cadcad-hack-2-predator-prey-sandbox-model

June 22, 2021

0.1 Table of Contents

---

1 Dependencies

[1]:

```python
# cadCAD standard dependencies

# cadCAD configuration modules
from cadCAD.configuration.utils import config_sim
from cadCAD.configuration import Experiment

# cadCAD simulation engine modules
from cadCAD.engine import ExecutionMode, ExecutionContext
from cadCAD.engine import Executor

# cadCAD global simulation configuration list
from cadCAD import configs
```

```python
# Included with cadCAD
import pandas as pd
```

[2]:
```python
# Additional dependencies

# For analytics
import numpy as np
# For visualization
import plotly.express as px
```

2 Modelling

2.1 1. State Variables

[3]:
```python
initial_state = {
    'predator_population': 15, # Initial number of predators
    'prey_population': 100, # Initial number of preys
}
initial_state
```

[3]:
```
{'predator_population': 15, 'prey_population': 100}
```

2.2 2. System Parameters

[4]:
```python
system_params = {
    # Parameters describing the interaction between the two populations:

    # A parameter used to calculate the rate of predator birth
    "predator_birth_parameter": [0.01],
    # A parameter used to calculate the rate of predator death
    "predator_death_parameter": [1.0],
    # A parameter used to calculate the rate of prey birth
    "prey_birth_parameter": [0.6, 1.0],
    # A parameter used to calculate the rate of prey death
    "prey_death_parameter": [0.03],

    # Parameters used for random Numpy variable tuning
    # These parameters scale the random variable used to create the random␣
 ↪birth / death rate

    "random_predator_birth": [0],
    "random_predator_death": [0],
    "random_prey_birth": [0],
    "random_prey_death": [0],

    # Parameter used as conversion factor between 1 unit of time and 1 timestep
```

```
    # 10 timesteps == 1 unit of time, i.e. every 10 cadCAD model timesteps, 1␣
↪unit of actual model time passes

    "dt": [0.1],
}
```

2.3 3. Policy Functions

Predator birth example:

Given a `predator_birth_parameter` of `0.01`, and a prey population of `100`, the `birth_rate` ends up being `0.01 * 100 = 1`.

This results in an increase in predators of `1 * predator_population` predators per unit of time, i.e. the predator population doubles at each unit of time!

[5]:
```python
def p_predator_births(params, substep, state_history, previous_state):
    '''Predator Births Policy Function
    The predator birth rate (rate of predators born per unit of time) is a␣
↪product of
    the prey population and the predator birth parameter plus a random variable.

    i.e. the larger the prey population, the higher the predator birth rate
    '''
    # Parameters
    dt = params['dt']
    predator_birth_parameter = params['predator_birth_parameter']
    random_predator_birth = params['random_predator_birth']

    # State Variables
    predator_population = previous_state['predator_population']
    prey_population = previous_state['prey_population']

    # Calculate the predator birth rate
    birth_rate = prey_population * (predator_birth_parameter + np.random.
↪random() * random_predator_birth)
    # Calculate change in predator population
    births = birth_rate * predator_population * dt

    return {'add_to_predator_population': births}


def p_predator_deaths(params, substep, state_history, previous_state):
    '''Predator Deaths Policy Function
    The predator death rate (rate of predators that die per unit of time) is a␣
↪function of
    the predator death parameter plus a random variable.
```

```
    i.e. the larger the predator death parameter, the higher the predator death␣
↪rate
    '''
    # Parameters
    dt = params['dt']
    predator_death_parameter = params['predator_death_parameter']
    random_predator_death = params['random_predator_death']

    # State Variables
    predator_population = previous_state['predator_population']

    # Calculate the predator death rate
    death_rate = predator_death_parameter + np.random.random() *␣
↪random_predator_death
    # Calculate change in predator population
    deaths = death_rate * predator_population * dt

    return {'add_to_predator_population': -1.0 * deaths}


def p_prey_births(params, substep, state_history, previous_state):
    '''Prey Births Policy Function
    The prey birth rate (rate of preys born per unit of time) is a function of
    the prey birth parameter plus a random variable.

    i.e. the larger the prey birth parameter, the higher the prey birth rate
    '''
    # Parameters
    dt = params['dt']
    prey_birth_parameter = params['prey_birth_parameter']
    random_prey_birth = params['random_prey_birth']

    # State Variables
    prey_population = previous_state['prey_population']

    # Calculate the prey birth rate
    birth_rate = prey_birth_parameter + np.random.random() * random_prey_birth
    # Calculate change in prey population
    births = birth_rate * prey_population * dt

    return {'add_to_prey_population': births}


def p_prey_deaths(params, substep, state_history, previous_state):
    '''Prey Deaths Policy Function
    The prey death rate (rate of preys that die per unit of time) is a product␣
↪of
```

```
    the predator population and the prey death parameter plus a random variable.

    i.e. the larger the predator population, the higher the prey death rate
    '''
    # Parameters
    dt = params['dt']
    prey_death_parameter = params['prey_death_parameter']
    random_prey_death = params['random_prey_death']

    # State Variables
    prey_population = previous_state['prey_population']
    predator_population = previous_state['predator_population']

    # Calculate the prey death rate
    death_rate = predator_population * (prey_death_parameter + np.random.
↪random() * random_prey_death)
    # Calculate change in prey population
    deaths = death_rate * prey_population * dt

    return {'add_to_prey_population': -1.0 * deaths}
```

2.4 4. State Update Functions

```
[6]: def s_predator_population(params, substep, state_history, previous_state,␣
↪policy_input):
    '''Predator Population State Update Function
    Take the Policy Input `add_to_predator_population`
    (the net predator births and deaths)
    and add to the `predator_population` State Variable.
    '''
    # Policy Inputs
    add_to_predator_population = policy_input['add_to_predator_population']

    # State Variables
    predator_population = previous_state['predator_population']

    # Calculate updated predator population
    updated_predator_population = predator_population +␣
↪add_to_predator_population

    return 'predator_population', updated_predator_population


def s_prey_population(params, substep, state_history, previous_state,␣
↪policy_input):
    '''Prey Population State Update Function
    Take the Policy Input `add_to_prey_population`
```

```
    (the net prey births and deaths)
    and add to the `prey_population` State Variable.
    '''
    # Policy Inputs
    add_to_prey_population = policy_input['add_to_prey_population']

    # State Variables
    prey_population = previous_state['prey_population']

    # Calculate updated prey population
    updated_prey_population = prey_population + add_to_prey_population

    return 'prey_population', updated_prey_population
```

2.5 5. Partial State Update Blocks

[7]:
```
partial_state_update_blocks = [
    {
        # Configure the model Policy Functions
        'policies': {
            # Calculate the predator birth rate and number of births
            'predator_births': p_predator_births,
            # Calculate the predator death rate and number of deaths
            'predator_deaths': p_predator_deaths,
            # Calculate the prey birth rate and number of births
            'prey_births': p_prey_births,
            # Calculate the prey death rate and number of deaths
            'prey_deaths': p_prey_deaths,
        },
        # Configure the model State Update Functions
        'variables': {
            # Update the predator population
            'predator_population': s_prey_population,
            # Update the prey population
            'prey_population': s_predator_population
        }
    }
]
```

3 Simulation

3.1 6. Configuration

[8]:
```
sim_config = config_sim({
    "N": 1, # the number of times we'll run the simulation ("Monte Carlo runs")
    "T": range(400), # the number of timesteps the simulation will run for
    "M": system_params # the parameters of the system
```

```
})
```

[9]:
```python
del configs[:] # Clear any prior configs
```

[10]:
```python
experiment = Experiment()
experiment.append_configs(
    initial_state = initial_state,
    partial_state_update_blocks = partial_state_update_blocks,
    sim_configs = sim_config
)
```

3.2 7. Execution

[11]:
```python
exec_context = ExecutionContext()
simulation = Executor(exec_context=exec_context, configs=configs)
raw_result, tensor_field, sessions = simulation.execute()
```

```
                  ___________    ____
  ________ __ ___/ / ____/   |  / __ \
 / ___/ __` / __  / /   / /| | / / / /
/ /__/ /_/ / /_/ / /___/ ___ |/ /_/ /
\___/\__,_/\__,_/\____/_/  |_/_____/
by cadCAD

Execution Mode: local_proc
Configuration Count: 1
Dimensions of the first simulation: (Timesteps, Params, Runs, Vars) = (400, 9,
2, 2)
Execution Method: local_simulations
SimIDs   : [0, 0]
SubsetIDs: [0, 1]
Ns       : [0, 1]
ExpIDs   : [0, 0]
Execution Mode: parallelized
Total execution time: 0.04s
```

3.3 8. Output Preparation

[12]:
```python
# Convert raw results to a Pandas DataFrame
df = pd.DataFrame(raw_result)

# Insert cadCAD parameters for each configuration into DataFrame
for config in configs:
    # Get parameters from configuration
    parameters = config.sim_config['M']
    # Get subset index from configuration
    subset_index = config.subset_id
```

```python
    # For each parameter key value pair
    for (key, value) in parameters.items():
        # Select all DataFrame indices where subset == subset_index
        dataframe_indices = df.eval(f'subset == {subset_index}')
        # Assign each parameter key value pair to the DataFrame for the␣
↪corresponding subset
        df.loc[dataframe_indices, key] = value

df.head(10)
```

[12]:

```
   predator_population  prey_population  simulation  subset  run  substep  \
0            15.000000       100.000000           0       0    1        0
1            15.000000       101.500000           0       0    1        1
2            15.022500       103.022500           0       0    1        1
3            15.067906       104.560883           0       0    1        1
4            15.136628       106.107996           0       0    1        1
5            15.229083       107.656124           0       0    1        1
6            15.345679       109.196979           0       0    1        1
7            15.486813       110.721693           0       0    1        1
8            15.652857       112.220816           0       0    1        1
9            15.844148       113.684335           0       0    1        1

   timestep  predator_birth_parameter  predator_death_parameter  \
0         0                      0.01                       1.0
1         1                      0.01                       1.0
2         2                      0.01                       1.0
3         3                      0.01                       1.0
4         4                      0.01                       1.0
5         5                      0.01                       1.0
6         6                      0.01                       1.0
7         7                      0.01                       1.0
8         8                      0.01                       1.0
9         9                      0.01                       1.0

   prey_birth_parameter  prey_death_parameter  random_predator_birth  \
0                   0.6                  0.03                    0.0
1                   0.6                  0.03                    0.0
2                   0.6                  0.03                    0.0
3                   0.6                  0.03                    0.0
4                   0.6                  0.03                    0.0
5                   0.6                  0.03                    0.0
6                   0.6                  0.03                    0.0
7                   0.6                  0.03                    0.0
8                   0.6                  0.03                    0.0
9                   0.6                  0.03                    0.0
```

	random_predator_death	random_prey_birth	random_prey_death	dt
0	0.0	0.0	0.0	0.1
1	0.0	0.0	0.0	0.1
2	0.0	0.0	0.0	0.1
3	0.0	0.0	0.0	0.1
4	0.0	0.0	0.0	0.1
5	0.0	0.0	0.0	0.1
6	0.0	0.0	0.0	0.1
7	0.0	0.0	0.0	0.1
8	0.0	0.0	0.0	0.1
9	0.0	0.0	0.0	0.1

3.4 9. Analysis

[13]:
```
# Visualize how the predator and prey populations change over time

# Notice that the populations are more chaotic when the prey birth rate is␣
↪higher,
# and the system is more stable when it is lower.

px.line(
    df,
    x='timestep', # Variable on the horizontal axis
    y=['predator_population', 'prey_population'], # Variables on the vertical␣
↪axis
    line_group='run', # One line for each MC run
    facet_row='prey_birth_parameter', # Create a figure for each␣
↪`prey_birth_parameter` parameter sweep
    log_y=True, # Use log scale on the vertical axis
    height=800,
)
```

[14]:
```
# Visualize how the predator and prey populations compare

# Notice that they tend to have cycles around a fixed point
# which means that their populations are never static,
# but rather cyclic with ups and downs.

px.line(
    df,
    x='predator_population', # Variable on the horizontal axis
    y='prey_population', # Variable on the vertical axis
    color='run', # Color by MC run
    facet_row='prey_birth_parameter', # Create a figure for each␣
↪`prey_birth_parameter` parameter sweep
    log_x=True, # Use log scale on the horizontal axis
    log_y=True, # Use log scale on the vertical axis
```

```
    height=800,
)
```