Project Report - Emily Palmer

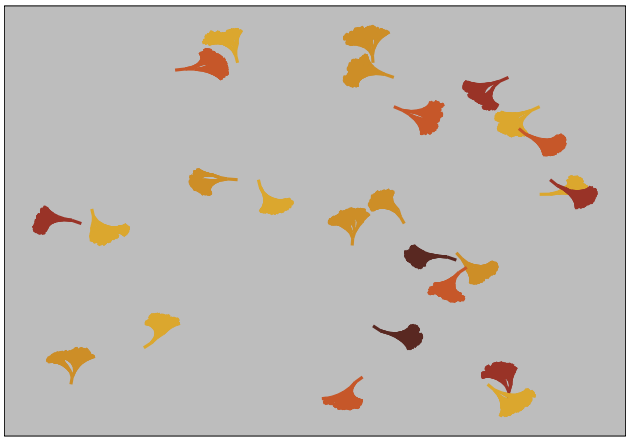
Motivation

I initially had wanted to use this project to make tree landscape pictures. After generating and plotting several trees in one plot, essentially zoomed out trees, they looked more like leaves than evergreen trees that make up the mountainscapes of the pacific northwest, so I decided to make leaf plots instead.

Package and results

For this project I created the `leafart` package, which generates leaf/tree pictures. This package contains three main functions, intended for use in the following way.

```
get_ginkgo_params() %>%
  create_leaf_pile() %>%
  plot_leaves()
```



`get_ginkgo_params()` can be replaced with `get_params()` for more general parameter values. `create_leaf_pile(param)` creates a data frame containing multiple randomly generated leaves based on parameter restraints, and `plot_leaves()` plots the leaves. The process to create these leaves and the helper functions that are combined to make the `create_leaf_pile()` are described below. There are associated help documents with each function that also explain the process.

Growing a leaf

A leaf starts with a stalk. We consider this initial stalk to be the zeroth "layer". We keep track of the stalk start and endpoint so we can draw a line between them. This initial stalk also contains angle, length, and time information for future growth. The following shows the data representation of the initial stalk.

```
initial_leaf
```

```
## # A tibble: 1 x 7
##     x_0   y_0   x_1   y_1 angle length iter_n
##   <dbl> <dbl> <dbl> <dbl> <dbl>  <int>  <int>
## 1     0     0     0     1    90      1      1
```

Branches of the next layer are created by splitting off at the endpoints of the previous layer. The way the leaf/tree branches depends on the parameter values we pass to the growing function. These parameters could

be single values, or a vector of values to be sampled from, which creates the variation in the leaves. The function `get_params()` will give us a list of parameter values. We can specify what values we want, or we can use its defaults. It will default to making one tree. Here we specify one layer, and two splits, which will create a tree/leaf with the initial stalk, and one additional layer of growth (two branches).

To grow one branch, we use the function `one_branch()`. This function returns the growth of the new branch. Its start point will be the endpoint of the branch it grew from.

```
params <- get_params(n_layer = 1, split = 2, scale = seq(.1,.9,1), angle = -40:20)
one_branch(initial_leaf, params)
```

```
## # A tibble: 1 x 7
##     x_0   y_0    x_1   y_1 angle length iter_n
##   <dbl> <dbl>  <dbl> <dbl> <dbl>  <dbl>  <int>
## 1     0     1 0.0454  1.09    63    0.1      2
```

To grow all the branches in this second layer we use the function `grow_leaf_layers()`. This function iterates the function `one_branch()` over the number of new branches we want (the parameter splits). This combines each branch (row) into one data frame that contains one row per new branch. Since we specified we want 2 splits, we will get 2 new branches.

```
grow_leaf_layers(initial_leaf, params)
```

```
## # A tibble: 2 x 7
##     x_0   y_0    x_1   y_1 angle length iter_n
##   <dbl> <dbl>  <dbl> <dbl> <dbl>  <dbl>  <int>
## 1     0     1 0.0259  1.10    75    0.1      2
## 2     0     1 0.0602  1.08    53    0.1      2
```

The function `grow_leaf()` will do this entire process, creating the initial stalk and growing the new layer (depending on the parameters). It accumulates calls to `grow_leaf_layers()` to do so. The initial location and rotation of the initial stalk are randomly placed. Each element of the list is a different layer of the tree.

```
(leaf <- grow_leaf(params))
```

```
## [[1]]
## # A tibble: 1 x 7
##     x_0   y_0   x_1   y_1 angle length iter_n
##   <int> <int> <dbl> <dbl> <dbl>  <int>  <int>
## 1     2    26     2    27    90      1      1
##
## [[2]]
## # A tibble: 2 x 7
##     x_0   y_0   x_1   y_1 angle length iter_n
##   <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <int>
## 1     2    27  2.02  27.1    76    0.1      2
## 2     2    27  2.06  27.1    54    0.1      2
```

This format is easy to grow from, but hard to plot. The function `rake_leaves()` will combine the list elements and convert the four start and endpoint columns into just `x` and `y` columns which are the aesthetics used for the plotting function. It also keeps track of a `step` column which is used as a grouping variable that keeps track of what layer and branch the points belong to decide which dots to connect.

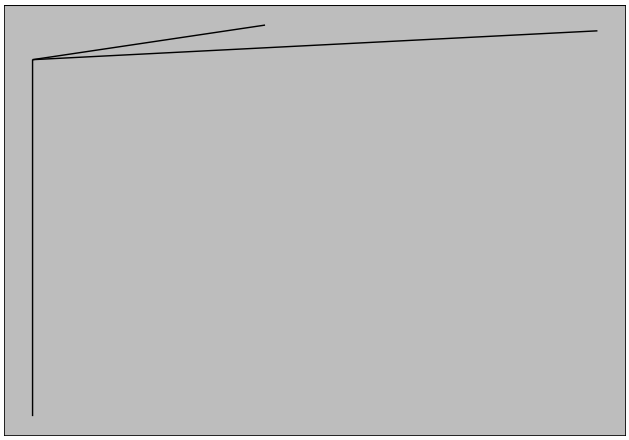
The `plot_leaves()` function will plot a leaf after it is "raked".

```
(raked_leaf <- rake_leaves(leaf))
```

```
## # A tibble: 6 x 3
##       x     y  step
```

```
##   <dbl> <dbl> <int>
## 1  2     26       1
## 2  2     27       1
## 3  2     27       2
## 4  2.02  27.1     2
## 5  2     27       3
## 6  2.06  27.1     3
```

```
plot_leaves(raked_leaf)
```



As we can see from the resulting image, there is the initial stalk, and one layer consisting of 2 split off branches at random lengths and angles.

The function `create_leaf_layers()` creates multiple leaves and combines this growing and raking process.

Attempts at speeding up runtime

Creating and plotting the leaves will take a moment to run, especially if we are creating many leaves which each have many layers. I tried to improve this run time by instead of generating each tree individually, we generate a smaller number of distinct trees, and then replicate that smaller number of trees to create the full leaf pile. These duplicated trees then need to be also randomly rotated and placed, using the functions `rotate_leaf()` and `move_leaf()`. However, this process actually took longer. This exploration can be found in the document here

Tests

There are a couple tests for the key functions. They are included in the test folder, and mostly test the dimensions of output of the functions, which ensures the leaf is growing in the expected manor given parameter constraints.

Examples

Examples of using this code and resulting images from different parameter specifications can be found in the package README. All functions have associated help pages created from roxygen. Users can also utilize the shiny app to explore the code as well.

Acknowledgements and sources

Big thank you to Charlotte Wickham for advice and help with code. Thanks to the members of OSSSO for being a great study group. Finally thanks to Danielle Navarro's flametree package for providing inspiration and ideas, especially regarding how to transform the data into a plottable format.