cupidpython

Release 1.0.0

Bernard Salanie

Nov 11, 2021

CONTENTS

CHAPTER

ONE

MATCHING WITH PERFECTLY TRANSFERABLE UTILITY

1.1 README

`CupidPython` contains Python code to compute the equilibrium and to estimate the parameters in separable one-to-one, bipartite matching models with perfectly transferable utility—see Galichon and Salanié 2021 for a general study.

It contains:

- in `ipfp_solvers.py`: implementations of the Iterative Projection Fitting Procedure (IPFP) algorithm for several variants and extensions of the Choo and Siow 2006 model and for a class of nested logit models.
- in `poisson_glm.py`: a function that estimates the original version of the Choo and Siow model (homoskedastic, with singles) for a semilinear surplus, using Poisson GLM.
- in `min_distance.py`: a function that applies a minimum distance estimator to separable, semilinear models with a user-supplied entropy function.

I also created a Streamlit app that demonstrates the basic Choo and Siow model. You can find a (hopefully) working version of the app here.

This package is released under the MIT license. I hope it is useful to you.

Bernard Salanié — Nov 7, 2021

bsalanie at columbia.edu

1.2 Module choo_siow

1.3 Module choo_siow_gender_heteroskedastic

1.4 Module choo_siow_heteroskedastic

1.5 Module cupid_streamlit

1.6 Module entropy

1.7 Module ipfp_solvers

1.8 Module matching_utils

1.9 Module min_distance_utils

Utility programs used in *min_distance.py*.

class min_distance_utils.**MDEResults**(*X: int*, *Y: int*, *K: int*, *number_households: int*, *estimated_coefficients: numpy.ndarray*, *varcov_coefficients: numpy.ndarray*, *stderrs_coefficients: numpy.ndarray*, *estimated_Phi: numpy.ndarray*, *test_statistic: float*, *test_pvalue: float*, *ndf: int*, *parameterized_entropy: Optional[bool] = False*)

Bases: object

The results from estimation and testing.

K: int

X: int

Y: int

estimated_Phi: numpy.ndarray

estimated_coefficients: numpy.ndarray

ndf: int

number_households: int

parameterized_entropy: Optional[bool] = False

stderrs_coefficients: numpy.ndarray

test_pvalue: float

test_statistic: float

varcov_coefficients: numpy.ndarray

1.10 Module min_distance

1.11 Module model_classes

1.12 Module monte_carlo_simul

1.13 Module nested_logit

1.14 Module nests_parameters

Nests and nest parameters for our two-level nested logit 0 is the first nest, all other nests and nest parameters are type-independent

1.15 Module poisson_glm

1.16 Module utils

This module contains some utility programs used by the package.

utils.**ScalarFunctionAndGradient**
Type of f(v, args, gr) that returns a scalar value and also a gradient if gr is True

alias of Callable[[numpy.ndarray, List, Optional[bool]], Union[float, Tuple[float, numpy.ndarray]]]

utils.**bs_error_abort**(*msg: str = 'error, aborting'*)
Report error and exits with code 1

Parameters **msg** – specifies the error message

Returns nothing

utils.**bs_name_func**(*back: int = 2*)
Get the name of the current function, or further back in the stack

Parameters **back** – 2 for the current function, 3 for the function that called it, etc

Returns the name of the function requested

utils.**check_gradient_scalar_function**(*fg: Callable[[numpy.ndarray, List, Optional[bool]], Union[float, Tuple[float, numpy.ndarray]]]*, *p: numpy.ndarray*, *args: List*, *mode: str = 'central'*, *EPS: float = 1e-06*) → Tuple[numpy.ndarray, numpy.ndarray]
Checks the gradient of a scalar function.

Parameters

- **fg** – should return the scalar value, and the gradient if its *gr* argument is *True*
- **p** – where we are checking the gradient
- **args** – other arguments passed to *fg*
- **mode** – "central" or "forward" derivatives

- **EPS** – the step for forward or central derivatives

Returns the analytic and numeric gradients

utils.**der_nppow**(*a: numpy.array*, *b: Union[int, float, numpy.array]*) → numpy.array
evaluates the derivatives in a and b of element-by-element a^b

Parameters

- **a** (*np.array*) –
- **b** (*Union[int, float, np.array]*) – if an array, should have the same shape as *a*

Returns a pair of two arrays of the same shape as *a*

utils.**describe_array**(*v: numpy.ndarray*, *name: str = 'The array'*) → collections.namedtuple
Descriptive statistics on an array interpreted as a vector

Parameters

- **v** – the array
- **name** – its name

Returns a *DescribeResult* namedtuple

utils.**npexp**(*a: numpy.ndarray*, *deriv: bool = False*, *bigx: float = 30.0*, *verbose: bool = False*) →
Union[numpy.ndarray, Tuple[numpy.ndarray, numpy.ndarray]]
C^2 extension of $\exp(a)$ above *bigx*

Parameters

- **a** – a Numpy array
- **deriv** – if *True*, the first derivative is also returned
- **bigx** – an upper bound
- **verbose** – whether diagnoses are printed

Returns upper bound:math:*exp(a)* C^2-extended above *bigx*, with its derivative if *deriv*

Return type bigx

utils.**nplog**(*a: numpy.ndarray*, *deriv: bool = False*, *eps: float = 1e-30*, *verbose: bool = False*) →
Union[numpy.ndarray, Tuple[numpy.ndarray, numpy.ndarray]]
C^2 extension of $\ln(a)$ below *eps*

Parameters

- **a** – a Numpy array
- **deriv** – if *True*, the first derivative is also returned
- **eps** – a lower bound
- **verbose** – whether diagnoses are printed

Returns $\ln(a)$ C^2-extended below *eps*, with its derivative if *deriv*

utils.**npmaxabs**(*a: numpy.ndarray*) → float
The maximum absolute value in an array

Parameters **a** – the array

Returns $\max|a|$

utils.**nppow**(*a: numpy.ndarray*, *b: Union[int, float, numpy.ndarray]*, *deriv: bool = False*) → Union[numpy.array, Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]]

Evaluates a^b element-by-element

Parameters

- **a** – a Numpy array
- **b** – if an array, it should have the same shape as *a*
- **deriv** – if *True*, the first derivatives wrt *a* and *b* are also returned

Returns an array of the same shape as *a*, and if *deriv* is *True*, the derivatives wrt *a* and *b*

utils.**nprepeat_col**(*v: numpy.ndarray*, *n: int*) → numpy.ndarray

Creates a matrix with *n* columns, all equal to *v*

Parameters

- **v** – a vector of size *m*
- **n** – the number of columns requested

Returns a matrix of shape *(m, n)*

utils.**nprepeat_row**(*v: numpy.ndarray*, *m: int*) → numpy.ndarray

Creates a matrix with *m* rows, all equal to *v*

Parameters

- **v** – a vector of size *n*
- **m** – the number of rows requested

Returns a matrix of shape *(m, n)*

utils.**print_stars**(*title: Optional[str] = None*, *n: int = 70*) → None

Prints a starred line, or two around the title

Parameters

- **title** – an optional title
- **n** – the number of stars on the line

Returns nothing

utils.**test_matrix**(*x: numpy.ndarray*, *fun_name: Optional[str] = None*) → Tuple[int, int]

Tests that *x* is a matrix; aborts otherwise

Parameters

- **x** – a potential matrix
- **fun_name** – the name of the calling function

Returns the shape of *x* if it is a matrix

utils.**test_vector**(*x: numpy.ndarray*, *fun_name: Optional[str] = None*) → int

Tests that *x* is a vector; aborts otherwise

Parameters

- **x** – a potential vector
- **fun_name** – the name of the calling function

Returns the size of *x* if it is a vector

PYTHON MODULE INDEX

INDEX