# Assyria Documentation

***Release 0.1***

**Hortacsu**

**Dec 05, 2017**

# CONTENTS:

CHAPTER

# ONE

# ESTIMATION

**class** estimate.**Estimate**(*build_type*, *lat=(36*, *42)*, *lng=(27*, *45)*)

**export_results**(*varlist*)
varlist is in jhwi format: (zeta, useless, long_known, long_unknown, lat_known, lat_unknown, a) Exports zeta.csv, coordinates.csv, cities.csv, simulation.csv

**fetch_dist**(*lat_guess*, *lng_guess*, *full_vars=False*)
Wrapper

**full_to_short_i**()
returns the indices to select short varlist from full varlist

**gen_data**(*len_sim*, *perturb*, *rank=None*, *full_vars=False*)
rank: int. Process number in parallelized computing. Returns simulation dataframe sorted by objective value

**get_best_result**(*results*)
results: pd.DataFrame. It is the output of the parallelized execution. returns the row with minimum objective function value.

**get_bounds**(*constr*, *full_vars=False*)
Returns (lb, ub), where lb and ub are lists for the bounds.

**get_coordinate_pairs**(*lat_guess*, *lng_guess*, *full_vars=False*)
full_vars: bool. If True, the known coordinates are included as variables of the objective and gradient.

This is an alternative implementation of the fetching distance process,

Leverages the fact that the iticount data is sorted according to id_jhwi_j first, then by id_jhwi_i, and the coordinates are sorted according to id_jhwi.

**get_errors**(*varlist*, *full_vars=False*)
varlist = np.array([zeta, alpha, lat_guess, lng_guess]).

Returns the difference of data and model trade shares.

**get_size**(*zeta*, *alpha*, *distances*, *scale_kanes=False*, *theta=4.0*)

**Returns the fundamental size of cities:** Size_i proportional to L_i T_i^(1/theta)

**get_size_variance**(*varlist*, *scale_kanes=False*, *var_type='white'*)
Applies Delta Method to get the variance-covariance matrix of the city size estimates.

Return the variance-covariance matrix of sizes.

**get_variance**(*varlist*, *var_type='white'*, *full_vars=False*)
Computes the variance-covariance matrix of the estimators according to the white formula. (see paper)

**get_variance_gmm**(*varlist*, *full_vars=False*)
Computes the variance-covariance matrix of the estimators according to the GMM formula. (see notes.pdf)

**haversine_approx**(*coord_i*, *coord_j*)
coord_i, coord_j: np.array. 2 columns, len(iticount) rows. First column (column 0) is latitude, second column (column 1) is longitude.

Returns the approximation of the Haversine formula described in the estimation section of the paper.

**initial_cond**(*len_sim=None*, *perturb=None*, *full_vars=False*)
len_sim: int. Specifies the number of draws to take. perturb: float. Specifies a percentage deviation from the default initial value.

Returns default initial condition if perturb is not specified, and an array of perturbed values of dimension (len_sim, numvars)

**output_to_jhwi**(*output*)
Returns the initial value to evaluate the MATLAB objective function.

**replace_id_coord**(*constr*, *drop_wahsusana=False*)
constr: pd.DataFrame. Specifies upper and lower bounds for coordinates of cities.

Replaces the city id with its corresponding latitudes and longitudes in the constraints datasets.

**resolve**(*result*)
result: pd.DataFrame. Output of self.get_best_result

Recursively digs into the coordinates results if the maximum number of iterations was reached. Otherwise it returns the best solution.

**s_ij_model**(*zeta*, *alpha*, *distances*)
zeta: float. The distance elasticity of trade. alpha: np.array. One for each importing city. distances: np.array. Contains distances between all j, i pairs, excluding j, j pairs.

Idea: cast elements as matrix, add over axis=0, repeat (number of cities - 1), divide elements by this new 1-dim array.

Returns np.array: the model-predicted trade shares

**simulate_contour_data**(*varlist*, *size=20000*, *var_type='white'*, *full_vars=False*)
varlist is taken to be the point estimate vector.

Gets the var_type variance matrix using varlist and simulates size draws from a normal distribution with mean varlist and variance.

**solve**(*x0*, *constraint_type='static'*, *full_vars=False*)
x0: list. It is the initial value. constraint_type: str. One of 'static' or 'dynamic'. Returns a one-row dataframe with optimization information.

**sqerr_sum**(*varlist*, *full_vars=False*)
Returns the value of the objective function given the data and the model trade shares.

**tile_nodiag**(*arr*)
arr: np.array. A 1-dim array of length self.num_cities.

Returns an array repeating arr the number of times given by self.num_cities, but extracting value in index j on repetition j.

example: If arr = np.array([1, 2, 3]) then self.tile_nodiag(arr) returns np.array([2, 3, 1, 3, 1, 2]).

# INDEX