Package 'Karen'

June 9, 2022

Title Kalman Reaction Networks

Version 1.0

Description In this package we implemented a stochastic framework that combines biochemical reaction networks with extended Kalman filter and Rauch Tung Striebel smoothing.
This framework allows to investigate the dynamics of cell differentiation from high-dimensional clonal tracking data subject to measurement noise, false negative errors, and systematically unobserved cell types.
Our tool can provide statistical support to biologists in gene therapy clonal tracking studies for a deeper understanding of clonal reconstitution dynamics.

License `use_mit_license()`, `use_gpl3_license()` or friends to pick a license

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.2

Imports Matrix, parallel, gaussquad, splines, scales, tmvtnorm, MASS

Depends R (>= 3.5.0)

LazyData true

NeedsCompilation no

Author Luca Del Core [aut, cre, cph] (<https://orcid.org/0000-0002-1672-6995>),
Danilo Pellin [aut] (<https://orcid.org/0000-0002-2647-0508>),
Marco Grzegorczyk [aut, ths] (<https://orcid.org/0000-0002-2604-9270>),
Ernst Wit [aut, ths] (<https://orcid.org/0000-0002-3671-9610>)

Maintainer Luca Del Core <l.del.core@rug.nl>

R topics documented:

get.cdn *Get the cell differentiation network from a fitted Kalman Reaction Network.*

Description

This function returns the cell differentiation network from a Kalman Reaction Network previously fitted on a clonal tracking dataset.

Usage

```
get.cdn(res.fit, edges.lab = FALSE, AIC = FALSE, cell.cols = NULL)
```

Arguments

res.fit	A list returned by get.fit() containing the information of a fitted Kalman Reaction Network.
edges.lab	(logical) Defaults to FALSE, in which case the labels (weights) will not be printed on the network edges.
AIC	(logical) Defaults to FALSE, in which case the Akaike Information Criterion is not reported.
cell.cols	Color legend for the cell types. Defaults to NULL, in which case no color legend for the cell types is provided.

Examples

```
cat("\nInstall/load packages")
inst.pkgs <- installed.packages() ## installed packages

l.pkgs <- c("expm",
            "Matrix",
            "parallel",
            "gaussquad",
            "splines",
            "scales",
            "mvtnorm",
            "tmvtnorm",
            "MASS",
            "igraph",
            "stringr",
            "Karen")
lapply(l.pkgs, function(pkg){
  if(!(pkg %in% rownames(inst.pkgs))){
    install.packages(pkg)
  }
})
## load packages
lapply(l.pkgs, function(pkg){library(pkg,character.only=TRUE)})

rm(list = ls())

rcts <- c("HSC->P1", ## reactions
          "HSC->P2",
```

```
            "P1->T",
            "P1->B",
            "P1->NK",
            "P2->G",
            "P2->M",
            "T->0",
            "B->0",
            "NK->0",
            "G->0",
            "M->0"
            ,"HSC->1"
            ,"P1->1"
            ,"P2->1"
)

cnstr <- c("theta\\[\\'HSC->P1\\'\\]=(theta\\[\\'P1->T\\'\\] + theta\\[\\'P1->B\\'\\] + theta\\[\\'P1->NK\\'\
       "theta\\[\\'HSC->P2\\'\\]=(theta\\[\\'P2->G\\'\\] + theta\\[\\'P2->M\\'\\])") ## reaction constraints
latsts <- c("HSC", "P1", "P2") ## latent cell types

ctps <- unique(setdiff(c(sapply(rcts, function(r){ ## all cell types
  as.vector(unlist(strsplit(r, split = "->", fixed = T)))
}, simplify = "array")), c("0", "1")))

########## TRUE PARAMETERS ##########
th.true <- c(0.65, 0.9, 0.925, 0.975, 0.55, 3.5, 3.1, 4, 3.7, 4.1, 0.25, 0.225, 0.275) ## dynamic parameters
names(th.true) <- tail(rcts, -length(cnstr))
s2.true <- 1e-8 ## additonal noise
r0.true <- .1 ## intercept noise parameter
r1.true <- .5 ## slope noise parameter
phi.true <- c(th.true, r0.true, r1.true) ## whole vector parameter
names(phi.true) <- c(names(th.true), "r0", "r1")

########## SIMULATION PARAMETERS ##########
S <- 1000 ## trajectories length
nCL <- 3 ## number of clones
X0 <- rep(0, length(ctps)) ## initial condition
names(X0) <- ctps
X0["HSC"] <- 100
ntps <- 30 ## number of time-points
f_NA <- .75 ## fraction of observed data

############################
## SIMULATE TRAJECTORIES ##
############################

XY <- get.sim.trajectories(rct.lst = rcts,
                           constr.lst = cnstr,
                           latSts.lst = latsts,
                           ct.lst = ctps,
                           th = th.true,
                           S = S,
                           nCL = nCL,
                           X0 = X0,
                           s2 = s2.true,
                           r0 = r0.true,
                           r1 = r1.true,
                           f = f_NA,
```

```
                          ntps = ntps,
                          trunc = FALSE)

####################################
## Fitting Karen on simulated data ##
####################################

nProc <- 1 # number of cores
cat(paste("\tLoading CPU cluster...\n", sep = ""))
cat(paste("Cluster type: ", "PSOCK\n", sep = ""))
cpu <- Sys.getenv("SLURM_CPUS_ON_NODE", nProc) ## define cluster CPUs
hosts <- rep("localhost",cpu)
cl <- makeCluster(hosts, type = "PSOCK") ## make the cluster
rm(nProc)

## mean vector of the initial condition:
m_0 <- replicate(nCL, X0, simplify = "array")
colnames(m_0) <- 1:nCL
## covariance matrix of the initial condition:
P_0 <- Diagonal(length(ctps) * nCL, 1e-5)
rownames(P_0) <- colnames(P_0) <- rep(1:nCL, each = length(ctps))
## Fit Karen on the simulated data:
res.fit <- get.fit(rct.lst = rcts,
                   constr.lst = cnstr,
                   latSts.lst = latsts,
                   ct.lst = ctps,
                   Y = XY$Y[,setdiff(ctps, latsts),],
                   m0 = m_0,
                   P0 = P_0,
                   cl = cl,
                   list(nLQR = 3,
                        lmm = 25,
                        pgtol = 0,
                        relErrfct = 1e-9,
                        tol = 1e-9,
                        maxit = 1000,
                        maxitEM = 10,
                        trace = 1,
                        FORCEP = FALSE))

stopCluster(cl) ## stop the cluster

#########################
## Visualizing results ##
#########################

library(devtools)
source_url("https://raw.githubusercontent.com/jevansbio/igraphhack/master/igraphplot2.R")
environment(plot.igraph2) <- asNamespace('igraph')
environment(igraph.Arrows2) <- asNamespace('igraph')

## Cell differentiation network
legend_image <- as.raster(matrix(colorRampPalette(c("lightgray", "red", "black"))(99), ncol=1))
pdf(file = paste(getwd(), "f", f_NA, "_diffNet.pdf", sep = ""), width = 5, height = 5)
layout(mat = matrix(c(1,1,1,2), ncol = 1))
par(mar = c(0,0,3,0))
get.cdn(res.fit = res.fit,
```

```
        edges.lab = F)
plot(c(0,1),c(-1,1),type = 'n', axes = F,xlab = '', ylab = '')
text(x=seq(0,1,l=5), y = -.2, labels = seq(0,1,l=5), cex = 2, font = 2)
rasterImage(t(legend_image), 0, 0, 1, 1)
dev.off()
```

`get.fit` *Fit the state-space model to a clonal tracking dataset*

Description

This function fits a state-space model to a clonal tracking dataset using an extended Kalman filter approach.

Usage

```
get.fit(
  rct.lst,
  constr.lst,
  latSts.lst,
  ct.lst,
  Y,
  m0,
  P0,
  cl = getDefaultCluster(),
 control = list(nLQR = 3, lmm = 25, pgtol = 0, relErrfct = 1e-05, tol = 1e-09, maxit =
    1000, maxitEM = 10, trace = 1, FORCEP = TRUE)
)
```

Arguments

`rct.lst`	A list of biochemical reactions defining the cell differentiation network. A differentiation move from cell type "A" to cell type "B" must be coded as "A->B" Duplication of cell "A" must be coded as "A->1" Death of cell "A" must be coded as "A->0".
`constr.lst`	List of linear constraints that must be applied to the biochemical reactions. For example, if we need the constraint "A->B = B->C + B->D", this must be coded using the following syntax c("theta\[\'A->B\'\]=(theta\[\'B->C\'\] + theta\[\'B->D\'\])").
`latSts.lst`	List of the latent cell types. If for example counts are not available for cell types "A" and "B", then latSts.lst = c("A", "B").
`ct.lst`	List of all the cell types involved in the network formulation. For example, if the network is defined by the biochemical reactions are A->B" and "A->C", then ct.lst = c("A", "B", "C").
`Y`	A 3-dimensional array whose dimensions are the time, the cell type and the clone respectively.
`m0`	mean vector of the initial condition x_0
`P0`	covariance matrix of the initial condition x_0

`cl` An object of class "cluster" specifying the cluster to be used for parallel execution. See makeCluster for more information. If the argument is not specified, the default cluster is used. See setDefaultCluster for information on how to set up a default cluster.

`control` A a list of control parameters for the optimization routine:

- "nLQR"(defaults to 3) is an integer giving the order of the Gauss-Legendre approximation for integrals.
- "lmm"(defaults to 25) is an integer giving the number of BFGS updates retained in the "L-BFGS-B" method.
- "pgtol"(defaults to 0 when check is suppressed) is a tolerance on the projected gradient in the current search direction of the "L-BFGS-B" method.
- "relErrfct"(defaults to 1e-5) is the relative error on the function value for the "L-BFGS-B" optimization. That is, the parameter "factr" of the optim() function is set to relErrfct/.Machine$double.eps.
- "tol"(defaults to 1e-9) is the relative error tolerance for the expectation-maximization algorithm of the extended Kalman filter optimization. That is, the optimization is run until the relative error of the function and of the parameter vector are lower than tol.
- "maxit"(defaults to 1000) The maximum number of iterations for the "L-BFGS-B" optimization.
- "maxitEM"(defaults to 10) The maximum number of iterations for the expectation-maximization algorithm.
- "trace"(defaults to 1) Non-negative integer. If positive, tracing information on the progress of the optimization is produced. This parameter is also passed to the optim() function. Higher values may produce more tracing information: for method "L-BFGS-B" there are six levels of tracing. (To understand exactly what these do see the source code: higher levels give more detail.)
- "FORCEP"(defaults to TRUE) Logical value. If TRUE, then all the covariance matrices involved in the algorithm are forced to be positive-definite and it helps the convergence of the optimization.

Value

A list containing the following:

- "fit"The output list returned by the optim() function (See documenttion of optim() for more details).
- "bwd.res"First two-order moments of the estimated smoothing distribution.
- "m0.res"Mean vector of the smoothing distribution at time t = 0.
- "P0.res"Covariance matrix of the smoothing distribution at time t = 0.
- "AIC"Akaike Information Criterion (AIC) of the fitted model.
- "cloneChunks"List containing the chunks of clones that have been defined for parallel-computing.
- "V"The net-effect matrix associated to the differentiation network.
- "Y"The complete clonal tracking dataset that includes also the missing cell types.
- "rct.lst"The list of biochemical reactions.
- "constr.lst"The linear constraints applied on the reactions.
- "latSts.lst"The missing/latent cell types.

Examples

```
cat("\nInstall/load packages")
inst.pkgs <- installed.packages() ## installed packages

l.pkgs <- c("expm",
            "Matrix",
            "parallel",
            "gaussquad",
            "splines",
            "scales",
            "mvtnorm",
            "tmvtnorm",
            "MASS",
            "igraph",
            "stringr",
            "Karen")
lapply(l.pkgs, function(pkg){
  if(!(pkg %in% rownames(inst.pkgs))){
    install.packages(pkg)
  }
})
## load packages
lapply(l.pkgs, function(pkg){library(pkg,character.only=TRUE)})

rm(list = ls())

rcts <- c("HSC->P1", ## reactions
          "HSC->P2",
          "P1->T",
          "P1->B",
          "P1->NK",
          "P2->G",
          "P2->M",
          "T->0",
          "B->0",
          "NK->0",
          "G->0",
          "M->0"
          ,"HSC->1"
          ,"P1->1"
          ,"P2->1"
)

cnstr <- c("theta\\[\\'HSC->P1\\'\\]=(theta\\[\\'P1->T\\'\\] + theta\\[\\'P1->B\\'\\] + theta\\[\\'P1->NK\\'\
       "theta\\[\\'HSC->P2\\'\\]=(theta\\[\\'P2->G\\'\\] + theta\\[\\'P2->M\\'\\])") ## reaction constraints
latsts <- c("HSC", "P1", "P2") ## latent cell types

ctps <- unique(setdiff(c(sapply(rcts, function(r){ ## all cell types
  as.vector(unlist(strsplit(r, split = "->", fixed = T)))
}, simplify = "array")), c("0", "1")))

########## TRUE PARAMETERS ##########
th.true <- c(0.65, 0.9, 0.925, 0.975, 0.55, 3.5, 3.1, 4, 3.7, 4.1, 0.25, 0.225, 0.275) ## dynamic parameters
names(th.true) <- tail(rcts, -length(cnstr))
s2.true <- 1e-8 ## additonal noise
r0.true <- .1 ## intercept noise parameter
```

```
r1.true <- .5 ## slope noise parameter
phi.true <- c(th.true, r0.true, r1.true) ## whole vector parameter
names(phi.true) <- c(names(th.true), "r0", "r1")

########## SIMULATION PARAMETERS ##########
S <- 1000 ## trajectories length
nCL <- 3 ## number of clones
X0 <- rep(0, length(ctps)) ## initial condition
names(X0) <- ctps
X0["HSC"] <- 100
ntps <- 30 ## number of time-points
f_NA <- .75 ## fraction of observed data

############################
## SIMULATE TRAJECTORIES ##
############################

XY <- get.sim.trajectories(rct.lst = rcts,
                           constr.lst = cnstr,
                           latSts.lst = latsts,
                           ct.lst = ctps,
                           th = th.true,
                           S = S,
                           nCL = nCL,
                           X0 = X0,
                           s2 = s2.true,
                           r0 = r0.true,
                           r1 = r1.true,
                           f = f_NA,
                           ntps = ntps,
                           trunc = FALSE)

####################################
## Fitting Karen on simulated data ##
####################################

nProc <- 1 # number of cores
cat(paste("\tLoading CPU cluster...\n", sep = ""))
cat(paste("Cluster type: ", "PSOCK\n", sep = ""))
cpu <- Sys.getenv("SLURM_CPUS_ON_NODE", nProc) ## define cluster CPUs
hosts <- rep("localhost",cpu)
cl <- makeCluster(hosts, type = "PSOCK") ## make the cluster
rm(nProc)

## mean vector of the initial condition:
m_0 <- replicate(nCL, X0, simplify = "array")
colnames(m_0) <- 1:nCL
## covariance matrix of the initial condition:
P_0 <- Diagonal(length(ctps) * nCL, 1e-5)
rownames(P_0) <- colnames(P_0) <- rep(1:nCL, each = length(ctps))
## Fit Karen on the simulated data:
res.fit <- get.fit(rct.lst = rcts,
                   constr.lst = cnstr,
                   latSts.lst = latsts,
                   ct.lst = ctps,
                   Y = XY$Y[,setdiff(ctps, latsts),],
                   m0 = m_0,
```

```
                    P0 = P_0,
                    cl = cl,
                    list(nLQR = 3,
                         lmm = 25,
                         pgtol = 0,
                         relErrfct = 1e-9,
                         tol = 1e-9,
                         maxit = 1000,
                         maxitEM = 10,
                         trace = 1,
                         FORCEP = FALSE))

stopCluster(cl) ## stop the cluster
```

`get.sim.trajectories` *Simulate a clonal tracking dataset from a given cell differentiation network.*

Description

This function simulates clone-specific trajectories for a cell differentiation network associated to a set of (constrained) biochemical reactions, cell types, and missing/latent cell types.

Usage

```
get.sim.trajectories(
  rct.lst,
  constr.lst,
  latSts.lst,
  ct.lst,
  th,
  S,
  nCL,
  X0,
  s2 = 1e-08,
  r0 = 0,
  r1 = 0,
  f = 0,
  ntps,
  trunc = FALSE
)
```

Arguments

`rct.lst`	A list of biochemical reactions defining the cell differentiation network. A differentiation move from cell type "A" to cell type "B" must be coded as "A->B" Duplication of cell "A" must be coded as "A->1" Death of cell "A" must be coded as "A->0".
`constr.lst`	List of linear constraints that must be applied to the biochemical reactions. For example, if we need the constraint "A->B = B->C + B->D", this must be coded using the following syntax c("theta\[\'A->B\'\]=(theta\[\'B->C\'\] + theta\[\'B->D\'\])").

latSts.lst	List of the latent cell types. If for example counts are not available for cell types "A" and "B", then latSts.lst = c("A", "B").
ct.lst	List of all the cell types involved in the network formulation. For example, if the network is defined by the biochemical reactions are A->B" and "A->C", then ct.lst = c("A", "B", "C").
th	The vector parameter that must be used for simulation. The length of th equals the number of unconstrained reactions plus 2 (for the noise parameters (ρ_0, ρ_1)). Only positive parameters can be provided.
S	The length of each trajectory.
nCL	An integer defining the number of distinct clones.
X0	A p-dimensional vector for the initial condition of the cell types, where p is the number of distinct cell types provided in ct.lst.
s2	(defaults to 1e-8) A positive value for the overall noise variance.
r0	(defaults to 0) A positive value for the intercept defining the noise covariance matrix $R_k = \rho_0 + \rho_1 G_k X_k$).
r1	(defaults to 0) A positive value for the slope defining the noise covariance matrix $R_k = \rho_0 + \rho_1 G_k X_k$).
f	(defaults to 0) The fraction of measurements that must be considered as missing/latent.
ntps	Number of time points to consider from the whole simulated clonal tracking dataset.
trunc	(defaults to FALSE) Logical, indicating whether sampling from a truncated multivariate normal must be performed.

Value

A list containing the following:

- "X"The simulated process.
- "Y"The simulated noisy-corrupted measurements.

Examples

```
cat("\nInstall/load packages")
inst.pkgs <- installed.packages() ## installed packages

l.pkgs <- c("expm",
            "Matrix",
            "parallel",
            "gaussquad",
            "splines",
            "scales",
            "mvtnorm",
            "tmvtnorm",
            "MASS",
            "igraph",
            "stringr",
            "Karen")
lapply(l.pkgs, function(pkg){
  if(!(pkg %in% rownames(inst.pkgs))){
    install.packages(pkg)
```

```
  }
})
## load packages
lapply(l.pkgs, function(pkg){library(pkg,character.only=TRUE)})

rm(list = ls())

rcts <- c("HSC->P1", ## reactions
          "HSC->P2",
          "P1->T",
          "P1->B",
          "P1->NK",
          "P2->G",
          "P2->M",
          "T->0",
          "B->0",
          "NK->0",
          "G->0",
          "M->0"
          ,"HSC->1"
          ,"P1->1"
          ,"P2->1"
)

cnstr <- c("theta\\[\\'HSC->P1\\'\\]=(theta\\[\\'P1->T\\'\\] + theta\\[\\'P1->B\\'\\] + theta\\[\\'P1->NK\\'\
       "theta\\[\\'HSC->P2\\'\\]=(theta\\[\\'P2->G\\'\\] + theta\\[\\'P2->M\\'\\])") ## reaction constraints
latsts <- c("HSC", "P1", "P2") ## latent cell types

ctps <- unique(setdiff(c(sapply(rcts, function(r){ ## all cell types
  as.vector(unlist(strsplit(r, split = "->", fixed = T)))
}, simplify = "array")), c("0", "1")))

########## TRUE PARAMETERS ##########
th.true <- c(0.65, 0.9, 0.925, 0.975, 0.55, 3.5, 3.1, 4, 3.7, 4.1, 0.25, 0.225, 0.275) ## dynamic parameters
names(th.true) <- tail(rcts, -length(cnstr))
s2.true <- 1e-8 ## additonal noise
r0.true <- .1 ## intercept noise parameter
r1.true <- .5 ## slope noise parameter
phi.true <- c(th.true, r0.true, r1.true) ## whole vector parameter
names(phi.true) <- c(names(th.true), "r0", "r1")

########## SIMULATION PARAMETERS ##########
S <- 1000 ## trajectories length
nCL <- 3 ## number of clones
X0 <- rep(0, length(ctps)) ## initial condition
names(X0) <- ctps
X0["HSC"] <- 100
ntps <- 30 ## number of time-points
f_NA <- .75 ## fraction of observed data

############################
## SIMULATE TRAJECTORIES ##
############################

XY <- get.sim.trajectories(rct.lst = rcts,
                           constr.lst = cnstr,
                           latSts.lst = latsts,
```

```
                    ct.lst = ctps,
                    th = th.true,
                    S = S,
                    nCL = nCL,
                    X0 = X0,
                    s2 = s2.true,
                    r0 = r0.true,
                    r1 = r1.true,
                    f = f_NA,
                    ntps = ntps,
                    trunc = FALSE)
```

`get.sMoments` *Get the first two-order smoothing moments from a fitted Kalman Reaction Network.*

Description

This function returns the first two-order smoothing moments from a Kalman Reaction Network previously fitted on a clonal tracking dataset.

Usage

```
get.sMoments(res.fit, X = NULL, cell.cols = NULL)
```

Arguments

`res.fit`	A list returned by get.fit() containing the information of a fitted Kalman Reaction Network.
`X`	Stochastic process. A 3-dimensional array whose dimensions are the time, the cell type and the clone respectively.
`cell.cols`	Color legend for the cell types. Defaults to NULL, in which case no color legend for the cell types is provided.

Examples

```
cat("\nInstall/load packages")
inst.pkgs <- installed.packages() ## installed packages

l.pkgs <- c("expm",
            "Matrix",
            "parallel",
            "gaussquad",
            "splines",
            "scales",
            "mvtnorm",
            "tmvtnorm",
            "MASS",
            "igraph",
            "stringr",
            "Karen")
lapply(l.pkgs, function(pkg){
```

```
  if(!(pkg %in% rownames(inst.pkgs))){
    install.packages(pkg)
  }
})
## load packages
lapply(l.pkgs, function(pkg){library(pkg,character.only=TRUE)})

rm(list = ls())

rcts <- c("HSC->P1", ## reactions
          "HSC->P2",
          "P1->T",
          "P1->B",
          "P1->NK",
          "P2->G",
          "P2->M",
          "T->0",
          "B->0",
          "NK->0",
          "G->0",
          "M->0"
          ,"HSC->1"
          ,"P1->1"
          ,"P2->1"
)

cnstr <- c("theta\\[\\'HSC->P1\\'\\]=(theta\\[\\'P1->T\\'\\] + theta\\[\\'P1->B\\'\\] + theta\\[\\'P1->NK\\'\
       "theta\\[\\'HSC->P2\\'\\]=(theta\\[\\'P2->G\\'\\] + theta\\[\\'P2->M\\'\\])") ## reaction constraints
latsts <- c("HSC", "P1", "P2") ## latent cell types

ctps <- unique(setdiff(c(sapply(rcts, function(r){ ## all cell types
  as.vector(unlist(strsplit(r, split = "->", fixed = T)))
}, simplify = "array")), c("0", "1")))

########## TRUE PARAMETERS ##########
th.true <- c(0.65, 0.9, 0.925, 0.975, 0.55, 3.5, 3.1, 4, 3.7, 4.1, 0.25, 0.225, 0.275) ## dynamic parameters
names(th.true) <- tail(rcts, -length(cnstr))
s2.true <- 1e-8 ## additonal noise
r0.true <- .1 ## intercept noise parameter
r1.true <- .5 ## slope noise parameter
phi.true <- c(th.true, r0.true, r1.true) ## whole vector parameter
names(phi.true) <- c(names(th.true), "r0", "r1")

########## SIMULATION PARAMETERS ##########
S <- 1000 ## trajectories length
nCL <- 3 ## number of clones
X0 <- rep(0, length(ctps)) ## initial condition
names(X0) <- ctps
X0["HSC"] <- 100
ntps <- 30 ## number of time-points
f_NA <- .75 ## fraction of observed data

############################
## SIMULATE TRAJECTORIES ##
############################

XY <- get.sim.trajectories(rct.lst = rcts,
```

```
                            constr.lst = cnstr,
                            latSts.lst = latsts,
                            ct.lst = ctps,
                            th = th.true,
                            S = S,
                            nCL = nCL,
                            X0 = X0,
                            s2 = s2.true,
                            r0 = r0.true,
                            r1 = r1.true,
                            f = f_NA,
                            ntps = ntps,
                            trunc = FALSE)

####################################
## Fitting Karen on simulated data ##
####################################

nProc <- 1 # number of cores
cat(paste("\tLoading CPU cluster...\n", sep = ""))
cat(paste("Cluster type: ", "PSOCK\n", sep = ""))
cpu <- Sys.getenv("SLURM_CPUS_ON_NODE", nProc) ## define cluster CPUs
hosts <- rep("localhost",cpu)
cl <- makeCluster(hosts, type = "PSOCK") ## make the cluster
rm(nProc)

## mean vector of the initial condition:
m_0 <- replicate(nCL, X0, simplify = "array")
colnames(m_0) <- 1:nCL
## covariance matrix of the initial condition:
P_0 <- Diagonal(length(ctps) * nCL, 1e-5)
rownames(P_0) <- colnames(P_0) <- rep(1:nCL, each = length(ctps))
## Fit Karen on the simulated data:
res.fit <- get.fit(rct.lst = rcts,
                   constr.lst = cnstr,
                   latSts.lst = latsts,
                   ct.lst = ctps,
                   Y = XY$Y[,setdiff(ctps, latsts),],
                   m0 = m_0,
                   P0 = P_0,
                   cl = cl,
                   list(nLQR = 3,
                        lmm = 25,
                        pgtol = 0,
                        relErrfct = 1e-9,
                        tol = 1e-9,
                        maxit = 1000,
                        maxitEM = 10,
                        trace = 1,
                        FORCEP = FALSE))

stopCluster(cl) ## stop the cluster

#########################
## Visualizing results ##
#########################
```

```
## simulated data and smoothing moments
par(mar = c(2,5,2,2), mfrow = c(1,3))
get.sMoments(res.fit = res.fit, X = XY$X)
```

`get.sMoments.avg` *Get the clone-average of the first two-order smoothing moments from a fitted Kalman Reaction Network.*

Description

This function returns the clone-average of the first two-order smoothing moments from a Kalman Reaction Network previously fitted on a clonal tracking dataset.

Usage

```
get.sMoments.avg(res.fit, X = NULL, cell.cols = NULL)
```

Arguments

`res.fit`	A list returned by get.fit() containing the information of a fitted Kalman Reaction Network.
`X`	Stochastic process. A 3-dimensional array whose dimensions are the time, the cell type and the clone respectively.
`cell.cols`	Color legend for the cell types. Defaults to NULL, in which case no color legend for the cell types is provided.

Examples

```
cat("\nInstall/load packages")
inst.pkgs <- installed.packages() ## installed packages

l.pkgs <- c("expm",
            "Matrix",
            "parallel",
            "gaussquad",
            "splines",
            "scales",
            "mvtnorm",
            "tmvtnorm",
            "MASS",
            "igraph",
            "stringr",
            "Karen")
lapply(l.pkgs, function(pkg){
  if(!(pkg %in% rownames(inst.pkgs))){
    install.packages(pkg)
  }
})
## load packages
lapply(l.pkgs, function(pkg){library(pkg,character.only=TRUE)})

rm(list = ls())
```

```
rcts <- c("HSC->P1", ## reactions
          "HSC->P2",
          "P1->T",
          "P1->B",
          "P1->NK",
          "P2->G",
          "P2->M",
          "T->0",
          "B->0",
          "NK->0",
          "G->0",
          "M->0"
          ,"HSC->1"
          ,"P1->1"
          ,"P2->1"
)

cnstr <- c("theta\\[\\'HSC->P1\\'\\]=(theta\\[\\'P1->T\\'\\] + theta\\[\\'P1->B\\'\\] + theta\\[\\'P1->NK\\'\
       "theta\\[\\'HSC->P2\\'\\]=(theta\\[\\'P2->G\\'\\] + theta\\[\\'P2->M\\'\\])") ## reaction constraints
latsts <- c("HSC", "P1", "P2") ## latent cell types

ctps <- unique(setdiff(c(sapply(rcts, function(r){ ## all cell types
  as.vector(unlist(strsplit(r, split = "->", fixed = T)))
}, simplify = "array")), c("0", "1")))

########## TRUE PARAMETERS ##########
th.true <- c(0.65, 0.9, 0.925, 0.975, 0.55, 3.5, 3.1, 4, 3.7, 4.1, 0.25, 0.225, 0.275) ## dynamic parameters
names(th.true) <- tail(rcts, -length(cnstr))
s2.true <- 1e-8 ## additonal noise
r0.true <- .1 ## intercept noise parameter
r1.true <- .5 ## slope noise parameter
phi.true <- c(th.true, r0.true, r1.true) ## whole vector parameter
names(phi.true) <- c(names(th.true), "r0", "r1")

########## SIMULATION PARAMETERS ##########
S <- 1000 ## trajectories length
nCL <- 3 ## number of clones
X0 <- rep(0, length(ctps)) ## initial condition
names(X0) <- ctps
X0["HSC"] <- 100
ntps <- 30 ## number of time-points
f_NA <- .75 ## fraction of observed data

############################
## SIMULATE TRAJECTORIES ##
############################

XY <- get.sim.trajectories(rct.lst = rcts,
                           constr.lst = cnstr,
                           latSts.lst = latsts,
                           ct.lst = ctps,
                           th = th.true,
                           S = S,
                           nCL = nCL,
                           X0 = X0,
                           s2 = s2.true,
```

```
                            r0 = r0.true,
                            r1 = r1.true,
                            f = f_NA,
                            ntps = ntps,
                            trunc = FALSE)

####################################
## Fitting Karen on simulated data ##
####################################

nProc <- 1 # number of cores
cat(paste("\tLoading CPU cluster...\n", sep = ""))
cat(paste("Cluster type: ", "PSOCK\n", sep = ""))
cpu <- Sys.getenv("SLURM_CPUS_ON_NODE", nProc) ## define cluster CPUs
hosts <- rep("localhost",cpu)
cl <- makeCluster(hosts, type = "PSOCK") ## make the cluster
rm(nProc)

## mean vector of the initial condition:
m_0 <- replicate(nCL, X0, simplify = "array")
colnames(m_0) <- 1:nCL
## covariance matrix of the initial condition:
P_0 <- Diagonal(length(ctps) * nCL, 1e-5)
rownames(P_0) <- colnames(P_0) <- rep(1:nCL, each = length(ctps))
## Fit Karen on the simulated data:
res.fit <- get.fit(rct.lst = rcts,
                   constr.lst = cnstr,
                   latSts.lst = latsts,
                   ct.lst = ctps,
                   Y = XY$Y[,setdiff(ctps, latsts),],
                   m0 = m_0,
                   P0 = P_0,
                   cl = cl,
                   list(nLQR = 3,
                        lmm = 25,
                        pgtol = 0,
                        relErrfct = 1e-9,
                        tol = 1e-9,
                        maxit = 1000,
                        maxitEM = 10,
                        trace = 1,
                        FORCEP = FALSE))

stopCluster(cl) ## stop the cluster

#########################
## Visualizing results ##
#########################

## simulated data and clone-average smoothing moments
par(mar = c(2,5,2,2), mfrow = c(1,3))
get.sMoments.avg(res.fit = res.fit, X = XY$X)
```

---

nearestPD *Nearest Positive Definite Matrix*

Description

This function first check if a matrix A is positive definite, typically a correlation or variance-covariance matrix. If A is not positive definite, this function computes the nearest positive definite matrix of A using the function nearPD from package Matrix.

Usage

```
nearestPD(A, ...)
```

Arguments

A	numeric $n \times n$ approximately positive definite matrix, typically an approximation to a correlation or covariance matrix. If A is not symmetric (and ensureSymmetry is not false), symmpart(A) is used.
...	Further arguments to be passed to nearPD (see package Matrix for details).

Value

The nearest positive definite matrix of A.

Y_CT *Clonal tracking data from clinical trials*

Description

A dataset containing clonal tracking cell counts from three different clinical trials.

Usage

```
Y_CT
```

Format

A list containing the clonal tracking data for each clinical trial (WAS, $\beta 0 \beta E$, $\beta S \beta S$). Each clonal tracking dataset is a 3-dimensional array whose dimensions identify

1 time, in months

2 cell types: T, B, NK, Macrophages(M) and Granulocytes(G)

3 unique barcodes (clones)

Source

https://github.com/BushmanLab/HSC_diversity

Y_RM *Rhesus Macaque clonal tracking dataset*

Description

A dataset containing clonal tracking cell counts from a Rhesus Macaque study.

Usage

```
Y_RM
```

Format

A list containing clonal tracking data for each animal (ZH33, ZH17, ZG66). Each clonal tracking dataset is a 3-dimensional array whose dimensions identify

1 time, in months

2 cell types: T, B, NK, Macrophages(M) and Granulocytes(G)

3 unique barcodes (clones)

Source

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3979461/bin/NIHMS567927-supplement-02.xlsx

Index