

# Fitting Bayesian time series models

FISH 507 – Applied Time Series Analysis

Eric Ward

# Overview of today's material

- ▶ Quick overview of Stan
- ▶ Manipulating and plotting Stan output
- ▶ Examples of time series models

# Review of models we've used so far

## Models

- ▶ Regression
- ▶ ARMA models
- ▶ State Space Models
- ▶ Dynamic Factor Analysis
- ▶ Dynamic Linear Models
- ▶ MARSS models (multivariate time series models)

# Why Bayesian?

- ▶ Complex hierarchical models
  - ▶ Non-linear models
  - ▶ Hierarchical or shared parameters
  - ▶ Non-normal data
- ▶ Inference: what's the probability that the data are less than some threshold?
- ▶ No bootstrapping!
  - ▶ We get credible intervals for parameters and states simultaneously

# Using STAN

## What is Stan?

- ▶ Powerful, cross-platform and cross-language (R, Julia, Matlab, etc) that allows users to write custom code that can be called directly from R
- ▶ Estimation can be fully or approximate Bayesian inference, or maximum likelihood optimization (BFGS)
- ▶ Useful links:
  - ▶ Stan homepage
  - ▶ Stan manual
  - ▶ rstan

## Options for using Stan in this class

- ▶ Write your own code (based on examples in the manual, etc)
- ▶ Use an existing package
- ▶ Use our bundled code to get started with simple models (we'll start here)

## Existing packages: rstanarm and bmrs

- ▶ Both packages very flexible, and allow same syntax as basic lm/glm or lmer models, e.g.

```
rstan::stan_lm  
rstan::stan_glm  
rstan::stan_glmer
```

- ▶ Vignettes brms rstanarm

## Potential limitations

brms includes autocorrelated errors, non-normal data, non-linear smooths, etc.

- ▶ But doesn't allow for AR processes on parameters / latent variables
- ▶ Also doesn't include multivariate data



# Advantages

brms offers notation that should be very familiar to run many classes of models,

```
brms::brm(y ~ x * z + (1|group), data=d)
brms::brm(y01 ~ x * z + (1|group), data=d, family = binomial)
brms::brm(bf(y ~ s(x)), data=d)
```

## To install code for this class

- ▶ We'll need to install these packages to run Stan,

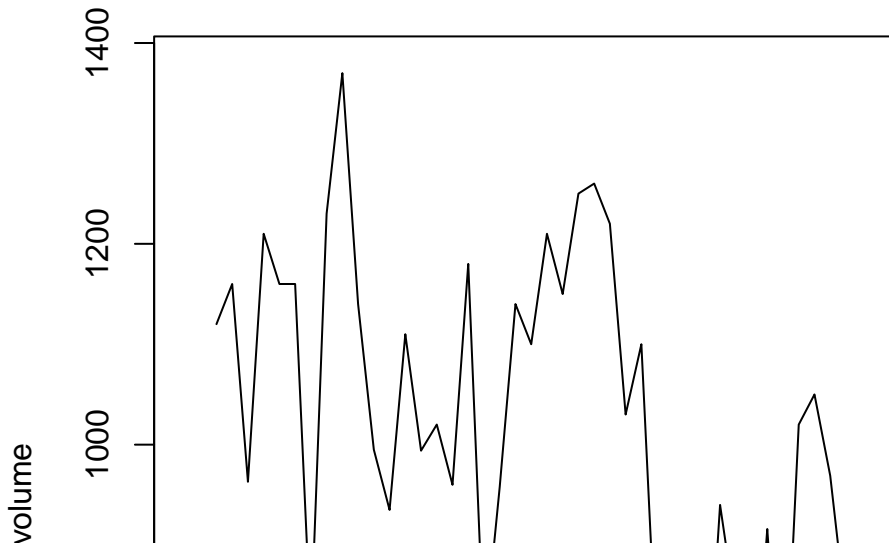
```
install.packages("rstan", repos = "https://cloud.r-project.org")  
install.packages("devtools", repos = "https://cloud.r-project.org")
```

- ▶ And then we can install our custom package for the class with bundled Stan time series models

```
devtools::install_github(repo="nwfsc-timeseries/atsar")  
install.packages("atsar")
```

## Working with Stan output

- ▶ We'll start with a simple example, using a classic dataset of flow on the Nile River



## Working with Stan output

Fitting a regression model to this data doesn't make a lot of sense, but it will introduce us to the basic functionality of our wrapper functions. We'll start with a function named `fit_stan`

```
library(atsar)
lm_intercept = fit_stan(y = as.numeric(Nile),
                        x = rep(1, length(Nile)),
                        model_name = "regression")
```

## Working with Stan output

The output of the fitted model can be examined a number of ways. Starting with the simple summaries,

```
lm_intercept
```

This model ran 3 MCMC chains (the default) with warmups of 500, followed by 500 more iterations. These latter were stored (so 1500 parameter samples total)

## Working with Stan output

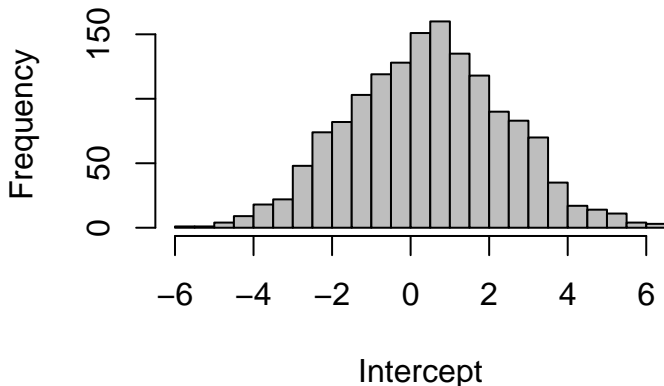
But we're probably more interested in the values for individual parameters. We can pull these out in a few different ways, and plot them with either base graphics or ggplot. As a first option for getting values we can manipulate, we can use the `extract` function,

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

## Plotting with Stan output

Then we can do all kinds of things with this output, like making a histogram

```
hist(pars$beta, 40, col="grey", xlab="Intercept", main="")
```



or calculate summary statistics

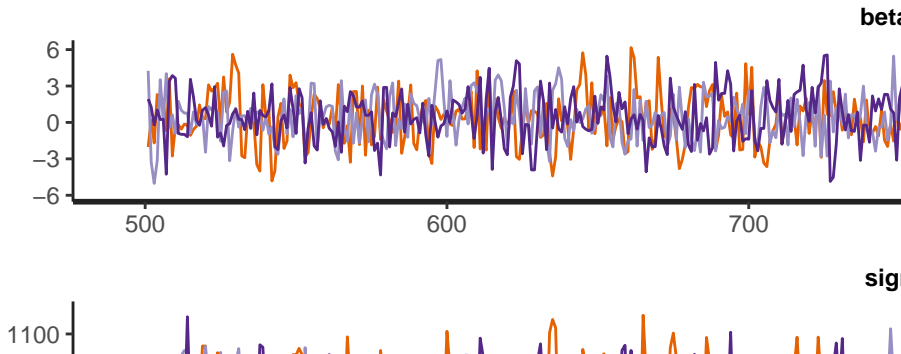
```
quantile(pars$beta, c(0.025,0.5,0.975))
```

## Plotting with Stan output

The object `lm_intercept` is a `stanfit` object, which means there's a lot of other plotting functionality from the `rstan` package we can use

- First, let's make some traceplots

```
rstan::traceplot(lm_intercept,  
                 pars = c("beta[1]", "sigma"), nrow=2, ncol=
```

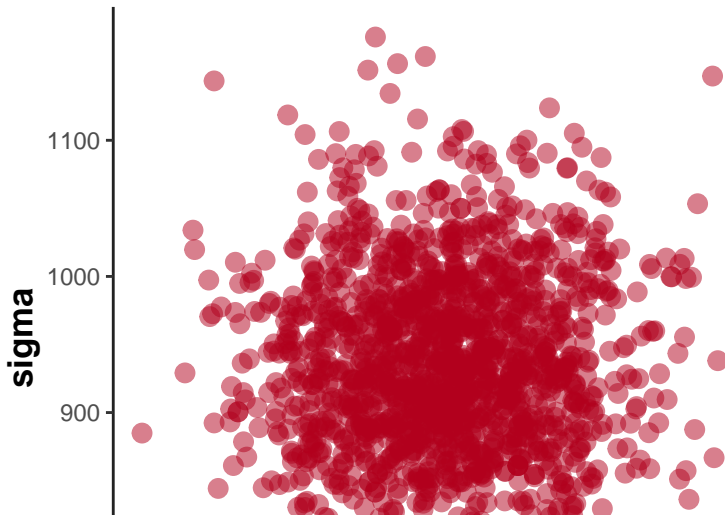




## Plotting with Stan output

- ▶ Second, we can examine the correlation between parameters with a pairs plot,

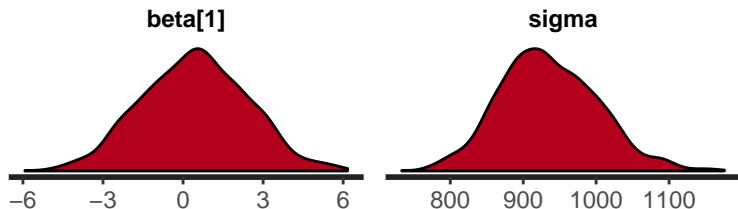
```
rstan::stan_scatter(lm_intercept, pars = c("beta[1]", "sigma"))
```



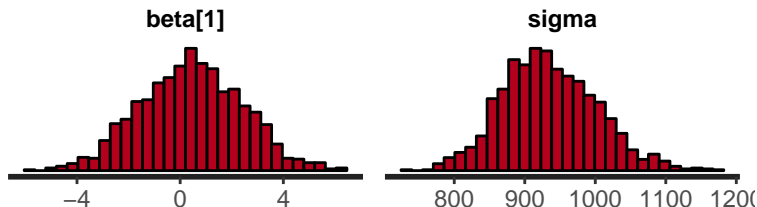
## Plotting with Stan output

- ▶ Third, maybe we want to make some density plots or histograms

```
rstan::stan_dens(lm_intercept, pars = c("beta[1]", "sigma"))
```



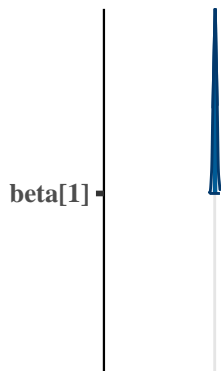
```
rstan::stan_hist(lm_intercept, pars = c("beta[1]", "sigma"))
```



## Plotting with Stan output

- Fourth, we can look at implementing some of the plots included in bayesplot

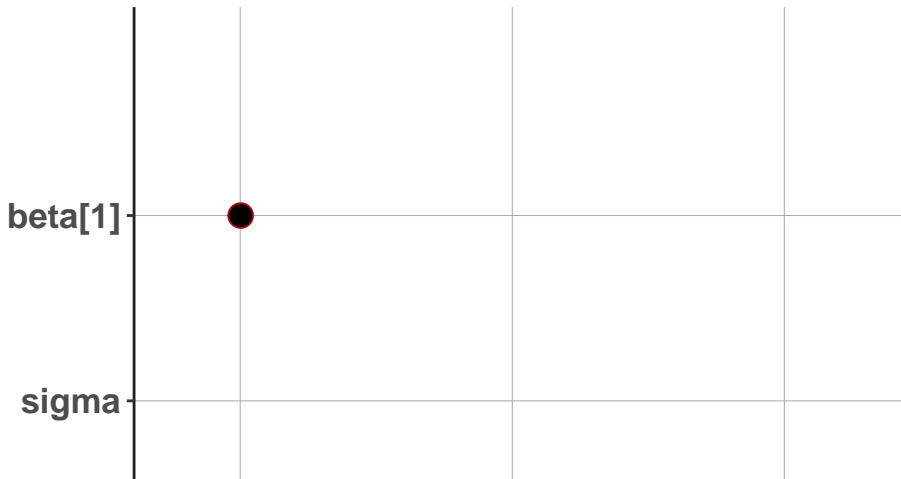
```
library(bayesplot)
mcmc_areas(as.matrix(lm_intercept),
           pars = c("beta[1]", "sigma"),
           prob = 0.8)
```



## Plotting with Stan output

- ▶ Another way to show the same uncertainties is with dotplots and credible intervals,

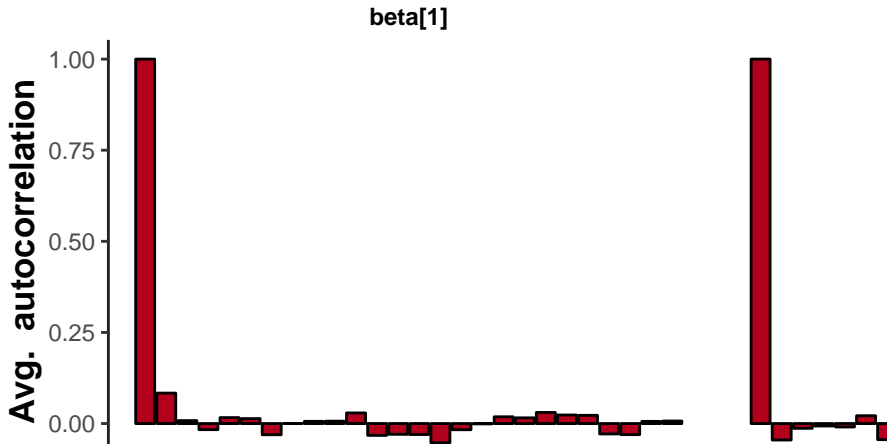
```
rstan::stan_plot(lm_intercept, pars=c("beta[1]", "sigma"))
```



## Plotting with Stan output

- ▶ We can also look at the autocorrelation of each parameter, which is another useful diagnostic for convergence (we want very low levels of autocorrelation)

```
rstan::stan_ac(lm_intercept, pars=c("beta[1]", "sigma"))
```



# Plotting with Stan output

These plots only the tip of the iceberg for plotting. For more great examples of the kinds of plots available, see these vignettes:

- ▶ Examples on Stan
- ▶ Jonah Gabry's introduction to bayesplot
- ▶ Matthew Kay's introduction to bayesplot and tidybayes

## Tidy summaries from Stan output

Using the broom package, we can also extract some tidy summaries of the output

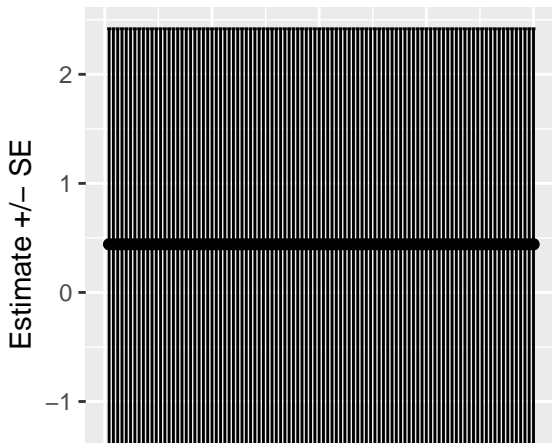
```
coef = broom::tidy(lm_intercept)
head(coef)
```

```
## # A tibble: 6 x 3
##   term      estimate std.error
##   <chr>      <dbl>      <dbl>
## 1 beta[1]      0.441      1.98
## 2 sigma     935.      66.9
## 3 pred[1]      0.441      1.98
## 4 pred[2]      0.441      1.98
## 5 pred[3]      0.441      1.98
## 6 pred[4]      0.441      1.98
```

## Tidy summaries from Stan output

These tidy summaries can then be fed into ggplot for example

```
coef = broom::tidy(lm_intercept)
ggplot(coef[grep("pred",coef$term),], aes(x = 1:100,y=estimate))
  geom_point() + ylab("Estimate +/- SE") + xlab("") +
  geom_errorbar(aes(ymin=estimate-std.error, ymax=estimate+std.error))
```





## Preserving chain order

For models with multiple chains, we might want to preserve the chain ID to look at individual chain diagnostics. Remember that

- ▶ Each chain is independent
- ▶ `extract` defaults to merging samples from all chains together, e.g.

```
extract(object, pars, permuted = TRUE)
```

- ▶ But summaries can be generated for each combination of parameters-chains by setting

```
extract(object, pars, permuted = FALSE)
```

## More time series models: random walk

This model should be familiar,

$$E[Y_t] = E[Y_{t-1}] + e_{t-1}$$

- ▶ We'll fit model to temperature data
- ▶ Note that the use of the argument `model_name` and `est_drift`. By not estimating drift, we assume the process is stationary with respect to the mean

```
data(airquality)
Temp = airquality$Temp # air temperature
rw = fit_stan(y = Temp, est_drift = FALSE, model_name = "rw")
```

## More time series models: random walk

Did the model converge?

- ▶ One quick check is to look at the value of R-hat for each parameter (generally should be small,  $< 1.05$  or smaller)

```
rw_summary <- summary(rw, pars = c("sigma"),  
                      probs = c(0.1, 0.9))$summary  
print(rw_summary)
```

```
##           mean      se_mean      sd      10%      90%  
## sigma 5.764972 0.01281503 0.328861 5.358552 6.205832 658
```

## More time series models: univariate state space models

State equation:

$$x_t = \phi x_{t-1} + \varepsilon_{t-1}$$

where  $\varepsilon_{t-1} \sim \text{Normal}(0, q)$

Observation equation:

$$Y_t \sim \text{Normal}(x_t, r)$$

- ▶ Let's compare models with and without the AR parameter  $\phi$  in the process model

## More time series models: univariate state space models

We can first run the model with  $\phi$ ,

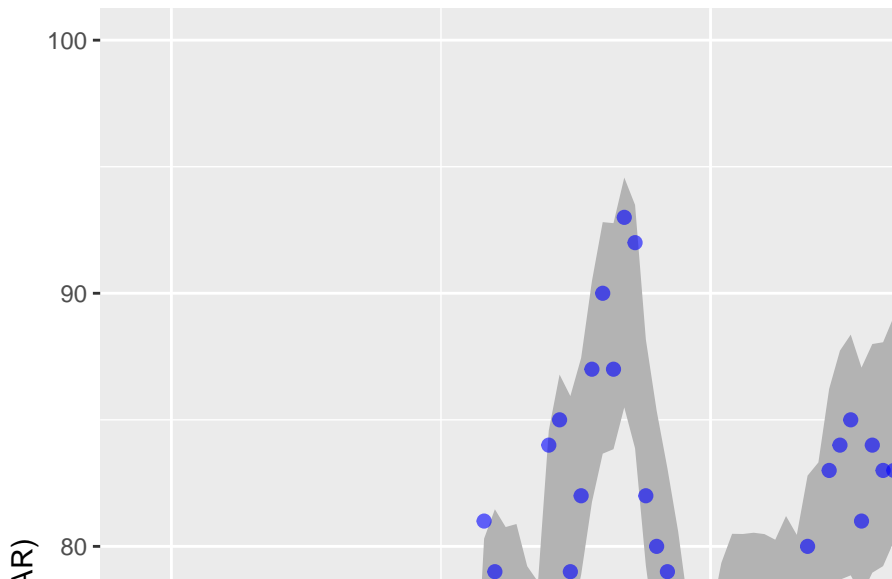
```
ss_ar = fit_stan(y = Temp, est_drift=FALSE, model_name = "s
```

then without,

```
ss_rw = fit_stan(y = Temp, est_drift=FALSE, model_name = "s
```

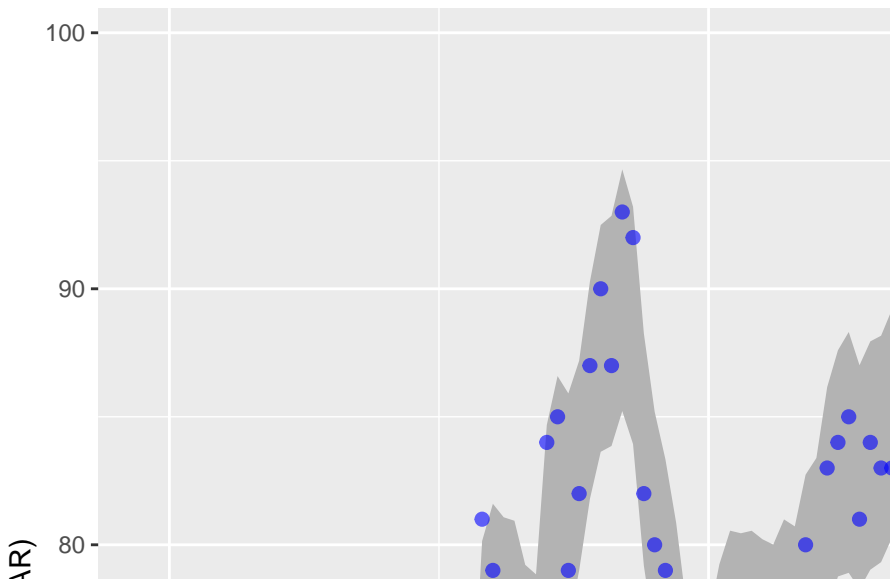
## More time series models: univariate state space models

Estimates from the AR(1) state space model



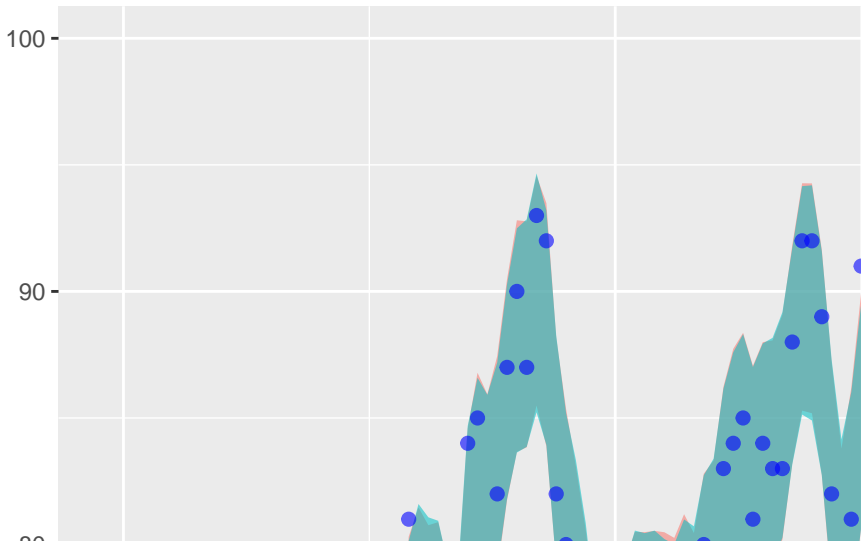
# More time series models: univariate state space models

Estimates from the RW state space model



## More time series models: univariate state space models

Estimates from both models (note the difference in credible interval widths)



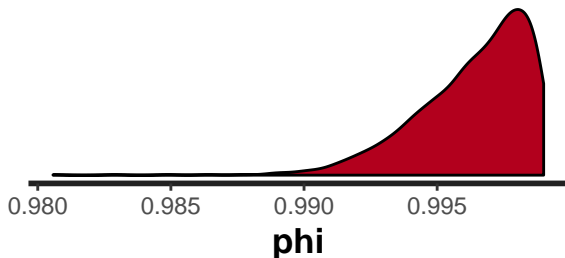


## More time series models: univariate state space models

We might be also interested in looking at posterior estimates for these models.

- What is the posterior distribution of  $\phi$ ?

```
rstan::stan_dens(ss_ar, c("phi"))
```



- This shows with a value near 1, the behavior of the model should be very similar to the random walk (from the predictions, it is!). The multi-modal distribution probably is an indicator of convergence issues

## More time series models: univariate state space models

We might also be interested in derived quantities.

- For example, what's the probability of temperature exceeding 90 degrees?

```
pars = extract(ss_ar)
p = length(which(pars$pred > 90)) / length(pars$pred)
print(round(p,3))
```

```
## [1] 0.077
```

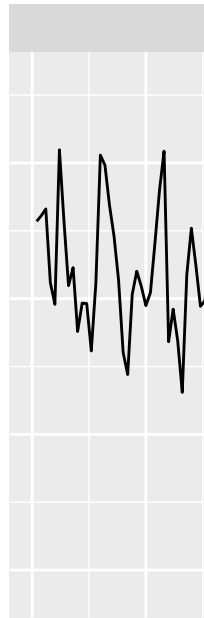
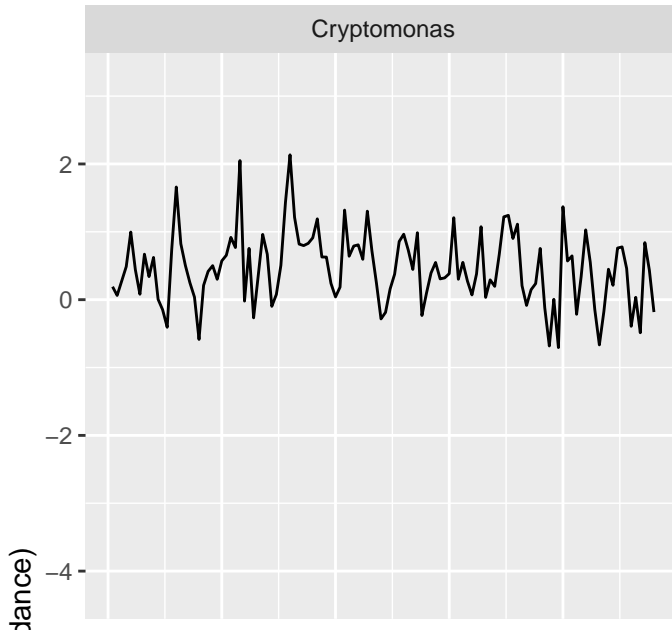
# Dynamic Factor Analysis (DFA)

As an example of Bayesian DFA, we'll load the WA plankton dataset. There's 3 versions of the data, and we'll use the one that's been transformed. As a reminder,

- + 0s replaced with NAs
- + data have been z-scored
- + we only use data from 1980-1989 for simplicity

```
dat = lakeWAp planktonTrans
plankdat = dat[dat[, "Year"] >= 1980 & dat[, "Year"] < 1990, ]
phytoplankton = c("Cryptomonas", "Diatoms", "Greens",
                  "Unicells", "Other.algae")
dat.spp.1980 = plankdat[, phytoplankton]
```

# Dynamic Factor Analysis (DFA)



## Running the model

For starters, we'll try a 3-trend model,

```
mod_3 = fit_dfa(y = t(dat.spp.1980), num_trends=3)
```

# Trends need to be rotated (like MARSS)

Why? Identifiability

- ▶ Try re-ordering time series, and running DFA on each new dataset
- ▶ Results will be sensitive to time series order, BUT not after rotation
- ▶ Like MARSS we'll use varimax rotation

Use function we've written, `rotate_trends`

## Attributes of rotated object

`Z_rot`, rotated  $Z$  matrix for each MCMC draw

`trends`, rotated trends for each MCMC draw

`Z_rot_mean`, mean  $Z$  across draws

`trends_mean`, mean trends across draws

`trends_lower`, lower 2.5% on trends

`trends_upper`, upper 97.5% on trends

## Other variance structures

By default, the structure is diagonal and equal

Diagonal and unequal or shared variances can also be specified using `varIndx` argument. The diagonal and unequal structure would be called with

```
mod_3 = fit_dfa(y = t(dat.spp.1980), num_trends=3, varIndx
```



## Uncertainty intervals on states

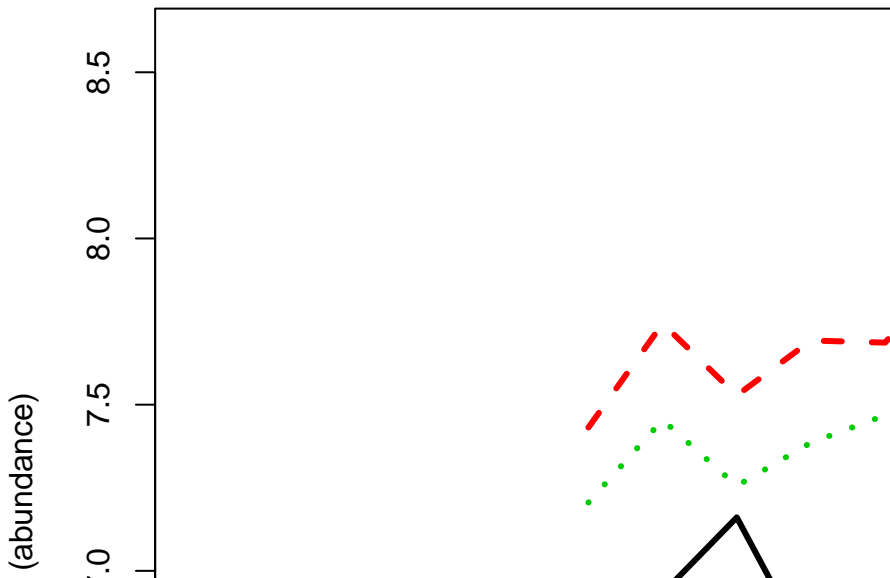
We often just present the trend estimates for DFA – but not uncertainty

Let's look at effect of missing data on DFA for the harbor seal dataset

```
data("harborSealWA")
```

## Model

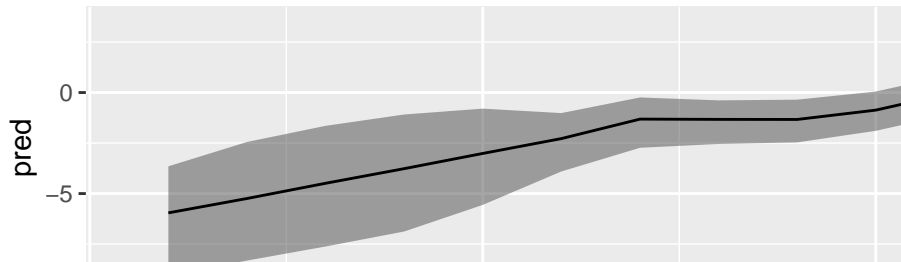
Assume single trend for the population



## Extracting the predicted trend

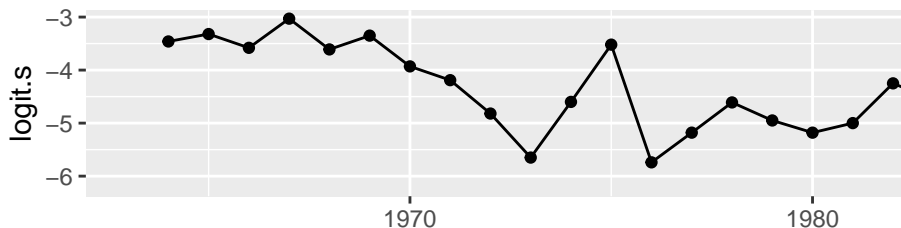
We'll extract predictions from the best model,

```
pars = extract(fit_dfa(y = t(harborSealWA[,-1]), num_trends = 2))
df = data.frame("time"=1:nrow(harborSealWA),
               "pred"=apply(pars$x[,1,], 2, mean),
               "low"=apply(pars$x[,1,], 2, quantile,0.025),
               "hi"=apply(pars$x[,1,], 2, quantile,0.975))
ggplot(df, aes(time, pred)) +
  geom_ribbon(aes(ymin=low,ymax=hi), fill="grey30", alpha=0.5) +
  geom_line()
```



## Fitting a DLM with time varying intercept

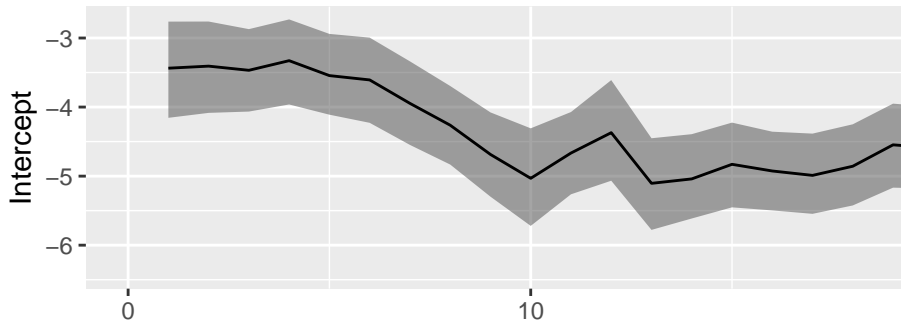
For comparison to MARSS, we'll use Mark's example of logit-transformed survival from the Columbia River. We can think about setting the DLM up in the slope or the intercept. For this first example, we'll do the latter.



```
mod = fit_stan(y = SalmonSurvCUI$logit.s,  
               model_name="dlm-intercept")
```

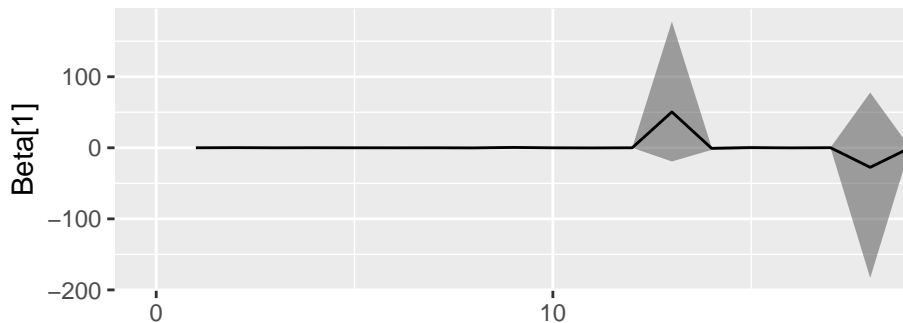
## Fitting a DLM with time varying intercept

```
df = data.frame("year"=1:42,  
"pred"=apply(extract(mod, "intercept", permuted=FALSE), 3, FUN=function(x) {  
"low"=apply(extract(mod, "intercept", permuted=FALSE), 3, FUN=function(x) {  
"hi"=apply(extract(mod, "intercept", permuted=FALSE), 3, FUN=function(x) {  
ggplot(df, aes(year,pred)) +  
  geom_ribbon(aes(ymin=low,ymax=hi),fill="grey30",alpha=0.5)  
  geom_line() + ylab("Intercept") + xlab("Time")
```



## Fitting a DLM with single intercept and time-varying slope

```
mod = fit_stan(y = SalmonSurvCUI$logit.s,  
x = SalmonSurvCUI$CUI.apr, model_name="dlm-slope")
```



## Fitting a DLM time-varying intercept and time-varying slope

- Use `model.matrix()` to specify  $x$

```
lmmod = lm(SalmonSurvCUI$logit.s ~ SalmonSurvCUI$CUI.apr)
x = model.matrix(lmmod)
```

```
lmmod = lm(SalmonSurvCUI$logit.s ~ SalmonSurvCUI$CUI.apr)
mod = fit_stan(y = SalmonSurvCUI$logit.s,
               x = model.matrix(lmmod),
               model_name="dlm")
```

# Summary

- ▶ Bayesian implementation of time series models in Stan can do everything that MARSS can do and more!
- ▶ Very flexible language, great developer community
- ▶ Widely used by students in SAFS / UW / QERM / etc
- ▶ Please come to us with questions or modeling issues!