

minimal_prey_predator

March 10, 2021

1 Prey-Predator simulation in cadCAD

Danilo Lessa Bernardineli

This is an cadCAD simulation of the Lotka-Volterra prey-predator model. This is a minimal example contained inside a notebook, and it can be used for quick experimentations

1.1 Dependences

```
[1]: %%capture
      !pip install cadcad
```

```
[2]: import pandas as pd
      import matplotlib.pyplot as plt
      import numpy as np
      from cadCAD.configuration import Experiment
      from cadCAD.configuration.utils import config_sim
      from cadCAD.engine import ExecutionMode, ExecutionContext, Executor
```

1.2 Definitions

1.2.1 Initial conditions and parameters

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

      params = {
          "prey_birth_rate": [1.0],
          "predator_birth_rate": [0.01],
          "predator_death_const": [1.0],
          "prey_death_const": [0.03],
          "dt": [0.1] # Precision of the simulation. Lower is more accurate / slower
      }

      simulation_parameters = {
```

```

    'N': 7,
    'T': range(200),
    'M': params
}

```

1.2.2 Policies

```

[4]: def p_predator_births(params, step, sL, s):
    dt = params['dt']
    predator_population = s['predator_population']
    prey_population = s['prey_population']
    birth_fraction = params['predator_birth_rate'] + np.random.random() * 0.0002
    births = birth_fraction * prey_population * predator_population * dt
    return {'add_to_predator_population': births}

def p_pre_y_births(params, step, sL, s):
    dt = params['dt']
    population = s['prey_population']
    birth_fraction = params['prey_birth_rate'] + np.random.random() * 0.1
    births = birth_fraction * population * dt
    return {'add_to_pre_y_population': births}

def p_predator_deaths(params, step, sL, s):
    dt = params['dt']
    population = s['predator_population']
    death_rate = params['predator_death_const'] + np.random.random() * 0.005
    deaths = death_rate * population * dt
    return {'add_to_predator_population': -1.0 * deaths}

def p_pre_y_deaths(params, step, sL, s):
    dt = params['dt']
    death_rate = params['prey_death_const'] + np.random.random() * 0.1
    prey_population = s['prey_population']
    predator_population = s['predator_population']
    deaths = death_rate * prey_population * predator_population * dt
    return {'add_to_pre_y_population': -1.0 * deaths}

```

1.2.3 State update functions

```

[5]: def s_pre_y_population(params, step, sL, s, _input):
    y = 'prey_population'
    x = s['prey_population'] + _input['add_to_pre_y_population']
    return (y, x)

```



```
Execution Mode: local_proc
Configuration Count: 1
Dimensions of the first simulation: (Timesteps, Params, Runs, Vars) = (200, 5,
7, 2)
Execution Method: local_simulations
SimIDs   : [0, 0, 0, 0, 0, 0, 0]
SubsetIDs: [0, 0, 0, 0, 0, 0, 0]
Ns        : [0, 1, 2, 3, 4, 5, 6]
ExpIDs    : [0, 0, 0, 0, 0, 0, 0]
Execution Mode: parallelized
Total execution time: 0.10s
```

1.2.6 Results

```
[8]: import plotly.express as px
```

```
[9]: df = pd.DataFrame(records)

fig = px.line(df,
               x=df.prey_population,
               y=df.predator_population,
               color=df.run.astype(str))

fig.show()
```