Seamlesssim Vignette

Introduction

`seamlesssim` is an R package to simulate seamless clinical trials (usually in the oncology setting) and assess some basic operating features of these trials. A seamless trial is one in which the trial aims to estimate both the maximum tolerated dose (MTD) for some treatment and the efficacy of that treatment at the MTD. After completing a seamless trial, trial sponsors may proceed to Phase III with more confidence about what the efficacy of the treatment will be. By using a seamless framework that blends Phases I and II, sponsors may be able to save time, money, and patient resources.

In Boonstra, Braun, and Chase (2020), we gave a broad definition of seamless trials and made some recommendations about features of good seamless trials. In our conception of the seamless framework, we separate the trial into 4 modules. Module 1 provides a first estimate of dose toxicity. In Module 2, efficacy is estimated at the MTD from Module 1. In Module 3, toxicity is re-estimated to obtain a more stable estimate. Finally, in Module 4, efficacy is re-estimated. Although we consider 4 modules, only Module 1 is mandatory. If Modules 2-4 are skipped, the seamless trial reduces to a Phase I trial; if Modules 3-4 are skipped; the seamless trial is a standard Phase I/II trial. Other combinations and configurations can be used to encapsulate most of the Phase I, II, and I/II designs that we have seen in the literature.

We consider several ways to estimate toxicity and efficacy within each of these modules. `seamlesssim` considers 4 different approaches for estimating toxicity in Modules 1 and 3 (the continual reassessment method or CRM; the 3+3 design; a de-escalation scheme; and a fixed dose assignment scheme) and 6 different approaches for estimating efficacy in Modules 2 and 4 (a Bayesian interval test; a Bayesian isotonic regression; an inverted score test; a minimum number of responders threshold; a minimum percentage of responders threshold; or no efficacy analysis). For more information on these approaches and their strengths and weaknesses, please see Boonstra, Braun, and Chase (2020). Here, we demonstrate how to specify designs and trial features in `seamlesssim` in order to simulate a trial with those characteristics, and then how to generate some basic summary statistics of the simulated trials.

Installation

First, we need to install the package by running `devtools::install_github(elizabethchase/seamlesssim)`, then load it into R by running `library(seamlesssim)`. To call up documentation on `seamlesssim`, we can use the ? operator to read more about functions. For example, we could run:

```
library(seamlesssim)
?sim_empiric_dec
```

to obtain information on the `sim_empiric_dec` function. The same could be done for all of the functions in the package.

Specifying Designs and Scenarios

Our first task is to specify the goals of the trial. What is our toxicity threshold, or the maximum amount of toxicity we can tolerate, and how much can we deviate from that? What amount of efficacy do we want to see? In our example, we set our toxicity threshold at 0.25, meaning that the true, unknown MTD is defined as the dose level such that the probability of dose-limiting toxicity is no greater than 25%; however, we allow

toxicity to go 5% above this (`tox_delta_no_exceed` = 0.05) during dose exploration. We set our efficacy threshold at 0.2, so that at the true MTD, the true efficacy rate is at least 20%.

```
primary_objectives <- c(tox_target = 0.25,
                        tox_delta_no_exceed = 0.05,
                        eff_target = 0.20)
```

Next, we specify what we think the true dose/efficacy curves of our treatment might be. Here, we are considering 5 different doses. Hopefully, clinicians will have some sense of what the true values of toxicity and efficacy on the doses are, but we might consider multiple scenarios to be sure. We consider 3 scenarios for how toxicity and efficacy is distributed across our 5 doses:

```
tox_eff_curves <- list(
  list(tox_curve = c(0.08,0.11,0.17,0.25,0.35),
       eff_curve = c(0.10,0.18,0.25,0.35,0.38),
       scenario = 1),
  list(tox_curve = c(0.08,0.11,0.17,0.25,0.35),
       eff_curve = c(0.10,0.10,0.10,0.35,0.35),
       scenario = 2),
  list(tox_curve = c(0.03,0.08,0.18,0.30,0.39),
       eff_curve = c(0.10,0.18,0.25,0.35,0.38),
       scenario = 3)
)
```

To interpret one of these: in scenario 1, we believe that dose 1 has 8% toxicity and 10% efficacy, while dose 5 has 35% toxicity and 38% efficacy.

Finally, we need to specify which designs we want to consider, and we need to provide the necessary parameters for each design we're considering. We will consider 3 different designs. They are:

1. Our first design uses 3+3 in Module 1, and then uses a Bayesian interval test in Module 2. For the interval test, it sets its prior parameters as estimating that all 5 doses will have 10% efficacy (`prior_mean`), but its confidence in this prior is low–`prior_n_per` is only 1. Its threshold is `prob_threshold` = 30%, so in order for the trial to proceed, the posterior probability of any of the doses meeting the efficacy threshold of 20% must be at least 30%. In Module 3, it then uses a dose-deescalation scheme in 35 patients (`n` = 35) to continue checking toxicity, and after 10 patients (`first_patient_look` = 10) it will check to see if toxicity exceeds 33% (`thresh_decrease` = 0.33). If so, it will de-escalate the dose. Finally, the design finishes off in Module 4 by using another Bayesian interval test to check efficacy at the end of the trial, with the same prior as was used in Module 2, but now with a higher posterior probabiltiy threshold of 38%, and using all of the data from Modules 1-4.

2. Our second design uses a CRM to assign doses to 25 patients in Module 1. It sets the CRM skeleton and scale parameters (for more information, see Boonstra, Braun, and Chase (2020) and Cheung (2019)), and requires that at least 3 patients be assigned to each dose before escalation (`dose_cohort_size` = 3), and that at least 6 patients receive doses before the trial can stop for toxicity (`earliest_stop` = 6). It uses a Bayesian interval test to assess efficacy in Module 2, with the same prior as in Design 1, but with a slightly higher posterior probability threshold of 35%. In Module 3, we continue the CRM from Module 1 in an additional 35 patients, to continue to assess toxicity. Finally, in Module 4, we do the same Bayesian interval test as in Module 2, but now using all of the data from Modules 1-5, and with a much higher posterior probability threshold of 75%: in order to pass our efficacy standard, the posterior probability of the MTD meeting our efficacy threshold of 20% must be at least 75%.

3. Our third design uses a CRM in Module 1 with the same parameters as in Design 2. It skips Module 2 entirely (so no interim efficacy analysis) and continues its CRM in Module 3 in an additional 35 patients. Then, in Module 4, it does a final efficacy analysis using a Bayesian isotonic regression. For the isotonic regression, it assumes a very small scale parameter (`alpha_scale` = 1e7), which corresponds to a fairly noninformative prior on the efficacy across the different doses. For more information on this choice, see

Boonstra, Owen, and Kang (2020). The probability of efficacy threshold is quite high (`prob_threshold` = 0.87)–in order to advance to further study, the posterior probability that the MTD is effective in at least 20% of patients must be greater than 87%.

```
design_list = list(
  list(##Design 1
    module1 = list(
      name = "3pl3",
      starting_dose = 1
    ),
    module2 = list(
      name = "bayes",
      prob_threshold = 0.3,
      prior_mean = rep(0.1, 5),
      prior_n_per = 1,
      include_stage1_data = T
    ),
    module3 = list(
      name = "empiric",
      n = 35,
      rule = "local",
      first_patient_look = 10,
      thresh_decrease = 0.33
    ),
    module4 = list(
      name = "bayes",
      prob_threshold = 0.38,
      prior_mean = rep(0.1, 5),
      prior_n_per = 1,
      include_stage1_data = T
    )
  ),
  #
  list(##Design 2
    module1 = list(
      name = "crm",
      n = 25,
      starting_dose = 1,
      skeleton = c(0.08,0.11,0.17,0.25,0.35),
      beta_scale = 0.6,
      dose_cohort_size = 3,
      dose_cohort_size_first_only = T,
      earliest_stop = 6
    ),
    module2 = list(
      name = "bayes",
      prob_threshold = 0.35,
      prior_mean = rep(0.1, 5),
      prior_n_per = 1,
      include_stage1_data = T
    ),
    module3 = list(
      name = "continue_crm",
      n = 35
```

```
    ),
    module4 = list(
      name = "bayes",
      prob_threshold = 0.75,
      prior_mean = rep(0.1, 5),
      prior_n_per = 1,
      include_stage1_data = T
    )
  ),
  #
  list(##Design 3
    module1 = list(
      name = "crm",
      n = 25,
      starting_dose = 1,
      skeleton = c(0.08,0.11,0.17,0.25,0.35),
      beta_scale = 0.6,
      dose_cohort_size = 3,
      dose_cohort_size_first_only = T,
      earliest_stop = 6
    ),
    module3 = list(
      name = "continue_crm",
      n = 35
    ),
    module4 = list(
      name = "bayes_isoreg",
      prob_threshold = 0.87,
      alpha_scale = 1e7,
      include_stage1_data = T
    )
  )
)
```

We label the designs something descriptive, although this could be skipped:

```
design_labels <- c("3pl3:bayes::emp:bayes", "crm:bayes::crm:bayes", "crm::isoreg")
```

Running the Simulator

Now, we run the simulator. Because we aren't running this in parallel, we leave `array_id` equal to 1. If we were calling this function many times in parallel, we would assign a different `array_id` to each run. We set `n_sim` to 5, so we will simulate 5 trials for each scenario and design. Note that because this function is computationally intensive, running more than about 30 simulations will take a while. We input our `primary_objectives`, `design_list`, and `design_labels` from above. We leave the `stan_args` as their default (`NA`), although more experienced STAN users may want to modify these. We leave `sim_labels` as the default, so each simulated trial will get a numeric id, and we leave `do_efficient_simulation` as TRUE, so that simulation results will be repeated for similar designs. We use a `seed` of 1.

Because `twostage_simulator` can only consider one scenario at a time, we call the function three times, with `tox_eff_curves` varying for each true scenario. (Note that for more than 3 scenarios, we may want to do these repeated calls in parallel, or consider using `lapply`.)

```
mysims_scen1 <- twostage_simulator(array_id = 1,
                                   n_sim = 5,
                                   primary_objectives = primary_objectives,
                                   dose_outcome_curves = tox_eff_curves[[1]],
                                   design_list = design_list,
                                   stan_args = NA,
                                   sim_labels = NULL,
                                   design_labels = design_labels,
                                   do_efficient_simulation = T,
                                   random_seed = 1)

mysims_scen2 <- twostage_simulator(array_id = 1,
                                   n_sim = 5,
                                   primary_objectives = primary_objectives,
                                   dose_outcome_curves = tox_eff_curves[[2]],
                                   design_list = design_list,
                                   stan_args = NA,
                                   sim_labels = NULL,
                                   design_labels = design_labels,
                                   do_efficient_simulation = T,
                                   random_seed = 1)

mysims_scen3 <- twostage_simulator(array_id = 1,
                                   n_sim = 5,
                                   primary_objectives = primary_objectives,
                                   dose_outcome_curves = tox_eff_curves[[3]],
                                   design_list = design_list,
                                   stan_args = NA,
                                   sim_labels = NULL,
                                   design_labels = design_labels,
                                   do_efficient_simulation = T,
                                   random_seed = 1)
```

Processing Simulation Results

Each of these function calls will result in a list with many elements. Of these elements, the three most important items for our purposes are:

1. `patient_data`. A complete sheet of all the patient-level data from the simulated trials. This will contain information on which treatment each patient was assigned to, whether or not they had toxicity/response, and whether or not they received the final recommended treatment.
2. `sim_data_stage1`. A sheet with summary information at the end of Stage 1 (Modules 1 and 2). This includes the estimated MTD at the end of Stage 1, any evidence of efficacy at the end of Stage 1, how many patients were enrolled and how many were enrolled to the MTD, and similar.
3. `sim_data_stage2`. The same as `sim_data_stage1`, but at the end of Stage 2 (Modules 1-4).

For users with a good grasp of data management and plotting techniques in R, these data can be used to create most summary statistics and plots that would be useful in a trial protocol. However, for users who are less confident in R (or who want a quick synopsis of the simulation results before doing more personalized descriptives), it may be useful to use the `twostage_results` function, which provides some basic plots and tables from the three datasets described above.

To use the function, we can do one of three things. For a large number of files, we can save the `patient_data`

and `sim_data_stage2` output as `.csv` files in separate folders. For a moderate number of files, we can save the raw output from the `twostage_simulator` function as `.Rds` files in their own dedicated folder. If there are other `.Rds` files in the folder that weren't produced by `twostage_simulator`, the function will not work properly. For a small number of files (and what we do here), we leave all of the files in the workspace and input them to `twostage_results` as a character vector giving their names. We include a comment giving example code for how we would save the data to a folder as `.csv` or `.Rds`for interested users.

```
#How to save the .csv results to patientdatfolder and stage2datfolder and then
#read it into twostage_results:
#write_csv(mysims_scen1$patient_data, path = "patientdatfolder/scen1.csv")
#write_csv(mysims_scen1$sim_data_stage2, path = "stage2datfolder/scen1.csv")
#write_csv(mysims_scen2$patient_data, path = "patientdatfolder/scen2.csv")
#write_csv(mysims_scen2$sim_data_stage2, path = "stage2datfolder/scen2.csv")
#write_csv(mysims_scen3$patient_data, path = "patientdatfolder/scen3.csv")
#write_csv(mysims_scen3$sim_data_stage2, path = "stage2datfolder/scen3.csv")
#myresults <- twostage_results(csv = TRUE,
 #                             stage2folder = "stage2datfolder",
 #                             finaldatfolder = "patientdatfolder",
 #                             dose_outcome_curves = tox_eff_curves,
 #                             primary_objectives = primary_objectives,
 #                             design_labels = design_labels)

#How to save the .Rds results to myfolder and then read it into twostage_results:
#saveRDS(mysims_scen1, file = "myfolder/scen1.Rds")
#saveRDS(mysims_scen2, file = "myfolder/scen2.Rds")
#saveRDS(mysims_scen3, file = "myfolder/scen3.Rds")
#myresults <- twostage_results(csv = FALSE,
 #                             files = "myfolder",
 #                             filepath = TRUE,
 #                             primary_objectives = primary_objectives,
 #                             design_labels = design_labels)

#How to read the results into twostage_results from the workspace:
myfiles <- c("mysims_scen1", "mysims_scen2", "mysims_scen3")
myresults <- twostage_results(files = myfiles,
                              filepath = FALSE,
                              primary_objectives = primary_objectives,
                              design_labels = design_labels)
```

```
## 1
## 2
## 3
```

From `twostage_results`, we get a list containing the items `tables` and `plots`. `tables` contains:

1. `generating_params_for_display`. This is a table giving the true efficacy and toxicity of each dose for each scenario considered.
2. `acc_dose_rec_table`. This is a table giving which proportion of trials for each design and scenario combination recommended an acceptable dose (a dose meeting the toxicity and efficacy standards, if not the best dose) at the end of the trial.
3. `mean_patients_table`. This is a table giving the mean number of patients enrolled for each design and scenario combination across the different simulated trials.

And `plots` contains:

4. `gen_params_plot_blue`. This is the same information as in `generating_params_for_display`,

but in plot form and in a blue color palette. If there are fewer than `design_per_page` designs and `scen_per_page` scenarios, it will be a list with only one element, which can be printed as shown below; for more designs/scenarios, there may be many plots stored in this list.

5. `gen_params_plot_redgreen`. This is the same information as in `generating_params_for_display`, but in plot form and in a red-green color palette. If there are fewer than `design_per_page` designs and `scen_per_page` scenarios, it will be a list with only one element, which can be printed as shown below; for more designs/scenarios, there may be many plots stored in this list.
6. `acc_dose_rec_plot`. This is a list of plots giving which proportion of trials for each design and scenario combination recommended an acceptable dose, unacceptable dose, or made no recommendation at all. If there are fewer than `design_per_page` designs and `scen_per_page` scenarios, it will be a list with only one element, which can be printed as shown below; for more designs/scenarios, there may be many plots stored in this list.
7. `dose_over_time_plot`. This is a list of filled barplots giving the distribution of dose assignments by time, where time is measured by patient number. If there are fewer than `design_per_page` designs and `scen_per_page` scenarios, it will be a list with only one element, which can be printed as shown below; for more designs/scenarios, there may be many plots stored in this list.
8. `n_patients_plot`. This is a boxplot of the number of patients enrolled for each design and scenario combination across the different simulated trials.
9. `n_patients_RP2D_plot`. This is a boxplot of the number of patients who received the final recommended dose for each design and scenario combination across the different simulated trials. Note that the final recommended dose may not be safe or effective–this is merely a measure of how much patient data the design will yield for the final dose it recommends.
10. `prop_patients_accep_plot`. This is a boxplot of the number of patients who received an acceptable dose (a dose meeting the toxicity and efficacy standards, if not the best dose) for each design and scenario combination across the different simulated trials.

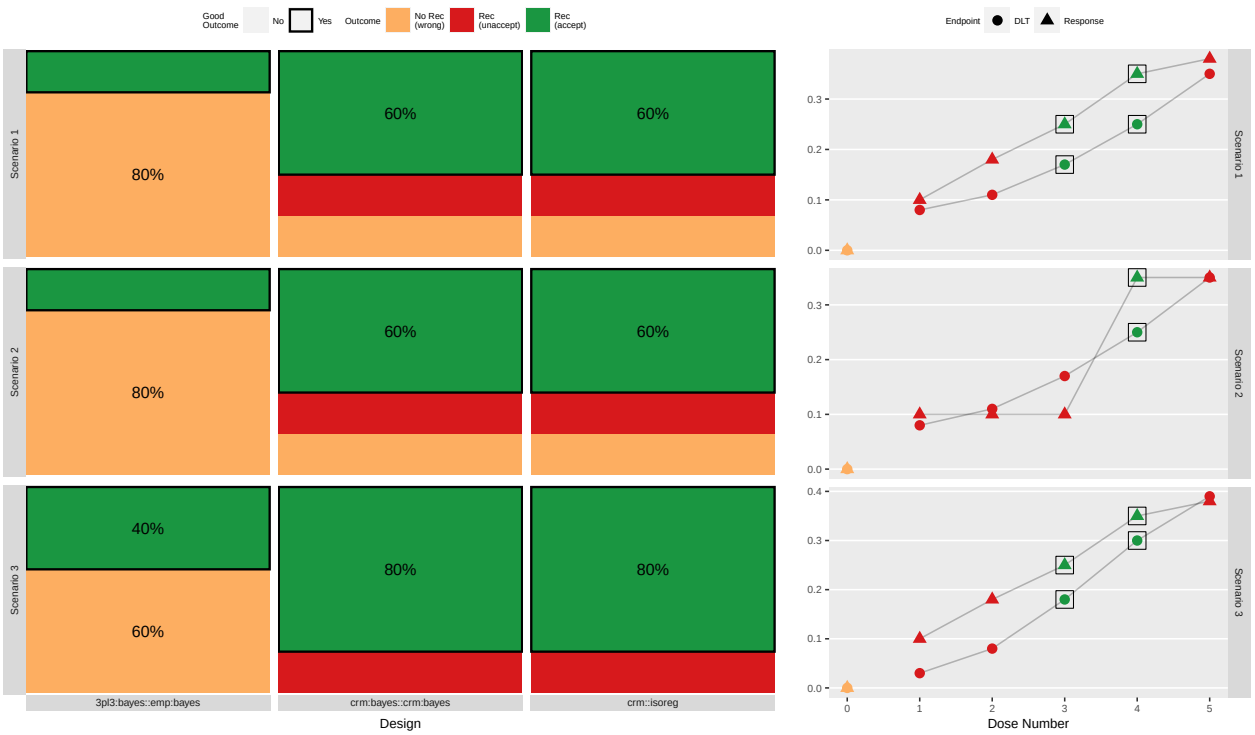
Although we could print the results of `twostage_results` as one big print, we will instead do some formatting to get a more sleek presentation, by using the `kable` package on the tables and the `cowplot` package on the plots. For example:

```
kable(myresults$tables$acc_dose_rec_table, caption="Proportion of Trials That
      Select Acceptable Dose")
```

Table 1: Proportion of Trials That Select Acceptable Dose

scenario	3pl3:bayes::emp:bayes	crm:bayes::crm:bayes	crm::isoreg
Scenario 1	0.2	0.6	0.6
Scenario 2	0.2	0.6	0.6
Scenario 3	0.4	0.8	0.8

```
#We use acc_dose_rec_plot[[1]], gen_params_plot_redgreen[[1]] here to call the first element of each
#list of plots and then place them side by side:
plot_grid(myresults$plots$acc_dose_rec_plot[[1]], myresults$plots$gen_params_plot_redgreen[[1]],
          ncol = 2, nrow=1, rel_widths = c(5,3))
```




```
kable(myresults$tables$mean_patients_table, caption="Mean Number of Patients Enrolled")
```

Table 2: Mean Number of Patients Enrolled

scenario	3pl3:bayes::emp:bayes	crm:bayes::crm:bayes	crm::isoreg
Scenario 1	37.8	60	60
Scenario 2	30.8	60	60
Scenario 3	39.6	60	60

Using Standalone Simulator Functions

Although not the primary purpose of `seamlesssim`, all of the subfunctions used in `twostage_simulator` can be used independently outside of `twostage_simulator`. So it is possible to simulate a simple 3+3 design trial, or a straightforward CRM trial, using the functions `sim_3pl3` and `titesim_phil`, respectively. Here, we demonstrate how to simulate a 3+3 trial using `sim_3pl3`. We would input a toxicity curve, using the toxicity curve from Scenario 1 above. We do 5 simulated trials, as above. Because we are running it outside of `twostage_simulator`, the `stage_label` is unimportant, so we leave it as the default value of 1. We do the same with `sim_specific_start_id` and `sim_specific_dose_start`, so the first patient (rather than the second or forty-second patient) is assigned to the default of dose 1 (rather than dose 2 or 4). We use a seed of 1.

```
tox_curve <- tox_eff_curves[[1]]$tox_curve
my3pl3 <- sim_3pl3(n_sim = 5,
                   true_tox_curve = tox_curve,
                    sim_specific_dose_start = NULL,
                    seed = 1)
```

From this, we get all of the patient data in `all_results`, the estimated MTD in each of our 5 simulated trials in `estMTD`, and the total number of patients enrolled in each simulated trial in `enrollment`. We could use `ggplot2` and other packages to create plots and tables of these results.

Similar analyses can be done using the functions `bayesian_isotonic`, `sim_empiric_dec`, `sim_twostage_crm`, and `titesim_phil`.

References

Boonstra, PS, TM Braun, and EC Chase. 2020. "A Modular Framework for Seamless Oncology Trials." *Arxiv*.

Boonstra, PS, DR Owen, and J Kang. 2020. "The Isotonic Horseshoe Prior for Modeling Binary Outcomes." *Arxiv*.

Cheung, Ken. 2019. *Dfcrm: Dose-Finding by the Continual Reassessment Method*. https://CRAN.R-project.org/package=dfcrm.