Single Species Example from Gulf of Alaska RACE Bottom Trawl Survey

Dr. Curry J. Cunningham

January 23, 2018

Contents

1 Purpose

The purpose of this document is to describe how to generate a model-based index of abundance unsing the spatio-temporal delta-GLMM in the VAST package.

Specifics of this Example:

- Uses RACE bottom trawl survey data.
  - Data are available from the /data folder
- Single species implementation.
- Gulf of Alaska survey data.

---

2 Setup

2.1 Install required packages

```
devtools::install_github("nwfsc-assess/geostatistical_delta-GLMM")
devtools::install_github("james-thorson/VAST")
devtools::install_github("james-thorson/utilities")
```

2.2 Load required packages

```
require(dplyr)
require(VAST)
require(TMB)
require(FishData)
# require(tidyverse)
```

2.3 Setup model

2.3.1 Define species of interest (based on species code) and survey name.

Species are selected by defining the vector `species.codes` in combination with the `combineSpecies` variable. While most species will have a single species code, there are some examples (i.e. GOA Dusky Rockfish) that require multiple species codes to be combined for a single species index. In this later case `combineSpecies = FALSE` would be specified.

Here are some examples to choose from:

number	name	species.code	include	survey	Region
1	Pacific ocean perch	30060	Y	GOA	Gulf_of_Alaska
2	Pacific ocean perch	30060	Y	AI	Aleutian_Islands
3	Walleye pollock	21740	Y	GOA	Gulf_of_Alaska
4	Walleye pollock	21740	Y	AI	Aleutian_Islands
5	Pacific cod	21720	Y	GOA	Gulf_of_Alaska
6	Pacific cod	21720	N	EBS_SHELF	Eastern_Bering_Sea
7	Pacific cod	21720	Y	AI	Aleutian_Islands
8	Northern rockfish	30420	Y	GOA	Gulf_of_Alaska

number	name	species.code	include	survey	Region
9	Northern rockfish	30420	Y	AI	Aleutian_Islands
10	Dover sole	10180	Y	GOA	Gulf_of_Alaska
11	Big skate	420	Y	GOA	Gulf_of_Alaska
12	Atka mackerel	21921	Y	AI	Aleutian_Islands
13	Harlequin rockfish	30535	Y	GOA	Gulf_of_Alaska
14	Arrowtooth flounder	10110	Y	GOA	Gulf_of_Alaska
15	Arrowtooth flounder	10110	Y	EBS_SHELF	Eastern_Bering_Sea
16	Spiny dogfish	310	Y	GOA	Gulf_of_Alaska

Example: Northern Rockfish in the Gulf of Alaska

```
species.codes = c(30420)
survey = "GOA"
```

Other `survey` variable specifications include the Eastern Bering Sea shelf survey `"EBS_SHELF"` and the Aleutian Islands survey `"AI"`.

If multiple `species.codes` need to be comined into single index, set `combineSpecies=TRUE`

```
combineSpecies = FALSE
```

Next, we will define the `Region`, for spatial extrapolation.

```
if(survey=="GOA") { Region = 'Gulf_of_Alaska' }
if(survey=="EBS_SHELF") { Region = "Eastern_Bering_Sea" }
if(survey=="AI") { Region = "Aleutian Islands" }
```

2.3.2 Observation reference location settings

Define what location reading from survey tow will be used Here we identify whether our spatial reference (lat/lon) for the observation will be the `"start"` or `"end"` of the survey tow.

- Note: `lat_lon.def = "start"` is usually best as in rare instances `"end"` location is not recorded.

```
lat_lon.def = "start"
```

2.3.3 Spatial settings

The following settings define the spatial resolution for the model (defined by number of knots `n_x`), and whether to use a grid or mesh approximation through the `Method` variable.

```
Method = c("Grid", "Mesh", "Spherical_mesh")[2]
grid_size_km = 25
n_x = c(100, 250, 500, 1000, 2000)[1]
Kmeans_Config = list( "randomseed"=1, "nstart"=100, "iter.max"=1e3 )
```

2.3.4 Define strata limits

Here we can define the latitude and longitude designations for strata, if strata-specific indices are desired. We will not stratify in this example.

```
#Basic - Single Area
strata.limits = data.frame(STRATA = c("All_areas"))
```

2.3.5 VAST version settings

Define which version of VAST you will be using, i.e. which version of CPP code will be referenced for the TMB model.

```
Version = "VAST_v4_0_0"
```

2.3.6 Model settings

Bias correction

Define whether to implement epsilon bias correction estimator for nonlinear transformation of random effects, through the `bias.correct` variable. See Thorson and Kristensen (2016)

*Note: Bias correction is computationally intensive, especially for models with high spatial complexity i.e. high `n_x`.

```
bias.correct = FALSE
```

Spatio-temporal variation, autocorrelation, and overdispersion

The following settings define whether to include spatial and spatio-temporal variation (`FieldConfig`), whether its autocorre- lated (`RhoConfig`), and whether there's overdispersion (`OverdispersionConfig`). In `FieldConfig`, `Omega1` and `Omega2` are `ON/OFF = 1/0` switches for spatial random effects in the **(1)** positive catch rate and **(2)** encounter probability components of the delta model. `Epsilon1` and `Epsilon2` are ON/OFF switches for the spatio-temporal random effects. In `RhoConfig`, `Beta1` and `Beta2` are autocorrelation specifications for intercepts, while `Epsilon1` and `Epsilon2` are the same specifications for spatio-temporal random effects, for **(1)** positive catch rate and **(2)** encounter probability components of the delta model.

```
FieldConfig = c(Omega1 = 1, Epsilon1 = 1, Omega2 = 1, Epsilon2 = 1)
RhoConfig  = c(Beta1 = 0, Beta2 = 0, Epsilon1 = 0, Epsilon2 = 0)
OverdispersionConfig  = c(Delta1 = 0, Delta2 = 0)
```

*See documentation for `Data_Fn()` within VAST for specification of `RhoConfig` and `OverdispersionConfig`.

```
?VAST::Data_Fn
```

2.3.7 Observation model settings

The `ObsModel` vector is used to specify the assumed observation model, where **first** element specifies the distribution for **positive catch rates** and **second** element specifies the functional form for **encounter probabilities**. Here we specify the conventional delta-model using logit-link for encounter probability and log-link for positive catch rates.

```
ObsModel = c(1,0)
```

Alternatives:

`ObsModel` **Specification**	Distribution for **Positive Catch Rates**
`ObsModel[1]=0`	Normal
`ObsModel[1]=1`	Lognormal
`ObsModel[1]=2`	Gamma
`ObsModel[1]=5`	Negative binomial
`ObsModel[1]=6`	Conway-Maxwell-Poisson (likely to be very slow)
`ObsModel[1]=7`	Poisson (more numerically stable than negative-binomial)

`ObsModel` **Specification**	Distribution for **Positive Catch Rates**
`ObsModel[1]=8`	Compound-Poisson-Gamma, where the expected number of individuals is the 1st-component, the expected biomass per individual is the 2nd-component, and SigmaM is the variance in positive catches (likely to be very slow)
`ObsModel[1]=9`	Binned-Poisson (for use with REEF data, where 0=0 individual; 1=1 individual; 2=2:10 individuals; 3=>10 individuals)
`ObsModel[1]=10`	Tweedie distribution, where epected biomass (lambda) is the product of 1st-component and 2nd-component, variance scalar (phi) is the 1st component, and logis-SigmaM is the power

*See documentation for `Data_Fn()` within VAST for specification of `ObsModel`.

2.3.8 Save settings

DateFile is the folder that will hold my model outputs.

```
DateFile = paste0(getwd(), "/VAST_output/")

#Create directory
dir.create(DateFile, recursive=TRUE)
```

2.4 Specify model outputs

The following settings define what types of output we want to calculate.

```
Options = c(SD_site_density = 0, SD_site_logdensity = 0,
            Calculate_Range = 1, Calculate_evenness = 0, Calculate_effective_area = 1,
            Calculate_Cov_SE = 0, Calculate_Synchrony = 0,
            Calculate_Coherence = 0)
```

---

3 Prepare the data

- Note: This section can be replace by function `create_VAST_input()`, from `R/create-VAST-input.r`

3.1 Load RACE data

To create the input data files for VAST model, first we must load RACE survey data. Two data files are necessary **(1)** catch data and **(2)** haul data.

3.1.1 Load data

Catch data

```
catch = readRDS("data/race_base_catch.rds")
```

Haul data

```
haul = readRDS("data/race_base_haul.rds")
```

Limit haul dataset to only abundance hauls

```
haul = haul[haul$ABUNDANCE_HAUL=='Y',]
```

3.1.2 Join datasets

We need to join haul information to catch data, creating list `catchhaul`.

```
catchhaul = right_join(x=catch, y=haul, by=c("HAULJOIN"))
```

3.1.3 Add in zero observations for catch weight, for no catches.

Drawing on Jim Thorson's code from FishData

- Note: `species.codes` are now treated as a factor.

```
catchhaul.2 = FishData::add_missing_zeros(data_frame=catchhaul, unique_sample_ID_colname="HAULJOIN",
                                          sample_colname="WEIGHT", species_colname="SPECIES_CODE",
                                          species_subset=species.codes,
                                          if_multiple_records="First",
                                          Method="Fast")
```

3.1.4 Attach cruise info

Cruise info is necessary to add year of survey and name ####Load cruise info

```
cruise.info = read.csv("data/race_cruise_info.csv", header=TRUE, stringsAsFactors=FALSE)
```

3.1.4.1 Attach cruise info

We use an inner_join because it removes hauls WITHOUT identified Year and Survey

```
catchhaul.3 = inner_join(x=catchhaul.2,
                         y=cruise.info[,c("Cruise.Join.ID","Year","Survey")],
                         by=c("CRUISEJOIN.x"="Cruise.Join.ID"))
```

3.1.4.2 Limit to survey of interest

We need to limit our dataset to only the survey of interest

```
catchhaul.3 = catchhaul.3[catchhaul.3$Survey==survey,]
```

3.1.4.3 Aggregate multiple species.codes

If we are combining multiple species into a single index, we need add these data together.

```
if(combineSpecies==TRUE) {
  catchhaul.4 = data.frame(catchhaul.3 %>% group_by(HAULJOIN) %>%
                             mutate('WEIGHT'=sum(WEIGHT, na.rm=TRUE)))
  #Since we have aggregated, only retain rows for 1st listed species code
  catchhaul.5 = catchhaul.4[catchhaul.4$SPECIES_CODE==species.codes[1],]
}else {
  catchhaul.5 = catchhaul.3
}
```

3.2 Standardize data

In order to standardize the survey catch data, we must calculate effort as area swept per tow.

3.2.1 Calculate effort

Input effort is in $ kilometers^2 $

```
catchhaul.5$effort = catchhaul.5$NET_WIDTH*catchhaul.5$DISTANCE_FISHED/1000
```

3.3 Add species names

First, we load the list describing both species names and `species.codes`

```
species.code.data = read.csv("data/race_species_codes.csv",
                             header=TRUE, stringsAsFactors=FALSE)
```

Next, join this to our overall survey data list

```
load.data = merge(x=catchhaul.5, y=species.code.data[,c("Species.Code","Common.Name")],
                  by.x="SPECIES_CODE", by.y="Species.Code")
```

3.4 Build Data_Geostat

Now, we will create the list `Data_Geostat` which is the input for the VAST model.

```
Data_Geostat = NULL
```

3.4.1 Add elements to Data_Geostat list

If you ar running for multiple species add the species name.

```
if(length(species.codes) > 1) {
  Data_Geostat$spp = load.data$Common.Name
}
Data_Geostat$Catch_KG = as.numeric(load.data$WEIGHT)
Data_Geostat$Year = as.integer(load.data$Year)
Data_Geostat$Vessel = "missing"
Data_Geostat$AreaSwept_km2 = as.numeric(load.data$effort)
Data_Geostat$Pass = 0
```

3.4.2 Define location of samples

Depending on `lat_lon.def` specification we will either use the **start**, **end**, or **mean** location recorded for a survey haul.

- Note: Using the starting location of each haul is probably best, as: `lat_lon.def="start"`.

```
if(lat_lon.def=="start") {
  Data_Geostat$Lat = load.data$START_LATITUDE
  Data_Geostat$Lon = load.data$START_LONGITUDE
}
if(lat_lon.def=="end") {
  Data_Geostat$Lat = load.data$END_LATITUDE
  Data_Geostat$Lon = load.data$END_LONGITUDE
}
if(lat_lon.def=="mean") {
  Data_Geostat$Lat = rowMeans(cbind(load.data$START_LATITUDE,
                                     load.data$END_LATITUDE), na.rm=TRUE)

  Data_Geostat$Lon = rowMeans(cbind(load.data$START_LONGITUDE,
                                     load.data$END_LONGITUDE), na.rm=TRUE)
}
```

Finally, we must ensure this `Data_Geostat` is a proper data frame.

```
Data_Geostat = data.frame(Data_Geostat)
```

To double check lets see how `Data_Geostat` looks…

```
kable(head(Data_Geostat))
```

Catch_KG	Year	Vessel	AreaSwept_km2	Pass	Lat	Lon
0.00	2005	missing	0.028	0	52.6	-170
40.32	2005	missing	0.020	0	52.6	-170
3.84	2005	missing	0.024	0	52.7	-169
0.00	2005	missing	0.021	0	53.2	-168
0.00	2005	missing	0.019	0	53.2	-168
1.05	2005	missing	0.021	0	53.1	-168

3.5 Create the extrapolation grid

We also generate the extrapolation grid appropriate for a given region. For new regions, we use Region="Other".

- Note: We are not defining strata limits, but could do so based on latitude and longitude definitions.

```
Extrapolation_List = SpatialDeltaGLMM::Prepare_Extrapolation_Data_Fn(Region = Region,
    strata.limits = strata.limits)
```

3.6 Create spatial list

Next, generate the information used for conducting spatio-temporal parameter estimation, bundled in list `Spatial_List`.

```r
Spatial_List = SpatialDeltaGLMM::Spatial_Information_Fn(grid_size_km = grid_size_km,
    n_x = n_x, Method = Method, Lon = Data_Geostat[,
        "Lon"], Lat = Data_Geostat[, "Lat"], Extrapolation_List = Extrapolation_List,
    randomseed = Kmeans_Config[["randomseed"]], nstart = Kmeans_Config[["nstart"]],
    iter.max = Kmeans_Config[["iter.max"]], DirPath = DateFile,
    Save_Results = TRUE)
```

3.7 Update `Data_Geostat` with knot references

We then associate each of our haul observations with its appropriate knot.

```r
Data_Geostat = cbind(Data_Geostat, knot_i = Spatial_List$knot_i)
```

---

4 Build and run model

4.1 Build model

To estimate parameters, we first build a list of data-inputs used for parameter estimation. Data_Fn has some simple checks for buggy inputs, but also please read the help file ?Data_Fn.

```r
if (length(species.codes) > 1 & combineSpecies == FALSE) {
    # MULTISPECIES
    TmbData = VAST::Data_Fn(Version = Version, FieldConfig = FieldConfig,
        OverdispersionConfig = OverdispersionConfig,
        RhoConfig = RhoConfig, ObsModel = ObsModel,
        c_i = as.numeric(Data_Geostat[, "spp"]) - 1,
        b_i = Data_Geostat[, "Catch_KG"], a_i = Data_Geostat[,
            "AreaSwept_km2"], v_i = as.numeric(Data_Geostat[,
            "Vessel"]) - 1, s_i = Data_Geostat[, "knot_i"] -
            1, t_i = Data_Geostat[, "Year"], a_xl = Spatial_List$a_xl,
        MeshList = Spatial_List$MeshList, GridList = Spatial_List$GridList,
        Method = Spatial_List$Method, Options = Options)
} else {
    # SINGLE SPECIES
    TmbData = VAST::Data_Fn(Version = Version, FieldConfig = FieldConfig,
        OverdispersionConfig = OverdispersionConfig,
        RhoConfig = RhoConfig, ObsModel = ObsModel,
        c_i = rep(0, nrow(Data_Geostat)), b_i = Data_Geostat[,
            "Catch_KG"], a_i = Data_Geostat[, "AreaSwept_km2"],
        v_i = as.numeric(Data_Geostat[, "Vessel"]) -
            1, s_i = Data_Geostat[, "knot_i"] - 1,
        t_i = Data_Geostat[, "Year"], a_xl = Spatial_List$a_xl,
        MeshList = Spatial_List$MeshList, GridList = Spatial_List$GridList,
        Method = Spatial_List$Method, Options = Options)
}
```

```
##   Omega1 Epsilon1   Omega2 Epsilon2
##        1        1        1        1
## Delta1 Delta2
##     -1     -1
```

4.2 Build TMB object

Next, we build and compile the TMB object for estimation.

- Note: Compilation may take some time… **be patient**.

```
TmbList = VAST::Build_TMB_Fn(TmbData = TmbData, RunDir = DateFile,
    Version = Version, RhoConfig = RhoConfig, loc_x = Spatial_List$loc_x,
    Method = Method)
Obj = TmbList[["Obj"]]
```

4.2.1 Do estimation

Fit VAST model to the data by optimizing the TMB function.

```
Opt = TMBhelper::Optimize(obj = Obj, lower = TmbList[["Lower"]],
                          upper = TmbList[["Upper"]], getsd = TRUE, savedir = DateFile,
                          bias.correct = bias.correct)
```

4.2.2 Save output

Save outputs from estimation

```
Report = Obj$report()

Save = list("Opt"=Opt, "Report"=Report, "ParHat"=Obj$env$parList(Opt$par),
            "TmbData"=TmbData)

save(Save, file=paste0(DateFile,"Save.RData"))
```

5 Diagnostic plots

We first apply a set of standard model diagnostics to confirm that the model is reasonable and deserves further attention. If any of these do not look reasonable, the model output should not be interpreted or used.

5.1 Plot data

It is always good practice to conduct exploratory analysis of data. Here, I visualize the spatial distribution of data. Spatio-temporal models involve the assumption that the probability of sampling a given location is statistically independent of the probability distribution for the response at that location. So if sampling "follows" changes in density, then the model is probably not appropriate!

```
SpatialDeltaGLMM::Plot_data_and_knots(Extrapolation_List = Extrapolation_List,
    Spatial_List = Spatial_List, Data_Geostat = Data_Geostat,
    PlotDir = DateFile)
```

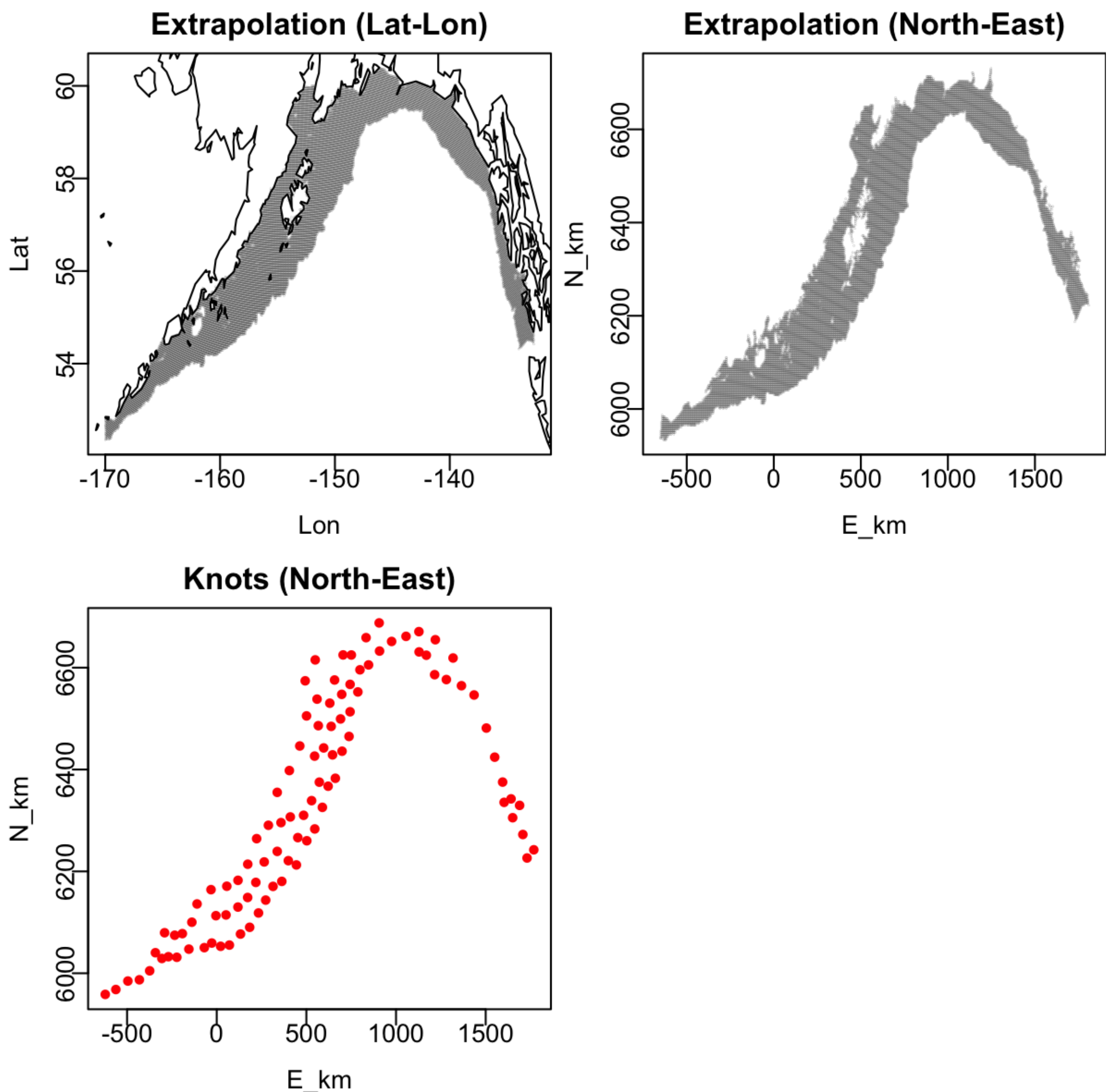


Figure 1: Spatial extent and location of knots

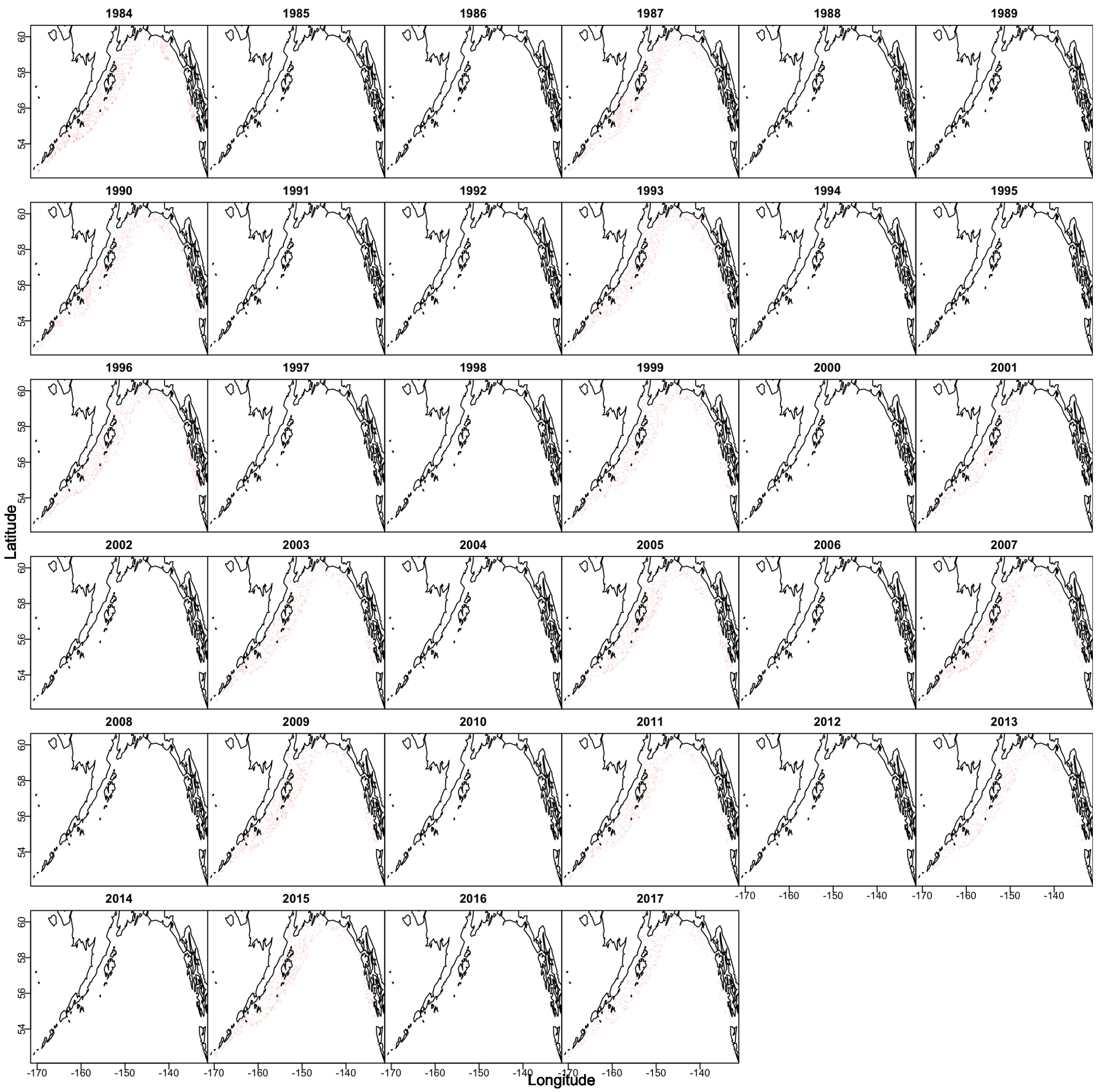


Figure 2: Spatial distribution of catch-rate data

5.2 Convergence

Here we print the diagnostics generated during parameter estimation, and confirm that (1) no parameter is hitting an upper or lower bound and (2) the final gradient for each fixed-effect is close to zero. For explanation of parameters, please see `?Data_Fn`.

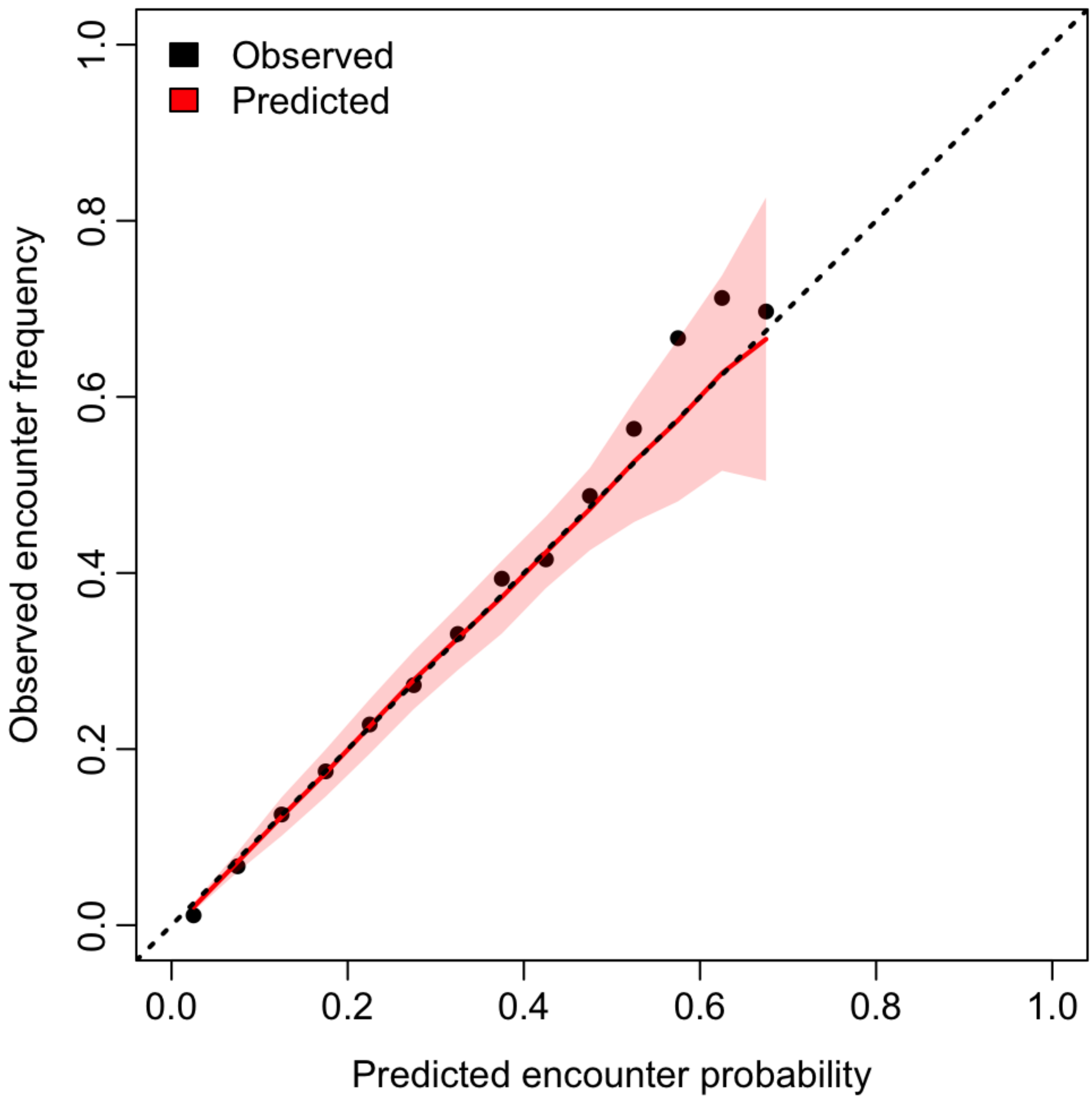
```
pander::pandoc.table( Opt$diagnostics[,c('Param','Lower','MLE','Upper','final_gradient')] )
```

Param	Lower	MLE	Upper	final_gradient
ln_H_input	-50	0.3012	50	1.061e-05
ln_H_input	-50	0.3485	50	-1.816e-05
beta1_ct	-50	-3.236	50	0.000125
beta1_ct	-50	-2.37	50	-0.0001177
beta1_ct	-50	-2.692	50	0.0001576
beta1_ct	-50	-2.681	50	3.22e-06
beta1_ct	-50	-2.837	50	4.197e-05
beta1_ct	-50	-3.023	50	0.0001386
beta1_ct	-50	-2.939	50	0.0001299
beta1_ct	-50	-3.122	50	9.499e-05
beta1_ct	-50	-2.933	50	-0.0001939
beta1_ct	-50	-3.107	50	2e-04
beta1_ct	-50	-3.182	50	0.0001528
beta1_ct	-50	-3.399	50	-2.718e-06
beta1_ct	-50	-3.244	50	-4.472e-05
beta1_ct	-50	-3.515	50	2.056e-05
beta1_ct	-50	-3.014	50	-8.154e-05
L_omega1_z	-50	-2.133	50	2.552e-05
L_epsilon1_z	-50	-0.4589	50	-0.0004237
logkappa1	-6.746	-4.8	-2.552	-5.14e-05
beta2_ct	-50	5.757	50	3.324e-05
beta2_ct	-50	5.997	50	1.477e-05
beta2_ct	-50	6.269	50	8.916e-05
beta2_ct	-50	6.192	50	0.0001281
beta2_ct	-50	6.862	50	-2.133e-05
beta2_ct	-50	6.415	50	8.467e-05
beta2_ct	-50	6.819	50	-0.0002633
beta2_ct	-50	6.289	50	-8.712e-05
beta2_ct	-50	6.992	50	-6.196e-05
beta2_ct	-50	6.912	50	7.674e-05
beta2_ct	-50	6.316	50	1.278e-05
beta2_ct	-50	6.784	50	-9.042e-05
beta2_ct	-50	7.152	50	-4.315e-05
beta2_ct	-50	6.604	50	-0.0001142
beta2_ct	-50	6.861	50	-0.00017
L_omega2_z	-50	-1.565	50	0.0001609
L_epsilon2_z	-50	-1.319e-07	50	-7.755e-06
logkappa2	-6.746	-4.823	-2.552	0.0002792
logSigmaM	-50	0.7437	10	0.002941

5.3 Diagnostics for encounter-probability component

Next, we check whether observed encounter frequencies for either low or high probability samples are within the 95% predictive interval for predicted encounter probability

```
Enc_prob = SpatialDeltaGLMM::Check_encounter_prob(Report = Report,
    Data_Geostat = Data_Geostat, DirName = DateFile)
```



5.4 Diagnostics for positive-catch-rate component

We can visualize fit to residuals of catch-rates given encounters using a Q-Q plot. A good Q-Q plot will have residuals along the one-to-one line.

- Note: In this plot, the red line should fall along the 1:1 line.

```
Q = SpatialDeltaGLMM::QQ_Fn(TmbData = TmbData, Report = Report,
    FileName_PP = paste0(DateFile, "Posterior_Predictive.jpg"),
    FileName_Phist = paste0(DateFile, "Posterior_Predictive-Histogram.jpg"),
    FileName_QQ = paste0(DateFile, "Q-Q_plot.jpg"),
    FileName_Qhist = paste0(DateFile, "Q-Q_hist.jpg"))
```

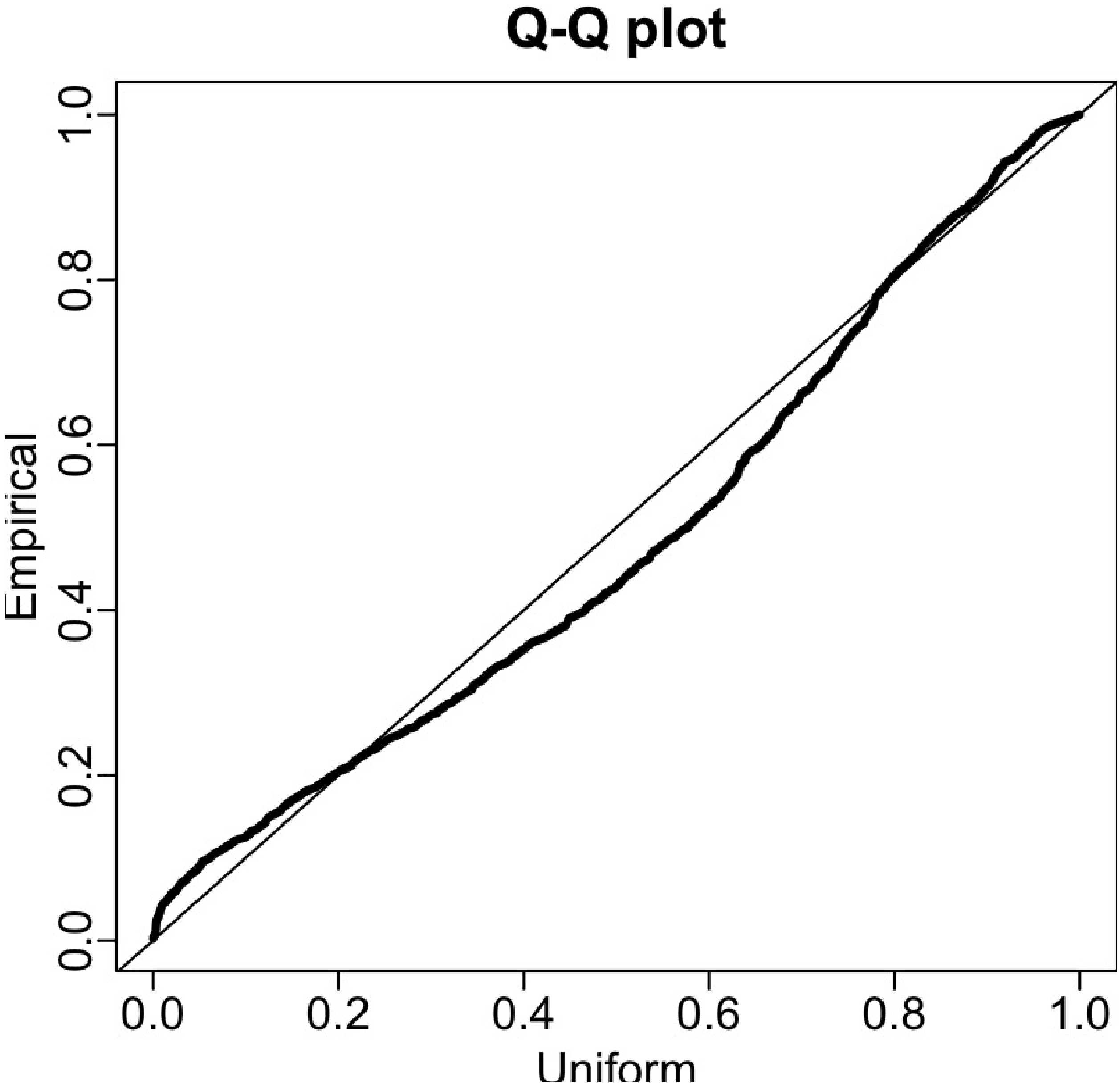


Figure 3: Quantile-quantile plot indicating residuals for “positive catch rate” component

5.5 Diagnostics for plotting residuals on a map

Finally, we visualize residuals on a map. To do so, we first define years to plot and generate plotting inputs. useful plots by first determining which years to plot (`Years2Include`), and labels for each plotted year (`Year_Set`). Some general insights:

- `MapDetails_Fn` - Tries to automatically detect size of plots to do, determine what states to plot nearby.
- `Year_Set` - Range of years sampled (bookeeping)
- `Years2Include` - Years with surveys (bookeeping)

```
# Get region-specific settings for plots
MapDetails_List = SpatialDeltaGLMM::MapDetails_Fn( "Region"=Region,
                                                  "NN_Extrap"=Spatial_List$PolygonList$NN_Extrap,
                                                  "Extrapolation_List"=Extrapolation_List )
# Decide which years to plot
Year_Set = seq(min(Data_Geostat[,'Year']),max(Data_Geostat[,'Year']))
Years2Include = which( Year_Set %in% sort(unique(Data_Geostat[,'Year'])))
```

We then plot Pearson residuals. If there are visible patterns (areas with consistently positive or negative residuals accross or within years) then this is an indication of the model "overshrinking" results towards the intercept, and model results should then be treated with caution.

```
SpatialDeltaGLMM:::plot_residuals(Lat_i = Data_Geostat[,
    "Lat"], Lon_i = Data_Geostat[, "Lon"], TmbData = TmbData,
    Report = Report, Q = Q, savedir = DateFile, MappingDetails = MapDetails_List[["MappingDetails"]],
    PlotDF = MapDetails_List[["PlotDF"]], MapSizeRatio = MapDetails_List[["MapSizeRatio"]],
    Xlim = MapDetails_List[["Xlim"]], Ylim = MapDetails_List[["Ylim"]],
    FileName = DateFile, Year_Set = Year_Set, Rotate = MapDetails_List[["Rotate"]],
    Cex = MapDetails_List[["Cex"]], Legend = MapDetails_List[["Legend"]],
    zone = MapDetails_List[["Zone"]], mar = c(0, 0,
        2, 0), oma = c(3.5, 3.5, 0, 0), cex = 1.8)
```

Encounter Probability Residuals

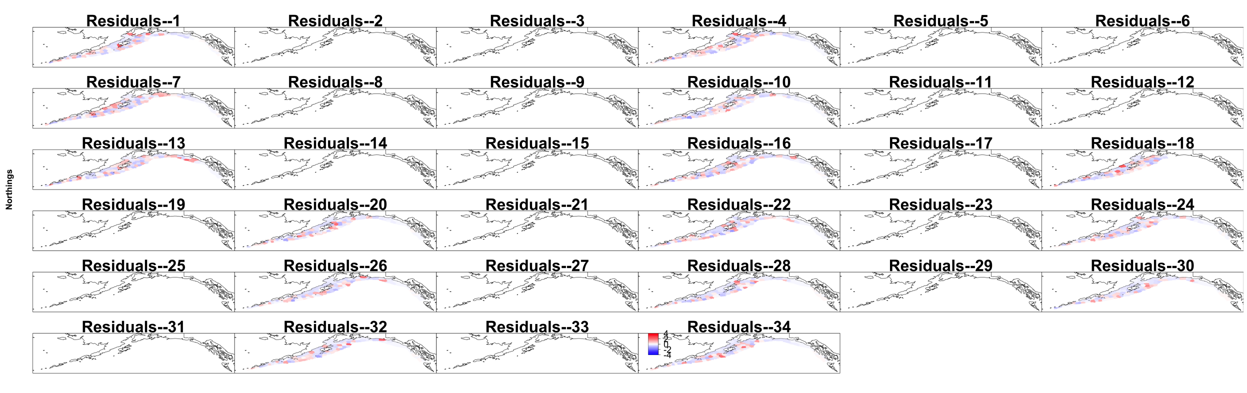


Figure 4: Pearson residuals for encounter-probability by knot

Catch Rate Residuals

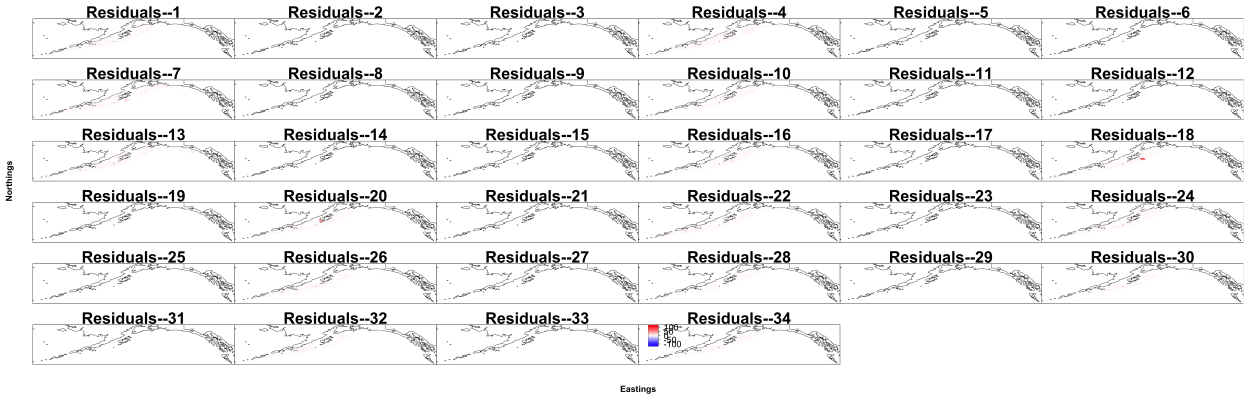


Figure 5: Pearson residuals for catch rate by knot

5.6 Model selection

To select among models, we recommend using the Akaike Information Criterion, AIC, via `Opt$AIC=2.271\times 10^{4}`.

6 Model output

Last but not least, we generate pre-defined plots for visualizing results

6.1 Direction of "geometric anisotropy"

We can visualize which direction has faster or slower decorrelation (termed "geometric anisotropy")

```
SpatialDeltaGLMM::PlotAniso_Fn(FileName = paste0(DateFile,
    "Aniso.png"), Report = Report, TmbData = TmbData)
```

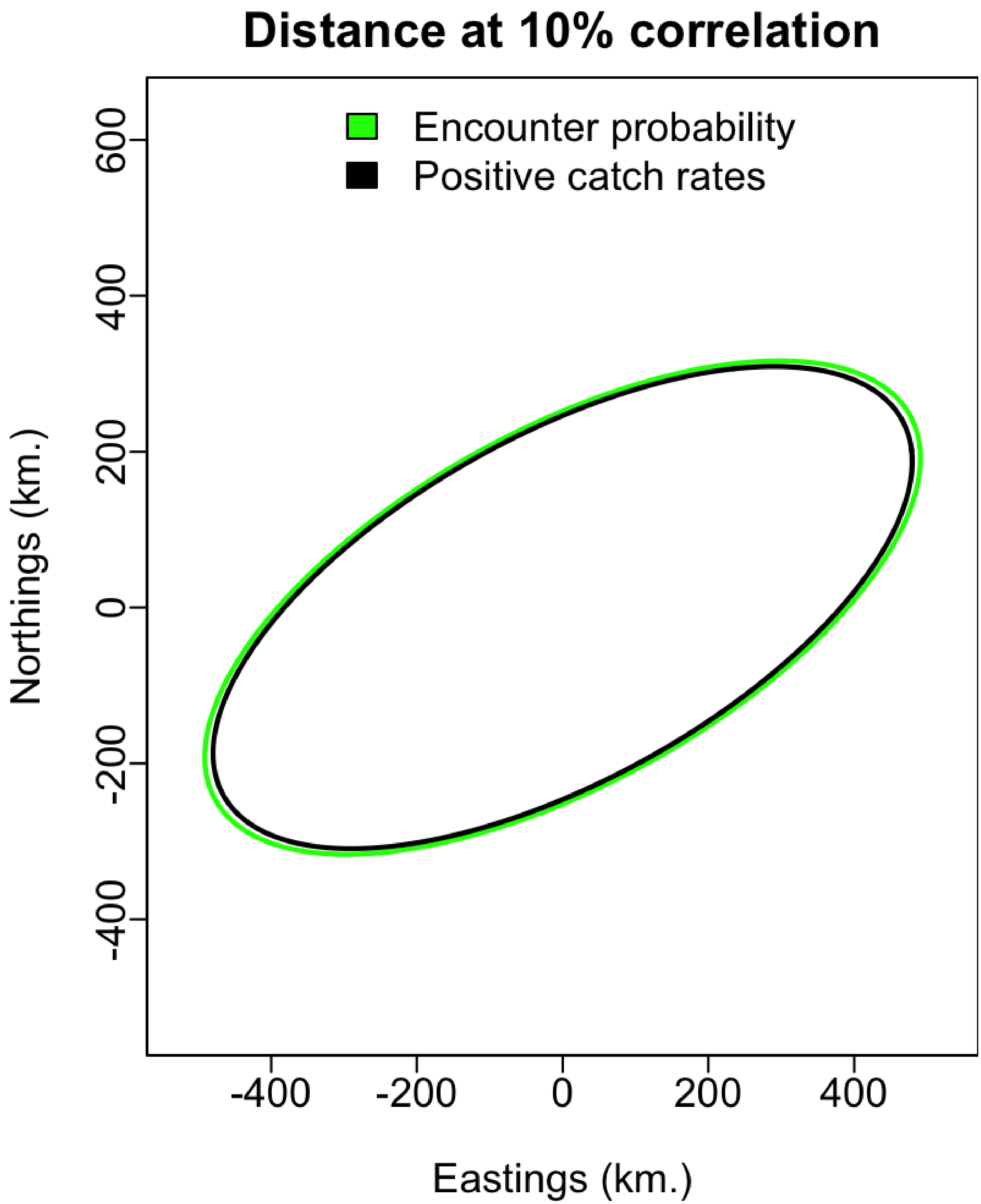


Figure 6: Decorrelation distance for different directions

6.2 Density surface for each year

We can visualize many types of output from the model. Here I only show predicted density, but other options are obtained via other integers passed to `plot_set` as described in `?PlotResultsOnMap_Fn`

```
SpatialDeltaGLMM::PlotResultsOnMap_Fn(plot_set = c(3),
    MappingDetails = MapDetails_List[["MappingDetails"]],
    Report = Report, Sdreport = Opt$SD, PlotDF = MapDetails_List[["PlotDF"]],
    MapSizeRatio = MapDetails_List[["MapSizeRatio"]],
    Xlim = MapDetails_List[["Xlim"]], Ylim = MapDetails_List[["Ylim"]],
    FileName = DateFile, Year_Set = Year_Set, Years2Include = Years2Include,
    Rotate = MapDetails_List[["Rotate"]], Cex = MapDetails_List[["Cex"]],
    Legend = MapDetails_List[["Legend"]], zone = MapDetails_List[["Zone"]],
    mar = c(0, 0, 2, 0), oma = c(3.5, 3.5, 0, 0), cex = 1.8,
    plot_legend_fig = FALSE)
```

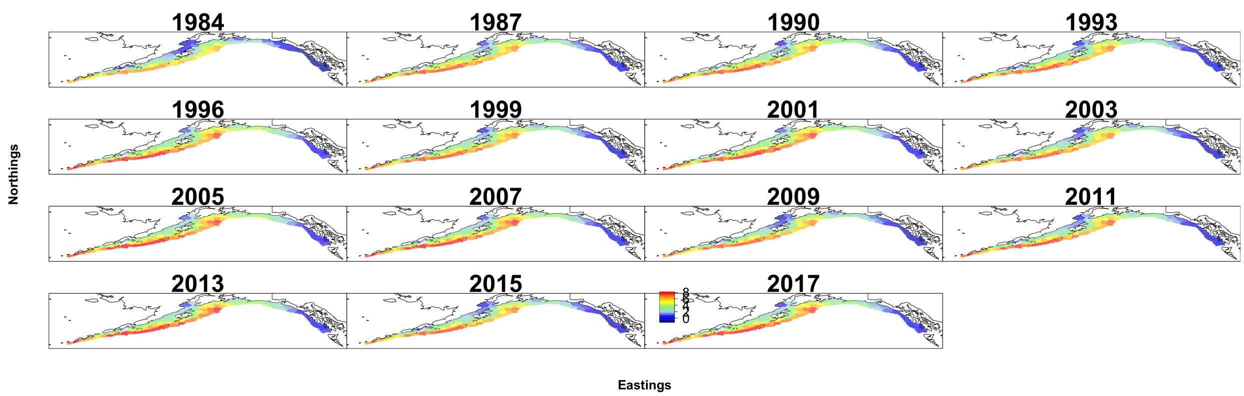


Figure 7: Density maps for each year

6.3 Index of abundance

The index of abundance is generally most useful for stock assessment models.

```
Index = SpatialDeltaGLMM::PlotIndex_Fn(DirName = DateFile,
    TmbData = TmbData, Sdreport = Opt[["SD"]], Year_Set = Year_Set,
    Years2Include = Years2Include, strata_names = strata.limits[,
        1], use_biascorr = TRUE)
pander::pandoc.table(Index$Table[, c("Year", "Fleet",
    "Estimate_metric_tons", "SD_log", "SD_mt")])
```

Year	Fleet	Estimate_metric_tons	SD_log	SD_mt
1984	All_areas	39153	0.1973	7726
1985	All_areas	0	NA	NA
1986	All_areas	0	NA	NA
1987	All_areas	79590	0.1788	14231
1988	All_areas	0	NA	NA
1989	All_areas	0	NA	NA
1990	All_areas	85304	0.2191	18694
1991	All_areas	0	NA	NA
1992	All_areas	0	NA	NA
1993	All_areas	85284	0.2014	17175
1994	All_areas	0	NA	NA
1995	All_areas	0	NA	NA
1996	All_areas	131030	0.2123	27823
1997	All_areas	0	NA	NA
1998	All_areas	0	NA	NA
1999	All_areas	84751	0.2245	19025
2000	All_areas	0	NA	NA
2001	All_areas	129965	0.2367	30758
2002	All_areas	0	NA	NA
2003	All_areas	73774	0.2173	16034
2004	All_areas	0	NA	NA
2005	All_areas	160774	0.204	32793
2006	All_areas	0	NA	NA
2007	All_areas	131814	0.2078	27393
2008	All_areas	0	NA	NA
2009	All_areas	73864	0.2132	15745
2010	All_areas	0	NA	NA
2011	All_areas	97047	0.2555	24794
2012	All_areas	0	NA	NA
2013	All_areas	162460	0.2574	41818
2014	All_areas	0	NA	NA
2015	All_areas	76595	0.2479	18985
2016	All_areas	0	NA	NA
2017	All_areas	138899	0.2489	34567

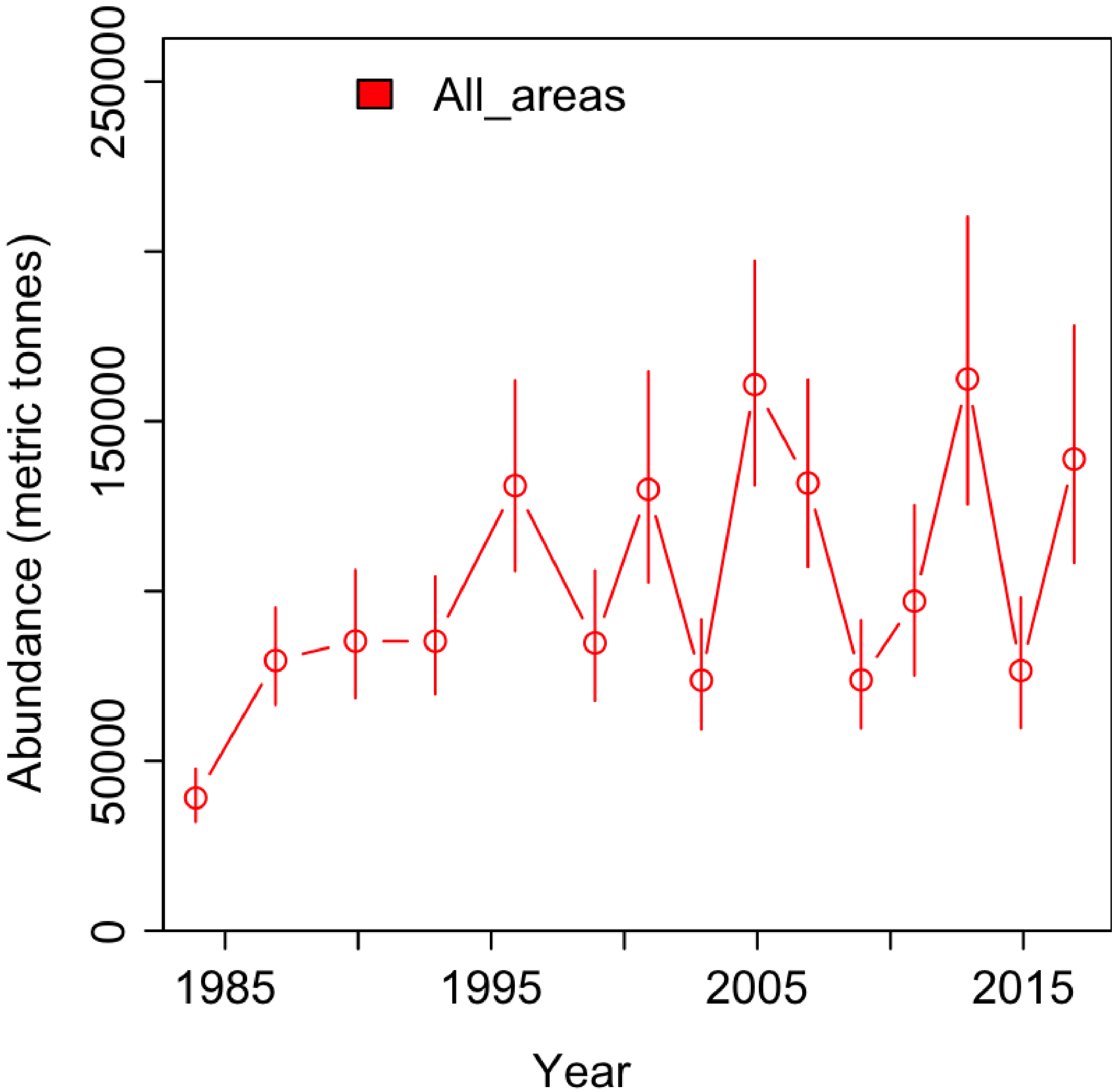


Figure 8: Index of abundance plus/minus 1 standard error

7 Comparing geostatistical and design-based indices

A logical next question is: How do these model-based biomass indices from VAST compare with the design-based indices we have used in the past? To answer this question we will calculate the design-based estimates for these same survey data. But first, we need a simple way to extract the VAST index..

7.1 Extracting VAST index

To return the **model-based** index values directly we can use a function called `get_VAST_index()` from the /R folder. `get_VAST_index()` is a simple function to retreive VAST index value and variance estimates.

- Note: This function only a simplified version of the PlotIndex_Fn() function in the SpatialDeltaGLMM package.

```
source("R/get-VAST-index.r")
vast_est = get_VAST_index(TmbData=TmbData, Sdreport=Opt[["SD"]], bias.correct=bias.correct, Data_Geosta
#Limit to years with observations
vast_est = vast_est[Years2Include,]
```

7.2 Calculate design-based estimate

To return the to calculate the **desing-based** index we can use a function called `calc_design_based_index()` from the /R folder, which returns the Biomass Index, Variance, SD, and CV.

`calc_design_based_index()` calls/requires the function `load_RACE_data()` from the /R folder, which loads and combines the catch and trawl datasets.

- Note: The `calc_design_based_index()` is currently only implemented for the single species case, not combining species codes or multiple species.

```
source("R/load-RACE-data.R")
source("R/calc-design-based-index.R")

db_est = calc_design_based_index(species.codes=species.codes, survey=survey)
```

```
## Species to include: 30420

## Number of samples to include for each species: 31306

## Finished processing for 30420
```

7.3 Plot comparison

Lets plot a simple comparison of the indices we have estimated, with uncertainty presented shaded regions describing +1SD and +2SD.

```
png(paste0(DateFile,"/VAST DB Index Comparison.png"), height=8, width=9, units='in', res=500)
  par(mfrow=c(1,1), oma=c(0,0,0,0), mar=c(4,4,3,1))

  y.lim = c(0, max(vast_est$Estimate_metric_tons+2*vast_est$SD_mt,
                   db_est$Biomass+2*db_est$SD))
  x.lim = c(min(Year_Set), max(Year_Set))
```

```
  #Survey years to plot
  years = Year_Set[Years2Include]

  #Plot it out
  plot(x=NULL, y=NULL, xlim=x.lim, ylim=y.lim, xlab='Year', ylab='Abundance (metric tonnes)',
       main=paste0(' n=', TmbData$n_x, ' knots'))
  grid(col='black')

  legend('top', legend=c('Design-based', 'VAST'), fill=c('blue', 'red'), ncol=2, bg='white')

  #Design-based
  polygon(x=c(years, rev(years)), y=c(db_est$Biomass+2*db_est$SD, rev(db_est$Biomass-2*db_est$SD)),
          border=FALSE, col=rgb(0,0,1, alpha=0.25))

  polygon(x=c(years, rev(years)), y=c(db_est$Biomass+1*db_est$SD, rev(db_est$Biomass-1*db_est$SD)),
          border=FALSE, col=rgb(0,0,1, alpha=0.25))

  lines(x=years, y=db_est$Biomass, lwd=2, col='blue')
  points(x=years, y=db_est$Biomass, pch=21, bg='blue')

  #VAST
  polygon(x=c(years, rev(years)), y=c(vast_est$Estimate_metric_tons+2*vast_est$SD_mt,
                                      rev(vast_est$Estimate_metric_tons-2*vast_est$SD_mt)),
          border=FALSE, col=rgb(1,0,0, alpha=0.25))

  polygon(x=c(years, rev(years)), y=c(vast_est$Estimate_metric_tons+1*vast_est$SD_mt,
                                      rev(vast_est$Estimate_metric_tons-1*vast_est$SD_mt)),
          border=FALSE, col=rgb(1,0,0, alpha=0.25))

  lines(x=years, y=vast_est$Estimate_metric_tons, lwd=2, col='red')
  points(x=years, y=vast_est$Estimate_metric_tons, pch=21, bg='red')
dev.off()
```

```
## pdf
##   2
```

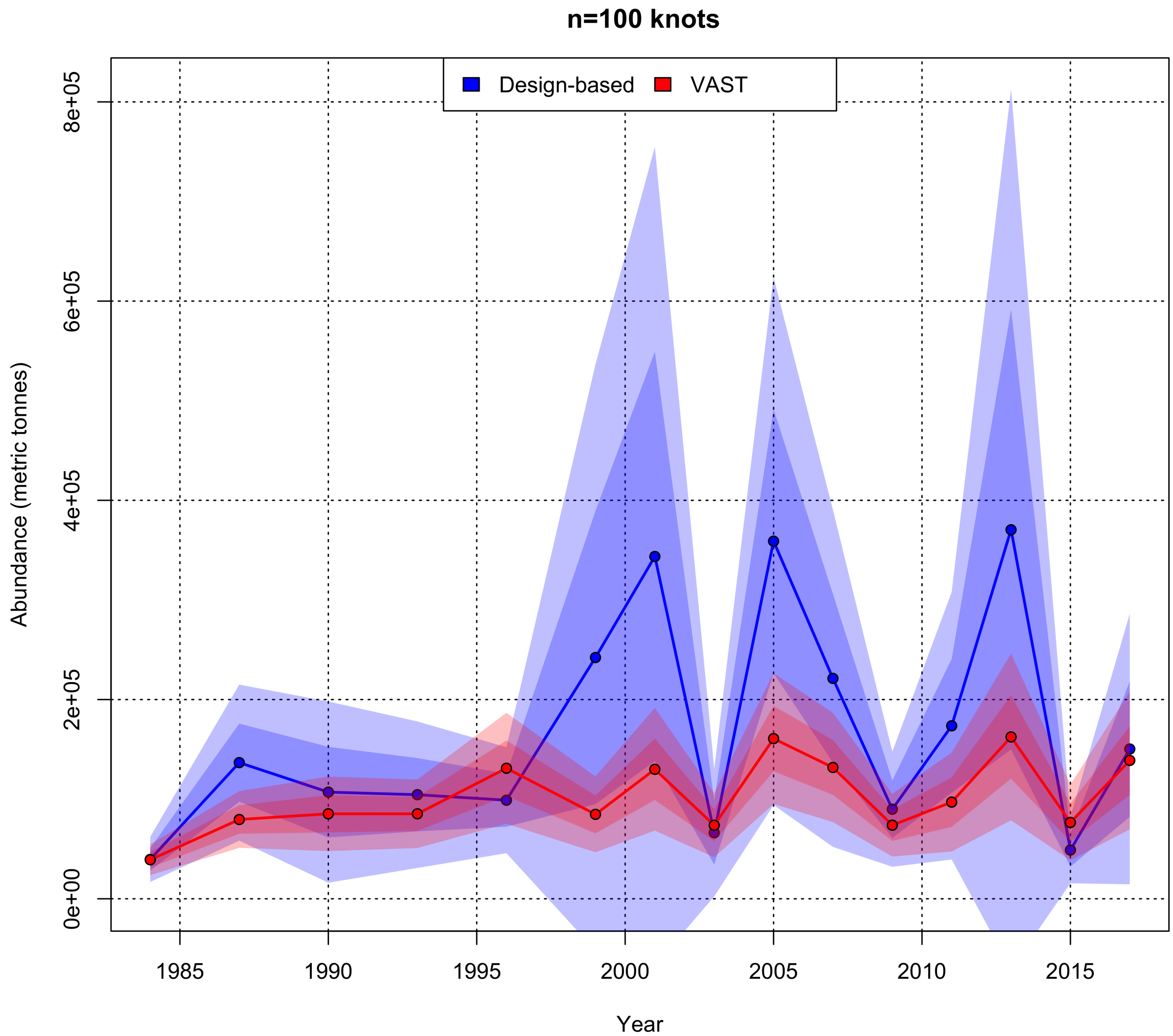


Figure 9: Comparison of VAST and design-based indices for RACE GOA bottom trawl. Index uncertainty presented as +1SD and +2SD.