Developing *descem*: a Package for Discrete Event Simulation in R

Javier Sanchez Alvarez & Valerie Aponte Ribero
(Hoffmann-La Roche)

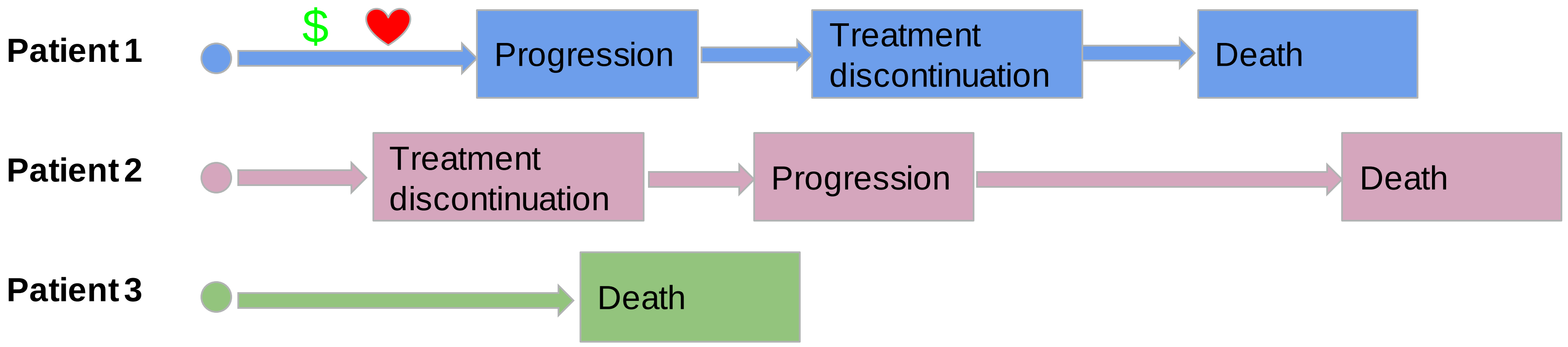
01.07.2021

Why *descem* for Discrete Event Simulation for CEA?

- Context and perspective:
  - DES not really used for HTA submissions but increasingly relevant, harder to approach for modellers, not widespread use relative to other more popular models (AUC, Markov)
  - Current existing packages have a non-HE focus or assume advanced user knowledge
- Our focus:
  - Compare interventions with an interest in LYs/QALYs/Costs and ICERs
  - Industry, HTA bodies, targeted to beginner/intermediate R users
  - Focus on clarity/accessibility, easy adaptation, modelling of capacity constraints are not required

How does DES work?

- Models the system as a series of ‘events’ (e.g. a disease progression or treatment discontinuation) that occur over time
- Moves forward in time at discrete intervals
- Patients modelled as independent entities each of which can be given associated attribute information

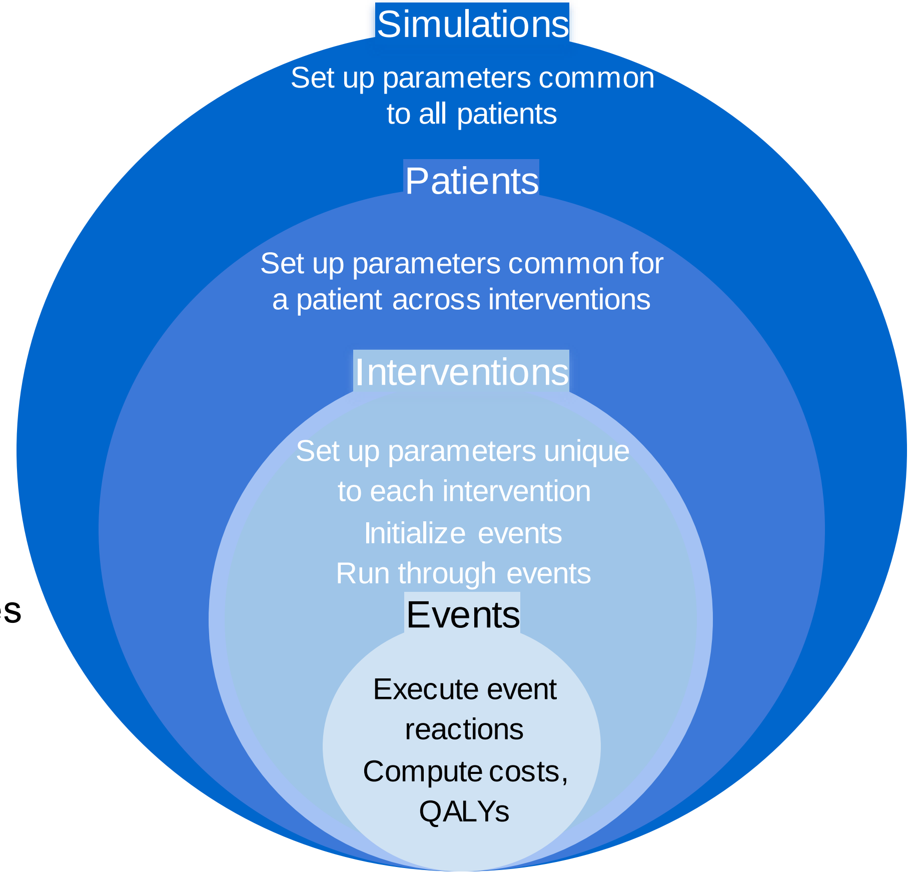


Steps to run a model in *descem*

- Parameters
    - common within simulation (e.g. unit costs)
    - common for a patient across interventions (e.g. patient characteristics)
    - specific to each patient and intervention (e.g. flags)
- Initial events and time to event
- Declaration of reaction to each event
- (optional) Utilities and costs
- Run the model and check results

Model engine: Loop description and order of execution

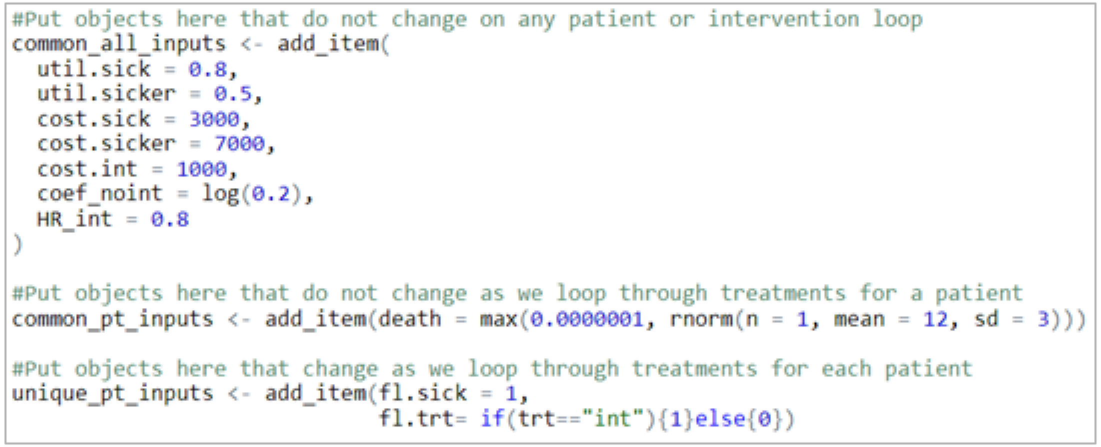
- Four nested loops:
  - per simulation (PSA)
  - per patient
  - per intervention
  - per event
- For each event, compute discounted qalys/lys/costs between previous and current event and then add/update events/items as defined in the reaction
- Each patient is "cloned" for each intervention (same basic characteristics) to compare "apples to apples"


Simulations
Set up parameters common to all patients
Patients
Set up parameters common for a patient across interventions
Interventions
Set up parameters unique to each intervention
Initialize events
Run through events
Events
Execute event reactions
Compute costs, QALYs

What the user sees in *descem*: setting parameters

- Parameters can be assigned at the simulation, patient and intervention level through add_item()
- Features:
  - Use lazy evaluation: all the inputs are only evaluated when the model is run
  - External data can also be introduced directly

Setting parameters through add_item()

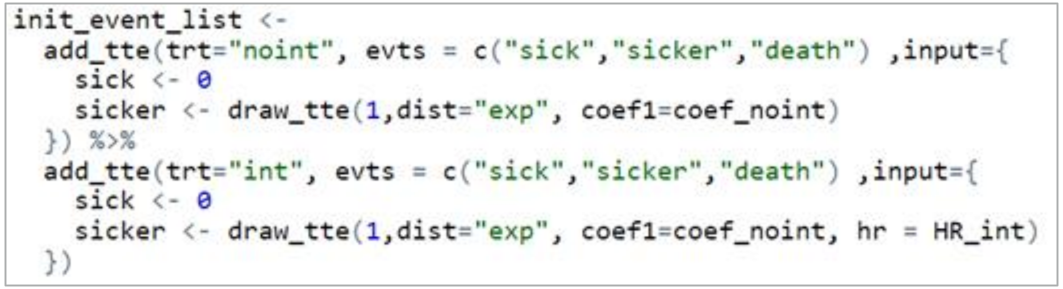


```
#Put objects here that do not change on any patient or intervention loop
common_all_inputs <- add_item(
  util.sick = 0.8,
  util.sicker = 0.5,
  cost.sick = 3000,
  cost.sicker = 7000,
  cost.int = 1000,
  coef_noint = log(0.2),
  HR_int = 0.8
)

#Put objects here that do not change as we loop through treatments for a patient
common_pt_inputs <- add_item(death = max(0.0000001, rnorm(n = 1, mean = 12, sd = 3)))

#Put objects here that change as we loop through treatments for each patient
unique_pt_inputs <- add_item(fl.sick = 1,
                             fl.trt= if(trt=="int"){1}else{0})
```

What the user sees in *descem*: setting events and reactions

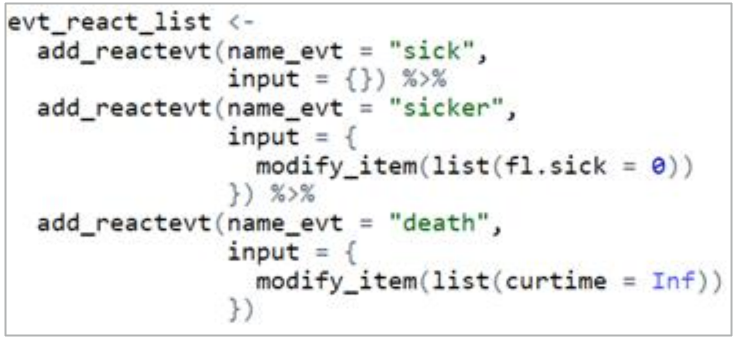
- Initial event times defined by using add_tte()
- Reactions are set for each event type through add_reactevt(). Use modify_item(), new_event() and modify_event() in the reactions
- Use of pipe to chain different interventions or events
- Expressions allow users to have full flexibility in coding. Debugging through browser() can also be implemented in these code chunks (helps verify values!)

Setting initial event times through add_tte()



```r
init_event_list <-
  add_tte(trt="noint", evts = c("sick","sicker","death") ,input={
    sick <- 0
    sicker <- draw_tte(1,dist="exp", coef1=coef_noint)
  }) %>%
  add_tte(trt="int", evts = c("sick","sicker","death") ,input={
    sick <- 0
    sicker <- draw_tte(1,dist="exp", coef1=coef_noint, hr = HR_int)
  })
```

Setting reactions to each event through add_reactevt()

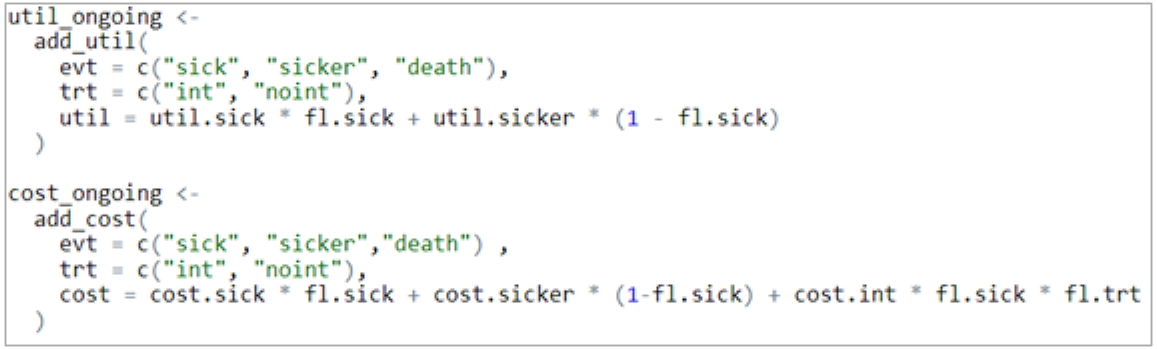


```r
evt_react_list <-
  add_reactevt(name_evt = "sick",
               input = {}) %>%
  add_reactevt(name_evt = "sicker",
               input = {
                 modify_item(list(fl.sick = 0))
               }) %>%
  add_reactevt(name_evt = "death",
               input = {
                 modify_item(list(curtime = Inf))
               })
```

What the user sees in *descem*: adding utilities and costs (optional)

- Utilities and costs are optional and can be added through add_util() and add_cost().
- Utilities and costs can be continuous, instantaneous and through cycles and will be discounted.
- We can define the specific equation that defines the utility/costs.

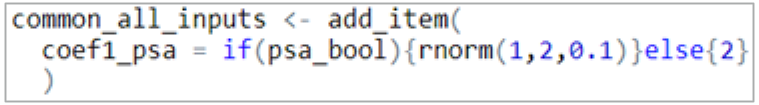
Setting utilities and costs through add_util() and add_cost()



```r
util_ongoing <-
  add_util(
    evt = c("sick", "sicker", "death"),
    trt = c("int", "noint"),
    util = util.sick * fl.sick + util.sicker * (1 - fl.sick)
  )

cost_ongoing <-
  add_cost(
    evt = c("sick", "sicker","death") ,
    trt = c("int", "noint"),
    cost = cost.sick * fl.sick + cost.sicker * (1-fl.sick) + cost.int * fl.sick * fl.trt
  )
```

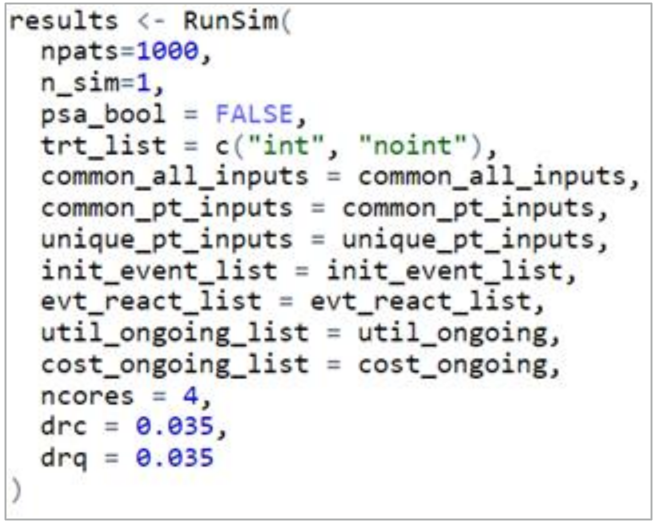
What the user sees in *descem*: running the model

- Run model using RunSim()
- PSA can be easily implemented (boolean variable) →
- Using multiple cores makes computing time manageable (100,000 patients simulated < 100 seconds), though efficiency gains can be expected in the future



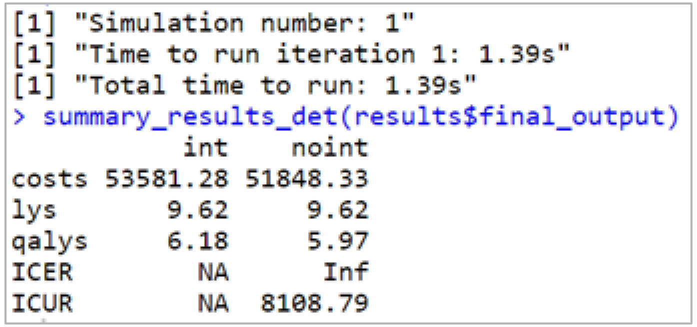
```
common_all_inputs <- add_item(
  coef1_psa = if(psa_bool){rnorm(1,2,0.1)}else{2}
  )
```

The model can be run through RunSim()



```
results <- RunSim(
  npats=1000,
  n_sim=1,
  psa_bool = FALSE,
  trt_list = c("int", "noint"),
  common_all_inputs = common_all_inputs,
  common_pt_inputs = common_pt_inputs,
  unique_pt_inputs = unique_pt_inputs,
  init_event_list = init_event_list,
  evt_react_list = evt_react_list,
  util_ongoing_list = util_ongoing,
  cost_ongoing_list = cost_ongoing,
  ncores = 4,
  drc = 0.035,
  drq = 0.035
)
```

Summary function available

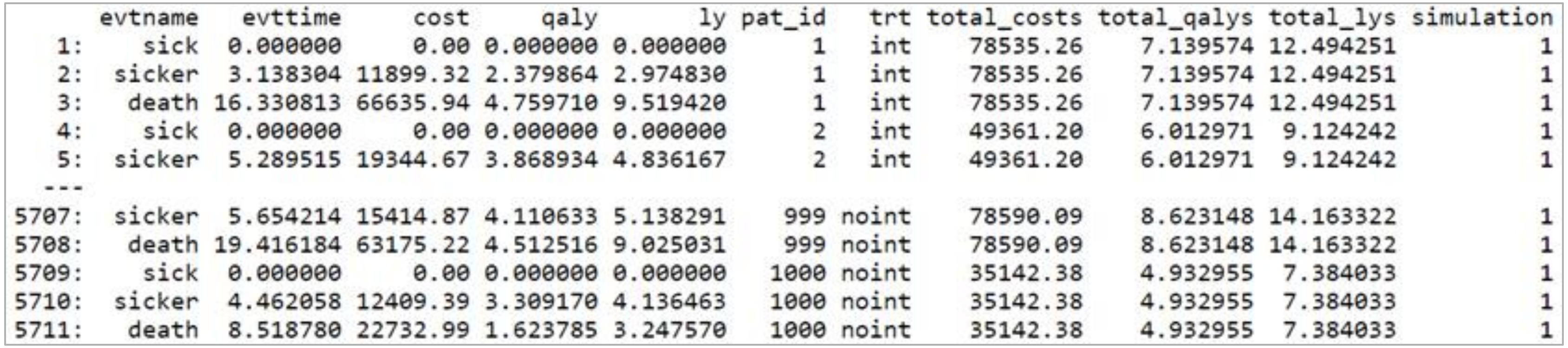


```
[1] "Simulation number: 1"
[1] "Time to run iteration 1: 1.39s"
[1] "Total time to run: 1.39s"
> summary_results_det(results$final_output)
           int    noint
costs 53581.28 51848.33
lys       9.62     9.62
qalys     6.18     5.97
ICER        NA      Inf
ICUR        NA  8108.79
```

What the user sees in *descem*: the output

- The output contains all the simulated data. Additional variables of interest can also be exported through the *input_out* argument in RunSim()

The output has all the relevant information and the user can add any extra desired variable/parameter



```
       evtname   evttime     cost     qaly       ly pat_id   trt total_costs total_qalys total_lys simulation
   1:     sick  0.000000     0.00 0.000000 0.000000      1   int    78535.26    7.139574 12.494251          1
   2:   sicker  3.138304 11899.32 2.379864 2.974830      1   int    78535.26    7.139574 12.494251          1
   3:    death 16.330813 66635.94 4.759710 9.519420      1   int    78535.26    7.139574 12.494251          1
   4:     sick  0.000000     0.00 0.000000 0.000000      2   int    49361.20    6.012971  9.124242          1
   5:   sicker  5.289515 19344.67 3.868934 4.836167      2   int    49361.20    6.012971  9.124242          1
  ---
5707:   sicker  5.654214 15414.87 4.110633 5.138291    999 noint    78590.09    8.623148 14.163322          1
5708:    death 19.416184 63175.22 4.512516 9.025031    999 noint    78590.09    8.623148 14.163322          1
5709:     sick  0.000000     0.00 0.000000 0.000000   1000 noint    35142.38    4.932955  7.384033          1
5710:   sicker  4.462058 12409.39 3.309170 4.136463   1000 noint    35142.38    4.932955  7.384033          1
5711:    death  8.518780 22732.99 1.623785 3.247570   1000 noint    35142.38    4.932955  7.384033          1
```

Learnings from developing the package

- Independent code development + frequent feedback sessions are important
- Focus on clarity/accessibility meant tradeoffs and challenges
    - Understanding the added value (vs. other packages) and setting objectives
    - Accessibility: Thinking of what’s intuitive for the final user
    - Flexibility in model design (e.g. allow numeric, characters, lists, matrices…): one could also run a Markov/hybrid model
    - Speed: standardized processing (e.g. using C++) vs. flexible/general evaluation
        - Profiling helps with “low-hanging fruit” optimizations

QR code to descem Github home site

Conclusion: Why use *descem*?

- *descem* is a new package for DES without capacity constraints for CEA with a focus on accessibility and adaptability
- *descem* can be a good solution for modellers who want both flexibility in disease modelling (where individual patient characteristics matter) and clarity in their code, which facilitates discussions, adaptations, validation and sharing
- Only first steps, still needs to be applied in a real case. Looking for feedback/ideas - community-driven development for wider acceptance.
- Potential new features: efficiency gains in engine, more informative diagnostics, ready to use plots and other CE tools...
- Medium/long term idea is to have a wider community acceptance of package to increase trust among HTA bodies or other stakeholders, reducing burden (validation of engine) and increasing efficiency (focusing discussion on assumptions and inputs)