Tutorial: Aufgabe 1

Öffnen Sie die (Tinn-R) Textdatei „all_functions.r" und übertragen Sie den Code aller R-Funktionen (z.B. per „Select all" und „Copy+Paste") in die R Console.

Wichtig: Die R-Pakete ‚ROC' und ‚limma' werden benötigt und müssen ggf. vorher installiert werden.

Wir betrachten die Funktion:

make_test_Data <- function(m_obs, var_noise)

Die Funktion erzeugt einen synthetischen Testdatensatz für den RAF-pathway mit 11 Knoten (Variablen) und 20 Kanten. Die Zusammenhänge zwischen Eltern- und Kinderknoten werden durch einfache lineare Regressionsbeziehungen modelliert. Die Regressionskoeffizienten werden zufällig gesetzt und der generierte Datensatz kann als multivariat-normalverteilt angesehen werden, wobei durch den RAF-pathway Graphen eine Menge bestimmter bedingter Unabhängigkeitsrelationen induziert wird. Mehr Details enthalten die Folien des Tutorials.

Die Input Argumente lauten:

m_obs = Anzahl der unabhängigen Beobachtungen, die generiert werden sollen

var_noise = Varianz σ^2 der Gauß-verteilten Zufallsfehler mit Erwartungswert 0

Erzeugen Sie einen Testdatensatz mit m=100 Beobachtungen, bei denen der Zufallsfehler die Varianz 1 aufweist. Der Output ist eine 11x100 Matrix. Jede Zeile entspricht einem Knoten (Protein) und jede Spalte einer unabhängigen Beobachtung des Netzwerks.

Tutorial: Aufgabe 2

Wir betrachten den structure MCMC Algorithmus bzw. die R-Funktion:
strMCMC <- function(Data, incidence, iterations, step_save)

Diese Funktion führt eine „structure MCMC“ Simulation durch.

Die Input-Argumente sind:

Data = **eine Datenmatrix, wobei Data[i,j] die j-te Realisierung des i-ten Knoten ist.**

incidence = **der Start-Graph, welcher hier als (Incidence-)Matrix übergeben wird.**
Incidence[i,j]=1 bedeutet, dass es eine Kante von X_i nach X_j gibt und
Incidence[i,j]=0 bedeutet, dass es keine Kante von X_i nach X_j gibt

iterations = **die Anzahl der durchzuführenden MCMC Iterationen (structure MCMC moves).**

step_save = **dieser Parameter regelt, nach wie vielen MCMC Schritten ein Graph gesampelt wird.**
step_save=10 z.B. bedeutet, dass in der MCMC Simulation jeder 10-te Graph gesampelt (gespeichert) wird.

Starten Sie drei unabhängige MCMC Simulationen zur Inferenz des Datensatzes, der in Aufgabe 1 generiert wurde. Starten Sie jeweils von einem Ausgangsgraphen ohne Kanten und führen Sie 20,000 MCMC Iterationen aus. Dies kann pro Lauf einige Minuten Rechenzeit in Anspruch nehmen.
Wenn Sie jeden 100-ten Graphen sampeln (speichern) erhalten Sie eine Graphenstichprobe der Größe 200.

Machen Sie sich mit der Ausgabe der Funktion strMCMC.R vertraut. Ausgegeben wird eine Liste, in der die gesampelten Graphen (an erster Stelle [[1]]) und deren BGe score (marginale Likelihood) (an zweiter Stelle [[2]]) gespeichert wurden. Zu beachten ist, dass die ersten Listeneinträge der Initialisierung entsprechen; hier also dem leeren Graph und der Score des leeren Graphen. Die Liste enthält daher jeweils 201 Graphen/Scores.

Tutorial: Aufgabe 3

Konvergenzdiagnostik basierende auf Trace-Plots der Graphen Scores

Eine notwendige (nicht hinreichende) Bedingung für Konvergenz der MCMC Simulation, ist dass sich die Scores der Graphen, die während der drei (unabhängigen) MCMC Läufe erzeugt wurden, nach einer bestimmten Zeit (dem Burn-In) nicht mehr systematisch unterscheiden.

Die Funktion strMCMC gibt eine Liste mit 2 Elementen aus und die zu plottenden Scores finden sich an zweiter Stelle (symbolisch [[2]]) dieser Liste.

Auch sollte für jedem einzelnen Lauf gelten, dass nach dem „Burn-In" Stationarität vorliegt. Auch systematische Veränderungen der Score-Mittelwerte oder Score-Varianzen einer einzelnen MCMC Simulation deuten darauf hin, dass die Markovkette noch nicht konvergiert hat.

Erzeugen Sie daher als Konvergenzdiagnostik einen "Traceplot", in welchem die Indicis der Graphen (x-Achse) gegen die BGe Scores (y-Achse) abgetragen werden. Bei Verwendung von drei verschiedenen Farben können alle drei Kurven in einem einzigen Plot dargestellt werden.

Gibt es Indizien, die hier bei Ihren Simulationen gegen (hinreichende) Konvergenz sprechen?

Tutorial: Aufgabe 4

Konvergenzdiagnostik basierend auf den marginalen aposteriori Wahrscheinlichkeiten der „directed edge relation features“:

Eine weitere notwendige (aber ebenfalls nicht hinreichende) Bedingung für Konvergenz der MCMC Simulation, ist dass sich die Schätzungen der marginalen aposteriori Wahrscheinlichkeiten der „directed edge relation features“ von MCMC Lauf zu MCMC Lauf nicht zu stark unterscheiden.

Berechnen Sie deshalb aus den drei MCMC Graphenstichproben die marginalen posterior Wahrscheinlichkeiten aller „directed edge relation features“ und tragen Sie diese Wahrscheinlichkeiten in Scatter Plots (Lauf 1 vs. Lauf 2, Lauf 2 vs. Lauf 3 und Lauf 1 vs. Lauf 3) gegeneinander ab.

Streuen die Punkte um die Diagonale oder gibt es starke Abweichungen?
Wenn die marginalen Wahrscheinlichkeiten unkorreliert sind, spricht dies gegen hinreichende Konvergenz.

Anleitung:
Matrizendarstellungen der marginalen Kantenwahrscheinlichkeiten erhalten Sie mit der R-Funktion:
cpdag_list <- function(list.inc,E)

list.inc sind die Graphen die mit der Funktion strMCMC erzeugt wurden (genauer der erste Listeneintrag (symbolisch [[1]]) des Outputs von strMCMC)
E>0 ist der Start der Sampling Phase (d.h. der E-te Graph ist der erste bei der Berechnung zu berücksichtigende Graph bzw. der erste Graph nach der Burn-In Phase)

Im dritten Listeneintrag (symbolisch [[3]]) des Outputs finden sich die marginalen Kantenwahrscheinlichkeiten. Die Nichtdiagonalelemente dieser Matrix sollten zu einem Vektor transformiert werden. Die Vektoren, die man für die drei Läufe per Transformation erhält, können dann jeweils paarweise in einem Scatter Plots gegeneinander abgetragen werden.
Tipp: Übersichtlicher werden die Streudiagramme, wenn man jeweils auch noch eine diagonale Referenzlinie einträgt.

Tutorial: Aufgabe 5

Beurteilung der Güte der Netzwerkrekonstruktion

Erzeugen Sie als erstes den „wahren" RAF-pathway Graphen mit der R-Funktion:
make_true_Net <- function()

Die Netzwerkrekonstruktion kann zum Beispiel mit Hilfe einer Receiver-Operator-Characteristic Curve beurteilt werden.

Nutzen Sie zur Erstellung einer ROC Kurve die R-Funktion:
draw_ROC <- function(postP, trueEdges)

Und berechnen Sie die area under the ROC Kurve (AUROC) mit der Funktion:
compute_AUROC <- function(postP, trueEdges)

Die Input-Argumente sind jeweils:
postP = Die geschätzten marginalen aposteriori Wahrscheinlichkeiten der gerichteten Kanten in Matrixform; der dritte Listeneintrag (symbolisch [[3]]) des Outputs von cpdag_list
trueEdges = der wahre Graph als (Incidence-)Matrix, hier der Output von make_true_net

Wie beurteilen Sie die Güte der Netzwerkrekonstruktion?

Optional können Sie jetzt testen welche Effekte die verschiedenen Parameter auf die Güte der Netzwerkrekonstruktion haben. (i) Was passiert, wenn die Anzahl der Beobachtungen m auf 10 oder 1000 gesetzt wird? (ii) Was passiert wenn für m=100 die Varianz 6^2 des Noise-Terms auf 0.1 oder 5 gesetzt wird? (iii) Ebenfalls interessant wäre eine Untersuchung, ob m einen Effekt auf die Konvergenz der MCMC Läufe hat? Reichen 20000 MCMC Iterationen bei m=1000 Beobachtungen aus?

```
# Aufgabe 1
m_obs     <- 100
var_noise <- 1
Data      <- make_test_Data(m_obs,var_noise)

# Aufgabe 2
n_nodes    <- nrow(Data)
iterations  <- 20000
step_save  <- 100

start_incidence <- matrix(0,n_nodes,n_nodes)

Sim_1 <- strMCMC(Data, start_incidence, iterations, step_save)
Sim_2 <- strMCMC(Data, start_incidence, iterations, step_save)
Sim_3 <- strMCMC(Data, start_incidence, iterations, step_save)

# Aufgabe 3
n_1 <- length(Sim_1[[2]])
n_2 <- length(Sim_2[[2]])
n_3 <- length(Sim_3[[2]])

plot(1:n_1,Sim_1[[2]], type="l", ylab="log P(D|G)", xlab="graphs", main="Trace Plots")
lines(1:n_2,Sim_2[[2]], col="blue")
lines(1:n_3,Sim_3[[2]], col="red")
```

```
# Aufgabe 4
cpdags_1 <- cpdag_list(Sim_1[[1]],1)
cpdags_2 <- cpdag_list(Sim_2[[1]],1)
cpdags_3 <- cpdag_list(Sim_3[[1]],1)

edges_mat_1 <- cpdags_1[[3]]
edges_mat_2 <- cpdags_2[[3]]
edges_mat_3 <- cpdags_3[[3]]

edges_vec_1 <- numeric(0)
edges_vec_2 <- numeric(0)
edges_vec_3 <- numeric(0)

for (i in 1:n_nodes){
      for (j in 1:n_nodes){
                              if (abs(i-j)>0){
                                          edges_vec_1 <- c(edges_vec_1,edges_mat_1[i,j])
                                          edges_vec_2 <- c(edges_vec_2,edges_mat_2[i,j])
                                          edges_vec_3 <- c(edges_vec_3,edges_mat_3[i,j])
                                          }
                               }
                             }

par(mfrow=c(2,2))
plot(edges_vec_1,edges_vec_2,pch=4, xlab="Run 1", ylab="Run 2", main="1 versus 2")
lines(0:1,0:1, type='l', col="red")
plot(edges_vec_2,edges_vec_3,pch=4, xlab="Run 2", ylab="Run 3", main="2 versus 3")
lines(0:1,0:1, type='l', col="red")
plot(edges_vec_1,edges_vec_3,pch=4, xlab="Run 1", ylab="Run 3", main="1 versus 3" )
lines(0:1,0:1, type='l', col="red")
```

```
# Aufgabe 4b – Slightly modified to visualise clusters of points
eps      <- 0.05
n_edges  <- length(edges_vec_1)

par(mfrow=c(2,2))

edges_vec_1b <- edges_vec_1 + eps*rnorm(n_edges)
edges_vec_2b <- edges_vec_2 + eps*rnorm(n_edges)
edges_vec_3b <- edges_vec_3 + eps*rnorm(n_edges)

plot(edges_vec_1b,edges_vec_2b, pch=4, xlab="Run 1", ylab="Run 2" , main="1 versus 2")
lines(0:1,0:1, type='l', col="red")
plot(edges_vec_2b,edges_vec_3b, pch=4, xlab="Run 2", ylab="Run 3", main="2 versus 3")
lines(0:1,0:1, type='l', col="red")
plot(edges_vec_1b,edges_vec_3b, pch=4, xlab="Run 1", ylab="Run 3" , main="1 versus 3")
lines(0:1,0:1, type='l', col="red")

# Aufgabe 5
true_Net <- make_true_Net()

par(mfrow=c(2,2))
void <- draw_ROC(edges_mat_1, true_Net)
void <- draw_ROC(edges_mat_2, true_Net)
void <- draw_ROC(edges_mat_3, true_Net)

AUC_1 <- compute_AUROC(edges_mat_1,true_Net)
AUC_2 <- compute_AUROC(edges_mat_2,true_Net)
AUC_3 <- compute_AUROC(edges_mat_3,true_Net)
```