# Univariate time series examples - best practices

Eric Ward

2020-02-03

## Data

I'm putting together a simple example with simulated data. This is interesting because the observed response variable has a trend in increasing variance. The mechanism is a time-varying but stationary covariate, combined with a non-stationary coefficient between the covariate and response.

```
set.seed(123)
x = scale(cumsum(rnorm(50)))
b = cumsum(rnorm(n=length(x),0,0.02))
y = rnorm(3+x*b, 0.1)
dat = data.frame(time=seq(1,length(x)),x=x,y=y,b=b)

g1 = ggplot2::ggplot(dat,aes(time,x)) + geom_point() +
  xlab("Time") + ylab("x covariate")
g2 = ggplot2::ggplot(dat,aes(time,b)) + geom_point() +
  xlab("Time") + ylab("b coefficient")
g3 = ggplot2::ggplot(dat,aes(time,y)) + geom_point() +
  xlab("Time") + ylab("Data (y)")
gridExtra::grid.arrange(g1,g2,g3,nrow=3)
```

### Linear regression and time series models

```
# start with linear regression, no breakpoint in year 30
fit <- brm(y ~ x * b,data=dat)
dat$breaks = c(rep("pre",30),rep("post",20))
dat$pred = predict(fit,newdata=dat)[,"Estimate"]
dat$model = "lm"
results = dat

# next linear regression, fixed breakpoint in year 30
fit = brm(y ~ x * b + b*breaks,data=dat)
dat$pred = predict(fit,newdata=dat)[,"Estimate"]
dat$model = "lm - fixed breakpoint"
results = rbind(results,dat)

# add linear time series model, with AR(1) and MA(1) components
fit = arima(dat$y, xreg=dat$x, order = c(1,0,1))
dat$pred = fitted(fit)
dat$model = "ARIMA (1,0,1)"
results = rbind(results,dat)
```

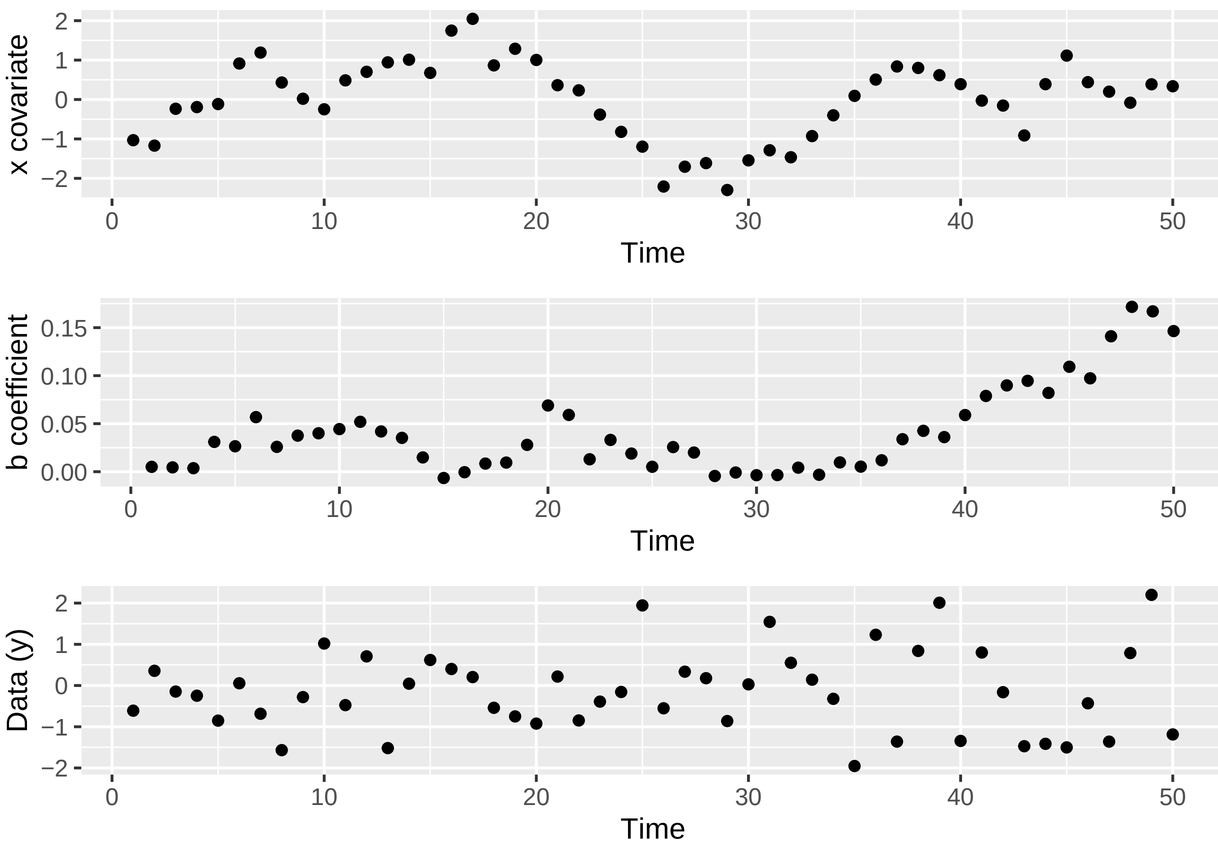


Figure 1: Simulated data and covariate. The predicted data at each time step is a linear function of the covariate, y(t) = b0 + b1(t) * x(t)

### DLM models

Dynamic linear models are state space models, and can have random walks in intercept, slope or both. Either are considered process variation.

```
# Random walk on slope here:
fit = fit_stan(y = dat$y, x = dat$x, model_name="dlm-slope")
dat$pred = apply(rstan::extract(fit)$pred,2,mean)
dat$model = "dlm-slope"
results = rbind(results,dat)

# time varying slope, constant intercept
fit = fit_stan(y = dat$y, x = dat$x, model_name="dlm-intercept")
dat$pred = apply(rstan::extract(fit)$pred,2,mean)
dat$model = "dlm-intercept"
results = rbind(results,dat)

# time varying slope and intercept. some issues with identifiability
fit = fit_stan(y = dat$y, x = model.matrix(lm(dat$y~dat$x)), model_name="dlm")
dat$pred = apply(rstan::extract(fit)$pred,2,mean)
dat$model = "dlm-both"
results = rbind(results,dat)
```

### Genaralized Additive Models

```
# GAM - simple
fit = gam(y ~ s(x), data=dat)
dat$pred = fit$fitted.values
dat$model = "gam s(x)"
results = rbind(results,dat)

# GAM, AR errors
fit = gamm(y ~ s(x),correlation = corARMA(p=1), data=dat)
dat$pred = fit$gam$fitted.values
dat$model = "gam s(x) & MA"
results = rbind(results,dat)

# GAM, interaction between x and time
fit = gam(y ~ s(x) + s(time) + ti(x,time), data=dat)
dat$pred = fit$fitted.values
dat$model = "gam te(x,time)"
results = rbind(results,dat)

# GAM with time-varying coefficient
fit = gam(y ~ s(time,by=x), data=dat)
dat$pred = fit$fitted.values
dat$model = "gam time varying"
results = rbind(results,dat)
```

**Hidden Markov Models (HMMs)**

Hidden Markov Models or discrete state switching models have an underlying latent state that each time point can be assigned to, and this state evolves as a Markov process with transition matrix Gamma (m x m). Though we could have > 2 states in the model, estimation becomes more difficult, so we'll restrict this to the 2-state model for now. Each state in the model might have a different mean or variance. Two examples are below, (1) each state is allowed to have a separate regression coefficient, and (2) each state is allowed to have separate regression coefficients and residual errors.

```
# Hidden markov model / 2 regime switching for b 1 coefficient
jagsscript = cat("
model {
  # priors for intercept
  B0 ~ dnorm(0,1);
  # priors for regression coefficient
  for(i in 1:2) {
    B1[i] ~ dnorm(0,1);
  }
  # prior for obs Error
  obsTau ~ dgamma(0.001,0.001);
  obsSigma <- 1/sqrt(obsTau);

  # markov switching
  alpha[1] <- 1;
  alpha[2] <- 1;
  p[1:2] ~ ddirch(alpha[1:2]);
  Gamma[1,1:2] ~ ddirch(alpha[1:2]);
  Gamma[2,1:2] ~ ddirch(alpha[1:2]);
  z[1] ~ dcat(p[1:2]);
  for(n in 2:nT) {
    z[n] ~ dcat(Gamma[z[n-1],]);
  }

  # evaluate the likelihood
  for(n in 1:nT) {
  pred[n] <- B0 + B1[z[n]]*x[n];
  y[n] ~ dnorm(pred[n],obsTau);
  }
  }  ",file="jags_switching.txt")

jags.data = list("y"=dat$y,"x"=dat$x,"nT"=nrow(dat))
jags.params=c("B0","B1","pred","z")
model.loc=("jags_switching.txt")

jags.model = jags(jags.data, inits = NULL,
  parameters.to.save= jags.params,
  model.file=model.loc,
  n.chains = 3,
  n.burnin = 20000,
  n.thin = 1,
  n.iter = 30000)
attach.jags(jags.model, overwrite=TRUE)
dat$pred = apply(pred,2,mean)
dat$model = "HMM - b"
```

```r
results = rbind(results,dat)

# Hidden markov model / 2 regime switching for b 1 coefficient
jagsscript = cat("
  model {
  # priors for intercept
  B0 ~ dnorm(0,1);
  # priors for regression coefficient
  for(i in 1:2) {
  B1[i] ~ dnorm(0,1);
  }
  # prior for obs Error
  obsTau[1] ~ dgamma(0.001,0.001);
  obsSigma[1] <- 1/sqrt(obsTau[1]);
  obsTau[2] ~ dgamma(0.001,0.001);
  obsSigma[2] <- 1/sqrt(obsTau[2]);

  # markov switching
  alpha[1] <- 1;
  alpha[2] <- 1;
  p[1:2] ~ ddirch(alpha[1:2]);
  Gamma[1,1:2] ~ ddirch(alpha[1:2]);
  Gamma[2,1:2] ~ ddirch(alpha[1:2]);
  z[1] ~ dcat(p[1:2]);
  for(n in 2:nT) {
  z[n] ~ dcat(Gamma[z[n-1],]);
  }

  # evaluate the likelihood
  for(n in 1:nT) {
  pred[n] <- B0 + B1[z[n]]*x[n];
  y[n] ~ dnorm(pred[n],obsTau[z[n]]);
  }
  }  ",file="jags_switching_var.txt")

jags.data = list("y"=dat$y,"x"=dat$x,"nT"=nrow(dat))
jags.params=c("B0","B1","pred","z")
model.loc=("jags_switching_var.txt")

jags.model = jags(jags.data, inits = NULL,
  parameters.to.save= jags.params,
  model.file=model.loc,
  n.chains = 3,
  n.burnin = 20000,
  n.thin = 1,
  n.iter = 30000)
attach.jags(jags.model, overwrite=TRUE)
dat$pred = apply(pred,2,mean)
dat$model = "HMM - b & var"
results = rbind(results,dat)
```

### Plotting Results

```
# reorder levels to be ordered in the order they were fit
results$model = factor(results$model,
  levels = unique(results$model))
```

```
# plot residuals across modeling approaches
ggplot(results, aes(time,y-pred)) + geom_point() +
  facet_wrap(~model) + xlab("Time") + ylab("Y - E[Y]")
```

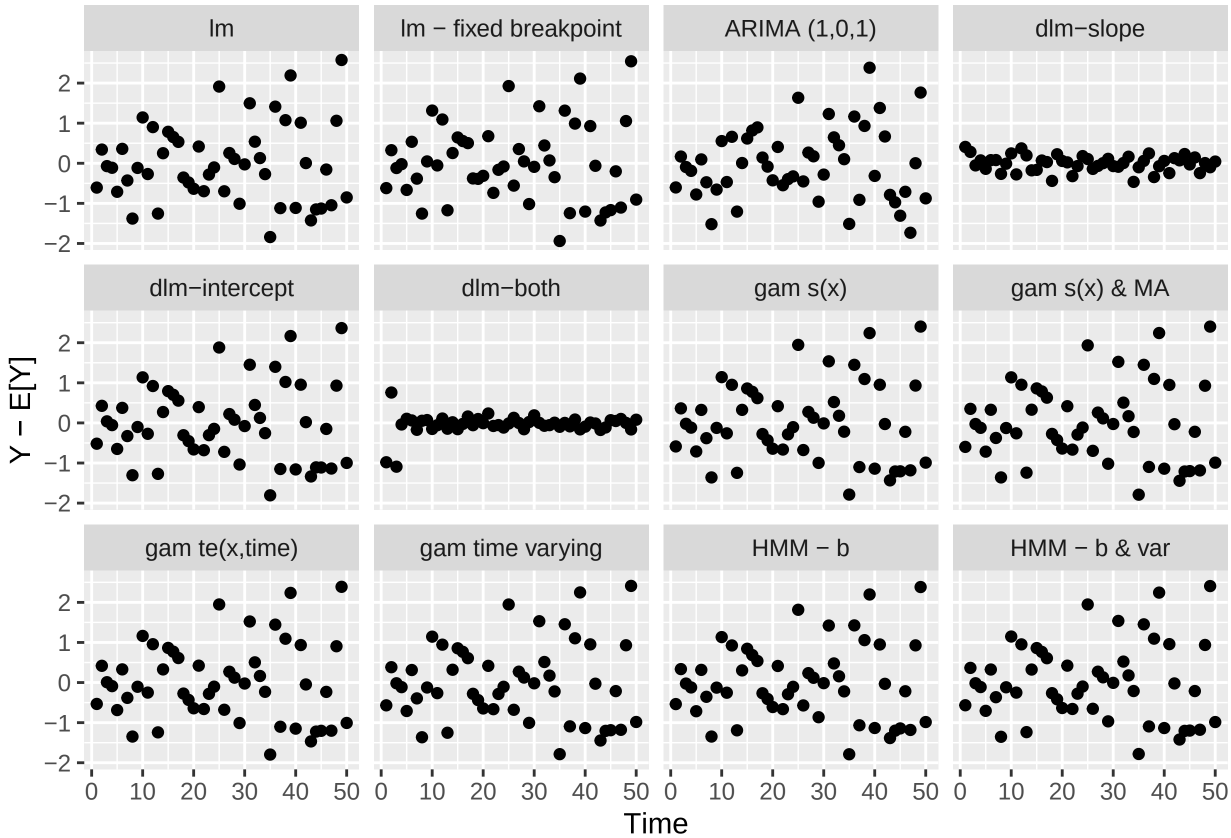


Figure 2: Residuals over time across models

```
# plot x versus residuals across modeling approaches
ggplot(results, aes(x,y-pred)) + geom_point() +
  facet_wrap(~model) + xlab("Covariate (x)") + ylab("Y - E[Y]")
```

```
# plot x versus residuals across modeling approaches
ggplot(results, aes(pred,y)) + geom_point() +
  facet_wrap(~model) + xlab("Predicted") + ylab("Observed")
```

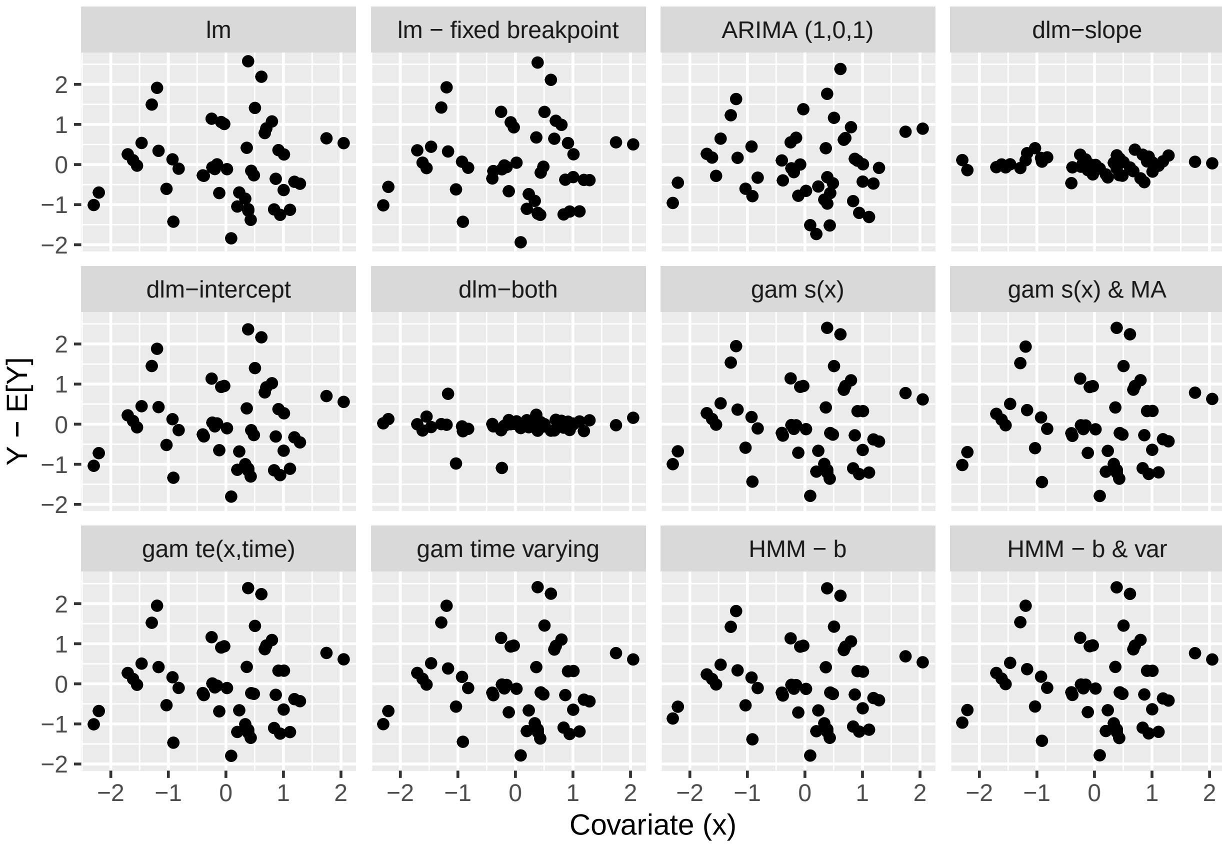


Figure 3: Residuals versus covariate value across models

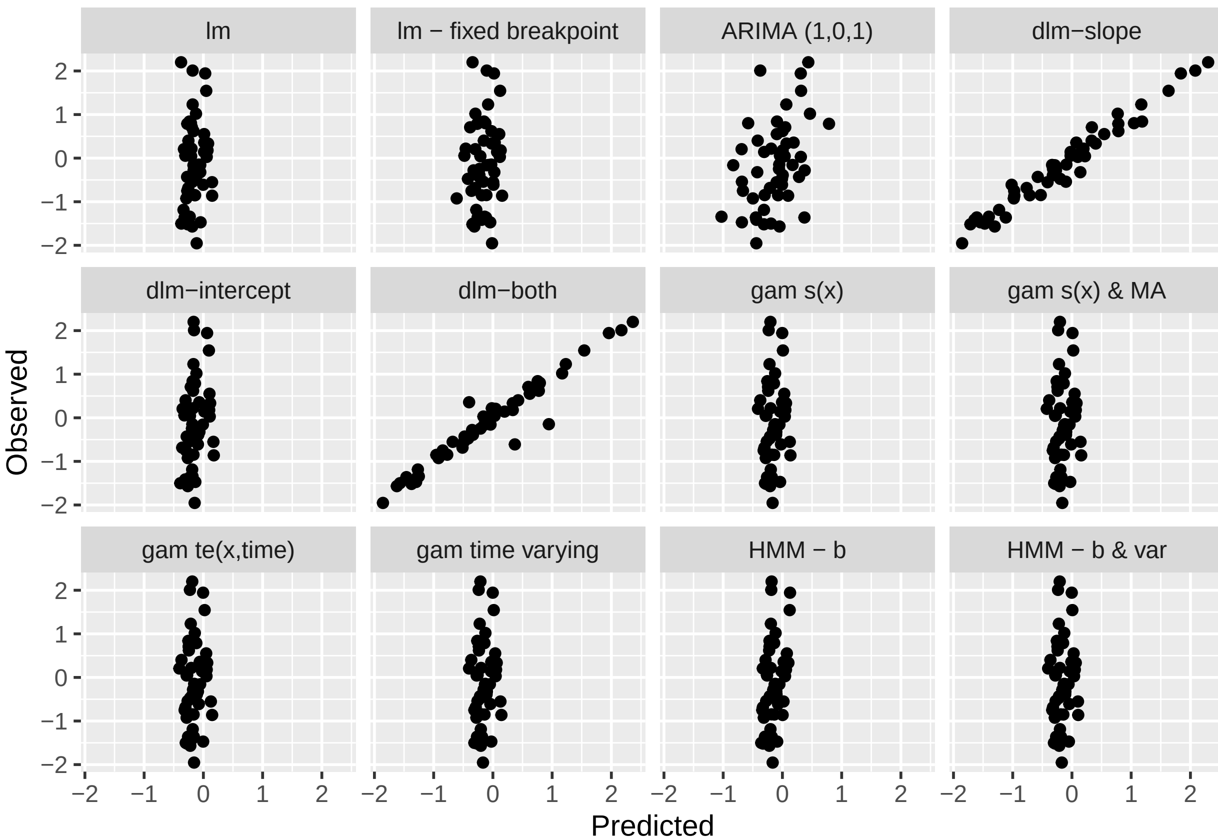


Figure 4: Predicted vs observed across models