An introduction to programming a patient level simulation (PLS) in R

Dr Joe Moss

Senior Statistician

October 2020

Providing Consultancy &
Research in Health Economics

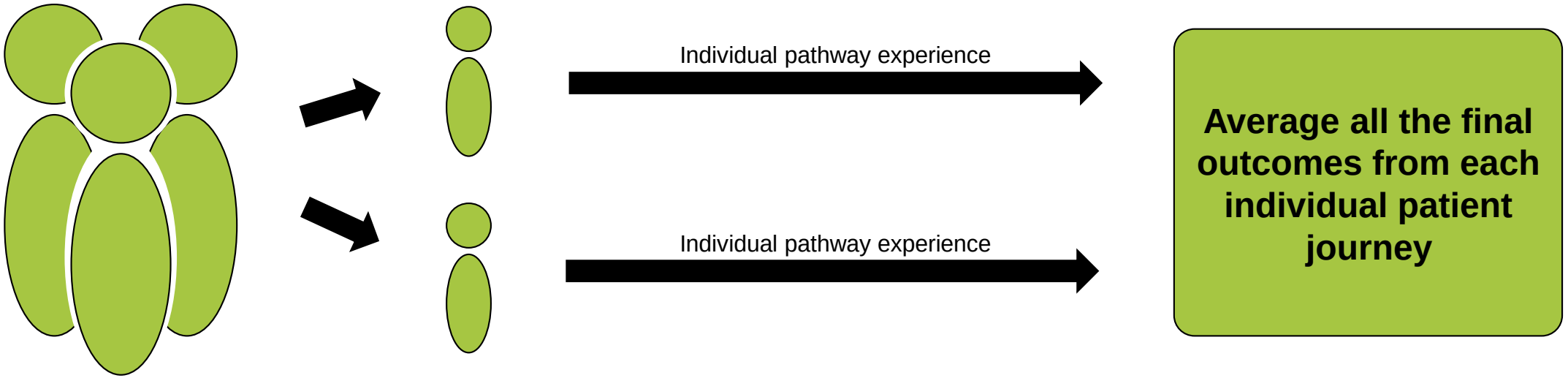
UNIVERSITY of York

Investors in People | Health & Wellbeing Award

YHEC
York Health Economics Consortium

What is a patient level simulation (PLS) model?

- A type of model in which outcomes are estimated for modelled patients one at a time.
- Outcomes are based on a random selection of patients.
- Estimate the mean costs and benefits for a group of people by considering the costs and benefits for each individual patient in the cohort.



Why would you use a PLS model?

- Allows individual patient histories to be recorded and incorporated.
- Model non-linear relationship between patient characteristics and model outcomes.
- Often considered more intuitive or more flexible than cohort models.

- However they often require additional computational power.

NICE DSU TECHNICAL SUPPORT DOCUMENT 15: COST-EFFECTIVENESS MODELLING USING PATIENT-LEVEL SIMULATION

REPORT BY THE DECISION SUPPORT UNIT

April 2014

Sarah Davis, Matt Stevenson, Paul Tappenden, Allan Wailoo

School of Health and Related Research, University of Sheffield

Original Research Article | Published: 27 October 2016

Simulation Modelling in Ophthalmology: Application to Cost Effectiveness of Ranibizumab and Aflibercept for the Treatment of Wet Age-Related Macular Degeneration in the United Kingdom

Lindsay Claxton, Robert Hodgson, Matthew Taylor, Bill Malcolm & Ruth Pulikottil Jacob

PharmacoEconomics **35**, 237–248(2017) | Cite this article

722 Accesses | **9** Citations | **2** Altmetric | Metrics

Abstract

Background

Previously developed models in ophthalmology have generally used a Markovian structure. There are a number of limitations with this approach, most notably the ability to base patient outcomes on best-corrected visual acuity (BCVA) in both eyes, which may be overcome using a different modelling structure. Simulation modelling allows for this to be modelled more precisely, and therefore may provide more accurate and relevant estimates of the cost effectiveness of ophthalmology interventions.

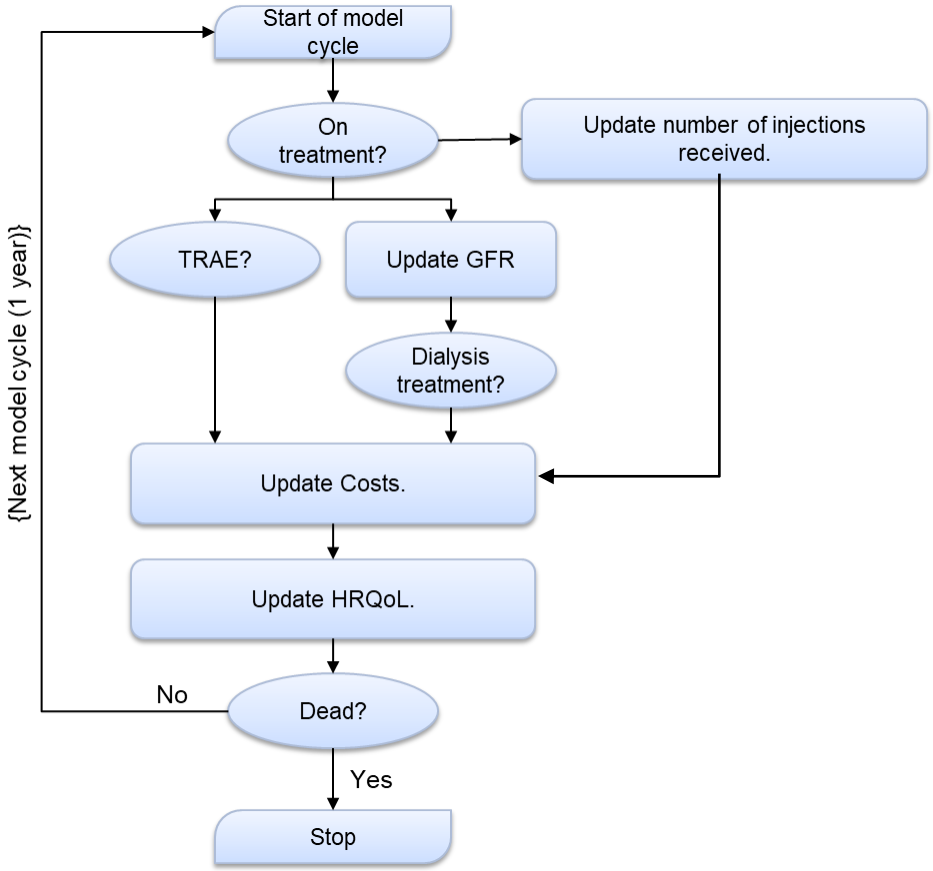
How would you create a PLS model?

- There are three main approaches to PLS modelling:
  - Decision tree – Each individual follows a unique path through the decision tree base on statistical distributions.
  - State-transition models – Uses statistical distributions to determine whether a patient experiences a particular transition.
  - Discrete Event Simulation (DES) – Uses time to event data to schedule patient events, outcomes are evaluated each time an event occurs.

- All approaches:
  - Translate the patient pathway into a series of sequential events.
  - Track the paths of individual patients.
  - Discount costs and QALYs accrued over time.

Example model structure

- Today's example model is completely fictional.
- Decision tree based PLS.
- Treating patients with chronic kidney disease with five possible therapies designed to delay dialysis.



Overview of the steps involved in creating a PLS

Step 1

Simulate patient characteristics

Step 2

Code functions to turn patient characteristics into outcomes.

Step 3

Loop the functions over each patient.

Step 4

Condense simulation results.

Step 5

Analyse outcomes.

Step 6

Model stability validation checks.

Patient numbers and characteristics

- Determining the number of patients to simulate can be tricky.
- It's a balance of the impact of simulating the patient and the computational time required.
- For example, the 10,000th patient is going to be more impactful than the 1,000,000th patient.

- One way to reduce the number of patients that need to be simulated is to use each set of patient characteristics in each treatment arm (i.e. generate them prior to the simulation).
- This means you're not relying on the same patient characteristics to be generate randomly.

- Use clinical trial based summary data to simulate patient characteristics.
  - This can be done independently or dependently.
- Loop the function to generate characteristics for the required number of patients.
- Store results in a list for future use.

Patient numbers and characteristics (Part 2)

```r
## Baseline character function ==========================================================================================

# Generates random patient characteristics

func_Patient_Char_Gen <- function(age, # BL age (mean and SD)
                                  sex, # gender
                                  gfr, # BL gfr (mean and SD)
                                  coda, # Treatment effect coda
                                  gfr.decline, # rate of decline in gfr when not on treatment
                                  mortality.rate, # individual mortality rate
                                  injection.info, # shape and scale info for injection numbers
                                  hrqol.info){ # coefficients and cholesky

  ## Step 1 - Create a list to fill

  Temp_List <- list(
    BL_Age = 0,
    Sex = 0,
    GFR = 0,
    GFR_Decline = 0,
    Coda = 0,
    Treatment_Effect = list(
      Drug_X = 0,
      Drug_A = 0,
      Drug_B = 0,
      Drug_C =0,
      Drug_D =0
    ),
    Mortality_Rate = 0,
    Injection_N = list(
      Drug_X = 0,
      Drug_A = 0,
      Drug_B = 0,
      Drug_C =0,
      Drug_D =0
    ),
    HRQoL_Info = list(
      Intercept = 0,
      Log_GFR = 0,
      Sex = 0,
      Cholesky = 0
    )
  )
```

Patient numbers and characteristics (Part 3)

```r
  ## Step 2 - generate results

  Temp_List[["BL_Age"]] <- rnorm(1, age[1], age[2])
  Temp_List[["Sex"]] <- sample(c(0, 1), 1, prob = c(sex, 1 - sex))
  Temp_List[["GFR"]] <- rnorm(1, gfr[1], gfr[2])
  Temp_List[["GFR_Decline"]] <- rnorm(1, gfr.decline[1], gfr.decline[2])

  Temp_List[["Coda"]] <- sample(c(1:length(coda$Drug_A)), 1)
  Temp_List[["Treatment_Effect"]][["Drug_X"]] <- coda$Drug_X[Temp_List[["Coda"]]]
  Temp_List[["Treatment_Effect"]][["Drug_A"]] <- coda$Drug_A[Temp_List[["Coda"]]]
  Temp_List[["Treatment_Effect"]][["Drug_B"]] <- coda$Drug_B[Temp_List[["Coda"]]]
  Temp_List[["Treatment_Effect"]][["Drug_C"]] <- coda$Drug_C[Temp_List[["Coda"]]]
  Temp_List[["Treatment_Effect"]][["Drug_D"]] <- coda$Drug_D[Temp_List[["Coda"]]]

  Temp_List[["Mortality_Rate"]] <- rnorm(1, mortality.rate[1], mortality.rate[2])

  Temp_List[["Injection_N"]][["Drug_X"]] <- rgamma(n = 1, shape = injection.info[1], rate = injection.info[6])
  Temp_List[["Injection_N"]][["Drug_A"]] <- rgamma(n = 1, shape = injection.info[2], rate = injection.info[7])
  Temp_List[["Injection_N"]][["Drug_B"]] <- rgamma(n = 1, shape = injection.info[3], rate = injection.info[8])
  Temp_List[["Injection_N"]][["Drug_C"]] <- rgamma(n = 1, shape = injection.info[4], rate = injection.info[9])
  Temp_List[["Injection_N"]][["Drug_D"]] <- rgamma(n = 1, shape = injection.info[5], rate = injection.info[10])


  Temp_List[["HRQoL_Info"]][["Cholesky"]] <- as.vector(t(hrqol.info[[4]]))


  ## Step 3 - Generate some random coefficient for the HRQoL regression

  z <- rnorm(n = 3, mean = 0, sd = 1)

  Temp_List[["HRQoL_Info"]][["Intercept"]] <- hrqol.info[[1]] + sum(Temp_List[["HRQoL_Info"]][["Cholesky"]][1:3]*z)
  Temp_List[["HRQoL_Info"]][["Log_GFR"]] <- hrqol.info[[2]] + sum(Temp_List[["HRQoL_Info"]][["Cholesky"]][4:6]*z)
  Temp_List[["HRQoL_Info"]][["Sex"]] <- hrqol.info[[3]] + sum(Temp_List[["HRQoL_Info"]][["Cholesky"]][7:9]*z)

  ## Step 4 - Return list

  return(Temp_List)

}
```

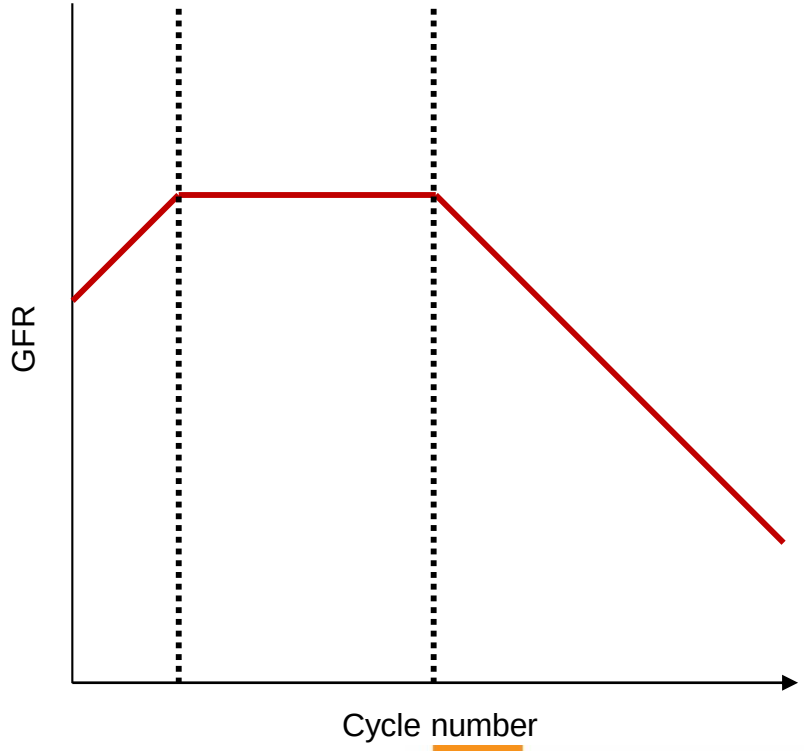
Patient numbers and characteristics (Part 4)

Out_Patient_Char[[1]]	list [9]	List of length 9
BL_Age	double [1]	74.03882
Sex	double [1]	1
GFR	double [1]	38.33483
GFR_Decline	double [1]	2.55211
Coda	integer [1]	5790
Treatment_Effect	list [5]	List of length 5
Drug_X	double [1]	16.9626
Drug_A	double [1]	11.81244
Drug_B	double [1]	10.80533
Drug_C	double [1]	16.61799
Drug_D	double [1]	14.60989
Mortality_Rate	double [1]	1.04999
Injection_N	list [5]	List of length 5
Drug_X	double [1]	4.927932
Drug_A	double [1]	8.620437
Drug_B	double [1]	7.017668
Drug_C	double [1]	7.36421
Drug_D	double [1]	7.488229
HRQoL_Info	list [4]	List of length 4
Intercept	double [1]	0.005600082
Log_GFR	double [1]	0.1795698
Sex	double [1]	-0.03957668
Cholesky	double [9]	0.004732 -0.001152 -0.000540 0.000000 0.000414 -0.001505 ...

Create a series of functions and wrap them in a parent function

- Create a series of functions which will translate patient characteristics into outcomes over time.
- These functions should follow an individual patient.
- Wrap them in a parent function that can be used to loop over all patients.

```
## GFR over time ==========================================================================================

# Simple loop to track a patient's GFR depending on treatment status

func_Treat_Effect <- function(bl.gfr, gfr.decline, treat.effect, treat.name, cycle.num, treat.check,...){

  # Calc Baseline GFR and treatment effect once instead of every time its needed

  GFR_Increase <- bl.gfr + treat.effect[[treat.name]]

  # Loop and logic

  V_GFR_Change = vector(mode = "numeric", length = max(cycle.num)+1)

  for(i in 1:(max(cycle.num)+1)){

    if(cycle.num[i] == 0){V_GFR_Change[i] <- bl.gfr}

    else if(cycle.num[i] > 0 & treat.check[i] == 1){V_GFR_Change[i] <- GFR_Increase}

    else if(cycle.num[i] > 0 & treat.check[i] == 0){V_GFR_Change[i] <- max(V_GFR_Change[i-1] - gfr.decline, 0)}

  }

  # return output

  return(V_GFR_Change)

}
```



Parallel computing can increase speed of main simulation

- Once all the necessary functions are set up to model individual patients, it needs to be efficiently looped over all patients.
- Parallel computing can have a substantial impact on running times (especially when using large patient numbers).
- Packages “doSNOW” and “foreach” can be used to implement parallel computing.

Step 1 – Set up a parallel computing cluster

```
cl <- parallel::makePSOCKcluster(parallel::detectCores() - 1)
registerDoSNOW(cl)
```

Parallel computing can increase speed of main simulation (Part 2)

Step 2 – Run the main simulation function for every patient

```
Temp_List_Drug_X <- foreach(
  i = 1:Input_MS_Patient_Number,
  .packages = c("purrr"),
  .export = c(ls(.GlobalEnv)),
  .inorder = FALSE
) %dopar% {
  Func_Patient_Results_Single(
    treat.name = "Drug_X",
    time.horizon = Input_MS_Time_Horizon,
    bl.age =  Out_Patient_Char[[i]][["BL_Age"]],
    treat.len = Input_MS_Treat_Length,
    inj.num = Out_Patient_Char[[i]][["Injection_N"]],
    bl.gfr = Out_Patient_Char[[i]][["GFR"]],
    gfr.decline = Out_Patient_Char[[i]][["GFR_Decline"]],
    treat.effect = Out_Patient_Char[[i]][["Treatment_Effect"]],
    mort.rate = Out_Patient_Char[[i]][["Mortality_Rate"]],
    sex = Out_Patient_Char[[i]][["Sex"]],
    mort.gen = Input_Mort_Gen,
    trae.stroke = Input_TRAE_Stroke,
    trae.mi = Input_TRAE_MI,
    dialysis.start = Input_MS_Dialysis_Threshold,
    cost.trt = Input_RC_Treatment,
    cost.stroke = c(Input_RC_Stroke, Input_RC_Stroke_LT),
    cost.mi = c(Input_RC_MI, Input_RC_MI_LT),
    cost.dialysis = Input_RC_Dialysis,
    discon.rate = c(Input_DC_Cost, Input_DC_Benefits),
    hrqol.info = Out_Patient_Char[[i]][["HRQoL_Info"]][-4]
  )
}
```

Information we need to pass to each cluster to run the function.

This simple bit of code tells the foreach package to run the following function across all of the clusters.

The target functions which should be run in parallel.

Outputs are returned as a list.

```
# stop parallel
stopCluster(cl)
```

Always remember to close your clusters when finished.

Parallel computing can increase speed of main simulation (Part 3)

Out_Sim_Results	list [5]	List of length 5
Drug_X	list [5000]	List of length 5000
[[1]]	list [23]	List of length 23
V_Cycle_N	double [61]	0 1 2 3 4 5 ...
V_Age	double [61]	53.8 54.8 55.8 56.8 57.8 58.8 ...
V_Treat_Check	double [61]	0 1 1 1 1 1 ...
V_Inj_N	double [61]	0.00 4.39 4.39 4.39 4.39 4.39 ...
V_GFR	double [61]	47.1 63.9 63.9 63.9 63.9 63.9 ...
V_Mortality	double [61]	0 0 0 0 0 0 ...
V_Mortality_Lag	double [61]	1 1 1 1 1 1 ...
V_AE_Stroke	double [61]	0 0 0 0 0 0 ...
V_AE_Stroke_Lag	double [61]	0 0 0 0 0 0 ...
V_AE_MI	double [61]	0 0 0 0 0 0 ...
V_AE_MI_Lag	double [61]	0 0 0 0 0 0 ...
V_Dialysis	double [61]	0 0 0 0 0 0 ...
V_Cost_Treat	double [61]	0 1010 1010 1010 1010 1010 ...
V_Cost_Stroke	double [61]	0 0 0 0 0 0 ...
V_Cost_MI	double [61]	0 0 0 0 0 0 ...
V_Cost_Dialysis	double [61]	0 0 0 0 0 0 ...
V_Cost_Disc_Treat	double [61]	0 976 1919 2829 3710 4560 ...
V_Cost_Disc_Stroke	double [61]	0 0 0 0 0 0 ...
V_Cost_Disc_MI	double [61]	0 0 0 0 0 0 ...
V_Cost_Disc_Dialysis	double [61]	0 0 0 0 0 0 ...
V_Cost_Disc_Total	double [61]	0 976 1919 2829 3710 4560 ...
V_HRQoL	double [61]	0.656 0.711 0.711 0.711 0.711 0.711 ...
V_HRQoL_Disc	double [61]	0.656 1.343 2.006 2.647 3.266 3.865 ...
[[2]]	list [23]	List of length 23
[[3]]	list [23]	List of length 23

Condense results into a list for easy computing

- After the simulation results have been generated, condense each patient into a single value for each output – makes generating results much faster.
- Calculate a rolling mean for costs and QALYs (for stability checks).

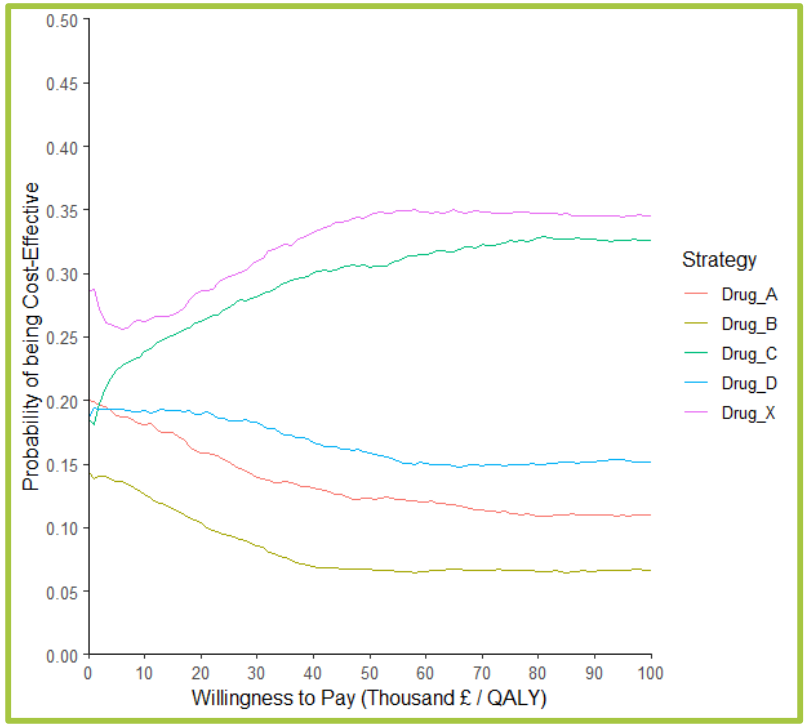
```r
## Condense results ==========================================================================================

# Function to extract key results to display

func_Condense_Results <- function(...){

  message("Condensing results")

  Output <- list(
    Max = list( # Find the max cost and HRQoL for each patient
      Cost = map(.x = Out_Sim_Results, ~map_dbl(.x = .x, ~max(.x[["V_Cost_Disc_Total"]], na.rm = TRUE))),
      HRQoL = map(.x = Out_Sim_Results, ~map_dbl(.x = .x, ~max(.x[["V_HRQoL_Disc"]], na.rm = TRUE)))
    ),
    Mean = list(
      Cost = 0,
      HRQoL = 0
    ),
    Rolling_Mean = list(
      Cost = 0,
      HRQoL = 0
    )
  )

  Output[["Mean"]] <- map(.x = Output[["Max"]], ~map_dbl(.x = .x, ~mean(.x, na.rm = TRUE))) # Find the average for each treatment arm

  Output[["Rolling_Mean"]][["Cost"]] <- map(.x = Output[["Max"]][["Cost"]], ~cummean(.x))
  Output[["Rolling_Mean"]][["HRQoL"]] <- map(.x = Output[["Max"]][["HRQoL"]], ~cummean(.x))

  message("Results condensed")

  return(Output)

}
```

Out_Sim_Summary	list [3]	List of length 3
Max	list [2]	List of length 2
Cost	list [5]	List of length 5
Drug_X	double [5000]	4560 252924 406206 77749 414383 228958 ...
Drug_A	double [5000]	157696 272225 221397 7938 363049 87624 ...
Drug_B	double [5000]	341505 219085 271172 203033 183828 99750 ...
Drug_C	double [5000]	210147 227906 295193 377048 168193 10365 ...
Drug_D	double [5000]	159879 282103 317205 262838 347681 56053 ...
HRQoL	list [5]	List of length 5
Drug_X	double [5000]	6.94 11.49 10.16 11.35 7.97 12.04 ...
Drug_A	double [5000]	11.22 10.69 9.49 11.96 7.42 12.57 ...
Drug_B	double [5000]	8.99 10.03 11.14 10.94 6.02 12.42 ...
Drug_C	double [5000]	11.56 10.76 10.02 7.49 12.49 12.05 ...
Drug_D	double [5000]	11.38 10.86 9.66 11.64 7.47 13.00 ...

Generating results

- After results have been condensed they can be used/plotted very quickly.
- “dampack” by Fernando Alarid-Escudero (gknowlt) can some helpful functions for assessing fully incremental cost-effectiveness.

CEAC

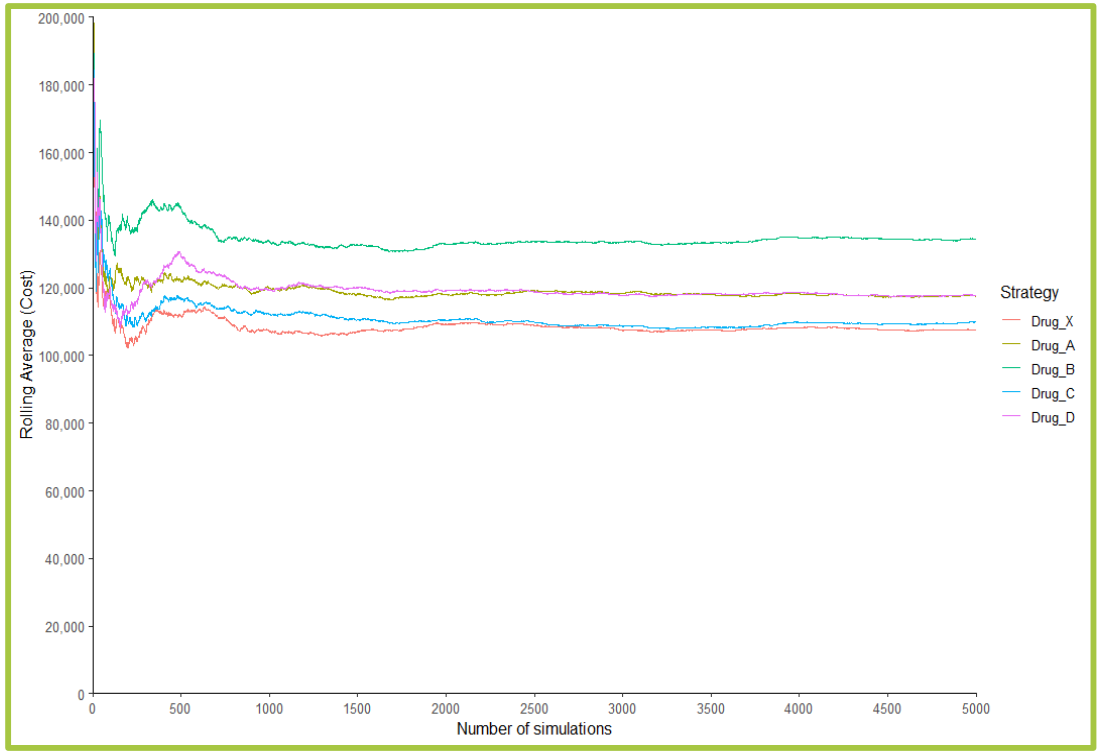


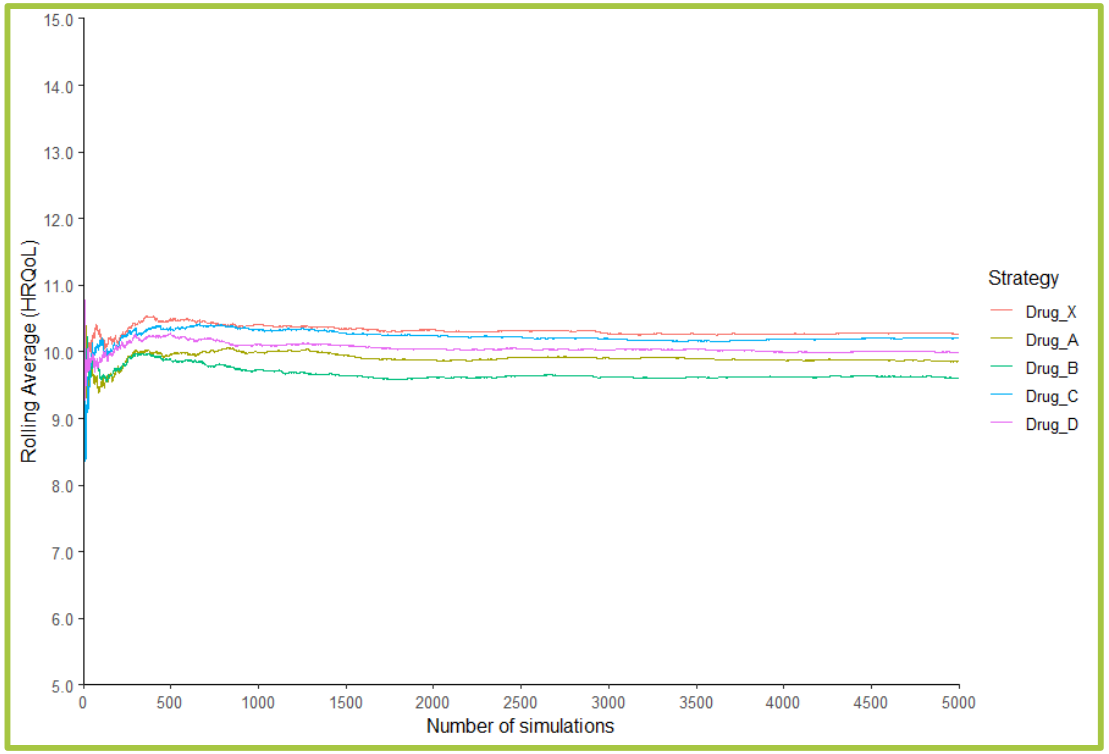
Full incremental cost-effectiveness analysis

Strategy	Cost	Effect	Inc_Cost	Inc_Effect	ICER	Status
Drug_X	107,396	10.27	NA	NA	NA	Non-dominated
Drug_C	109,896	10.21	NA	NA	NA	Dominated
Drug_A	117,576	9.86	NA	NA	NA	Dominated
Drug_D	117,757	9.99	NA	NA	NA	Dominated
Drug_B	134,460	9.61	NA	NA	NA	Dominated

Simulate enough patients to achieve model stability

- Vital step in any PLS is to check model stability.
- Plot the rolling average for total costs and QALYs.





Thank you

Joe.Moss@york.ac.uk

Website: www.yhec.co.uk

http://tinyurl.com/yhec-facebook

http://twitter.com/YHEC1

http://www.linkedin.com/company/york-health-economics-consortium

http://www.minerva-network.com/

Providing Consultancy &
Research in Health Economics

University of York

Investors in People | Health & Wellbeing Award

YHEC
York Health Economics Consortium