Oscope: a statistical pipeline for identifying oscillatory genes in unsynchronized single cell RNA-seq experiments

Ning Leng and Christina Kendziorski

October 17, 2016

Contents

1 Introduction

Oscope (as detailed in Leng* *et al.*, 2015 **?**) is a statistical pipeline for identifying oscillatory genes in unsynchronized single cell RNA-seq (scRNA-seq) experiments. Oscope capitalizes on the fact that cells from an unsynchronized population represent distinct states in a system. Oscope utilizes co-regulation information among oscillators to identify groups of putative oscillating genes, and then reconstructs the cyclic order of samples for each group, defined as the order that specifies each sample's position within one cycle of the oscillation, referred to as a base cycle. The reconstructed order is based on minimizing distance between each gene's expression and its gene-specific profile defined by the group's base cycle allowing for phase shifts between different genes. For different groups of genes following independent oscillatory processes and/or having distinct frequencies, the cyclic orders need not be the same.

The flowchart of Oscope is shown in Figure **??**. As shown, Oscope first fits a sinusoidal function to all gene pairs and choose those with significantly high sine scores. Once candidate genes are identified, K-medoids is applied to cluster genes into groups with similar frequencies, but possibly different phases. Then, for each group, Oscope recovers the cyclic order which orders cells by their position within one cycle of the oscillatory process underlying the group.

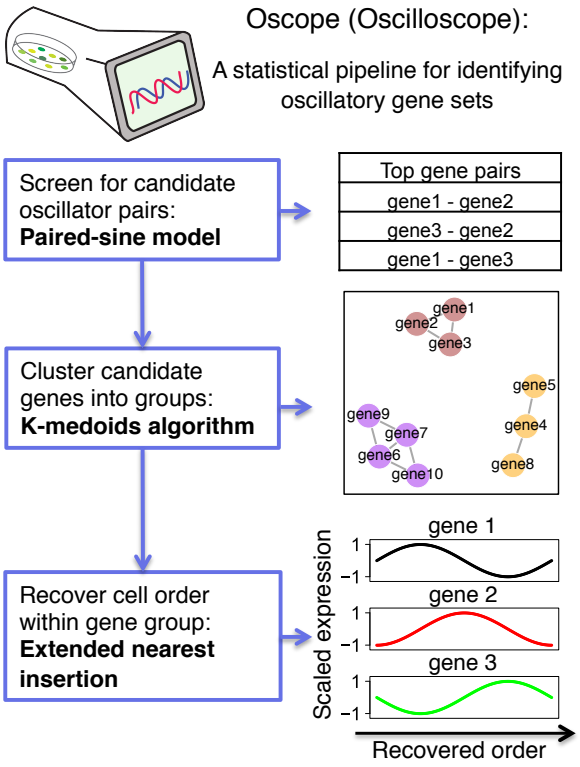

Figure 1: The Oscope flowchart.

2 Run Oscope

Before analysis can proceed, the Oscope package must be loaded into the working space:

```
> library(Oscope)
```

2.1 Required inputs

Data: The object `Data` should be a $G - by - S$ matrix containing the expression values for each gene and each sample, where G is the number of genes and S is the number of samples. These values should exhibit estimates of gene expression across samples. Counts of this nature may be obtained from RSEM (**?**), Cufflinks (**?**), or a similar approach. Cross-sample library size normalization should be performed. An cross-sample library size normalization by median normalization are shown in section **??**.

The object `OscopeExampleData` is a simulated data matrix containing 500 rows of genes and 30 columns of samples. The genes are named `g1, g2, ...` and the samples are named `S1, S2, ...` Among the 500 genes, gene g1-g120 are simulated as oscillators. Two groups of oscillators (g1-g60 and g61-g120) are simulated following independent frequencies and orders. The data set also include a gene group (g301-g305) that has purely linear correlation between genes (but not oscillating). The other genes are simulated as noise.

```
> data(OscopeExampleData)
> str(OscopeExampleData)
```

```
 num [1:500, 1:30] 982 1038 901 1524 895 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:500] "g1" "g2" "g3" "g4" ...
  ..$ : chr [1:30] "S1" "S2" "S3" "S4" ...
```

```
> set.seed(10)
```

2.2 Normalization

Oscope requires cross-sample normalization to be applied to adjust for sequencing depth differences among different samples. Here, the library size factors may be obtained via the function `MedianNorm`, which reproduces the median normalization approach in DESeq (**?**).

```
> Sizes <- MedianNorm(OscopeExampleData)
```

If quantile normalization is preferred, library size factors may be obtained via the function `QuantileNorm` (for example, `QuantileNorm(GeneMat,.75)` for Upper-Quartile Normalization in **?**).

To obtain the normalized expression matrix, user may used the GetNormalizedMat() function:

```
> DataNorm <- GetNormalizedMat(OscopeExampleData, Sizes)
```

2.3 Pre-processing

It is well-accepted that scRNA-seq suffers from high level of technical noise. It is also known that the low expressers are more affected by the noises. Therefore, we suggest users to apply Oscope on a subset of genes with high mean and high variance to reduce the effects from technical noises. Note that once the base cycle order is recovered, a user may apply ordinary time-series oscillatory gene detection algorithms based on the recovered orders to identify oscillatory genes in the genes with lower mean and variance.

Function CalcMV() may be used to calculate the estimated mean and variance of genes, as well as select genes with high mean and high variance. For example:

```
> MV <- CalcMV(Data = OscopeExampleData, Sizes = Sizes)
> str(MV$GeneToUse)
 chr [1:171] "g4" "g6" "g7" "g9" "g11" "g15" "g19" "g22" "g30" ...
> DataSubset <- DataNorm[MV$GeneToUse,]
```

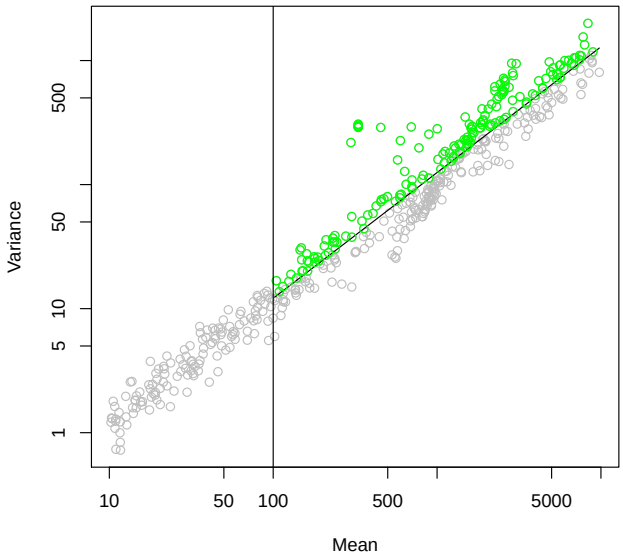

Figure 2: Mean-Variance plot generated by CalcMV() function.

Figure **??** shows the output figure of the CalcMV() function. By default, CalcMV() defines genes with mean expression greater than 100 as high expressers. To change it, a user may specify parameter `MeanCutLow` to another number. To define the high variance genes, the CalcMV() function will fit a linear regression on log(variance) $\sim$ log(mean) on genes with high mean. Genes with variance above this line are considered as genes with high mean and high variance (marked in green in Figure **??**). The upper bound of mean may be specified using `MeanCutHigh`.

While working with a normalized data set, a user may specify `Sizes = NULL` and `NormData = TRUE`. For example:

```
> MV2 <- CalcMV(Data = DataNorm, Sizes = NULL, NormData = TRUE)
> str(MV2$GeneToUse)
> DataSubset2 <- DataNorm[MV2$GeneToUse,]
```

The CalcMV() function can also take unnormalized data set. By setting `Sizes = NULL` and `NormData = FALSE`, the CalcMV() function will calculate the library size factor via MedianNorm() function first, then calculate mean and variance after adjusting for library sizes. A user can also input pre-defined library size factor for unnormalized data via parameter `Sizes`.

2.4 Rescaling

Since the paired-sine model in Oscope requires input values to be between -1 and 1, a rescaling step is needed prior to apply Oscope. Function `NormForSine` may be used for the rescaling. For example:

```
> DataInput <- NormForSine(DataNorm)
```

The NormForSine() function will rescale the expression measurements to values between -1 and 1. To reduce the influences of potential outliers, the default setting in NormForSine() function will impute the extreme values in each gene to its upper/lower bound. By default settings,a gene's upper (lower) bound is set to be its 95th (5th) quantile of expression.

These two quantile thresholds may be changed via parameters `qt1` and `qt2`. If `qt1` and `qt2` are set as 0 and 1, no outlier imputation will take place.

2.5 Oscope: paired-sine model

We developed a paired-sine model to identify gene pairs that are oscillating following the same process. Genes following the same process are assumed to have same frequency, but allow for phase shifts. To apply the paired-sine model, user may use the `OscopeSine()` function.

```
> SineRes <- OscopeSine(DataInput)
> str(SineRes)
```

The `OscopeSine` function can be parallelized by setting `parallel=TRUE` (see below). A user may change the settings, such as the number of cores, via the parameter `parallelParam`.

```
> SineRes <- OscopeSine(DataInput, parallel=TRUE)
> str(SineRes)
```

```
List of 3
 $ SimiMat : num [1:500, 1:500] 0 0.668 0.413 0.177 0.156 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:500] "g1" "g2" "g3" "g4" ...
  .. ..$ : chr [1:500] "g1" "g2" "g3" "g4" ...
 $ DiffMat : num [1:500, 1:500] 0 0.215 0.387 0.666 0.698 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:500] "g1" "g2" "g3" "g4" ...
  .. ..$ : chr [1:500] "g1" "g2" "g3" "g4" ...
 $ ShiftMat: num [1:500, 1:500] 0 0.0908 0.2083 0.2961 0.4471 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:500] "g1" "g2" "g3" "g4" ...
  .. ..$ : chr [1:500] "g1" "g2" "g3" "g4" ...
```

The output of `OscopeSine()` contains 3 matrices. SimiMat shows the sine score between each pair of input genes. The higher the sine score, the more likely that two genes are oscillating following the same process. DiffMat shows the distance (dissimilarity estimates) between each pair of genes. Note that sine score = -log10 (distance) for any pair of genes. ShiftMat shows the estimated phase shift between each pair of genes.

If only high mean high variance genes are of interest, a user may run the paired-sine model on the genes defined in section **??**:

```
> DataInput2 <- NormForSine(DataSubset)
> SineRes2 <- OscopeSine(DataInput2)
```

2.6 Oscope: K-medoids algorithm

Oscope incorporated a K-medoids algorithm to cluster candidate oscillatory gene pairs identified by the paired-sine model into gene groups. Function `OscopeKM()` may be used to apply the K-medoids algorithm:

```
> KMRes <- OscopeKM(SineRes, maxK = 10)
> print(KMRes)
```

```
$cluster1
[1] "g301" "g302" "g303" "g304" "g305"

$cluster2
[1] "g61" "g62" "g78" "g79" "g92" "g93"
```

```
$cluster3
 [1] "g5"  "g6"  "g15" "g16" "g21" "g22" "g24" "g25" "g29" "g30" "g32" "g33" "g53" "g54"
```

Input of `OscopeKM()` function is required to be the output of the `OscopeSine()` function. The K-mediods algorithm uses the distance matrix estimated in the paired-sine model as the dissimilarity metric. By setting `maxK = 10`, `OscopeKM()` function will search for the optimal K among 2-10 by maximizing the Silhouette distance. By default settings, the top 5% genes will be used in the K-medoids clustering. The percentage may be changed by setting parameter `quan`. We define a gene's minimal distance as the shortest distance between the gene and any other genes. The top genes are defined as those that have the shortest minimal distances. The distances may be calculated using the `OscopeSine()` function described in section **??**. If `maxK` is not specified, the maximum K will be set to the integer part of (number of top genes)/10. In this example, the optimal number of clusters is 3. The 3 clusters contain 5, 6 and 14 genes, respectively.

2.7 Flag clusters with small within-cluster sine scores and/or small within-cluster phase shifts

To infer the significance of each group, Oscope evaluates each group's sine score distribution using permuted data. For each group, Oscope permutes cell order for each gene independently. By default, Oscope only takes groups whose median sine score in original data is greater than its 90^{th} quantile in permuted data. Function `FlagCluster()` will flag groups who fail to meet this criteria.

To avoid detecting gene groups with purely linear relationship, we suggest users to only consider gene clusters with some within-group phase differences. For any pair of genes gi, gj within a group, define $\upsilon_{gi,gj} = min((\pi - \eta_{gi,gj}), \eta_{gi,gj})$, in which $\eta_{gi,gj} = \psi_{gi,gj} \mod \pi$. Oscope's default takes groups whose 90^{th} quantile of $\upsilon_{gi,gj}$'s is greater than $\pi/4$ for further order recovery. Function `FlagCluster()` could also flag gene clusters with small within-cluster phase shifts.

For example:

```
> ToRM <- FlagCluster(SineRes, KMRes, DataInput)
```

```
       0%       25%       50%       75%      100%
0.9791176 1.1432542 1.2087869 1.2774145 1.3284732
        0%        25%        50%        75%       100%
-1.1774749 -1.1307845 -1.0983760 -1.0174618 -0.9428653
        0%        25%        50%        75%       100%
-0.4779631 -0.3909184 -0.2329971  0.2340577  0.7903933
        0%        25%        50%        75%       100%
-1.1817922 -1.1400641 -1.0811097 -1.0312815 -0.9303043
         0%         25%         50%         75%        100%
-0.53898937 -0.38557307 -0.28651353  0.03803695  0.91732422
        0%        25%        50%        75%       100%
-1.1803460 -1.1037469 -1.0652287 -1.0161345 -0.9172967
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.02988 0.03512 0.03860 0.03876 0.04118 0.05105
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.07956 0.14460 1.20900 0.82760 1.44700 1.53700
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.06996 0.33790 0.77400 0.75470 1.12400 1.52000
```

```
> print(ToRM$FlagID_bysine)
```

```
integer(0)
```

```
> print(ToRM$FlagID_byphase)
```

```
[1] 1
```

```
> print(ToRM$FlagID) # all flagged clusters
```

```
[1] 1
```

The `FlagCluster()` function requires outputs of `OscopeSine()` and `OscopeKM()`. Summary statistics will be displayed in R console. In the sine score analysis part, the function will show summary statistics of the sine scores in the original data and permuted data. In the phase shift analysis part, the function will show summary statistics of the υ's within each cluster. Cluster 1 is flagged in this example because of the lack of within-cluster phase shift. This is expected since gene 301-305 are simulated as linearly correlated but are not oscillating. To exclude cluster 1 in the order recovery step:

```
> KMResUse <- KMRes[-ToRM$FlagID]
> print(KMResUse)
```

```
$cluster2
[1] "g61" "g62" "g78" "g79" "g92" "g93"

$cluster3
 [1] "g5"  "g6"  "g15" "g16" "g21" "g22" "g24" "g25" "g29" "g30" "g32" "g33" "g53" "g54"
```

2.8 Oscope: extended nearest insertion

Oscope reconstructs the base cycle order for each of the selected gene clusters. To reconstruct the base cycle orders, the `OscopeENI()` function may be used:

```
> ENIRes <- OscopeENI(KMRes = KMResUse, Data = DataInput, NCThre = 100)
> print(ENIRes)
```

The `OscopeENI` function can also be parallelized by setting `parallel=TRUE` (see below). A user may change the settings, such as the number of cores, via the parameter `parallelParam`.

```
> ENIRes <- OscopeENI(KMRes = KMResUse, Data = DataInput, NCThre = 100, parallel=TRUE)
```

The `OscopeENI()` requires gene lists and rescaled expression matrix as inputs. The `OscopeENI()` function will perform the extended nearest algorithm and the 2-opt algorithm. Parameter `NCThre` may be used to define the iteration stopping criteria of the 2-opt algorithm. Here we set `NCThre = 100` for demonstration purpose. By setting `NCThre = 100`, The 2-opt algorithm will be stopped when there are no updates for 100 iterations. The default setting of `NCThre` is 1000.

Once the recovered cell orders are obtained, a user may reorder the expression matrix and apply ordinary time series methods (e.g. FFT) on all the genes to find weaker oscillators (and oscillators with lower mean or variance, if only high mean high vanriance genes are used in previous steps). For example, the reordered data set may be obtained by:

```
> DataNorm2 <- DataNorm[,ENIRes[["cluster2"]]]
```

To visualize the recovered base cycle profiles of 6 genes in cluster 2:

```
> par(mfrow = c(3,2))
> for(i in 1:6)
+ plot(DataNorm[KMResUse[["cluster2"]][i], ENIRes[["cluster2"]]],
+ xlab = "Recovered order", ylab = "Expression",
+ main = KMResUse[["cluster2"]][i])
```

To visualize the recovered base cycle profiles of 6 genes in cluster 3:

```
> par(mfrow = c(3,2))
> for(i in 1:6)
+ plot(DataNorm[KMResUse[["cluster3"]][i], ENIRes[["cluster3"]]],
+ xlab = "Recovered order", ylab = "Expression",
+ main = KMResUse[["cluster3"]][i])
```

3 Session info

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

```
> print(sessionInfo())
```

```
R version 3.3.1 (2016-06-21)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.9.5 (Mavericks)

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] Oscope_1.4.0        BiocParallel_1.8.0  cluster_2.0.5       EBSeq_1.14.0
[5] testthat_1.0.2      gplots_3.0.1        blockmodeling_0.1.8

loaded via a namespace (and not attached):
 [1] snow_0.4-2          gtools_3.5.0        crayon_1.3.2        bitops_1.0-6
 [5] R6_2.2.0            magrittr_1.5        KernSmooth_2.23-15 gdata_2.17.0
 [9] BiocStyle_2.2.0     tools_3.3.1         parallel_3.3.1      caTools_1.17.1
```