Quick look at the margins command

Jean-Philippe Gauvin
jean-philippe.gauvin@umontreal.ca

October 9, 2012

Abstract

In this document I try to give an overview of the `margins` command in Stata. I start with a basic explanation of the difference between a predictive margin and a marginal effect. I then explain how to use the `marginsplot` command to visualise results. Finally, I explain how to run multiple models with `margins` and overlay the results on one graph.

1 Predictive Margins

In Stata 12, the `margins` command can do multiple things, one of them being a *predictive margin*, or more generally an *adjusted prediction*. This means that the `margins` command can, after running a model, predict the expected value of an estimate, while holding every other variable either at their means or their observed value.

For example:

```
regress y x1 x2 x3 x4 x5
margins, at(x2=3) atmeans
```

In this example, I predict $\hat{y}$ (i.e. the outcome of y) when $x2$ is 3, while holding every other variables constant at their mean. If I used the `asobserved` argument instead of `atmeans`, Stata would run a simulation for each observation and then average the effect.

The adjustment is not limited to one variable; it can also be used to run profile simulations.

For example:

```
regress income female i.region i.ideology i.education
margins, at(female=1 region=3 ideology=3 education=1)
```

In this example, the answer given by `margins` will be the linear prediction of income when the respondent is a female living in region 3 while having a right-leaning ideology with low education.

The interesting thing with `margins` is that you can repeat the process and compute that adjusted prediction for different values of x. Also, it is not limited to linear models.

For example:

```
logit GW_xp01 c.hierarchy##i.RISK ideology female
margins RISK, at(hierarchy=(1 2 3 4 5 6)) atmeans
```

In this example, GW_xp01 is a dichotomous variable where 0 means that the expert is not found credible and 1 means that the expert is found to be credible. hierarchy is a continuous variable going from 1 to 6. RISK is a dichotomous variable, where 0 means that the respondent was in the control group and 1 means that he was in the experiment group. We use the ## argument to tell Stata to use the whole multiplication term in the interaction, as it should. In sum, we are estimating a *predicted probability*: that of finding the expert credible depending on the group (control or experiment), at different values of hierarchy, while maintaining every other covariate at its mean.

The marginsplot command then provides the graphical representation of those adjusted predictions.

2 Marginal Effect

The *marginal effect* (also called *partial change* or *partial derivative*) is the difference in prediction of the presence and absence of a variable. The idea is simple to grasp when thinking of a dummy variable in a logit model. Since the y outcome is dichotomous, the prediction is the probability of getting a 1 rather than a 0. When taking the marginal effect of a dummy variable, we try to get the effect of that variable when it is one, minus its effect when it is zero. In other words, the marginal effect of x_1 is the difference in the probability of y being 1 given that x is 1, minus the probability that $y = 1$ given that x is equal to 0 – as shown in equation 1:

$$m_e(x_1) = Pr(y = 1|x = 1) - Pr(y = 1|x = 0) \tag{1}$$

But the marginal effect might not always be appropriate in the case of dependent categorical variable. To understand this specifically, let

$$Pr(y = 1|\mathbf{x}) = F(\mathbf{x}\beta) \tag{2}$$

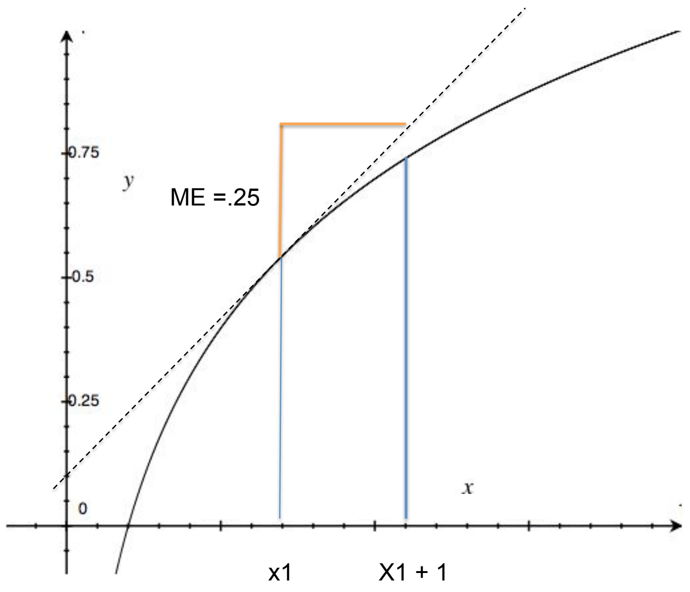
where F is any cumulative density function (as Φ for the normal distribution in the case of probit or Λ for the logistic distribution in the case of logit). Hence, the marginal effect is also called the *partial change in the probability* since it is computed by simple calculus, ie. by taking the partial derivative of equation 2 with respect to x_k:

$$\frac{\partial Pr(y = 1|\mathbf{x})}{\partial x_k} = \frac{\partial F(\mathbf{x}\boldsymbol{\beta})}{\partial x_k} = \frac{dF(\mathbf{x}\boldsymbol{\beta})}{d\mathbf{x}\boldsymbol{\beta}}\frac{\partial \mathbf{x}\boldsymbol{\beta}}{\partial x_k} = f(\mathbf{x}\boldsymbol{\beta})\beta_k \tag{3}$$

As we can see from this transformation, we start with the cdf (F) and end up with the probability density function or pdf (f). In Stata, we would use the option dydx(*) after the margins command, leaving the star in to provide marginal effects for all covariates, or placing one or several covariates between the brackets.

As showed in Figure 1, the derivative method means that we must be careful, since we are estimating the effect from the tangent and not the slope itself. In other words, there is no problem in linear relationships, but there might be some discrepancies in non-linear relationships like in logits and probits. The following graph provides an exaggerated example. Long (1997) suggests alternatives to the marginal effect, like the discrete change. Indeed, using discrete change will actually "touch" the slope, compared to the orange line in the following figure who only touches the derived tangent.

Figure 1: Measuring the marginal effect



2.1 Marginal Effects and Linear Models

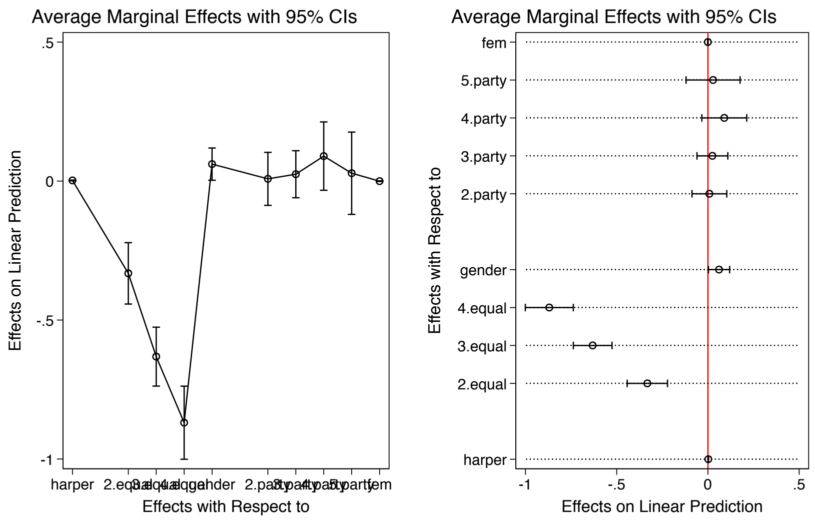
The last section took the first derivative of a logistic function with the `margins` command. But is it possible to use it with a linear model? The short answer is yes, but the results would not be very useful. Indeed, the marginal effect of a linear relationship is the beta coefficient. This is because, as explained in the last section, a marginal effect – or partial change in the function – is computed from the derivative of the function at a certain point on the curve. In other words, the first derivative measures the slope of the tangent. But if the relationship is linear, the result will be the slope of the regression line, hence the beta coefficient.

But if this result does not seem useful, the `marginsplot` command provide a neat way to present regression results. However, using it directly after taking the marginal effects of the linear prediction slope might not produce the best graphic, as can be seen from the left portion of figure 2. Indeed, the connected line makes it uninterpretable, while the x-axis is impossible to read. But the graphic can be greatly improve by flipping it horizontally, recasting the graphic as a dot plot and setting a line at 0 on the x-axis. This can be done by specifying the following options:

```
marginsplot, recast(dot) horizontal xline(0, lcolor(red))
```

The `recast` command reshapes the graphic in a different format – in this case, a dot plot. The `horizontal` option flips the graph on its side. Finally, the `xline` option creates a vertical line on the x-axis at value 0 and formats it in red. If no color is necessary, `xline(0)` can be used on its own. The result can be seen in the right portion of figure 2. Notice that the betas are now standardized, which facilitates reading. With their confidence interval, the variables that are significantly different from zero are easy to identify. The next step would be to reorder the variables by effect size.

Figure 2: Marginsplot with Regression Results



3 One Limit of the Margins Command

While `margins` can be very useful at isolating a factor that explains the dependant variable, it can only simulate one model at a time. This can be very unpractical when trying to estimate multinomial logit regression models, since it will only predict one outcome at a time. The same thing applies to ordered logits.

Say we want to know how the perception of a specific leader influences the probability of voting for different parties. We could run the following model in Stata:

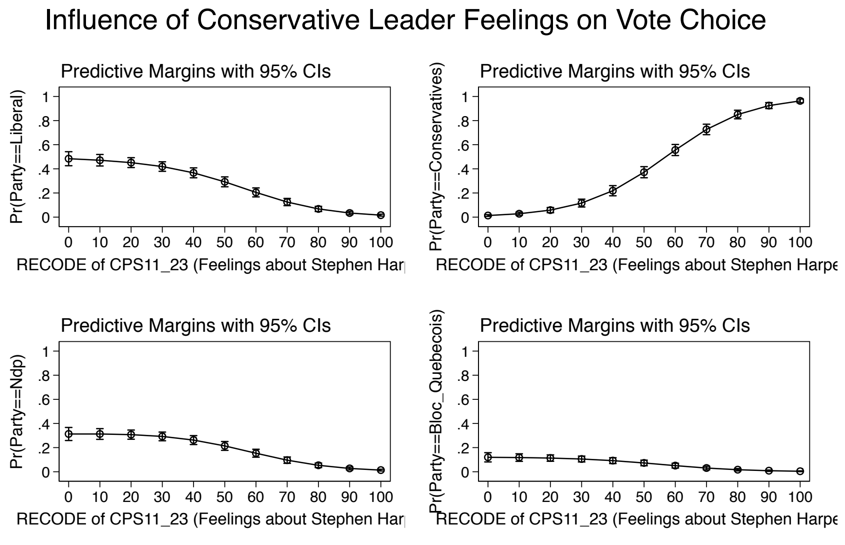
```
mlogit party gender##c.fem rightscale Cons_leader, rr
```

Where `Cons_leader` is a thermometer scale (from 0 to 100) of the perception of the Conservative leader in this election. In a model where there are four parties, we could use a loop to create and combine four different graphics:

```
forvalues i=1/4 {
   margins, at(Cons_leader=(0(10)100)) predict(outcome(‘i’))
   marginsplot
   graph save margins‘i’.gph
}
graph combine margins1.gph margins2.gph margins3.gph margins4.gph, ///
ycommon ti(Influence of Conservative Leader Feelings on Vote Choice)
```

This would give us a graphic similar to the one in Figure 3. While it is not an absolutely ugly graph – except for the axis labels – most people would want to simplify interpretation by overlaying the different slopes on one and only graph. Fortunately, it is possible to create those

Figure 3: Example graph



slopes manually by extracting the data from Stata's inner memory. But this requires knowledge of how Stata handles its data. In fact, when posting results, it actually stores data in different matrices. We can then run the following syntax to gather the relevant data for the slopes. I will go into a line by line explanation in the next paragraph.

```
set more off
mlogit party gender##c.fem rightscale Cons_leader, rr

forvalues j=1/4 {
  margins, at(Cons_leader=(0(10)100)) predict(outcome(‘j’)) atmeans vsquish post

  mat t=J(11,3,.)

     mat a = (0\10\20\30\40\50\60\70\80\90\100)      /* get the "at" values    */

  forvalues i=1/11 {
    mat t[‘i’,1] = _b[‘i’._at]                      /* get probability estimates */
    mat t[‘i’,2] = _b[‘i’._at] - 1.96*_se[‘i’._at]   /* compute lower limit       */
    mat t[‘i’,3] = _b[‘i’._at] + 1.96*_se[‘i’._at]   /* compute upper limit       */
  }

  mat t=t,a
  mat colnames t = prob_party‘j’ lci_party‘j’ uci_party‘j’ at_party‘j’
  svmat t, names(col)
}
```

In order to save some place, I use a loop within a loop to repeat the process multiple times. The first step is to run the model we want to estimate, hence the `mlogit` command. The results include four different outcomes for four different political parties. I then use a loop to repeat the extraction procedure four times, one for each party. The `'j'` argument will be repeatedly replace by 1, 2, 3 and then 4.

Inside the first loop, we run the `margins` command, setting our predicted probability at 11 values of `Cons_leader`, being from 0 to 100, by increments of 10. I also use the `vsquish` argument to synthesise the output and the `post` argument to save the results in matrices.

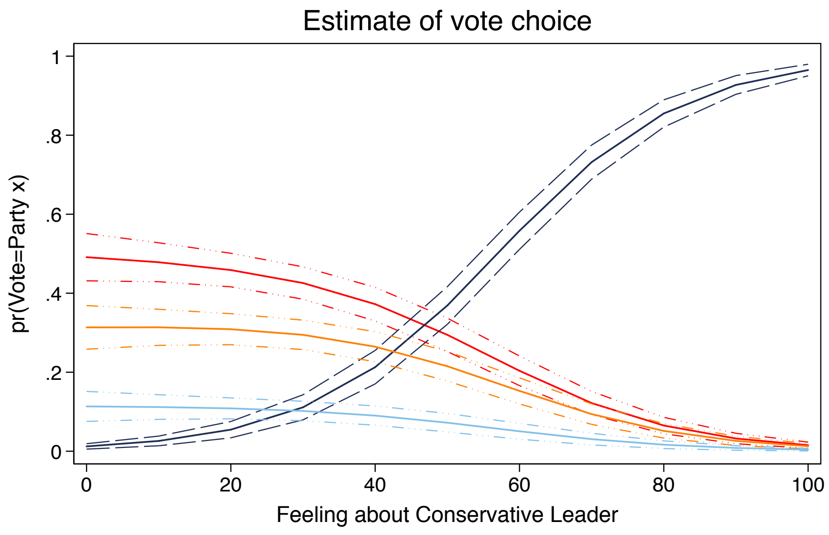
Afterwards, we need three informations in order to create the slopes. The first is in the first column of the results, being the predicted probability. These estimates will become our datapoints in the final graph. We also need two other informations: the lower and upper limits of the confidence interval so to create slopes refering to those intervals.

We start by typing `mat t=J(11,3,.)` to create a new `t` matrix composed of eleven rows and three columns, all being empty. We then create an `a` vector that consists of all the `_at` values from the first column. We then use a loop to extract everything automatically in eleven steps – one for each `_at` value – and store it in our `t` matrix. The first line extracts the first beta – specified by the `_b` argument – and stores it in first column of the matrix. The second and third step computes the lower and upper limit of the confidence interval by subtracting and adding to the beta 1.96 times its standard error – specified by the `_se` argument. This is the classic formula to compute confidence interval at a 95% level of certainty.

Finally, we reset the `t` matrix by appending the `a` vector to it. We then change the column names so they are more easily interpretable. And as a last step, we save the columns as variables, using the column names. We now have twelve variables – four variables for each party – each consisting of eleven rows, for the values from 0 to 100 by increments of 10. We can finally graph those slopes with the following syntax.

```
twoway  (line prob_party1 at_party1, lcolor(red) lpattern(solid)) /// party 1 set
(line lci_party1 at_party1, lcolor(red) lpattern(dash_3dot) lwidth(thin)) ///
(line uci_party1 at_party1, lcolor(red) lpattern(dash_3dot) lwidth(thin)) ///
///
(line prob_party2 at_party2, lcolor(dknavy) lpattern(solid) ) /// party 2 set
(line lci_party2 at_party2, lcolor(dknavy) lpattern(longdash) lwidth(thin)) ///
(line uci_party2 at_party2, lcolor(dknavy) lpattern(longdash) lwidth(thin))  ///
///
(line prob_party3 at_party3, lcolor(orange) lpattern(solid) ) /// party 3 set
(line lci_party3 at_party3, lcolor(orange) lpattern(dash_3dot) lwidth(thin)) ///
(line uci_party3 at_party3, lcolor(orange) lpattern(dash_3dot) lwidth(thin))  ///
///
(line prob_party4 at_party4, lcolor(eltblue) lpattern(solid) ) /// party 4 set
(line lci_party4 at_party4, lcolor(eltblue) lpattern(dash_3dot) lwidth(thin)) ///
(line uci_party4 at_party4, lcolor(eltblue) lpattern(dash_3dot) lwidth(thin))  ///
, legend(off)  ///
xtitle(Feeling about Conservative Leader) ytitle("pr(Vote=Party x)") ///
ti(Estimate of vote choice)
```

Figure 4: Better Visual Representation



This syntax might seem, yet again, complex. In fact, it is pretty straightforward. It is composed of only one command: `scatter` and multiple slopes. Hence, this syntax is one huge line of command, separated by breaks – identified by three slashes (`///`) – to make it more easy to read. As explained earlier, we are not creating four slopes but twelve: four slopes for the different parties and 8 slopes for the confidence intervals. Hence, I create the line with the `line` argument. Each line and its options are contained in brackets. I start by plotting the slope of the estimate, using `lcolor` to set the color and `lpattern` to choose a special design. For the confidence intervals, I also add the `lwidth` option so the graph is more easily read. I then add axis titles and a graph title.

The result can be seen in Figure 4. While it is not the most wonderful figure in the world, it is better than figure 3. But more importantly, the technique can be used with any models where you wish to overlay different slopes with `margins` results, such as multinomial logits, ordered logits or simply different logits.