Most important R functions in TB Vx model

Source file **TBVx-hi-level-fncs-v1.R**

`set.paths()`

Description

The purpose of `set.paths()` is to create an R environment with file paths to the XML file, parameter file, targets file etc. This function also initializes the model log.

Usage

```
set.paths = function(mydir          = here(),
                     countries      = "countries",
                     countrycode    = NA,
                     xml            = NA,
                     parameters     = NA,
                     targets        = NA,
                     baseline       = NA,
                     VXa.incidence.data=NA)
```

Arguments

`mydir`	the ‘root’ directory
`countries`	the name of the countries folder (which will be appended to `mydir`)
`countrycode`	3 letter ISO code of the country (e.g. ZAF)
`xml`	the filename of the XML input file
`parameters`	the filename of the parameters file (which may overwrite parameters in the xml inputfile)
`targets`	the filename of the targets.csv file
`baseline`	the filename (without extent) of the XML file of the baseline
`VXa.incidence.files`	a vector of filenames (including path from the country folder) to replace the VXa incidence files specified in the XMLinput.xml file

Details

The only required parameters are `countrycode` and `xml`.

If the `parameters` argument is specified, the values of the parameters designated as constant in this file (usually `input.csv`) will overwrite the values in the XML.

If the `targets` argument is specified, the output will contain a `hits` element with a comparison between targets and model values.

If the `baseline` argument is specified, the model run will use the adjustments that were made to the population size over time (and written to files) in the current simulation run to take into account reduced mortality in an intervention run with TB vaccines.
This correction requires the attribute value `econ.output="true"` in the XML inputfile (in <output><detailed.output>).

The optional argument `VXa.incidence.files` is a vector of filenames that will replace the file names in <VXa><VXa.incidence><incidence.data>. Note: only the filenames will be replaced, not the additional attributes. *Use with care*.

The directory structure of a specific country (ZAF as an example) and the required files are:

ZAF/data/demographics.csv
ZAF/data/ZAF_deathrates.csv
ZAF/data/HIV-incidence.txt
ZAF/data/ART-incidence.txt
ZAF/data/ZAF_all_contacts_2020.txt

NOTE: the path and filename from the country folder depends on the specified path and filename in the XML input file e.g. <population file="data/demographics.csv"/>

ZAF/parameters/XMLinput.xml
ZAF/parameters/input.csv
ZAF/parameters/target.csv

NOTE: the filenames depend on the parameters passed to the intialize() function. The path is fixed.

ZAF/output The folder where output will be written to

ZAF/logs The folder where the model log will be written to

The initialize function also:

- sets up the paths environment
- sets up the global function modlog()

Result

`paths`	an environment with the paths to the various files and folders

Example

See runcountry-from-RStudio-new-examples.R

Source file **TBVx-run-v1.R**

`run()`

Description

The purpose of `run()` is to read the XML input file, update parameter values (if applicable), intialize the model parameter (i.e. setting up the matrices), do one model run, and return the output.

Usage

```
run = function(mdl=NULL,
               new.parameter.values=NULL,
               write.to.file=F,
               write.xml=NULL,
               combine.stocks.and.flows=F,
               baseline=NULL,
               output.format='txt',
               sample.parameters = F,
               exclude.intermediates=T)
```

Arguments

`mdl`	a model environment with initialized paths
`new.parameter.values`	a named vector of parameter values that will overwrite (part of) the non-constant parameters in the parameters file (usually input.csv)
`write.to.file`	a logical that determines if output will be written to files
`write.xml`	the name of a file to write the modified XML to
`combine.stocks.and.flows`	if T stocks and flows will be written to a single file
`baseline`	if not NULL, a reference to an output structure produced by a baseline run
`output.format`	either ‘txt’ for tab-delimited text, ‘fst’ for fst, or ‘parquet’ for parquet
`sample.parameters`	if T, values of the non-constant parameters in input,csv will be sampled from the distribution specified in the ‘distr’ column of input.csv
`exclude.intermediates`	if T, intermediate targets will be omitted in the ‘hits’ output

Details

If the argument `new.parameter.values` is not supplied while the paths$parameters is not NA, ***all*** parameters from the parameters file (input.csv) will be used to overwrite the parameter values in the XML.

Result

`output`	a list with the following elements: `stocks, flows, hits` (only if targets is specified) and additional elements if econ.output="true" in the XML

Example

See runcountry-from-RStudio.R

Source file: **TBVx-hi-level-fncs-v1.R**

```
set.options=function()

redirect.errors=function(paths)

setup.model.log=function(paths)

set.paths = function(mydir         = here(),
                     countries     = "countries",
                     countrycode   = NA,
                     xml           = NA,
                     parameters    = NA,
                     targets       = NA,
                     baseline      = NA,
                     VXa.incidence.files=NA)

modify.input.csv = function(input.csv=NULL,new.parameter.values=NULL)

selected.parameters = function(parameters.df=NULL, constant=NA)

constant.parameters = function(parameters.df=NULL)

fitted.parameters = function(parameters.df=NULL)

sample.fitted.parameters = function(selected.parameters=NULL)

get.target.hits = function(output=NULL,model.params=NULL, exclude.intermediates=T)

write.targets = function(output=NULL,model.params=NULL, output.format='txt')

write.stocks.and.flows = function(output=NULL,model.params=NULL, output.format='txt')

write.econ.output = function(output=NULL,model.params=NULL,output.format='txt')

write.updated.xml = function(parameters=NULL, xml=NULL)

merge.stocks.and.flows=function(output)
write.merged.stocks.and.flows = function(onerun=NULL,model.params=NULL,output.format='txt')
```

Source file: **TBVx-initialization-fncs-v20.R**

optimize.params=function(Ti)

fractions.by.stage.at.start=function(p, df=NULL)

fractions.by.stage.at.birth=function(p)

replace.NAs.by=function(nodes,attrib,replace=0,first=NA,n=NA)

check.sum.to.1 = function(v)

create.approx.func.from.rownames=function(df,offset=0.0)

get.deathrate.allcauses=function(p, t)

get.bgdeathrate=function(p, t)

create.approx.fun=function(row, df=NULL)

calc.birthrate = function(fname=NA, countrycode=NA)

scale.population = function(pop=NA,fname=NA)

seed.initial.population=function(parms, df)

init.constants = function(p,xml)

create.contacts.matrices=function(paths=NULL, model.params=NULL)

initialize.incidence=function(paths=NULL,p=NULL)

initialize.TB.transmission=function(p=NULL)

initialize.TB.infectivity=function(p=NULL)

initialize.treatment=function(p=NULL,dim=NA)

initialize.progression=function(p=NULL,dim=NA)

read.model.parameters=function(paths=NULL)

set.baseline.in.model.parameters=function(params=NULL,output=NULL)

initialize.model.parameters=function(params=NULL)

```
read.model.parameters = function(paths=NULL)
```

Arguments

paths	an environment with paths to various files, country code etc. See the function set.paths()

Result

output	an environment with initialized constants and initialized data structures related to demography

`initialize.model.parameters = function(params=NULL)`

Arguments

`params`	An environment with the result of calling the function read.model.parameters(). The params argument contains e.g. data frames from files read but needs to be parsed further.

Result

`output`	an environment with initialized data structures required to run the simulation ; see below for a complete description of that environment

The environment that results from `initialize.model.parameters()`

`AGES` Named num [1:16] 0 5 10 15 20 25 30 35 40 45 ...	lower limits of 5 year age groups
`aging.matrix` num [1:16, 1:16] -0.2 0.2 0 0 0 0 0 0 0 0 ...	matrix used to move the population to the next higher age group once a year ; i.e. the result of multiplying the 16x16 aging matrix with the (in this case) 240 x 16 matrix of state variables
`ALIVE` logi [1:240] TRUE TRUE TRUE TRUE TRUE TRUE ...	a logical vector that defines the 'alive' states, derived from state names not ending in 'dead'
`at.birth` Named num [1:240] 0.4 0 0 0 0 0 0 0 0 0 ...	a vector with the distribution of newborns over states (sums to 1.0)
`ATBIRTH` List of 5 $ VXa : num 1 $ SES : num [1:2] 0.4 0.6 $ RISK: num 1 $ HIV : num [1:10] 1 0 0 0 0 0 0 0 0 0 $ TB : num [1:12] 1 0 0 0 0 0 0 0 0 0 ...	List with distribution of newborns by state ; used to calculate at.birth vector
`at.start` num [1:240, 1:16] 0.321475 0.037363 0.002855 0.030072 0.000481 ...	distribution of the population by state for each of the age groups at the start of the simulation ; each age group (column) sums to 1.0
`birthrate`	a function that calculates the row index

function (v)	from the current year

`contacts`
List of 3
$ defined: logi TRUE
$ M :Formal class 'dgeMatrix' [package "Matrix"] with 4 slots
$ MplustM:Formal class 'dgeMatrix' [package "Matrix"] with 4 slots
	M is the contacts matrix (from the paper by Kiesha Prem et al) and MplustM is M + t(M) which is used to rebalance the contacts matrix to the updated population composition which is done once per year. See the function update.contact.matrix(parms, alive.pop) in TBVx-age-related-fncs-v##.R)

`DAYSPERYEAR` num 365.2425	number of days per year
`deathfn` function (v)	function to select the deathrate vector of the current year from the deathrate matrix (below)
`deathrate` num [1:151, 1:16] 0.0527 0.0527 0.0527 0.0527 0.0512 ...	deathrate by year (row) and age group (column) derived from data (e.g. IND_deathrates.csv)
`DIMLENGTHS` Named int [1:5] 1 2 1 10 12	lengths of the dimensions
`DIMNAMES` chr [1:5] "VXa" "SES" "RISK" "HIV" "TB"	names of the dimensions
`DIMNAMESLIST` List of 5 $ VXa : chr "never" $ SES : chr [1:2] "low" "high" $ RISK: chr "risk1" $ HIV : chr [1:10] "HIV-" "HIVu1" "HIVu2" "HIVd1" ... $ TB : chr [1:12] "Un" "Uc" "Lf" "Ls" ...	list of state names by dimension
`hash` chr e.g. "9f1dfb8bec73a9aaa1a92064f9a924a9"	hash of this environment of initialized parameters
`HIV` chr [1:10] "HIV-" "HIVu1" "HIVu2" "HIVd1" "HIVd2" "ARTn1" "ARTn2" "ARTs1" "ARTs2" ...	names of HIV states
`HIVp` <environment: 0x00000247f45b9a58>	environment with HIV progression parameters
`HIVTB` chr [1:120] "HIV--Un" "HIV--Uc" "HIV--Lf" "HIV--Ls" "HIV--Ds" "HIV--Dc" "HIV--T" ...	names of the combination of HIV and TB states

`HIVtr`
List of 4
$ diag :<environment: 0x00000247fa8085d8>
$ treat :<environment: 0x00000247fa132138>
$ supp :<environment: 0x00000247f6beadc8>
$ HIVARTdead:<environment: 0x00000247f8198c88>
	list of HIV treatment environments; the names of these environments are derived from the XML

	distribution of newborns over HIV states (sums to 1) ; used to generate ATBIRTH list
`inci` List of 1 $ HIV:List of 1	List of incidence lists ; one for each supported dimension. Used for generating incidence matrices (function create.incidence.matrix()) Each of the list elements is an environment with all that is required to properly execute the transitions specified in the incidence files
`inci.dim.names` List of 5 $ VXa: chr [1:240] "never" "never" "never" "never" ... $: NULL $: NULL $ HIV: chr [1:240] "HIV-" "HIV-" "HIV-" "HIV-" ... $: NULL	List of dimensions that (currently) support incidence files
`Infc` <environment: 0x00000247f7f6e150>	Environment with parameters related to TB infectiousness ; most important is the matrix with infectiousness by state (a 240x240 matrix in this case)
`intervention` logi FALSE	logical to flag intervention run
`intervention.start` num 1e+06	default value of intervention start ; overwritten if applicable
`intervention.start.from` logi NA	logical indicating if the intervention start is derived from incidence file
`iHIV`: num [1:10] 1 0 0 0 0 0 0 0 0 0 `iRISK` : num 1 `iSES` : num [1:2] 0.4 0.6 `iTB` : num [1:12] 1 0 0 0 0 0 0 0 0 0 ... `iVXa` : num 1	distribution of newborns over dimension states ; used to generate `ATBIRTH`
`nAGES` : int 16	sizes of (combined) dimensions

nHIV : int 10 nHIVTB : int 120 nRISK : int 1 nRISKHIVTB : int 120 nSES : int 2 nSESRISKHIVTB : int 240 nTB : int 12 nVXa : int 1 nVXaSESRISKHIVTB : int 240	
paths : <environment: 0x00000247f063b038>	environment with paths to various files ; see set.paths() function
popfname : chr e.g. "C:/Users/roelb/GitHub/tbvax-update-current/tbvax/countries-examples/ZAF/data/demographics.csv"	filename of UNPOP demographics file (i.e. population development over time)
PROGRESSION : List of 5 $ HIV :<environment: 0x00000247f45b9a58> $ HIVtr:List of 4 $ RISK : NULL $ SES : NULL $ VXa : NULL	List with HIV, RISK, SES and VXa progression and in addition HIV treatment. Produced by PROGRESSION = list(HIV=HIVp,HIVtr=HIVtr,RISK=RISKp,SES=SESp,VXa=VXap) where HIVp etc have been initialized separately. TB progression and treatment are in TBp and TBtr
RISK : chr "risk1"	names of RISK states
RISKHIVTB : chr [1:120] "risk1-HIV--Un" "risk1-HIV--Uc" "risk1-HIV--Lf" "risk1-HIV--Ls" ...	names of combined RISKHIVTB states
RISKp : NULL	RISK progression
run.params : <environment: 0x00000247e2608590>	Environment with parameters that are related to running the simulation rather than the model itself (i.e. states, transitions etc.).
SES : chr [1:2] "low" "high"	names of RISK states
SESp : NULL	SES progression
SESRISKHIVTB : chr [1:240] "low-risk1-HIV--Un" "low-risk1-HIV--Uc" "low-risk1-HIV--Lf" ...	names of combined SESRISKHIVTB states

`TB` : chr [1:12] "Un" "Uc" "Lf" "Ls" "Ds" "Dc" "T" "R" "TBdead" "TBHIVdead" "Rdead" ...	names of TB states
`TBp` : <environment: 0x00000247fda44480>	RISK progression
`TBtr` : List of 5 $ `RDs` :<environment: 0x00000247f604d680> $ `LfLsDs`:<environment: 0x00000247f1c2f4c0> $ `init` :<environment: 0x00000247fbc78108> $ `nodead`:<environment: 0x00000247f8de4928> $ `died` :<environment: 0x00000247f25667c0>	List of TB treatment matrices
`Tm` : <environment: 0x00000247f74b9888>	TB transmission
`unpopdem` : num [1:151, 1:16] 2024 2106 2218 2326 2419 ...	UNPOP population by year and age group (i.e. adjusted to modelled age groups)
`unpopdemfn` : function (v)	function that returns the row index for the current year
`VXa` : chr "never"	names of VXa states
`VXap` : NULL	VXa progression
`VXaSESRISKHIVTB` : chr [1:240] "never-low-risk1-HIV--Un" "never-low-risk1-HIV--Uc" ...	names of combined VXaSESRISKHIVTB states

`xml` : List of 2
$ doc :List of 2
$ schema:List of 2
	List with
	XML document and
	XML Schema

`xmlfilename` : chr "C:/Users/roelb/GitHub/tbvax-update-current/tbvax/countries-examples/ZAF/parameters/Area1/XMLinput_2910_m.xml"	file name of XML input file
`y.ini` : num [1:240, 1:16] 650.584 75.613 5.779 60.859 0.974 ...	initial population by state and age group
`y.ini.read` : num [1:240, 1:16] 2024 0 0 0 0 ...	the initial population read from demographics.csv for the start year of the simulation, before seeding ; seeding creates y.ini

Source file: **TBVx-derivs-v24.R**

add.vaccination.results = function(t,p,dvacc,rownamesZ,subject="VXa")

aging.and.births.and.update.contact.matrix.event <- function(t,X,p)

aging.and.births.and.update.contact.matrix.event is called once a year, at the start of a new year, and takes care of:

- aging: moving (part of) the contents of the state matrix Y (after converting the vector X) to the next higher age group
- births: adding newborns
- adjusting the population composition to UNPOP data (baseline) or reusing recorder population adjustments (intervention)
- rescaling the population (usually in 1950)
- updating the contacts matrix (i.e. balancing the contacts matrix to ensure equal number of contacts between ego - alter age groups and vice versa which is necessary if population composition changes)
- incidence in case of incidence as a yearly event rather than a rate

derivs.deSolve=function(t,X,p)

derivs.deSolve() calculates the derivatives at time **t**, with state vector **X** and parameters **p**

After converting the vector X to a matrix Y with
Y = Y.matrix.from.X.vector(X,p)

The derivative dY is calculated and updated by calling

```
dY = matmul.by.age.group(p$TBp,Y)                    # TB progression
dY = dY + derivs.Tm(t,Y,p,fn=matmul.by.age.group)    # TB transmission
if (!is.null(p$TBtr)){                               # TB treatment
   for (i in seq_along(p$TBtr)){
     dY = dY + derivs.Tr(t,Y,p$TBtr[[i]],fn=matmul.by.age.group)
   }
 }
 for (i in seq_along(p$inci$TB)){                    # TB incidence
   dY = dY + derivs.inci(t,Y,p,p$inci$TB[[i]])
 }
 if (p$nHIV>1){
   dY = dY + matmul.by.age.group(p$HIVp, Y)          # HIV: progression
   if (!is.null(p$HIVtr)){                           # HIV treatment
     for (i in seq_along(p$HIVtr)){
       dY = dY + derivs.Tr(t,Y,p$HIVtr[[i]],fn=matmul.by.age.group)
     }
   }
   for (i in seq_along(p$inci$HIV)){                 # HIV incidence
     dY = dY + derivs.inci(t,Y,p,p$inci$HIV[[i]])
   }
```

```
  }
  if (p$nRISK>1 & !is.null(p$RISKp)){                    # RISK: progression
    dY = dY + matmul.by.age.group(p$RISKp, Y)
  }
  if (p$nSES>1  & !is.null(p$SESp)){
    dY = dY + matmul.by.age.group(p$SESp, Y)             # SES: progression
  }
  if (p$nVXa>1){
    if(!is.null(p$VXap))
      dY = dY + matmul.by.age.group(p$VXap, Y)           # VXa: progression
    for (i in seq_along(p$inci$VXa)){
      dvac = derivs.inci(t,Y,p,p$inci$VXa[[i]])          # VXa: incidence
      rownames(dvac)=p$VXaSESRISKHIVTB
      rows = calc.indices.for.dim(p,"VXa")==which(p$VXa == "vac")
      dY = dY + dvac
    }

  }

# return a list with two vectors:
# dY : a vector (after converting the matrix dY)
# dY.HIVTBdeaths : a vector with HIV and TB deaths
list(dY=as.numeric(dY), dY.HIVTBdeaths=as.numeric(colsums(dY[!p$ALIVE,])))
```

derivs.Tr=function(t,Y,Tr,fn)

derivs.Tr is a function that calculates the contribution of treatment to the derivatives of Y

foi=function(t,Y,parms)

foi is a function that calculates the force of infection at time t, for state Y and using parms to get a reference to (yearly updated) contact matrices

derivs.Tm=function(t,Y,parms,fn)

derivs.Tm() calculates the contribution of transmission to the derivatives of Y at time t with parameters parms and a reference to a function fn that calculates susceptibility
Within derivs.Tm() the function foi() is called to calculate the force of infection.

derivs.inci.common=function(t=NA,Y=NA,p=NA,p.inci=NA)

derivs.inci.common() calculates the contribution to the derivatives of Y at time t with parameters p and p.inci

p.inci is an environment derived from options in the XML e.g.

```
<HIV.incidence>
        <incidence.data file="data/HIV-incidence.txt" times="1980,2051"
          values="lambdaH,lambdaH" proportions="false" denominator="susc"/>
</HIV.incidence>
```

and from the file that defines the incidence (e.g. “data/HIV-incidence.txt”)
The most important data structure is p.inci$inci.matrix which is based on the incidence data file.

derivs.inci=function(t=NA,Y=NA,p=NA,p.inci=NA,once=F)

derivs.inci.in.out=function(t=NA,Y=NA,p=NA,p.inci=NA,once=F)

scale.alive.population=function(Y,parms)

colsums = function(M)

Y.matrix.from.X.vector = function(X,p)

run.deSolve = function(params = NULL)

run.deSolve is called from run.model()
within run.deSolve the deSolve function rk() is called with derivs.deSolve() as the argument that provides a reference to the derivs function that produces the derivatives at a specific time point.

calc.vac.flows=function(p)

run.model = function(model.params=NULL)

The main function that runs the model with initialized model.params as an argument.
run.model calls run.deSolve() internally which returns the state variables over time.
From these state variables the derivs are recreated to produce flows.
All output is created from the state variables over time.

demography.from.model.run=function(out,params)