Appendix A: R code to generate the data and estimate the opt-out models

This appendix presents the code written in the R statistics program [25] needed to generate the DCE datasets, estimate the models, and conduct the Monte Carlo analysis presented in this paper. It is intended to provide a resource for practitioners who are currently using, or considering to use, DCEs and are keen to explore potential opt-out effects present in their own data. Indeed, while the code relates to the analysis presented in this paper, it can easily be adapted to suit other DCE datasets.

The syntax to generate all DCE datasets is given in Box A1. We define the `generate.choices` function, which has three arguments: the random number generator is controlled using the argument `random.seed`; the DCE dataset of interest (i.e., a value 1–3 to generate data to mirror the multinomial logit model with an opt-out constant, nested logit model and independent availability logit, respectively) is specified using the argument `dataset`; and, the opt-out definition (i.e., a value 1–3 for none, baseline and participant-specific, respectively) is defined by using the argument `optout.defn`.[7] For the purposes of illustration, we use an orthogonal main-effects experimental design. The `generate.choices` function returns an object that is a list that includes `random.seed` as well as the following components: `n.tasks` for the number of experimentally designed choice tasks per participant; `alt.1`, `alt.2` and `alt.3`, which give the matrix of attribute levels in first, second and third alternative, respectively (`alt.3` is the opt-out alternative); `choice`, which is the vector of simulated choice outcomes; and, `parameters`, which gives the dataset-specific vector of parameters used to generate the data.

The syntax for our candidate models is given in Box A2. The first of these models, `mnlasc.model`, conforms with the multinomial logit model with an opt-out alternative-specific constant, as in Eq. 3. A nested logit model, as described in Eq. 4, is expressed in the function `nested.model`. The function `ial.model` is an independent availability logit model, as expressed in Eq. 5. Finally, `combined.model` is the combined specification, as outlined in Eq. 6. Each model function has the argument `coeff`, which is the vector of unknown parameters that maximises the log-likelihood.

[7] Within the `generate.choices` function we use the package **fExtremes** [40] to generate random errors draws from the type I extreme value distribution and the package **support.CEs** [41] for generating the experimental design generated.

```r
## function for generating a synthetic DCE dataset
generate.choices <- function(random.seed, dataset, optout.defn) {
  treatment.parameters <- true.parameters
  if (treatment %in% c(2:3, 5:6, 8:9)) {
    treatment.parameters[5] = 0
  }
  if (treatment %in% c(1, 3, 4, 6, 7, 9)) {
    treatment.parameters[6] = 1
  }
  if (treatment %in% c(1:2, 4:5, 7:8)) {
    treatment.parameters[7:9] = c(0, 0, 1)
  }
  ## package for generating generalized extreme value distributions
  require(support.CEs)
  ## generate a L^MA experimental design
  exp.design <- Lma.design(attribute.names = attributes, nalternatives = 2, nblocks = 2, row.renames = FALSE, seed = random.seed)
  n.tasks <- exp.design$design.information$nquestions
  n.blocks <- exp.design$design.information$nblocks
  alt.1 <- matrix(as.numeric(as.matrix(exp.design$alternatives$alt.1[rep(1:(n.tasks * n.blocks), N/n.blocks), -c(1:3)])), ncol = 4)
  alt.2 <- matrix(as.numeric(as.matrix(exp.design$alternatives$alt.2[rep(1:(n.tasks * n.blocks), N/n.blocks), -c(1:3)])), ncol = 4)
  ## generate treatment opt-out alternative (alternative 3)
  if (treatment %in% 1:3) {
    alt.3 <- matrix(0, nrow = N * n.tasks, ncol = 4)
  }
  if (treatment %in% 4:6) {
    alt.3 <- matrix(rep(c(0, 1, 0, attributes$Cost[1]), N * n.tasks), ncol = 4, byrow = TRUE)
  }
  if (treatment %in% 7:9) {
    set.seed(random.seed + 1)
    alt.3 <- cbind(rep(sample(attributes$Efficacy, N, replace = TRUE), each = n.tasks), rep(sample(attributes$Effects, N, replace = TRUE), each = n.tasks),
      rep(sample(attributes$Monitoring, N, replace = TRUE), each = n.tasks), rep(sample(attributes$Cost, N, replace = TRUE), each = n.tasks))
  }
  ## generate participant-specific deviate for processing rules - U(0,1)
  set.seed(random.seed + 2)
  processing.deviate <- rep(runif(N), each = n.tasks)
  processing.strategy.1 <- ifelse(processing.deviate <= treatment.parameters[7], 1, 0)
  processing.strategy.2 <- ifelse(processing.deviate > treatment.parameters[7] & processing.deviate <= sum(treatment.parameters[7:8]), 1, 0)
  ## multiplying indicator variable with minus infinity (set here as log(1e-100))
  ignore.alt1 <- log(1e-100) * processing.strategy.1
  ignore.alt2 <- log(1e-100) * processing.strategy.1
  ignore.alt3 <- log(1e-100) * processing.strategy.2
  ## generate observable component of utility
  v1 <- alt.1 %*% treatment.parameters[1:4] + ignore.alt1
  v2 <- alt.2 %*% treatment.parameters[1:4] + ignore.alt2
  v3 <- alt.3 %*% treatment.parameters[1:4] + treatment.parameters[5] + ignore.alt3
  ## package for generating generalized extreme value distributions
  require(fExtremes)
  if (treatment %in% c(2, 5, 8)) {
    set.seed(random.seed + 3)
    error.nonSQ.nest <- rgev(N * n.tasks, xi = 0, mu = 0, beta = sqrt(1 - treatment.parameters[6]^2))
    set.seed(random.seed + 4)
    error1 <- rgev(N * n.tasks, xi = 0, mu = 0, beta = treatment.parameters[6]) + error.nonSQ.nest
    set.seed(random.seed + 5)
    error2 <- rgev(N * n.tasks, xi = 0, mu = 0, beta = treatment.parameters[6]) + error.nonSQ.nest
  }
  if (treatment %in% c(1, 3:4, 6:7, 9)) {
    set.seed(random.seed + 3)
    error1 <- rgev(N * n.tasks, xi = 0, mu = 0, beta = 1)
    set.seed(random.seed + 4)
    error2 <- rgev(N * n.tasks, xi = 0, mu = 0, beta = 1)
  }
  set.seed(random.seed + 6)
  error3 <- rgev(N * n.tasks, xi = 0, mu = 0, beta = 1)
  ## utility
  utility1 <- v1 + error1
  utility2 <- v2 + error2
  utility3 <- v3 + error3
  ## choice variable
  choice <- apply(cbind(utility1, utility2, utility3), 1, which.max)
  list(random.seed = random.seed, n.tasks = n.tasks, alt.1 = alt.1, alt.2 = alt.2, alt.3 = alt.3, choice = choice, parameters = treatment.parameters)
}
```

```r
## function for multinomial logit model with alternative-specific constant
mnlasc.model <- function(coeff) {
  util1 <- choice.data$alt.1 %*% coeff[1:4]
  util2 <- choice.data$alt.2 %*% coeff[1:4]
  util3 <- choice.data$alt.3 %*% coeff[1:4] + coeff[5]
  choice.prob <- (exp(util1) * (choice.data$choice == 1) + exp(util2) * (choice.data$choice == 2) + exp(util3) * (choice.data$choice == 3))/(exp(util1) +
    exp(util2) + exp(util3))
  choice.prob.prod <- apply(matrix(choice.prob, nrow = n.tasks), 2, prod)
  log(choice.prob.prod)
}
## function for nested logit model
nested.model <- function(coeff) {
  util1 <- choice.data$alt.1 %*% coeff[1:4]
  util2 <- choice.data$alt.2 %*% coeff[1:4]
  util3 <- choice.data$alt.3 %*% coeff[1:4]
  iv <- log(apply(exp(cbind(util1, util2)), 1, sum))
  mu <- 1/(1 + abs(coeff[5]))
  marginal.probs <- exp(cbind(mu * iv, util3))/apply(exp(cbind(mu * iv, util3)), 1, sum)
  conditional.probs <- cbind(exp(cbind(util1, util2)/mu)/apply(exp(cbind(util1, util2)/mu), 1, sum), 1)
  choice.prob <- conditional.probs[, 1] * marginal.probs[, 1] * (choice.data$choice == 1) + conditional.probs[, 2] * marginal.probs[, 1] *
    (choice.data$choice == 2) + conditional.probs[, 3] * marginal.probs[, 2] * (choice.data$choice == 3)
  choice.prob.prod <- apply(matrix(choice.prob, nrow = n.tasks), 2, prod)
  log(choice.prob.prod)
}
## function for independent availability logit model
ial.model <- function(coeff) {
  util1 <- choice.data$alt.1 %*% coeff[1:4]
  util2 <- choice.data$alt.2 %*% coeff[1:4]
  util3 <- choice.data$alt.3 %*% coeff[1:4]
  prob.alts <- exp(c(0, coeff[5:6]))/sum(exp(c(0, coeff[5:6])))
  choice.prob.prod1 <- apply(matrix(ifelse(choice.data$choice == 3, 1, 0), nrow = n.tasks), 2, prod)
  choice.prob2 <- (exp(util1) * (choice.data$choice == 1) + exp(util2) * (choice.data$choice == 2))/(exp(util1) + exp(util2))
  choice.prob.prod2 <- apply(matrix(choice.prob2, nrow = n.tasks), 2, prod)
  choice.prob3 <- (exp(util1) * (choice.data$choice == 1) + exp(util2) * (choice.data$choice == 2) + exp(util3) * (choice.data$choice == 3))/(exp(util1) +
    exp(util2) + exp(util3))
  choice.prob.prod3 <- apply(matrix(choice.prob3, nrow = n.tasks), 2, prod)
  choice.prob.prod <- prob.alts %*% rbind(choice.prob.prod1, choice.prob.prod2, choice.prob.prod3)
  log(choice.prob.prod)
}
## function for combined model
combined.model <- function(coeff) {
  util1 <- choice.data$alt.1 %*% coeff[1:4]
  util2 <- choice.data$alt.2 %*% coeff[1:4]
  util3 <- choice.data$alt.3 %*% coeff[1:4] + coeff[5]
  iv <- log(apply(exp(cbind(util1, util2)), 1, sum))
  mu <- 1/(1 + abs(coeff[6]))
  prob.alts <- exp(c(0, coeff[7:8]))/sum(exp(c(0, coeff[7:8])))
  marginal.probs <- exp(cbind(mu * iv, util3))/apply(exp(cbind(mu * iv, util3)), 1, sum)
  conditional.probs <- cbind(exp(cbind(util1, util2)/mu)/apply(exp(cbind(util1, util2)/mu), 1, sum), 1)
  choice.prob.prod1 <- apply(matrix(ifelse(choice.data$choice == 3, 1, 0), nrow = n.tasks), 2, prod)
  choice.prob2 <- (exp(util1) * (choice.data$choice == 1) + exp(util2) * (choice.data$choice == 2))/(exp(util1) + exp(util2))
  choice.prob.prod2 <- apply(matrix(choice.prob2, nrow = n.tasks), 2, prod)
  choice.prob3 <- conditional.probs[, 1] * marginal.probs[, 1] * (choice.data$choice == 1) + conditional.probs[, 2] * marginal.probs[, 1] *
    (choice.data$choice == 2) + conditional.probs[, 3] * marginal.probs[, 2] * (choice.data$choice == 3)
  choice.prob.prod3 <- apply(matrix(choice.prob3, nrow = n.tasks), 2, prod)
  choice.prob.prod <- prob.alts %*% rbind(choice.prob.prod1, choice.prob.prod2, choice.prob.prod3)
  log(choice.prob.prod)
}
```

Before generating the synthetic DCE datasets, in Box A3, we specify the attributes and their levels, the true parameters, the number of synthetic participants, as well as the number of replications. For readers wishing to execute the syntax for testing purposes, we note that smaller values for `N` and, especially, `n.replications` will significantly reduce computational time.[8]

For each replication, we retrieve the parameter estimates that maximise the log-likelihood of our candidate models for every treatment (multiple replications can be achieved by varying the argument `random.seed` in the `generate.choices` function). The syntax for this process is also given in Box A3.[9]

[8] For those interested in Since the experimental design is specified to consist of two blocks, `N` needs to be an even number.

[9] All models are estimated using the package **maxLik** [35]. It is important to be mindful of the vulnerability to local maxima meaning that there can be uncertainty that some of these models reach a unique maximum. Thus, to reduce the possibility of reaching a local maximum, rather than a global maximum, it is advisable to start the estimation iterations from a variety of random starting points. We did this for the analysis presented

```r
## define the attributes
attributes <- list(Efficacy = 0:1, Effects = 0:1, Monitoring = 0:1, Cost = 1:4)
## specify true parameters for simulation
true.parameters <- c(1.5, -0.9, 1.1, -0.5, 0.3, 0.5, 0.3, 0.2, 0.5)
names(true.parameters) <- c("beta.Efficacy", "beta.Effects", "beta.Monitoring", "beta.Cost", "gamma", "mu", "phi.c1", "phi.c2", "phi.c3")
## number of synthetic participants
N <- 350
## number of replications
n.replications <- 1000
## package for maximum likelihood estimation
require(maxLik)
## create an array to store the results
results <- array(0, c(11, 4, 3, 3, n.replications))
## generation of choices and model estimation for each treatment and replication
for (r in 1:n.replications) {
  for (d in 1:3) {
    for (f in 1:3) {
      treatment <- (d - 1) * 3 + f
      ## generate choices
      choice.data <- generate.choices(random.seed = r, dataset = d, optout.defn = f)
      n.tasks <- choice.data$n.tasks
      ## estimate the candidate models
      mnlasc.result <- maxBFGS(mnlasc.model, start = choice.data$parameters[1:5])
      nested.result <- maxBFGS(nested.model, start = c(choice.data$parameters[1:4], (1 - choice.data$parameters[6])/true.parameters[6]))
      ial.result <- maxBFGS(ial.model, start = c(choice.data$parameters[1:4], ifelse(d <= 2, 0, -0.4), ifelse(d <= 2, 10, 0.5)))
      combined.result <- maxBFGS(combined.model, start = c(choice.data$parameters[1:5], (1 - choice.data$parameters[6])/true.parameters[6],
        ifelse(d <= 2, 0, -0.4), ifelse(d <= 2, 10, 0.5)))
      ## store the parameter estimates
      results[c(1:7, 9), 1, d, f, r] <- c(mnlasc.result$maximum, 5, mnlasc.result$estimate, 1)
      results[c(1:6, 8:9), 2, d, f, r] <- c(nested.result$maximum, 5, nested.result$estimate[1:4], 1/(1 + abs(nested.result$estimate[5])),1)
      results[c(1:6, 9:11), 3, d, f, r] <- c(ial.result$maximum, 6, ial.result$estimate[1:4], exp(c(0, ial.result$estimate[5:6]))/
        sum(exp(c(0, ial.result$estimate[5:6]))))
      results[, 4, d, f, r] <- c(combined.result$maximum, 8, combined.result$estimate[1:5], 1/(1 + abs(combined.result$estimate[6])),
        exp(c(0, combined.result$estimate[7:8]))/sum(exp(c(0, combined.result$estimate[7:8]))))
    }
  }
}
```

This syntax produces a five-dimensional array called `results` with eleven rows to store the log-likelihood, number of parameters and the estimated parameters, four columns (one for each candidate model), three slices (one for each of the DCE data generation process) in the third dimension, three slices (one for each of the opt-out definition) in the fourth dimension and one thousand slices (one for each replication) in the fifth dimension of the array. For example `results[,4,2,3,17]` contains the log-likelihood, number of parameters and estimated parameters for the combined model for the nested logit DCE dataset with participant-specific opt-out levels in the seventeenth replication.

in the paper, however given our desire to keep the syntax as succinct as possible to avoid confusion, this process is not shown in Box A3. This said, our own evaluations reveal that any susceptibility to local maxima of these models across all treatments does not appear to influence the main results.