Documentation for Version 2.1 of V++

Estimation software written by Vladislav Slanchev

Jared Ashworth

Duke University

Tyler Ransom

Duke University

Version of Documentation: April 7, 2016

1 Model Setup

This program estimates multi-state discrete choice models that can allow for selection and unobserved random components which have a factor structure, where the (random) factor is distributed either discrete, normal, or mixture of normals. The class of models which can be estimated are .... (Heckman, 1995?). The software also computes marginal effects for variables of interest in various outcome equations, conditional on the factor.

1.1 Equations

The general form of the model is a system of equations

$$\begin{aligned} d_{ijt} &= X_{it}\beta_j + \alpha_j \xi_{it} + \varepsilon_{ijt}, \\ y_{ijt} &= Z_{it}\delta_j + \kappa_j \xi_{it} + \nu_{ijt} \end{aligned}$$

where ξ_{it} is a vector of (possibly) time-varying random factors each distributed $N(0,1)$[1] and mutually independent; d_{ijt} is a categorical dependent variable; y_{ijt} is a continuous dependent variable; X_{it} and Z_{it} are matrices of regressors associated with d_{ijt} and y_{ijt}, respectively; β_j, δ_j, α_j and κ_j are vectors of parameters to be estimated (the latter two are called *factor loadings*); ε_{ijt} is an idiosyncratic shock assumed to be distributed standard Type I Extreme Value (mean Euler-Mascheroni constant[2] and standard deviation $\pi/\sqrt{6}$); and ν_{ijt} is a random shock assumed to be distributed mean-zero Normal with standard deviation σ_j.

This system of equations allows for estimation of a selection correction model[3] in the presence of unobserved heterogeneity. Unobserved heterogeneity is accounted for by the random factor(s) ξ_{it}, which are linked across outcome equations. In particular, the outcome variable y_{ijt} may only be observed for certain outcomes.

2 Program files

The majority of Vlad++ is done behind the scenes in the prok folder. This folder should be left alone, and a project folder should be created to call the program from this folder. To call the program, execute the following files in the following order:

1. `vladppadd.m` file (this must be modified to point to the location of the `prok` folder; called automatically in `regri.m` and `opt_rout.m`)
2. An m-file to import data into Matlab (named, e.g., `data_import.m`)
3. `descri.m` file (which declares matrix indices for model elements)
4. `regri.m` file (which declares exactly how each variable behaves in the model)
   (a) includes a line to read in the data file resulting from the script in 1. (named, e.g., `data_import.mat`)
5. `form_model5.m` file (which converts the model behavior information into data Matlab can read)
6. `form_input.m` file (which modifies the data for compatibility with C++ mex files)
   (a) `form_input_sh.m` file (only modifies how restrictions enter the data for compatibility with C++ mex files; requires `form_input.m` to have already been run and numerical intergration to be used)
7. `opt_rout.m` file (which calls the optimization routine)
8. `simple_marginaleff3.m` (which calculates marginal effects)

[1] Other distributions are also supported, including finite mixture, $N(\mu, \sigma^2)$, and mixture of normals
[2] The Euler-Mascheroni constant is approximately 0.5772.
[3] Originally proposed by Heckman (1979).

2.1 Software Compilation

If desired, users can re-compile all required mex files by running the file `mexer.m`. This re-compiles the C++ object files, sets the number of multi-processing threads, and then recreates the matlab mex files. This can be done for the files `prok2.cpp`, `prok3.cpp` and `marg.cpp`. Note that the version of matlab in the path directory must be correct. Also, the user can specify the number of multi-processing threads. It can be helpful to start at 1 and then choose higher numbers, normally numbers that are multiples of the number of processors. **Note that invocation of openmp has not been tested and has been known to cause stability issues with this software on certain servers**.

prok2.cpp

At the command line, type:

```
g++ -c -fPIC -O3 -mfpmath=sse -march=core2 -fopenmp -I/opt/matlabR2009b/extern/include
prok2_calc_one_2A.cpp
```

Note that you may have to add a line to the file prok2_calc_one2A.cpp (as well as the marg.cpp): `#include <string.h>` (also, prok2, prok3, and marg_main). This may have to happen on other files as well (prok3, prok3, marg_main).

In Matlab, type:

```
setenv('OMP_NUM_THREADS', '1');
mex prok2.cpp prok2_calc_one_2A.o CFLAGS="\$CFLAGS -fopenmp" LDFLAGS="\$LDFLAGS -fopenmp"
```

prok3.cpp

At the command line, type:

```
g++ -c -fPIC -O3 -mfpmath=sse -march=core2 -fopenmp -I/opt/matlabR2009b/extern/include
prok3_calc_one_A2.cpp
```

In Matlab, type:

```
setenv('OMP_NUM_THREADS', '8');
mex prok3.cpp prok3_calc_one_A2.o CFLAGS="\$CFLAGS -fopenmp" LDFLAGS="\$LDFLAGS -fopenmp"
```

marg.cpp

At the command line, type:

```
g++ -c -fPIC -O3 -mfpmath=sse -march=core2 -fopenmp -I/opt/matlabR2009b/extern/include
marg_main.cpp
```

In Matlab, type:

```
setenv('OMP_NUM_THREADS', '8');
mex marg.cpp marg_main.o CFLAGS="\$CFLAGS -fopenmp" LDFLAGS="\$LDFLAGS -fopenmp"
```

2.2 Model Declaration

Users declare the role of each variable in the model in `regri.m`. This file can be quite complicated, because the user needs to explicitly specify the relationship between dependent variables and endogenous regressors in each time period of the model. Details on how to accomplish this comprise the majority of the documentation (section §3).

2.3 Estimation

The optimization routine in `opt_rout.m` calls `fminunc` on a Mex function written in C++. Thus, optimization options such as convergence criteria and maximum iterations can be controlled via `fminunc`. Standard errors can also be

obtained by asking `fminunc` to report the numerical Hessian matrix upon completion of the optimization routine.[4]

2.4 Marginal Effects

Marginal effects are recovered after estimation in the file `simple_marginaleff3.m`. Details on how to customize marginal effects are found in section §5.

3 Model Specification

3.1 Basic Idea

The marginal effects in this model are computed by integrating out the unobserved heterogeneity distribution and simulating the model. In order to compute marginal effects for this complex model, it is necessary for the user to specify the exact relationships among all of the variables in the model. This is particularly important for variables that are endogenous (e.g. lagged dependent variables). Specifically, for each column of data used in estimation, the user must specify all of the roles which that column of data plays in the model. The file `regri.m` is where the user specifies these relationships.

3.2 Variables and Roles

The main tools used in the `regri.m` file are indices named `varnumber` and `rownumber`. The matrix that summarizes all of the model information is called `stvar` and has dimensions `max(rownumber)` by `numberrest` by `max(varnumber)`[5], where `num_dv` is the number of dependent variables in the model in each time period, and `numberrest` is the number of parameter restrictions in the model. So `stvar(r,n,v)` is the value for the `n`th flag of the `r`th row of the `v`th variable.[6]

varnumber corresponds to 1 column of data/1 variable

rownumber corresponds to each role that `varnumber` can play

Thus, for each `varnumber`, the user will have many `rownumbers` which indicate the different roles in the model played by that variable.

3.3 Types of variables

Variables are classified into five types (numbered 0-4 by the software, respectively):

0. simple regressor (predetermined, non-time-varying variable)
    - e.g. race, sex, IQ, mother's education
1. dependent variable
    - e.g. test score, decision to attend college
2. regressor (or part of a sum) resulting from a discrete choice — used only in `varnumbers` that are discrete choice outcomes
    - e.g. a worker choosing to work today raises his experience level tomorrow by 1 unit — currently this is the only possible law of motion for state variables.

[4] The developer remarks that the derivest_suite (MathWorks File Exchange) provides some more benefits and should be used instead of `fminunc` to get the numerical Hessian. The derivest_suite takes longer than `fminunc`, but may be more precise.

[5] Note that, if no restrictions are present, the second dimension has 27 elements (basically, the number of model elements in section 3.4.2).

[6] For the school-to-work project, it is $768 \times 72 \times 19,391$. `max(varnumber)` is artifically large because our numbering system creates gaps in the `varnumber` sequence. `max(rownumber)`=num_dv*T because non-time varying variables are only introduced once in the model for every outcome, with $T = 192$ and `num_dv`$= 4$

3. a variable which creates a condition for state dependent outcomes (mainly used for censoring purposes)

    - i.e. wages are only observed when an individual works, so one of the roles of the work choice is that it turns on the possibility for someone to receive a wage

4. a sum of endogenous or state variables — used to indicate that a right-hand-side `varnumber` is actually a state variable

    - i.e. work experience is a state variable, so its value will change if the worker's previous choice changes (e.g. in a counterfactual simulation setting)

3.4 Indices in the `stvar` matrix

As mentioned previously, the `stvar` matrix contains all information about the model. Rather than confusing the user with numbers for the index references, the software contains role names as pointers to indicate the location in `stvar` where information can be found. These names are listed for reference in section 3.4.2, but before going there, let's look at a simple example to illustrate what we've discusses so far.

3.4.1 Example

Let's go through an example in order to see how some of the main roles are implemented in `regri.m`. This excerpt shows the initialization of the (multinomial) choice variable in our model (with categories 1-3).

```
1  varnumber=t*100+name.Choice;
2  eval(['varval' num2str(varnumber) ' = [choice(:,2:3,t) choice(:,1,t)];']);
3  eval(['msvar'  num2str(varnumber) ' = 2*(missingFlag(:,1,t)==0)-1;']);
4  namevar{varnumber,1}='Pre HS Choice';
5  stvar(1,discrete,varnumber)=1;
6
7  rownumber=1; % Enters model as a DV
8  stvar(rownumber,present  ,varnumber)=1;
9  stvar(rownumber,dependent,varnumber)=1;
10 stvar(rownumber,numfac   ,varnumber)=1;
11 stvar(rownumber,numfac+1 ,varnumber)=0; %id of the factor
12 if t>1
13     stvar(rownumber,obscoeff  ,varnumber)=100+name.Choice;
14     stvar(rownumber,unobscoeff,varnumber)=100+name.Choice;
15 end
16 stvar(rownumber,numcond   ,varnumber)=1;
17
18 stvar(1,numrows,varnumber)=rownumber;
```

And below is a line-by-line explanation:

1. The `varnumber` is initialized as t times 100 + `name.Choice`, where t is the time period [this is part of a loop], 100 is simply a placeholder (since in any given time period there will be more than 10 varnumbers), and `name.Choice` is a variable that holds the number of the dependent variable within each time period.
2. `varval101`, for example, is the choice variable in period 1 for dependent variable 1. Notice that the base category is the last column of the matrix, and that the matrix is $N \times J$.
3. `msvar101`, for example, is a subsetting variable that indicates if an observation should be ignored. `msvar` should equal -1 for missing observations and 1 for non-missing observations. `msvar` can also take on values of 0, but this is only applicable for the missing data imputation capabilities of the software, which are normally not invoked.
4. `namevar` keeps track of the name of the variable, for convenience
5. `discrete` indicates that the dependent variable here is discrete
6. n/a
7. declaring `rownumber` 1
8. `present` is a quick way for the user to drop a variable (0 means variable is excluded, 1 means included)
9. `dependent` is where the user applies the roles listed in section 3.3. It always goes in `rownumber` 1 of any `varnumber`. Here, it is 1 because that is the code for a dependent variable
10. `numfac` lists the number of unobserved factors entering the right hand side of this dependent variable
11. the id of the factor always comes in element `numfac`+1 (factor id's begin with the number 0)
12. n/a
13. `obscoeff` is how the user pools observable coefficient estimates across time periods. In this example, we set all of coefficients for periods 2 through T equal to those of period 1; thus, the coefficients are constant across time
14. `unobscoeff` is the same as above, but for factor loadings
15. n/a
16. `numcond` is the number of conditions for variables resulting from switching modes
17. n/a
18. `numrows` tells the program how many `rownumbers` are in the given `varnumber`

3.4.2 Model Elements

Because one example can't cover all possibilities, below is an ordered list of model elements with a short explanation for each.

#	Flag Name	Explanation
Main Indices		
1	numrows	number of entries for that variable (must be assigned at the end of each varnumber)
2	present	indicates that the variable enters part of the model
3	idmodel	id of the outcome in which the variable is a regressor
4	dependent	0 = simple regressor (or part of a sum) 1 = dependent variable 2 = regressor (or part of a sum) resulting from discrete choice 3 = creating a condition for state dependent outcomes

		4 = sum of endogenous or state variables (could be ones excluding each other)
5	alter	if dependent==2 | 3 – relevant alternative
6	obscoeff	restrict all **observables'** coefficients to be the same as the ones for some other outcome (only for LHS variables) 0 = no constraint >0 = id number of the other LHS variable
7	unobscoeff	restrict all **unobservables'** coefficients to be the same as the ones for some other outcome (only for LHS variables) 0 = no constraint >0 = id number of the other LHS variable
8	varcoeff	restrict the **variance** estimate of the idiosyncratic shock to be the same as the one for some other outcome (only for LHS variables) 0 = no constraint >0 = id number of the other LHS variable -1 = no variance
9	discrete	indicates that the variable is discrete (must go in row 1)
10	interaction	if dependent!=1 (regressor) – varnumber for interaction, use same varnumber for quadratic term
11	multiplier	multiplier for the interaction
12	minival	if dependent==3 – minumum value for a condition to be satisfied
13	maxival	if dependent==3 – maximum value for a condition to be satisfied
14	auxvariable	id when auxiliary variable
15	margtype	type of outcome according to marginal effects definition 0 = final outcome 1 = continous continuing as a regressor 2 = discrete continuing as a regressor 3 = discrete leading to switching modes -1 = continuous leading to auxilary (sum) -2 = discrete leading to auxiliary (sum)
16	location	location if regressor
17	interaction	if dependent!=1 (regressor) – varnumber for interaction, use same varnumber for quadratic term (DUPLICATE INDEX)
18	idequation	id of the equation in which the variable goes
19	visitinter	if interaction visited – relevant location of the row where the variable with which it interacts stays
20	locnointer	location of the variable among the regressors of that outcome when not interacted
21	locincloutc	if interacted variable endogenous – the order of the interaction among the other outcomes for which it is a regressor - estimation
22	locxincloutc	if interacted variable endogenous – the order of the interaction among the other outcomes for which it is a regressor - marginal effects
23	average	average of the variable (for marginal effects)
24	chng	>0 = change for the marginal effects (must go in row 1) 0 = average/100
25	numfac	if dependent==1 – number of factors
Other Indices		
	numcond	Number of conditions that must hold for variable to be present. Used in conjunction with other variables that have dependent==3. (must go in row 1)

3.5 Parameter Constraints

The main principle with parameter constraints is that restrictions that affect a whole equation need to be put into `rownumber` 1 of the dependent variable, whereas restrictions on separate independent variables can be put on the

respective variables.

3.5.1 Pooling estimates across time

See `obscoeff`, `unobscoeff` and `varcoeff` in the table in section 3.4.2, as well as lines 13 and 14 in the example in section 3.4.1.

3.5.2 Specifying specific parameter constraints

If, in addition to constraints across time, a user wants to specify constraints on individual parameters, the following variables may need to be generated, depending on what type of restriction the user wishes to impose.

#	Flag Name	Explanation
Restriction indices		
	numberrest	number of restrictions being implemented in this `rownumber`. Called once per `rownumber`. (Note: the reference variable `restnum` separates restrictions for a given `rownumber`. There need to be `numberrest` calls of `restnum`, going from 1 to `numberrest`.
1	typeRest	type of restriction 0=equal to a fixed value 1=setting parameter(s) equal to other parameter(s) (Note: both variables must be introduced before the restriction in added; place restriction on second variable)
2	typePa	if dependent==1 – type of the parameter(s) 1=all observables for the whole alternative 2=single factor loading 3=all factor loadings for the whole alternative
3	numAltRestA	number of the alternative where the current parameter(s) appear(s) (1 if the outcome is continous)
4	parRestA	if dependent==1 – factor id for a restricted factor loading
if typeRest==0		
5	valRest	value to which the parameter is restricted
if typeRest==1		
6	numRegRestB	if dependent!=1 – `varnumber` of the other parameter that the current parameter is being equated to (required, even if same `varnumber`) if dependent==1 – `varnumber` of the other outcome that contains alternatives to equate to the parameters in `numAltRestA` of the current outcome (required, even if same `varnumber`)
7	numAltRestB	if dependent!=1 – number of the alternative where the other parameter appears (1 if the outcome is continous; required, even if same alternative) if dependent==1 – number of the alternative which contains the parameters to equate to the parameters in numAltRestA (1 if the outcome is continous; required, even if same alternative)
8	parRestB	if dependent!=1 – `rownumber` of numRegRestB that indicates outcome equation (required, even if same `rownumber`) if dependent==1 – factor id for a restricted factor loading (required)
9	sizeRest	number of possible cells for one restiction. All restriction indices are reserved for each restriction regardless of whether they are used or not.

3.5.3 Specifying specific parameter constraints on the parameter of an interacted variable

In the event that a resrtiction needs to be implemented for a parameter associated with an interaction, the restriction should be placed on the variable with the largest `varnumber`. As an example, if you interact `varnumber` 77 with `varnumber` 89, you should put the restriction only on `varnumber` 89.

3.6 Conditioning

Vlad++ allows the user to tell it the conditions for the variables to be turned on through `numcond`, which controls the censoring of the model. See the second to last row of the table in section 3.4.2.

3.6.1 Details

section 3.3 discusses the different types of dependent variables. One of these types is 3, indicating that the variable is creating a condition. When the condition holds, a 1 is passed to the receiving variable, else a 0 is passed. The receiving variable then assigns the number of conditions that need to occur to `numcond` in the first row. If the numbers add up, then the rest of the receiving variable is implemented. Otherwise, it is ignored for that observation.

There are many conditions that can be used, though the most useful will be associated with `minival`/`maxival` and with `alter`. The following example adds a dummy for high school graduation before the choice variable and a wage variable after the choice variable. Both uses of the conditioning statement are used here and are explained below in the next two subsections:

```
varnumber=t*100+name.gradHS;
eval(['varval' num2str(varnumber) ' = [gradHS(:,1,t)];']);
eval(['msvar'  num2str(varnumber) ' = 2*(missedInt(:,1,t)==0)-1;']);
namevar{varnumber,1}='gradHS';
stvar(1,discrete,varnumber)=1;

rownumber=1; % sum of previous degree status & whether gradutated
stvar(rownumber,present  ,varnumber)=1;
stvar(rownumber,dependent,varnumber)=4;

rownumber=2; % activating condition for risk set 1 - no high school diploma
stvar(rownumber,present  ,varnumber)=1;
stvar(rownumber,idmodel  ,varnumber)=t*100+name.Choices1;
stvar(rownumber,dependent,varnumber)=type.d.condition;
stvar(rownumber,minival  ,varnumber)=-0.05;
stvar(rownumber,maxival  ,varnumber)=0.05;

rownumber=3; % activating condition #1 for risk set 2 - high school diploma
stvar(rownumber,present  ,varnumber)=1;
stvar(rownumber,idmodel  ,varnumber)=t*100+name.Choices2;
stvar(rownumber,dependent,varnumber)=type.d.condition;
stvar(rownumber,minival  ,varnumber)=0.95;
stvar(rownumber,maxival  ,varnumber)=1.05;
stvar(1,numrows,varnumber)=rownumber;


varnumber=t*100+name.Choices1;
eval(['varval' num2str(varnumber) ' = [choice(:,2:3,t) choice(:,1,t)];']);
eval(['msvar'  num2str(varnumber) ' = 2*(missingFlag(:,1,t)==0)-1;']);
namevar{varnumber,1}='Pre HS Choice';
stvar(1,discrete,varnumber)=1;

rownumber=1; % Enters model as a DV
stvar(rownumber,present  ,varnumber)=1;
stvar(rownumber,dependent,varnumber)=1;
stvar(rownumber,numfac   ,varnumber)=1;
stvar(rownumber,numfac+1 ,varnumber)=0; %id of the factor
if t>1
    stvar(rownumber,obscoeff  ,varnumber)=100+name.Choices1;
    stvar(rownumber,unobscoeff,varnumber)=100+name.Choices1;
end
stvar(rownumber,numcond   ,varnumber)=1;

rownumber=2; % Creates condition for wage to be turned on; choice 2
stvar(rownumber,present ,varnumber)=1;
stvar(rownumber,idmodel ,varnumber)=t*100+name.Wage1;
stvar(rownumber,dependent,varnumber)=3;
stvar(rownumber,alter ,varnumber)=1;

stvar(1,numrows,varnumber)=rownumber;


varnumber=t*100+name.Wage1;
eval(['varval' num2str(varnumber) ' = [lnWage(:,:,t)];']);
eval(['msvar' num2str(varnumber) ' = 2*([missedInt(:,1,t)==0].*[working(:,1,t)==1])-1;']);

namevar{varnumber,1}='Wage';

rownumber=1; % Enters model as a DV
stvar(rownumber,present ,varnumber)=1;
```

3.6.2 `minival/maxival`

Oner main opportunity to condition is when a variable takes on a specific value, including dummy variables, i.e. an individual faces different choice sets depending on whether they have graduated high school or not. However, due to the way the program is structured, it is not as simple as saying whether a variable equals X. Rather, Vlad++ wants to know whether the variable is in $(X-\varepsilon, X+\varepsilon)$.

11. This `rownumber` gives the condition needed for an observation to be included in the first set of choices, aka those who have NOT graduated high school.
12. Indicates that this `rownumber` is `present` in the model
13. `idmodel` indicates that this `rownumber` is referring to the wage equation
14. `dependent` is set equal to 3, referencing that this `rownumber` is having this variable act as a condition for when an observation should enter the wage equation
15. First half of the `condition`, setting `minival` to $X-\varepsilon$ (-0.05), to allow for a test of whether the dummy for graduating high school is equal to 0.
16. Second half of the `condition`, setting `maxival` to $X+\varepsilon$ (0.05), to allow for a test of whether the dummy for graduating high school is equal to 0.

3.6.3 `alter`

You can use `alter` when you want an individual to enter an equation if a specific choice was made, i.e. the wage only enters the model when an individual chooses a working activity.

18. Adds a new `rownumber`
19. Indicates that this `rownumber` is `present` in the model
20. `idmodel` indicates that this `rownumber` is referring to the wage equation
21. `dependent` is set equal to 3, referencing that this `rownumber` is having this variable act as a condition for when an observation should enter the wage equation
22. Creates a condition for `alter` to be equal to 1. If this is true, then this sends a 1 to the wage equation. Else it sends a 0.
27. Adds a new `varnumber` for the wage equation
28. Reads in the wage data
29. Reads in dummies for missing wage data
30. Calls the variable "wage"
31. n/a
32. First (and only) `rownumber`
33. Indicates that this `rownumber` is `present` in the model
34. `dependent` is set equal to 1, which introduces the variable as a true dependent variable
35. No factors
36. n/a
37. if
38. if
39. if
40. n/a
41. Here is the conditioning statement. `numcond` must be equal to 1 for an observation to be included in the wage equation. This happens if the sum of all the conditions being sent to the wage variable are 1. In this, the only one is the one appearing on line 22. So if that is true, then the observation is included in the wage equation.

4 Unobserved Heterogeneity

When estimating the model with unobserved heterogeneity, the user has two options for integrating out the unobserved factors: *(i)* Simulated Maximum Likelihood Estimation (SMLE); and *(ii)* Gauss-Hermite Quadrature. As a rule of thumb, Gauss-Hermite Quadrature will run more quickly than SMLE if the dimension of factors is two or less.

4.1 Factor Distributions

The default distribution for the unobserved factor is standard normal. However, the user can also estimate model specifications in which the factor is distributed discrete, non-standard normal, or a mixture of normals.

4.2 Numerical Integration

As mentioned above, the user can estimate the model using SMLE or Gauss-Hermite Quadrature. Within SMLE, the user can specify that the random draws be a random sequence, a Halton Sequence, or a Scrambled Halton Sequence.

The user can also specify how many random draws should be made, or how many quadrature points should be used (if using Gauss-Hermite Quadrature)

4.3 Specifying Unobserved Heterogeneity

To inform the software that a factor should be included in a given dependent variable equation, the user should issue the following in `rownumber` 1 of the dependent variable:

```
rownumber=1; % Enters model as a DV
stvar(rownumber,present ,varnumber)=1;
stvar(rownumber,dependent,varnumber)=1; %this is a dependent variable
stvar(rownumber,numfac ,varnumber)=1; %number of factors included in this equation
stvar(rownumber,numfac+1 ,varnumber)=0; %ids of those factor
```

Because the software runs with C++ (hence the clever name V++), the factor id's begin numbering from 0 instead of 1. Hence, in the simple case with only one factor in the model, the factor id is always 0.

To control options related to estimation with random factors, the user adjusts the following auxiliary variables found at the top of the `regri.m` file.

Flag Name	Explanation
typeEst	type of numerical integration.
	0=SMLE
	1=Gauss-Hermite Quadrature
typeRand	type of simulated draws (for SMLE)
	1=random (use if Quadrature)
	2=Halton Sequence
	3=Scrambled Halton Sequence
factorType	distribution of the random factor
	1=standard normal
	2=discrete
	3=normal with non-standard mean and/or variance
	4=mixture of normals
factorEval	number of simulation draws (SMLE) or quadrature points (Gauss-Hermite)
	must be a positive integer; set to 1 for discrete factor
factorPoints	number of distributions in the mixture, or number of mass points in the discrete distribution
	must be an integer larger than 1
drawFileCount	number of files in which the random draws are stored
	used only for SMLE; otherwise 1 by default

5 Marginal Effects

Marginal effects are implemented by running the file `simple_marginaleff3.m` after estimation. This file reads the model estimates (from `opt_rout.m`) and evaluates marginal effects for all parameters in a dependent variable equation of interest (e.g. wages at time period 3). The default is to evaluate at the means of all the regressors and the mean(s) of the unobserved factor(s). It also allows for user-control over how any simulation is performed.

`simple_marinaleff3.m` is a useful setup file, which calls `simple_marinaleff4.m`, which actually gets the marginal effects. Only one dependent variable equation can be sent to `simple_marinaleff4.m` at a time. However, a list of equations of interest can be looped over to easily generate the desired output.

Most importantly, remember that the reason the setup is so complicated is to allow for marginal effects calculations. Incorrect specifications in the `regri.m` file may not cause errors until you get here, or they can calculate the incorrect effect.

6 Example

The following is an example of how to set up the data matrices and model specification for a simple multinomial logit with one wage equation. The example data is `y97_test.dta`, a dataset from the NLSY97.

The example will go through how to estimate parameters in the choice model (the multinomial variable) as well as how to estimate parameters in the wage equation. The dataset consists of annual observations on 601 males from age 14 until 28 (or survey attrition). Because of the overlapping birth cohorts in the survey, $T = 11$. There are two outcome equations (a discrete choice and a wage).

6.1 Setup

6.1.1 Stata equivalent

The model can be estimated in Stata (using "long" panel format instead of "wide" panel format) by issuing the following commands:

1. `mlogit activity black hispanic AFQT m_AFQT Mhgc m_Mhgc c.age##c.age experSchOnly experSchWork experPT experFT experOther, base(1)`
2. `reg log_wage black hispanic c.age##c.age experSchOnly experSchWork experPT experFT experOther`

6.1.2 Required Model Statements

For this example, the model specification is quite complex, even though the Stata implementation is simple.

`See regri.m`

6.1.3 Introducing Constraints

For this example, let us impose the following constraints:

1. `experPT` has the same coefficient in choice alternatives 2 and 5
2. all parameters are the same in choice alternatives 3 and 4
3. `m_Mhgc` is constrained to be 0 in choice alternative 2
4. `black` is constrained to be -0.2 in choice alternative 5
5. `experPT` is constrained to be equal to `experFT` in the wage equation

Stata In Stata these constraints would be invoked as

1. `constraint 1 [2=5]: experPT`
2. `constraint 2 [3=4]: _cons black hispanic AFQT m_AFQT Mhgc m_Mhgc age c.age#c.age experSchOnly experSchWork experPT experFT experOther`
3. `constraint 3 [2] : m_Mhgc`
4. `constraint 4 [5] : black = -.2`
5. `constraint 5 experFT = experPT`

The constrained model would then be estimated via the following commands:

1. `mlogit activity black hispanic AFQT m_AFQT Mhgc m_Mhgc c.age##c.age experSchOnly experSchWork experPT experFT experOther, base(1) constraints(1/4)`
2. `cnsreg log_wage black hispanic c.age##c.age experSchOnly experSchWork experPT experFT experOther, constraints(5)`

Vlad++ The following code accomplishes these constraints in the Vlad++ program:

`See regri_restrict.m`

7 FAQ

Q: How are my regressors ordered?

A: Regressors are ordered by `varnumber` in each time period.

Q: What's the policy on dependent?

A: `dependent` should always be declared in rownumber 1 of any `varnumber`. Moreover, every `varnumber` needs a `dependent` declaration.

Q: My "type B" restrictions won't work. Any ideas?

A: Make sure that you are following the order: `typeRest`, then `numAltRestA`, then `numRegRestB`, then `numAltRestB`, then `parRestB`.

7.1 Debugging

Users may run into issues when using the software. Below is a list of steps that can be taken to rectify such scenarios.

Problem: The likelihood at convergence different than what you know to be true in other software programs

Solution: If the likelihood is quite different, it means that either the data are different, or the model is different. See next problem.

Problem: Data may not have been properly read in.

Solution: Check to make sure by comparing summary stats, as in the example files. Make sure the `msvar` matrix is correctly specifying missing values. Note that the software drops either converts `NaN` values to 0 (if an exogenous variable), imputes them or assigns them to be 1 (if endogenous), or drops them altogether (if dependent variable), so it is of the utmost importance that the user make sure that the data is the same.

Problem: A simple OLS regression isn't coming up with the same answers as other software.

Solution: A simple way of checking this is by running OLS on `model_all.mat` (generated by `regri.m`). e.g. `b=reg1(dmydata1==1)\dv1(dymdata1==1)`. This is also a quick way to make sure that variables are in the same order and have the same values as expected by the user.

Other avenues to explore when debugging:

- **Typo in `varnumbers` and/or `rownumbers`:** Check that they are sequentially numbered (note: this is not required, but is helpful)
- `numcond` **incorrectly specified:** `numcond` controls the censoring of the model and always needs to be in `rownumber` 1, and needs to list the number of "`dependent==3`" conditions are associated with an equation. For example, if the wage only enters the model when choice = 2, then you need to put `numcond=1` in the first rownumber of both the choice *and* the wage.