BDI

November 6, 2024

[1]:
```
import agentpy as ap
import numpy as np
import seaborn as sns
import pandas as pd
import random
import matplotlib.pyplot as plt
```

[2]:
```
def gini(x):
    """ Calcular el Coeficiente de Gini """
    x = np.array(x)
    mad = np.abs(np.subtract.outer(x, x)).mean()
    rmad = mad / np.mean(x)
    return 0.5 * rmad
class BaseWealthAgent(ap.Agent):
    """ Agente base con riqueza """
    def setup(self):
        self.wealth = 1
        self.strategy_name = "Base"

    def wealth_transfer(self):
        pass
```

[3]:
```
class BDIAgent(BaseWealthAgent):
    def setup(self):
        super().setup()
        self.strategy_name = "BDI"
        self.wealth = 2
        self.beliefs = {'wealth': self.wealth}
        self.desires = {'save_wealth': True}
        self.intentions = []

    def update_beliefs(self):
        # Update beliefs based on current wealth
        self.beliefs['wealth'] = self.wealth

    def generate_options(self):
        # Generate possible options based on beliefs and desires
        options = []
```

```
        if self.beliefs['wealth'] > 2:
            options.append('transfer')
        return options

    def filter_options(self, options):
        # Filter options to form intentions
        if 'transfer' in options:
            self.intentions = ['transfer']
        else:
            self.intentions = []

    def create_plan(self):
        # Create a plan based on intentions
        if 'transfer' in self.intentions:
            partner = list(self.model.agents.random(n=1))[0]
            return lambda: self.transfer_wealth(partner)
        return lambda: None

    def transfer_wealth(self, partner):
        partner.wealth += 1
        self.wealth -= 1

    def wealth_transfer(self):
        self.update_beliefs()
        options = self.generate_options()
        self.filter_options(options)
        plan = self.create_plan()
        plan()
```

[4]:
```
class RiskTakingAgent(BaseWealthAgent):
    def setup(self):
        super().setup()
        self.strategy_name = "RiskTaker"
        self.wealth = 4

    def wealth_transfer(self):
        if self.wealth > 2:
            partner = list(self.model.agents.random(n=1))[0]
            transfer = min(self.wealth - 1, 3)
            partner.wealth += transfer
            self.wealth -= transfer
```

[5]:
```
class DeductiveReasoningAgent(BaseWealthAgent):
    def setup(self):
        super().setup()
        self.strategy_name = "DeductiveReasoning"
        self.beliefs = {'partner': None}
```

```
        self.actions = [self.wealth_transfer]
        self.rules = [self.rule_1]

    def see(self, agents):
        per = agents.random()
        self.beliefs['partner'] = per

    def next(self):
        for act in self.actions:
            for rule in self.rules:
                if rule(act):
                    return act
        return None

    def action(self, act):
        if act is not None:
            act()

    def step(self):
        self.see(self.model.agents)
        a = self.next()
        self.action(a)

    def rule_1(self, act):
        rule_validation = [False, False, False]
        if self.wealth > 0:
            rule_validation[0] = True
        if self.beliefs["partner"] is not None:
            rule_validation[1] = True
        if act == self.wealth_transfer:
            rule_validation[2] = True
        return all(rule_validation)

    def wealth_transfer(self):
        if self.beliefs['partner'] is not None:
            self.beliefs['partner'].wealth += 1
            self.wealth -= 1
```

[6]:
```
class WealthModel(ap.Model):
    """ Un modelo de transferencias de riqueza entre diferentes estrategias de␣
↪agentes """

    def setup(self):
        # Crear una lista de agentes basada en los parámetros del modelo
        self.agents = ap.AgentList(self, self.p.agents['BDI'], BDIAgent) + \
                      ap.AgentList(self, self.p.agents['RiskTaker'],␣
↪RiskTakingAgent) + \
```

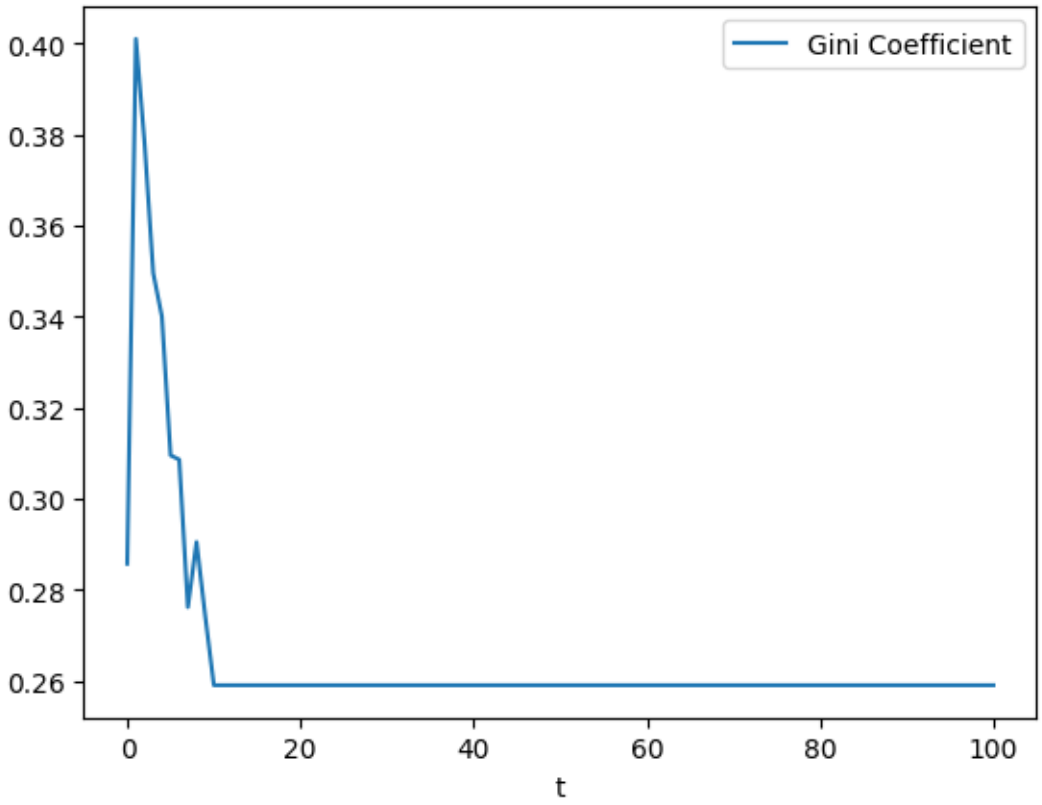
```
                    ap.AgentList(self, self.p.agents['DeductiveReasoning'],␣
↪DeductiveReasoningAgent)

    def step(self):
        # Cada agente realiza una transferencia de riqueza
        self.agents.wealth_transfer()

    def update(self):
        # Calcular el Coeficiente de Gini para la distribución de riqueza actual
        wealths = [agent.wealth for agent in self.agents]
        self.record('Gini Coefficient', gini(wealths))

    def end(self):
        # Registrar la riqueza final de cada agente
        self.agents.record('wealth')
```

[7]:
```
# Parámetros del Modelo
parameters = {
    'agents': {
        'BDI': 10,
        'RiskTaker': 10,
        'DeductiveReasoning': 10
    },
    'steps': 100,
    'seed': 42,
}
```

[8]:
```
# Correr el Modelo
model = WealthModel(parameters)
results = model.run()
```

```
Completed: 100 steps
Run time: 0:00:00.018926
Simulation finished
```

[9]:
```
# Visualización de Resultados
data = results.variables.WealthModel
ax = data.plot()
```



Evolución de la Desigualdad Económica en el Modelo de Distribución de Riqueza Multi-Agente

Este gráfico muestra cómo cambia el coeficiente de Gini, que mide la desigualdad de riqueza, a lo largo de 100 pasos de simulación.

Observaciones clave: - Al principio, hay mucha volatilidad con una desigualdad máxima de 0.40 en los primeros pasos. - Después del paso 20, la desigualdad se estabiliza rápidamente alrededor de 0.26. - La estabilización rápida sugiere que el sistema alcanza un equilibrio. - El coeficiente de Gini final bajo indica una distribución de la riqueza relativamente equitativa.

Esto significa que las estrategias de interacción de los agentes llevan rápidamente a un patrón de distribución de riqueza estable, a pesar de las diferentes condiciones iniciales y modelos de comportamiento.

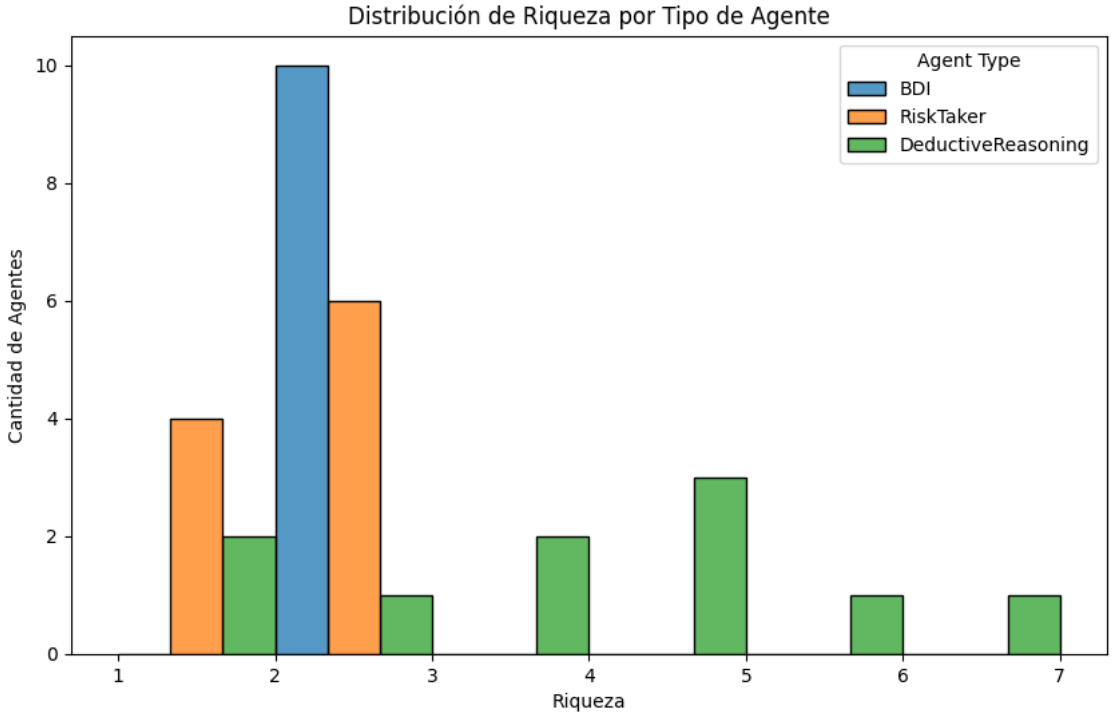
Parámetros: - 30 agentes en total (10 de cada tipo) - 100 pasos de simulación

Significado: - Demuestra la naturaleza autorreguladora del sistema en la distribución de la riqueza.

[10]:
```
# Extraer datos de riqueza y tipo de agente después de la simulación
agent_data = [(agent.strategy_name, agent.wealth) for agent in model.agents]
df = pd.DataFrame(agent_data, columns=["Agent Type", "Wealth"])
```

[11]:
```
# Crear histograma de la distribución de riqueza por tipo de agente
plt.figure(figsize=(10, 6))
```

```
sns.histplot(data=df, x="Wealth", hue="Agent Type", multiple="dodge",␣
 ↪binwidth=1)
plt.title("Distribución de Riqueza por Tipo de Agente")
plt.xlabel("Riqueza")
plt.ylabel("Cantidad de Agentes")
plt.show()
```



Comparación de Estrategias de Agentes en la Acumulación de Riqueza

Este histograma muestra cómo se distribuye la riqueza al final entre tres tipos de agentes (BDI, Tomadores de Riesgos y Razonamiento Deductivo), revelando cómo les fue económicamente y qué tan efectivas fueron sus estrategias.

Puntos clave: - Agentes BDI (Azul): * La mayoría tiene entre 2-3 unidades de riqueza * Manejan su dinero de manera consistente y conservadora * La mayoría mantiene una riqueza estable en un rango medio

- Tomadores de Riesgos (Naranja):
  - Tienen entre 1-3 unidades de riqueza
  - Hay más variación en sus resultados
  - Esta estrategia no lleva a acumular mucha riqueza
- Agentes de Razonamiento Deductivo (Verde):
  - Tienen un rango de riqueza más amplio (1-7 unidades)
  - Son los únicos que alcanzan los niveles más altos de riqueza
  - Son los más exitosos en acumular riqueza

– Muestran una mayor adaptabilidad en sus estrategias