c-simulations Documentation

Release 0.3.3

Gregory McWhirter

June 20, 2014

Contents

This is a framework for running two types of game theory simulations in C. It can run n-population replicator dynamics simulations or Herrnstein-Roth-Erev urn learning simulations.

It builds both static and shared versions of two libraries that should be linked to the program that actually runs the simulations.

Contents:

CHAPTER 1

Replicator Games

The replicator_game.c and replicator_game.h files handle the functionality of defining the game being played for replicator dynamics simulations.

1.1 Constants

CACHE_NONE
This constant indicates that nothing should be cached. It is a `cache_mask`.

CACHE_PROFILES
This constant indicates that only the strategy profiles should be cached. It is a `cache_mask`.

CACHE_PAYOFFS
This constant indicates that only the payoffs should be cached. It is a `cache_mask`.

CACHE_ALL
This constant indicates that both the profiles and payoffs should be cached. It is equivalent to `CACHE_PROFILES` | `CACHE_PAYOFFS`. It is a `cache_mask`.

1.2 Types

cache_mask
This type is an unsigned int. It is specially named to indicate that one of `CACHE_NONE`, `CACHE_PROFILES`, `CACHE_PAYOFFS`, or `CACHE_ALL` should be used.

double ***(*payoff_function)** (int *players*, int **strategy_profile*)
This type defines the signature for a payoff function that takes a strategy profile and returns an array of payoff values.

strategyprofiles_t
This is an alias for the `StrategyProfiles` struct.

game_t
This is an alias for the `Game` struct.

payoffcache_t
This is an alias for the `PayoffCache` struct.

struct **StrategyProfiles**
This struct holds the information for strategy profiles (tuples of interaction possibility)

int **StrategyProfiles.count**
This is the number of profiles the struct holds.

int **StrategyProfiles.size**
This is the size of each profile.

int* **StrategyProfiles.types**
This is an array of types (the number of types for player i is in the ith entry). The `size` member indicates the size of this array.

int **StrategyProfiles.has_cached_info**
This is a flag to indicate whether the struct has cached information stored. It is used for the `StrategyProfiles_destroy()` function.

int** **StrategyProfiles.profiles**
This is an array of the possible strategy profiles. It has size stored in `count` and each element has size from the `size` member.

int*** **StrategyProfiles.player_strategy_profiles**
This is an array of the possible strategy profiles sorted by players participating in them. It has the size defined by the `size` member. The first dimension corresponds to the player. The second dimension corresponds to the number of strategies for that player (size `count`/`types`[i]). The third dimension is a list of the profiles in which that strategy participates, represented by indices referring to the `StrategyProfiles.profiles` list.

struct **Game**

This struct holds data about the game being played under the dynamics.

int **Game.populations**
This is how many populations the game has.

int **Game.players**
This is how many players there are in the game.

int* **Game.types**
This is a list, for each player, how many strategies that player has.

payoff_function **Game.payoffs**
This is the function that returns a payoff vector for a certain strategy profile in the game.

struct **PayoffCache**
This is a struct that holds a cache of pre-calculated payoff vectors.

int **PayoffCache.count**
This is how many items are in the cache.

int **PayoffCache.has_cached_info**
This is a flag to indicate that the cache has information in it that should be freed.

int **PayoffCache.free_profiles**
This is a flag to indicate that the profiles are cached and should be freed.

payoff_function **PayoffCache.payoffs**
This is the payoff function that generates the payoffs.

strategyprofiles_t* **PayoffCache.profiles**
This is the pointer to the cache of strategy profiles.

double** **PayoffCache.payoff_cache**
This is the cache of payoff vectors. Each payoff vector is an array of doubles, and the collection is an array of those arrays.

1.3 Functions

1.3.1 StrategyProfiles

strategyprofiles_t * **StrategyProfiles_create** (int *players*, int **types*, cache_mask *cache*)
This creates a `strategyprofiles_t` struct for the requisite number of `players`.

The `types` parameter is a list of number of strategies that each player has.

The `cache` parameter is how much of the profiles and payoffs to cache.

int * **StrategyProfiles_getProfile** (strategyprofiles_t **sprofs*, int *num*)
This returns the strategy profile corresponding to the `num`th entry in the :c:data:`sprofs` array.

int * **StrategyProfiles_getPlayerProfile** (strategyprofiles_t **sprofs*, int *player*, int *strategy*, int *num*)
This returns the `num`th strategy profile that player :c:data:`player`s strategy `strategy` is involved in.

int **StrategyProfiles_getPlayerProfileNumber** (strategyprofiles_t **sprofs*, int *player*, int *strategy*, int *num*)
This returns the index in the profile list of `sprofs` that the `num`th strategy profile of player :c:data:`player`s strategy `strategy` is involved in.

void **StrategyProfiles_destroy** (strategyprofiles_t **sprofs*)
This frees all data associated with `sprofs`.

1.3.2 Game

game_t * **Game_create** (int *players*, int *populations*, int **types*, payoff_function *payoffs*)
This creates a `game_t` struct based on the requested data.

The number of populations must either be 1 or equal to the number of players.

The parameter `types` is a list of the number of strategies for each player.

The parameter `payoffs` is the payoff function for the game.

void **Game_destroy** (game_t **game*)
This frees all data associated with `game`.

strategyprofiles_t * **Game_StrategyProfiles_create** (game_t **game*, cache_mask *cache*)
This creates a `strategyprofiles_t` struct from the data already present in a `game_t` struct.

1.3.3 PayoffCache

payoffcache_t * **PayoffCache_create** (game_t **game*, strategyprofiles_t **profiles*, cache_mask *do_cache*)
This creates a `payoffcache_t` struct based on the provided information.

double * **PayoffCache_getPayoffs** (payoffcache_t **cache*, int *profile_index*)
This returns the payoffs for the cached profile index `profile_index`.

void **PayoffCache_destroy** (payoffcache_t **cache*)
This frees all data associated with `cache`.

CHAPTER 2

Replicator Populations

2.1 Types

population_t
This is a shortcut for a `Population` struct.

popcollection_t
This is a shortcut for a `PopCollection` struct.

struct **Population**
This struct holds the data relevant to a single replicator population

int **Population.size**
This member determines how many entries are in the `Population.proportions` array.

double* **Population.proportions**
This member holds the population proportions.

struct **PopCollection**
This struct holds a collection of `population_t` structs.

int **PopCollection.size**
This determines how many populations are collected.

int* **PopCollection.pop_sizes**
This is an array of the sizes of each of the populations collected.

popcollection_t** **PopCollection.populations**
This is an array of the populations.

2.2 Functions

2.2.1 Population

population_t * **Population_create** (int *size*)

void **Population_destroy** (population_t **pop*)

int **Population_equal** (population_t **pop1*, population_t **pop2*, double *effective_zero*)

void **Population_copy** (population_t **target*, population_t **source*)

void **Population_randomize** (population_t **pop*)

void **Population_serialize** (population_t *pop, FILE * target_file)

population_t * **Population_deserialize** (FILE * source_file)

2.2.2 PopCollection

popcollection_t * **PopCollection_create** (int num_pops, int *sizes)

popcollection_t * **PopCollection_clone** (popcollection_t *original)

void **PopCollection_destroy** (popcollection_t *coll)

int **PopCollection_equal** (popcollection_t *coll1, popcollection_t *coll2, double effective_zero)

void **PopCollection_copy** (popcollection_t *target, popcollection_t *source)

void **PopCollection_randomize** (popcollection_t *coll)

void **PopCollection_serialize** (popcollection_t *coll, FILE * target_file)

popcollection_t * **PopCollection_deserialize** (FILE * source_file)

CHAPTER 3

Replicator Simulations

3.1 Types

void **(*cb_func)** (game_t *game, int *generation*, popcollection_t *generation_pop, FILE *outfile*)

3.2 Functions

void **replicator_dynamics_setup** ()

popcollection_t * **replicator_dynamics** (game_t *game, popcollection_t *start_pops, double *alpha*, double *effective_zero*, int *max_generations*, cache_mask *caching*, cb_func *on_generation*, FILE *outfile*)

double **earned_payoff** (int *player*, int *strategy*, popcollection_t *pops, strategyprofiles_t *profiles, payoffcache_t *payoff_cache*)

double **average_earned_payoff** (int *player*, popcollection_t *pops, strategyprofiles_t *profiles, payoffcache_t *payoff_cache*)

void **update_population_proportions** (double *alpha*, int *player*, population_t *pop, popcollection_t *curr_pops, strategyprofiles_t *profiles, payoffcache_t *payoff_cache*, int *threads*)

CHAPTER 4

Urn-Learning Games

4.1 Types

unsigned int * **(*urn_interaction)** (unsigned int *players*, urncollection_t ****player_urns*, rk_state **random_state*)

urngame_t

struct **UrnGame**

unsigned int **Urngame.num_players**

unsigned int** **Urngame.types**

urncollection_t** **Urngame.player_urns**

urn_interaction **Urngame.interaction_function**

4.2 Functions

urngame_t * **UrnGame_create** (unsigned int *players*, unsigned int **num_urns*, unsigned int ***types*, double ****initial_counts*, urn_interaction *func*)

void **UrnGame_destroy** (urngame_t **urngame*)

unsigned int * **default_urnlearning_interaction** (unsigned int *players*, urncollection_t ****player_urns*, rk_state **rand_state_ptr*)

void **UrnGame_copy** (urngame_t **source*, urngame_t **target*)

urngame_t * **UrnGame_clone** (urngame_t **urngame*)

CHAPTER 5

Urn-Learning Urns

5.1 Types

urn_t

urncollection_t

struct **Urn**

unsigned int **Urn.types**

double* **Urn.counts**

double* **Urn.proportions**

struct **UrnCollection**

unsigned int **UrnCollection.num_urns**

urn_t** **UrnCollection.urns**

5.2 Functions

5.2.1 Urn

urn_t * **Urn_create** (unsigned int *types*, double **initial_counts*)

void **Urn_destroy** (urn_t * *urn*)

void **Urn_update** (urn_t **urn*, double **count_updates*)

unsigned int **Urn_select** (urn_t **urn*, double *random_draw*)

unsigned int **Urn_randomSelect** (urn_t **urn*, rk_state **rand_state_ptr*)

void **Urn_display** (urn_t * *urn*, char **prefix*, FILE **outfile*)

urn_t * **Urn_clone** (urn_t **urn*)

void **Urn_copy** (urn_t **source*, urn_t **target*)

5.2.2 UrnCollection

urncollection_t * **UrnCollection_create** (unsigned int *num_urns*, unsigned int * *types*, double **initial_counts*)

void **UrnCollection_destroy** (urncollection_t *urnc)

urncollection_t * **UrnCollection_clone** (urncollection_t *urnc)

void **UrnCollection_copy** (urncollection_t *source, urncollection_t *target)

CHAPTER 6

Urn-Learning Simulations

6.1 Types

double ** **(*payoff_function)**(unsigned int *players*, unsigned int **types*, unsigned int * *state_action_profile*)

6.2 Functions

void **urnlearning_dynamics**(urngame_t *urngame, unsigned long *max_iterations*, payoff_function *payoffs*)

Examples:

Universal Deception simulations (Replicator)

Self-Deception simulations (Urn-Learning)

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*