Sample size analysis using `powerly`

Mihai A. Constantin

2023-06-08

Table of contents

Settings things up

Before we proceed, we need to install the `powerly` package and make it available in our `R` session.

> Tip
>
> You can find out more information about `powerly` at powerly.dev. Make sure to keep an eye on the website for updates and new features.

```
# Install `powerly`.
install.packages("powerly")
```

Now, we can go ahead and load the package into our session.

```
# Load `powerly`.
library(powerly)
```

> **Tip**
>
> Like always, it is a good idea to first check what packages you have installed before proceeding. Nevertheless, the command above will not reinstall packages that are already installed and up-to-date, i.e., unless we force this behavior.

Description

In this tutorial we are going to get familiar with the workflow of running a sample size analysis using `powerly`. While we will not use a time series model for the analysis (i.e., still work in progress), we use an equally interesting and challenging example, i.e., a *regularized network model*. More specifically, we will learn how to run a sample size analysis for a *Gaussian Graphical Model* (GGM) estimated using the *Least Absolute Shrinkage and Selection Operator* (LASSO; Friedman et al., 2008) and the *Extended Bayesian Information Criterion* (EBIC; Foygel & Drton, 2010). If you are curios to learn more about this methodology, you can check out the paper by Epskamp & Fried (2018).

Selecting the true model

Let's start by selecting the network model that we will use as *true model*. In this case, our true model Θ takes the form a a $p \times p$ matrix of edge weights θ_{ij}, where p is the number of nodes in the network. We can generate a true model using the `generate_model()` function provided by `powerly`.

> **Tip**
>
> Check out the documentation for `generate_model()` by running `?generate_model`, or by visiting the package website.

> **Note**
>
> We may also manually specify our true model matrix Θ. However, this is a rather complicated task, since, in this case, Θ is a matrix of *partial correlation coefficients* that represents a complex interplay between the architecture of the network and the strength of the relationships between the nodes. For this reason, it is often more convenient to generate a true model using the `generate_model()` function based on some hyperparameters, such as the number of nodes, the edge density, and the proportion of positive edges in the network.

Suppose we are interested in a GGM true model consisting of `10` nodes with an edge density of `0.4`, and a proportion of positive edges of `0.9`.

> **Note**
>
> We may also set a seed to ensure that we all use the same true model.
>
> ```
> # Set the seed.
> set.seed(20031993)
> ```

```
# To get a matrix of edge weights.
true_model <- generate_model(
    type = "ggm",
    nodes = 10,
    density = .4,
    positive = .9
)
```

Let's continue by plotting our network to get a better idea of what it looks like. For this we can use the `qgraph` package (Epskamp et al., 2012).

> **Tip**
>
> Note that you do not need to install `qgraph` since it is already a dependency of `powerly`. You can simply access its exported functions by using the `qgraph::` prefix.

```
# Plot the true model.
qgraph::qgraph(
    true_model,
    layout = "spring",
    theme = "colorblind"
)
```

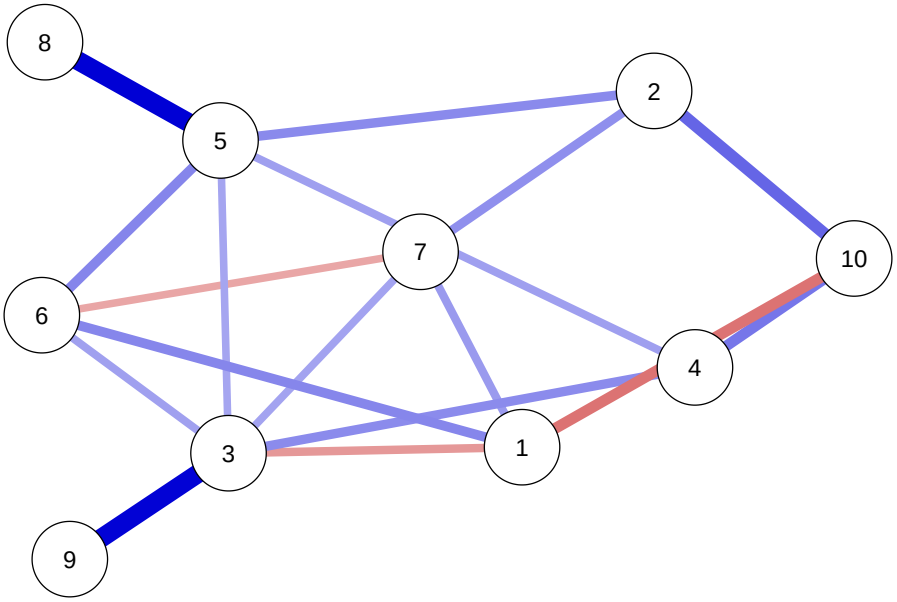

Figure 1: The GGM true model.

Specifying the performance measure and the statistic

As discussed in the lecture (i.e., `Slides > Sample size planning for intensive longitudinal designs`), the choice of performance measure should be driven by the research question. For our example, let's suppose we are interested in correctly recovering the network structure. This implies, that we may be interested in *sensitivity*, as the proportion of edges in the true network structure that were correctly estimated to be non-zero (i.e., present), which is defined as:

$$\text{SEN} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

where TP represents the true positive rate, and FN the false negatives. As for the target value of the performance measure, let's suppose we are interested in a SEN of `0.6`.

Finally, we need to specify the statistic, which is used to determined how we want to observe the performance measure. More specifically, suppose we are interested in obtaining a sample size that will allow us to detect a SEN of `0.6` with a probability of `0.8`. In other words, if we are are to collect, say 100 data sets with the optimal sample size, we want to be able to detect a SEN of `0.6` in at least `80` of them.

We can specify the performance measure and the statistic in the `powerly` function using the `measure` and `statistic` arguments, respectively. And their corresponding target values using the `measure_value` and `statistic_value` arguments.

> **Tip**
>
> Check out the documentation of the `powerly` function for a detailed description of all the arguments. You can also use the `?powerly` command in the `R` console to open the documentation.

Running the sample size analysis

At this point, we have a good idea about the true model that we want to recover, the performance measure that we want to use, and the statistic that we want to use to observe the performance measure. We can now run the sample size analysis.

```
# Run the sample size analysis.
results <- powerly(
    range_lower = 300,
    range_upper = 1000,
    samples = 30,
    replications = 60,
    measure = "sen",
    statistic = "power",
    measure_value = .6,
    statistic_value = .8,
    model = "ggm",
    model_matrix = true_model,
    cores = 9,
    verbose = TRUE
)
```

```
Method run completed (4.4468 sec):
 - converged: yes
 - iterations: 2
 - recommendation: 525
```

Check the results

To get the information about the results, we can simply use the `plot` method and indicate the method step that we want to investigate.

For Step 1 of the method.

```
# Plot Step 1.
plot(results, step = 1)
```

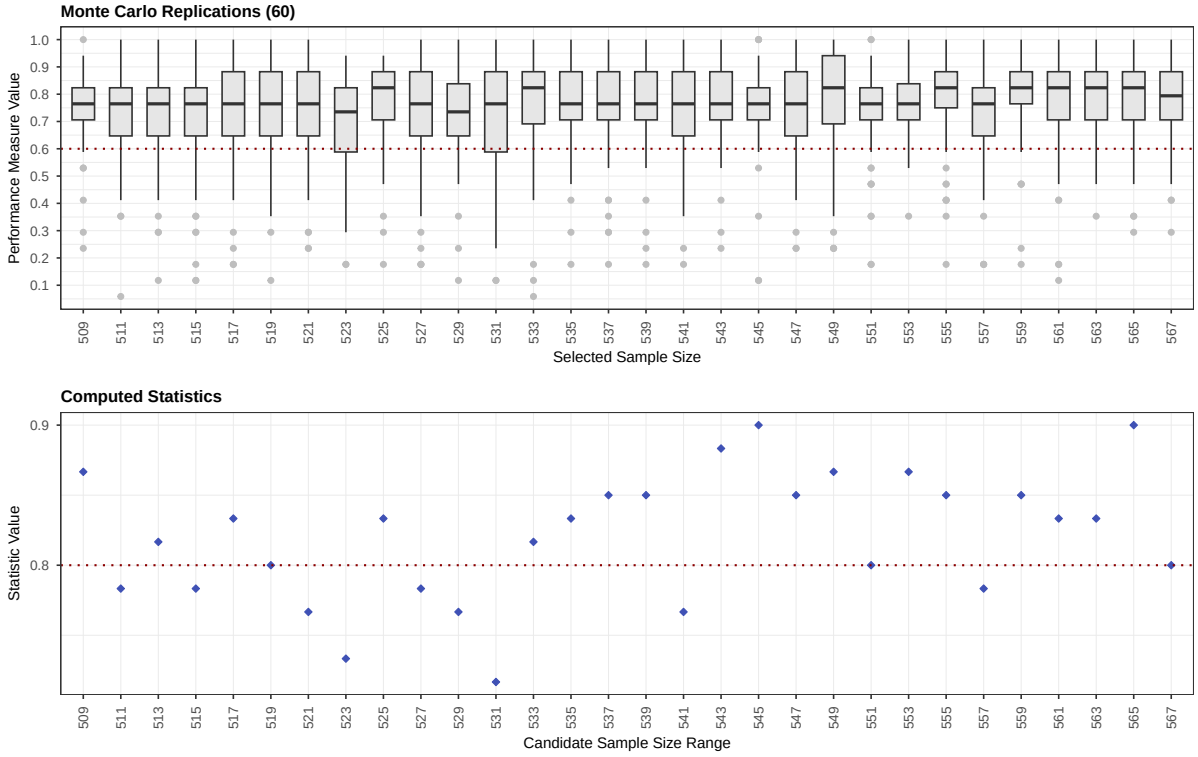

Figure 2: Step 1 results.

For Step 2 of the method.

```
# Plot Step 2.
plot(results, step = 2)
```

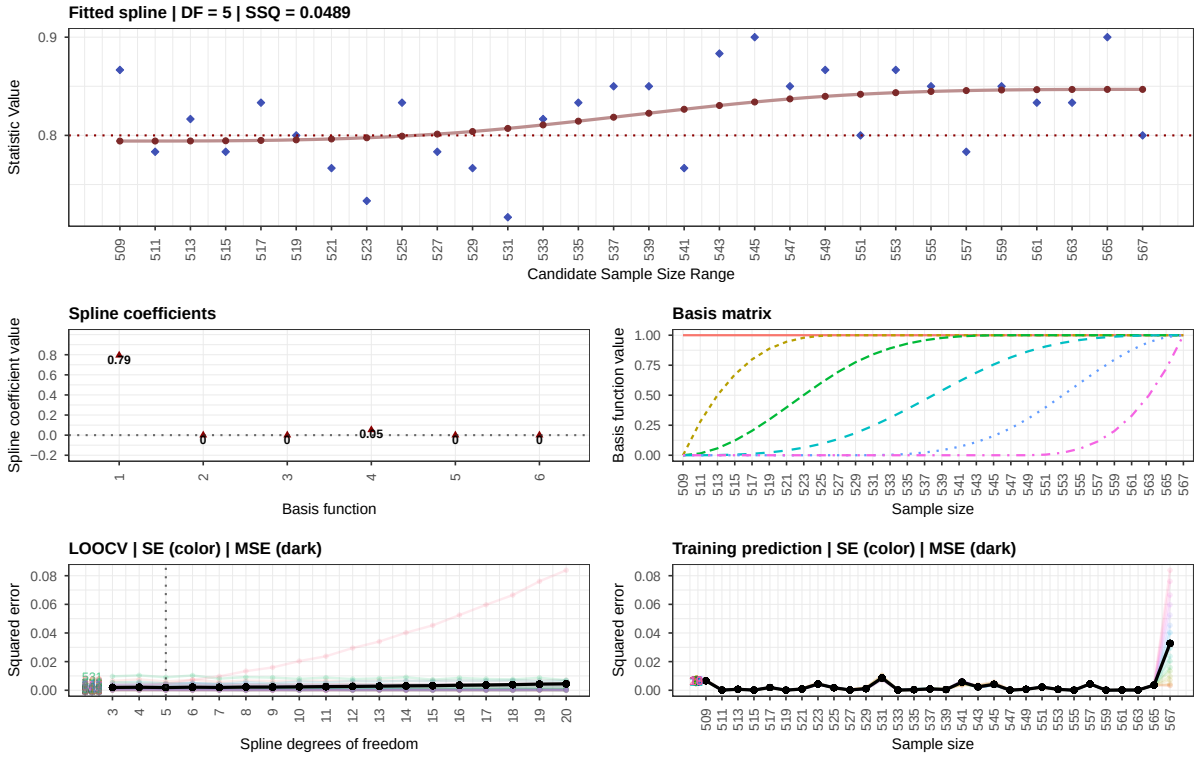

Figure 3: Step 2 results.

For Step 3 of the method.

Arguably, the most important step of the method is Step 3, which is where we can see the actual optimal sample recommendation.

```
# Plot Step 3.
plot(results, step = 3)
```

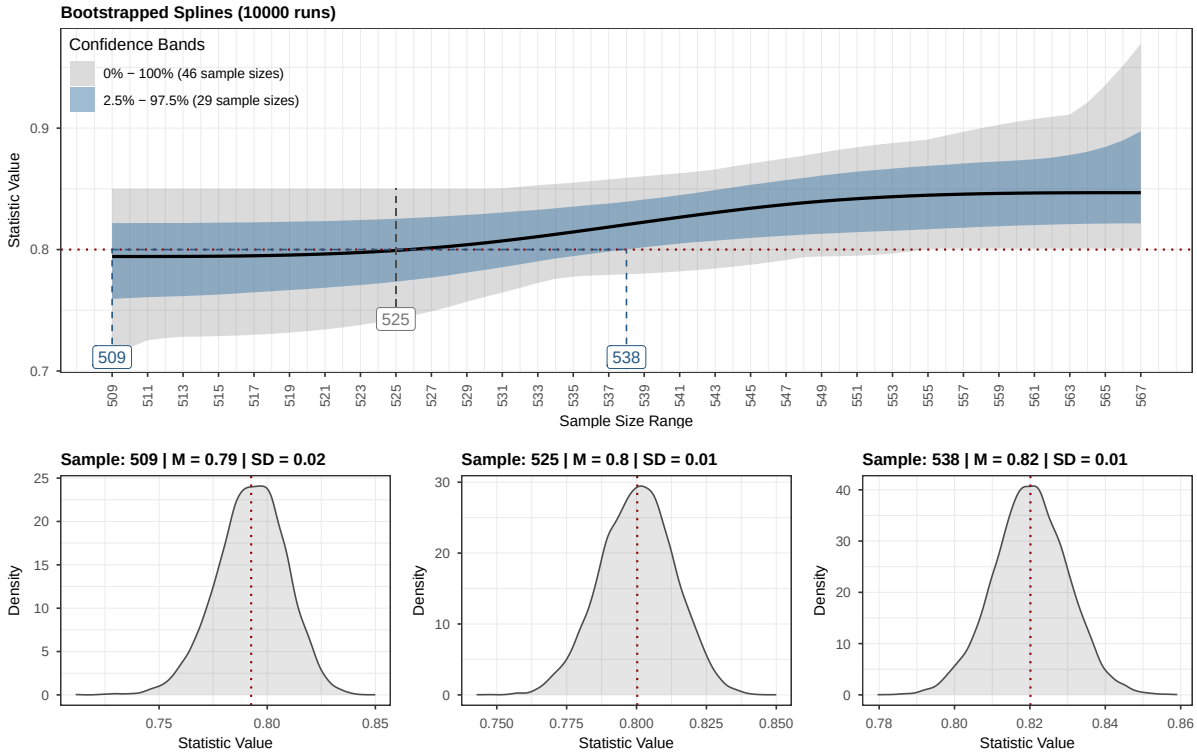

Figure 4: Step 3 results.

Additionally, we can also use the `summary` method to get a summary of the results.

```
# Get a summary of the results.
summary(results)
```

```
Method run completed (4.4468 sec):
 - converged: yes
 - iterations: 2
 - recommendation: 2.5% = 509 | 50% = 525 | 97.5% = 538
```

Validating the recommendation

It also good practice to validate the optimal sample size recommendation using the `validate` function by passing in the output obtained from the `powerly` function, followed by plotting the validation results.

```r
# Run validation.
validation <- validate(
    method = results,
    replications = 3000,
    cores = 9
)
```

```
Running the validation...

Validation completed (1.9732 sec):
 - sample: 525
 - statistic: 0.8013333
 - measure at 20th pert.: 0.647
```

```r
# Plot validation.
plot(validation)
```

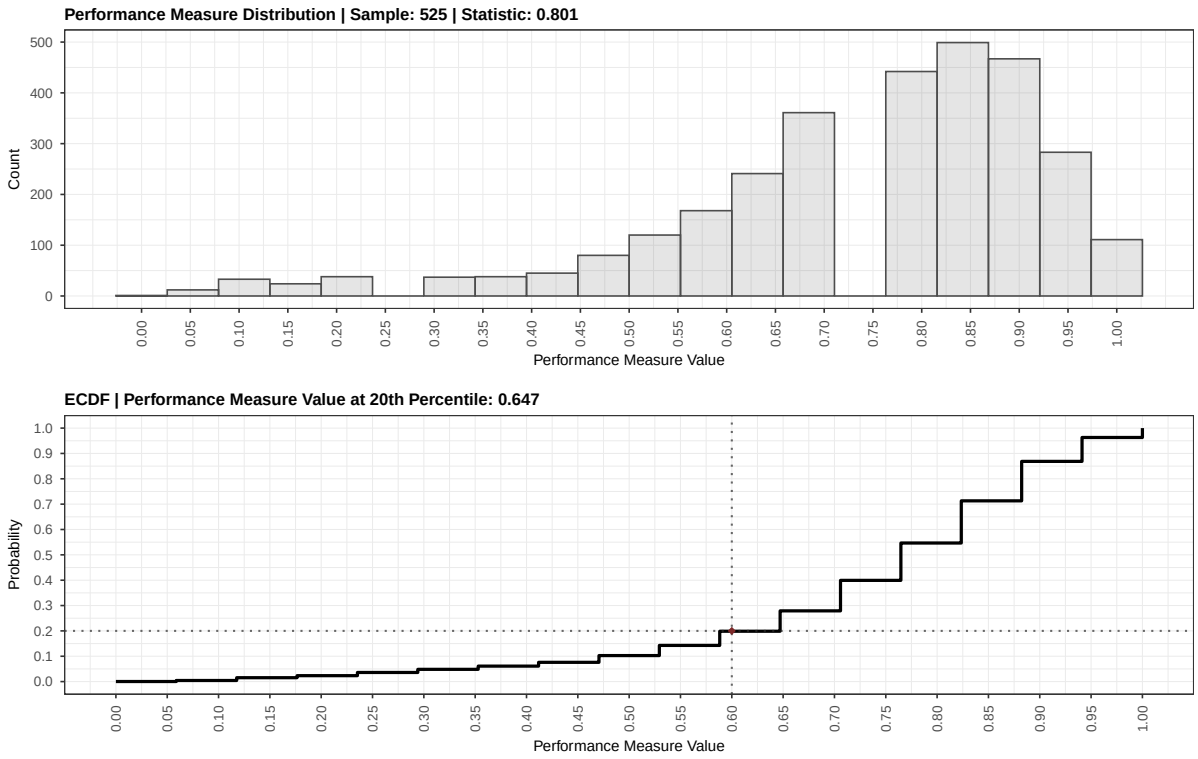

Figure 5: The validation results.

References

Epskamp, S., Cramer, A. O. J., Waldorp, L. J., Schmittmann, V. D., & Borsboom, D. (2012). Qgraph: Network Visualizations of Relationships in Psychometric Data. *Journal of Statistical Software*, *48*(4). https://doi.org/10.18637/jss.v048.i04

Epskamp, S., & Fried, E. I. (2018). A Tutorial on Regularized Partial Correlation Networks. *Psychological Methods*, *23*(4), 617–634. https://doi.org/10.1037/met0000167

Foygel, R., & Drton, M. (2010). Extended Bayesian information criteria for Gaussian graphical models. *Advances in Neural Information Processing Systems*, *23*, 604–612.

Friedman, J., Hastie, T., & Tibshirani, R. (2008). Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, *9*(3), 432–441. https://doi.org/10.1093/biostatistics/kxm045