

GPs, GMRFs, and Splines, Oh My!

Nonparametric modeling with correlated Gaussian effects

Devin S. Johnson, Ph.D.

NOAA Fisheries Marine Mammal Laboratory

Seattle, Washington

Email: devin.johnson@noaa.gov

Data Science Meetup

April 9, 2019



NOAA
FISHERIES

Code, etc...

Everything presented here is available on my github account

user:

dsjohnson

repo:

data_science_tak

Why GPs?

- You don't have to assume functional forms
- No need for training data (i.e., cross-validation) for parameter estimation
- Works quite well at prediction
- Certain GPs are equivalent to a infinite node NN

Why not GPs

- They are computationally burdensome (scales like n^3)
- Estimation can be numerically tricky
- There's actually not much software out there for complex data analysis using GPs (in the base form)

The basics

Multivariate normal distribution

$$x \sim N(m, V)$$

m is the expected value (mean) of x

$V = [\sigma_{ij}]$ is the variance-covariance matrix

- $\sigma_{ii} = \text{var}(x_i)$
- $\sigma_{ij} = \text{cov}(x_i, x_j)$

V has some mathy constraints such as nonnegative-definiteness, that is $a' Va \geq 0$ for any a .

Some properties

$$x \sim N(m, V)$$

- $Ax \sim N(Am, AVA')$
- Decompose $V = M\Lambda M'$ where Λ is diagonal ($\lambda_1 > \dots > \lambda_n$) if λ_k s are small for after some p , then $\tilde{V} = M\tilde{\Lambda}M'$ is a low rank approximation created by setting $\lambda_k = 0$ for $k > p$

Bayesian inference

Bayes rule

$$p(x|y) = \frac{p(y|x)p(x)}{\int p(y|x)p(x)dx}$$

Typically,

- $x = \theta$ and represents a set of parameters
- $p(y|x)$ is a probability model interest
- $p(x)$ is the **prior** distribution of x before data is collected
- $p(x|y)$ is the **posterior** distribution of x after learning from y .

Gaussian processes

Gaussian processes (the mathy version)

GPs are prior distributions over functions

$$f(x) \sim \mathcal{GP}(m(x), k(x, x'))$$

characterized by

- $m(x)$ = mean function (usually constant or 0) and
- $k(x, x')$ = the covariance function

Gaussian process because for any x_1, \dots, x_n
 $[f(x_1), \dots, f(x_n)] \sim N(0, K)$

Covariance functions

Linear

$$k(x, x') = \xi x \cdot x'$$

Squared exponential

$$k(x, x') = \xi \exp\{-\theta(x - x')^2/2\}$$

Exponential

$$k(x, x') = \xi \exp\{-\theta(x - x')\}$$

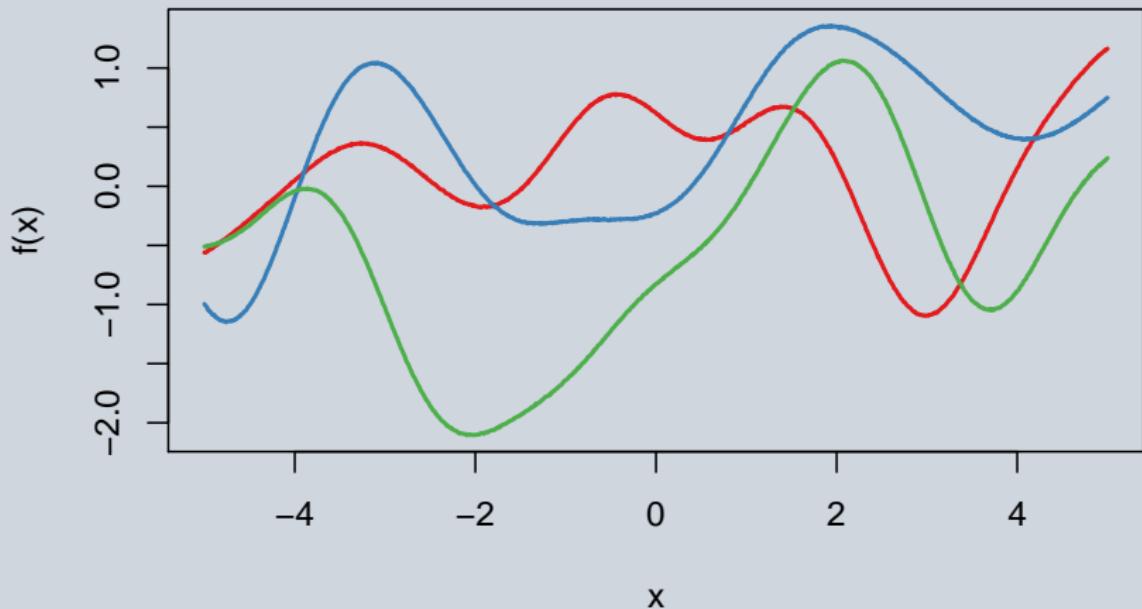
Construct a valid function

$$f(x) = \int b(u - x)w(u)du$$

$$k(x, x') = \int b(u - x)b(x' - u)du$$

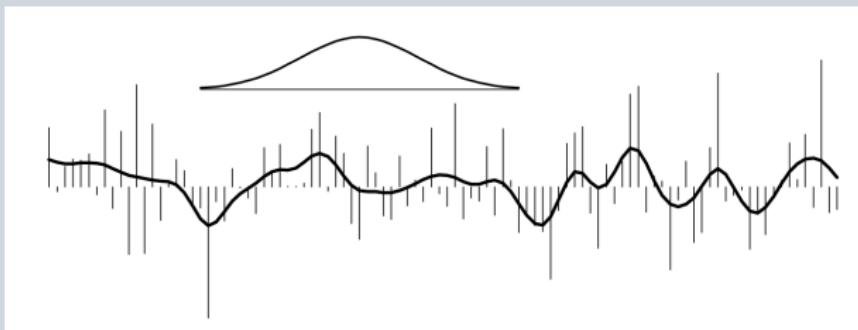
Draws of a GP using squared-exponential

3 GP realisations from the kernel
 $\text{rbf}(x)$



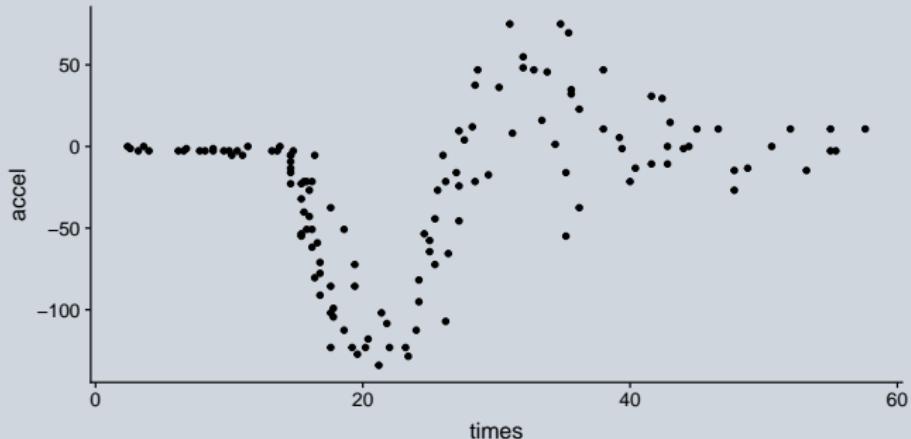
Gaussian process regression (in practice)

- Gaussian processes are usually too burdensome to work with for real problems
- Process convolution provides a way out
- Set $f(x) = \sum b(u_k - x)w(u_k)$



The motorcycle data

```
library(MASS); data(mcycle)
```



Model:

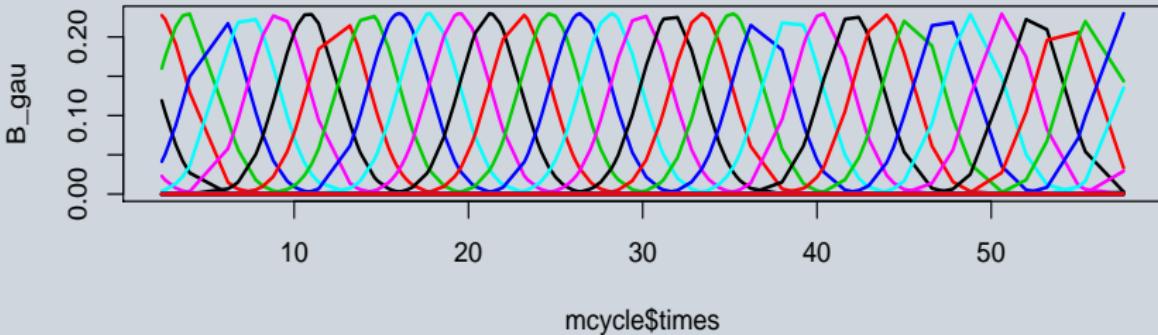
$$\text{accel} = f(\text{times}) + \text{error}$$

```

u <- seq(-10, 75, length=50)
sd_fac <- 1

B_gau <- outer(
  mcycle$times,u,
  FUN=function(x,y,sd){dnorm(x,y, sd)},
  sd=sd_fac*diff(u[1:2])
)

```



$$\mathbf{f}(\mathbf{x}) = \mathbf{B}\mathbf{w}; w(u_k) \sim N(0, \xi^2)$$

$$\mathbf{f}(\mathbf{x}) \sim N(\mathbf{0}, \xi^2 \mathbf{B}\mathbf{B}')$$

Model

$$\mathbf{y} = \mathbf{B}\mathbf{w} + \boldsymbol{\epsilon}$$

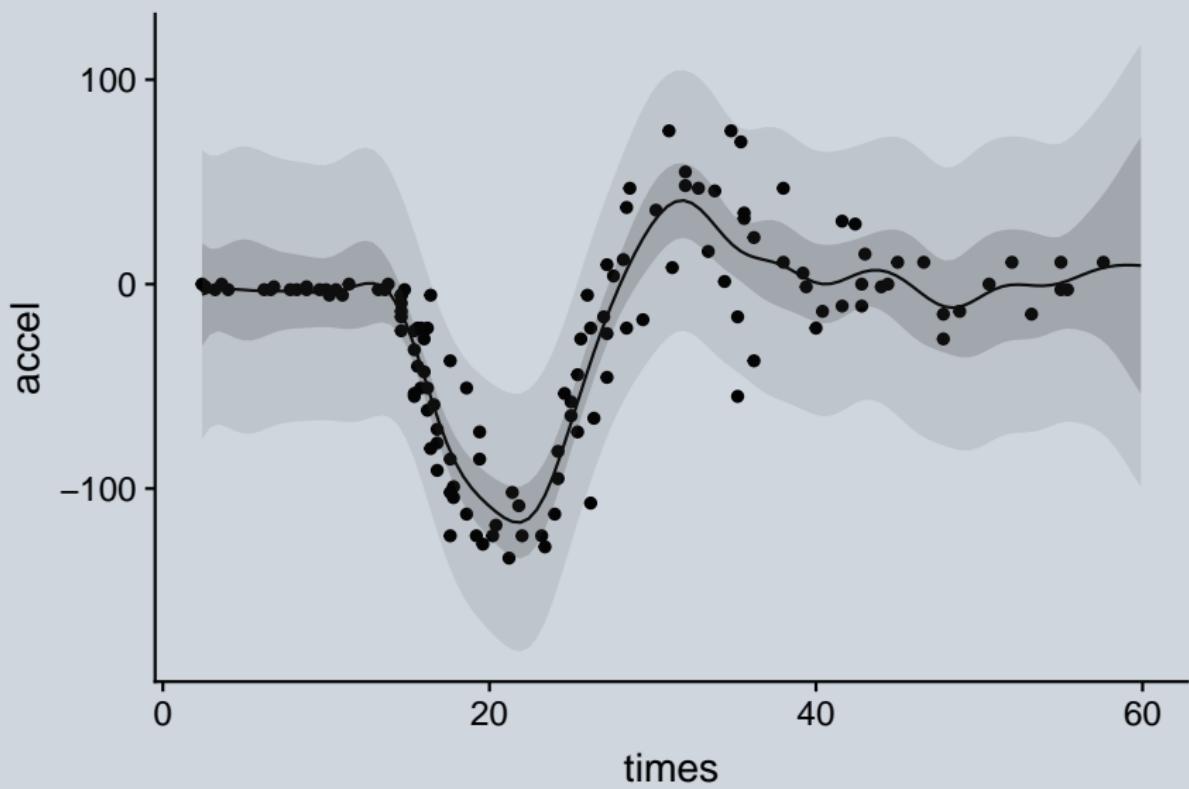
$$(n \times 1) = (N \times p)(p \times 1) + (n \times 1)$$

$$w_k \sim \text{i.i.d.} N(0, \xi^2) \text{ and } \epsilon_i \sim \text{i.i.d.} N(0, \sigma^2)$$

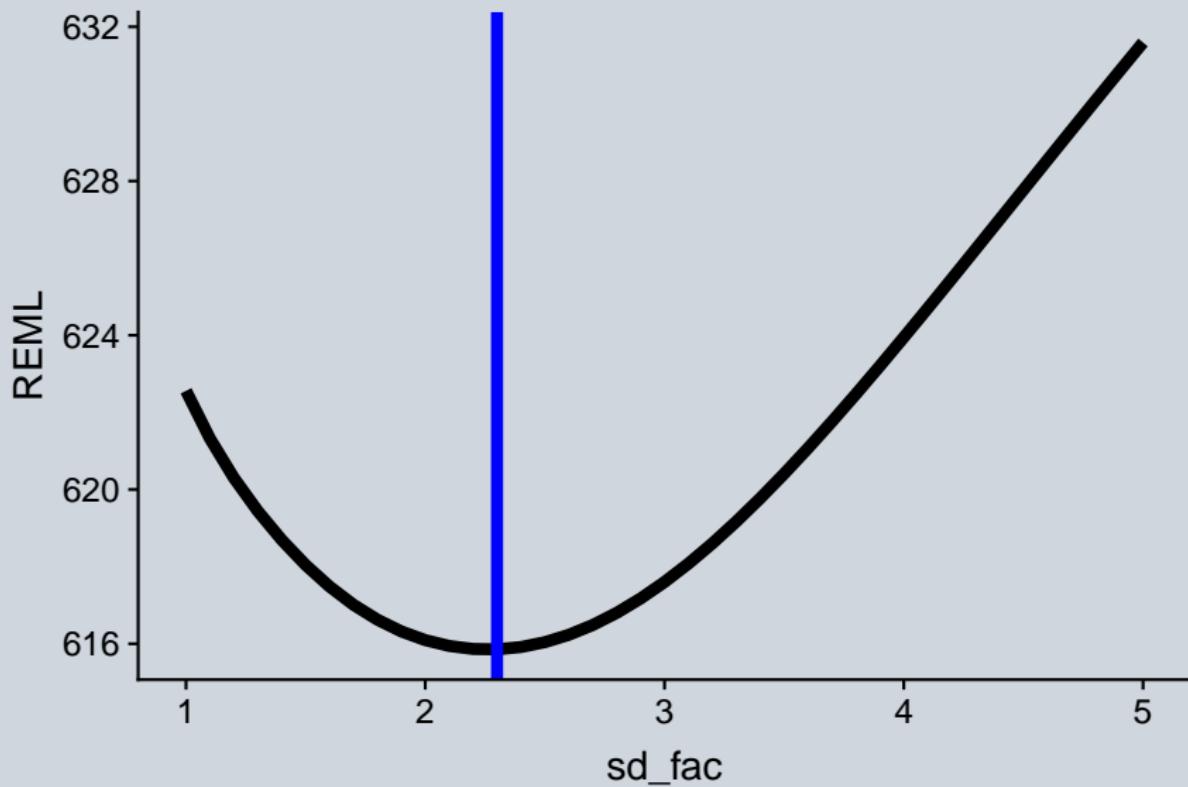
mgcv syntax

```
fit_gau <- mgcv::gam(  
  accel ~ times + B_gau,  
  data=mcycle,  
  paraPen=list(B_gau=list(S=diag(length(u)))),  
  method="REML"  
)
```

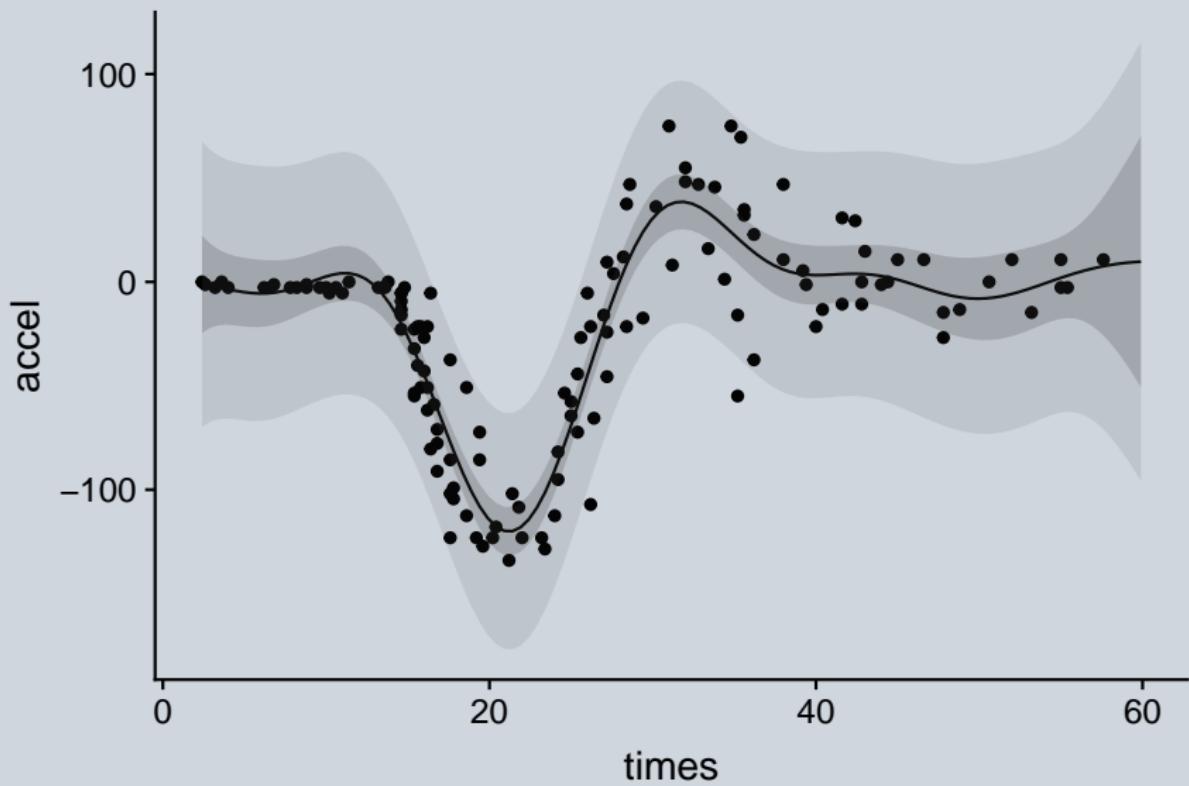
Some prediction



Choosing the optimal kernel



Some better predictions?



Ok, where's the Bayesian part?

- $\hat{f}(x)$ and $\text{var}(\hat{f}(x))$ are the Best Linear Unbiased Estimator
- The BLUP is equivalent to the mean and variance of $p(f|\xi, \mathbf{y})$, the posterior distribution of f (conditioned on ξ)
- mgcv can approximate the variance of $p(f|\mathbf{y})$, the true posterior
- $p(f|\xi, \mathbf{y})$ is normal for normal response, approximate for non-normal (e.g., Poisson)

Penalized splines

Nonparametric regression modeling

$$y_i = f(x_i|\theta) + \epsilon_i$$

- The form of $f(\cdot|\beta)$ is unknown
- $\epsilon_i \sim N(0, \sigma)$

estimate (β, σ) by minimizing **penalized likelihood**

$$\hat{\beta} = \operatorname{argmax}_{\beta} \left\{ \sum_i \log \mathcal{L}(y_i|\beta, \sigma) + J_{\lambda}(\beta) \right\}$$

where J_{λ} is a penalty that keeps the parameters from overfitting (i.e., keeps the wiggliness down)

Gaussian surface interpretation

There are many different types of penalized spline smoother: cubic, thin-plate, P-splines, B-splines, etc.

But they all look like this:

$$f(x) = \sum_k^p \beta_k b_k(x)$$

$$J_\lambda(\beta) = -\lambda \frac{1}{2} \beta' \mathbf{S} \beta$$

Looks familiar, hmmm.....

$$\mathbf{f}(x) \sim N(\mathbf{0}, \lambda^{-1} \mathbf{B} \mathbf{S}^{-1} \mathbf{B}')$$

Modeling northern fur seal pup migration

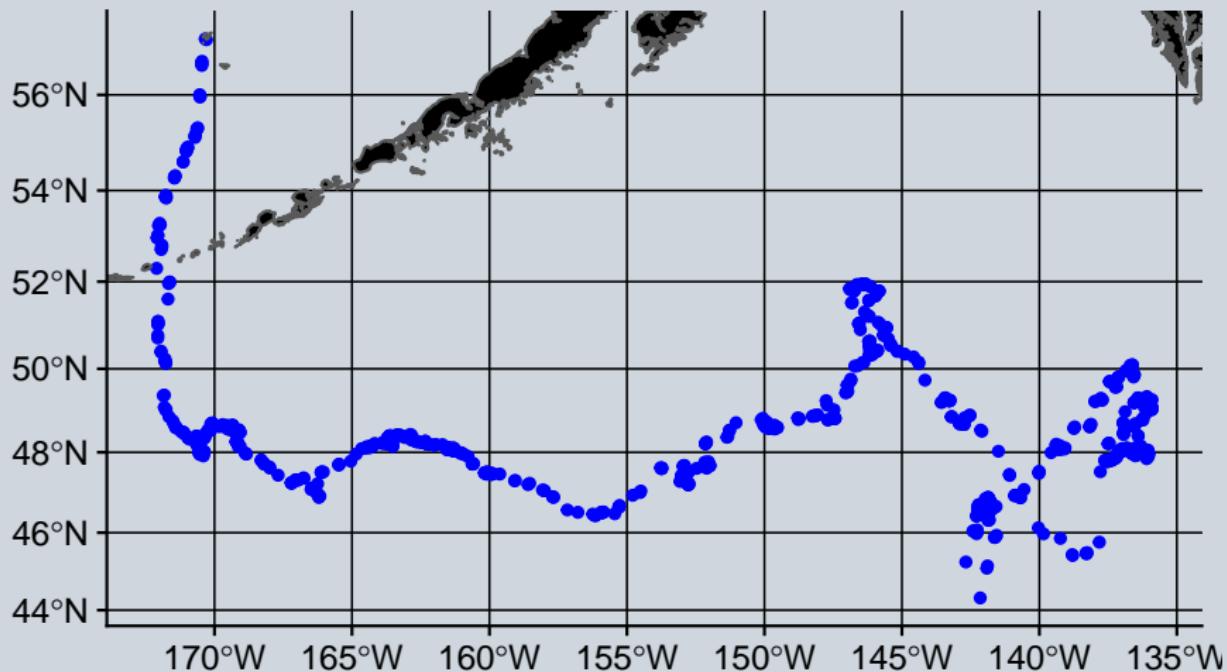


The data: pup 355

```
library(crawl); data("northernFurSeal")
head(northernFurSeal)

##                                     GMT      long     lat loc_class
## 1 2005-11-10 14:52:00 -170.292 57.109            3
## 2 2005-11-10 14:59:00 -170.336 57.126            1
## 3 2005-11-11 01:17:00 -170.452 56.679            2
## 4 2005-11-11 02:02:00 -170.474 56.645            1
## 5 2005-11-11 02:28:00 -170.454 56.632            1
## 6 2005-11-11 03:53:00 -170.460 56.597            1
```

The data: pup 355



Stack data for fitting

```
head(nfs_stack)
```

hour	loc	quality	coord	
1	314342.9	4420274	3	X
2	314343.0	4415376	1	X
3	314353.3	4402463	2	X
4	314354.0	4400014	1	X
5	314354.5	4402241	1	X
6	314355.9	4401573	1	X

>

Notice the coords are projected now

Model location error

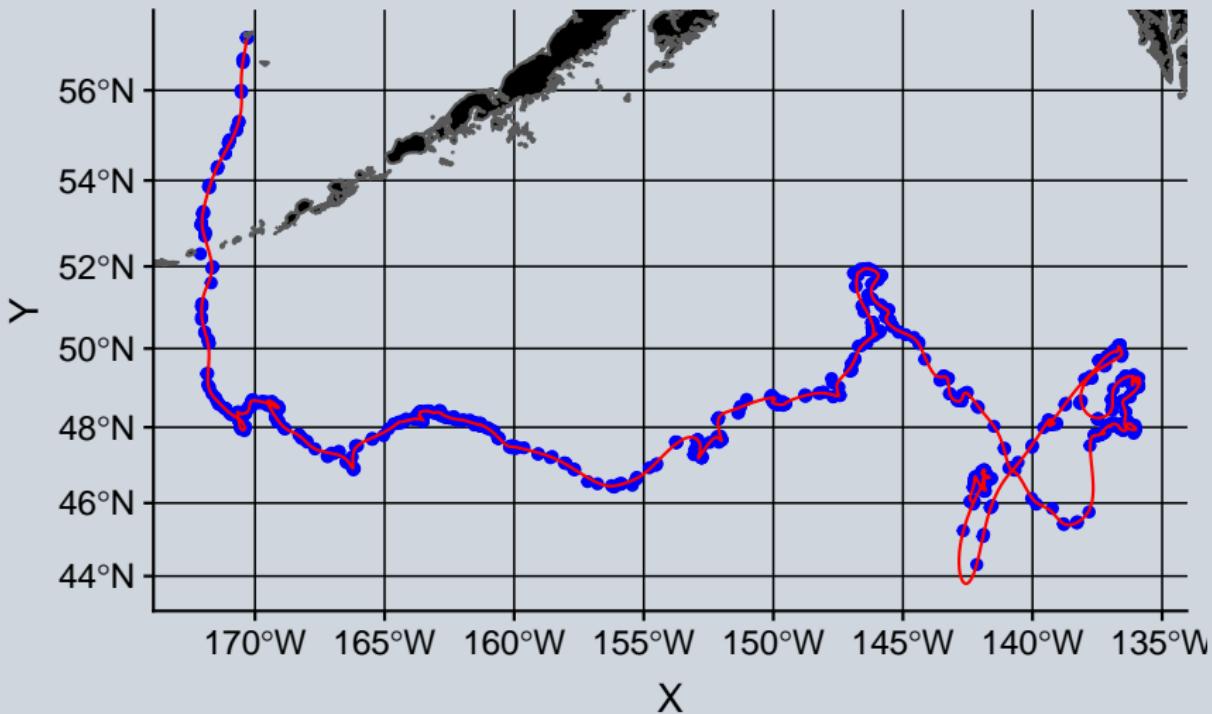
$\text{var}(\epsilon) = \exp(2\gamma_g v)$ where 'g' indexes a group and 'v' is a covariate

```
## define Argos error variance form
nfs_stack$ones <- 1
# necessary to trick the variance function
fix <- c('3'=log(150), '1'=log(500), '2'=log(250))
var_func <- varExp(
  fixed = fix,
  form = formula(~ones|quality)
)
```

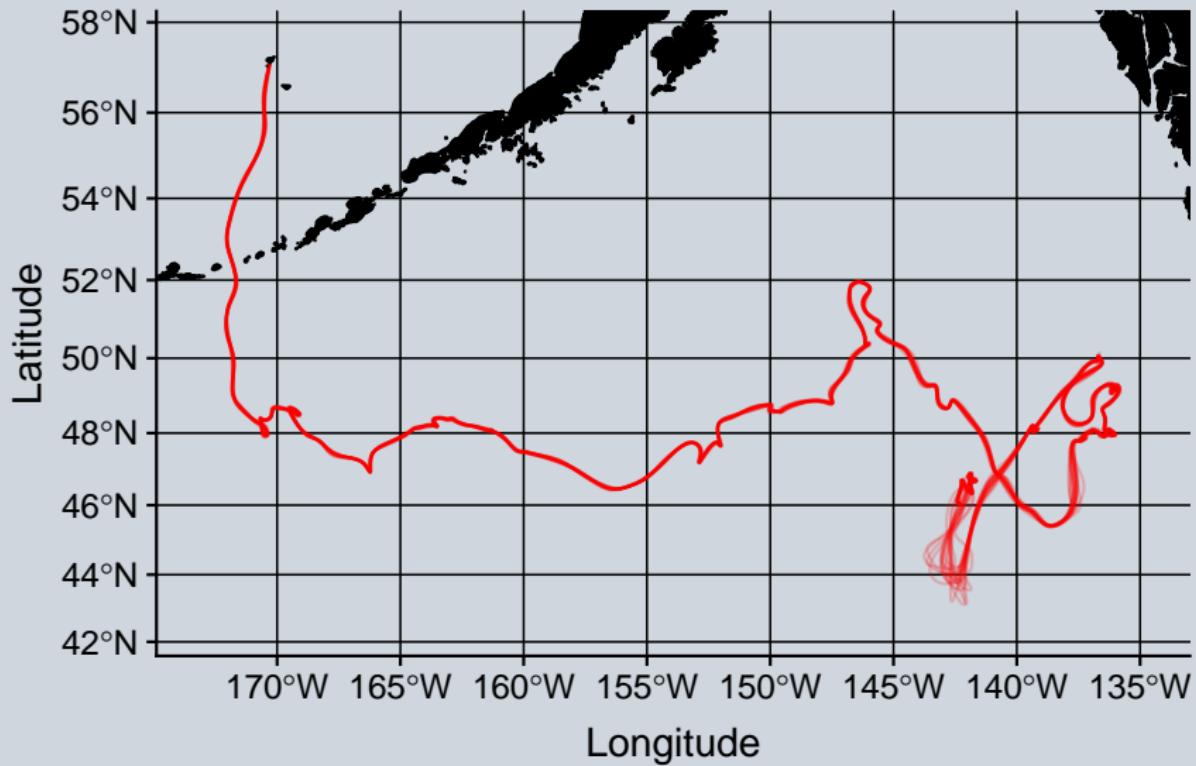
Fit model and sample from posterior paths

```
fit = mgcv::gamm(  
  loc ~ 0 + coord +  
    s(hour, by=coord, k=200, bs=c("ps","ps")),  
  weights = var_func,  
  data=nfs_stack, method="REML")  
### Make some simulated paths  
B = predict(  
  fit[["gam"]], newdata=newdata, type="lpmatrix")  
beta = coef(fit$gam)  
V = vcov(fit$gam, unconditional = T)  
beta_smp = t(rmvnorm(20, beta, V))  
paths <- foreach(i=1:20)%do%{  
  p <- B %*% beta_smp[,i] %>%  
    matrix(., nrow=nrow(newdata)/2, ncol=2)  
  p  
}
```

Predicted path



Path posterior samples



Markov random fields

Modeling functions discretely

- $f(x)$ only makes sense for, fixed x_1, \dots, x_k
- Because we only have a few neighbors, lets model each location conditioned on it's neighbors

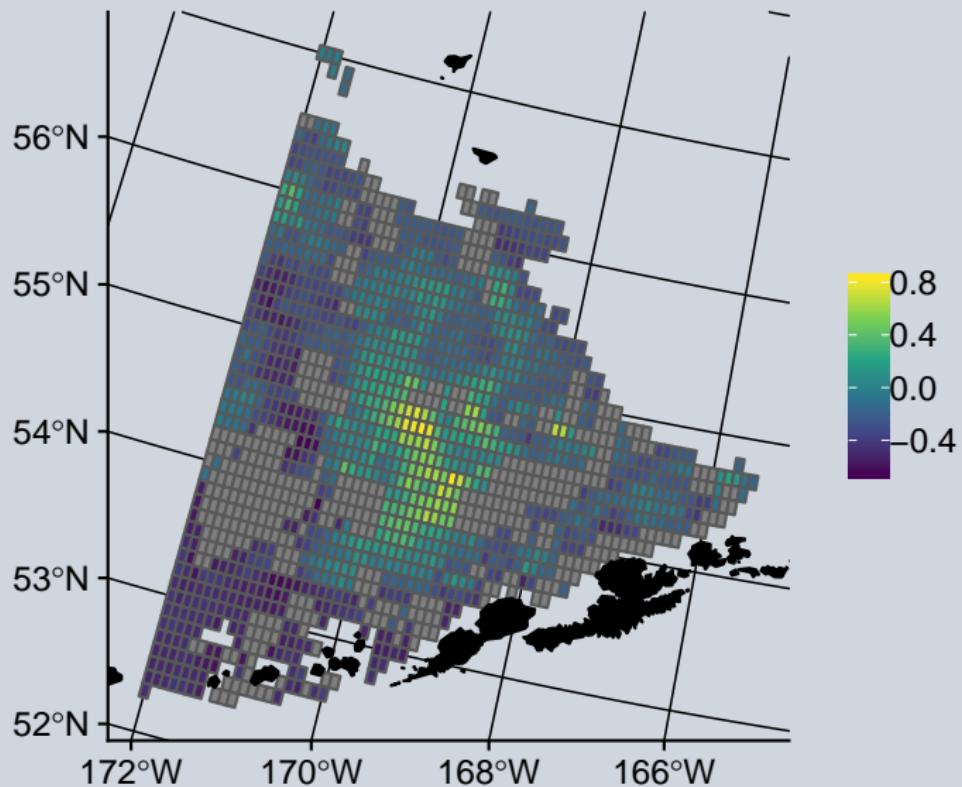
$$f(x_i) \sim N(\bar{f}_i, \xi^2/n_i)$$

- \bar{f}_i mean $f(x_j)$ for all x_i neighbors
- n_i = number of neighbors of x_i
- This is known as the Intrinsic Conditionally Auto(R)egressive model, ICAR

Markov random fields

- MRFs are based on the previous type of neighborhood structure
- Just like GPs, there are many different versions, ICAR is just a very common example
- Models are built on the precision matrix (inverse on the covariance matrix)
- Thus they can be very computationally efficient because there is no need to invert the covariance matrix in the likelihood

Remote sensed CHI data

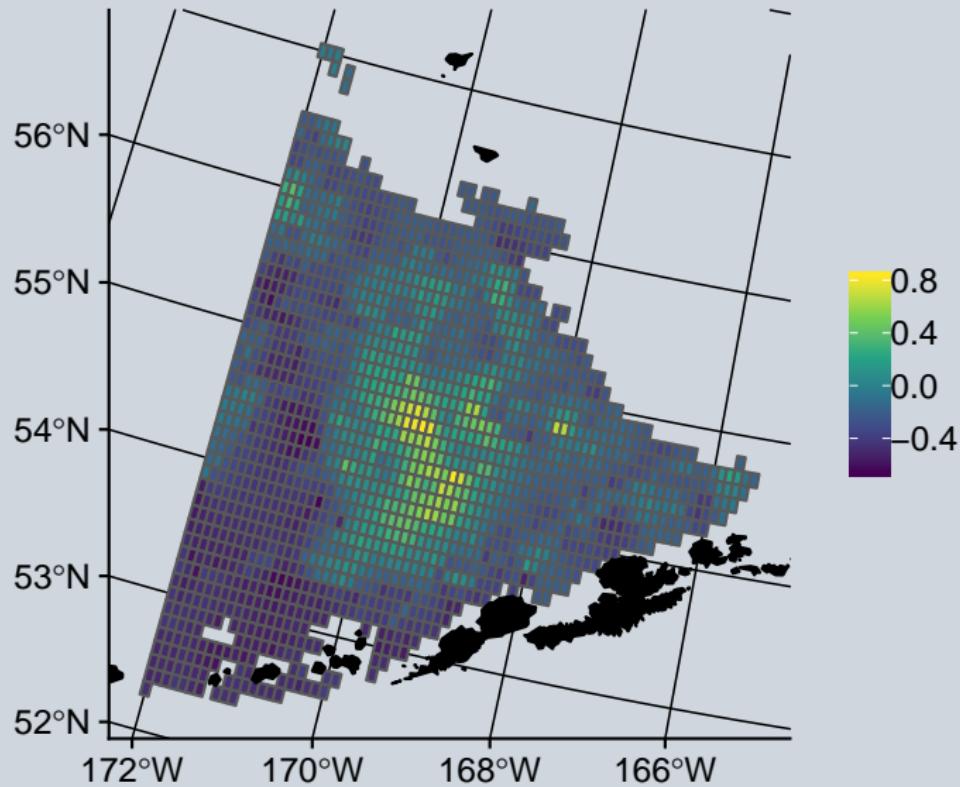


mgcv, not the best choice for MRFs

```
nb <- chl %>% spdep::poly2nb() %>%
  `names<-`(`chl$cell`) %>% `class<-`("list")

chl <- chl %>%
  mutate(
    cell = factor(cell),
    weight = ifelse(is.na(log10chl), 0, 1),
    log10chl = ifelse(is.na(log10chl),
                      mean(log10chl, na.rm=T), log10chl)
  )
### Fit a GMRF model
fit <- mgcv:::gam(log10chl ~ s(cell, bs="mrf", xt=list(nb=nb)),
                    data=chl, weights=weight)
### Predict missing values
chl <- bind_cols(chl, predict(fit, se.fit=T) %>%
  as.data.frame())
```

CHI prediction





The End