A quick introduction to Karen

L. Del Core

l.del.core@rug.nl

September 14, 2022

Abstract

This document reviews some key functionalities of the R package Karen. Section 1 shows how to simulate a clonal tracking dataset from a stochastic quasi-reaction network. In particular, we show how to simulate clone-specific trajectories, following a given set of biochemical reactions. Subsequently, Section 2 shows how to fit a Kalman Reaction Network to a simulated clonal tracking dataset. Finally in Section 3 we show how to visualize the results.

1 Simulating clonal tracking datasets

A clonal tracking dataset compatible with Karen's functions must be formatted as a 3-dimensional array Y whose ijk-entry Y_{ijk} is the number of cells of clone k for cell type j collected at time i. The function `get.sim.trajectories()` can be used to simulate clone-specific trajectories given an initial condition X_0 for a set of observed `ct.lst` and latent `latSts.lst`, and obeying to a particular cell differentiation network defined by a list `rct.lst` of biochemical reactions, subject to a set of linear constraints `constr.lst`. In particular, our package considers only three cellular events, such as cell duplication ($X_{it} \to 1$), cell death ($X_{it} \to \emptyset$) and cell differentiation ($X_{it} \to X_{jt}$) for a clone-specific time counting process

$$X_t = (X_{1t}, \dots, X_{Nt}) \tag{1}$$

observed in N distinct cell lineages. The time counting process Y_t for a single clone in a time interval $(t, t + \Delta t)$ evolves according to a set of biochemical reactions defined as

$$v_k = \begin{cases} (0 \dots 1_i \dots 0)' & \text{dup. of the } i\text{-th cell type} \\ (0 \dots -1_i \dots 0)' & \text{death of the } i\text{-th cell type} \\ (0 \dots -1_i \dots 2_j \dots 0)' & \text{diff. of the } i\text{-th type into the } j\text{-th type} \end{cases} \tag{2}$$

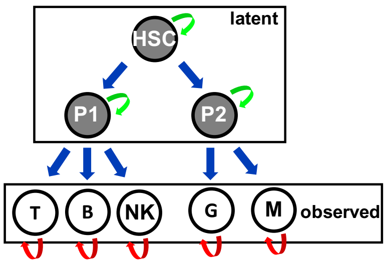


Figure 1: Cell differentiation structure of five observed cell types (white nodes) and three latent cell types (grey nodes). Duplication, death and differentiation moves are indicated with green, red and blue arrows respectively.

with the k-th hazard function given by

$$h_k(X_t, \theta_i) = \begin{cases} X_{it}\alpha_i & \text{for duplication} \\ X_{it}\delta_i & \text{for death} \\ X_{it}\lambda_{ij} & \text{for differentiation} \end{cases} \tag{3}$$

Finally, the net-effect matrix and hazard vector are defined as

$$V = \left[v_1 \cdots v_K\right]; \qquad h(X_t;\theta) = \left[h_1(X_t;\theta) \cdots h_K(X_t;\theta)\right]' \tag{4}$$

Finally we assume that the simulated measurements y_ks are noisy-corrupted and subject to the measurement model

$$\begin{aligned} g_k(x(t_k), r_k) &= G_k x(t_k) + r_k; \quad & r_k \sim \mathcal{N}_d(0, R_k); \\ R_k &= \rho_0 I_d + \rho_1 diag(G_k x(t_k)) \quad & \forall k = 1, \ldots, K \end{aligned} \tag{5}$$

with a time-dependent selection matrix G_k which selects only the measurable cells of $x(t_k)$ with an additive noise r_k having a time-dependent covariance matrix R_k where ρ_0 and ρ_1 are simulation parameters, and $diag(\cdot)$ is a diagonal matrix with diagonal equal to its argument.
The cellular events of duplication, death and differentiation are respectively coded in the package with the character labels "A->1", "A->0", and "A->B", where A and B are two distinct cell types. The following R code chunk shows how to simulate clone-specific trajectories of cells via a Euler-Maruyama simulation algorithm. As an illustrative example we focus on a simple cell differentiation network structure from Figure 1 having eight synthetic cell types. Here we assume that the hematopoietic stem cells HSC, and the two intermediate progenitors P1 - P2 are latent cell types that cannot be measured.

```
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)

XY$X ## process
XY$Y ## measurements
```

2 Fitting a Kalman Reaction Network

The following R code chunk shows how to fit a Kalman reaction network on the previously simulated clonal tracking dataset.

```
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 <- parallel::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 <- Matrix::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-5,
                             tol = 1e-3,
                             maxit = 100,
                             maxitEM = 10,
                             trace = 1,
                             verbose = TRUE,
                             FORCEP = FALSE))
parallel::stopCluster(cl) ## stop the cluster
```

3 Visualizing results

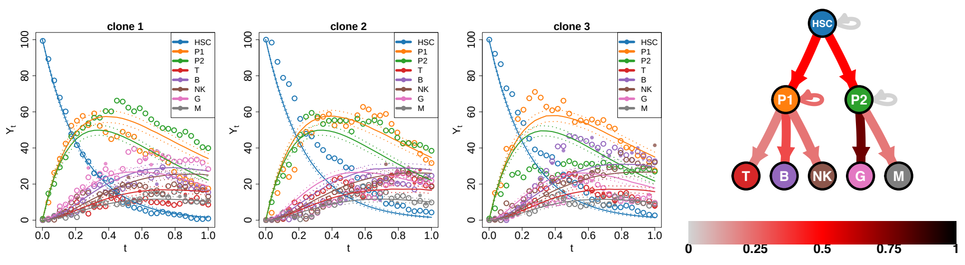
The main graphical output of Karen are a cell differentiation network and the first two order moments of the smoothing distribution. The following R code chunk shows how to obtain these from a previously fitted Kalman Reaction Network.

```
## define color legend for cell types:
cell.cols <-  c("#1F77B4", "#FF7F0E", "#2CA02C", "#E7B928
   ↪ ", "#D62728", "#9467BD", "#8C564B", "#E377C2", "#7
   ↪ F7F7F")
names(cell.cols) <- c("HSC", "P1", "P2", "P3", "T", "B",
   ↪ "NK", "G", "M")

## simulated data and smoothing moments
oldpar <- par(no.readonly = TRUE)
  par(mar = c(5,5,2,2), mfrow = c(ceiling(nCL/ceiling(
     ↪ sqrt(nCL))),ceiling(sqrt(nCL))))
  get.sMoments(res.fit = res.fit, X = XY$X, cell.cols =
     ↪ cell.cols)
par(oldpar)

## Cell differentiation network
oldpar <- par(no.readonly = TRUE)
legend_image <- grDevices::as.raster(matrix(
  grDevices::colorRampPalette(c("lightgray", "red", "
     ↪ black"))(99), ncol=1))
layout(mat = matrix(c(1,1,1,2), ncol = 1))
par(mar = c(1,0,3,0))
get.cdn(res.fit = res.fit,
        edges.lab = F,
        cell.cols = cell.cols)
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)
par(oldpar)
```


clone 1
clone 2
clone 3
HSC
P1
P2
T
B
NK
G
M
Y_t
t