Documentation

This app allows conducting the Equality Equivalence Test (EET) as introduced by Kerschbamer (2013, 2015) in different variants and parameterizations by simply altering the documented variables in `config.py`. As a ready-made application, the EET app taps the full potential of oTree, implying straightforward setup and usage, seamless integration into existing projects, responsive graphical design, multi-language support, and automated testing.

Installation

To install the app to your local *oTree* directory, copy the "eet" folder to your *oTree* project and extent the session configurations in your `settings.py` at the root of the *oTree* directory by something like

```
SESSION_CONFIG = [
   ...
   {
      'name': 'eet',
      'display_name': 'Equality Equivalence Test',
      'num_demo_participants': 2,
      'app_sequence': ['eet'],
   },
   ...
]
```

Please note that global settings as `REAL_WORLD_CURRENCY_CODE`, `USE_POINTS`, as well as `SESSION_CONFIG_DEFAULTS` (including `participation_fee` and `real_world_currency_per_point`) are – as for all *oTree* apps – specified in *oTree*'s `settings.py` rather than the application itself but do affect the display of currency figures as well as the calculations of payoffs and amounts to pay.

Setup

To set up the task, alter pre-defined variables in the file `config.py` at the root of the app's directory match your desired features. Any combination of the variables described below is operable. By that means, several different variations of the task are easily implementable. The following variables can be specified:

Task Parametrization

`e` (integer/float field):

`e` determines the test's locus of the equal-material-payoff allocation $(m,o) = (e,e)$. Note that option 'right' implies a payoff (in points or currency units specified in `settings.py`) equal to `e` for both players in all binary decisions.

`g` (integer/float field):

`g` is the test's 'gap' variable characterizing the vertical distance to (e,e). The 'other' person's payoff in option 'left' is determined by $(e+g)$ for the x-list and $(e-g)$ for the y-list. Note that `g` is restricted to be strictly smaller than `e` (to prevent negative payoffs).

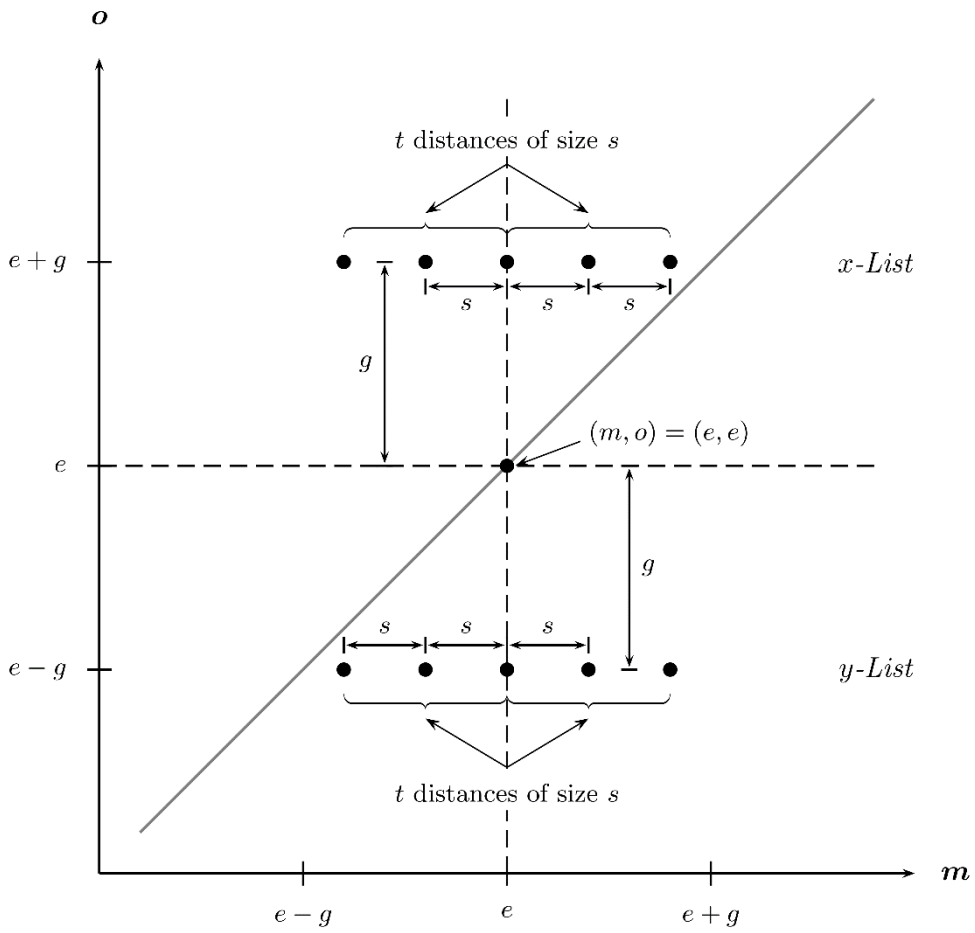
`s` (integer/float field):

`s` is the test's 'step size' variable characterizing the horizontal distance between two adjacent allocations. That is, the 'my' payoff varies in steps of size `s` around the locus payoff `e`.

`t` (integer field):

`t` is the test's 'test size' variable determining the number of steps (of size `s`) which are made to the left and to the right starting from the point just above or below the equal-payoff allocation $(m,o) = (e,e)$. `t` is restricted to be smaller or equal to *g/s* and must be larger or equal to 1.

The test parametrization, i.e., the way in which the values assigned to `e`, `g`, `s`, and `t` translate into the material payoff allocations in the equality equivalence test, is graphically illustrated in the figure below.



List Variations and Modifications of Parametrization

`asymmetric_s` (Boolean field):

`asymmetric_s = True` replaces the symmetric default step size between two adjacent decisions to increase the power of the test to discriminate between selfish and different variants of non-selfish behavior. Asymmetric step-sizes imply that step sizes are smaller at the center but larger at the borders. The specific functional form to determine asymmetric steps can be modified in `models.py:65`. By default, the implemented function implies that the most left/right 'my' payoff remains identical to the symmetric test version but 'my' payoffs in between double for all adjacent allocations.

`asymmetric_t` (Boolean field) and **`a`** (integer field):

`asymmetric_t = True` replaces the symmetric default test size specification by extending the x-list to the left and the y-list to the right in order to examine whether subjects put more weight on the material payoff of the passive person than on their own material payoff. `a` determines how many choices are added on each list. Note that `asymmetric_t = True` implies that the number of choices per list increases from $n = (2t + 1)$ to $n = (2t + a + 1)$.

`multipele_lists` (Boolean field) and **`multiple_g`** (list field):

`multiple_lists = True` implements multi-list versions of the equality equivalence test, potentially allowing for more precise estimates of the shape of indifference curves. The additional lists differ in the gap variable `g`, which are set as the elements of the variable (list) `multiple_g`. In case of symmetric versions of the test, the 'my'-payoffs for additional lists are scaled in such a way that all allocations lie on rays starting at the equal payoff allocation; this can be easily changed by adapting the determination of 'my'-payoffs in `models.py:54`.

`reduced_form` (Boolean field):

`reduced_form = True` implements a short version of the test with only two binary choices per list, i.e., a total of only four decisions to be made. These choices are constructed the same way as in the standard (symmetric) version of the test. However, the test size `t` is set to 1 and the allocation including a 'my'-payoff equal to `e` is dropped from both lists. Note that the reduced form test is sufficient to discriminate between all nine archetypes of distributional preferences but does not allow for measuring preference intensities.

Role Assignment

`role_fixed` (Boolean field):

If `role_fixed = True`, roles are assigned ex ante (i.e. when creating the session) depending on the player's id's. If `player.id_in_group = 1`, the participant is assigned the role of the active player, if `player.id_in_group = 2`, the participant is assigned the role of the passive player.

`role_uncertain` (Boolean field):

If `role_uncertain = True`, each player decides in the role of the active player. Only after both players in a group have made their decisions, to application randomly determines which player's decisions are relevant for payout, i.e., who is actually the active and who the passive player.

`role_double` (Boolean field):

If `role_double = True`, each player decides in the role of the active player and, at the same time, each player is assigned the role of a passive player as well. That is, subjects get two payoffs: one as an active and one as a passive player.

Group Matching

`group_randomly` (Boolean field):

If `group_randomly = True`, players will be randomly matched in pairs (independent of a player's id). If `group_randomly = False`, players will be matched in ascending order of their `id_in_subsession`, i.e. player 1 and 2 will form a group, player 3 and 4, etc.

`group_by_arrival` (Boolean field):

If `group_by_arrival = True`, players will be matched according to their arrival time on the server. That is, the first player arriving will be matched with the second, the third arriving with the forth, etc. If `group_by_arrival = False`, group matching is determined by the argument of `group_randomly`.

`instructions` (boolean field):

If `instructions = True`, a separate template `Instructions.html` is rendered prior to the task in round one. If `instructions = False`, the task starts immediately (e.g. in case of printed instructions). Please note that the instructions included serve only exemplary purposes and need to be adjusted to your settings in `config.py`.

`results` (Boolean field):

Determines whether a results page summarizing the task's outcome is rendered or not. If `results = True`, a separate view `Results.html` containing all relevant information (i.e., the randomly picked decision to be payed, the active player's choice determining the payments, and the payoff) is rendered. If `results = False`, no separate page is displayed.

`one_page` (Boolean field):

If `one_page = True`, both the x- and y-list will be displayed on a single page/screen (in separate tables). If `one_page = False`, each of the two lists will be shown on separate, subsequent pages. The ordering of the x- and y-list is determined by the argument of `counterbalance`.

`counterbalance` (Boolean field):

If `counterbalance = False`, the ordering of lists is deterministic: the x-list is displayed before the y-list. If `counterbalance = True`, the ordering of the x- and y-lists on screen will be randomized for each active player.

`one_choice_per_page` (Boolean field):

If `one_choice_per_page = True`, all decision problems of both the x- and y-list will be displayed sequentially on separate pages. Technically, the decisions are separated into rounds, i.e., the first choice is made in round 1, the second choice in round 2, etc. Accordingly, the dataset for download contains `n = 4t + 2` rows, one for each round, for each subject with a single observation corresponding to the respective choice. If `one_choice_per_page = False`, all decision pairs are displayed on one page or on two separate pages, depending on the argument of `one_page`.

`shuffle_lists` (Boolean field):

If `shuffle_lists = False`, choices in the x- and y-list will be displayed in ascending order of 'my' payoffs. If `shuffle_lists = True`, the order of choices within each of the two lists will be randomly determined for each active player. If `one_choice_per_page = True`, lists will be shuffled jointly (making `counterbalance` obsolete).

`enforce_consistency` (Boolean field):

If `enforce_consistency = True`, subjects are enforced to answer the choice list without preference reversals (implemented via *JavaScript*). In particular, all options ‘Left’ below a selected option ‘Left’ are automatically selected; similarly, all options ‘Right’ above a selected option ‘Right’ are automatically checked, implying consistent choices. Therefore, `enforce_consistency = True` implies a single switching point and thereby imposes strict monotonicity of revealed preferences and enforces transitivity. Note that enforcing consistency is only reasonably implementable for single-page choice lists in ascending order of ‘my’-payoffs, i.e., if `one_choice_per_page = False` and `shuffle_lists = False`.

`revise_decision` (Boolean field):

If `revise_decision = True`, subjects will have the opportunity to revise the decisions they have made. That is, after all choices have been submitted, participants will face one or more screens (depending on the number of lists) summarizing their decisions in tabular form and allowing them to change the inputs they have made previously. This might in particular be reasonable if `one_choice_per_page = True` and/or `shuffle_lists = True` to provide subjects an opportunity to correct potential inconsistencies.

References

Kerschbamer, R., 2013. The geometry of distributional preferences and a non-parametric identification approach. *Working Papers in Economics and Statistics* 2013-25.

Kerschbamer, R., 2015. The geometry of distributional preferences and a non-parametric identification approach: The equality equivalence test. *European Economic Review* 76, 85–103.