EUROMOD
Advanced Functions

Fiscal Policy Analysis Unit
Joint Research Centre
European Commission

20 July 2022

Introduction

- Who is this course for?

This course assumes that you have already completed the standard EUROMOD training and ideally already have some hands-on experience with using EUROMOD.

- Structure of the course

In this course we will try to go through many of the advanced functionalities of EUROMOD in a short time. As such, although there will be some examples for each function, it will not be possible to get into the full detail of all available parameters of each function. The main goal of the course is to gain an understanding of what is possible within EUROMOD, and to know where to look for detailed help once you need it.

Course Outline

- Section 1 – Input data manipulation
- Section 2 – Looping
- Section 3 – Aggregate functions
- Section 4 – Other advanced functions

Section 1 – Input data manipulation

- KeepUnit
- DropUnit
- AddHHMembers
- Uprate
- DefInput

KeepUnit / DropUnit

Both the KeepUnit and DropUnit functions allow for dropping observations from the original Input Data. The difference is that with KeepUnit you specify a condition for which observations to keep (and drop everything else), while with DropUnit you specify a condition for which observations to drop (and keep everything else).

The condition can be based on any formula (e.g. “dag<15” or “drgn=3”), and you can also select to drop whole TUs, in which case you can specify who needs to match the condition (which works exactly like Who_Must_Be_Elig).

Policy	*Grp/No*	**SL_demo**	*Comment*
KeepUnit		on	
keep_cond		{il_earns>0}#1 & {il_pension=0}#1 & {dag>14} & {dag<66}	keep household if there is at least one person
#_level	1	individual_sl	with positive earnings, not receiving pensions
keep_cond_who		one	and aged between 15 and 65
TAX_UNIT		household_sl	

AddHHMembers

The AddHHMembers function is designed to add new observations to the original Input Data. Specifically it allows adding new members to existing households only.

You can chose to add Partners, Children or Other members, depending on which you will be expected to specify the condition based on the Partner, Parents and Household respectively. You are allowed to specify the values of any number of (existing) variables for the new member, the default being 0 for all. The function will also automatically set the correct values for idPartner or idFather & idMother depending on what type of member you are adding.

Policy	*Grp/No*	**SL_demo**	*Comment*
AddHHMembers		**on**	
Add_Who	1	Child	
ParentCond	1	{IsPartner#2} & {dgn=0} & {dag>=30} & {dag<=35} & {nDepChildrenInTu#2<=1}	
#_Level	2	tu_sben_family_sl	
dgn	1	1	

Uprate

The Uprate function is used to uprate monetary variables to the price level of the current system year. Most commonly it is used by applying a single factor to a single variable, something which is very well explained in the standard EUROMOD course. However it also allows for:

- aggregate uprating for variables that are composed of other variables
- use of regular expressions to uprating multiple variables at once
- multiple factors per variable based on conditions
- use of the DBYearVar parameter to allow for uprating datasets that include observations from multiple years

Policy	*Grp/No*	**SL_demo**	*Comment*
Uprate		on	
dataset		sl_demo_a1	
RegExp_Def	1	yem*	uprate all employment variables ...
RegExp_Factor	1	cpi	... with cpi
RegExp_Def	2	x1*	uprate expenditure variables of the first COICOP group ...
RegExp_Factor	2	cpi	... with cpi
RegExp_Def	3	x[2-9]*	uprate all other expenditure variables ...
RegExp_Factor	3	1.3	... with 1.3

Policy	*Grp/No*	**SL_demo**	*Comment*
Uprate		on	
dataset		sl_demo_a1	
yem	1	1.02	employment income ...
Factor_Condition	1	{drgn1=1}	... in region 1
yem	2	1.025	
Factor_Condition	2	{drgn1=3}	... in region 2
yem	3	1.03	
Factor_Condition	3	{drgn1=3}	... in region 3

DefInput

The DefInput function allows for merging one or more variables from a file into the original Input Data. It can be used in either Input or Lookup mode:

- When used in Input mode, the file is expected to have two or more variables as columns. In this case, you will have to chose one matching variable (e.g. idhh), and will import the values of all other variables.
- When used in Lookup mode, only one variable is merged, based on the combination of the values of two variables. In this case the file is expected to be a table where row headers are values of the reference variable, and column headers are values of a second reference variable.

Policy	**SL_demo**	*Comment*
DefInput	on	
Path	c:\SomeFolder	
File	SomeInputFile.txt	
RowMergeVar	dag	
ColMergeVar	dgn	
InputVar	sin01_s	
DoRanges	yes	

File SomeInputFile.txt:

age/gender	0	1
10	100	101
30	300	301
80	800	801
200	2000	2001

Extract of Output:

idperson	drg	dgn	sin01_s
101	31	1	801
102	30	0	300
103	2	0	100
201	99	1	2001
…	…	…	…

Policy	**SL_demo**	*Comment*
DefInput	on	
Path	c:\SomeFolder	
File	SomeInputFile.txt	
RowMergeVar	dgn	

File SomeInputFile.txt:

dgn	sin01_s	sin02_s	$SomeVar
0	4711	111	1234
1	1147	999	9876

Extract of Output:

idperson	dgn	sin01_s	sin02_s	$SomeVar
101	0	4711	111	1234
102	1	1147	999	9876
…	…	…	…	…
123401	0	4711	111	1234

Section 2 – Looping

- Loop
- UnitLoop
- Store
- Restore

Loop / UnitLoop

The functions Loop and UnitLoop allow you to repeat part of the tax-benefit calculations. This can be very useful when calculating marginal tax rates, or making budget neutral reforms. The main difference between the two, is that:

- Loop will repeat the calculations for all individuals, based on a general condition, or for a given number of iterations.
- UnitLoop will repeat the calculations for each eligible individual of a TU.

Both functions allow you to specify the policies or functions that will be included in the loop with two ways, inclusive or exclusive (e.g. First_Func vs Start_After_Func).

Policy	**SL_demo**	*Comment*
Loop	on	
loop_id	turn	
first_func	ExampleStore_loopstart	assuming that policy is called ExampleStore
last_func	ExampleStore_loopend	
num_iterations	3	

Policy	**SL_demo**	*Comment*
UnitLoop	on	
loop_id	unit	loop over the tax-benefit calculations
start_after_pol	tudef_sl	as often as there are persons with
stop_before_pol	output_std_sl	positive employment income
elig_unit	individual_sl	assuming that this policy is placed at the beginning of the loop
elig_unit_cond	yem>0	(i.e. after tudef_sl)

Store / Restore

The functions Store and Restore are mainly designed to support the Loop & UnitLoop functions. They serve two purposes:

- To backup the values of specific variables and to set them back to their original (or previous) values
- To generate new suffixed variables to store the values of specific variables after each iteration

They can be used on single variables, or on incomelists, in which case they will store separately the values of each variable in that incomelist.

Store	**on**	
post_loop	turn	store variable yem before each iteration of the loop
var	yem	
...	**on**	
...	...	do something with yem
Restore	**on**	
postloop	turn	set yem back to its value before the first iteration
iteration	1	

Store	**on**	
postloop	turn	store the results of each turn of the loop
var	tin_s	store model-calculated income tax
il	il_sic	store model-calculated insurance contrib. (tscee_s+tscse_s)
DefOutput	**on**	
file	example_out	
var	tin_s_turn1	contains value of tin_s after the 1st iteration of the loop
var	tin_s_turn2	contains value of tin_s after the 2nd iteration of the loop
var	tin_s_turn3	contains value of tin_s after the 3rd iteration of the loop
var	tin_s_turn	contains value of tin_s after the last iteration of the loop
var	tscee_s_turn1	contains value of tscee_s after the 1st iteration of the loop
var	tscee_s_turn2	contains value of tscee_s after the 2nd iteration of the loop
var	tscee_s_turn3	contains value of tscee_s after the 3rd iteration of the loop
var	tscee_s_turn	contains value of tscee_s after the last iteration of the loop
var	tscse_s_turn1	contains value of tscse_s after the 1st iteration of the loop
var	tscse_s_turn2	contains value of tscse_s after the 2nd iteration of the loop
var	tscse_s_turn3	contains value of tscse_s after the 3rd iteration of the loop
var	tscse_s_turn	contains value of tscse_s after the last iteration of the loop
il	il_sic_turn1	contains value of il_sic after the 1st iteration of the loop
il	il_sic_turn2	contains value of il_sic after the 2nd iteration of the loop
il	il_sic_turn3	contains value of il_sic after the 3rd iteration of the loop
il	il_sic_turn	contains value of il_sic after the last iteration of the loop

Section 3 – Aggregate functions

- DefIl
- IlVarOp
- CumulativeSum
- Totals
- IlArithOp

DefIl

The DefIl function lets you define incomelists that contain a number of variables, so that you can easily use their sum. This is something well explained in the standard EUROMOD course. One of the less known features of DefIl, that can be very handy especially in combination with other functions in this section, and hence is worth mentioning in this course, is the ability to specify the included variables by means of regular expressions.

DefIl		switch	Baseline parameter list: ad valorem excises
Name		il_tco_base_v	
Warn_If_No...		no	
RegExp_Def	1	$tco_base_v_[0-9]+	
RegExp_Factor	1	+	
DefIl		switch	Commodity list: il_xs = income shares (all goods)
Name		il_xs	
Warn_If_No...		no	
RegExp_Def	1	xs[0-9]+	
RegExp_Factor	1	+	

IlVarOp

The IlVarOp function allows for performing basic arithmetic operations to components of an incomelist. The operations can be multiplication, addition or setting negatives to zeroes, and they can be applied to either all of the variables in the incomelist, or to a single variable based on its value (min, max, minpos).

Policy	**SL_demo**	*Comment*
ILVarOp	on	
operator_il	il_earns	all variables within the incomelist il_earns
operand	101%	are increased by 1%
operation	mul	parameter could be skipped as 'mul' is default
sel_var	all	parameter could be skipped as 'all' is default

Policy	**SL_demo**	*Comment*
ILVarOp	on	
operator_il	il_earns	the value of the variable yot is added
operand	yot	to the smallest variable within il_earns
operation	add	
sel_var	min	

CumulativeSum

The CumulativeSum function allows for calculating the cumulative sum of a given variable or incomelist over all observations. It can sort the observations based on any number of nested variables and incomelists, and the sum can be weighted or unweighted, absolute or relative with any combination of the two pairs.

CumulativeSum		**on**	
SortingVar	1	eq_dispy	sort households according to equivalised disposable income
SummingVar		dwt	cumulatively sum the weight
SummingAbsolute		no	as a relative value
OutputVar		cumRelWt	
TAX_UNIT		household_sl	
CumulativeSum		**on**	
SortingVar	1	eq_dispy	sort households according to equivalised disposable income
SummingVar		eq_dispy	cumulatively sum the equivalised disposable income
SummingAbsolute		no	as a relative value
SummingWeighted		yes	weighted
OutputVar		cumRelInc	
TAX_UNIT		household_sl	
ArithOp		**on**	
Formula		(dwt/$sum_dwt) * (cumRelWt - cumRelInc) * 2	the gini formula calculation
Output_Var		gini	
TAX_UNIT		household_sl	
Totals		**on**	
Varname_Sum		$sum	
Agg		gini	calculate the total gini ($sum_gini)
TAX_UNIT		household_sl	

Totals

The Totals function allows for the calculation of aggregates of variables or incomelists over all observations or a selected subgroup. It can perform calculations such as Sum, Mean, Median, Min, Max, Count, Deciles etc. using either the weighted or unweighted values of the selected variable or incomelist.

Policy	**SL_demo**	*Comment*
Loop	**on**	**loop function could be placed wherever desired**
loopid	budgneut	
first_pol	sic_sl	
last_pol	cb_reform_sl	assumes that the present policy is called cb_reform_sl
breakcond	{$totnew_ils_dispy<$totold_ils_dispy}	exit condition: tax increase covers raised expenditure
Totals	**on**	**compute total disposable income before the reform**
run_cond	{loopcount_budgneut=1}	
varname_sum	$totold	
agg_il	ils_dispy	
TAX_UNIT	individual_sl	
Totals	**on**	**compute total disp. inc. after each increase of the tax rate**
varname_sum	$totnew	
agg_il	ils_dispy	
TAX_UNIT	individual_sl	

IlArithOp

The IlArithOp function is designed to perform more complex operations between the components of different analogue incomelists, and write the results into the variables of an analogue output incomelist. It requires a base incomelist whose variables it will iterate through and try to match them in the other incomelists based on the given prefixes/suffixes. It will then generate a new incomelist and a new set of variables to fill it and the values of the new variables will be calculated using the full flexibility of the Formula field.

Policy	*Grp/No*	**SL_demo**	*Comment*
DefIL		**on**	**define price incomelist**
Name		il_prices	
pApples		+	
pCoffee		+	
pBread		+	
DefIL		**on**	**define quantity incomelist**
Name		il_quantities	
qApples		+	
qCoffee		+	
qBread		+	
IlArithOp		**on**	**calculate expenditure**
Base_ILName		il_prices	
Base_Prefix		p	all vars in il_prices must start with 'p'
ILName	1	il_quantities	
Prefix	1	q	all vars in il_quantities must start with 'q'
Formula		BASE_IL_COMPONENT * IL_COMPONENT[il_quantities]	all vars of il_prices are multiplied by the respective vars in il_quantities
Out_ILName		il_expenditure	incomelist holding the result variables
Out_Prefix		x	all vars in il_expenditure start with 'x'

Section 4 – Other advanced functions

- Advanced formulas
- DefConst
- Scale
- CallProgramme
- Break

Advanced formulas

The formulas (and conditions) in any EUROMOD function are a very flexible tool that can do much more than simple arithmetic operations:

- They allow for a number of different numerical operators such as: +, -, *, /, \, ^, %
- They allow for a number of different logical operators such as: !, &, |, <, >, =, <=, >=, !=
- Conditions and Booleans can be directly evaluated within the formula to a 0/1 value:

  E.g. “IsChild * benefit”, “(yem<threshold)*topup”
- They can contain inline functions like Excel: ABS(1), MIN(2), MAX(2), LN(1), LOG(1/2), LOG10(1), EXP(1), SQRT(1), CEILING(1), FLOOR(1), ROUND(1/2), POWER(2), IF(3)
- Specifically the IF inline function allows for extra flexibility that can substitute whole functions:

  E.g. “if(yem<threshold, topup, 0)”, or “if(b=0, 0, a/b)”

DefConst

The DefConst function allows for the definition of constants. It is one of the most common functions which is also covered in the standard EUROMOD course. However one of the relatively new additions to it, namely the Condition parameter, is less known. This gives the DefConst function an increased flexibility in defining conditional constants that would otherwise require the use of much bigger code.

Policy	*Grp/No*	**SL_demo**	*Comment*
DefConst		on	
$MinWage		1000#m	
$ChBen		200#m	
$UnivCredit		500#m	
Condition	1	drgn1 = 8	London
$MinWage	1	1200#m	
$ChBen	1	250#m	
$UnivCredit	1	600#m	
Condition	2	drgn1 >10	Rural UK
$MinWage	2	800#m	
$ChBen	2	180#m	
$UnivCredit	2	450#m	

Scale

The Scale function allows the user to scale monetary variables and monetary parameters. In the case of variables, it will effect all existing monetary variables and the effect is cumulative. In the case of parameters, it will affect all the parameters following this function and the effect is not cumulative.

Policy	*Grp/No*	**SL_demo**	*Comment*
Scale		**on**	
FactorVariables		1.5	
FactorParameter		2	

CallProgramme

The CallProgramme function allows you to call an external application. Note that the function is only available under Windows as it uses platform specific code.

Policy	**SL_demo**	*Comment*
CallProgramme	**on**	
Programme	Excel.exe	
Argument	&Output\SomeExcelWorkbook	
RepByEMPath	&	

Break

The Break function allows the user to break the run at any point inside the spine. This function is only intendent to be used during the development stage of a system to help the developer locate and fix bugs in such system. When called, it will immediately stop execution, ignoring everything that follows, but still producing a detailed output of all the existing variables and incomelists, and optionally all TU info.

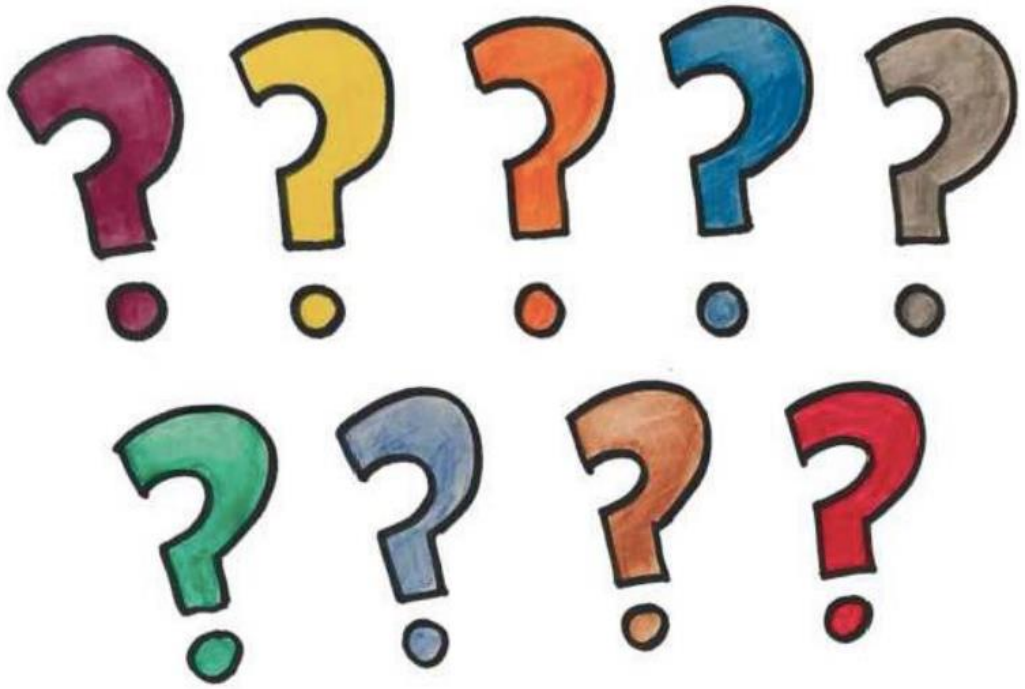
Policy	*Grp/No*	**SL_demo**	*Comment*
Break		**on**	
ProduceOutput		yes	
OutputFileName		C:\MyFiles\debugging.txt	
ProduceTUinfo		yes	

Final Remarks

- Always consult the EUROMOD Help. If unsure, use the context-sensitive keys F5, F6 to see the function description and parameters respectively. It is recommended that you also have a look at the keyboard shortcuts page.
- Keep in touch!
  - Visit our website https://euromod-web.jrc.ec.europa.eu/
  - Subscribe to the JRC EUROMOD Newsletter https://euromod-web.jrc.ec.europa.eu/about/newsletters
  - Send us feedback JRC-EUROMOD@ec.europa.eu

European Commission

Questions and discussion



Thank you

JRC-EUROMOD@ec.europa.eu

European Commission