ipfp_python

Release 1.0.0

Bernard Salanie

Apr 22, 2021

CONTENTS

CHAPTER

ONE

MODULE ESTIMATE_CS_FUVL

Estimation of the Choo and Siow 2006 model: in its original version (homoskedastic with singles).

We minimize the $F(u, v, \lambda) - \hat{\mu} \cdot \Phi^{\lambda}$ function of Galichon–Salanie (2020, Proposition 5.)

`estimate_cs_fuvl.`**`estimate_cs_fuvl`**(*muxy: numpy.ndarray*, *nx: numpy.ndarray*, *my: numpy.ndarray*, *bases: numpy.ndarray*) $\rightarrow$ scipy.optimize.optimize.OptimizeResult

this estimates the parameters and equilibrium utilities in a semilinear homoskedastic Choo-Siow model.

Parameters

- **muxy** (*np.ndarray*) – the numbers of matches in each *(x,y)* cell, a *(X,Y)* matrix
- **nx** (*np.ndarray*) – the numbers of men in each x cell, a *X*-vector
- **my** (*np.ndarray*) – the numbers of women in each *y* cell, a *Y*-vector
- **bases** (*np.ndarray*) – the values of the K basis functions in each cell, a *(X,Y,K)* array

Returns

a *scipy.optimize.OptimizeResult* object *resus*. *resus.x* has the estimates of u, v, and λ in that order:

- u_x the expected utility of men of type *x*
- v_y the expected utility of women of type *y*
- λ_k the coefficient of basis function *k*

CHAPTER

TWO

MODULE IPFP_SOLVERS

Implementations of the IPFP algorithm to solve for equilibrium and do comparative statics in several variants of the Choo and Siow 2006 model:

- homoskedastic with singles (as in CS 2006)
- homoskedastic without singles
- gender-heteroskedastic: with a scale parameter on the error term for women
- gender- and type-heteroskedastic: with a scale parameter on the error term for women

each solver, when fed the joint surplus and margins, returns the equilibrium matching patterns, the adding-up errors on the margins, and if requested (gr=True) the derivatives of the matching patterns in all primitives.

ipfp_solvers.**ipfp_hetero_solver**(*Phi*, *men_margins*, *women_margins*, *tau*, *tol=1e-09*, *gr=False*, *verbose=False*, *maxiter=1000*)

solve for equilibrium in a in a gender-heteroskedastic Choo and Siow market

given systematic surplus and margins and a scale parameter dist_params[0]

Parameters

- **Phi** (*np.array*) – matrix of systematic surplus, shape (ncat_men, ncat_women)
- **men_margins** (*np.array*) – vector of men margins, shape (ncat_men)
- **women_margins** (*np.array*) – vector of women margins, shape (ncat_women)
- **tau** (*float*) – a positive scale parameter for the error term on women
- **tol** (*float*) – tolerance on change in solution
- **gr** (*boolean*) – if True, also evaluate derivatives of muxy wrt Phi
- **verbose** (*boolean*) – prints stuff
- **maxiter** (*int*) – maximum number of iterations
- **dist_params** (*np.array*) – array of one positive number (the scale parameter for women)

Returns

- (muxy, mux0, mu0y) the matching patterns
- marg_err_x, marg_err_y the errors on the margins
- and the gradients of (muxy, mux0, mu0y) wrt (men_margins, women_margins, Phi, dist_params[0]) if gr=True

`ipfp_solvers.`**`ipfp_heteroxy_solver`**(*Phi*, *men_margins*, *women_margins*, *sigma_x*, *tau_y*, *tol=1e-09*, *gr=False*, *maxiter=1000*, *verbose=False*)

solve for equilibrium in a in a gender- and type-heteroskedastic Choo and Siow market

given systematic surplus and margins and a scale parameter dist_params[0]

Parameters

- **`Phi`** (*`np.array`*) – matrix of systematic surplus, shape (ncat_men, ncat_women)
- **`men_margins`** (*`np.array`*) – vector of men margins, shape (ncat_men)
- **`women_margins`** (*`np.array`*) – vector of women margins, shape (ncat_women)
- **`sigma_x`** (*`np.array`*) – an array of positive numbers of shape (ncat_men)
- **`tau_y`** (*`np.array`*) – an array of positive numbers of shape (ncat_women)
- **`tol`** (*`float`*) – tolerance on change in solution
- **`gr`** (*`boolean`*) – if True, also evaluate derivatives of muxy wrt Phi
- **`verbose`** (*`boolean`*) – prints stuff
- **`maxiter`** (*`int`*) – maximum number of iterations

Returns

- (muxy, mux0, mu0y) the matching patterns
- marg_err_x, marg_err_y the errors on the margins
- and the gradients of (muxy, mux0, mu0y) wrt (men_margins, women_margins, Phi, dist_params) if gr=True

`ipfp_solvers.`**`ipfp_homo_nosingles_solver`**(*Phi*, *men_margins*, *women_margins*, *tol=1e-09*, *gr=False*, *verbose=False*, *maxiter=1000*)

solve for equilibrium in a Choo and Siow market without singles

given systematic surplus and margins

Parameters

- **`Phi`** (*`np.array`*) – matrix of systematic surplus, shape (ncat_men, ncat_women)
- **`men_margins`** (*`np.array`*) – vector of men margins, shape (ncat_men)
- **`women_margins`** (*`np.array`*) – vector of women margins, shape (ncat_women)
- **`tol`** (*`float`*) – tolerance on change in solution
- **`gr`** (*`boolean`*) – if True, also evaluate derivatives of muxy wrt Phi
- **`verbose`** (*`boolean`*) – prints stuff
- **`maxiter`** (*`int`*) – maximum number of iterations

Returns

- muxy the matching patterns, shape (ncat_men, ncat_women)
- marg_err_x, marg_err_y the errors on the margins
- and the gradients of muxy wrt Phi if gr=True

`ipfp_solvers.`**`ipfp_homo_solver`**(*Phi*, *men_margins*, *women_margins*, *tol=1e-09*, *gr=False*, *verbose=False*, *maxiter=1000*)

solve for equilibrium in a Choo and Siow market

given systematic surplus and margins

Parameters

- **Phi** (*np.array*) – matrix of systematic surplus, shape (ncat_men, ncat_women)
- **men_margins** (*np.array*) – vector of men margins, shape (ncat_men)
- **women_margins** (*np.array*) – vector of women margins, shape (ncat_women)
- **tol** (*float*) – tolerance on change in solution
- **gr** (*boolean*) – if True, also evaluate derivatives of muxy wrt Phi
- **verbose** (*boolean*) – prints stuff
- **maxiter** (*int*) – maximum number of iterations

Returns

- (muxy, mux0, mu0y) the matching patterns
- marg_err_x, marg_err_y the errors on the margins
- and the gradients of (muxy, mux0, mu0y) wrt (men_margins, women_margins, Phi) if gr=True

CHAPTER THREE

MODULE IPFP_UTILS

some utility programs used by ipfp_solvers

ipfp_utils.**der_npexp**(*arr: numpy.array*, *bigx: float = 30.0*, *verbose: bool = False*) → numpy.array
derivative of C^2 extension of $\exp(a)$ above *bigx*

Parameters

- **arr** (*np.array*) – a Numpy array
- **bigx** (*float*) – upper bound

Returns derivative of $\exp(a)$ C^2-extended above *bigx*

ipfp_utils.**der_nplog**(*arr: numpy.array*, *eps: float = 1e-30*, *verbose: bool = False*) → numpy.array
derivative of C^2 extension of $\ln(a)$ below *eps*

Parameters

- **arr** (*np.array*) – a Numpy array
- **eps** (*float*) – lower bound

Returns derivative of $\ln(a)$ C^2-extended below *eps*

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

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

ipfp_utils.**describe_array**(*v: numpy.array*, *name: str = 'v'*)
descriptive statistics on an array interpreted as a vector

Parameters

- **v** (*np.array*) – the array
- **name** (*str*) – its name

Returns the *scipy.stats.describe* object

ipfp_utils.**npexp**(*arr: numpy.array*, *bigx: float = 30.0*, *verbose: bool = False*) → numpy.array
C^2 extension of $\exp(a)$ above *bigx*

Parameters

- **arr** (*np.array*) – a Numpy array

- **bigx** (*float*) – upper bound

Returns $\exp(a)$ C^2-extended above *bigx*

`ipfp_utils.`**`nplog`**(*arr: numpy.array*, *eps: float = 1e-30*, *verbose: bool = False*) → numpy.array

C^2 extension of $\ln(a)$ below *eps*

Parameters

- **arr** (*np.array*) – a Numpy array
- **eps** (*float*) – lower bound

Returns $\ln(a)$ C^2-extended below *eps*

`ipfp_utils.`**`npmaxabs`**(*arr: numpy.array*) → float

maximum absolute value in an array

Parameters **arr** (*np.array*) – Numpy array

Returns a float

`ipfp_utils.`**`nppow`**(*a: numpy.array*, *b: Union[int, float, numpy.array]*) → numpy.array

evaluates a**b element-by-element

Parameters

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

Returns an array of the same shape as *a*

`ipfp_utils.`**`nprepeat_col`**(*v: numpy.array*, *n: int*) → numpy.array

create a matrix with *n* columns equal to *v*

Parameters

- **v** (*np.array*) – a 1-dim array of size *m*
- **n** (*int*) – number of columns requested

Returns a 2-dim array of shape *(m, n)*

`ipfp_utils.`**`nprepeat_row`**(*v: numpy.array*, *m: int*) → numpy.array

create a matrix with *m* rows equal to *v*

Parameters

- **v** (*np.array*) – a 1-dim array of size *n*
- **m** (*int*) – number of rows requested

Returns a 2-dim array of shape *(m, n)*

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

prints a starred line, or two around the title

Parameters

- **title** (*str*) – title
- **n** (*int*) – number of stars on line

Returns nothing

PYTHON MODULE INDEX

e

i

INDEX