Appendix 2. R-codes for post-processing the outputs of ALUAM-AB

With the code below, Fig. 2, 3 and 4 from the paper were created. The input table for the codes below is "appendix4.csv", which is also supplied in the supplementary materials as an example. This table contains input variables and results from the ALUAM-AB simulations in which direct payments were varied. Some additional processing of the ALUAM-AB output was necessary to create the columns that are based on the linear regression between the time steps and the area of intensive or extensive agriculture (i.e. the columns sigBetaIntensive, sigBetaExtensive, areaIntensiveC10Y, areaExtensiveC10Y, betaIntensive and betaExtensive). However, since the output of every agent-based model is formatted differently, it is not possible to provide a generic R-script for these analysis steps. Below is a short description of the different columns in appendix4.csv:

Factor: The multiplication factor with which the direct payments were multiplied relative to the level of direct payments in 2001.
areaIntensive2002: The area of intensive agriculture in 2002 (ha)
areaExtensive2002: The area of extensive agriculture in 2002 (ha)
sigBetaIntensive: The p-value of the regression coefficient of the years regressed against the area of intensive agriculture.
sigBetaExtensive: The p-value of the regression coefficient of the years regressed against the area of extensive agriculture.
areaIntensiveC10Y: The change of the area of intensive agriculture over a 10-year period (ha; betaIntensive * 10).
areaExtensiveC10Y: The change of the area of extensive agriculture over a 10-year period (ha; betaExtensive * 10).
betaIntensive: The regression coefficient of the years regressed against the area of intensive agriculture (ha*year^{-1}).
betaExtensive: The regression coefficient of the years regressed against the area of extensive agriculture (ha*year^{-1}).

<u>Direction field plots</u>

```
#######################################
#### INPUT SETTINGS                ####
#######################################

# Specify the file in which the results of the ABM simulations are stored.
ResultsFile = "...\\appendix4.csv"
# Specify the folder in which the output figures should be stored
outputLoc = "D:\\..."

#######################################
#### CODE BODY                     ####
#######################################

# Load libraries
library(ggplot2)
library(scales)

#######################################
#### Load function                 ####
#######################################

# Define Multiple plot function
# From: http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects)
```

```
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  library(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

  if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}

#Read the results file
scenarioTabSel = read.csv(ResultsFile)

#Plot the direction field plots for intensive agriculture
intensivePlot = ggplot(data=scenarioTabSel, aes(x=Factor, y=areaIntensive2002, colour =
areaIntensiveC10Y))+
  geom_segment(aes(xend=Factor, yend=areaIntensive2002+areaIntensiveC10Y), arrow =
arrow(length = unit(0.3,"cm"), type = "closed"), size = 1.0)+
  xlab("Multiplication factor direct payments")+
  ylab("Area intensive agriculture (ha)")+
  scale_colour_gradientn(colours=c("Green","Light green","white","pink","red"), guide =
FALSE,values=rescale(c(max(scenarioTabSel$areaIntensiveC10Y),5,0,-
5,min(scenarioTabSel$areaIntensiveC10Y))))+
  theme_dark()+
  theme(legend.position='none',text=element_text(size=25))

#Plot the direction field plots for extensive agriculture
extensivePlot = ggplot(data=scenarioTabSel, aes(x=Factor, y=areaExtensive2002, colour =
areaExtensiveC10Y))+
  geom_segment(aes(xend=Factor, yend=areaExtensive2002+areaExtensiveC10Y), arrow =
arrow(length = unit(0.3,"cm"), type = "closed"), size = 1.0)+
  xlab("Multiplication factor direct payments")+
  ylab("Area extensive agriculture (ha)")+
  scale_colour_gradientn(colours=c("Green","Light green","white","pink","red"), guide = FALSE,
values=rescale(c(max(scenarioTabSel$areaExtensiveC10Y),5,0,-
5,min(scenarioTabSel$areaExtensiveC10Y))))+
  theme_dark()+
  theme(text=element_text(size=25))

#Save the plots
```

```
setwd(outputLoc)  #Set the workspace
pdf("DirectionFieldPlot.pdf",width=20,height=10)
multiplot(intensivePlot,extensivePlot, cols = 2)
dev.off()

#Save the plots
setwd(outputLoc)  #Set the workspace
jpeg("DirectionFieldPlot.jpg",width=40,height=20,units = "cm",res=300)
multiplot(intensivePlot,extensivePlot, cols = 2)
dev.off()
```

Stability landscapes

```
########################################
#### INPUT SETTINGS                 ####
########################################

# Specify the file in which the results of the ABM simulations are stored.
ResultsFile = "...\\appendix4.csv"
# Specify the folder in which the output figures should be stored
outputLoc = "D:\\..."

# Open an online plotly account. Follow the instructions from here: https://plot.ly/r/getting-
started/
Sys.setenv("plotly_username"="XXXXXXXXXX") #Enter plotly user-name
Sys.setenv("plotly_api_key"="XXXXXXXXXX") #Enter plotly password

########################################
#### CODE BODY                      ####
########################################

# Load libraries
library(raster)
library(plotly)
library(mmand)

########################################
#### Load functions                 ####
########################################

# Define the function to create the stability landscapes
# Requires:
#   x: vector with the values for the x-axis variable (i.e. system stressor)
#   y: vector with the values for the y-axis variable (i.e. system state)
#   beta: vector with the regression coefficient values (i.e. positive or negative growth)
#   size_raster: (optional) the numbers of rows and columns in the grid covering the state
space
#   MWsigma: (optional) the sigma parameters used to define the Gaussian weighting kernel
#   xlab: (optional) the label for the x-axis in the 3D stability landscape
#   ylab: (optional) the label for the y-axis in the 3D stability landscape
stabilityLS = function(x,y,beta,size_raster=c(25,25),MWsigma=c(1.5,1.5),xlab="x",ylab="y")
{
  # set up an 'empty' raster via an extent object
  e <- extent(c(min(x),max(x),min(y),max(y)))
  r <- raster(e, ncol=size_raster[1], nrow=size_raster[2])

  # Convert the beta values to a raster by taking the mean value. This raster will have holes
in it.
  x <- rasterize(data.frame(x,y),r,beta,fun = mean)

  # Create a Gaussian kernel with sigma = c(1.5,1.5) to use for the focal function below
  weights = gaussianKernel(sigma = MWsigma)

  # Apply focal statistics with a window of size_window x size_window and apply the mean of
this moving window.
  x_focal = focal(x, w=weights, fun = mean, pad = TRUE, padValue = NA, na.rm=TRUE)

  # Transform the output of the focal to a matrix
  z = as.matrix(x_focal)

  # Per column calcualte the cumulative sum of the column.
```

```
  cs = apply(z, 2, cumsum)

  # Define a function to rescale each column with a score range transformation.
  sr = function(x){
    return((x-min(x))/(max(x)-min(x)))
  }

  #Apply the sr function to the columns in cs.
  st_cs = apply(cs,2,sr)

  #Copy the values of the rescaled variables to a raster.
  dem = x_focal
  values(dem) = st_cs

  # Transform the coordinates of the raster to a dataframe and the z values to a matrix.
  xyz = as.data.frame(rasterToPoints(dem))
  z = matrix(xyz$layer, nrow = size_raster[1],byrow = TRUE)

  # Set a vector of colours to create the surface plot.
  cols2 <- c("#a6bddb","#feb24c","#f03b20")

  # Make a plot of the surface plot.
  p <- plot_ly(x = sort(unique(xyz$x)), y = sort(unique(xyz$y),decreasing = TRUE), z = ~z) %>%
    add_surface(colors = cols2, showscale=FALSE) %>%
    layout(scene = list(
      xaxis = list(title = xlab),
      yaxis = list(title = ylab),
      zaxis = list(title = "", zeroline = TRUE, showline = FALSE, showticklabels =
FALSE,showgrid = TRUE)
    ))
  return(p)
}

#Read the results file
scenarioTabSel = read.csv(ResultsFile)

# Make the stability landscape for intensive agriculture
plotInt = stabilityLS(x = scenarioTabSel$Factor,
                      y=scenarioTabSel$areaIntensive2002,
                      beta=scenarioTabSel$betaIntensive,
                      size_raster=c(25,25),
                      MWsigma=c(2,2),
                      xlab="",
                      ylab="")

# Plotly interactive plot
plotInt

# Make the stability landscape for extensive agriculture
plotExt = stabilityLS(x = scenarioTabSel$Factor,
                      y=scenarioTabSel$areaExtensive2002,
                      beta=scenarioTabSel$betaExtensive,
                      size_raster=c(25,25),
                      MWsigma=c(2,2),
                      xlab="",
                      ylab="")

# Plotly interactive plot
plotExt
```

Support vector machine classifications and bifurcation diagrams

```
#######################################
#### INPUT SETTINGS                ####
#######################################

# Specify the file in which the results of the ABM simulations are stored.
ResultsFile = "...\\appendix4.csv"
# Specify the folder in which the output figures should be stored
outputLoc = "D:\\..."
```

```
#######################################
#### CODE BODY                     ####
#######################################

# Load libraries
library(e1071)
library(ggplot2)

#######################################
#### Load functions                ####
#######################################

# Define Multiple plot function
# From: http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects)
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  library(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

  if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}

# Define the function to perform the Support Vector Machine tuning
# Requires:
#   x: Name of the x-axis variable in data (i.e. system stressor)
#   y: Name of the y-axis variable in data (i.e. system state)
#   beta: Name of the variable in data that contains the regression coefficients (i.e.
positive or negative growth)
#   data: Dataframe with the x, y and beta variables
#   res: Number of increments along the x- and y-axis for which predictions should be made.
SVMmod = function(x,y,beta,data,res=100)
{
  # Add a column with positive or negative correlations
  data$PosNeg = -1
  data$PosNeg[data[beta]>0] = 1
```

```
  data$PosNeg[data[beta]==0] = 0
  data$PosNeg = as.factor(data$PosNeg)

  # Only select those observations with a positive or negative growth
  svm_inputDF = data[data$PosNeg!=0,c("PosNeg",x,y)]

  # Tune the SVM with a 10-fold crossvalidated grid-search for the hyperparameters gamma and
cost.
  obj <- tune(svm, as.formula(paste("PosNeg ~", x,"+",y)), data=svm_inputDF,
              ranges = list(gamma = seq(0.01,1,length.out = 8), cost = seq(0.01,2,length.out =
8)),
              tunecontrol = tune.control(sampling = "cross", cross = 10)
  )

  # Make predictions for each area in the state space.
  x_range = seq(min(data[x]),max(data[x]),length.out = res)
  y_range = seq(min(data[y]),max(data[y]),length.out = res)
  xy_grid = expand.grid(x_range,y_range)
  names(xy_grid) = c(x,y)
  xy_grid$xy_pred = predict(obj$best.model, xy_grid)

  # Put the output in a list-object
  output = list(obj,data,xy_grid)
  return(output)
}

#Read the results file
scenarioTabSel = read.csv(ResultsFile)

# Define the colours for the points in the SVM-plots
colPoints = c("#e6550d","#31a354","#636363")
names(colPoints) = c(-1,1,0)

# Define the colours for the polygons in the SVM-plots
colPoly = c("#e6550d","#31a354")
names(colPoly) = c(-1,1)

# Run the SVM classifications for extensive agriculture
ExtMod = SVMmod(x = "Factor",
                y = "areaExtensive2002",
                beta = "betaExtensive",
                data = scenarioTabSel,
                res = 100)
print(ExtMod[[1]])

# Make a plot of the SVM results
ExtPlot = ggplot()+
  geom_point(data = ExtMod[[2]], aes(x = Factor, y = areaExtensive2002, colour = PosNeg),
show.legend = FALSE)+
  scale_color_manual(values = colPoints)+
  geom_raster(data = ExtMod[[3]], aes(x = Factor, y = areaExtensive2002,fill =
xy_pred),alpha=0.1, show.legend = FALSE)+
  scale_fill_manual(values = colPoly)+
  stat_contour(data = ExtMod[[3]], aes(x = Factor, y = areaExtensive2002, z =
as.numeric(xy_pred)), bins = 1, size = 2, color = "black")+
  xlab("Multiplication factor direct payments")+
  ylab("Initial area extensive agriculture (ha)")+
  theme(text = element_text(size=25))

# Run the SVM classifications for intensive agriculture
IntMod = SVMmod(x = "Factor",
                y = "areaIntensive2002",
                beta = "betaIntensive",
                data = scenarioTabSel,
                res = 100)
print(IntMod[[1]])

# Make a plot of the SVM results
IntPlot = ggplot()+
  geom_point(data = IntMod[[2]], aes(x = Factor, y = areaIntensive2002, colour = PosNeg),
show.legend = FALSE)+
  scale_color_manual(values = colPoints)+
```

```
  geom_raster(data = IntMod[[3]], aes(x = Factor, y = areaIntensive2002,fill =
xy_pred),alpha=0.1, show.legend = FALSE)+
  scale_fill_manual(values = colPoly)+
  stat_contour(data = IntMod[[3]], aes(x = Factor, y = areaIntensive2002, z =
as.numeric(xy_pred)), bins = 1, size = 2, color = "black")+
  xlab("Multiplication factor direct payments")+
  ylab("Initial area intensive agriculture (ha)")+
  theme(text = element_text(size=25))

#Save the plots
setwd(outputLoc)  #Set the workspace
pdf("SVMplot.pdf",width=20,height=10)
multiplot(IntPlot,ExtPlot, cols = 2)
dev.off()

#Save the plots
setwd(outputLoc)  #Set the workspace
jpeg("SVMplot.jpg",width=40,height=20,units = "cm",res=300)
multiplot(IntPlot,ExtPlot, cols = 2)
dev.off()

#Print the SVM-classification results for extensive agriculture
print(paste("The best cost hyperparameter for extensive agriculture was:",
ExtMod[[1]]$best.parameters$cost, sep = " "))
print(paste("The best gamma hyperparameter for extensive agriculture was:",
ExtMod[[1]]$best.parameters$gamma, sep = " "))
print(paste("The lowest classification error for extensive agriculture was:",
ExtMod[[1]]$best.performance, sep = " "))

#Print the SVM-classification results for intensive agriculture
print(paste("The best cost hyperparameter for extensive agriculture was:",
IntMod[[1]]$best.parameters$cost, sep = " "))
print(paste("The best gamma hyperparameter for extensive agriculture was:",
IntMod[[1]]$best.parameters$gamma, sep = " "))
print(paste("The lowest classification error for extensive agriculture was:",
IntMod[[1]]$best.performance, sep = " "))
```