dynsim: An R implementation of dynamic simulations of autoregressive relationships

Christopher Gandrud, Laron K. Williams, and Guy D. Whitten

Abstract

This paper is a short introduction in how to use the **dynsim** R package for calculating dynamic simulations with autoregressive time series using the approach laid out in Williams and Whitten (2012). Building on the `Zelig` package, `dynsim` depicts long-run simulations of dynamic processes for a variety of substantively interesting scenarios, with and without the presence of exogenous shocks. The package includes the `dynsim` function for calculating the dynamic simulations and provides a function (`dynsimGG`) for graphically depicting the dynamic simulations with the `ggplot2` package.

Introduction

Two recent trends in the social sciences have drastically improved the interpretation of statistical models. The first trend is researchers providing substantively meaningful quantities of interest when interpreting models rather than putting the burden on the reader to interpret tables of coefficients King et al. (2000). With this goal in mind, Gary King and his co-authors have produced a series of packages, including the R package **Zelig** (Owen et al., 2013), that ease the calculation and presentation of substantive quantities of interest. The second trend is to completely interpret and present all of the inferences available from one's model. This is seen most obviously in the case of time-series models with an autoregressive series, where the effects of an explanatory variable have both a short- and a long-term component. A more complete interpretation of these models therefore requires additional work ranging from the presentation of long-term multipliers (De Boef and Keele, 2008) to dynamic simulations (Williams and Whitten, 2012).

We feel that these two trends can be naturally combined to allow scholars to easily depict long-term dynamic processes through simulations. Dynamic simulations depict long-run simulations of dynamic processes for a variety of substantively interesting scenarios, with and without the presence of exogenous shocks. In this short paper we introduce the **dynsim** (Gandrud et al., 2014) which makes it easy for researchers to implement this approach in R.[1]

In the next section we discuss the dynamic simulations with autoregressive time series approach. We then lay out the **dynsim** process and syntax for implementing this approach. Finally, we illustrate how to use the package with examples.

Dynamic simulations

Assume that we estimate the following partial adjustment model: $Y_t = \alpha_0 + \alpha_1 Y_{t-1} + \beta_0 X_t + \epsilon_t$, where Y_t is a continuous variable, X_t is an explanatory variable and ϵ_t is a random error term. The short-term effect of X_1 on Y_t is simple, β_0. This is the inference that social science scholars most often make, and unfortunately, the only one that they usually make (Williams and Whitten, 2012). However, since the model incorporates a lagged dependent variable (Y_{t-1}), a one-unit change in X_t on Y_t also has a long-term effect by influencing the value of Y_{t-1} in future periods. The appropriate way of calculating the long-term effect is with the long-term multiplier, or $\kappa_1 = \frac{\beta_0}{(1-\alpha_1)}$. We can then use the long-term multiplier to calculate the total effect that X_t has on Y_t distributed over future time periods. Of course,

[1]There is also a Stata (StataCorp., 2013) implementation documented in Williams and Whitten (2011).

the long-term multiplier will be larger as β_0 or α_1 gets larger in size.

We can use graphical depictions to most effectively communicate the dynamic properties of autoregressive time series across multiple time periods. The intuition is simple. For a given scenario of values of the explanatory variables, calculate the predicted value at time t: $\tilde{y} = \boldsymbol{X_C}\tilde{\beta} + \tilde{\epsilon}$, where $\tilde{\beta}$ is a vector of simulated effect coefficients, $\boldsymbol{X_C}$ is a matrix of user-specified values of variables, including y_{t-1}, and $\tilde{\epsilon}$ is one draw from $N(0, \tilde{\sigma}^2)$ (King et al., 2000). At each subsequent observation of the simulation, the predicted value of the previous scenario $(\tilde{y}|\mathbf{X_C})$ replaces the value of y_{t-1} to calculate $\tilde{y}_t$. Inferences such as long-term multipliers and dynamic simulations are based on *estimated* coefficients that are themselves uncertain. It is therefore very important to also present these inferences with the necessary measures of uncertainty (such as confidence intervals).

Dynamic simulations offer a number of inferences that one cannot make by simply examining the coefficients. First, one can determine whether or not the confidence interval for one scenario overlaps across time, suggesting whether or not there are statistically significant changes over time. Second, one can determine whether the confidence intervals of different scenarios overlap at any given time period, indicating whether the scenarios produce statistically different predicted values. Finally, if one includes exogenous shocks, then one can determine the size of the effect of the exogenous shock as well as how quickly the series then returns to its pre-shock state. These are all invaluable inferences for testing one's theoretical expectations.

dynsim process & syntax

Use the following four step process to simulate and graph autoregressive relationships with **dynsim**:

1. Estimate your linear model using `zelig` from the **Zelig** package (Owen et al., 2013).
2. Set up starting values for simulation scenarios and (optionally) shock values at particular iterations (e.g. points in forecasted time).
3. Simulate these scenarios based on the estimated model using the `dynsim` function.
4. Plot the simulation results with the `dynsimGG` function.

Before looking at examples of this process in action, let's look at the `dynsim` and `dynsimGG` syntax.

The `dynsim` command currently has seven arguments. The first–`obj`–is used to specify the model object estimated with `zelig`. You specify the lagged dependent variable with the `ldv` argument. The object containing the starting values for the simulation scenarios are set with `scen`. `n` allows you to specify the number of simulation iterations. These are equivalent to simulated 'time periods'. `scen` sets the values of the variables in the model at 'time' $n = 0$. To specify the level of statistical significance for the confidence intervals use the `sig` argument. By default it is set at `0.95` for 95 percent significance levels. You can adjust the number of simulations you draw for each point in time–i.e. each value of `n`–with the `num` argument. By default 1,000 simulations are drawn. Adjusting the number of simulations allows you to change the processing time. There is a trade-off however between the amount of time it takes to draw the simulations and the resulting information you have about about the simulations' probability distributions (King et al., 2000, 349). Finally you can identify the object that contains the shock values with the `shocks` argument.

Let's examine the basic characteristics of objects that can be used with the `scen` and `shocks` arguments. Objects for use with `scen` should be a list of data frames. Each data frame contains starting values for a different scenario. As such, each data frame will have one row with the starting values and as many columns as there are independent variables in the model estimated with `zelig`. Each column should be given a name that matches the names of a variable in the estimation model. If you have entered an interaction using *[2] then you only need to specify starting values for the base variables not the interaction term. The interaction will be calculated automatically.

shocks objects are simply data frames with the first column called times containing the iteration number (as in n) when a shock should occur. Note each shock must be at a unique time that cannot exceed n. The following columns are named after the shock variable(s), as they are labeled in the zelig model. The values will correspond to the variable values at each shock time. You can include as many shock variables as there are variables in the estimation model. Again only values for the base variables, not the interaction terms need to be specified.

Once the simulations have been run you will have a dynsim class object. These contain seven elements.

- scenNumber: The scenario number.
- time: The time points.
- shock. ...: Columns containing the values of the shock variables at each point in time.
- ldvMean: Mean of the simulation distribution.
- ldvLower: Lower bound of the simulation distribution's central interval set with sig.
- ldvUpper: Upper bound of the simulation distribution's central interval set with sig.
- ldvLower50: Lower bound of the simulation distribution's central 50 percent interval.
- ldvUpper50: Upper bound of the simulation distribution's central 50 percent interval.

You can easily convert dynsim objects into data frames and plot them with any available method in R. To convert an example dynsim object called DynOut into a data frame use:

```
class(DynOut) <- "data.frame"
```

A more convenient approach to plotting dynsim class objects is to use the dynsimGG command. This command draws on **ggplot2** (Wickham and Chang, 2013) to plot the simulation distributions across time. The distribution means are represented with a line. The range of the central 50 percent interval is represented with a dark ribbon. The range of the interval defined by the sig argument in dynsim, e.g. 95%, is represented with a lighter ribbon.

The primary dynsimGG argument is obj. Use this to specify the output object from dynsim that you would like to plot. The remaining arguments control the plot's aesthetics. For instance, the size of the central line can be set with the lsize argument and the level of opacity for the lightest ribbon[3] with the alpha argument. Please see the **ggplot2** documentation for more details on these arguments. You can change the color of the ribbons and central line with the color argument. If only one scenario is going to be plotted then you can manually set the color using a variety of formats, including hexadecimal color codes. If more than one scenario is going to be plotted, then select a color palette from those available in the **RColorBrewer** package (Neuwirth, 2011).[4] The plot's title, y-axis and x-axis labels can be set with the title, ylab, and xlab arguments, respectively.

There are three arguments that allow you to adjust the look of the scenario legend. leg.name allows you to choose the legend's name and leg.labels lets you change the scenario labels. This must be a character vector with new labels in the order of the scenarios in the scen object. legend allows you to hide the legend entirely. Simply set legend = FALSE.

[2]For example, an interaction between Var1 and Var2 entered into zelig as Var1*Var2.

[3]The darker 50 percent central interval ribbon is created by essentially doubling the opacity set by alpha.

[4]To see the full list, after loading **RColorBrewer** in your R session, simply enter brewer.pal.info into your console.

Finally, if you included shocks in your simulations you can use the `shockplot.var` argument to specify *one* shock variable's fitted values to include in a plot underneath the main plot. Use the `shockplot.ylab` argument to change the y-axis label.

The output from `dynsimGG` is generally a **ggplot2** gg class object.[5] Because of this you can further change the aesthetic qualities of the plot using any relevant command from **ggplot2**. Additionally, you can combine these plots created with `dynsimGG` with other **ggplot2** plots using the **gridExtra** (Auguie, 2012) package.

Examples

To get a better sense of how **dynsim** works, let's go through a few examples. We will use the Grunfeld (1958) data set. It is included with **dynsim**. To load the data use:

```
data(grunfeld, package = "dynsim")
```

The linear regression model we will estimate is:

$$I_{it} = \alpha + \beta_1 I_{it-1} + \beta_2 F_{it} + \beta_3 C_{it} + \mu_{it} \tag{1}$$

where I_{it} is real gross investment for firm i in year t. I_{it-1} is the firm's investment in the previous year. F_{it} is the real value of the firm and C_{it} is the real value of the capital stock.

In the `grunfeld` data set real gross investment is denoted `invest`, the firm's market value is `mvalue`, and the capital stock is `kstock`. There are 10 large US manufacturers from 1935-1954 in the data set (Baltagi, 2001). The variable identifying the individual companies is called `company`. We can easily create the investment one-year lag using the `slide` command from the **DataCombine** package (Gandrud, 2014). Here is the code:

```
grunfeld <- slide(grunfeld, Var = "invest", GroupVar = "company",
               NewVar = "InvestLag")
```

The new lagged variable is called `InvestLag`.

Dynamic simulation without shocks

Now that we have created our lag variable we, can begin to create dynamic simulations with **dynsim** by estimating the underlying linear regression model using **Zelig**, i.e.:

```
M1 <- zelig(invest ~ InvestLag + mvalue + kstock,
            model = "ls", data = grunfeld, cite = FALSE)
```

We can use the resulting model object `M1` in the `dynsim` command to run our dynamic simulations. We first need to create a list object containing data frames with starting values for each simulation scenario. Imagine we want to run three contrasting scenarios with the following fitted values:

- **Scenario 1**: mean lagged investment, market value and capital stock held at their 95th percentiles,
- **Scenario 2**: all variables held at their means,
- **Scenario 3**: mean lagged investment, market value and capital stock held at their 5th percentiles.

We can create a list object for the `scen` argument containing each of these scenarios with the following code:

[5]This is not true if you include a shock plot.

```
# Create simulation scenarios
attach(grunfeld)
Scen1 <- data.frame(InvestLag = mean(InvestLag, na.rm = TRUE),
                    mvalue = quantile(mvalue, 0.95),
                    kstock = quantile(kstock, 0.95))
Scen2 <- data.frame(InvestLag = mean(InvestLag, na.rm = TRUE),
                    mvalue = mean(mvalue),
                    kstock = mean(kstock))
Scen3 <- data.frame(InvestLag = mean(InvestLag, na.rm = TRUE),
                    mvalue = quantile(mvalue, 0.05),
                    kstock = quantile(kstock, 0.05))
detach(grunfeld)

# Combine into a single list
ScenComb <- list(Scen1, Scen2, Scen3)
```

In this example we won't introduce shocks, so now we run our simulations:

```
Sim1 <- dynsim(obj = M1, ldv = "InvestLag", scen = ScenComb, n = 20)
```

This took 1000 draws of the coefficients to produce 20 dynamic simulation iterations for each of the three scenarios. In the resulting `Sim1` object we have retained the middle 95% and 50%. We will return to presenting and exploring the contents of `Sim1` later in this section.

Dynamic simulation with shocks

Now let's include fitted shock values. In particular, let's see how a company with capital stock in the 5th percentile is predicted to change its gross investment when its market value experiences shocks compared to a company with capital stock in the 95th percentile. We will use market values for the first company in the `grunfeld` data set over the first 15 years as the shock values. To create the shock data use the following code:

```
# Keep only the mvalue for the first company for the first 15 years
grunfeldsub <- subset(grunfeld, company == 1)
grunfeldshock <- grunfeldsub[1:15, "mvalue"]

# Create data frame for the shock argument
grunfeldshock <- data.frame(times = 1:15, mvalue = grunfeldshock)
```

Now we can simply add `grunfeldshock` to the `dynsim` `shocks` argument.

```
Sim2 <- dynsim(obj = M1, ldv = "InvestLag", scen = ScenComb, n = 15,
               shocks = grunfeldshock)
```

We can easily incorporate interactions between the shock variable and another exogenous variable. To include, for example, an interaction between the firm's market value (the shock variable) and the capital stock (another exogenous variable) we simply need to rerun the basic parametric model like so:

```
M2 <- zelig(invest ~ InvestLag + mvalue*kstock,
            model = "ls", data = grunfeld, cite = FALSE)
```

We then run `dynsim` as before. The only change is that we use the fitted model object `M2` that includes the interaction.

```
Sim3 <- dynsim(obj = M2, ldv = "InvestLag", scen = ScenComb, n = 15,
               shocks = grunfeldshock)
```

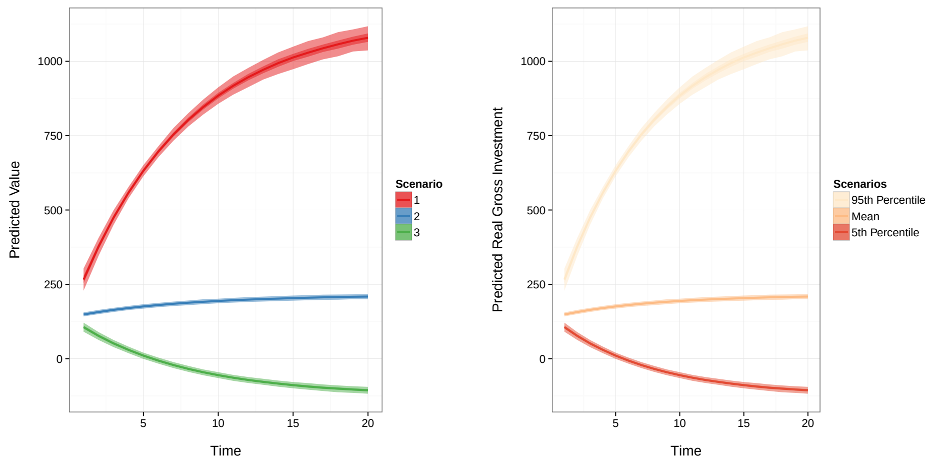

Figure 1: An Example of a Dynamic Simulation of Three Scenarios

Plotting simulations

Probably the easiest and most effective way to communicate your `dynsim` simulation results is with the package's built-in plotting capabilities. These are accessed with the `dysnsimGG` command. Let's start by plotting the simulations from our basic model. In an earlier example we put these results into `Sim1`. Using `dynsimGG`'s default settings we create the left-panel in Figure 1 with the following code:

```
dynsimGG(Sim1)
```

We can make a number of aesthetic changes. For example, we can re-title the legend using the `leg.name` argument and change the legend values by adding a vector of the desired label names in the order of our scenarios using the `leg.labels` argument. We can also change the color palette using the `color` argument. Let's use the 'orange-red' palette denoted by `OrRd`. Finally, we can change the y-axis label with the `ylab` argument so that it is more descriptive. Adding these changes using the following code creates the right-panel in Figure 1.

```
# Create legend labels vector
Labels <- c("95th Percentile", "Mean", "5th Percentile")

# Plot
dynsimGG(Sim1, leg.name = "Scenarios", leg.labels = Labels, color = "OrRd",
         ylab = "Predicted Real Gross Investment\n")
```

Plotting simulations with shock variables is very similar to what we have already seen. The one major change we can make is to include a plot of one shock variable's fitted values underneath the main plot. To do this simply use the `shockplot.var` argument to specify which variable to plot. Use the `shockplot.ylab` argument to change the y-axis label. For example, the following code will create Figure 2:

```
dynsimGG(Sim2, leg.name = "Scenarios", leg.labels = Labels, color = "OrRd",
       ylab = "Predicted Real Gross Investment\n", shockplot.var = "mvalue",
       shockplot.ylab = "Firm Value")
```

Similarly, we can plot the simulations from scenarios where market value shocks were interacted with capital stock. This can be seen in Figure 3. The code is almost identical to the code chunk we just saw. The only difference is that we use the `Sim3` model object.

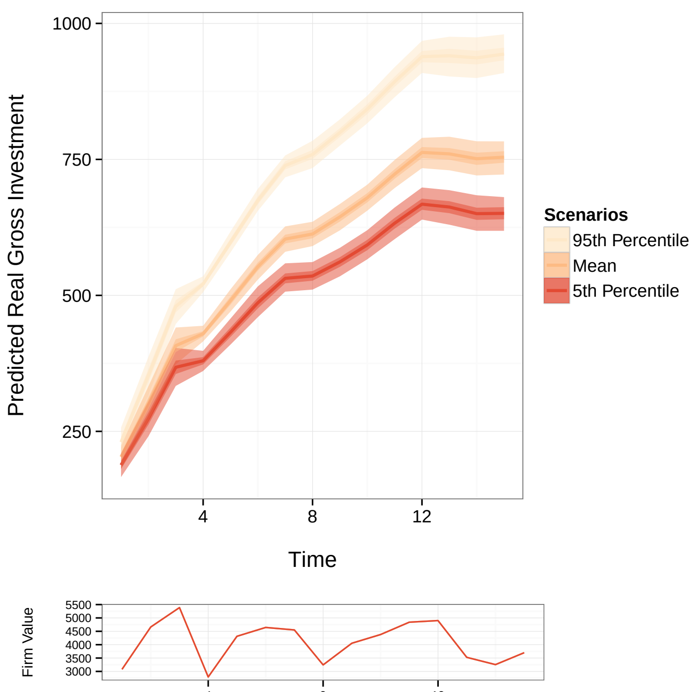

Figure 2: An Example of a Dynamic Simulation with the Inclusion of a Shock Variable

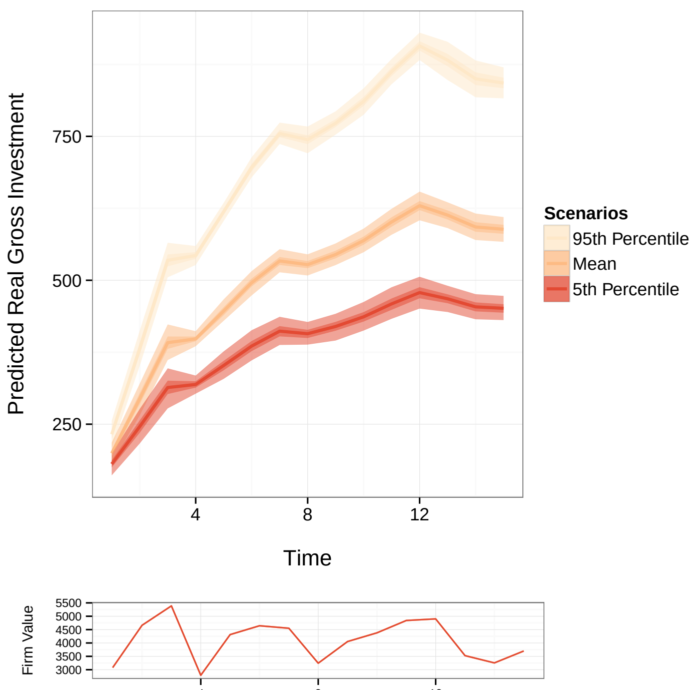

Figure 3: An Example of a Dynamic Simulation Where a Shock Variable Is Interacted with an Exogenous Variable

Summary

In this short paper we have demonstrated how the R package **dynsim** makes it easy implement Williams and Whitten's (2012) approach to more completely interpreting results from autoregressive time-series models where the effects of explanatory variables have both short- and long-term components. Hopefully this will lead to more meaningful investigations and more useful presentations of results estimated from these relationships.

Bibliography

B. Auguie. *gridExtra: functions in Grid graphics*, 2012. URL `http://CRAN.R-project.org/package=gridExtra`. R package version 0.9.1. [p4]

B. Baltagi. *Econometric Analysis of Panel Data*. Wiley and Sons, Chichester, UK, 2001. [p4]

S. De Boef and L. Keele. Taking time seriously. *American Journal of Political Science*, 52(1): 184–200, 2008. [p1]

C. Gandrud. *DataCombine: R tools for making it easier to combine and clean data sets.*, 2014. URL `http://christophergandrud.github.io/DataCombine/`. R package version 0.1.20. [p4]

C. Gandrud, G. D. Whitten, and L. K. Williams. *dynsim: An R implementation of dynamic simulations of autoregressive relationships*, 2014. R package version 0.2.3. [p1]

Y. Grunfeld. *The Determinants of Corporate Investment*. PhD thesis University of Chicago, 1958. [p4]

G. King, M. Tomz, and J. Wittenberg. Making the Most of Statistical Analyses: Improving Interpretation and Presentation. *American Journal of Political Science*, 44(2):347–361, 2000. [p1, 2]

E. Neuwirth. *RColorBrewer: ColorBrewer palettes*, 2011. URL `http://CRAN.R-project.org/package=RColorBrewer`. R package version 1.0-5. [p3]

M. Owen, K. Imai, G. King, and O. Lau. *Zelig: Everyone's Statistical Software*, 2013. URL `http://CRAN.R-project.org/package=Zelig`. R package version 4.2-1. [p1, 2]

StataCorp. Stata statistical software: Release 13, 2013. [p1]

H. Wickham and W. Chang. *ggplot2: An implementation of the Grammar of Graphics*, 2013. URL `http://CRAN.R-project.org/package=ggplot2`. R package version 0.9.3.1. [p3]

L. K. Williams and G. D. Whitten. Dynamic Simulations of Autoregressive Relationships. *The Stata Journal*, 11(4):577–588, 2011. [p1]

L. K. Williams and G. D. Whitten. But Wait, There's More! Maximizing Substantive Inferences from TSCS Models. *Journal of Politics*, 74(03):685–693, 2012. [p1, 8]

Christopher Gandrud
Hertie School of Governance
Friedrichstraße 180
Berlin, 10117
Germany
`gandrud@hertie-school.org`

Laron K. Williams
University of Missouri
Department of Political Science
103 Professional Building

Columbia, MO 65211-6030
United States
`williamslaro@missouri.edu`

Guy D. Whitten
Texas A & M University
Allen Building 2070
College Station, TX 77843-4348
United States
`g-whitten@pols.tamu.edu`