SDE Examples

AR(1) SDE model (no covariates)

AR(1) SDE models are used widely in oceanography. Example applications include Cummins and Lagerloef (2002) estimating Ekman forcing on pyrocline depth variability, ecosystem response to climate forcing (Di Lorenzo 2010, Di Lorenzo & Ohman 2013). These models appear to generally ignore observation error, and fit parameters using least squares (e.g. Chhak et al. 2009). 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
```

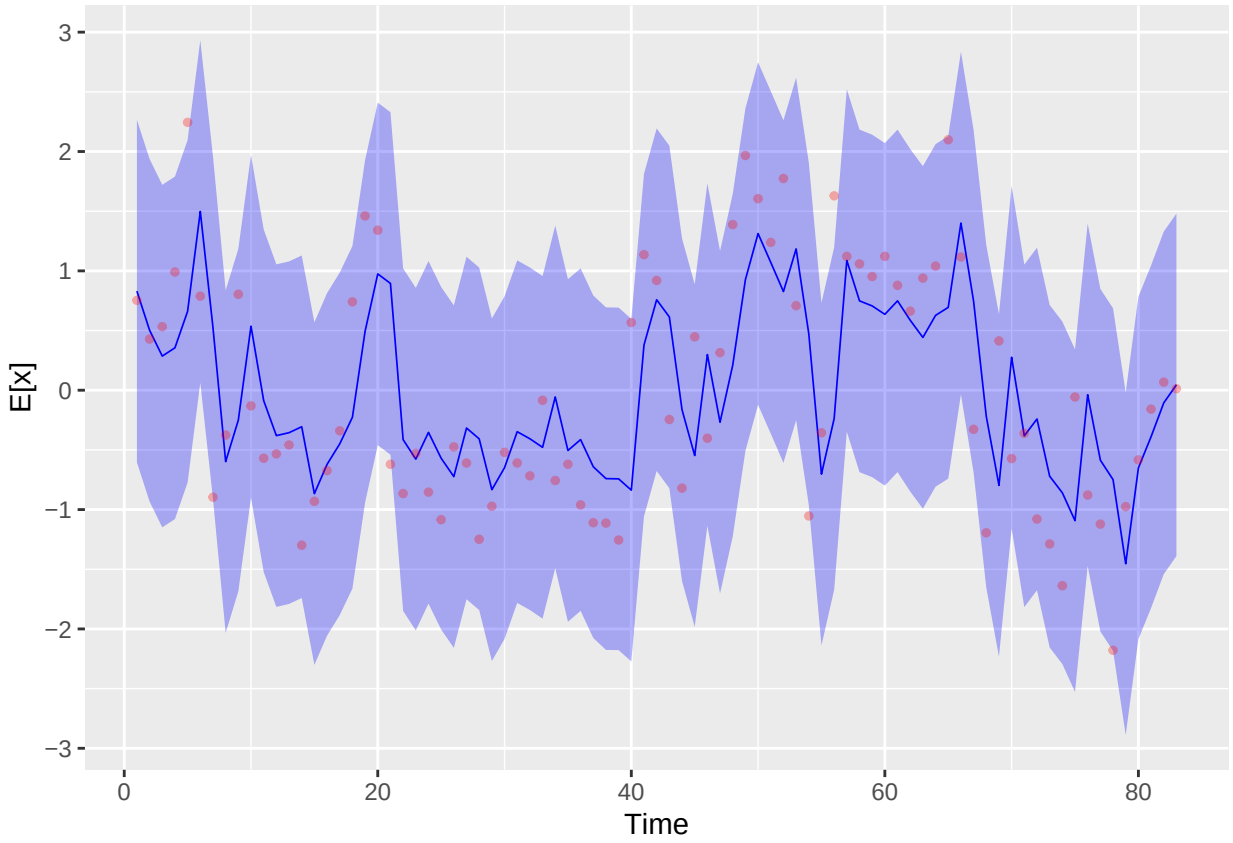


Figure 1: Maximum likelihood estimates of SDE fit to SST anomalies

And of course this gives perfectly correlated values,

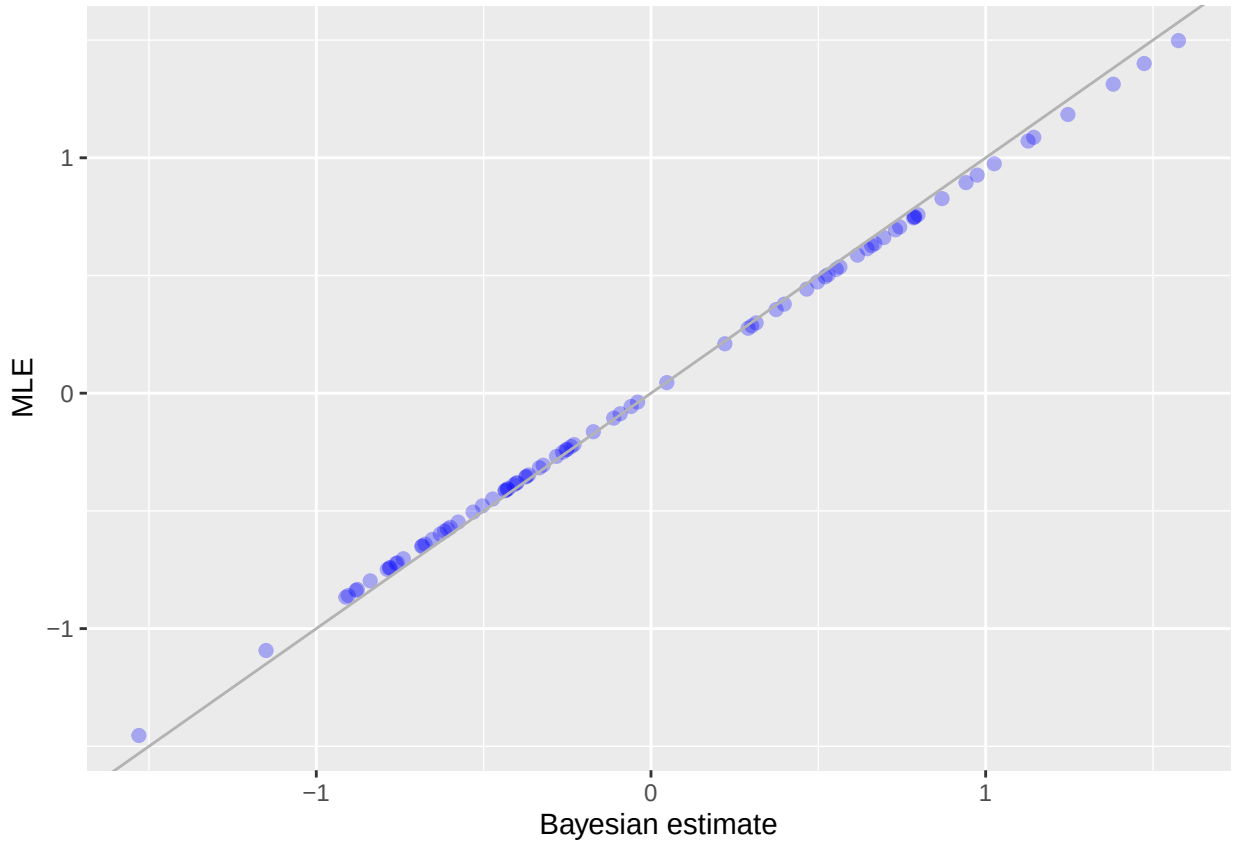


Figure 2: Bayesian analytical estimates vs MLE analytical estimates

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).

```
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.

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

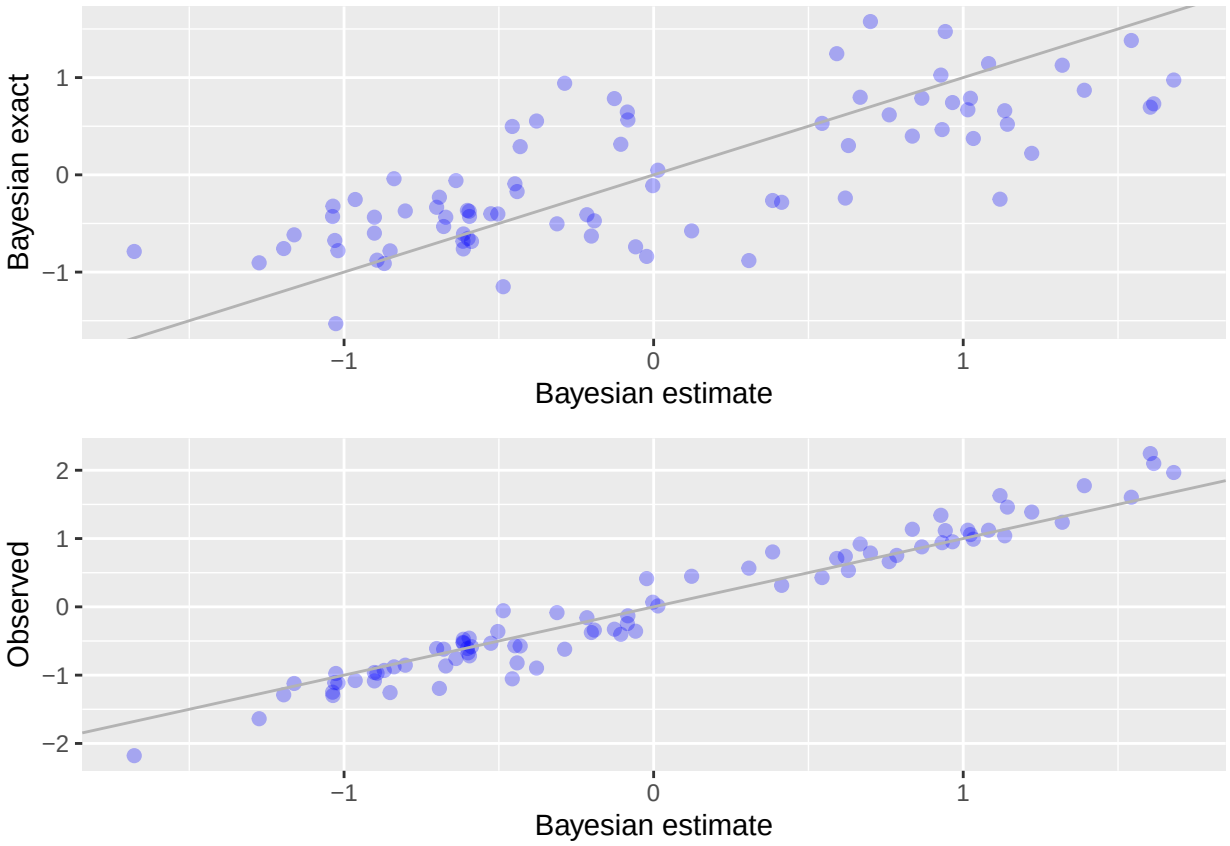
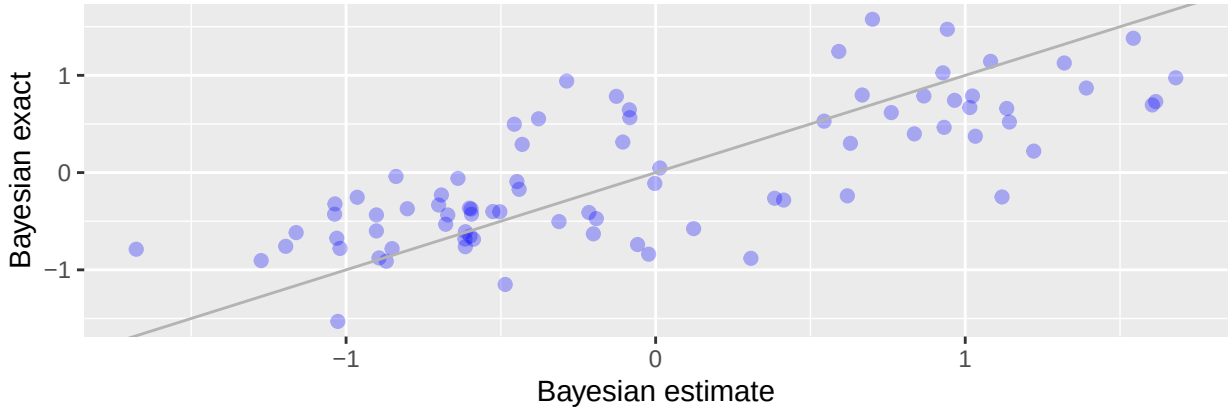
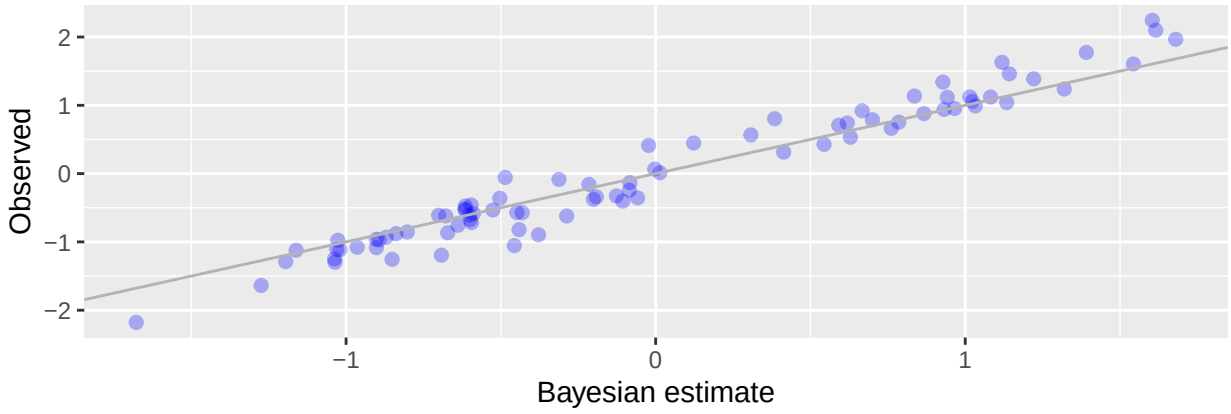


Figure 3: Bayesian analytical estimates vs Bayesian approximation

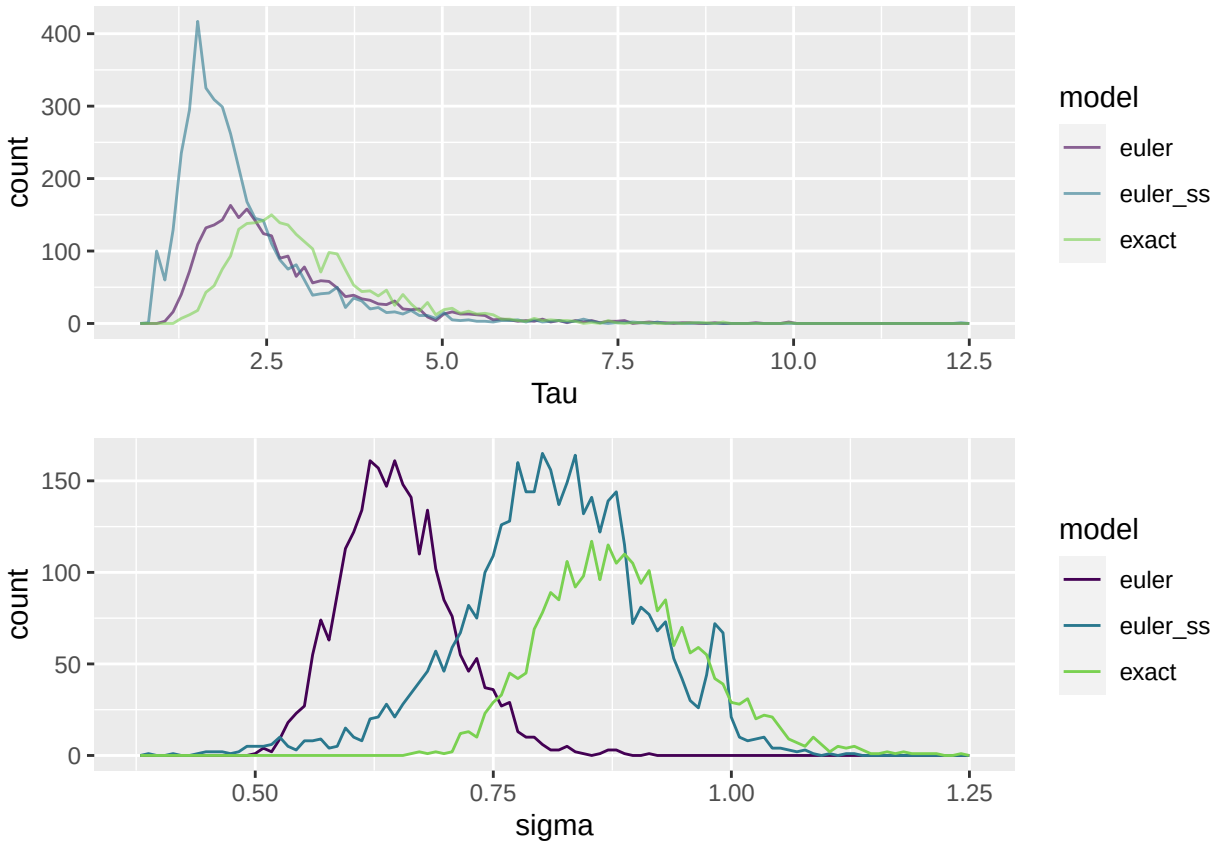
```
data_list = list(
  N = nrow(d),
  M = 2, # resolution, should be higher than 10 (more like 100 or 1000)
  obs_y = c(d$sst.anom)
)

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,



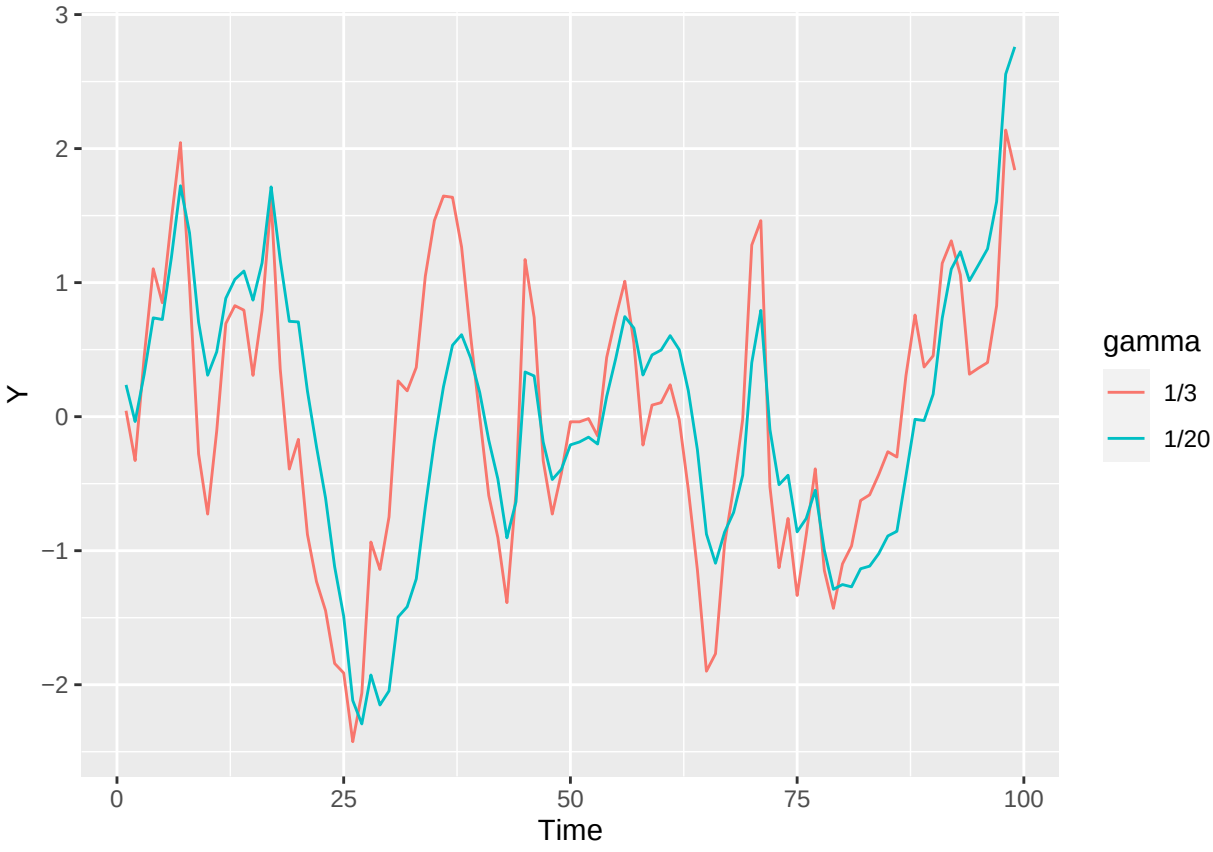
Red noise time series

Red noise time series can be created from white noise time series by including the AR component. This example uses the same terminology in oceanography models (DiLorenzo et al.), where f is a forcing variable (white noise), and γ is the '1/(decorrelation timescale)'.

```
ar_ls = function(time,forcing,gamma) {
  #S(t+1) = (1-GAMMA*DT)*S(t) + F(t)*DT
  forcing = c(forcing - mean(forcing))
  T=length(forcing)
  sig = 0

  for(t in 1:(T-1)) {
    #sig[t+1] = -theta*sig[t] + forcing[t]
    sig[t+1] = (1-gamma)*sig[t] + forcing[t]
  }

  # next estimates are linearly de-trended
  sig = sig - lm(sig ~ time)$fitted.values
  # interpolate output on the original time grid
  s.sig=(sig[-1]+sig[-T])/2 # midpoint
  # final step is normalize
  s.sig=s.sig/sd(s.sig)
  return(s.sig)
}
```

We can plot several red noise series with the same deviations, but different values of gamma,

```
set.seed(123)
x = rnorm(100)
df = rbind(data.frame("t"=1:99, "y"=ar_ls(1:100, x, 1/3),"gamma"="1/3"),
data.frame("t"=1:99, "y"=ar_ls(1:100, x, 1/20),"gamma"="1/20"))

ggplot(df, aes(t,y,group=gamma,col=gamma)) + geom_line() +
  xlab("Time") + ylab("Y")
```



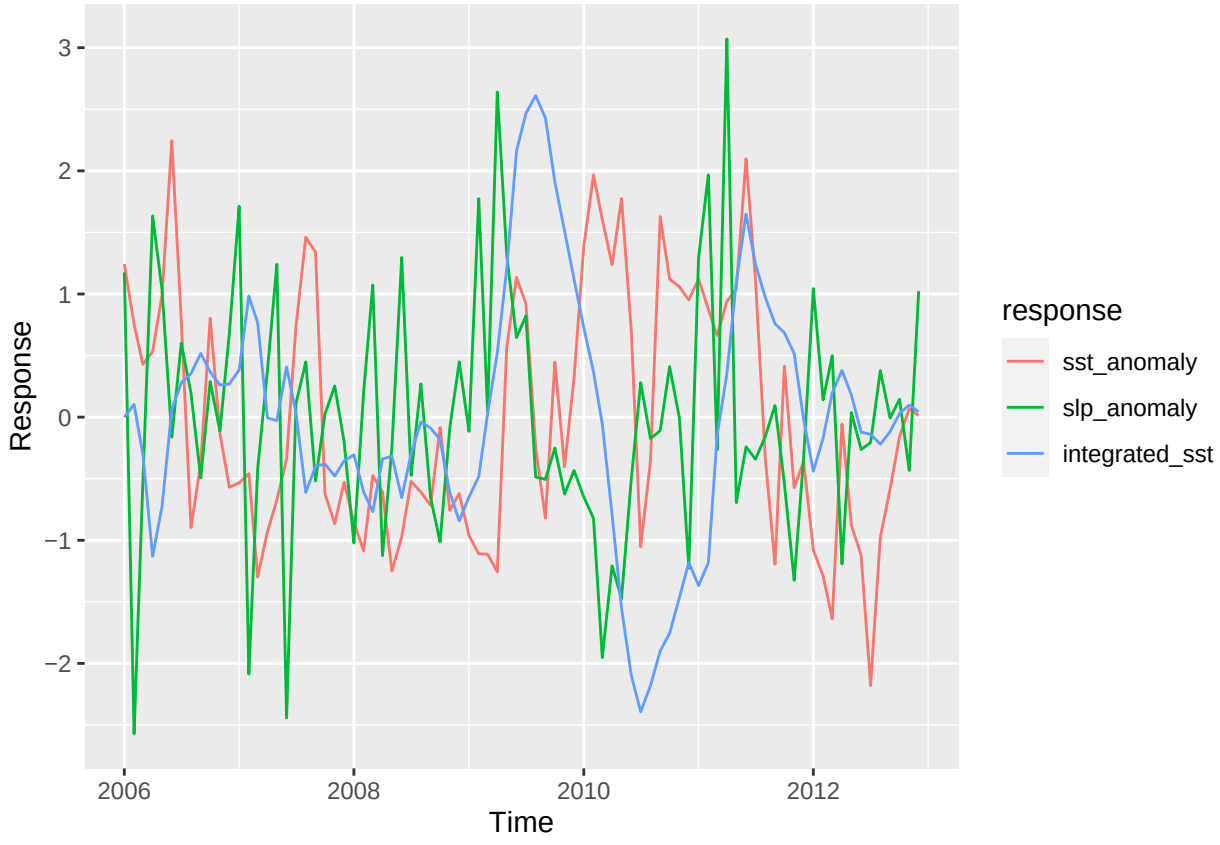
AR(1) SDE model (covariates)

In the above approaches, the white noise deviations were all estimated. In this extension, we'll treat them all as known (referred to in oceanography literature as forcing variables, e.g. Cummings and Lagerloef 2002).

In this first example, we'll use the same least squares approach that has been used previously (Chhak et al. 2009, Di Lorenzo et al. 2010). In addition to finding the solution with least squares, note that this approach isn't estimating the process variance (because of the standardization). This function is the basic AR-1 model and can be embedded in optim() to to optimization. Using our original example,

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

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

d$date = lubridate::parse_date_time(x = paste(d$year,d$month,"01"),orders="ymd",tz="Pacific")
```

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

calc_ss = function(theta) {
  pred_ts = ar_ls(1:nrow(d), forcing=d$slp.anom.lag2, theta)
  ss = -sum((pred_ts - d$sst.anom[-1])^2) # return -SS for optim
}

o = optimize(f=calc_ss, interval = c(0,1))

pred_ts = ar_ls(1:nrow(d), forcing=d$slp.anom.lag2, gamma = o$minimum)

df = rbind(data.frame("t"=d$date, "y"=d$sst.anom,"response"="sst_anomaly"),
  data.frame("t"=d$date, "y"=d$slp.anom.lag2,"response"="slp_anomaly"),
  data.frame("t"=d$date, "y"=c(0,as.numeric(pred_ts)),"response"="integrated_sst"))

ggplot(df, aes(t, y, group=response,col=response)) + geom_line() +
  xlab("Time") + ylab("Response")
```



Bayesian implementation

We can't do the exact Bayesian equivalent – because the approach above uses the sum of squares, rather than likelihood. But we can tweak the above approach to have an estimated observation error component.

```r
data_list = list(
  N = nrow(d),
  M = 1, # resolution, should be higher than 10 (more like 100 or 1000)
  obs_y = c(d$sst.anom),
  obs_x = c(d$slp.anom.lag2)
)

fit = stan("code/ar1_forcing_ss.stan",
  data = data_list,
  pars = c("sigma","pred_y","tau","obs_sigma"),
  iter=3000,
  chains=1,
  control=list(adapt_delta=0.99, max_treedepth=20))
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Users/eric.ward/Library/R/
## In file included from <built-in>:1:
## In file included from /Users/eric.ward/Library/R/3.6/library/StanHeaders/include/stan/math/prim/mat/
## In file included from /Users/eric.ward/Library/R/3.6/library/RcppEigen/include/Eigen/Dense:1:
## In file included from /Users/eric.ward/Library/R/3.6/library/RcppEigen/include/Eigen/Core:88:
## /Users/eric.ward/Library/R/3.6/library/RcppEigen/include/Eigen/src/Core/util/Macros.h:613:1: error: u
## namespace Eigen {
## ^
## /Users/eric.ward/Library/R/3.6/library/RcppEigen/include/Eigen/src/Core/util/Macros.h:613:16: error:
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Users/eric.ward/Library/R/3.6/library/StanHeaders/include/stan/math/prim/mat/
## In file included from /Users/eric.ward/Library/R/3.6/library/RcppEigen/include/Eigen/Dense:1:
## /Users/eric.ward/Library/R/3.6/library/RcppEigen/include/Eigen/Core:96:10: fatal error: 'complex' fil
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'ar1_forcing_ss' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 6.1e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.61 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 1: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 1: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 1: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 1: Iteration: 1200 / 3000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1500 / 3000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1501 / 3000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1800 / 3000 [ 60%]  (Sampling)
## Chain 1: Iteration: 2100 / 3000 [ 70%]  (Sampling)
## Chain 1: Iteration: 2400 / 3000 [ 80%]  (Sampling)
## Chain 1: Iteration: 2700 / 3000 [ 90%]  (Sampling)
```

```
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.179908 seconds (Warm-up)
## Chain 1:                0.173672 seconds (Sampling)
## Chain 1:                0.35358 seconds (Total)
## Chain 1:
```

```
pars = rstan::extract(fit)
```

We can plot the values versus the SS approach above and see the 2 methods agree. Note: in the SDE above, the process variance was estimated, but here it's fixed at 1 so the integrated SST will be on the same scale as the sum of squares.

```
df = rbind(data.frame("t"=d$date, "y"=c(0,as.numeric(pred_ts)),"method"="sum_squares"),
  data.frame("t"=d$date, "y"=apply(pars$pred_y,2,mean),"method"="bayesian")
  )

ggplot(df, aes(t, y, group=method,col=method)) + geom_line() +
  xlab("Time") + ylab("Integrated SST")
```

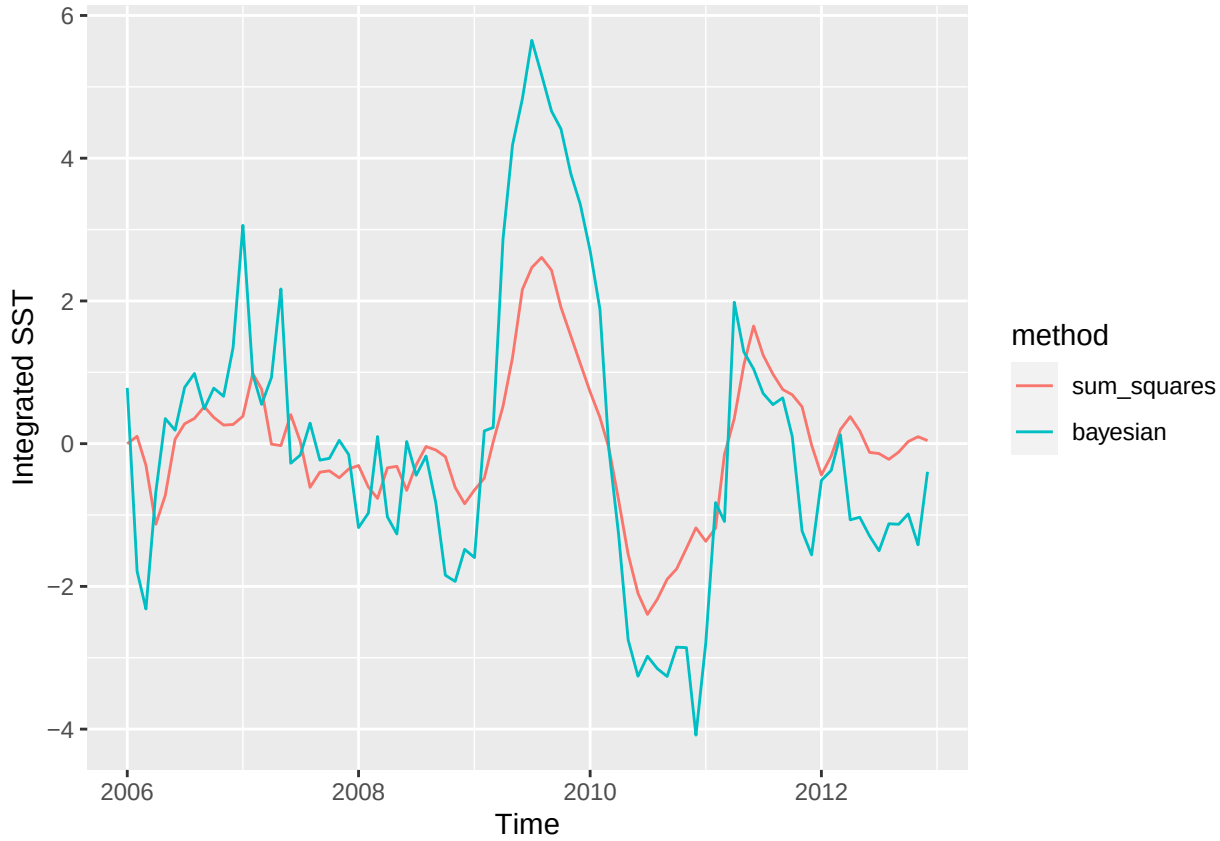


Figure 4: Bayesian vs SS estimates when forcing variable rather than white noise is included

Finally, we can put some credible intervals on our estimated integrated SST.

```
df = data.frame("t"=d$date,
  "y"=apply(pars$pred_y,2,mean),
  "low"=apply(pars$pred_y,2,quantile,0.025),
  "hi"=apply(pars$pred_y,2,quantile,0.975))
```

```
ggplot(df, aes(t, y)) +
  geom_ribbon(aes(ymin = low, ymax = hi),fill="darkblue",alpha=0.5) +
  geom_line(col="darkblue") +
  xlab("Time") + ylab("Integrated SST")
```