tigre User Guide

Antti Honkela, Pei Gao, Jonatan Ropponen,
Magnus Rattray, and Neil D. Lawrence

September 20, 2010

1 Abstract

The *tigre* package implements our methodology of Gaussian process differential equation models for analysis of gene expression time series from single input motif networks. The package can be used for inferring unobserved transcription factor (TF) protein concentrations from expression measurements of known target genes, or for ranking candidate targets of a TF.

2 Citing *tigre*

The *tigre* package is based on a body of methodological research. Citing *tigre* in publications will usually involve citing one or more of the methodology papers (Lawrence et al., 2007; Gao et al., 2008; Honkela et al., 2010) that the software is based on as well as citing the software package itself.

3 Introductory example analysis - Drosophila development

In this section we introduce the main functions of the *puma* package by repeating some of the analysis from the PNAS paper (Honkela et al., 2010)[1].

3.1 Installing the *tigre* package

The recommended way to install *tigre* is to use the `biocLite` function available from the bioconductor website. Installing in this way should ensure that all appropriate dependencies are met.

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("tigre")
```

To load the package start R and run

```
> library(tigre)
```

[1] Note that the results reported in the paper were run using an earlier version of this package for MATLAB, so there can be minor differences.

3.2 Loading the data

To get started, you need some preprocessed time series expression data. If the data originates from Affymetrix arrays, we highly recommend processing it with `mmgmos` from the *puma* package. This processing extracts error bars on the expression measurements directly from the array data to allow judging the reliability of individual measurements. This information is directly utilised by all the models in this package.

To start from scratch on Affymetrix data, the .CEL files from `ftp://ftp.fruitfly.org/pub/embryo_tc_array_data/` may be processed using:

```
> # Names of CEL files
> expfiles <- c(paste("embryo_tc_4_", 1:12, ".CEL", sep=""),
+               paste("embryo_tc_6_", 1:12, ".CEL", sep=""),
+               paste("embryo_tc_8_", 1:12, ".CEL", sep=""))
> # Load the CEL files
> expdata <- ReadAffy(filenames=expfiles,
+                     celfile.path="embryo_tc_array_data")
> # Setup experimental data (observation times)
> pData(expdata) <- data.frame("time.h" = rep(1:12, 3),
+                              row.names=rownames(pData(expdata)))
> # Run mmgMOS processing (requires several minutes to complete)
> drosophila_mmgmos_exprs <- mmgmos(expdata)
> drosophila_mmgmos_fragment <- drosophila_mmgmos_exprs
```

This data needs to be further processed to make it suitable for our models. This can be done using

```
> drosophila_gpsim_fragment <-
+   processData(drosophila_mmgmos_fragment,
+               experiments=rep(1:3, each=12))
```

Here the last argument specifies that we have three independent time series of measurements.

In order to save time with the demos, a part of the result of this is included in this package and can be loaded using

```
> data(drosophila_gpsim_fragment)
```

3.3 Learning individual models

Let us now recreate some the models shown in the plots of the PNAS paper (Honkela et al., 2010):

```
> # FBgn names of target genes
> targets <- c('FBgn0003486', 'FBgn0033188', 'FBgn0035257')
> # Load gene annotations
> library(annotate)
> aliasMapping <- getAnnMap("ALIAS2PROBE",
+                   annotation(drosophila_gpsim_fragment))
> # Get the probe identifier for TF 'twi'
> twi <- get('twi', env=aliasMapping)
> # Load alternative gene annotations
```

```
> fbgnMapping <- getAnnMap("FLYBASE2PROBE",
+                  annotation(drosophila_gpsim_fragment))
> # Get the probe identifiers for target genes
> targetProbes <- mget(targets, env=fbgnMapping)
> st_models <- list()
> # Learn single-target models for each gene individually
> for (i in seq(along=targetProbes)) {
+   st_models[[i]] <- GPLearn(drosophila_gpsim_fragment,
+                             TF=twi, targets=targetProbes[i],
+                             useGpdisim=TRUE, quiet=TRUE)
+ }
> # Learn a joint model for all targets
> mt_model <- GPLearn(drosophila_gpsim_fragment, TF=twi,
+                     targets=targetProbes,
+                     useGpdisim=TRUE, quiet=TRUE)
> # Display the joint model parameters
> show(mt_model)

Gaussian process driving input single input motif model:
  Number of time points:
  Driving TF: 143396_at
  Target genes (3):
    148227_at
    152715_at
    147995_at
  Basal transcription rate:
    Gene 1: 31.4901487775542
    Gene 2: 0.0077988721171123
    Gene 3: 1.48241754617025e-06
  Kernel:
    Multiple output block kernel:
    Block 1
    Normalised version of the kernel.
    RBF inverse width: 0.7718466 (length scale 1.138242)
    RBF variance: 1.754535
    Block 2
    Normalised version of the kernel
    DISIM decay: 0.07285956
    DISIM inverse width: 0.7718466 (length scale 1.138242)
    DISIM Variance: 1
    SIM decay: 2005.556
    SIM Variance: 0.001472529
    RBF Variance: 1.754535
    Block 3
    Normalised version of the kernel
    DISIM decay: 0.07285956
    DISIM inverse width: 0.7718466 (length scale 1.138242)
    DISIM Variance: 1
    SIM decay: 0.4985881
    SIM Variance: 0.03226757
```

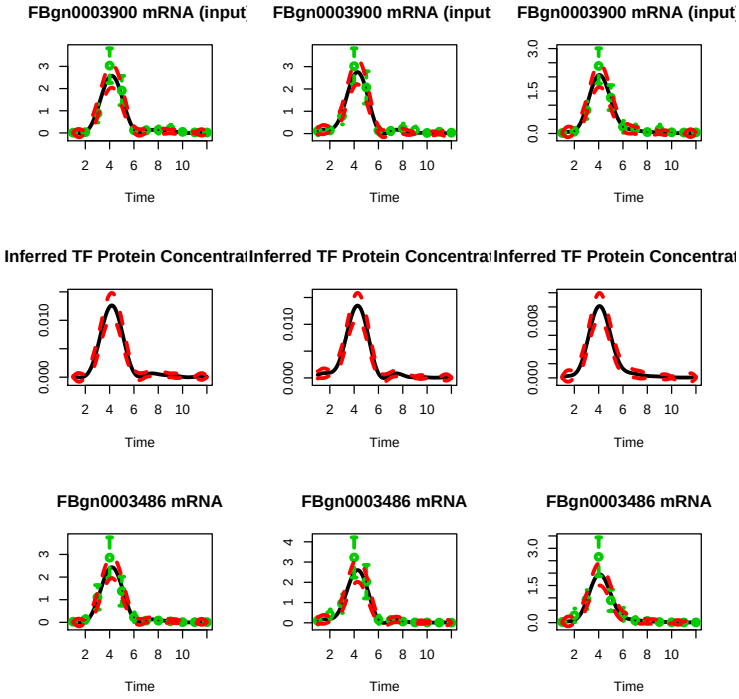

Figure 1: Single target models for the gene FBgn0003486. The models for each repeated time series are shown in different columns.

```
   RBF Variance: 1.754535
   Block 4
   Normalised version of the kernel
   DISIM decay: 0.07285956
   DISIM inverse width: 0.7718466 (length scale 1.138242)
   DISIM Variance: 1
   SIM decay: 0.0002446364
   SIM Variance: 0.003265287
   RBF Variance: 1.754535
  Log-likelihood: -31.84288
```

3.4 Visualising the models

The models can be plotted using commands like

```
> GPPlot(st_models[[1]], nameMapping=getAnnMap("FLYBASE",
+                          annotation(drosophila_gpsim_fragment)))
> GPPlot(mt_model, nameMapping=getAnnMap("FLYBASE",
+                    annotation(drosophila_gpsim_fragment)))
```

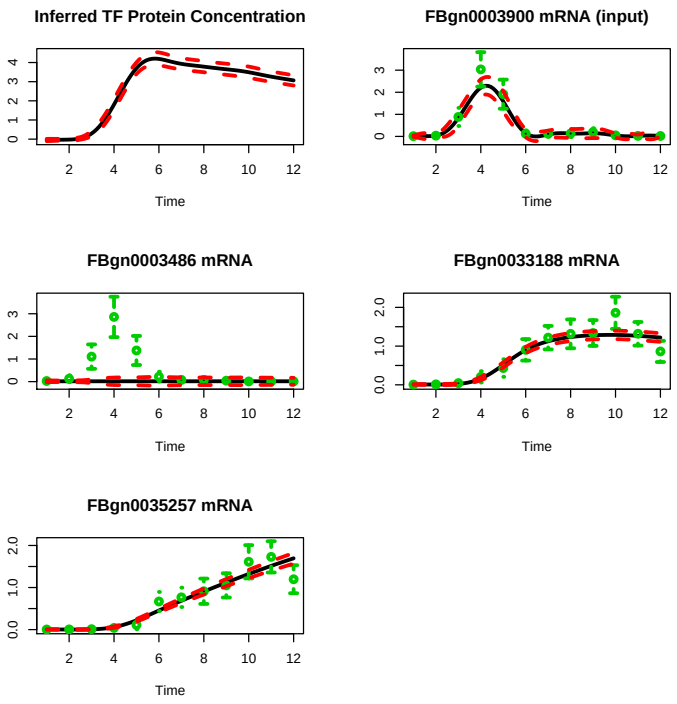

Figure 2: Multiple-target model for all the example genes. The call creates independent figures for each repeated time series.

3.5 Ranking the targets

Bulk ranking of candidate targets can be accomplished using

```
> ## Rank the targets, filtering weakly expressed genes with average
> ## expression z-score below 1.8
> scores <- GPRankTargets(drosophila_gpsim_fragment, TF=twi,
+                         testTargets=targetProbes,
+                         options=list(quiet=TRUE),
+                         filterLimit=1.8)
> ## Sort the returned list according to log-likelihood
> scores <- sort(scores, decreasing=TRUE)
> write.scores(scores)
```

```
"log-likelihood" "null_log-likelihood"
"147995_at" 6.75970270587137 -487.893231050121
"152715_at" -1.51920691642152 -539.73619673943
"148227_at" -1.52526566233162 -73.4806804255218
```

To save space, `GPRankTargets` does not return the models by default. If those are needed later e.g. for plotting, they can be recreated using the inferred parameters saved together with the ranking using

```
> topmodel <- generateModels(drosophila_gpsim_fragment,
+                            scores[1])
> show(topmodel)
```

```
[[1]]
Gaussian process driving input single input motif model:
  Number of time points:
  Driving TF: 143396_at
  Target genes (1):
    147995_at
  Basal transcription rate:
    Gene 1: 0.000135154902328578
  Kernel:
    Multiple output block kernel:
    Block 1
    Normalised version of the kernel.
    RBF inverse width: 0.760276 (length scale 1.146870)
    RBF variance: 1.804526
    Block 2
    Normalised version of the kernel
    DISIM decay: 0.01789634
    DISIM inverse width: 0.760276 (length scale 1.146870)
    DISIM Variance: 1
    SIM decay: 0.01952063
    SIM Variance: 0.002722844
    RBF Variance: 1.804526
  Log-likelihood: 6.759703
```

3.6 Ranking using known targets with multiple-target models

Ranking using known targets with multiple-target models can be accomplished simply by adding the `knownTargets` argument

```
> ## Rank the targets, filtering weakly expressed genes with average
> ## expression z-score below 1.8
> scores <- GPRankTargets(drosophila_gpsim_fragment, TF=twi,
+                         knownTargets=targetProbes[1],
+                         testTargets=targetProbes[2:3],
+                         options=list(quiet=TRUE),
+                         filterLimit=1.8)
> ## Sort the returned list according to log-likelihood
> scores <- sort(scores, decreasing=TRUE)
> write.scores(scores)

"log-likelihood" "null_log-likelihood"
"152715_at" -28.0896855288335 -539.73619673943
"147995_at" -206.238519107964 -487.893231050121
```

3.7 Running ranking in a batch environment

`GPRankTargets` can be easily run in a batch environment using the argument `scoreSaveFile`. This indicates a file to which scores are saved after processing each gene. Thus one could, for example, split the data to, say, 3 separate blocks according to the reminder after division by 3 and run each of these independently. The first for loop could then be run in parallel (e.g. as separate jobs on a cluster), as each step is independent of the others. After these have all completed, the latter loop could be used to gather the results.

```
> for (i in seq(1, 3)) {
+   targetIndices <- seq(i,
+     length(featureNames(drosophila_gpsim_fragment)), by=3)
+   outfile <- paste('ranking_results_', i, '.Rdata', sep='')
+   scores <- GPrankTargets(preprocData, TF=twi,
+                           testTargets=targetIndices,
+                           scoreSaveFile=outfile)
+ }
> for (i in seq(1, 3)) {
+   outfile <- paste('ranking_results_', i, '.Rdata', sep='')
+   load(outfile)
+   if (i==1)
+     scores <- scoreList
+   else
+     scores <- c(scores, scoreList)
+ }
> show(scores)
```

4 Experimental feature: Using non-Affymetrix data

Using non-Affymetrix data, or data without associated uncertainty information for the expression data in general, requires more because of two reasons

- noise variances need to be estimated together with other model parameters; and
- weakly expressed genes cannot be easily filtered *a priori*.

The first of these is automatically taken care of by all the above functions, but the latter requires some extra steps after fitting the models.

In order to get started, you need to create an `ExpressionTimeSeries` object of your data set. This can be accomplished with the function

```
> procData <- processRawData(data, times=c(...),
+                            experiments=c(...))
```

Filtering of weakly expressed genes requires more care and visualising the fitted models is highly recommended to avoid mistakes.

Based on initial experiments, it seems possible to perform the filtering based on the statistic `loglikelihoods(scores) - baseloglikelihoods(scores)`, but selection of suitable threshold is highly dataset specific.

References

Pei Gao, Antti Honkela, Magnus Rattray, and Neil D Lawrence. Gaussian process modelling of latent chemical species: applications to inferring transcription factor activities. *Bioinformatics*, 24(16):i70–i75, Aug 2008. doi: 10.1093/bioinformatics/btn278. URL `http://dx.doi.org/10.1093/bioinformatics/btn278`.

Antti Honkela, Charles Girardot, E. Hilary Gustafson, Ya-Hsin Liu, Eileen E M Furlong, Neil D Lawrence, and Magnus Rattray. Model-based method for transcription factor target identification with limited data. *Proc Natl Acad Sci U S A*, Apr 2010. doi: 10.1073/pnas.0914285107. URL `http://dx.doi.org/10.1073/pnas.0914285107`.

Neil D. Lawrence, Guido Sanguinetti, and Magnus Rattray. Modelling transcriptional regulation using Gaussian processes. In B. Schölkopf, J. C. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems*, volume 19, pages 785–792. MIT Press, Cambridge, MA, 2007.