Fitting multivariate Bayesian time series models

FISH 507 – Applied Time Series Analysis

Eric Ward

11 Feb 2021

Overview of today's material

- Using STAN for MAP estimation
- Multivariate time series models
- DFA models
- Writing our own Stan code

MAP (maximum a posteriori) estimation

- Point estimate of unknown parameters
- Optimization algorithms: BFGS, Newton, etc
- Estimates similar to maximum likelihood, but also incorporate prior
- Quick for checking models, etc

MAP estimation in Stan

- We left off with a Bayesian DLM with random walk in intercept

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

MAP estimation in Stan

- Switching this to MAP estimation

```
map_fit = fit_stan(y = SalmonSurvCUI$logit.s,
                   model_name="dlm-intercept", map_estimation=T
```

MAP estimation in Stan

```r
print(map_fit)
```

```
## $par
##              x0       beta[1]   pro_dev[1]   pro_dev[2]
## -3.460000e+00 -1.161100e+00  1.040939e-01 -1.933172e-01
##    pro_dev[4]    pro_dev[5]   pro_dev[6]   pro_dev[7]
## -4.312462e-01  1.933172e-01 -4.312462e-01 -1.933172e-01
##    pro_dev[9]   pro_dev[10]  pro_dev[11]  pro_dev[12]
## -6.171283e-01  7.807044e-01  8.030103e-01 -1.650632e+00
##   pro_dev[14]   pro_dev[15]  pro_dev[16]  pro_dev[17]
##  4.238109e-01 -2.527996e-01 -1.710113e-01  1.338350e-01
##   pro_dev[19]   pro_dev[20]  pro_dev[21]  pro_dev[22]
## -2.304936e-01 -6.691753e-02  3.494581e-01 -8.030104e-01
##   pro_dev[24]   pro_dev[25]  pro_dev[26]  pro_dev[27]
##  1.263998e-01 -1.858821e-01  6.691752e-01 -7.435279e-01
##   pro_dev[29]   pro_dev[30]  pro_dev[31]  pro_dev[32]
##  6.691740e-02  8.922337e-01 -5.204696e-02 -5.279050e-01
##   pro_dev[34]   pro_dev[35]  pro_dev[36]  pro_dev[37]
```

MAP estimation in Stan

- Check that model converged

```
map_fit$return_code
```

```
## [1] 0
```

- MAP value when converged

```
map_fit$value
```

```
## [1] 615.844
```

MAP estimation in Stan

- grep or other string matching functions needed

```r
grep("pred",names(map_fit$par))
```

```
##  [1] 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62
## [26] 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
```

```r
pred = map_fit$par[grep("pred",names(map_fit$par))]
```

MAP estimation in Stan

- grep or other string matching functions needed

```
grep("pred",names(map_fit$par))
```

```
##  [1] 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62
## [26] 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
```

```
pred = map_fit$par[grep("pred",names(map_fit$par))]
```

MAP estimation in Stan

- grep or other string matching functions needed

```r
grep("pred",names(map_fit$par))
```

```
##  [1] 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62
## [26] 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
```

```r
pred = map_fit$par[grep("pred",names(map_fit$par))]
```

MAP estimation in Stan

- include SEs of estimates, via Hessian

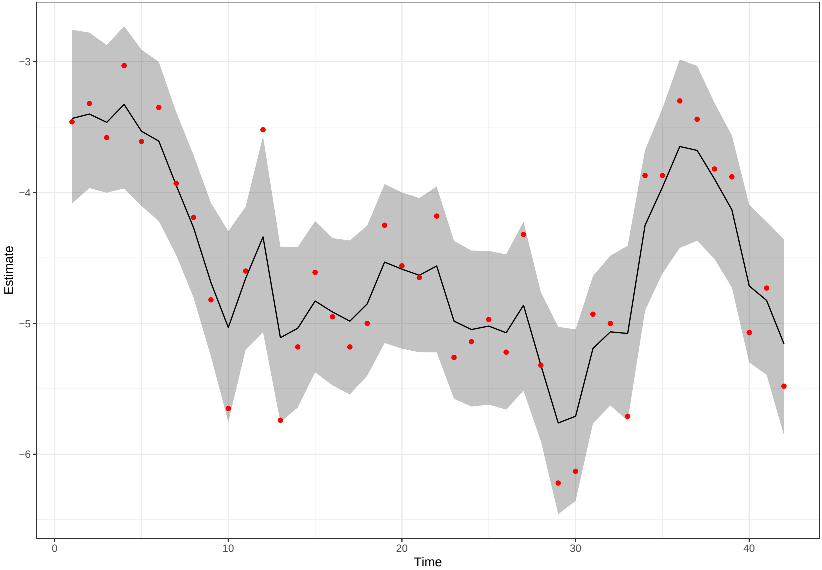
```
map_fit = fit_stan(y = SalmonSurvCUI$logit.s,
     model_name="dlm-intercept",
     map_estimation=TRUE,
     hessian=TRUE)
```

- or change algorithm (“LBFGS” default) to “BFGS” or “Newton”

```
map_fit = fit_stan(y = SalmonSurvCUI$logit.s,
     model_name="dlm-intercept",
     map_estimation=TRUE,
     hessian=TRUE,
     algorithm="BFGS")
```

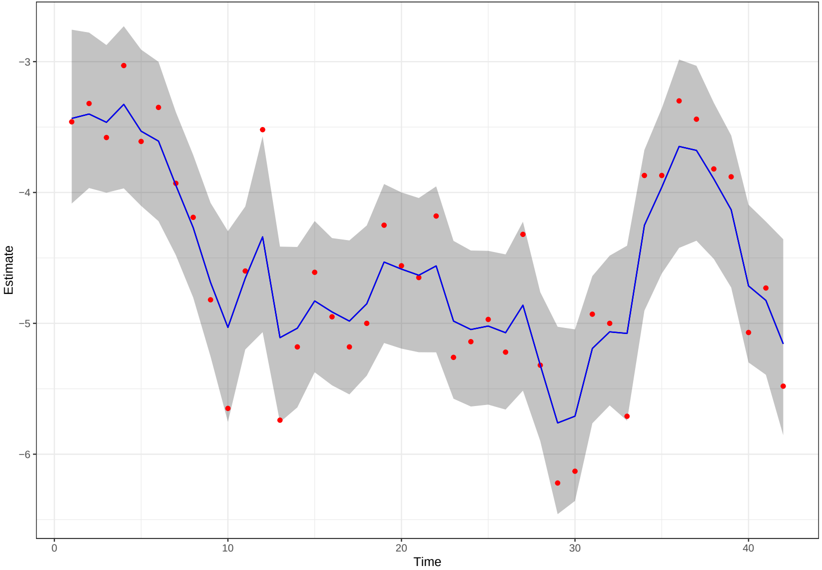
MAP estimation in Stan

- Posterior means, 95% CIs



MAP estimation in Stan

- MAP estimates = posterior



Multivariate time series models

- Multivariate state space models
- Stan package for fitting models similar to MARSS

```r
devtools::install_github("nwfsc-timeseries/tvvarss")
```

Multivariate time series models

- Most models in tvvarss() are exactly the same as MARSS
- Takes in data matrix y
- `family` argument defaults to Gaussian, but can be many others

```
tvvarss::tvvarss(y = y, family = "poisson",...)
```

Multivariate time series models

- Like MARSS, variances can be shared (default) or unique by species, or time series

```
tvvarss::tvvarss(y = y, shared_r = R, shared_q = Q,...)
```

Multivariate time series models

- Optional 'process' argument acts as `Z` in MARSS and maps time series to latent state processes

```
tvvarss::tvvarss(y = y, shared_r = R, shared_q = Q,...)
```

Multivariate time series models

- Perhaps most exciting feature is time-varying interactions (AR coefficients)
- Time-varying vector autoregressive models
- AR coefficients behave like a DLM

```
tvvarss::tvvarss(y = y, dynamicB = TRUE,...)
```

- Time-varying parameters are data hungry! We'll dive into this more later in the quarter

Bayesian DFA models

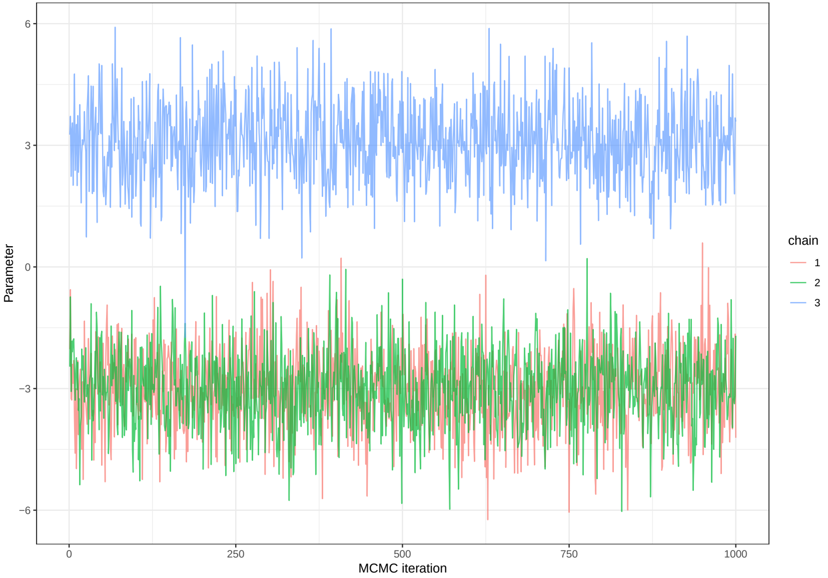
- DFA models from `atsar` bundled with other DFA code we've developed

```
devtools::install_github("fate-ewi/bayesdfa")
```

Bayesian DFA models

- DFA poses interesting identifiability challenges
- Bayesian models generally involve > 1 MCMC chain
- Direction of DFA loadings / trends don't have meaning

Bayesian DFA models



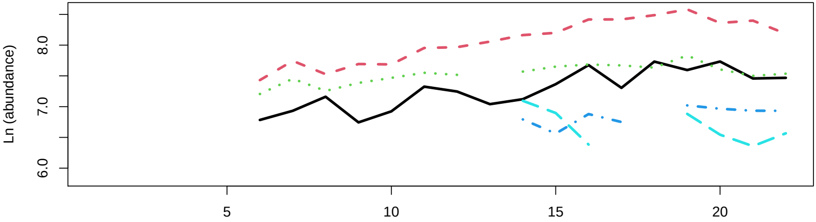
Bayesian DFA models

- Solution 1: priors for identifiability
- Solution 2: post-hoc ‘chain flipping’ before convergence tests run

Bayesian DFA models

- Fitting DFA models will give very similar answers to using MARSS

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



Bayesian DFA models

We'll extract predictions from the best model,

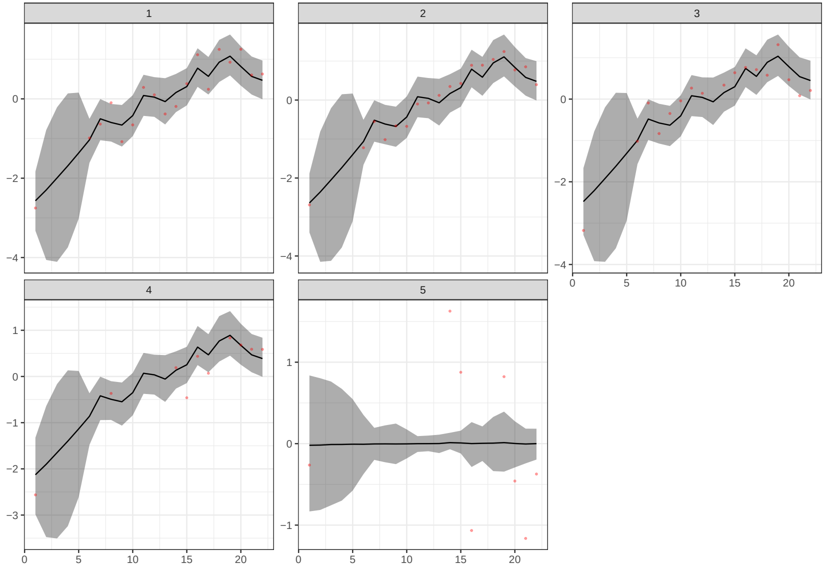
```
fit = bayesdfa::fit_dfa(y = t(harborSealWA[,-1]), num_trend
```

And as an `stanfit` object, we can extract summaries of states `x`

```
pars = extract(fit$model)
```

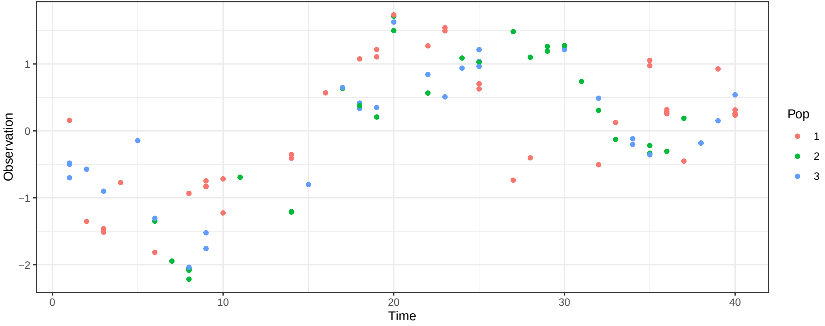
Bayesian DFA models

```
bayesdfa::plot_fitted(fit) + theme_bw()
```



Bayesian DFA models

- DFA extension 1: long format data (optional), replicate observations



Bayesian DFA models

- DFA extension 2: temporal extremes

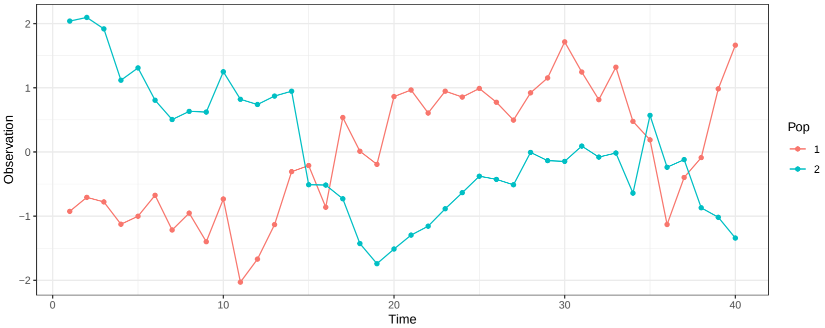
$$x_t = x_{t-1} + \delta_{t-1}$$

$$\delta_{t-1} \sim Normal(0, q)$$

- But do these deviations have to be normal? NO!

$$\delta_{t-1} \sim Student - t(\nu, 0, q)$$

Bayesian DFA models



Bayesian DFA models

- Extremes example: coho salmon body size in coastal fisheries

Bayesian DFA models

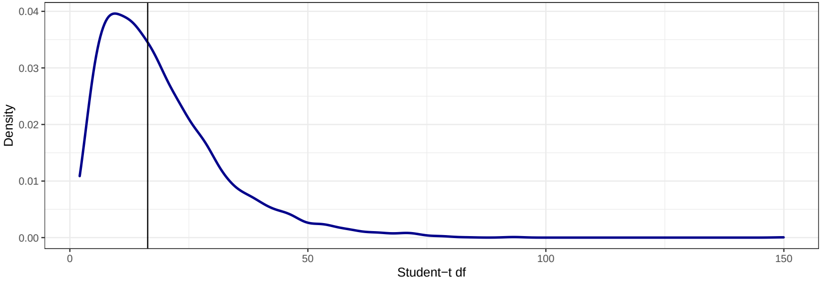
- Fitting the model

```
coho = readRDS("coho_mean_length.rds")
# rename variables
coho=dplyr::rename(coho, obs=mean_length,time=recovery_year
fit = fit_dfa(y = coho, data_shape="long",estimate_nu = TRU
```

Bayesian DFA models

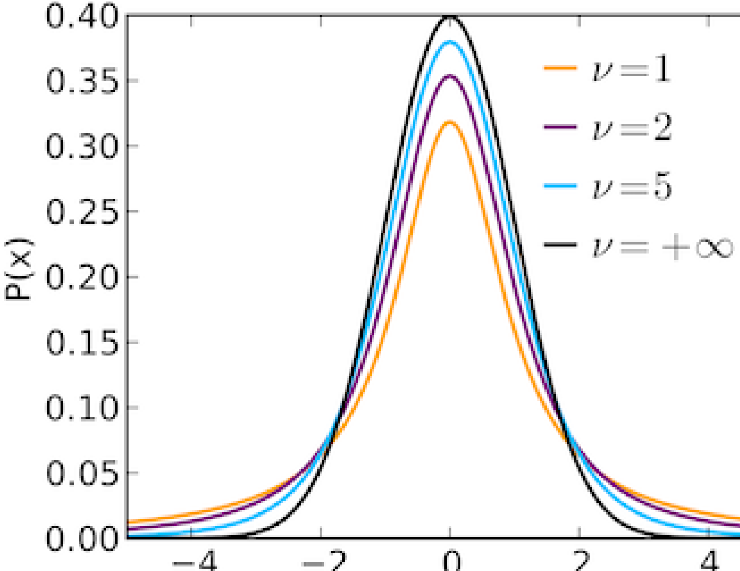
- Let's look at df parameter, ν
- Mean $\sim$ 19.5, median $\sim$ 16.3

```
pars = extract(fit$model)
```



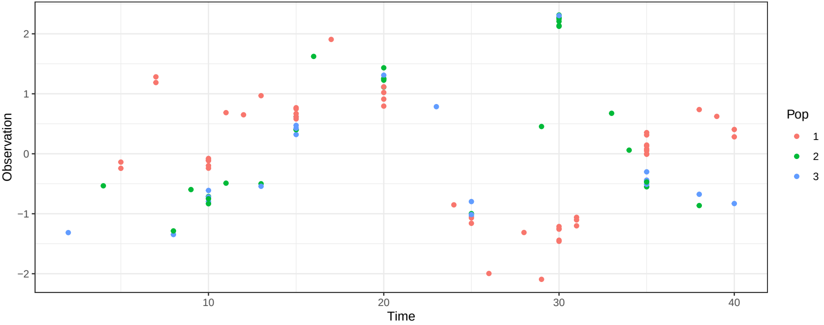
Bayesian DFA models

- ν parameter ranges from 2 (heavy tails) to normal (30)



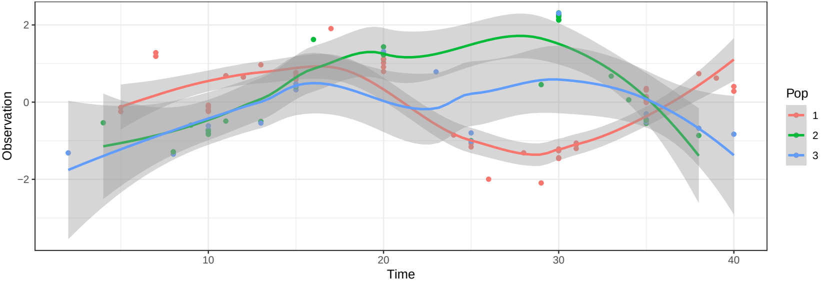
Bayesian DFA models

- DFA extension 2: increased flexibility in trends
- Sampling might be 'chunky'



Bayesian DFA models

- AR models (conventional DFA) may not be well suited
- Alternative approach: use Gaussian Process models to model smooth trends
- Feddern et al. 2021 (Global Change Biology)



Bayesian DFA models

- What's a Gaussian Process model?
- Instead of modeling process

$$x_t = x_{t-1}...$$

- We model the covariance of the $\mathbf{x}$
- But there are a lot of $\mathbf{x}$!!

Bayesian DFA models

- For model with T timesteps
- Σ has $T * (T-1)/2$ parameters
- GP models implement covariance functions, e.g. exponential, Gaussian, Matern
- Model covariance as function of space or distance in time

Bayesian DFA models

- Example: Gaussian covariance (aka 'squared exponential')

$$\Sigma_{t,t+3} = \sigma^2 exp(\frac{(t-(t+3))^2}{2\tau^2})$$

- σ^2 controls variability
- τ controls how quickly covariance decays between points

Bayesian DFA models

- Implementation

```
fit_dfa(y= y, num_trends = 2, ..., trend_model = c("gp"))
```

Bayesian DFA models

- DFA extension 3: alternate constraints on loadings matrix
- Conventional constraints on loadings matrix

```
##             [,1]       [,2]       [,3]
## [1,] -0.56047565  0.0000000  0.0000000
## [2,] -0.23017749  0.4609162  0.0000000
## [3,]  1.55870831 -1.2650612  0.4007715
## [4,]  0.07050839 -0.6868529  0.1106827
## [5,]  0.12928774 -0.4456620 -0.5558411
```

Bayesian DFA models

- Alternative: compositional DFA model

```
##              [,1]       [,2]       [,3]
## [1,] 0.6871757 0.24516978 0.06765452
## [2,] 0.1491745 0.06014499 0.79068047
## [3,] 0.3885419 0.59306319 0.01839489
## [4,] 0.2919682 0.30545457 0.40257721
## [5,] 0.3760087 0.33262301 0.29136825
```

Bayesian DFA models

- Under compositional DFA model, time series are true mixtures of underlying trends

$$\mathbf{Y} = \mathbf{Z}\mathbf{x}$$

```
fit_dfa(y = y, num_trends = 2, ..., z_model = "proportion")
```

- When might this be useful?
- Stable isotope data, environmental monitoring, pollutant data, etc

Writing our own Stan scripts

- Stan scripts always start with a data block
- Data needs to be typed

```
data {
  int<lower=0> N;
  int<lower=0> K;
  real y[N];
  int P;
  int y_int[N];
  matrix[N, K] x;
}
```

- Stan manual

Writing our own Stan scripts

- transformed data block is optional

```
transformed data {
  int zeros[N];
  for(i in 1:N) {
    zeros[i] = 0;
  }
}
```

Writing our own Stan scripts

- parameters block containts any parameters

```
parameters {
  real x0;
  vector[K] beta0;
  vector[K] pro_dev[N-1];
  real<lower=0> sigma_process[K];
  real<lower=0> sigma_obs;
}
```

Writing our own Stan scripts

- parameters block containts any parameters

```
parameters {
  real x0;
  vector[K] beta0;
  vector[K] pro_dev[N-1];
  real<lower=0> sigma_process[K];
  real<lower=0> sigma_obs;
}
```

Writing our own Stan scripts

- transformed parameters block contains derived quantities
- examples: predicted states for state space time series model

```
transformed parameters {
  vector[N] pred;
  pred[1] = x0;
  for(i in 2:N) {
    pred[i] = phi*pred[i-1] + sigma_process*pro_dev[i-1];
  }
}
```

Writing our own Stan scripts

- model block contains priors on parameters
- likelihood (or data model)

```
model {
  x0 ~ normal(0,10);
  phi ~ normal(0,1);
  sigma_process ~ student_t(3,0,2);
  sigma_obs ~ student_t(3,0,2);
  pro_dev ~ std_normal();
  for(i in 1:N) {
    y[i] ~ normal(pred[i], sigma_obs);
  }
}
```

Writing our own Stan scripts

- generated data block (optional)
- includes quantities useful for model selection, prediction, forecasts, etc

```
generated quantities {
  vector[N] log_lik;
  for (n in 1:N) log_lik[n] = normal_lpdf(y[n] | pred[n], s
}
```

Writing our own Stan scripts

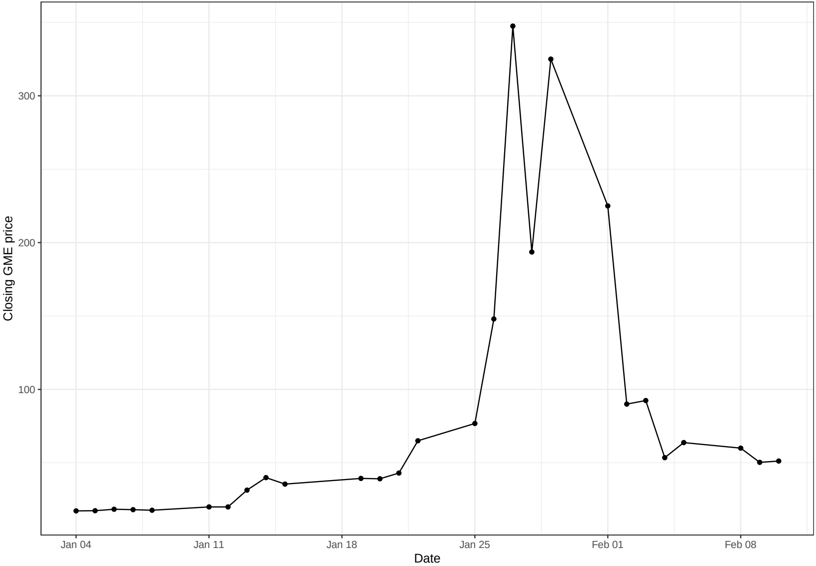
- Ok – let's do this in practice
- Problem # 1: let's take our state space random walk model and modify it to better handle extreme events with the Student - t distribution
- Mathematically

$$\delta_{t-1} \sim Normal(0, q)$$

becomes

$$\delta_{t-1} \sim Student - t(\nu, 0, q)$$

Writing our own Stan scripts



Writing our own Stan scripts

- You could do this in RStudio (File -> New File -> Stan File)
- Scripts for the `atsar` package linked below

https://github.com/nwfsc-timeseries/atsar/tree/master/inst/stan

- We're just going to modify this code (ss_ar.stan) rather than start from scratch

Writing our own Stan scripts

- 2 things need to change
- Change process error deviations from Normal to Student-t distribution
- Add ν as a parameter, with constraints (> 2)

Writing our own Stan scripts

```
parameters {
real<lower=2> nu;
...
}
```

Writing our own Stan scripts

```
model{
...
nu ~ student_t(3, 2, 3);
pro_dev ~ student_t(nu, 0, 1);
//pro_dev ~ std_normal();
...
}
```

Writing our own Stan scripts

- Done! Now we can fit the model

```
y = gme$Close
N = length(y)
n_pos = length(which(!is.na(y)))
pos_indx = c(which(!is.na(y)),0,0)

fit = stan(file = "ss_ar_t.stan",
  data = list(y = y, N = N, n_pos = n_pos, pos_indx = pos_i
```

Writing our own Stan scripts