Global Farm Size, Version 1, 2000. Preliminary Release.

Zia Mehrabi and Vincent Ricciardi

April 17, 2024

Contents

1 Aim of document

This document was first created for the peer reviewed publication: Mehrabi, Z., McDowell, M.J., Ricciardi, V. et al. 2020. The global divide in data-driven farming. Nature Sustainability, 4:154–160. https://doi.org/10.1038/s41893-020-00631-0

The aim of this document is to create a global map of farm sizes. We do this by merging two currently available sources of information on the distribution of farm sizes across the world: national level data on farm size distributions and subnational data on field size distributions. The resulting map is a "best guess" of where different sizes of farms are distributed globally. We caution that this map should not be used for detailed country level analysis, but is intended for global assessment studies, where results are typically aggregated at the regional or global level. This document explains the underlying data, pre-processing, and algorithm we developed to create this map. We were driven to create this map as a work that can be built upon and improved as higher resolution and more accurate data become available.

2 Reproducibility

Here we call the **R** package `renv` (**R-renv**). This will create a local library on your computer and install a copy of the packages required by this project as they existed on CRAN by the specified version number, and update the **R** session to use these packages. This helps make our analysis fully reproducible on your machine.

Note, the 'renv.lock' file needs to be in the top level of this project's directory. This code block needs to only be run when initially setting up your project then can be commented out.

```
> # Uncomment during first run
> # install.packages('renv')
> # renv::init()
```

We also set the seed of the entire document to ensure the same results when randomly sampling.

```
> set.seed(123)
```

Note that the **R** version used here is 4.3.1 (2023-06-16).

```
[[1]]
[1] TRUE

[[2]]
[1] TRUE

[[3]]
[1] TRUE

[[4]]
[1] TRUE

[[5]]
[1] TRUE

[[6]]
[1] TRUE

[[7]]
[1] TRUE

[[8]]
```

```
[1] TRUE

[[9]]
[1] TRUE

[[10]]
[1] TRUE

[[11]]
[1] TRUE

[[12]]
[1] TRUE

[[13]]
[1] TRUE

[[14]]
[1] TRUE

[[15]]
[1] TRUE

[[16]]
[1] TRUE

[[17]]
[1] TRUE

[[18]]
[1] TRUE

[[19]]
[1] TRUE

[[20]]
[1] TRUE

[[21]]
[1] TRUE
```

For the analysis in this document we will be using the following packages: SexprR-data.table (citeR-data.table).

3 Data

In this analysis we leverage two key datasets. **Lowder** (Lowder, hereafter) contains farm size distributions (in terms of agricultural area and the number of farms) for 100 countries. **Lesiv** (Lesiv, hereafter) contains a crowd-sourced point data of categorical field size classes (ranging from very small to large fields). We use Lesiv's qualitative field size classes to spatially disaggregate the Lowder farm size classes. Links and access dates to these input data and other ancillary data sets used in our analysis as highlighted below:

1. Global field size data from Lesiv, retrieved from from http://pure.iiasa.ac.at/id/eprint/15526/ on October 12th, 2018.

2. Country farm size distributions from Lowder, retrieved from http://iopscience.iop.org/article/10.1088/1748-9326/11/12/124010 on July 12th, 2018.

3. Global cropland and pastureland from **Ramankutty** (Ramankutty, hereafter), retrieved from www.earthstat.org on July 12th 2018.

4. Country boundaries, retrieved internally in R through **R-rworldmap**.

3.1 Country boundaries

First we make a raster of the world country data from the `rworldmap` package.

```
> world  <- rworldmap::getMap(resolution = 'low')
> lookup <- as.data.frame(cbind(as.character(
+   world@data[,'ISO3']),
+   world@data[,'ADMIN'],
+   as.character(world@data[,'ADMIN'])))
> raster.world <- raster(res = c(0.0833282, 0.0833282))
> extent(raster.world) <- extent(world)
> world.raster <- rasterize(as(world, 'SpatialPolygons'),
+                           raster.world,
+                           field = world@data[, 'ADMIN'],
+                           fun   = 'first')
> world.rast <- writeRaster(world.raster,
+                           'data/tmp/worldrast.tif',
+                           format    = 'GTiff',
+                           overwrite = TRUE)
> world  <- raster('data/tmp/worldrast.tif')
```

3.2 Country farm size distributions

Lowder's data contains two variables at the national level, the amount of agricultural area per farm size class, and the number of farms per farm size class. We are interested in the amount of agricultural area per farm size class for our analysis. This data is distributed as the World Census of Agriculture's (WCA) farm size classes, which are: 0-1 ha, 1-2 ha, 2-5 ha, 5-10 ha, 10-20 ha, 20-50 ha, 50-100 ha, 100-200 ha, 200-500 ha, 500-1000 ha, 1000 ha. This data is read in here and columns relabeled.

```
> # Load Lowder's distribution dataset
> lwd <- read.csv('data/lowder/Lowder_2016_dist.csv',
+                 header = T,
+                 na.strings = c('','NA'))
> # Subset only agricultural area
> lwd <- lwd[which(lwd$Holdings..agricultural.area != 'Holdings'), ]
> lwd <- lwd[, c(1,5:15)]
> names(lwd) <- c('country', '0_1', '1_2', '2_5',
+                 '5_10', '10_20', '20_50', '50_100', '100_200',
+                 '200_500', '500_1000', '1000_5000')
> # Ensure variable type is numeric
> for (i in names(lwd)[2:length(names(lwd))]) {
+   lwd[[i]] <- as.numeric(lwd[[i]])
+ }
```

Next we need to calculate the cumulative sum per country across Lowder's farm size classes. To do this we first convert the data from wide to long format, and set classes for countries without data to zero area.

```
> lwd <- melt(lwd, id.vars = c('country'))
> lwd[is.na(lwd)] <- 0
```

Then we calculate the proportional agricultural area for each farm size class for each country. We note that Lowder contains some countries without farm size distributions and some countries do not have a distribution for each farm size class, and so we remove those cases here.

```
> lwd <- lwd %>%
+   group_by(country) %>%
+   mutate(total = sum(value),
+          perc = value / total) %>%
+    filter(value > 0) %>%
+   select(country, variable, perc)
```

Finally, cast dataframe back into wide form.

```
> lwd <- dcast(lwd, country ~ variable)
> write.csv(lwd, 'data/tmp/lwd.csv')
```

3.3 Global agricultural area

We use Ramankutty's 10 km^2 cropland map (crop, hereafter) and pastureland map (pasture, hereafter) as our reference layer of the distribution of agricultural land at the subnational level. We read in these files here and unionize the crop and pasture maps to create an agricultural land raster (ag, hereafter).

```
> crop <- paste0('data/Ramankutty_2008_cropland/',
+                'CroplandPastureArea2000_Geotiff/',
+                'Cropland2000_5m.tif')
> pasture <- paste0('data/Ramankutty_2008_cropland/',
+                   'CroplandPastureArea2000_Geotiff/',
+                   'Pasture2000_5m.tif')
> crop <- raster(crop)
> pasture <- raster(pasture)
> # Here we take a union of crop and pasture areas
> # Set NA to 0, ensure same origin, find sum
> crop_tmp <- crop
> pasture_tmp <- pasture
> crop_tmp[is.na(crop_tmp)] <- 0
> pasture_tmp[is.na(pasture_tmp)] <- 0
> origin(crop_tmp) <- origin(pasture_tmp) <- c(0.0, 0.0)
> ag <-mosaic(crop_tmp, pasture_tmp, fun = sum)
> ag[ag[] == 0] <- NA
```

3.4 Global field sizes

Here we interpolate the crowd-sourced field size data point data onto Ramankutty's agricultural area data. While the original sampling frame for this field size was on a unionized cropland map, we extend the distribution here in an attempt to better match and account for the inclusion of non-cropland in Lowder's farm size data (noting that later we will clip the resulting product to cropland only). We use kk-nearest neighbor (kkNN) on a 0.01 degree grid (kkNN is a weighted variant of kNN that we use to try to account as best as possible for Lesiv's sparsely sampled points in Africa).

3.4.1 Read in Data

First, we read in the estimated dominant field sizes point data and subset to not include NA values or values coded as having no fields.

```
> fs <- read.csv("data/Fritz_2019/Global Field Sizes/estimated_dominant_field_sizes.csv")
```

3.4.2 Pre-process Data

We recode Lesiv's data according to their coding as detailed below. And then convert this data into a spatial object.

- 3502 - Very large fields with an area of greater than 100 ha
- 3503 - Large fields with an area between 16 ha and 100 ha
- 3504 - Medium fields with an area between 2.56 ha and 16 ha
- 3505 - Small fields with an area between 0.64 ha and 2.56 ha
- 3506 - Very small fields with an area less than 0.64 ha
- 3507 - no fields
- NA - skipped

```
> fs[which(fs$field_size == 3507), 5] <- NA
> fs_clean <- fs %>%
+   na.omit() %>%
+   dplyr::mutate(field_size_fac = as.factor(field_size)) %>%
+   dplyr::mutate(fs_verbatim_fac = recode(field_size_fac,
+                                           "3502" = "very large",
+                                           "3503" = "large",
+                                           "3504" = "medium",
+                                           "3505" = "small",
+                                           "3506" = "very small"))%>%
+   dplyr::select(-rowid,-sampelid) %>%
+   st_as_sf(coords = c("x", "y"),
+            crs = "+proj=longlat +ellps=WGS84")
```

3.4.3 Interpolate

We make a target raster grid to interpolate onto using the agricultural area raster. We use a very simple method of interpolation based on the latitude and longitude.

```
> ag_GR0 <- ag
> ag_GR0[] <- ifelse(ag_GR0[] > 0, 1, NA)
> target_pts <- rasterToPoints(ag, spatial = TRUE)
> target_pts_agGR0 <- rasterToPoints(ag_GR0, spatial = TRUE)
```

We then train the kkNN model using different types of kernels and number of points used in the classification procedure. We use 10 fold cross validation to identify the best parameters to use in the final model (noting that CV does not approximate the test error as would be estimated using an independent hold-out set).

```
> fs_train <- data.frame(field_size = fs_clean$fs_verbatim_fac,
+                        x = st_coordinates(fs_clean)[,1],
+                        y = st_coordinates(fs_clean)[,2])
> fs_interp_train <- kknn::train.kknn(field_size ~.,
+                                     data = fs_train,
+                                     ks = seq(5,51, 5),
+                                     distance = 2,
+                                     kernel = c("gaussian",
+                                                "triangular",
+                                                "rectangular",
+                                                "epanechnikov",
+                                                "optimal"),
+                                     kcv = 10)
```

Next we summarize and plot the results of the model fitting. We note the the minimal classifications error is 0.429748947326607. Crudely, with 5 classes we would expect a raw misclassification error of 80%, which suggests the model has in the best case helped reduce blind uncertainty by around 53.7186184158258%. We also plot the error from the different models below.

Future versions improvements to this model may be made through identifying ancillary features that do a better job of predicting field size classes than simply latitude and longitude alone. Similarly, using an ordinal response, instead of the nominal response defaulted here, is likely to improve predictions also.

We use the best fitting model parameters to predict the field size classes across the agricultural area grid, and then extract the fitted values and the associated probabilities.

```
> fs_interp_final <- kknn::kknn(field_size ~ .,
+                               train = fs_train,
+                               test = target_pts_agGRO,
+                               kernel = fs_interp_train$best.parameters$kernel,
+                               k = fs_interp_train$best.parameters$k)
> target_pts_agGRO_res <- target_pts_agGRO %>%
+   # extract the interpolated class at each
+   # grid cell with the kknn::fitted function
+   as.data.frame() %>%
+   # only retain the probability of the
+   # interpolated size class, discard the others
+   mutate(field_size_fit = fitted(fs_interp_final),
+          prob = apply(fs_interp_final$prob, 1, function(input) max(input)))
```

3.4.4 Rasterize

Here we convert results back to a raster and save the results of the fitted classes.

```
> target_pts_agGRO_res_spdf <- target_pts_agGRO_res
> coordinates(target_pts_agGRO_res_spdf) <- ~ x + y
> fs_interp_final_ras <- rasterize(target_pts_agGRO_res_spdf,
+                                  ag,
+                                  as.integer(
+                                  target_pts_agGRO_res_spdf$field_size_fit),
+                                  progress = "text")
> fs_interp_final_ras_prob <- rasterize(target_pts_agGRO_res_spdf,
+                                       ag,
+                                       target_pts_agGRO_res_spdf$prob,
+                                       progress = "text")
> lesiv <- fs_interp_final_ras
```

4 Create common dataframe

To run our algorithm to create a field size map we need to ensure all rasterized datasets (i.e., field size, crop, pasture, ag, and country boundaries) are all in the same resolution, have the same coordinate reference system (crs), and have the same spatial extent, which we do in this section.

4.1 Convert to equal area

Here we set each dataset to have an equal area (Eckert IV) at a 8.4 km^2 resolution. To interpolate, we use nearest neighbor for the categorical data and bilinear for the continuous data.

```
> rast.list.ngb <- list(lesiv, world)
> rast.list.bil <- list(crop, pasture, ag)
> resValue <- 8439
> rast.list.ngb.eq <- lapply(rast.list.ngb, function(x)
+   projectRaster(
+     x,
+     res    = c(resValue, resValue),
+     crs    = '+proj=eck4 +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0',
+     method = 'ngb',
+     over   = T))
> rast.list.bil.eq <- lapply(rast.list.bil, function(x)
+   projectRaster(
+     x,
+     res    = c(resValue, resValue),
+     crs    = '+proj=eck4 +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0',
+     method = 'bilinear',
+     over   = TRUE))
> rast.list <- c(rast.list.ngb.eq, rast.list.bil.eq)  # Combine data
> names(rast.list) <- c('lesiv', 'world', 'crop', 'pasture', 'ag')
```

4.2 Set extents

Next, we need to make sure that all the rasters have the same spatial extents. The country boundaries include Antarctica, while the cropland/pastureland/ag and field size data do not (e.g., crop ymin is -8501789, while the country boundaries ymin is -8503388). To ensure all the datasets match we need to expand the cropland/pastureland/ag and Lesiv extents to match the country boundaries extent. This means extending the cropland/pastureland/ag and Lesiv data to also include the country boundary grid cells; we do this by adding rows of NA grid cells.

```
> ex1 <- extent(rast.list$world)
> # Extend
> rast.list.ex <- lapply(rast.list,
+                         function(x) extend(x, ex1))
> # Crop
> rast.list.c  <- lapply(rast.list.ex,
+                         function(x) crop(x, ex1))
> # Force equal extent
> rast.list.et <- lapply(rast.list.c,
+                         function(x) setExtent(x, ex1,
+                                               keepres = TRUE))
```

4.3 Create stack

Now that all the layers' resolution, crs, and spatial extent are equal, we can create a raster stack and proceed with the analysis.

```
> rast.all <- stack(rast.list.et)
```

4.4 Finalize data set

We now convert the data in the raster stack into a dataframe, which we will use for later analysis.

```
> df <- raster::extract(rast.all, 1:ncell(rast.all), df = T)
> df <- as.data.frame(df)
> df$ID <- 1:nrow(df)
```

Set any crop, pasture value under 0.01 to NA, to account for sparse agricultural areas.

```
> df$crop <- ifelse(df$crop >= 0.01, df$crop, NA)
> df$pasture <- ifelse(df$pasture >= 0.01, df$pasture, NA)
> df$ag <- ifelse(df$ag >= 0.01, df$ag, NA)
```

Next we add back in the country names (presently they are only country ISO3 IDs), then subset the data to only contain countries in the Lowder dataset (this speeds up processing).

```
> world  <- getMap(resolution = 'low')
> lookup <- as.data.frame(
+   cbind(
+     as.character(world@data[,'ISO3']),
+     world@data[,'ADMIN'],
+     as.character(world@data[,'ADMIN'])))
> matched    <- lookup[match(df$world, lookup$V2), ]
> df$ISO3    <- matched[[1]]
> df$country <- matched[[3]]
```

We then match Lowder's country names to the global country boundary country names.

```
> countries <- data.frame(names = sort(unique(df$country)))
> lwdNames  <- data.frame(names = unique(lwd$country))
> lwdNames$names  <- str_trim(lwdNames$names)
> lwdNames$indata <- lwdNames$names %in% unique(countries$name)
> # List of names in lowder that do not match dataset's names
> # note Guadeloupe, Martinique, Reunion has no data
> # lwdNames[which(lwdNames$indata == FALSE), ]
>
> changeFrom <- c('Korea Rep. of',
+                 "Lao People's Democratic Republic",
+                 'Viet Nam',
+                 'Serbia',
+                 'Bahamas',
+                 'Venezuela (Bolivarian Republic of)',
+                 'St. Kitts & Nevis',
+                 'Iran (Islamic Republic of)',
+                 "Côte d'Ivoire",
+                 'Virgin Islands United States')
> changeTo   <- c('South Korea',
+                 'Laos',
+                 'Vietnam',
+                 'Republic of Serbia',
+                 'The Bahamas',
+                 'Venezuela',
+                 'Saint Kitts and Nevis',
+                 'Iran',
+                 'Ivory Coast',
+                 'United States Virgin Islands')
> require(plyr)
> lwd$country <- mapvalues(str_trim(lwd$country),
+                          from = changeFrom,
+                          to   = changeTo)
> detach('package:plyr')
```

Subset the dataset to only the countries containing farm size distributions.

```
> lwdNames <- unique(lwd$country)
> df_lwd <- df[which(df$country %in% lwdNames), ]
> df_not_lwd <- df[which(!df$country %in% lwdNames), ]
```

5 Match Field Sizes to Farm Sizes

5.1 Algorithm

Here are the steps we used to create the farm size map, with uncertainty at the level of the distribution of farm to field size matching. The algorithm makes the assumption that smaller field sizes are owned by smaller farms (although we note that the uncertainty in this assumption grows with farm size, as while smaller farms can not own large fields, large farms can own many small fields).

1. Group pixels by country
2. Assign pseudo-random number to pixels
3. Reorder the pixels in a nested fashion: first by interpolated field size classes, and then by the pseudo-random numbers
4. Compute pixel wise agricultural area from Ramankutty
5. Compute cumulative agricultural area from Ramankutty
6. Divide cumulative sum of agricultural area by total area in each country to get cumulative proportional sums of agricultural area from Ramankutty
7. Assign pixel fields sizes to farm size classes by matching the cumulative proportional agricultural area from Lowder to the cumulative proportional agricultural area for interpolated Lesiv product where pixels are ordered from smallest to largest fields
8. For countries not in Lowder, hard code farm size based on field sizes.
9. Convert result to a raster
10. Save raster
11. Iterate steps 1 to 10 100x

Steps 1 to 6 are written below.

```
> cumsumSkipNA <- function(x) {
+     FUNC <- match.fun('cumsum')
+     x[!is.na(x)] <- FUNC(x[!is.na(x)])
+     return(x)
+ }
> df_lwd$lesiv <- factor(df_lwd$lesiv)
> levels(df_lwd$lesiv) <- c('5', '4', '3', '2', '1')
> df_lwd$lesiv <- as.integer(as.character(df_lwd$lesiv))
> steps_1_6 <- function(df_lwd) {
+
+   dat <- df_lwd %>%
+     mutate(pseudorandom = sample(1:n(), n(), replace = F))  %>%
+     arrange(country, lesiv, pseudorandom) %>%
+     group_by(country) %>%
+     mutate(crop = crop * resValue^2 * 0.0001,       # new
+            pasture = pasture * resValue^2 * 0.0001, # new
+            ag = crop + pasture,                     # new
+            ag_area = ag * resValue^2 * 0.0001, # original was this line only
+            cumsum_area = cumsumSkipNA(ag_area),
+            prop_sums = ifelse(
+                is.na(cumsum_area),
+                NA,
+                cumsum_area / max(cumsum_area, na.rm = T)
+                )
+     )
+
+   return(dat)
+ }
```

Step 7 is written below.

```
> cumsumSkipNA <- function(x) {
+     FUNC <- match.fun('cumsum')
+     x[!is.na(x)] <- FUNC(x[!is.na(x)])
+     return(x)
+ }
> step_7 <- function(dat) {
+
+   tmp <- lwd[, 2:length(lwd)]
+   tmp <- t(tmp)
+   tmp <- apply(tmp, FUN = function(x) cumsumSkipNA(x), MARGIN = 2)
+   tmp <- t(tmp)
+   tmp <- as.data.frame(tmp)
+   tmp$country <- lwd$country
+
+   dat <- merge(dat, tmp, by = 'country')
+
+   for (x in unique(dat$country)) {
+
+       tmp <- dat[which(dat$country == x), ]
+
+       tmp$farm <- ifelse(tmp$prop_sums <= tmp$`0_1`
+                          & !is.na(tmp$`0_1`),
+                          '0_1',
+                   ifelse(tmp$prop_sums <= tmp$`1_2`
+                          & !is.na(tmp$`1_2`),
+                          '1_2',
+                   ifelse(tmp$prop_sums <= tmp$`2_5`
+                          & !is.na(tmp$`2_5`),
+                          '2_5',
+                   ifelse(tmp$prop_sums <= tmp$`5_10`
+                          & !is.na(tmp$`5_10`),
+                          '5_10',
+                   ifelse(tmp$prop_sums <= tmp$`10_20`
+                          & !is.na(tmp$`10_20`),
+                          '10_20',
+                   ifelse(tmp$prop_sums <= tmp$`20_50`
+                          & !is.na(tmp$`20_50`),
+                          '20_50',
+                   ifelse(tmp$prop_sums <= tmp$`50_100`
+                          & !is.na(tmp$`50_100`),
+                          '50_100',
+                   ifelse(tmp$prop_sums <= tmp$`100_200`
+                          & !is.na(tmp$`100_200`),
+                          '100_200',
+                   ifelse(tmp$prop_sums <= tmp$`200_500`
+                          & !is.na(tmp$`200_500`),
+                          '200_500',
+                   ifelse(tmp$prop_sums <= tmp$`500_1000`
+                              & !is.na(tmp$`500_1000`),
+                              '500_1000',
+                      ifelse(tmp$prop_sums <= tmp$`1000_5000`
+                              & !is.na(tmp$`1000_5000`),
+                              '1000_5000',
+                              '1000_5000'
+                              )))))))))))
+
+       if (x == as.character(unique(dat$country)[1])) {
+         out <- tmp
+       } else {
+         out <- rbind(out, tmp)
+       }
+   }
+
+   out <- out[,c(2,length(out))]
+
+   dat <- merge(df, out,
+                by = 'ID',
+                all.x = T,
+                all.y = T)
+
+   return(dat)
+
+ }
```

Step 8 is written below. For countries not in Lowder, we hard code the farm size matched to that field size class. We take the upper end of the given range and match it to Lesiv's codes (e.g., code 3502 becomes 200 ha)

1 - 3502 - Very large fields with an area of greater than 100 ha - 100-200 ha

2 - 3503 - Large fields with an area between 16 ha and 100 ha - 50-100 ha

3 - 3504 - Medium fields with an area between 2.56 ha and 16 ha - 5-10 ha

4 - 3505 - Small fields with an area between 0.64 ha and 2.56 ha - 1-2 ha

5 - 3506 - Very small fields with an area less than 0.64 ha - 0-1 ha

NA - 3507 - no fields;

```
> step_8 <- function(dat) {
+
+   field_2_farm <- data.frame(
+     lesiv = c(1,2,3,4,5),
+     # lesiv = c(5,4,3,2,1),
+     farm_avg = c('100_200',
+                  '50_100',
+                  '5_10',
+                  '1_2',
+                  '0_1'))
+
+   dat <- merge(dat, field_2_farm, by = 'lesiv', all.x = T)
+
+   # Make sure that any inf are NA
+   dat2 <- dat %>% rationalize()
+
+   # Make sure farm and farm_avg cats are same type
+   dat$farm <- as.character(dat$farm)
+   dat$farm_avg <- as.character(dat$farm_avg)
+
+   dat$farm_final <- ifelse(dat$ag >= 0.01 & !is.na(dat$farm),
+                            dat$farm,
+                     ifelse(dat$ag >= 0.01 & !is.na(dat$lesiv),
+                            dat$farm_avg, NA))
+   return(dat)
+ }
```

Step 9: Generate farm size raster

```
> step_9 <- function(dat, x = 'farm_final') {
+   # Places farm size values into a raster
+   # Params:
+   #   df
+   # Return:
+   #   raster object
+
+   if (x == 'farm_final') {
+     dat$rasterValues <- as.numeric(
+       str_split_fixed(dat[[x]], '_', 2)[, 2])
+   } else {
+     dat$rasterValues <- dat[[x]]
+   }
+
+   # Clip to cropland area
+   dat$rasterValues <- ifelse(is.na(dat$crop), NA, dat$rasterValues)
+
+   dat <- dat[order(dat$ID), ]
+   rast_final <- rast.all@layers[[1]]
+   rast_final <- setValues(rast_final, dat$rasterValues, dat$ID)
+
+   return(rast_final)
+ }
```

Step 10: Write raster

```
> step_10 <- function(rast_final, i) {
+   # Writes raster
+   # Params:
+   #   rast_final
+   #   i: random number
+   # Returns
+   #   Writes raster to file
+
+   writeRaster(rast_final,
+               filename  = paste0(
+                 'raster_out_forCleanRunTest/farmSize_',
+                 # 'raster_out_forPublication/farmSize_agarea_',
+               format(Sys.time(), "%Y%m%d"), '_', i, '.tif'),
+               format    = 'GTiff',
+               overwrite = TRUE)
+ }
```

5.2 Resampled Results

Step 11: Repeat steps 1-10 100x. Note, 100 runs is illustrative. For more stable results increase to > 1k.

```
> # i = 1
> # for (i in 1:100) {  # use for full run
> for (i in 1:2) {      # use for test run
+   dat1 <- steps_1_6(df_lwd)
+   dat2 <- step_7(dat1)
+   dat3 <- step_8(dat2)
+   rast <- step_9(dat3)
+   step_10(rast, i)
+ }
```

5.3 Sanity Check

We sanity check that the proportional farm size distributions per country equals Lowder's. We use one run as an example. As we can see the data is very close to the one-to-one line (but not exactly as would be expected based on small differences in row matching in our algorithm). Since the sampling scheme randomly shuffles the data, this plot will be slightly different for each sample.

5.4 Final Output

Here is the mode of the 100 farm size maps generated.

Session information

```
> sessionInfo()
```

```
R version 4.3.1 (2023-06-16)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Sonoma 14.3

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LAPACK versi

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/Los_Angeles
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] zoo_1.8-12        viridis_0.6.4     viridisLite_0.4.2 lubridate_1.9.2
 [5] forcats_1.0.0     stringr_1.5.0     dplyr_1.1.2       purrr_1.0.2
 [9] readr_2.1.4       tidyr_1.3.0       tibble_3.2.1      ggplot2_3.4.3
[13] tidyverse_2.0.0   sf_1.0-14         rworldmap_1.3-6   rgdal_1.6-7
[17] reshape2_1.4.4    raster_3.6-26     sp_2.1-2          kknn_1.3.1
[21] magrittr_2.0.3    lintr_3.1.0       leaflet_2.2.0     hablar_0.3.2
[25] gmodels_2.18.1.1  ggthemes_4.2.4    foreach_1.5.2     data.table_1.14.8
[29] renv_1.0.3        formatR_1.14      printr_0.3        knitr_1.43

loaded via a namespace (and not attached):
 [1] DBI_1.1.3          gridExtra_2.3      remotes_2.4.2.1    rlang_1.1.1
 [5] maptools_1.1-8     e1071_1.7-14       compiler_4.3.1     gdata_2.19.0
 [9] callr_3.7.3        vctrs_0.6.3        maps_3.4.1         pkgconfig_2.0.3
[13] crayon_1.5.2       fastmap_1.1.1      backports_1.4.1    labeling_0.4.2
[17] utf8_1.2.3         tzdb_0.4.0         ps_1.7.5           xfun_0.40
[21] terra_1.7-65       R6_2.5.1           stringi_1.7.12     Rcpp_1.0.11
[25] iterators_1.0.14   fields_15.2        Matrix_1.5-4.1     igraph_1.5.1
[29] timechange_0.2.0   tidyselect_1.2.0   rstudioapi_0.15.0  codetools_0.2-19
[33] processx_3.8.2     lattice_0.21-8     plyr_1.8.8         withr_2.5.0
[37] foreign_0.8-84     desc_1.4.2         units_0.8-5        proxy_0.4-27
[41] xml2_1.3.5         pillar_1.9.0       KernSmooth_2.23-21 rex_1.2.1
[45] generics_0.1.3     rprojroot_2.0.3    hms_1.1.3          munsell_0.5.0
[49] scales_1.2.1       gtools_3.9.4       class_7.3-22       glue_1.6.2
[53] lazyeval_0.2.2     tools_4.3.1        dotCall64_1.0-2    grid_4.3.1
[57] cyclocomp_1.1.1    crosstalk_1.2.0    colorspace_2.1-0   cli_3.6.1
[61] spam_2.9-1         fansi_1.0.4        gtable_0.3.3       digest_0.6.33
[65] classInt_0.4-10    htmlwidgets_1.6.2  farver_2.1.1       htmltools_0.5.6
[69] lifecycle_1.0.3    MASS_7.3-60
```