

We thank Lily L. Tsai and Teppei Yamamoto for their helpful comments and suggestions.

### References

- Adida, L. Claire, Karen E. Ferree, Daniel N. Posner, and Amanda Lea Robinson. 2016. "Who's Asking? Interviewer Coethnicity Effects in African Survey Data." *Comparative Political Studies*. Forthcoming. <http://dx.doi.org/10.1177/0010414016633487> (August 1, 2016).
- Berinsky, Adam, Tesalia Rizzo, Leah R. Rosenzweig, and Elisha Heaps. 2015. "Attribute Affinity: US Natives' Attitudes Toward Immigrants." Working paper.
- Carlson, Elizabeth. 2015. "Ethnic Voting and Accountability in Africa: A Choice Experiment in Uganda." *World Politics* 67(2): 353-85.
- Hainmueller, Jens, and Daniel J. Hopkins. 2014. "The Hidden American Immigration Consensus: A Conjoint Analysis of Attitudes toward Immigrants." *American Journal of Political Science* 59(3): 529-48.
- Hainmueller, Jens, Daniel J. Hopkins, and Teppei Yamamoto. 2014. "Causal Inference in Conjoint Analysis: Understanding Multidimensional Choices via Stated Preference Experiments." *Political Analysis* 22(1): 1-30.
- Hainmueller, Jens, Dominik Hangartner, and Teppei Yamamoto. 2015. "Validating Vignette and Conjoint Survey Experiments against Real-world Behavior." *Proceedings of the National Academy of Sciences* 112(8): 2395-400.
- Hartman, Alexandra, and Ben Morse. 2015. "Wartime Violence, Empathy, and Inter-group Altruism: Theory and evidence from the Ivoirian refugee crisis in Liberia." Working paper.
- Horiuchi, Yusaku, Daniel M Smith, and Teppei Yamamoto. 2015. "Identifying Multidimensional Policy Preferences of Voters in Representative Democracies: A Conjoint field experiment in Japan." *SSRN* 2627907.
- Meyer, Alexander, and Leah R. Rosenzweig. 2016. "Replication Data for: Conjoint Analysis Tools for Developing Country Contexts." <http://dx.doi.org/10.7910/DVN/DURDD3> Harvard Dataverse, V1. (August 1, 2016).
- Rosenzweig, Leah R., and Lily L. Tsai. 2016. "Using Facebook for Online Experiments." Working paper.
- Rosenzweig, Leah R., and Lily L. Tsai. N.d. "Evaluating Candidates: A Conjoint Experiment in Tanzania." Working paper.

## Visualize Dynamic Simulations of Autoregressive Relationships in R

**Christopher Gandrud**

City University London and Hertie School of Governance  
[christopher.gandrud@city.ac.uk](mailto:christopher.gandrud@city.ac.uk)

**Laron K. Williams**

University of Missouri  
[williamslaro@missouri.edu](mailto:williamslaro@missouri.edu)

**Guy D. Whitten**

Texas A&M University  
[g-whitten@tamu.edu](mailto:g-whitten@tamu.edu)

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). The second trend is a movement to more completely interpret and present 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 short- and a long-term components. 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).

These two trends can be combined to allow scholars to easily depict the long-term implications from estimates of dynamic processes through simulations. Dynamic simulations can be employed to depict long-run simulations of dynamic processes for a variety of substantively-interesting scenarios, with and without the presence of exogenous shocks. We introduce *dynsim* (Gandrud et al. 2016) which makes it easy for researchers to implement this approach in R.<sup>1</sup>

### 1. 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  $\epsilon_t$  is a random error term. The short-term effect of  $X_1$  on  $Y_t$  is simple,  $\beta_0$ . 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

<sup>1</sup>There is also a Stata (StataCorp. 2015) implementation documented in Williams and Whitten (2011).

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, the long-term multiplier will be larger as  $\beta_0$  or  $\alpha_1$  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  $t$ :  $\tilde{y} = \mathbf{X}_C\tilde{\beta} + \tilde{\epsilon}$ , where  $\tilde{\beta}$  is a vector of simulated effect coefficients,  $\mathbf{X}_C$  is a matrix of user-specified values of variables, including  $y_{t-1}$ , and  $\tilde{\epsilon}$  is one draw from  $N(0, \tilde{\sigma}^2)$ . 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 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.

## 2. dynsim process and syntax

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

1. Estimate the linear model using the core R function `lm`.
2. Set up starting values for simulation scenarios and (optionally) shock values at particular iterations (e.g. points in simulated 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` function has seven arguments. The first—`obj`—is used to specify the model object. The lagged dependent variable is identified with the `ldv` argument. The object containing the starting values for the simulation scenarios is identified 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. Note that `dynsim` currently depicts uncertainty in the systematic component of the model, rather than forecast uncertainty. The practical implication of this is that the confidence intervals will not grow over the simulation iteration. The number of simulations drawn for each point in time—i.e. each value of `n`—is adjusted 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 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, p. 349). Finally the object containing the shock values is identified with the `shocks` argument.

Objects for use with `scen` can be either a list of data frames—each data frame containing starting values for a different scenario—or a data frame where each row contains starting values for different scenarios. In both cases, the data frames have as many columns as there are independent variables in the estimated model. 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 simulated values for the interaction will be found automatically.

`shocks` objects are 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 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. Because `dynsim` objects are also data frames you can plot them with any available method in R. They contain at least seven columns:

- `scenNumber`: The scenario number.

<sup>2</sup>For example, an interaction between `Var1` and `Var2` entered into the model as `Var1*Var2`.

- **time**: The time points.
- **shock**. . . .: Optional 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.

The `dynsimGG` function is the most convenient plotting approach. This function draws on `ggplot2` (Wickham and Chang 2015) 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<sup>3</sup> 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 plotted then you can manually set the color using a variety of formats, including hexadecimal color codes. If more than one scenario is plotted, then select a color palette from those available in the `RColorBrewer` package (Neuwirth 2014).<sup>4</sup> 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`.

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.<sup>5</sup> Because of this you can further change the aesthetic qualities of the plot using any relevant function from `ggplot2` using the `+` operator. You can also convert them to interactive graphics with `ggplotly` from the `plotly` (Sievert et al. 2016) package.

### 3. Examples

The following examples demonstrate how `dynsim` works. They 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:

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

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 within each `company` group using the `slide` function from the `DataCombine` package (Gandrud 2016). Here is the code:

```
library(DataCombine)

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

The new lagged variable is called `InvestLag`. The reason we use `slide` rather than R's core `lag` function is that the latter is unable to lag a grouped variable. You could of course use any other appropriate function to create the lags.

#### 3.1. Dynamic simulation without shocks

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

```
M1 <- lm(invest ~ InvestLag + mvalue + kstock, data = grunfeld)
```

The resulting model object—`M1`—is used in the `dynsim` function to run the dynamic simulations. We first 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,

<sup>3</sup>The darker 50 percent central interval ribbon is created by essentially doubling the opacity set by `alpha`.

<sup>4</sup>To see the full list, after loading `RColorBrewer` in your R session, simply enter `brewer.pal.info` into your console.

<sup>5</sup>This is not true if you include a shock plot.

- **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:

```
> 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)

ScenComb <- list(Scen1, Scen2, Scen3)
```

To run the simulations without shocks use:

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

### 3.2. Dynamic simulation with shocks

Now we include fitted shock values. In particular, we will examine 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 add `grunfeldshock` to the `dynsim` shocks argument.

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

Interactions between the shock variable and another exogenous variable can also be simulated for. To include, for example, an interaction between the firm's market value (the shock variable) and the capital stock (another exogenous variable) we need to rerun the parametric model like so:

```
M2 <- lm(invest ~ InvestLag + mvalue*kstock, data = grunfeld)
```

We then use `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)
```

### 3.3. Plotting simulations

The easiest and most effective way to communicate `dynsim` simulation results is with the package's built-in plotting capabilities, e.g.:

```
dynsimGG(Sim1)
```

We can make a number of aesthetic changes. The following code adds custom legend labels, the 'orange-red' color palette—denoted by `OrRd`—, and relabels the y-axis to create Figure 1.

```
Labels <- c("95th_Percentile", "Mean", "5th_Percentile")
dynsimGG(Sim1, leg.name = "Scenarios", leg.labels = Labels,
  color = "OrRd",
  ylab = "Predicted_Real_Gross_Investment\n")
```

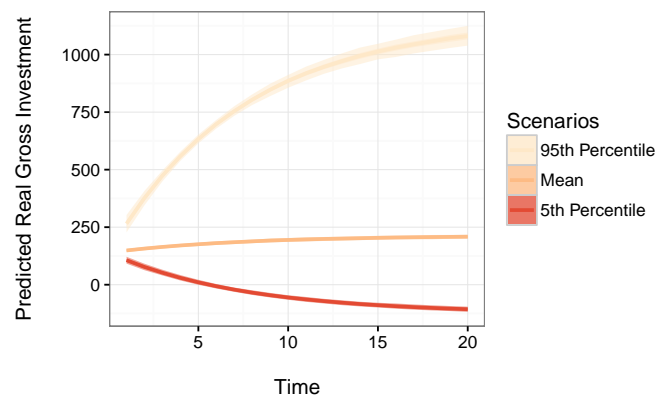


Figure 1: Three Dynamic Simulations Plotted with Custom Scenario Labels and Color Palette

When plotting simulations with shock values another plot can be included underneath the main plot showing one shock variable's fitted values. To do this 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 creates 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")
```

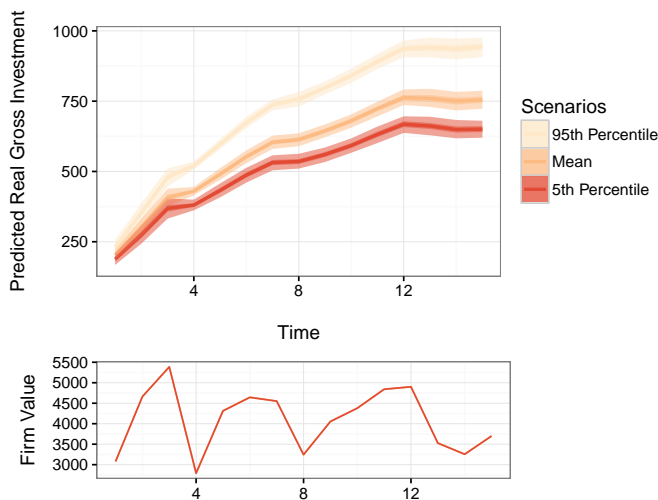


Figure 2: An Example of a Dynamic Simulation with the Inclusion of a Shock Variable

#### 4. Summary

We have demonstrated how the R package `dynsim` makes it easy to 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.

#### References

- Baltagi, Badi H. 2001. *Econometric Analysis of Panel Data*. Chichester, UK: Wiley and Sons.
- De Boef, Suzanna, and Keele, Luke. 2008. "Taking Time Seriously." *American Journal of Political Science* 52(1): 184-200.

## Embrace your Fallibility: Thoughts on Code Integrity

Nicholas Eubank  
Stanford University  
[nicholaseubank@stanford.edu](mailto:nicholaseubank@stanford.edu)

*Two years ago, I wrote a piece about my experiences over two years testing the code for papers being published in the Quarterly Journal of Political Science, which found problems in the code of many papers. The piece was first published*

- Gandrud, Christopher. 2016. *DataCombine: Tools for Easily Combining and Cleaning Data Sets*. R package version 0.2.21. <https://CRAN.R-project.org/package=DataCombine> (August 1, 2016).
- Gandrud, Christopher, Williams, Laron K., and Whitten, Guy D. 2016. *dynsim: Dynamic Simulations of Autoregressive Relationships*. R package version 1.2.2. <https://CRAN.R-project.org/package=dynsim> (August 1, 2016).
- Grinfeld, Yehuda. (1958). "The Determinants of Corporate Investment." Ph.D.diss. The University of Chicago.
- King, Gary, Tomz, Michael, and Wittenberg, Jason. 2000. "Making the Most of Statistical Analyses: Improving Interpretation and Presentation." *American Journal of Political Science* 44(2): 347-61.
- Neuwirth, Erich. 2014. *RColorBrewer: ColorBrewer palettes*. R package version 1.1-2. <https://CRAN.R-project.org/package=RColorBrewer> (August 1, 2016).
- Sievert, Carson, Chris Parmer, Toby Hocking, Scott Chamberlain, Karthik Ram, Marianne Corvellec, and Pedro Despouy. 2016. *plotly: Create Interactive Web Graphics via 'plotly.js'*. R package version 3.4.13. <https://cran.r-project.org/package=plotly> (August 1, 2016).
- StataCorp. 2015. *Stata Statistical Software: Release 14*. College Station, TX: StataCorp LP.
- Wickham, Hadley, and Winston Chang. 2015. *ggplot2: An implementation of the Grammar of Graphics*. R package version 1.0.1. <https://cran.r-project.org/package=ggplot2> (August 1, 2016).
- Williams, Laron K., and Guy D. Whitten. 2011. "Dynamic Simulations of Autoregressive Relationships." *The Stata Journal* 11(4): 577-88.
- Williams, Laron K., and Guy D. Whitten. 2012. "But Wait, There's More! Maximizing Substantive Inferences from TSCS Models." *Journal of Politics* 74(3): 685-93.

*in The Political Methodologist (Eubank 2014), and later in PS: Politics and Political Science (Eubank 2016). This piece is an extension of that article based on conversations that article sparked and my own experiences over the past two years.*

It's natural to think that the reason we find problems in the code behind published papers is carelessness or inattention on behalf of authors, and that the key to minimizing problems in our code is to be more careful. The truth, I have come to believe, is more subtle: humans are effectively