DYNAMIC DECISION MODELING: REPLICATION OF RUST AND ROTHWELL (1995)

PAUL SCHRIMPF

This assignment will attempt to replicate the result of Rust and Rothwell (1995).

Problem 1: Start by reading Rust and Rothwell (1995). You may also want to read the very related Rothwell and Rust (1995). There is nothing to hand in for this question.

Problem 2 (Explore the data): The file loadRRdata.R downloads Rust and Rothwell's data, loads it into R, and creates the state and action variables described in the paper. To check that the data is correct, reproduce Table I and Figures 3, 4, and 7. If you'd like you could also try to recreate some of the other figures from either Rust and Rothwell (1995) or Rothwell and Rust (1995).

There are many ways to create Table I. You could use the data.table package. You could also use the "aggregate" function. For example, to create the first three rows:

```
df <- plantData
df$percent.time.operating <- df$hrs.operating/df$hrs.total
df$percent.time.refueling <- df$hrs.refuel/df$hrs.total
df$percent.time.forced.out <- df$hrs.forced.out/df$hrs.total
df$percent.time.planned.out <- df$hrs.plan.out/df$hrs.total
df$era <- "1975-79"
df$era[df$year>=80] <- "1980-83"
df$era[df$year>=84] <- "1984-93"
tab1 <- aggregate(. ~ era, df[,c("era", "percent.time.operating",
                          "percent.time.refueling", "percent.time.forced.out",
                          "percent.time.planned.out")], FUN=mean)
```

I suggest using ggplot2 to create the figures. For example, for Figure 3,

```
fig3 <- ggplot(data=aggregate(. ~ year, plantData, mean),
               aes(x=year,y=num.forced.out)) + geom_line()
```

If the results do not match the paper, comment on whether you think the differences are important. You could also double check the loadRRdata.R code for whether all the variables are being defined correctly. I did not make any deliberate mistakes, but I may have overlooked something.

Problem 3 (Transition probabilities): The first step in estimating the dynamic model is to estimate the state transition probabilities, which are denoted by $p(x'|x,a,\psi)$ in the paper. The details of how this is done are described more clearly in Rothwell and Rust (1995) than in Rust and Rothwell (1995). The transitions of duration (d_t in their paper, `duration` in my code) and spell type (r_t in their paper, `spelltype` in my code) are deterministic. The only thing that is stochastic is the signal received by the power plant operator (f_t in their paper, `npp.signal` in my code). If a plant is currently refueling, it can only go back to operating if it does not receive a "continue refueling" signal ($f_t = 2$ or `npp.signal=cont.refuel`). This happens with probability $p_{ro}(x)$. In the paper they allow p_{ro} to depend on the duration of the current refueling spell and the age of the power plant. We will ignore age to simplify. The timing of decisions is as follows:

- Period begins state is (r_t, d_t). The signal, f_t and shocks ϵ_t are observed
- Action, a_t, is chosen based on signal, shocks, and state
- Next state, (r_{t+1}, d_{t+1}), is determined based on action

Date: Due: March 6th, 2018.

- Proceed to time $t+1$

loadRRdata.R follows this timing convention.

Forced outage signals can occur next period during either spell type, unless the current signal is "continue refueling" and the current spell is "refueling." We will estimate the probability of a forced outage signal given the duration of the current spell using a logit. Be careful with your treatment of what Rothwell and Rust (1995) call "major problem spells" (see pages 2 and 47).

The R command for estimating a logit is glm, which stands for generalized linear model. The following code estimates the probability of a forced signal and then plots the results.

```
# estimate P(forced outage | duration)
pof.glm <- glm( (npp.signal=="forced.outage") ~ as.factor(dur.cens),
                family=binomial(link=logit) ,
                data=subset(plantData,npp.signal!="cont.refuel"))
# plot estimated probability along with 95% confidence bands
md <- max(subset(plantData,npp.signal!="cont.refuel")$dur.cens)
df <- data.frame(dur.cens=1:md)
tmp <- predict(pof.glm,newdata=df,type="response", se.fit=TRUE)
df$p <- tmp$fit
df$se.p <- tmp$se.fit
df$lo <- df$p - df$se.p*1.96
df$hi <- df$p + df$se.p*1.96
fig10.of <- ggplot(data=df, aes(x=dur.cens, y=p)) +
  geom_line() + geom_line(aes(y=lo),linetype="dashed") +
  geom_line(aes(y=hi),linetype="dashed")
rm(df,tmp)
```

This code allows $p_{of}(d)$ to be a completely flexible function of d (the `as.factor(dur.cens)` puts in dummies for each duraction length). Based on Rothwell and Rust (1995), it looks like Rust and Rothwell specified $p_{of}(d)$ to be a quadratic function of d. You can choose to specify $p_{of}(d)$ however you think is best.

(1) Similarly estimate $p_{ro}(d)$. For this you will only want to use the observation when the spell type is "refueling" and the outcome is getting a signal other than "cont.refuel."

Problem 4 (Calculating the likelihood): Equations (4) and (5) of Rust and Rothwell (1995) are the key ingredients in the likelihood. Equation (4) gives the conditional choice probabilities implied by the model given the choice-specific value function. Equation (5) defines the choice specific value function. Equation (5) is

$$v_t(x,a) = u(x,a,\phi) + \beta \int \log\left[\sum_{a' \in A(x')} \exp(v_{t+1}(x',a'))\right] p(dx'|x,a,\psi) \tag{1}$$

To calculate this we need to:

- Find the flow utility given a state, x, actions a, and parameters ϕ.
- Take the sum over feasible actions in a given state to calculate $\sum_{a' \in A(x')}$
- Take the sum over states that are reachable given action a and state x to calculate the integral
- Use our estimates of $p_{ro}(d)$ and $p_{of}(d)$ to calculate $p(dx'|x,a,\psi)$

You may choose to organize your code however you would like. I suggest writing a function that given ϕ calculates a number of states $\times$ number of actions matrix of $u(x,a;\phi)$. Something like:

```
flow.utility <- function(phi) {
  u <- matrix(0,nrow=n.states,ncol=n.actions)
  colnames(u) <- levels(plantData$action)
  ## Add code to calculate u
  return(u)
```

```
}
```

For the value function, I would write a function that returns a $T\times$ number of states $\times$ number of actions array that contains the choice specific value function. The function could be something like:

```
## Calculate choice specific value function
## v(t,x,a) =  u(x,a) + beta*E[max_{a'} v(t+1,x',a') + e(a') | x, a]
choice.value <- function( u, discount, p.x, T) {
  v <- array(dim=c(T,nrow(u),ncol(u)))
  ## Add code to calculate v
  return(v)
}
```

Given the above two functions, the likelihood can be calculated as

```
likelihood <- function(phi) {
  u <- flow.utility(phi)
  v <- choice.value(u, discount, P, 480)
  #v <- choiceValue(u,feasible.action, discount, P, 480)
  sumExpV <- matrix(NA,nrow=dim(v)[1],ncol=dim(v)[2]) # T by S
  for(t in 1:dim(v)[1]) {
    sumExpV[t,] <- rowSums(exp(v[t,,])*feasible.action)
  }
  ccp <- exp(v[cbind(plantData$age, plantData$state.index,
                     plantData$action.index)]) /
          sumExpV[cbind(plantData$age,plantData$state.index)]
  return(sum(log(ccp),na.rm=TRUE))
}
```

For organizing states and actions into array indices, it is useful to assign each state vector and combination of actions a scalar index. Actions are already a scalar, so it's easy for them, but states are a vector so some works is needed. The following code implements a mapping from state vector to indices and vice-versa. It may be useful for writing the flow.utility and choice.value functions.

```
# Given vector of variables that define a state, create a function
# that returns an index for each unique combination of them, and a
# function that given an index returns a state vector.
# The two resulting functions are inverses of one another
vector.index.converter <- function(data, state.vars) {
  nv <- rep(NA,length(state.vars))
  state.levels <- list()
  for (s in 1:length(state.vars)) {
    state.levels[[s]] <- sort(unique(data[,state.vars[s]]))
    nv[s] <- length(state.levels[[s]])
  }
  si <- function(state) {
    stopifnot(length(state)==length(nv))
    sn <- as.numeric(state)
    fac <- 1
    index <- 0
    for (i in 1:length(nv)) {
      index <- index + (sn[i]-1)*fac
      fac <- fac*nv[i]
    }
    return(index+1)
  }

  sv <- function(index) {
```

```
    stopifnot(index<=n.states)
    state <- data[1,state.vars]
    li <- rep(NA,length(state.vars))
    fac <- prod(nv)
    index <- index - 1
    for (i in 1:length(nv)) {
      li[i] <- index %% nv[i] + 1
      index <- index %/% nv[i]
      state[[i]] <- state.levels[[i]][li[i]]
    }
    state[[1]] <- state.levels[[1]][li[1]]
    return(state)
  }
  return(list(index=si, vector=sv))
}

state.fn <- vector.index.converter(plantData,state.vars)
action.fn <- vector.index.converter(plantData, action.vars)

# state.fn$vector(i) returns a state vector with index i
# state.fn$index(state) returns index for state vector
```

(1) Check the value function by verifying that the choice specific value varies sensibly with the state and action. You can use the value of ϕ estimated by Rust and Rothwell. (Their estimates include month dummies, but we will leave them out). You should recreate something like figures 13 and 14.

(2) Your initial code for the likelihood function might be very slow. To identify what part of the code is taking the most time, we can use R's profiler.

```
Rprof("rrLike.prof", line.profiling=TRUE) # start the profiler
likelihood(phi.post)
Rprof(NULL) # stop the profiler

summaryRprof("rrLike.prof") # show the results
```

What part of the likelihood function is slow? Can you speed it up? https://csgillespie.github.io/efficientR/performance.html and https://www.r-bloggers.com/strategies-to-speedu provide some good suggestions for speeding up R code.

- When all else fails, time consuming portions of R code can be rewritten in C++, which can be much faster. A speedup of 100x is not unusual. I wrote a version of the choice.value function in C++ in the file choiceValue.cpp. You need not understand this code. To compile the code on Windows, you must first install (separately not as an R package) the Rtools program. The following code snipper compiles the C++ code and makes it callable from R.

```
library(Rcpp)
Sys.setenv(PKG_CXXFLAGS="-O3")
sourceCpp("choiceValue.cpp",showOutput=TRUE,verbose=TRUE,rebuild=TRUE)
```

It creates a function `choiceValue(u,feasible.action,discount,P,T)` which should give the same result as the choice.value function written in R.

(3) Profile the likelihood using your optimised code. Compare the results with your original code.

Problem 5 (Estimation): Now with a likelihood that hopefully does not take too long to evaluate, we can try to maximize the likelihood. For example,

```
library("nloptr") # library for optimization

phi.names <-
  c("exit","refuel","f.r","duration","shutdown","run1.25","run26.50","run51.75","
      run76.99","run100",
    "f.s","f.100")
# parameter estimates from Rust & Rothwell (1995)
phi.pre <- c(0,-1.82, -2.33, -0.05, -0.04, -1.82, -0.96, -0.15, 1.52,
             2.93, -4.03, -3.44)
names(phi.pre) <- phi.names
phi.post <- c(0, -3.44, -3.09, -0.06, -0.54, -2.12, -1.58, -0.74,
              0.54, 2.93, -4.04, -5.89)
names(phi.post) <- phi.names

# payoff of exit is set to 0 and run100 to 2.93 as in RR
free.params <- c("refuel", "f.r", "duration", "shutdown", "run1.25",
                 "run26.50", "run51.75","run76.99", "f.s","f.100")

# bounds on parameters - for some parameter values can easily get
# overflow in the value calculation from taking exp(v[t+1,,]),
# bounding the parameters to a reasonable set ensures this doesn't
# occur during maximization
phi.lb <- phi.post
phi.lb[] <- -Inf
phi.ub <- phi.post
phi.ub[] <- Inf

phi.lb["exit"] <- 0
phi.ub["exit"] <- 0
phi.ub[c("f.r","f.s","f.100")] <- 0 # forced outages are costly
phi.ub[c("refuel")] <- 0 # refueling is costly
phi.ub["duration"] <- 0 # longer duration is costly
phi.ub[c("shutdown","run1.25","run26.50",
         "run51.75","run76.99","run100")]<- 2.93 # no action pays
                                         # more than running 100%

mle.nm <- nloptr(x0=phi.post[free.params], eval_f=function(x) {
  phi <- phi.post
  phi[free.params] <- x
  -likelihood.cpp(phi, subset(plantData,major.problem.spell==FALSE & spell.type!="
      exit")) },
              opts=list(algorithm="NLOPT_LN_NELDERMEAD",
                  print_level=3,
                  maxeval=100000,
                  xtol_rel=1e-6,
                  xtol_abs=1e-12),
              lb=phi.lb[free.params],
              ub=phi.ub[free.params])
```

(1) Explore the sensitivity of the maximization to initial value and/or the choice of optimization algorithm. How confident are you in the results? How do the results compare to those of Rust and Rothwell (1995)?
(2) Assess the fit the estimates by producing something like figures 11 and 12 of Rust and Rothwell (1995).

Problem 6 (Extensions): This problem is meant to be challenging. You should make an honest attempt to complete at least one the parts. You need not answer more than one part.

(1) Inference: read carefully the section on inference in the solutions to assignment 1. Use one or more methods to calculate confidence regions for the parameters of this model. Add confidence bands to your versions of figures 11 and 12 of Rust and Rothwell (1995).
(2) Alternate estimation method: estimate this model using an alternative estimator. The estimator used above in maximum likelihood with a nested fixed point. Possible estimation approaches that have been discussed in lecture include: (i) discrete Euler equations, (ii) a 2-step estimator for dynamic discrete choice, such as pseudo likelihood, (iii) MPEC. Compare the results. Comment on any notable differences in programming difficulty or computation time.
(3) New data: the NRC's website contains daily "Power Reactor Status Reports" going back to 1999. https://www.nrc.gov/reading-rm/doc-collections/event-status/reactor-status/ Scrape this website to create a dataset similiar to the one used by Rust and Rothwell (1995). Estimate the model using this updated data. Discuss the differences.

References

Rothwell, Geoffrey Scott and John Philip Rust. 1995. "A Dynamic Programming Model of U.S. Nuclear Power Plant Operations." Microeconomics, EconWPA. URL http://EconPapers.repec.org/RePEc:wpa:wuwpmi:9502001.

Rust, John and Geoffrey Rothwell. 1995. "Optimal response to a shift in regulatory regime: The case of the US nuclear power industry." Journal of Applied Econometrics 10:75. URL http://search.proquest.com.ezproxy.library.ubc.ca/docview/218751608?accountid=14656. Name - Nuclear Regulatory Commission; Copyright - Copyright Wiley Periodicals Inc. Dec 1995; Last updated - 2011-10-21; CODEN - JAECET; SubjectsTermNotLitGenreText - US.