Simulation study to test vivax relatedness model

To-do: change plot N_alleles to N_alleles_Choices

Determines whether to run the full suite of simulations (takes a long time to run).

```
RUN_MODELS = F
PLOT_RESULTS = T
```

Simulation 1: Effective Complexity of Infection

We want to know how adding extra noisy parasites into an infection will affect recurrence state inference as a function of the number of markers typed. Note that the evidence for relapse increases with the number of markers, `M`, if `K_poly_markers` is less than `max(M)`:

- when `M <= K_poly_markers` the noisy parasite will be a stranger in relation to the other parasites in the same infection
- when `K_poly_markers < M < 2*K_poly_markers` the noisy parasite will be a sibling of the other parasites in the same infection.
- when `K_poly_markers << M`, the noisy will approach a clone of the other parasites in the same infection, but will be considered a sibling under the model

Outline of simulation is as follows:

- Simulate data for N individuals, with M markers which are polyallelic for a given number K (this controls complexity of problem). We do this for two episodes where there are underlying clonal or sibling relationships between episodes.
- Compute resulting recurrence state estimates
- Plot resulting recurrence state estimates as a function of the problem complexity, and the effective MOI

```
# Setup simulation study parameters
N_alleles_Choices = c(13,4,28) # Marker cardinality (set to match mean and range our panel)
K_indivs = 1000 # Number of indiviuduals
Ms = seq(3,15, by = 3) # Number of markers
K_poly_markers = 3 # Number of polyallelic markers
Tn_fixed = 2 # Total number of episodes
MOI_1_max = 2 # Maximum COI of primary episode
MOI_2_max = 2 # Maximum COI of recurrent episode
CLI_prior = c('C' = 1/3, 'L' = 1/3, 'I' = 1/3) # Discrete uniform prior on recurrence states
States = c('C','I','L')

# Enumerate all combinations of COI complexity and numbers of markers
settings = expand.grid(1:MOI_1_max, 1:MOI_2_max, Ms)
names(settings)= c('MOI_1', 'MOI_2', 'M')
settings = settings[settings$MOI_1+settings$MOI_2<4,]
settings$MOI_pattern <- paste(settings$MOI_1, settings$MOI_2, sep = "_")
JOBS = nrow(settings)
```

```
# All simulation parameter settings possible
if(RUN_MODELS){
  # iterate over cardinality of markers
  for(N_alleles in N_alleles_Choices){
    # iterative over simulation scenario
    for(related_type in c('Sibling','Stranger','Clone')){
      # iterate over parameter settings
```

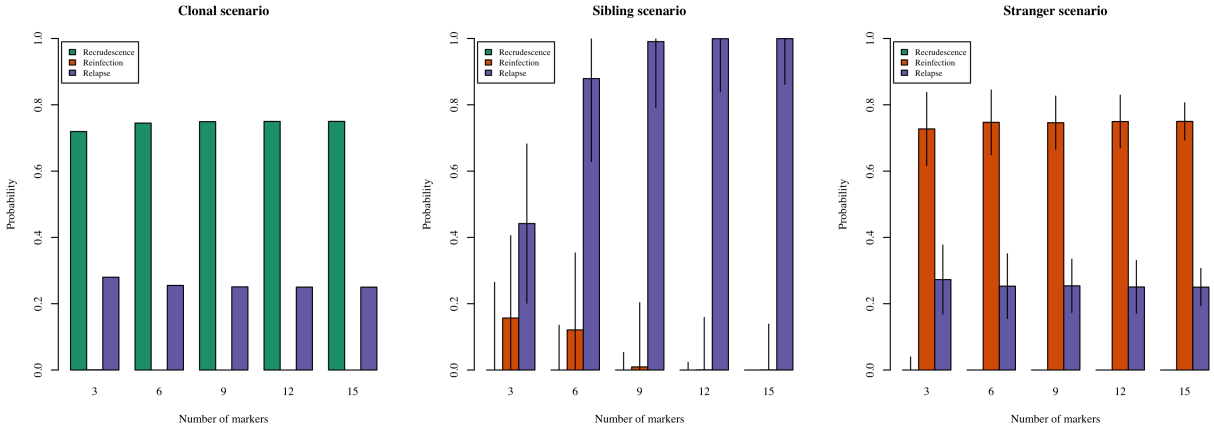
```
      thetas_all = foreach(s = 1:JOBS, .combine = rbind,
                           .packages = c('dplyr','Matrix','gtools',
                                         'igraph','matrixStats','doParallel')
      ) %do% { # parallisation happening inside the function

        MOI_1 = settings$MOI_1[s] # MOI of primary infection
        MOI_2 = settings$MOI_2[s] # MOI of recurrent infection
        M = settings$M[s] # Number of markers

        MS_markers = sapply(1:M, function(x) paste0('MS',x)) # Marker names
        FS = lapply(MS_markers, function(x) table(1:N_alleles)/N_alleles) # Marker frequencies
        names(FS) = MS_markers # Discrete uniform distribution on the alleles

        ########################################################
        # create a dataframe to store the simulated MS data
        ########################################################
        MS_data = BuildSimData(Tn = Tn_fixed,
                               MOIs = c(MOI_1, MOI_2),
                               M = M,
                               N = K_indivs,
                               N_alleles = N_alleles,
                               K_poly_markers = K_poly_markers,
                               relatedness = related_type)
        eps_ids = unique(MS_data$Episode_Identifier[MS_data$Episode>1])
        P_matrix = data.frame(Episode_Identifier=eps_ids,
                              C = rep(CLI_prior['C'],K_indivs),
                              L = rep(CLI_prior['L'],K_indivs),
                              I = rep(CLI_prior['I'],K_indivs))


        ###################################
        # Run the model on the data
        ###################################
        TH = post_prob_CLI(MSdata = MS_data,
                           cores = 7,
                           Fs = FS,
                           verbose = F,
                           p = P_matrix)

        TH$setting = s # Add setting number for plotting
        TH # return results
      }
      writeLines(paste0('********** Done for ',related_type,' **********'))
      fname = paste0('SimulationOutputs/Posterior_Probs_N*=',
                     N_alleles,'_',
                     related_type,'_EffectMOI.RData')
      save(thetas_all, file = fname)
    }
  }
}
```

Plot results

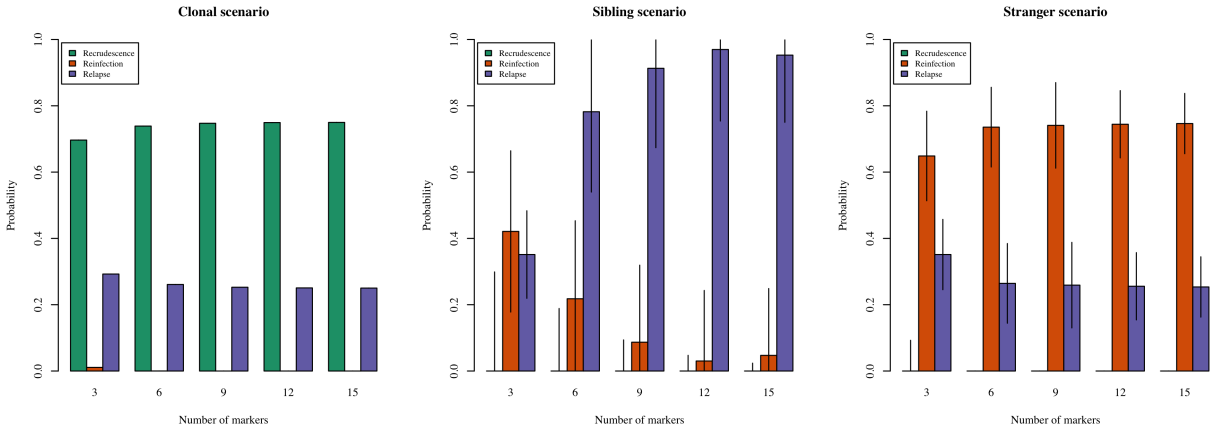
```
if(PLOT_RESULTS){

  # Alternative vizualisation: show one row in main plot
  State_names = list(L='Relapse',I='Reinfection',C='Recrudescence')
  mycols = brewer.pal(length(State_names), 'Dark2')
  related_type_names = c('Clone' = 'Clonal scenario',
                         'Sibling' = 'Sibling scenario',
                         'Stranger' = 'Stranger scenario')
  par(mfrow = c(1,3), family = 'serif', pty = 's', oma = c(0,0,3,0))
  BESIDES = T # Allows visualisation of error


  for(pattern in unique(settings$MOI_pattern)){
    JOBS_pattern <- which(settings$MOI_pattern == as.character(pattern))
    for(N_alleles in c(10,4,5)){ # N_alleles_Choices

      # Iterate over simulation scenarions (types of data)
      for(related_type in names(related_type_names)){

        # Load data
        fname = load(paste0('SimulationOutputs/Posterior_Probs_N*=',N_alleles,'_',
                            related_type,'_EffectMOI.RData'))

        # Summary plot
        X <- sapply(JOBS_pattern, function(x){
          ind <- thetas_all$setting == x
          apply(thetas_all[ind,States], 2, median)
        })

        sds <- sapply(JOBS_pattern, function(x){
          ind <- thetas_all$setting == x
          apply(thetas_all[ind,States], 2, sd)
        })


        # Barplot
        Z = barplot(X, beside = BESIDES, col = mycols[1:3], ylim = c(0,1),
                    names.arg = settings[JOBS_pattern,'M'],
                    ylab = 'Probability', xlab = 'Number of markers',
                    main = related_type_names[related_type])
        legend(ifelse(BESIDES,'topleft','top'), fill = mycols[1:3],
               legend = State_names[States], inset = 0.01,cex = 0.75)

        # Title
        mtext(text = sprintf('Marker cardinality: %s \n COI of first and second infection: %s and %s res
                             N_alleles, strsplit(pattern, split = '_')[[1]][1],
                             strsplit(pattern, split = '_')[[1]][1]), side = 3, outer = T)

        # Add error bars (+/- sd)
        if(BESIDES){
          rownames(Z) <- States
```

```
          for(state in States){
            segments(x0 = Z[state,], x1 = Z[state,],
                     y0 = X[state,] - sds[state,], y1 = X[state,] + sds[state,])
          }
        }
      }
    }
  }
}
```

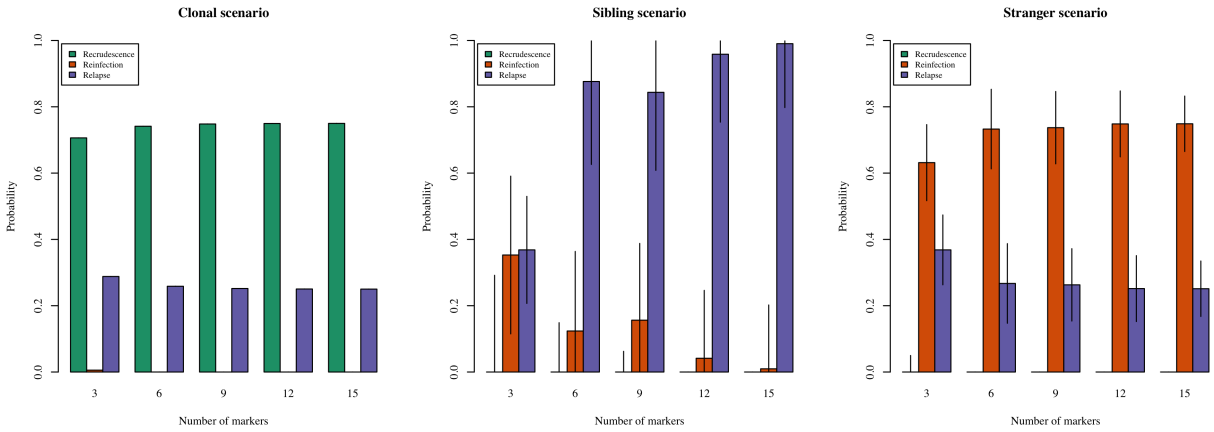
Marker cardinality: 10
COI of first and second infection: 1 and 1 respectively



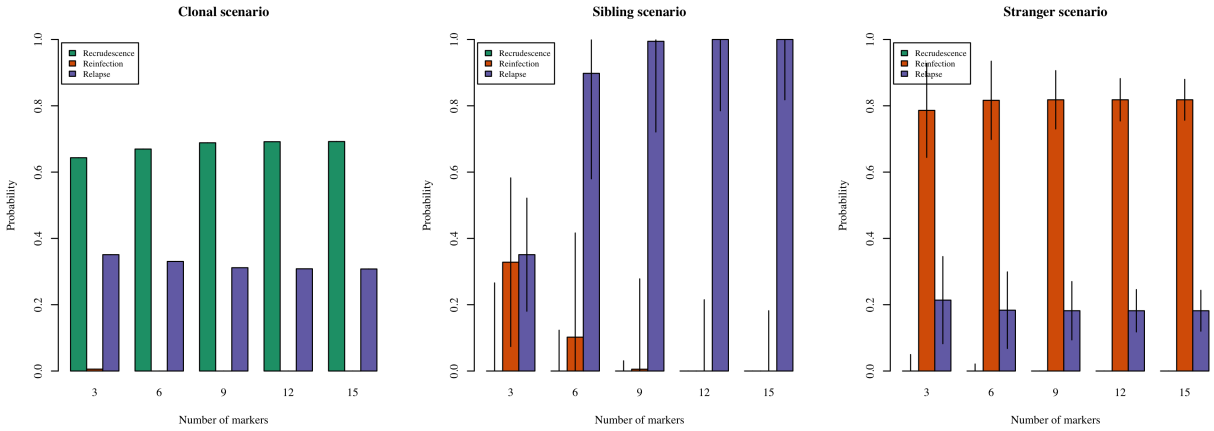
Marker cardinality: 4
COI of first and second infection: 1 and 1 respectively



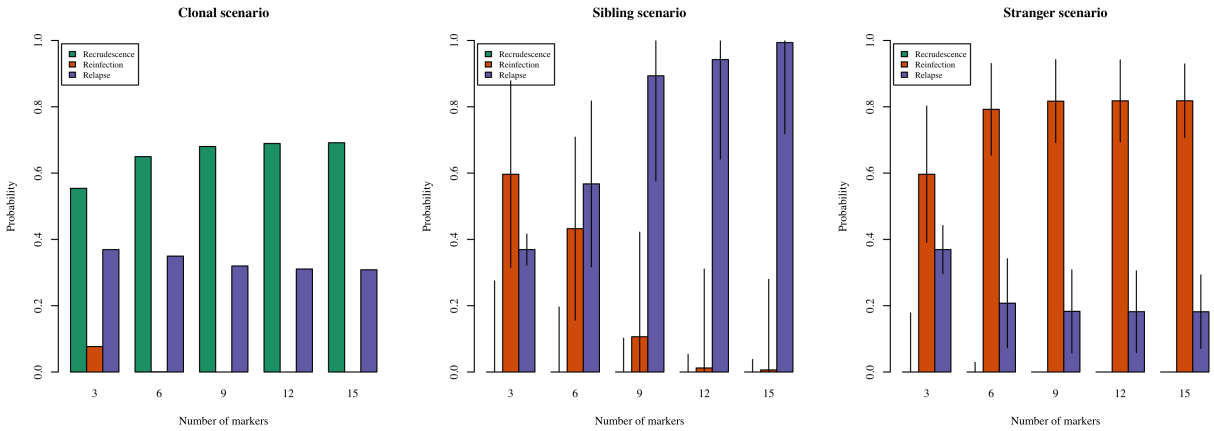
Marker cardinality: 5
COI of first and second infection: 1 and 1 respectively



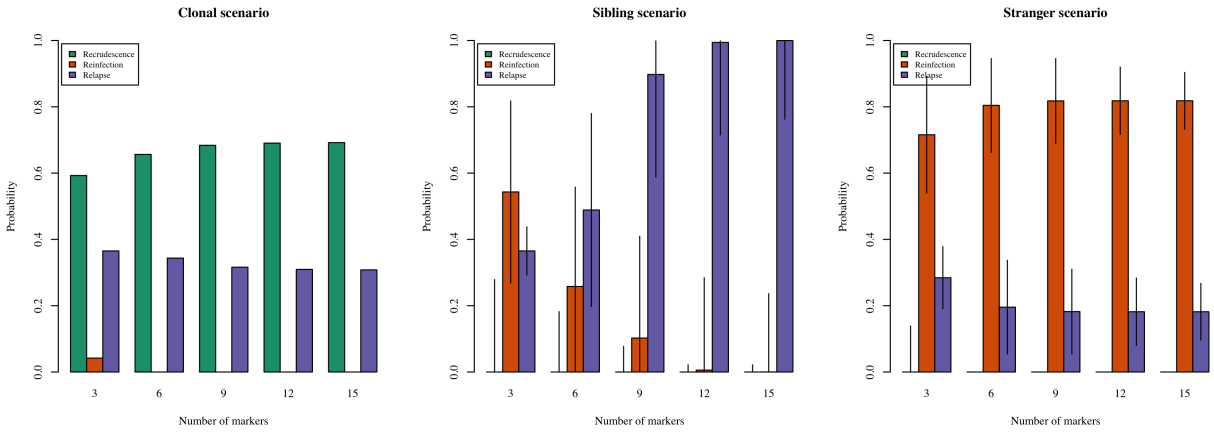
Marker cardinality: 10
COI of first and second infection: 2 and 2 respectively



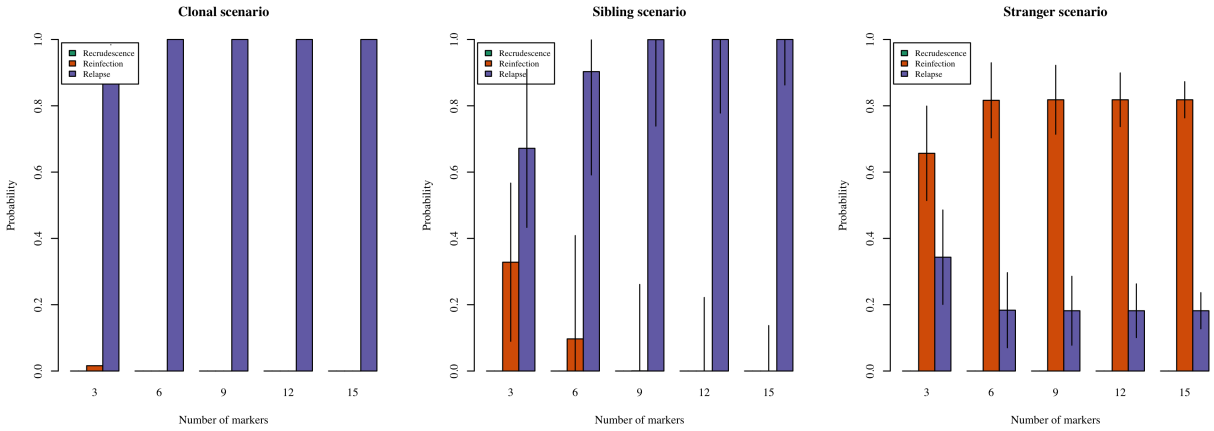
Marker cardinality: 4
COI of first and second infection: 2 and 2 respectively



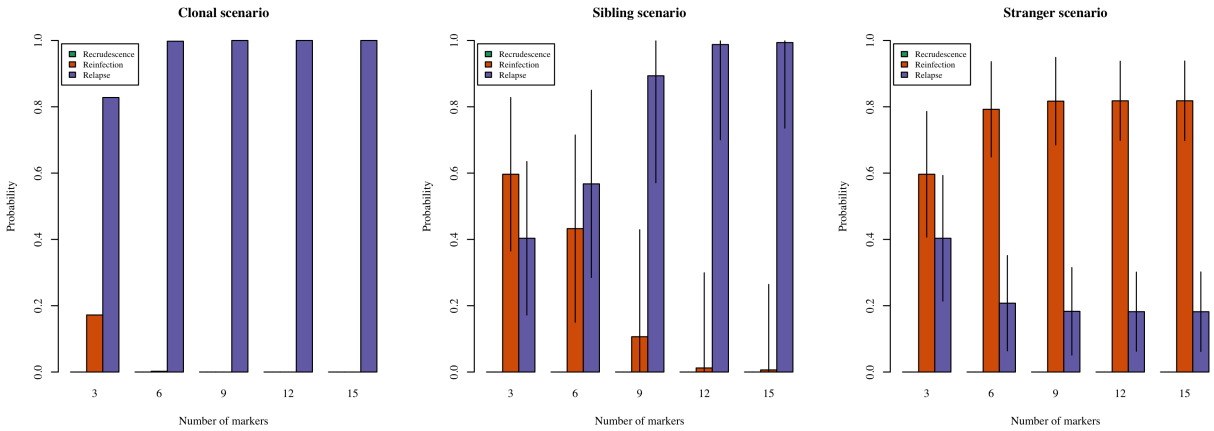
Marker cardinality: 5
COI of first and second infection: 2 and 2 respectively



Marker cardinality: 10
COI of first and second infection: 1 and 1 respectively



Marker cardinality: 4
COI of first and second infection: 1 and 1 respectively



Marker cardinality: 5
COI of first and second infection: 1 and 1 respectively