Appendix S4

Shelton, A.O., Z.J. Gold, A.J. Jensen, E. D'Agnese, E.A. Allan, A. Van Cise, R. Gallego, A. Ramón-Laca, M. Garber-Yonts, K. Parsons, and R.P. Kelly. 2022. Toward Quantitative Metabarcoding. Ecology.

Introduction

This document accompanies "Toward Quantitative Metabarcoding" by Shelton et al. (2022) and provides a toy example for implementing the model with empirical data. We present an example from the Pacific fish example in the main text to illustrate implementation in practice. This example assumes that sequencing data has been appropriately generated and classified into taxa. Data are the number of reads associated with each taxa in each sample.

Pacific Fish

This example involves using data produced by Zack Gold using mock communities constructed from species from the California current. We use three mock communities from the "Ocean" group: Ocean even, Ocean skew 1, and Ocean skew 2. There are 18 species in this set of species and as their names suggest, Ocean even has equal abundance across all species (5.66% each) while the two skew communities have 4 species with relatively high abundance (18.75%) and 8 species with lower contributions (3.125% each). In the example below will use the even communities as a known mock community and the remaining as unknown samples.

First we need to load needed libraries and read into R the observed sequence data.

```r
# you will need these packages to make this program run.
library(dplyr)
library(compositions)
library(tidyverse)
library(rstan)
library(here)
library(ggsci)

# This is the data from Ocean communities.
dat <- readRDS("Ocean_mock_community.RDS")
# Take a brief look at the data:
head(dat)
```

```
## # A tibble: 6 x 6
##   ID_mifish             community      tech_rep Cycles nReads start_conc_ng
##   <chr>                 <chr>          <chr>    <chr>   <dbl>         <dbl>
## 1 Bathylagoides wesethi Oceanic_Even   1        39       4129            30
## 2 Bathylagoides wesethi Oceanic_Even   2        39       4850            30
## 3 Bathylagoides wesethi Oceanic_Even   3        39       2862            30
## 4 Bathylagoides wesethi Skew_Oceanic_1 1        39          0             0
## 5 Bathylagoides wesethi Skew_Oceanic_1 2        39          0             0
## 6 Bathylagoides wesethi Skew_Oceanic_1 3        39          0             0
```

Here the columns describe the taxa (ID_mifish), mock community (community), technical replicate number (tech_rep), number of PCR cycles (Cycles), number of observed reads (nReads), and known starting DNA concentration (start_conc_ng). This is the entirety of data that is required for running the statistical model. You will note that the starting DNA concentration is not strictly necessary; below we will convert these starting concentrations into proportion of DNA from each species in the mock community. These proportions will be used in the statistical model. The data includes only the three Ocean communities (even, skew 1, and skew 2) for a single number of PCR cycles. It includes three technical replicates for each community.

```
dat %>% distinct(community,tech_rep)
```

```
## # A tibble: 9 x 2
##   community      tech_rep
##   <chr>          <chr>
## 1 Oceanic_Even   1
## 2 Oceanic_Even   2
## 3 Oceanic_Even   3
## 4 Skew_Oceanic_1 1
## 5 Skew_Oceanic_1 2
## 6 Skew_Oceanic_1 3
## 7 Skew_Oceanic_2 1
## 8 Skew_Oceanic_2 2
## 9 Skew_Oceanic_2 3
```

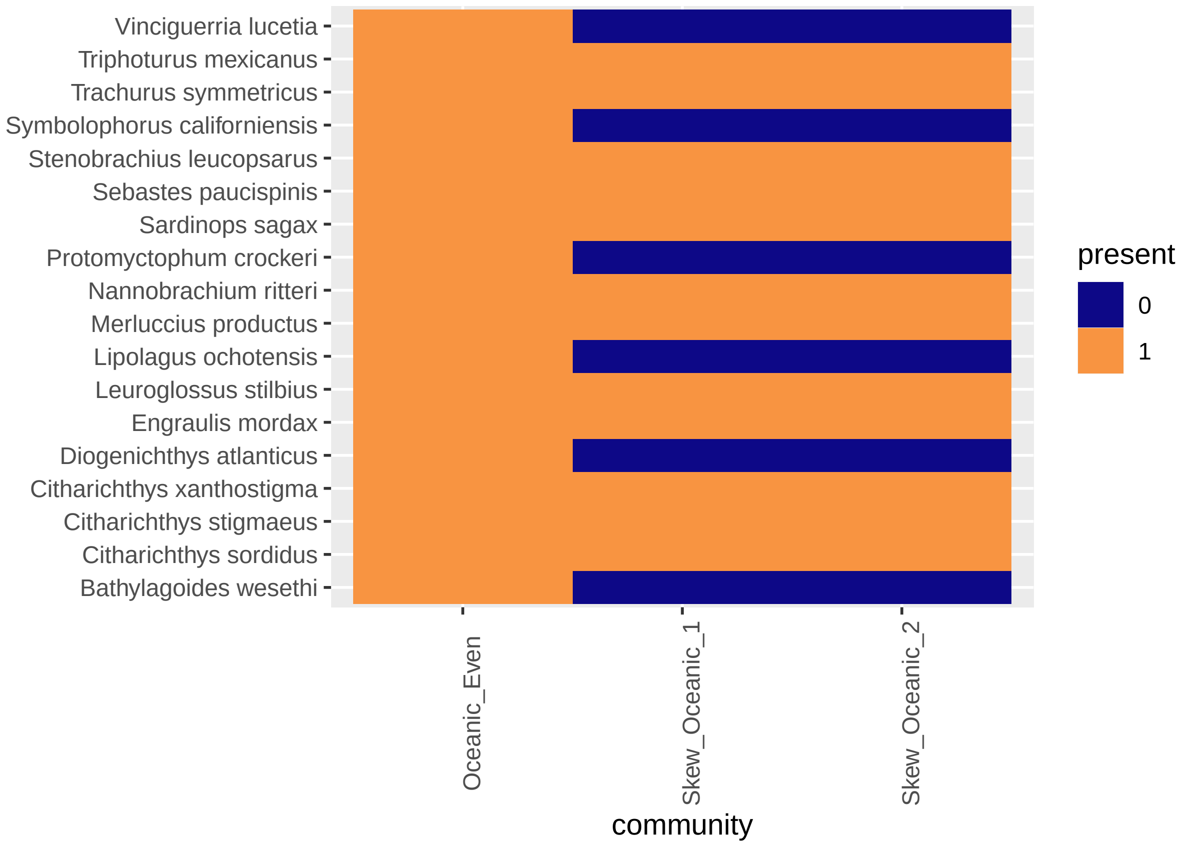
and includes 18 species, each of which is present in at least one community.

```
dat %>% distinct(ID_mifish)
```

```
## # A tibble: 18 x 1
##    ID_mifish
##    <chr>
##  1 Bathylagoides wesethi
##  2 Citharichthys sordidus
##  3 Citharichthys stigmaeus
##  4 Citharichthys xanthostigma
##  5 Diogenichthys atlanticus
##  6 Engraulis mordax
##  7 Leuroglossus stilbius
##  8 Lipolagus ochotensis
##  9 Merluccius productus
## 10 Nannobrachium ritteri
## 11 Protomyctophum crockeri
## 12 Sardinops sagax
## 13 Sebastes paucispinis
## 14 Stenobrachius leucopsarus
## 15 Symbolophorus californiensis
## 16 Trachurus symmetricus
## 17 Triphoturus mexicanus
## 18 Vinciguerria lucetia
```

We can make a figure showing which species occur in each sample

```
sp.by.comm.slim <-  dat %>%
  group_by(species=ID_mifish,community) %>%
  summarise(conc = sum(start_conc_ng)) %>%
  mutate(conc2 = ifelse(conc>0,1,0)) %>%
  arrange(species,community) %>% group_by(community) %>%
  mutate(tot_conc = sum(conc),prop=conc/tot_conc) %>%
  dplyr::select(-conc,-prop,-tot_conc) %>%
  pivot_wider(names_from = community,values_from = conc2,values_fill = 0) %>%
  pivot_longer(!species, names_to = "community",values_to= "present") %>%
  mutate(present = as.factor(present))

# Make some grids of species to understand overlap.
ggplot(sp.by.comm.slim) +
  geom_tile(aes(x=community,y=species,fill=present)) +
  scale_fill_viridis_d(option = "plasma",end = 0.75) +
  scale_y_discrete("") +
  theme(axis.text.x = element_text(angle=90) )
```



Setting up data frames for analysis

As in the main text of the manuscript, we will treat the skew communities as field samples of unknown composition and the even community as the mock community of known composition.

```
#set up environmental/unknown samples
  env <- dat %>%
          rename("Species" = "ID_mifish") %>%
          group_by(community,tech_rep) %>%
          mutate(propReads = nReads/sum(nReads), #calculate proportion of reads
               totReads = sum(nReads)) %>%  #calculate total reads for community
          group_by(Species) %>%
          mutate(totalSpeciesReads = sum(nReads)) %>%
          add_tally(nReads > 0, name = "totalOccurrences") %>%
          filter(totalSpeciesReads > 0)
  #assign most common species in the observed sequences to be the reference species
  mostCommon <- env %>%
    group_by(Species) %>%
    tally(nReads > 0) %>%
    arrange(desc(n)) %>%
    head(1) %>%
    pull(Species)
  #make the most common species the reference species
  env$Species[env$Species == mostCommon] <- paste0("zRefSpecies_", mostCommon)
  env <- env %>%
    arrange(Species, community)

# set up mock communities
  mc <- dat %>%
    filter(str_detect(community, "Even")) %>%
    rename("Species" = "ID_mifish")  #mock comm samples
  #make the most common species the reference species
    mc$Species[mc$Species == mostCommon] <- paste0("zRefSpecies_", mostCommon)

  # Filter so that you only keep species in the environment samples that are in the mock community.
  # It is ok to include species that are only in the mock community.
  env <- env %>%
      filter(str_detect(community, "Even",negate = TRUE)) %>%
      filter(Species %in% mc$Species)%>% #limit to species occurring in mock community dataset
      arrange(Species, community)

  #double check
  sum(!mc$Species %in% unique(env$Species)) # This can be non-zero
```

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

```
  sum(!env$Species %in% unique(mc$Species)) # this had better be zero.
```

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

```
  # Make a single species list:
    sp.list   <- data.frame(Species = sort(unique(mc$Species))) %>% mutate(sp_idx =1:length(Species))
    N_species <- nrow(sp.list)

    comm.mock.list <- mc %>% group_by(community, tech_rep,Cycles) %>%
                        summarise(n=length(tech_rep)) %>%
                        ungroup() %>%
                        mutate(id=1:length(n))
    comm.env.list   <- env %>% group_by(community, tech_rep,Cycles) %>%
                        summarise(n=length(tech_rep)) %>%
                        ungroup() %>% mutate(id=1:length(n))

    #make a list of species that are in mock community but not environment,
    # expand grid to make it so the the environmental samples get padded with all the
    # missing species for all of the communities and all tech replicates.

    sp.comm.mc  <- expand_grid(Species = sp.list$Species, id = comm.mock.list$id) %>%
                          left_join(.,sp.list %>% dplyr::select(Species,sp_idx)) %>%
                          left_join(.,comm.mock.list %>%
                          dplyr::select(community,tech_rep,Cycles,id) ) %>%
```

```r
                          dplyr::select(-id)
    sp.comm.env <- expand_grid(Species = sp.list$Species, id = comm.env.list$id) %>%
                          left_join(.,sp.list %>% dplyr::select(Species,sp_idx)) %>%
                          left_join(.,comm.env.list %>%
                          dplyr::select(community,tech_rep,Cycles,id) ) %>%
                          dplyr::select(-id)

    # convert data from long-form to wide-form
    # merge in species and indices first to make pivoting more efficient.
    mc  <- left_join(sp.comm.mc,mc) %>%
              mutate(nReads = ifelse(is.na(nReads),0,nReads),
              start_conc_ng = ifelse(is.na(start_conc_ng),0,start_conc_ng))
    env <- left_join(sp.comm.env,env) %>%
              mutate(nReads = ifelse(is.na(nReads),0,nReads),
              start_conc_ng = ifelse(is.na(start_conc_ng),0,start_conc_ng))

    # this will be the unknown sample data
    sample_data <- env %>%
      ungroup() %>%
      dplyr::select(community, sp_idx, nReads,tech_rep, Cycles) %>%
      arrange(sp_idx) %>%
      pivot_wider(names_from = "sp_idx", values_from = "nReads", values_fill = 0) %>%
      mutate(Cycles = as.numeric(Cycles))

    # Make a simple data structure that will only be used for make posterior predictions
    sample_data_small <- sample_data %>% filter(tech_rep==1)

    # this will be the mock community sample data
    mock_data <- mc %>%
      ungroup() %>%
      dplyr::select(community, sp_idx, nReads,tech_rep, Cycles) %>%
      arrange(sp_idx) %>%
      pivot_wider(names_from = "sp_idx", values_from = "nReads", values_fill = 0) %>%
      mutate(Cycles = as.numeric(Cycles))

    mock_data_small <- mock_data %>% filter(tech_rep==1)

#proportions
p_mock <- mc %>%
  select(community, tech_rep, sp_idx, start_conc_ng, Cycles) %>%
  arrange(sp_idx) %>%
  group_by(community, tech_rep, Cycles) %>%
  mutate(prop_conc = start_conc_ng/sum(start_conc_ng)) %>%
  select(-start_conc_ng) %>% #, -Species) %>%
  pivot_wider(names_from = "sp_idx", values_from = "prop_conc", values_fill = 0) %>%
  ungroup() %>%
  arrange(community)
  #select(-community)

p_mock_small <- mc %>%
  filter(tech_rep == 1) %>%
  select(community, sp_idx, start_conc_ng, Cycles) %>%
  arrange(sp_idx) %>%
  group_by(community) %>%
  mutate(prop_conc = start_conc_ng/sum(start_conc_ng)) %>%
  select(-start_conc_ng) %>%  # -Species) %>%
  pivot_wider(names_from = "sp_idx", values_from = "prop_conc", values_fill = 0) %>%
  ungroup() %>%
  arrange(community)
  #select(-community, -Cycles)

  #calculate additive log ratios
  # this means reference species is 0 and all other species are relative to that species.
  # add a trivial amount of proportion to each species (1e-12) to avoid logarithms of zero.
  alr_mock_true_prop <- p_mock[,4:(ncol(p_mock)-1)]*0
  for(i in 1:nrow(p_mock)){
    alr_mock_true_prop[i,] <- alr(p_mock[i,4:(ncol(p_mock))] + 1e-12)
  }
  alr_mock_true_prop[,N_species] <- 0 #adding explicit reference species column

  alr_mock_true_prop_small <- p_mock_small[,3:(ncol(p_mock_small)-1)]*0
  for(i in 1:nrow(p_mock_small)){
    alr_mock_true_prop_small[i,] <- alr(p_mock_small[i,3:(ncol(p_mock_small))] + 1e-12)
  }
  alr_mock_true_prop_small[,N_species] <- 0
```

Create design matrices for fitting the model

This section mostly makes matrices that are useful for performing matrix multiplication in the estimation model.

```
# Make DESIGN MATRICES for the betas (site-specific relative abundance)
# and the alphas (amplification efficiencies)

  ##### Mock communities first
  # Make a vector of PCR cycles.
  N_pcr_mock <- mock_data$Cycles

  # For the mock communities, you only need the relative amplification efficiencies.
  # efficiencies (alphas)
  formula_a <- community ~ Cycles -1
  model_frame <- model.frame(formula_a, mock_data)
  model_vector_a_mock <- model.matrix(formula_a, model_frame) %>% as.numeric()

  N_obs_mock       <- nrow(mock_data)
  # For the unknown communities, you need to make a design matrix for the sites (betas)
  # and second component for the amplification efficiencies.

  # use sample_data
  N_pcr_samp <- sample_data$Cycles

  if(length(unique(sample_data$community))==1){
    formula_b <- Cycles ~ 1
  } else {
    formula_b <- Cycles ~ community
  }
  model_frame <- model.frame(formula_b, sample_data)
```

```r
  model_matrix_b_samp <- model.matrix(formula_b, model_frame)

  # This is a model frame for making a single prediction for each site.
  model_frame <- model.frame(formula_b, sample_data_small)
  model_matrix_b_samp_small <- model.matrix(formula_b, model_frame)

  # efficiencies (alphas)
  # for all observations
  formula_a <- community ~ Cycles -1
  model_frame <- model.frame(formula_a, sample_data)
  model_vector_a_samp <- model.matrix(formula_a, model_frame) %>% as.numeric()
  # for a single observation for each site
  model_frame <- model.frame(formula_a, sample_data_small)
  model_vector_a_samp_small <- model.matrix(formula_a, model_frame) %>% as.numeric()

  #counters
  N_obs_samp_small <- nrow(model_matrix_b_samp_small)
  N_obs_samp <- nrow(sample_data)
  N_b_samp_col <- ncol(model_matrix_b_samp)
```

Model estimation

Ok. The previous sections generate the data that is needed to feed an estimation model. Here we make the lists and such that we will feed the stan program.

```r
stan_data <- list(
  N_species = N_species,   # Number of species in data
  N_obs_samp = N_obs_samp, # Number of observed samples
  N_obs_mock = N_obs_mock, # Number of observed mock samples
  N_obs_samp_small = N_obs_samp_small, # Number of observed samples

  # Observed data of community matrices
  sample_data = sample_data %>% select(-community,-Cycles,-tech_rep),
  mock_data   = mock_data  %>% select(-community,-Cycles,-tech_rep),

  # True proportions for mock community
  #mock_true_prop = p_mock_all %>% dplyr::select(contains("sp")),
  alr_mock_true_prop = alr_mock_true_prop,
  alr_mock_true_prop_small = alr_mock_true_prop_small,

  # Design matrices: field samples
  N_b_samp_col = N_b_samp_col,
  model_matrix_b_samp = model_matrix_b_samp,
  model_matrix_b_samp_small = model_matrix_b_samp_small,
  model_vector_a_samp = model_vector_a_samp,
  model_vector_a_samp_small = as.array(model_vector_a_samp_small),

  # Design matrices: mock community samples
  model_vector_a_mock = model_vector_a_mock,

  # Priors
  alpha_prior = c(0,0.1),  # normal prior
  beta_prior = c(0,5),    # normal prior
  tau_prior = c(1.5,1.5)   # relatively diffuse gamma prior for over dispersion
)

# These are the parameters that are to be monitored during optimization or MCMC:
stan_pars <- c(
  "alpha", # efficiencies relative to the reference species
  "beta",  # parameters for each site (NOT )
  "eta_samp", # overdispersion random effects for each species-site combination (field samples)
  "eta_mock", # overdispersion random effects for each species-site combination (mock samples)
  "tau", # sd of random effects (one for each species (less the reference species))
  "mu_samp", # Predicted proportions for each species-site (unknown samples)
  "mu_mock", # Predicted proportions for each species-site (mock samples)
  "int_samp_small" # this is the predicted intercept for each site
)

# This is an initial value generator for the amplification efficiencies.
stan_init_f2 <- function(n.chain,N_species){
  A <- list()
  for(i in 1:n.chain){
    A[[i]] <- list(
      # tau = runif(N_species-1,0.1,0.5),
      alpha_raw = runif(N_species-1,-0.5,0.5)
    )
  }
  return(A)
}
```

Fitting the model

There are two paths to estimating the model. You can either use an optimizer to get a maximum likelihood esitmate (fast but limited / mediocre descriptions of uncertainty) or you can run a full Bayesian analysis with Hamiltonian MCMC (slow but you can do all sorts of things with the output). Both models call the same code and should provide very similar results. If you are running the MCMC option, be sure you know something about the details of MCMC – at the very least how to check that the model has converged. I can point out first steps below, but ask if you have questions.

Ok. Let's do the optimizer version first. This model requires replication for all of the observations. There is a second version of the model for unreplicated observations and there are some approaches for using a mixture of replicated and unreplicated samples.

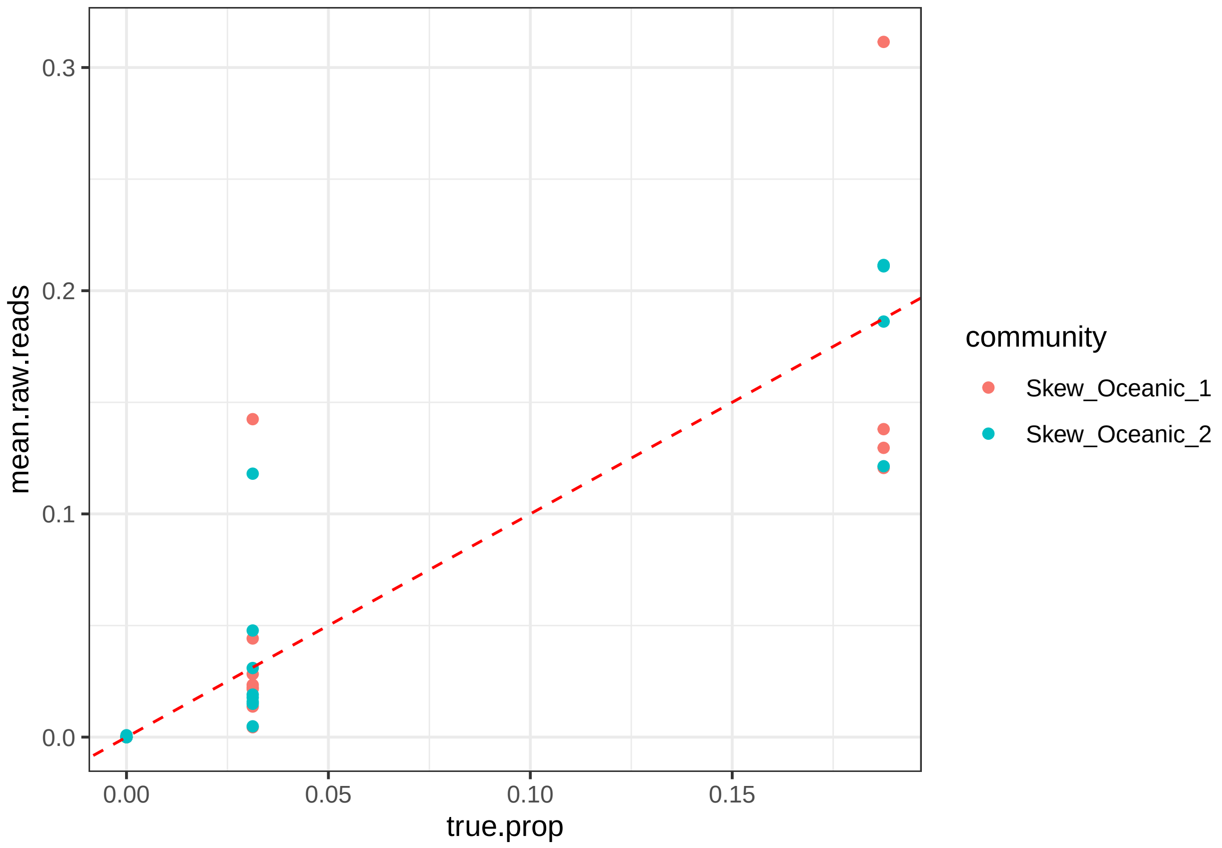
```
  # This calls the Stan code and compiles the model.  May take a few minutes
 M <- stan_model("quant_metabar_multinom.stan")
 likMod <- optimizing(M, data=stan_data, iter=100000,
                     draws=1000, #
                     verbose=T,
# These are optimizer settings.  You likely don't need to play with them
                     # ,tol_obj = 1e-8,
                     # tol_rel_obj = 1e-8,
                     # tol_grad = 1e-8,
                     # tol_param=1e-8,
                     # tol_rel_grad=1e-8,
                     algorithm="LBFGS",
                     hessian = TRUE)
```

```
## Chain 1: Initial log joint probability = -3.17492e+06
## Chain 1:     Iter      log prob        ||dx||      ||grad||       alpha      alpha0  # evals  Notes
## Chain 1: Error evaluating model log probability: Non-finite gradient.
## Error evaluating model log probability: Non-finite gradient.
## Error evaluating model log probability: Non-finite gradient.
## Error evaluating model log probability: Non-finite gradient.
##
## Chain 1:      999       -614829    0.00129893       1085.23      0.5828      0.5828     1058
## Chain 1:     Iter      log prob        ||dx||      ||grad||       alpha      alpha0  # evals  Notes
## Chain 1:     1999       -614637    0.00108025       151.657           1           1     2110
## Chain 1:     Iter      log prob        ||dx||      ||grad||       alpha      alpha0  # evals  Notes
## Chain 1:     2999       -614604     0.0012732       41.7386      0.7873      0.7873     3163
## Chain 1:     Iter      log prob        ||dx||      ||grad||       alpha      alpha0  # evals  Notes
## Chain 1:     3460       -614600   0.000287036       22.5824           1           1     3651
## Chain 1: Optimization terminated normally:
## Chain 1:   Convergence detected: relative gradient magnitude is below tolerance
```

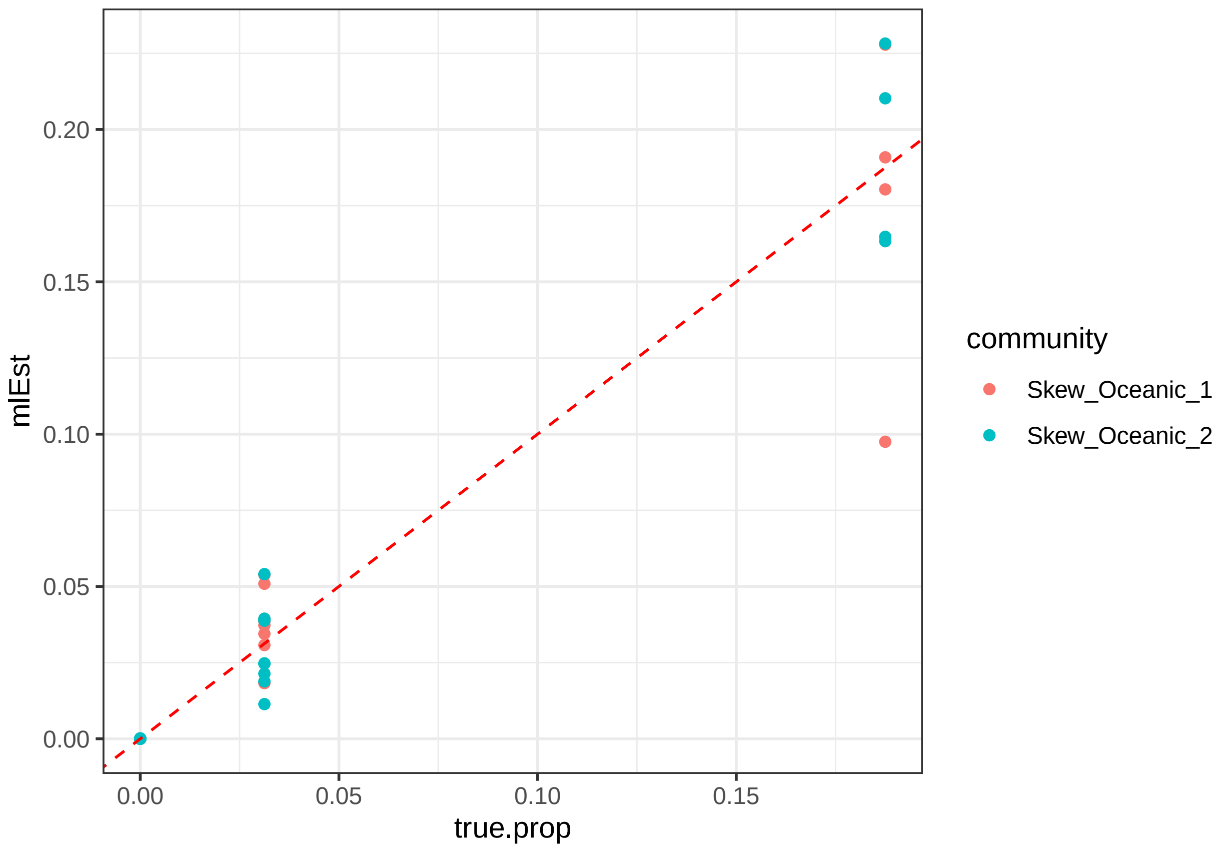
```r
# Extract point estimates
 # Estimated proportions for each species-sample combination
estProportions <- data.frame(mlEst = likMod$par[grepl("int_samp_small",names(likMod$par))]) %>%
   mutate(community= rep(sample_data_small$community,N_species)) %>%
   mutate(comm_idx = rep(1:length(unique(env$community)), times = N_species)) %>%
   mutate(sp_idx = rep(1:N_species, each = length(unique(env$community)))) %>%
   mutate(nom=rownames(.)) %>%
   left_join(.,mc %>% distinct(Species,sp_idx))

 # Estimated alphas for each species
estAlpha <- data.frame(mlEst = likMod$par[grepl("alpha",names(likMod$par))]) %>%
  mutate(nom=rownames(.)) %>%
  filter(!grepl("_raw",nom)) %>%
  mutate(sp_idx=1:nrow(.)) %>%
  left_join(.,mc %>% distinct(Species,sp_idx) )

# You can extract the other parameters using similar code.


# This is how you summarize values from the likelihood fit
# (inverting the hessian, assuming multivariate normality, etc.)
# this is what the "draws" argument does in the optimizing function.
# Big picture it looks ok, but seems like it has very heavy tails... which can happen
# when the assumptions about multi-variate normality for the estimate
# don't interact well with the transformations involved in this problem.
prop_draw <- likMod$theta_tilde[,grepl("int_samp_small",colnames(likMod$theta_tilde))]

PROBS <- c(0.025,0.05,0.25,0.75,0.95,0.975)
prob.nom <- paste0("q",PROBS)
prop_draw_a <- prop_draw %>% as.data.frame() %>%
                  pivot_longer(cols=everything(),names_to="sample", values_to ="prop") %>%
                  group_by(sample) %>%
                  summarise(q=quantile(prop,probs=PROBS),Mean=mean(prop),Median = median(prop)) %>%
                  mutate(prob.nom = prob.nom)

prop_draw_summary <- prop_draw_a %>%
                      pivot_wider(.,id_cols=c("sample","q"),names_from="prob.nom",values_from="q") %>%
                      left_join(.,prop_draw_a %>% distinct(sample,Mean,Median)) %>%
                      left_join(.,estProportions %>% dplyr::select(sample=nom,community,Species))
```

OK. Let's plot the results a little so we can assess how the model is doing.

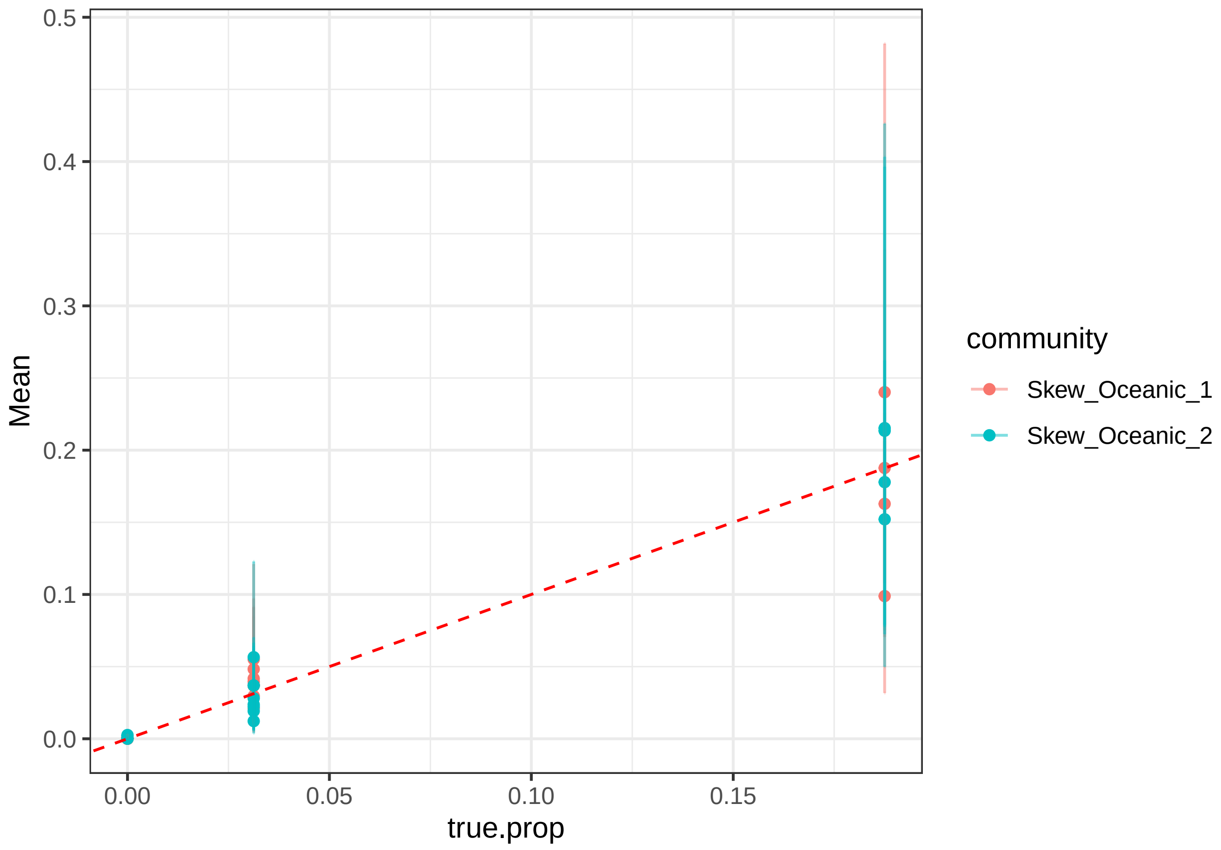
```
# Derive summary of starting proportions for the skew communities

result.dat <- env %>% group_by(community,Species) %>%
                  summarise(conc = mean(start_conc_ng),mean.raw.reads = mean(propReads)) %>%
                  group_by(community) %>%
                  mutate(total.conc = sum(conc)) %>%
                  ungroup() %>%
                  mutate(true.prop = conc / total.conc) %>%
                  left_join(.,estProportions) %>%
                  left_join(.,prop_draw_summary)

# Make a couple of plots.
# First Compare mean reads across replicates against true proportions
# perfect would be on the dashed line
ggplot(result.dat) +
    geom_point(aes(x=true.prop,y=mean.raw.reads,color=community)) +
    geom_abline(intercept = 0,slope=1,color="red",linetype="dashed") +
    theme_bw()
```



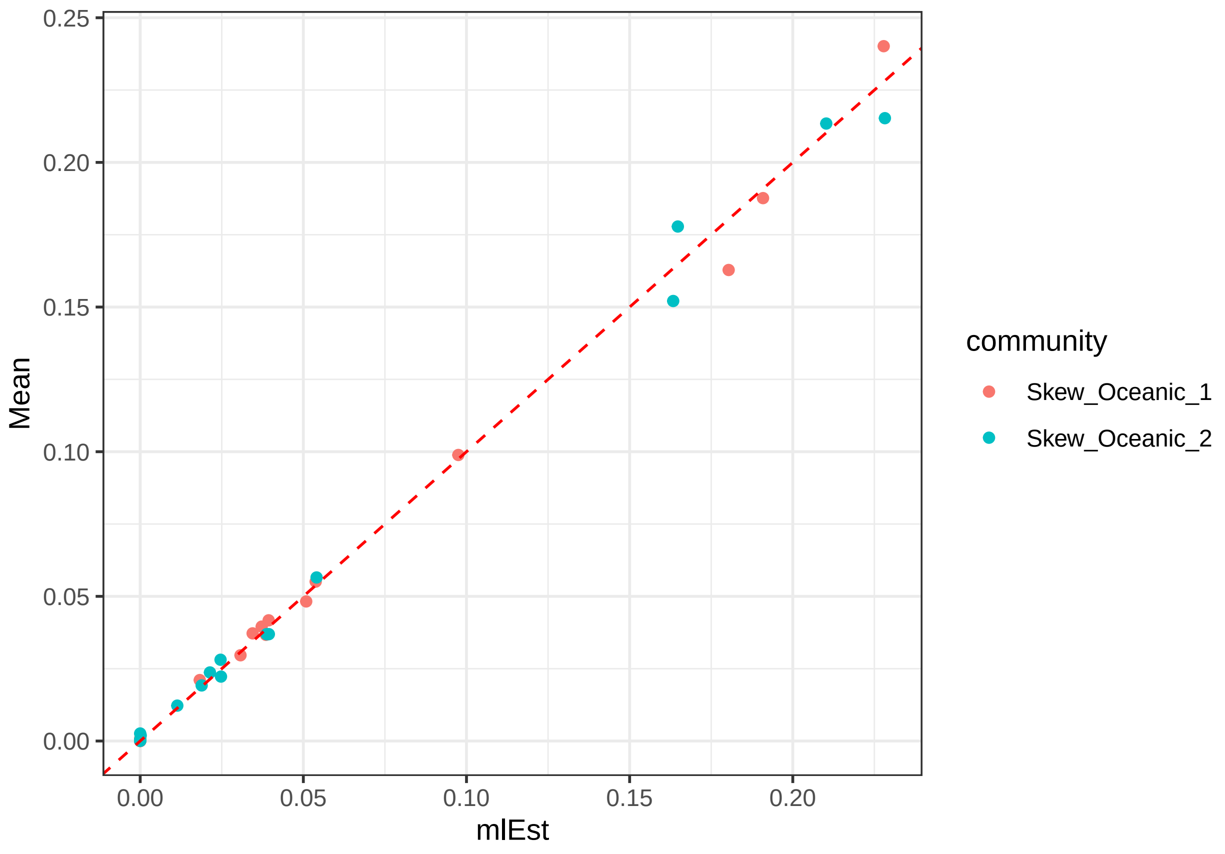
```
# not so hot.

# Better if you use point estimates from the model.
ggplot(result.dat) +
    geom_point(aes(x=true.prop,y=mlEst,color=community)) +
    geom_abline(intercept = 0,slope=1,color="red",linetype="dashed") +
    theme_bw()
```



```
# Or you can use the information from the draws to provide some uncertainty bounds.
# Note that there are some mighty big error bars there.
ggplot(result.dat) +
    geom_point(aes(x=true.prop,y=Mean,color=community)) +
    geom_errorbar(aes(x=true.prop,ymin=q0.05,ymax=q0.95,Mean,color=community),alpha=0.5,width=0) +
    geom_abline(intercept = 0,slope=1,color="red",linetype="dashed") +
    theme_bw()
```



```r
# note that the estimates from the maximum likelihood and the draws are close by not identical.
ggplot(result.dat) +
    geom_point(aes(x=mlEst,y=Mean,color=community)) +
    geom_abline(intercept = 0,slope=1,color="red",linetype="dashed") +
    theme_bw()
```



Bayesian version

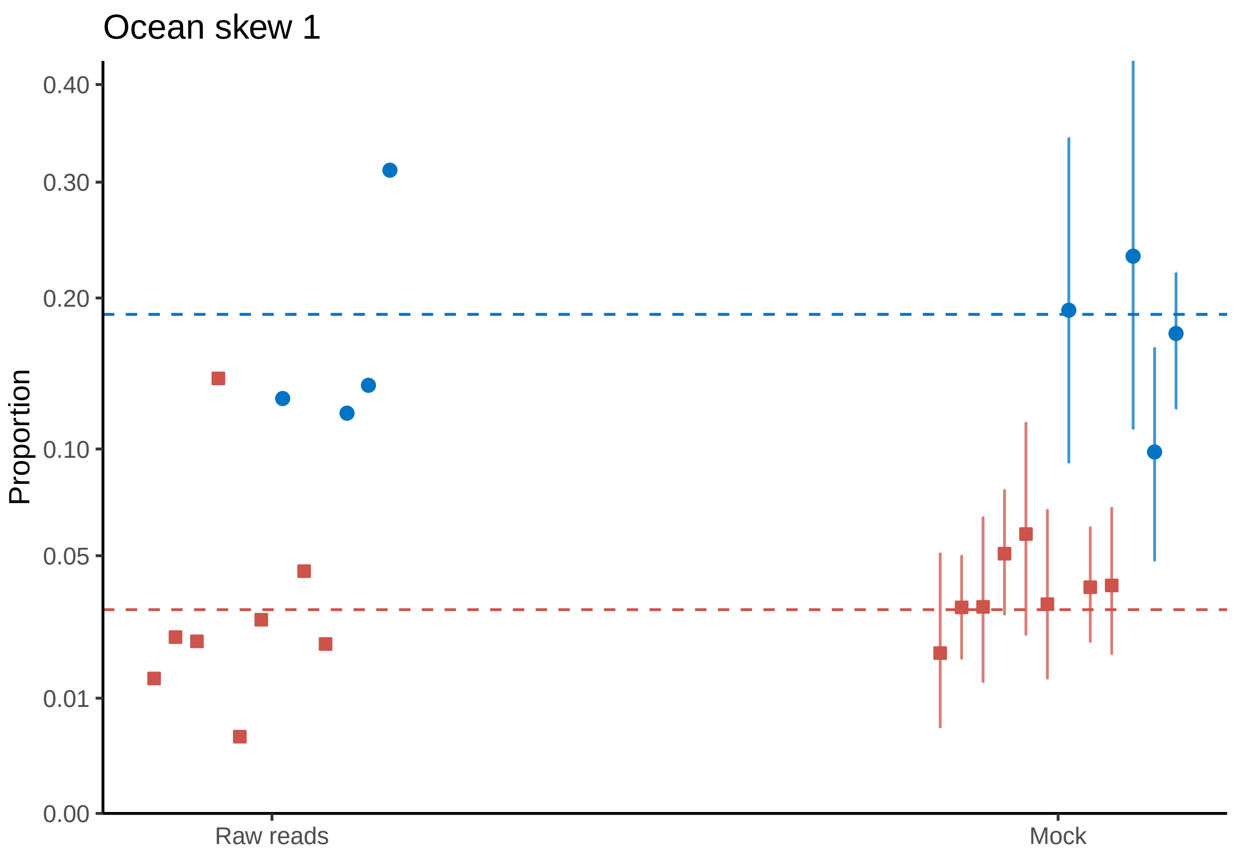
OK. This is the full Bayesian version. On my computer it takes a minute or two. I have commented out the actual estimation and included summarized output as a

```
  ##########################################
#Bayesian Estimation
# This is a very small problem so this runs pretty quick.
N_CHAIN = 3
Warm = 500
Iter = 1000
Treedepth = 13
Adapt_delta = 0.8

# This is the code to actually run the model.  It is commented out for speed

# rstan_options(auto_write = TRUE)
# options(mc.cores = parallel::detectCores())
# # you may have to specify a correct path for this to work...
# stanMod = stan(file = "quant_metabar_multinom.stan" ,data = stan_data,
#                verbose = FALSE, chains = N_CHAIN, thin = 1,
#                warmup = Warm, iter = Warm + Iter,
#                control = list(adapt_init_buffer = 175,
#                               max_treedepth=Treedepth,
#                               stepsize=0.01,
#                               adapt_delta=Adapt_delta,
#                               metric="diag_e"),
```

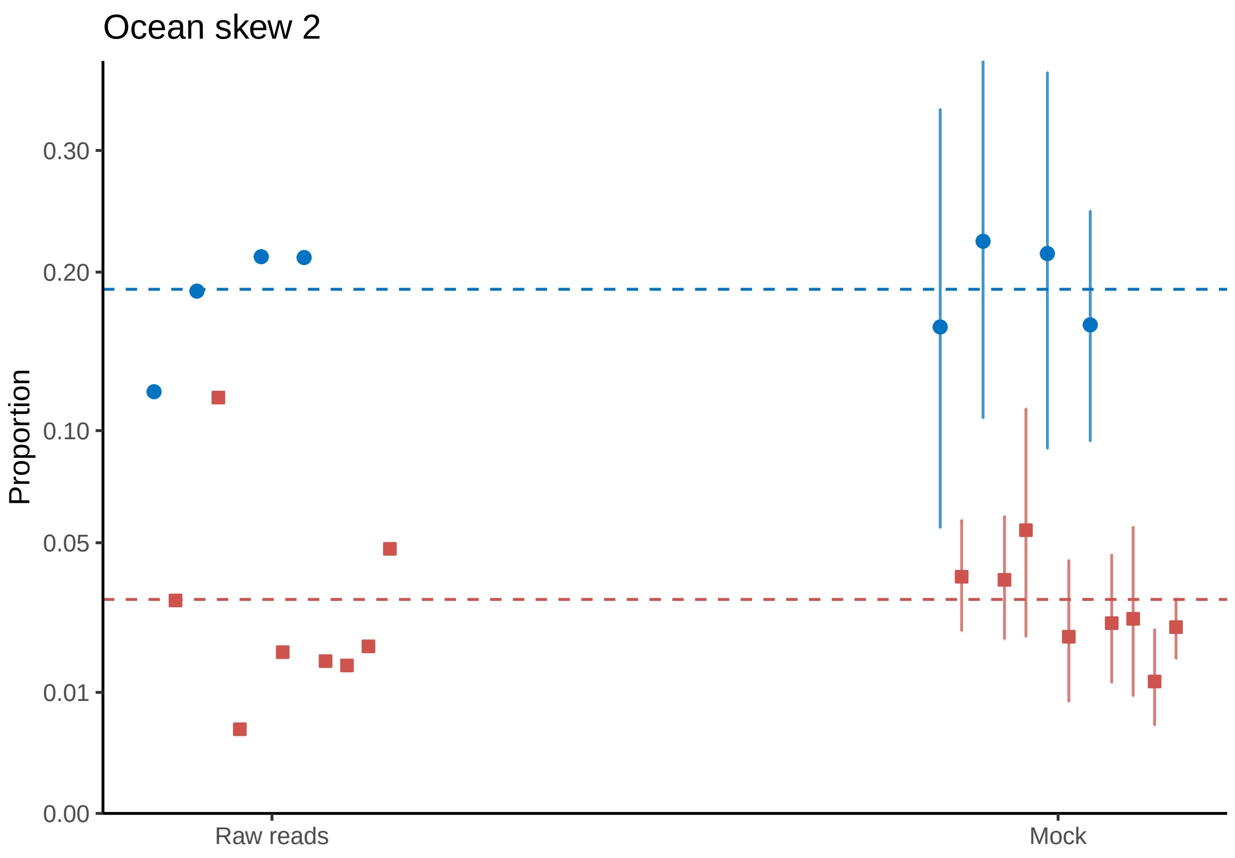
```r
#                 pars = stan_pars,
#                 #refresh = 10,
#                 boost_lib = NULL,
#                 init = stan_init_f2(n.chain=N_CHAIN,N_species=N_species)
#                 #sample_file = paste0("./tmpF.csv")
# )
#
# pars <- rstan::extract(stanMod, permuted = TRUE)
# samp_params <- get_sampler_params(stanMod)
#
# stanMod_summary <- list()
# stanMod_summary[["alpha"]] <- summary(stanMod,pars="alpha")$summary
# stanMod_summary[["tau"]] <- summary(stanMod,pars=c("tau"))$summary
# stanMod_summary[["beta"]] <- summary(stanMod,pars="beta")$summary
# stanMod_summary[["eta_samp_raw"]] <- summary(stanMod,pars="eta_samp")$summary
# stanMod_summary[["eta_mock_raw"]] <- summary(stanMod,pars="eta_mock")$summary
# stanMod_summary[["mu_samp"]] <- summary(stanMod,pars="mu_samp")$summary
# stanMod_summary[["mu_mock"]] <- summary(stanMod,pars="mu_mock")$summary
# stanMod_summary[["int_samp_small"]] <- summary(stanMod,pars="int_samp_small")$summary
#
# Output <- list(
#
#   env = env,  #environmental sample data
#   mc = mc, #mock data
#   Species = unique(mc$Species),
#
#   # Input data
#   p_true = p_mock,
#   p_samp_all = sample_data,
#   p_mock_all = mock_data,
#
#   # stan input objects
#   stan_data = stan_data,
#   Warm=Warm,
#   Iter=Iter,
#
#   # Fitted Objects
#   stanMod = stanMod, # Full stan Model fitted object
#   pars = pars, # MCMC output
#   samp_params=samp_params, # Sampler information
#   stanMod_summary = stanMod_summary # posterior summaries.
# )
#
# # This saves things to file so you don't have to run the model every time.
# save(Output,file=paste0("Fish_Oceanic_crossValidate",".Rdata"))
```

```r
# Extract estimates and make basic plots.
# read in the file created in the previous chunk.
load("Fish_Oceanic_crossValidate.Rdata")
mock2 <- Output # This is the cross validation that uses only the 39 cycle even communities

#########################################################
# Mock2
#########################################################
```

```r
# summarize raw estimates from reads for each species.
mock2.raw <- mock2$env %>% group_by(community,Cycles,tech_rep) %>%
  mutate(sum.ng = sum(start_conc_ng),
         true.prop = start_conc_ng / sum.ng) %>%
  ungroup() %>%
  group_by(Species,community,Cycles,true.prop) %>%
  summarise(simple.Mean=mean(propReads),
            simple.N = length(tech_rep)) %>%
  replace_na(list(raw.Mean=0,raw.SD=0,raw.SE=0))

# extract predicted proportions from the posterior
COM <- data.frame(community = levels(mock2$env$community %>% as.factor()))
COM$comm_idx <- 1:nrow(COM)
SP  <- mock2$env %>% distinct(Species,sp_idx) %>% as.data.frame()

# These are the predicted intercepts for the posteriors
beta_posterior <- mock2$stanMod_summary[["int_samp_small"]][, c(1,4:8)]
colnames(beta_posterior) <- paste0("mock2.",substr(colnames(beta_posterior),1,nchar(colnames(beta_poste
colnames(beta_posterior)[1] <- "mock2.mean"
beta_posterior <- as.data.frame(beta_posterior)

mock2.post <-expand.grid(comm_idx = COM$comm_idx,sp_idx =SP$sp_idx) %>%
  arrange(comm_idx,sp_idx) %>%
  left_join(.,COM) %>%
  left_join(.,SP) %>%
  bind_cols(.,beta_posterior)

# Combine the raw estimates and posterior estimates
mock2.all <- full_join(mock2.raw,mock2.post)

mock2.all <- mock2.all %>% mutate(true.cat= ifelse(true.prop>0.1,"big","small"))

# pull out just ocean.skew.dat for plotting.
spread=0.15
ocean.skew1.dat <- mock2.all %>%
  filter(true.prop > 0,community=="Skew_Oceanic_1")
ocean.skew1.dat <- bind_cols(ocean.skew1.dat,
                             data.frame(offset=     seq(-spread,spread,length.out=nrow(ocean.skew1.dat)

ocean.skew2.dat <- mock2.all %>%
  filter(true.prop > 0,community=="Skew_Oceanic_2")
ocean.skew2.dat <- bind_cols(ocean.skew2.dat,
                            data.frame(offset= seq(-spread,spread,length.out=nrow(ocean.skew2.dat)))

# Make plots
BREAKS <- c(0.0,0.01,0.05,0.10,0.20,0.30,0.40,0.6,0.8)
x.labs <- c("Raw reads","Mock")
x.at   <- c(1,2)

skew_plot <- function(dat,
                      BREAKS=BREAKS,x.labs=x.labs,x.at=x.at){

    shape.val = c(21,22)
    col.val = pal_jco(palette = c("default"), alpha = 1)(10)[c(1,4)]

  skew.plot <-  ggplot(dat) +
    geom_point(aes(x=1+offset,y=simple.Mean,shape=true.cat,fill=true.cat,color=true.cat),size=2) +
    geom_errorbar(aes(x=2+offset,
                      ymin= mock2.2.5,
                      ymax= mock2.97.5,color=true.cat),width=0,alpha=0.75)   +
    geom_point(aes(x=2+offset,mock2.mean,shape=true.cat,fill=true.cat,color=true.cat),size=2) +
    scale_shape_manual(values =c(21,22)) +
    scale_fill_manual(values= col.val, "True value") +
    scale_color_manual(values= col.val,"True value") +
    scale_y_continuous("Proportion",
                       trans="sqrt",
                       # trans="log",
                       breaks = BREAKS,expand=c(0,NA),limits = c(0,NA)) +
    geom_hline(aes(yintercept = true.prop,color=true.cat),linetype="dashed") +
    #geom_point(aes(x=0.70,y=true.prop,shape=true.cat,fill=true.cat),size=3) +
    scale_x_continuous(name=NULL,breaks=x.at,labels = x.labs) +
    theme_classic() +
    theme(legend.position = "none")

  return(skew.plot)
}


ocean.skew.1 <- skew_plot(dat=ocean.skew1.dat,
                          BREAKS=BREAKS,x.labs=x.labs,x.at=x.at)
print(ocean.skew.1  +ggtitle("Ocean skew 1"))
```



```
ocean.skew.2 <- skew_plot(dat=ocean.skew2.dat,
                          BREAKS=BREAKS,x.labs=x.labs,x.at=x.at)
print(ocean.skew.2 +ggtitle("Ocean skew 2"))
```



Other versions

OK. The above is for a model with technical replicates. If you don't have technical replication, you can't estimate some of the parameters. A simplified model version for that scenario involves using the file *quant_metabar_no_overdispersion.stan*. You would change the required stan model files like so:

```
# These are the parameters that are to be monitored during optimization or MCMC:
stan_pars <- c(
  "alpha", # efficiencies relative to the reference species
  "beta",  # parameters for each site (NOT )
  "mu_samp", # Predicted proportions for each species-site (unknown samples)
  "mu_mock", # Predicted proportions for each species-site (mock samples)
  "int_samp_small" # this is the predicted intercept for each site
)
```

and everything else can be used as above. Essentially, you can't estimate the random effects that allow for over-dispersion with a single replicate. If you have a mix of replication and no replication, that is trickier but we can likely work something out if you want to work on that.

There are two other files that might be of interest that allow you to fit an equivalent model assuming no amplification differences among species. These are:

quant_metabar_no_mock_no_alpha.stan

which allows you to fit a model with over-dispersion in the observations and

quant_metabar_no_overdispersion_no_alpha.stan

which does not allow for over-dispersion.