","tau","obs_sigma"),
  iter=8000,
  chains=1,
  control=list(adapt_delta=0.99, max_treedepth=20))
pars = rstan::extract(fit_3)
states$bayes_euler_ss = apply(pars$pred_y,2,mean)[-1]
```

Again, these approaches yield similar results – and near perfect fits to the data.



All three of the Bayesian approaches have similar estimates of tau and sigma,


300
200
100
0
count
5
10
Tau
model
euler
euler_ss
exact


200
100
0
count
0.5
1.0
sigma
model
euler
euler_ss
exact