On the empirical identification of time preferences in discrete choice dynamic programming models. Documentation.

Sofia Badini

21 February 2021

CONTENTS

This project is my Master's thesis, presented at the Department of Economics at the University of Bonn in February 2021.

CHAPTER

ONE

INTRODUCTION

This thesis assesses the practical identification of the time preference parameters in a discrete choice dynamic model of occupational choice, after introducing empirically-motivated exclusion restrictions that influence the size of the agents' choice set.

In particular, agents may or may not face future employment restrictions that depend deterministically on their educational choices: Future-oriented agents take the restrictions into account when deciding on their level of education.

The same identification strategy is used in a setting with exponential discounters and in a setting where agents discounts quasi-hyperbolically and are completely naïve.

Time preference parameters in structural models of dynamic discrete choices are underidentified, which is especially problematic for counterfactual analysis, since information on time preferences are needed to make any statement about the behavioral response of the agents to a policy intervention.

In the literature, both theoretical arguments for identification and empirical identification strategies exploit, with mixed results, variables that leave the per-period utility function unaffected, while being relevant to the agents' decisions. The intuition is that comparing the behavioral response of similar agents to different (expected) futures may reveal information on their time preferences.

This project uses the template by Hans-Martin von Gaudecker [3].

CHAPTER

TWO

MODEL SPECIFICATIONS

The directory *src.model_specs* contains yaml and csv files with model specifications.

In particular, it contains the `params` and `options` needed to simulate a discrete choice dynamic programming model with the open-source software respy, see Gabler and Raabe (2020) [1].

CHAPTER

THREE

DATA SIMULATION

Documentation of the code in *src.data_simulation*.

3.1 Simulation of main datasets

Simulate main datasets for the two different specifications of the model based on [2] (KW94).

3.2 Simulation of counterfactual data

Get counterfactual prediction for 2000 US dollars tuition subsidy for different parametrization of the model with hyperbolic discounting and choice restrictions based on Keane and Wolpin (1994) [2].

Looking at the bivariate distribution of the time preference parameters, it seems that many combinations of beta (present bias) and delta (discount factor) are compatible with the empirical data. Therefore, I study the extent to which the counteractual predictions of these competing parametrizations differ.

simulate_life_cycle_data (*params*, *options*)
Generate simulated life-cycle data (100 DataFrame).

Parameters

- **params** (*pd.DataFrame*) – DataFrame containing model parameters.
- **options** (*dict*) – Dictionary containing model options.

Returns List of pd.DataFrames.

simulate_life_cycle_df (*params*, *options*, *sim_seed*, *sol_seed*, *col_to_keep*)
Simulate life cycle dataset, store choices and wages (mean and std).

Parameters

- **params** (*pd.DataFrame*) – DataFrame containing model parameters.
- **options** (*dict*) – Dictionary containing model options.
- **sim_seed** (*int*) – Seed for simulation.
- **sim_seed** – Seed for solution.
- **col_to_keep** (*list*) – Columns of the simulate data from which to compute relevant moments (choice and wages).

Returns pd.DataFrame.

CHAPTER

FOUR

ANALYSIS

Documentation of the code in *src.analysis*. This is the core of the project.

4.1 Assessing the validity of the choice restrictions

Regressions to check effect of restrictions.

4.2 Method of Simulate Moments

Empirical moments to compute for exponential and hyperbolic model specification respectively.

Compute moments and weighting matrix for empirical datasets.

Given observed moments and weighting matrix in *OUT_ANALYSIS*, "msm_estimation", generate values of Method of Simulated moment criterion function for different value of selected parameters, keeping all other parameters fixed.

The goal is to study whether the resulting univariate distributions (in particular, those of the time-preference patameters) have a minimum around the true parameter value and are reasonably smooth.

get_univariate_distribution(*params*, *params_base*, *crit_func*, *steps*)

Get values of criterion function for different values of selected parameters, keeping all other *params* fixed.

Parameters

- **params** (*pd.DataFrame*) – Model parameters.
- **params_base** (*pd.DataFrame*) – Dataframe of parameters whose values will be varied to compute the criterion function. Need to have column "upper" and "lower" specifying respectively the maximum and minimum parameter's value for which the criterion function is computed.
- **crit_func** (*func*) – Crtierion function.
- **steps** (*numpy.ndarray*) – Step size. Determine the number of parameters' values for which the criterion function is evaluated.

Returns dict

Given observed moments and weighting matrix in *OUT_ANALYSIS*, "msm_estimation", generate values of Method of Simulated Moments criterion function for combinations of discount factor and present bias values.

The goal is to study the bivariate distribution of the time preference parameters around the combination of true parameter values.

get_bivariate_distribution(*params*, *crit_func*, *grid_delta*, *grid_beta*)

Compute value of criterion function for differen value fo discount factor and present bias parameter.

Parameters

- **params** (*pd.DataFrame*) – DataFrame containing model parameters.
- **crit_func** (*dict*) – Dictionary containing model options.
- **grid_delta** (*np.array*) – Values of discount factor.
- **grid_beta** (*np.array*) – Values of present-bias parameter.

Returns pd.DataFrame

4.3 Counterfactual analysis

Compute effect of 2000 US dollar tuition subsidy predicted by different model specifications.

compute_subsidy_effect(*data_without_subsidy*, *data_with_subsidy*, *moment*)

Compute effect of subsidy subsidy on a certain moment (i.e. years of education).

Parameters

- **data_without_subsidy** (*list of dataset*) – List of dataset, without subsidy subsidy.
- **data_with_subsidy** (*list of dataset*) – List of dataset, with subsidy subsidy. Must have the same length of *data_without_subsidy*.
- **moment** (*func*) – Moment of interest.

Returns numpy.ndarray

compute_subsidy_effect_on_experience(*data_without_subsidy*, *data_with_subsidy*)

Compute effect of subsidy subsidy on occupational experience (i.e. education, occupation A and occupation B).

Parameters

- **data_without_subsidy** (*list of dataset*) – List of dataset, without subsidy subsidy.
- **data_with_subsidy** (*list of dataset*) – List of dataset, with subsidy subsidy. Must have the same length of *data_without_subsidy*.
- **mom_func** (*func*) – Function to compute moment of interest.

Returns dict

CHAPTER

FIVE

VISUALIZATION AND RESULTS FORMATTING

Documentation of the code in *src.final*.

5.1 Visualization module

Visualization module.

compare_choice_probabilities(*df*, *policy_dict*, *color_dict*, *label_dict*)
Plot choice probabilities, comparing behavior of restricted and unrestricted agents.

Parameters

- **df** (*pd.DataFrame*) – Dataframe with choice probabilities.
- **policy_dict** (*dict*) – Dictionary, from policies to policies.
- **color_dict** (*dict*) – Dictionary, from choices to colors.
- **label_dict** (*dict*) – Dictionary, from choices to labels.

Returns Matplotlib figure.

get_custom_cmap()
Generate custom cmap.

plot_average_predicted_choices(*df_emp*, *data_dict*, *moments_dict*, *style_dict*, *plot_observed=True*)
Plot moments of interest for list of pd.DataFrame.

Parameters

- **df_emp** (*pd.DataFrame*) – Empirical dataset.
- **data_dict** (*dict*) – Dictionary of data.
- **moments_dict** (*dict*) – Dictionary of moments to plot.
- **style_dict** (*dict*) – Dictionary to style graphical elements of the plot.
- **plot_observed** (*bool*) – Whether the moment of the empirical datasets should be visualized in the plot.

Returns Matplotlib figure.

plot_counterfactual_predictions(*data_dict*, *style_dict*, *title_dict*, *mom*, *ylabel*)
Plot average predicted effect of tuition subsidy in each period, plus simulation noise, on specified moment.

Parameters

- **data_dict** (*dict*) – Dictionary of data.
- **style_dict** (*dict*) – Dictionary to style graphical elements of the plot.
- **title_dict** (*dict*) – Dictionary of titles.
- **mom** (*str*) – Moment to plot. Can be "A", "B", or "Edu".
- **ylabel** (*str*) – Label of y-axis.

Returns Matplotlib figure.

plot_heatmap2d(*df*, *threshold*)

Plot 2D heatmap from criterion values for combinations of different discount factor and present bias values.

Parameters

- **df** (*pd.DataFrame*) – Data. Need to have three columns: "beta", "delta" and "val".
- **threshold** (*float*) – Value of criterion that makes colormap switch.
- **title** (*string*) – Title of plot.

Returns Matplotlib figure.

plot_heatmap3d(*df_heatmap*)

Plot 3D heatmap.

plot_ridgeplot(*df*, *groups*, *value_col*, *xlim=(-0.2, 1)*, *figsize=(16, 10)*, *col_palette=None*, *bw=0.015*, *kernel='epanechnikov'*, *first_group_is_complete=True*, *ticker_value=0.25*, *hspace=-0.25*)

Create overlapping densities plot.

Parameters

- **df** (*pd.DataFrame*) – Data frame containing a value column (name passed as argument), and a group column (must be called "group"), that classifies each row to a single group. Note that the group must be categorical.
- **groups** (*list*) – List of unique group names contained in df["group"]. The order of this list determines the order of the subplots.
- **value_col** (*str*) – Name of value column.
- **xlim** (*tuple*) – x-axis limits.
- **figsize** (*tuple*) – Figure size.
- **col_palette** (*list*) – Color palette. If None, defaults to itertools.cycle(seaborn.cubehelix_palette(10, rot=-.3, light=.7)).
- **bw** (*float*) – Bandwidth for the density estimation kernel.
- **first_group_is_complete** (*bool*) – If after getting the unique ordered groups the first group represents the complete sample.
- **ticker_value** (*float*) – Float passed to matplotlib.ticker.MultipleLocator.
- **hspace** (*float*) – Float passed to grid_spec.GridSpec().update.

Returns The ridge plot figure.

Return type fig (matplotlib.figure.Figure)

plot_univariate_distribution(*results_dict*, *title_dict*, *params_base*, *adjustment=False*, *pad=0.1*, *set_title=True*)

Plot parameters' univariate distribution.

Parameters

- **results_dict** (*dict*) – Dictionary of results, where parameters' names are the keys and values of criterion function are the values.
- **title_dict** (*dict*) – Dictionary mapping parameters to plot title.
- **params_base** (*pd.DataFrame*) – DataFrame of parameters to be plotted.
- **adjustment** (*bool*) – Whether to adjust the plot yticklabels. Default is False.
- **pad** (*float*) – Where to position the new yticklabel, if *adjustment* is True. Default is 0.1.
- **set_title** (*bool*) – Whether to set title for the plot. Default is True.

5.2 Figures

5.2.1 Conditional choice probabilities (p. 19)

Plot conditional choice probabilities in main datasets.

5.2.2 Method of Simulated Moment criterion (p. 24-28, 42, 43)

Plot criterion values univariate distribution for selected parameters.

Plot heatmap criterion.

5.2.3 Counterfactual predictions (p. 29-31, 33)

Plot counterfactual predictions.

5.3 Tables

Generate tables. In particular: Summary statistics of empirical moments (p. 20), choice restrictions regression results (p. 21), counterfactual predictions (p. 32).

CHAPTER

SIX

THESIS

Documentation of the code in *src.paper*.

- `main.tex` contains the final thesis.

CHAPTER

SEVEN

CODE LIBRARY

Code shared across different modules.

Functions shared across modules, mostly useful to compute moments from the 'empirical' datasets or other quantities needed to obtain the criterion function (e.g. the weighting matrix).

calc_choice_probabilities(*df*)
Compute choice probabilities by period.

calc_restricted_choice_probabilities(*df*)
Compute choice probabilities for agents under one choice restriction.

calc_restricted_wage_distribution(*df*)
Compute per-period mean and std of wages for agents under one choice restriction.

calc_unrestricted_choice_probabilities(*df*)
Compute choice probabilities for unrestricted agents.

calc_unrestricted_wage_distribution(*df*)
Compute mean and standard deviation of wages, by period.

calc_very_restricted_choice_probabilities(*df*)
Compute choice probabilities for agents under two choice restrictions.

calc_very_restricted_wage_distribution(*df*)
Compute per-period mean and std of wages for agents under two choice restrictions.

calc_wage_distribution(*df*)
Compute mean and standard deviation of wages, by period.

get_weighting_matrix(*data*, *empirical_moments*, *calc_moments*, *n_bootstrap_samples*, *n_observations_per_sample*, *replace_missing_weights=None*)
Compute a diagonal weighting matrix for estimation with MSM. Weights are the inverse bootstrap variances of the observed sample moments.

Parameters

- **data** (*pandas.DataFrame*) – Dataframe containing individual observations. Must contain index named "Identifier" by which observations are sampled.
- **empirical_moments** (*dict*) – Dictionary containing empirical moments in the form of pandas.DataFrame or pandas.Series.
- **calc_moments** (*dict*) – Dictionary containing moment functions.
- **n_bootstrap_samples** (*int*) – Number of samples that should be boostrapped.

- **n_observations_per_sample** (*int*) – Observations per bootstrap sample.
- **replace_missing_weights** (*None or float*) – Can be used to replace missing weights with a float value. If none, in cases where where weights are computed to be missing/infinite (i.e. if variances are 0), weights are set to zero.

Returns

Diagonal weighting matrix with dimensions RxR where R denotes the number of moments.

Return type numpy.array

CHAPTER

EIGHT

REFERENCES

BIBLIOGRAPHY

[1] Janos Gabler and Tobias Raabe. Respy - a framework for the simulation and estimation of eckstein-keane-wolpin models. https://respy.readthedocs.io/en/latest/, 2020. URL: https://github.com/OpenSourceEconomics/respy.

[2] Michael P. Keane and Kenneth I. Wolpin. The solution and estimation of discrete choice dynamic programming models by simulation and interpolation: monte carlo evidence. *The Review of Economics and Statistics*, 76(4):648–672, 1994. URL: http://www.jstor.org/stable/2109768, doi:10.2307/2109768.

[3] Hans-Martin von Gaudecker. Templates for reproducible research projects in economics. https://doi.org/10.5281/zenodo.2533241, 2019.

PYTHON MODULE INDEX

INDEX