AI Investigation: Using "Prisoners Dilemma"to experiment with using finding Equilibrium of simple game theory games

AI Investigation Part 5

Goal: test basic theory of using simulation to determine equilibrium of game theory games

Also experiments a new way to present the winloss function

Creator: `Lang (Ron) Chen 2022`

In [1]:

```
import random
import numpy as np
import statistics as s
import scipy.stats
from collections import defaultdict
```

Define Game

If both confesses, then each get 10 years in prison;

If A confesses and B does not, then A gets 1 year and B gets 25 years;

Vice versa if B confesses and A does not;

If both do not confess then they each get 3 years;

In [2]:

```
CHOICE = {'Confess': 0, 'Not Confess':1}
OUTCOME = [[-10, -1],
           [-25, -3]]
```

This is an experiment on upgrading the winloss function from using a host of if-else to a more lightweight structure. However this wouldn't work as well for games with more than two players, or if the CHOICEs are not discrete.

Also if two players have different payoffs then need two different OUTCOMEs, but this would still be more concise than writing two distinct winloss functions with massive if-else overheads

In [3]:

```
def winloss(player1, player2):
    return OUTCOME[CHOICE[player1]][CHOICE[player2]]
```

In [4]:

```
def validation(choice, CHOICE):
    if "Inconclusive" in choice:
        return 'Inconclusive'
    if choice not in CHOICE:
        return False
    return True
```

Simulation

In [5]:
```
RUNS = 1000000

data1 = defaultdict(list)
data2 = defaultdict(list) # Using defaultdict makes the code more adaptable to different g

sample = list()
victory = list()

choicekeys = list(CHOICE.keys())
for i in range(RUNS):
    obs1 = random.sample(choicekeys, 1)[0]
    obs2 = random.sample(choicekeys, 1)[0]

    data1[obs1].append(winloss(obs1, obs2))
    data2[obs2].append(winloss(obs2, obs1))

# This time we record the data for both players because we are interested in each's domina
# i.e. we are not as interested in, or equally interested in each's dominant strategy as w
```

Algorithm for final choice

First: Manipulate data so that it is in a dictionary and the dictionary value is [mean, stdev, length]

In [6]:
```
stats1 = dict()
for choice in CHOICE:
    tmp = list()
    tmp.append(s.mean(data1[choice]))
    tmp.append(s.stdev(data1[choice]))
    tmp.append(len(data1[choice]))

    stats1[choice] = tmp

stats1
```

Out[6]:
```
{'Confess': [-5.509312564171198, 4.499994858484077, 500614],
 'Not Confess': [-14.01057298362389, 11.000005932262946, 499386]}
```

In [7]:
```
stats2 = dict()
for choice in CHOICE:
    tmp = list()
    tmp.append(s.mean(data2[choice]))
    tmp.append(s.stdev(data2[choice]))
    tmp.append(len(data2[choice]))

    stats2[choice] = tmp

stats2
```

Out[7]:
```
{'Confess': [-5.5080158479744705, 4.49999735386907, 500758],
 'Not Confess': [-14.007403223286502, 11.000008525455318, 499242]}
```

In [8]:
```
def final_choice(stats):

    stats.sort(key = lambda x:x[1][0], reverse = True)

    # Then tests whether the second, third and so forth values contain 0 within their joi
    inconclusive_list = [stats[0][0]]
    inconclusive = False
    for i in range(1, len(stats)):
```

```python
        if in_range(construct_CI(stats[0], stats[i])):
            inconclusive_list.append(stats[i][0])
            inconclusive = True
        else: # Because all values are sorted, if the current choice's joint Confidence In
            break

    if inconclusive: # If inconclusive, return statement with the list of 'drawed' choices
        return f'Inconclusive: the following came to a draw {inconclusive_list}'

    return stats[0][0] # Else, return the dominant strategy
```

In [9]:

```python
def construct_CI(stat1, stat2):
    """ Uses Welch's approximation to construct a joint CI of two means, unknown populatio

    xbar1 = stat1[1][0]
    xbar2 = stat2[1][0]
    s1 = stat1[1][1]
    s2 = stat2[1][1]
    n = stat1[1][2]
    m = stat2[1][2]

    r = (s1**2 /n + s2**2 /m)**2 /(s1**4 /(n**2 * (n-1)) + s2**4 /(m**2 * (m-1)))

    q = scipy.stats.t.ppf(0.95, df = r)

    poolsd = np.sqrt(s1**2 /n + s2**2 /m)

    CI = (xbar1 - xbar2 - q * poolsd, xbar1 - xbar2 + q * poolsd)

    print(f'{stat1[0]} {stat2[0]}: {CI}')
    print('\n')

    return CI
```

In [10]:

```python
def in_range(CI):
    """ Helper function to check whether 0 is within the Confidence Interval """

    if CI[0] <= 0 and CI[1] >= 0:
        return True
    return False
```

In [11]:

```python
stats1
```

Out[11]:

```
{'Confess': [-5.509312564171198, 4.499994858484077, 500614],
 'Not Confess': [-14.01057298362389, 11.000005932262946, 499386]}
```

In [12]:

```python
result1 = final_choice(list(stats1.items()))
result1
```

```
Confess Not Confess: (8.473601980530878, 8.528918858374505)
```

Out[12]:

```
'Confess'
```

In [13]:

```python
stats2
```

Out[13]:

```
{'Confess': [-5.5080158479744705, 4.49999735386907, 500758],
 'Not Confess': [-14.007403223286502, 11.000008525455318, 499242]}
```

In [14]:
```
result2 = final_choice(list(stats2.items()))
result2
```

```
Confess Not Confess: (8.47172607945118, 8.527048671172885)
```

Out[14]:
```
'Confess'
```

Validation

In [15]:
```
validation(result1, CHOICE)
```

Out[15]:
```
True
```

In [16]:
```
validation(result2, CHOICE)
```

Out[16]:
```
True
```

Emperical Testing

In [17]:
```
victory1 = list()
victory2 = list()
for i in range(RUNS):
    victory1.append(winloss(result1, result2))
    victory2.append(winloss(result2, result1))

Equilibrium = (s.mean(victory1), s.mean(victory2))
Equilibrium
```

Out[17]:
```
(-10, -10)
```

Thus, the equilibrium of this game is (-10, -10), where both convicts' dominant strategy is to confess