We can compare each run to see how the yields have changed with each change we have made.

Function: multiple flow outputs - *ComPlot*

The above will create a chart for one of the flows. However, it is also useful to be able to visualize the flows in relation to one another. We therefore provide another function: comPlot.

This requires the same input as before, but this time we will introduce some of the other flow outputs, which should have been saved into separate folders as described earlier. Current maximum of 3 flows.

```
#Read in the example flows:
flow1 <- read.csv("./Run_1/saveFlow1", sep="", header = F) #Files have been
saved in separate folders
flow2 <- read.csv("./Run_2/saveFlow1", sep="", header = F)
flow3 <- read.csv("./Run_3/saveFlow1", sep="", header = F)
#flow4 <- read.csv("./saveFlow4", sep="")

#Run function with new flows
comPlot(flow.data.1 = flow1, flow.data.2 = flow2,
        flow.data.3 = flow3, 35)
```

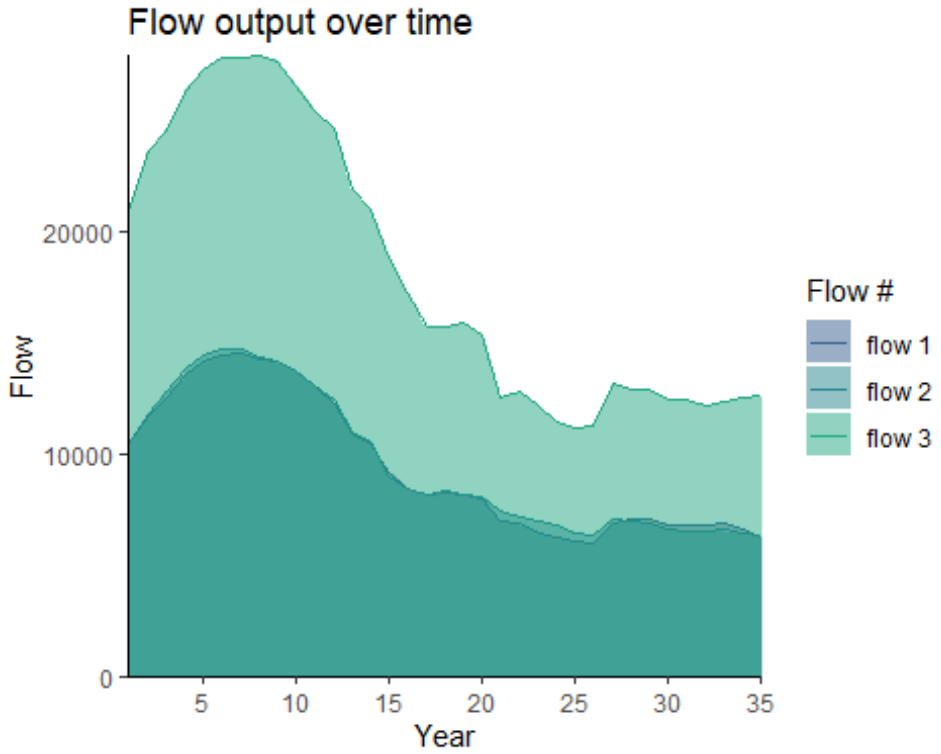

Figure 26. The amount of flow (habitat area) attained over time as a result of the three separate Habplan management schedule (flow1, flow2, and flow3). Each flow is represented by a different shade of blue.

```
#Read in the example flows:
flow1 <- read.csv("./Run_1/saveFlow2", sep="") #Files have been saved in
separate folders
flow2 <- read.csv("./Run_2/saveFlow2", sep="")
flow3 <- read.csv("./Run_3/saveFlow2", sep="")
#flow4 <- read.csv("./saveFlow4", sep="")

#Run function with new flows
comPlot(flow.data.1 = flow1, flow.data.2 = flow2,
        flow.data.3 = flow3, 35)
```

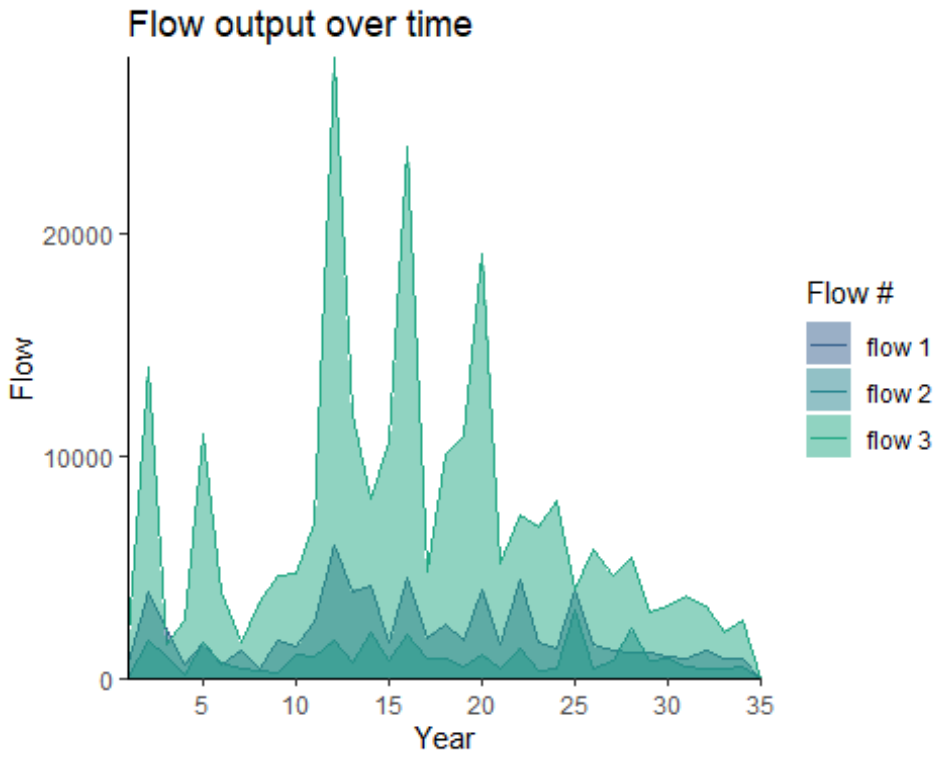

Figure 27. The amount of flow (harvested pine pulpwood) attained over time as a result of the three separate Habplan management schedule (flow1, flow2, and flow3). Each flow is represented by a different shade of blue.

By visualizing the flows in comparison to one another from each run, we can now clearly see the little differences found between the first and second run, but the changes we made to the third run have greatly increased our averge HSI and harvested pine pulpwood - ultimately providing more habitat for our species of interest, but maximizing yield and profits also.

Function: saving the top schedule to shapefile - *StandSched*

From this example, we are happy with the final run and will apply the recommended regime to each stand based on the output schedule from Habplan. Our next function pulls the information from the best schedule saved in the working directory (from Habplan), and attaches this to the stand shapefile.

As long as the file is still in the working directory (saveSched), then all we need to do is input the stand shapefile and run the function.

```
#Run function to add schedule to shapefile
standSched(site.shp)
```

If an error message comes up that states the following:

Error: unable to find an inherited method for function 'writeVector' for signature 'x = "data.frame", filename = "character"'

then run the next section in R to manually save the top schedule as a shapefile.

```
#Read in saved schedule from habplan run
sched <- read.csv("./saveSched")
#Change column headings so that it's easier to work with
colnames(sched) <- c("id", "StdID", "sched")
#Remove any blank spaces that may have been brought in from the import
sched$StdID <- gsub(" ", "", sched$StdID)
#Add the new column
new.shp <- merge(site.shp, sched, by = "StdID")
#Save to the working directory
writeVector(new.shp, "./Site_with_schedule.shp", filetype="ESRI Shapefile",
            overwrite=TRUE)
```

A new .shp file can now be found in the working directory titled Site_with_schedule.shp.

For the purposes of creating flow files, running Habplan, and comparing possible management schedules output from Habplan, you can now stop at this point of the vignette. However, next we provide two additional functions for creating custom HSI flow files, and including a blocksize flow subcomponent to the Habplan run.

Function: HSI calculation - *HSIcalc*

We are going to work through two examples of using basal area (BA) to calculate HSI. We have provided a function that will compute an HSI based on a user-specified formula, and combine this with our stand data. We provide examples of simple HSI formulas, but any formula can be used, so long as the inputs are present in the std.data dataset. We use a single independent variable (BA), but a function of multiple independent variables can also be used.

Our first example looks at a negative linear effect of BA on HSI. The equation will need to be provided in form of an R function.

```
#Create function to apply equation for HSI
#Define the independent variable (x), slope (m), and y-intercept (b) of the
linear equation
hsi.func <- function(x = std.data$BA, m = -0.02, b = 2) {
  return(m * x + b)
  }
```

We can now insert this equation into the *HSIcalc* function, which calculates HSI from the minimum to maximum values of all x values in the std.data dataset, and then adds a new data column to our stand data for HSI values.

```
#The function will create a new column with the HSI values
new.data <- HSIcalc(std.data, hsi.func)

#Summarise the HSI data
summary(new.data$HSI)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#> 0.05188 0.40419 0.61323 0.54688 0.73023 0.88072
#Plot data against BA to show relationship
ggplot(data = new.data) +
  geom_point(aes(x = BA, y = HSI, color = HSI),
             size = 3) +
  scale_color_viridis_c(option = "plasma") +
  theme_classic()
```

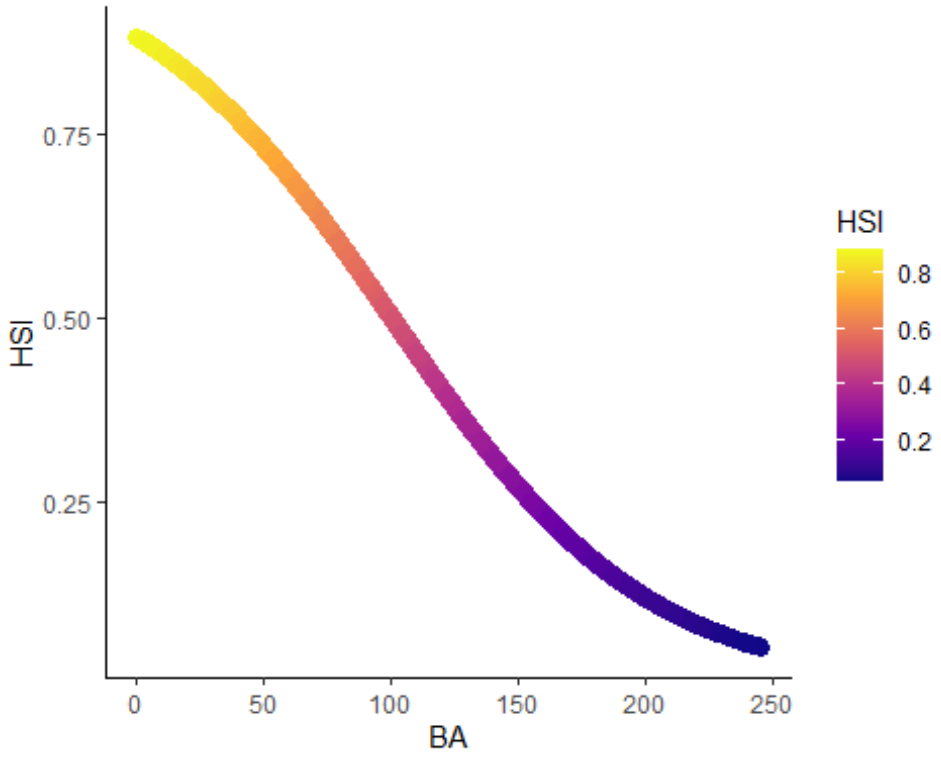

Figure 28. A negative linear effect of BA on HSI. Color represents change in HSI values.

Our second example looks at a quadratic relationship of BA on HSI. Again, the equation will need to be provided in form of an R function:

```
#Create function to apply equation for HSI
#Use a, b, and c to change shape of graph
hsi.func <- function(x = std.data$BA, a = -0.0006, b = 125, c = 3.5) {
  return(a * (x - b)^2 + c)
  }
```

We can now insert this equation into the *HSIcalc* function to add a column to our stand data for HSI values.

```
#The function will create a new column with the HSI values
new.data <- HSIcalc(std.data, hsi.func)

#Summarise the HSI data
summary(new.data$HSI)
#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#> 0.002816 0.394109 0.774463 0.661753 0.938009 0.970688
#Plot data against BA to show relationship
ggplot(data = new.data) +
  geom_point(aes(x = BA, y = HSI, color = HSI),
             size = 3) +
```

```
  scale_color_viridis_c(option = "plasma") +
  theme_classic()
```

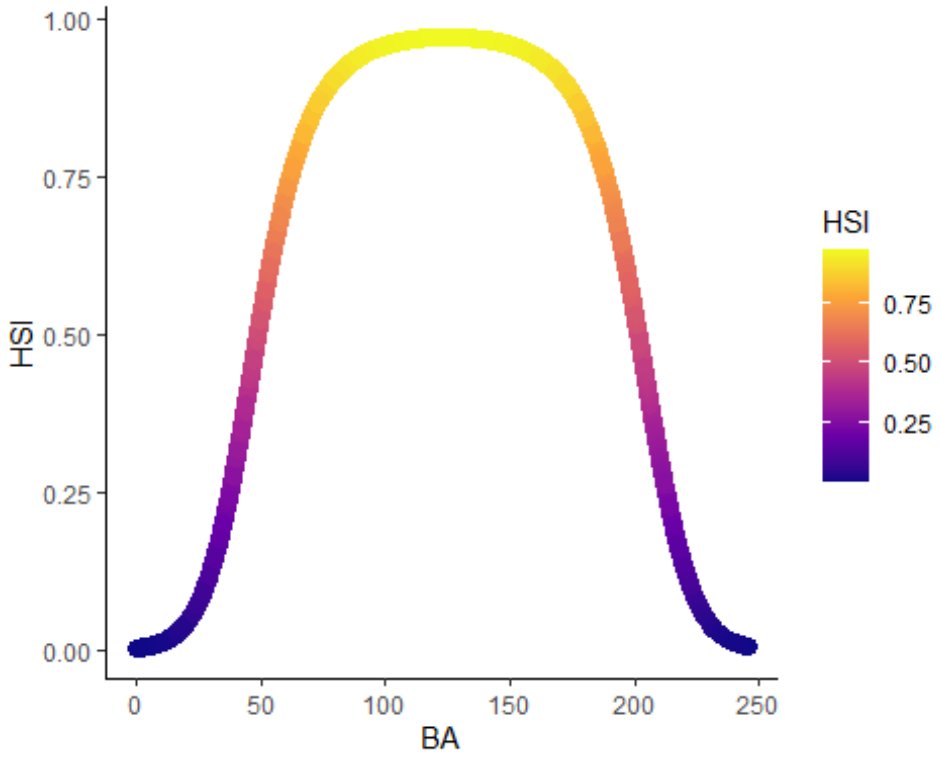

Figure 29. A quadratic effect of BA on HSI. Color represents change in HSI values.

Function: create block data - *HabBlock*

We will not be including the block data for the purpose of this vignette due to processing time. However, below we provide instructions for including a block size constraints to the Habplan run. Block size constraints are a sub-component of flow components. For a comprehensive guide to the purpose of block size constraints, please see section 9 of the Habplan user manual.

We can now run the function *HabBlock* from the HabplanR package. This takes the SpatVector shapefile, and stand data/info from earlier in the vignette and creates block data (defines which polygons are adjacent to each other). The block data will save directly to the working directory.

```
#HabBlock will create a block data file in the working directory
HabBlock(std.data = std.data, std.info = std.info, site.shp = site.shp,
         block.title = "block1")

#Since this block size component file is within the working directory
already,
#we just need to specify the file name below (b1.file), and input the
```

```
#specific parameters for the component.

#Info for f1 block size component (green up) ----
b1.file <- "block1.txt"
b1.notBlock <- "1-4" #which regimes do NOT contribute to block sizes
b1.greenUp <- "3" #3 year green up period
b1.min <- "1" #minimum allowable block size
b1.max <- "1000" #maximum allowable block size
b1.goal <- "1" #a value of 1.0 means all blocks must comply
b1.weight <- "1.0" #good practice to start this at 1
b1.title <- "block1.txt"
#Combine f1 components for writing
block1 <- c('<block title="BK1(1) Component">',
            paste0('<file value="', b1.file, '" />'),
            paste0('<notBlock value="', b1.notBlock, '" />'),
            paste0('<greenUp value="', b1.greenUp, '" />'),
            paste0('<min value="', b1.min, '" />'),
            paste0('<max value="', b1.max, '" />'),
            paste0('<goal value="', b1.goal, '" />'),
            paste0('<weight value="', b1.weight, '" />'),
            paste0('<title value="', b1.title, '" />'),
            '<bounds height="330" width="366" x="553" y="443" />',
            "</block>")
```

You can include as many block size components as there are flow components, which can be included as above but changing the block title and file to match the number of the flow component.

We can then include block size constraints to the project file using the *writeProj* function.

```
#Provide each of these flow component objects to the function and run
writeProj(f1.comp = f1.comp, block1 = block1, f2.comp= f2.comp)
```

Member-only version

Using the linear programming component of Habplan

We will now use the parameters from our third Habplan run, and use the linear programming component of Habplan to compare management solutions.

Function: creating biological type II data - *HabBio2*

To move forward with using the linear programming model of Habplan, we first need biological type II data. For more information, see section 11 (page 36) of the Habplan manual. We have developed a function embedded within *HabplanR* to create one of these files.

The biological type II data works by assigning weights to each combination of stand ID and regime. A higher weighting on a regime for a specific stand will prioritize that regime for

that stand. The below function assigns an equal weight to each regime. For the function to run, we need to provide the std.data, and std.info file we have used previously.

```
#Run function to create Bio2 file
HabBio2(std.data = std.data, std.info = std.info)
```

We now have a new .dat file saved in the working directory called Biol2_data. If any regimes need to be weighted greater than others for specific stands, then the Biol2_data file should be opened with a note application, and the corresponding number 1 (right hand column of data file), should be replaced with a number 2.

This file can now be treated as an additional flow component for the Habplan run; however, the parameters to set in the project file are different than other flow components. We will go through a new run incorporating the biological type II data below. We will use the same parameters as the third run exampled above.

```
#Info for f1 component ----
f1.file <- paste0(wd,"HSI.dat")
f1.bygone <- ""
f1.time0 <- "10000"
f1.goal0 <- "0.1"
f1.thlo <- "5000"
f1.thhi <- "10000"
f1.goalplus <- ".05"
f1.goalf <- "0.5"
f1.slope <- "0.0"
f1.weightf <- "1.0"
f1.weight0 <- "1.0"
f1.model <- "1,15000;20,5000;30,2000;"
f1.title <- "Breed.dat"
#Combine f1 components for writing
f1.comp <- c('<flow title="F1 Component">',
             paste0('<file value="', f1.file, '" />'),
             paste0('<bygone value="', f1.bygone, '" />'),
             paste0('<time0 value="', f1.time0, '" />'),
             paste0('<goal0 value="', f1.goal0, '" />'),
             paste0('<threshLo value="', f1.thlo, '" />'),
             paste0('<threshHi value="', f1.thhi, '" />'),
             paste0('<goalPlus value="', f1.goalplus, '" />'),
             paste0('<goalF value="', f1.goalf, '" />'),
             paste0('<slope value="', f1.slope, '" />'),
             paste0('<weightF value="', f1.weightf, '" />'),
             paste0('<weight0 value="', f1.weight0, '" />'),
             paste0('<model value="', f1.model, '" />'),
             paste0('<title value="', f1.title, '" />'),
             '<bounds height="330" width="366" x="553" y="443" />',
             "</flow>")
```

```
#Info for f2 component ----
f2.file <- paste0(wd, "Harv_P_Pulp_Tons.dat")
f2.bygone <- ""
f2.time0 <- "1000"
f2.goal0 <- "0.1"
f2.thlo <- "2000"
f2.thhi <- "100000"
f2.goalplus <- ".05"
f2.goalf <- "0.5"
f2.slope <- "0.0"
f2.weightf <- "1.0"
f2.weight0 <- "1.0"
f2.model.1 <- "1,100;20,2000;30,2000;"
f2.title <- "Harv_P_Pulp_Tons.dat"
#Combine f2 components for writing
f2.comp <- c('<flow title="F2 Component">',
             paste0('<file value="', f2.file, '" />'),
             paste0('<bygone value="', f2.bygone, '" />'),
             paste0('<time0 value="', f2.time0, '" />'),
             paste0('<goal0 value="', f2.goal0, '" />'),
             paste0('<threshLo value="', f2.thlo, '" />'),
             paste0('<threshHi value="', f2.thhi, '" />'),
             paste0('<goalPlus value="', f2.goalplus, '" />'),
             paste0('<goalF value="', f2.goalf, '" />'),
             paste0('<slope value="', f2.slope, '" />'),
             paste0('<weightF value="', f2.weightf, '" />'),
             paste0('<weight0 value="', f2.weight0, '" />'),
             #paste0('<model value="1,', f2.target, ';', f2.next.year, ',',
             #      f2.next.target, '" />'),
             paste0('<title value="', f2.title, '" />'),
             '<bounds height="331" width="368" x="1260" y="440" />',
             "</flow>")

#Info for bio2 component ----
bio2.1.file <- "Flows/Biol2_test.dat"
goal.val <- "0"
weight <- "1"
sum.val <- "1"
max.val <- "1"
bio2.title <- "Biol2_test.dat"
#Combine bio2 components for writing
bio2.1 <- c('<biol2 title="Bio2-1 Component">',
             paste0('<file value="', bio2.1.file, '" />'),
             paste0('<goalKind value="max" />'),
             paste0('<goal value="', goal.val, '" />'),
             paste0('<weight value="', weight, '" />'),
             paste0('<sum value="', sum.val, '" />'),
             paste0('<max value="', max.val, '" />'),
```

```
            paste0('<title value="', bio2.title, '" />'),
            "</biol2>")
```

The biological 2 component will be incorporated into a new project file for Habplan to read. To do this, we need to assign the component as a new argument in the *writeProj* function. Additionally, we need to set a new configuration. The new configuration now includes a 1 in the corresponding biological 2 component position.

```
#Need a new configuration to include bio2 component
config <- "2,0,0,0,0,0,1,0" #now includes an additional 1 in second to last
position

#Provide each of these flow component objects to the function and run
writeProj(f1.comp = f1.comp, f2.comp= f2.comp, bio2.1 = bio2.1)
```

We now have the configuration and project parameters ready for our linear programming run. However, we need an additional file for the linear programming solver.

Function: creating an MPS file for lp_solve - *lpMPS*

Habplan uses an additional free software for running the linear programming component, lp_solve, which works by reading MPS files. These files are difficult to interpret, but we provide a function below to create one of these files in preparation for the Habplan run.

For the function to work, we need to provide our std.data as many of the functions before, and the nyear (35). We have also provided the option to change the filename of the output MPS file via the filename argument. The two final arguments are th.hi (upper threshold), and th.lo (lower threshold). Both of these arguments are bounded between 0-100, and now represent a percentage of positive or negative deviation, respectively. In our example below, we are allowing a positive deviation of 50% for our model targets, and a negative deviation of only 10%.

```
#Create an MPS file for linear programming
lpMPS(std.data = std.data, nyear = 35, filename = "lpMPS",
      th.hi = 50, th.lo = 10)
```

Let's open habplan and run through the linear programming example.

IMPORTANT - MUST RENAME OR MOVE SAVEFLOW1 AND SAVESCHED FROM WORKING DIRECTORY IF YOU DO NOT WANT THE FILES TO BE DELETED

```
#Open Habplan (if run from here, R functionality will cease until Habplan is
closed)
shell("h", wait=TRUE)
```