Homework 2 is designed to give you more experience with making graphs in R. The specific problem involves the human contrast sensitivity function, but the solution is much more general.

Generally to plot data you need data for the x-axis and data for the y-axis. There are several ways to provide these: via individual vectors, or by creating a data frame with the x- and y- numbers in them. The first approach is shown below. First create the x-values, then create the y-values. The length of the x and y vectors must be the same.

```r
x <- c(1, 2, 3, 5, 8, 10, 20, 30, 50)
y <- c(0.1, 0.01, 0.005, 0.004, 0.01, 0.05, 0.2, 0.8, 1.0)
```

Since the homework assignment involves contrast thresholds, \( c \), but you are asked to plot contrast sensitivity \( (1/c) \), you should transform the y data into values that you need (shown below):

```r
recip <- 1 / y
```

Now you can make plots. I strongly recommend that you define an R function for each of the plots you want to draw. That way you can keep refining and improving the code that makes the plots and then to draw it you only have to issue a single command. Here are the definitions for the first plot. The plot command is simple but can quickly become complex the nicer you want to make it. Below is the code that specifies a function to plot \( x \) as a function of \( y \), plot both lines and data points, make the y axis logarithmic, and finally provide appropriate labels for the graph. there are many other options available. Type `help(plot)` in R to see some of them.

```r
plot1 <- function() {
  plot(y ~ x,
       type = "b",
       log = "y",
       xlab = "The x-axis",
       ylab = "The y-axis",
       main = "A lin-log Plot")
}
```

Here is the definition of the plot function for the same data but with logarithmic scales on both the x- and y-axes.

```r
plot2 <- function() {
  plot(y ~ x,
       type = "b",
       log = "xy",
       xlab = "The x-axis",
       ylab = "The y-axis",
       main = "A log-log Plot")
}
```

Now that `plot1()` and `plot2()` have been defined you can make these plots simply by giving the `plot1()` or the `plot2()` command, just as you would for any other R command. The commands below create a single graph with two plots arranged in with one row and two columns:
par(mfcol = c(1, 2))
plot1()
plot2()

A lin–log Plot

![Lin-log Plot](image)

A log–log Plot

![Log-log Plot](image)

par(mfcol = c(1, 1))

Whoops. I forgot that we were supposed to plot the reciprocal of y, not y itself. Since we already computed the reciprocal, the easiest way is to assign the object \( y \) to contain \( 1/y \) instead of \( y \), then reissue the plot commands:

\[
y <- recip 
\]

par(mfcol = c(1, 2))
plot1()
plot2()
The homework can be done in an analogous fashion. You can spend hours making your graphs look beautiful, add color, reference lines, etc. but at this stage make them simple, but clear!!

Hint about functions

The way the functions are defined above does not make them very flexible: they only can plot vectors that are named x and y, and when I redefined y to contain recip, I destroyed the original contents of y. A more flexible way to define these plot functions is to provide arguments that will handle different x and y vectors. Here are revised versions:

```r
plot1 <- function(xplt = x, yplt = y) {
  plot(yplt ~ xplt,
       type = "b",
       log = "y",
       xlab = "The x-axis",
       ylab = "The y-axis",
       main = "A lin-log Plot")
}
```

and

```r
plot2 <- function(xplt = x, yplt = y) {
  plot(yplt ~ xplt,
       type = "b",
       log = "y",
       xlab = "The x-axis",
       ylab = "The y-axis",
       main = "A log-log Plot")
}
```
When you evoke the plot1() or plot2() command without any arguments, they will look for objects x and y. But if you give the command with new vectors they will use those data.

```r
y <- c(0.1, 0.01, 0.005, 0.004, 0.01, 0.05, 0.2, 0.8, 1.0)
recip <- 1 / y
par(mfcol = c(1, 2))
plot1()
plot2(x, recip)
```