Population modelling for census support

Code

Tutorial 2: How to model large-scale spatial variations?

Edith Darin and Douglas Leasure

Compiled on 21/08/2021

WorldPop

Code

Introduction

In tutorial 1, we modelled population count with a compound Poisson-Lognormal. This modelling assumes however that the observations are independent and identically distributed.

Tutorial 2 will explore the spatial patterns of population counts and how they are survey siteed across space.

We will analyse the grouping of our observations and unravel their hierarchical structure. We will then see how Bayesian models offer a great modelling toolbox to represent hierarchical data. We will finally try different hierarchical modelling structures to see which one fits the best our data.

Hierarchical models define complex parameter space. We will thus start discussing about efficiency in model coding and how to tune the Markov Chains Monte Carlo algorithm.

Goals

1. Understand the concept of grouped observations

2. Estimate population density in a no-pooling framework
3. Estimate population density in a partial pooling framework
4. Address some MCMC estimation issues

Supporting readings

- An Introduction to Hierarchical Modeling (http://mfviz.com/hierarchical-models/) by Michael Freeman. Great visualisation of grouped data and goals of general hierarchical modelling
- Bayes rules Book on hierarchical model (https://www.bayesrulesbook.com/chapter-15.html) by Alicia A. Johnson, Miles Ott, Mine Dogucu. Good explanation of hierarchically modelling in `stan` with many examples
- Brief guide to stan's warning (https://mc-stan.org/misc/warnings.html): Tuning Parameters for Hamiltonian Monte Carlo in `stan`

Extra package

We will use the `bayesplot` package that offers more flexibility in evaluating Bayesian estimations, `plotly` for interactive plotting and `extratDistr` for additional probability distributions.

Hide

```
install.packages('bayesplot') # additional evaluations of stan models
install.packages('plotly') # interactive plot
install.packages('extraDistr') # additional probability distribution
```

Hierarchical structure in the data

Let's unravel the survey siteing patterns in population count. Figure 1 shows the spatial variation of population density across Nigeria. We can already observed survey siteed patterns: some regions (in the North and in the South East) have a higher population density in average.

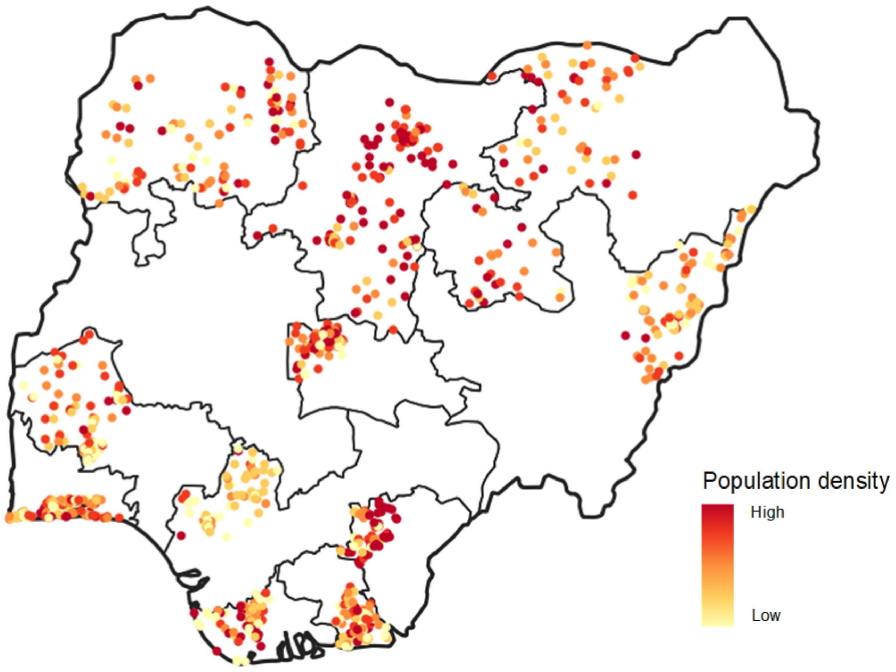


Figure 1: Map of survey site population densities overlaid with Nigeria region boundaries

Let's plot the population density distribution by regions:

Code

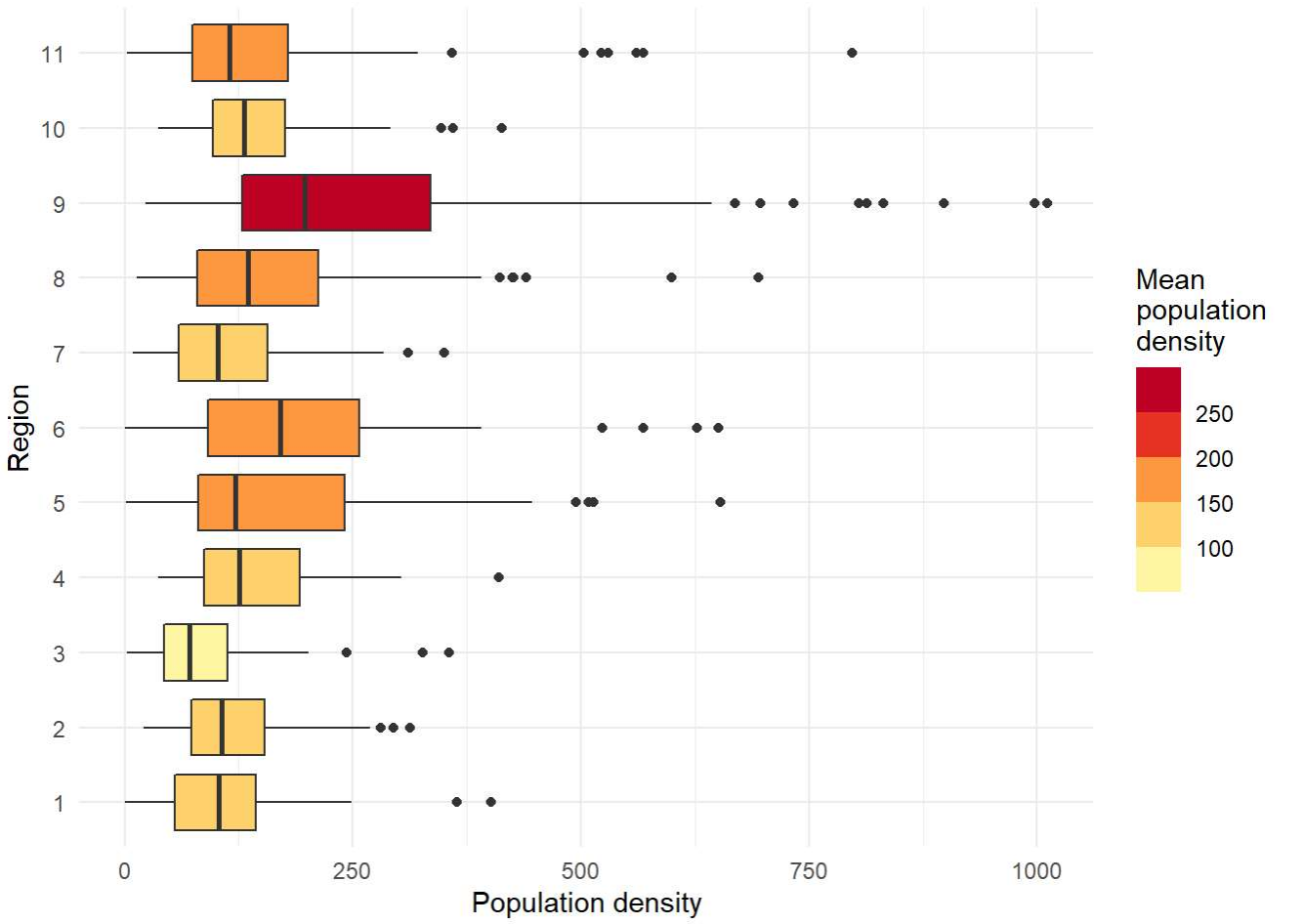


Figure 2: Boxplot of population densities per region

In addition to the 11 regions, the data provides two other **Nigerian administrative divisions**: state (15) and local (standing for local government areas, 223), which gives extra flexibility for grouping the survey sites. The underlying assumption: the administrative structure of a country gives information on the population density variation.

Another key grouping is the `type` attribute indicated in the data. It refers to a settlement map based on satellite imagery (Pleiades, Airbus and WorldView2, DigitalGlobe) classified into six **settlement types** pictured in Figure 3 Weber et al. (2018).

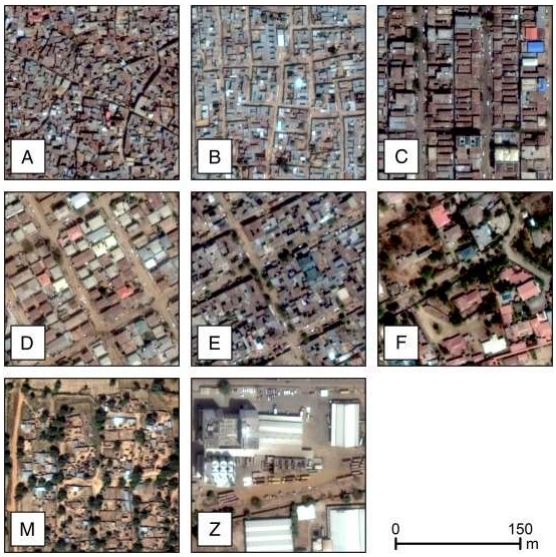


Figure 3: Exemplars of the urban residential (A–F), rural residential (M), and non-residential (Z) types for Kano and Kaduna states, Nigeria, from Weber et al (2018)

Code

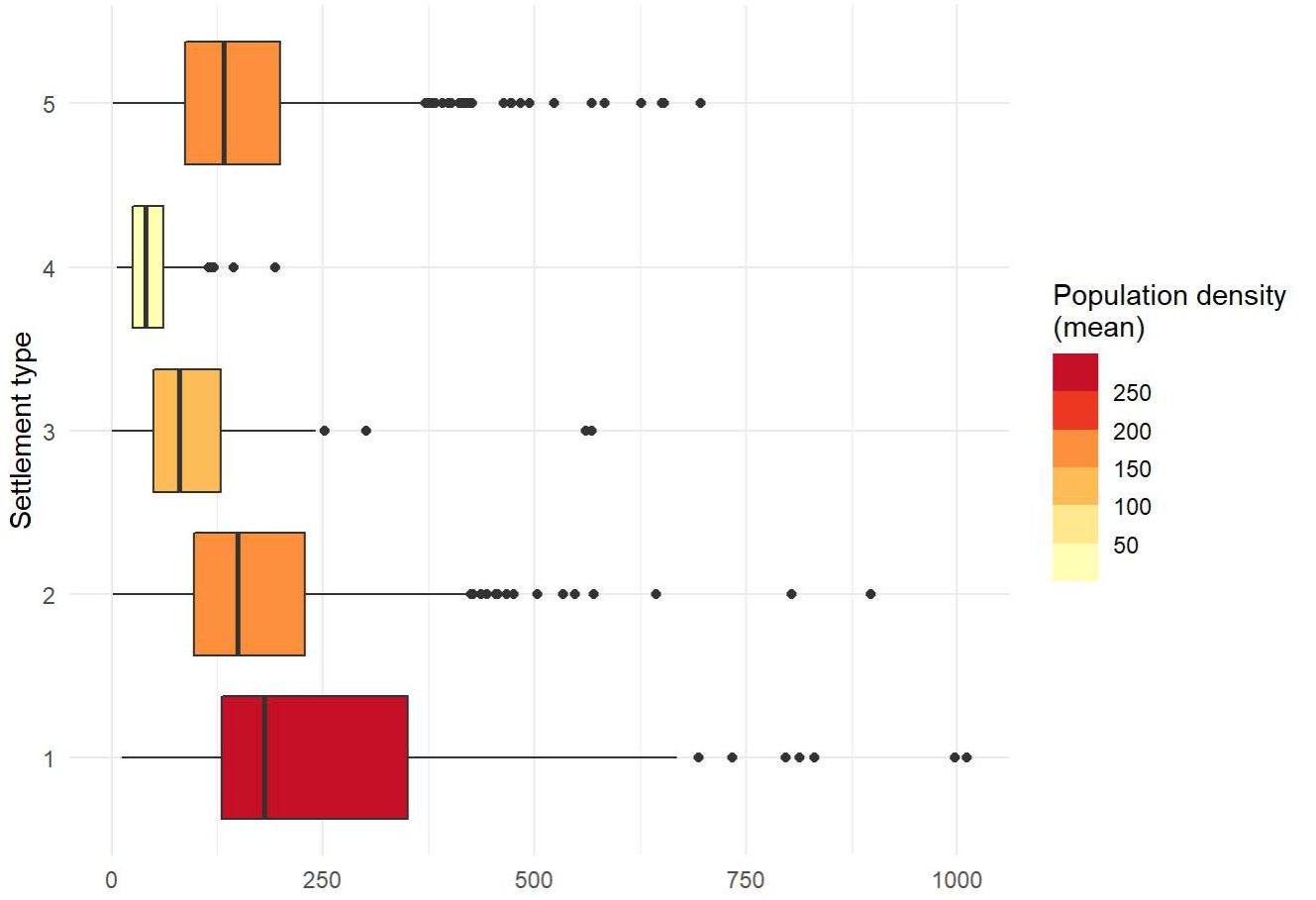


Figure 4: Boxplot of population densities per settlement type

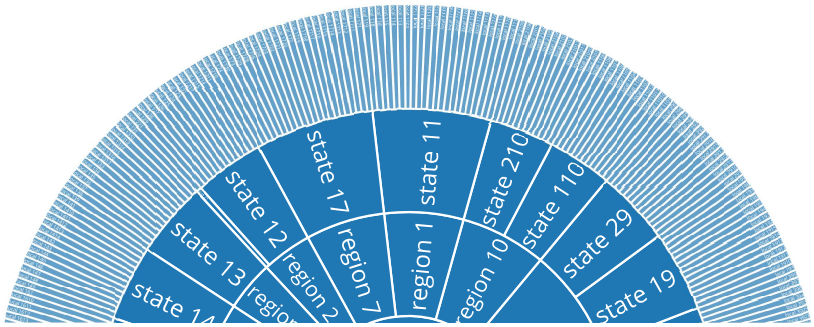
We see that settlement type clearly stratifies the population density, settlement type 1 having a substantial higher population density than settlement type 4.

Note that out of the eight settlement types displayed in Figure 3, only five types are actually present in the data. No sites were surveyed in the non-residential type area.

Full picture of the data grouping

Figure 5 shows the full structure of the survey sites, with the following nesting: settlement type, region, state and local government area. When hovering on the schema, you can see how many survey survey sites are in each grouping. You can access the detailed structure by clicking on a specific group.

Code



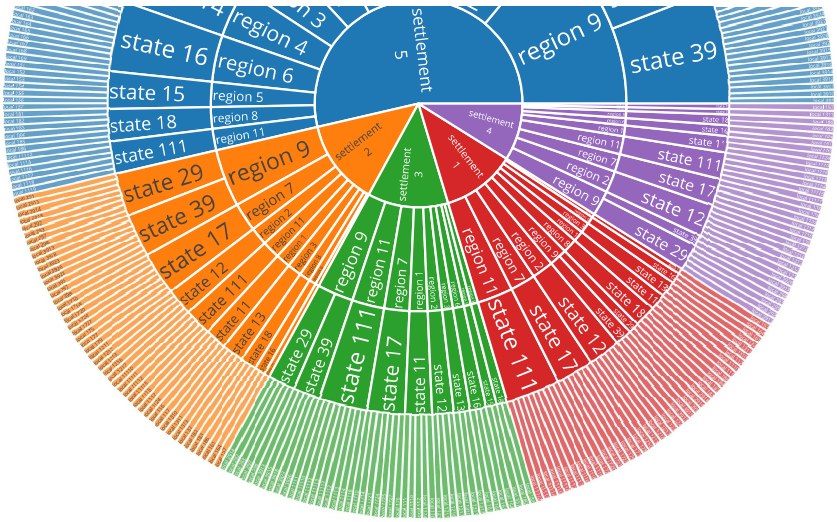
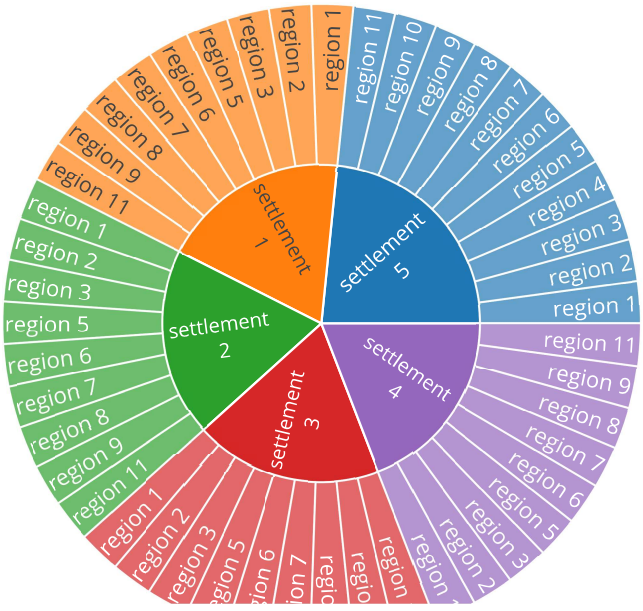


Figure 5: Full grouping structure of the survey

Note that:

1. not every state and local government area are present in the data due to sampling limitations
2. not every group combination is represented in the data. Some region/state/local governmental area do not have survey sites belonging to all settlement types. Figure 6 shows the regions with missing settlement type. We can't assess from this analysis if those settlement types are missing because they have not being sampled in our survey or they don't exist (for example a remote region that does not contain any highly urban settlement type).

Code



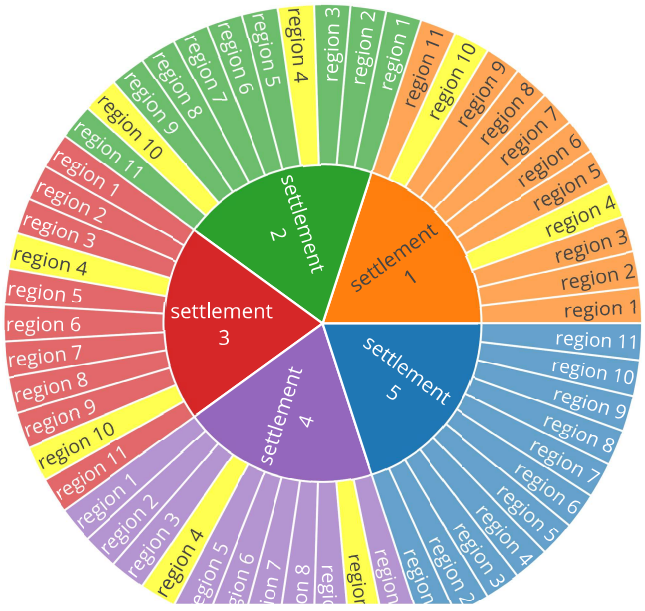


Figure 6: Full grouping structure of the survey with missing combination in yellow

Hierarchical structure in the model

When all observations are treated together and indistinctly in the model, it is called **complete pooling**. This is how we modelled population count in tutorial 1. It violates the assumption of independence between the observations and it might produce spurious conclusions that are valid at global level but invalid at group level, through diluting group-specific patterns (cf. the ecological fallacy (https://en.wikipedia.org/wiki/Ecological_fallacy)).

When a model is fit for each data grouping independently, it is called **no pooling** and create a model per grouping. It reduces drastically the sample size and makes it impossible to extrapolate for a grouping that is not present in the observations sample.

The purpose of hierarchical modelling is to aim at **partial pooling** where information is shared between group while accounting for the hierarchical structure of the data. It provides insights on:

- between-group variability: population densities differ from one region to the other
- within-group variability: some regions have a greater variability than others (cf region 9 in Figure 2)

Let's recall our previous model:

$$population \backsim Poisson(pop\_density * settled\_area)$$
$$pop\_density \backsim Lognormal(\alpha, \sigma)$$
$$\alpha \backsim Normal(0, 100)$$
$$\sigma \backsim Uniform(0, 100)$$

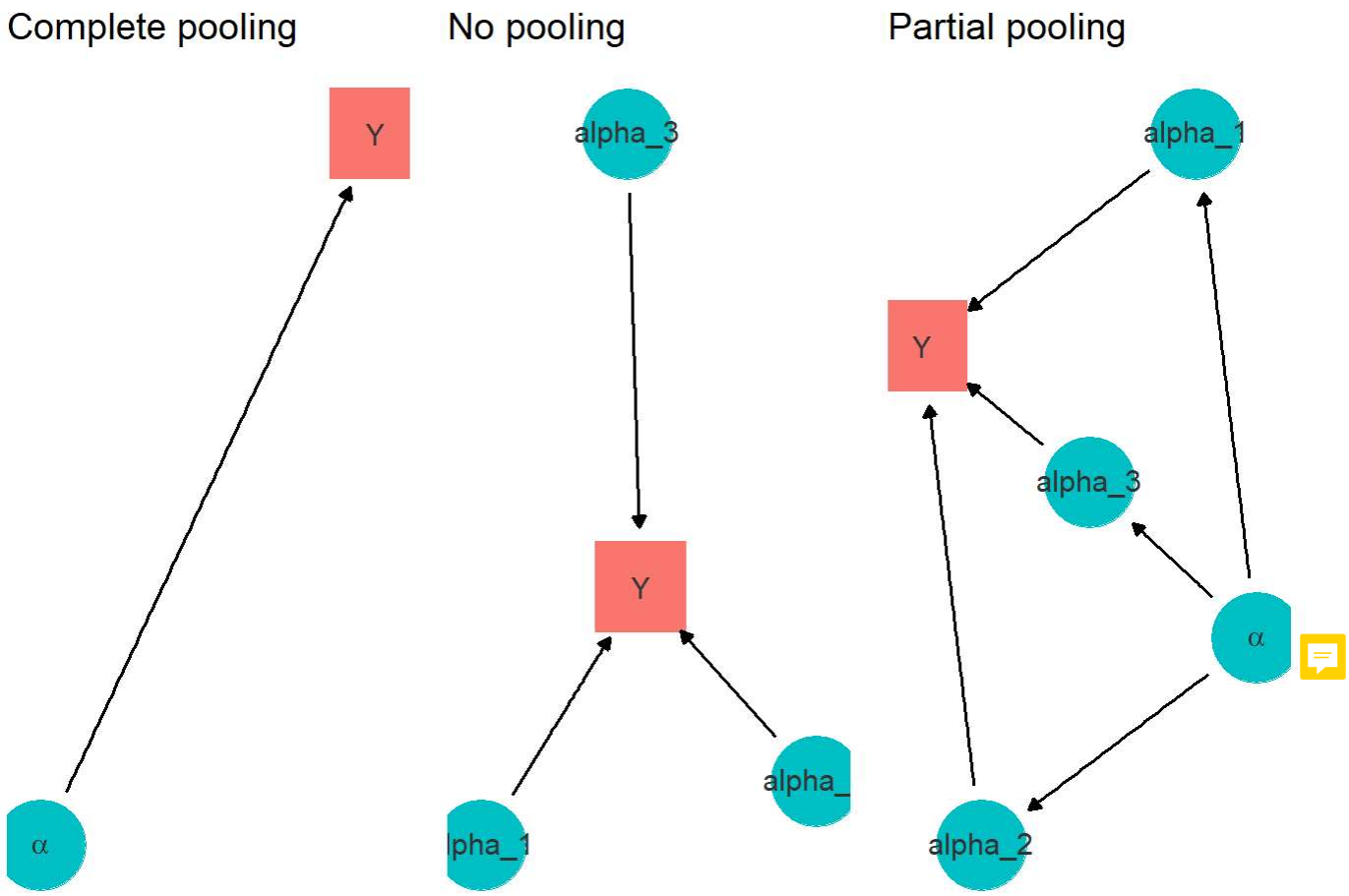
A hierarchical intercept by settlement type

We want to differentiate the parameters according to the hierarchical structure of the data.

A first parameter is α, that is the median on the log scale of the population density distribution (the moment of order 1 of a lognormal (https://en.wikipedia.org/wiki/Log-normal_distribution#Properties)). Modelling α hierarchically means acknowledging that the median differs depending on the grouping, for example different for settlement type 1 from settlement type 4.

Let's see the different scenarios for estimating α by settlement type (we will show only for three settlement types for sake of simplicity):

Code



We will see the difference between the three models: complete ooling, no pooling and partial pooling.

Model 2 from tutorial 1 is based on complete pooling: one α for every settlement type.

No-pooling

Let's implement a no-pooling framework, that is α **will be estimated independently for each settlement type** t. Equation (1) represents its formal model, whereby α is indexed by t the settlement type and each α_t has a identical but independent prior.

$$population \backsim Poisson(pop\_density * settled\_area) \quad (1)$$
$$pop\_density \backsim Lognormal(\alpha_t, \sigma)$$
$$\alpha_t \backsim Normal(0, 100)$$
$$\sigma \backsim Uniform(0, 100)$$

How do you write it in Stan? We will show only the code that is impacted.

Hide

```
// Model 1: Independent alpha by settlement type
data{
  ...
  int<lower=0> type[n]; // settlement type
  int<lower=0> ntype; // number of settlement types
}
parameters{
  ...
  // independent intercept by settlement type
  vector[ntype] alpha_t;
}
model{
  // population totals
  ...
  pop_density ~ lognormal( alpha_t[type], sigma );

  // independent intercept by settlement type
  alpha_t ~ normal(0, 100);
  ...
}
generated quantities{
  ...
   for(idx in 1:n){
     density_hat[idx] = lognormal_rng( alpha_t[type[idx]], sigma );
     population_hat[idx] = poisson_rng(density_hat[idx] * area[idx]);
   }
}
```

In the `data block` , we declare the observation structure of settlement type.

In the `parameters block` , α becomes a vector of size 5, the number of settlement type in the data.

In the `model block`, we choose the same prior for every α_t, five normal distribution centered on zero and with standard deviation 100.

Notice how the indexing works in the `generated quantities block` .

Once the model is written, we prepare the corresponding data, adding the settlement type:

Hide

```
# prepare data for stan
stan_data_model1 <- list(
  population = data$N,
  n = nrow(data),
  area = data$A,
  type = data$type,
  ntype= n_distinct(data$type))
```

And run the model in a similar fashion as in tutorial 1:

Hide

```
# mcmc settings
chains <- 4
warmup <- 250
iter <- 500
seed <- 1789

# parameters to monitor
pars <- c('alpha_t','sigma', 'population_hat', 'density_hat')

# mcmc
fit1 <- rstan::stan(file = file.path('tutorial2_model1.stan'),
                    data = stan_data_model1,
                    iter = warmup + iter,
                    chains = chains,
                    warmup = warmup,
                    pars = pars,
                    seed = seed)
```

No warnings is spit by `stan`, the model has converged.

Figure 7 shows that the estimated $\hat{\alpha}_t$ vary by settlement type. We also see how the previously estimated α was averaging between the different patterns.

Code

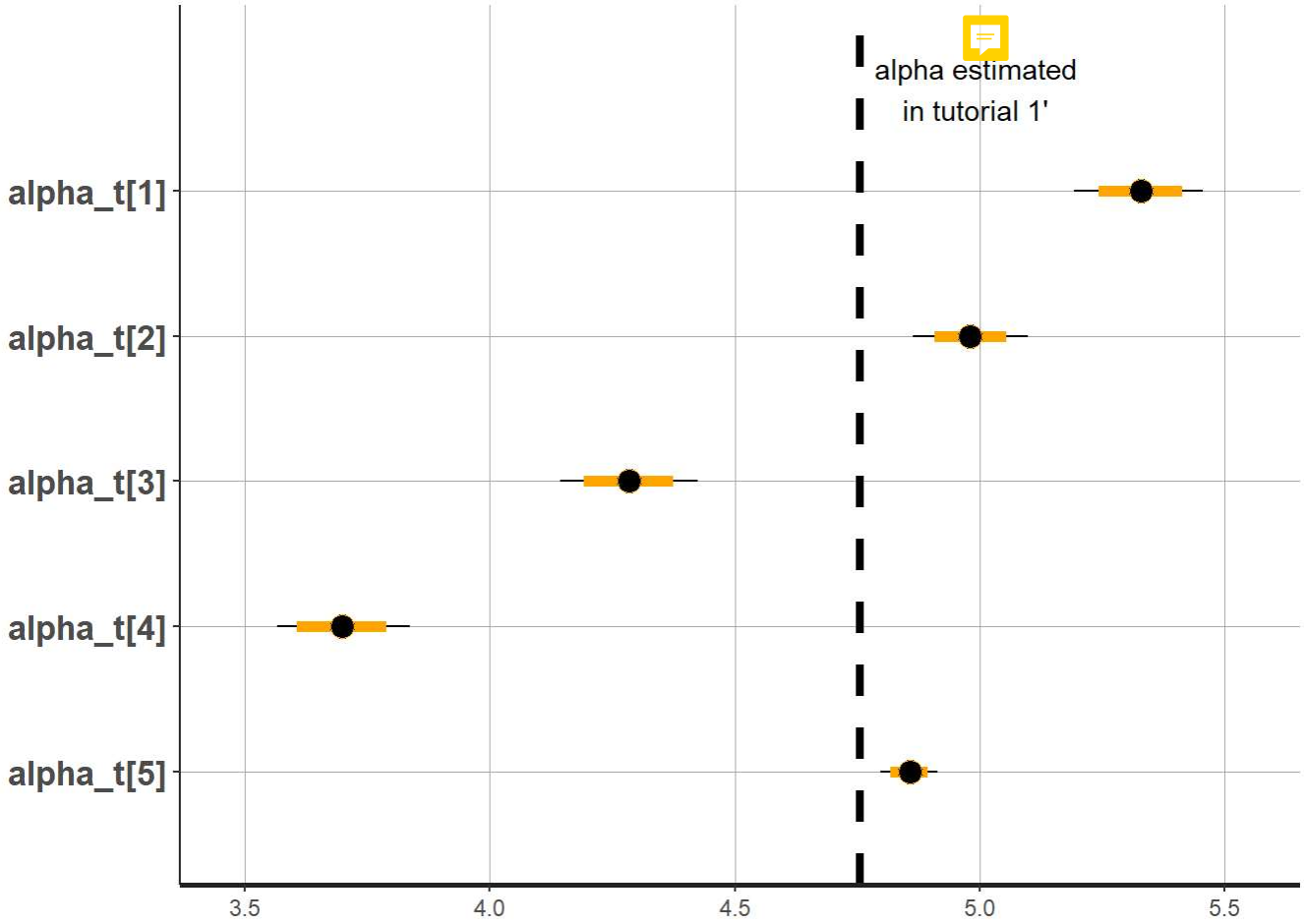


Figure 7: Intercept estimated independantly by settlement type

We draw the observed vs predicted plot to see the impact of estimating α by settlement type:

Code

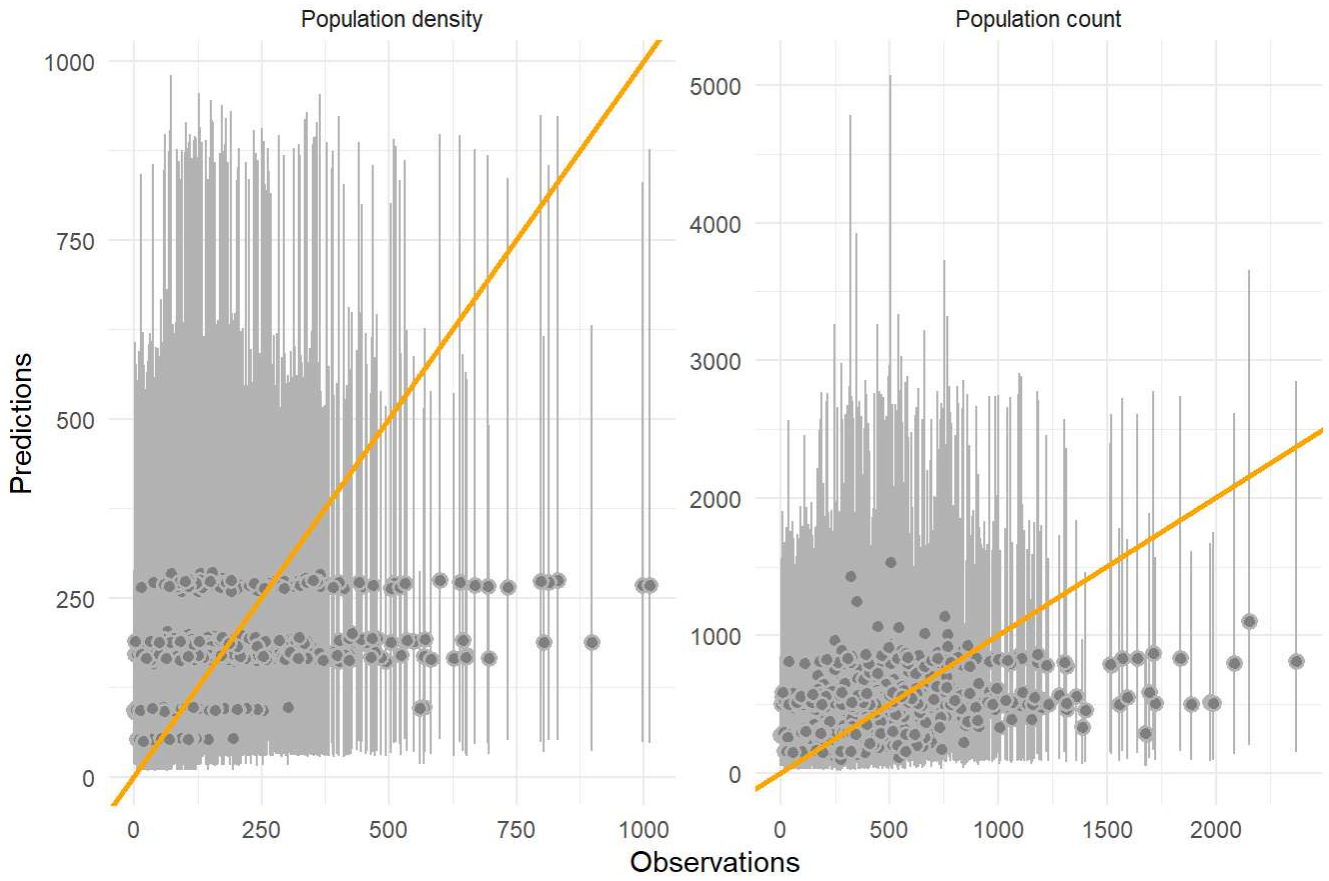


Figure 8: Comparison of observations with predictions from model 1

We see that the predicted population density has now five modes that corresponds to the five settlement types.

Partial pooling

Let's try now **partial pooling**. Partial-pooling involves a hierarchical set-up for the priors, that is, each α_t prior distribution will depend on an overarching α prior distribution. In other words, α defines a country-wide distribution that constrains each α_t to be in a similar range.

$$
\begin{aligned}
population \backsim Poisson(pop\_density * settled\_area) \\
pop\_density \backsim Lognormal(\alpha_t, \sigma) \\
\alpha_t \backsim Normal(\alpha, \nu_\alpha) \\
\alpha \backsim Normal(0, 100) \\
\nu_\alpha \backsim Uniform(0, 100) \\
\sigma \backsim Uniform(0, 100)
\end{aligned}
\tag{2}
$$

In `stan` we will write the mode like that:

Hide

```
// Model 2: Hierarchical alpha by settlement type
parameters{
  ...
  // hierarchical intercept by settlement
  vector[ntype] alpha_t;
  real alpha;
  real<lower=0> nu_alpha;
}
model{
  ...
  // hierarchical intercept by settlement
  alpha_t ~ normal(alpha, nu_alpha);
  alpha ~ normal(0, 100);
  nu_alpha ~ uniform(0, 100);
}
```

We add the new parameters to the parameters to monitor and we run the model:

Hide

```
pars <- c('alpha_t','alpha', 'nu_alpha', 'population_hat', 'density_hat')

# mcmc
fit2 <- rstan::stan(file = file.path('tutorial2_model2.stan'),
                    data = stan_data_model1,
                    iter = warmup + iter,
                    chains = chains,
                    warmup = warmup,
                    pars = pars,
                    seed = seed)
```

```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior varia
nces and tail quantiles may be unreliable.
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess
```

We get a warning about the Tail Effective Samples Size. For more information read here (%5B%3Chttps://mc-stan.org/misc/warnings.html#tail-ess%3E%5D(https://mc-stan.org/misc/warnings.html#tail-es)). A simple answer is to run the algorithm for longer by tweaking `warmup` and `iter` options for more iterations:

Hide

```
warmup <- 500
iter <-1000

# mcmc
fit2 <- rstan::stan(file = file.path('tutorial2_model2.stan'),
                    data = stan_data_model1,
                    iter = warmup + iter,
                    chains = chains,
                    warmup = warmup,
                    pars = pars,
                    seed=seed)
```

Tuning the Markov chains did result in the model convergence.

We can now compare the estimated parameters between a no pooling and a partial pooling framework.

Code

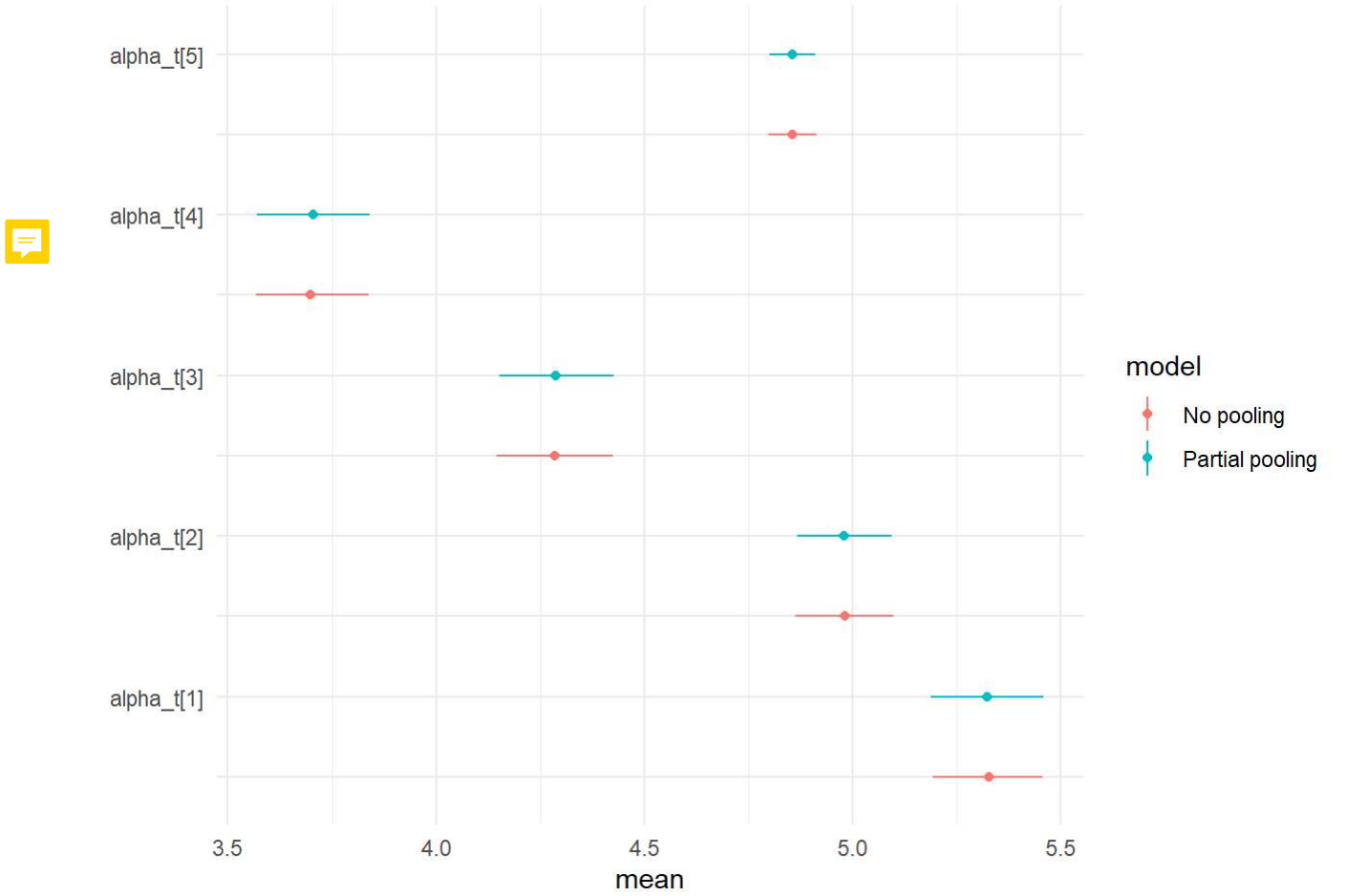


Figure 9: Comparison between alpha_t estimated independently and hierarchically

There are not striking differences between the two models. Table 1 shows how incorporating the settlement type grouping has a direct effect on the prediction.

Code

Table 1: Goodness-of-metrics comparison between complete pooling, no pooling, and partial pooling

model	Bias	Inaccuracy	Imprecision	Mean Confidence Interval Size
Complete pooling	61.83258	265.7909	337.7841	1800.707
No pooling	46.21468	234.3674	307.3298	1495.538
Partial pooling	46.04422	234.2140	307.1502	1499.183

Estimating α by settlement type does improve the model as shown by smaller residuals (*Bias* and *Imprecision*) and less variation (*Inaccuracy*). The hierarchical model does reduce slightly *Bias* but not strikingly, which indicates that the number of survey sites per settlement type was big enough to estimate the α_t independently, in a no-pooling framework.

We will see however in the next section what is a second advantage of hierarchical modelling.

A hierarchical intercept by settlement type and region

Taking into account the settlement type stratification of population count leads to more accurate predictions and reduces prediction uncertainty (see *Mean Confidence Interval Size* in Table 1).

We can include further grouping in the modelling, such as administrative divisions.

Let's include region in the modelling:

$$population \backsim Poisson(pop\_density * settled\_area)$$
$$pop\_density \backsim Lognormal(\alpha_{t,r}, \sigma)$$
$$\alpha_{t,r} \backsim Normal(\alpha_t, \nu_t)$$
$$\alpha_t \backsim Normal(\alpha, \nu)$$
$$\alpha \backsim Normal(0, 100)$$
$$\nu \backsim Uniform(0, 100)$$
$$\nu_t \backsim Uniform(0, 100)$$
$$\sigma \backsim Uniform(0, 100)$$

The model is translated in `stan` as :

Hide

```
// Model 3: Hierarchical alpha by settlement type and region

data{
  ...
  int<lower=1> nregion; //number of regions
  int<lower=1,upper=nregion> region[n]; // region
}
parameters{
  ...
  // hierarchical intercept by settlement and region
  real alpha;

  vector[ntype] alpha_t;
  vector[nregion] alpha_t_r[ntype];

  real<lower=0> nu_alpha;
  real<lower=0> nu_alpha_t;
}
transformed parameters{
  vector[n] pop_density_mean;

  for(idx in 1:n){
    pop_density_mean[idx] = alpha_t_r[type[idx],region[idx]];
  }
}
model{
  ...
  pop_density ~ lognormal(pop_density_mean, sigma );

  // hierarchical intercept by settlement and region
  alpha ~ normal(0, 100);
  nu_alpha ~ uniform(0, 100);
  nu_alpha_t ~ uniform(0, 100);

  alpha_t ~ normal(alpha, nu_alpha);

  for(t in 1:ntype){
    alpha_t_r[t,] ~ normal(alpha_t[t], nu_alpha_t);
  }
}
generated quantities{
  ...
  density_hat[idx] = lognormal_rng( alpha_t_r[type[idx], region[idx]], sigma );
}
```

Note the new `transformed parameters` block: it is useful to keep your `model` block tidy with only the stochastic relationships. We add the region indexing to the data list prepared for stan:

Hide

```
# prepare data for stan
stan_data_model3 <- list(
  population = data$N,
  n = nrow(data),
  area = data$A,
  type = data$type,
  ntype= n_distinct(data$type),
  region = data$region,
  nregion = n_distinct(data$region)
  )
```

And run the model with the new parameters to monitor:

Hide

```
pars <- c('alpha','alpha_t','alpha_t_r','nu_alpha', 'sigma','population_hat',  'dens
ity_hat')

# mcmc
fit3 <- rstan::stan(file = file.path('tutorial2_model3.stan'),
                    data = stan_data_model3,
                    iter = warmup + iter,
                    chains = chains,
                    warmup = warmup,
                    pars = pars,
                    seed = seed)
```

No problem of model convergence.

Let's look at the estimated $\hat{\alpha}_{t,r}$. Since there are 1 $\hat{\alpha}$, 5 $\hat{\alpha}_t$ and 5x11 $\hat{\alpha}_{t,r}$, we will zoom in two settlement types in particular, 1 and 4. To do so, remember how the parameter names are constructed:

- `alpha_t` is a vector of dimension 5, such that `alpha_t[1]` represents $\hat{\alpha_1}$, the estimate associated to settlement 1
- `alpha_t_r` is a vector of dimension 5*11, such that `alpha_t[1,1]` represents $\hat{\alpha}_{1,1}$, the estimate associated to settlement 1 and region 1.

Hide

```
alpha_4_r <- paste0('alpha_t_r[4,', 1:11, ']')
alpha_1_r <- paste0('alpha_t_r[1,', 1:11, ']')

stan_plot(fit3, pars=c('alpha','alpha_t[1]','alpha_t[4]', alpha_1_r, alpha_4_r),
          fill_color='orange')
```

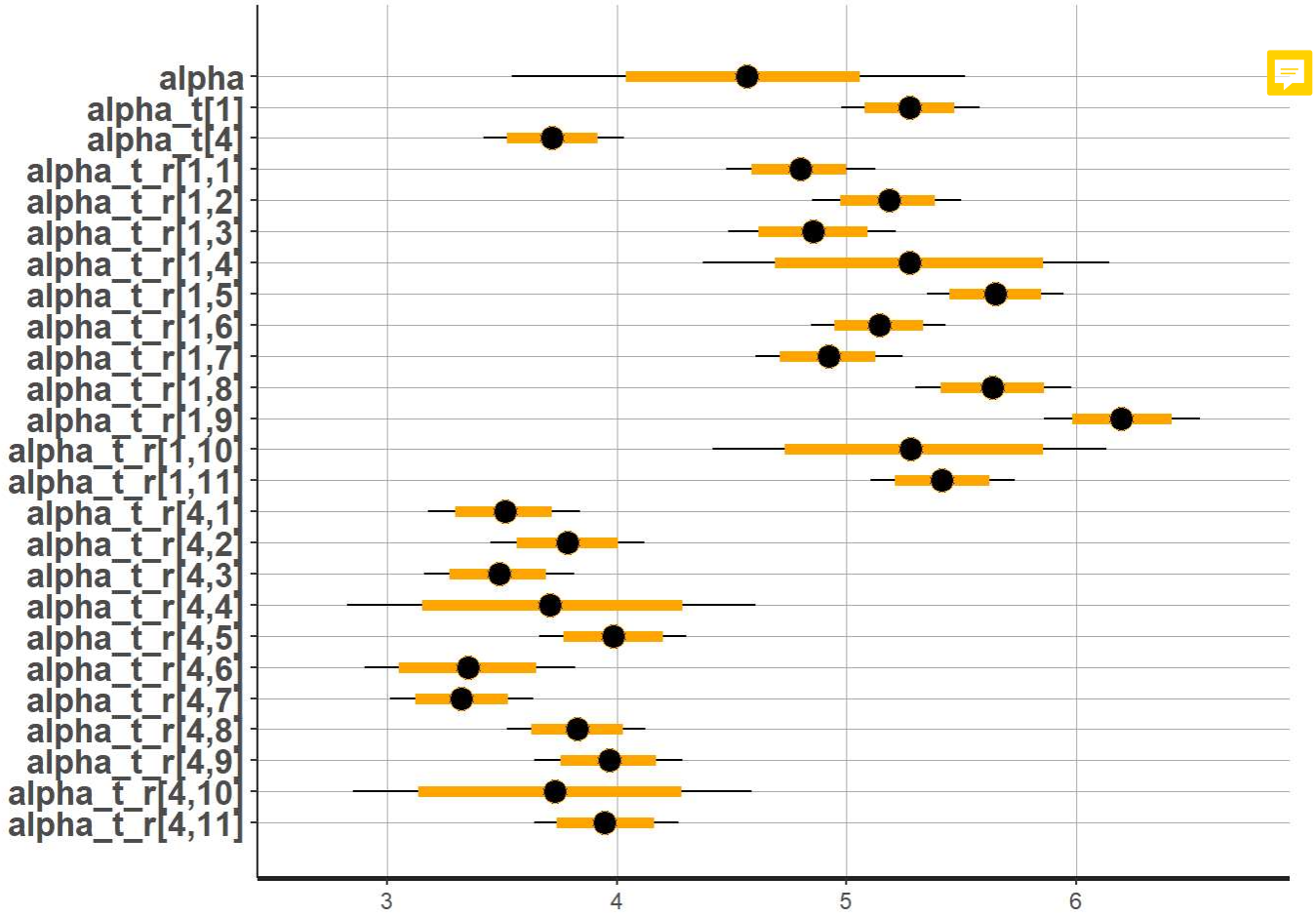


Figure 10: Estimated alpha by region for settlement 1 and 4

We see that:

1. $\hat{\alpha}$ has a estimated distribution that overlaps the two settlement types, and wider confidence interval to account for all the uncertainty
2. $\hat{\alpha}_1$ and $\hat{\alpha}_2$ have two distinctive patterns
3. the two settlement-level patterns mask regional disparities

A fourth remark: look closely at region 4 and 10, that is to say $\hat{\alpha}_{1,4}$, $\hat{\alpha}_{1,10}$,$\hat{\alpha}_{2,4}$ and $\hat{\alpha}_{2,10}$. We see that the confidence intervals are larger than for any other region.

This is because those two regions comprise only one settlement type (5) in the data (see Figure 6) and thus neither settlement type 1 nor settlement type 4. Therefore there is no data to inform those $\alpha_{t,r}$, but because of the hierarchy, they can be estimated from the global parameter α_t. And indeed we see that their estimated means closely match the respective $\hat{\alpha}_t$.

This feature of hierarchical setting is a **great advantage compared to the non-pooling setting**. When predicting population count across Nigeria, we might find out settlement type in a region that was not represented in our dataset, typically type 1, 2, 3 or 4 in region 4 or 10. In a non-pooling setting we won't have any parameter estimated because the models are estimated independently per level combination that is for a type x region. In a partial-pooling setting or hierarchical model we are covered by the global α_t. Obviously there will be more uncertainty in the prediction for this specific area because no data could be used to fit the model.

This ability of a hierarchical model holds also if an entire grouping was not sampled, typically if a region was not included in the survey. However it means that this missing region will be estimated with observations from the other sampled regions. If the missing region has very different characteristics, the model will not be able to capture them.

A hierarchical intercept by settlement type, region, state and local government area

To maximize the information retrieved from the reported administrative divisions, we can use a hierarchy with four levels: settlement type, region, state and local government area.

The additional administrative level however are organised in a *nested setting* which is particular case from hierarchical data.

Indeed we were previously in a *crossed hierarchical setting*: we would expect each settlement type to be present in every region. We are not however expecting each state to be present in every region. The hierarchy is strict.

To account for a nested setting, the **indexing is key**. We need the indexing to be nested, that is *dense* for each level. Luckily our data has already been coded this way:

Hide

```
data %>% group_by(region, state) %>% summarise(`Number of survey sites`=n()) %>%
  kbl() %>% kable_minimal()
```

region	state	Number of survey sites
1	1	101
2	1	102
2	2	1
3	1	100
4	1	43
5	1	99
6	1	101
7	1	102
8	1	96
9	1	42
9	2	81
9	3	86
10	1	42
10	2	39
11	1	106

Each state's index is nested within the parent region.

Including the additional levels of hierarchy in the `stan` code looks very similar to previous models:

Hide

```
// Model 4: Hierarchical alpha by settlement type , region, state local
data{
  ...
  int<lower=1> nstate[nregion]; //number of states
  int<lower=1,upper=max(nstate)> state[n]; // state

  int<lower=1> max_nlocal; // max local with data in one state
  int<lower=0> nlocal[nregion, max(nstate)]; // number of local per region, per stat
e
  int<lower=1,upper=max_nlocal> local[n]; // local

}
parameters{
  ...

  // hierarchical intercept by settlement, region, state, local
  real alpha;
  vector[ntype] alpha_t;
  vector[nregion] alpha_t_r[ntype];
  vector[max(nstate)] alpha_t_r_s[ntype, nregion];
  vector[max_nlocal] alpha_t_r_s_l[ntype, nregion, max(nstate)];

  real<lower=0> nu_alpha;
  real<lower=0> nu_alpha_t;
  real<lower=0> nu_alpha_r;
  real<lower=0> nu_alpha_s;
}
transformed parameters{
  ...
  for(idx in 1:n){
    pop_density_mean[idx] = alpha_t_r_s_l[type[idx], region[idx], state[idx], local
[idx]];
  }
}
model{
  ...
  // hierarchical intercept by settlement, region, state and local
  alpha ~ normal(0, 100);

  alpha_t ~ normal(alpha, nu_alpha);

  for(t in 1:ntype){
    alpha_t_r[t,] ~ normal(alpha_t[t], nu_alpha_t);
    for(r in 1:nregion){
      alpha_t_r_s[t,r,1:nstate[r]] ~ normal(alpha_t_r[t,r], nu_alpha_r);
      for(s in 1:nstate[r]){
        alpha_t_r_s_l[t,r,s,1:nlocal[r,s]] ~ normal(alpha_t_r_s[t,r,s], nu_alpha_s);
      }
    }
  }
```

```
  nu_alpha ~ uniform(0, 100);
  nu_alpha_t ~ uniform(0, 100);
  nu_alpha_r ~ uniform(0, 100);
  nu_alpha_s ~ uniform(0, 100);
  ...
}

generated quantities{
  ...
  density_hat[idx] = lognormal_rng( alpha_t_r_s_l[type[idx], region[idx], state[id
x], local[idx]], sigma );
}
```

The data preparation is slightly more complex. 1. `nstate` should indicate the number of states per region and thus be an array of size `nregion`.

Hide

```
# prepare data for stan

# providing number of state by region
nstate <-  data %>% group_by(region) %>% summarise(nstate=n_distinct(state)) %>% ung
roup() %>% select(nstate) %>% unlist(use.names = F)
nstate
```

```
##  [1] 1 2 1 1 1 1 1 1 3 2 1
```

2. `nlocal` should indicate the number of local government areas per region per state and thus be a matrix of size `nregion x nstate`. Naturally this matrix will have NA (there is not the same number of local areas in each state). We replace those NAs by zero because `stan` does not accept NAs in the data.

Hide

```
# providing number of local areas by region and state
nlocal <-  data %>% group_by(region, state) %>% summarise(nlocal=n_distinct(local))
 %>% ungroup() %>%
  pivot_wider(id_cols=region, names_from = state, values_from = nlocal) %>% select(-
region)
nlocal[is.na(nlocal)] <- 0 # stan doesn't accept NA in the data
nlocal
```

```
## # A tibble: 11 x 3
##      `1`   `2`   `3`
##    <int> <int> <int>
##  1    24     0     0
##  2    17     1     0
##  3    17     0     0
##  4    14     0     0
##  5     7     0     0
##  6    13     0     0
##  7    30     0     0
##  8     6     0     0
##  9    15    18    29
## 10    12    12     0
## 11    17     0     0
```

We see that `nlocal` has 11 rows (the number of regions) and 3 columns (the maximum number of states per region).

Hide

```
stan_data_model4 <- list(
  population = data$N,
  n = nrow(data),
  area = data$A,
  type = data$type,
  ntype= n_distinct(data$type),
  region = data$region,
  nregion = n_distinct(data$region),
  nstate = nstate,
  state = data$state,
  nlocal = nlocal,
  max_nlocal= max(nlocal, na.rm=T),
  local = data$local
  )
```

To Doug: here starts the issue

I wanted to show the nested indexing and also some priors tweaking. i tried (settlement type, region, state and local government area) and (settlement type, region and local government area) - no success. With some mcmc parameters/prior distribution combination i achieved no divergence but i have mixing errors (not visible on the traceplot). so i could maybe end up on the mixing error and saying that the data doesnt support this modelling?

Other question: i'm considering adding a section on hierarchical variance. Suitable? Mentioned or fully developed?

Addressing convergence issues

If we use the same mcmc setting as previously, the model takes more than 5 hours to run and fail to converge. There are a few adaptations that we can add:

1. Adapt initialisation

Markov chains need be initialised to start exploring the parameter space, that is to be given a position where to begin their walks. As from now we have left it to `stan` to choose the starting point whose default is *to randomly generate initial values between -2 and 2 on the unconstrained support*. To speed up the process we can give the algorithm an hinch about where to begin.

For that purpose we write a function that takes as input our data and adds small random variations to it.

Hide

```
inits <- function(md, nchains=chains){

  set.seed(md$seed)
  inits.out <- list()

  for (c in 1:nchains){
    inits.i <- list()
    # intercept
    inits.i$alpha <- runif(1, 4, 6)
    inits.i$alpha_t <- runif(md$ntype, 0, 2.5)
    inits.i$alpha_t_r <- matrix(runif(md$ntype * md$nregion, 3, 6), nrow=md$ntype, n
col=md$nregion)
    inits.i$alpha_t_r_s <- array(runif(md$ntype * md$nregion * max(md$nstate), 3, 6
),
                                 dim=c(md$ntype, md$nregion, max(md$nstate)))
    inits.i$alpha_t_r_s_l <- array(runif(md$ntype * md$nregion * max(md$nstate)*md$m
ax_nlocal, 3, 6),
                                 dim=c(md$ntype, md$nregion, max(md$nstate), md$m
ax_nlocal))
    inits.i$nu_alpha <- runif(1, 0, 1)
    inits.i$nu_alpha_t <- runif(1, 0, 1)
    inits.i$nu_alpha_r <- runif(1, 0, 1)
    inits.i$nu_alpha_s <- runif(1, 0, 1)
    # variance
    inits.i$sigma <- runif(1, 0.1, 0.5)
    # population density
    inits.i$pop_density <- rlnorm(md$n, log(md$population / md$area), 0.25)

    inits.out[[c]] <- inits.i
  }
return(inits.out)}
```

2. Constrain `sigma`

The prior distribution for sigma (`halfnormal(0,100)`) comprise unlikely values. A hint is given by the observed log of the geometric standard deviation of population density (that is σ in a Lognormal): 0.87. We want to constrain the prior to be around 1 but we still want extreme

values to be covered to avoid overfitting our data. The Cauchy distribution (https://en.wikipedia.org/wiki/Cauchy_distribution) is a good candidate. Figure 11 compares different prior distributions for σ.

Code

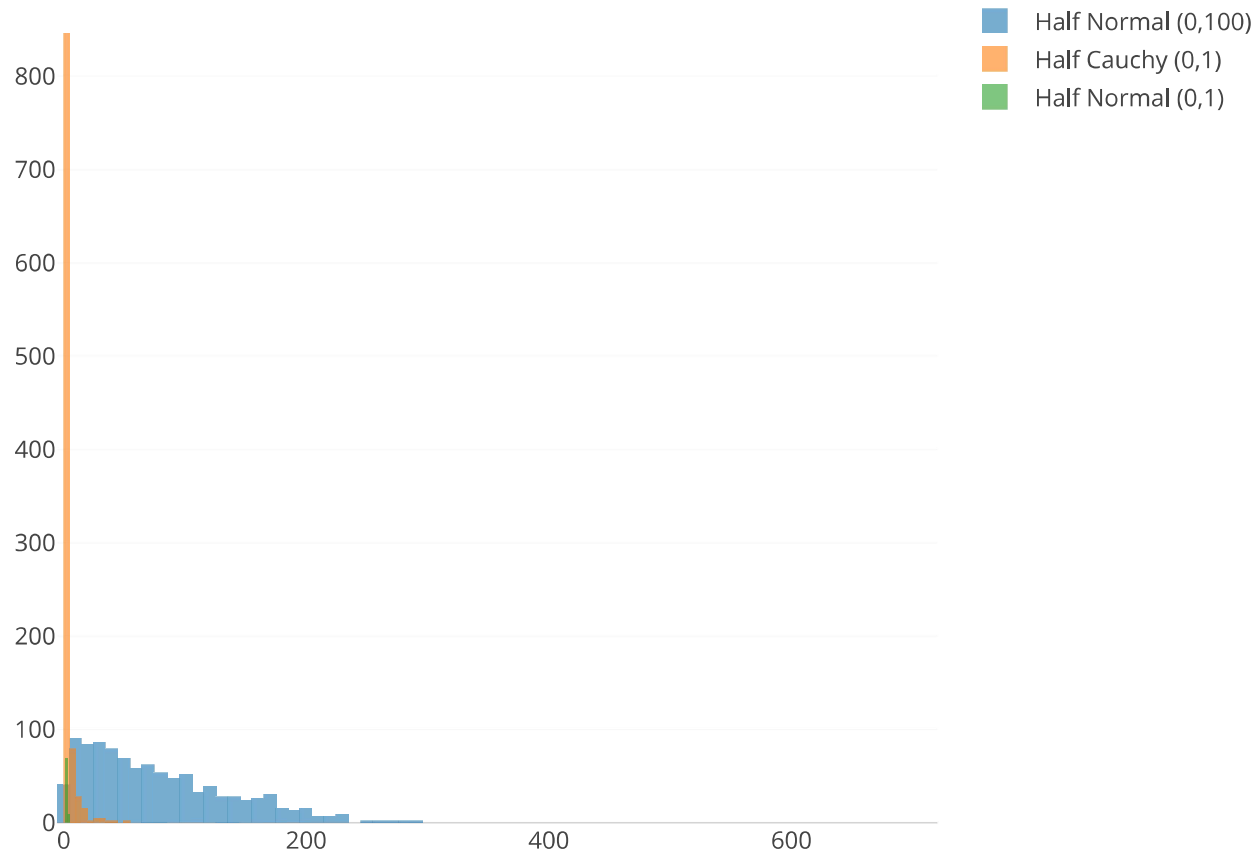


Figure 11: Prior distributions comparison for sigma

Don't hesitate to use the interactive tools to understand the difference between a Half Normal (0,1) and a Half Cauchy (0,1). Below how we integrate the change in `stan`.

Hide

```
// Model 5: Hierarchical alpha by settlement type , region, state local with a Cauch
y prior for sigma

model{
  ...

  // variance with Cauchy prior
  sigma ~ cauchy(0, 1);
}
```

3. Control sampler behaviour

The option `control` in the `stan` call allows to adjust the way the chains are exploring the space. A critical setting in our context is `max_treedepth` : it controls t*he cap on the depth of the trees that it evaluates during each iteration. When the maximum allowed tree depth is reached it indicates that NUTS is terminating prematurely to avoid excessively long execution time*.

We will also increase the running time by modifying `iter` and `warmup` .

We now gather all the three modifications and run the model with full hierarchy.

Hide

```
pars <- c('alpha','alpha_t','alpha_t_r','alpha_t_r_s', 'nu_alpha','nu_alpha_t', 'nu_
alpha_r', 'sigma','population_hat',  'density_hat')

warmup <- 500
iter <- 500
# mcmc
start <- Sys.time ()
fit5 <- rstan::stan(file = file.path('tutorial2_model5.stan'),
                    data = stan_data_model4,
                    iter = warmup + iter,
                    chains = chains,
                    warmup = warmup,
                    pars = pars,
                    seed = seed,
                    init = inits(stan_data_model4),
                    control = list(max_treedepth = 12))
time <- Sys.time () - start
time
beep()
fit <- list()
fit$fit <- fit5
fit$time <- time
saveRDS(fit, 'fit5.rds')
```

Attempt with only settlement type region and local

Hide

```
nlocal <-  data %>% group_by(region) %>% summarise(nlocal=n_distinct(local)) %>% ung
roup() %>%
  select(nlocal) %>% unlist(use.names = F)


stan_data_model6 <- list(
  population = data$N,
  n = nrow(data),
  area = data$A,
  type = data$type,
  ntype= n_distinct(data$type),
  region = data$region,
  nregion = n_distinct(data$region),
  nlocal = nlocal,
  local = data$local
  )
inits6 <- function(md, nchains=chains){

  set.seed(md$seed)
  inits.out <- list()

  for (c in 1:nchains){
    inits.i <- list()
    # intercept
    inits.i$alpha <- runif(1, 3, 5)
    inits.i$alpha_t <- runif(md$ntype, 3, 5)
    inits.i$alpha_t_r <- matrix(runif(md$ntype * md$nregion, 3, 5), nrow=md$ntype, n
col=md$nregion)
    inits.i$alpha_t_r_l <- array(runif(md$ntype * md$nregion * max(md$nlocal), 3, 5
),
                                 dim=c(md$ntype, md$nregion, max(md$nlocal)))
    inits.i$nu_alpha <- runif(1, 0, 1)
    inits.i$nu_alpha_t <- runif(1, 0, 1)
    inits.i$nu_alpha_r <- runif(1, 0, 1)
    inits.i$nu_alpha_s <- runif(1, 0, 1)
    # variance
    inits.i$sigma <- runif(1, 0.1, 0.5)
    # population density
    inits.i$pop_density <- rlnorm(md$n, log(md$population / md$area), 0.25)

    inits.out[[c]] <- inits.i
  }
return(inits.out)}
pars <- c('alpha','alpha_t','alpha_t_r','alpha_t_r_l', 'nu_alpha','nu_alpha_t', 'nu_
alpha_r', 'sigma','population_hat',  'density_hat')

warmup <- 500
iter <- 500
# mcmc
start <- Sys.time ()
fit6 <- rstan::stan(file = file.path('tutorial2_model6.stan'),
```

```
                    data = stan_data_model6,
                    iter = warmup + iter,
                    chains = chains,
                    warmup = warmup,
                    pars = pars,
                    seed = seed,
                    init = inits6(stan_data_model6),
                    control = list(max_treedepth = 12))
time <- Sys.time () - start
time
```

References

Weber, Eric M., Vincent Y. Seaman, Robert N. Stewart, Tomas J. Bird, Andrew J. Tatem, Jacob J. McKee, Budhendra L. Bhaduri, Jessica J. Moehl, and Andrew E. Reith. 2018. “Census-Independent Population Mapping in Northern Nigeria.” *Remote Sensing of Environment* 204 (January): 786–98. https://doi.org/10.1016/j.rse.2017.09.024 (https://doi.org/10.1016/j.rse.2017.09.024).