ResistanceGA

ResistanceGA

An R Package for Optimizing Resistance Surfaces using Genetic Algorithms

Background

With this vignette/tutorial, hopefully you'll get an idea of what each of the functions in this package can do, as well present an example (using simulated data) of how you can optimize resistance surfaces in isolation as well as simultaneously to create novel resistance surfaces. This 'package' (I use that term very loosely) has largely been developed from functions I wrote to conduct different landscape genetic analyses. See Peterman et al. (2014) for the original conception of optimizing resistance surfaces using optimization functions. This approach was limited to optimization of continuous surfaces in isolation. Since that paper, I've further developed the optimization method to utilize genetic algorithms, implemented using the `ga` function from the GA package in R. By moving to genetic algorithms, much more complex parameter space can be effectively searched, which allows for the optimization of categorical resistance surfaces, as well as optimization of multiple resistance surfaces simultaneously.

This package fills a void in the landscape genetics toolbox. There are various methods proposed for determining resistance values (reviewed by Spear et al., 2010). Previously utilized methods generally searched a limited parameter space and/or relied on expert opinion. Graves et al. (2013) utilized optimization functions and interindividual genetic distances to determine resistance values, but found that the data generating values were rarely recoverable. I have not assessed the ability of functions/methods utilized in this package to optimize resistance surfaces as in Graves et al. (2013), but do note that very different methods of scaling, transforming, and combining resistance surfaces are utilized in `ResistanceGA`.

A few words of caution. I have made every effort to run and test each function with simulated data, but I make no guarantees concerning function performance and stability. Data formatting can be a challenge, and I have tried to simplify the process as much as possible. If you choose to optimize using CIRCUITSCAPE,

Please make sure you carefully read through the CIRCUITSCAPE documentation, as well as other relevant papers by Brad McRae to get a more complete understanding of resistance modeling and circuit theory. If errors occur, start by making sure that you are providing function inputs in the correct format. If a function does not work, there likely will not be a helpful error message to help you troubleshoot. Depending on interest and use, these are features that may be added in the future. Lastly, this is not a fast process. Even with the 50x50 pixel simulated landscapes used in this tutorial, each (CIRCUITSCAPE) optimization iteration takes 0.75–1.00 seconds to complete (Intel i7 3.4 GHz processor, 24 GB RAM). Optimizing using least cost paths is ~3x faster than with CIRCUITSCAPE. To further reduce optimization time when using least cost paths, the optimization can be run in parallel. It appears that under most circumstances the resolution of the landscape can be reduced without loss of information (McRae et al., 2008). If you want to use the optimization procedures in `ResistanceGA`, but are working with a large landscape, I might suggest reducing the resolution first. Depending upon whether you are optimizing a single surface or multiple surfaces simultaneously, the genetic algorithms typically run for 50–300 generations. `ga` settings will vary for each run, but there will typically be 50–150 offspring (i.e. different parameter value realizations) per generation. This means that 2500–4.5×10^4 iterations will be needed to complete the optimization. This can be a **LONG** process! If you encounter issues while executing any of these functions, or would like some other functionality incorporated, please let me know (bill.peterman@gmail.com). I am eager to make this as accessible, functional, and as useful as possible, so any and all feedback is appreciated.

References

- Graves, T. A., P. Beier, and J. A. Royle. 2013. Current approaches using genetic distances produce poor estimates of landscape resistance to interindividual dispersal. Molecular Ecology 22:3888–3903.
- McRae, B. H., B. G. Dickson, T. H. Keitt, and V. B. Shah. 2008. Using circuit theory to model connectivity in ecology, evolution, and conservation. Ecology 89:2712–2724.
- Peterman, W. E., G. M. Connette, R. D. Semlitsch, and L. S. Eggert. in press. Ecological resistance surfaces predict fine scale genetic differentiation in a terrestrial woodland salamander. Molecular Ecology 23:2402–2413.

- Spear, S. F., N. Balkenhol, M. J. Fortin, B. H. McRae, and K. Scribner. 2010. Use of resistance surfaces for landscape genetic studies: considerations for parameterization and analysis. Molecular Ecology 19:3576–3591.

Setup

Install necessary software and packages

If you want to optimize using CIRCUITSCAPE, this package requires that you have CIRCUITSCAPE v4.0 or higher installed on your Windows machine. At this point in time, `ResistanceGA` has only been tested on *WINDOWS* operating systems. Ability to run on Mac or Linux systems is in testing. You will also need to have R >= v3.0 installed. I would highly recommend installing R studio when working with R.

R Packages
This package consists of several wrapper functions for implementing functions from other packages, and these will all be imported when `ResistanceGA` is installed.

Demonstrations

Continuous surface transformations

First, install `ResistanceGA` from GitHub. This will require the `devtools` package

```
# Install 'devtools' package, if needed
if(!("devtools" %in% list.files(.libPaths()))) {
    install.packages("devtools", repo = "http://cran.rstudio.com", dep = TRUE)
}

library(devtools) # Loads devtools

install_github("wpeterman/ResistanceGA") # Download package
```

Load `ResistancaGA` and clear your workspace.

```
library(ResistanceGA)
rm(list = ls())
```

There are 8 different transformations that can be applied to continuous surfaces. Since the publication of Peterman et al. (2014), I have added Reverse Ricker and Inverse-Reverse Ricker transformation to better cover parameter space. I still think that there are more flexible ways to optimize surfaces, and I'm

3

continuing to develop these as I have time.
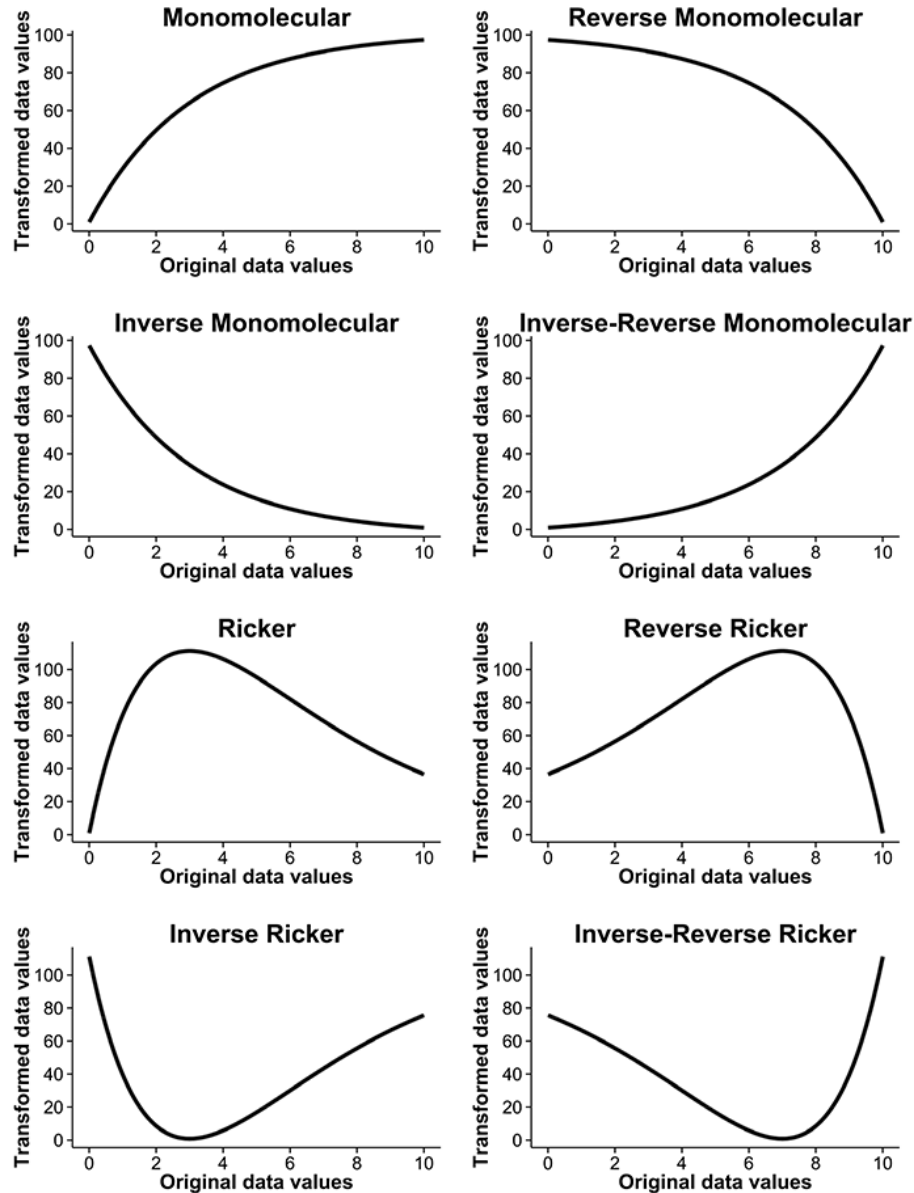
All of these figures were made with the `Plot.trans` function. This function returns a ggplot object, which allows you to manipulate some aspects of the plot, as well as determine the resistance value at different levels of your original surface.

```
Ricker.plot <- Plot.trans(PARM=c(1.5, 200),Resistance=c(1,10),transformation="Ricker")
```
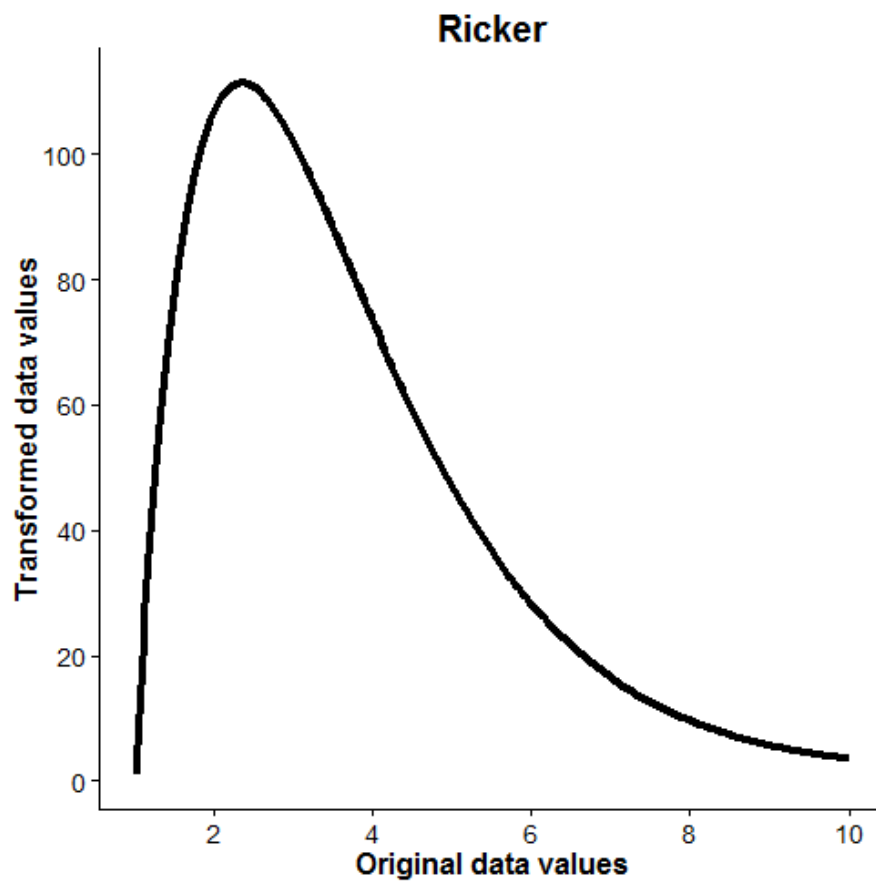
Ricker

```
# Change title of plot
Ricker.plot$labels$title<-"Ricker Tansformation"
Ricker.plot
```

Ricker Tansformation
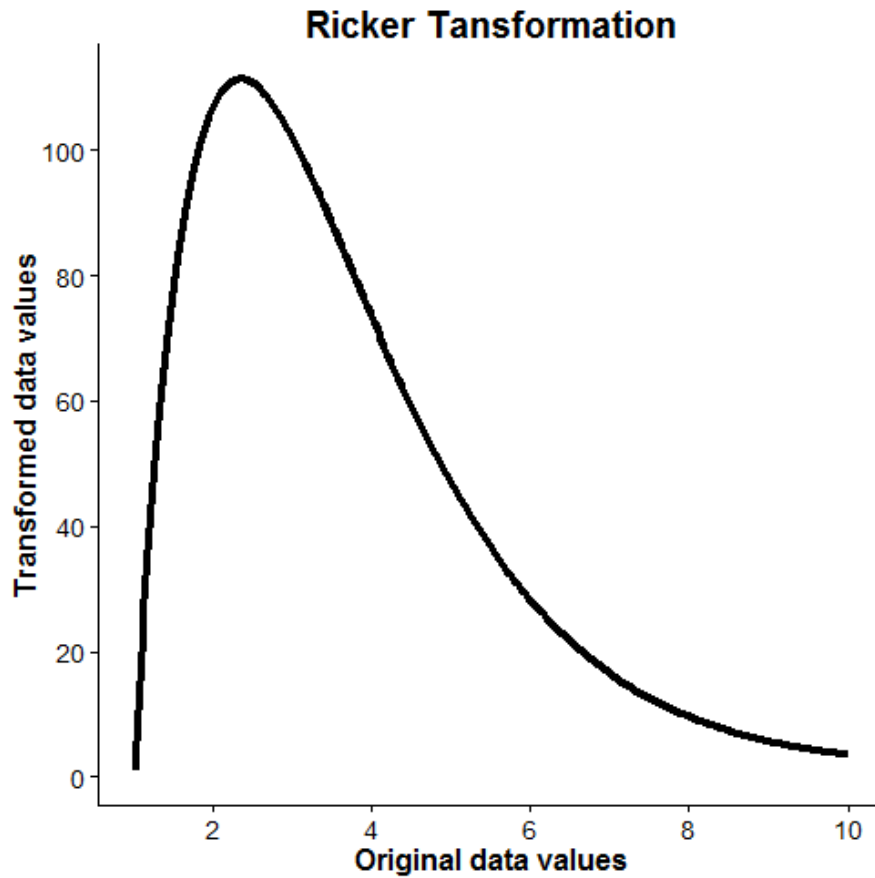

```
# Find original data value that now has maximum resistance
Ricker.plot$data$original[which(Ricker.plot$data$transformed==max(Ricker.plot$data$transform
```

```
## [1] 2.356784
```

Example Function Use

Single surface optimization (with simulated data)

Make a directory to write ASCII files, CIRCUITSCAPE batch files, and results.
It is critical that there are **NO SPACES** in the specified directory as this will
cause functions that interact with CIRCUITSCAPE to fail.

```
if("ResistanceGA_Examples"%in%dir("C:/")==FALSE)
```

```
  dir.create(file.path("C:/", "ResistanceGA_Examples"))

# Create a subdirectory for the first example
dir.create(file.path("C:/ResistanceGA_Examples/","SingleSurface"))

write.dir <- "C:/ResistanceGA_Examples/SingleSurface/"      # Directory to write .asc files

# Give path to CIRCUITSCAPE .exe file
# Default = '"C:/Program Files/Circuitscape/cs_run.exe"'
CS.program <- paste('"C:/Program Files/Circuitscape/cs_run.exe"')
```

Load resistance surfaces and export as *.asc* file for use with CIRCUITSCAPE.
These surfaces were made using the `RandomFields` package

```
data(resistance_surfaces)
continuous <- resistance_surfaces[[2]]
writeRaster(continuous,filename=paste0(write.dir,"cont.asc"),overwrite=TRUE)
```

Load the example sample location data and export as *.txt* file. This is formatted
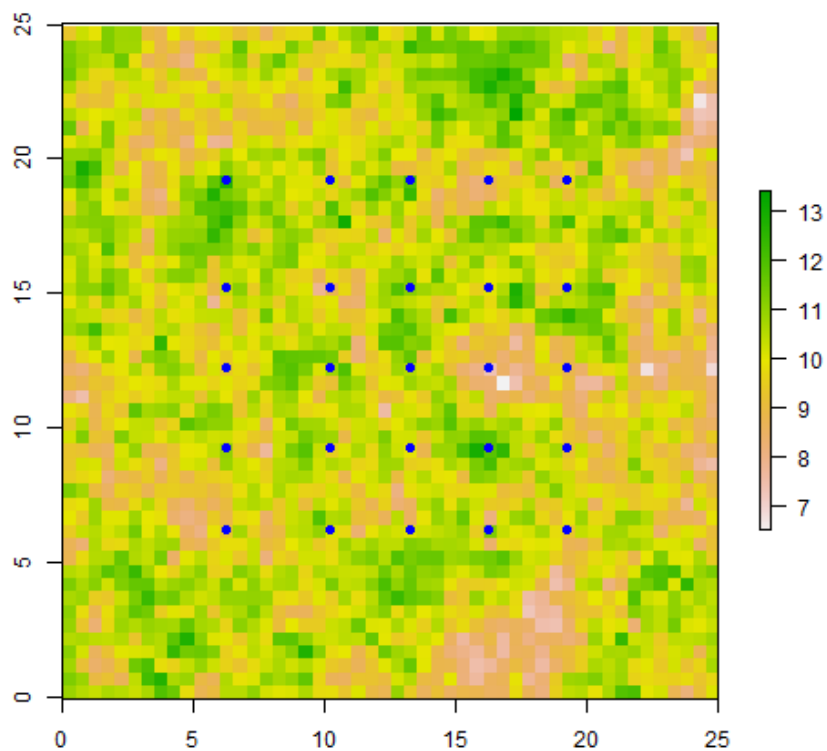for input into CIRCUITSCAPE

```
data(samples)
write.table(samples,file=paste0(write.dir,"samples.txt"),sep="\t",col.names=F,row.names=F)

# Create a spatial points object for plotting
sample.locales <- SpatialPoints(samples[,c(2,3)])
```

Plot surface and overlay the sample points

```
plot(continuous)
plot(sample.locales, pch=16, col="blue", add=TRUE) # Add points
```

Prepare data for optimization

Run the `GA.prep` and `CS.prep` functions

```
# Set the random number seed to reproduce the results presented
GA.inputs <- GA.prep(ASCII.dir=write.dir,
                max.cat=500,
                max.cont=500,
                seed = 555)

CS.inputs <- CS.prep(n.POPS=length(sample.locales),
                CS_Point.File=paste0(write.dir,"samples.txt"),
                CS.program=CS.program)
```

Note that **response** was not defined in `CS.prep` because it has not been made

8

yet.

Transform raw continuous surface using the `Resistance.tran` function to apply one of the eight transformations, and then view the transformation using `Plot.trans`. Note that `Plot.trans` returns a `ggplot2` object as well as the plot. Therefore you can manipulate and modify the plot as desired.

```
r.tran <- Resistance.tran(transformation="Monomolecular", shape=2, max=275, r=continuous)

plot.t <- Plot.trans(PARM=c(2,275), Resistance=continuous, transformation="Monomolecular")
```

Monomolecular
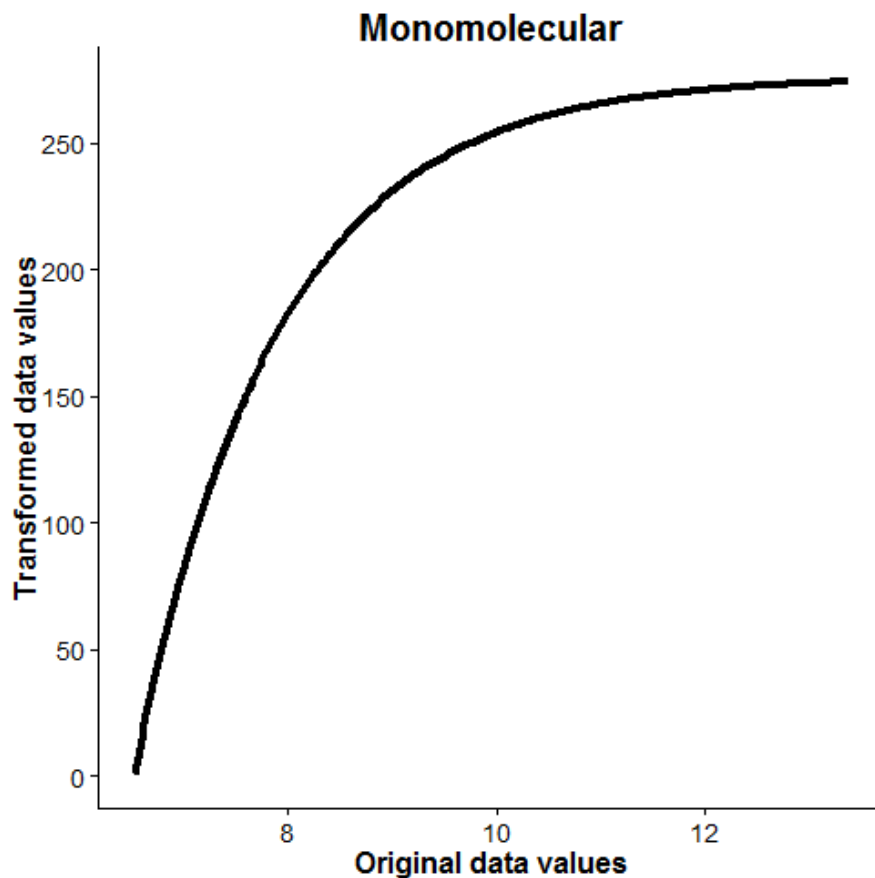
Run the transformed resistance surface through CIRCUITSCAPE to get effective resistance between each pair of points. `Run.CS` returns the lower half of the pairwise resistance matrix for use with the optimization prep functions. This will be our response that we optimize on.

```
# Create the true resistance/response surface
```

```
CS.response <- Run_CS(CS.inputs=CS.inputs,GA.inputs=GA.inputs, r=r.tran)
```

Rerun `CS.prep` including the newly created `CS.response`

```
CS.inputs <- CS.prep(n.POPS=length(sample.locales),
                     response=CS.response,
                     CS_Point.File=paste0(write.dir,"samples.txt"),
                     CS.program=CS.program)
```

Run the Single surface optimization function (`SS_optim`). Running this example with the default settings took 147 iterations and ~72 minutes to complete on a computer with an Intel i7 3.4 GHz processor. The data generating values have been precisely recovered.

```
SS_RESULTS <- SS_optim(CS.inputs=CS.inputs,
                       GA.inputs=GA.inputs)
```

View the results and compare with truth

```
SS_table <- data.frame(c("Monomolecular", 2.0, 275),
                       t(SS_RESULTS$ContinuousResults[c(3:5)]))
colnames(SS_table) <- c("Truth", "Optimized")

SS_table
                 Truth      Optimized
Equation Monomolecular Monomolecular
shape                2      1.999999
max                275      274.9982
```

If you get the error:

```
Error in initializePtr() :
  function 'dataptr' not provided by package 'Rcpp'
```

Reinstall the `Rcpp` package and execute the `SS_optim` function again.

After executing the function, the console will be updated to report the time to complete each iteration as well as AICc of each iteration. If you do not wish to view updates at each iteration of the optimization, set `quiet = TRUE` in `GAp.prep`

What the `SS_optim` function does:

- Read each .asc file that is in the specified ASCII.dir and determines whether it is a categorical or continuous surface. A surface is considered categorical if it contains 15 or fewer unique values.

- Optimize each resistance surface

  - Categorical surfaces: Each optimized value represents the resistance of that category to current flow

  - Continuous surfaces: Each continuous surface is first rescaled so that values range from 0–10 (note that relative spacing is preserved during rescaling). The genetic algorithm then tests different combinations of the transformation equation, shape parameter value, and maximum resistance value. When the genetic algorithm has finished optimization, the optimized parameters can be passed to a second optimization function that uses `nlm` to fine-tune the shape and maximum value parameters (`nlm = TRUE`). However, this approach may lead to overfitting and the default is `nlm = FALSE`.

- Several summary outputs are generated

  - In the 'Results' directory (located in the directory with the .asc files), a final optimized resistance .asc file has been made, along with the CIRCUITSCAPE results (.out files).

  - Summary tables for continuous surfaces (ContinuousResults.csv), categorical surfaces (CategoricalResults.csv), and the AICc of all surfaces (All_Results_AICc.csv)

  - MLPE_coeff_Table.csv contains the model coefficients from the fitted mixed effects model for each surface

  - In the 'Plots' directory there is a 4-panel figure with different model diagnostic plots generated from the fitted mixed effects model of each optimized resistance surface. If a continuous surface was optimized, there is also a plot showing the relationship of the transformed resistance surface with the original data.

- The returned object is a named list containing the tables described above.

To view the AICc response surface for the Monomolecular optimization of this surface, you can run `Grid.Search`. This function is only relevant for single continuous surfaces.

```
Grid.Results <- Grid.Search(shape=seq(1,4,by=0.1),
                            max=seq(50,500,by=75),
                            transformation="Monomolecular",
                            Resistance=continuous,
                            CS.inputs,
                            GA.inputs=GA.inputs)
```
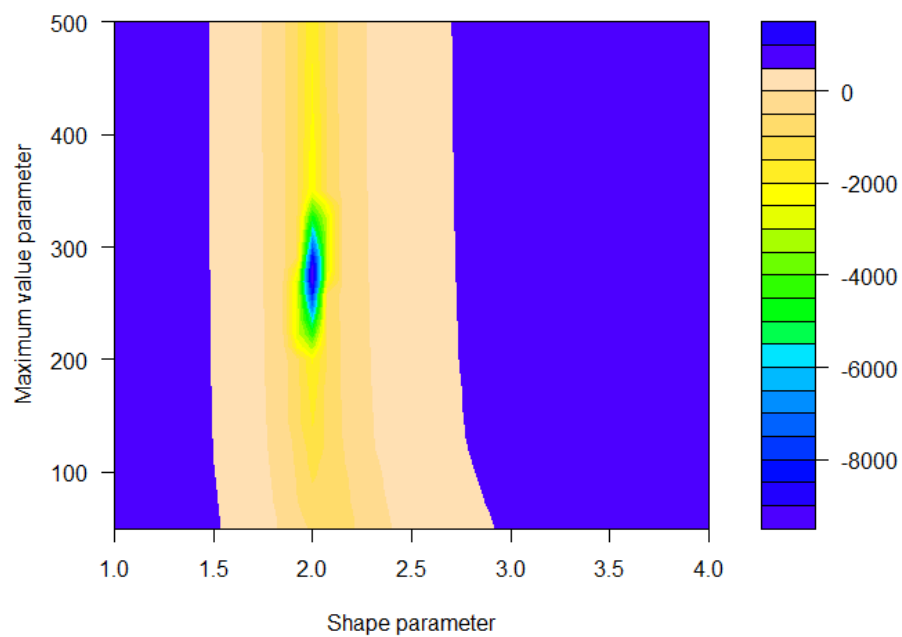

You can change the color scheme and color breaks by manually recreating the response surface from the generated data [default = topo.colors(20)]

```
filled.contour(Grid.Results$Plot.data,
               col=rainbow(30),
               xlab="Shape parameter",
               ylab="Maximum value parameter")
```
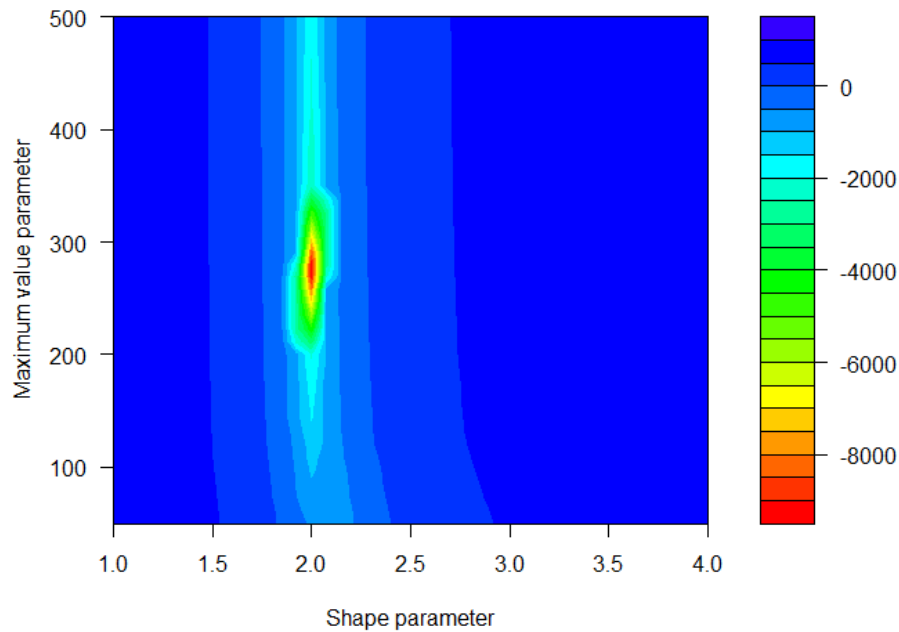
Note that actual response surfaces tend to be slightly flatter, and the maximum value for a single surface is more difficult to identify precisely. If you were to add some random noise to the CS.response, the single surface optimization generally would do a good job of recovering the transformation and shape parameters, but the true maximum value may remain elusive. Occasionally the algorithm will get 'stuck' trying to optimize on an incorrect transformation. If this happens, rerun the optimization. Of course, you may not know that a surface wasn't correctly optimized when using real data. For this reason, it is good practice to run all optimizations at least twice to confirm parameter estimates.

Minimum code for running ResistanceGA

The example above outlines how the functions can be used while simulating and generating data with known parameter values. When analyzing your own data, it is not necessary to to use the `r.tran` or `Run_CS` functions. You should only have to use the following functions (although you may want to change settings from defaults).

```
GA.inputs <- GA.prep(ASCII.dir=write.dir)

CS.inputs <- CS.prep(n.POPS=length(sample.locales),
```

```
                        response=CS.response,
                        CS_Point.File=paste0(write.dir,"samples.txt"),
                        CS.program=CS.program)


SS_RESULTS <- SS_optim(CS.inputs=CS.inputs,
                        GA.inputs=GA.inputs)
```

Optimzation using least cost paths

The above optimization can also be done using cost distances calculated in `gdistance`. This approach uses least cost paths between points, so it is a simpler representation of connectivity. However, optimization using `gdistance` is ~3x faster than optimization with CIRCUITSCAPE. This optimization took 86 iterations and 7 minutes to complete when run in parallel on 4 cores (`parallel = 4` in `GA.prep`)

```
# Import data
data(resistance_surfaces)
continuous <- resistance_surfaces[[2]]

data(samples)
sample.locales <- SpatialPoints(samples[,c(2,3)])

# Set the random number seed to reproduce the results presented
# Run in parallel on 4 cores
GA.inputs <- GA.prep(ASCII.dir=continuous,
                     Results.dir=write.dir,
                     max.cat=500,
                     max.cont=500,
                     seed = 555,
                     parallel = 4)

gdist.inputs <- gdist.prep(n.POPS=length(sample.locales),
                           samples=sample.locales)

# Transform resistance surface
r.tran <- Resistance.tran(transformation="Monomolecular", shape=2, max=275, r=continuous)

# Create the true resistance/response surface
gdist.response <- Run_gdistance(gdist.inputs=gdist.inputs, r=r.tran)

# Rerun `gdist.prep` to include response
gdist.inputs <- gdist.prep(n.POPS=length(sample.locales),
                           response=lower(as.matrix(gdist.response)),
```

```
                              samples=sample.locales)

# Run optimization
SS_RESULTS.gdist <- SS_optim(gdist.inputs=gdist.inputs,
                             GA.inputs=GA.inputs)

# Grid search of response surface
Grid.Results.gdist <- Grid.Search(shape=seq(1,4,by=0.1),
                                  max=seq(50,500,by=75),
                                  transformation="Monomolecular",
                                  Resistance=continuous,
                                  gdist.inputs=gdist.inputs,
                                  GA.inputs=GA.inputs)
```

Simultaneous optimization of multiple surfaces

First, make a new directory to write ASCII files, CIRCUITSCAPE batch files, and results.

```
if("ResistanceGA_Examples"%in%dir("C:/")==FALSE)
  dir.create(file.path("C:/", "ResistanceGA_Examples"))

# Create a subdirectory for the second example
dir.create(file.path("C:/ResistanceGA_Examples/","MultipleSurfaces"))

# Directory to write .asc files and results
write.dir <- "C:/ResistanceGA_Examples/MultipleSurfaces/"
```

Extract other resistance surfaces from the 'resistance_surfaces' raster stack

```
data(resistance_surfaces)
data(samples)
sample.locales <- SpatialPoints(samples[ ,c(2,3)])
```
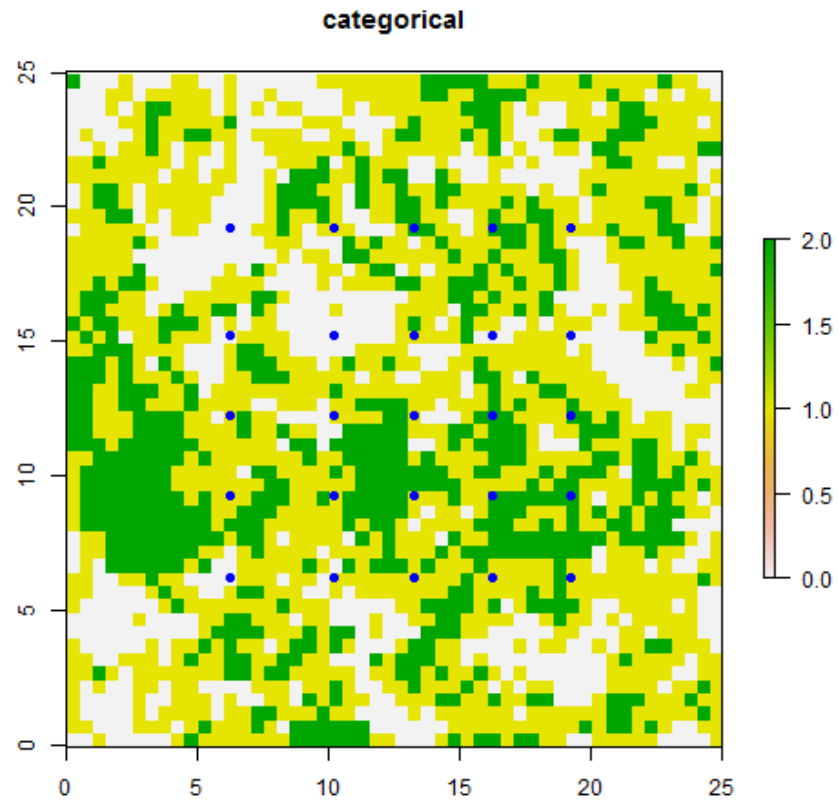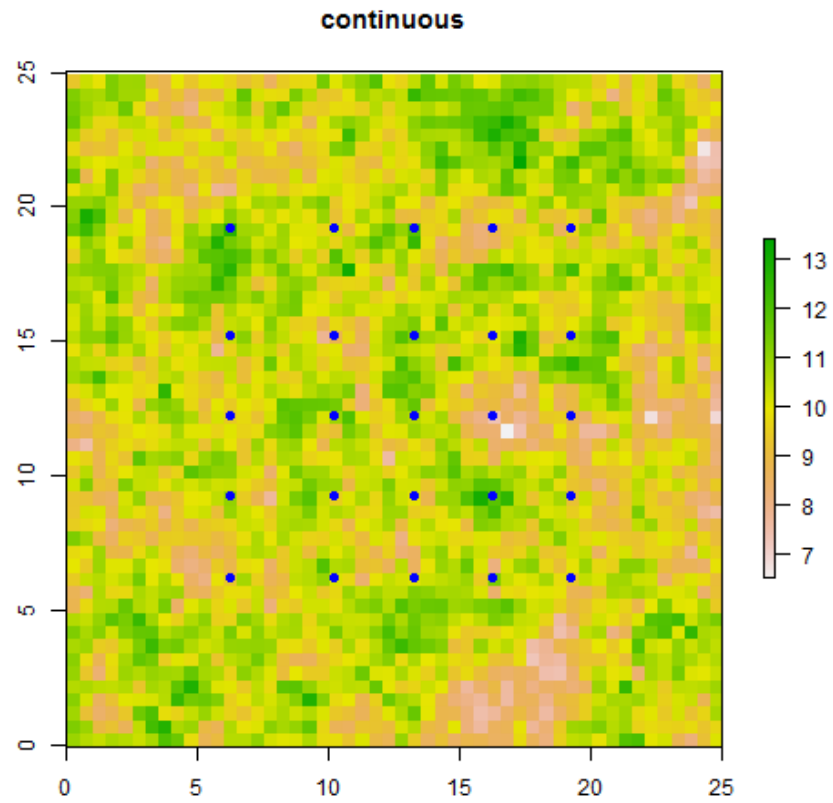
Visualize each surface:

```
plot(resistance_surfaces[[1]],main = resistance_surfaces[[1]]@data@names)
plot(sample.locales, pch=16, col="blue", add=TRUE)
```
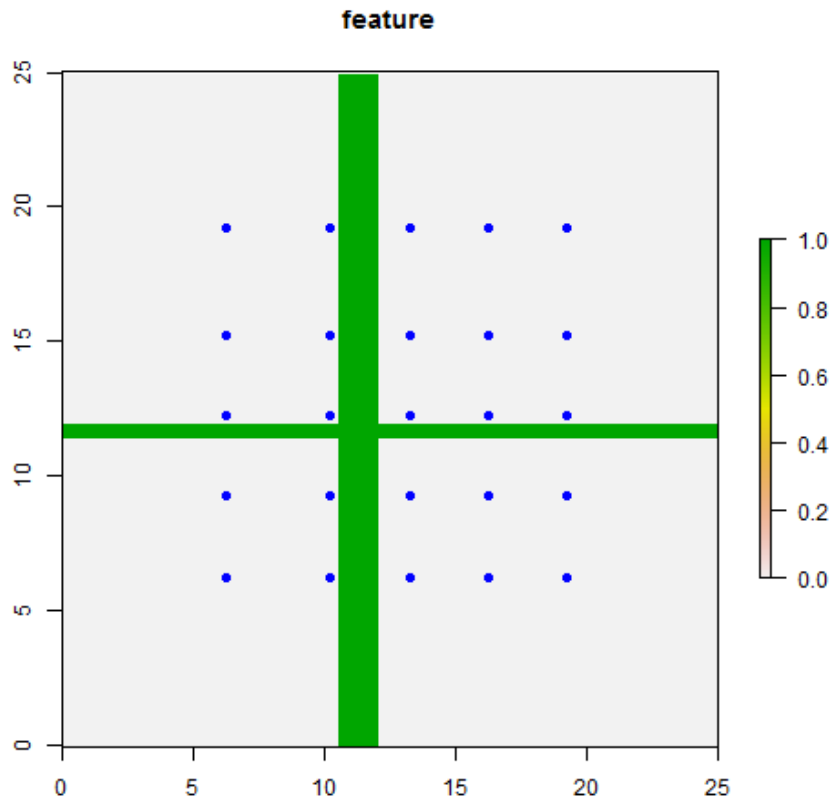
categorical

```
plot(resistance_surfaces[[2]],main = resistance_surfaces[[2]]@data@names)
plot(sample.locales, pch=16, col="blue", add=TRUE)
```

continuous

```
plot(resistance_surfaces[[3]],main = resistance_surfaces[[3]]@data@names)
plot(sample.locales, pch=16, col="blue", add=TRUE)
```

feature

Write all three surfaces to a directory for use with CIRCUITSCAPE and run the
`GA.prep` function (needed to combine surfaces). Also write the sample location
file to the "MultipleSurfaces" directory.

```
writeRaster(categorical,filename=paste0(write.dir,"cat.asc"),overwrite=TRUE)
writeRaster(continuous,filename=paste0(write.dir,"cont.asc"),overwrite=TRUE)
writeRaster(feature,filename=paste0(write.dir,"feature.asc"),overwrite=TRUE)

write.table(samples,file=paste0(write.dir,"samples.txt"),sep="\t",col.names=F,row.names=F)
```

Run `GA.prep`

```
GA.inputs <- GA.prep(ASCII.dir=write.dir,
                     max.cat=500,
                     max.cont=500,
                     seed = 555,
```

```
                    quiet = TRUE)
```

Transform, reclassify, and combine the three resistance surfaces together. Use an
"Inverse-Reverse Monomolecular" transformation of the continuous surface. Visualize this transformation using `Plot.trans`. The first value of `PARM` refers to the
shape parameter, and the second value refers to the maximum value parameter.
Look in the help file for `Plot.trans` for transformation names/numbers.

```
plot.t <- Plot.trans(PARM=c(3.5,400),Resistance=continuous,transformation="Reverse Ricker")
```
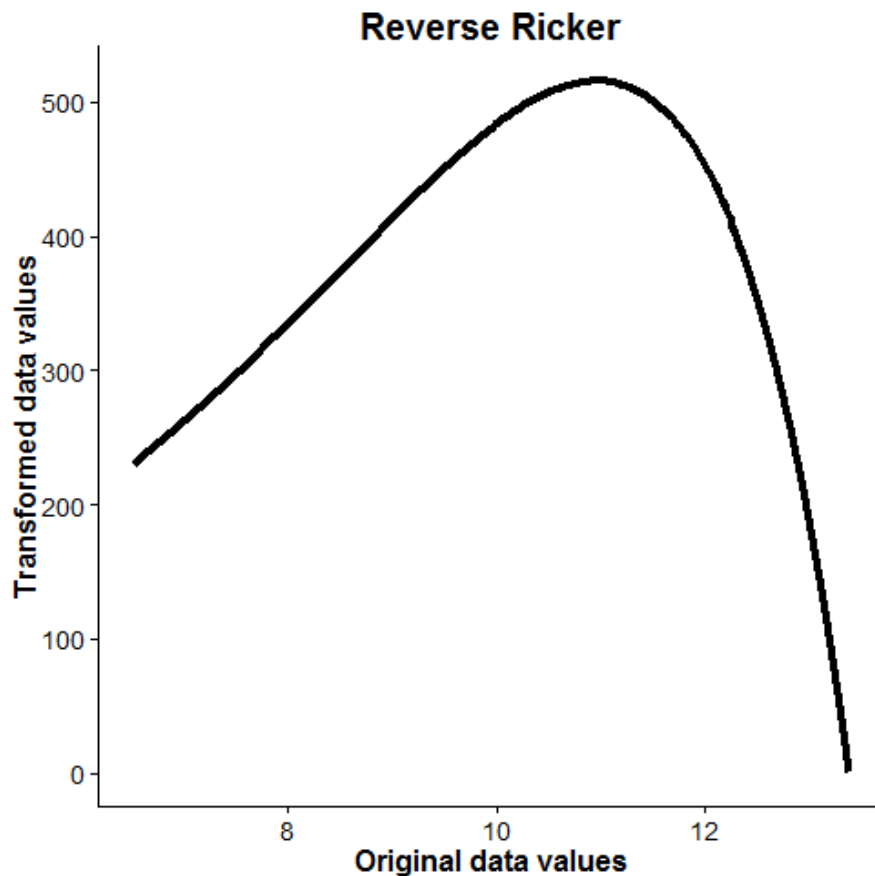

Combine raster surfaces together using `Combine_Surfaces`. Note that the .asc
files are read in alphabetically. You can check the order of surfaces by inspecting
`GA.inputs$layer.names`. First, define the parameters that will be passed to
`Combine_Surfaces`.

```
PARM <- c(1, 250, 75, 6, 3.5, 150, 1, 350)
```

```
# PARM<- c(1,   # First feature of categorical
#          250, # Second feature of categorical
#          75,  # Third feature of categorical
#          6,   # Transformation equation for continuous surface
#          3.5,  # Shape parameter
#          150, # Scale parameter
#          1,   # First feature of feature surface
#          350) # Second feature of feature surface

# Combine resistance surfaces
Resist <- Combine_Surfaces(PARM=PARM,
                           CS.inputs=CS.inputs,
                           GA.inputs=GA.inputs,
                           out=NULL,
                           rescale = TRUE)

# View combined surface
plot(Resist,  main = "scaled composite resistance")
```

scaled composite resistance
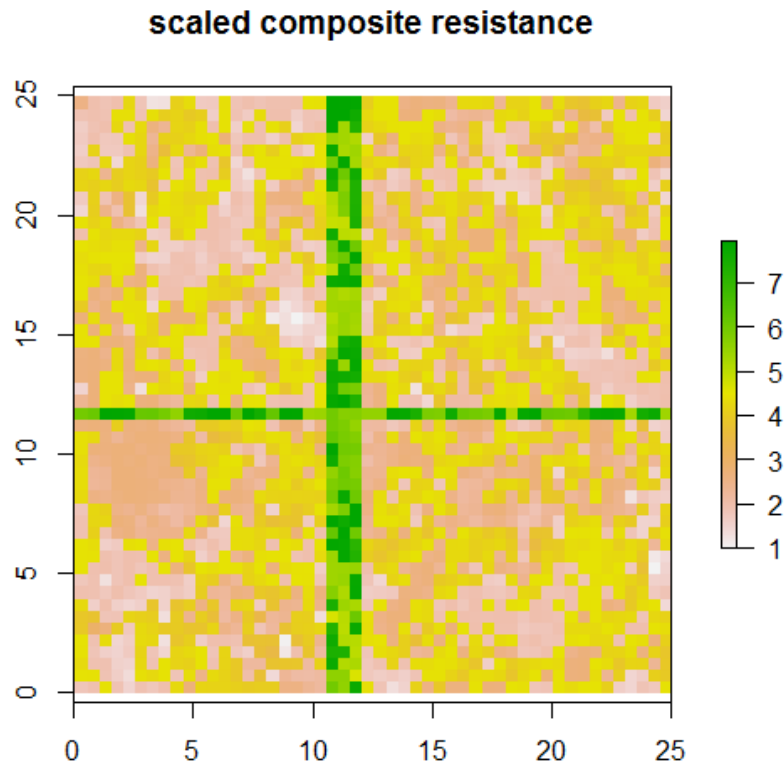
Generate new CS response surface by using `Run_CS` and run `CS.prep` to add response

```
# Create the true resistance/response surface
CS.response <- Run_CS(CS.inputs=CS.inputs, GA.inputs=GA.inputs, r=Resist)

CS.inputs<-CS.prep(n.POPS=length(sample.locales),
                   response=CS.response,
                   CS_Point.File=paste0(write.dir,"samples.txt"),
                   CS.program=CS.program)
```

Run `MS_optim`. Running this multisurface example with the default settings took 257 iterations and ~5 hours to complete on a computer with an Intel i7 3.4 GHz processor.

```
Multi.Surface_optim <- MS_optim(CS.inputs=CS.inputs,
                                GA.inputs=GA.inputs)
```

After executing the function, the console will be updated to report the time to complete each iteration as well as AICc of each iteration.

What the `MS_optim` function does:

- Read all .asc files that are in the specified ASCII.dir, makes a raster stack, and determines whether each is a categorical or continuous surface. A surface is considered categorical if it contains 15 or fewer unique values.

- Transformation and resistance values are chosen for each surface, all surfaces are added together, and AICc from the mixed effects model is calculated.

- Several summary outputs are generated

    - In the 'Results' directory (located in the directory with the .asc files), a final optimized resistance .asc file has been made (the name is a combination of the layers optimized, separated by "."), along with the CIRCUITSCAPE results (.out files).

    - 'Multisurface_Optim_Summary.txt' provides a text summary of the model parameters and results of the multisurface optimization

    - A .csv file with the fitted MLPE model coefficients
    - In the 'Plots' directory there is a 4-panel figure with different model diagnostic plots generated from the fitted mixed effects model of each optimized resistance surface.

- The `GA` object from the optimization is returned and can be further explored.

The multisurface optimization procedure has done a pretty good job of recovering the relative data generating values. You'll notice that we have not exactly recovered the values, but that the relative relationship among surfaces is preserved (see below).

```
Summary.table <- data.frame(PARM,round(t(Multi.Surface_optim@solution),2))
colnames(Summary.table)<-c("Truth", "Optimized")
row.names(Summary.table)<-c("Category1",
                            "Category2",
                            "Category3",
                            "Transformation",
                            "Shape",
                            "Max",
```

```
                  "Feature1",
                  "Feature2")


Summary.table
               Truth Optimized
Category1        1.0      1.00
Category2      250.0    304.31
Category3       75.0     91.15
Transformation   6.0      6.28
Shape            3.5      3.50
Max            150.0    183.12
Feature1         1.0      1.00
Feature2       350.0    426.14
```

The optimized values are ~1.22 times greater than the data generating values. However, if we rescale both the true and optimized resistance surfaces to have a minimum value of 1, we see that the surfaces are identical. The values for the 3-class categorical surface are the first three values listed, the continuous surface values $= 4$–6 , and the feature surface values $= 7$–8. Note that the first value for continuous surfaces identifies the transformation used (the fourth value, here), and is always rounded down ($6 =$ Reverse Ricker). Visualize and test the equivalence of simulated and optimized resistance surfaces:

```
# Make combined, optimized resistance surface.
optim.resist <- Combine_Surfaces(PARM=Multi.Surface_optim@solution, CS.inputs =  CS.inputs,(
ms.stack <- stack(Resist, optim.resist)
names(ms.stack) <- c("Truth", "Optimized")
plot(ms.stack)

# Correlation between the two surfaces
pairs(ms.stack)
```
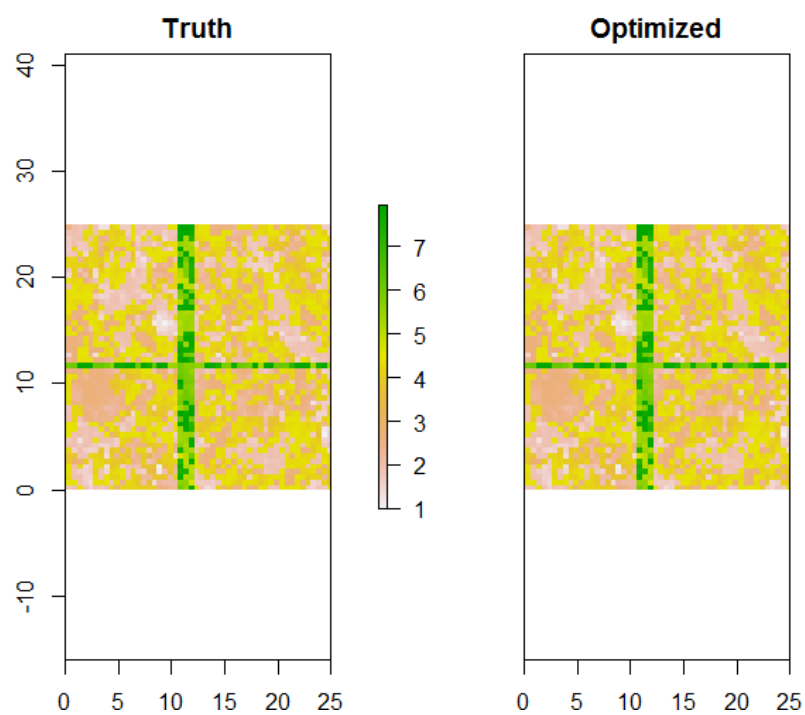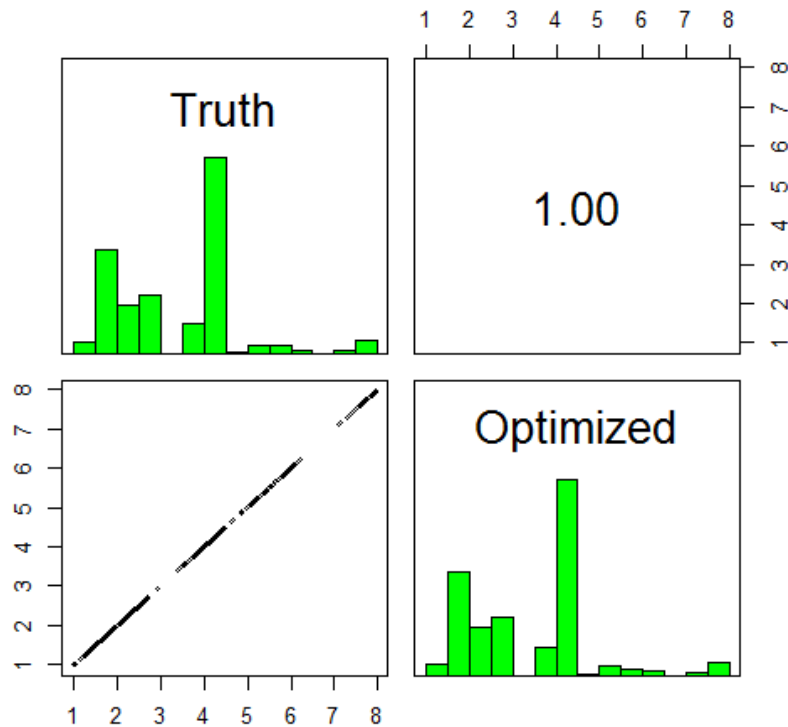
If you want to create a `CIRCUITSCAPE` current map from either the true or optimized surfaces, this can be done by setting `CurrentMap=TRUE` and `output="raster"` in `Run_CS`.

```
Resist.true <- Run_CS(CS.inputs=CS.inputs,
                      GA.inputs=GA.inputs,
                      r=Resist,
                      CurrentMap=TRUE,
                      output="raster")

Resist.opt <- Run_CS(CS.inputs=CS.inputs,
                     GA.inputs=GA.inputs,
                     r=optim.resist,
                     CurrentMap=TRUE,
                     output="raster")
```

```
# We can confirm that, like the resistance surfaces above, the CIRCUITSCAPE current maps are
cs.stack <- stack(Resist.true, Resist.opt)
names(cs.stack) <- c("Truth", "Optimized")
pairs(cs.stack)
```
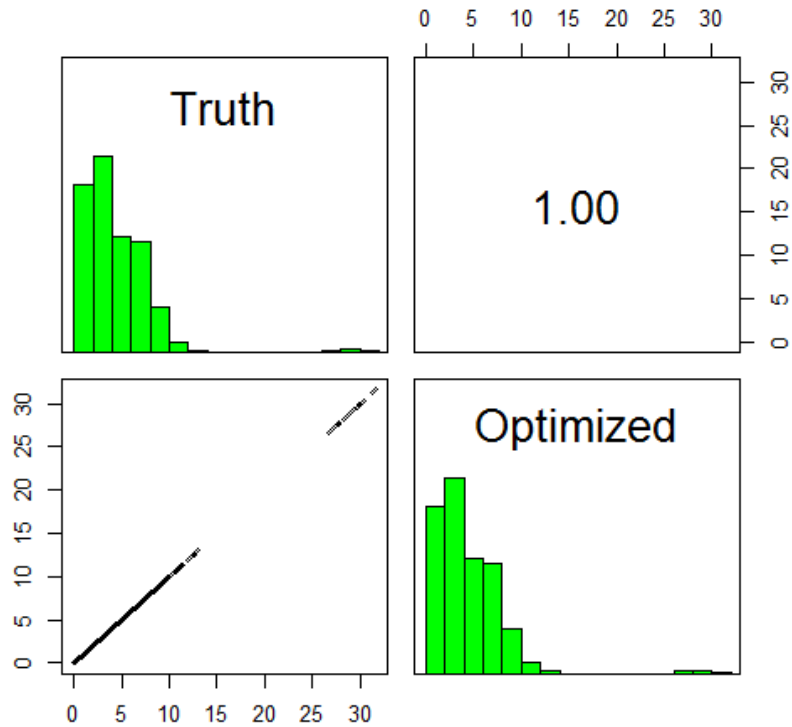

The optimization often converges on a highly correlated solution, but one that
results in relative resistance values that are identical to those of the simulated
data. This is important to understand, and interpretation of resistance values
should be made with this fact in mind.

Multisurface optimization using least cost paths

The multisurface optimization can also be run in parallel when using least cost
paths. This can be done by setting `parallel = TRUE` or `parallel = #cores`

in `GA.prep`. This example took 210 iterations to run in parallel on 4 cores, and completed in 38 minutes.

```
# Run `gdist.prep`
gdist.inputs<-gdist.prep(n.POPS=length(sample.locales),
                         samples=sample.locales)


GA.inputs <- GA.prep(ASCII.dir=resistance_surfaces,
                     Results.dir=write.dir,
                     max.cat=500,
                     max.cont=500,
                     seed = 999,
                     parallel = 4)

# Combine resistance surfaces
Resist <- Combine_Surfaces(PARM=PARM,
                           gdist.inputs=gdist.inputs,
                           GA.inputs=GA.inputs,
                           out=NULL,
                           rescale=TRUE)

# Create the true resistance/response surface
gd.response <- Run_gdistance(gdist.inputs=gdist.inputs,
                             r=Resist)

# Run `CS.prep` functions
gdist.inputs<-gdist.prep(n.POPS=length(sample.locales),
                         response=lower(as.matrix(gd.response)),
                         samples=sample.locales)

# Run `MS_optim`
Multi.Surface_optim.gd <- MS_optim(gdist.inputs=gdist.inputs,
                                   GA.inputs=GA.inputs)

summary(Multi.Surface_optim.gd)
```

Comments on multiple surface optimization:

- If the optimized resistance values are near the maximum value specified in `GA.prep`, it is recommended that you increase the maximum value and rerun the optimization.

- If the optimization ends very quickly (e.g., <40 iterations), you may want to increase the probability of mutation (`pmutation`) and/or the probability of crossover (`pcrossover`). These can be adjusted using `GA.prep`. I

have not extensively tested these settings to determine optimal values, but found that the current defaults (pmutation = 0.125, pcrossover = 0.85) have generally worked quite well with simulated data and produced reproducible estimates with real data. Alternatively, because this is a stochastic optimization, just rerun the optimization (make sure you have not set a seed!)

- Any and all settings of the `ga` function can be adjusted or customized. The main change made from the default setting for optimization of resistance surfaces was to use the "gareal_blxCrossover" method. This greatly improved the search of parameter space.

- As mentioned above concerning single surface optimization: this is a stochastic optimization process and optimized values will likely differ from run to run. Despite the time involved, it is advised to run all optimizations at least twice to confirm parameter estimates/relative relationship among resistance surfaces.

- While there is no established framework for how optimization of resistances surface can or should be done, below is a flowchart of how an analysis might proceed:
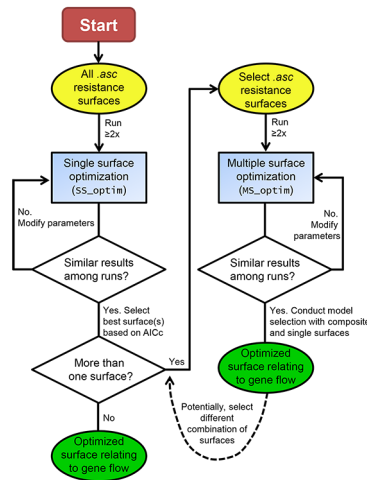
Summary

Hopefully this vignette/tutorial has demonstrated the functions present in this package and how they can be used together to optimize resistance surfaces in isolation or in combination. These methods require no *a priori* assumptions by the researcher. Optimization is conducted solely on the genetic distance data provided. The goal of this package is to make these methods accessible and useful to others. Development and advancement will continue as long as there is interest and there remains a need. Please contact me (bill.peterman@gmail.com) if you

encounter issues with any of these functions, need assistance with interpretation, or would like other features added.

Acknowledgements