SINGLE Package Vignette

Ricardo Pio Monti, Christoforos Anagnostopoulos and Giovanni Montana

December 8, 2013

Given multiple longitudinal time series we are often interested in quantifying the relationship between the time series over time. For example, given fMRI data corresponding to various regions of interest in the brain we may be interested in measuring their statistical dependencies over time. These relationships can subsequently be summarised as graphs or networks where each node corresponds to a time series (e.g., a BOLD time series for a region of interest) and edges between nodes (or their absence) provide information regarding the nature of the relationship. Here we present a short tutorial and introduction to using the `R` package `SINGLE` to estimate such dynamic graphs over time from noisy time series data. The `SINGLE` package provides an implementation of the Smooth Incremental Graphical Lasso Estimation algorithm, full details of which can be found in [Monti et al., 2013].

1 Introduction

There is an increasing interest in summarising the relationships between time series using undirected graphs. Here each node represents a time series (e.g., this can be the BOLD time series for a brain region) and the edge structure serves to summarise the statistical relationship between nodes. Edge structure is commonly estimated using partial correlations. In this case, the absence of an edge between two nodes implies that the two nodes are conditionally independent given all other nodes.

However it is often the case that only a global graph is estimated using the entire time series. Whilst this may be appropriate in some scenarios, it is often the case that we expect the statistical dependencies between nodes to change over time. A clear example of this can be seen by considering fMRI time series of a subject performing alternating tasks: we naturally expect the relationship between brain regions to change depending on task.

In this vignette, we introduce the `SINGLE` package which can be used to estimate dynamic graphs from noisy time series data. The remainder of this vignette is organised as follows: in Section 2 we give a brief motivating example to show the capabilities of the SINGLE algorithm. In Section 3 we give a brief background description of the SINGLE algorithm. Finally, in Section 4 we give a more detailed description of each of the functions in the `SINGLE` package.

2 Motivating example

Here we present a brief motivational example to show the capabilities of the SINGLE algorithm and the functionality of the `SINGLE` package.

First we simulate non-stationary data in order to test the performance of the SINGLE algorithm. We simulate the data using the `generate_random_data` funciton:

```
> library('SINGLE')
> set.seed(1)
> sim = generate_random_data(ROI=5, length_=50, seg=3, sparsity=.1, str=-.6)
```

Here `sim` is a list of length two. The first entry contains a matrix of simulated non-stationary data and the second contains an array which summaries the true correlation structure over time. Thus `sim$data` is a matrix with 5 columns and 150 rows. For example, this could correspond to fMRI data from 5 regions of interest over a time. As mentioned previously each column represents a node in our graph and each row is a chronologically ordered observation. Thus `sim$data[i,j]` is the ith observation of the jth node.

The true partial correlation structure over time is stored in the array `sim$true_cov`. We also note that this dataset contains two changepoints, one at the 50th observation and the other at the 100th. We can view the correlation structure for at the first observations as follows:

```
> sim$true_cov[,,1]

     [,1] [,2] [,3] [,4] [,5]
[1,]  1.0    0    0    0 -0.6
[2,]  0.0    1    0    0  0.0
```

```
[3,]  0.0    0    1    0  0.0
[4,]  0.0    0    0    1  0.0
[5,] -0.6    0    0    0  1.0
```

We can now run the Smooth Incremental Graphical Lasso Estimation (SINGLE) algorithm to estimate the network structure over time. The SINGLE algorithm requires the input of three intuitive parameters. Parameters λ_1 and λ_2 control the level of sparsity and temporal homogeneity respectively. They can be estimated by minimising AIC over a given range of values. The final parameter is h, the choice of width for the Gaussian kernel. This can be estimated using the `choose_h` function which calculates the look-ahead log-likelihood.

```
> data = sim$data
> h_G = choose_h(data=data, sample_size=30, kernel="gaussian",
+                h_lower=20, h_upper=100, h_step=10)
> h_W = choose_h(data=data, sample_size=30, kernel="window",
+                h_lower=20, h_upper=100, h_step=10)
```

We note that the choice of `kernel` allows the user to alternative between a Gaussian kernel and a sliding window.

The SINGLE algorithm can then be implemented in one of two ways. Firstly, we can simply input the data as well as the estimated width together with λ_1 and λ_2 parameters as follows:

```
> S = SINGLE(data=data, h=h_G, l1=.75, l2=0.5)
```

This automatically uses a Gaussian kernel to estimate sample covariance matrices. Alternatively, we can directly supply the `SINGLE` function with an array `C` of sample covariance matrices. This allows us to run the SINGLE algorithm on any arbitrary choice of sample covariance matrices. For example, we could use a sliding window to estimate covariance matrices. This can be achieved using the `get_kern_cov` function as follows:

```
> C_slid = get_kern_cov(data=data, h=h_W, kernel="window")
> S_slid = SINGLE(data=data, C=C_slid, l1=.75, l2=0.5)

Sample covariance matrices provided
```

This gives the `SINGLE` function greater flexibility and allows the user to easily compare performance across difference widths, h, as well as using different kernels. Of course the values of λ_1 and λ_2 would need to be estimated for each choice of h and kernel. The `SINGLE` function also returns the AIC and BIC for specified parameters λ_1 and λ_2.

We can see that the estimated partial correlations accurately reflect the true network structure (stored in `sim$true_cov`). We also note that the estimated partial correlation is lower than 0.6, this is due to the presence of the sparsity (λ_1) penalty. Given this is a simulated example we can assess the performance of the SINGLE algorithm using precision, recall and F scores over time using the `precision_recall` function:

```
> plotSINGLE(object=S, index=c(1,2,3,4,5), x.axis = seq(1,150), n.row=2,
+            col.names=seq(1,5), fix.axis=TRUE)
```

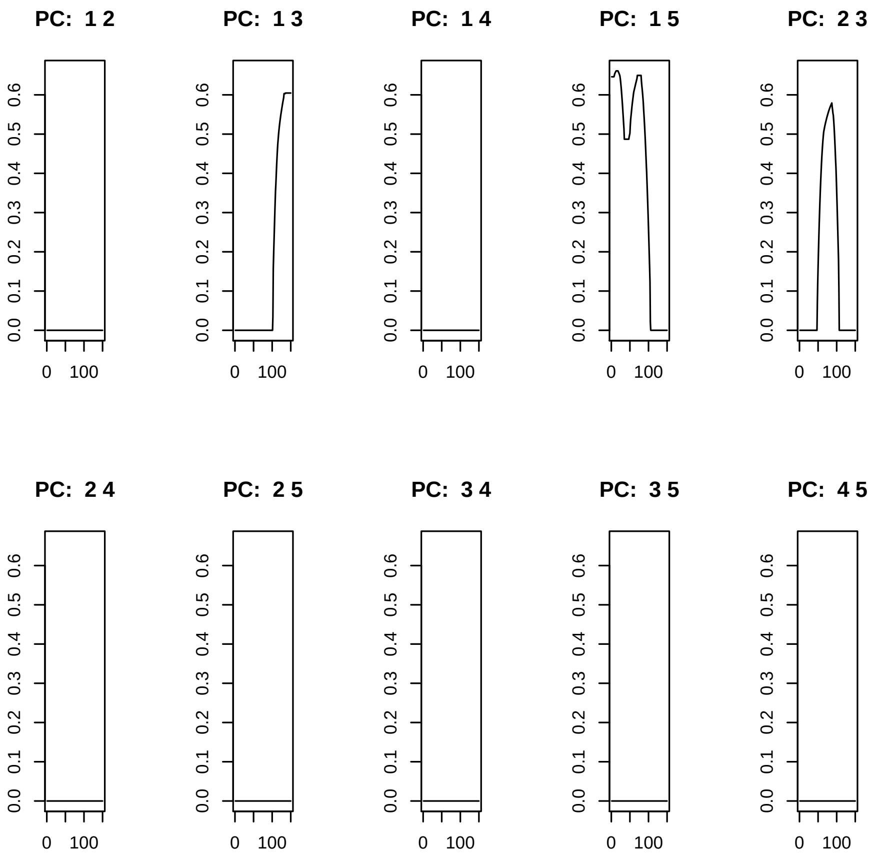

Figure 1: Estimated partial correlations for simulated example

3 Background

We assume to have obtained time series denoted by $X_1, \ldots, X_T$, where each vector $X_i \in \mathbb{R}^{1\times p}$ contains the measurements for each of the p variables of interest at the ith time point. We are interested in inferring a sequence of graphs $\{G_1, G_2, \ldots, G_T\}$ where each $G_i = (V, E_i)$ corresponds to the functional connectivity between nodes, V, at time i. The edge structure, E, is determined using partial correlations.

The `SINGLE` algorithm has the following two desirable properties: (a) each graph G_i is sparse thus allowing for accurate and interpretable estimates of the underlying graphs, and (b) the structure of the estimates graphs varies smoothly over time[1].

At any given time point, we assume that the random vector X_i follows a multivariate Gaussian distribution, however both the mean and the covariance of this distribution are assumed to be dependent on the time index. We write S_i to denote the estimated covariance

[1]we note that the extent of sparsity and smoothness can be determined by the user

```
> result = precision_recall(true_cov=sim$true_cov, estimated_cov=S$P_)
> plot(result$F1, type='l', ylim=c(0,1), ylab='',
+      main='F Score', xlab='Time')
```

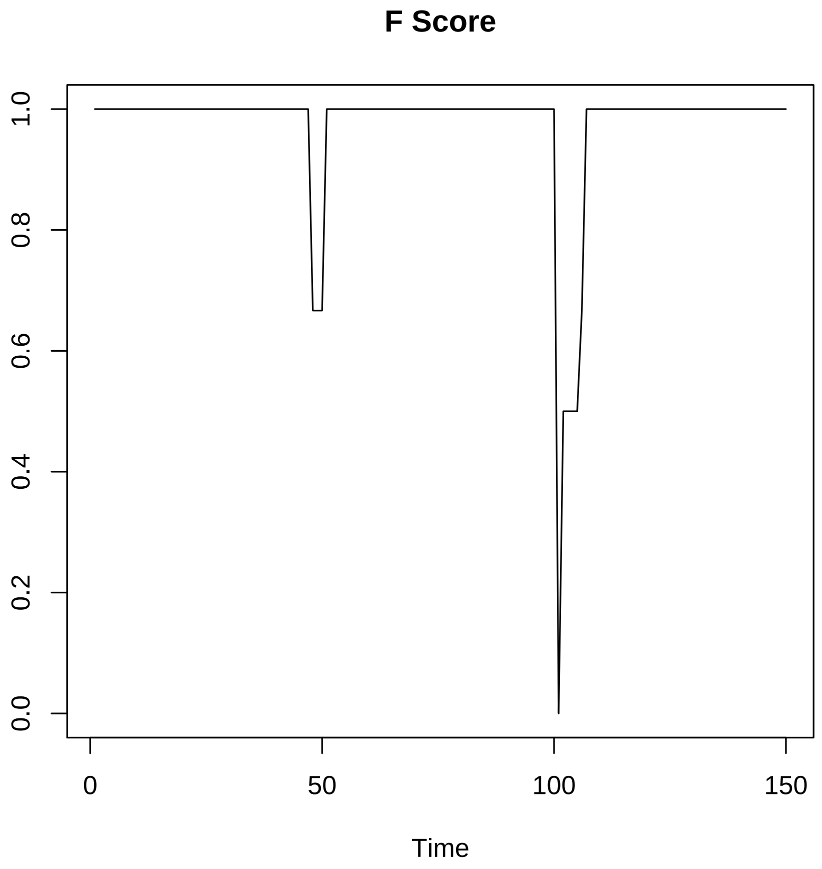

Figure 2: Plot of the F score over time for the SINGLE algorithm on simulated data

matrix at time i.

The set of partial correlations is summarised in the precision (inverse covariance) matrix. Thus our objective is equivalent to obtaining a sequence of time-dependent estimates of precision matrices, $\hat{\Theta}_1, \ldots, \hat{\Theta}_T$, where each $\hat{\Theta}_i = S_i^{-1}$. The Smooth Incremental Graphical Lasso (SINGLE) obtains these estimates by solving a constrained optimisation problem which balances goodness-of-fit, sparsity and temporal smoothness. This is achieved by formulating the following objective function:

$$f(\{\hat{\Theta}\}) = \sum_{i=1}^{T} \left[-\log \det \hat{\Theta}_i + \text{trace } (S_i \hat{\Theta}_i) \right] + \lambda_1 \sum_{i=1}^{T} ||\hat{\Theta}_i||_1 + \lambda_2 \sum_{|i-j|<k} ||\hat{\Theta}_i - \hat{\Theta}_j||_1, \qquad (1)$$

where $\{\hat{\Theta}\} = \{\hat{\Theta}_1, \ldots, \hat{\Theta}_T\}$ contains all the precision matrices indexed by time. By taking a closer look at the objective function we can gain a clear understanding of what the SINGLE algorithm is looking to achieve. The first sum is proportional to the sum of negative log likelihoods of the estimated precision matrices. The first penalty term regularised by λ_1 corresponds to the Graphical Lasso penalty and ensures sparsity in the estimated graphical structure by imposing a penalty on the sum of absolute entries in each $\hat{\Theta}_i$. On the other hand it, the second penalty function regularised by λ_2, ensures smoothness by penalising the differences between temporally adjacent networks. This penalty can be seen as an extension of the Fused Lasso penalty from the context of penalised regression (i.e., in the Fused Lasso we penalise $|\beta_i - \beta_{i+1}|$ and here this is extended to the difference over graphs). It is the choice of parameter k that governs the extent over which we expect there to be smoothness in the functional connectivity structure over time. It follows that a large choice of k will penalise changes over a longer period of time and thus result in *smoother* estimates.

The SINGLE estimation procedure consists of two independent steps performed in sequence: initially, recursive estimates of the covariance matrices $S_1, \ldots, S_T$ are obtained using a Gaussian kernel; then an iterative optimisation algorithm is run until convergence in order to produce a sequence of estimated graphs. Full details of the SINGLE algorithm are given in [Monti et al., 2013].

4 The `SINGLE` package

In this section we give a more detailed description of each of the functions contained in the `SINGLE` package.

4.1 `generate_random_data`

This function allows for the generatation of data with a random correlation structure that is piecewise continuous. This is achieved using Vector Autoregressive Processes (VAR). The choice of VAR processes here is motivated by their ability to encode autocorrelation within a time series as well as cross-correlations across time series. Network structure can be simulated according to either Erdos-Renyi random graphs or Barabasi-Albert scale-free networks. This choice is specified by `mode` input in the `generate_random_data` funciton.

First a random correlation structure is simulated. This is then used to simulate a VAR process with the given correlation structure.

As shown in the motivational example, we can easily simulate data using this function:

```
> sim_ER = generate_random_data(ROI=5, length_=50, mode='ER', seg=3, sparsity=.1)
> sim_BA = generate_random_data(ROI=5, length_=50, mode='BA', seg=3, sparsity=.1)
```

We note that while the number of piecewise continuous segments of data and their lengths can be specified using `seg` and `length_` respectively. Random data with segments of different lengths can be generated using the `generate_random_data` command several times as follows:

```
> sim1 = generate_random_data(ROI=5, length_=25, mode='BA', seg=1, sparsity=.15)
> sim2 = generate_random_data(ROI=5, length_=75, mode='BA', seg=1, sparsity=.15)
> data = rbind(sim1$data, sim2$data)
```

This function depends on both the `igraph` [Csardi and Nepusz, 2006] and `dse` packages [Gilbert, 2006] for simulating random networks and VAR processes respectively.

4.2 `SINGLE`

The `SINGLE` function is the main function within the `SINGLE` package and provides an implementation of the SINGLE algorithm. We note that this function requires the input of the following user-specified parameters:

1. `h`: This is the width, h, used in the Gaussian kernel. The choice of this radius is particularly important. Setting h to be too large will result in overly smooth estimates whereas choosing h to be too small will result in noisy estimation. The optimal choice of h can be estimated using the `choose_h` function.

2. `l1`: This is value of λ_1 in the SINGLE objective function. Increasing the value of λ_1 will increase the sparsity of the estimated dynamic networks.

3. `l2`: This is the value of λ_2 in the SINGLE objective. Increasing the value of λ_2 will encourage sparsity in the difference of temporally adjacent networks, thus resulting in *smoother* estimates.

4. `C`: This is an optional input of an array of estimated sample covariance matrices. These can be estimated in any arbitrary manner.

As shown in Section 2 the SINGLE algorithm can be run as follows:

```
> set.seed(1)
> sim = generate_random_data(ROI=5, length_=50, seg=3, sparsity=.1, str=-.6)
> data = sim$data
> S = SINGLE(data=data, h=50, l1=.75, l2=0.5)
```

Since the SINGLE algorithm relies on the Fused Lasso for one step of its iterative optimisation this function depends on the `flsa` package [Hoefling, 2013].

4.3 `precision_recall`

The `precision_recall` function allows us to measure the performance of the SINGLE algorithm. Precision is defined as the fraction of reported edges which are true edges while recall is defined as the fraction of true edges which are correctly reported. It follows that the closer precision and recall are to 1 the better the performance. Finally we also define the F score as follows:

$$F = 2 \cdot \frac{P \cdot R}{P + R},$$

where P and R refer to the precision and recall respectively.

This function calculates the precision and recall for an iteration of the SINGLE algorithm at each observation. That is, at the ith observation the precision, recall and F score is calculated by comparing the true correlation structure at the time and the correlation structure estimated by the SINGLE algorithm.

The `precision_recall` function is run directly on the array of estimated precision matrices (`S$P_` in the example above) as follows:

```
> result = precision_recall(true_cov=sim$true_cov, estimated_cov=S$P_)
```

4.4 `plotSINGLE`

The `plotSINGLE` function allows for the visualisation of the estimated correlation structures over time. This is achieved by plotting the pairwise partial correlations between each of the nodes. Since there may be a potentially large number of nodes (resulting in an even larger number of pairwise correlations to plot) the `plotSINGLE` function allows the user to specify a subset of nodes to plot using its `index` command. In the example from before we plot the partial correlations for all nodes. Alternatively, we could plot only the partial correlations between the 1st, 3rd and 5th nodes as follows:

```
> plotSINGLE(object=S, index=c(1,2,3,5), x.axis = seq(1,150), n.row=2,
+             col.names=c(1,2,3,5), fix.axis=TRUE)
```

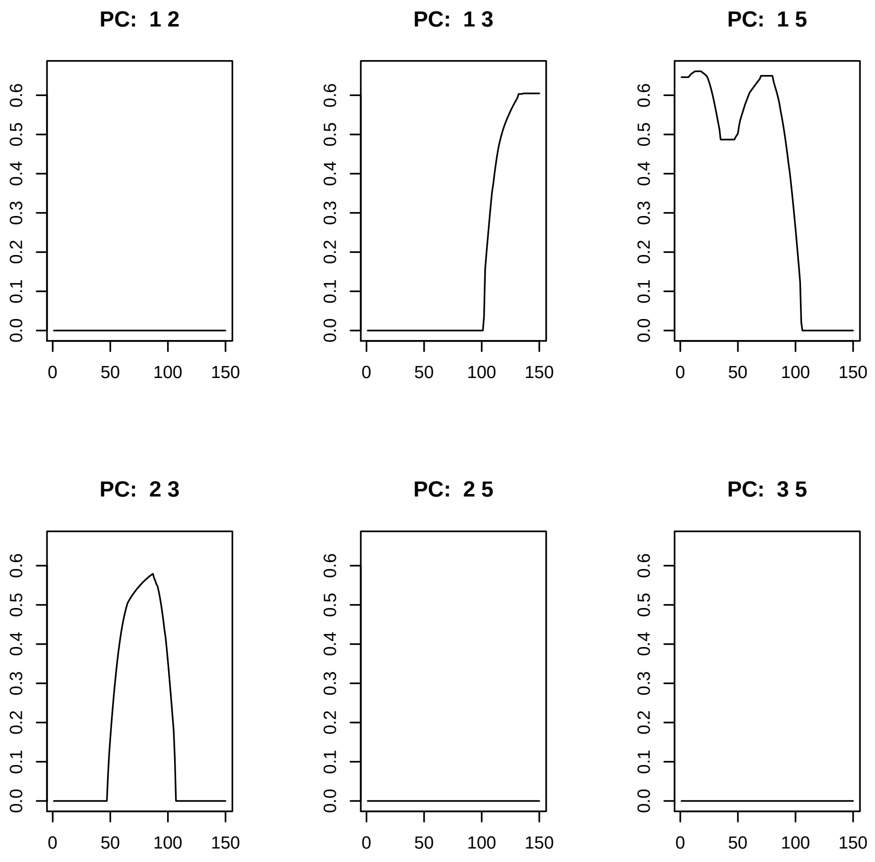

Figure 3: Estimated partial correlations for simulated example discussed previously

4.5 `choose_h`

The `choose_h` function allows the user to estimate the optimal kernel width using look-ahead log-likelihood. We define

$$\mathcal{L}_{-1}(i;h) = -\frac{1}{2} \log \det (S_{i-1}) - \frac{1}{2}(X_i - \mu_{i-1})^T S_{i-1}^{-1}(X_i - \mu_{i-1}) \tag{2}$$

as the estimated look-ahead log-likelihood for the ith observation for some fixed choice of h. We note that both μ_{i-1} and S_{i-1} are estimated with the ith observation removed. It follows that calculating $\mathcal{L}_{-1}(h)$ for all observations is computationally expensive, thus in order to save computational effort we sample a subset of observations $R \subseteq \{2, \dots, T\}$. The optimal value of h can then be estimated by choosing h to maximise the following score function:

$$CV(h) = \sum_{i \in R} \mathcal{L}_{-1}(i; h), \tag{3}$$

where in order to reduce the variability of estimated look-ahead log-likelihoods we use the same sample R to calculate each $CV(h)$.

The `sample_size` input in the `choose_h` function determines the size of subset R and the inputs `h_lower`, `h_upper` and `h_step` determine the values over which we calculate $CV(h)$.

4.6 `get_kern_cov`

The `get_kern_cov` function allows the user to estimate sample covariance matrices for a given data using either a Gaussian kernel or sliding window of any given width. The following formula is used to estimate sample covariance matrices:

$$\mu_i = \frac{\sum_{j=1}^{T} K_h(i,j) \cdot X_j}{\sum_{j=1}^{T} K_h(i,j)}, \tag{4}$$

$$S_i = \frac{\sum_{j=1}^{T} K_h(i,j) \cdot (X_j - \mu_j)^T (X_j - \mu_j)}{\sum_{j=1}^{T} K_h(i,j)}. \tag{5}$$

where $K_h(\cdot, \cdot)$ is a kernel function.

References

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL `http://igraph.sf.net`.

P. D. Gilbert. *Brief User's Guide: Dynamic Systems Estimation distributed with the dse package*, 2006. URL `http://cran.r-project.org/web/packages/dse/vignettes/dse-guide.pdf`.

H. Hoefling. *flsa: Path algorithm for the general Fused Lasso Signal Approximator*, 2013. URL `http://CRAN.R-project.org/package=flsa`. R package version 1.05.

R. P. Monti, P. Hellyer, D. Sharp, R. Leech, C. Anagnostopoulos, and G. Montana. Estimating Time-varying Brain Connectivity Networks from Functional MRI Time Series. *Preprint*, 2013.