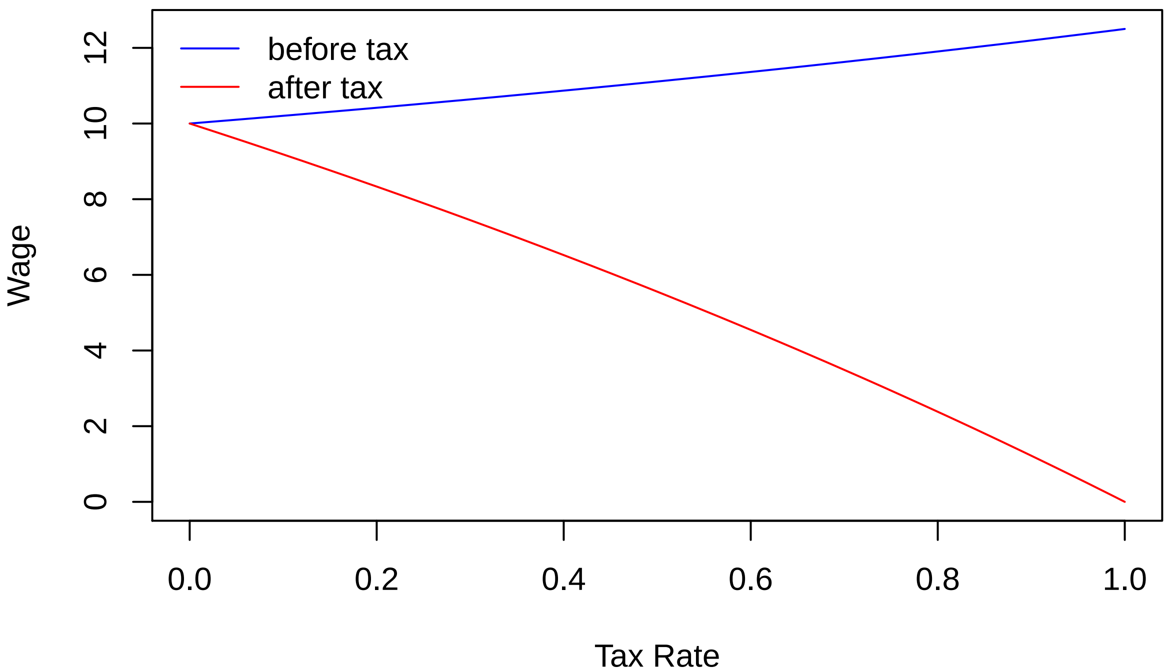
week8

giraffewhale

2017-12-01

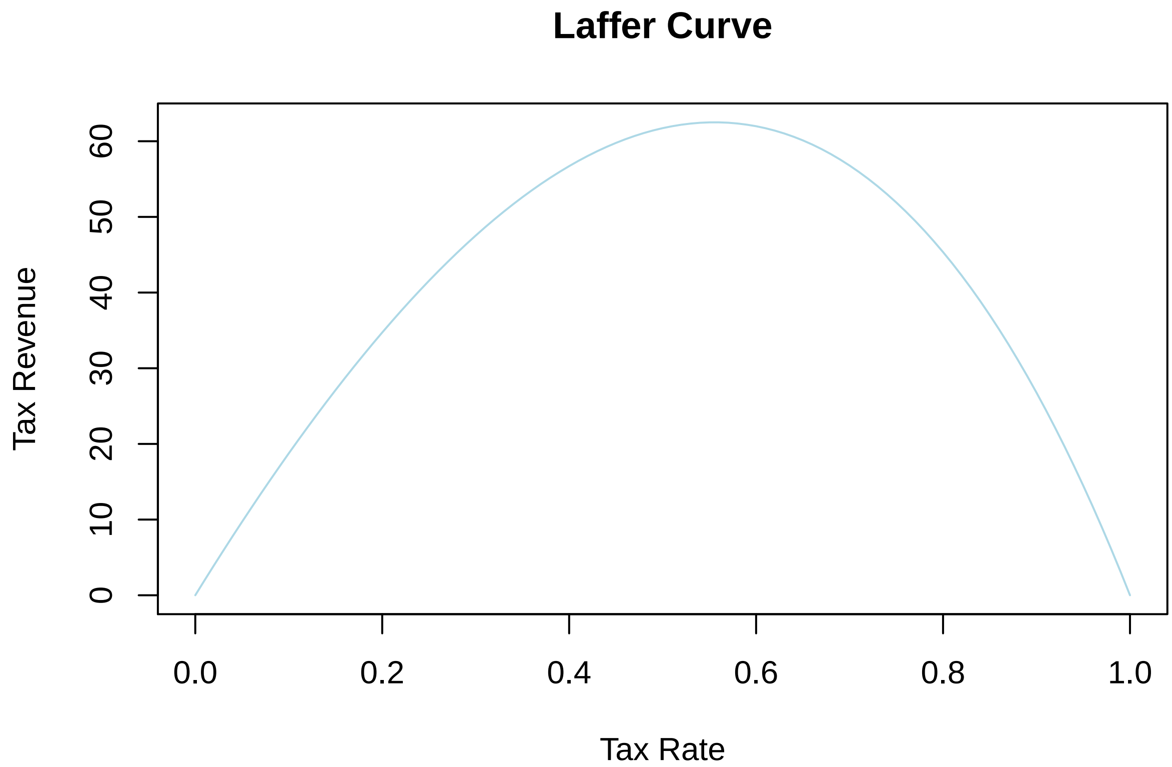
5.3

```
tau <- seq(0,1,0.01)
wb <- 50/(5-tau)
wf <- 50*(1-tau)/(5-tau)
x <- tau
y1 <- wb
y2 <- wf
plot(x,y1,ylim=c(min(y2),max(y1)),type="l",col="blue",xlab = "Tax Rate",ylab = "Wage")
lines(x,y2,col="red")
text.legend=c("before tax","after tax")
col2<-c("blue","red")
legend("topleft",legend=text.legend,lty = c(1,1), col=col2,bty="n",horiz=F)
```



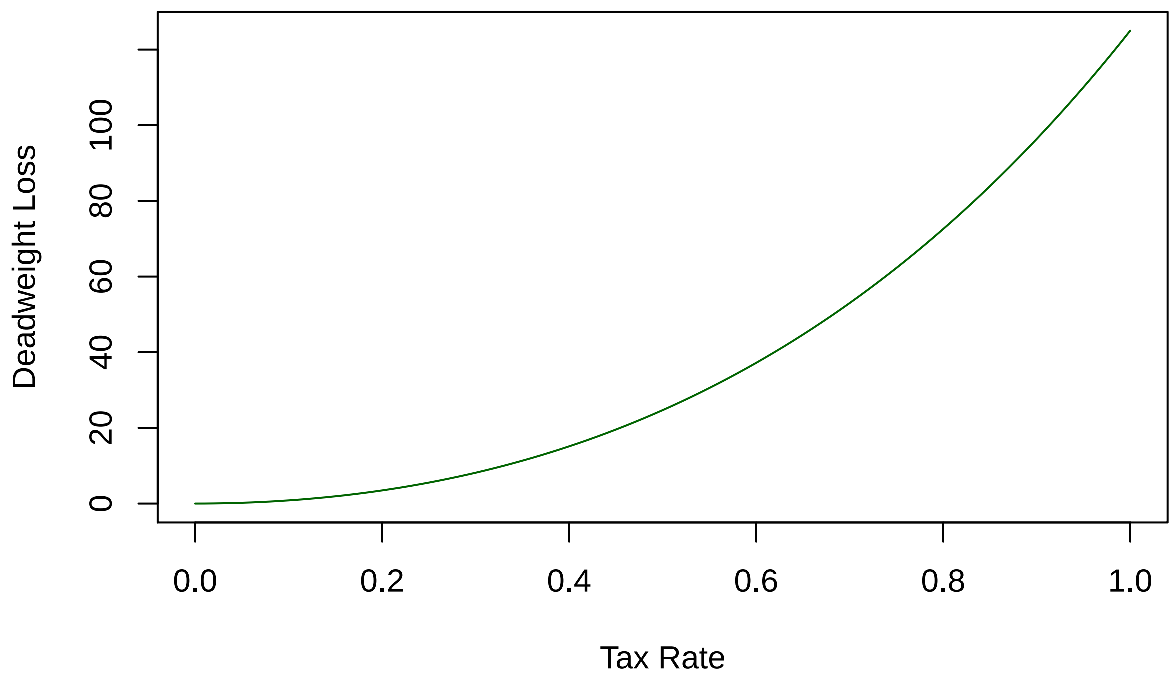
#5.4

```
tr <- tau*(1-tau)*5000/(5-tau)^2
plot(tau,tr,type = "l",col="light blue",xlab = "Tax Rate",ylab = "Tax Revenue",main = "Laffer Curve")
```



#5.7

```
dwl <- 2000*tau^2/(5-tau)^2
plot(tau,dwl,type = "l",col="dark green",xlab = "Tax Rate",ylab = "Deadweight Loss",main = "")
```

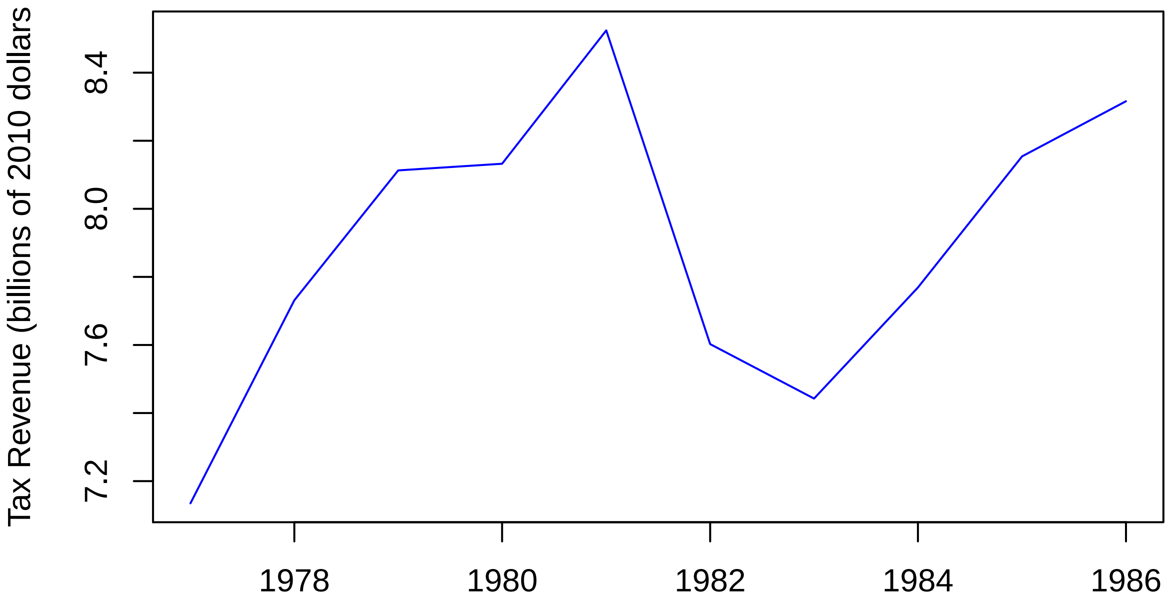


#

5.9&5.10

```
taxreceipt19771986 <- read.csv("taxreceipt19771986.csv")
taxreceipt19972006 <- read.csv("taxreceipt19972006.csv")
GDPdeflator19771986 <- read.csv("GDPdeflator19771986.csv")
GDPdeflator19972006 <- read.csv("GDPdeflator19972006.csv")
```

```
realtaxreceipt19771986 <- taxreceipt19771986$W006RC1A027NBEA/GDPdeflator19771986$USAGDPDEFAISMEI
realtaxreceipt19972006 <- taxreceipt19972006$W006RC1A027NBEA/GDPdeflator19972006$USAGDPDEFAISMEI
timeprd1 <- as.Date(taxreceipt19771986$DATE)
timeprd2 <- as.Date(taxreceipt19972006$DATE)
plot(timeprd1,realtaxreceipt19771986,type="l",col="444",xlab ="",ylab="Tax Revenue (billions of 2010 dol
```




```
plot(timeprd2,realtaxreceipt19972006,type="l",col="666",xlab ="",ylab="Tax Revenue (billions of 2010 dol
```

Optimazation

General-purpose Optimization

Description

General-purpose optimization based on Nelder–Mead, quasi-Newton and conjugate-gradient algorithms. It includes an option for box-constrained optimization and simulated annealing.

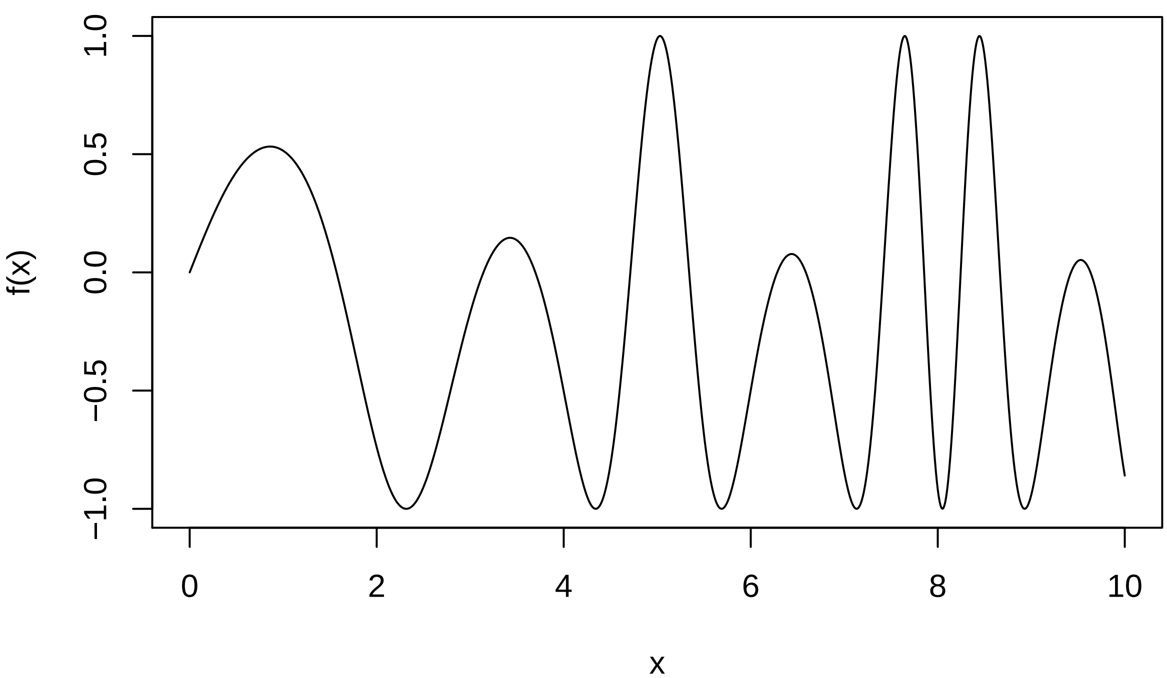
Usage

optim(par, fn, gr = NULL, . . . , method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"), lower = -Inf, upper = Inf, control = list(), hessian = FALSE)

optimHess(par, fn, gr = NULL, . . . , control = list()) Arguments

par:Initial values for the parameters to be optimized over. fn:A function to be minimized (or maximized), with first argument the vector of parameters over which minimization is to take place. It should return a scalar result. gr:A function to return the gradient for the "BFGS", "CG" and "L-BFGS-B" methods. If it is NULL, a finite-difference approximation will be used. For the "SANN" method it specifies a function to generate a new candidate point. If it is NULL a default Gaussian Markov kernel is used. . . . Further arguments to be passed to fn and gr. method: The method to be used. See 'Details'. Can be abbreviated. lower, upper:Bounds on the variables for the "L-BFGS-B" method, or bounds in which to search for method "Brent". control:A list of control parameters. See 'Details'. hessian:Logical. Should a numerically differentiated Hessian matrix be returned?

```
x<-seq(0,10,0.01)
f <- function(x){sin(x*cos(x))}
plot(x,f(x),type = "l")
```




```
optim(2, f)$par
```

```
## Warning in optim(2, f): one-dimensional optimization by Nelder-Mead is unreliable:
## use "Brent" or optimize() directly
```

```
## [1] 2.316016
```

```
optim(4, f)$par
```

```
## Warning in optim(4, f): one-dimensional optimization by Nelder-Mead is unreliable:
## use "Brent" or optimize() directly
```

```
## [1] 4.342236
```

```r
optim(6, f)$par
```

```
## Warning in optim(6, f): one-dimensional optimization by Nelder-Mead is unreliable:
## use "Brent" or optimize() directly

## [1] 5.688647
```

```r
optim(8, f)$par
```

```
## Warning in optim(8, f): one-dimensional optimization by Nelder-Mead is unreliable:
## use "Brent" or optimize() directly

## [1] 7.132227
```

Suppose we have a function:

$$y = 100 - 20x + x^2$$

How do we find x^* that minimizes y?

1. Define a Function

Firts, let us define the function $y(x)$ in `R`. Here's how:

```r
y <- function(x) 100 - 20*x + x^2
```

That's it! We have created the function $y(x)$. Let's test it:

```r
y(5)
```

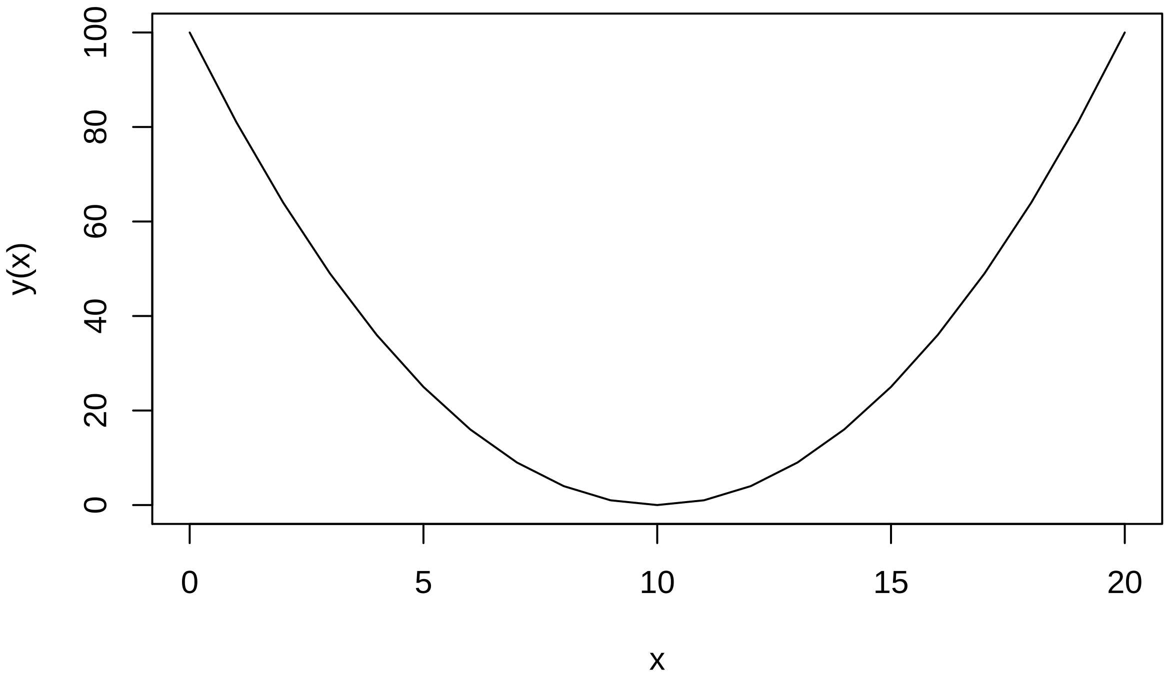
```
## [1] 25
```

Is this correct? Yes:

$$y(5) = 100 - 20 \times 5 + 5^2 = 25$$

Now let's plot this function:

```r
x = 0:20
plot(x,y(x),type="l")
```



2. Finding the Minimum

Now let's find the minimum point of $y(x)$. To do so, we use the function `optim`:

```
x0 = 0 #the initial value of x that we want optim to start searching from.
result <- optim(x0, y) #Optim will start from x0 and then search through many values of x to find the
xstar = result$par #the variable "result"" contains many values. "result$par" is what we want, which is
xstar
```

```
## [1] 9.9
```

That's it! We have found the x^* that minimizes $y(x)$. This x^* is a **numerical solution**, since we have found it by using computer programs to search for the best value. If we calculate by hand, we will get the theoretical solution, or **analytical solution**. Note that in this case, the numerical solution is $x^* = 9.9$, which is close to the analytical solution, which is $x^* = 10$.

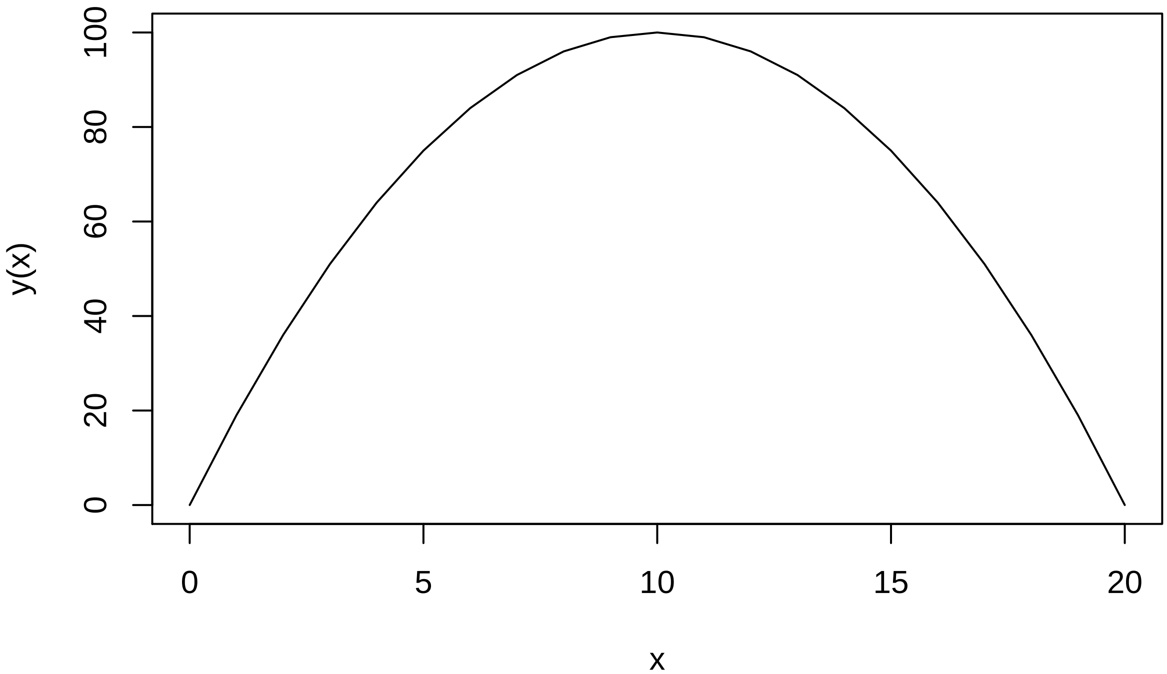
3. Finding the Maximum

`optim` is for finding *minimum* values. How do we find the *maximum* value of a function?

Let's define the following function:

$$y = 10x - x^2$$

```
y <- function(x) 20*x - x^2
x = 0:20
plot(x,y(x),type="l")
```



How do we find the maximum point of $y(x)$? Easy. We can still use `optim`, but let's define another function $z(x) = -y(x)$:

```
z <- function(x) -y(x)
```

Then we can just use the `optim` to find the *minimum* point of $z(x)$, which will be the same as finding the *maximum* point of $y(x)$:

```
x0 = 0
result <- optim(x0, z)
xstar = result$par
xstar
```

```
## [1] 9.9
```

There we go! We have found the x^* that maximizes $y(x)$!

4. Constrained Optimization

In many cases, function $y(x)$ is only defined on $x \in [a, b]$, i.e. x has to be between a and b. In this case, if we are searching for the x^* that minimizes or maximizes $y(x)$, we will only search within the range $[a, b]$. This is called **constrained optimization**.

For example, let

$$y(p) = p(1 - p)$$

, where $p \in [0, 1]$

To find the p^* that minimizes $y(p)$, we again use `optim`, but this time, we add a lowerbound and an upperbound for p.

```
y <- function(p) p*(1-p)
z <- function(p) -y(p) #minimize z = maximize y
result <- optim(0.5, z, lower=0, upper=1) #0.5 is initial value; lower and upper set lowerbound and upp
```

```
pstar = result$par
pstar
```

```
## [1] 0.5
```