Math9

Ryan J. Cooper

11/13/2020

```r
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.6.3
```

```
##
## 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
```

```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.6.3
```

Finite Mathmatics in R

I recently enrolled in Math 9, finite mathematics and teh Santa Rosa Junior College. This transfer-level math class is intended as a broad survey of mathematics subjects "other than calculus". As such, it's a great fit for anyone interested in R - the subjects of matrix operations, probability, expected value, statistcs, set theory are essential to the understanding of R. Many important financial formulas are also covered in this course, providing a solid foundation for the subjects of interest, annuities, sinking funds, amoritization, present value, and other key conepts in finance. I took the course with Professor Wyndham Galbraith.

In this paper, I will demonstrate some of the ways that matrix operations can be applied in R, and highlight the natural fit of R to the basics of set theory. R is well-suited to act as a financial calculator platform.

The course text is "Finite Mathematics, An Applied Approach" by Michael Sullivan. The textbook has an online component offered through the Wiley elearning platform. Mr. Galbraith set up the Wiley platfor m to be "integrated" with SRJC's LMS platform, Canvas, so they work well in conjunction, although there are some slight quirks in the way the two programs interact. Together the syste provides a very nice platform, with canvas giving you all you course scheduling, grading, and general structure, and Wiley providing all the actual math learning content.

For this analysis, I will use GGPlot2 for visualizations and the pipe operator from DPLYR.

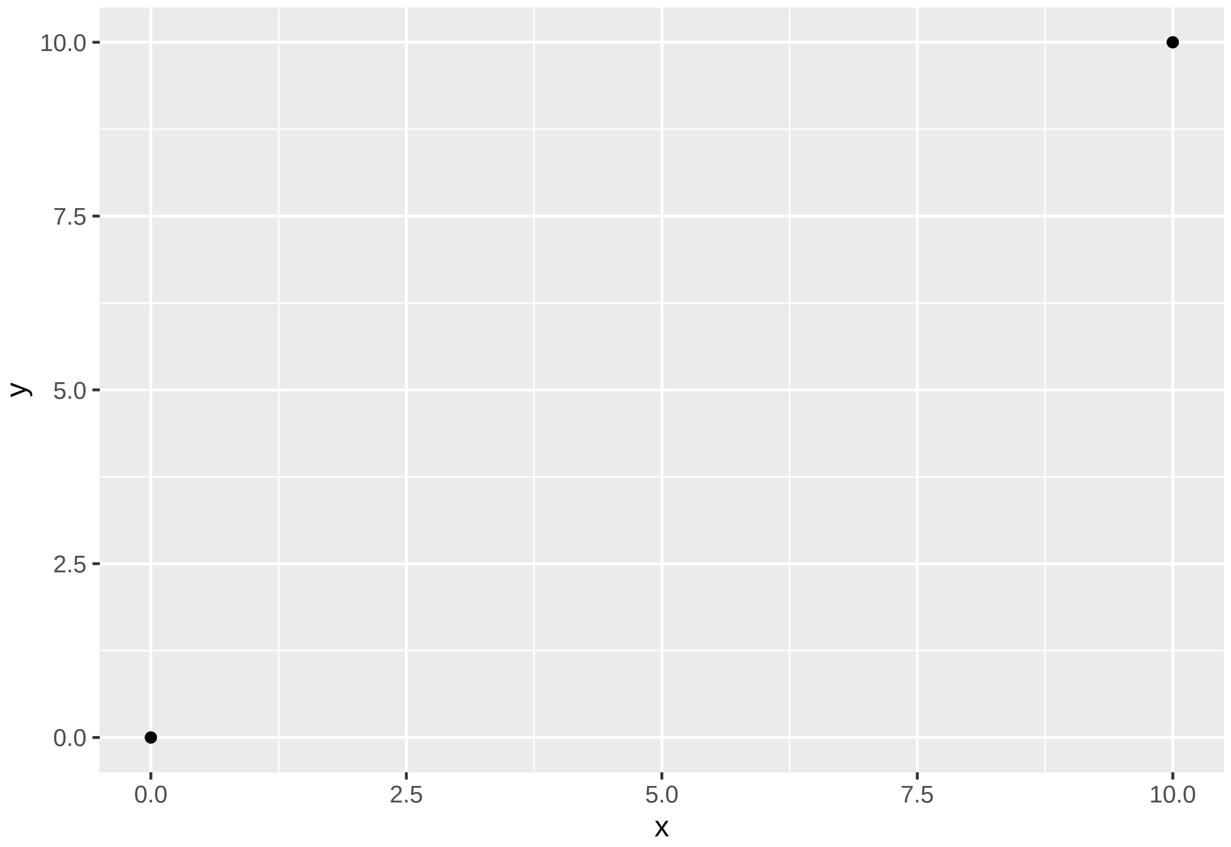
Chapter 1 Linear Equations

This chapter was not covered in the course - but I am including all chapetrs in the wiley text just to ensure the content has a place in my analysis. I suspect other instructors may include this chapter - but it is pretty basic stuff like graphing a linear equation and finding the slope of a line.

Graphing Points on a Cartesian Plane

Create two points and build a simple plot with just those 2 points at {0,0} and {10,10}

```r
x_points <- c(0,10)
y_points <- c(0,10)
coords <- data.frame(x=x_points,y=y_points)

simple_plot <- ggplot(data=coords, aes(x=x, y=y)) +
  geom_point()
simple_plot
```

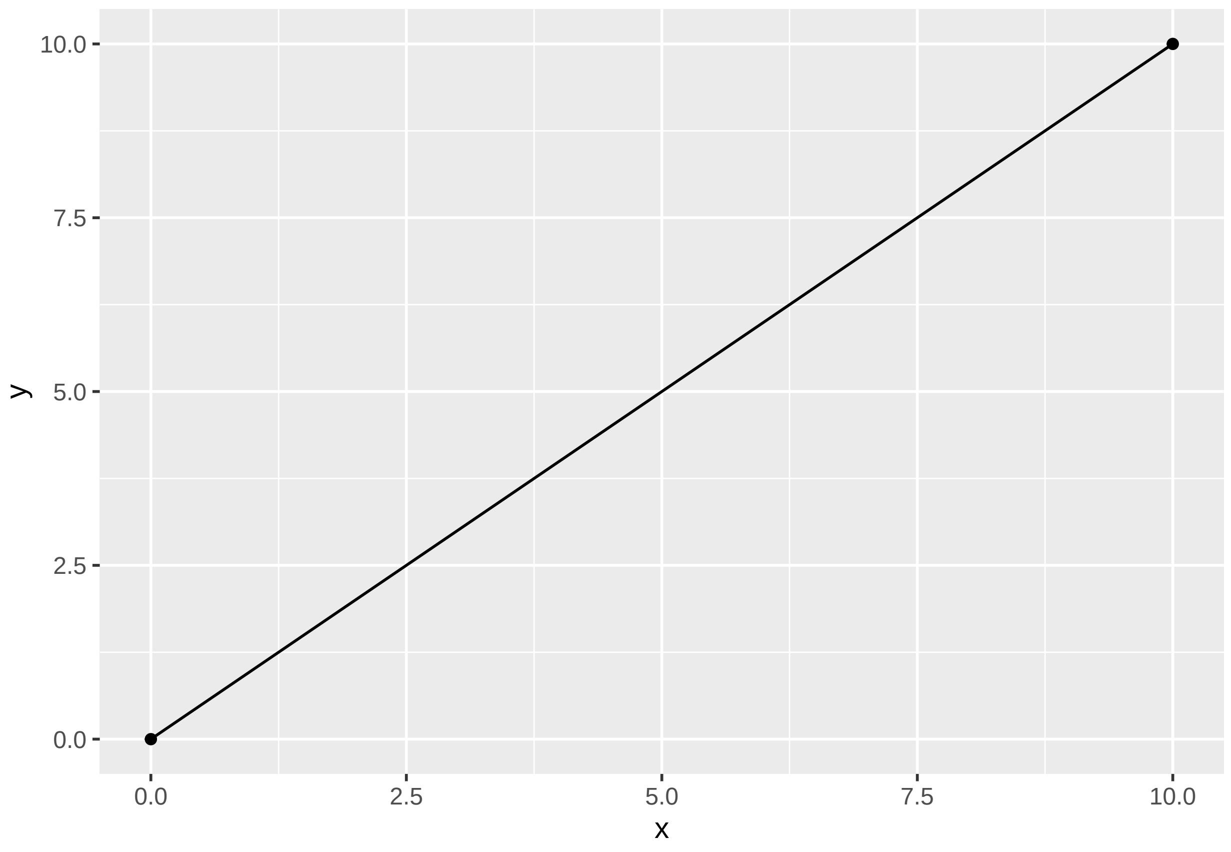


Graphing a Line between Points

Create two points and build a simple plot with just those 2 points at {0,0} and {10,10}

To add a line between these two points, simply add a geom_line object.

```r
x_points <- c(0,10)
y_points <- c(0,10)
coords <- data.frame(x=x_points,y=y_points)

simple_plot <- ggplot(data=coords, aes(x=x, y=y)) +
  geom_point() + geom_line()
simple_plot
```

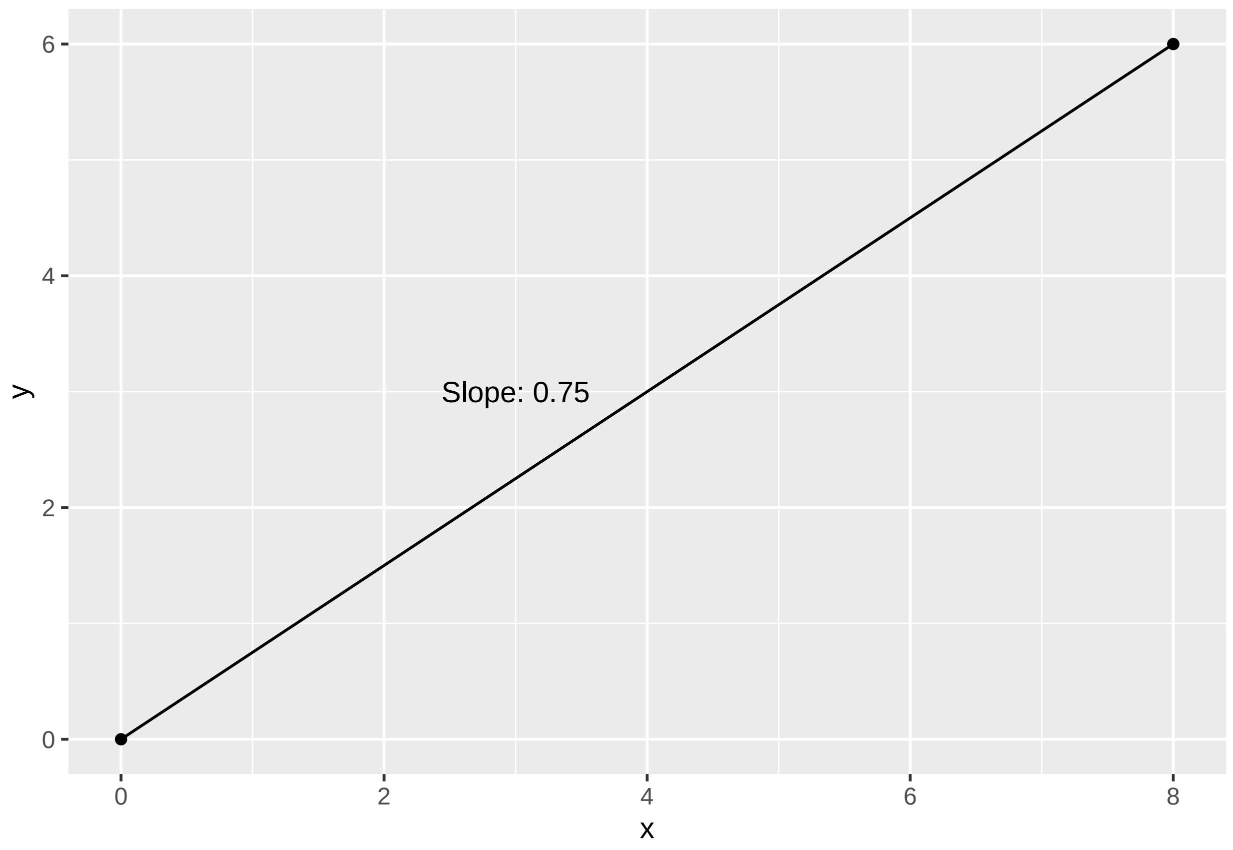


Slope of a Line

Our first equations will be to cal;culate the slope of a line. For each formal equation we create, we will create a function that encapsulates that function. Calculate the slope of a line and display it on a chart as an annotation:

```r
#x/y points
x_points <- c(0,8)
y_points <- c(0,6)
coords1 <- data.frame(x=x_points,y=y_points)

#slope of a line equation
slope_line <- function(coords1){
  m <- (coords1$y[2] - coords1$y[1]) / (coords1$x[2] - coords1$x[1])
  m
}
```

```r
#run the function and set the output to variable m
m <- slope_line(coords1)

#plot
simple_plot <- ggplot(data=coords1, aes(x=x, y=y)) +
  geom_point() +
  geom_line() +
  annotate("text", x=3,y=3,label = paste("Slope:",m))
simple_plot
```



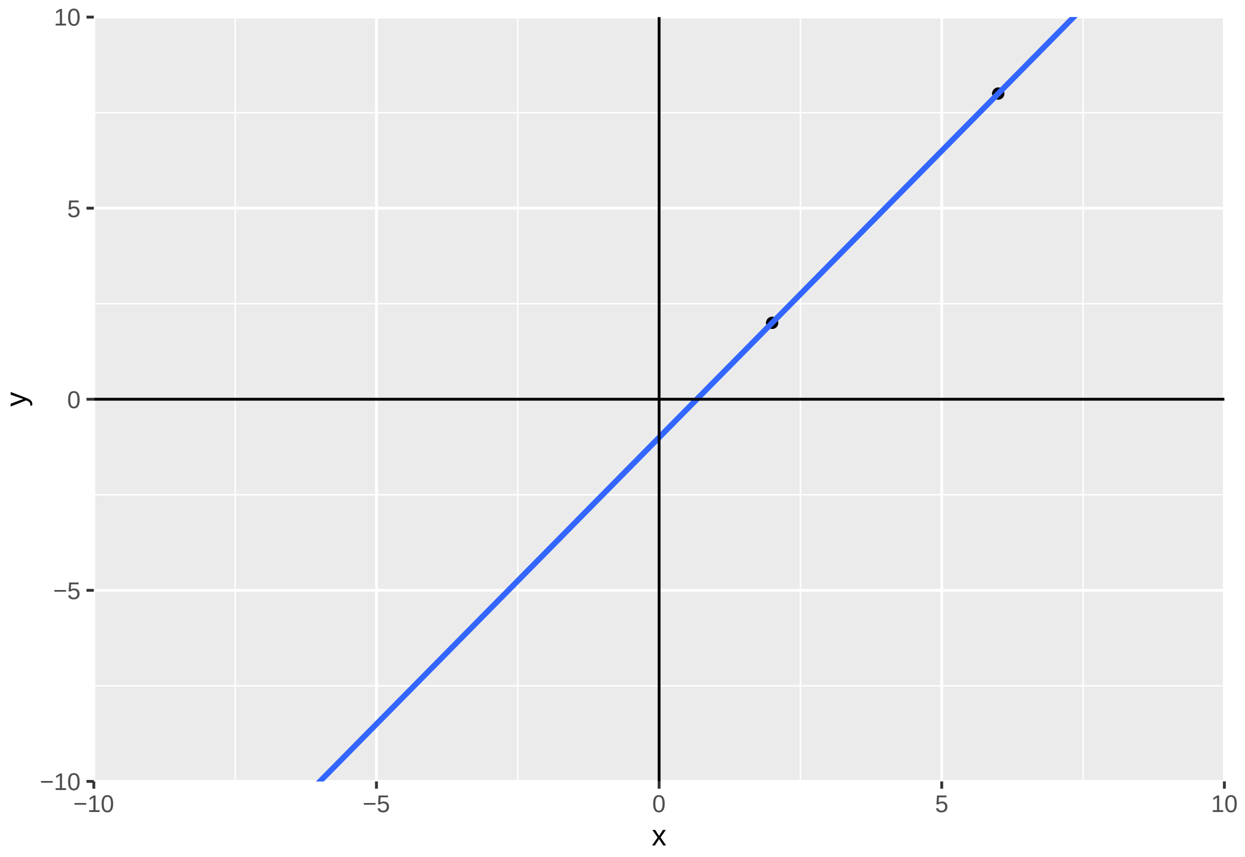
Graphing a Line with geom_smooth

This is a variant of the above line graphs. the geom_smooth produces a *linear regression* line between n points. This will draw a line, based on your points, using a model of your choosing. We will start with a linear model (a straight line). This object will be important in later chapters as we explore plots that contain many points. In ths section we are introducing the coord_cartesian object. This object will "zoom in" on a portion of the area, without dropping any points. This is different than than scale object which will set the scale of the area and remove items outside the limits.

```r
#x/y points
x_points <- c(2,6)
y_points <- c(2,8)
coords1 <- data.frame(x=x_points,y=y_points)
```

```r
#set bounds of focus area
upper_bound <- 10
lower_bound <- -10
left_bound <- -10
right_bound <- 10

#set limits to extend beyond the bounds
lower_bound_f <- ifelse(lower_bound>0,0,lower_bound)
left_bound_f <- ifelse(left_bound>0,0,left_bound)
upper_bound_f <- ifelse(upper_bound<0,0,upper_bound)
right_bound_f <- ifelse(right_bound<0,0,right_bound)

#plot
simple_plot <- ggplot() +
  geom_point(data=coords1, aes(x=x, y=y)) +
  xlim(left_bound_f*2, right_bound_f*2) +
  ylim(lower_bound_f*2, upper_bound_f*2) +
  geom_smooth(data=coords1, aes(x=x, y=y),method="lm",fullrange=TRUE) +
  geom_hline(yintercept=0) +
  geom_vline(xintercept=0) +
  coord_cartesian(xlim = c(left_bound, right_bound),
                  ylim = c(lower_bound, upper_bound),
                  expand = FALSE)
simple_plot
```

```
## `geom_smooth()` using formula 'y ~ x'

## Warning in qt((1 - level)/2, df): NaNs produced

## Warning: Removed 27 rows containing missing values (geom_smooth).

## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -
## Inf
```

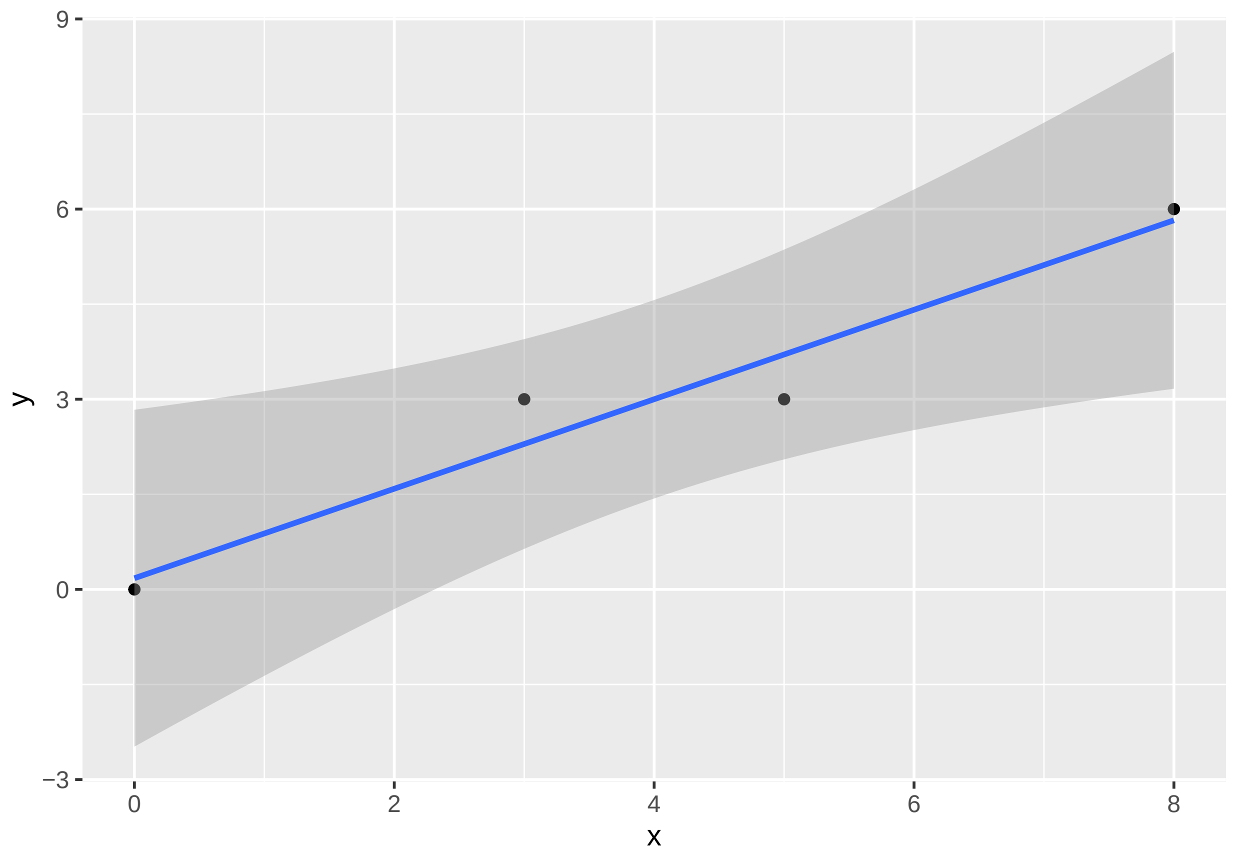


Graphing a Linear Equation with 3 or more Points with geom_smooth

This geom_smooth produces a linear regression line between 3 points. The shaded area represents the standard error of this regression line using a 95% confidence interval by default.

```
#x/y points
x_points <- c(0,3,5,8)
y_points <- c(0,3,3,6)
coords1 <- data.frame(x=x_points,y=y_points)

#plot
simple_plot <- ggplot(data=coords1, aes(x=x, y=y)) +
  geom_point() +
  geom_smooth(method="lm")
simple_plot
```

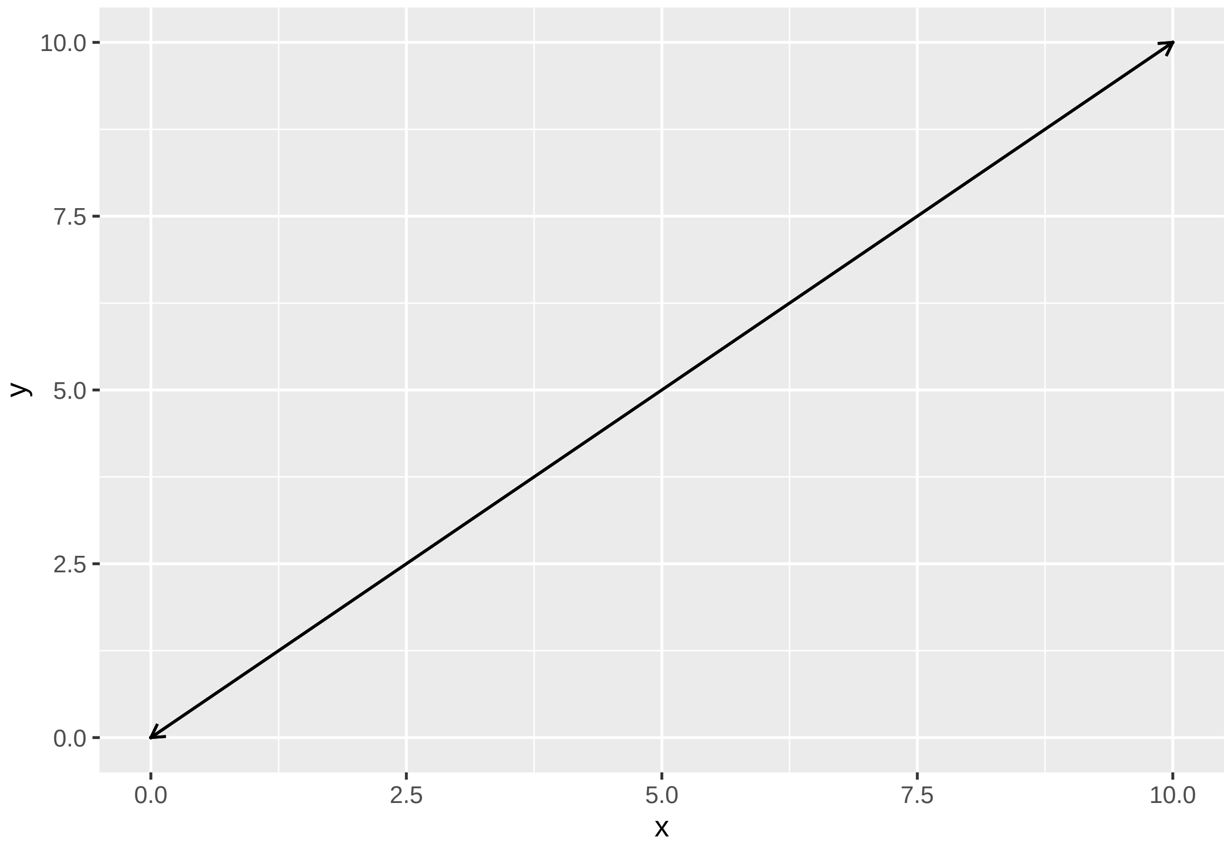
```
## `geom_smooth()` using formula 'y ~ x'
```



Graphing a line with arrows with geom_segment

Another way to represent an infinite line that continues into an endless space, is to use the geom_segment object. This geom_segment object produces a line between 2 points with an arrowhead on the end. You can add two lines that are both covering the same x/y space, but going opposite directions, with points size set to 0, to simulate the appearance of an infinite line.

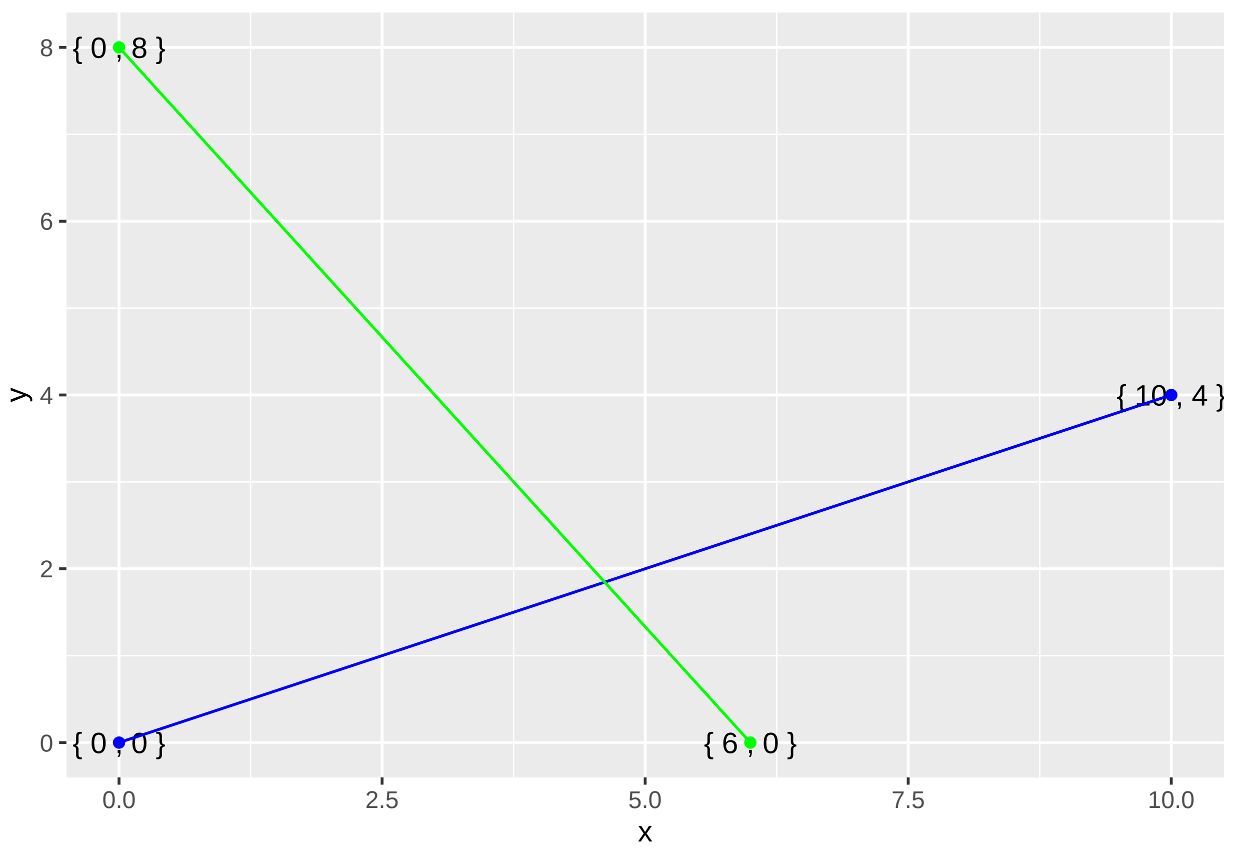
```
x_points <- c(0,10)
y_points <- c(0,10)
coords <- data.frame(x=x_points,y=y_points)

simple_plot <- ggplot(data=coords, aes(x=x, y=y)) +
  geom_point(size=0) +
  geom_segment(x=coords$x[1],xend=coords$x[2],y=coords$y[1],yend=coords$y[2],arrow = arrow(length = uni
  geom_segment(x=coords$x[2],xend=coords$x[1],y=coords$y[2],yend=coords$y[1],arrow = arrow(length = uni
simple_plot
```



Graphing 2 Lines

Create 2 sets of 2 points and draw a line between them. In this example, the two equations are being treated as separate data frames. This enables us to pass these into the ggplot objects **data** parameter for each element we want to draw. Note that how the data parameter has moved out of the ggplot() object, and into the individual geom elements. We are also adding some text to show the x,y coordinates of each point.

```
x_points <- c(0,10)
y_points <- c(0,4)
coords1 <- data.frame(x=x_points,y=y_points)

x_points <- c(0,6)
y_points <- c(8,0)
coords2 <- data.frame(x=x_points,y=y_points)

multi_plot <- ggplot() +
  geom_text(data=coords1, aes(x=x, y=y, label=paste("{",x,",",y,"}")),color="black") +
  geom_point(data=coords1, aes(x=x, y=y),color="blue") +
  geom_line(data=coords1, aes(x=x, y=y),color="blue") +
  geom_text(data=coords2, aes(x=x, y=y, label=paste("{",x,",",y,"}")),color="black") +
  geom_point(data=coords2, aes(x=x, y=y),color="green") +
  geom_line(data=coords2, aes(x=x, y=y),color="green")
multi_plot
```

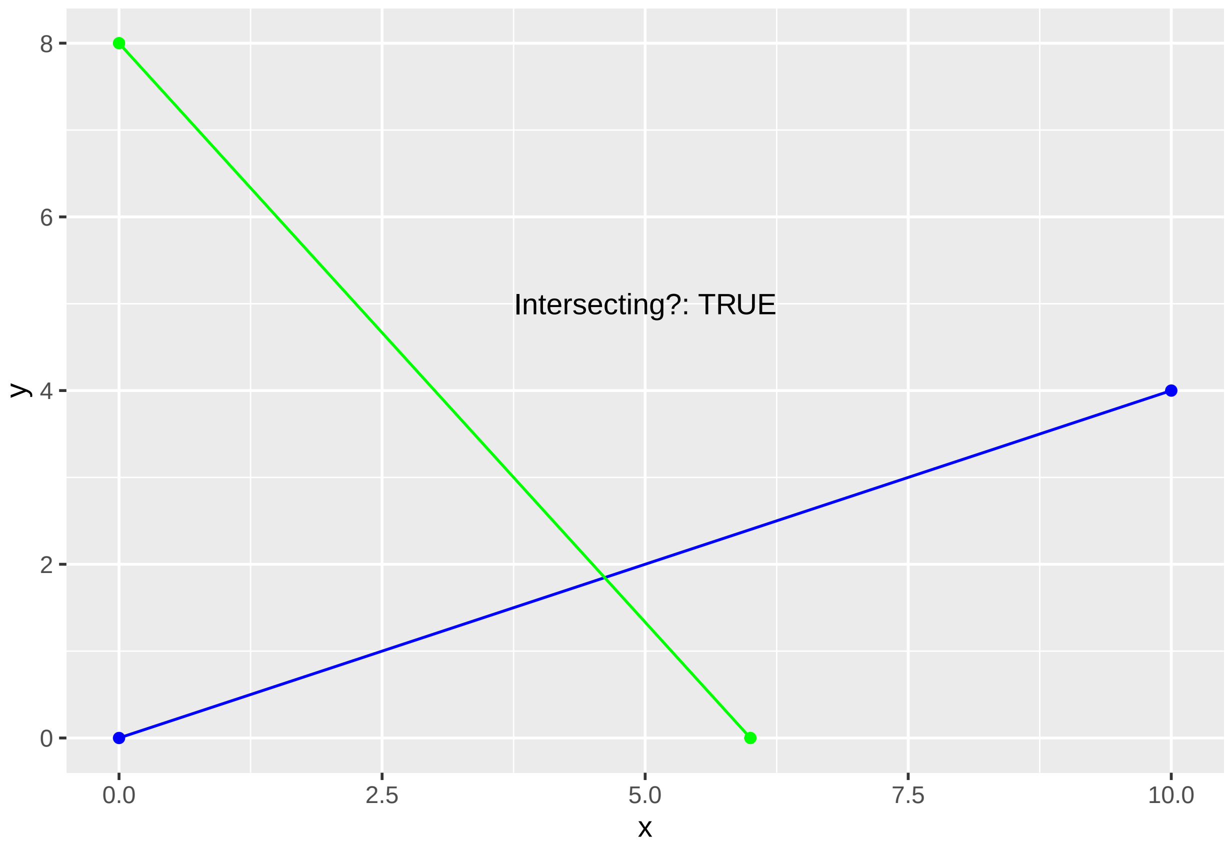


Check Intersection of 2 Lines

This function will to determine if 2 lines intersect by checking the slope of each line using our slope_line function on both sets of coordinates. If the slope is not eequal - the lines must intersect. The function will return TRUE if the lines intersect. *important note: This function assumes that the lines continue forever in both directions. It is only checking that the slope of teh two lines is not equal, not detecting any two segments overlap.

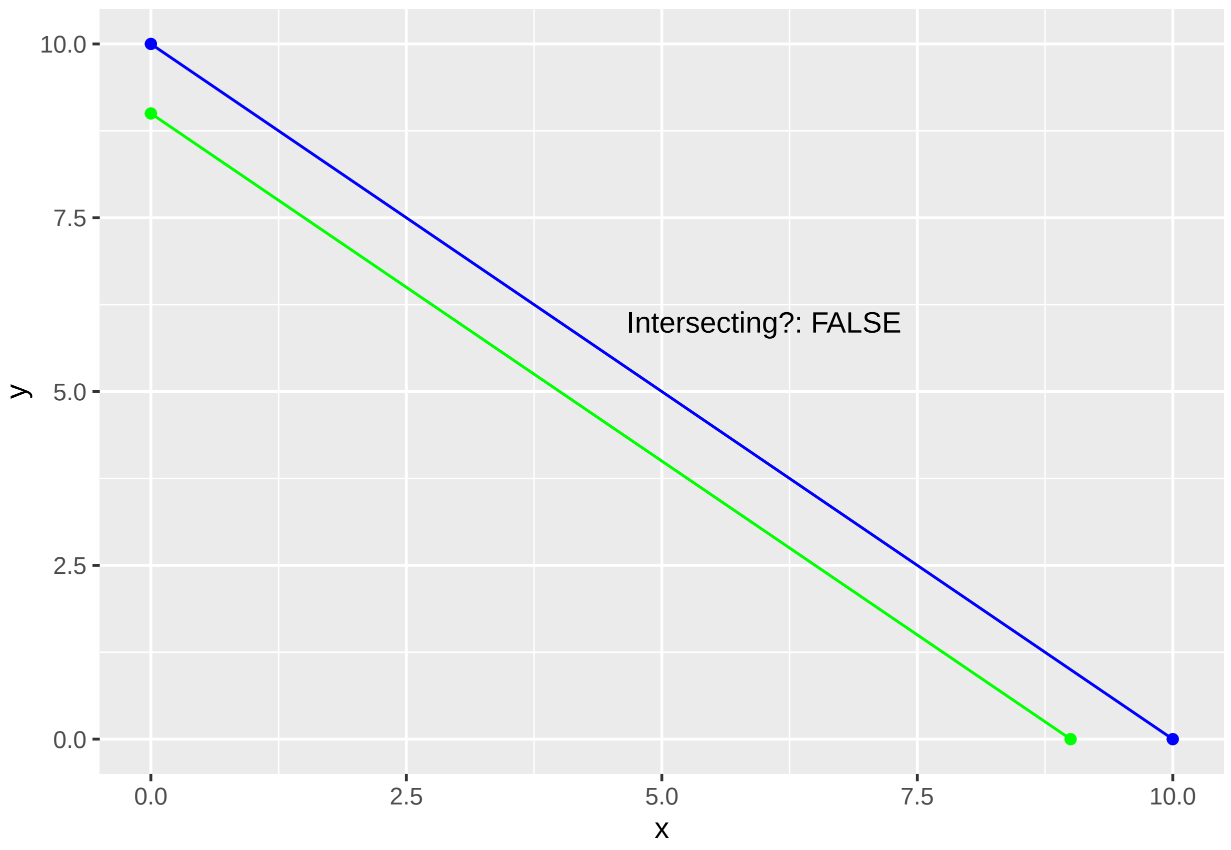
```r
#x/y points
x_points <- c(0,10)
y_points <- c(0,4)
coords1 <- data.frame(x=x_points,y=y_points)

x_points <- c(0,6)
y_points <- c(8,0)
coords2 <- data.frame(x=x_points,y=y_points)

#check if the slopes intersect
check_intersect <- function(coords1,coords2){
  m1 <- slope_line(coords1)
  m2 <- slope_line(coords2)
  is_intersecting <- ifelse(m1 == m2,FALSE,TRUE)
  is_intersecting
}
```

```
#run intersect check
intersects <- check_intersect(coords1,coords2)

#plot
multi_plot <- ggplot() +
  geom_point(data=coords1, aes(x=x, y=y),color="blue") +
  geom_line(data=coords1, aes(x=x, y=y),color="blue") +
  geom_point(data=coords2, aes(x=x, y=y),color="green") +
  geom_line(data=coords2, aes(x=x, y=y),color="green") +
  annotate("text", x=5,y=5,label = paste("Intersecting?:",intersects))
multi_plot
```



```
#x/y points
x_points <- c(0,10)
y_points <- c(10,0)
coords1 <- data.frame(x=x_points,y=y_points)

x_points <- c(0,9)
y_points <- c(9,0)
coords2 <- data.frame(x=x_points,y=y_points)

#run intersect check
intersects <- check_intersect(coords1,coords2)

#plot
```

```
multi_plot <- ggplot() +
  geom_point(data=coords1, aes(x=x, y=y),color="blue") +
  geom_line(data=coords1, aes(x=x, y=y),color="blue") +
  geom_point(data=coords2, aes(x=x, y=y),color="green") +
  geom_line(data=coords2, aes(x=x, y=y),color="green") +
  annotate("text", x=6,y=6,label = paste("Intersecting?:",intersects))
multi_plot
```



Chapter 2: Systems of Linear Equations

In this chapter we looked at how to solve a system of equations using substitution, elimination, and matrix methods.

Single Linear Expression

This code demonstrates how you might set up a linear equation. In the example below - you are trying to determine the values of x1 and x2 for each set, such that eval_linear would evaluate to TRUE. First we will establish the basic logic, then, we will build some code to simulate the steps required to solve a system of linear equations.

```
#8x + 6y = 3580

#define coefficients
```

```r
a1 <- 8
a2 <- 6

#define right hand side
b <- 3580

#set variables to 0
x1 <- 0
x2 <- 0

set1 <- data.frame(a1=a1,a2=a2,b=b,x1=x1,x2=x2)

#--------

#a linear equation with 2 variables has the form
eval_linear1 <- (set1$a1 * set1$x1) + (set1$a2 * set1$x2) == set1$b

eval_linear1
```

```
## [1] FALSE
```

Set of Linear Expressions

Expanding on the section above - this code represents the setup of a set of 2 linear equations each having 2 variables.

```r
#8x + 6y = 3580

#define coefficients
a1 <- 8
a2 <- 6

#define right hand side
b <- 3580

#set variables to 0
x1 <- 0
x2 <- 0

set1 <- data.frame(a1=a1,a2=a2,b=b,x1=x1,x2=x2)


#--------

#define coefficients
a1 <- 1
a2 <- 1

#define right hand side
b <- 525

#set variables to 0
```

```
x1 <- 0
x2 <- 0

set2 <- data.frame(a1=a1,a2=a2,b=b,x1=x1,x2=x2)

#--------


#a linear equation with 2 variables has the form
eval_linear1 <- (set1$a1 * set1$x1) + (set1$a2 * set1$x2) == set1$b
eval_linear2 <- (set2$a1 * set2$x1) + (set2$a2 * set2$x2) == set2$b

eval_linear1
```

```
## [1] FALSE
```

```
eval_linear2
```

```
## [1] FALSE
```

Shading an area above a line

This chart includes a polygon shading of the area above the line. This could be useful for linear programming and optimization problems.

```
buildPoly <- function(xr, yr, slope = 1, intercept = 0, above = TRUE){
    #Assumes ggplot default of expand = c(0.05,0)
    xrTru <- xr + 0.05*diff(xr)*c(-1,1)
    yrTru <- yr + 0.05*diff(yr)*c(-1,1)

    #Find where the line crosses the plot edges
    yCross <- (yrTru - intercept) / slope
    xCross <- (slope * xrTru) + intercept

    #Build polygon by cases
    if (above & (slope >= 0)){
        rs <- data.frame(x=-Inf,y=Inf)
        if (xCross[1] < yrTru[1]){
            rs <- rbind(rs,c(-Inf,-Inf),c(yCross[1],-Inf))
        }
        else{
            rs <- rbind(rs,c(-Inf,xCross[1]))
        }
        if (xCross[2] < yrTru[2]){
            rs <- rbind(rs,c(Inf,xCross[2]),c(Inf,Inf))
        }
        else{
            rs <- rbind(rs,c(yCross[2],Inf))
        }
    }
    if (!above & (slope >= 0)){
        rs <- data.frame(x= Inf,y= -Inf)
```

```r
        if (xCross[1] > yrTru[1]){
            rs <- rbind(rs,c(-Inf,-Inf),c(-Inf,xCross[1]))
        }
        else{
            rs <- rbind(rs,c(yCross[1],-Inf))
        }
        if (xCross[2] > yrTru[2]){
            rs <- rbind(rs,c(yCross[2],Inf),c(Inf,Inf))
        }
        else{
            rs <- rbind(rs,c(Inf,xCross[2]))
        }
    }
    if (above & (slope < 0)){
        rs <- data.frame(x=Inf,y=Inf)
        if (xCross[1] < yrTru[2]){
            rs <- rbind(rs,c(-Inf,Inf),c(-Inf,xCross[1]))
        }
        else{
            rs <- rbind(rs,c(yCross[2],Inf))
        }
        if (xCross[2] < yrTru[1]){
            rs <- rbind(rs,c(yCross[1],-Inf),c(Inf,-Inf))
        }
        else{
            rs <- rbind(rs,c(Inf,xCross[2]))
        }
    }
    if (!above & (slope < 0)){
        rs <- data.frame(x= -Inf,y= -Inf)
        if (xCross[1] > yrTru[2]){
            rs <- rbind(rs,c(-Inf,Inf),c(yCross[2],Inf))
        }
        else{
            rs <- rbind(rs,c(-Inf,xCross[1]))
        }
        if (xCross[2] > yrTru[1]){
            rs <- rbind(rs,c(Inf,xCross[2]),c(Inf,-Inf))
        }
        else{
            rs <- rbind(rs,c(yCross[1],-Inf))
        }
    }

    return(rs)
}
```

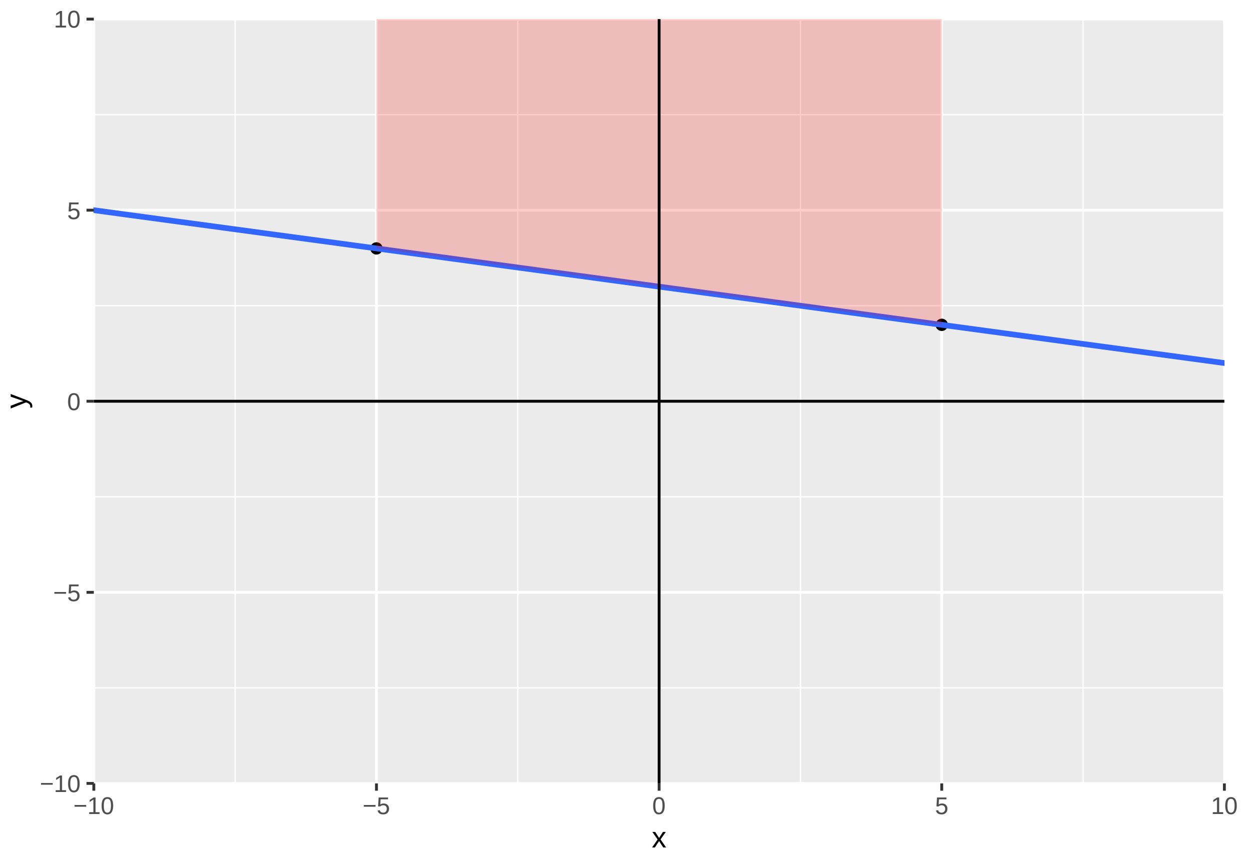
```r
#x/y points
x_points <- c(-5,5)
y_points <- c(4,2)
coords1 <- data.frame(x=x_points,y=y_points)

#set bounds of focus area
```

```r
upper_bound <- 10
lower_bound <- -10
left_bound <- -10
right_bound <- 10

#set limits to extend beyond the bounds
lower_bound_f <- ifelse(lower_bound>0,0,lower_bound)
left_bound_f <- ifelse(left_bound>0,0,left_bound)
upper_bound_f <- ifelse(upper_bound<0,0,upper_bound)
right_bound_f <- ifelse(right_bound<0,0,right_bound)

sl <- diff(coords1$y) / diff(coords1$x)
int <- coords1$y[1] - (sl*coords1$x[1])

#Build the polygon
datPoly <- buildPoly(range(coords1$y),range(coords1$x),
            slope=sl,intercept=int,above=FALSE)

min_x <- min(coords1$x)
max_x <- max(coords1$x)

df_poly_above <- coords1 %>%
  add_row(x = c(max_x, min_x),
                y = c(Inf, Inf))

#plot
simple_plot <- ggplot() +
  geom_point(data=coords1, aes(x=x, y=y)) +
  xlim(left_bound_f*2, right_bound_f*2) +
  ylim(lower_bound_f*2, upper_bound_f*2) +
  geom_smooth(data=coords1, aes(x=x, y=y),method="lm",fullrange=TRUE) +
  #geom_ribbon(data=coords1, aes(ymin = y, ymax = y + 10,x = x), fill = "grey70") +
  geom_polygon(data = df_poly_above,
             aes(x = x, y = y),
             fill = "red",
             alpha = 1/5) +
  geom_hline(yintercept=0) +
  geom_vline(xintercept=0) +
  coord_cartesian(xlim = c(left_bound, right_bound),
                ylim = c(lower_bound, upper_bound),
                expand = FALSE)
simple_plot
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning in qt((1 - level)/2, df): NaNs produced
```

```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -
## Inf
```



10
5
0
−5
−10
y
−10
−5
0
5
10
x