Representing SFC models as DAGs

In this post we provide an introductory demo for a graphical tool that exists as an extension of the pk-sfc package available here developed in R. The graphical tools main function is to extract Directed Acylic Graphs (DAGs) from a Stock Flow Consistent Model. The SFC models will be generated using eviews files containing the SFC models from Monetary Economics: An Integrated Approach to Credit, Money, Income Production and Wealth by Marc Lavoie and Wynne Godley. The eviews files are available from here and required files are available for download here.

Step 1: Set-up

If you do not already have R installed please follow this link and install the appropriate version. The code below will then install load some of the prerequisite functions and code.

```
library(PKSFC)
```

```
## Loading required package: expm
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'expm'
```

```
## The following object is masked from 'package:Matrix':
##
##     expm
```

```
## Loading required package: igraph
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
## Loading required package: networkD3
```

```
library("Rgraphviz")
```

```
## Loading required package: graph
```

```
## Loading required package: BiocGenerics
```

```
## Loading required package: parallel

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:igraph':
##
##     normalize, union

## The following objects are masked from 'package:stats':
##
##     IQR, mad, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, cbind, colnames,
##     do.call, duplicated, eval, evalq, Filter, Find, get, grep,
##     grepl, intersect, is.unsorted, lapply, lengths, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff,
##     sort, table, tapply, union, unique, unsplit

##
## Attaching package: 'graph'

## The following objects are masked from 'package:igraph':
##
##     degree, edges, intersection

## Loading required package: grid
```

Step 2: Load an eviews file

Using the model from chapter 6 ("gl06open.prg") as demo case, using the following code to parse the eviews file and extract the adjacency matrix for the system.

```
model <- sfc.model("ch6.txt",modelName="Chapter6_openmodel")
```

The adjacency matrix for the system can be found by using `$matrix`.

Step 3: Your very first DAG

The next segment of codes takes the adjacency matrix and returns a list of igraph objects corresponding to the original graph, the nodes that form cycles and the resulting DAG for the system.

```
graphs = generate.DAG.collapse( adjacency = model$matrix )
```

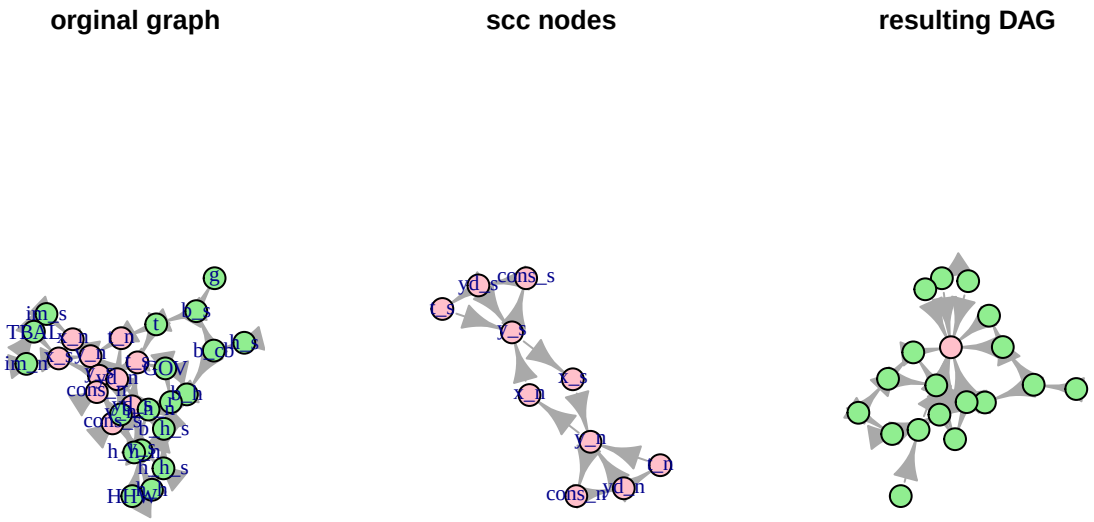
If instead of using an adjacency matrix, you wanted to use an igraph object, adding the argument `IGRAPH = TRUE` will allow the three graphs to be required with no errors.

```
graphs = generate.DAG.collapse(model$matrix)
```

Step 4: Generating graphics

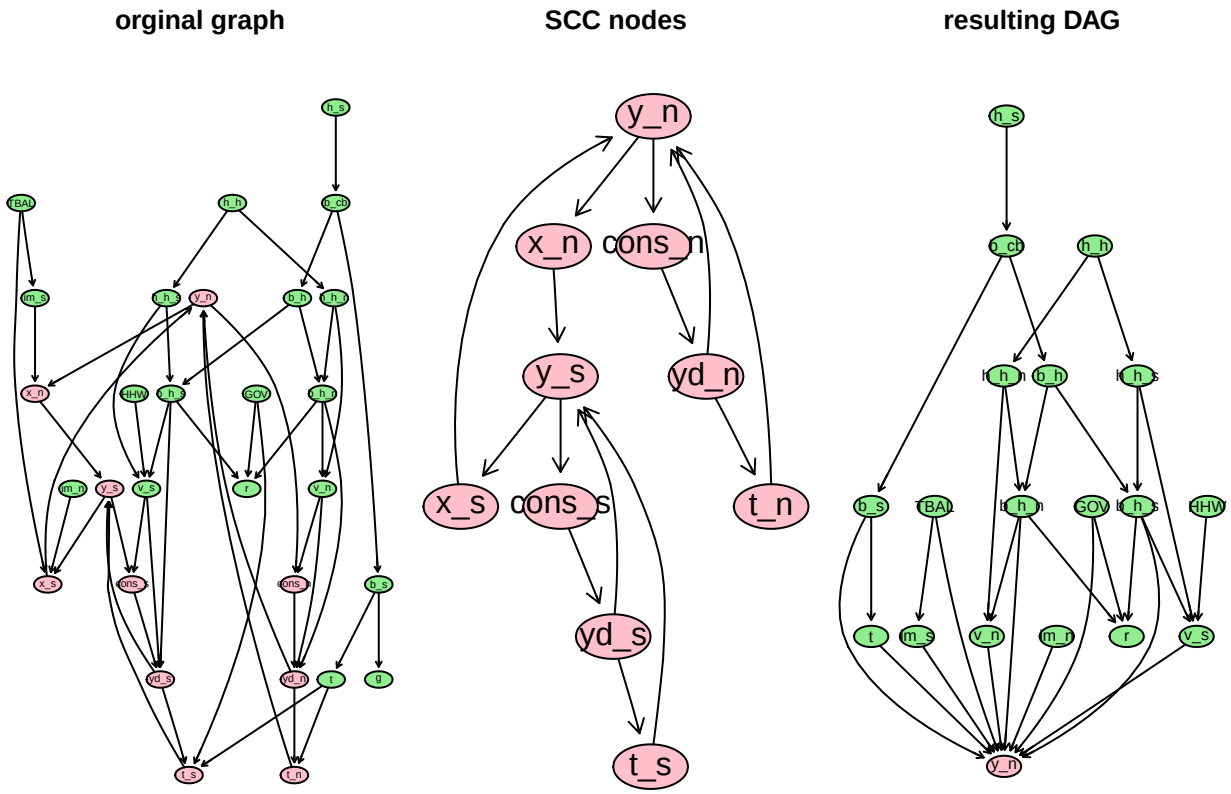
Two methods for generating graphical representations of DAGs from the models are presented here, first is the standard `plot` function using the igraph package.

```
par(mfrow = c(1,3))
# first plot the orgainal graph of the system
plot( graphs$orginal_graph,vertex.color = V(graphs$orginal_graph)$color,vertex.size = 20,main = "orginal
# just the nodes that form strongly connected componets
plot( graphs$SCC_graph, vertex.color = V(graphs$SCC_graph)$color, vertex.size = 20, main = "scc nodes")
# the resulting DAG when we take the condensation of the orginal graph
plot( graphs$DAG, vertex.color = unlist(V(graphs$DAG)$color), vertex.size = 20, vertex.label = NA, main
```



Or using the much nicer but less flexible function based in the Rgraphivz package to generate the plots we can delve into the actual structure of the system.

```
par(mfrow = c(1,3))
# first plot the orgianl grpah
plot_graph_hierarchy( graphs$orginal_graph, main = "orginal graph" )
# plot hte nodes that form the strongly connected compoent
plot_graph_hierarchy( graphs$SCC_graph, main = "SCC nodes" )
# plot the result ing DAG when we take the condensation of the graph
plot_graph_hierarchy( graphs$DAG, main = "resulting DAG" )
```



Nodes that do not form a cycle are green while nodes that form a cycle in the system are pink. To following segments of code will produce a vector of the green or pink nodes for inspection.

```
non_cycle_nodes = V(graphs$orginal_graph)[ V( graphs$orginal_graph )$color == "lightgreen" ]$name
cycle_nodes = V(graphs$orginal_graph)[ V(graphs$orginal_graph)$color == "pink" ]$name

head( non_cycle_nodes )
```

```
## [1] "im_n"  "im_s"  "v_n"   "v_s"   "h_h_n" "h_h_s"
```

To beauty of using igraph as the back bone for this tool is that it provides a lot of flexibility in how we can construct such graphical representations of SFC models. Models can be created using adjacency matrices, edges list or other graphical objects from other packages can also handled, for the most part, pain free!

Some useful reference materials:

The main documentation for the igraph package can be found here.

A brief introduction to igraph along with some of its basics can be found here here.