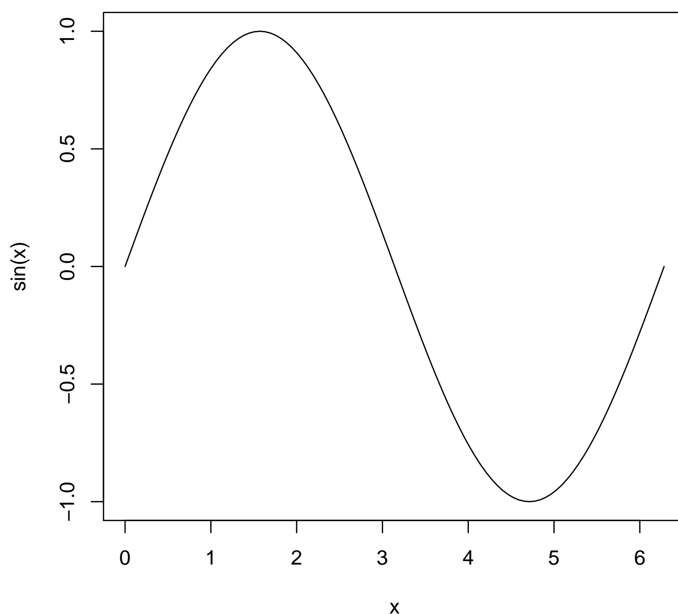
Some R Routines for Plotting Non-Vectorizing Functions

James H. Steiger
Psychology 311

1 Introduction

In Psychology 310, we were introduced to the idea of plotting functions of a single variable x using the `curve` function. A typical example would be plotting the function $\sin(x)$ over the range from 0 to 2π.

```
> curve(sin(x),0,2*pi)
```



However, on occasion you will discover that, despite the fact that you have apparently called the `curve` function correctly, it just doesn't work and delivers a strange error message.

Just such an example occurs with Psychology 311 Homework 2, problem 2b.

2 Problems with a Non-vectorizing Function

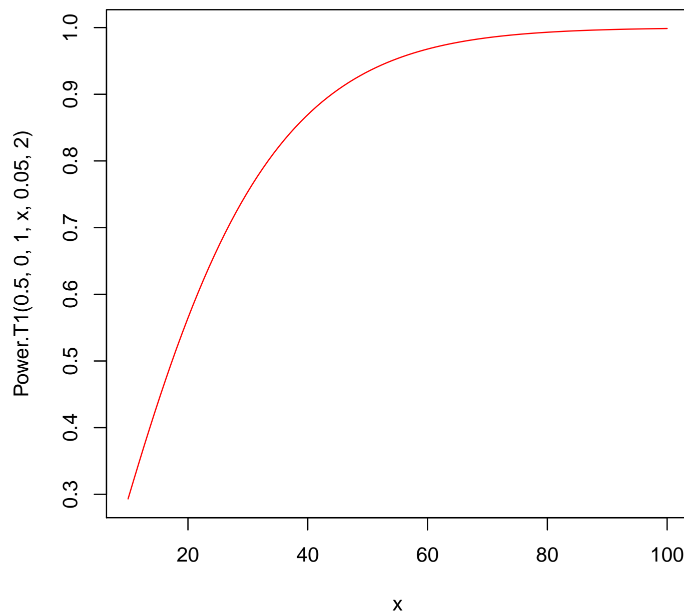
Suppose we load in the t distribution power calculation functions that we used in class. You can load them in with the command `source("Curve Code.txt")`

(the code is on the website with this handout). I show the t-function code below.

```
> ###Generic Function for T Rejection Point
> T.Rejection.Point <- function(alpha,df,tails){
+   if(tails==2)return(qt(1-alpha/2,df))
+   if((tails^2) != 1) return(NA)
+   return(tails*qt(1-alpha,df))
+ }
> ### Generic Function for T-Based Power
> Power.T <- function(delta,df,alpha,tails){
+   pow <- NA
+   R <- T.Rejection.Point(alpha,df,abs(tails))
+   if(tails==1)
+     pow <- 1 - pt(R,df,delta)
+   else if (tails==-1)
+     pow <- pt(R,df,delta)
+   else if (tails==2)
+     pow <-  pt(-R,df,delta) + 1-pt(R,df,delta)
+   return(pow)
+ }
> ### Power Calc for One-Sample T
> Power.T1 <- function(mu,mu0,sigma,n,alpha,tails){
+   delta = sqrt(n)*(mu-mu0)/sigma
+   return(Power.T(delta,n-1,alpha,tails))
+ }
> Power.T2 <- function(mu1,mu2,sigma,n1,n2,alpha,
+                      tails,hypo.diff=0){
+   delta = sqrt((n1*n2)/(n1+n2))*
+     (mu1-mu2-hypo.diff)/sigma
+   return(Power.T(delta,n1+n2-2,alpha,tails))
+ }
> Power.GT <- function(mus,ns,wts,sigma,alpha,
+                       tails,kappa0=0){
+      W = sum(wts^2/ns)
+      kappa = sum(wts*mus)
+      delta = sqrt(1/W) * (kappa-kappa0)/sigma
+      df = sum(ns)-length(ns)
+      return(Power.T(delta,df,alpha,tails))
+  }
```

Now, suppose you wish to plot a curve of power vs. sample size for the 1-sample t test, when $\alpha = 0.05$ and the test is 2-sided, with an $E_s = (\mu - \mu_0)/\sigma = 0.5$. If you write the code below, everything works out nicely.

```
> curve(Power.T1(.5,0,1,x,.05,2),10,100,col="red")
```



Suppose you try the same trick for a Generalized t statistic, testing the hypothesis that $\mu_1 + \mu_2 - 2\mu_3 = 0$, in a situation where $\sigma = 1$, and $\mu_1 = .4, \mu_2 = .4$, and $\mu_3 = .2$. If we assume equal n, and again assume $\alpha = 0.05$ and that the test is 2-sided, we would try this code in order to plot power for values of n from 100 to 1000:

```
curve(Power.GT(c(.4,.4,.2),c(x,x,x),c(1,1,-2),1,0.05,2),100,1000)
```

Unfortunately, you discover that the code "bombs" with the following error message:

```
Error in curve(Power.GT(c(0.4, 0.4, 0.2), c(x, x, x), c(1, 1, -2), 1,  :
  'expr' did not evaluate to an object of length 'n'
```

As is often the case with R, this error message is not very helpful. What is the problem?

If you Google the error message and examine the help file for the `curve` function, you may eventually discover that the `curve` function works as follows:

- It assumes that the function f(x) to be plotted is "vectorizing," that is, if the input x to the function is a vector instead of a single value, the function will return a vector of values of f(x) corresponding to each element in the vector. Most basic functions in R are vectorizing. And simple functions based on these R functions tend to also be vectorizing. For example,

```
> f <- function(x)x^2
> f(1:4)
[1]  1  4  9 16
```

- It creates a vector of input values for x, representing the range of the plot you want to draw.
- It then submits this vector of x values to the function, which (if it is vectorizing) returns a vector of results

It is easy to test whether a function is vectorizing. Input a vector and see what happens! Let's try this with the `Power.GT` function.

```
> x <- 1050:1055
> Power.GT(c(.4,.4,.2),c(x,x,x),c(1,1,-2),1,0.05,2)
[1] 0.5805505
```

Note that only one value was returned. The `Power.GT` function is not vectorizing. It processed only the first input value.

Now try the `Power.T1` function.

```
> x <- 105:110
> Power.T1(.1,0,1,x,0.05,2)
[1] 0.1738668 0.1751010 0.1763353 0.1775698 0.1788045 0.1800393
```

This function is vectorizing.

Looks like we have a problem. What can we do? In what follows, I describe one "quick and easy" solution to the problem.

3 Wrapper Functions

We can't plot power as a function of n using `Power.GT` in the typical way. Two things make our task difficult. First, the function is not vectorizing. Second, the function call is itself very messy. Only one thing is changing over the plot points, i.e., n the sample size. But the call is complex.

To solve this second problem and make things easier, we can try creating a temporary *wrapper function.*

```
> zz <- function(x){
+ Power.GT(c(.4,.4,.2),c(x,x,x),c(1,1,-2),1,0.05,2)
+ }
```

With the use of the `zz` function, if we want power for an n of 1000 in our situation, we just type `zz(1000)` instead of the long messy function call.

After creating our wrapper function, we can use it in conjunction with a new curve-plotting function that I've created for you.

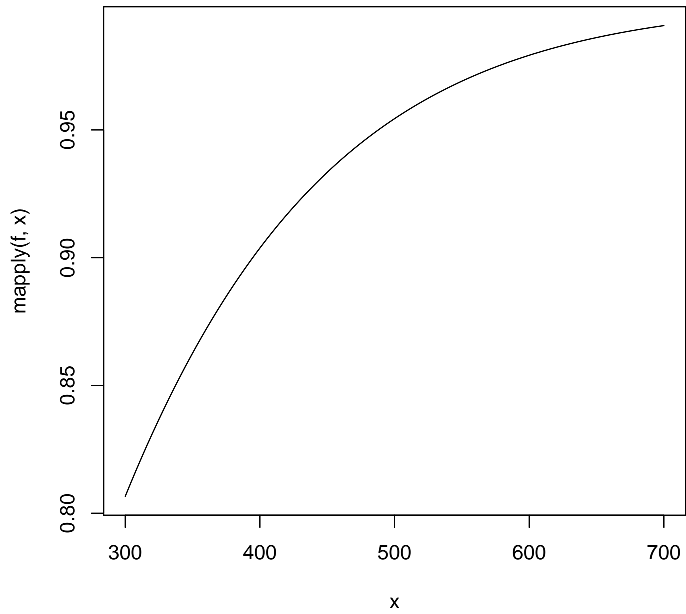
4 New Curve-Plotting Functions

Here are equivalents to the `curve` function that do not require that the function be vectorizing. These functions were in the `Curve Code.txt` file you loaded earlier, so you should not have to type them in.

```
> curve.js <- function(f,a,b,points=100,type='l',...){
+   ftext <- paste("g <- function(x){",f,"}")
+   eval(parse(text=ftext))
+   x <- seq(a,b,length=points)
+   plot(x,mapply(g,x),type,...)
+ }
> plot.curve <- function(f,a,b,points=100,type='l',...){
+   x <- seq(a,b,length=points)
+   plot(x,mapply(f,x),type,...)
+ }
```

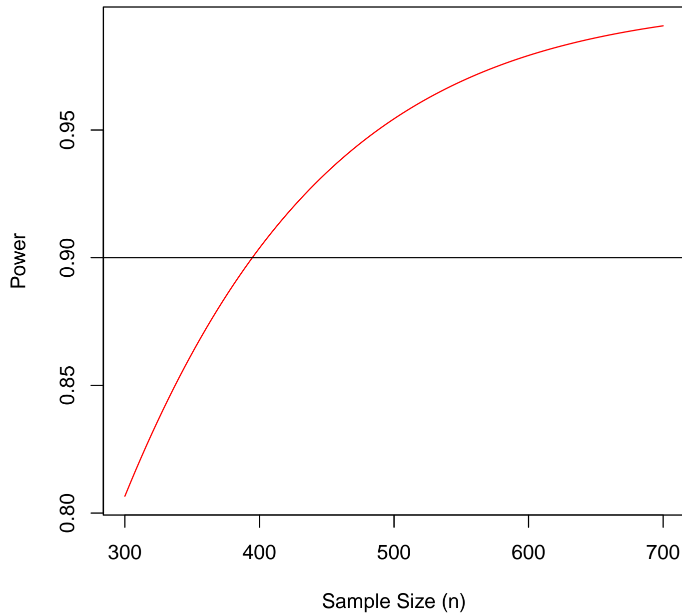
The `plot.curve` function requires as input the name of a function that has a single input variable x. We can easily plot a power curve using our wrapper function.

```
> plot.curve(zz,300,700)
```



You can touch up the graph with a variety of the standard graphics commands. For example,

```
> plot.curve(zz,300,700,col='red',xlab='Sample Size (n)',
+           ylab = 'Power')
> abline(h=.9)
```



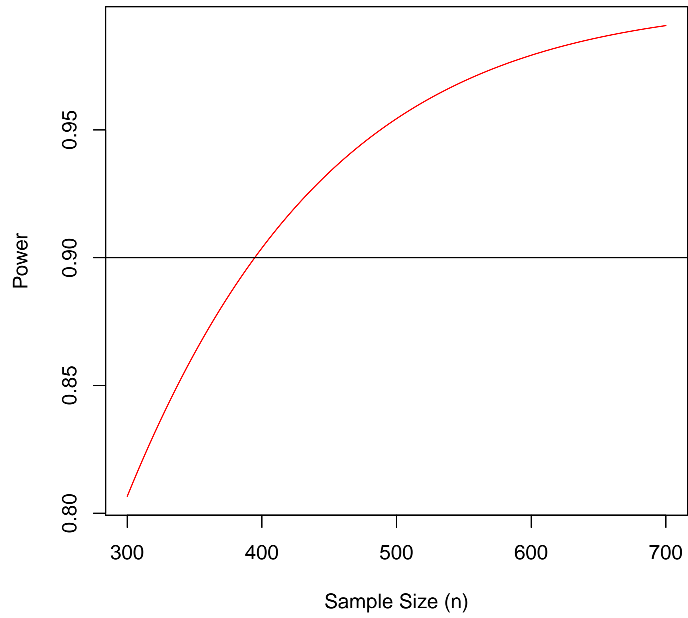
The advantage of using the wrapper function is that if you want to pin down the function value at a particular value of n, you can type it quickly. However, you also have to take the time to write the wrapper function. (All of about 30 seconds once you get the hang of it.)

An alternative is simply to fix R's `curve` function. I took a stab at it – standard disclaimers apply!

The `curve.js` function works like `curve` would work if it handled vectorizing functions. The only diffence is that you have to put the input function in quotes. This function does not require the use of a wrapper function. Study the code for the function above and see if you can figure out what it is doing.

Here is an example using the `curve.js` function.

```
> curve.js("Power.GT(c(.4,.4,.2),c(x,x,x),c(1,1,-2),1,0.05,2)",
+              300,700,col='red',xlab='Sample Size (n)',ylab = 'Power')
> abline(h=.9)
```



You may want to keep this function handy should you run into the problem with `curve` I described.