STATISTICAL RESERVOIR MODELING
(PETE 7285)

Report

Esmail Ansari

April 22, 2017

1 INTEGRATING MODELS FOR FACIES PREDICTION

1.1 PART A

The course codes were trimmed so that they only output scaled kapur data (function `scaleKapur`). Then, the kapur facies are counted using `table` function and probability (proportion) of each facies is computed.

```
#cleaning variables and closing plots
rm(list=ls())
if (!is.null(dev.list())) dev.off()

#loading packages
require(boot)
require(nnet)
require(RColorBrewer)
require(ggplot2)
require(lattice)
line.colors <- brewer.pal(N.facies,"Dark2")

#Counting facies using 'table' function
facies      = kapur["facies"]
tab.facies = table(facies)
N.facies    = length(tab.facies)
```

```
tot.obs    = sum(tab.facies)
p.facies   = tab.facies/tot.obs
```

For fitting a beta distribution to the facies, the total counts are used for estimating alpha and beta. `dbeta` function is then used for calculating the probability of facies proportions. The results are then tabulated to be used by *lattice* package.

```
for (i in 1:N.facies){
  #evaluating the parameters of beta function and facies pdf
  x     =  seq(0,1,0.001)
  alpha = tab.facies+1
  beta  =  tot.obs-tab.facies+1
  p.df  = dbeta(x,alpha[i],beta[i])

  #tidy up for lattice package
  current.size = length(x)
  sum.size.pri = current.size+sum.size.pri
  all.prior[(sum.size.pri-current.size):(sum.size.pri-1),1] = x
  all.prior[(sum.size.pri-current.size):(sum.size.pri-1),2] =
    p.df/sum(p.df)
  all.prior[(sum.size.pri-current.size):(sum.size.pri-1),3] =
   rep(paste("facies ",i),length(x))
}
#plotting priors using lattice package
line.colors = brewer.pal(N.facies,"Dark2")
myStrip     = function(which.panel, factor.levels, ...) {
  panel.rect(0, 0, 1, 1,col = line.colors[which.panel],border = 1)
  panel.text(x = 0.5, y = 0.5,font = 2,lab = factor.levels[which.panel])
}
print(xyplot(Probability ~ Proportion|Facies.type,
               data = all.prior[1:sum.size.pri-1,],
               type = "l",lwd = 3,strip = myStrip))
```

1.2 Part B

Function `prob.facies` was prepared for bootstraping using `boot` function. This function fits the multinomial regression to the facies and outputs the probability of probability of each facies.

```
#Defining the probability function to be bootstrapped
prob.facies     <-  function(formula, data, indices){
  selected.data    =  data[indices,]
```

```
  kapur.glm       =  multinom(formula,data=selected.data)
  pred.glm        =  fitted(kapur.glm)
  pred.glm        =  sortPredByLevels(pred.glm)
  most.likely.glm =  apply(pred.glm,1,which.max)
  tab.pred.facies =  table(most.likely.glm)
  tot.obs         =   sum(tab.pred.facies)
  all.fac.name    =  names(table(data$facies))
  N.facies        =   length(all.fac.name)
  idxToFac        =   all.fac.name
  names(idxToFac) =   as.character(1:N.facies)
  sel.fac.name    =   names(tab.pred.facies)
  names(tab.pred.facies) = idxToFac[sel.fac.name]
  no.occur.facies =   setdiff(all.fac.name,sel.fac.name)
  prob            =   rep(NA,N.facies)
  names(prob)     =  all.fac.name
  if (any(as.integer(no.occur.facies))) {
    prob[no.occur.facies] = 0
  }
  prob[sel.fac.name]   =  as.vector(tab.pred.facies)/tot.obs
return(prob)
}
```

Note that when we are using bootstraping for categorical data, we face two important challenges. First, some facies type may not be sampled and second some facies type may not be predicted. For both of these situations we should put the probability of such facies to zero. Here, facies 2 and 7 occur relatively few times and sometimes they may not be sampled or predictred. Below lines (which are in `prob.facies` function) are added for detecting unsampled facies and enforcing their probabilities as zero when bootstapping. Below chunck demonstrates the idea of naming existing facies according to the indeices that come out of `which.max` function.

```
all.fac.name    =  names(table(data$facies))
N.facies        =   length(all.fac.name)
idxToFac        =   all.fac.name
names(idxToFac) =   as.character(1:N.facies)
sel.fac.name    =   names(tab.pred.facies)
names(tab.pred.facies) = idxToFac[sel.fac.name]
no.occur.facies =   setdiff(all.fac.name,sel.fac.name)
prob            =   rep(NA,N.facies)
names(prob)     =  all.fac.name
if (any(as.integer(no.occur.facies))) {
  prob[no.occur.facies] = 0
```

And finally function `prob.facies` is prepared to be passed to function `boot` to be bootstrapped 300 times.

```
# Bootstrapping using boot function
form        = facies ~ caliper + ind.deep + gamma + r.deep + density

boot.result = boot(formula=form, statistic=prob.facies,
                   data=scaleKapur(kapur),R=N.Boot)
```

1.3 Part C

Posteriors are calculated by multiplying the priors and likelihoods. In below code `y` represents the likelihood and `p.df` represents the prior counts for the facies.

```
# Finding and plotting facies posteriors
posterior = p.df * y/sum(p.df * y)
```

and then we plot the posterior of each facies using *ggplot2* package.

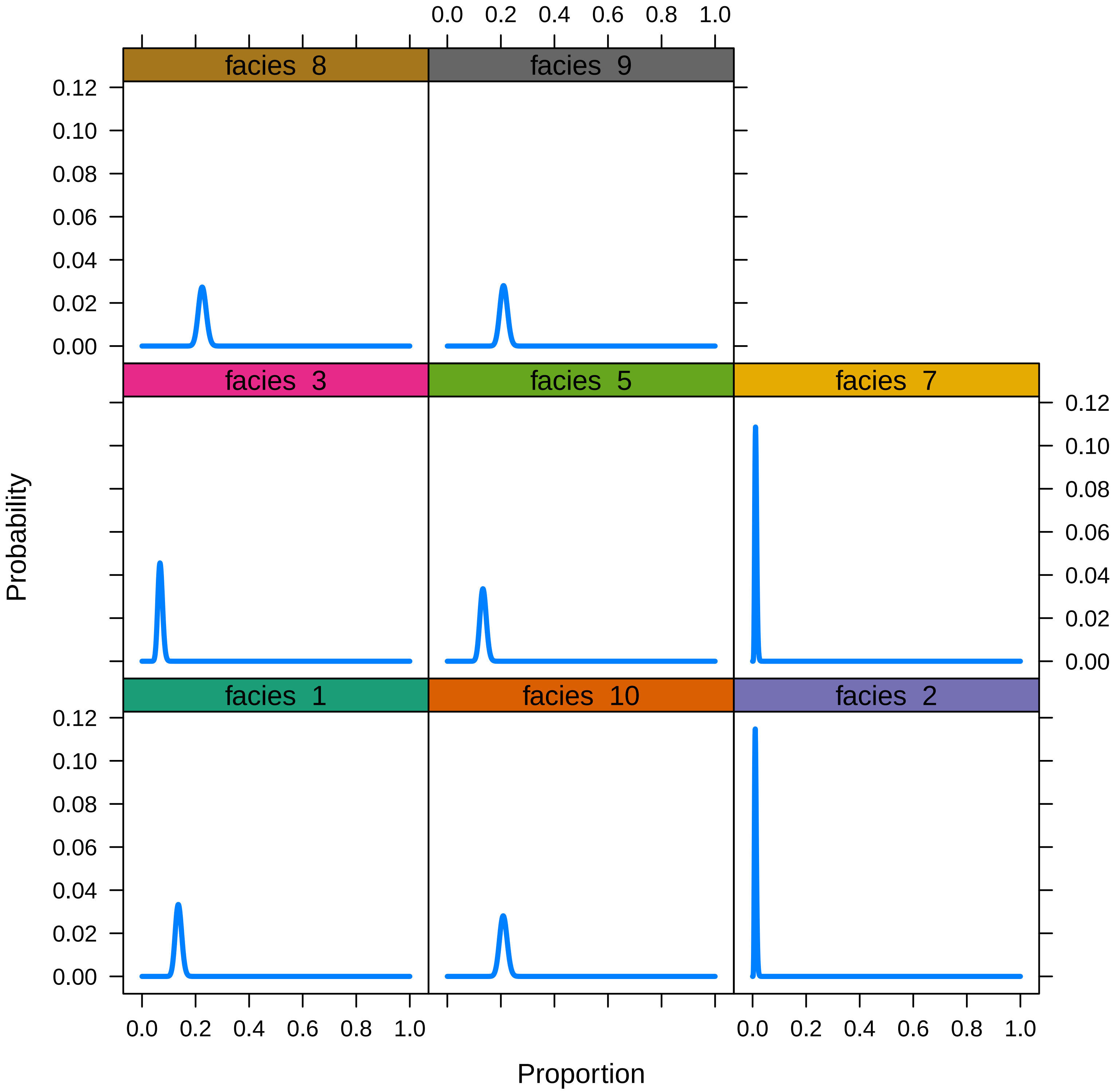
```
  df        = data.frame(x,prior,likelihood,posterior)

  g[[i]]    = ggplot( df, aes(x)) +
                     geom_line(aes(y=prior,colour="prior")) +
                     geom_line(aes(y=likelihood,colour="likelihood")) +
                     geom_line(aes(y=posterior,colour="posterior") )

  g[[i]]   = g[[i]] + xlab(paste("proportion of facies ",
                      names(tab.facies[i]))) + ylab("Probability")+
                      scale_colour_manual("",values=c("darkgreen","blue","red"))


require(gridExtra)
for (i in seq(1,8,2)){
  sidebysideplot = grid.arrange(g[[i]], g[[i+1]], nrow=2)
  print(sidebysideplot)
}
```

For facies 1 and 2, the likelihood improves our belief about the proportion of these facies and our model strengthens the priors. The likelihood weakens our confidence (prior) on probabilities of facies 5 and 7. Probabilities of facies 9 and 10 are not affected using bayesian analysis. This may be because their proportions are already high and likelihood also approves the priors to be the best representation of these facies. For facies 3, the likelihood hugely changes our understanding of it's probability. It inidcates that most probable proportion of facies 3 is 0.055 and our prior opinion of 0.07 may not be accurate based on new evidence (bootstrapped multinomial model).


facies 8
facies 9
facies 3
facies 5
facies 7
facies 1
facies 10
facies 2
Probability
Proportion
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.0
0.2
0.4
0.6
0.8
1.0

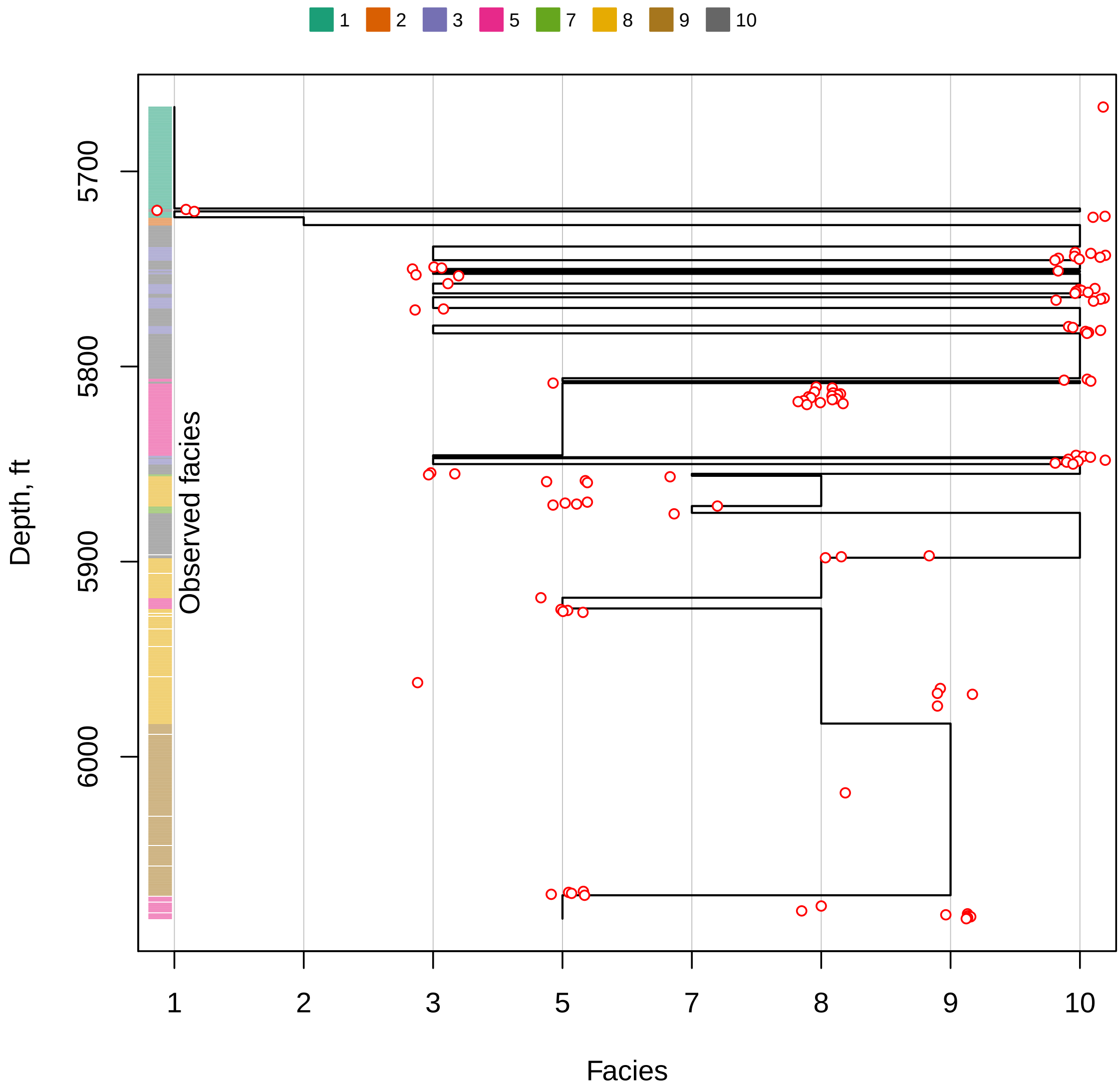
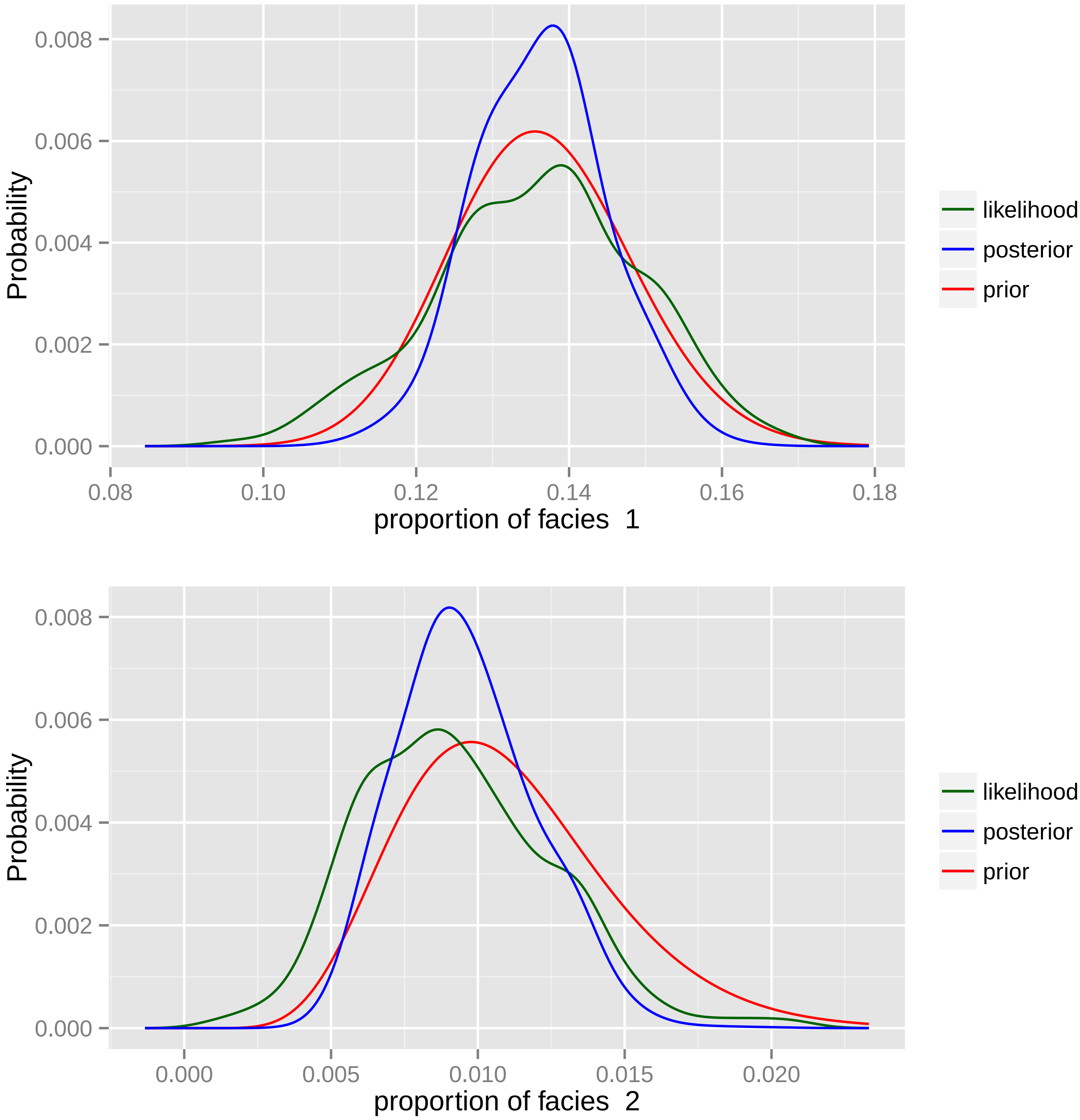
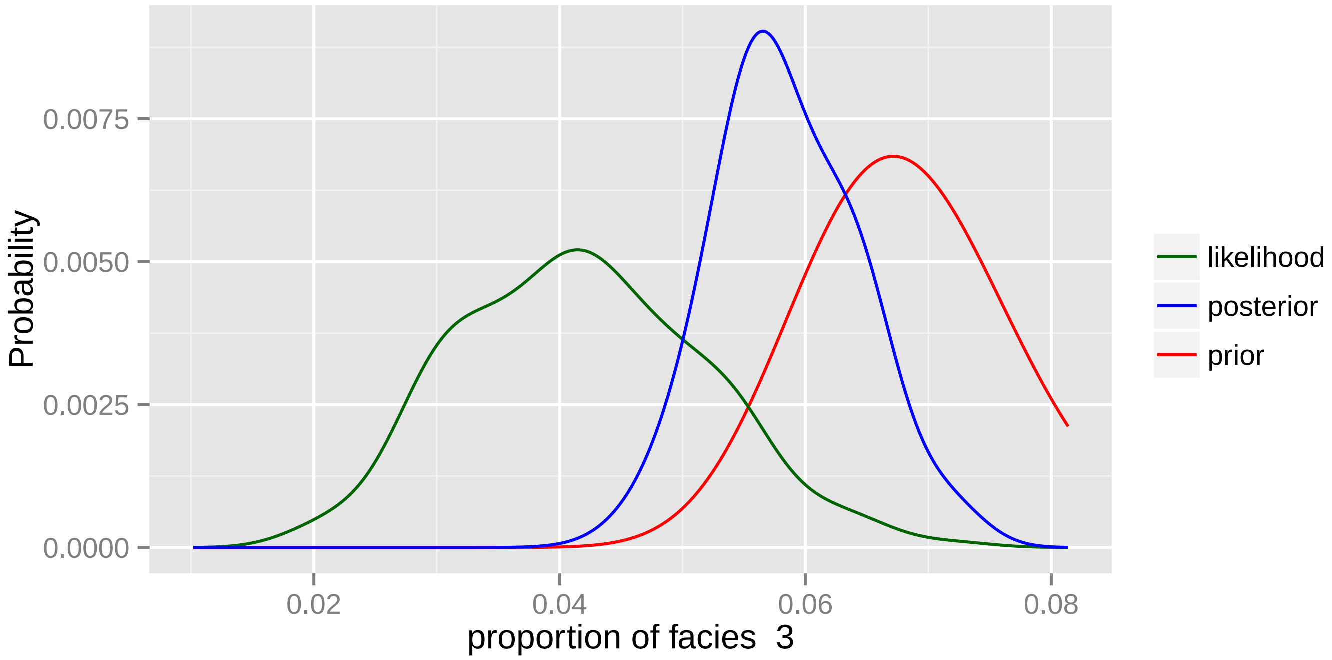
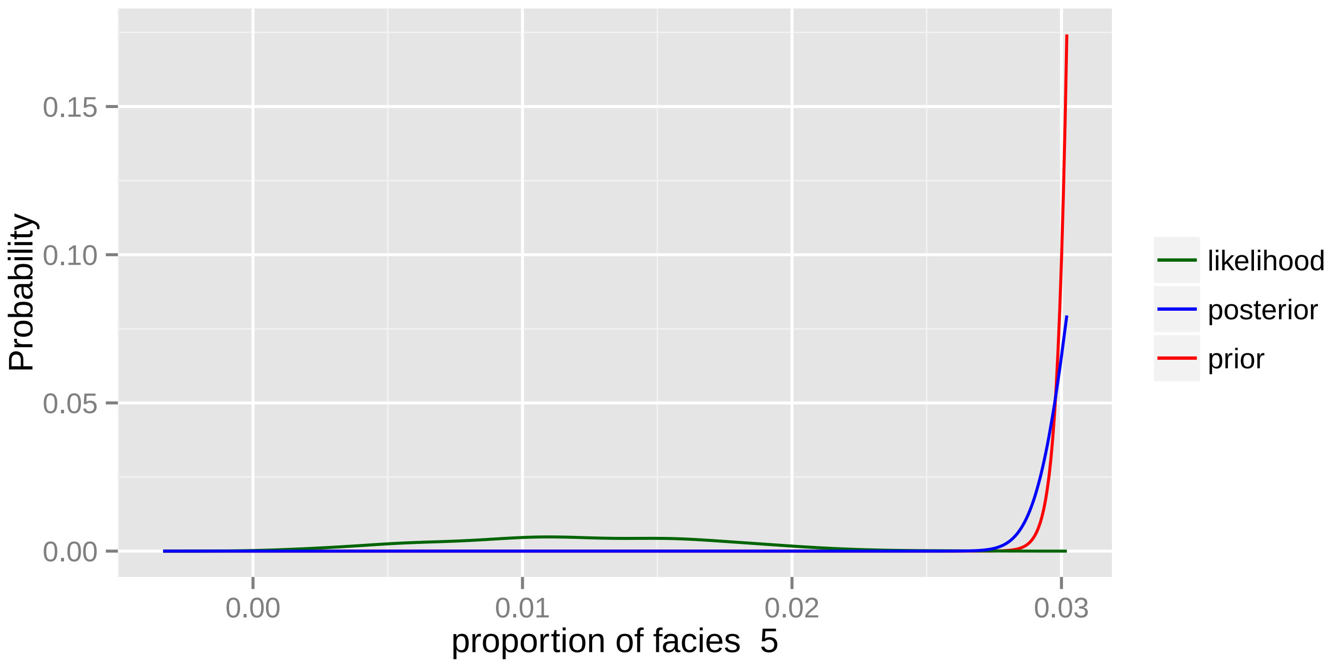


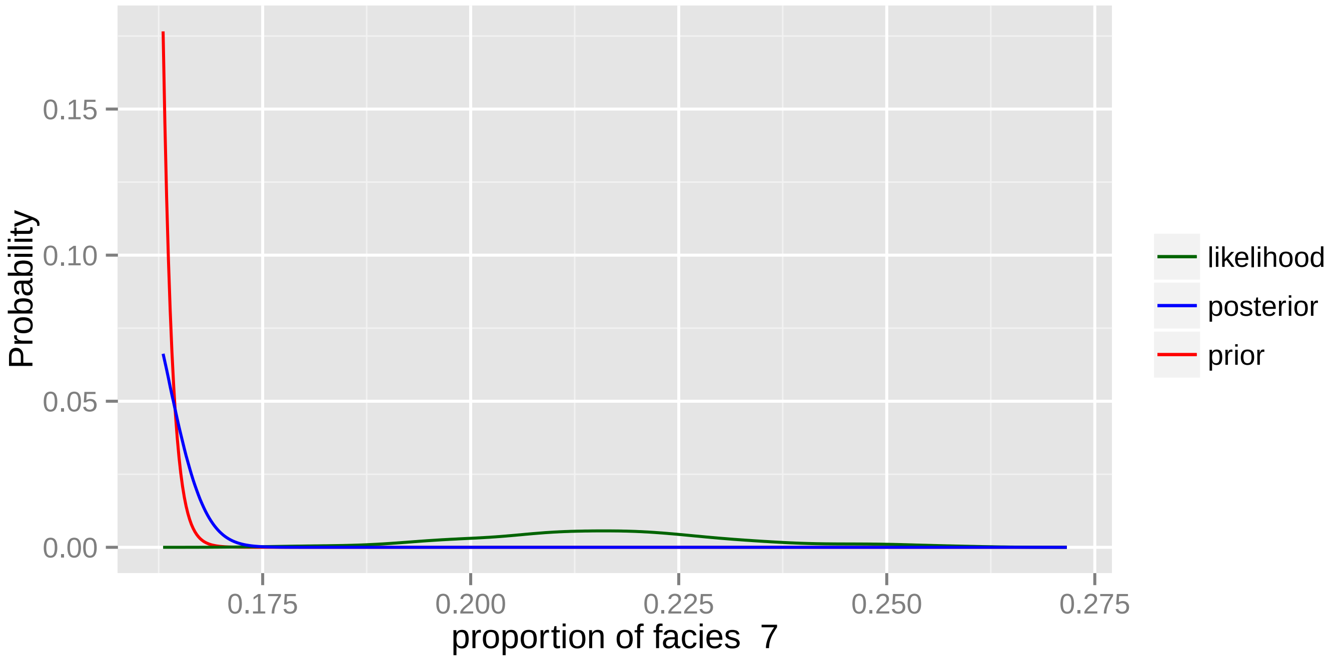
Figure 1.1: we just used linear discrimentant analysis instead of logistic regression to plot prediction vs. depth. Posteriors are not used in constructing this figure.



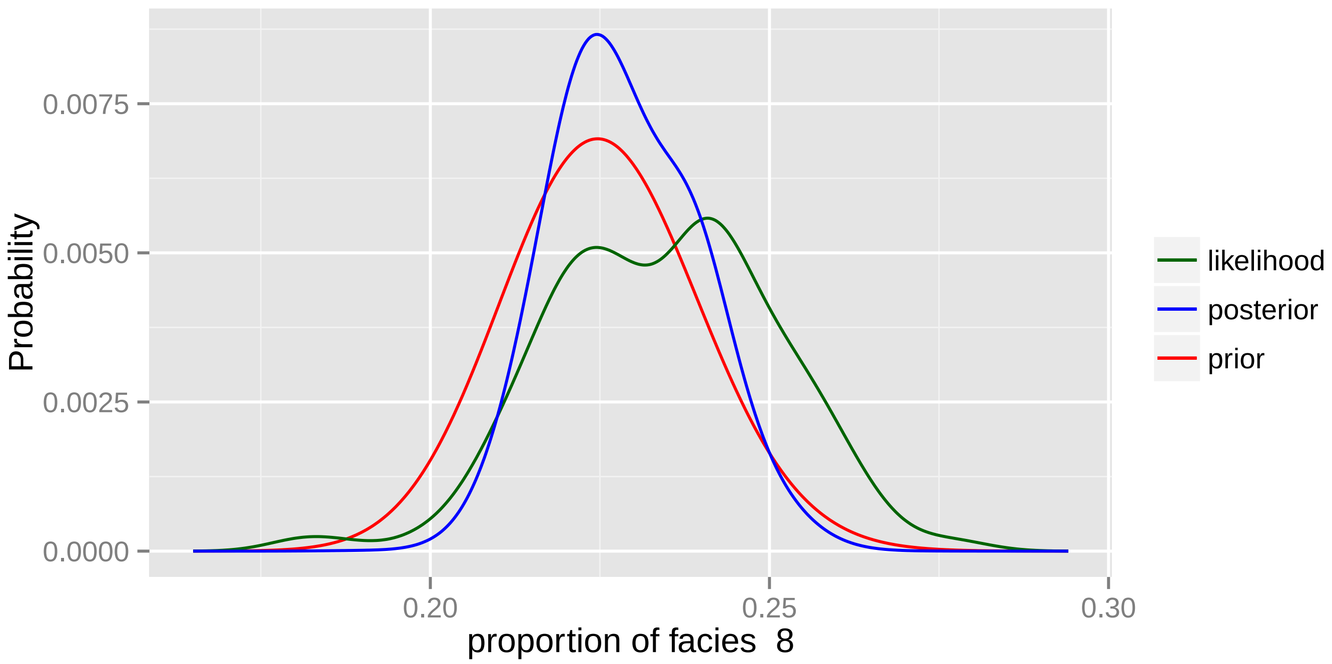
Probability
0.000
0.002
0.004
0.006
0.008
0.08
0.10
0.12
0.14
0.16
0.18
proportion of facies 1
likelihood
posterior
prior
Probability
0.000
0.002
0.004
0.006
0.008
0.000
0.005
0.010
0.015
0.020
proportion of facies 2
likelihood
posterior
prior


0.0075
0.0050
0.0025
0.0000
Probability
0.02
0.04
0.06
0.08
proportion of facies 3
likelihood
posterior
prior

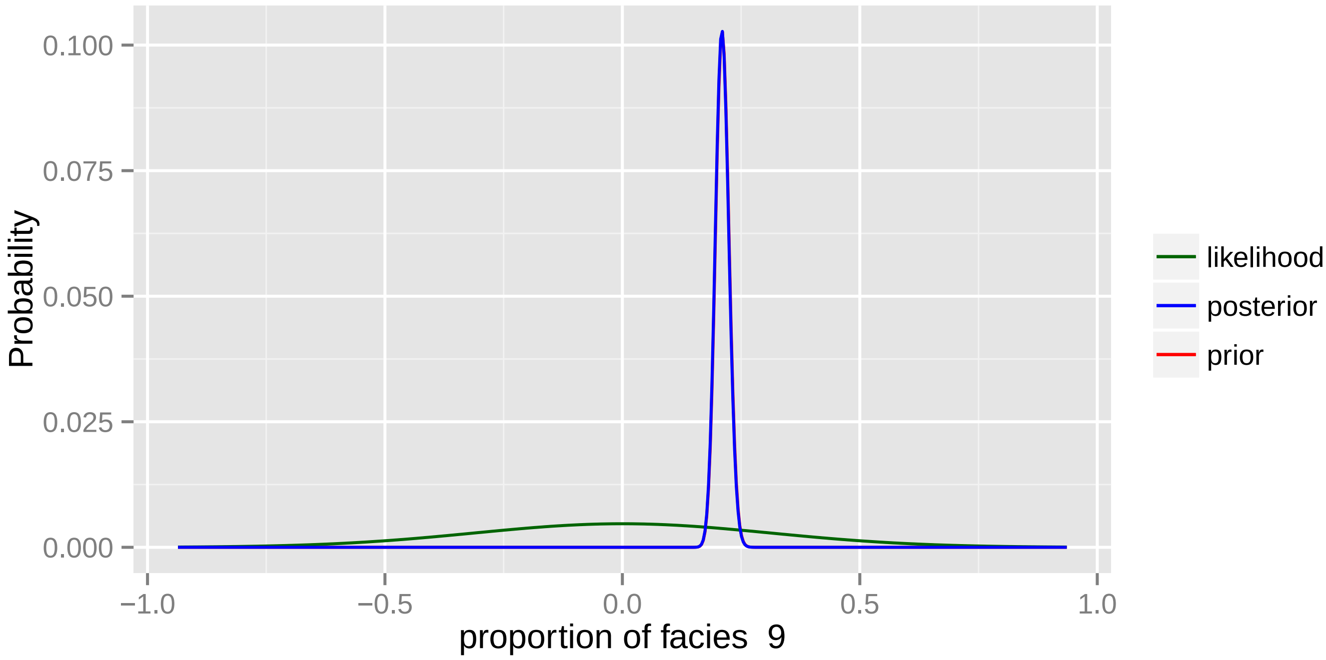

0.15
0.10
0.05
0.00
Probability
0.00
0.01
0.02
0.03
proportion of facies 5
likelihood
posterior
prior



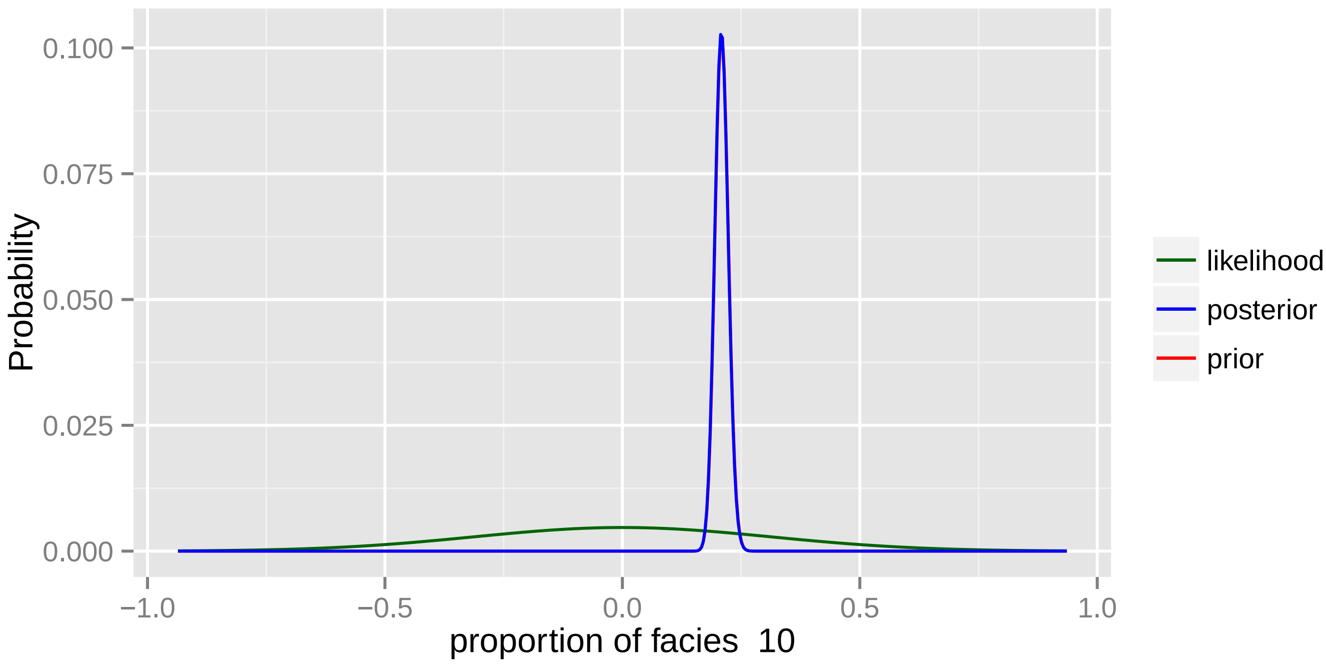
Probability
0.15
0.10
0.05
0.00
0.175
0.200
0.225
0.250
0.275
proportion of facies 7
likelihood
posterior
prior



Probability
0.0075
0.0050
0.0025
0.0000
0.20
0.25
0.30
proportion of facies 8
likelihood
posterior
prior



0.100
0.075
0.050
0.025
0.000
Probability
−1.0
−0.5
0.0
0.5
1.0
proportion of facies 9
likelihood
posterior
prior



0.100
0.075
0.050
0.025
0.000
Probability
−1.0
−0.5
0.0
0.5
1.0
proportion of facies 10
likelihood
posterior
prior

2 MARKOV CHAIN AS A STOCHASTIC GRAPH

Various assumptions used for solving this problem are summarized below.

1. For the work over, first arrived well, will be served first.
2. If multiple well damage at the same time, the oldest well(the well that has started the chain sooner) will be worked over first.
3. The reservoir engineer has estimated a value of N.true for each well. As soon as this value becomes zero (or less), the well is considered depleted and should be abandoned.
4. For constructing violin plot, only drilling risks are taken into consideration and reserovir engineer's prediction (N.true) will remain the same for all runs. If desired, this assumption can be easily eleminated.

`SetState.R` contains functions that make small changes to state transitions.

```
#Defining states, costs and state transition functions
rm(list=ls())
source(file.path('/Users/eansar2/Desktop/Geostatistics/TakeHome',
                 'SetState.R'))
costs <- list(Queue=0,Drill=-25,FinishedDrill=0,Complete=-25,Produce=-0.1,
              Shut_in=0,WaitForRig=0,WorkOver=-25,Abandon=-10)
states <-   c("Queue","Drill","FinishedDrill","Complete","Produce",
              "Shut_in","WaitForRig","WorkOver","Abandon")
```

Three additional properties (`current.state, next.state, drilling.flag`) are added to the course notes proposed well objects in order to track them more easily. All wells are in Queue at the begining (time zero).

```
#Initializing well objects
wells <- vector("list",N.wells)
for (i in 1:N.wells){
  wells[[i]] = list(current.state="Queue",next.state="Queue",
                    t.states=matrix(0,10000,N.states),
                    t.drill=sample(2:6,1,p=c(0,0.25,0.5,0.75,1)),
                    t.this=0,N.believe=runif(1,500,1000),
                    N.true=runif(1,500,1000),q=runif(1,0.5,3),value=0,
                    drilling.flag=FALSE)

  colnames(wells[[i]]$t.states) <- states
}
```

Algorithm 1 Markov Chain

```
Initialize variables
while not all wells abandoned do
    for each wells do
        if drilling job finished then
            Move the well out of drilling state
        end if
        if drilling is empty and this well is in queue then
            Drill this well
        end if
        if any well is damaged then
            Work over the earlier damaged well (oldest if multiple)
        end if
        Determine the next state of the well
    end for
    for all wells do
        Update the well's current state and adjust it's properties
        calculate the profit
        calculate N.ture
        if N.true<=0 then
            flag the well for abandoning
        end if
        if any well is in wait for rig state then
            determine possible workover wells and select only one based on assumptions
        end if
    end for
    time=time+1
end while
```

Initialize the transition matrix so that all the pathes between states are open at the beginning.

```
#Defining initial transition matrix
trans.prob = matrix(0,nrow=N.states,ncol=N.states,
                    dimnames=list(states,states))
trans.prob = initialize(trans.prob) #initializing transition matrix
```

`Drilling.flag` shows which well is being drilled. `wells.abandoned` records the wells that are absorbed.

```
#Initiallizing variables
drilling.flag      = rep(FALSE,N.wells)
wells.abandoned    = rep(FALSE,N.wells)
pos.workover.wells = rep(FALSE,N.wells)
def.N.true.wells   = rep(FALSE,N.wells)
waiting.time       = rep(0,N.wells)
workover.well      = 0
time               = 0
```

It is not necessary, but would be more understandable and sometimes useful, if we calculate the costs and some other paramters in a separate `for` loop. (i.e. after all wells are moved one step and before the end of the month).

```
#Main loop
while (!all(wells.abandoned)){
   #First for loop
   for (i in 1:N.wells){
     #move wells one time step
     ...
   }
   #Second for loop
   for (i in 1:N.wells){
     #for calculating the costs
     ...
   }
  ...
  wells.abandoned[[i]] = (wells[[i]]$current.state=="Abandon")
  time=time+1
}
```

The next state is sampled between all posible pathes weighted by their probabilities.

```
    #select the next state.
    pos.next.states <- names(which(  trans.prob[current.state,]!=0 ))
    prob.next.states  <- trans.prob[current.state,pos.next.states]
    selected.next.state <- sample(pos.next.states,1,p=prob.next.states)
    wells[[i]]$next.state <- selected.next.state
```

Below chunck demonstrates how workover wells are selected.

```
  #In the second "for" loop
    pos.workover.wells[i]=(wells[[i]]$current.state=="WaitForRig")
    waiting.time[i] = as.integer(pos.workover.wells[i])
                      *wells[[i]]$t.this

  #Before increasing time step, check for workover wells
  if (any(pos.workover.wells))
  {
    #First check which wells have arrived at WaitForRig state
    max.waiting.wells = which(waiting.time==
                              max(waiting.time[pos.workover.wells]))

    #If more than one well arrive at the same time, choose the well
    # has started the chain sooner(Oldest well)
    workover.well     = min(max.waiting.wells)
  }else{
    workover.well     = 0
  }
```

Below chunck is located at the begining of the first `for` loop to check whether the drilling state is empty.

```
    #check if any well is being drilled. If not, the next well can be
    #drilled.
    for (j in 1:N.wells)
       drilling.flag[j] =wells[[j]]$drilling.flag
    if (any(drilling.flag))
        trans.prob <- drill.close(trans.prob) else
        trans.prob <- drill.open(trans.prob)
```

The below code is added for abandoning depleted wells.

```
     #If N.true of the well is depleted, abandon that well.
     def.N.true.wells[i] = (wells[[i]]$N.true <= 0)
     if (def.N.true.wells[i])
        trans.prob <- well.abandon(trans.prob)  else
        trans.prob <- well.open(trans.prob)
```

Below code is for moving the well out of the drilling state into the finished drilling state if it's `t.drill` is fulfilled.

```
#If drilling is finished move the well out.
if (sum(wells[[i]]$t.states[,"Drill"])==wells[[i]]$t.drill)
   trans.prob <- Finished.drill.open(trans.prob) else
   trans.prob <- Finished.drill.close(trans.prob)
```

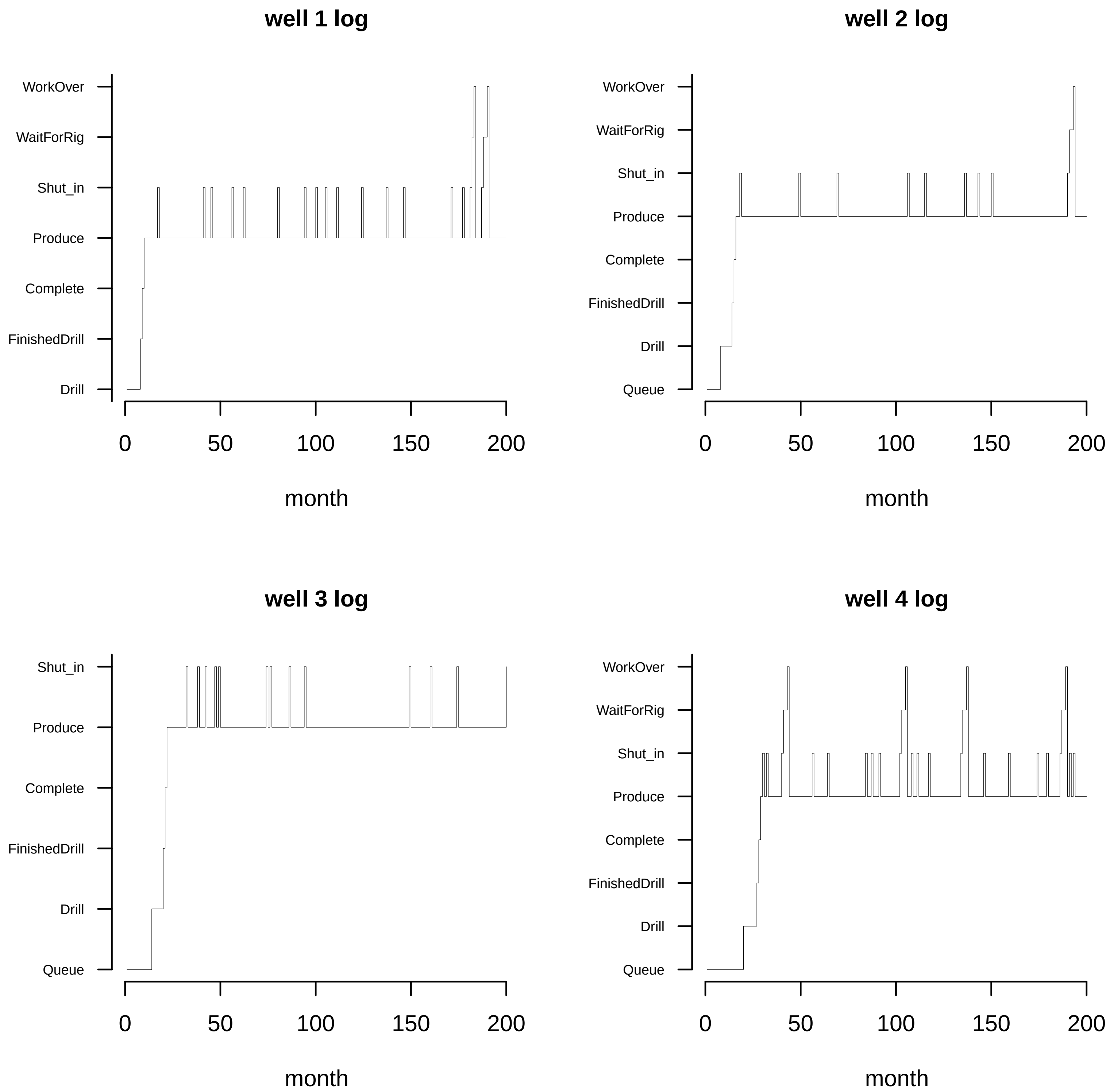
Below chunck should be clear.

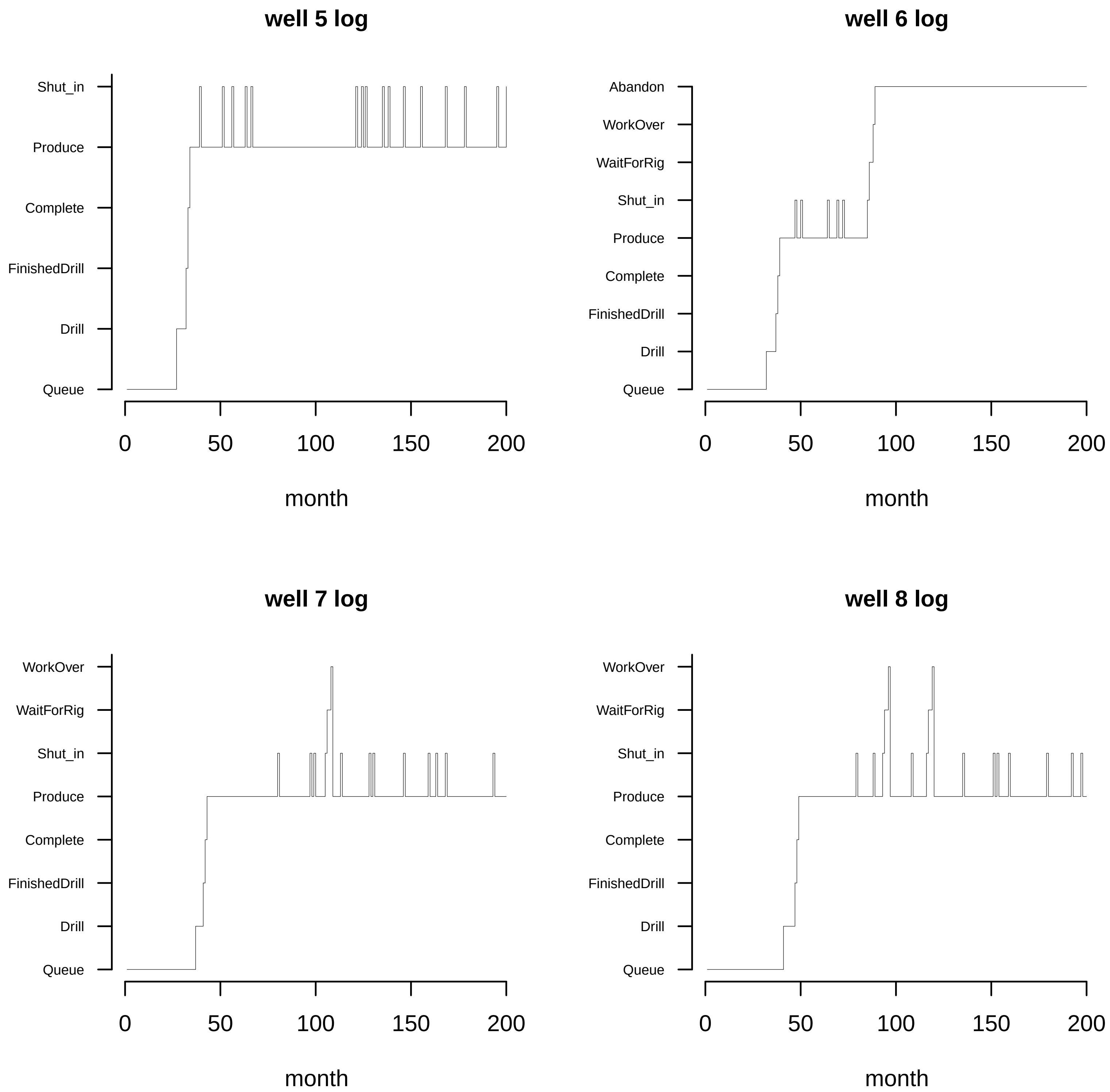
```
#WorkOver is only open for one well at time
if (i==workover.well)
   trans.prob <- WorkOver.open(trans.prob) else
   trans.prob <- WorkOver.close(trans.prob)
```

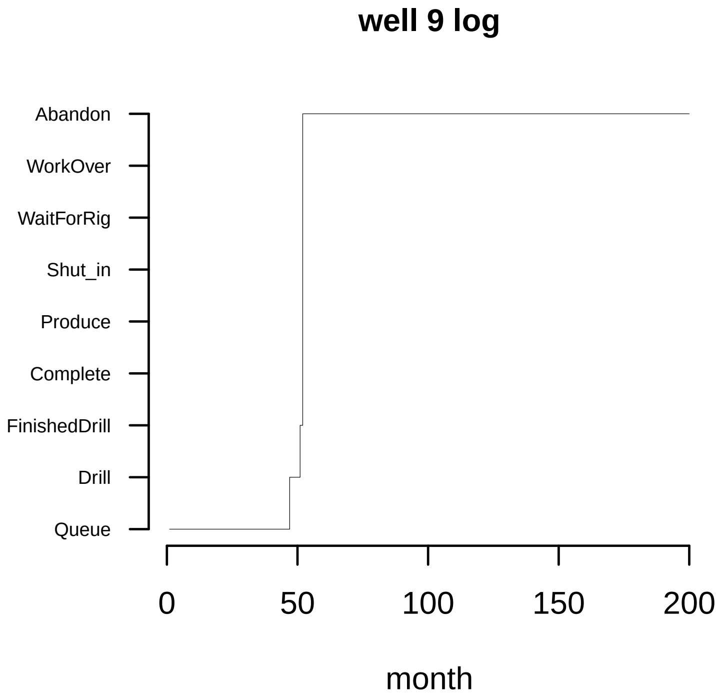
And finally the costs are calculated in the second `for` loop.

```
#calculate the profit and modify the value of the well
profit=as.integer(wells[[i]]$current.state=="Produce")*wells[[i]]$q
if (!(wells[[i]]$current.state=="Abandon"))
  wells[[i]]$value = (wells[[i]]$value+
                      costs[[wells[[i]]$current.state]]+profit)
#The cost of Abandon state is single time cost.
if ((wells[[i]]$current.state=="Abandon")&&(wells[[i]]$t.this==1))
  wells[[i]]$value = (wells[[i]]$value+
                      costs[[wells[[i]]$current.state]]+profit)
```

Log of the wells for a prolific field development (revenue of around 2100 million $ after all wells are abandoned) are shown below. As violen plot indicates this is one of the profitable develpments. The average revenue (after 1000 months) is around 600 million $.


well 1 log
WorkOver
WaitForRig
Shut_in
Produce
Complete
FinishedDrill
Drill
0
50
100
150
200
month
well 2 log
WorkOver
WaitForRig
Shut_in
Produce
Complete
FinishedDrill
Drill
Queue
0
50
100
150
200
month
well 3 log
Shut_in
Produce
Complete
FinishedDrill
Drill
Queue
0
50
100
150
200
month
well 4 log
WorkOver
WaitForRig
Shut_in
Produce
Complete
FinishedDrill
Drill
Queue
0
50
100
150
200
month


well 5 log
Shut_in
Produce
Complete
FinishedDrill
Drill
Queue
0
50
100
150
200
month
well 6 log
Abandon
WorkOver
WaitForRig
Shut_in
Produce
Complete
FinishedDrill
Drill
Queue
0
50
100
150
200
month
well 7 log
WorkOver
WaitForRig
Shut_in
Produce
Complete
FinishedDrill
Drill
Queue
0
50
100
150
200
month
well 8 log
WorkOver
WaitForRig
Shut_in
Produce
Complete
FinishedDrill
Drill
Queue
0
50
100
150
200
month


well 9 log
Abandon
WorkOver
WaitForRig
Shut_in
Produce
Complete
FinishedDrill
Drill
Queue
0
50
100
150
200
month

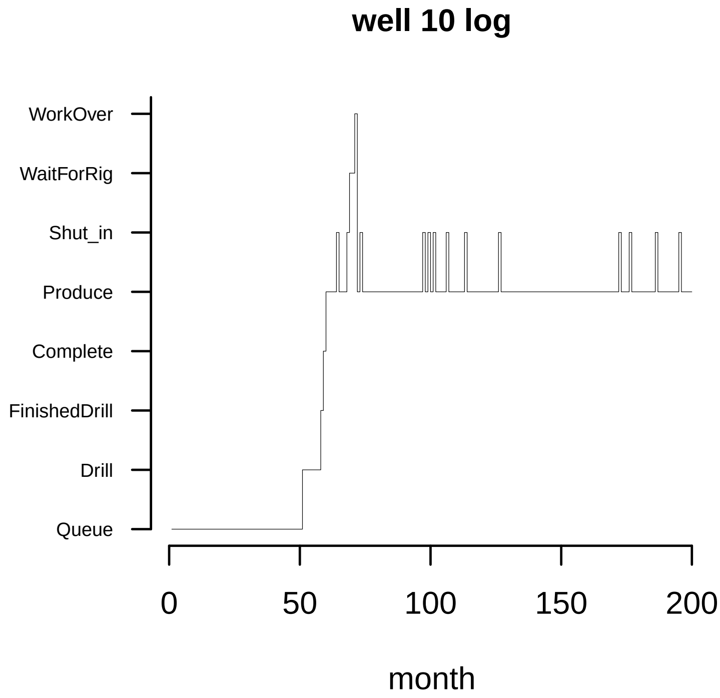

well 10 log
WorkOver
WaitForRig
Shut_in
Produce
Complete
FinishedDrill
Drill
Queue
0
50
100
150
200
month

Figure 2.1: Violin plot for the field develpment. The red line connects the mean of field net revenue calculated after each month. It turns out that the mean is equal to the median. The skewness of the kernell densities for all months are small and upward. It is more likely that we have to wait around 300 months to get our investment money back. If we are the most luckiest person on earth, this waiting time would decrease to 180 months and 3800 million $ is the maximum revenue we will attain. The most probable maximum cost would be around 1500 million and it happens after around 70 months. The maximum amount that we have to invest on field development does not depend (relatively) on whether we are lucky or not. After a 1000 month!(83 years), the 0.25 percent quantile of violins are roughly higher than zero which can be promising if we are interested in long term revenue.