# simulation_features_minimum_example

March 14, 2021

[1]:
```
import math
import pandas as pd
import plotly
from numpy import random

pd.options.plotting.backend = "plotly"

from cadCAD.configuration.utils import config_sim

from cadCAD.engine import ExecutionMode, ExecutionContext
from cadCAD.engine import Executor

from cadCAD.configuration import Experiment

from cadCAD import configs
del configs[:] # Clear any prior configs

experiment = Experiment()
```

[2]:
```
MONTE_CARLO_RUNS = 3

seeds = [random.RandomState(i) for i in range(MONTE_CARLO_RUNS)]
```

[3]:
```
initial_state = {
    'population': 50, # number of sheep
    'food': 1000 # tons of grass
}

system_params = {
    'reproduction_rate': [0.3],
    'death_rate': [0.03],
    'consumption_rate': [0.03],
    'growth_rate': [30.0],
}
```

[4]:
```
def s_population(params, substep, state_history, previous_state, policy_input):
    population = previous_state['population'] + policy_input['delta_population']
```

```
    return 'population', max(math.ceil(population), 0)

def s_food(params, substep, state_history, previous_state, policy_input):
    food = previous_state['food'] + policy_input['delta_food']
    return 'food', max(food, 0)

def p_reproduction(params, substep, state_history, previous_state):
    population_reproduction = params['reproduction_rate'] *␣
 ↪previous_state['food']
    return {'delta_population': population_reproduction}

def p_death(params, substep, state_history, previous_state):
    population_death = params['death_rate'] * previous_state['population']
    return {'delta_population': -population_death}

def p_growth(params, substep, state_history, previous_state):
    run = previous_state['run']
    food_growth = params['growth_rate'] * seeds[run - 1].rand()
    return {'delta_food': food_growth}

def p_consumption(params, substep, state_history, previous_state):
    food_consumption = params['consumption_rate'] * previous_state['population']
    return {'delta_food': -food_consumption}
```

[5]:
```
partial_state_update_blocks = [
    {
        'policies': {
            'reproduction': p_reproduction,
            'death': p_death,
            'consumption': p_consumption,
            'growth': p_growth
        },
        'variables': {
            'population': s_population,
            'food': s_food
        }
    }
]
```

[6]:
```
del configs[:]

SIMULATION_TIMESTEPS = 500

sim_config = config_sim({
    'N': MONTE_CARLO_RUNS,
    'T': range(SIMULATION_TIMESTEPS),
    'M': system_params
```

```
})

experiment.append_configs(
    initial_state = initial_state,
    partial_state_update_blocks = partial_state_update_blocks,
    sim_configs = sim_config,
)
```

[7]:
```
exec_mode = ExecutionMode()
exec_context = ExecutionContext()

simulation = Executor(exec_context=exec_context, configs=configs)
```

[8]:
```
raw_result, tensor_field, sessions = simulation.execute()
df = pd.DataFrame(raw_result)
sim_0 = df[df.simulation == 0]
```

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

Execution Mode: local_proc
Configuration Count: 1
Dimensions of the first simulation: (Timesteps, Params, Runs, Vars) = (500, 4,
3, 2)
Execution Method: local_simulations
SimIDs   : [0, 0, 0]
SubsetIDs: [0, 0, 0]
Ns       : [0, 1, 2]
ExpIDs   : [0, 0, 0]
Execution Mode: parallelized
Total execution time: 0.09s
```

[9]:
```
simulation=0
```

[10]:
```
seeds[simulation - 1].rand()
```

[10]:
```
0.26658984055727786
```

[11]:
```
df[['simulation','subset','run','substep']].drop_duplicates()
```

[11]:

| | simulation | subset | run | substep |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |

```
1               0       0    1         1
501             0       0    2         0
502             0       0    2         1
1002            0       0    3         0
1003            0       0    3         1
```

[12]:
```
import plotly.express as px

fig = px.line(df, x='timestep', y=['population', 'food'], facet_row='run',␣
↪height=800, width=1000, title='Monte Carlo Runs')
fig.layout.font.size = 20
fig.show()
```

### 0.1 Parameter Sweeps

[13]:
```
initial_state = {
    'population': 50, # number of sheep
    'food': 1000 # tons of grass
}

system_params = {
    'reproduction_rate': [0.1, 0.2, 0.3],
    'death_rate': [0.01, 0.02, 0.03],
    'consumption_rate': [0.01, 0.02, 0.03],
    'growth_rate': [10, 20, 30.0],
}
```

[14]:
```
def s_population(params, substep, state_history, previous_state, policy_input):
    population = previous_state['population'] + policy_input['delta_population']
    return 'population', max(math.ceil(population), 0)

def s_food(params, substep, state_history, previous_state, policy_input):
    food = previous_state['food'] + policy_input['delta_food']
    return 'food', max(food, 0)

def p_reproduction(params, substep, state_history, previous_state):
    population_reproduction = params['reproduction_rate'] *␣
↪previous_state['food']
    return {'delta_population': population_reproduction}

def p_death(params, substep, state_history, previous_state):
    population_death = params['death_rate'] * previous_state['population']
    return {'delta_population': -population_death}

def p_growth(params, substep, state_history, previous_state):
    run = previous_state['run']
```

```
    food_growth = params['growth_rate']
    return {'delta_food': food_growth}

def p_consumption(params, substep, state_history, previous_state):
    food_consumption = params['consumption_rate'] * previous_state['population']
    return {'delta_food': -food_consumption}
```

[15]:
```
partial_state_update_blocks = [
    {
        'policies': {
            'reproduction': p_reproduction,
            'death': p_death,
            'consumption': p_consumption,
            'growth': p_growth
        },
        'variables': {
            'population': s_population,
            'food': s_food
        }
    }
]
```

[16]:
```
MONTE_CARLO_RUNS = 1

# del configs[:]

SIMULATION_TIMESTEPS = 500

sim_config = config_sim({
    'N': MONTE_CARLO_RUNS,
    'T': range(SIMULATION_TIMESTEPS),
    'M': system_params,
})

experiment.append_configs(
    initial_state = initial_state,
    partial_state_update_blocks = partial_state_update_blocks,
    sim_configs = sim_config,
)
```

[17]:
```
exec_mode = ExecutionMode()
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: 2
Dimensions of the first simulation: (Timesteps, Params, Runs, Vars) = (500, 4,
3, 2)
Execution Method: local_simulations
SimIDs   : [0, 0, 0, 1, 1, 1]
SubsetIDs: [0, 0, 0, 0, 1, 2]
Ns       : [0, 1, 2, 0, 1, 2]
ExpIDs   : [0, 0, 0, 1, 1, 1]
Execution Mode: parallelized
Total execution time: 0.18s
```

[18]:
```
df = pd.DataFrame(raw_result)
df[['simulation','subset','run','substep']].drop_duplicates()
```

[18]:

| | simulation | subset | run | substep |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 501 | 0 | 0 | 2 | 0 |
| 502 | 0 | 0 | 2 | 1 |
| 1002 | 0 | 0 | 3 | 0 |
| 1003 | 0 | 0 | 3 | 1 |
| 1503 | 1 | 0 | 1 | 0 |
| 1504 | 1 | 0 | 1 | 1 |
| 2004 | 1 | 1 | 2 | 0 |
| 2005 | 1 | 1 | 2 | 1 |
| 2505 | 1 | 2 | 3 | 0 |
| 2506 | 1 | 2 | 3 | 1 |

[19]:
```
df = df[df.simulation == 1]
```

[20]:
```
df
```

[20]:

| | population | food | simulation | subset | run | substep | timestep |
|---|---|---|---|---|---|---|---|
| 1503 | 50 | 1000.00 | 1 | 0 | 1 | 0 | 0 |
| 1504 | 150 | 1009.50 | 1 | 0 | 1 | 1 | 1 |
| 1505 | 250 | 1018.00 | 1 | 0 | 1 | 1 | 2 |
| 1506 | 350 | 1025.50 | 1 | 0 | 1 | 1 | 3 |
| 1507 | 450 | 1032.00 | 1 | 0 | 1 | 1 | 4 |
| … | … | … | … | … | … | … | … |

```
3001        1000    99.96           1       2    3        1       496
3002        1000    99.96           1       2    3        1       497
3003        1000    99.96           1       2    3        1       498
3004        1000    99.96           1       2    3        1       499
3005        1000    99.96           1       2    3        1       500

[1503 rows x 7 columns]
```

[21]:
```
import plotly.express as px



fig = px.line(df, x='timestep', y=['population', 'food'], facet_row='subset',␣
 ↪height=800, width=1000, title='Parameter Sweep Results')
fig.layout.font.size = 20
fig.show()
```

### 0.2 A|B Testing

[22]:
```
import plotly.express as px

df = pd.DataFrame(raw_result)
fig = px.line(
    df,
    x='timestep',
    y=['population', 'food'],
    facet_row='run',
    facet_col='simulation',
    height=700,
    template='seaborn',
    title="A|B Testing"
)
fig.layout.font.size = 21

fig.update_layout(
    margin=dict(l=20, r=10, t=60, b=20),
)

fig.show()
```