TeaSetClass_vignette

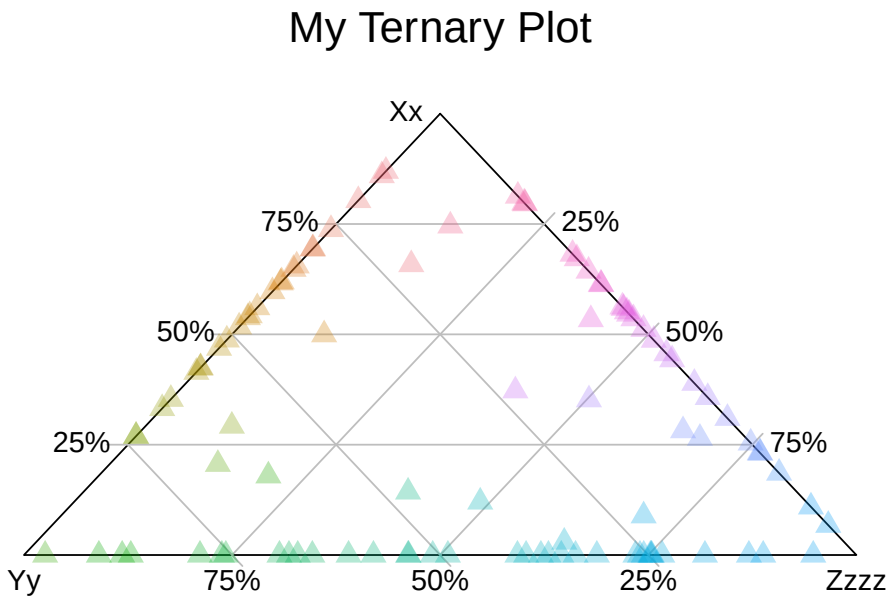
Christopher Pearson

```
library(TeaSet)
library(dplyr)
#>
#> Attaching package: 'dplyr'
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
```

The TeaSet Class

This class is built to create ternary graphs and help manipulate ternary data. Ternary data is used to show the ratios between three variables. However, there are not many tools to help construct a ternary graph. The TeaSet class looks to remedy this for R. In this class, the vectors i, j and k are used to indicate moving up, to the bottom-right, and the bottom-left respectively. Similar to three-dimensional coordinates, t-coordinates use an (x,y,z) format–albeit they cannot treat negatives in the same manner. To start, we will create some test data and storing it in a three-column matrix. This data can quickly be graphed with the quick_tea_plot() function.

```
testData <- as.data.frame(matrix(rnorm(300, sd=10),ncol=3))
names(testData) <- c("Xx","Yy","Zzzz")
quick_tea_plot(testData, main='My Ternary Plot')
```



Handling Negative Values and Data Normalization

The axes of a t-plot do not have room for negative values. To handle these, TeaSet redistributes negative values so that they add half their value to the other two coordinates. Thus, (-2,1,1) is treated as (0,2,2), while (-1,-1,-1) cancels out to (1,1,1). Another way that t-coordinates differ from normal 3d xyz-coordinates is that all values are normalized. In 3d space, (1,1,1) and (100,100,100) are far from one another, in a t-plot they occupy the same space. When graphed in two-dimensions, all t-points are divided by their maximimum value.

```
testData <- as.data.frame(matrix(rnorm(300, sd=10),ncol=3))
teaSet <- brew_tea(testData)
temp<- t(teaSet$p_redistribute_negatives())  # p_ methods are intended for internal class use, but are
temp<-as.data.frame(temp)
temp<-cbind(temp,t(teaSet$p_normalize_ternary_pt()))
names(temp) <- c("X_No_negs","Y_No_negs","Z_No_negs","X-Normal","Y-Normal","Z_Normal")
head(temp)
#>   X_No_negs Y_No_negs Z_No_negs  X-Normal  Y-Normal  Z_Normal
#> 1  0.000000  5.823316  5.992735 0.0000000 0.4928310 0.5071690
#> 2  1.514902  0.000000  5.085334 0.2295224 0.0000000 0.7704776
#> 3  6.213544 13.067105  0.000000 0.3222684 0.6777316 0.0000000
#> 4 23.822568  0.000000  8.892390 0.7281858 0.0000000 0.2718142
#> 5  5.870659  9.321309  0.000000 0.3864318 0.6135682 0.0000000
#> 6 18.062922 14.544233  0.000000 0.5539558 0.4460442 0.0000000
```

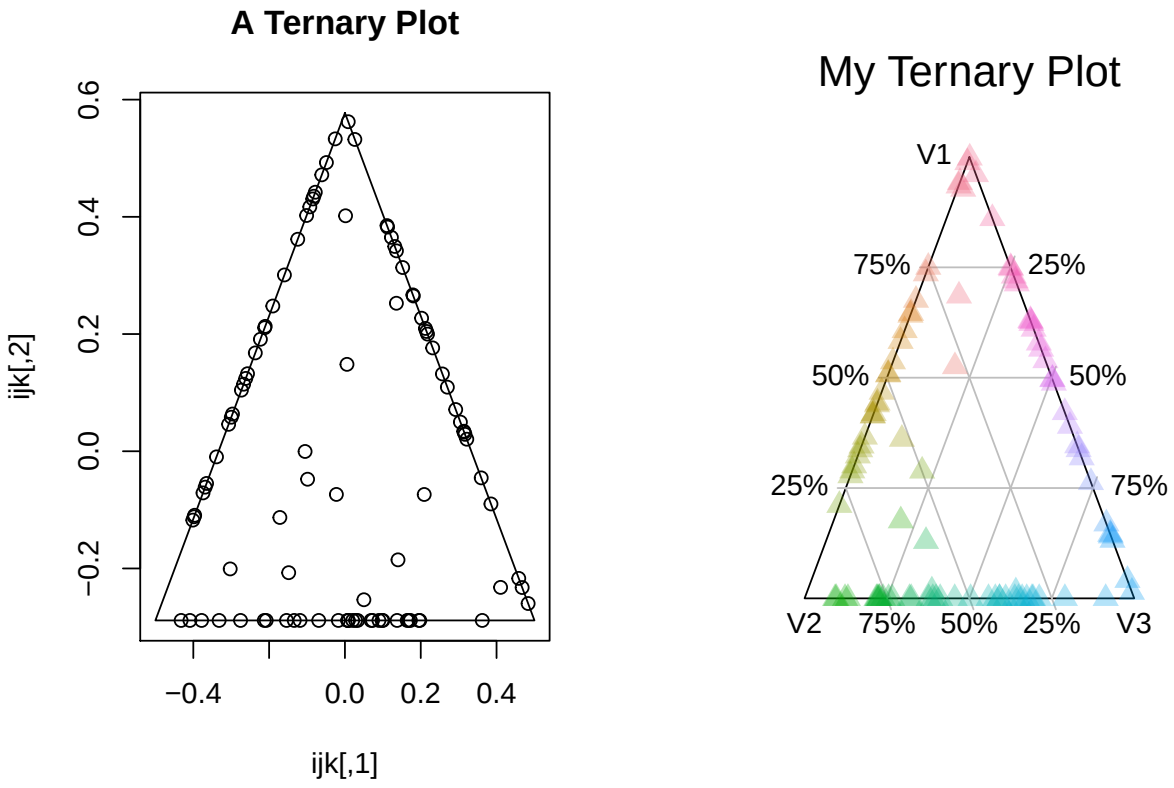
Getting XY Coordinates & IJK

The workhorse functions of this package are get_ijk() and get_xy(). The get_ijk() method provides the three vectors that the get_xy() method uses to calculate xy-coordinates. These points are also the vertices of the border triangle for the t-plot. With these two functions one can easily make a ternary plot; however, there are many additional steps if one wants a presentable t-plot.

quick_tea_plot()

The easiest way to plot ternary data is with the quick_tea_plot() function–which only needs a set of ternary data as an argument. Quick_tea_plot() simply calls the packages main plotting method, tea_plot() with a number of presets.

```
testData <- as.data.frame(matrix(rnorm(300, sd=10),ncol=3))
layout(matrix(c(1,2), nrow=1, byrow = TRUE))
ijk <- teaSet$get_ijk()
xy <- teaSet$get_xy()
plot(ijk, type = 'l', main ="A Ternary Plot")
points(xy)
quick_tea_plot(testData, main = "My Ternary Plot")
```

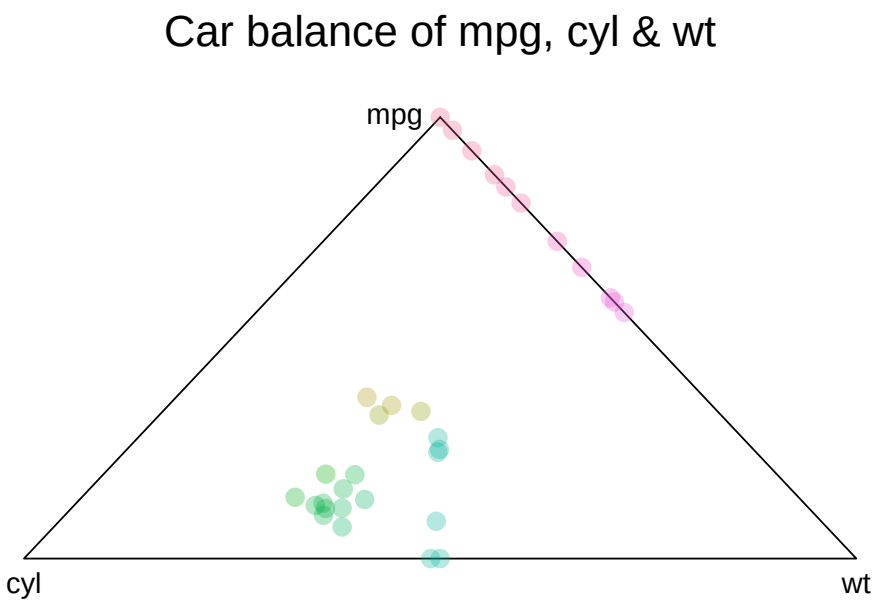


Plotting in Greater Depth: teaSet$tea_plot() and tea_plot(teaSet)

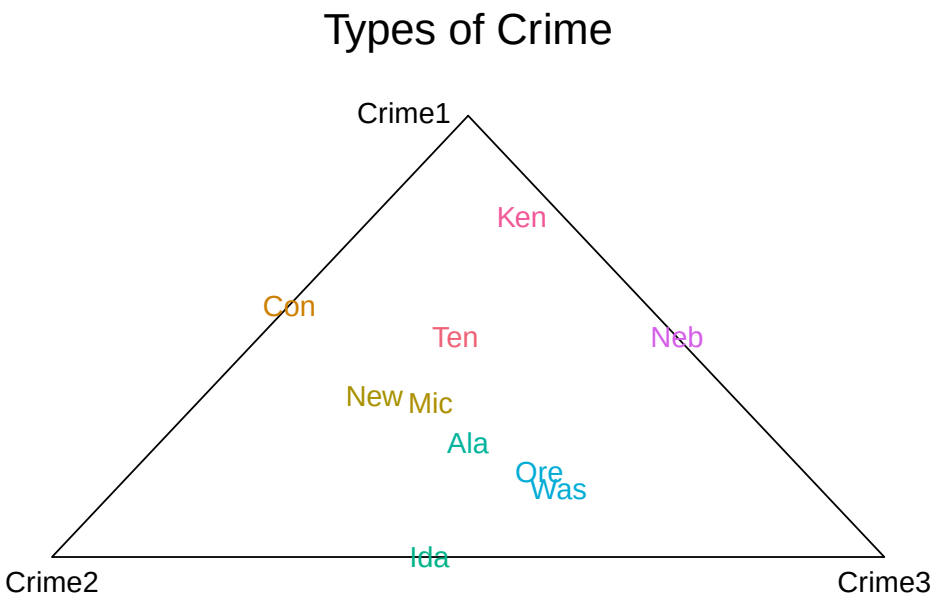
The main plotting function in TeaSet is tea_plot(), which can be accessed as a function or a method atta
Some standard plot() parameters work differently, or not longer work at all. The col parameter in parti

such as xlab and ylab no longer work--they do not make sense for a ternary plot. The xlab and ylab para
The newPlot variable, defaulting to TRUE, allows one to decide whether to use tea_plot() to start a new
There are several options for drawing ticks or similar lines to help distinguish values. Setting bulls

```
data(mtcars)
mtcars %>%
  select(mpg,cyl,wt) %>%
  normalize("minmax") %>%
  brew_tea() %>%
  tea_plot(main="Car balance of mpg, cyl & wt",col="gradient90",pch=16,cex=1.5)
```




```
data(USArrests)
testData<-USArrests[seq(2,50, by=5),]
names(testData)<-c("Crime1","Crime2","Pop","Crime3")
rownames(testData)<-substr(rownames(testData),1,3)
testData %>%
  select(Crime1,Crime2,Crime3) %>%
  normalize() %>%
  brew_tea() %>%
  tea_plot(main = "Types of Crime",dataLabels=TRUE,col="contrast",alpha=1)
```



myStretch, myCenter, and set_frame()

Those looking to incorporate a ternary plot into their visuals could be more interested in incorporating
Moving the frame around while graphing requires the use of the tea_plot() method, as one needs a TeaSet

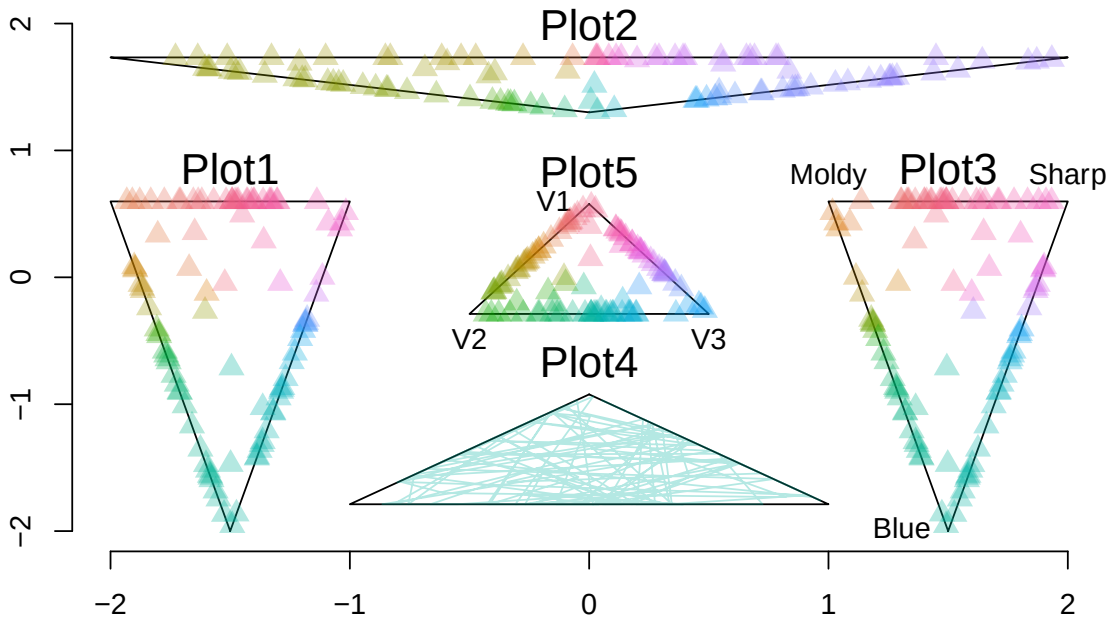
```
testData <- as.data.frame(matrix(rnorm(300, sd=10),ncol=3))
plot(c(-2,2),c(-2,2),type='n',frame.plot=FALSE,axes=TRUE,ylab="",xlab="")
# returns<- is simply to hide the return-values for set_frame()
returns<-teaSet$set_frame(xFrame = c(-2,-1),yFrame = c(1,-2),inplace=TRUE)
teaSet$tea_plot(newPlot=FALSE, main="Plot1",pch=17,cex=1.5,axis.labels=FALSE)

returns<-teaSet$set_frame(xFrame = c(-2,2), yFrame = c(1.8,1.3),inplace=TRUE)
teaSet$tea_plot(newPlot=FALSE, main="Plot2",pch=17,cex=1.5,axis.labels=c("","",""))

returns<-teaSet$set_frame(xFrame = c(2,1), yFrame = c(1,-2),inplace=TRUE)
teaSet$tea_plot(newPlot=FALSE,main="Plot3",pch=17,cex=1.5,axis.labels=c("Blue","Sharp","Moldy"))

returns<-teaSet$set_frame(center = c(0,-1.5),stretch=c(2,1),inplace=TRUE)
teaSet$tea_plot(newPlot=FALSE,main="Plot4",type = 'l',cex=1.5,axis.labels=NA) # not really built for li
#> Warning in polygon(...): graphical parameter "type" is obsolete

returns<-teaSet$set_frame(inplace=TRUE)
teaSet$tea_plot(newPlot=FALSE,main="Plot5",pch=17,cex=1.5,axis.labels=TRUE)
```



Drawing

Always trying to shift from an xy-coordinate system to ternary coordinates can be rather difficult, and
The tea_triangles() method draws a triangle similar to the area of a t-plot. If that area is stretched
The flexibility of tea_triangle() is shown with the tea_gradient_background() function, which creates r
The tea_lines() method can draw a line with only one value of x, y or z. With ternary coordinates, one
The tea_segments() method on the other hand simply takes two t-points as inputs (either as seperate inp

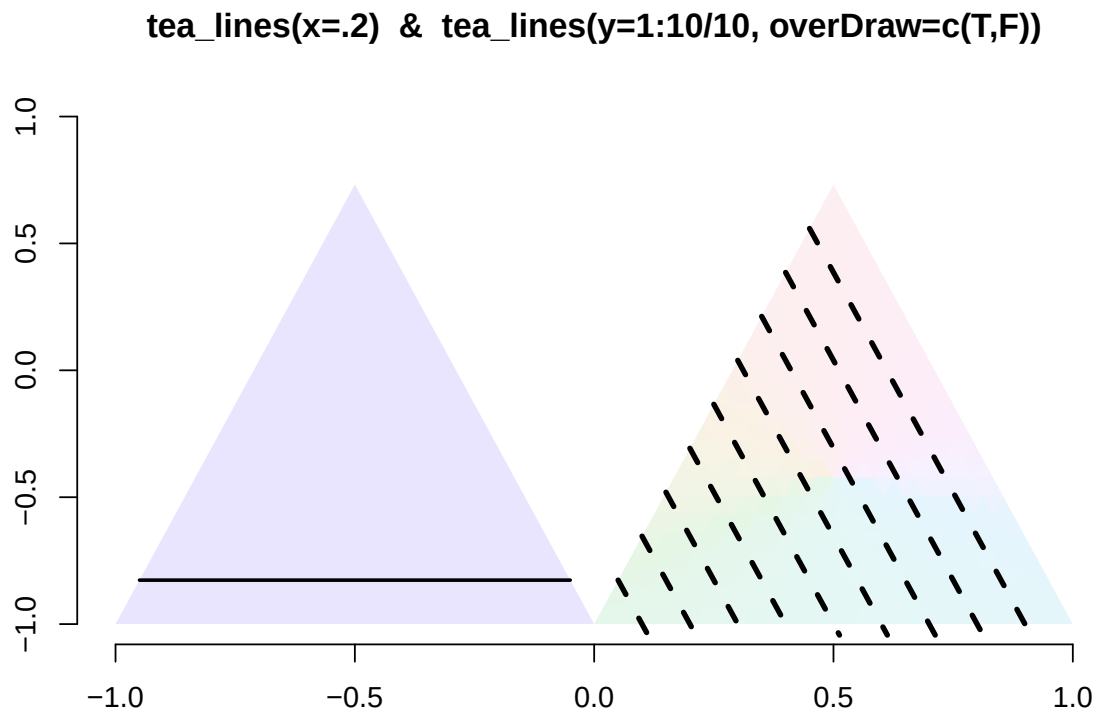
```
plot(c(-1,1),c(-1,1),type='n',frame.plot=FALSE,axes=TRUE,ylab="",xlab="", main = "tea_lines(x=.2)  &  t

returns<-teaSet$set_frame(xFrame = c(-1,0),yFrame=c(-1,1),inplace=TRUE)

returns<-teaSet$tea_triangle(alpha=0.2,border=NA)
returns<-teaSet$tea_lines(x=.1,col="black",lwd="2")

returns<-teaSet$set_frame(xFrame = c(0,1),yFrame=c(-1,1),inplace=TRUE)

returns<-teaSet$tea_gradient_background(rows=24)
returns<-teaSet$tea_lines(y=1:9/10, overDraw = c(TRUE,FALSE),col='black',lwd=3,lty=2)
```

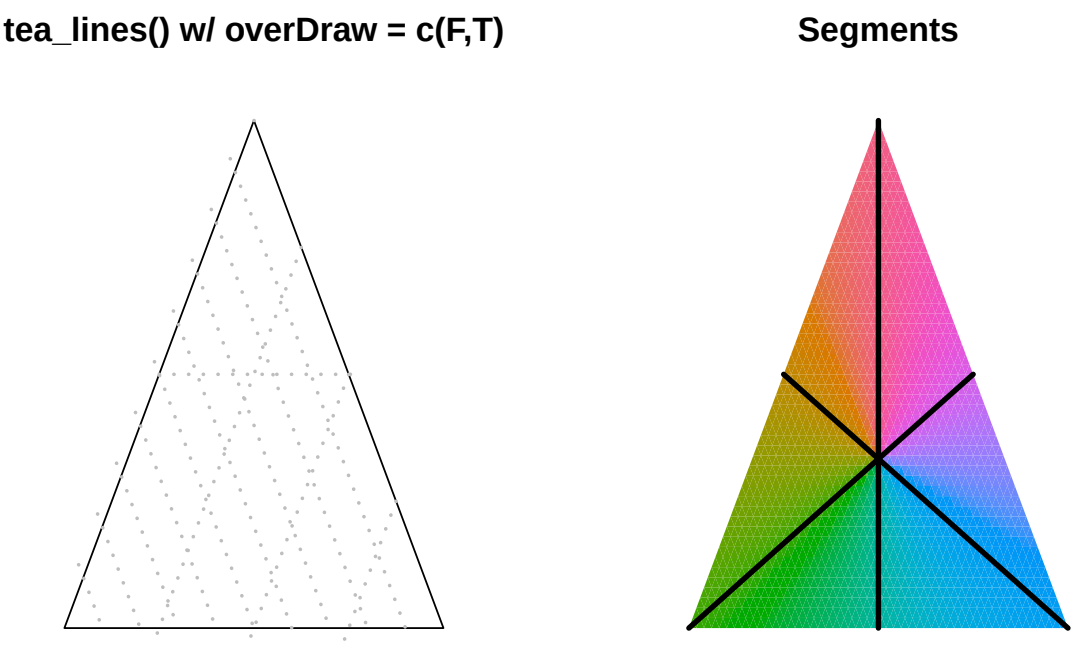



```
returns<-teaSet$set_frame(inplace=TRUE)
```

The tea_lines() function ultimately calls the tea_segments() function, which simply converts t-coordinates to xy-coordinates and then calls the normal segments() function to draw the line. The tea_segments() function works with a variety of inputs. The "normal" way to call it would be two 3-length lists representing t-points, but it also works with a 6-length list representing two t-points. Likewise, it can handle 3-column and 6-column matrices.

```
layout(matrix(1:2, nrow=1, byrow = TRUE))
plot(teaSet$get_ijk(), type = 'l', frame.plot=FALSE,axes=FALSE,ylab="",xlab="",main="tea_lines() w/ ove
returns<-teaSet$tea_lines(x=1:2/2,y=1:9/10,z=1:3/4, overDraw=c(FALSE,TRUE),lwd=2,lty=3)
returns<-test<-matrix(c(1,0,0,0,1,1,
               0,1,0,1,0,1,
               0,0,1,1,1,0),ncol=6,byrow=TRUE)

plot(teaSet$get_ijk(), type = 'n', frame.plot=FALSE,axes=FALSE,ylab="",xlab="",main="Segments")
teaSet$tea_gradient_background(rows=50,alpha=1)
returns<-teaSet$tea_segments(test,lwd=3)
```

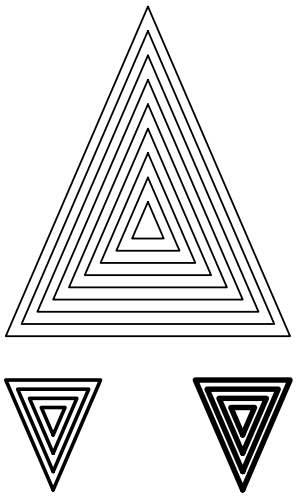


There is also a quick_triangle() function to allow the use of tea_triangles without needing to create a TeaSet object. It follows the same parameters: r to indicate the size of the similar triangle, and center to indicate where to draw it. However, this is best understood as a wrapper aroundthe polygon function, and does not rely on creating a TeaSet object within itself. As it does not rely on myStretch, these are always equilateral triangles. One final aspect of all of these function is that they have a draw=T/F paramter, and they return the coordinates used to draw them. Thus, one could call the tea_lines() function simply to get the coordinates to input into another geometric function. However, this is not the most wieldy way to draw polygons.

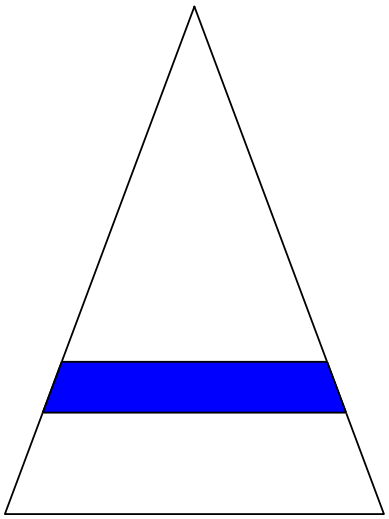
```
layout(matrix(1:2, nrow=1, byrow = TRUE))
plot(c(-1,1),c(-1,1),type='n',frame.plot=FALSE,axes=FALSE,ylab="",xlab="",main="quick_triangle()")
returns<-quick_triangle(1:9/6)
returns<-quick_triangle(-1*1:4/8,center=c(0.5,-0.75),lwd=3)
returns<-quick_triangle(-1*1:4/8,center = c(-0.5,-0.75),lwd=2)

plot(teaSet$get_ijk(),type='l',frame.plot=FALSE,axes=FALSE,ylab="",xlab="",main="tea_lines data fed int
poly <-(teaSet$tea_lines(x=c(.2,.3),draw=TRUE))
poly <- matrix(poly,ncol=2,byrow=TRUE)
polygon(poly[c(1,3,4,2),],col="blue")
```

quick_triangle()



tea_lines data fed into polygon



Conclusion

I hope this has been a useful demonstration of TeaSet's plotting capabilities. This is a new package though, so I would appreciate your feedback on any errors or features that should be added.