Homework 3

Benjamin Ridenhour

05-Oct-21

Contents

In this homework you were asked to do some simple optimization using the five criteria we discussed in class, and apply the five different optimization algorithms available in `optim`. The data and models came from problem 3.4.7 (on pg. 135) which has to do with estimating the weight (W) of a fish given its length (l) and girth (g).

Entering Data and Modifying Them

The data in the book are as follows:

```
fish <- data.frame("l" = c(14.5, 12.5, 17.25, 14.5, 12.625, 17.75, 14.125, 12.625),
                   "g" = c(9.75, 8.375, 11.0, 9.75, 8.5, 12.5, 9.0, 8.5),
                   "W" = c(27, 17, 41, 26, 17, 49, 23, 16))
head(fish)
```

```
##        l      g  W
## 1 14.500  9.750 27
## 2 12.500  8.375 17
## 3 17.250 11.000 41
## 4 14.500  9.750 26
## 5 12.625  8.500 17
## 6 17.750 12.500 49
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

fish = pd.DataFrame({'l': [14.5, 12.5, 17.25, 14.5, 12.625, 17.75, 14.125, 12.625],
                     'g': [9.75, 8.375, 11.0, 9.75, 8.5, 12.5, 9.0, 8.5],
                     'W': [27, 17, 41, 26, 17, 49, 23, 16]})
fish.head()
```

```
##         l       g   W
## 0  14.500   9.750  27
## 1  12.500   8.375  17
## 2  17.250  11.000  41
## 3  14.500   9.750  26
## 4  12.625   8.500  17
```

I asked you to make two modifications to the data. The first task was to add some error to the data using `rnorm`. This is easily accomplished using:

```r
set.seed(68459)
fish_noisy <- data.frame(apply(fish, c(1,2), function(x) rnorm(1,x)))
head(fish_noisy)
```

```
##          l         g        W
## 1 13.09223 10.049867 27.16572
## 2 12.11434  9.618248 16.71103
## 3 16.30117 11.791402 43.38272
## 4 13.69022 10.781787 25.12849
## 5 12.34343  8.837926 17.29332
## 6 18.22708 11.470993 50.15631
```

Note the second argument in `apply` where I specify `c(1,2)`. This tells `apply` that I want to apply my function to each element of the data frame. The call to `rnorm` indicates that I want 1 random with mean x; I did not specify a SD, therefore it uses the default of 1. Here I perform something similar in Python, but instead I simply add a N(0,1) random to each of the values in `fish`.

```python
rng = np.random
rng.seed(68459)
fish_noisy = fish + rng.normal(size=np.prod(fish.size)).reshape((-1,3))
fish_noisy.head()
```

```
##            l          g          W
## 0  12.837478   8.481034  26.386089
## 1  12.080876   6.967741  19.375344
## 2  16.339207  11.273321  41.243811
## 3  15.921797  10.499707  25.203026
## 4  12.329027   8.227127  16.471945
```

The next step of the homework to create a third data set with outliers in the weight variable. These aren't terribly specific instructions; we need to choose how many observations to alter the weight and by how much. The coefficient of variation is defined as the variance of a variable divided by its mean. For weight, the CV is 5.720806. For 3 of the data points, let's add something within ± 2CV.

```r
set.seed(15697)
fish_outlier <- fish_noisy
nChanges <- 3
changeMe  <- sample(1:nrow(fish_outlier), nChanges)
CV <- var(fish_outlier$W)/mean(fish_outlier$W)
fish_outlier$W[changeMe] <- fish_outlier$W[changeMe] + rnorm(nChanges,CV)*sample(c(-1,1),nChanges,repla
fish$W - fish_outlier$W #compare our final W to the original W
```

```
## [1] -0.1657152  0.2889739 -7.9644293  0.8715099  5.0217118 -1.1563084 -1.5374328
## [8] -6.9091701
```

Okay, our data look good and ready for our models.

```python
rng.seed(15697)
fish_outlier = fish_noisy.copy()
nChanges = 3
changeMe = rng.choice(len(fish.index), nChanges, replace = False)
CV = np.var(fish_outlier.W)/np.mean(fish_outlier.W)
fish_outlier.W[changeMe] = fish_outlier.W[changeMe] + rng.normal(CV,size=nChanges)*rng.choice([-1,1],nCh
fish.W - fish_outlier.W
```

```
## 0    0.613911
## 1   -7.385830
```

```
## 2   -0.243811
## 3    0.796974
## 4    0.528055
## 5    5.980621
## 6    1.287321
## 7   -0.551332
## Name: W, dtype: float64
```

Optimization with Different Criteria

Okay, first we need to grab the code from the lecture that has the different criteria.

```
chebyshev <- function(y, yhat) max(abs(y - yhat))
meanAD <- function(y, yhat) mean(abs(y - yhat))
medianAD <- function(y, yhat) median(abs(y - yhat))
leastSq <- function(y, yhat) mean(abs(y - yhat)^2)
llnorm <- function(y, yhat) sum(dnorm(y, yhat, log = T))
```

```
import scipy.stats as stats

def chebyshev(y, yhat):
  return(np.max(np.abs(y - yhat)))

def meanAD(y, yhat):
  return(np.mean(np.abs(y - yhat)))

def medianAD(y, yhat):
  return(np.median(np.abs(y - yhat)))

def leastSq(y, yhat):
  return(np.mean((y-yhat)**2))

def llnorm(y, yhat):
  return(np.sum(stats.norm.logpdf(yhat,y)))
```

We also need the two different models, $W = kl^3$ and $W = klg^2$. Let's write objective functions for these two models:

```
## for W = k l ^3
objOne <- function(k, data, dist = "chebyshev"){
  with(data,{
    W_hat <- k*l^3 #calculate the value of the first model for some parameter
    eval(call(dist, W, W_hat)) #use the distance function specified
  })
}

## for W = k l g ^2
objTwo <- function(k, data, dist = "chebyshev"){
  with(data,{
    W_hat <- k*l*g^2 #calculate the value of the first model for some parameter
    eval(call(dist, W, W_hat)) #use the distance function specified
  })
}

###Try it out
objOne(0.5,fish)
```

```
## [1] 2747.18
```

```
objTwo(0.5,fish)
```

```
## [1] 1337.719
```

```
def objOne(k, data, dist=chebyshev, sign = 1):
  W_hat = k * np.power(data.l,3) #note data should be a pandas data frame with columns named l and W
  return(sign*dist(data.W, W_hat))

def objTwo(k, data, dist=chebyshev, sign = 1):
  W_hat = k * data.l * np.power(data.g,2) #note data should be a pandas data frame with columns named l
  return(sign*dist(data.W, W_hat))

objOne(0.5, fish)
```

```
## 2747.1796875
```

```
objTwo(0.5, fish)
```

```
## 1337.71875
```

It looks like the objective functions are ready for use in optimization. The last thing we need for optimization is a guess at the parameter value. Let's just assume that the average solution for W is a good starting point.

```
#Model one guesses
guess_1 <- with(fish, mean(W / l^3))
guess_1_n <- with(fish_noisy, mean(W / l^3))
guess_1_o <- with(fish_outlier, mean(W / l^3))

#Model two guesses
guess_2 <- with(fish, mean(W / (l*g^2)))
guess_2_n <- with(fish_noisy, mean(W / (l*g^2)))
guess_2_o <- with(fish_outlier, mean(W / (l*g^2)))

c(guess_1,guess_1_n,guess_1_o)
```

```
## [1] 0.008424862 0.009415505 0.009465100
```

```
c(guess_2,guess_2_n,guess_2_o)
```

```
## [1] 0.01892890 0.01802916 0.01846227
```

```
#Model one guesses
guess_1 = np.mean(fish.W/np.power(fish.l,3))
guess_1_n = np.mean(fish_noisy.W/np.power(fish_noisy.l,3))
guess_1_o = np.mean(fish_outlier.W/np.power(fish_outlier.l,3))

#Model two guesses
guess_2 = np.mean(fish.W/(np.power(fish.g,2)*fish.l))
guess_2_n = np.mean(fish_noisy.W/(np.power(fish_noisy.g,2)*fish_noisy.l))
guess_2_o = np.mean(fish_outlier.W/(np.power(fish_outlier.g,2)*fish_outlier.l))

np.array([guess_1,guess_1_n,guess_1_o])
```

```
## array([0.00842486, 0.00854243, 0.00870341])
```

```
np.array([guess_2,guess_2_n,guess_2_o])
```

```
## array([0.0189289 , 0.02114615, 0.02173397])
```

It looks like the guesses don't vary much between the data sets, so we can just use one of the guesses for each model (~ 0.009 and ~ 0.018 for models 1 and 2, respectively). We can now perform optimization using the five criteria and the three data sets.

```
results <- data.frame()
critList <- c("chebyshev","meanAD","medianAD","leastSq","llnorm")
dataList <- c("fish","fish_noisy","fish_outlier")
for(i in critList){
  for(j in dataList){
    clist <- if(i == "llnorm") list(fnscale = -1) else list()
    newResult <- data.frame(optim(guess_1, objOne, data = get(j), dist = i, control = clist)[c("par","va
    newResult$Criteria <- i
    newResult$Data <- j
    results <- rbind(results, newResult)
  }
}
```

```
## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly
```

```
## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_1, objOne, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly
```

```
results
```

```
##            par       value convergence  Criteria         Data
## 1  0.008391364    2.072478           0 chebyshev         fish
## 2  0.008432437    8.242546           0 chebyshev   fish_noisy
## 3  0.009542590    7.628999           0 chebyshev fish_outlier
## 4  0.008528435    0.972877           0    meanAD         fish
## 5  0.009399487    3.913939           0    meanAD   fish_noisy
## 6  0.009399487    4.549453           0    meanAD fish_outlier
## 7  0.008688139    0.500000           0  medianAD         fish
## 8  0.010207819    1.670563           0  medianAD   fish_noisy
## 9  0.008623113    3.620832           0  medianAD fish_outlier
## 10 0.008436740    1.521043           0   leastSq         fish
## 11 0.008944423   25.954591           0   leastSq   fish_noisy
## 12 0.009304372   27.811641           0   leastSq fish_outlier
## 13 0.008436740  -13.435679           0    llnorm         fish
## 14 0.008944423 -111.169871           0    llnorm   fish_noisy
## 15 0.009304372 -118.598071           0    llnorm fish_outlier
```

```
### repeat for model 2

results2 <- data.frame()
for(i in critList){
  for(j in dataList){
    clist <- if(i == "llnorm") list(fnscale = -1) else list()
    newResult <- data.frame(optim(guess_2, objTwo, data = get(j), dist = i, control = clist)[c("par","va
    newResult$Criteria <- i
    newResult$Data <- j
    results2 <- rbind(results2, newResult)
  }
}
```

```
## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly
```

```
## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly

## Warning in optim(guess_2, objTwo, data = get(j), dist = i, control = clist): one-dimensional optimiza
## use "Brent" or optimize() directly
```

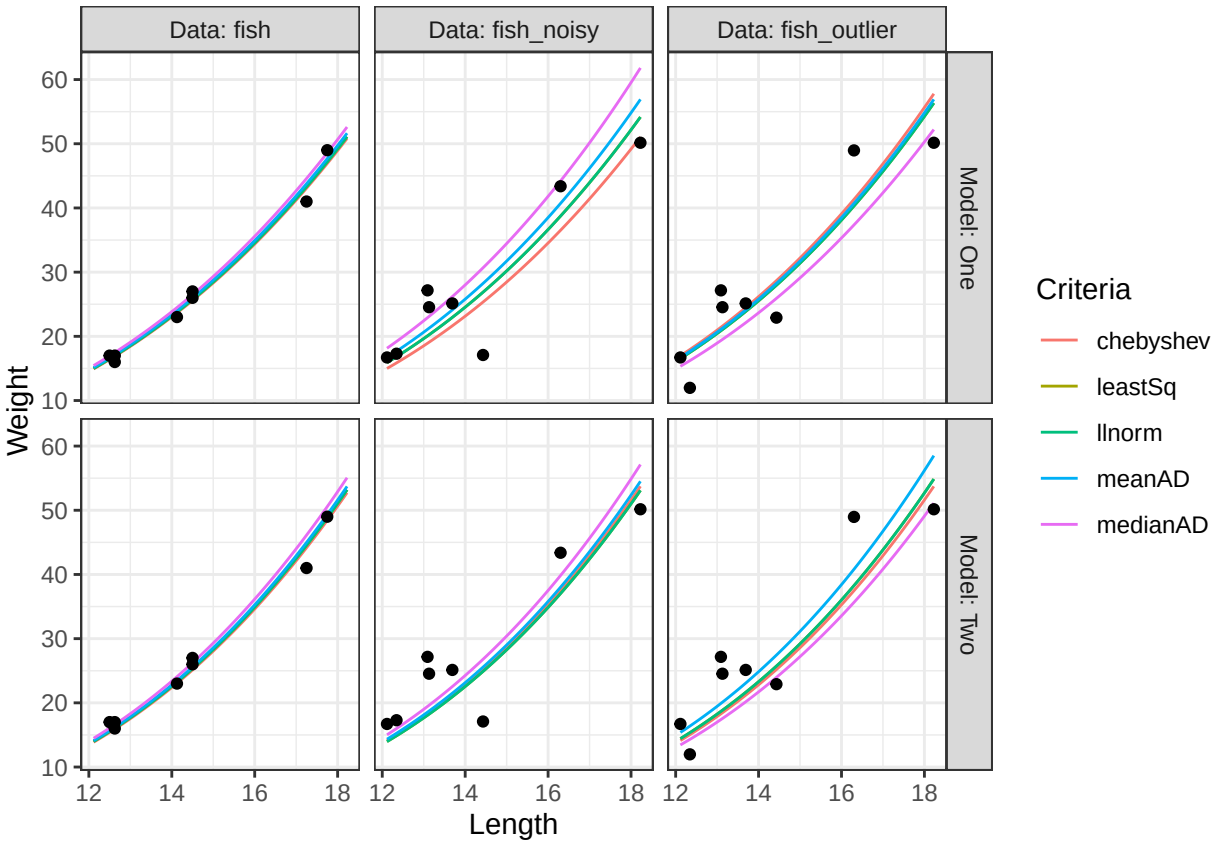
```r
results2
```

```
##           par        value convergence  Criteria         Data
## 1  0.01851590    2.3526893           0 chebyshev         fish
## 2  0.01886917    4.9007619           0 chebyshev   fish_noisy
## 3  0.01886421    6.2093066           0 chebyshev fish_outlier
## 4  0.01886236    1.1544172           0    meanAD         fish
## 5  0.01914114    2.8099014           0    meanAD   fish_noisy
## 6  0.02054409    4.6085680           0    meanAD fish_outlier
## 7  0.01933716    0.6464873           0  medianAD         fish
## 8  0.02005926    2.0637018           0  medianAD   fish_noisy
## 9  0.01794734    4.3795216           0  medianAD fish_outlier
## 10 0.01867508    2.2088872           0   leastSq         fish
## 11 0.01864492   11.8662431           0   leastSq   fish_noisy
## 12 0.01926348   25.6792581           0   leastSq fish_outlier
## 13 0.01867519  -16.1870570           0    llnorm         fish
## 14 0.01864515  -54.8164810           0    llnorm   fish_noisy
## 15 0.01926348 -110.0685406           0    llnorm fish_outlier
```

The last thing you were asked to do is plot the different solutions from the different criteria. First we need to create the data to plot, then pass those data to `ggplot`.

```r
modelOne <- function(l, k) k * l^3
modelTwo <- function(l, g, k) k * l * g^2

raw <- rbind(fish,fish_noisy, fish_outlier)
```

```
raw$Data <- rep(dataList, each = nrow(fish))

lVals <- seq(min(raw$l), max(raw$l), length=100)
gVals <- seq(min(raw$g), max(raw$g), length=100)


plotData <- data.frame()
for(i in critList){
  for(j in dataList){
    p1 <- subset(results, Data == j & Criteria == i)$par
    p2 <- subset(results2, Data == j & Criteria == i)$par
    plotData <- rbind(plotData,
                      data.frame(l=lVals, Criteria = i, Data = j,
                                 W = c(modelOne(lVals, p1), modelTwo(lVals,gVals,p2)),
                                 Model = rep(c("One","Two"), each = length(lVals)))
                      )
  }
}

library(ggplot2)
g <- ggplot(plotData, aes(x = l, y = W))
g + geom_line(aes(color=Criteria)) + geom_point(data = raw) +
  facet_grid(rows = vars(Model), cols= vars(Data), labeller = label_both) +
  theme_bw() + xlab("Length") + ylab("Weight")
```



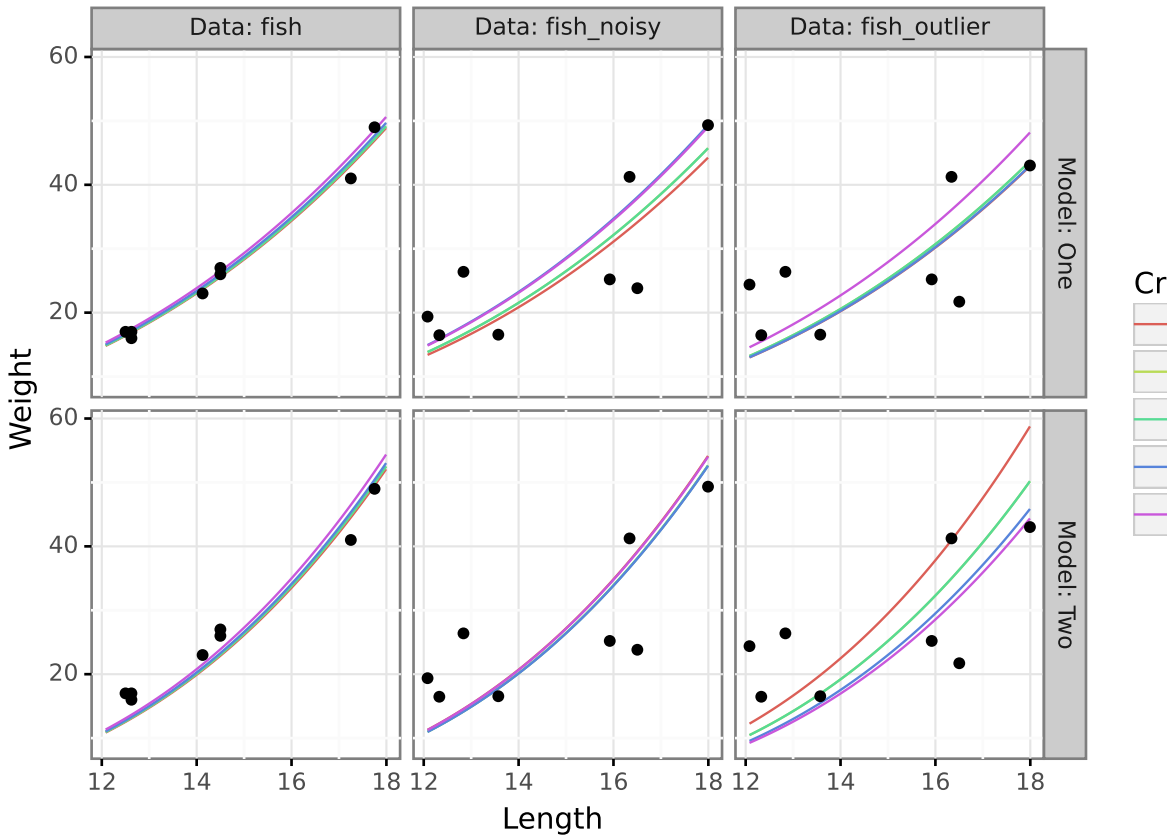
Doing all the above using Python instead:

```
import scipy.optimize as optim

results = pd.DataFrame(columns = ['Criteria', 'data','par','value','convergence'])

critList = [chebyshev, meanAD, medianAD, leastSq, llnorm]
dataList = {'fish':fish, 'fish_noisy':fish_noisy, 'fish_outlier':fish_outlier}

for i in critList:
  #iterating over a dictionary where j = keys and k = values
  for j, k in dataList.items():
    scaleVal = 1 if i.__name__ != 'llnorm' else -1
    newResult = optim.minimize(objOne, guess_1, args=(k,i,scaleVal), method = 'Nelder-Mead')
    results = results.append({'Criteria': i.__name__, 'data': j, 'par': newResult.x[0], 'value': newResu
                              'convergence': newResult.status },ignore_index=True)


results2 = pd.DataFrame(columns = ['Criteria', 'data','par','value','convergence'])

for i in critList:
  #iterating over a dictionary where j = keys and k = values
  for j, k in dataList.items():
    scaleVal = 1 if i.__name__ != 'llnorm' else -1
    newResult = optim.minimize(objTwo, guess_1, args=(k,i,scaleVal), method = 'Nelder-Mead')
    results2 = results2.append({'Criteria': i.__name__, 'data': j, 'par': newResult.x[0], 'value': newR
                              'convergence': newResult.status },ignore_index=True)
```

```
def modelOne(l,k): return k * np.power(l,3)
def modelTwo(l,g,k):  return k * l * np.power(g,2)

raw = pd.concat([fish,fish_noisy,fish_outlier])
raw['Data'] = np.repeat(list(dataList.keys()),len(fish.index))

lVals = np.linspace(np.min(raw.l), np.max(raw.l), 100)
gVals = np.linspace(np.min(raw.g), np.max(raw.g), 100)

plotData = pd.DataFrame(columns = ['l','Criteria','Data','W','Model'])

for i in critList:
  for j,k in dataList.items():
    p1 = results[(results.Criteria == i.__name__) & (results.data == j)].par.item()
    p2 = results2[(results.Criteria == i.__name__) & (results.data == j)].par.item()
    plotData = plotData.append(pd.DataFrame({'l':lVals, 'Criteria': i.__name__, 'Data': j, 'W': modelOne
    plotData = plotData.append(pd.DataFrame({'l':lVals, 'Criteria': i.__name__, 'Data': j, 'W': modelTwo


#plotnine adds ggplot functionality to python

import plotnine as p9

(
  p9.ggplot(plotData) + p9.aes(x = 'l', y = 'W')
   + p9.geom_line(p9.aes(color='Criteria')) + p9.geom_point(data=raw)
   + p9.facet_grid('Model ~ Data', labeller = p9.label_both)
```

```
   + p9.theme_bw() + p9.xlab("Length") + p9.ylab("Weight")
)
```

```
## <ggplot: (628946685)>
```



Testing Different `optim` Algorithms

The final part of the homework was to compare the various methods available to `optim` or `minimize` to perform the optimization. You were asked to make a table of the results. I will only do this for R, however, to do it in, Python, you can simply follow the same logic: Make another loop that goes over the various `minimize` algorithms (listed here).

```
algos <- c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN","Brent")

results3 <- data.frame()
for(i in critList){
  for(j in dataList){
    for(k in algos){
      clist <- if(i == "llnorm") list(fnscale = -1) else list()
      newResult <- suppressWarnings(data.frame(optim(guess_1, objOne, data = get(j), dist = i,
                                    control = clist, method = k, lower = 0, upper = 1)[c("value")]))
      newResult$Criteria <- i
      newResult$Data <- j
      newResult$Algorithm <- k
      results3 <- rbind(results3, newResult)
```

```
    }
  }
}

xtabs(value ~ Criteria + Algorithm + Data, data = results3)
```

```
## , , Data = fish
##
##            Algorithm
## Criteria            BFGS        Brent           CG     L-BFGS-B  Nelder-Mead
##   chebyshev    2.2444216    2.0724845    2.2444216    2.2444216    2.2444216
##   leastSq      1.5210427    1.5210427    1.5210427    1.5210427    1.5210427
##   llnorm     -13.4356792  -13.4356792  -13.4356792  -13.4356792  -13.4356792
##   meanAD       1.0014416    0.9728772    1.0014416    1.0014416    1.0014416
##   medianAD     0.7860166    0.5000000    0.7860166    0.7860166    0.7860166
##            Algorithm
## Criteria            SANN
##   chebyshev    2.2444216
##   leastSq      1.5210427
##   llnorm     -13.4356792
##   meanAD       1.0014416
##   medianAD     0.7860166
##
## , , Data = fish_noisy
##
##            Algorithm
## Criteria            BFGS        Brent           CG     L-BFGS-B  Nelder-Mead
##   chebyshev    8.2595448    8.2425454    8.2595448    8.2595448    8.2595448
##   leastSq     25.9545905   25.9545905   25.9545905   25.9545905   25.9545905
##   llnorm    -111.1698703 -111.1698703 -111.1698703 -111.1698703 -111.1698703
##   meanAD       3.9424201    3.9139390    3.9424201    3.9424201    3.9424201
##   medianAD     1.6743069    1.6705635    1.6743069    1.6743069    1.6743069
##            Algorithm
## Criteria            SANN
##   chebyshev    8.2595448
##   leastSq     25.9545905
##   llnorm    -111.1698703
##   meanAD       3.9424201
##   medianAD     1.6743069
##
## , , Data = fish_outlier
##
##            Algorithm
## Criteria            BFGS        Brent           CG     L-BFGS-B  Nelder-Mead
##   chebyshev    8.3105649    7.6290023    8.3105649    8.3105649    8.3105649
##   leastSq     27.8116406   27.8116406   27.8116406   27.8116406   27.8116406
##   llnorm    -118.5980705 -118.5980705 -118.5980705 -118.5980705 -118.5980705
##   meanAD       4.5867898    4.5494531    4.5867898    4.5867898    4.5867898
##   medianAD     3.6575449    3.6208316    3.6575449    3.6575449    3.6575449
##            Algorithm
## Criteria            SANN
##   chebyshev    8.3105649
##   leastSq     27.8116406
##   llnorm    -118.5980705
```

```
##   meanAD       4.5867898
##   medianAD     3.6575449
```

And for model 2:

```
### model 2

results4 <- data.frame()
for(i in critList){
  for(j in dataList){
    for(k in algos){
      clist <- if(i == "llnorm") list(fnscale = -1) else list()
      newResult <- suppressWarnings(data.frame(optim(guess_2, objTwo, data = get(j), dist = i,
                                    control = clist, method = k, lower = 0, upper = 1)[c("value")]))
      newResult$Criteria <- i
      newResult$Data <- j
      newResult$Algorithm <- k
      results4 <- rbind(results4, newResult)
    }
  }
}

xtabs(value ~ Criteria + Algorithm + Data, data = results4)
```

```
## , , Data = fish
##
##            Algorithm
## Criteria            BFGS        Brent           CG     L-BFGS-B  Nelder-Mead
##   chebyshev    2.4044291    2.3526947    2.4044291    2.4044291    2.4044291
##   leastSq      2.2088872    2.2088872    2.2088872    2.2088872    2.2088872
##   llnorm     -16.1870569  -16.1870569  -16.1870569  -16.1870569  -16.1870569
##   meanAD       1.1575997    1.1544174    1.1575997    1.1575997    1.1575997
##   medianAD     0.6779507    0.7042833    0.6779507    0.6779507    0.6779507
##            Algorithm
## Criteria            SANN
##   chebyshev    2.4044291
##   leastSq      2.2088872
##   llnorm     -16.1870569
##   meanAD       1.1575997
##   medianAD     0.6779507
##
## , , Data = fish_noisy
##
##            Algorithm
## Criteria            BFGS        Brent           CG     L-BFGS-B  Nelder-Mead
##   chebyshev    4.9958285    4.9007625    4.9958285    4.9958285    4.9958285
##   leastSq     11.8662431   11.8662431   11.8662431   11.8662431   11.8662431
##   llnorm     -54.8164808  -54.8164808  -54.8164808  -54.8164808  -54.8164808
##   meanAD       2.8537584    2.8098940    2.8537584    2.8537584    2.8537584
##   medianAD     2.0858739    2.0562112    2.0858739    2.0858739    2.0858739
##            Algorithm
## Criteria            SANN
##   chebyshev    4.9958285
##   leastSq     11.8662431
##   llnorm     -54.8164808
```

```
##   meanAD       2.8537584
##   medianAD     2.0858739
##
## , , Data = fish_outlier
##
##            Algorithm
## Criteria            BFGS        Brent           CG     L-BFGS-B  Nelder-Mead
##   chebyshev    6.2716805    6.2093096    6.2716805    6.2716805    6.2716805
##   leastSq     25.6792581   25.6792581   25.6792581   25.6792581   25.6792581
##   llnorm    -110.0685405 -110.0685405 -110.0685405 -110.0685405 -110.0685405
##   meanAD       4.6743136    4.6085683    4.6743136    4.6743136    4.6743136
##   medianAD     4.5145817    4.3795226    4.5145817    4.5145817    4.5145817
##            Algorithm
## Criteria            SANN
##   chebyshev    6.2716805
##   leastSq     25.6792581
##   llnorm    -110.0685405
##   meanAD       4.6743136
##   medianAD     4.5145817
```

One thing that jumps out looking at both models and all of our criteria is that the "Brent" method is always the best algorithm for our problem! This is because the Brent algorithm is specialized for 1-D optimization, like we are doing here. *Python has a separate function for 1-D optimization called* ***brent****.* Let's redo our results using Brent and plot them:

```
results <- data.frame()
critList <- c("chebyshev","meanAD","medianAD","leastSq","llnorm")
dataList <- c("fish","fish_noisy","fish_outlier")
for(i in critList){
  for(j in dataList){
    clist <- if(i == "llnorm") list(fnscale = -1) else list()
    newResult <- data.frame(optim(guess_1, objOne, data = get(j), dist = i,
                                  control = clist, method = "Brent", lower = 0, upper = 1)[c("par","valu
    newResult$Criteria <- i
    newResult$Data <- j
    results <- rbind(results, newResult)
  }
}
results
```

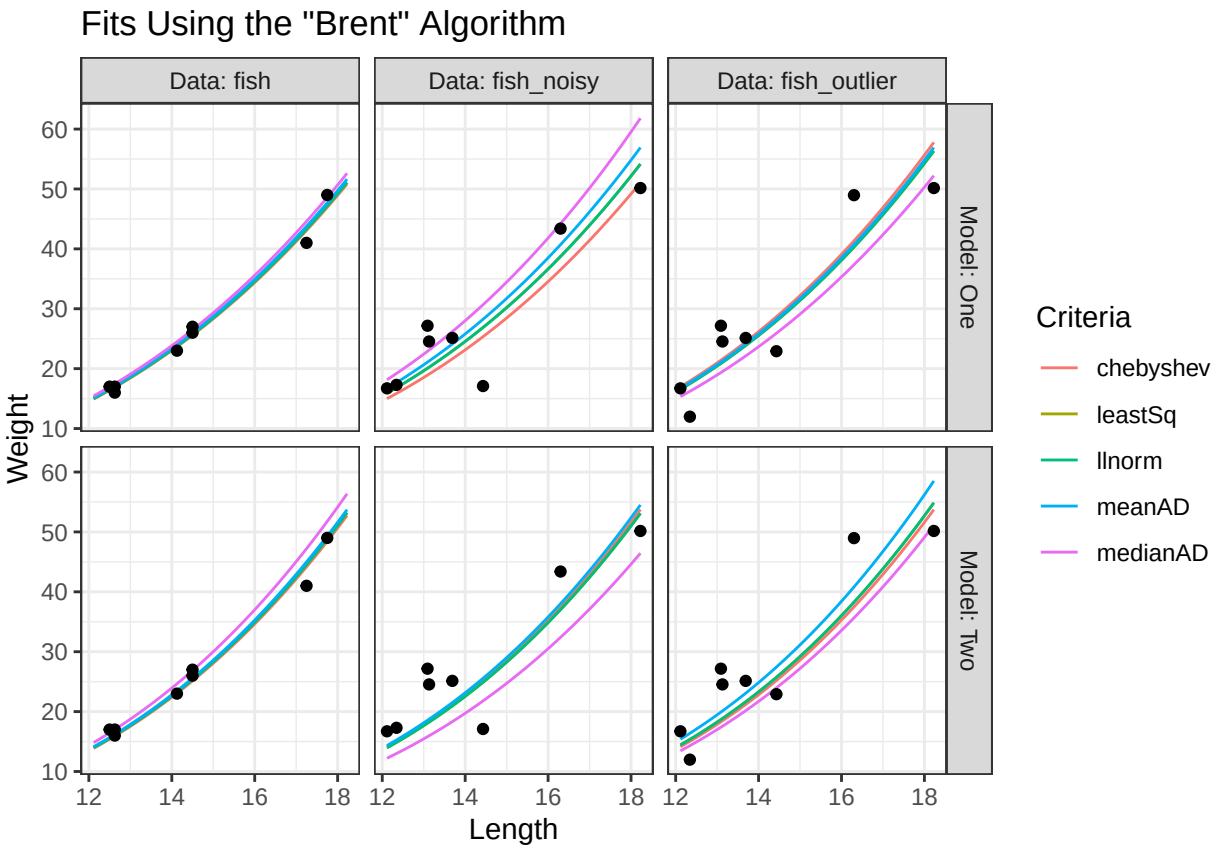
```
##            par        value convergence  Criteria         Data
## 1  0.008391363    2.0724845           0 chebyshev         fish
## 2  0.008432438    8.2425454           0 chebyshev   fish_noisy
## 3  0.009542589    7.6290023           0 chebyshev fish_outlier
## 4  0.008528434    0.9728772           0    meanAD         fish
## 5  0.009399487    3.9139390           0    meanAD   fish_noisy
## 6  0.009399486    4.5494531           0    meanAD fish_outlier
## 7  0.008688584    0.5000000           0  medianAD         fish
## 8  0.010207816    1.6705635           0  medianAD   fish_noisy
## 9  0.008623113    3.6208316           0  medianAD fish_outlier
## 10 0.008436761    1.5210427           0   leastSq         fish
## 11 0.008944351   25.9545905           0   leastSq   fish_noisy
## 12 0.009304406   27.8116406           0   leastSq fish_outlier
## 13 0.008436761  -13.4356792           0    llnorm         fish
## 14 0.008944351 -111.1698703           0    llnorm   fish_noisy
```

```
## 15 0.009304406 -118.5980705           0    llnorm fish_outlier
```

```
### repeat for model 2

results2 <- data.frame()
for(i in critList){
  for(j in dataList){
    clist <- if(i == "llnorm") list(fnscale = -1) else list()
    newResult <- data.frame(optim(guess_2, objTwo, data = get(j), dist = i,
                                  control = clist, method = "Brent", lower = 0, upper = 1)[c("par","valu
    newResult$Criteria <- i
    newResult$Data <- j
    results2 <- rbind(results2, newResult)
  }
}
results2
```

```
##           par        value convergence  Criteria         Data
## 1  0.01851590    2.3526947           0 chebyshev         fish
## 2  0.01886917    4.9007625           0 chebyshev   fish_noisy
## 3  0.01886421    6.2093096           0 chebyshev fish_outlier
## 4  0.01886237    1.1544174           0    meanAD         fish
## 5  0.01914111    2.8098940           0    meanAD   fish_noisy
## 6  0.02054409    4.6085683           0    meanAD fish_outlier
## 7  0.01979333    0.7042833           0  medianAD         fish
## 8  0.01631031    2.0562112           0  medianAD   fish_noisy
## 9  0.01794733    4.3795226           0  medianAD fish_outlier
## 10 0.01867511    2.2088872           0   leastSq         fish
## 11 0.01864500   11.8662431           0   leastSq   fish_noisy
## 12 0.01926353   25.6792581           0   leastSq fish_outlier
## 13 0.01867511  -16.1870569           0    llnorm         fish
## 14 0.01864500  -54.8164808           0    llnorm   fish_noisy
## 15 0.01926353 -110.0685405           0    llnorm fish_outlier
```

```
plotData <- data.frame()
for(i in critList){
  for(j in dataList){
    p1 <- subset(results, Data == j & Criteria == i)$par
    p2 <- subset(results2, Data == j & Criteria == i)$par
    plotData <- rbind(plotData,
                      data.frame(l=lVals, Criteria = i, Data = j,
                                 W = c(modelOne(lVals, p1), modelTwo(lVals,gVals,p2)),
                                 Model = rep(c("One","Two"), each = length(lVals)))
                      )
  }
}

g <- ggplot(plotData, aes(x = l, y = W))
g + geom_line(aes(color=Criteria)) + geom_point(data = raw) +
  facet_grid(rows = vars(Model), cols= vars(Data), labeller = label_both) +
  theme_bw() + xlab("Length") + ylab("Weight") + ggtitle("Fits Using the \"Brent\" Algorithm")
```



Who is the Winner?

This is a bit of challenge to answer because we do not know the truth. Clearly, regardless of the criteria, data set, or model, we should be using the Brent algorithm for our 1-D optimization. That leaves a choice of which of the criteria seemed to be best. Least squares and the normal log-likelihood produce identical estimates, so they're interchangeable. I would argue that the mean AD is more robust (changes less) with the noisier data and outliers, and therefore might be the best choice. However, depending on your purpose, you might choose something else!