Ruminant Feed Balance Modelling in R

January 2025


1 Introduction

This course is designed to guide learners on ruminant feed balance modelling in R. The course gives an overview of the concept of feed balance modelling, and introduces the main modelling steps based by Fraval et al. (2024). Learners are taught how to gather spatial and non-spatial data, pre-process it, and ultimately assess feed balances. The methodology has be implemented in Ethiopia, Burkina Faso and Nigeria. The codes and data in these course are for Nigeria. The course emphasizes practical learning, with learners working on hands-on sessions. Detailed explanations are provided, helping learners follow along with the material and providing a valuable resource for future reference.

2 Background

Feed balances are used to evaluate the adequacy of available livestock feed resources in meeting the dietary requirements of livestock (Mottet and Assouma 2024). Feed resources typically include natural grasses, browse, crop by-products (e.g., crop residues), and agro-industrial by-products. Feed balance assessments are conducted at specific geographical scales and over defined time periods. In this course, we focus on estimating feed (energy) balance for ruminant livestock at the national scale, while ensuring relevance at sub-national levels and across different time frames.

2.1 Objectives

The objectives of this course are to:-

- Teach participants where to collect spatial and non-spatial data required for livestock feed balance assessment for Nigeria.
- Teach participants how to import, edit, and export spatial and non-spatial data in preparation for feed balance modeling.
- Guide participants in evaluating livestock feed supply and determining livestock nutritional requirements.
- Equip participants with the skills to assess feed balances, allowing them to evaluate if livestock feed supply meets demand.

2.2 Learning Outcomes

By the end of the course, learners will be able to:-

- Locate and collect essential spatial and non-spatial data for livestock feed balance assessment for Nigeria
- Import, edit, and export spatial and non-spatial data for use in livestock feed balance assessments.
- Assess livestock feed supply and determine livestock nutritional requirements.
- Evaluate feed supply against livestock requirements to assess feed balances.

2.3 Approach

The approach involves estimating (i) feed availability using freely accessible geospatial data and (ii) ruminant livestock feed requirements, both assessed over time. Figure 1 provides an overview of the feed balance model design.

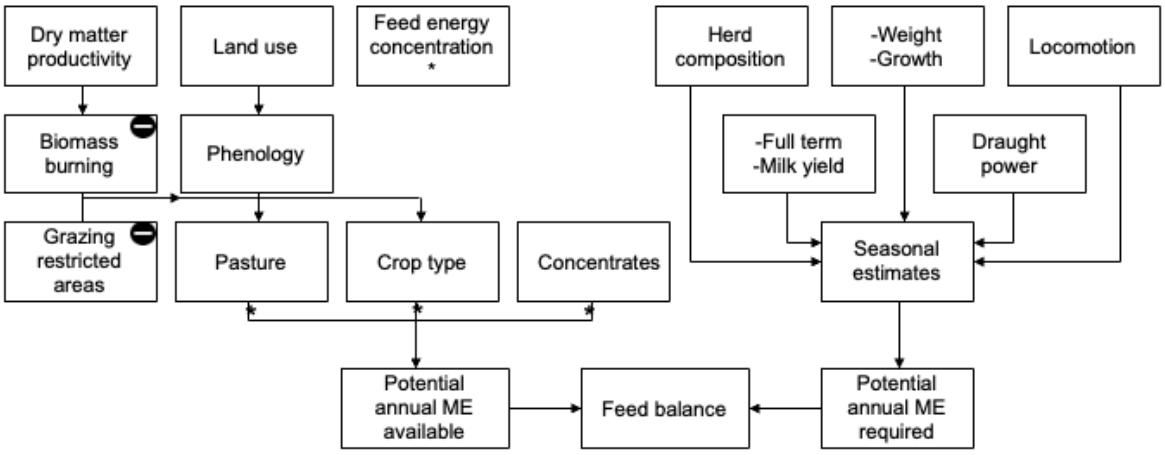

Figure 1: Feed balance model for ruminant livestock. A minus (−) indicates excluded biomass, while an asterisk (*) denotes inclusion of feed concentration values in subsequent calculations. ME = Metabolizable Energy.

The course is organized into three workflows: `1Data-download` for downloading data, `2Feed-geoprocessing` for estimating feed availability and `3Balance-estimates` for estimating feed balances.

3 Setting up R environment

Learners will need to install the following free and open source software:-

- R programming language (4.4+) https://cran.r-project.org.
- R Studio Desktop (3.6+) https://rstudio.com/products/rstudio/download/#download.

Create a new folder within your work space and name it `feed-balance-modelling`. Using R, we assign the path to the folder the variable name `root`.

For Linux/Unix systems

```
# Linux systems
root <- "/home/feed-balance-modelling"
```

For Windows systems

```
# Windows systems
root <- "C:/Documents/feed-balance-modelling"
```

Next, we set the country of interest.

```
country <- "Nigeria"
```

4 Data collation

This tutorial aims to guide learners through the key spatial and non-spatial data sources required for assessing feed balance. Learners will be introduced to a variety of datasets, and by the end of the tutorial, they will understand the necessary data requirements and know how to gather these datasets for feed balance modeling.

The tutorial leverages freely available spatial and non-spatial data to perform feed balance modeling. Recent advancements in R packages have enabled streamlined access to spatial data, including administrative boundaries, climatic variables, and environmental datasets. The required data is listed in Table 1.

Table 1: List of datasets required for feed balance modelling

Dataset	**Description**	**Availability**	**Spatial resolution**	**Year**	**Source**
Administrative boundaries	Administrative boundaries for target country	Global	NA	2023	GADM (2024)
Aggregation zones	Areas with similar livestock management practices	Regional	NA	2018	FMARD (2024)
Land use	Fractional cover of different land uses	Global	100 meter	2019	NA
Above ground dry matter productivity	Vegetation's overall growth rate	Global	300 meter	2020-2023	NA
Crop type and area	Location, extent, and patterns of feedable crops	Global	10 kilometer	2020	NA
Phenolology	Seasonal pattern of variation in vegetated land surfaces observed from remote sensing	Global	500 meter	2020-2023	NA
Burned areas	Burnt scars	Global	300 meter	2020-2023	NA
Protected areas	Marine and terrestrial protected areas	Global	NA	2024	NA
Tree cover	Forest and non-forest tree cover	Global, Regional	100 meter	2019	NA
Crop harvest index	Ratio of harvested product dry weight to total above-ground biomass dry weight at plant maturity	Local	NA	2024	NA
Feed parameters	Nutritional quality of feed items/type e.g., dry matter digestibility, crude protein	Local	NA	2024	NA
Livestock population	Type and number of livestock	Global	10 kilometer	2020	NA
Livestock parameters	Animal characteristics e.g., live weight, age	Local	NA	2024	NA

We provide brief reproducible examples illustrating how to download and utilize such data for feed balance modeling.

4.1 Administrative boundaries

We create a new folder under `feed-balance-modelling`, and name it `AdminBound` and assign it the variable name `outdir`. We can download administrative boundaries of world countries with the `geodata` R package. Here we use `geodata` to download the administrative boundaries for levels `0`, `1` and `2` for Nigeria from `Database of Global Administrative Areas` (GADM 2024), and store the data in `AdminBound` folder.

```
library(geodata)
library(sf)

country <- "Nigeria"

outdir <- paste0(root, "/src/1Data-download/SpatialData/inputs/AdminBound/",
 ↪  country)
dir.create(outdir, F, T)

admin_levels <- c("0", "1", "2")
for (admin_level in admin_levels) {
    aoi <- geodata::gadm(country = "NGA", level = admin_level, path =
 ↪  paste0(outdir),
        version = "latest") %>%
        sf::st_as_sf()
    write_sf(aoi, paste0(outdir, "/gadm40_NGA_", admin_level, ".shp"), append
     ↪  = FALSE)
    write_sf(aoi, paste0(outdir, "/aoi", admin_level, ".shp"), append =
     ↪  FALSE)
}
```

4.2 Aggregation zones

We use ecological and feed distribution zones as defined by FMARD (2024), the most recent version can be downloaded at https://drive.google.com/file/d/10HsGspftDNgq-fjAjeUwB8QsmHZWUJyT/view?usp=sharing.

```
library(googledrive)
```

```
outdir <- paste0(root,
 ↪  "/src/1Data-download/SpatialData/inputs/AggregationZones")
dir.create(outdir, F, T)

drive_deauth()
drive_user()

public_file <- drive_get(as_id("10HsGspftDNgq-fjAjeUwB8QsmHZWUJyT"))
drive_download(public_file, path = paste0(outdir,
 ↪  "/Ecological_and_Feed_Distribution.zip"),
    overwrite = TRUE)

unzip(zipfile = paste0(outdir, "/Ecological_and_Feed_Distribution.zip"),
 ↪  exdir = paste0(outdir,
    "/"))
```

4.3 Feed parameters

Several feed parameters are needed for model parameterization. In this course, we estimate the metabolizable energy (ME) concentration of feeds using species-specific estimates derived from literature reviews and databases such as the Sub-Saharan Africa Feeds composition Database (Link: https://feedsdatabase.ilri.org. Feed quality parameters include metabolizable Energy (ME), neutral detergent fibre (NDF), in vitro organic matter digestibility (IVOMD), and crude protein (CP). To facilitate model parameterization, a comprehensive feed parameter file has been prepared and is available for download at: https://drive.google.com/drive/folders/1SpB1p9i4MGU1gMahF4M3Uc-HZr8FGoqd

```
outdir <- paste0(root, "/src/1Data-download/Tables/inputs/", country,
 ↪  "/CropParams")
dir.create(outdir, F, T)

drive_deauth()
drive_user()

# folder link to id
public_folder <-
 ↪  "https://drive.google.com/drive/folders/1SpB1p9i4MGU1gMahF4M3Uc-HZr8FGoqd"
folder_id <- drive_get(as_id(public_folder))

# find files in folder
public_files <- drive_ls(folder_id)
```

```
for (i in 1:nrow(public_files)) {
    public_file <- public_files[i, ]
    file_name <- public_file$name
    drive_download(public_file, path = paste0(outdir, "/", file_name),
     ↪  overwrite = TRUE)
}
```

Other useful websites for feed parameters include:

- Feedipedia https://www.feedipedia.org
- Tropical forages http://www.tropicalforages.info

4.4 Livestock parameters

Several livestock parameters are needed for model parameterization. In this course, we utilize the Gridded Livestock of the World (GLW) database (Gilbert et al. 2018) to spatially disaggregate herds and flocks into species and categories including sheep, goats (adult and young), cattle (bulls, steers, cows, heifers, and calves), horses, and donkeys. We compile essential livestock parameters to characterize herd dynamics and productivity. These parameters include: live weight, age, daily weight gain, fertility rate, lactation length, daily milk yield, annual milk yield, daily walking distance, and proportion of population used for work. To facilitate model parameterization, a comprehensive feed parameter file has been prepared and is available for download at: https://drive.google.com/drive/folders/1-3N_kmMgcHr_tayylSxlMJAr-2PBGFXd

```
outdir <- paste0(root, "/src/1Data-download/Tables/inputs/", country,
 ↪  "/LivestockParams")
dir.create(outdir, F, T)

drive_deauth()
drive_user()

# folder link to id
public_folder =
 ↪  "https://drive.google.com/drive/folders/1-3N_kmMgcHr_tayylSxlMJAr-2PBGFXd"
folder_id = drive_get(as_id(public_folder))

# find files in folder
public_files = drive_ls(folder_id)

for (i in 1:nrow(public_files)) {
```

```
    public_file <- public_files[i, ]
    file_name <- public_file$name
    drive_download(public_file, path = paste0(outdir, "/", file_name),
     ↪  overwrite = TRUE)
}
```

Extended workflows and detailed scripts for collating other required datasets are available at https://github.com/ilri/ruminant-feed-balance/tree/main/src/1Data-download

5 Feed geoprocessing

This tutorial aims to guide learners through the geo-processiong techniques used for making the datasets we collated in the previous session ready for feed balance modelling. The workflow is comprised of R scripts that should be executed in numeric-alphabetic sequence - e.g 0a, 0b, 1, 2a, 2b etc.

Here we provide brief reproducible examples illustrating execute some of the R scripts.

5.1 Cropping rasters

The spatial datasets downloaded in the previous session included both vector and raster data. For example `dry matter productivity` herein referred to as `DMP` layers have a global extent, `aggregation zones` are at a national extent, the `tree cover` data is at a regional extent, and the `protected areas` data is provided as a vector layer.

To focus our analysis on the area of interest, we will crop the layers from their global or regional extents to a smaller extent. We will use the boundary of Nigeria as the cropping extent to refine the datasets for the analysis.

5.1.1 Dry Matter Productivity

We begin by processing the `Dry Matter Productivity` layers. We create an outputs folder `outdir` to store the clipped files. We then read in the Nigeria boundary layer `aoi` to define the area of interest. Next, we list all `NetCDF` files downloaded in the previous session. We iterate through the files, cropping each to the specified area of interest, and save the resulting clipped files in the `outdir` folder.

```r
# Load required packages
library(dplyr)
library(raster)
library(rgdal)
library(sf)

# output folder
outdir <- paste0(root, "/src/2Feed-geoprocessing/SpatialData/inputs/",
 ↪  country, "/Feed_DrySeason/DMP")
dir.create(outdir, F, T)

# read AOI
aoi <- read_sf(paste0(root,
 ↪  "/src/1Data-download/SpatialData/inputs/AdminBound/",
    country, "/aoi0.shp"))

nc_files <- list.files("/src/1Data-download/SpatialData/inputs/Feed/DMP",
 ↪  pattern = ".nc$",
    full.names = TRUE, recursive = TRUE)

for (nc_file in nc_files) {

    nc_name <- gsub(".{3}$", "", basename(nc_file))

    iDMP <- raster::raster(nc_file, varname = "DMP", ncdf = TRUE)
    iDMP <- crop(iDMP, extent(aoi))
    iDMP <- mask(iDMP, aoi)

    # save as GeoTIFF
    raster::writeRaster(iDMP, filename = paste0(outdir, "/", nc_name,
 ↪  ".tif"), overwrite = TRUE)

}
```

5.1.2 Protected areas

As mentioned earlier, the `protected areas` layer is provided as a vector layer and will be converted into a raster format. A reference layer from the `DMP` layer collection is used for clipping and resampling during this process.

```
# Libraries
library(terra)

indir <- paste0(root,
 ↪  "/src/1Data-download/SpatialData/inputs/ProtectedAreas")
outdir <- paste0(root, "/src/2Feed-geoprocessing/SpatialData/inputs/",
 ↪  country, "/ProtectedAreas")
dir.create(outdir, F, T)

# load livelihoods vectors
wdpaNGA <- vect(paste0(indir, "/WDPA_WDOECM_Oct2024_Public_NGA.shp"))

# reference raster? r <- rast(ext(wdpaNGA), resolution = 0.00297619, crs =
# crs(wdpaNGA))
dmpTemp <- rast(paste0(root, "/src/2Feed-geoprocessing/SpatialData/inputs/",
 ↪  country,

     ↪  "/Feed_DrySeason/DMP/c_gls_DMP300-RT6_202301100000_GLOBE_OLCI_V1.1.2.tif"))

# rasterize the SpatVector
wdpaNGA <- rasterize(wdpaNGA, dmpTemp, field = "STATUS_YR")

# Write output
writeRaster(wdpaNGA, paste0(outdir, "/WDPAGlobal.tif"), overwrite = TRUE)
```

5.1.3 Crop digestible fraction

We derive `crop digestible` fraction layers by calculating feedable versus non-feedable crops on using the `crop type and distribution` layers.

```
# Load libraries
library(terra)
library(dplyr)
library(readr)

# Runs with 16gb ram and 40+gb hdd space
terraOptions(tempdir = "/home/scratch/AUTemp")
terraOptions(memfrac = 0.5)
terraOptions(todisk = TRUE)

pathLU <- paste0(root, "/src/2Feed-geoprocessing/SpatialData/inputs/",
 ↪  country, "/Feed_DrySeason/LandUse")
```

```
filesLU <- list.files(path = pathLU, pattern = ".tif$", full.names = T)

pathSPAM <- paste0(root, "/src/2Feed-geoprocessing/SpatialData/inputs/",
 ↪  country,
    "/SPAM2020")
pathSPAMInter <- paste0(root, "/src/2Feed-geoprocessing/SpatialData/inputs/",
 ↪  country,
    "/SPAM2020/intermediate")
dir.create(pathSPAMInter, F, T)

# end of file name should be physical area_cropname_a
filesSPAM <- list.files(path = pathSPAM, pattern = "_a.tif$", full.names = T)

# stSPAM <- stack(filesSPAM)
stSPAM <- rast(filesSPAM)

### Calculate non-feed crops proportion from SPAM model.
tmpNonFeed <- read_csv(paste0(root,
 ↪  "/src/1Data-download/Tables/inputs/CropParams/crop_harvest index.csv"))
 ↪  %>%
    filter(Excluded != "0") %>%
    pull(codeSPAM) %>%
    unique()
tmpNonFeedIndex <- grep(pattern = paste(tmpNonFeed, collapse = "|"),
 ↪  names(stSPAM))
iSPAMtotalArea <- app(stSPAM, fun = sum, na.rm = TRUE)
iSPAMnonFeedArea <- app(stSPAM[[tmpNonFeedIndex]], fun = sum, na.rm = TRUE)

iSPAMnonFeedFrac <- (iSPAMnonFeedArea/iSPAMtotalArea)
iSPAMnonFeedFrac <- classify(iSPAMnonFeedFrac, cbind(NA, NA, 0), right =
 ↪  FALSE)  # Replace missing values with 0
iSPAMAnimalDigestCropFrac <- 1 - iSPAMnonFeedFrac

iSPAMAnimalDigestCropFrac <- writeRaster(iSPAMAnimalDigestCropFrac,
 ↪  paste0(pathSPAMInter,
    "/animal_digest_frac.tif"), overwrite = T)
```

5.1.4 Crop residue fraction

We derive `crop residue` fraction layers by combining data on `crop type and distribution` with `harvest index` obtained from published sources.

```
# Load libraries
library(dplyr)
library(raster)
library(rgdal)
library(readr)

# Runs with 16gb ram and 40+gb hdd space
rasterOptions(tmpdir = "/home/scratch/AUTemp")
rasterOptions(maxmemory = 1e+60)
rasterOptions(todisk = TRUE)

# setwd('/exports/eddie/scratch/sfraval/feed-surfaces/')
cropHI <- read_csv(paste0(root,
 ↪  "/src/1Data-download/Tables/inputs/CropParams/crop_harvest index.csv"))

pathSPAM <- paste0(root, "/src/2Feed-geoprocessing/SpatialData/inputs/",
 ↪  country,
    "/SPAM2020")
pathSPAMInter <- paste0(root, "/src/2Feed-geoprocessing/SpatialData/inputs/",
 ↪  country,
    "/SPAM2020/intermediate")

# end of file name should be physical area_cropname_a
filesSPAM <- list.files(path = pathSPAM, pattern = "_a.tif$", full.names = T)
stSPAM <- stack(filesSPAM)
gc()

crops <- sub(".*_a_(.*?)_a\\.tif$", "\\1", filesSPAM)

tmpCropIndex <- grep(pattern = paste(crops, collapse = "|"), names(stSPAM))
iSPAMcropArea <- calc(stack(stSPAM[[tmpCropIndex]]), fun = sum, na.rm = TRUE)

print("past 1")

stHI <- stack()
stSPAMcropProp <- stack()
gc()
for (i in 1:length(crops)) {
    tmpCropIndex <- grep(pattern = paste(crops[i], collapse = "|"),
 ↪  names(stSPAM))
    iSPAMtmpArea <- overlay(stSPAM[[tmpCropIndex]], fun = sum, na.rm = TRUE)
    icrop <- stSPAM[[tmpCropIndex]]
```

```
    icrop[icrop > 0] <- (1 - cropHI$harvest_index[cropHI$codeSPAM ==
 ↪  crops[i]])
    stHI <- stack(stHI, icrop)

    stSPAMcropProp <- stack(stSPAMcropProp, overlay(iSPAMtmpArea,
 ↪  iSPAMcropArea,
        fun = function(x, y) {
            (x/y)
        }))

    print(paste("Loop", i))
}
gc()

iSPAMcropResFrac <- weighted.mean(stHI, stSPAMcropProp, na.rm = T)
iSPAMcropResFrac <- reclassify(iSPAMcropResFrac, cbind(NA, NA, 0.8), right =
 ↪  FALSE)  #Assume that 80% is available for animals

print("past mean")

writeRaster(iSPAMcropResFrac, paste0(pathSPAMInter, "/crop_res_frac.tif"),
 ↪  overwrite = T)
```

We calculate the proportion of feed items i.e., cereals, roots, legumes, and oil crops using the `crop area and distribution` data. We write the new layers in the `SPAM2020` folder.

```
# Load libraries
library(terra)
library(readr)

terraOptions(tempdir = "/home/scratch/AUTemp")
terraOptions(memfrac = 0.5)
terraOptions(todisk = TRUE)

spamPath <- paste0(root, "/src/2Feed-geoprocessing/SpatialData/inputs/",
 ↪  country,
    "/SPAM2020")

cropLookup <- read_csv(paste0(root,
 ↪  "/src/1Data-download/Tables/inputs/CropParams/Crop classification_feed
 ↪  basket.csv"))
filesSPAM <- list.files(path = spamPath, pattern = "_a.tif$", full.names = T)
```

```
stCrops <- rast(filesSPAM)

# Index major feed types
indexCere <- grep(pattern =
 ↪  paste(cropLookup$codeSPAM[cropLookup$codeBasket_grouped_NGA ==
    "cere"], collapse = "|"), sub(".*_a_(.*?)_a\\.tif$", "\\1", filesSPAM))
indexRoots <- grep(pattern =
 ↪  paste(cropLookup$codeSPAM[cropLookup$codeBasket_grouped_NGA ==
    "roots"], collapse = "|"), sub(".*_a_(.*?)_a\\.tif$", "\\1", filesSPAM))
indexLeg <- grep(pattern =
 ↪  paste(cropLookup$codeSPAM[cropLookup$codeBasket_grouped_NGA ==
    "leg"], collapse = "|"), sub(".*_a_(.*?)_a\\.tif$", "\\1", filesSPAM))
indexOilc <- grep(pattern =
 ↪  paste(cropLookup$codeSPAM[cropLookup$codeBasket_grouped_NGA ==
    "oilc"], collapse = "|"), sub(".*_a_(.*?)_a\\.tif$", "\\1", filesSPAM))

# Extraction area for major feed categories
areaTotal <- app(stCrops, fun = sum, na.rm = T)
areaCere <- app(stCrops[[indexCere]], fun = sum, na.rm = T)
areaRoots <- app(stCrops[[indexRoots]], fun = sum, na.rm = T)
areaLeg <- app(stCrops[[indexLeg]], fun = sum, na.rm = T)
areaOilc <- app(stCrops[[indexOilc]], fun = sum, na.rm = T)

# Calculate proportions
propCere <- areaCere/areaTotal
propRoots <- areaRoots/areaTotal
propLeg <- areaLeg/areaTotal
propOilc <- areaOilc/areaTotal

# Write outputs
writeRaster(propCere, paste0(spamPath, "/propCereSPAM.tif"), overwrite =
 ↪  TRUE)
writeRaster(propRoots, paste0(spamPath, "/propRootsSPAM.tif"), overwrite =
 ↪  TRUE)
writeRaster(propLeg, paste0(spamPath, "/propLegSPAM.tif"), overwrite = TRUE)
writeRaster(propOilc, paste0(spamPath, "/propOilcSPAM.tif"), overwrite =
 ↪  TRUE)
```

Additional workflows and detailed scripts for preparing other files are available at:

- Landuse https://github.com/ilri/ruminant-feed-balance/blob/main/src/2Feed-geoprocessing/1bPrepareLanduse_clip.R

- Tree cover https://github.com/ilri/ruminant-feed-balance/blob/main/src/2Feed-geoprocessing/1cPrepareTreeCover_clip.R
- Digital Earth Africa land use https://github.com/ilri/ruminant-feed-balance/blob/main/src/2Feed-geoprocessing/1dPrepareDEALanduse_clip.R
- Phenology https://github.com/ilri/ruminant-feed-balance/blob/main/src/2Feed-geoprocessing/2aPreparePhenologyModis.R
- Burned area https://github.com/ilri/ruminant-feed-balance/blob/main/src/2Feed-geoprocessing/3aBurnedDaysclip.R
- SPAM https://github.com/ilri/ruminant-feed-balance/blob/main/src/2Feed-geoprocessing/4aSPAMclip.R
- Livestock population https://github.com/ilri/ruminant-feed-balance/blob/main/src/2Feed-geoprocessing/6prepareLivestockPopulation.R
- Livestock production systems https://github.com/ilri/ruminant-feed-balance/blob/main/src/2Feed-geoprocessing/7prepareLivestockSystems.R

5.1.5 Seasonal DM availability

We calculate the number of cropping days in a year by analyzing phenology data (e.g., green-up and senescence dates), over a time series `2020-2013`. We then define `wet` and `dry` periods. Next, we combine `crop residue`, `browse`, `natural grass` fractions, and `DMP` data to estimate feed availability for `dry` and `wet` season over the time series.

```
yearOffset <- (0 * 365)  # Base year = 2020

# Load libraries

library(dplyr)
library(raster)
library(rgdal)

rasterOptions(tmpdir = "/home/scratch/AUTemp")
rasterOptions(maxmemory = 5e+20)  # 6e+10 ~51GB allowed
rasterOptions(todisk = TRUE)

# read AOI
aoi <- readOGR(paste0(root,
 ↪  "/src/1Data-download/SpatialData/inputs/AdminBound/",
    country, "/aoi0.shp"))

yearList <- c("2020", "2021", "2022", "2023")

lapply(yearList, function(year) {
```

```
  cropOutdir <- paste0(root,
↪ "/src/2Feed-geoprocessing/SpatialData/inputs/", country,
      "/Cropping_days")
  dir.create(cropOutdir, F, T)
  FeedQuantityOutdir <- paste0(root,
↪ "/src/2Feed-geoprocessing/SpatialData/inputs/",
      country, "/Feed_DrySeason/Feed_quantity/", year)
  dir.create(FeedQuantityOutdir, F, T)

  pathPhen <- paste0(root, "/src/2Feed-geoprocessing/SpatialData/inputs/",
↪ country,
      "/Feed_DrySeason/PhenologyModis/", year, "/outputTif")
  filesPhenology <- list.files(path = pathPhen, pattern = ".tif$",
↪ full.names = T)

  pathDMP <- paste0(root, "/src/2Feed-geoprocessing/SpatialData/inputs/",
↪ country,
      "/Feed_DrySeason/DMP")
  filesDMP <- list.files(path = pathDMP, pattern = paste0("RT6_", year,
↪ ".*\\.tif$"),
      full.names = TRUE)
  stDMP <- stack(filesDMP)

  datesDMP <- sub(".*RT6_(.{8}).*", "\\1", filesDMP)
  datesDMP <- as.Date(datesDMP, "%Y%m%d")
  datesDMPdiff <- as.numeric(datesDMP - as.Date("1970/01/01"))  #convert to
↪ same date format as Modis phenology
  pathLU <- paste0(root, "/src/2Feed-geoprocessing/SpatialData/inputs/",
↪ country,
      "/Feed_DrySeason/LandUse")
  filesLU <- list.files(path = pathLU, pattern = "300.tif$", full.names =
↪ T)
  pathSPAM <- paste0(root, "/src/2Feed-geoprocessing/SpatialData/inputs/",
↪ country,
      "/SPAM2020")

  # end of file name should be physical area_cropname_a
  filesSPAM <- list.files(path = pathSPAM, pattern = "_a.tif$", full.names
↪ = T)
  iSPAMAnimalDigestCropFrac <- raster(paste0(root,
↪ "/src/2Feed-geoprocessing/SpatialData/inputs/",
```

```
    country, "/SPAM2020/animal_digest_frac.tif"))

rProtectedAreas <- stack(paste0(root,
↪ "/src/2Feed-geoprocessing/SpatialData/inputs/",
    country, "/ProtectedAreas/WDPAGlobal.tif"))
rNonProtectedAreas <- calc(rProtectedAreas, fun = function(x) {
    ifelse(x == 0, 1, 0)
})
rm(rProtectedAreas)

print("past protected")

stLU <- stack(filesLU)
LUcrops300DEA <- raster(paste0(pathLU, "/LUcrops300DEA.tif"))

stPhen <- stack(raster(grep("phenoGreenup1.tif", filesPhenology, value =
↪ TRUE)),
    raster(grep("phenoSenescence1.tif", filesPhenology, value = TRUE)),
     ↪  raster(grep("phenoGreenup2.tif",
        filesPhenology, value = TRUE)),
↪ raster(grep("phenoSenescence2.tif", filesPhenology,
        value = TRUE)))
gc()

## Crop land use to test area
LUcrops300DEA <- extend(LUcrops300DEA, extent(stDMP[[1]]))
LUcrops300DEA <- crop(LUcrops300DEA, extent(stDMP[[1]]))
LUcrops300DEA <- mask(LUcrops300DEA, aoi)
stLU <- extend(stLU, extent(stDMP[[1]]))
stLU <- crop(stLU, extent(stDMP[[1]]))
stLU <- mask(stLU, aoi)

## Revise grass and shrub area
diffCrop <- LUcrops300DEA - stLU$LUcrops300
stLU$LUgrassShrub300 <- sum(stLU$LUgrass300, stLU$LUshrub300, na.rm = T)
stLU$LUgrassShrub300 <- stLU$LUgrassShrub300 - LUcrops300DEA

stLU$LUcrops300 <- LUcrops300DEA

stPhen$phenoGreenup2 <- calc(stPhen$phenoGreenup2, fun = function(x) {
    ifelse(x > max(datesDMPdiff) + 30, NA, x)
})
```

```
    stPhen$phenoSenescence2 <- calc(stPhen$phenoSenescence2, fun =
↪   function(x) {
        ifelse(x > max(datesDMPdiff) + 30, NA, x)
    })
    stPhen$phenoGreenup1 <- calc(stPhen$phenoGreenup1, fun = function(x) {
        ifelse(x < min(datesDMPdiff) - 30, NA, x)
    })
    stPhen$phenoSenescence1 <- calc(stPhen$phenoSenescence1, fun =
↪   function(x) {
        ifelse(x < min(datesDMPdiff) - 30, NA, x)
    })
    gc()

    growing2 <- (stPhen$phenoSenescence2 - stPhen$phenoGreenup2)
    growing2 <- reclassify(growing2, c(365, Inf, 0))
    growingDays <- sum((stPhen$phenoSenescence1 - stPhen$phenoGreenup1),
↪   growing2,
        na.rm = T)
    growingDays <- reclassify(growingDays, c(300, Inf, 300))
    gc()

    writeRaster(growingDays, paste0(cropOutdir, "/croppingDays_", year,
     ↪  ".tif"),
        overwrite = T)

    print("past 0")

    names(stDMP) <- paste0("d", datesDMPdiff)

    stLU$LUtree300 <- reclassify(stLU$LUtree300, c(-Inf, 0, 0, 200, Inf, 0))
    stLU$LUtree300[is.na(stLU$LUtree300)] <- 0

    print("past 1")

    ##### Estimate total DMP per ha
    grassFracDry <- 1  # 0.33 #max 0.55
    grassFracWet <- 1  # 0.55 #max 0.55
    browseShrubFrac <- 1  #0.38 #max 0.38
    browseForestFrac <- 1
    iResidueUtil <- 1  #max 0.6
    iSPAMHarvestResidueFrac <- raster(paste0(root,
↪   "/src/2Feed-geoprocessing/SpatialData/inputs/",
```

```
    country, "/SPAM2020/crop_res_frac.tif"))
gc()

print("past overlay 1")

residueFrac <- iSPAMHarvestResidueFrac

shrubFrac <- raster(paste0(root,
↪ "/src/2Feed-geoprocessing/SpatialData/inputs/",
    country, "/TreeCover/treecover300m.tif"))/100

gc()

funGrowingGrassWet <- function(dmp, crops, grassShrub, forest, shrubFrac,
↪ greenup,
    senesence, greenup2, senesence2, nonprotected) {
    ifelse((greenup <= datesDMPdiff[i] & senesence >= datesDMPdiff[i]) |
     ↪ (greenup2 <=
        datesDMPdiff[i] & senesence2 >= datesDMPdiff[i]), (dmp * 9 *
↪ grassShrub *
        grassFracWet * (1 - shrubFrac)) + (dmp * 9 * forest *
↪ grassFracWet *
        (1 - shrubFrac) * nonprotected), NA)
}  #@feedFrac is the proportion of crops grown that have feedable
↪ residues - i.e. excluding coffee, tea, ect.
funGrowingGrassDry <- function(dmp, crops, grassShrub, forest, shrubFrac,
↪ greenup,
    senesence, greenup2, senesence2, nonprotected) {
    ifelse((greenup > datesDMPdiff[i]) | (senesence < datesDMPdiff[i] &
     ↪ senesence +
        60 > datesDMPdiff[i]) | (senesence2 < datesDMPdiff[i]), (dmp * 9
         ↪  * grassShrub *
        grassFracDry * (1 - shrubFrac)) + (dmp * 9 * forest *
↪ grassFracDry *
        (1 - shrubFrac) * nonprotected), NA)
}  #@feedFrac is the proportion of crops grown that have feedable
↪ residues - i.e. excluding coffee, tea, ect.
funGrowingBrowse <- function(dmp, crops, grassShrub, forest, shrubFrac,
↪ nonprotected) {
    (dmp * 9 * grassShrub * shrubFrac * browseShrubFrac) + (dmp * 9 *
↪ forest *
        nonprotected * shrubFrac * browseForestFrac)
```

```
    }
    funGrowingCrops <- function(dmp, crops, greenup, senesence, feedFrac,
↪   resFrac,
        greenup2, senesence2) {
        ifelse((greenup <= datesDMPdiff[i] & senesence >= datesDMPdiff[i]) |
         ↪  (greenup2 <=
            datesDMPdiff[i] & senesence2 >= datesDMPdiff[i]), (dmp * 9 *
↪   crops),
            NA)
    }  #@feedFrac is the proportion of crops grown that have feedable
↪   residues - i.e. excluding coffee, tea, ect.
    funGrowingAftermath <- function(dmp, crops, greenup, senesence, greenup2,
↪   senesence2,
        nonprotected) {
        ifelse((greenup > datesDMPdiff[i]) | (senesence < datesDMPdiff[i] &
         ↪  senesence +
            60 > datesDMPdiff[i]) | (senesence2 < datesDMPdiff[i]), (dmp * 9
             ↪  * crops),
            NA)
    }  #@feedFrac is the proportion of crops grown that have feedable
↪   residues - i.e. excluding coffee, tea, ect.

    for (i in 1:length(names(stDMP))) {

        iDMPGrassGrowing <- overlay(stDMP[[i]], stLU$LUcrops300,
↪   stLU$LUgrassShrub300,
            stLU$LUtree300, shrubFrac, stPhen$phenoGreenup1,
↪   stPhen$phenoSenescence1,
            stPhen$phenoGreenup2, stPhen$phenoSenescence2,
↪   rNonProtectedAreas, fun = funGrowingGrassWet)
        writeRaster(iDMPGrassGrowing, paste0(FeedQuantityOutdir,
         ↪  "/grassWetDMP",
            datesDMP[i], ".tif"), overwrite = TRUE)
        rm(iDMPGrassGrowing)
        gc()

        iDMPGrassDry <- overlay(stDMP[[i]], stLU$LUcrops300,
↪   stLU$LUgrassShrub300,
            stLU$LUtree300, shrubFrac, stPhen$phenoGreenup1,
↪   stPhen$phenoSenescence1,
            stPhen$phenoGreenup2, stPhen$phenoSenescence2,
↪   rNonProtectedAreas, fun = funGrowingGrassDry)
```

```
        writeRaster(iDMPGrassDry, paste0(FeedQuantityOutdir, "/grassDryDMP",
         ↪  datesDMP[i],
            ".tif"), overwrite = TRUE)
        rm(iDMPGrassDry)
        gc()

        iDMPBrowse <- overlay(stDMP[[i]], stLU$LUcrops300,
↪  stLU$LUgrassShrub300,
            stLU$LUtree300, shrubFrac, rNonProtectedAreas, fun =
↪  funGrowingBrowse)
        writeRaster(iDMPBrowse, paste0(FeedQuantityOutdir, "/browseDMP",
         ↪  datesDMP[i],
            ".tif"), overwrite = TRUE)
        rm(iDMPBrowse)
        gc()

        iDMPCropGrowing <- overlay(stDMP[[i]], stLU$LUcrops300,
↪  stPhen$phenoGreenup1,
            stPhen$phenoSenescence1, iSPAMAnimalDigestCropFrac, residueFrac,
↪  stPhen$phenoGreenup2,
            stPhen$phenoSenescence2, fun = funGrowingCrops)
        writeRaster(iDMPCropGrowing, paste0(FeedQuantityOutdir, "/cropDMP",
         ↪  datesDMP[i],
            ".tif"), overwrite = TRUE)
        rm(iDMPCropGrowing)
        gc()

        iDMPAftermath <- overlay(stDMP[[i]], stLU$LUcrops300,
↪  stPhen$phenoGreenup1,
            stPhen$phenoSenescence1, stPhen$phenoGreenup2,
↪  stPhen$phenoSenescence2,
            fun = funGrowingAftermath)
        writeRaster(iDMPAftermath, paste0(FeedQuantityOutdir,
         ↪  "/aftermathDMP", datesDMP[i],
            ".tif"), overwrite = TRUE)
        rm(iDMPAftermath)
        gc()
        print(paste("cycle", i))

    }

    gc()
```

```
    iDMPgrassWet <- stack(list.files(path = paste0(FeedQuantityOutdir),
↪   pattern = "grassWet",
        full.names = T))
    DMPgrassmeanWet <- mean(iDMPgrassWet, na.rm = T)
    writeRaster(DMPgrassmeanWet, paste0(FeedQuantityOutdir,
     ↪  "/DMPgrassWetmean_",
        year, ".tif"), overwrite = TRUE)

    iDMPgrassDry <- stack(list.files(path = paste0(FeedQuantityOutdir),
↪   pattern = "grassDry",
        full.names = T))
    DMPgrassmeanDry <- mean(iDMPgrassDry, na.rm = T)
    writeRaster(DMPgrassmeanDry, paste0(FeedQuantityOutdir,
     ↪  "/DMPgrassDrymean_",
        year, ".tif"), overwrite = TRUE)

    iDMPbrowse <- stack(list.files(path = paste0(FeedQuantityOutdir), pattern
↪   = "browse",
        full.names = T))
    DMPbrowsemean <- mean(iDMPbrowse, na.rm = T)
    writeRaster(DMPbrowsemean, paste0(FeedQuantityOutdir, "/DMPbrowsemean_",
     ↪  year,
        ".tif"), overwrite = TRUE)

    iDMPCropGrowing <- stack(list.files(path = paste0(FeedQuantityOutdir),
↪   pattern = "crop",
        full.names = T))
    DMPcropmean <- mean(iDMPCropGrowing, na.rm = T)
    writeRaster(DMPcropmean, paste0(FeedQuantityOutdir, "/DMPcropmean_",
     ↪  year, ".tif"),
        overwrite = TRUE)

    iDMPAftermath <- stack(list.files(path = paste0(FeedQuantityOutdir),
↪   pattern = "aftermath",
        full.names = T))
    DMPAftermean <- mean(iDMPAftermath, na.rm = T)
    writeRaster(DMPAftermean, paste0(FeedQuantityOutdir, "/DMPaftermean_",
     ↪  year,
        ".tif"), overwrite = TRUE)
    gc()
```

```
})
```

References

FMARD. 2024. “Ecologial and Feed Distribution Zones.”

Fraval, S., J. Y. Mutua, T. Amole, A. Tolera, T. Feyisa, P. K. Thornton, A. M. O. Notenbaert, et al. 2024. “Feed Balances for Ruminant Livestock: Gridded Estimates for Data Constrained Regions.” *Animal*, May, 101199. https://doi.org/10.1016/j.animal.2024.101199.

GADM. 2024. “Database of Global Administrative Boundaries (GADM).” https://gadm.org.

Gilbert, Marius, Gaëlle Nicolas, Giusepina Cinardi, Thomas P. Van Boeckel, Sophie O. Vanwambeke, G. R.William Wint, and Timothy P. Robinson. 2018. “Global Distribution Data for Cattle, Buffaloes, Horses, Sheep, Goats, Pigs, Chickens and Ducks in 2010.” *Scientific Data 2018 5:1* 5 (1): 1–11. https://doi.org/10.1038/sdata.2018.227.

Mottet, Anne, and Mohamed Habibou Assouma. 2024. “The Feed Balances Sheet: A Tool for Planning the Use of Resources and Enhancing Resilience in Tropical Grazing Livestock.” *Frontiers in Animal Science* 5 (March). https://doi.org/10.3389/fanim.2024.1354728.