chipDyno User Guide

Muhammad A. Rahman, Guido Sanguinetti, Magnus Rattray and Neil D. Lawrence

November 3, 2014

Abstract

Quantitative estimation of the regulatory relationship between transcription factors and genes is a fundamental stepping stone when trying to develop models of cellular processes. This task, however, is difficult for a number of reasons: transcription factors' expression levels are often low and noisy, and many transcription factors are post-transcriptionally regulated. It is therefore useful to infer the activity of the transcription factors from the expression levels of their target genes.

Contents

1 Mirroring the chipDyno repository from GitHub

The recommended way to clone the package from GitHub. Installation should ensure that all the dependencies are met.

```
$ clone https://github.com/SheffieldML/chipDyno.git
$ cd chipDyno/
```

2 Installation from .tar file

After downloading the package it can also be installed in an specific location. If we want to use the directory $/data/Rpackages/$ then creating a package directory, the installation can be done in the following way:

```
$ sudo R
...
> install.packages("chipDynoTest2*", lib="/data/Rpackages/")
> library(chipDynoTest2*, lib.loc="/data/Rpackages/")
```

Or from the command line:

```
$ R CMD INSTALL chipDyno*.tar.gz
```

3 Loading the package and getting help

The first step in any *chipDyno* analysis is to load the package. This package can be loaded when the *R* console is ready. At the *R* console type the following command:

```
> library("chipDyno")
```

Command `help` can be used to get help on any function. For example, to get help on the `chipDynoLikeStatGrad` type the following (both of them have the similar output):

```
> help(chipDynoLikeStatGrad)
> ?chipDynoLikeStatGrad
```

4 Input data and preparations

If the data is in Affymetrix CEL files then it may required to do some preprocessing. This CEL files can be extracted using the bioconductor package *puma* [Pearson et al., 2009]. *puma* usually stores the point expression in '*_exprs.csv' file and defferent levels of uncertainties in '*_se.csv' files as Comma-separated values.

Load the point expression levels data and it corresponding uncertainty. Here the data was stored in a simple text file

```
> rm(list=ls())
> source("../chipDynoLoadModules.R")
> file_data <-
+   "../data/MetabolData/YeastMetabolism_exprs.txt";
> file_vars <-
+   "../data/MetabolData/YeastMetabolism_se.txt";
> data = as.matrix(read.table(file_data))
> data[1:5,1:8]
```

```
            V1        V2        V3        V4        V5         V6         V7         V8
[1,]  3.032196  3.915297  3.513140  2.887694  2.624707  2.4336185  0.5146789  3.0277957
[2,]  7.619968  7.009363  6.327303  6.420305  6.474659  6.4259952  5.9848784  6.6202407
[3,]  9.367415  9.337176  9.346070  9.528845  9.870827  9.9068906  9.2679571  9.7358944
[4,] 11.116018 10.157765 10.559681 10.625835 10.832905 11.1418061 10.7627800 11.1802351
[5,] -1.746557 -1.909657  1.074617 -2.313146 -1.508921 -0.9244119 -1.7330923 -0.8551462
```

```
> vars=as.matrix(read.table(file_vars))
> vars[1:5,1:8]
```

```
            V1        V2        V3        V4        V5         V6        V7         V8
[1,] 0.5988238 0.3692819 0.4940527 0.6459059 0.8544423 0.82382426 1.4128785 0.62460252
[2,] 0.1291985 0.1683352 0.2438547 0.2245224 0.2532125 0.23656530 0.2989101 0.21290733
[3,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00000000 0.0000000 0.00000000
[4,] 0.1019400 0.1426169 0.1248059 0.1224968 0.1122887 0.09996799 0.1154624 0.09959699
[5,] 1.8253530 1.8705859 0.9995836 1.9992325 1.8095992 1.60575819 1.8439322 1.57351633
```

These differentially expressed data and its uncertainty doesn't contain any genes name or problem ID number. The gene Names and its corresponding problem IDs can be extracted from other two source (?) dictionary.txt and probeIDTu.txt.

```
> file_dictionary <-
+   "../data/MetabolData/dictionary.txt";
> dictionary <- read.table(file_dictionary, header = FALSE, sep = "\t",
+                     quote = "", dec = ".", col.names=c("IDs_temp", "ORF",
+                                  "ORF_temp","geneCN_temp"), na.strings = "NA",
+                                  colClasses = NA, nrows = -1, skip = 0, check.names = TRUE,
+                                  fill = TRUE, strip.white = FALSE, blank.lines.skip = FALSE,
+                                  comment.char = "#", allowEscapes = FALSE, flush = FALSE)
> IDs= matrix(dictionary$IDs_temp,,1)
> ORF= matrix(dictionary$ORF,,1)
> file_probeIDTu <-
+   "../data/MetabolData/probeIDTu.txt";
> probeIDTu <- read.table(file_probeIDTu, sep=" ", fill= TRUE, col.names=c("IDSn"))
> IDSnew <- matrix(probeIDTu$IDSn,,1)
```

The connectivity contains the evidence of connectivity between a gene and a transcription factor. `Connectivity2.txt` file depicts the relation between 6229 genes and 204 transcription factors. Load the connectivity file.

```
> file_dataChip <-
+   "../data/Connectivity2.txt";
> dataChip=as.matrix(read.table(file_dataChip))
```

From the annotation file we can find the gene names present in the connectivity information. Load the annotation file.

```
> file_annotation <-
+   "../data/annotations2.txt"
> probe_anno <- read.table(file_annotation, header = FALSE, sep = "\t", quote = "",
+                          dec = ".", col.names=c("prob", "anno"), na.strings = "NA",
+                          colClasses = NA, nrows = -1, skip = 0, check.names = TRUE,
+                          fill = TRUE, strip.white = FALSE, blank.lines.skip = FALSE,
+                          comment.char = "#", allowEscapes = FALSE, flush = FALSE)
> probeName2=matrix(probe_anno$prob,,1)
> annota=matrix(probe_anno$anno,,1)
```

The list of transcription factors present in the connectivity information can be found from the file

```
> file_transNames <-
+   "../data/Trans_Names2.txt"
```

```
> TransNames_tab <- read.table(file_transNames, header = FALSE, sep = "\t", quote = "",
+                              dec = ".", col.names=c("TN"), na.strings = "NA",
+                              colClasses = NA, nrows = -1, skip = 0, check.names = TRUE,
+                              fill = TRUE, strip.white = FALSE, blank.lines.skip = FALSE,
+                              comment.char = "#", allowEscapes = FALSE, flush = FALSE)
> TransNames = matrix(TransNames_tab$TN,,1)
```

In our experimental data set there are some genes for which the measurement of uncertainty is zero(i.e. vars[3,]). We would like to exclude those data.

```
> zeroValueRow = which(rowSums(vars)==0)
> data = data[-zeroValueRow, ]
> vars = vars[-zeroValueRow, ]
> probeName = ORF # Rename ORF
> probeName = matrix(probeName[-zeroValueRow, ],,1)
```

All the genes represent the connectivity information with the transcription factors might not present in the differentially expressed data set. We will like to exclude those genes. The following segment of code will exclude those genes. We will also exclude the redundant genes, their point expression data and uncertainty levels.

```
> noOfprobe=nrow(probeName)
> redundancy=matrix(mat.or.vec(noOfprobe,1),,1)
> redundancy[,]=1
> index=matrix(mat.or.vec(nrow(dataChip),1),,1)
> for (i in 1: nrow(probeName2)){
+         vec <- probeName==probeName2[i,1]
+         index[i]=colSums(vec)
+         if(index[i]>1){
+                 pippo=(which(vec))
+                 redundancy[pippo[2:length(pippo)]]=0
+                 }
+         }
> dataChip=dataChip[which(index!=0),]
> annota=annota[which(index!=0),]
> probeName2=probeName2[which(index!=0),]
> probeName2=matrix(probeName2,,1)
> probeName=probeName[which(redundancy!=0),]
> probeName=matrix(probeName,,1)
> data=data[which(redundancy!=0),]
> vars=vars[which(redundancy!=0),]
```

Again all the genes have the differentially expressed data may not have the connectivity information with the transcription factors. We will exclude those genes as well. We will also organize the genes connectivity information based on the differentially expressed genes index. So that in both data set (points expressions with uncertainty and connectivity information) have the common genes and aligned with the same index.

```
> preX=NULL
> annotation=NULL
> index=mat.or.vec(nrow(data),1)
> for (i in 1: nrow(data)){
+         index[i]=sum(probeName[i]==probeName2)
+         if (index[i]==1)
+                 preX=rbind(preX,dataChip[which(probeName[i]==probeName2),])
```

```
+                 annotation=rbind(annotation,annota[which(probeName[i]==probeName2)])
+ }
> data= data[which(index==1),]
> vars= vars[which(index==1),]
> probeName= probeName[which(index==1),]
> probeName=matrix(probeName,,1)
```

The connectivity matrix between genes and the transcription factors is a binary matrix. If there is an evidence that a gene is transcribed by a transcription factor or, a transcription factor is transcribing some other genes then the relation will be 1, the value will be 0 otherwise. We binarized the matrix by giving a value based on the suggestion of [Lee et al., 2002] when the associated p-value is less than 0.0001.

```
> X= mat.or.vec(nrow(preX),ncol(preX))
> I <- c(which(preX<1e-3))
> X[I] =1
```

In the experimental dataset there might have some genes which are not transcribing by any of the given transcription factors, again there might have some transcription factors which are not transcribing any of the present genes. We will exclude those genes and transcription factors.

```
> fakeX = rowSums(X)
> X=X[which(fakeX!=0),]
> annotation=annotation[which(fakeX!=0),]
> annotation = matrix(annotation,,1)
> effectX=colSums(X)
> TransNames=TransNames[which(effectX!=0),]
> TransNames=matrix(TransNames,,1)
> X=X[,which(effectX!=0)]; X[201:207,16:30]
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15]
[1,]    0    0    0    0    0    1    0    0    0     0     0     0     0     0     0
[2,]    0    1    0    0    0    0    0    0    0     0     0     0     0     0     0
[3,]    0    0    0    0    0    1    0    0    0     0     0     0     1     0     0
[4,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0
[5,]    0    0    0    0    0    1    0    0    0     0     0     0     1     0     0
[6,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0
[7,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0
```

```
> data= data[which(fakeX!=0),]; data[1:5,1:7]
```

```
             V1          V2        V3        V4         V5          V6         V7
[1,]  7.6199683  7.00936333  6.327303  6.420305  6.4746593  6.42599518  5.9848784
[2,] 11.1160178 10.15776528 10.559681 10.625835 10.8329051 11.14180612 10.7627800
[3,]  5.6794099  5.94811239  5.656218  6.013510  6.6347741  6.82920197  6.4394373
[4,] -0.8461938 -0.09323972  1.174838  2.204736  0.7630712  0.05440951  0.9871222
[5,] 11.2509533 11.10515487 10.940308 10.467595 10.3794020  9.96586556  9.8538724
```

```
> vars= vars[which(fakeX!=0),]; vars[1:5,1:7]
```

```
             V1         V2         V3         V4         V5         V6        V7
[1,] 0.12919850 0.16833525 0.24385468 0.22452239 0.25321253 0.23656530 0.2989101
[2,] 0.10194004 0.14261692 0.12480590 0.12249678 0.11228873 0.09996799 0.1154624
[3,] 0.35203433 0.30378077 0.36894665 0.29941130 0.24717867 0.20727438 0.2476667
[4,] 2.14604703 1.89233701 1.47218035 1.15147407 1.67454729 1.86261881 1.5501621
[5,] 0.06955393 0.07214869 0.07750579 0.09116104 0.09549444 0.10993607 0.1157719
```

Define some variables-

```
> nGenes= nrow(data) # Number of genes will be present in our experiment
> npts= ncol(data) # Number of time points
> nTrans = ncol(X) # Number of TF will be present in our experiment
> muIn = array(0, dim <-c(nTrans,1));
> annotations = annotation # Both of the variavle contain the same data
> transNames = TransNames # Both of the variavle contain the same data
```

5 Reduce Variables

The connectivity matrix is a sparse matrix. Using `chipReduceVariables` we can find the same length integer vectors having the row indices, column indices, the values of the non-zero entries of the sparce matrix, and the number of effective genes

```
> R_C_V_nEffectGenes = chipReduceVariables(X);
> R = R_C_V_nEffectGenes[[1]]; R[1:10]

 [1] 289 458  17  27  35  47  59  75  98 102

> C = R_C_V_nEffectGenes[[2]]; C[1:10]

 [1] 1 1 2 2 2 2 2 2 2 2

> V = R_C_V_nEffectGenes[[3]]; V[1:10]

 [1] 1 1 1 1 1 1 1 1 1 1

> nEffectGenes = R_C_V_nEffectGenes[[4]]; nEffectGenes

[1] 580
```

Initialize some variables-

```
> diagonal = array(0.5, dim <- c(1,nTrans))
> precs_mat = array(1, dim <- c(nrow(vars),ncol(vars)))
> precs = precs_mat/(vars^2)
> beta=3;
> gamma=pi/4;
> params=matrix(c(beta,gamma,t(muIn),0.1*t(V), diagonal),1,)
```

6 Likelihood Optimization

We choose to optimize the likelihood using scaled conjugate gradient algorithm implement in `Netlab` [Nabney, 2002].

```
> options = array(0, dim <- c(1,18))
> options[1]=1; #display error values
> options[2]=0.0001 # measure of the absolute precision
> options[3]=0.0001  #objective function values between two successive steps to satisfy condit
> options[14]= 2000  # maximum number of iterations
> #options[17]=0.1
```

Optimize all the parameters using scaled conjugate gradient algorithm. This process is time consuming. The following line of command may take several hours to run. Here we can reduce the number of iteration to speed up the process and later we will load some optimized data. Figure. 1 shows the mean value of the transcription factors activity.

```
> options[14]= 3 # Number if iteration; only for trial purpose.
> #params = SCGoptimNoise(params, options, data, precs, X, nEffectGenes, R, C)
```

Load the optimized value-

```
> require("Matrix")
> source("../chipDynoLoadModules.R")
> load("../ResultsTu_Final.RData")
> V=params[(3+nTrans):(length(params)-nTrans)]
> preSigma <- as.matrix(sparseMatrix(R, C, x=V, dims = c(nEffectGenes,nTrans)))
> diagonal = params[(length(params)-nTrans+1) :(length(params))];
> Sigma = t(preSigma)%*%preSigma + diag(diagonal*diagonal)
> beta = params[1]
> gamma = params[2]
> mu = params[3:(2+nTrans)]
> #save.image("ResultsTu_test.RData") # To save the data
```

7 Computation of posterior expectations for a given gene and TF

chipDynoStatPostEst and chipDynoStatPostEstNoise computes posterior expectations of transcription factor activity for a transcription factor and a gene given the uncertainty of the expression level is absent or present respectively.

```
> #rm(list=ls())
> load("../ResultsTu_Final.RData")
> source("../chipDynoLoadModules.R")
> annotations = annotation
> transNames=TransNames
> transName= "ACE2"
> geneName="YHR143W"
> #geneName="KRE32"
>
> npts=ncol(data);
> nTrans=ncol(X);
> c = class(geneName)
> v= mat.or.vec(length(annotations),1)
> if (c == 'character'){
+   for (i in 1: nrow(annotations)) {
+                 v[i] <- geneName==annotations[i,1]
+                 }
+         x=data.matrix(X[which(v==1),])
+         data=t(data[which(v==1),])
+         precs = precs[which(v==1),]
+
+ } else if (c=='integer'){
+         x=data.matrix(X[geneName,])
```

```
> plot(mu, type='s', col='red', xlab= "Transcription Factors", ylab= "Mean Value")
```

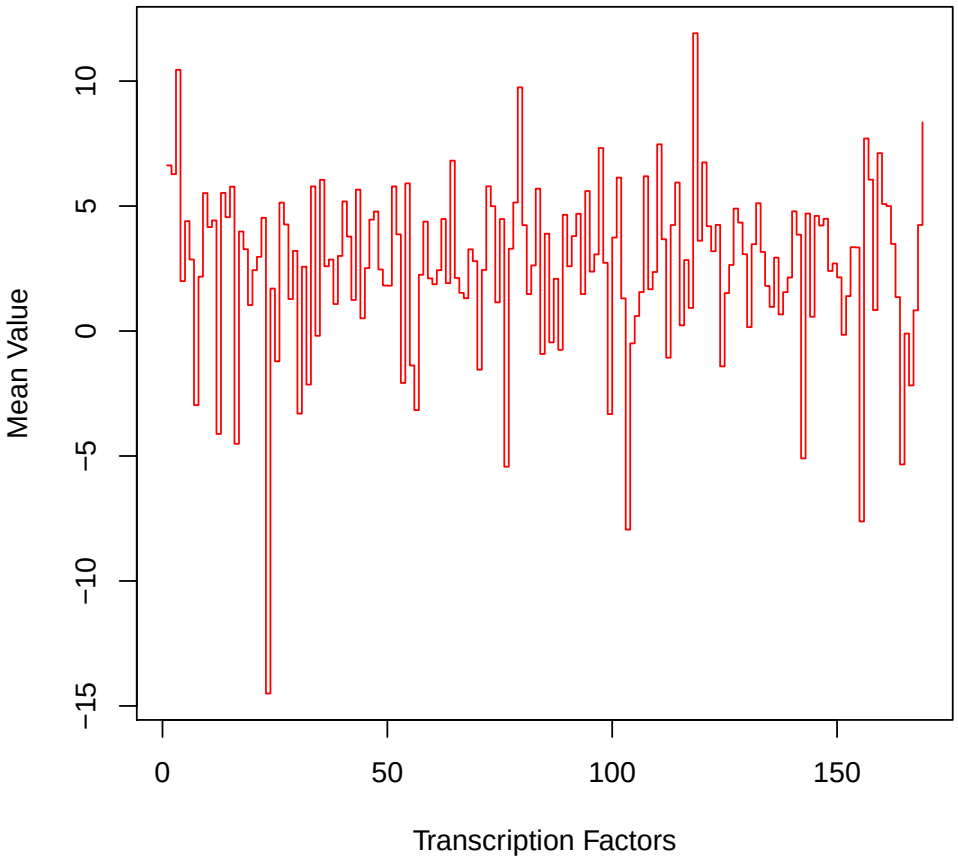

Figure 1: Posterior Mean of TFA

```
+         data=data[geneName,]
+         precs=precs[geneName,] ### t() ???
+
+ } else {
+         print('Error: Genes can be identified either by number or name')
+ }
> expectations=chipDynoStatPostEstNoise(data,x,Sigma,beta,precs,gamma,mu);
> expectations.b=expectations[[1]];
> expectations.tfError=expectations[[2]];
> expectations.tfErrorDiffs=expectations[[3]];
> index=which(transName==transNames);
> if (x[index,]== 0) {
+  print('Error: The gene selected is not a target of the transcription factor')
+ }
> tf=expectations.b[,index];
> ind=which(transName == transNames[which(x!=0)]);
> tfErrors=expectations.tfError[,ind];
> tfErrorsDiffs=expectations.tfErrorDiffs[,,ind];
```

```
> tf

 [1] 0.7626810 1.3371938 2.4517332 2.5973486 2.6137437 2.5888224 2.1749048 1.6951394
 [9] 2.1091986 1.0651819 1.0096594 0.9626544 0.8710457 1.5893280 2.6575006 2.5570126
[17] 2.3549824 2.6979573 2.2454622 1.8153979 1.3135126 1.2849328 1.4638310 1.0960801
[25] 1.0105326 1.9398684 2.4904246 2.7819267 2.5135438 2.3421242 2.1452290 2.2057685
[33] 1.6532838 1.3664687 1.2549145 0.9006209

> tfErrors

 [1] 2.773261 2.760529 2.758281 2.758231 2.758226 2.758233 2.758432 2.759117 2.758469
[10] 2.764042 2.762704 2.762955 2.770898 2.759345 2.758213 2.758245 2.758325 2.758204
[19] 2.758397 2.758859 2.760652 2.760853 2.759780 2.762497 2.768186 2.758657 2.758265
[28] 2.758191 2.758256 2.758334 2.758457 2.758394 2.759217 2.760254 2.760747 2.766046
```

8 Posterior expectations for a given TF : chipDynoExpectationsFastNoise

chipDynoExpectationsFast and chipDynoExpectationsFastNoise computes posterior expectations of transcription factor activity for a given transcription factor considering uncertainty of the expression level absent or present respectively. These functions first find all the genes gegulated by geven transcription factor and later the activity on different genes. The format of the function and arguments are-

```
> rm(list=ls())
> load("ResultsSpellman_Final.RData")
> #load("../ResultsTu_Final.RData")
> source("../chipDynoLoadModules.R")
> transNames = TransNames
> annotations = annotation
> nTrans=nrow(TransNames);
> lst=list();
> newX=array(0, dim <-c(dim(X)));
> newXVals=array(0, dim <-c(dim(X)));
> #i=2; name=TransNames[i,] # ACE2 for demSpellman
>
> name="ACE2"
> index=which(name == transNames);
> genesIn=which(X[,index]!=0);
> anno=annotations[which(X[,index]!=0)];
> nTargets=length(anno);
> npts=ncol(data);
> TF=array(0, dim=c(nTargets,npts));
> TFError=array(0, dim=c(nTargets,npts));
> TFErrorDiff=array(0, dim=c(npts,npts,nTargets));
> plotErrorBar <- function(y,SE, gnName){
+
+ add.error.bars <- function(x,y,SE,w,col=1){
+  x0 = x; y0 = (y-SE); x1 =x; y1 = (y+SE);
+  arrows(x0, y0, x1, y1, code=3,angle= 90,length=w,col=col);
+  }
+
+ x <- c(1:length(y));
+ plot(x,y, type = 'l', col='green4', las=1, xlab="time", ylab="TFA",
```

```
> M <- matrix(c(rep(1:4)), byrow=TRUE, nrow=2) # Choose the position by matrix setting!
> layout(M)
> k <- c(46,47,50,52)
> #for (i in 1:4){
> for (i in k){
+ #expectations = chipDynoExpectationsFast(data,X,Sigma,beta,gamma,mu,
+ #                    transNames, annotations, name, genesIn[i]);
+ expectations = chipDynoExpectationsFastNoise(data,X,Sigma,beta,precs,
+                     gamma,mu, transNames, annotations, name, genesIn[i]);
+ TF[i,] = expectations[[1]];
+ TFError[i,] = expectations[[2]];
+ gnName=annotation[genesIn[i]]
+ plotErrorBar(TF[i,],TFError[i,], gnName)
+ }
```

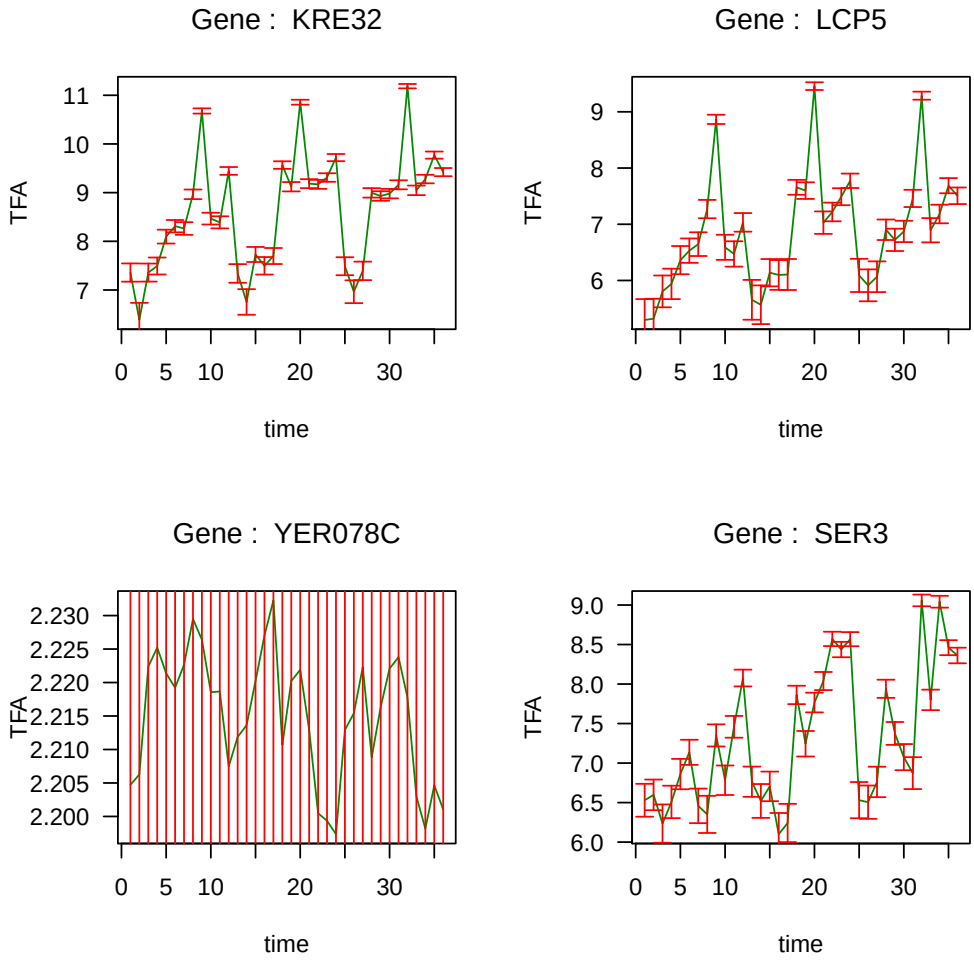

Figure 2: TFA Activities

```
+       main=bquote("Gene : " ~ .(gnName)));
+ add.error.bars(x,y,SE,0.05,col='red');
+ }
```

```
> M <- matrix(c(rep(1:4)), byrow=TRUE, nrow=2) # Choose the position by matrix setting!
> layout(M)
> k <- c(17,19, 28, 50)
> #for (i in 8:11){
> for (i in k){
+ expectations = chipDynoExpectationsFast(data,X,Sigma,beta,gamma,mu, transNames, annotations,
+ #expectations = chipDynoExpectationsFastNoise(data,X,Sigma,beta,precs, gamma,mu, transNames,
+ TF[i,] = expectations[[1]];
+ TFError[i,] = expectations[[2]];
+ gnName=annotation[genesIn[i]]
+
+ plotErrorBar(TF[i,],TFError[i,], gnName)
+ }
```

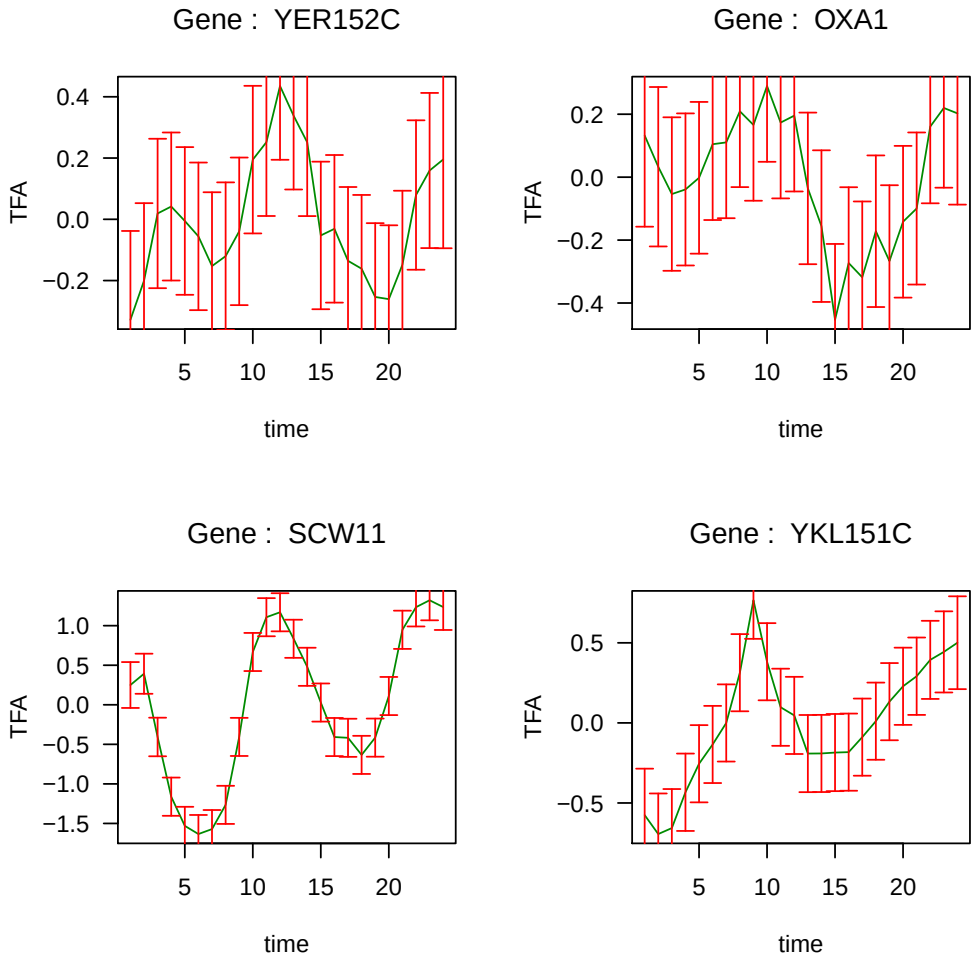

Figure 3: TFA Activities of ACE2

9 Maximum Transcription Factor Activity

chipDynoMaxDiff computes the maximum transcription factor activity of all the target or transcribed genes for a given transcription factor. For a specific target it also computes the variation over time.

```
> rm(list=ls())
> load("../ResultsTu_Final.RData")
```

```
> source("../chipDynoLoadModules.R")
> transNames = TransNames
> annotations = annotation
> nTrans=nrow(TransNames);
> lst=list();
> newX=array(0, dim <-c(dim(X)));
> newXVals=array(0, dim <-c(dim(X)));
> i=6; # TransNames[6,] is "AFT2"
> expectations =chipDynoTransFactNoise(data,X,Sigma,beta,precs, gamma, mu,
+                                      TransNames, annotation, TransNames[i,]);
> TF = expectations[[1]]
> TFError = expectations [[2]]
> TFErrorDiff = expectations [[3]]
> maxVars=chipDynoMaxDiff(TF,TFErrorDiff);
> #maxVars
```

10 TFAs with error bars :chipDynoTransFactNoise

chipDynoTransFactNoise provides transcription factor activities with error margin for a given transcription factor. For an example we are interested to find the activity of transcription factor ACE2. Using chipDynoTransFactNoise we can find it by following way-

```
> rm(list=ls())
> load("../ResultsTu_Final.RData")
> #load("../ResultsTu.RData")
> source("../chipDynoLoadModules.R")
> i=4; TransNames[i,]

[1] "ACE2"

> expectations =chipDynoTransFactNoise(data,X,Sigma,beta, precs, gamma, mu,
+                                      TransNames, annotation, TransNames[i,]);
> TransNames[i,] # Name of the transcription factor

[1] "ACE2"

> expectations[[1]][1:5,1:5] # Transcription facot activity on different genes

            [,1]      [,2]       [,3]       [,4]       [,5]
[1,]  1.11519852 1.4685515 1.92696131 2.06662345 2.09650500
[2,]  2.93071966 2.7465487 2.57092062 2.80903684 2.90590250
[3,] -0.03532873 0.1613834 0.05887395 0.06218098 0.07963148
[4,]  0.75657027 0.8134580 0.74290215 0.74045359 0.74227924
[5,]  1.97300550 1.9733308 1.96459567 1.96068312 1.96982907

> expectations[[2]][1:5,1:5] # Transcription facot activity error on different genes

         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 2.825080 2.824669 2.824563 2.824552 2.824549
[2,] 2.537083 2.537217 2.537447 2.537162 2.537087
[3,] 3.442080 3.441786 3.441868 3.441853 3.441868
[4,] 3.441750 3.441747 3.441751 3.441751 3.441751
[5,] 3.564012 3.564012 3.564012 3.564012 3.564012

> #expectations[[2]]
```

11 Significantly varying TFs :chipDynoActTransFactNoise

chipDynoActTransFact and chipDynoActTransFactNoise can find out transcription factor activity for all the transcription factors without or with uncertainty of expression level respectively

```
> rm(list=ls())
> load("../ResultsTu_Final.RData")
> source("../chipDynoLoadModules.R")
> nTrans=nrow(TransNames);
> lst=list();
> newX=array(0, dim <-c(dim(X)));
> newXVals=array(0, dim <-c(dim(X)));
> i=1
> expectations =chipDynoTransFactNoise(data,X,Sigma,beta,precs, gamma, mu,
+                                      TransNames, annotation, TransNames[i,]);
> expectations[[1]][1:5,1:5] # Transcription Factor activity

         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 3.808254 3.819266 3.580342 3.530811 3.639116
[2,] 7.992771 9.052932 8.876395 8.822699 9.228027
[3,] 6.278216 6.262015 5.851471 5.898305 6.425768
[4,] 9.367163 8.709554 7.501238 7.325142 7.652420
[5,] 5.749316 5.776974 5.347983 5.821693 6.240452

> expectations[[2]][1:5,1:5] # Transcription Factor activity error

           [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 3.28911886 3.2889320 3.2907785 3.2907267 3.2914427
[2,] 0.19198452 0.1224492 0.1338201 0.1347550 0.1192488
[3,] 0.23749561 0.2267321 0.2941456 0.2797092 0.2359230
[4,] 0.09053301 0.1132712 0.1933534 0.2011850 0.1860413
[5,] 3.69168251 3.6916071 3.6926496 3.6915638 3.6912072

> expectations[[3]][1:5,1:5,1] # Transcription Factor activity error difference

          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 1.0000000 0.4844291 0.6761753 0.8177611 0.9372513
[2,] 0.4844291 1.0000000 0.4897352 0.6755841 0.8188247
[3,] 0.6761753 0.4897352 1.0000000 0.4951315 0.6817655
[4,] 0.8177611 0.6755841 0.4951315 1.0000000 0.4971175
[5,] 0.9372513 0.8188247 0.6817655 0.4971175 1.0000000
```

12 Given Gene: lists activators in decreasing order

For a given gene we can find out the list of activators using the function chipDynoGeneActNoise. It will list all the activators in decreasing order with transcription factor activity and activity error.

```
> rm(list=ls())
> load("../ResultsTu_Final.RData")
> source("../chipDynoLoadModules.R")
> geneName="YHR143W";
> #geneName="AGA1";
```

```
> transNames=TransNames
> activators = chipDynoGeneActNoise(data, X, Sigma, beta, precs, gamma,mu,
+                     transNames, annotation, geneName)
> activators[[1]] # List of activators

[1] "ACE2" "FKH2" "FKH1" "MBP1"

> activators[[2]] # Maximum transcription factor activity

[1] 0.7841517 0.7258846 0.4750492 0.1858749

> activators[[3]] # Maximum transcription factor activity error

[1] 2.758191 3.198300 2.983093 3.743103
```

13 Session info

Here is the output of `sessionInfo` on the system on which this document was compiled:

```
> toLatex(sessionInfo())
```

- R version 3.0.2 (2013-09-25), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_GB.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_GB.UTF-8`, `LC_COLLATE=en_GB.UTF-8`, `LC_MONETARY=en_GB.UTF-8`, `LC_MESSAGES=en_GB.UTF-8`, `LC_PAPER=en_GB.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_GB.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: Matrix 1.1-2
- Loaded via a namespace (and not attached): BiocStyle 0.0.19, grid 3.0.2, lattice 0.20-24, tools 3.0.2

References

[Lee et al., 2002] Lee, T. I., Rinaldi, N. J., Robert, F., Odom, D. T., Bar-Joseph, Z., Gerber, G. K., Hannett, N. M., Harbison, C. T., Thompson, C. M., Simon, I., Zeitlinger, J., Jennings, E. G., Murray, H. L., Gordon, D. B., Ren, B., Wyrick, J. J., Tagne, J.-B., Volkert, T. L., Fraenkel, E., Gifford, D. K., and Young, R. A. (2002). Transcriptional regulatory networks in saccharomyces cerevisiae. *Science*, 298(14):799–804.

[Nabney, 2002] Nabney, I. T. (2002). Netlab: Algorithms for pattern recognition. *Springer*. London.

[Pearson et al., 2009] Pearson, R., Liu, X., Sanguinetti, G., Milo, M., Lawrence, N., and Rattray, M. (2009). puma: a bioconductor package for propagating uncertainty in microarray analysis. *BMC Bioinformatics*. 10:211.