The Agent-Based Model of Human Activity Patterns (ABMHAP): Documentation and Users Guide

Release 2018.06

Namdi Brandon

Jun 25, 2018

Contents

1 Running the ABMHAP Code **3**
- 1.1 How to Run the Code Using User-Defined Parameters . . . 3
  - 1.1.1 Setting the input parameters . . . 3
  - 1.1.2 Running the simulation . . . 7
  - 1.1.3 Interpreting the output . . . 7
- 1.2 How to Run the Code Using Parameters Derived from an Empirical Dataset . . . 8
  - 1.2.1 Setting the input parameters . . . 8
  - 1.2.2 Running the simulation . . . 9
    - 1.2.2.1 Running in **serial** . . . 10
    - 1.2.2.2 Running in **parallel** . . . 10
  - 1.2.3 Interpreting the output . . . 10

2 Code Documentation **13**
- 2.1 Source Directory . . . 13
  - 2.1.1 activity module . . . 13
  - 2.1.2 asset module . . . 15
  - 2.1.3 bed module . . . 17
  - 2.1.4 bio module . . . 17
  - 2.1.5 chad module . . . 19
  - 2.1.6 chad_code module . . . 22
  - 2.1.7 commute module . . . 23
  - 2.1.8 diary module . . . 27
  - 2.1.9 eat module . . . 30
  - 2.1.10 food module . . . 35
  - 2.1.11 home module . . . 35
  - 2.1.12 hunger module . . . 37
  - 2.1.13 income module . . . 40
  - 2.1.14 interrupt module . . . 41
  - 2.1.15 interruption module . . . 41
  - 2.1.16 location module . . . 43
  - 2.1.17 meal module . . . 44
  - 2.1.18 my_globals module . . . 45
  - 2.1.19 need module . . . 49
  - 2.1.20 occupation module . . . 50
  - 2.1.21 params module . . . 55
  - 2.1.22 person module . . . 61

2.1.23 rest module . . . 64
2.1.24 scheduler module . . . 67
2.1.25 sleep module . . . 68
2.1.26 social module . . . 70
2.1.27 state module . . . 73
2.1.28 temporal module . . . 75
2.1.29 transport module . . . 77
2.1.30 travel module . . . 78
2.1.31 universe module . . . 79
2.1.32 work module . . . 83
2.1.33 workplace module . . . 85
2.2 Run Directory . . . 85
2.2.1 main module . . . 85
2.2.2 main_params module . . . 86
2.2.3 scenario module . . . 86
2.2.4 singleton module . . . 88
2.3 Run_chad Directory . . . 89
2.3.1 analysis module . . . 89
2.3.2 analyzer module . . . 91
2.3.3 chad_demography module . . . 94
2.3.4 chad_demography_adult_non_work module . . . 99
2.3.5 chad_demography_adult_work module . . . 100
2.3.6 chad_demography_child_school module . . . 100
2.3.7 chad_demography_child_young module . . . 101
2.3.8 chad_parameter_figures notebook . . . 102
2.3.9 chad_params module . . . 104
2.3.10 commute_from_work_trial module . . . 107
2.3.11 commute_to_work_trial module . . . 108
2.3.12 data_counter notebook . . . 109
2.3.13 driver module . . . 115
2.3.14 driver_params module . . . 122
2.3.15 driver_result module . . . 123
2.3.16 eat_breakfast_trial module . . . 124
2.3.17 eat_dinner_trial module . . . 125
2.3.18 eat_lunch_trial module . . . 126
2.3.19 evaluation module . . . 128
2.3.20 fig_driver module . . . 132
2.3.21 figure_loader notebook . . . 133
2.3.22 figure_loader_with_without_variation notebook . . . 142
2.3.23 figure_residuals notebook . . . 149
2.3.24 longitude_plot notebook . . . 154
2.3.25 plot_graphs notebook . . . 157
2.3.26 my_debug module . . . 163
2.3.27 omni_trial module . . . 163
2.3.28 sleep_trial module . . . 167
2.3.29 trial module . . . 169
2.3.30 variation module . . . 175
2.3.31 work_trial module . . . 176
2.4 Plotting Directory . . . 177
2.4.1 plot_diary notebook . . . 177
2.4.2 plotter module . . . 183
2.5 Processing Directory . . . 186
2.5.1 commute_school notebook . . . 186
2.5.2 commute_work notebook . . . 188

2.5.3 count_records notebook 190
2.5.4 datum module 194
2.5.5 demographics notebook 202
2.5.6 demography module 210
2.5.7 eat_new notebook 211
2.5.8 school_new notebook 214
2.5.9 sleep_new notebook 215

3 Indices and tables **219**

Python Module Index **221**

The Agent-Based Model of Human Activity Patterns (ABMHAP, pronounced "ab-map") is one of the modules for the Life Cycle Human Exposure Model (LC-HEM) project as described in the United States Environmental Protection Agency (U.S. EPA) research plan, which may be found here. ABMHAP is a model capable of creating agents that simulate longitudinal human behavior. The current version of ABMHAP is able to simulate daily routines for the following behaviors:

1. Sleeping
2. Eating Breakfast
3. Eating Lunch
4. Eating Dinner
5. Working
6. Commuting to Work
7. Commuting from Work
8. Being Idle (i.e., time spent not doing the above activities)

ABMHAP requires the user to input parameters that describe the longitudinal variation in behavior of a single individual. Information on the design of the model, modeling assumptions, and limitations of the model are described in the publications:

1. Brandon et al. *Simulating Exposure-Related Behaviors using Agent-Based Models Embedded with Needs-Based Artificial Intelligence*, Journal of Exposure Science and Environmental Epidemiology, submitted 2018.
2. Brandon, N and Price, P. Simulating *Long-Term Patterns in Exposure-Related Behaviors using the Agent-Based Model of Human Activity Patterns*, Journal of Exposure Science and Environmental Epidemiology, submitted 2018.

The current version of ABMHAP is written in Python version 3.5.3. More information on the Python programming language may be found here. The Python libraries that must be installed in order for ABMHAP to run are listed below.

- matplotlib
- multiprocessing
- numpy
- pandas
- scipy
- sphinx
- statsmodels

Some of the features within ABMHAP also use Jupyter Notebook. More information on Project Jupyter may be found here.

ABMHAP is written by Dr. Namdi Brandon (ORCID: 0000-0001-7050-1538).

Disclaimer The United States Environmental Protection Agency through its Office of Research and Development has developed this software. The code is made publicly available to better communicate the research. All input data used for a given application should be reviewed by the researcher so that the model results are based on appropriate data for any given application. This model is under continued development. The model and data included herein do not represent and should not be construed to represent any Agency determination or policy.

CHAPTER 1

Running the ABMHAP Code

The current version of ABMHAP may be run in two different ways.

1. ABMHAP may run with user-defined parameters in order to simulate one household
2. ABMHAP may run with parameters derived from empirical data as a Monte-Carlo simulation in order to simulate multiple households

1.1 How to Run the Code Using User-Defined Parameters

The following describes how to run an ABMHAP simulation of **one household** (ABMHAP currenlty simulates one agent per household). In order to do so, the user must do the following:

1. set the input parameters of the simulation in the file `\run\main_params.py`
2. run the executable using `\run\main.py`

1.1.1 Setting the input parameters

In order to run ABMHAP, the user must set the following types of input parameters in `\run\main_params.py`:

1. input parameters that govern the general logistics of the simulation
2. input parameters that govern the the length of simulation duration
3. input parameters that define the behavior of the agent

For illustrative purposes, what follows is a demonstration of how to parametrize a run for ABMHAP.

The below code does the following:

- informs the algorithm to not print the output to the screen
- informs the algorithm to not plot the output
- informs the algorithm to not save the output to a file

- should the output file be saved, sets the output file to `\output_directory\outputfile.csv`

The user must set the input parameters that govern the general logistics of the simulation:

```
# whether (if True) or not (if False) the output of the simulation should
# print a message to screen
do_print     = False

# whether (if True) or not (if False) the output of the simulation should
# be plotted a message to screen
do_plot      = False

# whether (if True) or not (if False) the output of the simulation should
# be saved in a file
do_save      = False

# the name of the output file should the output be saved. The filename
# should have a ".csv" extension
fname        = 'output_directory\\outputfile.csv'
```

The following code shows how to set ABMHAP to run starting on Sunday, Day 0 starting from 16:00 and ending on Monday, Day 7 at 0:00. It's recommended that the user start running the code on a Sunday or Saturday at 16:00 in order to minimize potential activity conflicts at initiation.

The user must set the input parameters dealing with the duration of the simulation:

```
# the number of days for the simulation
num_days     = 7

# the number of additional hours
num_hours    = 8

# the number of additional minutes
num_min      = 0
```

The user must set the input parameters dealing with when in the simulation year the simulation should start:

```
# start the simulation on Sunday, Day 0 at 16:00
t_start      = WINTER * SEASON_2_MIN + 0 * WEEK_2_MIN \
             + SUNDAY * DAY_2_MIN + 16 * HOUR_2_MIN
```

where the following constants are useful in assigning input parameters that define the start time of the simulation:

```
# an agent-based model module with capabilities concerning time
import temporal

# the value of Sunday
SUNDAY          = temporal.SUNDAY

# convert one day into minutes
DAY_2_MIN       = temporal.DAY_2_MIN

# convert one hour into minutes
HOUR_2_MIN      = temporal.HOUR_2_MIN

# the number of minutes in one season (13 weeks)
SEASON_2_MIN    = temporal.SEASON_2_MIN
```

(continues on next page)

(continued from previous page)

```
# the number of minutes in one week
WEEK_2_MIN      = temporal.WEEK_2_MIN

# the winter season (has the value 0)
WINTER          = temporal.WINTER
```

The user must set the input parameters that govern the behavior of the agent. The input parameters will govern the agent's behavior for the following activities.

1. sleeping
2. eating breakfast
3. eating lunch
4. eating dinner
5. working
6. commuting to work
7. commuting from work

In order to set the sleeping behavior, the user must set the the mean and standard deviation for the start time and end time for the sleep activity. The agent's behavior for sleeping is set as follows:

```
# set the mean start time to be 22:00
sleep_start_mean      = np.array( [22 * HOUR_2_MIN] )

# set the standard deviation of the start time to be 30 minutes
sleep_start_std       = np.array( [30] )

# set the mean end time to be 8:00
sleep_end_mean        = np.array( [8 * HOUR_2_MIN] )

# set the standard deviation of the end time to be 15 minutes
sleep_end_std         = np.array( [15] )
```

In order to set the eat breakfast behavior, the user must set the mean and standard deviation for the start time and duration for the eat breakfast activity. The agent's behavior for eating breakfast is set as follows:

```
# set the mean start time to be 8:00
bf_start_mean        = np.array( [8 * HOUR_2_MIN] )

# set the standard deviation of the start time to be 10 minutes
bf_start_std         = np.array( [10] )

# set the mean duration to be 15 minutes
bf_dt_mean           = np.array( [15] )

# set the standard deviation of the duration to be 10 minutes
bf_dt_std            = np.array( [10] )
```

In order to set the eat lunch behavior, the user must set the mean and standard deviation for the start time and duration for the eat lunch activity. The agent's behavior for eating lunch is set as follows:

```
# set the mean start time to be 12:000
lunch_start_mean       = np.array( [12 * HOUR_2_MIN] )
```

(continues on next page)

(continued from previous page)

```
# set the standard deviation of start time to be 15 minutes
lunch_start_std        = np.array( [15] )

# set the mean duration to be 30 minutes
lunch_dt_mean          = np.array( [30] )

# set the standard deviation of duration to be 10 minutes
lunch_dt_std           = np.array( [10] )
```

In order to set the eat dinner behavior, the user must set the mean and standard deviation for the start time and duration for the eat dinner activity. The agent's behavior for eating dinner is set as follows:

```
# set the mean start time to be 19:00
dinner_start_mean       = np.array( [19 * HOUR_2_MIN] )

# set the standard deviation of start time to be 10 minutes
dinner_start_std        = np.array( [10] )

# set the mean of duration to be 45 minutes
dinner_dt_mean          = np.array( [45] )

# set the standard deviation of duration to be 5 minutes
dinner_dt_std           = np.array( [5] )
```

In order to set the work behavior, the user must set the mean and standard deviation for the start time and end time for the work activity. The agent's behavior for working is set as follows:

```
# set the mean start time to be 9:00
work_start_mean     = np.array( [9 * HOUR_2_MIN] )

# set the standard deviation of start time to be 15 minutes
work_start_std      = np.array( [15] )

# set the mean end time to be 17:00
work_end_mean       = np.array( [17 * HOUR_2_MIN] )

# set the standard deviation of end time to be 5 minutes
work_end_std        = np.array( [5] )
```

The user must set the agent's employment status. The agent's employment status is set as follows:

```
# an agent-based model module for functionality dealing with occupation
import occupation

# set the job identifier (job id) as standard job if the agent
# is supposed to work
job_id    = occupation.STANDARD_JOB

# OR set the job identifier (job id) as not having a job if the agent
# is NOT supposed to work
job_id    = occupation.NO_JOB
```

In order to set the commute to work behavior, the user must set the mean and standard deviation for the duration of the commute to work activity. The agent's behavior for commuting to work is set as follows:

```
# set the mean duration to be 30 minutes
commute_to_work_dt_mean     = np.array( [30] )

# set the standard deviation to be 10 minutes
commute_to_work_dt_std      = np.array( [10] )
```

In order to set the commute from work behavior, the user must set the mean and standard deviation for the duration of the commute from work activity. The agent's behavior for commuting from work is set as follows:

```
# set the mean duration to be 30 minutes
commute_from_work_dt_mean     = np.array( [30] )

# set the standard deviation to be 10 minutes
commute_from_work_dt_std      = np.array( [10] )
```

1.1.2 Running the simulation

After defining all of the parameters in the file `\run\main_params.py`, the code is run by doing the following:

1. go to the `\run` directory.
2. enter `python main.py` into the terminal (or command line)
3. press enter (or return)

1.1.3 Interpreting the output

ABMHAP outputs the record of the activities that the agent did during the simulation. This record is called an **activity diary**. An activity diary is a chronological record contains the following information about each activity: day, start time, end time, duration, and location.

Below is an example of the output of ABMHAP. Recall that ABMHAP saves the activity diary in terms of a .csv file

day	start	end	dt	act	loc
0	16	19	3	-1	0
0	19	19.75	0.75	4	0
0	19.75	22	2.25	-1	0
0	22	8	10	6	0
1	8	8.25	0.25	3	0
1	8.25	8.5	0.25	-1	0
1	8.5	9	0.5	2	1
1	9	12	3	7	3
1	12	12.5	0.5	5	3
1	12.5	17	4.5	7	3
1	17	17.5	0.5	1	1
1	17.5	19	1.5	-1	0
1	19	19.75	0.75	4	0
1	19.75	22	2.25	-1	0
1	22	8	10	6	0

where day, start, end, dt, act, and loc represent the day the activity starts, the start time of the activity (in hours), the end time of the activity (in hours), the duration of the activity (in hours), the activity identifier, and the location identifier, respectively. In the results, the time of day 16:30 is represented as 16.5.

The following table is an interpretation of the example output shown above. In the table, the duration is expressed in minutes.

Day	Start	End	Duration	Activity Code	Location Code
0	16:00	19:00	180	Idle	Home
0	19:00	19:45	45	Eat dinner	Home
0	19:45	22:00	135	Idle	Home
0	22:00	8:00	600	Sleep	Home
1	8:00	8:15	15	Eat breakfast	Home
1	8:15	8:30	15	Idle	Home
1	8:30	9:00	30	Commute to work	Out of doors
1	9:00	12:00	180	Work	Workplace
1	12:00	12:30	30	Eat lunch	Workplace
1	12:30	17:00	270	Work	Workplace
1	17:00	17:30	30	Commute from work	Out of doors
1	17:30	19:00	90	Idle	Home
1	19:00	19:45	45	Eat dinner	Home
1	19.45	22:00	135	Idle	Home
1	22:00	8:00	600	Sleep	Home

1.2 How to Run the Code Using Parameters Derived from an Empirical Dataset

ABMHAP may be also used to simulate agents whose parametrization is derived from an empirical dataset, as opposed to using user-defined parameters. Specifically, the current version of ABMHAP uses the Consolidated Human Activity Database (CHAD) to parametrize the behavior of agents. More information about CHAD may be found here. Currently, ABMAHP uses data from CHAD to parametrize agents that simulate the behaviors of people that represent the following demographic groups within the general United States population:

1. working adults (ages 18 and above)
2. non-working adults (ages 18 and above)
3. school-age children (ages 5 through 17)
4. preschool children (ages 1 through 4)

To simulate a demographic, we run ABMHAP via a Monte-Carlo simulation that creates **multiple households** (ABMHAP currently simulates one agent per household) and randomly parametrizes their behavior based on empirical distributions from CHAD data. In order to decrease the runtime of simulating multiple households, ABMHAP has the capability to run the Monte-Carlo simulations in **parallel**.

In order to run the Monte-Carlo simulations, the user must do the following:

1. set the input parameters of the simulation in the file `\run_chad\driver_params.py`
2. run the executable using `\run_chad\driver.py`

1.2.1 Setting the input parameters

In order to run ABMHAP, the user must set the following types of input parameters in `\run_chad\driver_params.py`:

1. input parameters that govern the general logistics of the simulation

2. input parameters that govern the the length of simulation duration
3. input parameters that define the demographic of the agents

For illustrative purposes, what follows is a demonstration of how to parametrize a run for ABMHAP. See, the earlier section “Setting the input parameters” in “How to Run the Code Using User-Defined Parameters” to understand how to set up the input parameters:

```
# the number of days for the simulation
num_days    = 364

# the number of additional hours
num_hours   = 8

# the number of additional minutes
num_min     = 0

# should the simulation plot results at the end of the run
do_plot         = False

# should the simulation print messages to the screen
do_print        = True

# should the simulation save the results (both input and output)
# of the simulation
do_save         = False
```

In addition, the user must define the demographic of the agents being simulated. This causes ABMHAP to use the empirical data from the respective demographic in CHAD to parametrize the agent:

```
# this causes ABMHAP to use empirical data from CHAD corresponding
# to the working adult demographic in order to parametrize the agents
demographic    = dmg.ADULT_WORK

# the path to the output directory where should the output results
# should be saved
fpath          = '\\output_directory'

# load input data from a previous simulation
do_load_trials  = False

# the file name for "trials" data without the .pkl extension,
# which will be used for saving the trial information
fname_load_trials_base = None
```

1.2.2 Running the simulation

After defining all of the parameters in the file `\run_chad\driver_params.py`, the code is run by doing the following:

1. go to the `\run_chad` directory.
2. enter the following into the terminal (or command line):

   ```
   python driver.py num_process num_hhld num_batch
   ```

   where

   - `num_process` is the total number of cores (i.e, processing units) used in the simulation

- `num_hhld` is the number of housholds to run per batch
- `num_batch` is the number of batches used per core

3. press enter (or return)

To run the code, do the following.

Note: This code may be run in **batches** in order to run many households while conserving memory. That is, instead of running 32 households at once (and keeping 32 households in memory), the program can run 2 batches of 16 households (for a total of 32 households). This halves the amount of memory used in the simulation compared to running the simulation of 1 batch of 32 households. We will shown how to run the code using "batches" below.

1.2.2.1 Running in serial

The following are examples on how to run the code in serial.

To run in serial with with 64 households per batch, 1 batch (implied)

> `python driver.py 1 64 1`

> `python driver.py 1 64`

To run in serial using 2 batches with 1 thread with 32 households per batch, 2 batches

> `python driver.py 1 32 2`

1.2.2.2 Running in parallel

The following are examples on how to run the code in parallel.

To run in parallel using 4 cores with 64 households total (16 household per core per batch), 1 batch (implied)

> `python driver.py 4 64 1`

> `python driver.py 4 64`

To run in parallel using 4 cores with 32 households per batch, 2 batches(8 households per core per batch)

> `python driver.py 4 32 2`

1.2.3 Interpreting the output

ABMHAP outputs the record of the activities that **each** agent did during the simulation. Each agent's record is called an **activity diary**. An activity diary is a chronological record contains the following information about each activity: day, start time, end time, duration, and location. The output is a combined record of every agent simulated where each agent is given a unique integer as an identifier.

Below is an example of the output of ABMHAP. Recall that ABMHAP saves the activity diary in terms of a .csv file

id	day	start	end	dt	act	loc
0	0	16	19	3	-1	0
0	0	19	19.75	0.75	4	0
⋮						
0	364	22	0	2	6	0
1	0	16	20	4	-1	0
1	0	20	20.5	0.5	4	0
⋮						
1	364	23	0	1	6	0

where id, day, start, end, dt, act, and loc represent the agent identifier, the day the activity starts, the start time of the activity (in hours), the end time of the activity (in hours), the duration of the activity (in hours), the activity identifier, and the location identifier, respectively. In the results, the time of day 16:30 is represented as 16.5. Each agent in the Mont-Carlo simulation is given a unique identifier via "id".

The following table is an interpretation of the example output shown above. In the table, the duration is expressed in minutes.

Identifier	Day	Start	End	Duration	Activity Code	Location Code
0	0	16:00	19:00	180	Idle	Home
0	0	19:00	19:45	45	Eat dinner	Home
⋮						
0	364	22:00	00:00	120	Sleep	Home
1	0	16:00	20:00	240	Idle	Home
1	0	20:00	20:30	30	Eat dinner	Home
⋮						
1	364	23:00	00:00	60	Sleep	Home

Given that ABMHAP is set to simulate working adults, the results from a ABMHAP simulation are saved in the following files:

1. `\output_directory\data_adult_work.csv`
2. `\output_directory\trials_adult_work.pkl`
3. `\output_directory\data_adult_work.pkl`

The `.csv` file contains the activity diary of all of the simulations. The other files are created for the following reason. Because these runs are capable of simulating large numbers of agents, in order to save memory space, the results from ABMHAP are saved in a compressed file format unique to python called "pickle" file format, which is denoted with a `.pkl` extension. The saved python objects correspond to

1. `\output_directory\trials_adult_work.pkl` contains the input data for each household being simulated. The file contains a list of *`trial.Trial`* objects.
2. `\output_directory\data_adult_work.pkl` contains the output data (i.e., the activity diaries) for each household being simulated. The file contains a *`driver_result.Driver_Result`* object

CHAPTER 2

Code Documentation

The following is the documentation of all of the files that are within the ABMHAP code. The code is divided into the following directories:

- `\source`, which handles the key components for ABMHAP
- `\run`, which handles code for running ABMHAP with user-defined parameters
- `\run_chad`, which handles running ABMHAP as a Monte-Carlo simulation with agents parameterized with empirical data from CHAD
- `\plotting`, which handles some plotting capability
- `\processing`, which handles parses the data from CHAD

2.1 Source Directory

These files are the key modules that are used to create the ABMHAP algorithm.

Contents:

2.1.1 activity module

This module contains code that governs the activities that the agent performs in order to satisfy its needs.

This module contains the following class: *`activity.Activity`*.

class `activity.`**`Activity`**

Bases: `object`

An activity enables a *`person.Person`* to address its satiation $n(t)$. This class's purpose is to encapsulate general capabilities that specific instances of activity will derive from.

Variables

- **`category`** (*`int`*) – an unique identifier naming the type of activity.

- **t_end** (*int*) – the end time of the activity [universal time, seconds]
- **t_start** (*int*) – the start time of the activity [universal time, seconds]
- **dt** (*int*) – the duration of the activity [seconds]

advertise(*the_need*, *dt*)

Calculates the advertised score of doing an activity. Let Ω be the set of all needs addressed by the activity. The score S is calculated by doing the following

$$S = \begin{cases} 0 & n(t) > \lambda \\ \sum_{j \in \Omega} W_j(n_j(t)) - W_j(n_j(t + \Delta t)) & n(t) \leq \lambda \end{cases}$$

where

- t is the current time
- Δt is the duration of the activity
- $n(t)$ is the satiation at time t
- λ is the threshold value of the need
- $W(n)$ is the weight function for the particular need

Parameters

- **the_need** (`need.Need`) – the primary need associated with the respective activity
- **dt** (*int*) – the duration Δt of doing the activity [minutes]

Returns score the score of the advertisement

Return type float

advertise_interruption()

Advertise the score if this activity interrupts another activity.

Note: This function should be overloaded in derived classes.

Returns score the advertised score

Return type float

end(*p*)

This function handles some of the common logistics in ending a specific activity assuming the activity ends without an interruption.

Currently the function does the following:

1. reset the *state.State* variable's start time to the current time
2. reset the *state.State* variable's end time to the current time

Parameters p (`person.Person`) – the person whose activity is ending

Returns None

halt(*p*)

This function handles some of the common logistics in ending a specific activity due to an interruption.

Currently the function does the following:

1. reset the *state.State* variable's start time to the current time
2. reset the *state.State* variable's end time to the current time

Parameters **p** (person.Person) – the person whose activity is being interrupted

Returns None

print_id()

This function represents the activity category as a string.

Return msg The string representation of the category

Return type str

start()

This function starts a specific activity.

Note: This function is meant to be overloaded by derived activity classes.

Returns None

toString()

This function represents the Activity object as a string.

Return msg the string representation of the activity object

Return type str

2.1.2 asset module

This module contains code that governs objects that enable access to activities (*activity.Activity*) that an agent may use in order to address a need.

This module contains the following class: *asset.Asset*.

class asset.**Asset**

Bases: object

An asset is an object that allows the agent to perform an activity. Each asset contains a list of activities that an agent can use to perform actions.

Variables

- **activities** (*dict*) – a dictionary of all the activities associated with this asset
- **category** (*int*) – a code that indicates the category type of asset
- **id** (*int*) – an identifier number for the asset
- **'location'** (location.Location) – the location of the asset
- **max_users** (*int*) – the maximum number of users that can simultaneously access the asset
- **num_users** (*int*) – the current number of users for the asset
- **status** (*int*) – the state of the asset

free()

This function changes the state of an asset once it is freed by a Person by doing the following:

1. decreases the number of users of the asset by 1
2. if the number of users is zero, the status of the asset is set to idle (`state.IDLE`)

Returns None

`initialize`(*people*)

This function initializes the asset at the beginning of the simulation.

Note: This function is meant to be overridden

Parameters **`people`** (*list[person.Person]*) – the Person objects who could be using the asset.

Returns None

`print_category`()

This function represents the category as a string.

Returns the string representation of the category

Return type str

`reset`()

This function does the following:

1. sets the number of users to zero
2. sets the asset's status to idle

Returns None

`toString`()

This function represents the asset as a string.

Return msg The string representation of the asset object.

Return type str

`update`()

This function changes the state of the asset once it is used by a person. The update does the following:

1. increases the number of people by 1
2. if the number of users is at the maximum number, set the asset's status to busy
3. if the number of users is less than the maximum number, set the asset's status to busy but able to be used by another agent

Returns None

2.1.3 bed module

This module contains code that enables the agent to use a bed. This class allows access to the sleep (*sleep.Sleep*) activity.

This module contains the following class: *bed.Bed*.

class bed.**Bed**

Bases: *asset.Asset*

This asset models a bed. It allows the agent to address the Rest (*rest.Rest*) need by doing the sleep (*sleep.Sleep*) activity.

2.1.4 bio module

This module contains information about a Person's (*person.Person*) biology.

This module contains the following class: *bio.Bio*.

class bio.**Bio**

Bases: object

This class holds the biologically relevant information for a person. This information is:

- age
- gender
- mean / standard deviation of start time for sleeping
- mean / standard deviation of end time for sleeping
- probability distribution function sleep start time / end time

Variables

- **age** (*int*) – the age [years]
- **gender** (*int*) – the gender
- **sleep_dt** (*int*) – the duration of time for a sleep event [minutes]
- **sleep_start_mean** (*int*) – the mean start time for a sleep event [minutes]
- **sleep_start_std** (*int*) – the standard deviation for start time for a sleep event [minutes]
- **sleep_start** (*int*) – the start time for sleep [minutes, time of day]
- **sleep_start_univ** (*int*) – the start time for sleep[minutes, universal time]
- **sleep_end_mean** (*int*) – the mean end time for a sleep event [minutes]
- **sleep_end_std** (*int*) – the standard deviation for end time for a sleep event [minutes]
- **sleep_end** (*int*) – the end time for sleep[minutes, time of day]
- **sleep_end_univ** (*int*) – the end time for sleep [minutes, universal time]
- **start_trunc** (*int*) – the number of standard deviations to allow when sampling sleep the truncated distribution for start time
- **end_trunc** (*int*) – the number of standard deviations to allow when sampling sleep the truncated distribution for end time

- **f_sleep_start** (*func*) – the distribution data for start time for sleep
- **f_sleep_end** (*func*) – the distribution data for end time for sleep

calc_awake_duration (*t*)
This function calculates the amount of time the person is expected to be awake.

Parameters **t** (*int*) – time of day [minutes]

Returns the duration [minutes] until the agent is expected to awaken

print_gender ()
This function returns a string representation of gender

Returns the string representation of gender

Return type str

set_sleep_params (*start_mean*, *start_std*, *end_mean*, *end_std*)
This function sets the biological sleep parameters themselves and the sleep parameter distribution functions.

Parameters

- **start_mean** (*int*) – the mean sleep start time [minutes]
- **start_std** (*int*) – the standard deviation of start time [minutes]
- **end_mean** (*int*) – the mean sleep end time [minutes]
- **end_std** (*int*) – the standard deviation of end time [minutes]

Returns None

toString (*do_decimal=False*)
This function represents the Bio object as a string.

Parameters **do_decimal** (*bool*) – This controls whether or not to represent the values in time in a decimal (hours) format where [1:30pm is 13.5] if True or as the minutes in the day if False [1:30pm is 13 * 60 + 30].

Return msg the string representation of the Bio object

Return type string

update_sleep_dt ()
This function sets the duration of sleep.

Returns None

update_sleep_end ()
This function samples the sleep end time distribution and sets the end time.

Returns None

update_sleep_end_univ (*time_of_day*, *t_univ*)
This function sets the end time for sleep in terms of universal time.

Parameters

- **time_of_day** (*int*) – the current time of day [minutes]
- **t_univ** (*int*) – the universal time [minutes]

Returns None

update_sleep_start()
This function samples the sleep start time distribution and sets the start time.

Returns None

update_sleep_start_univ(*time_of_day*, *t_univ*)
This function sets the start time for sleep in terms of universal time.

Parameters

- **time_of_day** (*int*) – the current time of day [minutes]
- **t_univ** (*int*) – the universal time [minutes]

Returns None

update_time_univ(*time_of_day*, *t_univ*, *t*)
This function updates a time t, which represents sleep start time or end time, to be in the next occurrence

Parameters

- **time_of_day** (*int*) – the current time of day [minutes]
- **t_univ** (*int*) – the universal time [minutes]
- **t** (*int*) – the time to be set[minutes, time of day]

Return out the time of the next event in universal time

Return type int

2.1.5 chad module

This file contains data from the Consolidated Human Activity Database (CHAD). This module contains constants necessary to access various files in the CHAD.

This module contains the following classes:

1. *chad.CHAD*
2. *chad.CHAD_RAW*.

class chad.**CHAD**(*fname*, *mode='r'*)
Bases: object

This object is in charge of accessing the compressed data files from CHAD.

Parameters

- **fname** (*str*) – the directory to the respective compressed data files
- **mode** (*str*) – the mode (read, write, or both) the zipfile will work under

Variables

- **fname_zip** (*str*) – the directory to the respective compressed data file (.zip)
- **mode** (*int*) – the mode (read, write, or both) the zipfile will work under
- **z** (*zipfile.Zipfile*) – object that holds the zipfile information

activity_times(*df*, *act_codes*)
This function finds the activity data (given by act_codes) in the dataframe df.

Parameters

- **df** (*pandas.core.frame.DataFrame*) – events data

- **act_codes** (*list*) – the list of CHAD activity codes specifying 1 general activity

Returns the activity data for the selected activity codes

Return type pandas.core.frame.DataFrame

get_data(*fname*)
Gets the decompressed data from the given file

Parameters **fname** (*str*) – the name of the file to decompressed

Return data the data

Return type pandas.core.frame.DataFrame

sum_time(*df*)
This function merges two similar adjacent activities into one activity. This function is used normally for the CHAD events data.

Parameters **df** (*pandas.core.frame.DataFrame*) – the dataframe corresponding to a specific CHAD identifier

Returns the dataframe where adjacent activities are merged into one activity

Return type pandas.core.frame.DataFrame

toString()
Represent the contents of the compressed file.

Returns a string representation of the CHAD object

Return type string

class chad.**CHAD_RAW**(*min_age=18*, *max_age=130*)
Bases: *chad.CHAD*

This is a specific instance of *chad.CHAD* that is made for accessing the raw CHAD data for accessing the questionnaire database and the events database.

Parameters

- **min_age** (*int*) – the minimum age [years] for the CHAD data age range
- **max_age** (*int*) – the maximum age [years] for the CHAD data age range

Variables

- **quest** (*pandas.core.frame.DataFrame*) – the CHAD questionnaire data
- **events** (*pandas.core.frame.DataFrame*) – the CHAD events data

clean_data()
This function cleans the data from the loaded CHAD .csv files for the format used for the ABM.

1. clean events
2. clean dates
3. set dates

Returns None

clean_dates()
This function is needed in order to remove data where there are no dates from the dataframes that represent the CHAD questionnaire data and the CHAD events data.

Returns None

clean_events()
This cleans the time information in the CHAD events data.

Returns None

convert_activity_code(*x*)
This function converts the activity code from a string into an integer. It also converts 'X' and 'U' into a numerical value.

Parameters **x** (*string*) – the activity code that needs to be converted

Returns None

convert_military_to_decimal_time(*x*)
This function converts military time [00 00 - 23 59] to decimal time [0.0 - 24).

Parameters **x** (*int*) – an integer representation of the military time where 09:00 is represented by 0900.

Returns the time converted into decimal time

Return type float

get_PID(*x*)
Given a CHADID, this function returns the PID. The PID is the CHADID stripped of the last character, which is a code for the day record.

Parameters **x** (*string*) – the CHADID

Returns the PID

Return type numpy.ndarray

get_data_by_age(*min_age*, *max_age*)
This function samples the CHAD data by age via the age range inputs.

Parameters

- **min_age** (*int*) – the minimum age [years]
- **max_age** (*int*) – the maximum age [years]

Returns None

get_events()
This function gets the raw CHAD events data and returns it in the appropriate column order.

Returns the CHAD events data

Return type pandas.core.frame.DataFrame

get_events_raw()
This function returns a data frame of the raw events data.

Return data the raw CHAD events data

Return type pandas.core.frame.DataFrame

get_quest()
This function returns a data frame of the raw questionnaire data. However, the data must have the date marked explicitly to be accepted.

Returns the CHAD questionnaire data in the correct column order

Return type pandas.core.frame.DataFrame

get_quest_raw()

This function returns a data frame of the raw questionnaire data and add the PID information.

Returns the raw CHAD questionnaire data

Return type pandas.core.frame.DataFrame

set_dates()

This function converts the date information in the CHAD questionnaire and CHAD events datafrmaes from strings to python datetime objects.

Returns None

set_times()

This function handles setting the time information for formatting in the CHAD questionnaire and events dataframes

1. converts the time from military time (0000 - 2359] to decimal time [0, 24) in the CHAD evnets and questionnaire data for the start time and end time
2. converts the duration to hours
3. drops the military time (start time and end time) and duration (minutes) from the respective data frames

Returns None

chad.**sample_stats**(*sample*)

This function takes sample data and returns the mean and the standard deviation.

Parameters **sample** (*pandas.core.frame.DataFrame*) – the data to analyze

Return s_mean the mean of the sample data

Return s_std the standard deviation of the sample data

Return type pandas.core.frame.DataFrame

Return type pandas.core.frame.DataFrame

2.1.6 chad_code module

This module contains activity codes found in the Consolidated Human Activity Database (CHAD).

The following general *chad_code* constants consist of groupings of CHAD activity codes

1. sleep
2. eat
3. **work**
    - work and income producing activities; work, general; work, income-related only; work, secondary (income-related); work breaks
4. **education**
    - general education and professional training, attending full-time school, attend day-care; attend school kindergarten - 12th grade
5. **commute to/ from work**

- travel to/ from work general; travel to/ from work by bus; travel to/ from work by foot; travel to/ from via motor vehicle; travel to/ from work via motor vehicle, by driving; travel to/from work via motor vehicle by driving via motor vehicle, by riding; travel to/ from work waiting

6. **commute to/ from school**

   - travel for education general; travel for education by bus; travel for education by foot; travel to/ from school via motor vehicle; travel for education via motor vehicle, by driving; travel for education via motor vehicle, by riding; travel for education, waiting

7. **All**

   - sleep + eat + work + education + commute to/ from work + commute to/ from school

2.1.7 commute module

This module contains about activities associated with commuting to and from work. This class is an Activity (*activity.Activity*) that gives a Person (*person.Person*) the ability to commute to/ from work/ school and satisfy the need Travel (*travel.Travel*).

This module contains the following classes:

1. *commute.Commute* (general commuting capability)
2. *commute.Commute_To_Work* (commute to work/ school)
3. *commute.Commute_From_Work* (commute from work/ school)

class commute.**Commute**

Bases: *activity.Activity*

This class allows for commuting. This class is to be derived from.

end(*p*, *local*)

This function handles the end of an activity.

Parameters

- **p** (person.Person) – the person of interest
- **local** (*int*) – the local location (work or home)

Returns None

end_commute(*p*)

This function ends the commuting activity.

Note: This function is to be overridden

Parameters **p** (person.Person) – the person of interest

Returns None

start(*p*)

This handles the start of the commute activity.

- If the current location of person is at home, the person is going to work, so set the location to `location.OFF_SITE`.
- If the current location of the person is off site, the person is going back home, so set the location to `location.HOME`.

Parameters **p** (person.Person) – the person of interest

Returns None

start_commute (*p*)

This function sets the variables pertaining to starting the commute activity by doing the following:

1. set the status of the person to `location.TRANSIT`
2. set the location of the asset to `location.TRANSIT`
3. set the person's state start time of the commute
4. set the person's state end time for the commute
5. update the asset
6. update the scheduler for the travel need for the end of the commute

Parameters **p** (person.Person) – the person of interest

Returns None

class commute.**Commute_From_Work**

Bases: *commute.Commute*

This class allows for the activity: commuting from work.

advertise (*p*)

This function calculates the score of to commute from work. It does this by doing the following:

1. calculate advertisement only if the person is located at work (off-site)
2. calculate the score S

$$S = \begin{cases} 0 & n_{travel}(t) > \lambda \\ W(n_{travel}(t)) - W(n_{travel}(t + \Delta t)) & n_{travel}(t) \leq \lambda \end{cases}$$

where

- t is the current time
- Δt is the duration of commuting from work [minutes]
- $n_{travel}(t)$ is the satiation for Travel at time t
- λ is the threshold value of Travel
- $W(n)$ is the weight function for Travel

Parameters **p** (person.Person) – the person of interest

Returns the advertised score

Return type float

calc_end_time (*p*)

1. calculate the end time (minutes, universal time) of the commute
2. set the the end time in the person's state

Parameters **p** (person.Person) – the person of interest

Returns None

end(*p*)

This function handles the end of an activity.

Parameters **p** (`person.Person`) – the person of interest

Returns None

end_commute(*p*)

This function sets the variables pertaining to ending the commute activity.

1. Sets the person's state to idle (`state.IDLE`)
2. Updates the asset's state and number of users
3. Sets the travel magnitude
4. Sets the work magnitude to η_{work} to allow for work to be the next activity, even if commute ends begin the work-start time
5. Sets the person's state's end time

Parameters

- **p** (`person.Person`) – person of interest
- **destination** (*int*) – a local location where the commute ends (home or workplace)

Returns None

start(*p*)

This handles the start of the commute activity.

- If the current location of person is at home, the person is going to work, so set the location to `location.OFF_SITE`
- If the current location of the person is off site, the person is going back home, so set the location to `location.HOME`

Parameters **p** (`person.Person`) – the person of interest

Returns None

class commute.**Commute_To_Work**

Bases: *commute.Commute*

This class allows for the activity: commute to work

advertise(*p*)

This function calculates the score of commuting to work by doing the following:

1. calculate advertisement only if the person is located at work (off-site)
2. calculate the score S

$$S = \begin{cases} 0 & n_{travel}(t) > \lambda \\ W(n_{travel}(t)) - W(n_{travel}(t + \Delta t)) & n_{travel}(t) \leq \lambda \end{cases}$$

where

- t is the current time

- Δt is the duration of commuting to work [minutes]
- $n_{travel}(t)$ is the satiation for Travel at time t
- λ is the threshold value of Travel
- $W(n)$ is the weight function for Travel

Parameters **p** (`person.Person`) – the person of interest

Return score the advertisement score

Return type float

calc_end_time (p)

Given the commute duration, store the end time. This function does the following:

1. calculate the end time [universal time] of the commute.
2. store the end time in the person.state

Parameters **p** (`person.Person`) – the person of interest

Returns None

end (p)

This function handles the logistics of ending the commute to work activity.

Parameters **p** (`person.Person`) – the person of interest

Returns None

end_commute (p)

This function handles the logistics concerning ending the commute. Specifically, this function does the following:

1. the asset is freed up from use
2. the magnitude of the travel need is set $n_{travel} = 1$
3. the person's state is set to idle (`state.IDLE`)
4. the person's location is set to the location of the job
5. the asset's location is set to the location of the job
6. the person's income need is set to $n_{income} = \eta_{work}$
7. update the commute to work duration
8. calculate the time until the next leave work event
9. update the schedule for the travel need

Parameters **p** (`person.Person`) – the person of interest

Returns

start (p)

This function handles the start of the commute to work activity. If the current location of person is at home, the person is going to work, so set the location to workplace location (`location.OFF_SITE`)

Parameters **p** (`person.Person`) – the person of interest

Returns None

start_commute (*p*)

This function sets the variables pertaining to starting the commute to work activity. Specifically, the function does the following:

1. set the person's status to `state.TRANSIT`
2. set the asset's location to `location.TRANSIT`
3. set the person's state start time to the current time
4. calculate the end time of commute to work
5. update the asset's update
6. update the scheduler for the travel need to take into account the end of the commute
7. update the scheduler for the income need to take into account the end of the commute

Parameters **p** (`person.Person`) – the person of interest

Returns None

2.1.8 diary module

This module contains code that governs the activity-diaries. Each activity diary contains dataframes that store the activity-diaries for each person. The activity-diaries are the output of the Agent-Based Model of Human Activity Patterns (ABMHAP) simulation.

This module contains class *`diary.Diary`*.

class diary.**Diary** (*t*, *act*, *local*)

Bases: `object`

This class represents the activity-diaries for a person.

Parameters

- **t** (*numpy.ndarray*) – the start times for each activity [universal time, minutes]
- **act** (*numpy.ndarray*) – the activity code done at each time step [integer] (flattened array)
- **local** (*numpy.ndarray*) – the history of location codes done by a person

Variables

- **colnames** (*list*) – the column names for the activity diary in order
- **df** (*pandas.core.frame.DataFrame*) – the activity-diary

create_activity_diary (*t*, *act*, *local*)

This function creates the activity diary for a given agent in the simulation.

The activity diary contains:

1. the start-time and end-time for each activity
2. the activity code

Parameters

- **t** (*numpy.ndarray*) – the simulation times [universal time, minutes]
- **act** (*numpy.ndarray*) – the activity code done at each time step [integer] (flattened array)

Returns a tuple containing the following: the array indices for each activity grouping, the activity diaries in a numerical format, the activity diary in a string format, and the column names for each data type

Each diary is a tuple that contains the following:

1. the day number of the start of the activity
2. the (start-time, end-time) for the activity event
3. the activity code for the activity event
4. the location of the event

get_day_end(*day_start*, *start*, *dt*)
This function gets the day that an activity ends.

Parameters

- **day_start** (*numpy.ndarray*) – the day an activity starts
- **start** (*numpy.ndarray*) – the time an activity starts [hours]
- **dt** (*numpy.ndarray*) – the duration for an activity [hours]

Returns the day an activity ends

Return type numpy.ndarray

get_weekday_data(*df=None*)
This function pulls out data that only corresponds to the weekday data.

Parameters **df** (*pandas.core.frame.DataFrame*) – the activity-diary of interest. If df is None, then use the dataframe associated with the diary object

Returns the activity-diary of data that occur on weekdays

get_weekday_idx(*df=None*)
Get the indices of the data that occurs on weekdays. An activity is considered to be on the weekday if the activity **ends** on Monday - Friday.

Parameters **df** (*pandas.core.frame.DataFrame*) – the activity-diary of interest. If df is None, then use the dataframe associated with the diary object

Returns boolean indices of which activities end during the weekend

Return type numpy.ndarray

get_weekend_data(*df=None*)
This function pulls out data that only corresponds to the weekend data.

Parameters **df** (*pandas.core.frame.DataFrame*) – the activity-diary of interest. If df is None, the use the dataframe associated with the current diary object

Returns an activity-diary of data that occurs on weekends

get_weekend_idx(*df=None*)
Get the indices of the data that occurs on weekend. An activity is considered to be on the weekend if the activity **ends** on Saturday or Sunday.

Parameters **df** (*pandas.core.frame.DataFrame*) – the activity-diary of interest. If df is None, then use the dataframe associated with the diary object

Returns boolean indices of which activities end during the weekend

Return type numpy.ndarray

group_activity(*t*, *y*)
This function groups activities in chronological order.

Parameters

- **t** (*numpy.ndarray*) – the start time for activities
- **y** (*numpy.ndarray*) – the activity code that corresponds with the respective time

Returns a list of each unique group-lists. Each group-list contains a tuple for (time step, activity code)

group_activity_indices(*groups*)
This function returns the indices for each continuous activity in chronological order.

Note: The output is the time step number **not** the value of time

Parameters **groups** (*list*) – a list of tuples of (time step, activity code)

Returns

group_activity_key(*x*)
This is the key function used in groupby in order to group consecutive time-step-activity pairs. This is necessary for creating an activity diary.

Parameters **x** (*tuple*) – the data in the form of (index, (time step, activity code))

Returns the key for sorting (, activity code)

Return type tuple

is_weekend(*day*)
This function returns true if a day is in the weekend and false if it's in a weekday.

Parameters **day** (*numpy.ndarray*) – the day of the weekd

Returns boolean index of whether or not a day is in the weekend (True) or not (False)

Return type numpy.ndarray

same_day(*start*, *dt*)
This function returns true if the activity starts and ends on the same day.

Parameters

- **start** (*numpy.ndarray*) – the time an activity starts [hours]
- **dt** (*numpy.ndarray*) – the duration of an activity, Δt [hours]

Returns a boolean index of whether or not an activity started and ended on the same day

Return type numpy.ndarray

toString()
This function expresses the Diary object as a string

Returns an expression of the diary as a string

Return type string

2.1.9 eat module

This module contains information about the activities associated with eating. This class is an Activity (*activity.Activity*) that gives a Person (*person.Person*) the ability to eat and satisfy the need Hunger (*hunger.Hunger*).

This module contains the following classes:

1. *eat.Eat* (general eating capabilities)
2. *eat.Eat_Breakfast* (eating breakfast)
3. *eat.Eat_Lunch* (eating lunch)
4. *eat.Eat_Dinner* (eating dinner)

class eat.**Eat**

Bases: *activity.Activity*

This class has general capabilities that allow the person to eat in order to satisfy *hunger.Hunger*. This class acts as a parent class and is expected to inherited.

advertise(*p*)

This function handles advertising the score to an agent. This function returns 0.

Note: This function should be overloaded when inherited.

Parameters **p** (person.Person) – the person of interest

Returns the score (0)

Return type float

advertise_help(*p*, *dt*)

This function does some of the logistics needed for *advertise()*.

This function does the following:

1. sets the suggested recharge rate for hunger
2. calculates the score

Parameters

- **p** (person.Person) – the person who is being advertised to
- **dt** (*float*) – the duration of the activity

Returns the score

Return type float

advertise_interruption(*p*)

This function calculates the score of an activity advertisement when a person is going to interrupt an ongoing activity in order to do an eating activity.

This function does the following:

1. temporarily sets the satiation of hunger $n_{hunger}(t) = \eta_{interruption}$
2. calculate the score advertised for the potential eating activity that will interrupt a current activity
3. restores the the satiation for hunger to the original value

Parameters **p** (`person.Person`) – the person of interest

Return score the value of the advertisement

Return type float

end(*p*)

This function ends the eat activity.

Parameters **p** (`person.Person`) – the person whose activity is ending

Returns None

end_meal(*p*)

This function ends the eat activity by doing the following:

1. frees the person's use of the asset
2. sets the state to idle (`state.IDLE`)
3. sets the satiation of hunger
4. set the current meal for the next day
5. set any skipped meals to be on the next day
6. find the the next meal
7. sets the decay rate of hunger
8. update the scheduler so that hunger will trigger the schedule to stop at the next meal
9. set the next meal to the current meal

Parameters **p** (`person.Person`) – The person whose meal is ending.

Returns None

set_end_time(*p*)

This function returns the end time of eating (universal time).

Parameters **p** (`person.Person`) – the person of interest.

Return t_end the end time of eating [minutes, universal time]

Return type int

start(*p*)

This function starts the eating activity.

Parameters **p** (`person.Person`) – The person whose activity is starting.

Returns None

start_meal(*p*)

This function starts the eat activity by doing the following:

1. sets the person's state to busy (`state.BUSY`)
2. set the decay rate of hunger to 0
3. store the start time to the state
4. sets the end time
5. sets the hunger recharge rate
6. updates the asset's state and number of users

7. update the schedule for the hunger need to trigger when the eat activity is scheduled to end

Parameters p (person.Person) – the person who is starting the meal

Returns None

test_func(*p*)

Note: This function is for debugging and has no practical purpose. This function will be removed in the future.

Parameters p (person.Person) – person of interest

Returns None

class eat.**Eat_Breakfast**

Bases: *eat.Eat*

This class is used to handle the logistics for eating breakfast.

advertise(*p*)

This function calculates the score of the activity's advertisement to a person. The activity advertise to the agent if the following conditions are met:

1. the current meal is breakfast
2. the agent's location is at home (`location.HOME`)
3. calculate the score S

$$S = \begin{cases} 0 & n_{hunger}(t) > \lambda \\ W(n_{hunger}(t)) - W(n_{hunger}(t + \Delta t)) & n_{hunger}(t) \leq \lambda \end{cases}$$

where

- t is the current time
- Δt is the duration of eating breakfast [minutes]
- $n_{hunger}(t)$ is the satiation for Hunger at time t
- λ is the threshold value of Hunger
- $W(n)$ is the weight function for Hunger

Parameters p (person.Person) – the person of interest

Return score the advertised score of doing the eat breakfast activity

Return type float

end_meal(*p*)

This function handles the logistics for ending the eat breakfast activity by doing the following:

1. call `eat.end_meal()`
2. if planning to skip lunch, update the lunch event to be the next day

Parameters p (person.Person) – the person who's meal is ending

Returns

start_meal(*p*)

This function handles the logistics for starting the eat activity by doing the following:

1. set the current meal to breakfast
2. call `eat.start_meal()`

Parameters p (`person.Person`) – the person who is starting the eat activity

Returns

class eat.**Eat_Dinner**

Bases: *eat.Eat*

This class is used to handle the logistics for eating dinner.

advertise(*p*)

This function calculates the score of an activities advertisement to a Person. This activity advertises to the agent if the following conditions are met

1. the current meal is lunch
2. the agent's location is at home (`location.HOME`)
3. calculate the score S

$$S = \begin{cases} 0 & n_{hunger}(t) > \lambda \\ W(n_{hunger}(t)) - W(n_{hunger}(t + \Delta t)) & n_{hunger}(t) \leq \lambda \end{cases}$$

where

- t is the current time
- Δt is the duration of eating dinner [minutes]
- $n_{hunger}(t)$ is the satiation for Hunger at time t
- λ is the threshold value of Hunger
- $W(n)$ is the weight function for Hunger

If the threshold is not met, score is 0. The advertisements assume that the duration of the activity is the mean duration.

Parameters p (`person.Person`) – the person of interest

Return score the advertised score of doing the eat dinner activity

Return type float

end_meal(*p*)

This function goes through the logistics of ending the dinner meal by doing the following:

1. calls `end.end_meal()`
2. if breakfast will be skipped, update the lunch event to be 2 days from the current day

Parameters p (`person.Person`) – the person who is finishing the eating dinner event

Returns None

start_meal(*p*)

This function goes through the logistics of starting the dinner meal by doing the following:

1. set the current meal to dinner

2. call `eat.start_meal()`

Parameters **p** (`person.Person`) – the person who is starting the eat dinner event

Returns None

class `eat.`**`Eat_Lunch`**

Bases: *`eat.Eat`*

This class is used to handle the logistics for eating lunch.

`advertise`(*p*)

This function calculates the score of an activities advertisement to a person. The activity advertise to the agent if the following conditions are met:

1. the current meal is lunch
2. the agent's location is at home (`location.HOME`) or the agent's location is at the workplace (`location.OFF_SITE`)
3. calculate the score S

$$S = \begin{cases} 0 & n_{hunger}(t) > \lambda \\ W(n_{hunger}(t)) - W(n_{hunger}(t + \Delta t)) & n_{hunger}(t) \leq \lambda \end{cases}$$

where

- t is the current time
- Δt is the duration of eating lunch [minutes]
- $n_{hunger}(t)$ is the satiation for Hunger at time t
- λ is the threshold value of Hunger
- $W(n)$ is the weight function for Hunger

If the threshold is not met, score is 0. The advertisements assume that the duration of the activity is the mean duration.

Parameters **p** (`person.Person`) – the person of interest

Return score the advertised score of doing the eat lunch activity

Return type float

`end_meal`(*p*)

This function ends the eat lunch activity by doing the following:

1. calls `eat.end_meal()`
2. if dinner is to be skipped, update the dinner event by doing the following:
   - **if the lunch is an interrupting activity**
     - set the time until the next lunch activity
     - update the schedule for the interruption to the next lunch activity
     - set the interruption state to False

Parameters **p** (`person.Person`) – The person whose meal is ending.

Returns None

start_meal(*p*)

This function handles the logistics for starting the eat activity by doing the following:

1. sets the current meal to lunch
2. call `eat.start_meal()`

Parameters **p** (`person.Person`) – the person starting the eat lunch event

Returns None

2.1.10 food module

This module contains information about the asset that allows for the eating activity.

This module contains the following class: *`food.Food`*.

class food.**Food**

Bases: *`asset.Asset`*

This class represents an asset that allows the agent to eat breakfast, eat lunch, and eat dinner.

Activities in this asset:

1. *`eat.Eat_Breakfast`*
2. *`eat.Eat_Lunch`*
3. *`eat.Eat_Dinner`*

2.1.11 home module

This module governs the control of assets used in the simulation. Mainly, the home contains all of the assets used in the simulation for the current version of the code.

This module contains the following class: *`home.Home`*

class home.**Home**(*clock*)

Bases: `object`

Contains all of the physical characteristics of a home/ residence. Currently, the home contains all of the assets within the simulation.

Parameters **clock** (`temporal.Temporal`) – the time

Variables

- **assets** (*`dict`*) – contains a list of all of the assets available in the home.
- **category** (*`int`*) – the type of home
- **clock** (`temporal.Temporal`) – the time
- **id** (*`int`*) – a unique home identification number
- **'location'** (`location.Location`) – the location of the home
- **population** (*`int`*) – the number of people who reside in a home
- **revenue** (*`float`*) – the household revenue

advertise (*p*, *do_interruption=False*, *locale=None*)
This function handles all of the activities' advertisements to a person. This occurs by looping through each asset in the home and collecting a list of advertisements for each activity in each asset. Specifically, the function does the following:

1. loop through each asset
2. if the asset is busy *and* is in the same location of the person
   - **for each activity in the given asset**
     (a) advertise for interrupting activities
     (b) advertise for non interrupting activities
     (c) collect the advertisements

Parameters

- **p** (`person.Person`) – a person to whom the assets are advertising
- **do_interruption** (*bool*) – a flag that indicates whether or not we should advertise for interruptions
- **locale** (*int*) – a local location identifier

Returns the advertisements (score, asset, activity, person) containing the various data for each advertisement: ("score", "asset", "activity", "person") coupling of data type (float, *asset.Asset*, *activity.Activity*, *person.Person*)

Return type dict

initialize (*people*)
Initialize the assets in the home.

Parameters **people** (*list*) – a list of people who reside in the home

Returns None

print_category ()
This function expresses the category variable as a string.

Returns string representation of the category

Return type str

reset ()
This function resets the each asset in the home.

Returns None

set_population (*people*)
Set the population of the house.

Parameters **people** (*list*) – the list of people living in the home

Returns None

set_revenue (*people*)
Sets the revenue of the home by adding the revenue of each person in the home.

Parameters **people** (*list*) – the list of people living in the home

Returns None

toString()
This function expresses the Home object as a string

Return msg the string representation of the home object

Return type str

2.1.12 hunger module

This module contains information about governing the need Hunger.

This module contains the class Hunger (*hunger.Hunger*).

class hunger.**Hunger**(*clock*, *num_sample_points*)
Bases: *need.Need*

This class governs the behavior of the need Hunger need. When Hunger is unstatisfied, the agent feels compelled to eat a meal in order to satisfy the need. Mathematically speaking, Hunger is modeled as linear-behaving need.

Parameters

- **clock** (temporal.Temporal) – the time
- **num_sample_points** (*int*) – the number of temporal nodes in the simulation

Variables

- **category** (*int*) – the category of the need
- **decay_rate** (*float*) – the decay rate of the Hunger need [need/minute]
- **recharge_rate** (*float*) – the recharge rate of the Hunger need [need/min]
- **suggested_recharge_rate** (*float*) – an approximate recharge rate used to calculate the end time of an event before rounding

decay(*status*)
This function decreases the satiation in Hunger by doing the following:

$$n(t+1) = n(t) + m_{decay}$$

> **Warning:** This function may be antiquated and **not used**

Parameters **status** (*int*) – indicates the current status of the person's state (not-used)

Returns None

decay_new(*dt*)
This function sets the default decrease in the Hunger need.

$$n(t + \Delta t) = n(t) + m_{decay} \, \Delta t$$

where

- t is the current time
- Δt is the duration of time to decay the satiation [minutes]
- $n(t)$ is the satiation for Hunger at time t
- m_{decay} the decay rate for Hunger

Parameters **dt** (*int*) – the duration of time [minutes] Δt used to decay the need

Returns None

initialize(*p*)

This function initializes the the Hunger need at the first step of the simulation. The function checks to see whether or not the current time implies that there should be an eating event. The Hunger object is set to the respective state.

This function does the following exactly:

1. initialize all of the meals
2. check to see if a meal should be occurring at the current time
3. if no meals should be occurring
   - figure out the next meal
   - calculate the decay rate for hunger until the next meal
   - calculate the amount of time until the next meal Δt
   - set the current meal
   - update the schedule for the hunger need to be the time the next meal starts
4. if a meal should be occurring
   - get the index of the meal that should be occurring
   - set the current meal
   - calculate the final time of the meal
   - calculate the duration until the end of the next meal Δt
   - set the recharge rate
   - update the scheduler for the hunger need to be the time the current meal should end
5. initialize the start time for each meal

Parameters **p** (`person.Person`) – the person whose hunger need is being initialized

Returns None

is_meal_time(*t*, *the_meal*)

This checks whether or not it is time for a meal.

Parameters

- **t** (*int*) – time of day [minutes]
- **the_meal** (`meal.Meal`) – the respective meal to see whether the current time implies that an eating event should happen

Returns True if the current time is within the time to eat. False, otherwise

Return type bool

is_meal_time_all(*t*, *meals*)

This function checks every meal and sees whether or not the current time implies that there should be an eventing event for a respective meal.

Parameters

- **t** (*int*) – the current time of day [minutes]
- **meals** (*list*) – a list of meals that a person has

Returns a list of boolean flags indicating True or False, indicating whether or not an eating event should occur for the respective meal

Return type list

perceive (*future_clock*)
This gives the result if eat is done now until a later time corresponding to clock.

Parameters **future_clock** (temporal.Temporal) – a clock at a future time

Return out the perceived hunger need association level

Return type float

reset ()
This function resets the values in order for the need to be used in the next simulation.

Returns

set_decay_rate (*t_start*)
This function calculates the decay rate of hunger to the next meal.

Parameters

- **dt** (*int*) – the amount of time Δt to the next meal [minutes]
- **t_start** (*int*) – the start time [in minutes] of the next meal

Returns None

set_decay_rate_new (*dt*)
This function calculates the decay rate of hunger to the next meal.

Parameters **dt** (*int*) – the amount of time Δt to the next meal [minutes]

Returns None

set_recharge_rate (*dt*)
This function calculates the recharge rate of hunger due to eating the current meal.

Parameters **dt** (*int*) – the amount of time Δt it takes to finish a meal [minutes]

Returns None

set_suggested_recharge_rate (*dt*)
This function sets the suggested recharge rate assuming a **linear function** behavior

The suggested recharge rate is based on the duration of the sleeping event and the threshold. The sleep duration is based on the biological data (no rounding).

Parameters **dt** (*int*) – The duration of time Δt of the eating event [minutes]

Returns None

toString ()

Represents the Hunger object as a string.

Return msg the string representation of the huger object

Return type str

2.1.13 income module

This is module contains code for governing the need to work/ be schooled.

This module contains the class *income.Income*.

class income.**Income**(*clock*, *num_sample_points*)

Bases: *need.Need*

This class governs the need dealing with work / school. Recall that income mathematically resembles a step function.

Parameters

- **clock** (temporal.Temporal) – the time
- **num_sample_points** (*int*) – the number of temporal node points in the simulation

decay(*p*)

This function decays the magnitude of the need. Income only decays after the job start time.

1. Find out if it is time to work
2. If it's time to work, set the satiation $n_{income} = \eta_{work}$

Parameters **p** (person.Person) – the person of interest

Returns None

initialize(*p*)

This function is used to initialize the agent's income need at the beginning of the simulation. This function initializes the Person to be at the workplace (location.OFF_SITE) if it is work time. This function does the following:

1. decay the income satiation
2. if the person is supposed to be at work
   - set the person to the workplace location
   - else, set the amount of time until the next work event
3. update the scheduler for the income need

Parameters **p** (person.Person) – the person of interest

Returns None

perceive(*clock*, *job*)

This gives the satiation of income **if** the income need is addressed now.

1. find out if the time associated with clock implies a work time for the person
2. **If it should be work time**
   - the perceived satiation is $\eta_{work} \leq \lambda$
   - else, the perceived satiation is 1.0

Parameters

- **clock** (temporal.Temporal) – the future time the activity the should be perceived to be done
- **job** (occupation.Occupation) – the job

Returns the satiation at the perceived time

Return type float

2.1.14 interrupt module

This module contains code for interrupting a current activity.

This module contains class *interrupt.Interrupt*.

class interrupt.**Interrupt**

Bases: *activity.Activity*

This class allows for the current activity to be interrupted by another activity.

advertise (*p*, *str_need*, *act*)

This function calculates the score of an activities advertisement to a Person. This function does the the following:

1. temporarily sets the value of the Need that must be immediately addressed to a low level.
2. send an advertisement is is made from the potentially interrupting activity
3. calculate the score from the potentially interrupting activity

Parameters

- **p** (person.Person) – the person who is being advertised to
- **str_need** (*int*) – the id of the Need that needs to be addressed, which could potentially cause an interrupting event
- **act** (activity.Activity) – the activity of interest that could immediately interrupt a current activity

Returns the value of the advertisement

Return type float

start (*p*)

This handles the start of an activity.

Parameters **p** (person.Person) – the person of interest

Returns None

2.1.15 interruption module

This class gives an agent the ability to interrupt a current activity.

This module contains class *interruption.Interruption*.

class interruption.**Interruption** (*clock*, *num_sample_points*)

Bases: *need.Need*

This class enables a Person to interrupt a current activity.

Parameters

- **clock** (temporal.Temporal) – the clock governing time in the simulation
- **num_sample_points** (*int*) – the number of time nodes in the simulation

Variables

- **category** (*int*) – the category of the interruption Need
- **activity_start** (*int*) – the category of the (interrupting) activity to start
- **activity_stop** (*int*) – the category of the (interrupted) activity to stop

decay(*p*)

This function sets the default decrease in the Interruption need

Parameters **p** (person.Person) – the person of interest

Returns None

get_time_to_next_work_lunch(*p*)

This function calculates the amount of time [in minutes] until the agent should eat lunch at work.

Parameters **p** (person.Person) – the person of interest

Returns the amount of time [minutes] until the next time the agent should eat lunch at work

initialize(*p*)

Initializes the need at the beginning of the simulation.

Parameters **p** (person.Person) – the person of interest

Returns None

is_lunch_time(*time_of_day*, *meals*)

This function indicates whether it is lunch time or not. This is used in the interruption to stop the work activity and begin the eat lunch activity.

Parameters

- **time_of_day** (*int*) – the time of day [minutes]
- **meals** (*list*) – a list of the meals (*meal.Meal*) for the agents; some of the entries in the list may be None.

Return is_lunch a flag indicating whether it is lunch time

perceive(*clock*)

This gives the result if sleep is done now until a later time corresponding to clock.

Parameters **clock** (temporal.Temporal) – a clock at a future time

Return out the perceived interruption magnitude

reset()

This function resets the Interruption need completely in order to re run the simulation. In this reset the history is also reset.

Returns

reset_minor()

This function resets the interruption need

Returns None

stop_work_to_eat(*p*)

This function checks to see if an interruption should occur to allow a person to start the eating activity while doing the work activity

An agent may stop the work activity to eat lunch if the following conditions are met:

1. the agent is hungry
2. the current activity is work
3. it is lunch time

Parameters **p** (`person.Person`) – the person of interest

Returns None

2.1.16 location module

This module is responsible for containing information about the concept of location.

This module contains class *`location.Location`*.

class `location.`**`Location`**(*geography=1*, *local=0*)

Bases: `object`

This class holds information relevant to the location of persons and assets.

Parameters

- **geography** (*int*) – the geographical location code
- **local** (*int*) – the local location code

Variables

- **geo** (*int*) – the geographical location code within the United States (e.g. north, south, eats, or west)
- **local** (*int*) – the local location code (e.g. home, off site, etc)

print_geo()

Returns the geographical location in a string format

Returns the string representation of the geographical location

Return type str

print_local()

Returns the local location in a string format

Returns the string representation of the local location

Return type str

reset()

This function resets the location to the default value, (`location.HOME`).

Returns None

toString()

This function represents the Location object as a string.

Return msg the string representation of the Location object

Return type str

2.1.17 meal module

This module contains contains information about various meals that an agent would eat in.

This module contains code for class `meal.Meal`.

class meal.**Meal**(*id=0*, *start_mean=390*, *start_std=10*, *start_trunc=1*, *dt_mean=15*, *dt_std=5*, *dt_trunc=1*)

Bases: `object`

This is class contains information about meals (breakfast, dinner, and lunch)

Variables

- **id** (*int*) – the meal type (breakfast, lunch, or dinner)
- **dt** (*int*) – the duration of a meal [minutes]
- **dt_mean** (*int*) – the mean duration of a meal [minutes]
- **dt_std** (*int*) – the standard deviation of meal duration [minutes]
- **dt_trunc** (*int*) – the number of standard deviation in the duration distribution
- **t_start** (*int*) – the start time of a meal [minutes, time of day]
- **t_start_univ** (*int*) – the start time of a meals [minutes, universal time]
- **start_mean** (*int*) – the mean start time of a meal [minutes, time of day]
- **start_std** (*int*) – the standard deviation of start time of a meal [minutes]
- **start_trunc** (*int*) – the number of standard deviation of in the start time distribution
- **f_start** – the start time distribution function
- **f_dt** – the duration distribution function
- **day** (*int*) – the day the meal should occur

initialize(*t_univ*)

At the beginning of the simulation, make sure that the meals are initialized to the proper times (`t_start_univ`) so that they relate to the simulation time (t_univ)

Parameters **t_univ** (*int*) – the simulation time [minutes, universal time]

Returns None

print_id()

This function returns a string representation of the meal id

Returns a string representation of the meal id

Return type str

set_meal(*id*, *start_mean*, *start_std*, *start_trunc*, *dt_mean*, *dt_std*, *dt_trunc*)

This function sets the values associated with the Meal object.

Parameters

- **id** (*int*) – the meal type (breakfast, lunch, or dinner)
- **start_mean** (*int*) – the mean start time of the meal [minutes, time of day]
- **start_std** (*int*) – the standard deviation of start time [minutes]
- **start_turnc** (*int*) – the number of standard deviations in the start time distribution
- **dt_mean** (*int*) – the mean duration of a meal [minutes]

- **dt_std** (*int*) – the standard deviation of meal duration [minutes]
- **dt_trunc** (*int*) – the number of standard deviations in the duration distribution

Returns None

toString()
This function returns a string representation of the Meal object.

Return msg a string representation of the Meal object

Return type str

update(*day*)
Given the day for the meal, update the meal. The following does the following:

1. Update the start time for the meal
2. Update the duration for the meal
3. Update the universal start time for the meal

Parameters **day** (*int*) – the day for the meal to occur

Returns None

update_day()
Update the day for the next meal, given the universal start time for the meal (`t_start_univ`).

Returns None

update_dt()
Sample the duration distribution to get a duration.

Returns None

update_start()
Sample the start time distribution to get a start time.

Returns None

update_start_univ(*day*)
Given the day for the next meal, update the universal start time for the meal.

Parameters **day** (*int*) – the day for the meal

Returns None

2.1.18 my_globals module

This module contains constants and functions that are used for general use.

This module contains information about the following constants:

1. Identifiers for activity codes
2. File names file paths for saving figures for the different demographics
3. File names file paths for saving figures for the different activities

my_globals.**check_filename_extension**(*fname*, *ext*)
This function returns whether or not the given file name has the given filename extension.

Parameters

- **fname** (*str*) – the file name

- **or str ext** (*list*) – a single (or list) of acceptable filename extensions

Returns

my_globals.**fill_out_data**(*t*, *y*)

This function takes an array of activity start times and activity codes from an activity diary and fills out the activity, minute-by-minute in between two adjacent activities.

Parameters

- **t** (*numpy.ndarray*) – the start times in an activity diary
- **y** (*numpy.ndarray*) – the activity codes in an activity diary

Returns

my_globals.**fill_out_time**(*t*)

This function takes an array of activity start times from an activity diary and fills out the time, minute-by-minute in between two adjacent activities

Example, if t = (0, 4, 7) (and $t_{final} = 10$) we get the following:

- (0, 1, 2, 3)
- (4, 5, 6)
- (7, 8, 9, 10)

Parameters **t** (*numpy.ndarray*) – the start times in the activity diary [minutes, universal time]

Returns None

my_globals.**from_periodic**(*t*, *do_hours=True*)

This function returns the time of day in a 24 hour format. It takes the time $t \in [-12, 12)$ and expresses it at time $x \in [0, 24)$ where 0 represents midnight. The same calculation can be done to represent the time in minutes

Parameters

- **t** (*float*) – the time in hours [-12, 12), or the respective minutes [-12 * 60, 12 * 60)
- **do_hours** (*bool*) – a flag to do the calculations in hours (if True)

Returns the time in [0, 24) or in minutes [0, 24 * 60)

my_globals.**get_ecdf**(*data*, *N=100*)

Given data, this function calculates the probability data from the empirical cumulative distribution function (ECDF).

Parameters

- **data** (*float*) – an array containing the relevant data to get the ECDF of
- **N** (*int*) – the amount of samples in calculating the ECDF results

Return y the ECDF

Return type float array

Return x the values sampled for the ECDF

Return type float array

my_globals.**group_time**(*t*)

This function takes data from an activity diary and groups that activity diary into , minute by minute arrays from start to end for each activity (start, start + 1, . . . end-1, end)

Parameters **t** (*numpy.ndarray*) – the start times from an activity diary [minutes, universal time]

Returns the grouped start/end pairs for ech activitiy

Return type list

my_globals.**hours_to_minutes**(*t*)

This function takes a duration of time in hours and converts the time rounded to the nearest minutes.

Parameters **t** (*float*) – a duration of time [hours]

Returns the time in minutes

my_globals.**initialize_random_number_generator**(*seed=None*)

This function initializes the random number generators with a specified seed, if given (i.e., that is if seed is not None). Both the *random* module and the *numpy.random* module's random number generator are seeded. This is useful for reproducing results.

Parameters **seed** (*int*) – the seed for the number generator

Returns

my_globals.**load**(*fname*)

This function loads data from a .pkl file.

Parameters **fname** (*str*) – the file name to be loaded from

Returns the data unpickled

my_globals.**sample**(*data*, *N*)

This function creates N samples of the empirical distribution of the values in the data array.

Parameters

- **data** (*numpy.ndarray*) – the data to sample
- **N** (*int*) – the number of data points to sample

Returns

my_globals.**sample_normal**(*std*, *dx*)

This function samples a normal distribution centered at zero assuming a max and minimum acceptable value [dx, -dx].

Parameters

- **std** (*float*) – the standard deviation
- **dx** (*float*) – the amount of total deviation from the mean allowd

Returns

my_globals.**save**(*x*, *fname*)

This function saves a python variable by pickling it.

Parameters

- **x** – the data to be saved
- **fname** (*str*) – the file name of the saved file. It must end with .pkl

my_globals.**save_diary_to_csv**(*df*, *fname*)

This function saves an activity diary as a .csv file. The output is changed from the original data in the following manor. We add + 1 minute to the end time so that 16:00 - 16:59 (original version) becomes 16:00 - 17:00 (saved version).

Parameters

- **df** (*pandas.core.frame.DataFrame*) – the activity-diary output of the simulation

- **fname** (*str*) – the file name of the saved file. It must end with a .csv extension

Returns

my_globals.**save_zip**(*out_file*, *source_dir*)
This function compresses an entire directory as a zip file.

Parameters

- **out_file** (*str*) – the filename of the save zip file without the .zip extension
- **source_dir** (*str*) – the directory to be compressed

Returns the name of the compressed directory

my_globals.**set_distribution**(*lower*, *upper*, *mu*, *std*)
This function sets the truncated normal probability distribution.

Parameters

- **lower** (*int*) – the lower bound in number of standard deviation from the mean
- **upper** (*int*) – the upper bound in number of standard deviation from the mean
- **mu** (*int*) – the mean
- **std** (*int*) – the standard deviation

Returns the function for the truncated normal distribution

my_globals.**set_distribution_dt**(*lower*, *upper*, *mu*, *std*, *x_min*)
This function set the truncated normal probability distribution subject to the fact that there is an assigned lowest value.

If the lowest value of the normal distribution is lower than the lowest allowed value, change the distribution so that the standard deviation allows the distribution to not be lower than the lowest allowed value.

Parameters

- **lower** (*int*) – the lower bound in number of standard deviation from the mean
- **upper** (*int*) – the upper bound in number of standard deviation from the mean
- **mu** (*int*) – the mean
- **std** (*int*) – the standard deviation
- **x_min** (*int*) – the lowest allowed value

Returns the function for the truncated normal distribution, the standard deviation of the distribution

Return type tuple

my_globals.**to_periodic**(*t*, *do_hours=True*)
This function returns the time of day in a periodic format. It takes the time $t \in [0, 24)$ and expresses it at time $x \in [-12, 12)$ where 0 represents midnight.

Parameters

- **t** (*float*) – the time in hours [0, 24)
- **do_hours** (*bool*) – a flag to do the calculations in hours (if True) or minutes if False

Returns the time in [-12, 12) or minutes [-12 * 60, 12 * 60)

Return type float

2.1.19 need module

This module contains information about governing the various needs that agents have in the simulation.

This module contains the class *need.Need*.

class need.**Need**(*clock*, *num_sample_points*)
Bases: object

This class holds general information about needs.

Parameters

- **clock** (temporal.Temporal) – the clock governing time in the simulation
- **num_sample_points** (*int*) – the number of time nodes in the simulation

Variables

- **category** (*int*) – the need- identifier
- **clock** (temporal.Temporal) – keeps track of the time
- **history** (*float*) – an array containing the magnitude level $[0, 1]$ of the need at all sample times.
- **magnitude** (*float*) – the magnitude of the need (the satiation)
- **t0** (*int*) – this keeps track of the last time the need was addressed
- **threshold** (*float*) – the threshold of the need

decay()
This calculates the amount of decay over a time step.

Note: This function is meant to be overridden.

Returns None

initialize()
This function initializes the state of the need at the very beginning of simulation.

Note: This function is meant to be overridden.

Returns None

print_category()
This function represents the category as a string.

Returns the string representation of the category

Return type str

reset()
This function resets the values in order for the need to be used in the next simulation. This function does the following:

1. sets the satiation to 1.0
2. sets the history to zero

Returns None

toString()

This function represents the Need object as a string.

Return msg the string representation of the Need object

Return type str

under_threshold(*n*)

Compares the value of anNeed's satiation to the threshold.

Parameters n (*float*) – the satiation

Returns True if the satiation is less than or equal to the threshold, False otherwise

Return type bool

weight(*n*)

This function calculates the weight function of a need.

Parameters n (*float*) – the satiation

Returns the weight due to the need

Return type float

2.1.20 occupation module

This module contains info needed for the occupation of a person. In addition, this file also contains functions useful for the module itself.This module contains constants relevant to the occupational information:

- job identifiers
- job categories
- default start time information
- default end time information
- default commuting to work information
- default commuting from work information
- default summer vacation (from school) information

This module contains class *occupation.Occupation*.

class occupation.**Occupation**

Bases: object

This class contains information relevant to an occupation of a Person.

Variables

- **category** (*int*) – the category of the job
- **id** (*int*) – the identifier for the job
- **commute_to_work_dt_mean** (*int*) – the mean duration to commute to work [minutes]
- **commute_to_work_dt_std** (*int*) – the standard deviation to commute to work [minutes]
- **commute_to_work_dt** (*int*) – the duration to commute to work [minutes]

- **commute_to_work_dt_trunc** (*int*) – the number of standard deviation in the commute to work duration distribution
- **commute_to_work_start** (*int*) – the start time for the commute to work activity [minutes]
- **dt_commute** (*int*) – the duration of the commute [minutes]
- **dt** (*int*) – the duration of the work activity [minutes]
- **commute_from_work_dt_mean** (*int*) – the mean duration to commute from work [minutes]
- **commute_from_work_dt_std** (*int*) – the standard deviation to commute from work [minutes]
- **commute_from_work_dt** (*int*) – the duration to commute from work [minutes]
- **commute_from_work_dt_trunc** (*int*) – the number of standard deviations in the commute from work duration distribution
- **t_start_mean** (*int*) – the mean start time for the job [minutes, time of day]
- **t_start_std** (*int*) – the standard deviation of the start time for the job
- **t_start** (*int*) – the start time for the job [minutes, time of day]
- **t_start_univ** (*int*) – the start time for the job [minutes, universal time]
- **work_start_trunc** (*int*) – the number of standard deviations in the work start time distribution
- **day_start** (*int*) – the day the work activity start [minutes]
- **t_end_mean** (*int*) – the mean end time for the job [minutes, time of day]
- **t_end_std** (*int*) – the standard deviation of the end time for the job
- **t_end** (*int*) – the end time for the job [minutes, time of day]
- **t_end_univ** (*int*) – the end time for the job [minutes, universal time]
- **work_end_trunc** (*int*) – the number of standard deviations in the work end time distribution
- **is_employed** (*bool*) – a flag saying whether the agent is employed (True) or not (False)
- **is_same_day** (*bool*) – a flag to see whether the start time and end time of a job are on the same day. If so, True. Else, False. If a person has `NO_JOB`, the flag is set to True
- **'location'** (`location.Location`) – the location of the Occupation
- **wage** (*float*) – the yearly wage for that job [U.S. dollars]
- **work_days** (*list*) – a list of ints, giving the days the job starts
- **f_commute_to_work_dt** – the commute to work duration distribution
- **f_commute_from_work_dt** – the commute from work duration distribution
- **f_work_start** – the work start time distribution
- **f_work_end** – the work end time distribution

is_summer_vacation (*week_of_year*)

This function returns True if the agent should not go to school due to summer vacation. False, otherwise.

Parameters **week_of_year** (*int*) – the week of the year

Returns

print_category()

This function represents the Occupation category as a string

Returns the string representation of a Occupation category

Return type str

print_id()

This function writes the Occupation id as a string

Returns a string representation of the job ID

Return type str

set_commute_distribution()

This function sets the following:

- commute to work duration distribution
- commute from work duration distribution.

Returns None

set_is_job()

This function checks to see if the current job is actually a job (eg. that it is not `NO_JOB`).

Sets self.is_job to True if the Occupation is `NO_JOB`, returns False otherwise

Returns None

set_is_same_day()

This function sets a flag indicating whether or not a job starts and ends on the same day. The function sets `is_same_day` to True if the Occupation start time and end time are within the same day. False, otherwise.

Returns None

set_job_params(*id_job*, *start_mean*, *start_std*, *end_mean*, *end_std*, *commute_to_work_dt_mean*, *commute_to_work_dt_std*, *commute_from_work_dt_mean*, *commute_from_work_dt_std*)

This function sets the Occupation parameters.

Parameters

- **id_job** (*int*) – the job identifier
- **start_mean** (*int*) – the mean start time for the occupation
- **start_std** (*int*) – the standard deviation of the start time for the occupation
- **end_mean** (*int*) – the mean end time for the occupation
- **end_std** (*int*) – the standard deviation for the end time
- **commute_to_work_dt_mean** (*int*) – the mean commute to work duration
- **commute_to_work_dt_std** (*int*) – the standard deviation of the commute to work duration
- **commute_from_work_dt_mean** (*int*) – the mean commute from work duration
- **commute_from_work_dt_std** (*int*) – the standard deviation to commute from work duration

Returns None

set_job_preset()

Sets Occupation to one of the following preset jobs:

- `NO_JOB`, the agent is unemployed
- `STANDARD_JOB`, the agent has a job
- `STUDENT`, the agent attends school (not including college / university)

Returns None

set_no_job()

Set the Occupation to having no job.

Parameters **job** (`occupation.Occupation`) – the job of which to set the attributes

Returns None

set_standard_job()

This function sets the Occupation to the default job. The job has the following characteristics:

- 9:00 - 17:00
- Monday through Friday
- wage of $40,000
- 30 minute commute to work
- 60 minute commute from work

Parameters **job** (`occupation.Occupation`) – the job of which to set the attributes

Returns None

set_student()

This function sets the Occupation to the default schooling behavior. This has the following characteristics:

- 8:00 - 15:00
- Monday through Friday
- wage of $0
- 30 minute commute to school
- 60 minute commute from school

Parameters **job** (`occupation.Occupation`) – the job of which to set the attributes

Returns None

set_work_distribution()

This function sets the following distributions for work:

- work start time distribution
- work end time distribution

Returns None

toString()

Represents the Occupation object as a string

Return msg The representation of the Occupation object as a string

Return type str

update_commute_from_work_dt()
Pull a commute from work duration from the respective distribution.

Returns None

update_commute_to_work_dt()
Pull a commute to work duration from the respective distribution. Also, update the commute to work start time place holder.

Returns None

update_commute_to_work_start()
Update the commute to work start time.

Returns None

update_work_dt()
Update the work duration

Returns None

update_work_end()
Update the work end time.

Returns None

update_work_start()
Update the work start time.

Returns None

occupation.**is_work_time**(*clock*, *job*, *is_commute_to_work=False*)
Given a clock and a job, this function says whether the clock's time corresponds to a time to be at work **or** a time to commute to work.

If $\Delta t > 0$, it indicates when it's time to commute to work.

Parameters

- **clock** (temporal.Temporal) – the time
- **job** (occupation.Occupation) – the job to inquiry
- **is_commute_to_work** (*bool*) – a flag indicating whether we are interested in calculating if it is time to commute to work

Returns a flag indicating if it is / is not work time (or commute time if is_commute_to_work is True)

Return type bool

occupation.**is_work_time_help**(*clock*, *job*)
Given a clock and a job, this function says whether the clock's time corresponds to a time at work.

Parameters

- **clock** (temporal.Temporal) – the time
- **job** (occupation.Occupation) – the job to inquiry

Returns is_work_time: a flag indicating if the time (clock) corresponds to a work time

Return type bool

`occupation.set_grave_shift(job)`

This function sets the Occupation to a grave shift.

- from 22:00 to 6:00
- Monday through Friday
- 30 minute commute to work
- 60 minute commute from work
- wage of $40,0000.

Parameters **job** (`occupation.Occupation`) – the job of which to set the attributes

Returns None

`occupation.set_no_job(job)`

Set the Occupation to having no job.

Parameters **job** (`occupation.Occupation`) – the job of which to set the attributes

Returns None

`occupation.set_standard_job(job)`

This function sets the Occupation to the standard job.

- 9:00 - 17:00
- Monday through Friday
- wage $40,000
- 30 minute commute to work
- 60 minute commute from work

Parameters **job** (`occupation.Occupation`) – the job of which to set the attributes

Returns None

`occupation.set_student(job)`

This function sets a job to the preset values of student occupation.

- 08:00 - 15:00
- Monday through Friday
- wage of $0
- 30 minute commute to school
- 60 minute commute from school

Parameters **job** (`occupation.Occupation`) – the job to set

Returns None

2.1.21 params module

The purpose of this module is to assign parameters necessary to run the Agent-Based Model of Human Activity Patterns (ABMHAP). This module also have constants used in default runs.

This module contains class *`params.Params`*.

class params.**Params**(*num_people*, *num_days*, *num_hours*, *num_min*, *do_minute_by_minute*, *t_start=960*, *dt=1*, *gender=None*, *sleep_start_mean=None*, *sleep_start_std=None*, *sleep_end_mean=None*, *sleep_end_std=None*, *job_id=None*, *do_alarm=None*, *dt_alarm=None*, *bf_start_mean=None*, *bf_start_std=None*, *bf_start_trunc=None*, *bf_dt_mean=None*, *bf_dt_std=None*, *bf_dt_trunc=None*, *lunch_start_mean=None*, *lunch_start_std=None*, *lunch_start_trunc=None*, *lunch_dt_mean=None*, *lunch_dt_std=None*, *lunch_dt_trunc=None*, *dinner_start_mean=None*, *dinner_start_std=None*, *dinner_start_trunc=None*, *dinner_dt_mean=None*, *dinner_dt_std=None*, *dinner_dt_trunc=None*, *work_start_mean=None*, *work_start_std=None*, *work_end_mean=None*, *work_end_std=None*, *commute_to_work_dt_mean=None*, *commute_to_work_dt_std=None*, *commute_from_work_dt_mean=None*, *commute_from_work_dt_std=None*)

Bases: `object`

This class contains the parameters that are needed to parametrize a household.

Note: Some of the class attributes are **not** really used and need to be phased out in future versions of the model. Some of these attribtues are:

- `dt`
- `do_alarm`
- `dt_alarm`

Parameters

- **dt** (*int*) – the step size [in minutes] in the simulation. **This is antiquated and will be removed in future versions.**
- **num_people** (*int*) – the number of people in the household
- **num_days** (*int*) – the number of days in the simulation
- **num_hours** (*int*) – the number of additional hours in the simulation
- **num_min** (*int*) – the number of additional minutes in the simulation
- **t_start** (*int*) – the start time [in minutes] in the simulation
- **gender** (*list*) – the gender of each person in the household
- **sleep_start_mean** (*list*) – the mean sleep start time [in minutes, time of day] for each person in the household
- **sleep_start_std** (*list*) – the standard deviation of sleep start time [in minutes] for each person in the household
- **sleep_end_mean** (*list*) – the mean sleep end time [in minutes, time of day] for each person in the household
- **sleep_end_std** (*list*) – the standard deviation of the sleep end time [in minutes] for each person in the household
- **job_id** (*list*) – the occupation identifier for each person in the household
- **do_alarm** (*list*) – a flag indicating whether or not a person uses an alarm for each person in the household
- **dt_alarm** (*list*) – the duration of time [in minutes] before an alarm goes off before its respective event

- **bf_start_mean** (*numpy.ndarray*) – the mean breakfast start time for each person in the household [minutes, time of day]
- **bf_start_std** (*numpy.ndarray*) – the standard deviation for breakfast start time for each person in the household [minutes]
- **bf_start_trunc** (*numpy.ndarray*) – the number of standard deviations used in the breakfast start time distribution for each person
- **bf_dt_mean** (*numpy.ndarray*) – the mean breakfast duration for each person in the household [minutes]
- **bf_dt_std** (*numpy.ndarray*) – the standard deviation for breakfast duration for each person in the household [minutes]
- **bf_dt_trunc** (*numpy.ndarray*) – the number of standard deviations used in the breakfast duration distribution for each person
- **lunch_dt_mean** (*numpy.ndarray*) – the mean lunch duration for each person in the household [minutes]
- **lunch_dt_std** (*numpy.ndarray*) – the standard deviation for lunch duration for each person in the household [minutes]
- **lunch_dt_trunc** (*numpy.ndarray*) – the number of standard deviations used in the lunch duration distribution for each person
- **lunch_start_mean** (*numpy.ndarray*) – the mean lunch start time for each person in the household [minutes, time of day]
- **lunch_start_std** (*numpy.ndarray*) – the standard deviation for lunch start time for each person in the household [minutes]
- **lunch_start_trunc** (*numpy.ndarray*) – the number of standard deviations used in the lunch start time distribution for each person
- **dinner_start_mean** (*numpy.ndarray*) – the mean dinner start time for each person in the household [minutes, time of day]
- **dinner_start_std** (*numpy.ndarray*) – the standard deviation for dinner start time for each person in the household [minutes]
- **dinner_start_trunc** (*numpy.ndarray*) – the number of standard deviations used in the dinner start time distribution for each person
- **dinner_dt_mean** (*numpy.ndarray*) – the mean dinner duration for each person in the household [minutes]
- **dinner_dt_std** (*numpy.ndarray*) – the standard deviation for dinner duration for each person in the household [minutes]
- **dinner_dt_trunc** (*numpy.ndarray*) – the number of standard deviations used in the dinner duration distribution for each person
- **work_start_mean** (*numpy.ndarray*) – the mean work start time for each person in the household [minutes, time of day]
- **work_start_std** (*numpy.ndarray*) – the standard deviation of work start time for each person in the household [minutes]
- **work_end_mean** (*numpy.ndarray*) – the work end time for each person in the household [minutes, time of day]

- **work_end_std** (*numpy.ndarray*) – the work standard deviation for each person in the household [minutes, time of day]
- **commute_to_work_dt_mean** (*numpy.ndarray*) – the mean duration for commuting to work [minutes] for each person in the household
- **commute_to_work_dt_std** (*numpy.ndarray*) – the standard deviation for commuting to work [minutes] for each person in the household
- **commute_from_work_dt_mean** (*numpy.ndarray*) – the mean duration for commuting from work [minutes] for each person in the household
- **commute_from_work_dt_std** (*numpy.ndarray*) – the standard deviation for commuting from work [minutes] for each person in the household

Variables

- **dt** (*int*) – the step size [in minutes] in the simulation **(this is antiquated and will be removed in future versions)**
- **num_people** (*int*) – the number of people in the household
- **num_days** (*int*) – the number of days in the simulation
- **num_hours** (*int*) – the number of additional hours in the simulation
- **num_min** (*int*) – the number of additional minutes in the simulation
- **t_start** (*int*) – the start time [in minutes] in the simulation
- **gender** (*list*) – the gender of each person in the household
- **sleep_start_mean** (*list*) – the mean sleep start time [in minutes, time of day] for each person in the household
- **sleep_start_std** (*list*) – the standard deviation of sleep start time [in minutes] for each person in the household
- **sleep_end_mean** (*list*) – the mean sleep end time [in minutes, time of day] for each person in the household
- **sleep_end_std** (*list*) – the standard deviation of the sleep end time [in minutes] for each person in the household
- **job_id** (*list*) – the occupation identifier for each person in the household
- **do_alarm** (*list*) – a flag indicating whether or not a person uses an alarm for each person in the household
- **dt_alarm** (*list*) – the duration of time [in minutes] before an alarm goes off before its respective event
- **breakfasts** (*list*) – the breakfast meal objects for each person in the household
- **lunches** (*list*) – the lunch meal objects for each person in the household
- **dinners** (*list*) – the dinner meal objects for each person in the household
- **work_start_mean** (*numpy.ndarray*) – the mean work start time for each person in the household [minutes, time of day]
- **work_start_std** (*numpy.ndarray*) – the standard deviation of work start time for each person in the household [minutes]
- **work_end_mean** (*numpy.ndarray*) – the work end time for each person in the household [minutes, time of day]

- **work_end_std** (*numpy.ndarray*) – the work standard deviation for each person in the household [minutes, time of day]
- **commute_to_work_dt_mean** (*numpy.ndarray*) – the mean duration for commuting to work [minutes] for each person in the household
- **commute_to_work_dt_std** (*numpy.ndarray*) – the standard deviation for commuting to work [minutes] for each person in the household
- **commute_from_work_dt_mean** (*numpy.ndarray*) – the mean duration for commuting from work [minutes] for each person in the household
- **commute_from_work_dt_std** (*numpy.ndarray*) – the standard deviation for commuting from work [minutes] for each person in the household

get_info_mean()

This function stores information about the mean values of the activity-parameters.

Returns a message about the the mean values

Return type str

Returns a list of activity codes in the chronological order in time of day

Return type list

get_info_std(*keys_ordered=None*)

This function stores information about the standard deviation values of the activity-parameters.

Parameters **keys_ordered** (*list*) – this is a list of the names of the activities that are in the same order as the information about the means

Returns a message about the the standard deviation values

Return type str

init_help(*val*, *default_val*)

This function assigns a default value to an attribute in case it was not assigned already. This is, function is particularly useful if the value to be assigned is an array depending on `num_people`. More specifically,

- if val is not None, return val
- if val is None, return the default value (default_val)

Parameters

- **val** – the value to be assigned
- **default_val** – the default value to assign in case val is None

Returns the non-None value

init_meal(*m_id*, *start_mean=None*, *start_std=None*, *start_trunc=None*, *dt_mean=None*, *dt_std=None*, *dt_trunc=None*)

This function returns the data for each person in the household for the respective meal given by "m_id":

- if specific parameters have been assigned, create meals with the respective parameters
- if specific parameters have not been assigned, create meals with the default meal parameters for each meal

Parameters

- **m_id** (*int*) – the identifier of meal type

- **start_mean** (*numpy.ndarray*) – the mean start time for the meal for each person in the household
- **start_std** (*numpy.ndarray*) – the standard deviation of start time for the meal for each person in the household
- **start_trunc** (*numpy.ndarray*) – the amount of standard deviations allowed before truncating the start time distribution for each person in the household
- **dt_mean** (*numpy.ndarray*) – the mean duration for the meal for each person in the household
- **dt_std** (*numpy.ndarray*) – the standard deviation for the meal for each person in the household
- **dt_trunc** (*numpy.ndarray*) – the amount of standard deviations allowed before truncating the duration distribution for each person in the household

Returns the meals for each person in the household

Return type list

init_meal_old(*id*, *start_mean=None*, *start_std=None*, *dt_mean=None*, *dt_std=None*)

This function returns the data for each person in the household for the respective meal given by "id".

Warning: This function may be **not** used because it is antiquated.

Parameters

- **id** (*int*) – the id of meal type
- **start_mean** (*numpy.ndarray*) – the mean start time for the meal for each person in the household
- **dt_mean** (*numpy.ndarray*) – the mean duration for the meal for each person in the household
- **dt_std** (*numpy.ndarray*) – the mean standard deviation for the meal for each person in the household

Returns the meals for each person in the household

Return type list

set_no_variation()

This function sets all of the standard deviations of the activity-parameters to zero for all agents being simulated.

Returns

set_num_steps()

This function calculates and sets the number of time steps ABMHAP will run.

Note: This function may be antiquated.

Return type None

tester()

> **Warning:** This function is just for testing. It checks to see whether the expected dinner time is before the expected end time for work.

Returns

toString()

This function represents the Params object as a string. For now, it prints the tuple (start time, duration, end time) in hours[0, 24] for the following activities:

1. eat breakfast
2. commute to work
3. work
4. eat lunch
5. commute from work
6. eat dinner
7. sleep

in order of start time. The commute activities only have duration information.

Returns the parameter information

2.1.22 person module

This module has code that governs information about the agent.

This module contains information about class `person.Person`.

class person.**Person**(*house*, *clock*, *schedule*)

Bases: `object`

This class contains all of the information relevant for a Person.

A person is parametrized by the following

- a place of residence
- a biology
- social behavior
- a location
- a history of activities and states
- Needs
  1. Hunger
  2. Rest
  3. Income
  4. Travel
  5. Interruption

Parameters

- **house** (`home.Home`) – the Home object the person resides in. (will need to remove this)
- **clock** (`temporal.Temporal`) – the time
- **schedule** (`scheduler.Scheduler`) – the schedule

Variables

- **'bio'** (`bio.Bio`) – the biological characteristics
- **clock** (`temporal.Temporal`) – keeps track of the current time. It is linked to the Universe clock
- **hist_state** (*numpy.ndarray*) – the state history [int] for each time step
- **hist_activity** (*numpy.ndarray*) – the activity history [int] for each time step
- **'home'** (`home.Home`) – this contains the place where the person resides
- **id** (*int*) – unique person identifier
- **'income'** (`income.Income`) – the need that concerns itself with working/school
- **'interruption'** (`interruption.Interruption`) – the need that concerns itself with interrupting an ongoing activity
- **'location'** (`location.Location`) – the location data of a person
- **needs** (*dict*) – a dictionary of all of the needs
- **'rest'** (`rest.Rest`) – the need that concerns itself with sleeping
- **socio** (`social.Social`) – the social characteristics of a Person
- **'state'** (`state.State`) – information about a Person's state
- **'travel'** (`travel.Travel`) – the need that concerns itself with moving from one area to another
- **hist_state** – the state of the person at each time step
- **hist_activity** – the activity code of the person at each time step
- **hist_local** (*numpy.ndarray*) – the location code of the person at each time step
- **H** (*numpy.ndarray*) – the satiation level for each need at each time step
- **need_vector** (*numpy.ndarray*) – the satiation level for each need at a given time step

get_diary()

This function output the result of the simulation in terms of an activity diary.

Returns the activity diary describing the behavior of the agent

Return type *diary.Diary*

print_basic_info()

This function expresses basic information about the Person object as a string by printing the following:

- person identifier
- home identifier
- age
- gender

Returns basic information about the Person

Return type str

reset()

This function rests the person at the beginning of a simulation by doing the following:

1. reset the history
2. reset the state
3. reset the location
4. reset the needs

Note: the clock needs to be set to the beginning of simulation

---

Returns None

reset_history()

This function resets the variables:

1. history of the state
2. history of the activity
3. history of the location

Returns None

reset_needs()

This function resets the needs.

Returns None

toString()

This function represents the Person object as a string.

Returns information about the Person

Return type str

update_history()

This function updates the history of the following values with their current values:

- state history
- location history
- activity history
- need (satiation) history

Returns

update_history_activity()

This function updates the activity history with the current values.

Returns None

update_history_needs()

This function updates the needs (satiation) history with the current values.

Returns None

2.1.23 rest module

This file contains information about the need dealing with Rest.

This module contains class `rest.Rest`.

class `rest.Rest`(*clock*, *num_sample_points*)

Bases: `need.Need`

This class contains relevant information about the rest need.

Parameters

- **clock** (`temporal.Temporal`) – this keeps track of the current time. It is linked to the Universe clock.
- **num_sample_points** (*int*) – the number of temporal nodes in the simulation

decay(*status*)

> **Warning:** This function is old and antiquated.

This function decays the Rest satiation. The satiation only decays if the person is **not** asleep. The decay in sleep

$$\delta = m_{decay} \Delta t$$
$$n(t + \Delta t) = n(t) + \delta$$

where

- m_{decay} is the decay rate
- Δt is the duration of time in 1 time step of simulation [minutes]
- δ is the amount of decay of rest
- $n(t)$ is the satiation at time t

Parameters **status** (*int*) – the current state of a person

Returns None

decay_new(*status*, *dt*)

This function decays Rests' satiation. The satiation only decays if the person is **not** asleep. The decay in sleep is calculated by

$$\delta = m_{decay} \Delta t$$
$$n(t + \Delta t) = n(t) + \delta$$

where

- t the current time
- Δt is the duration of time to decay the satiation [minutes]
- δ the change in the satiation for Rest

- m_{decay} is the decay rate for Rest
- $n(t)$ is the satiation of Rest at time t

Parameters

- **status** (int) – the current state of a person
- **dt** (int) – the duration of time Δt [minutes] used to decay the need

Returns None

initialize(p)

The purpose of this code is to help initialize Rest's satiation and whatever activity that goes with it, depending on any time the simulation begins.

Note: This code is a work in progress.

1. update the sleep start and end time
2. find out if the person should be asleep
3. if the Person is asleep,
   - sets the appropriate duration of sleep left to do
   - sets the rest magnitude to threshold
   - sets the rest recharge rate
   - sets the schedule to trigger when when the person is scheduled to wake up
4. if the Person is not asleep,
   - sets the decay rate
   - set the magnitude
   - sets the schedule to trigger when when the person is scheduled to start sleeping
5. update the schedule for the rest need

Parameters **p** (`person.Person`) – the person of interest

Returns None

is_workday(p)

This function indicates whether or not the sleep event resembles that from a person sleeping for a workday.

Parameters **socio** (`social.Social`) – the social characteristics of the person of interest

Returns True, if the sleep event resembles a workday. False, otherwise.

perceive(*future_clock*)

This functions gives the updated rest magnitude if sleep is done from now until a later time corresponding to clock.

$$\delta = m_{suggested}\Delta t$$

where

- δ is the amount of change in the satiation for Rest

- $m_{suggested}$ is the suggested recharge rate for Rest
- Δt **is the duration of time from now until the future time** given by future_clock

Parameters **`future_clock`** (`temporal.Temporal`) – a clock corresponding to a future time

Returns the perceived rest level

Return type float

`reset()`

This function resets the values in order for the need to be used in the next simulation

Returns None

`set_decay_rate`(*dt*)

This function sets the decay rate. The decay rate (m_{decay}) is assumed to be the slope of a linear function.

$$m_{decay} = -\frac{1-\lambda}{\Delta t}$$

where

- Δt is the duration of time expected to be awake
- λ is the Rest threshold
- m_{decay} is the decay rate for Rest

Parameters **`dt`** (*int*) – the duration of sleep Δt [minutes]

Returns None

`set_recharge_rate`(*dt*)

This function sets the recharge rate. The recharge rate ($m_{recharge}$) is assumed to be the slope of a linear function.

$$m_{recharge} = \frac{1-\lambda}{\Delta t}$$

where

- Δt is the duration of sleep
- λ is the threshold for Rest
- $m_{recharge}$ is the recharge rate for Rest

Parameters **`dt`** (*int*) – the duration of sleep after rounding Δt [minutes]

Returns None

`set_suggested_recharge_rate`(*dt*)

This function sets the "suggested" recharge rate. That is, the rate of recharge assuming exact arithmetic (there is no rounding in time, say to the nearest minute).

$$m_{suggested} = \frac{1-\lambda}{\Delta t}$$

where

- Δt is the duration of sleep

- λ is the Rest threshold
- $m_{suggested}$ is the suggested recharge rate for Rest

Parameters **dt** (*int*) – the duration of sleep Δt [minutes]

Returns None

should_be_asleep(*t_start*, *t_end*)

This function finds out if the person should be asleep for the initialization of the ABMHAP algorithm.

Parameters

- **t_start** (*int*) – start time of sleep [minutes, time of day]
- **t_end** (*int*) – end time of sleep [minutes, time of day]

Returns a flag indicating whether a person should be asleep (if True) or awake (if False)

Return type bool

toString()

Represent the Rest object as a string

Returns the representation of the Rest object

Return type str

2.1.24 scheduler module

This module contains code that is is responsible for controlling the scheduler for the simulation. Note that the simulation does **not** run continuously in from one adjacent time step to the next. Instead the simulation jumps forward in time (i.e. move across multiple time steps in time), stopping only at time steps in which an action could occur. The ability to jump forward in time is controlled by the scheduler.

The scheduler will trigger the simulation to stop skipping time steps for the following reasons:

1. an activity should start
2. an activity should end
3. a need is under threshold

This module contains class *scheduler.Scheduler*.

class scheduler.**Scheduler**(*clock*, *num_people*, *do_minute_by_minute=False*)

Bases: object

This class contains the code for the scheduler. The scheduler is in charge of jumping forward in time and stopping at only potentially relevant time steps. The scheduler keeps track of the needs for every person in in the household and stops at time steps where any person should have an action / need that needs to be addressed.

Parameters

- **clock** (temporal.Temporal) – the time
- **num_people** (*int*) – the number of people in the household

Variables

- **clock** (temporal.Temporal) – the time
- **A** (*numpy.ndarray*) – the schedule matrix of dimension (number of people x number of needs). This matrix contains the times [minutes, universal time] that the simulation should not skip over

- **dt** (*int*) – the duration of time between events
- **t_old** (*int*) – the time [minutes, universal time] of the prior event
- **do_minute_by_minute** (*bool*) – this flag controls whether the schedule should either go through time minute by minute (if True) or jump forward in time (if False). The default is to jump forward in time

get_next_event_time()
This function searches the schedule matrix and finds the next time that that model should handle.

Note: This function is only capable of handling **single-occupancy** households.

Returns the next time [minutes, time of day] that the model should address

Return type int

toString()
This function presents the Scheduler object as a string.

Returns a string representation of the object

update(*id_person*, *id_need*, *dt*)
This function updates the schedule matrix for a given person and need with the duration for the next event, for the respective person-need combination.

Parameters

- **id_person** (*int*) – the person identifier
- **id_need** (*int*) – the need identifier
- **dt** (*int*) – the duration to the next event

Returns None

2.1.25 sleep module

This module contains information about the activity dealing with sleeping. This class is an Activity (*activity.Activity*) that gives a person (*person.Person*) the ability to eat and satisfy the need Rest (*rest.Rest*).

This file contains class *sleep.Sleep*.

class sleep.**Sleep**
Bases: *activity.Activity*

This class is responsible for the act of sleeping, which satisfies the need *rest.Rest*.

advertise(*p*)
This function calculates the score of an activity advertisement to a Person

Parameters **p** (person.Person) – the person being advertised to

Returns the value of the advertisement

Return type float

end(*p*)
This handles the end of the sleep activity.

Parameters **p** (person.Person) – the person of interest

Returns None

end_sleep (*p*)

This function addresses logistics with a person waking up from sleep. More specifically, the function does the following:

1. free the asset from use
2. set the state of the person to idle (`state.IDLE`)
3. update the satiation
4. update the start time and end time
5. set the decay rate
6. update the schedule for the rest need

Parameters **p** (`person.Person`) – the person of interest

Returns None

is_workday (*p*)

This function indicates whether or not the sleep event resembles that from a person sleeping for a workday.

Parameters **p** (`person.Person`) – the person of interest

Returns True, if the sleep event resembles a workday. False, otherwise.

set_end_time (*p*)

This function returns the end time of sleeping. The end time t_{end} is set as follows:

$$\begin{cases} \Delta t & = \frac{1-n(t)}{m_{suggested}} \\ t_{end} & = t + \Delta t \end{cases}$$

where

- Δt is the duration of sleep [minutes]
- t_{end} is the end time of the sleep activity [universal time, minutes]
- $m_{suggested}$ is the suggested recharge rate for Rest
- $n(t)$ is the satiation of Rest at time t

Parameters **p** (`person.Person`) – the person of interest

Return t_end the end time of the sleep event [minutes, universal time]

Return type int

start (*p*)

This handles the start of the sleep activity.

Parameters **p** (`person.Person`) – the person of interest

Returns None

start_sleep (*p*)

This handles what happens when a person goes to sleep. Specifically, this function does the following:

1. the asset's status is updated.
2. the person's state is set to the sleep state (`state.SLEEP`)

3. the end time is calculated
4. the recharge rate is set (according to whether or not it is a workday / non-workday)

Parameters **p** (person.Person) – the person of interest

Returns None

toString()

This function represents the Sleep object as a string

Return msg the representation of the Sleep object

Return type str

2.1.26 social module

This module contains code that governs the social behavior/ characteristics relevant to a Person (*person.Person*).

This module contains class *social.Social*.

class social.**Social**(*age*, *num_meals=3*)

Bases: object

This class contains all of the relevant information governing the person's social behavior.

Note: The current version of ABMHAP does not have any "alarm" functionality / capability. The remnants of any code that governs the use of an alarm will be removed in future updates.

Parameters

- **age** (*int*) – the age of the person [years]
- **num_meals** (*int*) – the number of meals per day

Variables

- **is_child** (*bool*) – this flag is True if the person is a child, False otherwise
- **job** (occupation.Occupation) – the information pertaining the the job
- **num_meals** (*int*) – the number of meals per day a person will eat
- **meals** (*list*) – a list of the meals that a person eats (*meal.Meal*)
- **current_meal** (meal.Meal) – the meal that is currently being eaten **or** if the person is not eating a meal, it is the upcoming meal
- **next_meal** (meal.Meal) – the meal that is after the meal indicated by current_meal
- **uses_alarm** (*bool*) – indicates whether or not a person uses an alarm to wake up
- **is_alarm_set** (*bool*) – indicates whether or not an alarm is set for the current day
- **t_alarm** (*int*) – the time an alarm is supposed to go off [minutes, time of day]

duration_to_next_commute_event(*clock*)

This function is called in in order to calculate the amount of time until the next commute event by doing the following.

1. If the agent is unemployed, return infinity

2. If the time indicates that the agent should be currently working, set the duration to be the length of time remaining at work
3. If the time indicates that the agent should be currently commuting to work, set the duration to be the duration until the commute to work should start
4. If the time indicates that the agent should be currently commuting from work, set the duration to be the amount of time until the commute from work should end
5. Else, calculate the amount of time until the next commute to work event

Note: The only reason this code is place here is because the work activity and the commute activity use it.

Parameters **clock** (`temporal.Temporal`) – the current time

Returns the duration in time [mintues] until the next commute event

Return type int

duration_to_next_meal(*t_univ*)

This function calculates the amount of time until the next meal.

Parameters **t_univ** (*int*) – the current time [minutes, universal time]

Returns the duration to the next meal [minutes]

Return type int

Returns the scheduled next meal

Return type *meal.Meal*

duration_to_work_event(*clock*)

This function is called in in order to calculate the amount of time until the next work event.

1. If the person is employed, the duration to the next meal is set to infinity
2. If the current time is a workday before the time work starts,
   - set the duration to the amount of time until the start of work
3. Else,
   - set the duration until the next work event

Note: The only reason this code is place here is because the work activity and the commute activity use it.

Parameters **clock** (`temporal.Temporal`) – the current time

Returns the duration [minutes] until the next minutes

Return type int

get_current_meal(*time_of_day*)

This function gets the closest meal to the time of day.

Parameters **time_of_day** (*int*) – the time of day

Returns return the meal

Return type *meal.Meal*

get_meal(*id_meal*)
Get the specific meal given by a meal identifier.

Parameters **id_meal** (*int*) – the meal identifier

Returns the meal given by the id

Return type *meal.Meal*

get_next_meal(*clock*)
This function gets the next meal. The meal must occur after the current time.

Parameters **clock** (`temporal.Temporal`) – the current time

Returns the next meal

Return type *meal.Meal*

print_child_status()
This function represents the child status as a string.

Return msg the child/ adult status

Return type str

set_child_flag(*age*)
Sets the flag indicating whether a person is a child.

Parameters **age** (*int*) – the age of the person [years]

Returns None

set_job(*job_id*, *dt=0*)
This function sets the job and the alarm time (if used) that corresponds to the job. The alarm is set, if a person is using the alarm.

Note: The current version of ABMHAP has no alarm capability.

Parameters

- **job_id** (*int*) – job identifier
- **dt** (*int*) – the amount of time before the job start.

Returns None

set_work_alarm(*dt=0*)
This sets the alarm time due to work. If a person uses an alarm, the alarm is set to be "dt" minutes before work time.

Note: The current version of ABMHAP has no alarm capability.

Parameters **dt** (*int*) – the amount of time to wake up before the work event [minutes]

Returns None

test_func (*time_of_day*, *the_meal*)
This is used for testing.

Note: This function has no real purpose and will be deleted in future versions.

Parameters

- **time_of_day** (*int*) – the time of day in minutes
- **the_meal** (meal.Meal) – a meal object

Returns None

toString ()
Represents the Social object as a string.

Returns the representation of the Social object

Return type str

2.1.27 state module

This module contains code that governs information relevant to a person's state.

This module contains class *state.State*.

class state.**State** (*status=0*)
Bases: object

This class contains information relevant to a person state

Parameters **status** (*int*) – the status of the person

Variables

- **'activity'** (activity.Activity) – the particular activity of the asset
- **arg_start** (*list*) – the list of arguments for the start() function
- **arg_end** (*list*) – the list of arguments for the end() function
- **'asset'** (asset.Asset) – the Asset that is being used
- **asset_list** (*list*) –
- **is_init** (*bool*) – this is a flag indicating whether or not the agent is in the initialization state. This state only occurs during the first step of the simulation.
- **status** (*int*) – the status of a person
- **t_end** (*int*) – the end time of a state [minutes, universal time]
- **t_start** (*int*) – the start time of the current state [minutes, universal time]
- **round_dt** (*int*) – the amount of minutes [-1, 0, 1] to round an activity duration
- **dt_frac** (*float*) – the fraction of a minutes subtracted from rounding down from the true projected activity duration
- **do_interruption** (*bool*) – a flag indicating whether the person is interrupting an ongoing activity

end_activity()

This function ends an activity.

Returns None

halt_activity(*p*)

This function runs the halt activity. The function is used by interruptions to stop an activity **immediately** without giving benefits to the need that the halted activity addressed.

Parameters **p** (`person.Person`) – the person of interest

Returns None

print_activity()

The string representation of the activity. This function handles the possibility of the activity being None.

Returns the representation of the activity

Return type str

print_asset()

This function represents the asset as a string. This function handles the possibility of the asset being None.

Returns the representation of the asset

Return type str

print_status()

This function represents the status as a string.

Returns the representation of the status

Return type str

reset(*t_univ*)

Reset the state object to the default behavior at the beginning of the simulation.

Parameters **t_univ** (*int*) – the time of the beginning of the simulation in universal time [seconds]

Returns None

reset_rounding_parameters()

This function resets the rounding parameters to zero.

Returns None

reset_time_status(*t_start*, *status=0*)

This function resets the time information to the current time and sets the status. This function is usually used at the end of an activity.

Parameters

- **t_start** (*int*) – the start time [minutes, universal time]
- **status** (*int*) – the status of the person

Returns None

run_activity(*arg*, *func*)

This function allows an activity to start, end, or halt

Parameters

- **arg** (*list*) – arguments for the func() function
- **func** (*function*) – arguments for the func() function

Returns None

start_activity()

This function starts an activity

Returns None

toString()

This function represents the State object as a string.

Returns the representation of the State object

Return type str

2.1.28 temporal module

This file contains code that handles the time related aspects of this code.

This file contains code for class *temporal.Temporal*. This file also includes other functions that are accessed outside of the Temporal class.

class temporal.**Temporal**(*t_univ=0*)

Bases: object

This class handles all the time keeping responsibilities.

Universal time is the total amount of time in minutes elapsed from the start of the calendar year.

Day 0 at 0:00 corresponds to a universal time of 0

Day 1 at 0:00 corresponds to a universal time of 1 * 24 * 60

Day 359 at 0:00 corresponds to a universal time of 359 * 24 * 60

Parameters **t_univ** (*int*) – the time in universal time [minutes]

Variables

- **day** (*int*) – the day number in the simulation
- **day_of_week** (*int*) – a number 0, 1, 2, … 6 corresponding to days of the week where 0 is Sunday, 1 is Monday, … 6 is Saturday
- **dt** (*int*) – the step size in the simulation [minutes] (**antiquated**)
- **hour_of_day** (*int*) – the hour of the day [0, 23]
- **is_weekday** (*bool*) – a flag indicating if it's a weekday (Monday-Friday) if True. False, otherwise.
- **is_night** (*bool*) – a flag indicating if the time of day is after **dusk** and before **dawn** if True. False, otherwise.
- **min_of_day** (*int*) – the minute of the day [0, 60 - 1]
- **t_univ** (*int*) – the universal time [minutes]
- **time_of_day** (*int*) – the time of the day [minutes], [0, 1, … 24 * 60 -1]
- **season** (*int*) – the season
- **tic** (*int*) – indicates that current tick (each tick corresponds to a step of size dt)
- **step** (*int*) – indicates the current step in the simulation [0, … num_steps-1]

print_day_night()

Represents whether it's day or night as a string

Return msg daytime / nightime status (or an error message, if there is an error)

Return type str

print_day_of_week()

Represents the day of the week as a string

Return msg the day of the week (or an error message, if there is an error)

Return type str

print_season()

Represents the seasons as a string

Returns the season (or an error message, if there is an error)

Return type str

print_time_of_day_to_military()

Represents the time of day as military time.

Returns the time of day in military time

Return type str

reset(*t_univ*)

Reset the temporal object to the initial state.

Parameters **t_univ** (*int*) – The time [seconds, universal time] that the time should be reset to

Returns

set_day_of_week()

This function sets the day of the week. In addition, this function sets the day count, the day of the week, and a flag indicating whether it is a weekday or not.

Returns None

set_season()

This function sets the season. Day 0 is the beginning of winter.

Returns None

set_time()

This function sets all the time variable due to the universal time. This function sets

1. the time of day
2. the day of the week
3. the season
4. the tic.

Returns None

set_time_of_day()

Given the universal time, this function sets the time of day in minutes.

Returns None

toString()

This function represents the Temporal object as a string.

Return msg the representation of the temporal object

Return type str

update_time()
Increments the time by 1 time step.

> **Warning:** This function is outdated!

Returns None

temporal.**convert_cyclical_to_decimal**(*t*)
This function converts cyclical time to decimal time

Parameters **t** (*int*) – the time of day [minutes]

Return out the time of day in [hours]

Return type float

temporal.**convert_cylical_to_universal**(*day*, *time_of_day*)
This function converts a cyclical time to the universal time.

Parameters

- **day** (*int*) – the day of the year
- **time_of_day** (*int*) – the time of day [minutes]

Return t the time in universal time

Return type int

temporal.**convert_decimal_to_min**(*t*)
This function takes in the time of day as a decimal and outputs the time in minutes

Parameters **t** (*float*) – the time of day [0, 24) [hours]

Return out the time of day [minutes]

Return type int

temporal.**convert_universal_to_decimal**(*t_univ*)
This function takes in the universal time and converts it to the time of day in decimal format [0, 24)

Parameters **t_univ** (*int*) – the universal time [minutes]

Return out the universal time [hours]

Return type float

temporal.**print_military_time**(*t*)
Represents the time of day in military time assume that time is in minutes format.

Parameters **t** (*int*) – the time of day [minutes]

Return msg the time of day in military time 00:00

Return type str

2.1.29 transport module

This module contains information about the asset that allows a person to do the following activities:

1. commute to work

2. commute from work

This module contains code for `transport.Transport`.

class `transport.`**`Transport`**

Bases: `asset.Asset`

This class is an asset that allows for commuting.

Activities in this asset:

1. `commute.Commute_To_Work`
2. `commute.Commute_From_Work`

`initialize`(*people*)

This function sets the transport location according to whether or not the Person is commuting to or from work.

Note: This function just sets the transport object to be at the home

Parameters **`people`** (`list`) – a list of people in the simulation

Returns None

2.1.30 travel module

This module contains code for the need associated with the desire to move from one environment to another.

This file contains code for `travel.Travel`.

class `travel.`**`Travel`**(*clock*, *num_sample_points*)

Bases: `need.Need`

This class governs the need for traveling.

Parameters

- **`clock`** (`temporal.Temporal`) – the time
- **`num_sample_points`** (`int`) – the number of temporal nodes in the simulation

`decay`(*p*)

This function decays the satiation. Travel for commuting only decays when the work need is low

Parameters **`p`** (`person.Person`) – the person whose satiation is decaying

Returns None

`decay_work_commute`(*p*)

This decays the satiation level in order to commute to work. For the satiation to decay the person needs the following:

1. the agent should leave the home to go to work
2. the agent should leave work to go home

Parameters **`p`** (`person.Person`) – the person of interest

Returns None

initialize(*p*)

This function initializes the Travel by updating the *scheduler.Scheduler* for Travel

Parameters **p** (person.Person) – the person of interest

Returns None

perceive(*clock*, *job*)

This function gives the satiation for Travel if the Travel need is addressed now.

Note going to work can only happen according to work hours of the job.

Parameters

- **clock** (temporal.Temporal) – the time the need to travel is perceived
- **job** (occupation.Occupation) – the job of the person

Return mag the perceived magnitude of the need

Return type float

2.1.31 universe module

This module contains code that is responsible for running the simulation. This file contains *universe.Universe*.

class universe.**Universe**(*num_steps*, *dt*, *t_start*, *num_people*, *do_minute_by_minute=False*)

Bases: object

The Universe is the governing engine of the simulation. The Universe contains all agents and objects. The Universe is responsible for running the simulation itself.

Parameters

- **num_steps** (*int*) – the number of time steps in the simulation
- **dt** (*int*) – the step size in the simulation [minutes]
- **t_start** (*int*) – the start time for the simulation [minutes, universal time]
- **num_people** (*int*) – the number of people in the household

Variables

- **clock** (temporal.Temporal) – does the timekeeping in the simulation
- **"home"** (home.Home) – the home the persons live in
- **people** (*list*) – a list of all person objects created in the Universe object
- **t_start** (*int*) – the start time for the simulation [minutes, universal time]
- **t_end** (*int*) – the last time for the simulation [minutes, universal time]
- **schedule** (scheduler.Scheduler) – the schedule governing each agent's needs

address_needs(*do_interruption=False*)

This function checks the needs of the agents. The function uses a recursion loop to choose activities.

The recursion:

1. gather all of the advertisements (object-person pairings)
2. assigns 1 activity to the Person with the highest score
3. that Person starts the activity, thereby updating the state of available activities in the home

4. the recursion starts again, where the Home advertises to all remaining Person(s)

Note If no activity will be done this time step to a person, a person is set to the temporary status `state.IDLE_TEMP`, so that the home knows not to advertise to that person.

Parameters **do_interruption** (*bool*) – this flag indicates whether or not advertisements should be made for activities that will interrupt the current activity (if True). If False, the advertisements are made for non-interrupting activities.

Returns None

advertise(*do_interruption=False*)

This function obtains a list of all of the possible activities each person could potentially start in this time step.

Parameters **do_interruption** (*bool*) – this flag indicates whether to make advertisements due to an interrupting activity (if True) or not (if False).

Return ads ads is a list of dictionaries for advertisements:

Dictionary (score, asset, activity, person) containing the various data for each advertisement: (score, asset, activity, person) coupling where the data types are (float, *asset.Asset*, *activity.Activity*, *person.Person*)

Return type list

check_expired_activities()

This function checks for expired activities. If found, end the activities.

Returns None

decay_needs(*dt=None*)

This function decays the needs according to the default behavior. That is, assume the needs are not addressed earlier.

Parameters **dt** (*int*) – the number of minutes to decay the needs by. The default behavior is to use the scheduler's time. If a number is specified, then it should be the number of minutes until the end of the simulation.

Returns None

initial_step()

This function is supposed to run the first time step of the run() loop

1. store the current time
2. address the needs assuming interruption
3. address the needs assuming NO interruption
4. update the history
5. update the clock
6. decay the needs

Note: this function is **NOT** called on in the current implementation yet

Returns None

initialize_needs()

This function initializes the need state of each person at the beginning of simulation based on the current time.

The needs are initialized in this order (the order matters)

1. Rest
2. Hunger
3. Income
4. Travel
5. Interruption

Returns None

print_activity_info(*p*)

This function stores activity info used for testing / developing/ debugging as a string.

Parameters **p** (`person.Person`) – the person of interest

Returns None

reset(*t_univ*)

This code resets the simulation by initializing the agents, home, and clock to the beginning status of the simulation.

This code does the following:

1. reset the clock
2. reset the home
3. reset each person
4. initialize each person
5. initialize the home

Parameters

- **p** (`params.Params`) – the parameters
- **t_univ** (*int*) – the time of the beginning of the simulation [seconds]

Returns

run()

This function is responsible for running the simulation. Instead of running the simulation minute-by-minute, in an effort to reduce run-time, the simulation skips time steps and addresses the agent at times that actions should occur. These times are dictated by the scheduler.

The function proceeds as following:

While the current time is less than the final time:

1. check for expired activities for all agents. If activities should have expired, tell the agent to end them
2. start new activities by addressing the needs for all agents (assuming no interruption)
3. decay the satiation for Interruption for all agents
4. start new activities by addressing the needs for all agents (assuming interruptions only)

5. update the history of the status of each agent
6. find the next time to jump to in the simulation according to the scheduler
7. update the clock to the new time
8. decay the needs for all agents
9. Repeat

For the last time step:

1. update the clock
2. decay the needs for each agent
3. update the history of the status of each agent

Returns

`select_activity`(*ads*)

Given a list of activity advertisements, this function selects the person with the largest activity score and outputs the score, asset, activity, and person.

Parameters **`ads`** (*`list`*) – a list of advertisements for this time step

Return chosen the selected activity advertisement (score, asset, activity, person)

Return type dict

`set_alarm`()

This function sets the alarm for those Person(s) who use an alarm

Note: This function is **NOT** used. There is currently no alarm capability.

Returns None

`test_func`()

Note: This function is just for debugging. This has **no** use in the current version. This function will be removed in future versions.

Returns

`toString`()

Represent the Universe object as a string.

This function outputs the representation of:

1. the clock
2. the home
3. agent person residing in the home

Return msg a representation of the Universe object

Return type str

update_clock (*t*)
This function updates the clock by

1. setting the clock to the given time
2. updating the step of the simulation
3. storing the history of the time nodes used in the simulation

Parameters **t** (*int*) – the time the clock should be set to

Returns

update_history (*step*)
Update the histories for each Person by storing the following:

1. the current state's status
2. the current activity
3. the current satiation value for each needs
4. the current location

Parameters **step** (*int*) – the time step

Returns None

update_history_new ()
Update the histories of each person.

Returns None

2.1.32 work module

This module contains code that governs the activity that gives a person the ability to go to work/ school.

This file contains *work.Work*.

class work.**Work**
Bases: *activity.Activity*

This class allows a person to work / go to school in order to satisfy the need *income.Income*.

advertise (*p*)
This function calculates the score of the advertised work activity to a person

Parameters **p** (person.Person) – the person of interest

Return score

Return type float

end (*p*)
This function handles the end of an activity

Parameters **p** (person.Person) – the person of interest

Returns None

end_work (*p*)
This function sets the variables pertaining to coming back from work by doing the following:

1. free the asset from use

2. set the asset's state to `state.IDLE`
3. set the Income satiation to 1
4. decay the need Travel
5. sample the new work start time
6. sample the new work end time
7. update the scheduler to take into account the next work event

Parameters **p** (`person.Person`) – the person of interest

Returns None

halt (*p*)
This function handles an interruption of an Activity.

Parameters **p** (`person.Person`) – the person of interest

Returns None

halt_work (*p*)
This function interrupts the work behavior by doing the following:

1. frees the current asset
2. the asset's state is set to `state.IDLE`
3. the Interruption satiation is set to 1.0
4. the Interruption's activity start/ stop

Note No benefits of working are given while being interrupted

Parameters **p** (`person.Person`) – the person of interest

Returns None

set_end_time (*p*)
Calculates the end time of work.

Parameters **p** (`person.Person`) – the person of interest

Return t_end the end time [minutes, universal time]

Return type int

start (*p*)
This handles the start of an Activity

Parameters **p** (`person.Person`) – the person of interest

Returns None

start_work (*p*)
This function starts the work activity

- updates that asset's status and number of users
- changes the location of the Person
- updates that person's status
- calculates the end time of the work activity

- update the scheduler for the Income satiation
- update the scheduler for the Travel satiation
- set the day for the work period

Parameters **p** (person.Person) – the person of interest

Returns None

test_func (*p*)

Note: This function is **NOT** used.

Parameters **p** (person.Person) – the person of interest

Returns

2.1.33 workplace module

This module contains code for the asset that allows a person to go to work / school.

This file contains *workplace.Workplace*.

class workplace.**Workplace**

Bases: *asset.Asset*

This class allows a Person to go to work / school.

Activities in this asset: *work.Work*

2.2 Run Directory

These are the files needed to run an instance of ABMHAP with one agent parametrized by user-defined parameters.

The driver for these type of runs is main.py.

Contents:

2.2.1 main module

This is code is runs the simulation for the Agent-Based Model of Human Activity Patterns (ABMHAP) module of the Life Cycle Human Exposure Model (LC-HEM) project.

In order to run the code, do the following:

1. set the user-defined parameters of the simulation in `main_params.py`
2. run the code via

   ```
   > python main.py
   ```

main.**plot** (*p*, *d=None*)

This function plots figures related to the results of the simulation. Specifically, it does the following for the given agent:

1. plots the histograms about the activity data
2. plots cumulative distribution functions (CDFs) of the activity data
3. plots how the satiation changes over time for the all of the needs
4. plots how the weight function values change over time for all of the needs

---

Note: The satiation and weight function plots will **not** be correct unless the simulation was set to run minute by minute. That is, main_params.do_minute_by_minute is set to **True**.

Parameters

- **p** (`person.Person`) – the agent whose information is going to be plotted
- **d** (`diary.Diary`) – the activity diary of the respected agent

Returns

2.2.2 main_params module

This module is responsible for containing parameters that main.py uses to control the simulation. The user should set the parameters in this module **before** running the driver `main.py`

`main_params.`**`set_no_variation`**(*num_people*)

This function sets the standard deviations in all of the activity-parameters to zero.

Parameters **num_people** (*int*) – the number of people in the simulation

Returns a tuple of the standard deviations of all of the activity-parameters

2.2.3 scenario module

This file contains information to run the Agent-Based Model of Human Activity Patterns (ABMHAP) in in different simulation scenarios in which the agent has a user-defined parametrization.

The following classes are in this module

1. *scenario.Scenario*
2. *scenario.Solo*
3. *scenario.Duo*

class `scenario.`**`Duo`**(*hhld_params*)

Bases: *scenario.Scenario*

This class parametrizes / runs a simulation scenario for the cases where two Singleton (*singleton.Singleton*) persons live in the same residence.

Note: This scenario is used in order to check for activity conflicts among 2 agents living in the same household. Currently it is used primarily as a debugging tool.

Parameters **hhld_params** (`params.Params`) – the parameters for the household that contain relevant information for the simulation

class scenario.**Scenario**(*hhld_params*)

Bases: `object`

This class governs what a simulation scenario consists of.

Parameters **hhld_params** (`params.Params`) – the parameters for the household that contain relevant information for the simulation

Variables

- **id** (*int*) – the scenario identifier number
- **u** (`universe.Universe`) – the universe object for the simulation
- **'params'** (`params.Params`) – the parameters needed that control the simulation

activity_diary()

This function returns the activity diary for each person

Each person will attain the following tuple

1. grouping of the index for each activity
2. the day, (start-time, end-time), activity code, and location for each activity-event, in a numeric format
3. the same as above in a string format

Returns

default_location()

Sets the default location for all Person's to be be at the home. This location may be overridden later in the initialization of persons.

Returns None

initialize()

This function initializes the scenario before the simulation scenario is run

More specifically, the function does the following:

1. Sets the state and location for each person
2. Sets the home
3. Initialize the initial need-association states for the Person(s) and Home

Returns None

run()

This function initializes the scenario and then runs the ABMHAP simulation.

Returns None

set_home()

This function sets aspects of the home in order to run the simulation scenario.

More specifically, the function does the following

1. set the home revenue
2. set the home population

Returns None

set_state()

This function initializes the scenario in order to run the simulation. More specifically, this function does the following:

1. For each Person, the following is set:
   (a) identification number
   (b) the state

   Returns None

class scenario.**Solo**(*hhld_params*)

Bases: *scenario.Scenario*

This class parametrizes / runs a simulation scenario for the Singleton (*singleton.Singleton*) person.

Parameters **hhld_params** (params.Params) – the parameters for the household that contain relevant information for the simulation

2.2.4 singleton module

This file contains information for creating the default agent that represents a person that lives alone in the home. Singleton will be the name of this type of agent.

This module contains *singleton.Singleton*.

class singleton.**Singleton**(*house*, *clock*, *schedule*)

Bases: *person.Person*

Singleton default is a person that has the following characteristics

1. female
2. 30 years old
3. goes to bed at 22:00 and sleeps for 8 hours
4. lives alone and has no children
5. works the Standard Job
6. eats breakfast at 7:30 for 15 minutes, lunch at 12:00 for 30 minutes, and dinner at 19:00 for 45 minutes

Parameters

- **house** (home.Home) – the place of residence
- **clock** (temporal.Temporal) – the clock running in the simulation
- **schedule** (scheduler.Scheduler) – the schedule for the agent

print_params()

This function prints the activity-parameter means in chronological order of start time. This results in the ability to print the mean daily routine.

Returns a representation of the parameters of the agent in increasing values of start time

Return type str

set(*param*, *idx*)

This function sets the Singleton's parameters.

The function does the following:

1. sets the biology
2. sets the job information
3. sets the alarm
4. sets the meal information

Parameters

- **param** (params.Params) – parameters describing the household
- **idx** (*int*) – the respective index number of the person of interest in the household

Returns None

2.3 Run_chad Directory

These are the files needed to run an instance of ABMHAP as a monte-carlo simulation consisting of multiple households of agents parametrized with the data from the Consolidated Human Activity Database (CHAD).

These simulations may be run in parallel.The driver for these type of runs is driver.py.

Contents:

2.3.1 analysis module

This file contains capability for analyzing results from the comparisons between CHAD (Consolidated Human Activity Database) data and the performance of ABMHAP (Agent-Based Model of Human Activity Patterns).

> **Warning:** This modules is old and may or may not be used.

analysis.**get_error**(*chad_raw*, *chad_stats*, *col_name*, *abm_all*, *do_cyclical=False*)

> **Warning:** I do not think this function is used.

Parameters

- **chad_raw** (*pandas.core.frame.DataFrame*) – the CHAD activity data being compared to
- **chad_stats** (*pandas.core.frame.DataFrame*) – the relevant statistics for the CHAD activity of the person (PID) being modeled
- **col_name** (*str*) – the name of the column of the CHAD data being compared

  example col_name = “dt” would allow access for chad_raw[“dt”]
- **abm_all** – the ABM simulation data for the simulated person’s activity with respected to the quantity from col_name.

  example if col_name = “dt”, then abm_all should contain the duration data
- **do_cyclical** (*bool*) – indicates when to cast data in a “cyclical” form. As in, [0, 24 * HOURS_2_MIN - 1] [minutes]

Returns the L2 (sum of squares) absolute error for each agent, the L2 (sum of squares) relative error for each agent

Return type float array, float array

`analysis.get_moments(abm_data)`

This function takes in all of the ABMHAP simulation data [in minutes] for a particular activity and returns the moments (mean and standard deviation) [hours] for each person in the simulation.

Parameters **abm_data** (*list*) – the list of ABMHAP of activity data in minutes per person

Returns the mean and standard deviation for each person in the simulation

Return type numpy.ndarray, numpy.ndarray

`analysis.get_proper_data(df_dt, df_start, df_record, x)`

This function gets the duration, start time, and record data for a given activity

Warning: This function may not be used.

Parameters

- **df_dt** (*pandas.core.frame.DataFrame*) – the duration statistical data for a given activity
- **df_start** (*pandas.core.frame.DataFrame*) – the start time statistical data for a given activity
- **df_record** (*pandas.core.frame.DataFrame*) – the CHAD records for the given activity
- **x** (*chad_params.CHAD_params*) – the parameters that limit sampling the CHAD data

Returns

`analysis.get_verification_info(demo, key_activity, sampling_params, fname_stats=None)`

This function gets the CHAD parameters for each household

Note: Sometimes record dataframe can be null. I should remove the sampling of the record code **and** output

Parameters

- **demo** (*int*) – the demographic identifier
- **key_activity** (*int*) – the identifier for the activity (from my_globals)
- **sampling_params** (list of *chad_params.CHAD_params*) – the parameters that the limit the sampling of the CHAD data

Returns the activity moments of the start time data from CHAD used to verify the ABMHAP, the activity moments of the end time data from CHAD used to verify the ABMHAP, the activity moments of the duration data from CHAD used to verify the ABMHAP, the activity records data from CHAD used to verify the ABMHAP

Return type pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame

analysis.**save_figures**(*figs*, *fnames*)
This function saves figures in a python pickle file, so that the data may be accessed again.

Parameters

- **figs** (*list of figures*) – figures for duration and start time for an activity
- **fnames** (*list of str*) – file names to save the data in figs
- **fdir** (*str*) – the directory in which to save the files

Returns

2.3.2 analyzer module

For a given activity, this code compares how well the ABMHAP matches the CHAD data. This is useful for the quality assurance and quality control (QAQC). This code compares the distributions of mean activity-start-time and mean activity duration for the respective activity.

> **Warning:** I will need to update this definition

analyzer.**get_activity_data**(*df*, *act*)
This function returns the activity data from an activity diary of given respective agent.

Parameters

- **df** (*pandas.core.frame.DataFrame*) – the activity diary
- **act** (*int*) – ABMHAP activity code

Returns an activity diary containing information from the given activity

Return type pandas.core.frame.DataFrame

analyzer.**get_moments**(*abm_list*, *do_periodic=False*)
This function calculates both the mean and the standard deviation for start time, end time, and duration for a given activity for each agent simulated.

Parameters

- **abm_list** (*list of pandas.core.frame.DataFrame*) – the activity diary for a given activity for each agent
- **do_periodic** (*bool*) – this flag indicates whether (if True) or not (if False) to do the analysis on a time scale that is [-12, 12). This is useful for activities that may occur over midnight.

Returns information containing the the mean and standard deviation information for start time, end time, and duration for the simulated agents for a given activity

Return type numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray

analyzer.**get_simulation_data**(*df_list*, *act*)
This function obtains the simulation data for a given activity from each each agent in the simulation.

Parameters

- **df_list** (*list of pandas.core.frame.DataFrame*) – the activity diaries for each agent in the entire simulation
- **act** (*int*) – the ABMHAP activity code

Returns the activity diary containing information for the respective activity for each agent simulated

Returns list of pandas.core.frame.DataFrame

`analyzer.get_verify_fpath` (*fdir*, *act_codes*)

This function returns the directories corresponding to the specified activity codes where figures of activities will be stored for a specific simulation.

Parameters

- **fdir** – the file path to the directory of the figure data for a specific simulation.
- **act_codes** (*list of int*) – the ABMHAP activity codes for a given activity

Returns the directories corresponding to the specified activity codeds where figures of the activities will be stored

Return type list of str

`analyzer.load_plot_data` (*fname*)

This function loads the data from pickled (.pkl) figures. This assumes that the figures plotted the ABM data first and then the CHAD data.

Parameters **fname** (*str*) – the filename of the saved figure (.pkl)

Returns the x and y data for the ABM and CHAD data, respectively

Return type numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray

`analyzer.plot_cdf` (*data_abm*, *data_chad*, *xlabel*, *title*)

This function plots the CDF for data related to the ABM and CHAD

Parameters

- **data_abm** (*numpy.ndarray*) – the ABM data to be plotted
- **data_chad** (*numpy.ndarray*) – the CHAD data to be plotted
- **xlabel** (*str*) – the x-axis label
- **title** (*str*) – the title of the plot

Returns

`analyzer.plot_cdf_new` (*data_abm*, *data_chad*, *fid*, *title*, *xlabel*, *do_periodic=False*)

This function plots the cumulative distribution function (CDF) comparing the ABM and CHAD data for a given activity

Parameters

- **data_abm** (*numpy.ndarray*) –
- **data_chad** (*numpy.ndarray*) –
- **fid** (*int*) – the figure identifier
- **title** (*str*) – the title of the figure
- **xlabel** (*str*) – the label of the x-axis
- **do_periodic** (*bool*) – this flag indicates whether (if True) or not (if False) to convert the data to a time scale that is [-12, 12). This is useful for activities that may occur over midnight.

Returns the figure of the CDF

Return type matplotlib.figure.Figure analyzer.**plot_verify_dt**(*act*, *data_abm*, *data_chad*, *fid*, *do_save_fig=False*, *fpath=''*)

This function plots the cumulative distribution function (CDFs) in order to compare the duration data for the given activity from the ABMHAP simulation to the CHAD data.

Parameters

- **act** (*int*) – the ABMHAP activity data
- **data_abm** (*numpy.ndarray*) – the ABMHAP duration data
- **data_chad** (*numpy.ndarray*) – the CHAD duration data
- **fid** (*int*) – the figure identifier
- **do_save_fig** (*bool*) – a flag indicating whether (if True) or not (if False) to save the figure
- **fpath** (*str*) – the file path to the directory in which to save the figure

Returns

analyzer.**plot_verify_end**(*act*, *data_abm*, *data_chad*, *fid*, *do_save_fig=False*, *fpath=''*)

This function plots the cumulative distribution function (CDFs) in order to compare the end time data for the given activity from the ABMHAP simulation to the CHAD data.

Parameters

- **act** (*int*) – the ABMHAP activity data
- **data_abm** (*numpy.ndarray*) – the ABMHAP end time data
- **data_chad** (*numpy.ndarray*) – the CHAD end time data
- **fid** (*int*) – the figure identifier
- **do_save_fig** (*bool*) – a flag indicating whether (if True) or not (if False) to save the figure
- **fpath** (*str*) – the file path to the directory in which to save the figure

Returns

analyzer.**plot_verify_start**(*act*, *data_abm*, *data_chad*, *fid*, *do_save_fig=False*, *fpath=''*)

This function plots the cumulative distribution function (CDFs) in order to compare the start time data for the given activity from the ABMHAP simulation to the CHAD data.

Parameters

- **act** (*int*) – the ABMHAP activity data
- **data_abm** (*numpy.ndarray*) – the ABMHAP start time data
- **data_chad** (*numpy.ndarray*) – the CHAD start time data
- **fid** (*int*) – the figure identifier
- **do_save_fig** (*bool*) – a flag indicating whether (if True) or not (if False) to save the figure
- **fpath** (*str*) – the file path to the directory in which to save the figure

Returns

analyzer.**run**(*num_process*, *num_hhld*, *num_batch*)

This function runs the simulations.

Parameters

- **num_process** (*int*) – the number of processors (cores)
- **num_hhld** (*int*) – the number of households per core per batch
- **num_batch** (*int*) – the number of batches

Returns the results of the simulation

Return type *driver_result.Driver_Result*

analyzer.**verify**(*trial_code*, *demo*, *chad_param_list*, *df_list*, *do_plot*, *do_print=False*, *fdir=None*)

This code compares the results of the ABM to the CHAD data by comparing the cumulative distribution function (CDF) of the duration and start times predicted by the ABM and that of respective CDFs from the CHAD data.

Parameters

- **trial_code** (*int*) – the trial code identifier
- **demo** (*int*) – the demographic identifier
- **chad_param_list** (list of *chad_params.CHAD_params*) – that limit the CHAD parameters sampling in initializing the households
- **df_list** (*list of pandas.core.frame.DataFrame*) – contains the activity diaries for each household
- **do_plot** (*bool*) – a flag to indicate whether (True) or not (False) to plot
- **do_print** (*bool*) – a flag to indicate whether (True) or not (False) to print various messages to the screen
- **fdir** (*list*) – a list of file directories needed to save the figures

Returns

2.3.3 chad_demography module

This module contains code that handles accessing the Consolidated Human Activity Database (CHAD) data for various demographics.

This module contains *chad_demography.CHAD_demography*.

class chad_demography.**CHAD_demography**(*demographic*)

Bases: object

This class contains the common functionality with accessing the CHAD data files relevant to different demographics.

Parameters **demographic** (*int*) – the demographic identifier

Variables

- **demographic** (*int*) – the demographic identifier
- **fname_zip** (*str*) – the name of the file (.zip) that contains the CHAD data
- **fname_stats_commute_to_work** (*dict*) – the file names for the CHAD longitudinal data (start time, end time, duration, and records) to be sampled for commuting to work
- **fname_stats_commute_from_work** (*dict*) – the file names for the CHAD longitudinal data (start time, end time, duration, and records) to be sampled for commuting from work
- **fname_stats_eat_breakfast** (*dict*) – the file names for the CHAD longitudinal data (start time, end time, duration, and records) to be sampled for eating breakfast

- **fname_stats_eat_dinner** (*dict*) – the file names for the CHAD longitudinal data (start time, end time, duration, and records) to be sampled for eating dinner
- **fname_stats_eat_lunch** (*dict*) – the file names for the CHAD longitudinal data (start time, end time, duration, and records) to be sampled for eating lunch
- **fname_stats_school** (*dict*) – the file names for the CHAD longitudinal data (start time, end time, duration, and records) to be sampled for schooling
- **fname_stats_sleep** (*dict*) – the file names for the CHAD longitudinal data (start time, end time, duration, and records) to be sampled for sleeping
- **fname_stats_work** (*dict*) – the file names for the CHAD longitudinal data (start time, end time, duration, and records) to be sampled for working
- **n_commute_from_work** (*int*) – the minimum number of events needed in sampling from CHAD longitudinal data for commuting from work
- **n_commute_to_work** (*int*) – the minimum number of events needed in sampling from CHAD longitudinal data for commuting to work
- **n_eat_breakfast** (*int*) – the minimum number of events needed in sampling from CHAD longitudinal data for eating breakfast
- **n_eat_dinner** (*int*) – the minimum number of events needed in sampling from CHAD longitudinal data for eating dinner
- **n_eat_lunch** (*int*) – the minimum number of events needed in sampling from CHAD longitudinal data for eating lunch
- **n_school** (*int*) – the minimum number of events needed in sampling from CHAD longitudinal data for schooling
- **n_sleep** (*int*) – the minimum number of events needed in sampling from CHAD longitudinal data for sleeping
- **n_work** (*int*) – the minimum number of events needed in sampling from CHAD longitudinal data for working
- **work_start_mean_min** (*float*) – the minimum mean start time for working when sampling CHAD data
- **work_start_mean_max** (*float*) – the maximum mean start time for working when sampling CHAD data
- **work_start_std_max** (*float*) – the maximum standard deviation for start time for working when sampling CHAD data
- **work_end_mean_min** (*float*) – the minimum mean end time for working when sampling CHAD data
- **work_end_mean_max** (*float*) – the maximum mean end time for working when sampling CHAD data
- **work_end_std_max** (*float*) – the maximum standard deviating for end time for working when sampling CHAD data
- **work_dt_mean_min** (*float*) – the minimum mean duration for working when sampling CHAD data
- **work_dt_mean_max** (*float*) – the maximum mean duration for working when sampling CHAD data

- **work_dt_std_max** (*float*) – the maximum standard deviation for working when sampling CHAD data
- **school_start_mean_min** (*float*) – the minimum mean start time for schooling when sampling CHAD data
- **school_start_mean_max** (*float*) – the maximum mean start time for schooling when sampling CHAD data
- **school_start_std_max** (*float*) – the maximum standard deviation for start time for schooling when sampling CHAD data
- **school_end_mean_min** (*float*) – the minimum mean end time for schooling when sampling CHAD data
- **school_end_mean_max** (*float*) – the maximum mean end time for schooling when sampling CHAD data
- **school_end_std_max** (*float*) – the maximum standard deviating for end time for schooling when sampling CHAD data
- **school_dt_mean_min** (*float*) – the minimum mean duration for schooling when sampling CHAD data
- **school_dt_mean_max** (*float*) – the maximum mean duration for schooling when sampling CHAD data
- **school_dt_std_max** (*float*) – the maximum standard deviation for schooling when sampling CHAD data
- **commute_to_work_start_mean_min** (*float*) – the minimum mean start time for commuting to work when sampling CHAD data
- **commute_to_work_start_mean_max** (*float*) – the maximum mean start time for commuting to work when sampling CHAD data
- **commute_to_work_start_std_max** (*float*) – the maximum standard deviation for start time for commuting to work when sampling CHAD data
- **commute_to_work_end_mean_min** (*float*) – the minimum mean end time for commuting to work when sampling CHAD data
- **commute_to_work_end_mean_max** (*float*) – the maximum mean end time for commuting to work when sampling CHAD data
- **commute_to_work_end_std_max** (*float*) – the maximum standard deviating for end time for commuting to work when sampling CHAD data
- **commute_to_work_dt_mean_min** (*float*) – the minimum mean duration for commuting to work when sampling CHAD data
- **commute_to_work_dt_mean_max** (*float*) – the maximum mean duration for commuting to work when sampling CHAD data
- **commute_to_work_dt_std_max** (*float*) – the maximum standard deviation for commuting to work when sampling CHAD data
- **eat_breakfast_start_mean_min** (*float*) – the minimum mean start time for eating breakfast when sampling CHAD data
- **eat_breakfast_start_mean_max** (*float*) – the maximum mean start time for eating breakfast when sampling CHAD data

- **eat_breakfast_start_std_max** (*float*) – the maximum standard deviation for start time for eating breakfast when sampling CHAD data
- **eat_breakfast_end_mean_min** (*float*) – the minimum mean end time for eating breakfast when sampling CHAD data
- **eat_breakfast_end_mean_max** (*float*) – the maximum mean end time for eating breakfast when sampling CHAD data
- **eat_breakfast_end_std_max** (*float*) – the maximum standard deviating for end time for eating breakfast when sampling CHAD data
- **eat_breakfast_dt_mean_min** (*float*) – the minimum mean duration for eating breakfast when sampling CHAD data
- **eat_breakfast_dt_mean_max** (*float*) – the maximum mean duration for eating breakfast when sampling CHAD data
- **eat_breakfast_dt_std_max** (*float*) – the maximum standard deviation for eating breakfast when sampling CHAD data
- **commute_from_work_start_mean_min** (*float*) – the minimum mean start time for commuting from work when sampling CHAD data
- **commute_from_work_start_mean_max** (*float*) – the maximum mean start time for commuting from work when sampling CHAD data
- **commute_from_work_start_std_max** (*float*) – the maximum standard deviation for start time for commuting from work when sampling CHAD data
- **commute_from_work_end_mean_min** (*float*) – the minimum mean end time for commuting from work when sampling CHAD data
- **commute_from_work_end_mean_max** (*float*) – the maximum mean end time for commuting from work when sampling CHAD data
- **commute_from_work_end_std_max** (*float*) – the maximum standard deviating for end time for commuting from work when sampling CHAD data
- **commute_from_work_dt_mean_min** (*float*) – the minimum mean duration for commuting from work when sampling CHAD data
- **commute_from_work_dt_mean_max** (*float*) – the maximum mean duration for commuting from work when sampling CHAD data
- **commute_from_work_dt_std_max** (*float*) – the maximum standard deviation for commuting from work when sampling CHAD data
- **sleep_start_mean_min** (*float*) – the minimum mean start time for sleeping when sampling CHAD data
- **sleep_start_mean_max** (*float*) – the maximum mean start time for sleeping when sampling CHAD data
- **sleep_start_std_max** (*float*) – the maximum standard deviation for start time for sleeping when sampling CHAD data
- **sleep_end_mean_min** (*float*) – the minimum mean end time for sleeping when sampling CHAD data
- **sleep_end_mean_max** (*float*) – the maximum mean end time for sleeping when sampling CHAD data

- **sleep_end_std_max** (*float*) – the maximum standard deviating for end time for sleeping when sampling CHAD data
- **sleep_dt_mean_min** (*float*) – the minimum mean duration for sleeping when sampling CHAD data
- **sleep_dt_mean_max** (*float*) – the maximum mean duration for sleeping when sampling CHAD data
- **sleep_dt_std_max** (*float*) – the maximum standard deviation for sleeping when sampling CHAD data
- **eat_lunch_start_mean_min** (*float*) – the minimum mean start time for eating lunch when sampling CHAD data
- **eat_lunch_start_mean_max** (*float*) – the maximum mean start time for eating lunch when sampling CHAD data
- **eat_lunch_start_std_max** (*float*) – the maximum standard deviation for start time for eating lunch when sampling CHAD data
- **eat_lunch_end_mean_min** (*float*) – the minimum mean end time for eating lunch when sampling CHAD data
- **eat_lunch_end_mean_max** (*float*) – the maximum mean end time for eating lunch when sampling CHAD data
- **eat_lunch_end_std_max** (*float*) – the maximum standard deviating for end time for eating lunch when sampling CHAD data
- **eat_lunch_dt_mean_min** (*float*) – the minimum mean duration for eating lunch when sampling CHAD data
- **eat_lunch_dt_mean_max** (*float*) – the maximum mean duration for eating lunch when sampling CHAD data
- **eat_lunch_dt_std_max** (*float*) – the maximum standard deviation for eating lunch when sampling CHAD data
- **eat_dinner_start_mean_min** (*float*) – the minimum mean start time for eating dinner when sampling CHAD data
- **eat_dinner_start_mean_max** (*float*) – the maximum mean start time for eating dinner when sampling CHAD data
- **eat_dinner_start_std_max** (*float*) – the maximum standard deviation for start time for eating dinner when sampling CHAD data
- **eat_dinner_end_mean_min** (*float*) – the minimum mean end time for eating dinner when sampling CHAD data
- **eat_dinner_end_mean_max** (*float*) – the maximum mean end time for eating dinner when sampling CHAD data
- **eat_dinner_end_std_max** (*float*) – the maximum standard deviating for end time for eating dinner when sampling CHAD data
- **eat_dinner_dt_mean_min** (*float*) – the minimum mean duration for eating dinner when sampling CHAD data
- **eat_dinner_dt_mean_max** (*float*) – the maximum mean duration for eating dinner when sampling CHAD data

- **eat_dinner_dt_std_max** (*float*) – the maximum standard deviation for eating dinner when sampling CHAD data

set_dt_bounds (*start_min*, *start_max*, *end_min*, *end_max*)
This function calculates the bounds for duration time [expressed in hours}

Parameters

- **start_min** (*float*) – the minimum start time [hours]
- **start_max** (*float*) – the maximum start time [hours]
- **end_min** (*float*) – the minimum end time [hours]
- **end_max** (*float*) – the maximum end time [hours]

Returns the minimum duration, the maximum duration

Return type float, float

set_end_bounds (*start_min*, *start_max*, *dt_min*, *dt_max*)
This function calculates the bounds for end time [expressed in hours]

Parameters

- **start_min** (*float*) – the minimum start time [hours]
- **start_max** (*float*) – the maximum start time [hours]
- **dt_min** (*float*) – the minimum duration [hours]
- **dt_max** (*float*) – the maximum duration [hours]

Returns the minimum end time, the maximum end time

Return type float, float

2.3.4 chad_demography_adult_non_work module

This module contains code that handles accessing the Consolidated Human Activity Database (CHAD) data for the non-working adult demographic.

This module contains *chad_demography_adult_non_work.CHAD_demography_adult_non_work*.

class chad_demography_adult_non_work.**CHAD_demography_adult_non_work**
Bases: *chad_demography.CHAD_demography*

This class contains the common functionality with accessing the CHAD data files relevant to non-working adult demographic.

Variables

- **keys** – the ABMHAP activity codes for the activities simulated by the non-working adult demographic
- **fname_stats** (*dict*) – for a given ABMHAP activity code, access the file names for CHAD longitudinal data for the respective activity
- **eat_breakfast** (*chad_params.CHAD_params*) – sampling parameters for the eating breakfast activity within CHAD
- **eat_dinner** (*chad_params.CHAD_params*) – sampling parameters for the eating dinner activity within CHAD
- **eat_lunch** (*chad_params.CHAD_params*) – sampling parameters for the eating lunch activity within CHAD

- **'sleep'** (chad_params.CHAD_params) – CHAD sampling parameters for the sleep activity within CHAD
- **int_2_param** (*dict*) – for a given activity code, choose the proper sampling parameters for the respective activity

2.3.5 chad_demography_adult_work module

This module contains code that handles accessing the Consolidated Human Activity Database (CHAD) data for the working adult demographic.

This module contains *chad_demography_adult_work.CHAD_demography_adult_work*.

class chad_demography_adult_work.**CHAD_demography_adult_work**

Bases: *chad_demography.CHAD_demography*

This class contains the common functionality with accessing the CHAD data files relevant to working adult demographic.

Variables

- **keys** – the ABMHAP activity codes for the activities simulated by the working adult demographic
- **fname_stats** (*dict*) – for a given ABMHAP activity code, access the file names for CHAD longitudinal data for the respective activity
- **commute_to_work** (chad_params.CHAD_params) – sampling parameters for the commuting to work activity within CHAD
- **commute_from_work** (chad_params.CHAD_params) – sampling parameters for commuting from work activity within CHAD
- **'work'** (chad_params.CHAD_params) – sampling parameters for working activity within CHAD
- **eat_breakfast** (chad_params.CHAD_params) – sampling parameters for the eating breakfast activity within CHAD
- **eat_dinner** (chad_params.CHAD_params) – sampling parameters for the eating dinner activity within CHAD
- **eat_lunch** (chad_params.CHAD_params) – sampling parameters for the eating lunch activity within CHAD
- **'sleep'** (chad_params.CHAD_params) – CHAD sampling parameters for the sleep activity within CHAD
- **int_2_param** (*dict*) – for a given activity code, choose the proper sampling parameters for the respective activity

2.3.6 chad_demography_child_school module

This module contains code that handles accessing the Consolidated Human Activity Database (CHAD) data for the school-age children demographic.

This module contains *chad_demography_child_school.CHAD_demography_child_school*.

class chad_demography_child_school.**CHAD_demography_child_school**

Bases: *chad_demography.CHAD_demography*

This class contains the common functionality with accessing the CHAD data files relevant to school-age children demographic.

Variables

- **keys** – the ABMHAP activity codes for the activities simulated by the school-age children demographic
- **fname_stats** (*dict*) – for a given ABMHAP activity code, access the file names for CHAD longitudinal data for the respective activity
- **commute_to_work** (chad_params.CHAD_params) – sampling parameters for the commuting to work activity within CHAD
- **commute_from_work** (chad_params.CHAD_params) – sampling parameters for commuting from work activity within CHAD
- **'work'** (chad_params.CHAD_params) – sampling parameters for schooling activity within CHAD
- **eat_breakfast** (chad_params.CHAD_params) – sampling parameters for the eating breakfast activity within CHAD
- **eat_dinner** (chad_params.CHAD_params) – sampling parameters for the eating dinner activity within CHAD
- **eat_lunch** (chad_params.CHAD_params) – sampling parameters for the eating lunch activity within CHAD
- **'sleep'** (chad_params.CHAD_params) – CHAD sampling parameters for the sleep activity within CHAD
- **int_2_param** (*dict*) – for a given activity code, choose the proper sampling parameters for the respective activity

2.3.7 chad_demography_child_young module

This module contains code that handles accessing the Consolidated Human Activity Database (CHAD) data for the pre-school children demographic.

This module contains *chad_demography_child_young.CHAD_demography_child_young*.

class chad_demography_child_young.**CHAD_demography_child_young**

Bases: *chad_demography.CHAD_demography*

This class contains the common functionality with accessing the CHAD data files relevant to preschool chidlren demographic.

Variables

- **keys** – the ABMHAP activity codes for the activities simulated by the preschool children demographic
- **fname_stats** (*dict*) – for a given ABMHAP activity code, access the file names for CHAD longitudinal data for the respective activity
- **eat_breakfast** (chad_params.CHAD_params) – sampling parameters for the eating breakfast activity within CHAD
- **eat_dinner** (chad_params.CHAD_params) – sampling parameters for the eating dinner activity within CHAD

- **eat_lunch** (chad_params.CHAD_params) – sampling parameters for the eating lunch activity within CHAD
- **'sleep'** (chad_params.CHAD_params) – CHAD sampling parameters for the sleep activity within CHAD
- **int_2_param** (*dict*) – for a given activity code, choose the proper sampling parameters for the respective activity

2.3.8 chad_parameter_figures notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 20, 2018
```

WARNING:

this code may not be useful

This code plots the histograms of the distributions being sampled from the CHAD data for each activity.

Import

```
import sys
sys.path.append('..\\source')
sys.path.append('..\\processing')

# plotting capability
import matplotlib.pylab as plt

# zipfile capability
import zipfile

# ABMHAP modules

# general capability
import my_globals as mg
import chad_params as cp
import demography as dmg

import activity, analysis, chad, omni_trial, params
```

Run

```
# the demographic
demo = dmg.ADULT_WORK

# sets of activities
keys_all = mg.KEYS_ACTIVITIES
```

(continues on next page)

(continued from previous page)

```
# the activity codes related to not eating
keys_not_eat = [mg.KEY_SLEEP, mg.KEY_WORK, mg.KEY_COMMUTE_TO_WORK, mg.KEY_COMMUTE_
↪FROM_WORK]

# the activity codes of the eating activities
keys_eat = [mg.KEY_EAT_BREAKFAST, mg.KEY_EAT_LUNCH, mg.KEY_EAT_DINNER]

# the chosen group of activities
keys = keys_all
```

Loop through each activity and plot the histograms of start time, end time, and duration. Note: the limitations for each activity depends on which activity parameters are being sampled

```
# loop through each activity and plot the histograms of start time, end time, and␣
↪duration
# Note: the limitations for each activity depends on which activity parameters are␣
↪being sampled
for k in keys:

    # the CHAD limiting parameters
    s_params = cp.OMNI[k]

    # get the data
    stats_start, stats_end, stats_dt, record = analysis.get_verification_
↪info(demo=demo, key_activity=k,
                                                                  sampling_params=[s_params])
    # number of the bins
    num_bins = 24

    # create subplots
    fig, axes = plt.subplots(2, 2)

    # title
    fig.suptitle( activity.INT_2_STR[k] )

    #
    # plot the mean start time distribution
    #
    ax = axes[0, 0]
    if k == mg.KEY_SLEEP:
        ax.hist(mg.to_periodic(stats_start.mu.values, do_hours=True), bins=num_bins,␣
↪color='blue', label='start')
    else:
        ax.hist(stats_start.mu.values, bins=num_bins, color='blue', label='start')
    ax.set_xlabel('hours')
    ax.legend(loc='best')

    #
    # plot the mean end time distribution
    #
    ax = axes[0, 1]
    ax.hist(stats_end.mu.values, bins=num_bins, color='green', label='end')
    ax.set_xlabel('hours')
    ax.legend(loc='best')
```

(continues on next page)

(continued from previous page)

```
    #
    # plot the mean duration distribution
    #
    ax = axes[1, 0]
    ax.hist(stats_dt.mu.values, bins=num_bins, color='red', label='duration')
    ax.set_xlabel('hours')
    ax.legend(loc='best')

# show plots
plt.show()
```

2.3.9 chad_params module

The purpose of this module is to assign parameters necessary to run the ABMHAP initialized with data from the Consolidated Human Activity Database (CHAD).

This module contains *chad_params.CHAD_params*.

class chad_params.**CHAD_params**(*dt_mean_min=None*, *dt_mean_max=None*, *dt_std_max=None*, *start_mean_min=None*, *start_mean_max=None*, *start_std_max=None*, *end_mean_min=None*, *end_mean_max=None*, *end_std_max=None*, *N=1*, *do_solo=False*, *do_dt=False*, *do_start=False*, *do_end=False*)

Bases: `object`

This class holds sampling parameters for various activities in CHAD that are used to filter out what is considered "good" data for a given activity.

Parameters

- **dt_mean_min** (*float*) – the minimum mean duration to be sampled in hours [0, 24)
- **dt_mean_max** (*float*) – the maximum mean duration to be sampled in hours [0, 24)
- **dt_std_max** (*float*) – the maximum standard deviation of duration to be sampled in hours [0, 24)
- **start_mean_min** (*float*) – the minimum mean start time to be sampled in hours [0, 24)
- **start_mean_max** (*float*) – the maximum mean start time to be sampled in hours [0, 24)
- **start_std_max** (*float*) – the maximum standard deviation of start time to be sampled in hours [0, 24)
- **end_mean_min** (*float*) – the minimum mean end time to be sampled in hours [0, 24)
- **end_mean_max** (*float*) – the maximum mean end time to be sampled in hours [0, 24)
- **end_std_max** (*float*) – the maximum standard deviation of end time to be sampled in hours [0, 24)
- **N** (*int*) – the minimum amount of activity-events needed in sampling
- **do_solo** (*bool*) – a flag indicating whether to take single activity-events only
- **do_dt** (*bool*) – a flag indicating whether (if True) or not (if False) to sample duration data from CHAD

- **do_start** (*bool*) – a flag indicating whether (if True) or not (if False) to sample start time data from CHAD
- **do_end** (*bool*) – a flag indicating whether (if True) or not (if False) to sample end time data from CHAD

Variables

- **dt_mean_min** (*float*) – the minimum mean duration to be sampled in hours [0, 24)
- **dt_mean_max** (*float*) – the maximum mean duration to be sampled in hours [0, 24)
- **dt_std_max** (*float*) – the maximum standard deviation of duration to be sampled in hours [0, 24)
- **start_mean_min** (*float*) – the minimum mean start time to be sampled in hours [0, 24)
- **start_mean_max** (*float*) – the maximum mean start time to be sampled in hours [0, 24)
- **start_std_max** (*float*) – the maximum standard deviation of start time to be sampled in hours [0, 24)
- **end_mean_min** (*float*) – the minimum mean end time to be sampled in hours [0, 24)
- **end_mean_max** (*float*) – the maximum mean end time to be sampled in hours [0, 24)
- **end_std_max** (*float*) – the maximum standard deviation of end time to be sampled in hours [0, 24)
- **N** (*int*) – the minimum amount of activity-events needed in sampling
- **do_solo** (*bool*) – a flag indicating whether to take single activity-events only
- **do_dt** (*bool*) – a flag indicating whether (if True) or not (if False) to sample duration data from CHAD
- **do_start** (*bool*) – a flag indicating whether (if True) or not (if False) to sample start time data from CHAD
- **do_end** (*bool*) – a flag indicating whether (if True) or not (if False) to sample end time data from CHAD

get_dt(*df_stats*)

This function samples CHAD data for duration.

Parameters df_stats (*pandas.core.frame.DataFrame*) – the duration data from CHAD for a given activity

Returns the duration data from CHAD that satisfies statistical properties to use in ABMHAP.

Return type pandas.core.frame.DataFrame

get_end(*df_stats*)

This function samples CHAD data for end time.

Parameters df_stats (*pandas.core.frame.DataFrame*) – the end time data from CHAD for a given activity

Returns the end time data from CHAD that satisfies statistical properties to use in ABMHAP.

Return type pandas.core.frame.DataFrame

get_record(*df*, *do_periodic*)

Given a data frame of CHAD records, return the results where conditions are met according to the chad_param object.

Parameters

- **df** (*pandas.core.frame.DataFrame*) – the CHAD records from participants for a given activity
- **do_periodic** (*bool*) – a flag indicating whether (if True) or not (if False) to convert time to a [-12, 12) format due to an activity that could occur over midnight.

Returns the records from CHAD that satisfy the statistical data for duration, start time, and end time.

Return type pandas.core.frame.DataFrame

get_record_help(*x*, *lower*, *upper*, *do_periodic*)

This function finds the boolean indices of acceptable entries from an activity-parameter within the CHAD data.

Parameters

- **x** (*numpy.ndarray*) – data for a given activity-parameter (i.e., duration, start time, or end time)
- **lower** (*float*) – the lower bound of acceptable values
- **upper** (*float*) – the upper bound of acceptable values
- **do_periodic** (*bool*) – a flag indicating whether (if True) or not (if False) to convert time to a [-12, 12) format due to an activity that could occur over midnight.

Returns boolean indices of acceptable values, respectively

Return type numpy.ndarray of int

get_start(*df_stats*)

" This function samples CHAD data for start time.

Parameters df_stats (*pandas.core.frame.DataFrame*) – the start time data from CHAD for a given activity

Returns the start time data from CHAD that satisfies statistical properties to use in ABMHAP.

Return type pandas.core.frame.DataFrame

get_stats(*df*, *mean_min*, *mean_max*, *std_max*, *N*)

This function samples the CHAD longitudinal data and selects entries with the selected characteristics: the mean within the given range, within the maximum standard deviation, and having longitudinal data with at least N entries.

Parameters

- **df** (*pandas.core.frame.DataFrame*) – the duration statistical data for a given activity
- **mean_min** (*float*) –
- **mean_max** (*float*) –
- **std_max** (*float*) –
- **N** (*int*) –

Returns the CHAD data that satisfies the given statistical constraints

Return type pandas.core.frame.DataFrame

toString()

Represent the object as a string.

Returns the representation of the object as a string

Return type str

2.3.10 commute_from_work_trial module

This module contains code in order to run Monte-Carlo simulations to comparing the Agent-Based Model of Human Activity Patterns (ABMHAP) with the data from the Consolidated Human Activity Database (CHAD) for the **commute from work** activity.

This module contains class *commute_from_work_trial.Commute_From_Work_Trial*.

class commute_from_work_trial.**Commute_From_Work_Trial**(*parameters*, *sampling_params*, *demographic*)

Bases: *trial.Trial*

This class sets up runs for the ABMHAP initialized with data from CHAD to focus on the "commute from work" activity.

Parameters

- **parameters** (params.Params) – the parameters that describe the household
- **sampling_params** (chad_params.CHAD_params) – he sampling parameters used to filter "good" CHAD commute from work data
- **demographic** (*int*) – the demographic identifier

adjust_params(*commute_dt_mean*, *commute_dt_std*, *work_start_mean*, *work_start_std*, *work_end_mean*, *work_end_std*)

This function adjusts the values for the mean and standard deviation of both commute from work duration and start time in the key-word arguments based on the CHAD data that was sampled. These new values will be used in the runs.

Parameters

- **commute_dt_mean** (*numpy.ndarray*) – the commute duration mean [hours] for each person
- **commute_dt_std** (*numpy.ndarray*) – the commute duration standard deviation [hours] for each person
- **work_start_mean** (*numpy.ndarray*) – the mean work start time [hours] for each person
- **work_start_std** (*numpy.ndarray*) – the standard deviation of work start time [hours] for each person
- **work_end_mean** (*numpy.ndarray*) – the mean work end time [hours] for each person
- **work_end_std** (*numpy.ndarray*) – the standard deviation of work end time [hours] for each person

Returns

create_universe()

This function creates a universe object that simulations will run in. The only asset in this simulation for an agent to use is a *transport.Transport* and *workplace.Workplace*.

Returns the universe

Return type *universe.Universe*

initialize()

This function sets up the trial.

1. gets the CHAD data for commuting from work under the appropriate conditions
2. gets N samples the CHAD data for workng and commuting for the N trials
3. updates the *params* to reflect the newly assigned working and commuting parameters for the simulation

Returns

2.3.11 commute_to_work_trial module

This module contains code in order to run Monte-Carlo simulations to comparing the Agent-Based Model of Human Activity Patterns (ABMHAP) with the data from the Consolidated Human Activity Database (CHAD) for the **commute to work** activity.

This module contains class *commute_to_work_trial.Commute_To_Work_Trial*.

class commute_to_work_trial.**Commute_To_Work_Trial**(*parameters*, *sampling_params*, *demographic*)

Bases: *trial.Trial*

This class sets up runs for the ABMHAP initialized with data from CHAD to focus on the "commute to work" activity.

Parameters

- **parameters** (params.Params) – the parameters that describe the household
- **sampling_params** (chad_params.CHAD_params) – he sampling parameters used to filter "good" CHAD commute to work data
- **demographic** (*int*) – the demographic identifier

adjust_params(*commute_dt_mean*, *commute_dt_std*, *work_start_mean*, *work_start_std*, *work_end_mean*, *work_end_std*)

This function adjusts the values for the mean and standard deviation of both commute to work duration and start time in the key-word arguments based on the CHAD data that was sampled. These new values will be used in the runs.

Parameters

- **commute_dt_mean** (*numpy.ndarray*) – the commute duration mean [hours] for each person
- **commute_dt_std** (*numpy.ndarray*) – the commute duration standard deviation [hours] for each person
- **work_start_mean** (*numpy.ndarray*) – the mean work start time [hours] for each person
- **work_start_std** (*numpy.ndarray*) – the standard deviation of work start time [hours] for each person

- **work_end_mean** (*numpy.ndarray*) – the mean work end time [hours] for each person
- **work_end_std** (*numpy.ndarray*) – the standard deviation of work end time [hours] for each person

Returns

create_universe()

This function creates a universe object that simulations will run in. The only asset in this simulation for an agent to use is a `transport.Transport` and `workplace.Workplace`.

Returns the universe

Return type *universe.Universe*

initialize()

This function sets up the trial

1. gets the CHAD data for commuting to work under the appropriate conditions
2. gets N samples the CHAD data for workng and commuting for the N trials
3. updates the `params` to reflect the newly assigned working and commuting parameters for the simulation

Returns

2.3.12 data_counter notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 20, 2018
```

This file loads the activity-data assigned with each activity for the respective demographic group. For each activity, then the file counts the amount of Consolidated Human Acitivyt Databse (CHAD) individuals from both the single day and the longitudinal entries.

Import

```
import os, sys
sys.path.append('..\\source')
sys.path.append('..\\processing')

# plotting capability
import matplotlib.pylab as plt

# data frame capability
import pandas as pd

```

(continues on next page)

(continued from previous page)

```
# zipfile capability
import zipfile

# ABMHAP capability
import my_globals as mg
import chad_demography_adult_non_work as cdanw
import chad_demography_adult_work as cdaw
import chad_demography_child_school as cdcs
import chad_demography_child_young as cdcy
import demography as dmg

import activity, chad, datum
```

```
%matplotlib auto
```

```
Using matplotlib backend: Qt5Agg
```

Functions

```
def load_data(z, fnames):

    """
    This function loads the activity parameter data (start time, end time, \
    duration, and CHAD records) for an activity for the demographic.

    :param zipfile.Zipfile z: the ZipFile object for a given demographic group
    :param fnames: the file names for CHAD activity-moments data
    :type fnames: dict mapping int to str

    :return: the start time, end time, duration, and record data for a \
    given activity
    :rtype: numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray
    """

    start = pd.read_csv( z.open( fnames[chad.START], mode='r' ) )
    end = pd.read_csv( z.open( fnames[chad.END], mode='r' ) )
    dt = pd.read_csv( z.open( fnames[chad.DT], mode='r' ) )
    record = pd.read_csv( z.open( fnames[chad.RECORD], mode='r' ) )

    return start, end, dt, record

def filter_data(df, the_filter, start_periodic=False, end_periodic=False):

    """
    This function takes CHAD data for an activity and filters the CHAD data \
    the satisfy the sampling parameters. This function returns the CHAD data \
    suitable for use in parameterizing ABMHAP.

    :param pandas.core.frame.DataFrame df: the record data for a given activity
    :param the_filter: for a given activity code, get the respective parameters \
    for sampling CHAD data
    :type the_filter: dict mapping int to :class:`chad_params.CHAD_params`
    :param bool start_periodic: whether (if True) or not (if False) the start \
    time should be in a [-12, 12) format
    :param bool end_periodic: whether (if True) or not (if False) the end \
    time should be in a [-12, 12) format
```

(continues on next page)

(continued from previous page)

```
    :return: the CHAD data that satisfy the sampling parameters for the following:
    start time moments, end time moments, duration momments, and records
    :rtype: pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, \
    pandas.core.frame.DataFrame, pandas.core.frame.DataFrame
    """

    # the_filter are the sampling paramters for the activity

    # the start time and end time data
    x_start, x_end = df.start, df.end

    # change the start time data to a [-12, 12) format
    if start_periodic:
        x_start = mg.to_periodic(x_start, do_hours=True)

    # change the start time data to a [-12, 12) format
    if end_periodic:
        x_end = mg.to_periodic(x_end, do_hours=True)

    # the indices that satisfy the requirements for mean start time, end time, and
    # and duration respectively
    idx = ( x_start >= the_filter.start_mean_min ) & ( x_start <= the_filter.start_
↪mean_max ) \
    & ( df.end >= the_filter.end_mean_min ) & ( df.end <= the_filter.end_mean_max ) \
    & ( df.dt >= the_filter.dt_mean_min ) & ( df.dt <= the_filter.dt_mean_max )

    # get the record data that satisfy the proper sampling ranges
    record = df[idx]

    # the personal identifier values within the CHAD data
    pid = record.PID.values

    # obtain the duraation, start time, and end time values from the filtered CHAD␣
↪records
    dt, start, end = record.dt.values, record.start.values, record.end.values

    # the CHAD data that satisfy the sampling parameters for the start time moments
    stats_start = datum.get_stats(pid, start, do_periodic=start_periodic)

    # the CHAD data that satisfy the sampling parameters for the end time moments
    stats_end   = datum.get_stats(pid, end, do_periodic=start_periodic)

    # the CHAD data that satisfy the sampling parameters for the duration moments
    stats_dt    = datum.get_stats(pid, dt)

    return stats_start, stats_end, stats_dt, record

def get_activity_data(z, fnames, the_filter, start_periodic=False, end_
↪periodic=False):

    """
    This function loads CHAD data for an activity and filters the CHAD data \
    the satisfy the sampling parameters. This function returns the CHAD data \
    suitable for use in parameterizing ABMHAP.

    :param zipfile.Zipfile z: the ZipFile object for a given demographic group
```

(continues on next page)

(continued from previous page)

```
    :param fnames: the file names for CHAD activity-moments data
    :type fnames: dict mapping int to str
    :param the_filter: for a given activity code, get the respective parameters \
    for sampling CHAD data
    :type the_filter: dict mapping int to :class:`chad_params.CHAD_params`
    :param bool start_periodic: whether (if True) or not (if False) the start \
    time should be in a [-12, 12) format
    :param bool end_periodic: whether (if True) or not (if False) the end \
    time should be in a [-12, 12) format

    :return: the CHAD data that satisfy the sampling parameters for the following:
    start time moments, end time moments, duration momments, and records
    :rtype: pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, \
    pandas.core.frame.DataFrame, pandas.core.frame.DataFrame
    """

    # get the longitudinal data
    start, end, dt, record = load_data(z, fnames)

    # filter the records and get the moments
    stats_start, stats_end, stats_dt, record = \
    filter_data(record, the_filter, start_periodic=start_periodic, end_periodic=end_
↪periodic)

    return stats_start, stats_end, stats_dt, record

def get_fnames(demo, k, do_long):

    """
    For a demographic, this function obtains the file names of the \
    activity data for longitudinal or single-day data.

    :param demography.Demography demo: the demographic of choice to access the CHAD␣
↪data
    :param int k: the activity code
    :param bool do_long: whether (if True) to load the longitduinal data. If not␣
↪(False), \
    load the single-day data.

    :return: the file names for CHAD activity-moments data for longitudinal data \
    or single-day data
    :rtype: dict of int to str
    """

    # get the file names of the longitudinal data
    fnames = demo.fname_stats[k]

    if not do_long:
        # get the file names of the single-day data
        x = [ ( key, value.replace('longitude', 'solo') ) for key, value in fnames.
↪items() ]
        fnames = dict( x )

    return fnames

def plot(data, ax, label):
```

(continues on next page)

(continued from previous page)

```
    """
    This function gets data and plots the empiricial cumulative dsitribution \
    function (CDF) of the data.

    :param numpy.ndarray data: the data to create a CDF of
    :param matplotlib.axes._subplots.AxesSubplot ax: the subplot that's plotting
    :param str label: the label for the data
    """

    # get an empiricial CDF based on the data
    x, y = mg.get_ecdf(data)

    # plot the CDF
    ax.plot(x, y, label=label)

    # show legend
    ax.legend(loc='best')

    return
```

Run

Load data via demographic

```
# map a demographic type to the respective CHAD_demography object
chooser = {dmg.ADULT_WORK: cdaw.CHAD_demography_adult_work(),
           dmg.ADULT_NON_WORK: cdanw.CHAD_demography_adult_non_work(),
           dmg.CHILD_SCHOOL: cdcs.CHAD_demography_child_school(),
           dmg.CHILD_YOUNG: cdcy.CHAD_demography_child_young()}
```

```
# choose the demography
demo_type = dmg.CHILD_SCHOOL

# get the name of the compressed data file
fname_zip = dmg.FNAME_DEMOGRAPHY[demo_type]

# create the ZipFile object for the respective demographic group
z = zipfile.ZipFile( fname_zip )

# set the demographic object
demo = chooser[demo_type]

# store all of the activity-keys for the demographic
keys = demo.keys

# print flag
do_print = False
```

Count the number of CHAD persons for each activity

```
# if true, count the number of people with longitudinal data (at least 2 entries)
# if false, count the number of people with single data (only 1 entry)
do_long = True


# for each activity in the demographic, count the amount of data
for k in keys:
```

(continues on next page)

(continued from previous page)

```
    # set whether to set the time to periodic time [-12, 12) hours instead of [0, 24)␣
↪hours
    do_periodic = False
    if k == mg.KEY_SLEEP:
        do_periodic = True

    # sampling / filtering params
    the_filter = demo.int_2_param[k]

    # get the names of the statistics files
    fnames = get_fnames(demo, k, do_long)

    # load and filter data fitting for the demographic
    start, end, dt, record = get_activity_data(z, fnames, the_filter, start_
↪periodic=do_periodic)

    # print the activity
    if do_print:
        print( activity.INT_2_STR[k] )

        # count the number of longitudinal or single-day data, respectively
        if do_long:
            print( start[start.N > 1].shape)
        else:
            print( start[start.N == 1].shape)
```

```
..processingdatum.py:689: RuntimeWarning: invalid value encountered in double_
↪scalars
  cv   = std / np.abs(mu)
..processingdatum.py:689: RuntimeWarning: divide by zero encountered in␣
↪double_scalars
  cv   = std / np.abs(mu)
```

Plot the data

```
# create the subplots
fig, axes = plt.subplots(3)

# the title
fig.suptitle(activity.INT_2_STR[k])

#
# plot the start time data
#

# select the subplot
ax = axes[0]

# the start time data
plot(start.mu.values, ax, 'start')

#
# plot the end time data
#
```

(continues on next page)

(continued from previous page)

```
# select the subplot
ax = axes[1]

# the end time data
plot(end.mu.values, ax, 'end')

#
# plot the duration data
#

# select the subplot
ax = axes[2]

# the duration data
plot(dt.mu.values, ax, 'duration')

# show plots
plt.show()
```

2.3.13 driver module

This code runs the simulation for the Agent-Based Model of Human Activity Patterns (ABMHAP) module of the Life Cycle Human Exposure Model (LC-HEM) project. This code is the driver for seeing how well ABMHAP parameterized with empirical human behavior data from the Consolidated Human Activity Database (CHAD) compares to results seen in CHAD.

Note: This code may be run in batches in order to run many households while conserving memory. That is, instead of running 32 households at once (and keeping 32 households in memory), the program can run 2 batches of 16 households (for a total of 32 household). This halves the amount of memory used in the simulation compared to running the simulation of 1 batch of 32 households. We will shown how to run the code using "batches" below.

The driver can also be run in **parallel**. We will show how to do so below.

To run the code, do the following.

1. Set the simulation-centric parameters in driver_params.py
2. **Run the code as** > `python driver.py num_process num_hhld num_batch` where
    - `num_process` is the total number of cores (i.e, processing units) used in the simulation
    - `num_hhld` is the number of simulations to run per batch
    - `num_batch` is the number of batches used per core

The following are examples on how to run the code:

To run in **serial** with with 64 households per batch, 1 batch (implied)

> `python driver.py 1 64 1`

> `python driver.py 1 64`

To run in serial using 2 batches with 1 thread with 32 households per batch, 2 batches

> `python driver.py 1 32 2`

To run in **parallel** using 4 cores with 64 households total (16 household per core per batch), 1 batch (implied)

```
> python driver.py 4 64 1
```

```
> python driver.py 4 64
```

To run in parallel using 4 cores with 32 households per batch, 2 batches(8 households per core per batch)

```
> python driver.py 4 32 2
```

`driver.`**`create_trials`**(*num_hhld*, *num_days*, *num_hours*, *num_min*, *trial_code*, *chad_activity_params*, *demographic*, *num_people*, *do_minute_by_minute*, *do_print=False*)

This function creates the input data for each household in the simulation.

Parameters

- **`num_hhld`** (*`int`*) – the number of households simulated
- **`num_days`** (*`int`*) – the number of days in the simulation
- **`num_hours`** (*`int`*) – the number of additional hours
- **`num_min`** (*`int`*) – the number of additional minutes
- **`trial_code`** (*`int`*) – the trial identifier
- **`chad_activity_params`** (`chad_params.CHAD_params`) – the activity parameters used to sample "good" CHAD data
- **`demographic`** (*`int`*) – the demographic identifier
- **`num_people`** (*`int`*) – the number of people per household
- **`do_minute_by_minute`** (*`bool`*) – a flag for how the time steps progress in the scheduler
- **`do_print`** (*`bool`*) – flag whether to print messages to the console

Returns input data where each entry corresponds to the input for the respective household in the simulation

Return type list of *`trial.Trial`*

`driver.`**`delete_batch_files`**(*fname_base*, *num_batch*)

This function deletes the batch files.

Parameters

- **`fname_base`** (*`str`*) – the file name for the files without the ".pkl", that are the basis of the batch files that will be deleted
- **`num_batch`** (*`int`*) – the number of batches used in the code run

Returns

`driver.`**`get_batch_filenames`**(*fpath*, *fname*)

This file gets the file names for the batch saves.

Parameters

- **`fpath`** (*`str`*) – the name of the directory that the batch file names are stored
- **`fname`** (*`str`*) – the name of the file to save (.pkl)

Returns the batched file names

Return type list

`driver.`**`get_chad_demo`**(*demographic*)

Given the demographic, this function returns the respective CHAD_demography object.

Parameters **`demographic`** (*`int`*) – the demography identifier

Returns the respective CHAD_demography object

`driver.`**`get_cmd_line_params`**`()`

This function gets the parameters from the command line.

The order of arguments to be read on the command line in order:

1. the number of processors (threads)
2. the number of households per batch
3. the number of batches

Returns the number of processors, the, the total number of households to simulate, the number of batches

Return type int, int, int

`driver.`**`get_current_batch_size`**`(`*num_hhld*, *idx*, *max_batch_size*`)`

This function returns the number of households for the current batch if the total number of households is a multiple of the number of batches. Each batch contains max_batch_size amount of households. However, if not, the last batch will be smaller than the number of the max_batch_size.

Parameters

- **`num_hhld`** (*`int`*) – the total amount of households in the simulation
- **`idx`** (*`int`*) – the index of the current batch number
- **`max_batch_size`** (*`int`*) – the maximum number of households per batch

Returns the current batch size

Return type int

`driver.`**`get_fnames`**`(`*fpath*, *demographic*, *num_days*, *N*, *do_print=False*`)`

Given a directory, this function creates the file names that will be used to save the ABMHAP trials (input) and the ABMHAP data (output) according to the respective demographic.

Parameters

- **`fpath`** (*`str`*) – the directory in which to save the files
- **`demographic`** (*`int`*) – the demography identifier
- **`num_days`** (*`int`*) – the number of days in the simulation
- **`N`** (*`int`*) – the total number of households
- **`do_print`** (*`bool`*) – a flag to indicate whether (if True) or not (if False) to print a message to the screen

Returns the file name to save the trials data (".pkl" extension); the file name to save the data (".pkl" extension"), the file name to save the the basis (no ".pkl" extension) of the file name to save the trials data, the basis (no ".pkl" extension) of the file name to save the ABMHAP output data

Return type str, str, str, str

`driver.`**`get_loaded_trials_for_batch`**`(`*loaded_trials*, *i*, *batch_size*`)`

This function extracts the household input information from the pre-loaded input data for the respective batch.

Parameters

- **`loaded_trials`** (list of *`trial.Trial`*) – input data needed for the simulation
- **`i`** (*`int`*) – the current batch number

- **batch_size** (*int*) – the number of households in the batch

Returns the input data that corresponds to the batch number

Return type list of *trial.Trial*

driver.**get_max_batch_size**(*num_hhld*, *num_batch*)

This function returns the maximum number of households simulated per batch.

Parameters

- **num_hhld** (*int*) – the total number of households to simulate
- **num_batch** (*int*) – the number of batches

Returns the number of households to simulate per batch

Return type int

driver.**get_results**(*diaries*, *trials*)

This function takes the output and input from the simulation and converts the data into the appropriate output and input types.

Parameters

- **diaries** (list of *diary.Diary*) – each activity diary (output) in the Monte-Carlo simulation
- **trials** (list of *trial.Trial*) – the input data for each household simulation.

Returns the output, the input

Return type *driver_result.Driver_Result*, list of *params.Params*

driver.**initialize_trials**(*param_list*, *trial_code*, *chad_activity_params*, *demographic*)

This function initializes the trials (input parameters) for the simulation.

Parameters

- **param_list** – contains information on how to initialize the simulation for each household.
- **trial_code** (*int*) – the code of what trial to run
- **chad_activity_params** (chad_params.CHAD_params) – the activity parameters used to sample "good" CHAD data
- **demographic** (*int*) – this is the code for what demographic to run

Returns the initialized simulation scenarios

Return type list of *trial.Trial*

driver.**is_batch_file**(*fname*, *extensions*)

This function indicates whether or not the filename is a batch file. For example, given a file name called filename_b0000.pkl will return True. On the other hand, filename.pkl will return False.

Parameters

- **fname** (*str*) – the file name
- **extensions** (*str, list of str*) – the file extensions for the file name

Returns flag indicating whether the file name is a batch file

Return type bool

driver.**load_trials_for_batches**(*fname_load_trials_base*, *num_batch*, *do_print*)
This function loads pre-existing trials data.

Parameters

- **fname_load_trials** (*str*) – the filename (.pkl) of the trials data to load
- **num_batch** (*int*) – the number of batches
- **do_print** (*bool*) – a flag to indicate whether or not to print a message to screen about the logistics of loading the data

Returns the input data that has been loaded, the file name for the input data, the batch size

Return type list of *trial.Trial*, str, int

driver.**print_end**(*elapsed_time*)
Print the elapsed time for the simulation message.

Parameters **elapsed_time** (*float*) – the elapsed time for the simulation [seconds]

Returns

driver.**print_start**()
Print the message about starting the simulation.

Returns

driver.**print_starting_info**(*num_hhld*, *batch_size*, *num_batch*, *num_days*, *num_process*, *total_cpus*)
Print information before the beginning of the simulation.

Parameters

- **num_hhld** (*int*) – the total number of households
- **batch_size** (*int*) – the maximum number of households to simulate per batch
- **num_hhld_per_batch** (*int*) – the number of households per batch
- **num_days** – the number of days in the simulation
- **num_process** – the number of processors used
- **total_cpus** – the total amount of potential CPUs available.

Returns

driver.**run**(*num_process*, *trials*, *do_print=False*)
This function runs each simulation (in serial or parallel).

Parameters

- **num_process** (*int*) – the number of processors to use
- **trials** (list of *trial.Trial*) – the input for each simulation
- **do_print** (*bool*) – a flag indicating whether to print (if True) or not (if False)

Returns the results of the simulations, the input parameters

Return type diary_result.Diary_result, list of *params.Params*

driver.**run_batch**(*num_batch*, *num_hhld*, *num_process*, *num_days*, *num_hours*, *num_min*, *trial_code*, *chad_activity_params*, *demographic*, *num_people*, *do_minute_by_minute*, *do_print*, *do_save*, *fpath*, *do_load_trials=False*, *fname_load_trials_base=None*)
Run the simulation in batches.

Parameters

- **num_batch** (*int*) – the number of batches
- **num_hhld** (*int*) – the total number of households to simulate
- **num_process** (*int*) – the number of processors used
- **num_days** (*int*) – the number of days in the simulation
- **num_hours** (*int*) – the number of additional hours in the simulation
- **num_min** (*int*) – the number of additional minutes in the simulation
- **trial_code** (*int*) – the identifier for the trial being run
- **chad_activity_params** (chad_params.CHAD_params) – the activity parameters used to sample "good" CHAD data
- **demographic** (*int*) – the demographic identifier
- **do_print** (*bool*) – a flag indicating whether to print (if True) or not (if False)
- **do_save** (*bool*) – flag to save the output
- **fname_trials_base** (*str*) – the file name for the trials without the .pkl, which will be used for saving the trial information (.pkl)
- **fname_data_base** (*str*) – the file name for the ABMHAP without the .pkl, which will be used for saving the trial information (.pkl)
- **do_load_trials** (*bool*) – indicating whether (if True) or not (if False) to load trials from a saved file instead of creating a new set of trials
- **fname_load_trials_base** (*str*) – the file name for the ABMHAP trials without the .pkl, which will be used for saving the trial information (.pkl)

Returns the file name of the input data, the file name of the output data, the file name of the input data (no ".pkl"), the file name of the output data (no ".pkl")

Return type str, str, str, str

driver.**run_everything**(*num_process*, *num_hhld*, *num_batch*)

This code runs the Monte-Carlo simulations. More specifically, it

1. creates / loads the input data
2. runs the simulations
3. saves both the input and output data

Parameters

- **num_process** (*int*) – the number of processes
- **num_hhld** (*int*) – the number of households per core per batch
- **num_batch** (*int*) – the number of batches

Returns the file name for the input data, the file name for the output data

Return type str, str

driver.**run_parallel**(*num_process*, *trials*)

This function runs the simulation in parallel.

Parameters

- **num_process** (*int*) – the number of processors used

- **trials** (list of `trial.Trial`) – the input data

Returns the output of the simulations

Return type list of `diary.Diary`

`driver.`**`run_serial`**(*trials*, *do_print=False*)

This function runs the simulation in serial.

Parameters

- **trials** (list of `trial.Trial`) – the input data
- **do_print** (`bool`) – a flag whether or not to print the trial number

Returns the output of the simulations

Return type list of `diary.Diary`

`driver.`**`run_trials_parallel`**(*t*)

This function is called in order to run the trials in parallel.

Parameters **t** (`trial.Trial`) – the trial to run

Returns the results of the simulation

Return type *diary.Diary*

`driver.`**`save`**(*fname_data*, *fname_trials*, *fname_data_base*, *fname_trials_base*, *num_batch*, *do_print=False*)

This function saves the input and output from the simulation. It merges the data from the batch save files into one file for not only the trials data (input) but also the ABMHAP simulation data (output). Afterwards, the individual batch files are deleted.

Parameters

- **fname_data** (`str`) – the file name in which to save the ABMHAP data (output)
- **fname_trials** (`str`) – the file name in which to save the ABMHAP trials (input)
- **fname_data_base** (`str`) – the base (no ".pkl" extension) of the file name in which to save the ABMHAP data (output)
- **fname_trials_base** (`str`) – the base (no ".pkl" extension) of file name in which to save the ABMHAP trials (input)
- **do_print** (`bool`) – print flag

Returns

`driver.`**`save_batch_data_as_one_file`**(*fname*, *do_print=False*)

The function combines the ABMHAP data from the individual batch saves and saves them in one file.

Parameters **fname** (`str`) – the file name of the ABMHAP data (.pkl)

Returns

`driver.`**`save_batch_trials_as_one_file`**(*fname*, *do_print=False*)

This function combines the trial (input) data from the individual batch saves and saves them in one file.

Parameters

- **fname** (`str`) – the file name of the trials data (.pkl)
- **do_print** (`bool`) – print flag

Returns

`driver.save_diary_to_csv(fname)`

This function loads an activity diary from a compressed file format and saves it as a .csv file.

Parameters **fname** (*str*) – the pickle file that holds the activity diary file.

Returns

`driver.save_for_batch(result, fname, do_print=False)`

Save the data for the current batch.

Parameters

- **result** (`driver_result.Driver_Result`) – the result of the simulation for the current batch
- **fname** (*str*) – the file name to save the data for the current batch
- **do_print** (*bool*) – print flag

Returns

`driver.set_save_files_for_batch(fname_trials_base, fname_data_base, i, do_print=False)`

This function sets the save files (inputs and output files) for the current batch.

Parameters

- **fname_trials_base** (*str*) – the file name for the ABMHAP data without the .pkl, which will be used for saving the input data (.pkl)
- **fname_data_base** (*str*) – the file name for the ABMHAP data without the .pkl, which will be used for saving the output data (.pkl)
- **i** (*int*) – the current batch index
- **do_print** (*bool*) – flag indicating whether or not to print relevant information to the console

Returns the save file name for the input data, the save file name for the output data

Return type str, str

`driver.set_save_path(fpath, N, num_days)`

This function sets the save path for the data. Given a save path, the function appends it by adding an extension of the current year, month, day, number of households, and number of days in the format.

For example, if this code is being run to simulate 64 households for 100 days on July 4, 2017 and the file path is "output_path", the file path is set to the following: //output_path//2017_07_04//n0064_d100.

Parameters

- **fpath** (*str*) – the file path in which to save the data
- **N** (*int*) – the number of households
- **num_days** (*int*) – the number of days in the simulation

Returns the file directory in the format '//output_file_path//YYYY_MM_DD//nXXXX_dXXX'

Return type str

2.3.14 driver_params module

This module is responsible for containing parameters that driver.py uses to control the simulation. The user should set the parameters in this module **before** running the driver `driver.py`.

2.3.15 driver_result module

This module holds the results from running the Monte-Carlo simulations.

This module contains class `driver_result.Driver_Result` and `driver_result.Batch_Result`.

class `driver_result.`**Batch_Result**(*dr_list*)

Bases: `driver_result.Driver_Result`

This class holds the results from batch runs from the driver in one object.

Parameters **dr_list** (list of `driver_result.Driver_Result`) – the results from the simulation from each batch that was used.

class `driver_result.`**Driver_Result**(*diaries*, *chad_param_list*, *demographic*)

Bases: `object`

This class holds the result of running driver.run().

Parameters

- **diaries** (list of `diary.Diary`) – the activity diaries for each household in the simulation
- **chad_param_list** (list of `chad_params.CHAD_params`) – the CHAD parameters used for sampling the CHAD data
- **demographic** (*int*) – the demography identifier

Variables

- **diaries** – the activity diaries for each household in the simulation
- **chad_param_list** – the CHAD parameters used for sampling the CHAD data
- **demographic** (*int*) – the demography identifier
- **num_hhld** (*int*) – the number of households
- **num_people** (*int*) – the number of people in the simulation

add_id(*df_list*)

This function adds an integer identifier to each simulated agent's activity diary.

Parameters **df_list** (*list of pandas.core.frame.DataFrame*) – the activity diaries for the simulated agents

Returns the updated activity diaries for each agent

Return type list of pandas.core.frame.DataFrame

get_all_data()

This function returns the diaries as a pandas data frame.

Returns activity diaries for each person in the simulation

:rtype list of pandas.core.frame.DataFrame

get_combined_diary()

This function combines all of the activity diaries from the simulation into one.

Returns all of the activity diaries from the simulated agents combine into one* dataframe

Return type pandas.core.frame.DataFrame

2.3.16 eat_breakfast_trial module

This module contains code in order to run Monte-Carlo simulations to comparing the Agent-Based Model of Human Activity Patterns (ABMHAP) with the data from the Consolidated Human Activity Database (CHAD) for the **eat breakfast** activity.

This module contains class *eat_breakfast_trial.Eat_Breakfast_Trial*.

class eat_breakfast_trial.**Eat_Breakfast_Trial**(*parameters*, *sampling_params*, *demographic*)

Bases: *trial.Trial*

This class runs the ABMHAP simulations initialized with eat breakfast data from CHAD.

Parameters

- **paramters** (params.Params) – the parameters describing each person in the household
- **sampling_params** (chad_params.CHAD_params) – the sampling parameters used to filter "good" CHAD eat breakfast data
- **demographic** (*int*) – the demographic identifier

adjust_params(*start_mean*, *start_std*, *dt_mean*, *dt_std*)

This function adjusts the values for the mean and standard deviation of both eat breakfast duration and eat breakfast start time in the key-word arguments based on the CHAD data that was sampled. These new values will be used in the runs.

Parameters

- **start_mean** (*numpy.ndarray*) – the mean eat breakfast start time [hours] for each person
- **start_std** (*numpy.ndarray*) – the standard deviation of eat breakfast start time [hours] for each person
- **dt_mean** (*numpy.ndarray*) – the eat breakfast mean duration [hours] for each person
- **dt_std** (*numpy.ndarray*) – the eat breakfast standard deviation of duration [hours] for each person

Returns

create_universe()

This function creates a universe object that simulations will run in. The only asset in this simulation for an agent to use is a *food.Food*.

Returns the universe

Return type *universe.Universe*

initialize()

This function sets up the trial

1. gets the CHAD data for eat breakfast under the appropriate conditions for means and standard deviations for both eat breakfast duration and eat breakfast start time
2. gets N samples the CHAD data for eat breakfast duration and eat breakfast start time for the N trials
3. updates the *params* to reflect the newly assigned eat breakfast parameters for the simulation

Parameters

- **fname_dt** (*str*) – the filename of the duration statistics

- **fname_start** (*str*) – the filename of the start time statistics

Returns

initialize_person (*u*, *idx*)

This function creates and initializes a person with the proper parameters for the Eat Breakfast Trial simulation. This is important because it changes the meal structure towards having only 1 meal per day.

More specifically, the function does

1. creates a person
2. initializes the person's parameters to the respective values in *params*

Parameters

- **u** (universe.Universe) – the universe the person will reside in
- **idx** (*int*) – the index of the person's parameters in *params*

Return p the agent to simulate

Return type *person.Person*

2.3.17 eat_dinner_trial module

This module contains code in order to run Monte-Carlo simulations to comparing the Agent-Based Model of Human Activity Patterns (ABMHAP) with the data from the Consolidated Human Activity Database (CHAD) for the **eat dinner** activity.

This module contains class *eat_dinner_trial.Eat_Dinner_Trial*.

class eat_dinner_trial.**Eat_Dinner_Trial** (*parameters*, *sampling_params*, *demographic*)

Bases: *trial.Trial*

This class runs the ABMHAP simulations initialized with eat dinner data from CHAD.

Parameters

- **paramters** (params.Params) – the parameters describing each person in the household
- **sampling_params** (chad_params.CHAD_params) – the sampling parameters used to filter "good" CHAD eat dinner data
- **demographic** (*int*) – the demographic identifier

adjust_params (*start_mean*, *start_std*, *dt_mean*, *dt_std*)

This function adjusts the values for the mean and standard deviation of both eat dinner duration and eat dinner start time in the key-word arguments based on the CHAD data that was sampled. These new values will be used in the runs.

Parameters

- **start_mean** (*numpy.ndarray*) – the mean eat dinner start time [hours] for each person
- **start_std** (*numpy.ndarray*) – the standard deviation of eat dinner start time [hours] for each person
- **dt_mean** (*numpy.ndarray*) – the eat dinner mean duration [hours] for each person
- **dt_std** (*numpy.ndarray*) – the eat dinner standard deviation of duration [hours] for each person

Returns

create_universe()

This function creates a universe object that simulations will run in. The only asset in this simulation for an agent to use is a *food.Food*.

Returns the universe

Return type *universe.Universe*

initialize()

This function sets up the trial

1. gets the CHAD data for eat dinner under the appropriate conditions for means and standard deviations for both eat dinner duration and eat dinner start time
2. gets N samples the CHAD data for eat dinner duration and eat dinner start time for the N trials
3. updates the *params* to reflect the newly assigned eat dinner parameters for the simulation

Returns

initialize_person(*u*, *idx*)

This function creates and initializes a person with the proper parameters for the Eat Dinner Trial simulation. This is necessary because it changes the meal structure to having only one meal per day.

More specifically, the function does

1. creates a *singleton.Singleton* person
2. initializes the person's parameters to the respective values in *params*

Parameters

- **u** (universe.Universe) – the universe the person will reside in
- **idx** (*int*) – the index of the person's parameters in *params*

Return p the agent to simulate

Return type *person.Person*

2.3.18 eat_lunch_trial module

This module contains code in order to run Monte-Carlo simulations to comparing the Agent-Based Model of Human Activity Patterns (ABMHAP) with the data from the Consolidated Human Activity Database (CHAD) for the **eat lunch** activity.

This module contains class *eat_lunch_trial.Eat_Lunch_Trial*.

class eat_lunch_trial.**Eat_Lunch_Trial**(*parameters*, *sampling_params*, *demographic*)

Bases: *trial.Trial*

This class runs the ABMHAP simulations initialized with eat lunch data from CHAD.

Parameters

- **parameters** (params.Params) – the parameters describing each person in the household
- **sampling_params** (chad_params.CHAD_params) – the sampling parameters used to filter "good" CHAD eat lunch data
- **demographic** (*int*) – the demographic identifier

adjust_params (*start_mean*, *start_std*, *dt_mean*, *dt_std*)

This function adjusts the values for the mean and standard deviation of eat lunch start time in the key-word arguments based on the CHAD data that was sampled. These new values will be used in the runs.

Parameters

- **start_mean** (*numpy.ndarray*) – the mean eat lunch start time [hours] for each person
- **start_std** (*numpy.ndarray*) – the standard deviation of eat lunch start time [hours] for each person
- **dt_mean** (*numpy.ndarray*) – the eat lunch mean duration [hours] for each person
- **dt_std** (*numpy.ndarray*) – the eat lunch standard deviation of duration [hours] for each person

Returns

create_universe ()

This function creates a universe object that simulations will run in. The only asset in this simulation for an agent to use is a *food.Food*.

Returns the universe

Return type *universe.Universe*

initialize ()

This function sets up the trial

1. gets the CHAD data for eat lunch under the appropriate conditions for means and standard deviations for both eat lunch duration and eat lunch start time
2. gets N samples the CHAD data for eat lunch duration and eat lunch start time for the N trials
3. updates the *params* to reflect the newly assigned eat lunch parameters for the simulation

Parameters

- **fname_dt** (*str*) – the filename of the duration statistics
- **fname_start** (*str*) – the filename of the start time statistics

Returns

initialize_person (*u*, *idx*)

This function creates and initializes a person with the proper parameters for the Eat Lunch Trial simulation.

More specifically, the function does

1. creates a *singleton.Singleton* person
2. initializes the person's parameters to the respective values in *params*

Parameters

- **u** (universe.Universe) – the universe the person will reside in
- **idx** (*int*) – the index of the person's parameters in *params*

Return p the agent to be simulated

Return type *person.Person*

2.3.19 evaluation module

This module is used to for evaluating the accuracy of the Agent-Based Model of Human Activity Patterns (ABMHAP) simulation results vs. Consolidated Human Activity Database (CHAD) data.

`evaluation.`**`compare_abm_to_chad`**(*demo*, *df_list*, *trial_code*, *fidx=100*, *do_save=False*, *fpath=None*)

This function compares the results of the ABMHAP to the CHAD data by showing by

1. plotting cumulative distribution functions (CDF) of the predicted (ABMHAP) and observed (CHAD) single-day data for each activity
2. plotting the residual (that difference between the CDFs) between the predicted (ABMHAP) and observed (CHAD) data for each activity

Parameters

- **demo** (*int*) – the demographic identifier
- **df_list** (*list of pandas.core.frame.DataFrame*) – the ABMHAP activity diaries to compare
- **trial_code** (*int*) – the trial identifier
- **fidx** (*int*) – the figure identifier for the first figure in a series of figures
- **do_save** (*bool*) – a flag indicating whether (if True) or not (if False) to save the figures
- **fpath** (*str*) – the file path of the figures that are to be saved

Returns

`evaluation.`**`compare_abm_to_chad_help`**(*df_abm*, *df_obs*, *act_code*, *fidx*, *do_save*, *fpath*)

This function compares the results of the ABMHAP to the CHAD data for a given activity by

1. plotting cumulative distribution functions (CDF) of the predicted (ABMHAP) and observed (CHAD) single-day data for each activity
2. plotting the residual (that difference between the CDFs) between the predicted (ABMHAP) and observed (CHAD) data for each activity

Parameters

- **df_abm** (*pandas.core.frame.DataFrame*) – the predicted (ABMHAP) data for the respective activity
- **df_obs** (*pandas.core.frame.DataFrame*) – the single-day observed (CHAD) data for the respective activity
- **act_code** (*float*) – the activity code
- **fidx** (*int*) – the figure identifier of the first figure
- **do_save** (*bool*) – a flag indicating whether (if True) or not (if False) to save the figures
- **fpath** (*str*) – the file path of the figures that are to be saved

Returns the last figure identifier plotted

Return type int

`evaluation.`**`get_solo_data`**(*z*, *fname*)

This function gets the single-day data from individuals with only single-day records within CHAD.

Parameters

- **z** (*zipfile*) – the zipfile of the demographic data
- **fname** (*str*) – the file name for the CHAD individual records data

Returns the CHAD single-day data

Return type pandas.core.frame.DataFrame

evaluation.**plot** (*x*, *q*, *cdf*, *inv_cdf*, *act_code*, *fids*, *do_hours=True*, *dname=None*)

This function plots the following results of cumulative distribution function (CDF):

1. CDFs comparing the predicted and observed values
2. CDFs showing the residual
3. CDFs showing the scaled residual
4. Inverted CDFs comparing the predicted and observed values
5. Inverted CDFs showing the residual
6. Inverted CDFs showing the scaled residual

Parameters

- **x** (*numpy.ndarray*) – the range of values of the data
- **q** (*numpy.ndarray*) – the qunatiles
- **cdf** (*numpy.ndarray*) – the cumulative distribution function in units of percentage
- **inv_cdf** (*numpy.ndarray*) – the cumulative distribution function in units of time
- **act_code** (*numpy.ndarray*) – the activity codes of the respective activities
- **fids** (*numpy.ndarray*) – the figure identifiers
- **do_hours** (*bool*) – a flag indicating whether to plot the inverted CDF data in hours (if True) or minutes (if false)
- **dname** (*str*) – the name of the data to be plotted
- **off** (*float*) – the percentage in which to put a vertical line indicating both the bottom and top off-percentage of the data

Returns a figure containing CDFs comparing the predicted and observed values, a figure containing CDFs showing the residual, a figure containing CDFs showing the scaled residual, a figure containing Inverted CDFs comparing the predicted and observed values, a figure containing Inverted CDFs showing the residual, a figure containing Inverted CDFs showing the scaled residual

Return type matplotlib.figure.Figure, matplotlib.figure.Figure, matplotlib.figure.Figure matplotlib.figure.Figure, matplotlib.figure.Figure, matplotlib.figure.Figure

evaluation.**plot_predicted_observed** (*x*, *pred*, *obs*, *xlabel*, *ylabel*, *title*)

Plot the predicted (ABMHAP) and observed (CHAD) data.

Parameters

- **x** (*numpy.ndarray*) – the x-axis
- **pred** (*numpy.ndarray*) – the predicted (ABMHAP) values
- **obs** (*numpy.ndarray*) – the observed (CHAD) values from data
- **xlabel** (*str*) – the x-axis label
- **ylabel** (*str*) – the y-axis label

- **title** (*str*) – the title of the figure

Returns

`evaluation.`**plot_residual**(*x*, *res*, *xlabel=''*, *ylabel=''*, *title=''*, *color='r'*, *label='Residual'*)

This function plots the residual between cumulative distribution functions (CDFs) the ABMHAP and CHAD data.

Parameters

- **x** (*numpy.ndarray*) – the x-axis data
- **res** (*numpy.ndarray*) – the residual $r(x)$
- **xlabel** (*str*) – the x-axis label
- **ylabel** (*str*) – the y-axis label
- **title** (*str*) – the title of the plot
- **color** (*str*) – the color of the plot
- **label** (*str*) – the label of the plot

Returns

`evaluation.`**residual**(*pred*, *obs*, *x*)

This function analyzes the residual between predicted values and observed values. Given the predicted and observed values, this function does the following:

1. Compute the empirical cumulative distribution function (CDF) between the predicted and observed data in units [quantile vs hours]
2. Compute the residual in the CDF between observed and predicted data

$$r(x) = cdf_{observed}(x) - cdf_{predicted}(x)$$

3. Invert the residual so that the CDFs and residuals are in units [minutes vs quantile]

Parameters

- **pred** (*numpy.ndarray*) – the predicted (ABMHAP) values used to make the empirical CDF
- **obs** (*numpy.ndarray*) – the observed (CHAD) values used to make the empirical CDF
- **x** (*numpy.ndarray*) – the x-values
- **do_scaling** (*bool*) – this scales the inverted cdf residual by the standard deviation of the observed values

Returns the data for the cumulative distribution data (predicted, observed, residual, and scaled residual), the data for the inverted cumulative distribution data (predicted, observed, residual, and scaled residual)

Return type pandas.core.frame.DataFrame, pandas.core.frame.DataFrame

`evaluation.`**residual_analysis**(*pred*, *obs*, *N=1001*, *do_periodic=False*)

This function takes the predicted and observed values and computes the respective cumulative distribution functions (CDFs) in units percentage and the inverted CDF which is the CDF in units of minutes.

Parameters

- **pred** (*numpy.ndarray*) – the predicted values
- **obs** (*numpy.ndarray*) – the observed values
- **N** (*int*) – the number of points of the CDF vector
- **do_periodic** (*bool*) – a flag to see if the time data should be in a [-12, 12) hour format

Returns the x values, CDF of residual, inverted CDF of residual

Return type numpy.ndarray, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame

evaluation.**sample_activitiy_abm_work**(*df*)

This function is used in order to sample a random day of work activity data from the ABM. This function takes takes into account that 1 work "event" consists of multiple work activity-diary entries.

Note: This function assumes that df only contains work activity data and is **NOT** empty

Note: The duration data here is the end of the last event - minus the start of the first event. This is done to mimic how the duration data is stored in CHAD.

Parameters **df** (*pandas.core.frame.DataFrame*) – the diary of work activities for an individual

:return:the sampled work data :rtype: pandas.core.frame.DataFrame

evaluation.**sample_activity_abm**(*df_list*, *act*)

Given an activity type, this function looks at each activity diary and samples 1 event of that activity type should that diary have a matching activity-entry.

Note: Because the work activity technically occurs twice (1 event before lunch and 1 event after lunch), the activity needs to be merged as one event in order for the analysis to be correct.

Parameters

- **df_list** (*list of pandas.core.frame.DataFrame*) – the activity diaries
- **act** (*float*) – the activity code

Returns the sampled activities

Return type pandas.core.frame.DataFrame

evaluation.**save_figs_dt**(*figs*, *fpath*)

This function save plots about the activity duration.

Parameters

- **figs** (*tuple*) – a tuple of figures to save about activity duration data
- **fpath** (*str*) – the specific file path in which to plot the data

Returns

evaluation.**save_figs_end**(*figs*, *fpath*)

This function save plots about the activity end time.

Parameters

- **figs** (*tuple*) – a tuple of figures to save about activity end time data
- **fpath** (*str*) – the specific file path in which to plot the data

Returns

`evaluation.save_figs_start(figs, fpath)`

This function save plots about the activity start time.

Parameters

- **figs** (*tuple*) – a tuple of figures to save about activity start time data
- **fpath** (*str*) – the specific file path in which to plot the data

Returns

`evaluation.save_figures(act, figs_start, figs_end, figs_dt, fpath)`

This function saves the plotted figures about duration and start time data of the results from `compare_abm_to_chad()`.

Parameters

- **act** (*int*) – the activity code
- **figs_start** (*tuple*) – a tuple of figures to save about activity start time data about the random day sampling
- **figs_end** (*tuple*) – a tuple of figures to save about activity end time data about the random day sampling
- **figs_dt** (*tuple*) – a tuple of figures to save about activity duration data about the random day sampling
- **fpath** (*str*) – the general file path to plot the data

Returns

2.3.20 fig_driver module

Warning: This module is antiquated and not used.

This function is used to get the saved pickled plot data and plot them in subplots.

It is used to obtain cumulative distribution functions (CDFs) about the Agent-Based Model of Human Activity Patterns (ABMHAP) ABMHAP vs CHAD data for various activities and plot subplots with data from various activities instead of just 1

`fig_driver.compare_single_omni(fdirs_single, fdirs_omni, fid, nrows, ncols, activity_codes, do_chad=False)`

Plot a subplot of CDFs of single-activity and full simulation data for start time and duration.

Parameters

- **fdirs_single** (*list*) – the filenames of the pickled single-activity data
- **fdirs_omni** (*list*) – the filenames of the pickled for full-simulation data
- **fid** (*int*) – figure identifier
- **nrows** (*int*) – the number of rows in the suubplot
- **ncols** (*int*) – the number of columns in the subplot

- **activity_codes** (*list*) – the activity codes to plot
- **do_chad** (*bool*) – flag indicating whether or not to plot the CHAD data

Returns

fig_driver.**plot_cdfs** (*fdir*, *fid*)

fig_driver.**plot_cdfs2** (*fdirs*, *fid*, *nrows*, *ncols*, *activity_codes*)

2.3.21 figure_loader notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 20, 2018
```

This notebook loads the individual data about the cumuluative distribution functions (CDFs) comaparing the Agent-Based Model of Human Activity Patterns (ABMHAP) results to the Consolidated Human Activity Database (CHAD) data. The plots compare the distribution activity-parameter data from ABMHAP to CHAD. More specifically, the ABMAHP simulation data parameterized with CHAD longitduinal data are comared to the single-day data from CHAD. The following is plotted: 1. CDFs of ABMHAP vs. CHAD longitudianl data for activity-parameters 2. CDFs of ABMHAP vs CHAD single-day data for activity-parameters 3. Inverse CDFs of ABMHAP vs CHAD single-day data for ctivity-parameters 4. Residual of the Inverse CDF of ABMHAP vs CHAD single-day data for activity-parameters 5. Scaled Residual of the Quantile Functions of ABMHAP vs CHAD single-day data for activity-parameters

Import

```
import sys
sys.path.append('..\\source')
sys.path.append('..\\processing')
sys.path.append('..\\plotting')

# plotting capabilities
import matplotlib.pylab as plt
import matplotlib.ticker as ticker

# math capability
import numpy as np

# data frame capability
import pandas as pd

# python pickle capability
import pickle

# ABMHAP capability
import my_globals as mg
```

(continues on next page)

(continued from previous page)

```
import chad_demography_adult_work as cdaw
import chad_demography_adult_non_work as cdanw
import chad_demography_child_school as cdcs
import chad_demography_child_young as cdcy
import demography as dmg

import activity, analyzer, plotter, temporal
```

```
%matplotlib auto
```

```
Using matplotlib backend: Qt5Agg
```

define functions

```
def plot_subplots(data_list, do_cdf, main_title, legend, xlabels, ylabels, xunits,␣
↪yunits, colors, \
                  do_save=False, fname=None, linewidth=1):

    # the dimensions of a maximized figure. Base x Height [pixels]
    b_pixels, h_pixels = 2400, 1255
    my_dpi = 800

    b_in = b_pixels/my_dpi
    h_in = h_pixels/my_dpi


    # set the figure size for saving to custom if savinig
    if do_save:
        figsize, dpi = (b_in, h_in), my_dpi
    else:
        figsize, dpi = None, None

    # data_list is
    nrows, ncols = 3, len(data_list[0])

    if do_cdf:
        f, axes = plt.subplots(nrows, ncols, sharey=True, figsize=figsize, dpi=dpi)
    else:
        f, axes = plt.subplots(nrows, ncols, sharex=True, figsize=figsize, dpi=dpi)


    #
    # plot
    #
    for i , ax in enumerate(f.axes):

        # indices
        irow = i // ncols
        jcol = i % ncols

        # plot data
        temp = data_list[irow][jcol]

        for t, color in zip(temp, colors):

            x_data, y_data = t
```

(continues on next page)

(continued from previous page)

```
        if do_cdf and irow == 2:
            idx = x_data >= 0
            ax.plot(x_data[idx], y_data[idx], color=color, linewidth=linewidth)
        else:
            ax.plot(x_data, y_data, color=color, linewidth=linewidth)

        #
        # set the tick labels
        #
        ticksize=14
        ax.tick_params(axis='both', labelsize=ticksize)

        if irow == 2:
            ax.xaxis.set_major_locator(ticker.MaxNLocator(nbins=5))

        if do_cdf and irow in [0, 1]:
            # limit the xaxis to integernumbers
            x_all = [x.get_xdata() for x in ax.lines]
            x_all = np.hstack(x_all).flatten()
            x_min, x_max = np.floor( np.min(x_all) ), np.ceil( np.max(x_all))
            dx = abs(x_min - x_max) + 1
            nbins = np.ceil(dx/2)
            ax.xaxis.set_major_locator(ticker.MaxNLocator(nbins))

            ax.set_xlim(x_min, x_max)

            # set the xticks
            # testing
            x_min = np.round(x_min).astype(int)
            x_max = np.round(x_max).astype(int)
            dx = (x_max - x_min) / (5 - 1)
            dx = np.floor(dx).astype(int)
            xticks = np.arange(x_min, x_max, dx)
            ax.set_xticks(xticks)


# main title
fontsize_title = 18
f.suptitle(main_title, fontsize=fontsize_title)

# legend
f.legend( f.axes[0].lines, legend, 'best')

#
# set the x-axis labels
#

fontsize_label = 18
for ax, xlabel in zip( axes[nrows-1,:], xlabels) :
    ax.set_xlabel(xlabel, fontsize=fontsize_label)

    if not do_cdf:
        x_min, x_max = 0, 1
        ax.set_xlim(x_min, x_max)
        xticks = np.linspace(x_min, x_max, 3)
        ax.set_xticks(xticks)
        ##ax.set_xticks(xticks, fontsize=20)
```

(continues on next page)

(continued from previous page)

```
            #ax.set_xticklabels(labels=[], fontsize=20)

    # set x titles
    for ax, key in zip(axes[0,:], keys):
        #ax.set_title( activity.INT_2_STR[key], fontsize=fontsize_title )
        ax.set_title( activity.INT_2_STR[key], fontsize=14 )

    #
    # set the y-axis labels
    #
    for ax, ylabel in zip(axes[:, ncols-1], ylabels):
        ax.yaxis.set_label_position('right')
        ax.set_ylabel(ylabel, fontsize=fontsize_label, rotation=270, labelpad=20)

    for i, ax in enumerate(axes[:,0]):
        ax.yaxis.set_label_position('left')
        ax.set_ylabel(yunits[i], fontsize=fontsize_label)

        if do_cdf:
            y_min, y_max = 0, 1
            ax.set_ylim(y_min, y_max)

    if do_save and (fname is not None):
        f.savefig(fname, dpi=my_dpi)

    return
```

set up the parameters

```
#
# choose the deomography
#
demo = dmg.ADULT_NON_WORK

chooser = {dmg.ADULT_WORK: cdaw.CHAD_demography_adult_work(),
           dmg.ADULT_NON_WORK: cdanw.CHAD_demography_adult_non_work(),
           dmg.CHILD_SCHOOL: cdcs.CHAD_demography_child_school(),
           dmg.CHILD_YOUNG: cdcy.CHAD_demography_child_young(),
           }

# the CHAD demogramphy object
chad_demo = chooser[demo]

# the CHAD sampling parameters
s_params = chad_demo.int_2_param
```

```
# save the figures
do_save_fig = False

# whether or not to show the plots
do_show = True

# the linewidth
linewidth = 0.5
```

```
# use a custom figure directory
fpath = mg.FDIR_SAVE_FIG + '\\01_16_2018_no_variation\\n8192_d007'

chooser_fin = {dmg.ADULT_WORK: fpath + '\\adult_work',
       dmg.ADULT_NON_WORK: fpath + '\\adult_non_work',
       dmg.CHILD_SCHOOL: fpath + '\\child_school',
       dmg.CHILD_YOUNG: fpath + '\\child_young',
      }

fpath_figure_save = chooser_fin[demo]

# print the save figure directory
print('the figure save path:\t%s' % fpath_figure_save)

# different sets of activitiy data to plot
keys_all = chad_demo.keys

# eating activities
keys_eat = [mg.KEY_EAT_BREAKFAST, mg.KEY_EAT_LUNCH, mg.KEY_EAT_DINNER]

# non-eating activities
keys_not_eat = [ k for k in keys_all if k not in keys_eat ]
```

```
the figure save path:       ..my_datafig01_16_2018_no_variationn8192_
↪d007adult_non_work
```

Plotting

```
DO_ALL = 1
DO_MEALS = 2
DO_NOT_MEALS = 3

# (the activites to plot, part of the file name that matches the keys)
chooser_keys = { DO_ALL: (keys_all, 'all'), \
                DO_MEALS: (keys_eat, 'meals'),\
                DO_NOT_MEALS: (keys_not_eat, 'not_meals'),
               }
```

```
#
# set the activities to plot
#
plot_keys = DO_ALL

keys, fname_keys = chooser_keys[plot_keys]
name_keys = [ activity.INT_2_STR[k] for k in keys]


# labels on the right hand side of the plot
ylabels = ['Start Time', 'End Time', 'Duration']
```

Plot CDFs vs Longitudinal data

plot verification

```
fpaths = analyzer.get_verify_fpath(fpath_figure_save, keys)
```

```
#
# plot the verification cdf
#

# load the data
fname = '\\cdf_' + fname_keys + '.png'
data_list_all, fname_subplot = plotter.get_figure_data(fpaths, fpath_figure_save,␣
↪fname)

#
# plotting parameters
#
do_cdf = True

colors = ['blue', 'red']
legend = ['Predicted', 'Means (CHAD)']

xunits = 'Hours'
yunits = ['Quantile'] * 3

main_title = 'CDFs of Activity-parameters'

xlabels = [xunits] * len(keys)

#
# plot
#

plot_subplots(data_list=data_list_all, do_cdf=do_cdf, main_title=main_title,␣
↪legend=legend, \
                  xlabels=xlabels, ylabels=ylabels, xunits=xunits, yunits=yunits,␣
↪colors=colors, \
                  do_save=do_save_fig, fname=fname_subplot, linewidth=linewidth)

if do_show:
    plt.show()
else:
    plt.close()
```

```
C:UsersnbrandonAppDataLocalContinuumAnaconda3libsite-packagesmatplotliblegend.
↪py:338: UserWarning: Automatic legend placement (loc="best") not␣
↪implemented for figure legend. Falling back on "upper right".
  warnings.warn('Automatic legend placement (loc="best") not '
```

Plot CDFs vs random days

```
# choose the activities to plot
# get the figure directories
fpaths = [ (fpath_figure_save + mg.KEY_2_FDIR_SAVE_FIG[k] + mg.FDIR_SAVE_FIG_RANDOM_
↪DAY) for k in keys]
```

plot the cdf

```
#
# plot the CDF
#
```

(continues on next page)

(continued from previous page)

```
fname = '\\cdf_' + fname_keys + '.png'
fnames_load = ('\\cdf_start.pkl', '\\cdf_end.pkl', '\\cdf_dt.pkl')

# load the data
data_list_all, fname_subplot = plotter.get_figure_data(fpaths, fpath_figure_save,␣
↪fname, fnames_load=fnames_load)

#
# plotting parameters
#
do_cdf = True

colors = ['blue', 'red']
legend = ['Predicted', 'Observed']

xunits = 'Hours'
yunits = ['Quantile'] * 3

main_title = 'CDFs of Activity-parameters'

xlabels = [xunits] * len(keys)

#
# plot
#

plot_subplots(data_list=data_list_all, do_cdf=do_cdf, main_title=main_title,␣
↪legend=legend, \
                  xlabels=xlabels, ylabels=ylabels, xunits=xunits, yunits=yunits,␣
↪colors=colors, \
                  do_save=do_save_fig, fname=fname_subplot, linewidth=linewidth)

if do_show:
    plt.show()
else:
    plt.close()
```

```
C:UsersnbrandonAppDataLocalContinuumAnaconda3libsite-packagesmatplotliblegend.
↪py:338: UserWarning: Automatic legend placement (loc="best") not␣
↪implemented for figure legend. Falling back on "upper right".
  warnings.warn('Automatic legend placement (loc="best") not '
```

Plot the Inverse CDF

```
#
# plot the Inverse CDF
#

fname = '\\cdf_inv_' + fname_keys + '.png'
fnames_load = ('\\cdf_inv_start.pkl', '\\cdf_inv_end.pkl', '\\cdf_inv_dt.pkl')

# load the data
data_list_all, fname_subplot = plotter.get_figure_data(fpaths, fpath_figure_save,␣
↪fname, fnames_load=fnames_load)

#
# plotting parameters
```

(continues on next page)

(continued from previous page)

```
#
do_cdf = True

colors = ['blue', 'red']
legend = ['Predicted', 'Observed']

xunits = 'Hours'
yunits = ['Quantile'] * 3

main_title = 'Inverse CDFs of Activity-parameters'

xlabels = [xunits] * len(keys)

#
# plot
#

plot_subplots(data_list=data_list_all, do_cdf=do_cdf, main_title=main_title,␣
↪legend=legend, \
                  xlabels=xlabels, ylabels=ylabels, xunits=xunits, yunits=yunits,␣
↪colors=colors, \
                  do_save=do_save_fig, fname=fname_subplot, linewidth=linewidth)

if do_show:
    plt.show()
else:
    plt.close()
```

```
C:UsersnbrandonAppDataLocalContinuumAnaconda3libsite-packagesipykernel_
↪launcher.py:73: RuntimeWarning: divide by zero encountered in long_scalars
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-39-2c25156c1693> in <module>()
     28 #
     29
---> 30 plot_subplots(data_list=data_list_all, do_cdf=do_cdf, main_title=main_title,␣
↪legend=legend,                  xlabels=xlabels, ylabels=ylabels, xunits=xunits,␣
↪yunits=yunits, colors=colors,                  do_save=do_save_fig, fname=fname_
↪subplot, linewidth=linewidth)
     31
     32 if do_show:


<ipython-input-3-8a15175d88ba> in plot_subplots(data_list, do_cdf, main_title, legend,
↪ xlabels, ylabels, xunits, yunits, colors, do_save, fname, linewidth)
     71                 dx = (x_max - x_min) / (5 - 1)
     72                 dx = np.floor(dx).astype(int)
---> 73                 xticks = np.arange(x_min, x_max, dx)
     74                 ax.set_xticks(xticks)
     75


ValueError: Maximum allowed size exceeded
```

plot residuals

```
#
# plot the residuals ICDF
#


# recall that the residuals should be multiplied by -1
fname = '\\res_inv_' + fname_keys + '.png'
fnames_load = ('\\res_inv_start.pkl', '\\res_inv_end.pkl', '\\res_inv_dt.pkl')

data_list_all, fname_subplot = plotter.get_figure_data(fpaths, fpath_figure_save,␣
↪fname, fnames_load=fnames_load)
#
# plotting parameters
#

# residual plot (inverse CDF)
do_cdf = False
legend = ['Residual']
colors = ['Red']

xunits = 'Quantile'
yunits = ['Hours', 'Hours', 'Minutes']

main_title = 'Residual of the Inverse CDF'

xlabels = [xunits] * len(keys)

#
# plot the data
#
plot_subplots(data_list=data_list_all, do_cdf=do_cdf, main_title=main_title,␣
↪legend=legend, \
                  xlabels=xlabels, ylabels=ylabels, xunits=xunits, yunits=yunits,␣
↪colors=colors, \
                  do_save=do_save_fig, fname=fname_subplot, linewidth=linewidth)

if do_show:
    plt.show()
else:
    plt.close()
```

```
C:UsersnbrandonAppDataLocalContinuumAnaconda3libsite-packagesmatplotliblegend.
↪py:338: UserWarning: Automatic legend placement (loc="best") not␣
↪implemented for figure legend. Falling back on "upper right".
  warnings.warn('Automatic legend placement (loc="best") not '
```

plot the scaled residuals

```
#
# plot the residuals ICDF scaled
#

# recall that the residuals should be multiplied by -1
fnames = '\\res_inv_scaled' + fname_keys + '.png'

fnames_load = ('\\res_inv_scaled_start.pkl', '\\res_inv_scaled_end.pkl', \
```

(continues on next page)

(continued from previous page)

```
                '\\res_inv_scaled_dt.pkl')

data_list_all, fname_subplot = plotter.get_figure_data(fpaths, fpath_figure_save,␣
↪fname, fnames_load=fnames_load)

#
# plotting parameters
#Q
do_cdf = False

legend = ['Residual']
colors = ['Red']
xunits = 'Quantitle'
yunits = ['Standard Deviations'] * 3

main_title = 'Scaled Residual of the Quantile Functions'

xlabels = [xunits] * len(keys)

#
# plot the data
#

plot_subplots(data_list=data_list_all, do_cdf=do_cdf, main_title=main_title,␣
↪legend=legend, \
                  xlabels=xlabels, ylabels=ylabels, xunits=xunits, yunits=yunits,␣
↪colors=colors, \
                  do_save=do_save_fig, fname=fname_subplot, linewidth=linewidth)

if do_show:
    plt.show()
else:
    plt.close()
```

```
C:UsersnbrandonAppDataLocalContinuumAnaconda3libsite-packagesmatplotliblegend.
↪py:338: UserWarning: Automatic legend placement (loc="best") not␣
↪implemented for figure legend. Falling back on "upper right".
  warnings.warn('Automatic legend placement (loc="best") not '
```

2.3.22 figure_loader_with_without_variation notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 20, 2018
```

This notebook loads the individual data about the cumuluative distribution functions (CDFs) comaparing the Agent-

Based Model of Human Activity Patterns (ABMHAP) results to the Consolidated Human Activity Database (CHAD) data. The plots compare the distribution activity-parameter data from ABMHAP to CHAD. More specifically, the we compare the ABMHAP with intra-individual variation, ABMHAP without intra-individual variation, and CHAD single-day data.

This module loads and plots a figure with the following:

1. CDFs of ABMHAP with intra-individual variation vs. ABMHAP without intra-individual variation vs. CHAD longitudinal data for activity-parameters

Import

```
import sys
sys.path.append('..\\source')
sys.path.append('..\\processing')
sys.path.append('..\\plotting')

# plotting capability
import matplotlib.pylab as plt
import matplotlib.ticker as ticker

# math capability
import numpy as np

# data frame capability
import pandas as pd

# pickling capability
import pickle

# ABMHAP modules
import my_globals as mg
import demography as dmg
import activity, analyzer, plotter, temporal

import chad_demography_adult_work as cdaw
import chad_demography_adult_non_work as cdanw
import chad_demography_child_school as cdcs
import chad_demography_child_young as cdcy
```

```
%matplotlib auto
```

```
Using matplotlib backend: Qt5Agg
```

define functions

```
# plot subplots

def plot_subplots(data_list1, data_list2, data_list3, do_cdf, main_title, legend,␣
↪xlabels, ylabels, \
                        xunits, yunits, colors, do_save=False, fname=None,␣
↪linewidth=1):

    # the dimensions of a maximized figure. Base x Height [pixels]
    b_pixels, h_pixels = 2400, 1255
    my_dpi = 800

    b_in = b_pixels/my_dpi
```

(continues on next page)

(continued from previous page)

```
    h_in = h_pixels/my_dpi


    # set the figure size for saving to custom if savinig
    if do_save:
        figsize, dpi = (b_in, h_in), my_dpi
    else:
        figsize, dpi = None, None

    # data_list is
    nrows, ncols = 3, len(data_list1[0])

    if do_cdf:
        f, axes = plt.subplots(nrows, ncols, sharey=True, figsize=figsize, dpi=dpi)
    else:
        f, axes = plt.subplots(nrows, ncols, sharex=True, figsize=figsize, dpi=dpi)


    #
    # plot
    #
    alpha = 0.7
    for i , ax in enumerate(f.axes):

        # indices
        irow = i // ncols
        jcol = i % ncols

        # plot data
        temp1 = data_list1[irow][jcol]
        temp2 = data_list2[irow][jcol]
        temp3 = data_list3[irow][jcol]

        counter = 0

        # ii for testing if
        ii = 0

        for t1, t2, color in zip(temp1, temp2, colors):

            if ii == 0:
                x_data1, y_data1 = t1
                x_data2, y_data2 = t2

                if counter == 0:
                    c1 = 'blue'
                    c2 = 'k'
                    #c2 = 'green'
                else:
                    c1 = 'red'
                    c2 = 'red'

                if do_cdf and irow == 2:
                    idx = x_data1 >= 0

                    ax.plot(x_data1[idx], y_data1[idx], color=c1, linewidth=linewidth,
↪ alpha=alpha)
```

(continues on next page)

(continued from previous page)

```
                    ax.plot(x_data2[idx], y_data2[idx], color=c2, ls='--',␣
↪linewidth=linewidth, alpha=alpha)
                else:
                    ax.plot(x_data1, y_data1, color=c1, linewidth=linewidth,␣
↪alpha=alpha)
                    ax.plot(x_data2, y_data2, color=c2, ls='--', linewidth=linewidth,␣
↪alpha=alpha)

                # access the CHAD data
                x_data3, y_data3 = temp3[1]

                if (irow in [0, 1]) and jcol in [1, 4]:
                    x_data3 = mg.from_periodic(x_data3, do_hours=True)

                ax.plot(x_data3, y_data3, color='r', linewidth=linewidth, alpha=alpha)

                counter = counter + 1
                ii = ii + 1
            #
            # set the tick labels
            #
            ticksize=14
            ax.tick_params(axis='both', labelsize=ticksize)

            if irow == 2:
                ax.xaxis.set_major_locator(ticker.MaxNLocator(nbins=5))

            if do_cdf and irow in [0, 1]:
                # limit the xaxis to integernumbers
                x_all = [x.get_xdata() for x in ax.lines]
                x_all = np.hstack(x_all).flatten()
                x_min, x_max = np.floor( np.min(x_all) ), np.ceil( np.max(x_all))
                dx = abs(x_min - x_max) + 1
                nbins = np.ceil(dx/2)
                ax.xaxis.set_major_locator(ticker.MaxNLocator(nbins))

                ax.set_xlim(x_min, x_max)

                # set the xticks
                # testing
                x_min = np.round(x_min).astype(int)
                x_max = np.round(x_max).astype(int)
                dx = (x_max - x_min) / (5 - 1)
                dx = np.floor(dx).astype(int)
                xticks = np.arange(x_min, x_max, dx)
                ax.set_xticks(xticks)


    # main title
    fontsize_title = 18
    f.suptitle(main_title, fontsize=fontsize_title)

    # legend
    f.legend( f.axes[0].lines, legend, 'best')

    #
    # set the x-axis labels
```

(continues on next page)

(continued from previous page)

```
    #

    fontsize_label = 18
    for ax, xlabel in zip( axes[nrows-1,:], xlabels) :
        ax.set_xlabel(xlabel, fontsize=fontsize_label)

        if not do_cdf:
            x_min, x_max = 0, 1
            ax.set_xlim(x_min, x_max)
            xticks = np.linspace(x_min, x_max, 3)
            ax.set_xticks(xticks)
            ##ax.set_xticks(xticks, fontsize=20)
            #ax.set_xticklabels(labels=[], fontsize=20)

    # set x titles
    for ax, key in zip(axes[0,:], keys):
        #ax.set_title( activity.INT_2_STR[key], fontsize=fontsize_title )
        ax.set_title( activity.INT_2_STR[key], fontsize=14 )

    #
    # set the y-axis labels
    #
    for ax, ylabel in zip(axes[:, ncols-1], ylabels):
        ax.yaxis.set_label_position('right')
        ax.set_ylabel(ylabel, fontsize=fontsize_label, rotation=270, labelpad=20)

    for i, ax in enumerate(axes[:,0]):
        ax.yaxis.set_label_position('left')
        ax.set_ylabel(yunits[i], fontsize=fontsize_label)

        if do_cdf:
            y_min, y_max = 0, 1
            ax.set_ylim(y_min, y_max)

    if do_save and (fname is not None):
        f.savefig(fname, dpi=my_dpi)

    return
```

set up the parameters

```
#
# choose the deomography
#
demo = dmg.CHILD_YOUNG

chooser = {dmg.ADULT_WORK: cdaw.CHAD_demography_adult_work(),
           dmg.ADULT_NON_WORK: cdanw.CHAD_demography_adult_non_work(),
           dmg.CHILD_SCHOOL: cdcs.CHAD_demography_child_school(),
           dmg.CHILD_YOUNG: cdcy.CHAD_demography_child_young(),
           }

# the CHAD demogramphy object
chad_demo = chooser[demo]

# the CHAD sampling parameters
s_params = chad_demo.int_2_param
```

```
# save the figures
do_save_fig = False

# whether or not to show the plots
do_show = True

# the linewidth
linewidth = 1
```

```
#fpath1 = mg.FDIR_SAVE_FIG + '\\11_21_2017\\n8192_d364' # with variation
#fpath2 = mg.FDIR_SAVE_FIG + '\\01_11_2018\\n8192_d007_no_variation' # no variation

fpath1 = mg.FDIR_SAVE_FIG + '\\12_07_2017\\n8192_d364' # with variation
fpath2 = mg.FDIR_SAVE_FIG + '\\01_16_2018_no_variation\\n8192_d007' # no variation

#fpath_temp = mg.FDIR_SAVE_FIG + '\\with_without_variation'
#fpath1 = fpath_temp + '\\n8192_d007_with_variation'
#fpath2 = fpath_temp + '\\n8192_d364_no_variation'

fpath_figure_save1 = fpath1 + '\\child_young'
fpath_figure_save2 = fpath2 + '\\child_young'

# print the save figure directory
print('the figure save path 1:\t%s' % fpath_figure_save1)
print('the figure save path 2:\t%s' % fpath_figure_save2)

# different sets of activitiy data to plot
keys_all = chad_demo.keys

keys_eat = [mg.KEY_EAT_BREAKFAST, mg.KEY_EAT_LUNCH, mg.KEY_EAT_DINNER]

keys_not_eat = [ k for k in keys_all if k not in keys_eat ]
```

```
the figure save path 1:      ..my_datafig12_07_2017n8192_d364child_young
the figure save path 2:      ..my_datafig01_16_2018_no_variationn8192_
↪d007child_young
```

Plotting

```
DO_ALL = 1
DO_MEALS = 2
DO_NOT_MEALS = 3

# (the activites to plot, part of the file name that matches the keys)
chooser_keys = { DO_ALL: (keys_all, 'all'), \
                DO_MEALS: (keys_eat, 'meals'),\
                DO_NOT_MEALS: (keys_not_eat, 'not_meals'),
               }
```

```
#
# set the activities to plot
#
plot_keys = DO_ALL

keys, fname_keys = chooser_keys[plot_keys]
name_keys = [ activity.INT_2_STR[k] for k in keys]
```

(continues on next page)

(continued from previous page)

```
# labels on the right hand side of the plot
ylabels = ['Start Time', 'End Time', 'Duration']
```

Plot CDFs vs Longitudinal data

plot verification

```
# get the figure directory of ABMHAP runs with intra-individual variation
fpaths1 = analyzer.get_verify_fpath(fpath_figure_save1, keys)

# get the figure directory of ABMHAP runs with no intra-individual variation
fpaths2 = analyzer.get_verify_fpath(fpath_figure_save2, keys)
```

```
# load figure data with longitudinal data

# file names
fname = '\\cdf_' + fname_keys + '.png'

# load figure data
data_list_all1, fname_subplot1 = plotter.get_figure_data(fpaths1, fpath_figure_save1,
↪fname)
data_list_all2, fname_subplot2 = plotter.get_figure_data(fpaths2, fpath_figure_save2,
↪fname)
```

Get the data for a random single day

```
# load figure data of sinlge-day data

# file names
fname = '\\cdf_' + fname_keys + '.png'

fnames_load = ('\\cdf_start.pkl', '\\cdf_end.pkl', '\\cdf_dt.pkl')

# load figure data from ABMHAP figures with intra-individual variation
data_list_all_single_day1, fname_subplot1 = \
plotter.get_figure_data(fpaths1, fpath_figure_save1, fname, fnames_load=fnames_load,
↪do_single_day=True)

# load figure data from ABMHAP figures with no intra-individual variation
data_list_all_single_day2, fname_subplot2 = \
plotter.get_figure_data(fpaths2, fpath_figure_save2, fname, fnames_load=fnames_load,
↪do_single_day=True)
```

```
fpath_figure_save2
```

```
'..\my_data\fig\01_16_2018_no_variation\n8192_d007\child_young'
```

plot the cdf

```
#
# plot the verification cdf
#

#
# plotting parameters
```

(continues on next page)

(continued from previous page)

```
#
do_cdf = True

colors = ['blue', 'red']
legend = ['With Intra','No Intra', 'CHAD single day', 'CHAD means']

xunits = 'Hours'
yunits = ['Quantile'] * 3

main_title = 'CDFs of Activity-parameters'

xlabels = [xunits] * len(keys)

#
# plot
#

# set the data
data_list1 = data_list_all_single_day1 # with variaiton
data_list2 = data_list_all_single_day2 # no variation
data_list3 = data_list_all_single_day1 # acesses the CHAD random day data which is␣
↪encapsulated within
                                        # data_list[irow][icol][1]

# plot the data
plot_subplots(data_list1=data_list1, data_list2=data_list2, data_list3=data_list3, \
                   do_cdf=do_cdf, main_title=main_title, \
                   legend=legend, xlabels=xlabels, ylabels=ylabels, xunits=xunits,␣
↪yunits=yunits, colors=colors, \
                   do_save=do_save_fig, fname=fname_subplot1, linewidth=0.5)

if do_show:
    plt.show()
else:
    plt.close()
```

```
C:UsersnbrandonAppDataLocalContinuumAnaconda3libsite-packagesmatplotliblegend.
↪py:338: UserWarning: Automatic legend placement (loc="best") not␣
↪implemented for figure legend. Falling back on "upper right".
  warnings.warn('Automatic legend placement (loc="best") not '
```

2.3.23 figure_residuals notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 20, 2018
```

This file calculates the residuals in the cumaltive distribution functions (CDFs) for the activities in each demographic.

The file calculates the residuals = |cdf_predicted - cdf_observed| as a function of percentile from 0 to 1. Then the mean value for the residual plot is calculated which represents the expected deviation from the data for each percentile

Import

```
import sys
sys.path.append('..\\source')
sys.path.append('..\\processing')
sys.path.append('..\\plotting')

# plotting capability analysis
import matplotlib.pylab as plt

# math capability
import numpy as np

# python data compression
import pickle

# ABMHAP modules
import my_globals as mg
import chad_demography_adult_work as cdaw
import chad_demography_adult_non_work as cdanw
import chad_demography_child_school as cdcs
import chad_demography_child_young as cdcy
import demography as dmg

import activity, plotter
```

```
%matplotlib auto
```

```
Using matplotlib backend: Qt5Agg
```

define functions

```
def f(data, alpha=0):

    # create the residuals between the prediction (ABMHAP) and observation (CHAD)
    # data. Plot the quantiles of the data [alpha, 1 - alpha] percentiles of the data.

    # predicted data and observed data
    pred, obs = data

    # the x and y values for the predicted data and observed data
    x_pred, y_pred = pred
    x_obs, y_obs = obs

    # residual
    r = np.abs(y_pred - y_obs)

    # the number of data points
    m = len(r)

    # the bottom and top percentile
    bot, top = alpha/2, 1 - alpha/2
```

(continues on next page)

(continued from previous page)

```
    # get the percentiles within range
    x = x_pred
    idx = (x >= bot) & (x <= top)

    return x[idx], r[idx]

# get the moments
def get_moments(x):

    # the mean data
    mu = x.mean()

    # the standard deviation data
    std = x.std()

    return mu, std
```

set up the parameters

```
#
# choose the deomography
#
demo = dmg.CHILD_YOUNG

chooser = {dmg.ADULT_WORK: cdaw.CHAD_demography_adult_work(),
           dmg.ADULT_NON_WORK: cdanw.CHAD_demography_adult_non_work(),
           dmg.CHILD_SCHOOL: cdcs.CHAD_demography_child_school(),
           dmg.CHILD_YOUNG: cdcy.CHAD_demography_child_young(),
           }

# the CHAD demogramphy object
chad_demo = chooser[demo]

# the CHAD sampling parameters
s_params = chad_demo.int_2_param
```

```
# save the figures
do_save_fig = False

# whether or not to show the plots
do_show = True

# the linewidth
linewidth = 1.5
```

```
# choose the appropriate figure directory
fpath = mg.FDIR_SAVE_FIG + '\\12_07_2017\\n8192_d364'

chooser_fin = {dmg.ADULT_WORK: fpath + '\\adult_work',
       dmg.ADULT_NON_WORK: fpath + '\\adult_non_work',
       dmg.CHILD_SCHOOL: fpath + '\\child_school',
       dmg.CHILD_YOUNG: fpath + '\\child_young',
      }

fpath_figure_save = chooser_fin[demo]
```

(continues on next page)

(continued from previous page)

```
# print the save figure directory
print('the figure save path:\t%s' % fpath_figure_save)

# different sets of activitiy data to plot
keys_all = chad_demo.keys

# eating activities
keys_eat = [mg.KEY_EAT_BREAKFAST, mg.KEY_EAT_LUNCH, mg.KEY_EAT_DINNER]

# non eating activities
keys_not_eat = [ k for k in keys_all if k not in keys_eat ]
```

```
the figure save path:        ..my_datafig12_07_2017n8192_d364child_young
```

Load plotting data

```
DO_ALL = 1
DO_MEALS = 2
DO_NOT_MEALS = 3

# (the activites to plot, part of the file name that matches the keys)
chooser_keys = { DO_ALL: (keys_all, 'all'), \
                DO_MEALS: (keys_eat, 'meals'),\
                DO_NOT_MEALS: (keys_not_eat, 'not_meals'),
               }
```

```
#
# set the activities to plot
#
plot_keys = DO_ALL

keys, fname_keys = chooser_keys[plot_keys]
name_keys = [ activity.INT_2_STR[k] for k in keys]


# labels on the right hand side of the plot
ylabels = ['Start Time', 'End Time', 'Duration']
```

Load all data

```
# choose the activities to plot

# get the figure directories
fpaths = [ (fpath_figure_save + mg.KEY_2_FDIR_SAVE_FIG[k] + mg.FDIR_SAVE_FIG_RANDOM_
↪DAY) for k in keys]

# the file name (no file path) of the data to save
fname = fpath_figure_save + '\\cdf_inv_' + fname_keys + '.png'

# file name to load
fnames_load = ('\\cdf_inv_start.pkl', '\\cdf_inv_end.pkl', '\\cdf_inv_dt.pkl')

# load the data
data_list_all, fname_subplot = plotter.get_figure_data(fpaths, fpath_figure_save,␣
↪fname, fnames_load=fnames_load)
```

Load the data for a specific activity-data

```
idx = -1
start = data_list_start[idx]
end = data_list_end[idx]
dt = data_list_dt[idx]

f_end = fnames_end[idx]
f_start = fnames_start[idx]
f_dt = fnames_dt[idx]

print(f_start)
print(f_end)
print(f_dt)
```

```
..my_datafig12_07_2017n8192_d364child_youngsleeprandom_daycdf_inv_start.pkl
..my_datafig12_07_2017n8192_d364child_youngsleeprandom_daycdf_inv_end.pkl
..my_datafig12_07_2017n8192_d364child_youngsleeprandom_daycdf_inv_dt.pkl
```

plot the residuals

```
#
# plot the residuals
#

alpha = 0.05
plt.close('all')

for idx, k in enumerate(keys):

    print( activity.INT_2_STR[k] )

    # load the start time, end time, and duration data
    start = data_list_start[idx]
    end = data_list_end[idx]
    dt = data_list_dt[idx]

    # quantile, and residual data
    x_start, r_start = f(start, alpha=alpha)
    x_end, r_end = f(end, alpha=alpha)
    x_dt, r_dt = f(dt, alpha=alpha)

    # covert the residuals into minutes
    r_start = r_start * 60
    r_end = r_end * 60
    r_dt = r_dt

    # get the moments on the residuals for start time, end time, and duration
    mu_start, std_start = get_moments(r_start)
    mu_end, std_end = get_moments(r_end)
    mu_dt, std_dt = get_moments(r_dt)

    print('mu start: %.2f\t\tstd start: %.2f' % (mu_start, std_start))
    print('mu end: %.2f\t\tstd end: %.2f' % (mu_end, std_end))
    print('mu dt: %.2f\t\tstd dt: %.2f\n' % (mu_dt, std_dt))

    # create subplots
    fig, axes = plt.subplots(3)
```

(continues on next page)

(continued from previous page)

```
    # create title
    fig.suptitle( activity.INT_2_STR[k] )

    # plot data about start time
    ax = axes[0]
    ax.plot(x_start, r_start, label='start')
    ax.axhline(mu_start, ls='--')
    ax.legend(loc='best')

    # plot data about end time
    ax = axes[1]
    ax.plot(x_end, r_end, label='end')
    ax.axhline(mu_end, ls='--')
    ax.legend(loc='best')

    # plot data about duration
    ax = axes[2]
    ax.plot(x_dt, r_dt, label='dt')
    ax.axhline(mu_dt, ls='--')
    ax.legend(loc='best')

plt.show()
```

```
Eat Breakfast
mu start: 11.83             std start: 8.87
mu end: 8.20                std end: 9.31
mu dt: 3.79         std dt: 4.17

Eat Lunch
mu start: 12.39             std start: 8.78
mu end: 14.46               std end: 7.60
mu dt: 2.10         std dt: 1.56

Eat Dinner
mu start: 7.21              std start: 5.18
mu end: 8.86                std end: 4.73
mu dt: 3.24         std dt: 2.95

Sleep
mu start: 5.94              std start: 4.78
mu end: 5.88                std end: 5.57
mu dt: 13.44                std dt: 10.27
```

2.3.24 longitude_plot notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
```

(continues on next page)

(continued from previous page)

```
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 20, 2018
```

This module plots the daily activity-duration for each activity over time done by an agent in an Agent-Based Module of Human Activity Patterns (ABMHAP) simulation. An agent representing each demographic are shown in a combined subplot:

1. An agent representing a respective demographic has its activity behavior is plotted in a log10 scale over time
2. This function plots a histogram showing the amount of times each activity was done in an ABMHAP simulation.

import

```
import os, sys
sys.path.append('..\\source')
sys.path.append('..\\processing')
sys.path.append('..\\plotting')

# plotting capability
import matplotlib.pylab as plt

# math capabilitiy
import numpy as np

# dataframe capability
import pandas as pd

# ABMHAP capability
import my_globals as mg
import chad_demography_adult_non_work as cdanw
import chad_demography_adult_work as cdaw
import chad_demography_child_school as cdcs
import chad_demography_child_young as cdcy
import demography as dmg

import activity, plotter, temporal
```

```
%matplotlib auto
```

```
Using matplotlib backend: Qt5Agg
```

run

```
#
# get the file name
#

# variation
fpath = mg.FDIR_MY_DATA

# file paths for each demographic
fpath_adult_work = fpath + '\\11_21_2017\\n8192_d364'
fpath_adult_non_work = fpath + '\\11_27_2017\\n8192_d364'
fpath_child_school = fpath + '\\11_29_2017\\n8192_d364'
fpath_child_young = fpath + '\\12_07_2017\\n8192_d364'

```

(continues on next page)

(continued from previous page)

```
# full file names for each demographic
fname_adult_work = fpath_adult_work + '\\data_adult_work.pkl'
fname_adult_non_work = fpath_adult_non_work + '\\data_adult_non_work.pkl'
fname_child_school = fpath_child_school + '\\data_child_school.pkl'
fname_child_young  = fpath_child_young + '\\data_child_young.pkl'

# demographic chooser
chooser = {dmg.ADULT_WORK: cdaw.CHAD_demography_adult_work(),
           dmg.ADULT_NON_WORK: cdanw.CHAD_demography_adult_non_work(),
           dmg.CHILD_SCHOOL: cdcs.CHAD_demography_child_school(),
           dmg.CHILD_YOUNG: cdcy.CHAD_demography_child_young(),
          }
```

```
#
# load demographic information
#
adult_work     = mg.load(fname_adult_work)
adult_non_work = mg.load(fname_adult_non_work)
child_school   = mg.load(fname_child_school)
child_young    = mg.load(fname_child_young)
```

```
# set the data
data_all = (adult_work, adult_non_work, child_school, child_young)

# set the titles of the data
titles   = ('Working Adults', 'Non-working Adults', 'School-age Children', 'Pre-
↪school Children')
```

```
# th index of the agent whose chosen for each demgoraphic, respectively
idx = 2

# full simulation data
diary_demo_full = [ xx.diaries[idx][0].df for xx in data_all]

# simulation data set to 14 days
diary_demo_week = []
for xx in data_all:
    df = xx.diaries[idx][0].df
    diary_demo_week.append( df[df.day <= 14])
```

plot

```
#
# plot longitudinal plots of the daily activities
#
linewidth = 0.5
data = diary_demo_week
plotter.plot_longitude(data=data, titles=titles, linewidth=linewidth)
linewidth = None

plt.show()
```

```
#
# plot the distribution of how many times each activity was done
#
```

(continues on next page)

(continued from previous page)

```
for data, title in zip(data_all, titles):
    plotter.plot_count(data, chooser[data.demographic].keys, do_abs=True, title=title)
    plotter.plot_count(data, chooser[data.demographic].keys, do_abs=False,␣
↪title=title)

plt.show()
```

2.3.25 plot_graphs notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 20, 2018
```

This notebook plots graphs comparing results from the Agent-Based Model of Human Activity Patterns (ABMHAP) to the data from the Consolidated Human Activity Database (CHAD).

1. plots the graphs of a distribution of the mean values of the agent and compares it to the distribution of CHAD mean values from the longitudinaal data for each activity start time, end time, and duration. The plots are the following:
2. plots the graphs of a distribution of 1 randomly chosen day from each agent and compares it to the distribution of CHAD single-day data for each activity start time, end time, and duration. The plots are the following:
    (a) the CDF plots of the ABMHAP distribution and CHAD distribution
    (b) the inveted CDF plots of the ABMHAP distribution and CHAD distribution
    (c) the inverted residual plots of the ABMHAP distribution and CHAD distribution
    (d) the scaled inverted residual plots of the ABMHAP distribution and CHAD distribution
3. The results of the figures are saved in a suite of .pkl files

Import

```
import os, sys
sys.path.append('..\\source')
sys.path.append('..\\processing')

# plotting capbailities
import matplotlib.pylab as plt

# ABMHAP capabilities
import my_globals as mg
import chad_demography_adult_non_work as cdanw
import chad_demography_adult_work as cdaw
import chad_demography_child_school as cdcs
```

(continues on next page)

(continued from previous page)

```
import chad_demography_child_young as cdcy
import demography as dmg
import evaluation as ev

import activity, analysis, analyzer, zipfile
```

```
%matplotlib auto
```

load the data

```
#
# load the data
#

#
# Get filename to load the data
#

# get the file name
f_data_ending = '\\12_07_2017\\n8192_d364'

# the file path directory to load the data
fpath = mg.FDIR_MY_DATA + f_data_ending

# the full file name for loading the data
fname_load_data = fpath + '\\data_child_young.pkl'

print('Loading data from:\t%s' % fname_load_data)

# clear variables
fname, fpath = None, None

# load the data
x = mg.load(fname_load_data)

# get all of the data frames
df_list = x.get_all_data()

# demographic
demo = x.demographic
```

parameters for saving the data

```
#
# Get directory to save the figrues in
#

# file directory for saving the data
fpath = mg.FDIR_SAVE_FIG + f_data_ending

# map the demographic to the correct file directory
chooser_fout = {dmg.ADULT_WORK: fpath + '\\adult_work',
        dmg.ADULT_NON_WORK: fpath + '\\adult_non_work',
        dmg.CHILD_SCHOOL: fpath + '\\child_school',
        dmg.CHILD_YOUNG: fpath + '\\child_young',
       }
```

(continues on next page)

(continued from previous page)

```
# get the file directory to save the data
fpath_save_fig = chooser_fout[demo]

print('The directory to save the data:\t%s' % fpath_save_fig)

# clear variables
fpath = None
```

the plotting parameters

```
#
# plotting flags
#

# calculates the plots
do_plot = True

# save the figures
do_save_fig = False

# show the plots
do_show = False

# show extra print messages
do_print = False
```

```
#
# demography
#

# map the demograph;y identifiyer to the demographics object
chooser = {dmg.ADULT_WORK: cdaw.CHAD_demography_adult_work(),
           dmg.ADULT_NON_WORK: cdanw.CHAD_demography_adult_non_work(),
           dmg.CHILD_SCHOOL: cdcs.CHAD_demography_child_school(),
           dmg.CHILD_YOUNG: cdcy.CHAD_demography_child_young(),
          }

# choose the demography
chad_demo = chooser[demo]
```

plot

```
# CHAD parameters
chad_param_list = chad_demo.int_2_param

# get the activity codes for a given trial
act_codes = chad_demo.keys

# the directories for the respective activities. This is used for saving the figures
fdirs = analyzer.get_verify_fpath(fpath_save_fig, act_codes)

if fpath_save_fig is None:
    do_save_fig = False

# offset, used for figure identifiers
```

(continues on next page)

(continued from previous page)

```
off = 0

# number of days in the simulation
n_days = len( df_list[0].day.unique() )

fid = 0

for act, fpath in zip(act_codes, fdirs):

    print( activity.INT_2_STR[act])
    if (do_print):
        msg = 'starting analysis for the ' + activity.INT_2_STR[act] + ' activity ...'
        print(msg)

    # this is to see if the analysis of the moments for start time needs to be in [-
↪12, 12)
    # instead of [0, 24) format
    chooser     = {activity.SLEEP: True, }
    do_periodic = chooser.get(act, False)

    # get the CHAD data
    # this is here to access the data frames from t.initialize()
    f_stats = chad_demo.fname_stats[act]

    # the sampling parameters for 1 household
    s_params = chad_demo.int_2_param[act]

    # get the CHAD data
    chad_start, chad_end, chad_dt, chad_record = \
        analysis.get_verification_info(demo=demo, key_activity=act, fname_stats=f_
↪stats, \
                                       sampling_params=[s_params] )

    # plot the ABMHAP data
    df_abm         = ev.sample_activity_abm(df_list, act)
    abm_start_mean = df_abm.start.values
    abm_end_mean   = df_abm.end.values
    abm_dt_mean    = df_abm.dt.values

    # create the plots
    if (do_plot):

        print(fpath)
        #if s_params.do_start:
        fid = fid + 1
        analyzer.plot_verify_start(act, abm_start_mean, chad_start['mu'].values,␣
↪fid=fid, \
                                   do_save_fig=do_save_fig, fpath=fpath)

        #if s_params.do_end:
        fid = fid + 1
        analyzer.plot_verify_end(act, abm_end_mean, chad_end['mu'].values, fid=fid, \
                                 do_save_fig=do_save_fig, fpath=fpath)

        #if s_params.do_dt:
        fid = fid + 1
        analyzer.plot_verify_dt(act, abm_dt_mean, chad_dt['mu'].values, fid=fid, \
```

(continues on next page)

(continued from previous page)

```
                                                  do_save_fig=do_save_fig, fpath=fpath)

if do_show:
    plt.show()
else:
    plt.close('all')
```

Validation

```
# get the CHAD sampling parameters for the given demographioc
chad_param_list = x.chad_param_list

# get the sampling parameters
s_params = chad_param_list[0]

# get the figure index
fidx = 100

# save flag
do_save = False

print(fpath_save_fig)
```

Compare random events

```
# the activity codes
act_codes = chad_demo.keys
#act_codes = [mg.KEY_WORK]

# open the data
z = zipfile.ZipFile(chad_demo.fname_zip, mode='r')

# this flag allows the code to pick a random record from the longitudinal data (if␣
↪True)
# or single-day data (if False)
do_random_long = False

# for each activity, plot the corresponding plots
for act in act_codes:

    print( activity.INT_2_STR[act] )


    # periodic time flag [-12, 12)
    do_periodic = False

    # if the activity occurs over midnight (if True), set the
    #
    if act == activity.SLEEP:
        do_periodic = True

    # sample the ABM data
    df_abm  = ev.sample_activity_abm(df_list, act)

    # get the CHAD data
    # this is here to access the data frames from t.initialize()
    f_stats = chad_demo.fname_stats[act]
```

(continues on next page)

(continued from previous page)

```
    # get the file name data of the single name data
    if do_random_long == False:
        for k in f_stats.keys():
            f_stats[k] = f_stats[k].replace('longitude', 'solo')

    # the sampling parameters for 1 household
    s_params = chad_demo.int_2_param[act]

    # get the CHAD data
    stats_start, stats_end, stats_dt, record = \
        analysis.get_verification_info(demo=demo, key_activity=act, fname_stats=f_
↪stats, \
                                       sampling_params=[s_params])

    # grouby the CHAD records by identifier
    gb  = record.groupby('PID')
    pid = record.PID.unique()

    # return true if x is in pid
    f = lambda x: x in pid

    # indices of records within 'pid'
    i = record.PID.apply(f)

    # get the CHAD observations
    df_obs = record[i]

    # get teh CHAD records that satisfy the sampling parameters for the given activity
    df_obs_new = s_params.get_record(df_obs, do_periodic)

    # get the single day observations
    print(fpath_save_fig)
    fid_last     = ev.compare_abm_to_chad_help(df_abm=df_abm, df_obs=df_obs_new, act_
↪code=act, fidx=fidx, \
                                              do_save=do_save, fpath=fpath_save_fig)
    fidx         = fid_last + 1

z.close()

print('finished plotting...')

# show the plots
if do_show:
    plt.show()
else:
    # clear all of the plots
    plt.close('all')

fpath = None
```

2.3.26 my_debug module

Warning: This is not used as part of the ABMHAP module. This should be removed.

2.3.27 omni_trial module

This is the module that is in charge of running simulations comparing the Agent-Based Model of Human Activity Patterns (ABMHAP) with the data from the Consolidated Human Activity Database (CHAD) comparing the performance of ABMHAP with all of the activity data.

This module contains class *omni_trial.Omni_Trial*.

class omni_trial.**Omni_Trial**(*parameters*, *sampling_params*, *demographic*)

Bases: *trial.Trial*

This class runs the ABMHAP simulations initialized with all of the activity data from CHAD for a given demographic. For the respective demographic, the following activity-data from CHAD are used:

- commute from work
- commute to work
- eat breakfast
- eat dinner
- eat lunch
- sleep
- work

Parameters

- **params** (params.Params) – the parameters that describe the household:
- **sampling_parameters** (dict of activity code - *chad_params.CHAD_params*) – maps an activity code to the sampling parameters to the CHAD data for the respective activity
- **demographic** (*int*) – the demographic identifier

adjust_commute_from_work(*data*, *no_variation=False*)

This function adjusts the household parameters to reflect the sampled parameters (mean and standard deviation of start time, end time, and duration, respectively), from the CHAD data for the commuting from work activity.

Parameters

- **data** (*tuple of numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray*) – relevant parameters for each person in the household for commuting from work. The tuple contains the following: mean start time, standard deviation of start time, mean end time, standard deviation of end time, mean duration, and standard deviation of duration for each person in the household.
- **no_variation** (*bool*) – whether (if True) or not (if False) intra-individual variation is set to zero among the activities

Returns

adjust_commute_to_work(*data*, *no_variation=False*)

This function adjusts the household parameters to reflect the sampled parameters (mean and standard deviation of start time, end time, and duration, respectively), from the CHAD data for the commuting to work activity.

Parameters

- **data** (*tuple of numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray*) – relevant parameters for each person in the household for commuting to work. The tuple contains the following: mean start time, standard deviation of start time, mean end time, standard deviation of end time, mean duration, and standard deviation of duration for each person in the household.
- **no_variation** (*bool*) – whether (if True) or not (if False) intra-individual variation is set to zero among the activities

Returns

adjust_eat_breakfast(*data*, *no_variation=False*)

This function adjusts the household parameters to reflect the sampled parameters (mean and standard deviation of start time, end time, and duration, respectively), from the CHAD data for the eating breakfast activity.

Parameters

- **data** (*tuple of numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray*) – relevant parameters for each person in the household for eating breakfast. The tuple contains the following: mean start time, standard deviation of start time, mean end time, standard deviation of end time, mean duration, and standard deviation of duration for each person in the household.
- **no_variation** (*bool*) – whether (if True) or not (if False) intra-individual variation is set to zero among the activities

Returns

adjust_eat_dinner(*data*, *no_variation=False*)

This function adjusts the household parameters to reflect the sampled parameters (mean and standard deviation of start time, end time, and duration, respectively), from the CHAD data for the eating dinner activity.

Parameters

- **data** (*tuple of numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray*) – relevant parameters for each person in the household for eating dinner. The tuple contains the following: mean start time, standard deviation of start time, mean end time, standard deviation of end time, mean duration, and standard deviation of duration for each person in the household.
- **no_variation** (*bool*) – whether (if True) or not (if False) intra-individual variation is set to zero among the activities

Returns

adjust_eat_lunch(*data*, *no_variation=False*)

This function adjusts the household parameters to reflect the sampled parameters (mean and standard deviation of start time, end time, and duration, respectively), from the CHAD data for the eating lunch activity.

Parameters

- **data** (*tuple of numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray*) – relevant parameters for each person in the household for eating lunch. The tuple contains the following: mean start time, standard deviation of start time, mean end time, standard deviation of end time, mean duration, and standard deviation of duration for each person in the household.
- **no_variation** (*bool*) – whether (if True) or not (if False) intra-individual variation is set to zero among the activities

Returns

adjust_params (*x*)

This function adjusts the household parameters to reflect the sampled parameters (mean and standard deviation of start time, end time, and duration, respectively), from the CHAD data for simulating the respective demographic in ABMHAP.

Parameters **x** (*dict that maps int to a tuple: numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray*) – maps an activity code to the parameterizing CHAD data for each activity, respectively. The CHAD data are the mean and standard deviation of the start time, end time, and duration.

Returns

adjust_params_adult_non_work (*x, no_variation=False*)

For the non-working adult demographic, this function adjusts the household parameters to reflect the sampled parameters (mean and standard deviation of the activity start time, end time, and duration, respectively, for the following activities:

1. eat breakfast
2. eat lunch
3. eat dinner
4. sleep

Parameters

- **x** (*dict that maps int to a tuple: numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray*) – maps an activity code to the parameterizing CHAD data for each activity, respectively. The CHAD data are the mean and standard deviation of the start time, end time, and duration.
- **no_variation** (*bool*) – off or on intra-individual variation among the activities

Returns

adjust_params_adult_work (*x, no_variation=False*)

For the working adult demographic, this function adjusts the household parameters to reflect the sampled parameters (mean and standard deviation of the activity start time, end time, and duration, respectively, for the following activities:

1. sleep
2. eat breakfast
3. eat lunch
4. eat dinner
5. commute to work

6. commute from work
7. work

Parameters

- **x** (*dict that maps int to a tuple: numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray*) – maps an activity code to the parameterizing CHAD data for each activity, respectively. The CHAD data are the mean and standard deviation of the start time, end time, and duration.
- **no_variation** (*bool*) – off or on intra-individual variation among the activities

Returns

adjust_params_child_school(*x*, *no_variation=False*)

For the school-age children demographic, this function adjusts the household parameters to reflect the sampled parameters (mean and standard deviation of the activity start time, end time, and duration, respectively, for the following activities:

1. sleep
2. eat breakfast
3. eat lunch
4. eat dinner
5. commute To work
6. commute From work
7. work

Parameters

- **x** (*dict that maps int to a tuple: numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray*) – maps an activity code to the parameterizing CHAD data for each activity, respectively. The CHAD data are the mean and standard deviation of the start time, end time, and duration.
- **no_variation** (*bool*) – off or on intra-individual variation among the activities

Returns

adjust_params_child_young(*x*, *no_variation=False*)

For the preschool children demographic, this function adjusts the household parameters to reflect the sampled parameters (mean and standard deviation of the activity start time, end time, and duration, respectively, for the following activities:

1. eat breakfast
2. eat lunch
3. eat dinner
4. sleep

Parameters

- **x** (*dict that maps int to a tuple: numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray*) – maps an activity code to the parameterizing CHAD data for each activity, respectively. The CHAD data are the mean and standard deviation of the start time, end time, and duration.
- **no_variation** (*bool*) – off or on intra-individual variation among the activities

Returns

adjust_sleep(*data*, *no_variation=False*)

This function adjusts the household parameters to reflect the sampled parameters (mean and standard deviation of start time, end time, and duration, respectively), from the CHAD data for the sleeping activity.

Parameters

- **data** (*tuple of numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray*) – relevant parameters for each person in the household for sleeping. The tuple contains the following: mean start time, standard deviation of start time, mean end time, standard deviation of end time, mean duration, and standard deviation of duration for each person in the household.
- **no_variation** (*bool*) – whether (if True) or not (if False) intra-individual variation is set to zero among the activities

Returns

adjust_work(*data*, *no_variation=False*)

This function adjusts the household parameters to reflect the sampled parameters (mean and standard deviation of start time, end time, and duration, respectively), from the CHAD data for the working activity.

Parameters

- **data** (*tuple of numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray*) – relevant parameters for each person in the household for working. The tuple contains the following: mean start time, standard deviation of start time, mean end time, standard deviation of end time, mean duration, and standard deviation of duration for each person in the household.
- **no_variation** (*bool*) – whether (if True) or not (if False) intra-individual variation is set to zero among the activities

Returns

initialize()

This function initializes the parameters for the ABMHAP simulation based on the CHAD data for the given demographic.

Returns

2.3.28 sleep_trial module

This module contains code in order to run Monte-Carlo simulations to comparing the Agent-Based Model of Human Activity Patterns (ABMHAP) with the data from the Consolidated Human Activity Database (CHAD) for the **sleep** activity.

This module contains class *sleep_trial.Sleep_Trial*.

class sleep_trial.**Sleep_Trial**(*parameters*, *sampling_params*, *demographic*)

Bases: *trial.Trial*

This class runs the ABMHAP simulations initialized with sleep data from CHAD.

Parameters

- **parameters** (params.Params) – the parameters describing each person in the household
- **sampling_params** (chad_params.CHAD_params) – the sampling parameters used to filter "good" CHAD sleep data
- **demographic** (*int*) – the demographic identifier

adjust_params (*start_mean*, *start_std*, *end_mean*, *end_std*)

This function adjusts the values for the mean and standard deviation of both sleep duration and sleep start time in the key-word arguments based on the CHAD data that was sampled. These new values will be used in the runs.

Parameters

- **start_mean** (*numpy.ndarray*) – the mean sleep start time [hours] for each person
- **start_std** (*numpy.ndarray*) – the standard deviation of sleep start time [hours] for each person
- **end_mean** (*numpy.ndarray*) – the sleep mean end time [hours] for each person
- **end_std** (*numpy.ndarray*) – the sleep standard deviation of end time [hours] for each person

Returns

create_universe ()

This function creates a universe object that simulations will run in. The only asset in this simulation for an agent to use is a *bed.Bed*.

Returns the universe

Return type *universe.Universe*

initialize ()

This function sets up the trial

1. gets the CHAD data for sleep under the appropriate conditions for means and standard deviations for both sleep duration and sleep start time
2. gets N samples the CHAD data for sleep duration and sleep start time for the N trials
3. updates the *params* to reflect the newly assigned sleep parameters for the simulation

Returns

sample_start (*df*, *s_params*)

This function is used for sampling mean and standard deviation data from start times.

Parameters

- **df** (*pandas.core.frame.DataFrame*) – the statistical start time data
- **s_params** (chad_params.CHAD_params) – the parameters the limit the sampling of CHAD data

Returns the start time time data in the range [-12, 12) [hours]

Return type pandas.core.frame.DataFrame

2.3.29 trial module

This is the module that is in charge of running simulations comparing the Agent-Based Model of Human Activity Patterns (ABMHAP) with the data from the Consolidated Human Activity Database (CHAD).

This module contains the class *trial.Trial*.

class trial.**Trial**(*parameters*, *sampling_params*, *demographic*)

Bases: object

This class is sets up runs for the ABMHAP initialized with data from CHAD.

This is how to run a trial

1. create the Trial object via __init__()
2. initialize the Trial. That is, one must set up the distribution for sampling means and standard deviations) via initialize(). This is usually done by sending the appropriate files names to the function for the respective distributions.
3. create the universe for the simulation
4. add the people to the household
5. run the simulation

Parameters

- **params** (params.Params) – the parameters that describe the household
- **sampling_params** (chad_params.CHAD_params) – the sampling parameters used to filter "good" CHAD activity data
- **demographic** (*int*) – the demographic identifier used to parametrize the agent

Variables

- **id** (*int*) – the trial identifier
- **'params'** (params.Params) – the parameters that describe the household
- **sampling_params** (chad_params.CHAD_params) – the sampling parameters used to filter "good" CHAD data
- **num_samples** (*int*) – the number of ABMHAP samples (or trials) to be run
- **demographic** (*int*) – the demographic identifier used to parametrize the agent
- **fname** (*str*) – the name of the zipfile for the CHAD data

add_person_to_universe(*u*, *idx*)

This function creates a person and sets up the universe for simulation.

Note: This function currently only assumes that each simulation has only 1 person / household. This will need to be changed later. There will be conflicts with the idx and id.

Parameters

- **u** (universe.Universe) – the universe the simulation will run in
- **idx** (*int*) – the index for *params* to access to parametrize this person.

Return u the updated/ initialized universe

Return type *universe.Universe*

assign_chad_params (*z*, *f_stats*, *s_params*)

Assign the CHAD statistical parameters for a given activity to the agent.

Parameters

- **z** (*zipfile.ZipFile*) – the file name (.zip) for the demographic data
- **f_stats** (*a dictionary of int - str*) – the file names of the statistical data relevant to the start time, end time, duration, and CHAD records for a given activity
- **s_params** (chad_params.CHAD_params) – the parameters that limit the sampling of respective statistical data

Returns relevant parameters for each person in the household for a given activity. The tuple contains the following [in hours]: mean start time, standard deviation of start time, mean end time, standard deviation of end time, mean duration, and standard deviation of duration for each person in the household.

Rtype data tuple of numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray

check_spacing (*start_mean*, *start_std*, *end_mean*, *end_std*, *spacing*)

This is done to make sure the minimum end time does not overlap with plausible start times. The function returns the indices of agents with a parametrization that causes this overlap. This is a concern for activities like sleeping where the agent can be assigned to end too early after starting the sleep too quickly.

Parameters

- **start_mean** (*numpy.ndarray*) – the mean start time for the given activity for each person in the household
- **start_std** (*numpy.ndarray*) – the standard deviation of start time for the given activity for each person in the household
- **end_mean** (*numpy.ndarray*) – the mean end time for the given activity for each person in the household
- **end_std** (*numpy.ndarray*) – the standard deviation of end time for the given activity for each person in the household
- **spacing** (*float*) – the minimum amount

Returns the indices of the agents with improper parametrization

Return type numpy.ndarray

create_universe ()

This function creates a universe object that simulations will run in.

Return u the universe for the simulation to run in

Return type *universe.Universe*

get_chad_stats_data_dt (*z*, *fname*, *s_params*)

This function obtains the CHAD data for activity duration data that are suitable for ABMHAP simulation.

Parameters

- **z** (*zifpile.Zipfile*) – the zipfile of the activity data
- **fname** (*str*) – the file name for the data file for activity duration

- **s_params** (chad_params.CHAD_params) – the parameters that limit the sampling of respective statistical data for a given activity

Returns the CHAD data for activity duration suitable for ABMHAP simulation

Return type pandas.core.frame.DataFrame

get_chad_stats_data_end(*z*, *fname*, *s_params*)

This function obtains the CHAD data for activity end time data that are suitable for ABMHAP simulation.

Parameters

- **z** (*zifpile.Zipfile*) – the zipfile of the activity data
- **fname** (*str*) – the file name for the data file for activity duration
- **s_params** (chad_params.CHAD_params) – the parameters that limit the sampling of respective statistical data for a given activity

Returns the CHAD data for activity end time suitable for ABMHAP simulation

Return type pandas.core.frame.DataFrame

get_chad_stats_data_start(*z*, *fname*, *s_params*)

This function obtains the CHAD data for activity start time data that are suitable for ABMHAP simulation.

Parameters

- **z** (*zifpile.Zipfile*) – the zipfile of the activity data
- **fname** (*str*) – the file name for the data file for activity duration
- **s_params** (chad_params.CHAD_params) – the parameters that limit the sampling of respective statistical data for a given activity

Returns the CHAD data for activity duration suitable for ABMHAP simulation

Return type pandas.core.frame.DataFrame

get_diary(*u*)

This function takes the simulation data in terms of a list of *universe.Universe* and creates a list of *diary.Diary* that contain the activity diaries. One per each household in the simulation.

Parameters **u** (universe.Universe) – contains all of the simulation data

Returns the activity diaries (1 entry per person)

Return type list of *diary.Diary*

get_diary_help(*t*, *hist_act*, *hist_loc*)

This function takes data on the activity start times, activity codes, and location codes from an activity diary and fills out the activity, minute-by-minute in between two adjacent activities.

Parameters

- **t** (*numpy.ndarray*) – the start time from an activity diary
- **hist_act** (*numpy.ndarray*) – the activity codes from an activity diary
- **hist_loc** (*numpy.ndarray*) – the location codes from an activity diary

Returns the minute by minute information from an ABMHAP simulation for the following: time information, activity codes, and location codes

Return type numpy.ndarray, numpy.ndarray, numpy.ndarray

get_stats_data(*z*, *f_stats*, *s_params*)

Assign the CHAD statistical parameters for a given activity to the agent.

Parameters

- **z** (*zipfile.ZipFile*) – the file name (.zip) for the demographic data
- **f_stats** (*a dictionary of int - str*) – the file names of the statistical data relevant to the start time, end time, duration, and CHAD records for a given activity
- **s_params** (chad_params.CHAD_params) – the parameters that limit the sampling of respective statistical data

Returns relevant parameters for each person in the household for a given activity. The tuple contains the following [in hours]: mean start time, standard deviation of start time, mean end time, standard deviation of end time, mean duration, and standard deviation of duration for each person in the household.

Rtype data tuple of numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray

get_stats_data_dt (*df*, *num_people*, *n_data*)

This function samples the duration data from CHAD from a particular activity and gets the mean and standard deviation of duration for the respective activity for each person in the household.

Parameters

- **df** (*pandas.core.frame.DataFrame*) – duration CHAD data
- **num_people** (*int*) – the number of people in the household
- **n_data** (*int*) – the minimum number of data points per CHAD-person record used in sampling the CHAD data

Returns the mean and standard deviation [in hours] for a given activity for each person in the household

Return type numpy.ndarray, numpy.ndarray

get_stats_data_help (*df*, *num_people*, *n_data*)

This function samples the CHAD data to obtain information on the mean and standard deviation data. This is done by doing the following

1. creating an empirical distribution for the mean and standard deviation of the data
2. randomly choosing a value out of the distribution for each agent in the household

Parameters

- **df** (*pandas.core.frame.DataFrame*) – the CHAD statistical data
- **num_people** (*int*) – number of people in the household
- **n_data** (*int*) – the minimum number of data points per CHAD-person record used in sampling the CHAD data

Returns the mean and standard deviation [in hours] for a given activity for each person in the household

Return type numpy.ndarray, numpy.ndarray

get_stats_data_start_end (*df_start*, *df_end*, *num_people*, *n_data*)

This function samples data for activities that are parametrized by both start time and end time activity-parameters.

Parameters

- **df_start** (*pandas.core.frame.DataFrame*) – the CHAD data for start time [hours]
- **df_end** (*pandas.core.frame.DataFrame*) – the CHAD data for end time [hours]
- **num_people** (*int*) – the number of people in the household
- **n_data** (*int*) – the number of data points to be considered "longitudinal"

Returns the mean and standard deviation for the start time and end time respectively

Return type numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray

get_stats_data_start_end_help(*df_start*, *df_end*, *num_people*, *n_data*)

This function samples data for activities that are parametrized by both start time and end time activity-parameters.

Parameters

- **df_start** (*pandas.core.frame.DataFrame*) – the CHAD data for start time [hours]
- **df_end** (*pandas.core.frame.DataFrame*) – the CHAD data for end time [hours]
- **num_people** (*int*) – the number of people in the household
- **n_data** (*int*) – the number of data points to be or not be considered "longitudinal"

Returns the mean and standard deviation for the start time and end time respectively

Return type numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray

initialize(*demo*)

This function initializes each activity in the trial for a given demographic by using CHAD data to parametrize the activity-parameters (i.e., the mean and standard deviation of star time, end time, and duration).

Parameters demo (*chad_demography.CHAD_demography*) – contains much information about the demographic

Returns a dictionary containing a tuple of the mean duration, standard deviation of duration, mean start time, standard deviation of start time (in hours, float)

Return type a dictionary of int to numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.

initialize_person(*u*, *idx*)

This function creates and initializes an agent with the proper parameters for simulation.

More specifically, the function does

1. creates the agent
2. initializes the agent's parameters to the respective values in *params*

Parameters

- **u** (*universe.Universe*) – the universe the agent will reside in
- **idx** (*int*) – the index of the agent within the household

Return p the agent

Return type *singleton.Singleton*

pseudo_intraindividual_variation (*start_mean*, *end_mean*)

This function assigns intraindividual variation for start time and end time based data where there is **no** longitudinal data (hence the name "pseudo"). The variation is assigned by having the following assumptions:

1. Given that the mean start time and end time are assigned
2. Calculate the mean duration based on the mean start time and mean end time
3. **Calculate the variance of the start time and end time with the following assumptions**
    - assume that start time and end time are independent
    - variance of start time is equal to the variance of the end time
    - standard deviation of the duration is set to be the coefficient of variation times the previously calculated mean duration

These assumptions are expressed mathematically below where

- $X_{start}, X_{end}, X_{\Delta t}$ are random variables for the start time, end time, and duration, respectively
- σ^2, σ, c_v are the variance, standard deviation, and coefficient of variation
- $E[\cdot], Cov(\cdot, \cdot)$ are the expected value operator and covariance operator

Given X_{start} and X_{end},

Let,

$$X_{\Delta t} = X_{end} - X_{start}$$

Then,

$$\sigma_{\Delta t}^2 = \sigma_{start}^2 + \sigma_{end}^2 - 2 * Cov(X_{start}, X_{end})$$

Assuming X_{start} and X_{end} are independent, then,

$$\sigma_{\Delta t}^2 = \sigma_{start}^2 + \sigma_{end}^2$$

Assuming $\sigma_{start}^2 = \sigma_{end}^2$, then,

$$\sigma_{\Delta t}^2 = 2\sigma_{start}^2$$

Finally,

$$\sigma_{start} = \frac{\sigma_{\Delta t}}{\sqrt{2}}$$

$$\sigma_{start} = \sigma_{end} = \frac{c_v E[X_{\Delta t}]}{\sqrt{2}}$$

Parameters

- **start_mean** (*numpy.ndarray*) – the mean start time [in hours] for each person bing parametrized
- **end_mean** (*numpy.ndarray*) – the mean end time [in hours] for each person being parametrized

Returns standard deviation for start time and end time, respectively for each person being parametrized

Rytpe numpy.ndarray, numpy.ndarray

run()

This function runs 1 simulations of the ABMHAP using data from CHAD. The function can handle having more than 1 person in the household.

More specifically the function does the following for each simulation:

1. creates the universe
2. create / initialize the person
3. run the ABMHAP simulation
4. store the results / data from the simulation

Return u the results of the simulation

Return type *universe.Universe*

sample(*df*)

This function samples the statistical data (of activity moments) from the CHAD diaries.

The function samples the **distributions** of both the means and the the standard deviations independently of each other.

Parameters df (*pandas.core.frame.DataFrame*) – a list of statistical data (mean, standard deviation, coefficient of variation) for activity information (duration, start, or end)

Returns values for the mean, standard deviation, and coefficient of variation, respectively

Return type numpy.ndarray, numpy.ndarray, numpy.ndarray

2.3.30 variation module

Warning: This file as antiquated and needs to be **REMOVED**.

variation.**f**(*x*)

Parameters x (*numpy.ndarray*) – the standard deviation a

Returns

variation.**integrate_residual**(*result, df_obs, act_code, do_periodic, do_weekday, do_duration, N=10001*)

variation.**run_initial**(*trials, chad_param_list, do_print=True, num_cpu=1, pool=None*)

variation.**run_simulation**(*t_0, u_list, chad_param_list, num_cpu=1, pool=None, do_print=True*)

variation.**run_trial_parallel**(*t*)

variation.**run_uni_parallel**(*t*)

variation.**run_universe_parallel**(*u*)

variation.**sweep**(*x, chad_param_list, u_list*)

variation.**sweep_parallel**(*x*)

2.3.31 work_trial module

This module contains code in order to run Monte-Carlo simulations to comparing the Agent-Based Model of Human Activity Patterns (ABMHAP) with the data from the Consolidated Human Activity Database (CHAD) for the **work** activity.

This module contains class *work_trial.Work_Trial*.

class work_trial.**Work_Trial**(*parameters*, *sampling_params*, *demographic*)

Bases: *trial.Trial*

This class runs the ABM simulations initialized with work data from CHAD.

Parameters

- **paramters** (params.Params) – the parameters describing each person in the household
- **sampling_params** (chad_params.CHAD_params) – the sampling parameters used to filter "good" CHAD work data
- **demographic** (*int*) – the demographic identifier

adjust_params(*start_mean*, *start_std*, *end_mean*, *end_std*)

This function adjusts the values for the mean and standard deviation of both work duration and work start time in the key-word arguments based on the CHAD data that was sampled. These new values will be used in the runs.

Parameters

- **dt_mean** (*numpy.ndarray*) – the work duration mean [minutes] for each person
- **dt_std** (*numpy.ndarray*) – the work duration standard deviation [minutes] for each person
- **start_mean** (*numpy.ndarray*) – the mean work start time [minutes] for each person
- **start_std** (*numpy.ndarray*) – the standard deviation of start time [minutes] for each person

Returns

create_universe()

This function creates a universe object that simulations will run in. The assets that this simulation uses in *workplace.Workplace* and *transport.Transport()*.

Returns the universe

Return type *universe.Universe*

initialize()

This function sets up the trial.

1. gets the CHAD data for work under the appropriate conditions for means and standard deviations for both work duration and sleep start time
2. gets N samples the CHAD data for work duration and work start time for the N trials
3. updates the *params* to reflect the newly assigned sleep parameters for the simulation

Returns

2.4 Plotting Directory

These functions handle plotting capabilities.

Contents:

2.4.1 plot_diary notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 20, 2018
```

This file contains the functions necessary to visualize the activity diaries.

Import

```
import sys
sys.path.append('..\\source')

# plotting functions
import matplotlib.pylab as plt

# mathematical capability
import numpy as np

# dataframe capability
import pandas as pd

# agent-based model modules
import my_globals as mg
import activity, temporal
```

```
# plotting scheme
%matplotlib auto
```

```
Using matplotlib backend: Qt5Agg
```

Functions

```
def plot_activity_diary(df, show_legend=False, fontsize=8, dpi=300):

    """
    This function plots the activity diary for a given agent. The information is␣
↪represented
    in terms of horizontal barcharts in which the agent is performing an activity and␣
↪where
```

(continues on next page)

(continued from previous page)

```
    the x-axis is the time of day (in hours).

    :param pandas.core.frame.DataFrame df: the activity diary of a given agent
    :param bool show_legend: a flag indicating whether (if True) or not (if False) to␣
↪show \
    the legend in the plot
    :param int fontsize: the font size of the text within the plot
    :param int dpi: the resolution of the plot in dots per inch

    :return: a tuple of a list of the lines that were plotted AND a list of the␣
↪labels. This \
    information is used in plotting the legend seperately
    """

    # set the font size for ticks, labels, titles, and legend
    fontsize_ticks = fontsize
    fontsize_title = fontsize
    fontsize_label = fontsize
    fontsize_title = fontsize
    fontsize_legend = fontsize

    # set font axis parameters
    font_axis = {'family': 'serif',
        'color':  'black',
        'weight': 'normal',
        'size': fontsize_ticks,}

    #
    # plot horizontal bars using matplotlib
    #

    # create the plot
    f, ax = plt.subplots(dpi=dpi)

    # a list of the lines plotted
    lines = list()
    align = 'center'

    # the labels in chornological order
    labels = [ activity.INT_2_STR[x] for x in df.act.unique() ]

    # set the label for "no actviity" to "Idle"
    for i, x in enumerate(labels):
        if x == activity.INT_2_STR[activity.NO_ACTIVITY]:
            labels[i] = 'Idle'

    # the flag to indicate whether the figure lines will be used for the legend
    do_legend = [ (x, True) for x in df.act.unique()]
    do_legend = dict(do_legend)

    # plot the diaries
    for i in range( len(df) ):

        # get the activity entry
        x = df.iloc[i]

        # get the corresponding color and label
```

(continues on next page)

(continued from previous page)

```
        color = activity.INT_2_COLOR[x.act]
        label = activity.INT_2_STR[x.act]

        # for the first entry
        if i == 0:
            # plot the entry in the beginning of the bar chart
            p = ax.barh(x.day, x.start, color=color, label=label, left=x.start,␣
↪align=align)

        else:
            # if the activity starts on one day and ends on the next,
            if x.start > x.end:
                # plot the activity entry until midnight on the first days bar chart␣
↪and
                # and starting at midnight on the next day's bar chart
                p = ax.barh(x.day, x.start, left=df.iloc[i-1].end, color=color,␣
↪label=label, align=align)
                ax.barh(x.day+1, x.end, left=0, color=color, label=label, align=align)

            else:
                # add the activity entry to the current day's bar chart
                p = ax.barh(x.day, x.start, left=df.iloc[i-1].end, color=color,␣
↪label=label, align=align)

        # if it's the first time an activity is plotted, add it to the legend.
        if do_legend[x.act]:
            lines.append(p)
            do_legend[x.act] = False

    #
    # handle the text related to plotting
    #

    # set the title
    f.suptitle('Daily Activity Diary', fontsize=fontsize_title)

    # create the legend
    if show_legend:
        f.legend(lines, labels, 'best', fontsize=fontsize_legend)

    # set the x limits
    ax.set_xlim( [0, 24])

    # set the x tick-marks
    xticks = np.linspace(0, 24, 9)
    ax.set_xticks(xticks)

    # set the font size of the x ticks
    ax.tick_params(axis='both', labelsize=fontsize_ticks)

    # label axes
    ax.set_xlabel('Time [h]', fontdict=font_axis)
    ax.set_ylabel('Day', fontdict=font_axis)

    # invert yaxis
    ax.invert_yaxis()
```

(continues on next page)

(continued from previous page)

```
    return lines, labels

def plot_longitude(data, titles, linewidth=1):

    """
    This function plots a chart showing the amount of time spent during each activity.
↪ The x-axis is the
    time in hours and the y-axis is the duration (in minutes) represented in a log10␣
↪scale.

    :param list data: a list of dataframes where each dataframe represents an␣
↪activity diary of an agent.
    :param list titles: a list of titles for each plot
    :param int linewidth: the linewidth of the lines within the plot
    """

    # the number of rows and columns (the dimensions) for the subplots
    nrows, ncols = 1, 1

    #
    # create axes
    #
    f, ax = plt.subplots(nrows, ncols, sharex=True, sharey=True)


    # plot the graphs
    K = [ plot_longitude_help(ax, data[i], linewidth) for i, ax in enumerate(f.axes)]

    # the number of unique activities, including idle time
    K0 = data[0].act.unique()

    # a list of each activity expressed as a string
    keys = [ activity.INT_2_STR[k] for k in K0]
    print(keys)

    # show the legend
    f.legend( f.axes[0].lines, keys, 'best' )

    # the subplot title size
    fontsize_title=18

    # the tick size
    ticksize=14

    # for each plot, set the font size and the tick size
    for i, ax in enumerate(f.axes):
        ax.set_title(titles[i], fontsize=fontsize_title)
        ax.tick_params(axis='both', labelsize=ticksize)

    # set the main title
    f.suptitle('Daily Activity Duration', fontsize=fontsize_title)

    # write axes for x and y
    df = data[0]
    xlabel, ylabel = 'Day', 'Duration [minutes]'
    x_min, x_max = df.day.values[0], df.day.values[-1]
```

(continues on next page)

(continued from previous page)

```
    #
    # set the x and y axes
    #

    # the y-label size
    fontsize_label = 18

    # set the ylabel
    ax.set_ylabel(ylabel, fontsize=fontsize_label)

    return

def plot_longitude_help(ax, df, linewidth=1):

    """
    This function actually handles plotting the longitude plot. This is to be used in
    plot_longitude(). For each activity, the function plots the respectivie activity-
↪duration
    on a long10 scale on each day.

    :param matplotlib.axes._subplots.AxesSubplot ax: for plotting object
    :param pandas.core.frame.DataFrame df: the activity diary of a given agent
    :param int linewidth: the linewidth of the lines within the plot

    :return: a list of the unique activity codes in the activity diary
    """

    colors = activity.INT_2_COLOR

    # the days in the simulation
    days = df.day.unique()

    # the activities that were done by the person in the simulation
    keys = df.act.unique()

    # group activities by day
    gb = df.groupby('day')

    # for each activity, plot the duration
    for k in keys:

        # the duration data
        y = np.zeros(days.shape)

        # for each day
        for i, d in enumerate(days):

            # get the activity data for the given day
            temp = gb.get_group(d)
            temp = temp[temp.act == k]

            # if there the respectivie activity does not happen that day, return NaN
            # this allows python to avoid plotting the activity on that specific day
            if temp.size == 0:
                dt = np.nan
            else:
                dt = temp.dt.values.sum()
```

(continues on next page)

(continued from previous page)

```
            # convert the duration from hours to minutes
            y[i] = temporal.HOUR_2_MIN * dt

        # plot the data for the kth activity on a log10 scale
        ax.plot(days, np.log10(y), '-*', label=activity.INT_2_STR[k], color=colors[k],
↪ linewidth=linewidth)

    return keys
```

Run

Load Activity Diary

```
# the file name of the activity diary
fname = mg.FDIR_MY_DATA + '\\main_result.csv'

# load the activity diary as a dataframe
df = pd.read_csv(fname)
```

Plot the activity diary

```
# figure resolution [ dots per inch (dpi) ]
# dpi needs to be at least 300 for submission to some journals
dpi=300

# font size of text within the figure
fontsize = 8

# plot the activity diary
lines, labels = plot_activity_diary(df, dpi=dpi, fontsize=8)

# show the plot
plt.show()
```

Isolate the legend

```
# create the plot
fig, ax = plt.subplots(dpi=dpi)

# plot the legend
fig.legend(lines, labels, 'best', fontsize=fontsize)

# do not plot anything else
ax.set_xticks([])
ax.set_yticks([])
ax.axis('off')

# show the plot
plt.show()
```

```
C:UsersnbrandonAppDataLocalContinuumAnaconda3libsite-packagesmatplotliblegend.
↪py:338: UserWarning: Automatic legend placement (loc="best") not␣
↪implemented for figure legend. Falling back on "upper right".
  warnings.warn('Automatic legend placement (loc="best") not '
```

Longitudinal Activity-Duration Plots

```
#
# plot longitudinal plots of the daily activities
#

# the title
titles = ('Working Adult',)

# the activity data
data = (df,)

# the width of the lines in the plots
linewidth = 1

# plot the activity durations
plot_longitude(data=data, titles=titles, linewidth=linewidth)

# show the plot
plt.show()
```

```
['No Activity', 'Eat Dinner', 'Sleep', 'Commute to Work', 'Work', 'Eat Lunch',
↪'Commute from Work', 'Eat Breakfast']
```

```
C:UsersnbrandonAppDataLocalContinuumAnaconda3libsite-packagesmatplotliblegend.
↪py:338: UserWarning: Automatic legend placement (loc="best") not␣
↪implemented for figure legend. Falling back on "upper right".
  warnings.warn('Automatic legend placement (loc="best") not '
```

2.4.2 plotter module

This module contains information and functions for plotting various data related to the algorithm. In short, this module is a plotting library for the algorithm.

`plotter.`**`calc_log_weight`**(*w*)

This function calculates the log10 of the weights. To avoid the possibility of getting an error due to taking log10(w=0), we zero-valued weight values to None.

Parameters **w** (*numpy.ndarray*) – the values of the weights of a corresponding need

Returns the log10 for the non-zero values of the weights

`plotter.`**`calc_weight`**(*x*, *threshold=0.2*)

This function calculates the weight value corresponding to a given value of satiation and threshold value.

Parameters

- **x** (*numpy.ndarray*) – the satiation values from an agent
- **threshold** (*float*) – the threshold value for a need

Returns an array of the weight values

`plotter.`**`get_figure_data`**(*fpaths*, *fpath_figure_save*, *fname*, *fnames_load=None*, *do_single_day=False*)

This function gets figure data from the subplots of cumulative distribution functions (CDFs) of activity-parameters (start time, end time, and duration).

Parameters

- **fpaths** (*list of str*) – a list of file paths of the figure data for each activity to load

- **fpath_figure_save** (*str*) – the file path to save the figure
- **fname** (*str*) – the file name (no file path) to save the data
- **fnames_load** (*list of str*) – the ending of the file names of the figure files to load (start time, end time, duration)
- **do_single_day** (*bool*) – a flag indicating whether to load single-day (if True) or longitudinal(if False) figure data

Returns the x and y values of the lines in the figure for start time, end time, and duration plots

Return type list, str

`plotter.`**`get_satiation_and_weight`**(*p*, *start_day*, *end_day*)

This function obtains the satiation values and weight values for the agent during the simulation over the range of the selected days.

Parameters

- **p** (`person.Person`) – the agent whose satiation and weight values are to be plotted
- **start_day** (*int*) – the day to start plotting
- **end_day** (*int*) – the day to end plotting

Returns a tuple of an array of the selected time (in hours), a list of the satiation values, and a list of the weights for the respective times

`plotter.`**`load_fig_data`**(*fname*)

Load figure data. :param str fname: the file name of the figure to load. The file must be a .pkl file.

Returns the x and y values of the lines in the figure

Return type list

`plotter.`**`plot_activity_cdfs`**(*d*, *keys*)

This function plots the cumulative distribution function of start time, end time, and duration for each activity in the the simulation.

Parameters

- **d** (`diary.Diary`) – the results of the simulation
- **keys** (*list*) – list of activities to graph

Returns

`plotter.`**`plot_activity_histograms`**(*d*, *keys*)

This function plots the histograms of start time, end time, and duration for each activity in the the simulation.

Parameters

- **d** (`diary.Diary`) – the results of the simulation
- **keys** (*list*) – list of activities to graph

Returns

`plotter.`**`plot_count`**(*data*, *keys*, *do_abs=True*, *title=None*)

This function plots a histogram showing the amount of times each activity was done in an ABMHAP simulation.

Parameters

- **data** (*pandas.core.frame.DataFrame*) – the activity diary
- **keys** (*list*) – the activity codes

- **do_abs** (*bool*) – whether (if True) to plot a histogram of the number of agents or (if False) to plot a histogram of percentage of agents
- **title** (*str*) – the title of the plot

Returns

plotter.**plot_history**(*t*, *y_list*, *labels*, *colors*, *linestyles*, *ylabel*, *linewidth=None*)

This function plots information related to data related to needs (such as satiation and weight function values) over time.

Parameters

- **t** (*numpy.ndarray*) – the time values [hours] of interest
- **y_list** (*list*) – the satiation values for each need over time
- **labels** (*list*) – the labels that corresponds to the respective need
- **colors** (*list*) – the colors that corresponds to the respective need
- **linestyles** (*list*) – the line styles that corresponds to the respective need
- **ylabel** (*str*) – the y-axis label
- **linewidth** (*int*) – the line width for each line

Returns

plotter.**plot_longitude**(*data*, *titles*, *linewidth=1*)

This function plots the day-to-day variation of activity duration for each activity over time from an ABMHAP simulation. This is done for each demographic in order to compare their differences and daily behavior. Within each subplot, an agent representing a respective demographic has its activity behavior is plotted in a log10 scale over time.

Parameters

- **data** (*list of pandas.core.frame.DataFrame*) – the activity diaries of the agents to plot. Each agent represents a different demograhic.
- **titles** (*list of str*) – the names of the demographics that are being plot
- **linewidth** (*float*) – the line width of the plot lines

Returns

plotter.**plot_longitude_help**(*ax*, *df*, *linewidth=1*)

This function plots the day-to-day variation of activity duration for each activity over time from an ABMHAP simulation. Within each subplot, an agent has its activity behavior is plotted in a log10 scale over time.

Parameters

- **ax** (*matplotlib.figure.Figure*) – the subplot object
- **df** (*pandas.core.frame.DataFrame*) – the activity diary of an agent
- **linewidth** (*float*) – the line width of the plot lines

Returns

Return type list of int

plotter.**plot_satiation_and_weight**(*p*, *start_day*, *end_day*, *fid_satiation=100*, *fid_weight=101*)

This function plots the satiation values and weight values for the agent during the simulation.

> **Warning:** This function is best used when the simulation moves through time minute by minute. If not, the slopes in both the satiation and weight plots will **not** be accurate.

Parameters

- **p** (`person.Person`) – the agent whose satiation and weight values are to be plotted
- **start_day** (*int*) – the day to start plotting
- **end_day** (*int*) – the day to end plotting
- **fid_satiation** (*int*) – the figure identifier for the satiation plot
- **fid_weight** (*int*) – the figure identifier for the weights plot

Returns

`plotter.`**`separate_activities_into_days`**(*data*)

This function finds the activities tha occur over midnight and breaks down creates a new activity diary in which an activity occurring over midnight is split into two activities: one activity entry ending at midnight, and one activity entry starting at midnight.

Parameters **data** (*pandas.core.frame.DataFrame*) – the activity diary of an agent

Returns the new activity diary

Return type pandas.core.frame.DataFrame

2.5 Processing Directory

These functions handle the logistics in dealing with CHAD.

Contents:

2.5.1 commute_school notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 22, 2018
```

This file goes through the data from the Consoldiated Human Activity Database (CHAD) and gets information relevent to **commuting to school** and **commuting from school** and processes the data for use in the Agent-Based Model of Human Activity Patterns (ABMHAP) for the school-age children demographic. More specficially, this file does the following:

For a given demographic,

1. This function goes through the CHAD data and finds the commute activity data

2. The CHAD activity data are seperated into start time, end time, duration, and CHAD record data
3. The CHAD activity data is saved into longitudinal data and single-activity data

Import

```
import sys
sys.path.append('..\\source')

# ABMHAP capability
import demography as dmg
import datum
```

```
%matplotlib notebook
```

Load

```
#
# demographic
#
# the input file and output file directory
key = dmg.CHILD_SCHOOL

# the input file and output file directory
fname_input, fpath_output = dmg.INT_2_FIN_FOUT_LARGE[key]

# load the data
data = dmg.load(fname_input)
```

Processing data

```
# get the raw commute data
d, d_to_school, d_from_school = datum.analyze_commute_school(data)
```

Plotting

```
#
# choose to save longitudinal data or single-day data
#
# note that N for the LONGITUDINAL DATA is 1
# this was done becaause there is NOT ENOUGH LONGITUDINAL DATA for adults and working
#
chooser = {True: (1, fpath_output + '\\longitude'),
           False: (1, fpath_output + '\\solo'), }

# whether to save the longitudinal data (if True) or the single-day data (if False)
# there is not enough longitudinal data to have a longitudinal model
do_long = False
```

```
# save the longitude data
do_save = False

if do_save:

    N, fpath = chooser[do_long]

    # the directories the data should be saved in
    fpaths = [fpath + '\\commute_to_work', fpath + '\\commute_from_work']
```

(continues on next page)

(continued from previous page)

```
    # the dictionaries holding the data
    data_dict = [d_to_school, d_from_school]

    # save the data
    for fpath, d in zip(fpaths, data_dict):

        stats_dt, stats_start, stats_end, record = d['stats_dt'], d['stats_start'], d[
↪'stats_end'], d['data']

        if do_long:
            dt, start, end, rec = datum.get_longitude(stats_dt, stats_start, stats_
↪end, record, N=N)
        else:
            dt, start, end, rec = datum.get_solo(stats_dt, stats_start, stats_end,␣
↪record)

        datum.save(fpath, record=rec, stats_dt=dt, stats_start=start, stats_end=end)
```

2.5.2 commute_work notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 22, 2018
```

This file goes through the data from the Consoldiated Human Activity Database (CHAD) and gets information relevent to **commuting to work**, **commuting from work**, and **working** and processes the data for use in the Agent-Based Model of Human Activity Patterns (ABMHAP) for the working adult demographic. More specficially, this file does the following:

1. This function goes through the CHAD data and finds the commute and work-activity data
2. The data is chosen such that events are chosen such that the work events are sandwiched between the commute to work and commtue from work event
3. The CHAD activity data are seperated into start time, end time, duration, and CHAD record data
4. The CHAD activity data is saved into longitudinal data and single-activity data

Import

```
import sys
sys.path.append('..\\source')

# plotting capability
import matplotlib.pylab as plt
```

(continues on next page)

(continued from previous page)

```
# ABMHAP modules
import demography as dmg
import datum
```

```
%matplotlib notebook
```

Load

```
#
# demographic
#
# the input file and output file directory
key = dmg.ADULT_WORK

# the input file and output file directory
fname_input, fpath_output = dmg.INT_2_FIN_FOUT_LARGE[key]

# load the data
data = dmg.load(fname_input)
```

Processing data

```
# analyze the commuting data
d, d_to_work, d_from_work, d_at_work = datum.analyze_commute(data)
```

Saving Data

```
# choose to save longitudinal data or single-day data
#
# note that N for the LONGITUDINAL DATA is 1
# this was done becaause there is NOT ENOUGH LONGITUDINAL DATA for adults and working
#
chooser = {True: (1, fpath_output + '\\longitude'),
           False: (1, fpath_output + '\\solo'), }

# whether to save the longitudinal data (if True) or the single-day data (if False)
# there is not enough longitudinal data to have a longitudinal model
do_long = True
```

```
# save the longitude data
do_save = False

if do_save:

    N, fpath = chooser[do_long]

    # the directories the data should be saved in
    fpaths = [fpath + '\\commute_to_work', fpath + '\\commute_from_work', fpath +
↪'\\work']

    # the dictionaries holding the data
    data_dict = [d_to_work, d_from_work, d_at_work]

    # save the data
    for fpath, d in zip(fpaths, data_dict):
```

(continues on next page)

(continued from previous page)

```
        stats_dt, stats_start, stats_end, record = d['stats_dt'], d['stats_start'], d[
↪'stats_end'], d['data']

        if do_long:
            dt, start, end, rec = datum.get_longitude(stats_dt, stats_start, stats_
↪end, record, N=N)
        else:
            dt, start, end, rec = datum.get_solo(stats_dt, stats_start, stats_end,␣
↪record)

        datum.save(fpath, record=rec, stats_dt=dt, stats_start=start, stats_end=end)
```

2.5.3 count_records notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 22, 2018
```

This function reports the amount of records from the Consolidated Human Activity Database (CHAD) records for each activity for each demographic that are suitable for use within the Agent-Based Model of Human Activity Patterns (ABMHAP) code.

import

```
#
# import
#
import sys
sys.path.append('..\\source')
sys.path.append('..\\run_chad')

# math capability
import numpy as np

# data frame capability
import pandas as pd

# zipfile capability
import zipfile

# ABMHAP modules
import my_globals as mg
import chad_demography_adult_work as cdaw
import chad_demography_adult_non_work as cdanw
import chad_demography_child_school as cdcs
import chad_demography_child_young as cdcy
```

(continues on next page)

(continued from previous page)

```
import chad
```

define functions

```
def counter(demos, names, key):

    """
    This create a dataframe that contains the amount of CHAD records for the single-
↪entry \
    and longitdinal data.

    :param demos: the demographics to compare the results to
    :type demoos: list of demography.Demography
    :param names: the names of the demographcs, respectively
    :type names: list of str
    :param int key: the ABMHAP activity code

    :return: a table the shows how many individuals have single-entry and␣
↪longitudinal data \
    within each demographic
    :retype: pandas.core.frame.DataFrame
    """

    do_periodic = False

    if key == mg.KEY_SLEEP:
        do_periodic = True

    solo_count = np.zeros( (len(demos), ) )
    long_count = np.zeros( solo_count.shape)

    for i, demo in enumerate(demos):
        solo, long = f(demo.fname_zip, demo.fname_stats[key][chad.RECORD], demo.int_2_
↪param[key],
                       do_periodic)

        solo_count[i] = sum( solo == 1 )
        long_count[i] = sum( long >= 2)

    df = pd.DataFrame( np.vstack( (solo_count, long_count) ).T )
    df.columns = ('single', 'long')
    df.index = names

    return df

def f(fname_zip, fname_record, s_param, do_periodic):

    """
    This function opens the demographic data and counts the number of both the single-
↪entry \
    (solo) records and the longitudinal (multiple-entry) records that can be used␣
↪within \
    ABMHAP according to the sepcific activity's requirements for filtering CHAD data

    :param str fname_zip: the file name of the .zip file of the CHAD data for a␣
↪specific \
```

(continues on next page)

(continued from previous page)

```
    demographic
    :param str fname_record: the file name of the CHAD record data for a given␣
↪activity \
    within the specific demographic
    :param chad_params.CHAD_params: the CHAD sampling parameters for the specific␣
↪activity
    :param bool do_periodic: a flag to inicate whether (if True) or not (if False) \
    to express time of day in hours [-12, 12)

    :return: for each person within the deographic in the CHAD data, the number of␣
↪activity \
    instances from the single-entry record data, multiple-entry record data
    :rtype: numpy.ndarray, numpy.ndarray
    """

    # the zipfile of the data for the given demographic
    z = zipfile.ZipFile(fname_zip)

    # count the number of activity instances per PID for the multiple-entry records
    long = f_temp(z, fname_record, s_param, do_periodic)

    # count the number of activity instances per PID for the single-entry records
    solo = f_temp(z, fname_record.replace('longitude', 'solo'), s_param, do_periodic)

    return solo, long

def f_temp(z, fname_record, s_param, do_periodic):

    """
    This function reads the record file and counts the number of entries of a person␣
↪in \
    CHAD for a given activity with single-entry or multiple-entry data.

    :param zipfile.Zipfile:
    :param str fname_record: the file name of the CHAD record data for a given␣
↪activity \
    within the specific demographic
    :param chad_params.CHAD_params: the CHAD sampling parameters for the specific␣
↪activity
    :param bool do_periodic: a flag to inicate whether (if True) or not (if False) \
    to express time of day in hours [-12, 12)

    :return: the number of activity instances per PID
    :rtype: numpy.ndarray
    """

    # read the record file
    df      = pd.read_csv( z.open(fname_record, mode='r') )

    # filter the dataframe for valid values for the reocrds
    df      = s_param.get_record(df, do_periodic)

    # group the records by PID
    gb      = df.groupby('PID')

    # count the number of records per PID
    counts  = np.array( [ len(gb.get_group(u)) for u in df.PID.unique() ] )
```

(continues on next page)

(continued from previous page)

```
    return counts


def print_count(demo, key, do_periodic=False):

    """
    This function prints the counts of single-entry data and longitudinal data.

    :param demography.Demography: the demographic of interest
    :int key: activity code
    :param bool do_periodic: a flag to inicate whether (if True) or not (if False) \
    to express time of day in hours [-12, 12)

    :return:
    """

    # count the number of activity instances per PID for the given activity within
    # both the single-entry data and longitudinal data
    solo, long = f(demo.fname_zip, demo.fname_stats[key][chad.RECORD], demo.int_2_
↪param[key], \
                  do_periodic)

    # print the results
    print( 'solo: %d\tlong: %d' % (sum(solo == 1), sum(long >= 2) ) )

    return
```

load the demographics information

```
#
# load demographics
#
adult_work = cdaw.CHAD_demography_adult_work()
adult_non_work = cdanw.CHAD_demography_adult_non_work()
child_school = cdcs.CHAD_demography_child_school()
child_young = cdcy.CHAD_demography_child_young()
```

```
# set the demographics and names for the data frame rows
demos = [adult_work, adult_non_work, child_school, child_young]
names = ['adult_work', 'adult_non_work', 'child_school', 'child_young']

demos_work = [adult_work, child_school]
names_work = ['adult_work', 'child_school']
```

meals and sleep

```
# breakfast
bf = counter(demos, names, mg.KEY_EAT_BREAKFAST)

# lunch
lunch = counter(demos, names, mg.KEY_EAT_LUNCH)

# dinner
dinner = counter(demos, names, mg.KEY_EAT_DINNER)
```

(continues on next page)

(continued from previous page)

```
# sleep
sleep = counter(demos, names, mg.KEY_SLEEP)
```

commuting, working

```
work = counter(demos_work, names_work, mg.KEY_WORK)
commute_to_work = counter(demos_work, names_work, mg.KEY_COMMUTE_TO_WORK)
commute_from_work = counter(demos_work, names_work, mg.KEY_COMMUTE_FROM_WORK)
```

View

```
sleep
```

2.5.4 datum module

This module contains functions that analyze the raw data from the Consolidated Human Activity Database (CHAD) to be processed/ filtered for use by the Agent-Based Model of Human Activity Patterns (ABMHAP).

This function primarily encapsulates functions to analyze data to be used as an imported module. However, it may also be run as a main file.

`datum.`**`analyze_commute`**`(`*data*`)`

This function analyzes the commuting data to get information about BOTH commuting to work, commuting from work, **AND** working. The data are chosen from entries where a work event is sandwiched between a commuting to work event and a commuting from work event. The commuting data and working data are processed and filtered for use for ABMHAP.

Parameters **`data`** (`chad.CHAD_RAW`) – the raw CHAD data

Returns the raw CHAD commuting data also the data of people with both commute and work data, statistical data of commuting to work, statistical data of commuting from work, statistical data of working.

Return type dictionary, dictionary, dictionary, dictionary

`datum.`**`analyze_commute_school`**`(`*data*`)`

This function analyzes the commuting to school data to get information to get data about commuting to school and commuting from school. The commuting to school data are processed and filtered for use for ABMHAP.

Parameters **`data`** (`chad.CHAD_RAW`) – the raw CHAD data

Returns the raw CHAD commuting data also the CHAD commuting data modified to handle over night events, statistical data of commuting to school, statistical data of commuting from school

Return type dictionary, dictionary, dictionary

`datum.`**`analyze_eat`**`(`*data*`)`

This function analyzes the CHAD data for eating in order to get information on eating breakfast, eating lunch, and eating dinner data. The data are processed and filtered for use for ABMHAP for the respective activities.

Parameters **`data`** (`chad.CHAD_RAW`) – the raw CHAD data

Returns statistical data of eating breakfast, statistical data of eating lunch, statistical data of eating dinner

Return type dictionary, dictionary, dictionary

`datum.`**`analyze_education`**`(`*data*`)`

This function analyzes the CHAD data for schooling in order to get information on going to school. The data are processed and filtered for use for ABMHAP for the school activity, namely school data are only taken if the event is considered “fulltime”, (i.e., having a long enough duration) in order to avoid part-time school events.

Parameters **data** (`chad.CHAD_RAW`) – the raw CHAD data

Returns the CHAD schooling data for “fulltime” educational data.

Return type dictionary

`datum.`**`analyze_moments`**(*df*, *start_periodic=False*)

This function analyzes the data for each person by calculating the moments for duration, start time, and end time for the following three cases.

1. General (weekday and weekend)
2. Weekday
3. Weekend

Parameters **df** (*pandas.core.frame*) – the data in the form of CHAD records to analyze

Returns the statistical moments data for the following: general duration, general start time, general end time, weekday duration, weekday start time, weekday end time, weekend duration, weekend start time, weekend end time

Return type pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DatFrame, pandas.core.frame.DataFrame

`datum.`**`analyze_sleep`**(*data*)

This function analyzes the CHAD data for sleeping in order to get information on sleeping. The data are processed and filtered for use for ABMHAP for the sleep activity.

Parameters **data** (`chad.CHAD_RAW`) – the raw CHAD data

Returns the statistical data on CHAD sleep data

Return type dictionary

`datum.`**`analyze_work`**(*data*)

This function analyzes the CHAD data for working. The data are processed and filtered for use for ABMHAP for the work activity. Data in only chosen if the person surveyed in CHAD is marked as fulltime employed. This function does a statistical analysis of the following:

1. raw work data
2. longitudinal data
3. fulltime work data

> **Warning:** This function may be antiquated and not currently used. Instead see *`analyze_commute()`* for obtaining work information.

Parameters **data** (`chad.CHAD_RAW`) – the raw CHAD data

Returns statistical data on CHAD work data on the following: raw CHAD data, raw CHAD data after being processed for overnight activities, raw CHAD data after being processed for data from people employed fulltime

Return type dictionary, dictionary, dictionary datum.**filter_commute**(*df*, *start_min*, *start_max*, *end_max*)

This function finds indices of the data that satisfy the filters placed on the commuting data by limiting the data to be within the start time range and end time range.

Parameters

- **df** (*pandas.core.frame.DataFrame*) – the commuting data
- **start_min** (*float*) – the minimum start time [hours]
- **start_max** (*float*) – the maximum start time [hours]
- **end_max** (*float*) – the maximum end time [hours]

Returns indices of the commuting data that satisfy the filtering

Return type numpy.ndarray

datum.**get_commute_data**(*df_all*)

This function finds the following commuting data for BOTH commuting to work AND commuting from work.

Parameters **df_all** (*pandas.core.frame.DataFrame*) – the dataframe containing commuting and work data

Returns the commute to work data, the commute from work data, the work activity data

datum.**get_data_help**(*idx*, *stats_dt*, *stats_start*, *stats_end*, *record*)

This function returns statistical information from the activity duration, start time, end time, and the CHAD records from the given indices.

Parameters

- **idx** (*numpy.ndarray*) – the indices of the CHAD individuals to keep in the statistical data
- **stats_dt** (*pandas.core.frame.DataFrame*) – the statistical moments for the activity duration
- **stats_start** (*pandas.core.frame.DataFrame*) – the statistical moments for the start time activity duration
- **stats_end** (*pandas.core.frame.DataFrame*) – the statistical moments for the end time activity duration
- **record** (*pandas.core.frame.DataFrame*) – the CHAD records for a given activity

Returns the statistical data on duration, start time, and end time; the CHAD record data from the chosen individuals given by the indices.

Return type pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame

datum.**get_end_date**(*date*, *start*, *end*)

This function finds the date that an activity ends.

Parameters

- **date** – the date the activities start
- **start** (*numpy.ndarray*) – the start time of the activities
- **end** (*numpy.ndarray*) – the end time of activities

Type numpy.ndarray of datetime.timedelta

Returns the end date for an activity

Return type numpy.ndarray of datetime.timedelta

`datum.`**`get_fulltime_data`**(*df*, *start_min=4*)

This function finds the data from CHAD that pertain to individuals that are working fulltime. That is, activities starting with with a minimum given mean start time.

Parameters

- **df** (*pandas.core.frame.DataFrame*) – the CHAD work data
- **start_min** (*float*) – the minimum start time to be accepted [0, 24)

Returns the data frame of the workers

Return type pandas.core.frame.DataFrame

`datum.`**`get_longitude`**(*stats_dt*, *stats_start*, *stats_end*, *record*, *N=2*)

This function gets the longitudinal CHAD statistical data for duration, start time, and end time. This function also gets the CHAD record data from the respective statistical data.

Parameters

- **stats_dt** (*pandas.core.frame.DataFrame*) – the statistical moments for the activity duration
- **stats_start** (*pandas.core.frame.DataFrame*) – the statistical moments for the start time activity duration
- **stats_end** (*pandas.core.frame.DataFrame*) – the statistical moments for the end time activity duration
- **record** (*pandas.core.frame.DataFrame*) – the CHAD records for a given activity
- **N** (*int*) – the minimum number of activities to be considered longitudinal

Returns longitudinal data for statistical moments for activity duration, start time, and end time also longitudinal CHAD records

Return type pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame

`datum.`**`get_meals`**(*df*)

This function takes in eating data and separates that data into meals: breakfast, lunch, and dinner by filtering the data by minimum and maximum start time, end time, and duration.

Parameters **df** (*pandas.core.frame.DataFrame*) – CHAD data on the eating data

Returns breakfast data, lunch data, and dinner data

Return type pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame

`datum.`**`get_moments`**(*x*, *start_periodic*)

This function calculates data about the moments of start time, end time, and duration weekday + weekend data, weekday data, weekend data. Also there are the CHAD records for the following situations: daily data, weekday data, and weekend data.

Parameters

- **x** (*pandas.core.frame.DataFrame*) – the CHAD data to be analyzed
- **start_periodic** (*bool*) – a flag indicating whether start times should be analyzed in [-12, 12) if true or [0, 24) if false

Returns a dictionary of statistical moments for the following data: duration, start time, end time, weekday duration, weekday start time, weekday end time, weekend duration, weekend start time, weekend end time. Also there are the following CHAD records: daily records, weekend records, weekday records.

Return type dictionary of pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame

`datum.`**`get_skipped_meals`**(*df*)

For each person identified within CHAD, this function goes through activity data and finds, on a workday, and finds whether or not the individual skipped a meal (i.e., skipped breakfast, lunch, and/ or dinner).

Warning: This function is antiquated and not used.

Parameters **`df`** (*`pandas.core.frame.DataFrame`*) – CHAD activity data

Returns the activity data of people within CHAD where a meal was skipped

Return type pandas.core.frame.DataFrame

`datum.`**`get_solo`**(*stats_dt*, *stats_start*, *stats_end*, *record*)

This function gets the single-day (i.e. from individuals with only 1 entry) CHAD statistical data for duration, start time, and end time. This function also gets the CHAD record data from the respective statistical data.

Parameters

- **`stats_dt`** (*`pandas.core.frame.DataFrame`*) – the statistical moments for the activity duration
- **`stats_start`** (*`pandas.core.frame.DataFrame`*) – the statistical moments for the start time activity duration
- **`stats_end`** (*`pandas.core.frame.DataFrame`*) – the statistical moments for the end time activity duration
- **`record`** (*`pandas.core.frame.DataFrame`*) – the CHAD records for a given activity

Returns single-day data for statistical moments for activity duration, start time, and end time also longitudinal CHAD records

Return type pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame

`datum.`**`get_stats`**(*pid*, *data*, *do_periodic=False*)

This function gets the statistics about an activity-parameter (start time, end time, or duration) and stores the following data within a dataframe:

1. person identifier (PID)
2. the number of events (N)
3. the mean (mu)
4. the standard deviation (std)
5. the coefficient of variation (cv)

Parameters

- **pid** (*numpy.ndarray of str*) – the identifiers for the individuals within CHAD for a given activity
- **data** (*numpy.ndarray*) – the CHAD records for a given activity
- **do_periodic** (*bool*) – a flag whether (if True) or not (if False) time of day should be expressed in [-12, 12)

Returns the statistical results from an activity-parameter (start time, end time, or duration)

Return type pandas.core.frame.DataFrame

datum.**get_stats_individual**(*x*)

This function gets the data from the records and returns the following.

1. the mean (mu)
2. the standard deviation (std)
3. the coefficient of variation (cv)
4. the number of events (N)

Parameters **x** (*numpy.ndarray*) – the individual records data

Returns the mean, standard deviation, coefficient of variation, and number of entries

Return type numpy.ndarray, numpy.ndarray, numpy.ndarray, int

datum.**get_stats_weekend**(*pid*, *data*, *date*, *start*, *end*, *do_weekend=True*, *do_periodic=False*)

This function calculates the stats about the moments of the activity that occur on a weekends OR weekedays.

Parameters

- **pid** (*numpy.ndarray of str*) – the personal identifiers in the CHAD data
- **data** – the CHAD records of the activity data
- **date** (*numpy.ndarray of datetime.timedelta*) – the dates of the activity data
- **start** (*numpy.ndarray*) – the start time of the activity data
- **end** (*numpy.ndarray*) – the end time of the activity data
- **do_weekend** (*bool*) – a flag whether (if True) to use data that occurs on the weekend or (if False) and the weekday
- **do_periodic** (*bool*) – a flag whether (if True) or not (if False) time of day should be expressed in [-12, 12)

Returns the statistical data for an activity-parameter (i.e. start time, end time, and duration) that occurs on the weekend or weekday

Return type pandas.core.frame.DataFrame

datum.**get_weekend_index**(*date*, *start*, *end*)

This function gets the indices of activity information of the weekend data.

Parameters

- **date** (*numpy.ndarray of datetime.timedelta*) – the date of the activity information
- **start** (*numpy.ndarray*) – the start time of the activity information
- **end** (*numpy.ndarray*) – the end time of the activity information

Returns this function gets the indices of activities that occur during the weekend

Return type numpy.ndarray of bool

`datum.`**`get_weekend_index_df`**`(`*df*`)`

This function gets the boolean indices of weekend data from a dataframe.

Parameters **`df`** (*`pandas.core.frame.DataFrame`*) – CHAD activity record data

Returns the boolean indices of weekend data

Return type numpy.ndarray of bool

`datum.`**`histogram`**`(`*ax*, *x*, *bins=None*, *color='b'*, *label=''*, *alpha=1.0*`)`

This function plots a histogram of the data where the y axis corresponds to the relative frequency.

Parameters

- **`ax`** (*`matplotlib.figure.Figure`*) – the plotting axis (plt or from axes)
- **`x`** (*`numpy.ndarray`*) – the data to be plotted
- **`bins`** (*`numpy.ndarray`*) – the bins for the histogram
- **`color`** (*`str`*) – the color for the histogram
- **`label`** (*`str`*) – the label of the data
- **`alpha`** (*`float`*) – the alpha for plotting

Returns

`datum.`**`merge`**`(`*df_full*`)`

For each person in the activity data, the function does the following:

1. groups the contiguous daily activity data
2. merges data that occur over midnight into one event

Parameters **`df_full`** (*`pandas.core.frame.DataFrame`*) – the full set of the activity data

Returns a data frame that merges activities that occur over midnight

Return type pandas.core.frame.DataFrame

`datum.`**`merge_end_of_day`**`(`*df*`)`

This function takes longitudinal data and merges the data if the data starts before midnight and ends after midnight.

Parameters **`df`** (*`pandas.core.frame.DataFrame`*) – the activity records data

Returns activity events that start before midnight and end after midnight

Return type pandas.core.frame.DataFrame

`datum.`**`periodicity_CHADID`**`(`*df*`)`

This function combines entries for sleep with the periodicity assumption for a given day (CHADID).

If there are two events starting at 0:00 and ending in the morning AND another event starting in the evening and ending at 0:00 on the SAME DAY, we combine the two events into one event. We assume that the person goes to sleep on the same start time and wakes up at the same time (periodicity assumption).

Parameters **`df`** (*`pandas.core.frame.DataFrame`*) – sleep events for 1 CHADID

Returns return sleep data with the periodicity assumption for 1 CHADID

Return type pandas.core.frame.DataFrame

datum.**periodicity_PID**(*df*)

Perform the periodicity assumption for a given person by its person identifier (PID).

Parameters **df** (*pandas.core.frame.DataFrame*) – the sleep data of a person with 1 PID

Returns sleep data with the periodicity assumption

Return type list of pandas.core.frame.DataFrame

datum.**periodicity_sleep**(*data*)

Perform the periodicity assumption (i.e., expressing time as [-12, 12)) for an entire dataset of multiple entries.

Parameters **data** (*pandas.core.frame.DataFrame*) – the sleep data over many individuals

Returns sleep data with the periodicity assumption

Return type pandas.core.frame.DataFrame

datum.**save**(*fpath*, *record*, *stats_dt*, *stats_start*, *stats_end*)

This function saves the following information as a .csv file:

1. the statistical moments data for the activity duration ('stats_dt.csv')
2. the statistical moments data for the activity start time ('stats_start.csv')
3. the statistical moments data for the activity end time ('stats_end.csv')
4. the statistical moments data for the activity records ('record.csv')

Parameters

- **fpath** (*str*) – the file directory in which to save the data
- **record** (*pandas.core.frame.DataFrame*) – the CHAD records for a given activity
- **stats_dt** (*pandas.core.frame.DataFrame*) – the statistical moments for the activity duration
- **stats_start** (*pandas.core.frame.DataFrame*) – the statistical moments for the start time activity duration
- **stats_end** (*pandas.core.frame.DataFrame*) – the statistical moments for the end time activity duration

Returns

datum.**sequential_data**(*df*)

For a given PID, this function groups the data in terms of sets of data for consecutive days. This function assumes that all the data given is for a given (generalized) activity.

Note: In the data, it is not necessarily the case that if there are multiple days of consecutive activity, that all of them form 1 contiguous period. Ex. It is possible to have entries Jan 1, Jan 2, Jan 3, Feb 10, Feb 11. This function will group the data into 2 groups when this occurs.

Parameters **df** (*pandas.core.frame.DataFrame*) – the data of a specific PID for an activity

Returns a list of dataframes for sequential longitudinal-data

Return type list of pandas.core.frame.DataFrame datum.**sequential_days**(*date*, *start=None*, *end=None*)

This creates label indicating sequential days. This is done by writing a sequence where each group of consecutive dates have a label starting at 0.

Note: the following sequence of dates [0, 0, 1,1, 3, 4, 5, 10], would have the following sequence [0, 0, 0, 0, 1, 1, 1, 2]

Parameters

- **date** (*numpy.ndarray datetime.timedelta*) – the date of the activity data
- **start** (*numpy.ndarray*) – the start time of the activity data
- **end** (*numpy.ndarray*) – the end time of the activity data

Returns a sequence whose indices indicates sequential dates for an activity

Return type numpy.ndarray

2.5.5 demographics notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 22, 2018
```

This file does the following

1. Goes through the Consolidated Human Activity Database (CHAD) data and seprates CHAD into datasets of different demographic groups
2. Or loads saved datasets representing different demographic groups for CHAD
3. Saves data for each demographic group:
   - Saves the demographic data into the 'data_large' directory
   - Saves the demographic in a compressed form in the 'data' directory as zip files
4. For a given demographic group and a given collection of activities
   - prints the amount of individuals found doing each activity given by a unique CHAD code
   - plots the histogram and/or CDF of distributions of start time, end time, and duration for each specific activity given by a CHAD code
   - Saves the plots

import

```
#
# import
#
import sys
sys.path.append('..\\source')
sys.path.append('..\\run_chad')
import os

# plotting capabilities
import matplotlib.pylab as plt

# math capability
import numpy as np

# ABMHAP modules
import my_globals as mg
import demography as dmg

import chad, chad_code
```

functions

```
def plot_cdfs(df, codes, N=1000, linewidth=1, do_save=False, fpath=''):

    """
    This function plots the distribution of activity distrbution of \
    start time, end time, and duration as cumulative distribution \
    functions (CDFs) from the CHAD data of the given activity.

    :param pandas.core.frame.DataFrame df:
    :param codes: the CHAD activity codes
    :type codes: list of list of int
    :param int N: the number of points sampled within the empirical CDF
    :param int linewidth: the width of the plotted lines
    :param bool do_save: a flag indicating whether (if True) to save the \
    figures or not(if False)
    :param str fpath: the file directory to save the files in

    :return:
    """

    # codes: chad_codes for each activity

    figs, fnames = [], []

    # for each activity category within the CHAD codes
    for act in codes:

        # get the data w
        temp = df[df.act == act]
        gb = temp.groupby('PID')

        # get the mean duration data
        y_dt = np.array( [ gb.get_group(p).dt.mean() for p in temp.PID.unique() ] )

        # get the mean start time data
        y_start = np.array( [ gb.get_group(p).start.mean() for p in temp.PID.unique()
↪] )
```

(continues on next page)

(continued from previous page)

```
        # get the mean end time data
        y_end = np.array( [ gb.get_group(p).end.mean() for p in temp.PID.unique() ] )

        if len(y_dt) != 0:

            # create subplots
            fig, axes = plt.subplots(2,2)

            # create title
            fig.suptitle(chad_code.INT_2_STR[act])

            # plot the start time
            ax = axes[0, 0]
            x, y = mg.get_ecdf(y_start, N)
            ax.plot(x, y, color='blue', label='start', lw=linewidth)

            # plot the end time
            ax = axes[0, 1]
            x, y = mg.get_ecdf(y_end, N)
            ax.plot(x, y, color='purple', label='end', lw=linewidth)

            # plot the duration
            ax = axes[1, 0]
            x, y = mg.get_ecdf(y_dt, N)
            ax.plot(x, y, color='red', label='duration', lw=linewidth)

            # plot axis label and legend
            for ax in axes.flatten():
                ax.set_xlabel('Hours')
                ax.legend(loc='best')

            #
            # save
            #
            if do_save:
                # figure name
                fname = fpath + chad_code.INT_2_SAVE_FIG_FNAME[act]

                # split the file name into 2 parts from the back
                x = fname.rsplit('\\', maxsplit=1)

                # create the filename
                fname = x[0] + '\\cdf\\' + x[1]

                print(fname)

                # add list of figures and finle names
                figs.append(fig)
                fnames.append(fname)

    # save the figures
    if do_save:
        for fig, fname in zip(figs, fnames):
            os.makedirs(os.path.dirname(fname), exist_ok=True)
            fig.savefig(fname, dpi=800)
            plt.close(fig)
```

(continues on next page)

(continued from previous page)

```
    return

def plot_histograms(df, codes, num_bins=12, fpath='', do_save=False):

    """
    This function plots the distribution of activity distrbution of \
    start time, end time, and duration as histograms from the CHAD \
    data of the given activity.

    :param pandas.core.frame.DataFrame df:
    :param codes: the CHAD activity codes
    :type codes: list of list of int
    :param int num_bins: the number of bins within the histogram
    :param bool do_save: a flag indicating whether (if True) to save the \
    figures or not(if False)
    :param str fpath: the file directory to save the files in

    :return:
    """

    figs, fnames = [], []

    # for each activitiy within the CHAD activity codes
    for act in codes:

        # get the data w
        temp = df[df.act == act]
        gb = temp.groupby('PID')


        # get the mean duration data
        y_dt = np.array( [ gb.get_group(p).dt.mean() for p in temp.PID.unique() ] )

        # get the mean start time data
        y_start = np.array( [ gb.get_group(p).start.mean() for p in temp.PID.unique()␣
↪] )

        # get the mean end time data
        y_end = np.array( [ gb.get_group(p).end.mean() for p in temp.PID.unique() ] )

        if len(y_dt) != 0:
            # create subplots
            fig, axes = plt.subplots(2,2)

            # create title
            fig.suptitle(chad_code.INT_2_STR[act])

            # plot the start time
            ax = axes[0, 0]
            ax.hist(y_start, bins=num_bins, color='blue', label='start')

            # plot the end time
            ax = axes[0, 1]
            ax.hist(y_end, bins=num_bins, color='purple', label='end')
```

(continues on next page)

(continued from previous page)

```
            # plot the duration
            ax = axes[1, 0]
            ax.hist(y_dt, bins=num_bins, color='red', label='duration')

            # plot axis label and legend
            for ax in axes.flatten():
                ax.set_xlabel('Hours')
                ax.legend(loc='best')

            #
            # save
            #
            if do_save:

                # figure name
                fname = fpath + chad_code.INT_2_SAVE_FIG_FNAME[act]

                # split the file name into 2 parts from the back
                x = fname.rsplit('\\', maxsplit=1)

                fname = x[0] + '\\histo\\' + x[1]

                print(fname)
                # add list of figures and finle names
                figs.append(fig)
                fnames.append(fname)

    # save the figures
    if do_save:
        for fig, fname in zip(figs, fnames):

            os.makedirs(os.path.dirname(fname), exist_ok=True)
            fig.savefig(fname, dpi=800)
            plt.close(fig)
    return

def save(x, fname):

    """
    This function saves the data for a given demographic.

    :param chad.CHAD_RAW x: the data to be pickled
    :param str fname: the name of the file
    """

    # first, close the zip file. This is necessary to avoid an pickling error
    x.z.close()

    # pickle the data
    mg.save(x, fname)

    return
```

Load data

```
# set flags
```

(continues on next page)

(continued from previous page)

```
# flag to load pre-saved CHAD data(if True) or (if False) to process the CHAD data, \
# which takes substantially more time
do_load = True

# flag to show messages
do_print = True
```

```
#
# load all of the data
#
if do_load:
    all_data  = mg.load(dmg.FNAME_ALL)
else:
    all_data = dmg.get_all()
```

```
#
# get all of the data for working age adults
#
if do_load:
    adult = mg.load(dmg.FNAME_ADULT)
else:
    adult = dmg.get_adult()
```

```
#
# get data for working adults
#
if do_load:
    adult_work = mg.load(dmg.FNAME_ADULT_WORK)
else:
    adult_work = dmg.get_adult_work(adult)
```

```
#
# get data for non-working adults
#
if do_load:
    adult_non_work = mg.load(dmg.FNAME_ADULT_NON_WORK)
else:
    adult_non_work = dmg.get_adult_non_work(adult)
```

```
#
# children school
#
if do_load:
    child_school = mg.load(dmg.FNAME_CHILD_SCHOOL)
else:
    child_school = dmg.get_child_school()
```

```
#
# pre-school children
#
if do_load:
    child_young = mg.load(dmg.FNAME_CHILD_YOUNG)
else:
    child_young = dmg.get_child_young()
```

save data

save all the information for the demographics in data_large directory

```
# save all of the information for the following demographics

do_save = False

if do_save:
    x = [all_data, adult, adult_non_work, adult_work, child_school, child_young]
    fnames = [ dmg.FNAME_ALL, dmg.FNAME_ADULT, dmg.FNAME_ADULT_NON_WORK, dmg.FNAME_
↪ADULT_WORK, \
                dmg.FNAME_CHILD_SCHOOL, dmg.FNAME_CHILD_YOUNG ]

    # save all of the data
    for y, fname in zip(x, fnames):
        save(y, fname)
```

Compress the demographics direcotory information

```
#
# The demographic
#
demos = [dmg.ADULT_WORK, dmg.ADULT_NON_WORK, dmg.CHILD_SCHOOL, dmg.CHILD_YOUNG]
```

```
#
# compress the directory in the non-large data directory
#
do_compression = False

chooser_temp = {dmg.ADULT: (chad.FNAME_ADULT[:-4], chad.FDIR_ADULT_LARGE),
           dmg.ADULT_WORK: (chad.FNAME_ADULT_WORK[:-4], chad.FDIR_ADULT_WORK_LARGE),
           dmg.ADULT_NON_WORK: (chad.FNAME_ADULT_NON_WORK[:-4], chad.FDIR_ADULT_NON_
↪WORK_LARGE),
           dmg.CHILD_SCHOOL: (chad.FNAME_CHILD_SCHOOL[:-4], chad.FDIR_CHILD_SCHOOL_
↪LARGE),
           dmg.CHILD_YOUNG: (chad.FNAME_CHILD_YOUNG[:-4], chad.FDIR_CHILD_YOUNG_
↪LARGE),
          }

if do_compression:
    for d in demos:
        fname_out, fdir_src = chooser_temp[d]
        mg.save_zip(out_file=fname_out, source_dir=fdir_src)
```

printing information about the data

```
#
# get the data
#
code_groups = [ chad_code.SLEEP, chad_code.EAT, chad_code.EDUCATION, chad_code.WORK,␣
↪chad_code.COMMUTE, \
                chad_code.COMMUTE_EDU ]

# code_groups = [chad_code.SLEEP]

df_list = [ data.activity_times(data.events, codes) for codes in code_groups ]
```

```
#
# for each CHAD code, print information about the amount of data that is in the␣
↪respective demographic group
#
for df, codes in zip(df_list, code_groups):

    if do_print:
        print('data shape')
        print(df.shape)


        print('number of individuals: %d' % len( df.PID.unique() ) )

        for act in codes:
            temp = df[df.act == act]
            print('%s:\tIndividuals:\t%d\tCount:\t%d' % (chad_code.INT_2_STR[act],␣
↪len(temp.PID.unique()), \
                                                              len(temp) ) )

        print('\n')
```

plotting

```
chooser_fpath ={dmg.ALL: mg.FDIR_SAVE_FIG_ALL,
                dmg.ADULT: mg.FDIR_SAVE_FIG_ADULT,
                dmg.ADULT_WORK: mg.FDIR_SAVE_FIG_ADULT_WORK,
                dmg.ADULT_NON_WORK: mg.FDIR_SAVE_FIG_ADULT_NON_WORK,
                dmg.CHILD_SCHOOL: mg.FDIR_SAVE_FIG_CHILD_SCHOOL,
                dmg.CHILD_YOUNG: mg.FDIR_SAVE_FIG_CHILD_YOUNG,
               }

chooser_data = {dmg.ALL: all_data,
                dmg.ADULT: adult,
                dmg.ADULT_WORK: adult_work,
                dmg.ADULT_NON_WORK: adult_non_work,
                dmg.CHILD_SCHOOL: child_school,
                dmg.CHILD_YOUNG: child_young,
               }
```

```
#
# get data and fpath for saving
#
data = chooser_data[demo]
fpath = chooser_fpath[demo] + '\\chad'

print(fpath)
```

..my_datafigdemographicadult_workchad

```
# flags for figures

# plot the figures
do_plot = False

# save the figure plots
do_save_fig= False
```

```
#
# plot the histograms
#

if do_plot:

    for df, codes in zip(df_list, code_groups):

        plot_histograms(df, codes, num_bins=24, do_save=do_save_fig, fpath=fpath)

    plt.show()
```

```
#
# plot the CDFs
#
if do_plot:

    for df, codes in zip(df_list, code_groups):

        plot_cdfs(df, codes, linewidth=2, do_save=do_save_fig, fpath=fpath)

    plt.show()
```

2.5.6 demography module

This module handles the logistics of data dealing with demographics from the raw data from the Consolidated Human Activity Database (CHAD) data in order to be used in Agent-Based Model of Human Activity Patterns (ABMHAP).

`demography.`**`filter_adult`**(*x*, *do_work*)

This function goes through the adult CHAD data and filters the results if the data is supposed to be for working adult or non-working adults.

Parameters

- **x** (`chad.CHAD_RAW`) – CHAD data for adults
- **do_work** (*bool*) – a flag indicating whether to get data from working adults (if True) or non-working adults (if False)

Returns

`demography.`**`get_adult`**()

This function gets the CHAD data for adults.

Returns the raw CHAD data from individuals that correspond to adult age

Return type *chad.CHAD_RAW*

`demography.`**`get_adult_non_work`**(*adult*)

This function gets raw CHAD data from non-working adults.

Parameters **adult** (`chad.CHAD_RAW`) – the raw adult data from CHAD

Returns raw CHAD data from non-working adults

Return type *chad.CHAD_RAW*

`demography.`**`get_adult_work`**(*adult*)

This function gets raw CHAD data from working adults.

Parameters **adult** (`chad.CHAD_RAW`) – the raw adult data from CHAD

Returns raw CHAD data from working adults

Return type *chad.CHAD_RAW*

`demography.get_all()`

This function gets all of the raw CHAD data.

Returns all of the raw CHAD data

Return type *chad.CHAD_RAW*

`demography.get_child_school()`

This function gets the CHAD data for school-age children.

Returns the raw CHAD data from individuals that correspond to school-age children

Return type *chad.CHAD_RAW*

`demography.get_child_young()`

This function gets the CHAD data for preschool children.

Returns the raw CHAD data from individuals that correspond to preschool children

Return type *chad.CHAD_RAW*

`demography.load(fname)`

This function loads data given by the file name

Parameters **fname** (`str`) – the file name of the data to load

Returns the data

`demography.save(x, fname)`

This function saves the raw CHAD data for the given demographic as a .pkl file.

Parameters

- **x** (`chad.CHAD_RAW`) – the raw CHAD data to save for a given demographic
- **fname** (`str`) – the file name to save raw CHAD data for a given demographic

Returns

2.5.7 eat_new notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 22, 2018
```

This file goes through the data from the Consoldiated Human Activity Database (CHAD) and gets information relevent to **eating breakfast**, **eating lunch**, and **eating dinner** and processes the data for use in the Agent-Based Model of Human Activity Patterns (ABMHAP) for each demographic. More specficially, this file does the following:

For a given demographic,

1. This function goes through the CHAD data and finds the eat-activity data
2. The CHAD activity data are seperated into start time, end time, duration, and CHAD record data for the meals: breakfast, lunch, and dinner
3. The CHAD activity data is saved into longitudinal data and single-activity data

Import

```
import sys
sys.path.append('..\\source')

# plotting capability
import matplotlib.pylab as plt

# ABMHAP modules
import demography as dmg
import datum
```

```
%matplotlib notebook
```

Load data

```
#
# the demographic
#
key = dmg.CHILD_YOUNG

# the input file and output file directory
fname_input, fpath_output = dmg.INT_2_FIN_FOUT_LARGE[key]

# load the data
data = dmg.load(fname_input)
```

Process the data

```
# analyze the eat-activity data
d_breakfast, d_lunch, d_dinner = datum.analyze_eat(data)
```

Plot the distribution

```
#
# plot the distribution
#
d = d_dinner

temp = d['data']

ylabel = 'Relative Frequency'
xlabel = 'Time [h]'

fig, axes = plt.subplots(2,2)

# start time
ax = axes[0,0]

datum.histogram(ax, temp.start.values, color='b', label='start')
```

(continues on next page)

(continued from previous page)

```
ax.set_ylabel(ylabel)
ax.set_xlabel(xlabel)
ax.legend(loc='best')

# end time
ax = axes[0, 1]
datum.histogram(ax, temp.end.values, color='g', label='end')
ax.set_ylabel(ylabel)
ax.set_xlabel(xlabel)
ax.legend(loc='best')


# duration
ax = axes[1, 0]
datum.histogram(ax, temp.dt.values, color='r', label='duration')
ax.set_ylabel(ylabel)
ax.set_xlabel(xlabel)
ax.legend(loc='best')

plt.show()
```

Save the data

```
# choose to save longitudinal data or single-day data
chooser = {True: (2, fpath_output + '\\longitude'),
           False: (1, fpath_output + '\\solo'), }

# whether to save the longitudinal data (if True) or the single-day data (if False)
do_long = False
```

```
#
# save the data

do_save = False

if do_save:

    N, fpath = chooser[do_long]

    # the directories the data should be saved in
    fpaths = [fpath + '\\eat_breakfast', fpath + '\\eat_lunch', fpath + '\\eat_dinner
↪']

    # the dictionaries holding the data
    data_dict = [d_breakfast, d_lunch, d_dinner]


    # save the data
    for fpath, d in zip(fpaths, data_dict):

        stats_dt, stats_start, stats_end, record = d['stats_dt'], d['stats_start'], d[
↪'stats_end'], d['data']

        if do_long:
            dt, start, end, rec = datum.get_longitude(stats_dt, stats_start, stats_
↪end, record, N=N)
```

(continues on next page)

(continued from previous page)

```
        else:
            dt, start, end, rec = datum.get_solo(stats_dt, stats_start, stats_end,␣
↪record)

        datum.save(fpath, record=rec, stats_dt=dt, stats_start=start, stats_end=end)
```

2.5.8 school_new notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 22, 2018
```

This file goes through the data from the Consoldiated Human Activity Database (CHAD) and gets information relevent to ** school** and processes the data for use in the Agent-Based Model of Human Activity Patterns (ABMHAP) for the school-age children demographic. More specficially, this file does the following:

For school-age children demographic,

1. This function goes through the CHAD data and finds the school activity data
2. The CHAD activity data are seperated into start time, end time, duration, and CHAD record data
3. The CHAD activity data is saved into longitudinal data and single-activity data

import

```
import sys
sys.path.append('..\\source')

# ABMHAP modules
import demography as dmg
import datum
```

load data

```
#
# demographic
#
key = dmg.CHILD_SCHOOL

fname_input, fpath_output = dmg.INT_2_FIN_FOUT_LARGE[key]

# load the data
data = dmg.load(fname_input)
```

process the data

```
# dictionaries about the moments
d = datum.analyze_education(data)
```

save the data

```
# choose to save longitudinal data or single-day data
chooser = {True: (2, fpath_output + '\\longitude'),
           False: (1, fpath_output + '\\solo'), }

# whether to save the longitudinal data (if True) or the single-day data (if False)
do_long = True
```

```
#
# save the data
#
do_save = False

if do_save:

    N, fpath = chooser[do_long]

    # the directory the data should be saved in
    fpath = fpath + '\\education'

    # save the data
    stats_dt, stats_start, stats_end, record = d['stats_dt'], d['stats_start'], d[
↪'stats_end'], d['data']

    if do_long:
        dt, start, end, rec = datum.get_longitude(stats_dt, stats_start, stats_end,␣
↪record, N=N)
    else:
        dt, start, end, rec = datum.get_solo(stats_dt, stats_start, stats_end, record)

    datum.save(fpath, record=rec, stats_dt=dt, stats_start=start, stats_end=end)
```

2.5.9 sleep_new notebook

```
# The United States Environmental Protection Agency through its Office of
# Research and Development has developed this software. The code is made
# publicly available to better communicate the research. All input data
# used fora given application should be reviewed by the researcher so
# that the model results are based on appropriate data for any given
# application. This model is under continued development. The model and
# data included herein do not represent and should not be construed to
# represent any Agency determination or policy.
#
# This file was written by Dr. Namdi Brandon
# ORCID: 0000-0001-7050-1538
# March 22, 2018
```

This file goes through the data from the Consoldiated Human Activity Database (CHAD) and gets information relevent to **sleeping** and processes the data for use in the Agent-Based Model of Human Activity Patterns (ABMHAP) for each demographic. More specficially, this file does the following:

For a given demographic,

1. This function goes through the CHAD data and finds the sleep-activity data
2. The CHAD activity data are seperated into start time, end time, duration, and CHAD record data
3. The CHAD activity data is saved into longitudinal data and single-activity data

Import

```
import sys
sys.path.append('..\\source')

# plotting capability
import matplotlib.pylab as plt

# ABMHAP modules
import demography as dmg
import my_globals as mg
import datum
```

```
%matplotlib notebook
```

Load

```
#
# demographic
#
demo = dmg.CHILD_YOUNG

# the input file and output file directory
fname_input, fpath_output = dmg.INT_2_FIN_FOUT_LARGE[key]

# load the data
data = dmg.load(fname_input)
```

Process data

```
# analyze the data
d_slumber = datum.analyze_sleep(data)
```

```
# get the statistical data
d = d_slumber

slumber, stats_dt, stats_start, stats_end = d['data'], d['stats_dt'], d['stats_start
↪'], d['stats_end']

slumber_we, stats_we_dt, stats_we_start, stats_we_end = \
d['data_weekend'], d['stats_we_dt'], d['stats_we_start'], d['stats_we_end']

slumber_wd, stats_wd_dt, stats_wd_start, stats_wd_end = \
d['data_weekday'], d['stats_wd_dt'], d['stats_wd_start'], d['stats_wd_end']
```

save the data

```
# the minimum number of activity entries per individual to be considered longitudinal
N_long = 2

# there is not much longitudinal information of pre-school children
if demo in [dmg.CHILD_YOUNG]:
```

(continues on next page)

(continued from previous page)

```
    N_long = 1

# choose to save longitudinal data or single-day data
chooser = {True: (N_long, fpath_output + '\\longitude'),
           False: (1, fpath_output + '\\solo'), }

# whether to save the longitudinal data (if True) or the single-day data (if False)
do_long = True
```

```
# save the and solo data
do_save = False

if do_save:

    N, fpath = chooser[do_long]

    if do_long:
        data_all = datum.get_longitude(stats_dt, stats_start, stats_end, slumber, N=N)
        data_weekend = datum.get_longitude(stats_we_dt, stats_we_start, stats_we_end,␣
↪slumber_we, N=N)
        data_weekday = datum.get_longitude(stats_wd_dt, stats_wd_start, stats_wd_end,␣
↪slumber_wd, N=N)
    else:
        data_all = datum.get_solo(stats_dt, stats_start, stats_end, slumber)
        data_weekend = datum.get_solo(stats_we_dt, stats_we_start, stats_we_end,␣
↪slumber_we)
        data_weekday = datum.get_solo(stats_wd_dt, stats_wd_start, stats_wd_end,␣
↪slumber_wd)

    # the directories the data should be saved in
    fpath = fpath + '\\sleep'
    fpaths = [ fpath + '\\all', fpath + '\\non_workday', fpath + '\\workday' ]

    # the dictionaries holding the data
    data_list = [data_all, data_weekend, data_weekday]

    # save the data
    for fpath, d in zip(fpaths, data_list):

        stats_dt, stats_start, stats_end, record = d
        datum.save(fpath, record=record, stats_dt=stats_dt, stats_start=stats_start,␣
↪stats_end=stats_end)
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

activity, 13
analysis, 89
analyzer, 91
asset, 15

b

bed, 17
bio, 17

c

chad, 19
chad_code, 22
chad_demography, 94
chad_demography_adult_non_work, 99
chad_demography_adult_work, 100
chad_demography_child_school, 100
chad_demography_child_young, 101
chad_params, 104
commute, 23
commute_from_work_trial, 107
commute_to_work_trial, 108

d

datum, 194
demography, 210
diary, 27
driver, 115
driver_params, 122
driver_result, 123

e

eat, 30
eat_breakfast_trial, 124
eat_dinner_trial, 125
eat_lunch_trial, 126
evaluation, 128

f

fig_driver, 132
food, 35

h

home, 35
hunger, 37

i

income, 40
interrupt, 41
interruption, 41

l

location, 43

m

main, 85
main_params, 86
meal, 44
my_debug, 163
my_globals, 45

n

need, 49

o

occupation, 50
omni_trial, 163

p

params, 55
person, 61
plotter, 183

r

rest, 64

s

scenario, 86
scheduler, 67

singleton, 88
sleep, 68
sleep_trial, 167
social, 70
state, 73

t

temporal, 75
transport, 77
travel, 78
trial, 169

u

universe, 79

v

variation, 175

w

work, 83
work_trial, 176
workplace, 85