Introduction
Advanced model diagnostics
Gridded population prediction
Predicting distributions
Aggregating prediction
References

Population modelling for census support

Code

Tutorial 4: Advanced model diagnostics and prediction

Edith Darin and Douglas Leasure

Compiled on 21/08/2021

WorldPop

Code

Introduction

From Tutorial 1 to Tutorial 3, we built a **bottom-up population model**, modelling brick by modelling brick**.** In Tutorial 1 we model population count as a **Poisson-Lognormal compound,** accounting for settled area. In Tutorial 2, we added a **hierarchical random intercept** by settlement type and region, that differentiates parameter estimation by the natural clustering of the observations. In Tutorial 3 we eventually modelled small-scale variations in population density by adding **a linear regression based on geospatial covariates**. We covered thus all the fundamental modelling blocks of WorldPop bottom-up population model, described in Leasure et al. (2020).

We will cover in this tutorial advanced model diagnostics for a complete goodness-of-fit assessment. Once we check that the model successfully passes the different diagnostics, we will then predict population count for every grid cell of the study area. Predicting population will be the opportunity to talk about estimates uncertainty.

Goals

1. Understand posterior predictive check
2. Understand spatial autocorrelation
3. Understand cross-validation and overfitting
4. Predict population for every grid cell of the study area
5. Visualise prediction uncertainty
6. Aggregate prediction in custom spatial units

Supporting readings

- Evaluating regression models (https://www.bayesrulesbook.com/chapter-10.html) by the bayesrules book, on poseterior predictive check and the difference between correct estimation and correct modelling.

- Spatial autocorrelation (https://mgimond.github.io/Spatial/spatial-autocorrelation.html) explained by Manuel Gimond, Colby College
- Tidy data and Bayesian analysis make uncertainty visualization fun (https://www.mjskay.com/presentations/openvisconf2018-bayes-uncertainty-2.pdf) by Matthew Kay, University of Michigan. It hammers the importance of communicating uncertainty with estimates and explains how a Bayesian framework addresses that issue.
- You're fit and you know it: overfitting and cross-validation (https://medium.com/the-sound-of-ai/youre-fit-and-you-know-it-overfitting-and-cross-validation-90a3a9f67c74), by Andy Elmsley. Good introduction to cross-validation despite applying it to machine learning problems.

Extra packages

We will use some additional packages, mainly for GIS manipulation:

- `raster` for predicting the gridded population in a raster format
- `sf` for aggregating the prediction in custom spatial unit
- `tmap` to plot spatial data
- `RColorBrewer` to use nice color palette in plots

Hide

```
install.packages('raster')
install.packages('RColorBrewer')
install.packages('sf')
install.packages('tmap')
```

Advanced model diagnostics

Before predicting population count for the entire study area we want to make sure that the model is correct. We thus perform additional checks: on the estimated parameters with a *posterior predictive check*, on the spatial autocorrelation of the residuals with a *spatial correlogram* and on the predicted population count at study site with a *cross-validation approach*.

Let's load the last fitted model in Tutorial 3, that is with a random slope effect on `x_4`.

Hide

```
model_fit <- readRDS('tutorials/tutorial3/tutorial3_model2_fit.rds')
```

Posterior predictive check

Bayesien estimation relies on choosing priors for the parameters. Those priors should reflect expert knowledge about the quantity of interest that is, then, confronted with the observations. The prior model defines the space for the posterior estimation. We should thus ensure that the prior support did not overly constrain the posterior. This is called a **posterior predictive check.**

We need first to extract the **estimated posterior distribution** from the `stanfit` object. We do it using the `extract` function from `rstan`.

Hide

```
alphas <- data.frame(
    # extact parameter
    posterior=extract(model_fit, pars='alpha')$alpha,
    parameter = 'alpha'
    )
glimpse(alphas)
```

```
## Rows: 2,000
## Columns: 2
## $ posterior <dbl> 5.317130, 4.575211, 3.943799, 4.408326, 4.419462, 4.880423, ~
## $ parameter <chr> "alpha", "alpha", "alpha", "alpha", "alpha", "alpha", "alpha~
```

`alphas` contains the estimated alpha for each iteration post-warmup (500) of each Markov chain (4).

We want to recreate the **prior distribution**. In model 2 of Tutorial 3, we used as prior for α a

$$Normal(0, 100)$$

We sample from this prior distribution, the same number of draws that we have for the posterior distribution (that is 2000).

Hide

```
# retrieve stan parameter
post_warmup_draws <- model_fit@stan_args[[4]]$iter - model_fit@stan_args[[4]]$warmup
chains <- length(model_fit@stan_args)

# simulate from the prior distribution
alphas$prior <- rnorm(chains * post_warmup_draws, 0, 100)
```

We build a similar dataframe to `alphas` for contrasting the prior distribution with the posterior distribution of σ and bind together with `alphas` .

Code

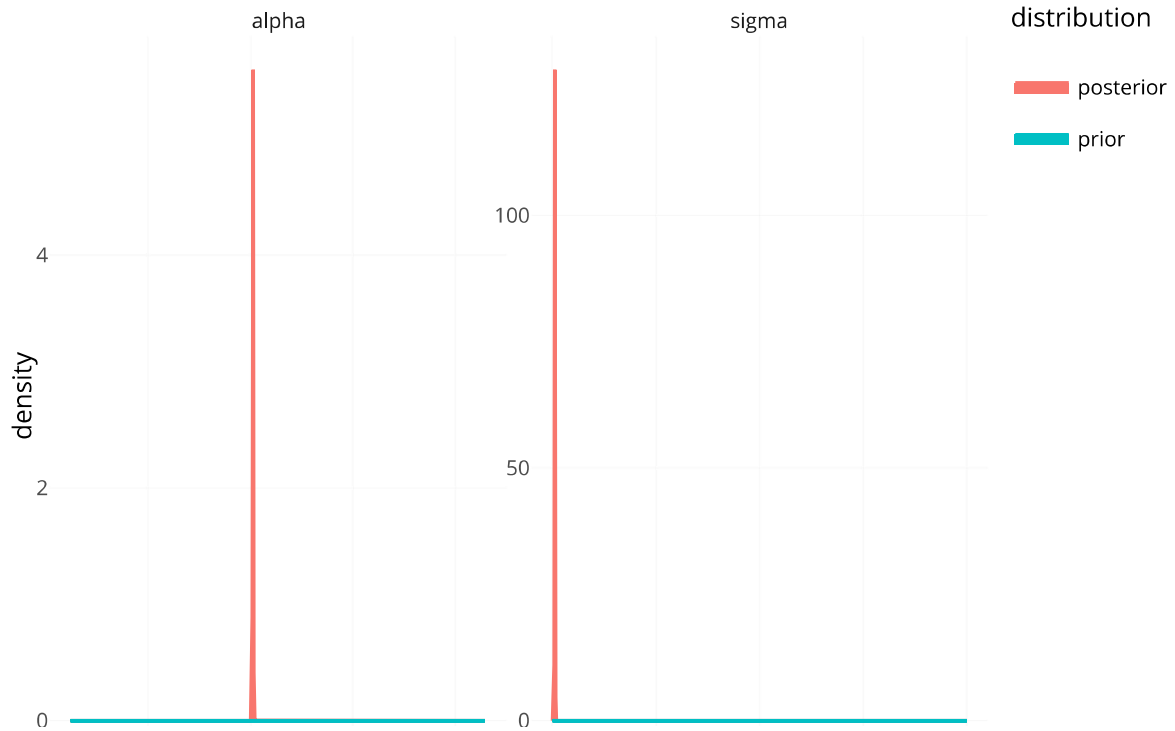


Figure 1: Posterior predictive checks for alpha and sigma

Figure 1 shows a posterior predictive check for both α and σ. It compares the prior distribution that we set in the model and the estimated posterior distribution. Don't hesitate to zoom in, as the range of values is very different between the prior and the posterior.

We confirm that our priors were not informed, with a very wide range of possible values. The unique constrain we can observe is the lower bound on σ at 0 but this is normal as a variance has to be positive. We could have indicated a lower range of values to speed up the model estimation.

Since the prior for α constrained the α_t, it is not strictly necessary to perform the same test for α_t.

> **Question**: Can you check the posterior distribution for β and interpret it?
>
> Remember that we have two parameters related to β: (1) `beta_random` and (2) `beta_fixed`. They are both of dimension 5 for two separate reasons: `beta_random` because there is 5 settlement types and `beta_fixed` because there is 5 covariates modelled with a fixed effect.
>
> Coding this question will help you to familiarize with the structure of a `stanfit` object.

▶
Click for the solution

Spatial auto-correlation of the residuals

After checking the parameter posterior, we now want to check **the assumption of independence between the observations** that is if the model is able to account for the correlations between observations. This is done by looking at the correlations of the residuals.

One correlation we are particular interesting in is the spatial auto-correlation (https://mgimond.github.io/Spatial/spatial-autocorrelation.html), that is when the outcome variable is highly determined by spatial locations. Population density is indeed likely to be more similar between close locations than distanced location.

In our model we took into account the spatial context through two components: the hierarchical structure and the geospatial covariates. We need to check if that is sufficient to address the spatial auto-correlation. To that end, we look at the spatial distribution of the residuals and use a spatial correlogram of Moran's I statistic (https://en.wikipedia.org/wiki/Moran's_I) calculated at ranges from 0 to 100 km. For more information, check the correlog function (https://www.rdocumentation.org/packages/pgirmess/versions/1.7.0/topics/correlog) from the `pgirmess` package.

The coordinates of the study sites were not communicated with the data for confidentiality reason so we will reproduce the correlogram from Leasure et al. (2020) where red dots indicate a statistically significant Moran's I statistic (P<0.05).

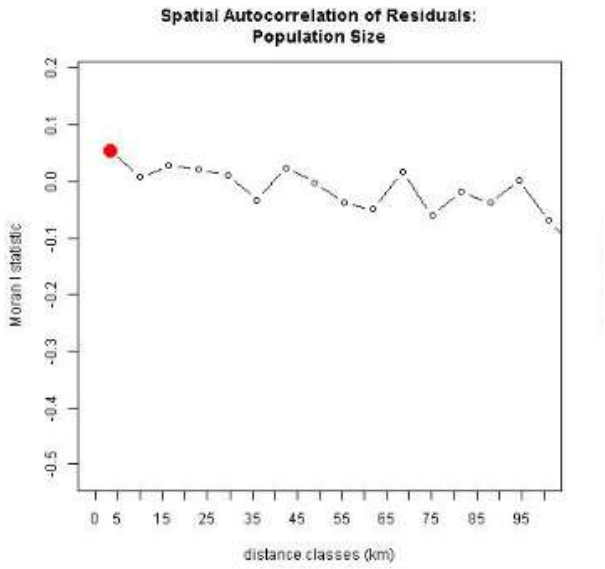


Figure 3: Example of a spatial correlogram between model residuals from Leasure et al. (2020).

Figure 3 shows that the Moran's I oscillates around zero or is not significant.

It means that the model correctly accounts for the spatial correlation of the observations.

Cross-validation of model predictions

After ensuring no spatial auto-correlation remains in the residuals, we revisit the **goodness-of-fit metrics based on population predictions**.

In tutorial 1 to 3, we check the goodness-of-fit of the modelled predictions **in sample**, that is predicting population and comparing it to the observations for the study sites that were already used to fit the model.

In-sample checks do not account for model overfitting. Overfitting occurs when the model does not only reproduce the information from the observations but also the noise. We want the model to be able to accurately predict population also unknown new study sites. This real-world scenario can be reproduced by keeping a percentage of the observations for model fitting and leaving the remaing study sites for predicting, a mechanism called **cross-validation.**

We need first to split our data in two, a training set (70%) that will be used for fitting the model and a predicting set (30%) that will be used for model prediction:

Hide

```
#load data
seed <- 1789
data <- readxl::read_excel(here('tutorials/data/nga_demo_data.xls'))
data <- data %>%
  mutate(
    id = as.character(1:nrow(data)),
    pop_density=N/A
  )
covs <- read_csv(here('tutorials/data/covs_scaled.csv'))

# sample observations
train_size <-  round(nrow(data)*0.7)
train_idx <- sample(1:nrow(data), size = train_size, replace=F)

# build train datasets
train_data <- data[train_idx,]
train_covs <- covs[train_idx,]

# build test datasets
test_data <- data[-train_idx,]
test_covs <- covs[-train_idx,]
```

In `stan` cross-validation can be done simultaneously with model fitting by providing the testing dataset to the `generated quantities` modelling block. Because all data has to be defined for each set it lengthens substantively the code.

Hide

```
data{

  int<lower=0> n_train; // number of microcensus study sites in training set
  int<lower=0> n_test; // number of microcensus study sites in predicting set

  int<lower=0> population_train[n_train]; // count of people

  vector<lower=0>[n_train] area_train; // settled area in training
  vector<lower=0>[n_test] area_test; // settled area in testing

    // fixed slope
  int<lower=0> ncov_fixed; // number of covariates -1
  matrix[n_train, ncov_fixed] cov_fixed_train; // covariates in training
  matrix[n_test, ncov_fixed] cov_fixed_test; // covariates in training

    // random slope
  vector[n_train] cov_random_train;
  vector[n_test] cov_random_test;

  int<lower=0> ntype; // number of settlement types
  int<lower=0> type_train[n_train]; // settlement type in train
  int<lower=0> type_test[n_test]; // settlement type in test

  int<lower=1> nregion; //number of regions
  int<lower=1,upper=nregion> region_train[n_train]; // region
  int<lower=1,upper=nregion> region_test[n_test]; // region
}
parameters{
  // population density
  vector<lower=0>[n_train] pop_density_train;

  // hierarchical intercept by settlement, region, state, local
  real alpha;
  vector[ntype] alpha_t;
  vector[nregion] alpha_t_r[ntype];
  real<lower=0> nu_alpha;
  real<lower=0> nu_alpha_t;

  // variance
  real<lower=0> sigma;

  // slope
  row_vector[ncov_fixed] beta_fixed;
  vector[ntype] beta_random;
}
transformed parameters{
  vector[n_train] pop_density_train_mean;
  vector[n_train] beta_train;

  for(idx in 1:n_train){
    beta_train[idx] = sum( cov_fixed_train[idx,] .* beta_fixed) + cov_random_train[idx] * beta_
random[type_train[idx]];
    pop_density_train_mean[idx] = alpha_t_r[type_train[idx], region_train[idx]] + beta_train[id
x];
  }
}
model{
```

```
  // population totals
  population_train ~ poisson(pop_density_train .* area_train);
  pop_density_train ~ lognormal( pop_density_train_mean, sigma );

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

  //slope
  beta_fixed ~ normal(0,10);
  beta_random ~ normal(0,10);

  // variance
  sigma ~ uniform(0, 100);
}
generated quantities{
  int<lower=-1> population_hat[n_test];
  real<lower=-1> density_hat[n_test];
  vector[n_test] beta_hat;

  for(idx in 1:n_test){
    beta_hat[idx] = sum( cov_fixed_test[idx,] .* beta_fixed) + cov_random_test[idx] * beta_rand
om[type_test[idx]];
    density_hat[idx] = lognormal_rng( alpha_t_r[type_test[idx], region_test[idx]] + beta_hat[id
x], sigma );
    if(density_hat[idx] * area_test[idx]<1e+09){
      population_hat[idx] = poisson_rng(density_hat[idx] * area_test[idx]);
    } else {
      population_hat[idx] = -1;
    }
  }
}
```

We prepare the data for `stan` and run the model:

Hide

```
# prepare data
stan_data_xval <- list(
  population_train = train_data$N,
  n_train = nrow(train_data),
  n_test = nrow(test_data),

  area_train = train_data$A,
  area_test = test_data$A,

  ntype= n_distinct(data$type),
  type_train = train_data$type,
  type_test = test_data$type,

  nregion = n_distinct(data$region),
  region_train = train_data$region,
  region_test = test_data$region,

  ncov_fixed = ncol(covs) -1,
  cov_fixed_train = train_covs %>% select(-x4),
  cov_fixed_test = test_covs %>% select(-x4),
  cov_random_train = train_covs$x4,
  cov_random_test = test_covs$x4,

  seed=seed
)
# mcmc setting
chains <- 4
warmup <- 500
iter <- 500
pars <- c('alpha','sigma','beta_fixed','beta_random','alpha_t', 'population_hat',  'density_ha
t')

# mcmc
fitxval <- rstan::stan(file = file.path('tutorials/tutorial4/tutorial3_model2xval.stan'),
                   data = stan_data_xval,
                   iter = warmup + iter,
                   chains = chains,
                   warmup = warmup,
                   pars = pars,
                   seed = seed)
```

```
## Warning: There were 1 divergent transitions after warmup. See
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and median
s may be unreliable.
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess
```

We now have access to two predictions sets: one, `model_fit` , from the in-sample prediction and one, `fitxval` , from the cross-validation prediction.

We compare the goodness-of-metrics from the two models:

Code

Table 1: Goodness-of-metrics computed in-sample vs cross-validation

Model	Bias	Inaccuracy	Imprecision	Mean Confidence Interval Size
Cross-validation	45.03971	203.4583	271.0803	1271.949
In-sample	33.26182	192.1447	274.5590	1240.142

Table 1 shows that the uncertainty increased because of the size of the dataset is subtantially smaller. The bias however even decreased. The model passed the cross-validation check.

A good practice is to always assess the model fit on the testing set to stay on the safe side of overfitting.

Gridded population prediction

We will cover in this section **model predictions for the entire study area** at the grid cell level.

It requires extracting the estimated parameters distributions: $\hat{\alpha}_{t,r}, \hat{\sigma}, \hat{\beta}^{fixed}, \hat{\beta}_t^{random}$. These estimated parameters are then combined with the input rasters following this modified model for prediction:

$$
\begin{aligned}
population\_predicted &\sim Poisson(\hat{pop\_density} * settled\_area) \\
\hat{pop\_density} &\sim Lognormal(\hat{\mu},\, \hat{\sigma}) \\
\hat{\mu} &= \hat{\alpha}_{t,r} + \hat{\beta}_t^{random} X^{random} + \hat{\beta}^{fixed} X^{fixed}
\end{aligned}
\tag{1}
$$

X and $settled\_area$ are the input data at grid cell level:

Grid-cell based model prediction requires thus 10 input rasters:

- settled area
- settlement type
- administrative region
- the six covariates
- a mastergrid that defines the grid cells for which to predict the population. These grid cells are (1) contained inside the study area

    2. considered as settled.

Unfortunately we are not allowed to redistribute the settlement map that was provided by Oak Ridge National Laboratory (2018). We will thus provide the code to show all the steps of population prediction but you will not be able to run it.

We will focus on region 8 that contains only 671 110 cells instead of the 166 412 498 cells for the entire country.

Preparing data for prediction

To predict gridded population, we convert the rasters set to a table with: in rows, the grid cells and in column, each raster value. We don't need to keep the spatial structure of the raster for model prediction. Every grid cell can be estimated independently as long as we know its administrative region, settlement type and covariates attributes.

Hide

```
library(raster)
wd_path <- 'Z:/Projects/WP517763_GRID3/Working/NGA/ed/bottom-up-tutorial/'

# function that loads the raster and transforms it into an array
getRasterValues <- function(x){
  return( raster(paste0(wd_path,x))[])
}

# get a list of the rasters
rasterlist <- list.files(wd_path,
                         pattern='.tif', full.names = F)
# apply function to raster list
raster_df <- as_tibble(sapply(rasterlist, getRasterValues))
```

The raster table contains 671 110 rows and 10 columns. 95% of the rows are NAs because they are not considered as settled.

Hide

```
settled <- raster_df %>% filter(mastergrid==1)
settled %>% head() %>% kbl() %>% kable_minimal()
```

mastergrid	settled_area	x1	x2	x3	x4	x5	x6	gridcell_id	settlementType
1	0.3764444	-0.0819475	0.0126183	4.663749	-0.1662194	-0.1876594	-0.1262890	47	5
1	0.0235278	-0.0629407	0.0189274	4.664463	-0.1640482	-0.1852137	-0.1265523	48	5
1	0.0411736	-0.0036624	0.0000000	4.661014	-0.1278646	-0.1549067	-0.1385387	90	5
1	0.0176458	0.0634254	0.0000000	4.662555	-0.1018127	-0.1279219	-0.1385805	91	5
1	0.0117639	0.0478863	1.2744479	4.871751	0.9074216	1.0802944	0.2204351	208	5
1	0.0999931	0.0948045	1.3028392	4.877525	0.8835372	1.0512134	0.2195443	209	5

We need to **scale the covariates** with the scaling factors computed during the fitting stage.

Hide

```
#load scaling factor
scale_factor <- read_csv('tutorials/data/scale_factor.csv')

scaled_covs <- settled %>%
  # subset covs column
  dplyr::select(starts_with('x'), gridcell_id) %>%

  # convert table to long format
  pivot_longer(-gridcell_id, names_to='cov', values_to = 'value') %>%

  # add scaling factor
  left_join(scale_factor) %>%

  # scale covs
  mutate(value_scaled= (value-cov_mean)/cov_sd) %>%
  dplyr::select(gridcell_id, cov, value_scaled) %>%

  # convert table back to wide format
  pivot_wider(names_from = cov, values_from = value_scaled)

# replace the covs with their scaled version
raster_df <- raster_df %>% dplyr::select(-starts_with('x')) %>%
  left_join(scaled_covs)
```

Extracting estimated parameters

The second step is to **extract the estimated parameter distributions**. We use the model fitted with the full observations dataset and not the one from the cross-validation for maximum information.

Hide

```
#extract betas
beta_fixed <- t(rstan::extract(model_fit, pars='beta_fixed')$beta_fixed)
beta_random <- t(rstan::extract(model_fit, pars='beta_random')$beta_random)
#extract alphas
alphas <- rstan::extract(model_fit, pars='alpha_t_r')$alpha_t_r
#extract sigmas
sigmas <-  rstan::extract(model_fit, pars='sigma')$sigma
```

We first focus on producing $\hat{\beta}^{fixed} X^{fixed}$ from Equation (1), the covariates modelled with a fixed effect.

$\hat{\beta}^{fixed}$ is a `2000 x 5` matrix for the 5 covariates and 2000 estimations. We transposed it to multiply by X^{fixed}, a `310 512 X 5` matrix of covariates for every settled cells.

Hide

```
# extract covariates modelled with a fixed effect
cov_fixed <- settled %>%
  dplyr::select(x1:x3, x5:x6) %>%
  as.matrix()

cov_fixed <- cov_fixed %*% beta_fixed
```

We then produce $\hat{\beta}_t^{random} X^{random}$, the covariates modelled with a random effect.

$\hat{\beta}_t^{random}$ is a `2000 x 5` matrix for the 5 settlement types. We need to associate each grid cell with the correct $\hat{\beta}_t^{random}$ based on the grid cell settlement type and then multiply by X^{random} the covariates values.

Hide

```
beta_random <- as_tibble(beta_random)
# add settlement type to beta random
beta_random$settlementType <- 1:5

# extract covariates modelled with a random effect
cov_random <- settled %>%
  # subset random covariate and settlement type
  dplyr::select(settlementType,x4) %>%
  # associate correct estimated beta_t
  left_join(beta_random) %>%
  # multiply cov by slope
  mutate(
    across(starts_with('V'), ~ .x*x4)
  ) %>%
  # keep only the estimates
  dplyr::select(-settlementType, -x4) %>%
  as.matrix()
```

We eventually need to associate the correct $\hat{\alpha}_{t,8}$ to each grid cell.

$\hat{\alpha}_{t,8}$ has a similar format as $\hat{\beta}_t^{random}$. We will thus proceed in the same way.

Hide

```
# subset alpha_t for region 8
alpha_t_8 <- as_tibble(t(alphas[,,8]))
# assign settlement type
alpha_t_8$settlementType <- 1:5

alpha_predicted <- settled %>%
  dplyr::select(settlementType) %>%
  left_join(alpha_t_8) %>%
  dplyr::select(-settlementType) %>%
  as.matrix()
```

We can finally compute $\hat{\mu}$ as the sum of $\hat{\alpha}_{t,r}$, $\hat{\beta}_t^{random} X^{random}$ and $\hat{\beta}^{fixed} X^{fixed}$

Hide

```
# sum mu components
mu_predicted <- alpha_predicted + cov_fixed + cov_random
```

Predicting population count for every grid cell

To estimate the grid-cell population, we need to simulate from Equation (1) with $\hat{\mu}$ and $\hat{\sigma}$.

Hide

```
predictions <- as_tibble(mu_predicted) %>%
  # add grid cell id and the settled area to mu
  mutate(
      gridcell_id= settled$gridcell_id,
      settled_area = settled$settled_area
    )  %>%
  # long format
  pivot_longer(
    -c(gridcell_id, settled_area),
    names_to = 'iterations', values_to = 'mu_predicted') %>%
  mutate(
    # add sigma iterations for every grid cell
    sigma=rep(sigmas, nrow(mu_predicted)),
    # draw density for a log normal
    density_predicted = rlnorm(n(), mu_predicted, sigma),
    # draw population count from a Poisson
    population_predicted = rpois(n(), density_predicted*settled_area)
    ) %>%
  dplyr::select(-mu_predicted,-sigma, -density_predicted) %>%
  # convert it back to hrid cell x iterations matrix
  pivot_wider(-c(iteration, population_predicted),
      names_from = iteration,
      values_from = population_predicted
    )
```

`predictions` contains 2000 population estimates for every settled grid cell.

Hide

```
predictions[1:5, ] %>%  dplyr::select('gridcell_id', 1:10) %>% kbl() %>% kable_minimal()
```

gridcell_id	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
47	375	114	124	45	143	189	51	298	137	105
48	5	11	3	9	4	7	7	16	28	1
90	12	29	20	8	15	13	9	23	17	9
91	4	5	1	6	15	2	4	3	6	8
208	7	4	8	1	4	8	2	12	2	6

Note: Model prediction is memory-intensive. We thus usually run it in parallel for blocks of grid cells.

Predicting distributions

Bayesian modelling is a stochastic modelling thus accounts for the unknowns, caused by for example:

- Weak predictors
- Errors in input data
- Incomplete sample representativity
- Observations variation

These unknowns result in prediction uncertainty. More precisely it results in predicting a distribution of the likely population count.

As seen in previous section, we have for every grid cell 2000 predictions. Figure 4 shows an example for five grids cells.

Code

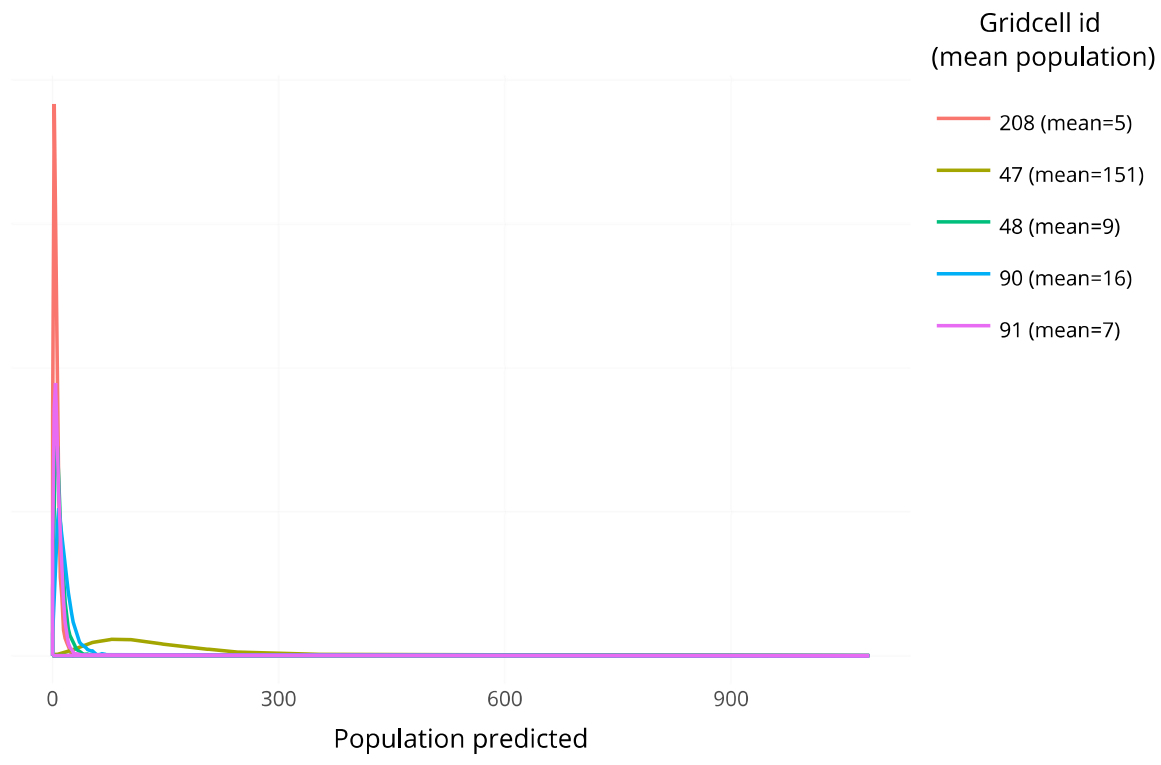


Figure 4: Example of predicted population count for 5 grid cells and their mean population count

A meaningful statistic to retrieve from the distribution is the mean, which can be considered as the *most likely value.*

Gridded population

The gridded bottom-up population that WorldPop released (https://wopr.worldpop.org/?/Population) are based on the mean value per grid cell.

Let's see how we can create a gridded population from our predictions.

We first compute the mean prediction for every grid cell:

Hide

```
mean_prediction <- tibble(
  mean_prediction = apply(predictions %>% dplyr::select(-gridcell_id), 1, mean)
  )
```

We then add the unsettled grid cell by joining with the `raster_df`, that contains every grid cell.

Hide

```
mean_prediction <- mean_prediction %>%
  # add gridcell_id
  mutate(gridcell_id = predictions$gridcell_id) %>%
  # join unsettled grid cell
  right_join(raster_df %>%
               dplyr::select(gridcell_id)) %>%
  # sort according to position in the raster
  arrange(gridcell_id)
```

To retrieve the raster structure we load a reference master - typically the mastergrid.

Hide

```
wd_path <- 'Z:/Projects/WP517763_GRID3/Working/NGA/ed/bottom-up-tutorial/'
mastergrid <- raster(paste0(wd_path, 'mastergrid.tif'))
```

And assign the predicted population count.

Hide

```
# create raster
mean_r <- mastergrid
# assign mean prediction
mean_r[] <- mean_prediction$mean_prediction
```

We can finally plot the raster to see our gridded population:

Hide

```
library(RColorBrewer)
# create color ramp
cuts <- quantile(mean_prediction$mean_prediction,
                 p=seq(from=0, to=1, length.out=7), na.rm=T)
col <-  brewer.pal(n = 7, name = "YlOrRd")

#plot mean prediction
plot(mean_r, col=col)
```

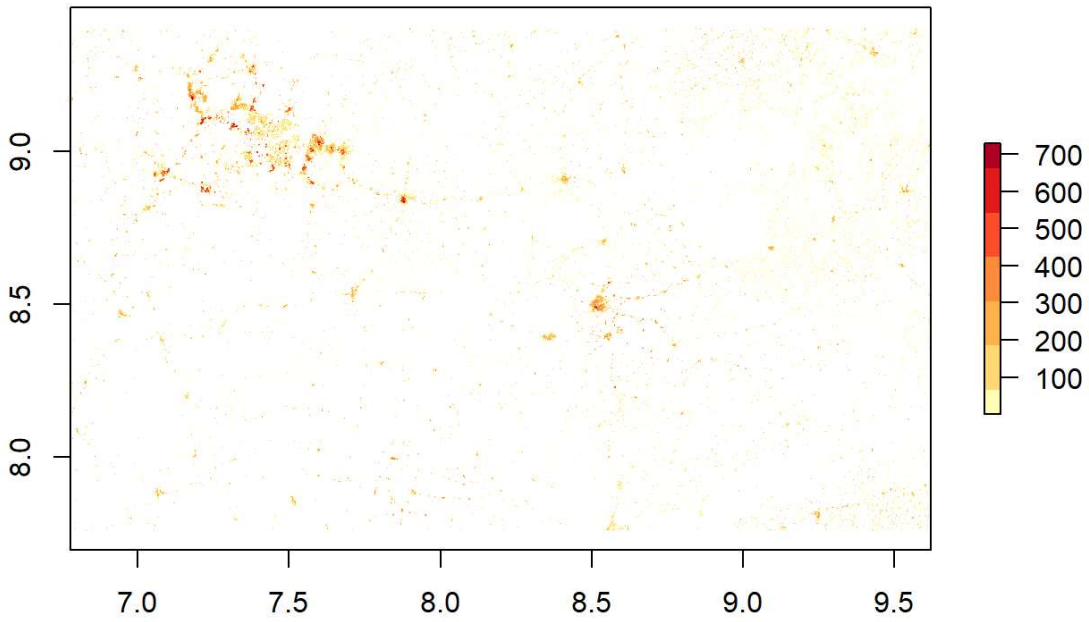


Figure 5: Gridded mean population prediction for region 8

Figure 5 shows typical population distribution: denser around cities and following the roads as seens by the lines of higher population count. In white are displayed the unsettled grid cells.

Gridded uncertainty

We can also retrieve from the full posterior distribution the 95% credible interval (CI) for every prediction and the related relative uncertainty computed as:

$$uncertainty = \frac{upper\_CI - lower\_CI}{mean\_prediction}$$

The relative uncertainty informs us about areas where the mean prediction is less reliable.

To compute the gridded uncertainty, we follow the same process as for the gridded mean prediction.

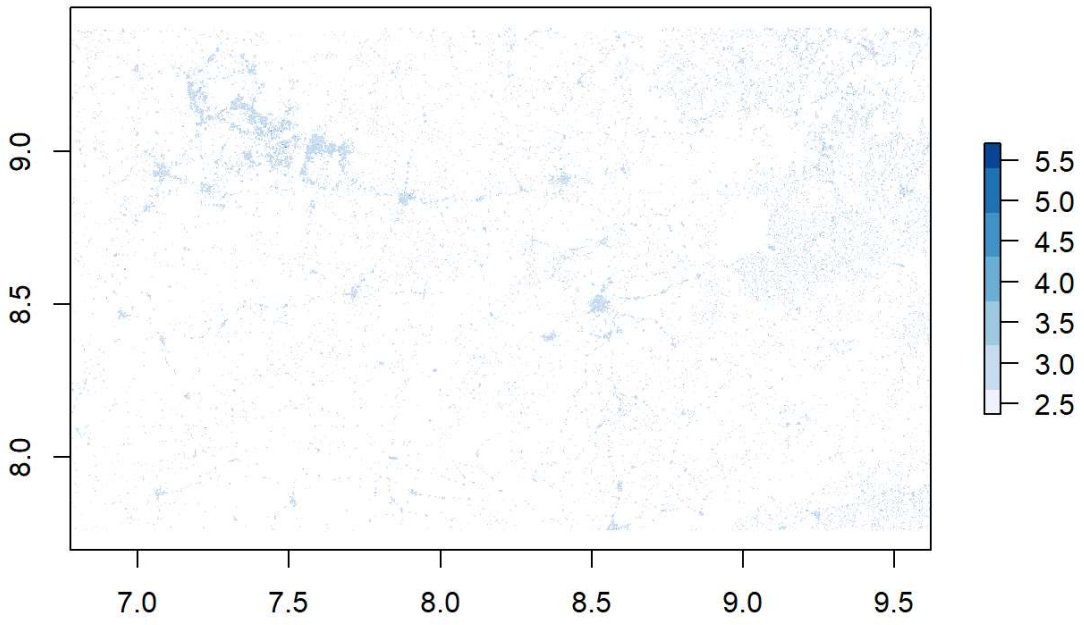
Hide

```
# retrieve the 95% credible interval
ci_prediction <- apply(predictions %>% dplyr::select(-gridcell_id),1, quantile, p=c(0.025, 0.97
5))

ci_prediction <- as_tibble(t(ci_prediction)) %>%
    # add gridcell_id
  mutate(gridcell_id = predictions$gridcell_id) %>%
  # join unsettled grid cell
  right_join(raster_df %>%
               dplyr::select(gridcell_id)) %>%
  # sort according to position in the raster
  arrange(gridcell_id) %>%
  mutate(
    mean_prediction = mean_prediction$mean_prediction,
    uncertainty = (`97.5%` - `2.5%`)/ mean_prediction
  )

# create uncertainty raster
uncertainty_r <- mastergrid
uncertainty_r[] <- ci_prediction$uncertainty

#plot uncertainty raster
cuts <- quantile(ci_prediction$uncertainty,
                 p=seq(from=0, to=1, length.out=7), na.rm=T)
col <-  brewer.pal(n = 7, name = "Blues")
plot(uncertainty_r, col=col)
```



We see that higher uncertainty can be observed on the outskirt of urban areas. This is often the case due to the rapid expansion of settlement extent that is difficult to capture.

Aggregating prediction

Gridded populations are great to visualise the spatial distribution of population across an area. But often we want to get aggregates, that is **a population estimate for a custom area** such as a city or the catchment area of a hospital.

Let's look at an example. We manually drew the extent of one city in our study area.

Disclaimer: the extents are not the official extent

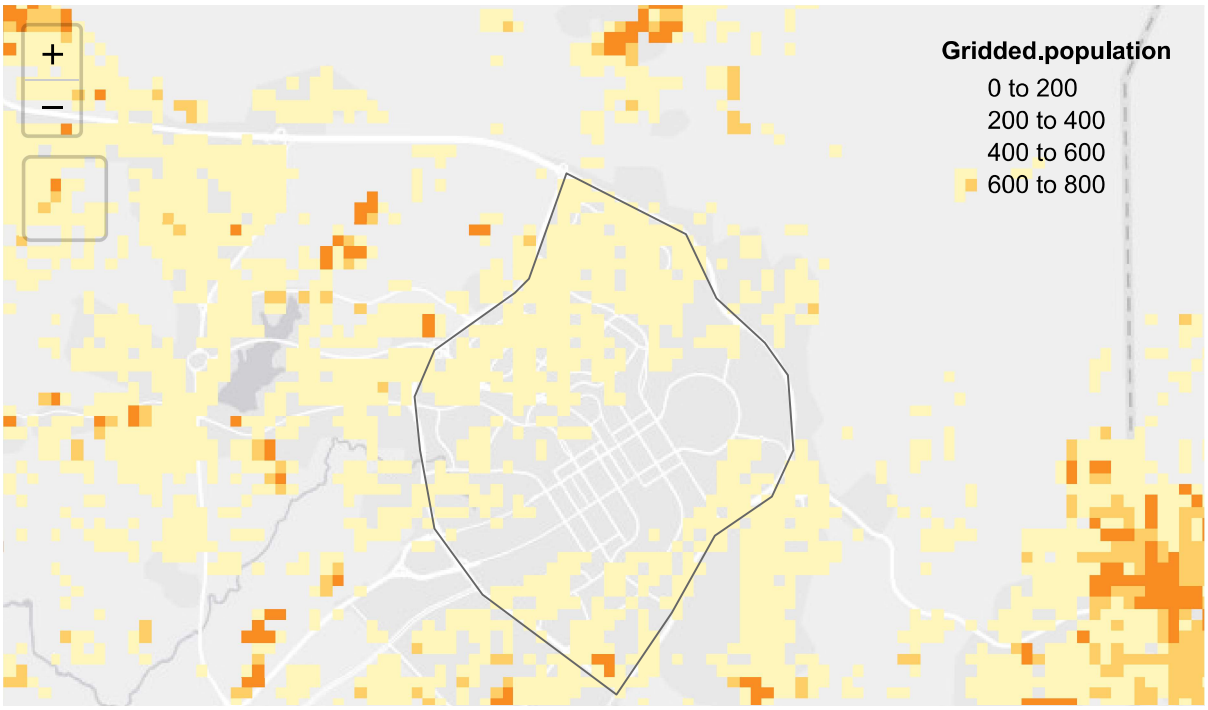
Hide

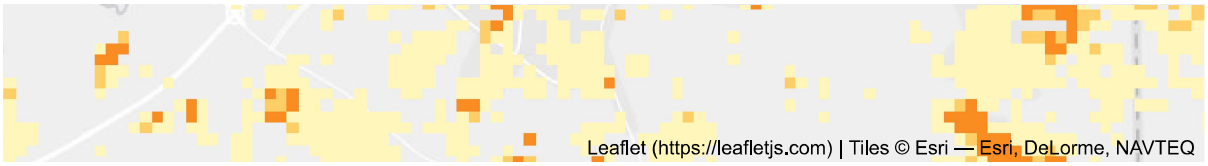
```
library(sf)
library(tmap)
tmap_options(check.and.fix = TRUE)
city <- st_read(paste0(wd_path, 'study_city.gpkg'))
```

```
## Reading layer `study_city' from data source
##   `Z:\Projects\WP517763_GRID3\Working\NGA\ed\bottom-up-tutorial\study_city.gpkg'
##   using driver `GPKG'
## Simple feature collection with 1 feature and 0 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 7.451997 ymin: 9.012191 xmax: 7.524984 ymax: 9.111175
## Geodetic CRS:  WGS 84
```

Hide

```
names(mean_r) <- 'Gridded population'
tmap_mode('view')
tm_shape(city)+
  tm_borders()+
  tm_shape(mean_r)+
  tm_raster()+
  tm_basemap('Esri.WorldGrayCanvas')
```





Getting the **mean population prediction for the city** corresponds to computing zonal statistic on the gridded population, that is summing up all the grid cells contained in the city.

Hide

```
mean_city <- extract(mean_r, city, fun=sum, na.rm=TRUE, df=TRUE)
mean_city %>%
  mutate(features = 'city', .after=ID) %>%
  rename("Mean population prediction"=Gridded.population) %>%
  kbl(caption='Mean population prediction for the city computed with the gridded population') %
>% kable_minimal(full_width = F)
```

Table 2: Mean population prediction for the city computed with the gridded population

ID	features	Mean population prediction
1	city	168303.3

Getting the uncertainty for that estimate is more complicated. Indeed we can't sum up uncertainty of the grid cell to retrieve the uncertainty of the aggregate because credible intervals are based on quantile computation, and quantiles can't be summed.

We need thus to reconstruct the full population prediction distribution for the city by aggregating all grid-cells population prediction distribution.

We first convert the city polygon to a raster with same extent as the gridded population. This raster is made of 1s for grid cells inside the city extent and 0s for grid cells outside the city.

Hide

```
city_r <- rasterize(city, mean_r)
```

We convert the city raster to an array and join to `raster_df` to get the grid cell id. We filter out the grid cells that belongs to the city and then merge it with the predictions:

Hide

```
city_prediction <- raster_df %>%
  # select grid cell id from raster df
  dplyr::select(gridcell_id) %>%
  # add city dummy
  mutate(city = city_r[]) %>%
  # keep grid cells inside the city
  filter(city==1) %>%
  # join predictions
  left_join(predictions) %>%
  # keep predictions
  dplyr::select(starts_with('V'))
```

`city_prediction` contains all the grid cells comprised in the city with the corresponding full population prediction distribution.

We first sum the predictions for every grid cells at each iteration.

Hide

```
city_prediction <- as_tibble(apply(city_prediction,2, sum, na.rm=T))
```

`city_prediction` becomes an array of size 2000 that contains thus the 2000 population totals for the city.

We can then derive meaningful statistics from the distribution of population total for the city.

Code

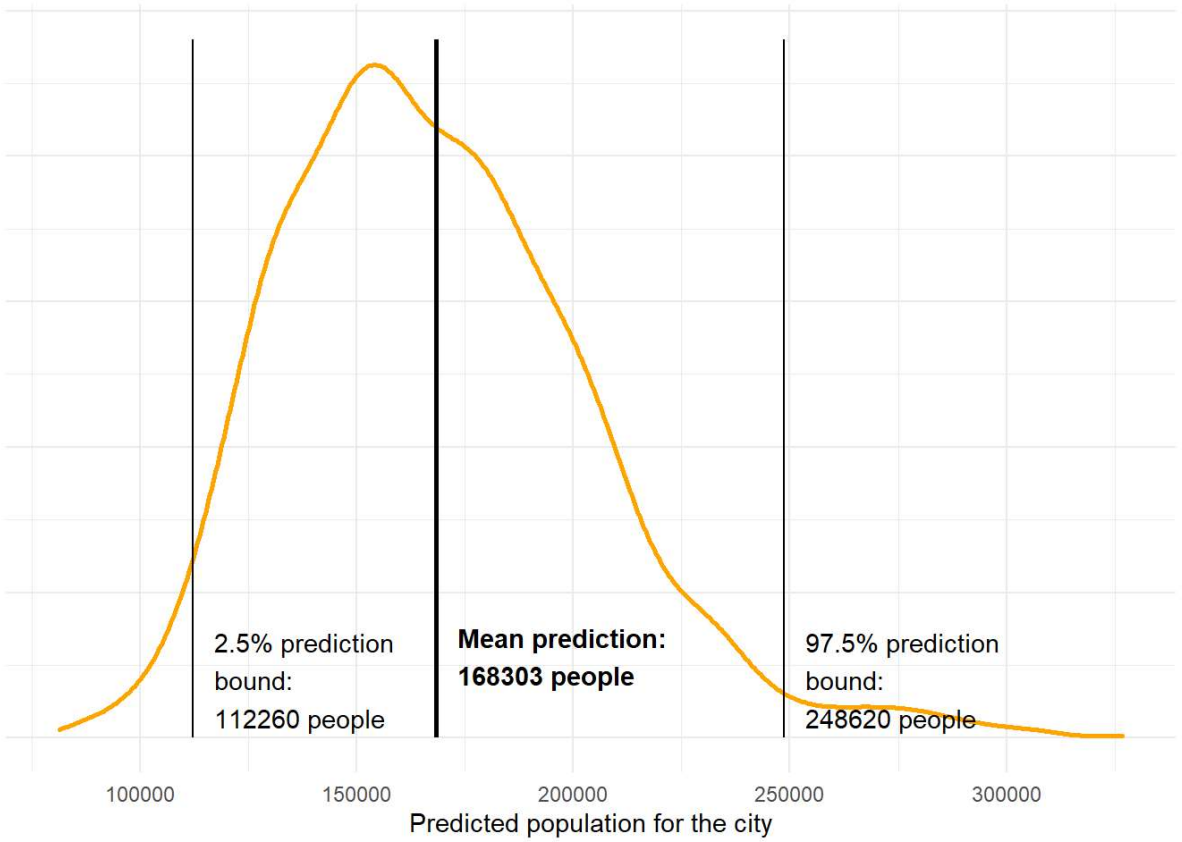


Figure 6: Full distribution of the predicted population total for the city

Figure 6 shows that the mean prediction matches the one computed with the gridded population raster. Furthermore we see that our model produces predictions with a very wide credible interval.

Having the full distribution can answer questions such as *"what is the probability that there is more than xx people in the area"* or *"with a 95% certainty, what is the maximum number of people in that area?"*

References

Leasure, Douglas R, Warren C Jochem, Eric M Weber, Vincent Seaman, and Andrew J Tatem. 2020. "National Population Mapping from Sparse Survey Data: A Hierarchical Bayesian Modeling Framework to Account for Uncertainty." *Proceedings of the National Academy of Sciences*.

Oak Ridge National Laboratory. 2018. "LandScan HD: Nigeria."