BUSULFAN CHIMERA CODE DOCUMENTATION

Current version, December 2015

Contents

Introduction

These notes serve as documentation of the busulfan chimera codes used in the PNAS paper. Every attempt has been made to be comprehensive in the sense of not just justifying each step, but trying to illustrate potential pitfalls avoided by doing things a certain way. That being said, the codes themselves are devoid of such examples. This is done to provide as slim and simple a set of codes as possible; the codes provided give the right answer as provided in the paper and do not contain much exploration of failed avenues.

Where analytics (such as ODE solving) are required, Mathematica is used with R being used as much as possible. Mathematica was originally used in almost all cases, including numeric solving, but these routines have since largely been replaced by R equivalents with the forerunner MM codes *not* provided.

Finally, there is much room for improvement in these codes. Not only are there for-loops galore but they are not as modular and adaptive as perhaps they could be. As much as I would have liked to provided 'software' rather than 'codes' where everything can be easily manipulated through runtime variables, this wasn't always possible (though the BrdU codes are much better at this, having been started once I understood R better).

Files

As of December 2015, the files in this set of codes are:

BusulfanPackages.R

Installs and/or loads required packages, is sourced in all other .R files.

DataPrepare.R

A basic data analysis routine which takes the raw counts from the .xlsx files and outputs rescaled donor fractions, total counts, and Ki67 percentage data.

BusulfanNaiveFit.R

First of two main fitting codes. Performs both the best fits and bootstraps. Requires the ratio and total count analytic functions to be hard coded.

AgeDependentLoss.R

Follows what is written in the PNAS paper to deal with the age-structured model.

Analysis.R

A slightly more involved analysis code. Reproduces all significant plots and data tables from the PNAS paper.

DataPrepare.R

Overview

This R file takes in two data files: ChimeraCountsCompiled NO WT.xlsx and ki67Data NO WT.xlsx with the latter only used towards the end of the code and is not required for the majority of outputs, with the former required for *all* calculations. As indicated in the names, these files do not contain wild type data.

This script does not currently take command line/run time arguments and so all modifications must be done to the file itself. Currently the output folder is ROutputs/ and so this must exist (will change).

Outputs

This routine will output the rescaled peripheral donor fraction (scaled to DP1 or SPx, see `thymic.precursors`), the total peripheral counts, the total precursor (DP1 or SPx) counts, and the Ki67+ percentages broken down by host/donor.

Currently we only consider Ki67 data from mice beyond 30wks post-BMT

The rescaled peripheral donor fraction is defined as

$$f_d(t) = \frac{1}{k(t)}\frac{N_d(t)}{N(t)} \tag{1}$$

where $k(t)$ can not only represent the chimerism in the animal (determined at SP or DP1 stages, in which case $k(t) = \chi$ is constant), but can be *any* cell type deemed a suitable precursor for the population in question e.g. naive cells as precursors for memory cells (hence the potential time dependence). This code allows one to specify a set of cells to be normalised only to a population in the thymus, as well as cell types we wish to have *peripheral* precursors *in addition* to thymic precursors.

Fractions rescaled to a thymic population are given for all cell types, and so thymic precursors are required for all cell types.

User defined toggles

The main variables one might like to change are:

`cell.Types.nai`

A vector of cell types to be investigated e.g. `c("4nai", "8nai"')`. In reality, this is a vector of cell types we want to only normalise to precursors in the thymus. Rescaled donor fractions of these cell types are often used by subsequent cell types i.e. memory (thus they are calculated first and separately).

thymic.precursors

A vector of 'precursor' types, each corresponding to an element in cell.types. Change spX to dp1 here to examine different normalisations. All cells, including those in `cell.Types.others` (defined below), require a thymic precursor to be specified.

cell.Types.others

Another vector of cell types, this time for cells we want to normalise to precursors in the periphery as well as the standard thymus normalisation. Currently functions for memory populations, should function just as well for others such as gdT/NKT/etc. A few vectors to fill manually but this gives maximum flexibility i.e. useful to normalise 8Tem to 8Tcm, for example.

Several cell types, notably 8Tem and 8Tcm, have less data than the others. This is handled by setting 'NA's to -1 and then removing all negative values prior to export.

peripheral.precursors

Cell types to act as precursors to cell types listed in `cell.Types.others`. Again, this normalized dataset is exported *in addition* to the standard thymic normalisation (spX or DP1).

cell.Types

Defined as `c("cell.Types.nai","cell.Types.others")`. This is the master cell type list.

t.Shift

Currently set to 42 (days) this determines the time of thymic reconstitution. All data prior to this time point are discarded from all outputs, so to investigate pre-reconstitution counts this needs to be reduced and the code re-run.

organ.Set

Vector of organ types to pool over. Can be LN or SP or both, but must be of type vector with string elements.

naughty.mice

Mouse IDs for mice that were deemed to be aberrant in their peripheral donor fraction behaviour. These remain in the .xlsx data files but are removed from all analysis and fits.

Function glossary (not exhaustive)

`collect.data(cell, organ.list, set)`

Takes in data.frame 'set' and calculates the total host and donor numbers for cell type 'cell' in each of the organs. Used by subsequent functions to calculate the donor frac and total counts.

`get.ki67.numbers(cell, organ, population)`

To get the ki67 positive percentage weighted average across the organs, we first compute the *number* of Ki67+ cells in each organ and sum them. Here 'population' refers to host/donor. This function returns the list `output=list(ki67+ cells, total cells)`.

`export.Table(frame, file.name)`

This function takes in a particular dataframe along with a file name. It assumes the first column of `frame` is a time vector and proceeds to take each subsequent column (corresponding to data for a given cell type) and bind it to this time vector making n frames, exporting each. Prior to export, each of these temporary 2-column frames has all negative values removed. Negative values are introduced to label missing data (i.e. where NAs appear).

BusulfanNaiveFit.R

Overview

This R file takes in the prepared data as well as analytic functions and outputs the best fits and bootstrap parameter estimates for CD4 and CD8 naive cells. Currently this is only able to deal with the naive case or similar, i.e. cells governed by $N'(t) = \Theta(t) - \lambda N(t)$ where $\Theta(t)$ is the involuting thymic input term, and λ is a constant net loss rate.

Once the cell type has been chosen, the code imports three bits of information: the total counts, the rescaled donor fraction, and the (thymic) precursor total counts. The time points in use are taken directly from this data, and if these data sets are of different length or their rows do not correspond to the same mice, errors/incorrect answers will ensue. So long as no sorting is done prior to the importing of the data from DataPrepare.R, everything should be okay.

Derivation of analytics

We begin without assuming the existence of any addition populations, and simply model host and donor cells according to

$$\begin{aligned} N_d'(t) &= \chi\Theta(t) - \lambda N_d(t), \\ N_h'(t) &= (1-\chi)\Theta(t) - \lambda N_h(t). \end{aligned} \tag{2}$$

Once the thymus can be said to have reconstituted (see `t.Shift`) we assume it has resumed involuting according to

$$\Theta(t) \ = \ e^{-\nu t}\Theta_0. \tag{3}$$

Thus $t = 0$ is defined to be some time after reconstitituion, and *not* equal to treatment time. Therefore we must allow for non-zero $N_d(0)$, and do so by defining the initial conditions

The definition of $N_d(0)$ is motivated by the expectation that the initial donor numbers are proportional to χ. Also, $0 \le \mu \le 1$ (can't have greater chimerism in periphery unless donor kinetically different).

$$\begin{aligned} N_0 &= N_h(0) + N_d(0), \\ N_d(0) &= \chi\mu N_0. \end{aligned} \tag{4}$$

Solving the above equations yields

$$\begin{aligned} f_d(t) &= \frac{1}{\chi}\frac{N_d(t)}{N_d(t)+N_h(t)}, \\ &= 1 - \frac{(\lambda-\nu)(1-\mu)}{\lambda - \nu - (1-e^{(\lambda-\nu)t})\Theta_0/N_0} \end{aligned} \tag{5}$$

We now include a population of 'incumbent' host cells that evolves according

$$\begin{aligned} I'(t) &= -q\lambda I(t), \\ \implies I(t) &= I_0 e^{-q\lambda t} \end{aligned} \tag{6}$$

This population has a different kinetic parameter, is entirely of type host, and has no thymic input. Defining $I_0 = \alpha N_0$ we can now write

Important: The inclusion of incumbents means N_0 is no longer the initial total counts, but is now initial **displacable** counts, with total counts $N_T(0) = I_0 + N_0$. This also means α is **not** the initial incumbent fraction. This is given by $\alpha/1+\alpha$ thus α **can be** > 1.

$$\begin{aligned} f_d(t) &= \frac{1}{\chi}\frac{N_d(t)}{N_d(t)+N_h(t)+I(t)}, \\ &= \left[1 + \frac{(\lambda-\nu)\left(1+e^{(1-q)\lambda t}\alpha + \mu\right)}{\mu(\lambda-\nu) - \left(1-e^{(\lambda-\nu)t}\right)}\right]^{-1} \end{aligned} \tag{7}$$

Note that with the inclusion of the incumbent population, μ is **no longer** the initial value of the rescaled donor fraction, rather $f_d(0) = N_0\mu/I_0+N_0$. However the code uses μ not $f_d(0)$ and so we keep the above formalism in these notes. Also note that the rescaled donor fractions as defined are completely scale free, i.e. only depend on parameters that are rates or fractions, rather than parameters that have units involving absolute cell counts.

The same cannot be said of the total peripheral count function, $N_T(t) = N_d(t) + N_h(t) + I(t)$, given by

$$N_T(t) = N_0\left(\alpha e^{-q\lambda t} + \frac{\Theta_0/N_0 e^{-\nu t}}{\lambda-\nu} + \frac{e^{-t\lambda}(\lambda-\nu-\Theta_0/N_0)}{\lambda-\nu}\right) \tag{8}$$

These equations are hard-coded into the code but there exists a toggle to set $q = 0$ to look at the case of lossless incumbents.

Fitting procedure preliminaries

The goal of the fitting procedure is to minimize the sum of the squared residuals. In the case of simultaneously fitting the counts and rescaled donor fraction we actually want to minimize the *product* of the SSRs of each set (see PNAS paper SI for discussion/derivation). Several functions are used to go from a set of parameters to a global SSR.

Wherever 'SSR' is mentioned it is almost certainly the global SSR, i.e. product of the two individual SSRs

A key point is that ν, the rate of thymic involution, is *not* fitted in the same breath as the other, peripheral, parameters. An inbuilt routine (`nls`) is used to estimate ν from the `precursorData`, and this estimate value of ν is then an *input* into the functions that try to minimise the SSRs from the peripheral total counts and donor fraction.

`header`

The variable `header` is ubiquitous in all numeric SSR minimisation codes I wrote. In its simplest form it is a list of strings corresponding to the names of parameters that may be varied in order to minimise the SSR. In order to facilitate a more modular code, where we can set toggles to include or remove certain parameters (based on say, choice of model - more about that in the BrdU code) we build the `header` up iteratively. The initial declaration includes the bare set of parameters common to all models being considered. Then a series of `if()` statements, based on toggles (`qZero` in this case), will add additional parameters to this header **while** adding the appropriate initial and boundary values to the `start, upper,` and `lower` vectors which define the search range for the `GenSA()` minimiser.

In this code there is also an 'outside' parameter ν the estimation of which is independent of the main SSR minimisation routine. This variable still appears in the header despite not having a corresponding entry in `start` etc, and this is dealt with in `TransformFunctions()`.

Function and variable glossary

Below we detail the functions used to go from a set of parameter values to a value of the global SSR. This is done in a 'bottom up' approach, i.e. we discuss the lowest level functions first, starting with the transformation of the data and ending with the calculation of the global SSR.

`ratioFn(t, pars)` and `totalCountFn(t, pars)`

Called by `TransformFunctions()`.

These functions correspond to the analytics derived above. The functions are evaluated with a set of parameters `pars`, given as a dataframe with the column names corresponding to the parameter names (see `header`), and a single or vector of time times `t`.

Transforms `T(x) and T2(x)`

Called by `TransformFunctions()`.

These two functions correspond to the transforms used on the counts and rescaled donor fraction respectively to ensure heteroskedasticity. The total counts are taken to be (in principle) unbounded and positive, and so $T(x) = \ln(x)$. The donor fractions are always $0 \leq f_d(t) \leq 1$ and so $T2(x) = \arcsin\left(\sqrt{x}\right)$.

Important note: For physically reasonable parameters values, the rescaled donor fraction is always less than 1. However, for certain extreme parameter sets the function may tend to 1 and due to small numerical instabilities may be evaluated to be slightly more than 1, e.g. 1.00001. In this case the $T2(x)$ function will return `NaN`. This is dealt with by replacing **all** ratio values > 1 with 1. In this code with its simple analytic formalism this is safe enough, but for other more numerically oriented codes it is advisable to only replace values $1 < x \leq 1 + \delta$ to ensure it's only the small numerical errors that are at play and not something more sinister.

`TransformFunctions(inVec, nu)`

Called by `residFn()`.

Function used to apply transforms to analytic curves only. These curves are evaluated at times in `tVec` which corresponds to the times taken from the experimental data. The vector `inVec` corresponds to a vector of numbers fed in by the minimizer (or user).

The minimizer itself does not know/care about the parameter labels; the order of values in `inVec` is simply determined by the order of the initial conditions and search range values fed in to the minimizer (see `findFit()`). We map these numeric values to their parameter names by converting it to a dataframe with column names given by `header`. Thus, the correspondence between/ordering of values in `header` and the initial/range vectors in `findFit()` determine which parameter gets which value. This will be laid out more clearly in `findFit()`.

As ν is not part of the peripheral SSR minimization, it must not be 'seen' by the `GenSA` minimizer, and thus is fed in as an additional argument and combined with the main parameter vector `inVec`. The inclusion of ν in `inVec` **must** match its location/inclusion in `header` otherwise there will be a very nasty mismatch when we set `names(pars)<-header`. As far as I can tell this is the best way to allow ν to be fed into the main routine (we cannot hard code it, need to bootstrap ν too!) without it being seen by `GenSA()`.

`residFn(parameters, cDat, rDat, nuVal)`

Called by `SSRfn()`.

Function to calculate the residuals, defined as the **predicted value minus the experimental** value (both defined on an appropriate-transform scale). This function returns a list of one dimensional residuals of the form `list(residC, residR)`.

`SSRfn(inVec,cDat,rDat,nuVal)`

Called by `findFit()`.

Primary function which is called to calculate the peripheral SSR. This function passes all arguments to `residFn(inVec,cDat,rDat,nuVal)` to get the residual values then calculates the SSR for both the total counts and rescaled donor fraction. The global SSR is then defined as the product of these two values and is returned.

findFit(dVec, nuVal)

Called in main code and `doBoot()`.

Function called to invoke numeric minimization of global peripheral SSR based on the experimental data in `dVec` and ν value given to `nuVal`. The list `dVec` contains two one-dimensional vectors of the **transformed** values of the total counts and rescaled donor fraction respectively. The data fed into this main and subsequent routines is defined on the transformed scale to make bootstrapping (where residuals are defined on the transformed scale) easier.

Within this function the `GenSA` simulated annealing function is called. This function requires the max number of iterations and initial temperature to be defined, as well as initial, lower, and upper values for all parameters to be estimated. These are defined **outside** the `findFit()` function so that one can vary which parameters are being fitted (i.e. set $q = 0$ and remove it from the list of fitted parameters).

As noted in the sections on `header` and `TransformFunctions()` the length of the `header` will not match the length of the initial condition vectors.

doBoot(pairedMice)

Called in parallel foreach in main code.

Function to take in best fit residuals and compute a boot strap replicate. Prior to declaration of this function, the best fit is run and the residuals calculated. These residuals are put into a 3-column dataframe containing the precursor, count, and donor fraction residuals respectively. The key point here is that the residuals in each row all correspond to the same mouse. The residuals in this dataframe are all defined on their respective **transformed** scales.

This function then takes a random sample (of rows) with replacement from `pairedMice`. The new data to fit to is then reconstructed by adding the **transformed** best fit values of the functions to this choice of residuals. The new precursor data is then used to extract a new value of ν which, along with the new count and ratio data, is fed into the `findFit()` function to return a best fit for that boot strap replicated. Currently only the parameter estimates, not the SSRs, for these bootstrap fits are returned.

Parallel implementation of bootstraping

Every effort has been made to parallelise where possible. In this code the simultaneous fitting of bootstrap replicates is achieved through the `parallel` and `doParallel` packages.

In principle one could endeavour to parallelise `GenSA()` itself as this would make all SSR minimizations, even those of a 'single replicated'-type, embarrassingly parallel. However, currently only globally independent processes, such as bootstrapping or fitting different models (BrdU) is done in parallel.

The variable `nLogicalCores` can be set manually or made to automatically detect the number of cores present on a system. The `makeCluster(nLogicalCores)` command then initializes a local parallel cluster which is regisgetered by `registerDoParallel`. Once this is done loops of the form `foreach() %do% {...` may be made parallel simply by changing this to `foreach() %do`**`par`**`% {...`

It is important to note that only variables called explicitly (i.e. not through several layers of functions) within the parallel loop will have their definitions exported to these parallel environments. Furthermore, any packages required by functions called must have their packages loaded within the loop.

AgeDependentLoss.R

Overview

This code follows the same logic as the BusulfanNaiveFit.R file, except now the functions are described by PDEs not ODEs. The code follows the same definitions used in the PNAS paper, and as such provides a good starting point for understanding this routine. Functions like calculate residuals and transform values are all identical in function if not in implementation, it is only the actual functions themselves and the addition of another parameter μ (**not** the same as the incumbent model μ), estimated from the thymic data, that makes this code different. It is much slower as numerical integrals (of the PDEs) are involved.

Settings to keep in mind

As noted in the PNAS paper, the initial total counts of the population N_0 are specified manually because when left as a free and fitted parameter it becomes unphysically large. Most of the key settings for this routine are at the start. The reconstitution time and treatment age are informed by the data and so should remain unchanged if the current data is being used. All that is left to choose is the model being fitted (determined by the value of p) and which cell type.

A key difference in the PDE approach is the use of pre-reconstitution data. In order to estimate mu we need to know how the donor counts in the thymus behave pre-reconstitution. However, the peripheral data (as well as the total precursor counts in the thymus) used to calculate the residuals and thus the data that is being fitted to is taken to be **post-**reconstitution. This is so we are fitting to the same peripheral data as in the incumbent case and can use AIC and other statistical methods to compare between them; AIC is contingent on the same data being used to discriminate between models.

In principle one could also use pre-reconstitution total precursor data to estimate ν.

Function and variable glossary

`functions`

This code differs from the incumbent case in that the functions that represent the total counts and donor fractions are prepared in Mathematica and exported as .csv files rather than hard coded. They stay in the .R script as unevaluated expressions until they are passed to `calculate.outputs()`. Specifically, `functions` is a list of the following expressions:

1. donor counts pre-reconstitution (thymically derived)
2. total counts pre-reconstitution (thymically derived)
3. donor post-recons
4. total post-recons
5. host pre-treatment (initial distribution of cells at time of treatment)

`fit.thymic.pararameters(precursor.counts,donor.precursor.dp1`

This function fits the involution rate ν in the normal way, as well as the reconstitution rate μ. The latter is extracted from the percentage of donor cells at stage DP1 pre-reconstitution.

`calculate.outputs(t,all.pars)`

This takes in a collection of parameters and calculates the number of donor and donor+host cells present at time t. This involves integrating the PDEs over a appropriately, which is done using the inbuilt `integrate()` function and calling `eval.fn()` to evaluate functions imported in the .csv file.

Analysis.R

Overview

This code takes the outputs of BusulfanNaive.R and performs analysis to produce plots and tables found in the paper. Currently this only contains routines to compute tables/figures for the incumbent model, with the age-dependent PDE and RTE models analysed separately.

Settings to keep in mind

- Currently this code assumes that the incumbents are **lossless**, i.e. $q = 0$ and so this is hardcoded throughout. This is reasonable as not only do the AIC values from the best fit comparing $q \neq 0$ with $q = 0$ favour the simpler case, but one can easily allow q to be free in `BusulfanNaive.R` and examine the resulting bootstrap values for q.

- Many of the plots require that we have $t = 0$ correspond to BMT and not reconstitution, and so once again we need to define `t.Shift` (=42d by default).

- The parameters produced by the fitting code output total **displacable** numbers N_0 scaled by 10^7 and so this factor is restored here.

- Currently the total count and peripheral ratio donor fraction functions are hardcoded here as well - this will be changed so that both the fitting file and this file import the same functions to avoid mismatch.

- No weeks $\leftrightarrow$ days converstions are done here. Whatever time scale is used in the imported data and parameter values is what is used so this needs to be consistent.

- When using Ki67 data to calculate ρ and δ we consider three values of $\sigma = \delta_Y/\delta_X \in \{0.1, 1, 10\}$ where X and Y are Ki67$^{\text{lo}}$ and Ki67$^{\text{hi}}$ populations respectively. These values may be changed in the vector `sigT`. Additionally, we assume that 20% of the thymically derived cells are Ki67$^{\text{hi}}$, which may be changed through variable `kP`.

Primary outputs

As stated the goal of this code is to output the tables and plot values found in the paper.

`physicalPars`

This dataframe corresponds to the first three rows of Table 1 in the PNAS paper but also contains some extra values of interest. Remember that the initial values such as N_0 and Θ_0 correspond to values **at reconstitution** i.e. at 42 days post-BMT. This is then done for a range of t values and plotted.

`countPlot`, `ratioPlot`, and `incumbentFracPlot`

These plots correspond to the first four panels in figure three (though less pretty as I suck at ggplot and the first round were Mathemagica + Illustrator).

Confidence intervals are calculated by computing all possible values for $f(t, \vec{\beta})$ where $\vec{\beta}$ is a choice of bootstrapped parameters, and then computing upper and lower quantiles.

`kinetic.Parameters`

This corresponds to the following list of dataframes

- `lifetime:` lifetime of displacable cells
- `interdiv:` interdivision time of displacable cells
- `incumtime:` lifetime=interdivision time of incumbent cells
- `pooledT1Delta, pooledT2Delta:` lifetime of pooled population taken at $t =$`t1,t2` (currently 98 and 308 days, i.e. 14 and 44 wks)
- `pooledT1Rho, pooledT2Rho:` interdivision time of pooled population

These values are calculated by starting with the equation for ρ (see PNAS paper for derivation) given by

$$\rho = \frac{\kappa(1+\kappa(\sigma-1)+\lambda T(\epsilon+\sigma-\epsilon\sigma)) - \lambda T\epsilon}{T(1-\kappa)(2+\kappa(\sigma-1))} \tag{9}$$

where κ corresponds to percentage of cells that are Ki67^{+}, ϵ is the percentage of source (thymic) cells that are Ki67^{+}, σ is the ratio of the loss rates for Ki67$^{\pm}$ cells, and T is the mean lifetime of Ki67 expression.

We observed that the Ki67% varied somewhat for early mice and so, as detailed in `DataPrepare.R`, we take Ki67 measurements from mice only after 30wks. We cannot create a best fit point estimate for ρ as such, as we do not have a 'best fit' Ki67 value (though in principle we could use the mean. Instead we take a Monte-Carlo approach and simultaneously sample parameters from

a.) our parameter bootstraps (to get λ),

b.) the Ki67 data, and

c.) a log-normal distrubtion of Ki67 lifetimes.

This is run 10^4 times and means and confidence intervals are taken. In the case of incumbent cells $\lambda = 0$ (lossless incumbent assumption) and we only sample from host Ki67, as they are purely of type host. However, this makes little difference in reality as we already showed that after 30wks host and donor Ki67 are pretty much indistinguishable (ANOVA, not shown here but easy to do).

To calculate the pooled population values of $1/\rho$ and $1/\delta$ we take the weighted population average

Note that we take the weighted average of the **mean times** not the **rates**

$$\frac{1}{\rho_T} = \frac{\alpha N_0}{N_T}\frac{1}{\rho_I} + \left(1 - \frac{\alpha N_0}{N_T}\right)\frac{1}{\rho_D} \tag{10}$$

and the same for δ.