Fitting Piecewise Regression by Genetic Algorithms

Jiahui Li

1 Introduction

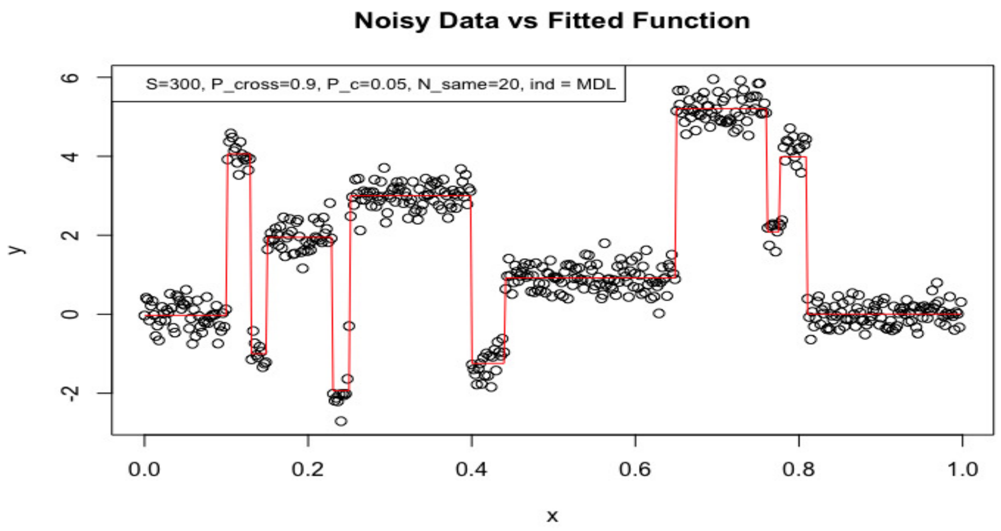
In this report, we will first develop automatic procedures for fitting piecewise constant regression using genetic algorithms. Then, we will research how the following parameters affect the behavior of algorithms in this problem.

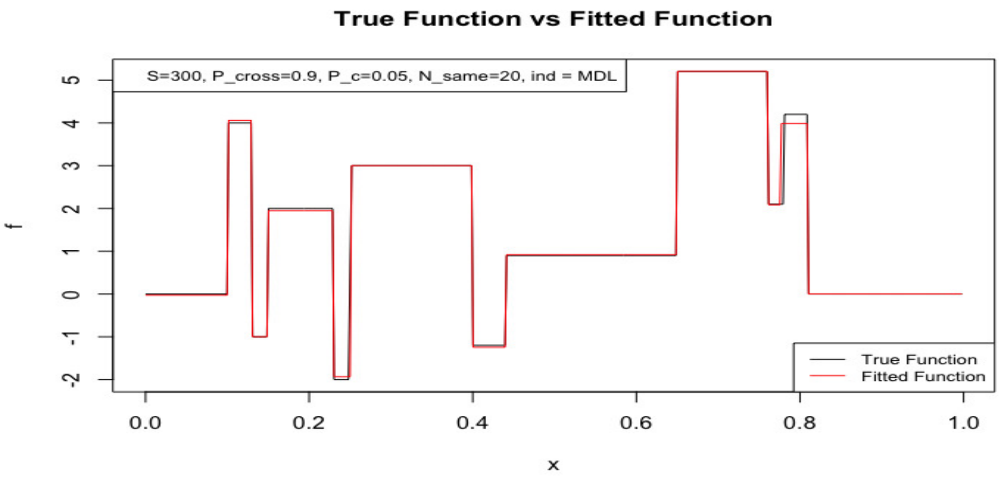
- The size of chromosomes, denoted by S.
- The probability of crossover, denoted by P_{cross}.
- The probability of mutation, denoted by P_c.
- The stopping criterion: stop when how many generations maintain the same best chromosome, denoted by N_{name}.
- Method for estimating: the minimum description length (MDL) principle or the Akaike information criterion (AIC).

2 Method

With S= 300, $P_{cross} = 0.9$, P_c=0.05, $N_{same} = 20$, and select the model with minimum MDL, after 107 iterations, we can find the "best" model with MDL = 124.6169.

We plot the noisy data vs the fitting piecewise constant function, and the true function vs the fitting function:

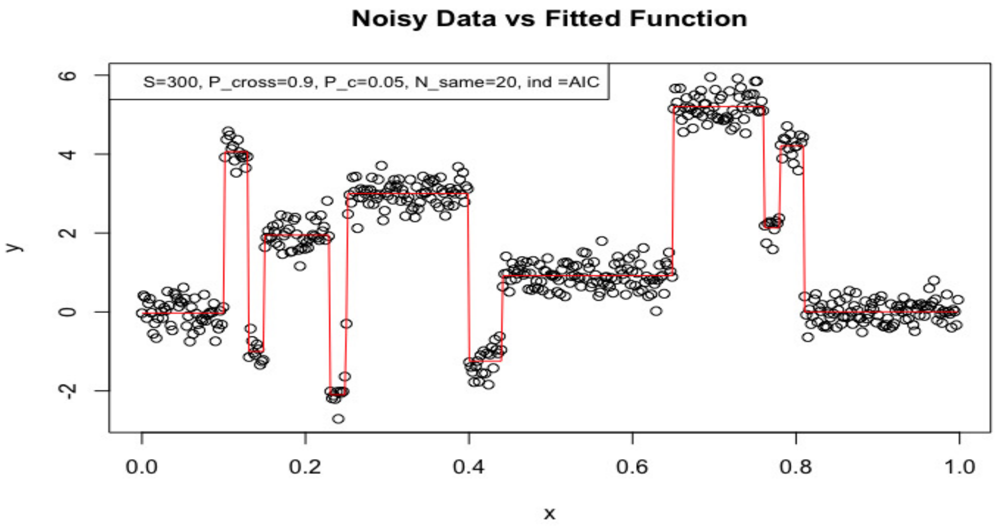


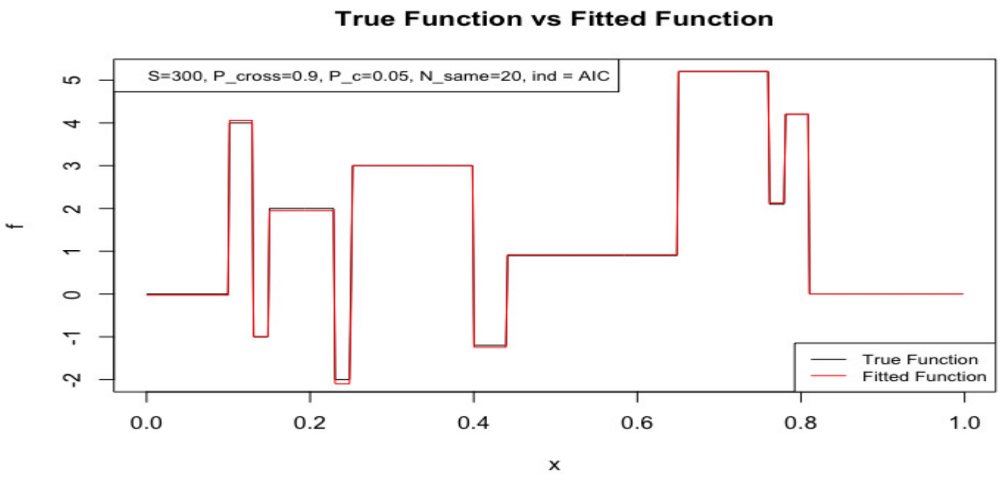


From the plots, we can clearly see that the fitting function fit the noisy points well and is almost the same with true function.

Then we try to select the model with minimum AIC with the same S, P_{cross}, P_c and N_{same}. After 104 iterations, we can find the "best" model with AIC = -1032.581.

We also plot the noisy data vs the fitting piecewise constant function, and the true function vs the fitting function:





From the plots, we can clearly see with AIC criterion, the fitting function fit the noisy points well too and is almost the same with true function, which seems better than the fitting one with MDL criterion.

Then, we want to change the value of the parameters to see what would happen. All the results are showed in the following tables (we show all of these plots in the appendix):

Different S						
S	P_{cross}	P_c	N_{same}	criterion	criterion value	iteration
100	0.9	0.05	20	MDL	305.4731	103
300	0.9	0.05	20	MDL	124.6169	107
500	0.9	0.05	20	MDL	124.1307	99
–	–	–	–	–	–	–
100	0.9	0.05	20	AIC	-721.2873	111
300	0.9	0.05	20	AIC	-1032.581	104
500	0.9	0.05	20	AIC	-1032.581	99

Different P_{cross}						
S	P_{cross}	P_c	N_{same}	criterion	criterion value	iteration
300	0.1	0.05	20	MDL	1125.035	121
300	0.5	0.05	20	MDL	791.3371	74
300	0.7	0.05	20	MDL	131.8142	252
300	0.9	0.05	20	MDL	124.6169	107
300	0.99	0.05	20	MDL	127.7425	94
–	–	–	–	–	–	–
300	0.1	0.05	20	AIC	882.72	53
300	0.5	0.05	20	AIC	40.7821	153
300	0.7	0.05	20	MDL	-1021.668	254
300	0.9	0.05	20	AIC	-1032.581	104
300	0.99	0.05	20	AIC	-1032.581	92

Different P_c						
S	P_{cross}	P_c	N_{same}	criterion	criterion value	iteration
300	0.9	0.01	20	MDL	123.0407	101
300	0.9	0.05	20	MDL	124.6169	107
300	0.9	0.1	20	MDL	130.9543	109
300	0.9	0.5	20	MDL	121.5256	105
300	0.9	0.8	20	MDL	149.8892	103
–	–	–	–	–	–	–
300	0.9	0.01	20	AIC	-1032.581	95
300	0.9	0.05	20	AIC	-1032.581	104
300	0.9	0.1	20	AIC	-1032.581	101
300	0.9	0.5	20	AIC	-1020.123	103
300	0.9	0.8	20	MDL	-1032.581	103

Different N_{same}						
S	P_{cross}	P_c	N_{same}	criterion	criterion value	iteration
300	0.9	0.05	3	MDL	128.6969	83
300	0.9	0.05	5	MDL	128.6969	85
300	0.9	0.05	10	MDL	124.6169	97
300	0.9	0.05	20	MDL	124.6169	107
300	0.9	0.05	30	MDL	124.6169	117
–	–	–	–	–	–	–
300	0.9	0.05	3	AIC	1237.359	9
300	0.9	0.05	5	AIC	-1032.581	89
300	0.9	0.05	10	AIC	-1032.581	94
300	0.9	0.05	20	AIC	-1032.581	104
300	0.9	0.05	30	AIC	-1032.581	114

3 Conclusion

From this experiment, we can obviously to see that with a good choice of parameter (eg.S= 300, $P_{cross} = 0.9$, P_c=0.05, $N_{same} = 20$), we can fit the piecewise constant function very well by Genetic Algorithms with both the minimum description length(MDL) principle and the Akaike information criterion(AIC).

To conduct this experiment, I think there two important things should be focused on.

The first one is how to generate the chromosome that could represent the information of our fitting problem. Besides to set the breakpoint as 1, and the other as 0, the most different thing in this problem is that the first gene of every chromosome should not be 1, because we should have at least one point in each interval of piecewise function, and if we set the first value of chromosome to be 1, that means there are no points in the first interval.

The second important thing is how to choose good parameters. We experiment with different values of parameters to try to find out this problem.

First, when we experiment with different values of S, the size of the population of chromosomes, we can find that with smaller S, such as 100, we couldn't find the "best" model. The reason I think is because with less population, we have less opportunities to perform crossover and mutation, and that will cause us to trap in the local "best" one. The second thing we find is that if the S is large enough, then the final result will not change as S increasing, but the iteration we need will be less, because we have more opportunities to get a better one in each generation so we need less generation to get the "best" one.

Second, when we experiment with different values of P_{cross}, the probability to perform a crossover operation, we can find that with small P_{cross}, we could not find the "best" model, and the results will be much far from the "best" one if we set P_{cross} less than 0.5. Because when P_{cross} is less than 0.5, which mean we will have smaller chance to perform crossover but larger chance to perform mutation, which will cause us have less chance to improve our model. When P_{cross} is larger than 0.7, the results we get will be closed the "best" one, but we can see from the iterations time, that it can save our time if we increase the value of P_{cross}. But to be noticed, from the MDL criterion and with $P_{cross} = 0.9$ and 0.99, we can see that it is not the larger P_{cross} is, the better the results is, because when the P_{cross} is very large, we will have very small chance to perform the mutation operation, which may cause us to trap in the local "best" one.

Third, when we experiment with different values of P_c, the probability to change the gene value when we perform mutation operation, we can find that this parameter will effect the result randomly. With larger P_c, you may get a better result or a worse result, what else, you may spend more time or less time. That's because, the effect of mutation operation is randomly, it may change the gene value to a better one or to a worse one.

Finally, when we experiment with different N_{same}, the stopping points when in how many generations, the best chromosome does not change, we can find that with small N_{same}, we could not find the "best" model. The reason is that sometimes we may trap in the local "best" one for more than one or two generation, so if the N_{same} is small, it will jump out the iteration when we stat in the local "best" for a while. So N_{same} should be big enough. But as we can see, when N_{same} is big enough, to increase it more is not useful to improve our find result because we have find a "best" one under that situation, and what's more, it will spend us more time, the meaningless time.

Above all, choosing parameter is the key problem, but if with a good choice of parameter, we can fit the model well by genetic algorithm.

4 Code

```
##### true function and data
truefunction<-function(x){
  t <- c(0.1, 0.13, 0.15, 0.23, 0.25, 0.4, 0.44, 0.65, 0.76, 0.78, 0.81)
  h <- c(4, -5, 3, -4, 5, -4.2, 2.1, 4.3, -3.1, 2.1, -4.2)
  temp <- 0
  for(i in 1:11) {
    temp <- temp + h[i]/2 * (1 + sign(x - t[i]))
  }
  return(temp)
}
n<-512
x<-(0:(n-1))/n
f<-truefunction(x)
set.seed(0401)
y<-f+rnorm(f)/3
plot(x,y)
lines(x,f)

##### The function to generate the initial population of size S
generate = function(S){
  chrom = matrix(numeric(n*S),nrow = S, ncol = n)
  chrom[,1] = 0
  for (i in 1:S){
    chrom[i,2:n] = sample (c(0,1),n-1,replace = TRUE)
  }
  return(chrom)
}

##### The function to compute MDL
MDL = function(chrom,fittedvalue){
  Bhat = sum(chrom)+1
  b = which(chrom==1)
  njsum = b-1
  njsum[Bhat] = length(chrom)
  nj = njsum[1]
  for (i in 2:Bhat){nj[i] = njsum[i]-njsum[i-1]}
  MDL = Bhat*log(n)+ 1/2 * sum(log(nj)) + n/2 * (1/n * sum((y-fittedvalue)^2) )
  return(MDL)
}

##### The function to compute AIC
AIC = function(chrom,fittedvalue){
  Bhat = sum(chrom)+1
  AIC = n*log(1/n * sum((y-fittedvalue)^2)) + log(n)*2*Bhat
  return(AIC)
}

##### The function to fit the yhat with chromosome
fit = function(chrom){
  b = which(chrom==1)
  f = numeric(length(chrom))
  for (j in 1:(sum(chrom)+1)){
    if (j==1){f[1:(b[j]-1)]= mean(y[1:(b[j]-1)])}
    else{
      if (j==sum(chrom)+1){f[b[j-1]:length(chrom)] = mean(y[b[j-1]:length(chrom)]) }
      else{f[b[j-1]:(b[j]-1)] = mean(y[b[j-1]:(b[j]-1)])}
    }
  }
  return(f)
}

##### The function to compute criterion value(MDF value or AIC value)
compute_criterion = function(S,chrom, ind){
  fittedvalues = matrix(numeric(n*S),nrow = S, ncol = n)
  criterion = numeric(S)
  if (ind=='MDL'){
    for (i in 1:S){
      fittedvalues[i,] = fit(chrom[i,])
      criterion[i] = MDL(chrom[i,],fittedvalues[i,])
    }
  }
  else{
    for (i in 1:S){
      fittedvalues[i,] = fit(chrom[i,])
      criterion[i] = AIC(chrom[i,],fittedvalues[i,])
    }
  }
  return(criterion)
}

##### The function to sort the criterion value(MDF value or AIC value)
```

```
sort_criterion = function(S, criterion){
  criterion$number = c(1:S)
  criterion = criterion[order(criterion$criterion ,decreasing = TRUE),]
  criterion$rank = c(1:S)
  return(criterion)
}

##### The function to perform a crossover operation
crossover = function(chrom,sorted_criterion){
  child = numeric(n)
  parents_ind = sample(sorted_criterion$number,2,prob = sorted_criterion$rank/sum(sort
  parents = chrom[parents_ind,]
  for (i in 2:n){
    child[i] = sample(parents[,i],1)
  }
  return(child)
}

##### The function to perform a mutation operation
mutation = function(chrom,sorted_criterion ,P_c){
  parent_ind = sample(sorted_criterion$number,1,prob = sorted_criterion$rank/sum(sorte
  parent = chrom[parent_ind,]
  child = parent
  for (i in 2:n){
    ind_c = sample(c(0,1),1,prob = c(P_c,1-P_c))
    if (ind_c==0){child[i] = 1- parent[i]}
  }
  return(child)
}

##### The function to produce offspring
Genetic_step = function(chrom,S,P_cross , P_c,ind){
  criterion = data.frame(criterion = compute_criterion(S,chrom,ind))
  sorted_criterion = sort_criterion(S,criterion)
  child = matrix(numeric(n*S),nrow = S, ncol = n)
  child[1,] = chrom[sorted_criterion$number[S],]
  for (i in 2:S){
    ind_pcross = sample(c(0,1),1, prob = c(P_cross , 1- P_cross))
    if(ind_pcross==0){child[i,] = crossover(chrom, sorted_criterion)}
    else{child[i,] = mutation(chrom,sorted_criterion ,P_c)}
  }
  return(child)
}

##### The function of minimization of Genetic Algorithms
Genetic = function(S,P_cross , P_c, N_same,ind){
  chrom = generate(S)
  best = numeric(n)
  count = 1
  iteration = 1
  while (count<N_same){
    offsprings = Genetic_step(chrom,S,P_cross , P_c,ind)
    criterion_new = data.frame(criterion = compute_criterion(S,offsprings ,ind))
    sorted_criterion_new = sort_criterion(S,criterion_new)
    best_new = offsprings[sorted_criterion_new$number[S],]
    if (sum(best==best_new)==n){count = count+1}else{
      count = 1
      best = best_new}
    #print(c(iteration,MDL(best,fit(best))))
    iteration = iteration+1
    chrom = offsprings
  }
  return(c(iteration-1,best))
}

##### Result
set.seed(1)
best = Genetic(S=300,P_cross=0.9, P_c=0.05, N_same=20,ind = 'MDL')
(iteration = best[1])
bestchrom = best[2:513]
bestfit = fit(bestchrom)
(bestMDL = MDL(bestchrom,bestfit))

plot(x,y,main="Noisy Data vs Fitted Function")
lines(x,bestfit,col = 'red')
legend('topleft',c('S=300, P_cross=0.9, P_c=0.05, N_same=20, ind = MDL'),cex= 0.8)

plot(x,f,type='l',main = 'True Function vs Fitted Function')
lines(x,bestfit,col='red')
legend('bottomright',c('True Function','Fitted Function'), lty = 1,col = c('black','re
legend('topleft',c('S=300, P_cross=0.9, P_c=0.05, N_same=20, ind = MDL'),cex= 0.8)

set.seed(1)
best = Genetic(S=300,P_cross=0.9, P_c=0.05, N_same=20,ind = 'AIC')
(iteration = best[1])
bestchrom = best[2:513]
bestfit = fit(bestchrom)
(bestMDL = AIC(bestchrom,bestfit))

plot(x,y,main="Noisy Data vs Fitted Function")
lines(x,bestfit,col = 'red')
legend('topleft',c('S=300, P_cross=0.9, P_c=0.05, N_same=20, ind =AIC'),cex= 0.8)

plot(x,f,type='l',main = 'True Function vs Fitted Function')
lines(x,bestfit,col='red')
legend('bottomright',c('True Function','Fitted Function'), lty = 1,col = c('black','re
legend('topleft',c('S=300, P_cross=0.9, P_c=0.05, N_same=20, ind = AIC'),cex= 0.8)

### different parameter
dif_par = function(S,P_cross, P_c, N_same,ind){
  set.seed(1)
  best = Genetic(S,P_cross, P_c, N_same,ind)
  iteration = best[1]
  bestchrom = best[2:513]
  bestfit = fit(bestchrom)
  if (ind=='MDL'){bestloss = MDL(bestchrom,bestfit)}else{
    bestloss = AIC(bestchrom,bestfit)
  }

  plot(x,y,main="Noisy Data vs True Function vs Fitted Function")
  lines(x,f)
  lines(x,bestfit,col = 'red')
  legend('topleft', paste(c("S=",S,'P_cross=',P_cross, 'P_c=',P_c, 'N_same=',N_same,'i
  legend('bottomright',c('True Function','Fitted Function'), lty = 1,col = c('black','
  return(c(iteration,bestloss))
}

### different S with MDL
dif_par(S=100,P_cross=0.9, P_c=0.05, N_same=20,ind = 'MDL')
dif_par(S=500,P_cross=0.9, P_c=0.05, N_same=20,ind = 'MDL')

### different P_cross with MDL
dif_par(S=300,P_cross=0.99, P_c=0.05, N_same=20,ind = 'MDL')
dif_par(S=300,P_cross=0.5, P_c=0.05, N_same=20,ind = 'MDL')
dif_par(S=300,P_cross=0.1, P_c=0.05, N_same=20,ind = 'MDL')

### different P_c with MDL
dif_par(S=300,P_cross=0.9, P_c=0.01, N_same=20,ind = 'MDL')
dif_par(S=300,P_cross=0.9, P_c=0.1, N_same=20,ind = 'MDL')
dif_par(S=300,P_cross=0.9, P_c=0.5, N_same=20,ind = 'MDL')

### different N_same with MDL
dif_par(S=300,P_cross=0.9, P_c=0.05, N_same=3,ind = 'MDL')
dif_par(S=300,P_cross=0.9, P_c=0.05, N_same=5,ind = 'MDL')
dif_par(S=300,P_cross=0.9, P_c=0.05, N_same=10,ind = 'MDL')
dif_par(S=300,P_cross=0.9, P_c=0.05, N_same=30,ind = 'MDL')
dif_par(S=300,P_cross=0.9, P_c=0.05, N_same=40,ind = 'MDL')

### different criterion with MDL
dif_par(S=300,P_cross=0.9, P_c=0.05, N_same=20,ind = 'MDL')
dif_par(S=300,P_cross=0.9, P_c=0.05, N_same=20,ind = 'MDL')

### different S with AIC
dif_par(S=100,P_cross=0.9, P_c=0.05, N_same=20,ind = 'AIC')
dif_par(S=500,P_cross=0.9, P_c=0.05, N_same=20,ind = 'AIC')

### different P_cross with AIC
dif_par(S=300,P_cross=0.99, P_c=0.05, N_same=20,ind = 'AIC')
dif_par(S=300,P_cross=0.5, P_c=0.05, N_same=20,ind = 'AIC')
dif_par(S=300,P_cross=0.1, P_c=0.05, N_same=20,ind = 'AIC')

### different P_c with AIC
dif_par(S=300,P_cross=0.9, P_c=0.01, N_same=20,ind = 'AIC')
dif_par(S=300,P_cross=0.9, P_c=0.1, N_same=20,ind = 'AIC')
dif_par(S=300,P_cross=0.9, P_c=0.5, N_same=20,ind = 'AIC')

### different N_same with AIC
dif_par(S=300,P_cross=0.9, P_c=0.05, N_same=3,ind = 'AIC')
dif_par(S=300,P_cross=0.9, P_c=0.05, N_same=5,ind = 'AIC')
dif_par(S=300,P_cross=0.9, P_c=0.05, N_same=10,ind = 'AIC')
dif_par(S=300,P_cross=0.9, P_c=0.05, N_same=30,ind = 'AIC')
dif_par(S=300,P_cross=0.9, P_c=0.05, N_same=40,ind = 'AIC')
```