MAS_2

November 13, 2020

Leonard Vorbeck

No Group

Student Number : 2709813

0.1 2.1.1

$C_c = 10$

$C_e = 20$

$U_c = 12$

$U_e = 18$

For example $U_i(C, C) \leftarrow U_c - \frac{C_c}{2} = 2$ and so on..

[12]:
```
A = ["C","E"]
pd.DataFrame([[(2,2),(-3,3)],[(3,-3),(-2,-2)]], columns=A, index=A)
```

[12]:
```
          C         E
C   (2, 2)   (-3, 3)
E  (3, -3)  (-2, -2)
```

0.2 2.1.2

E strictly dominates C for both agents, since $EU_i(E) > EU_i(C)$ for all agents, regardless of the counteragents action, which means the game always arrives at (E, E).

0.3 2.1.3

Introducing s changes the (joint)-payoff matrix of the game in the following way:

[13]:
```
pd.DataFrame([[(2,"2 + s"),("-3 + s",3)],[("3 + s",-3),(-2,"-2 + s")]],␣
↪columns=A, index=A)
```

[13]:
```
              C             E
C   (2, 2 + s)   (-3 + s, 3)
E  (3 + s, -3)  (-2, -2 + s)
```

Note : Alice is the column player, Bob is the row player. I will continue calling the row player (Bob) p_1 and the column player (Alice) p_2.

Now for this case $s \in (0, 2]$ determines the possible equilibria of the game. Within the possible value range of s, we can identify some thresholds that determine the games outcome. First, I inspect the problem visually.

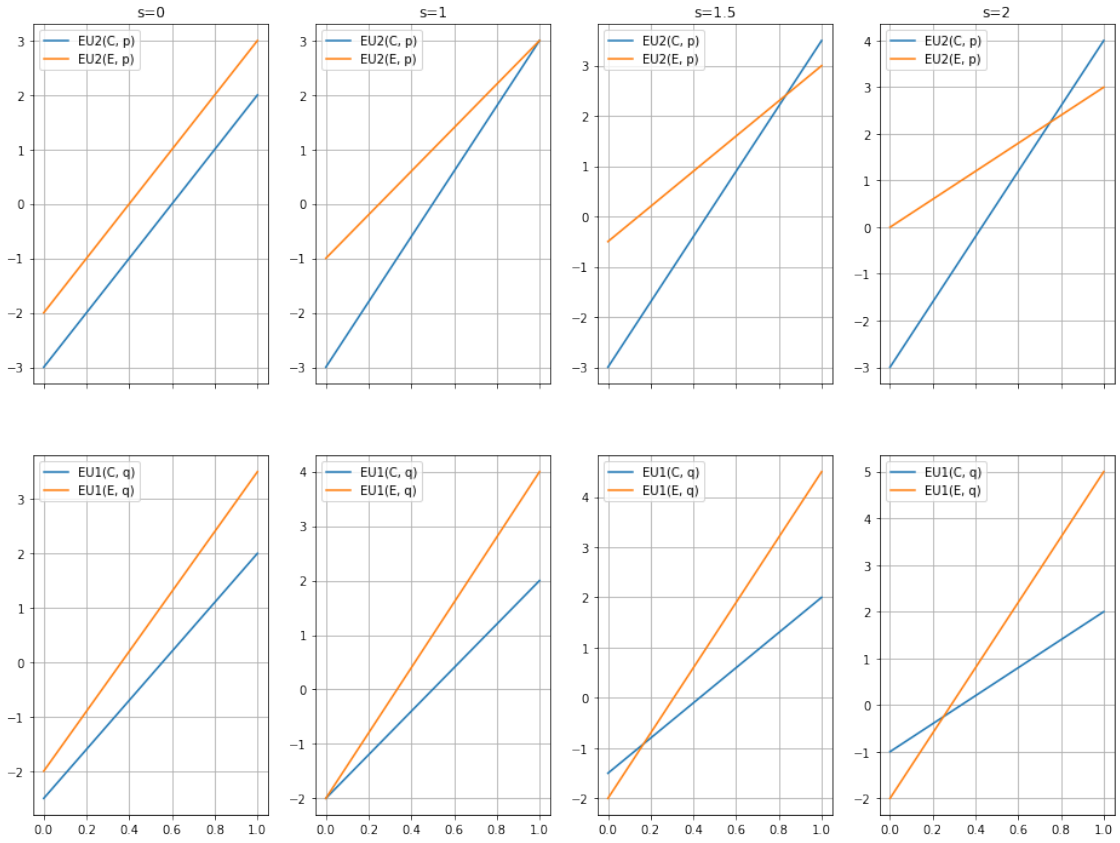
[91]:
```
def EU2_C(p, s) : return 5*p + s*p -3
def EU2_E(p, s) : return 5*p -2 + s -s*p

def EU1_C(q, s) : return 5*q -3 + s -s*q
def EU1_E(q, s) : return 5*q + s*q -2
```

[95]:
```
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(16,12), sharex=True)
s=0
pd.DataFrame({"EU2(C, p)" : [EU2_C(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.]],
                  "EU2(E, p)" : [EU2_E(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.
 ↪]]},
              index=[0, 0.25, 0.5, 0.75, 1.]).plot(grid=True, title="s=%s" % s,␣
 ↪ax=axes[0,0])
s=1
pd.DataFrame({"EU2(C, p)" : [EU2_C(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.]],
                  "EU2(E, p)" : [EU2_E(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.
 ↪]]},
              index=[0, 0.25, 0.5, 0.75, 1.]).plot(grid=True, title="s=%s" % s,␣
 ↪ax=axes[0,1])
s=1.5
pd.DataFrame({"EU2(C, p)" : [EU2_C(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.]],
                  "EU2(E, p)" : [EU2_E(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.
 ↪]]},
              index=[0, 0.25, 0.5, 0.75, 1.]).plot(grid=True, title="s=%s" % s,␣
 ↪ax=axes[0,2])
s=2
pd.DataFrame({"EU2(C, p)" : [EU2_C(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.]],
                  "EU2(E, p)" : [EU2_E(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.
 ↪]]},
              index=[0, 0.25, 0.5, 0.75, 1.]).plot(grid=True, title="s=%s" % s,␣
 ↪ax=axes[0,3])
s=0.5
pd.DataFrame({"EU1(C, q)" : [EU1_C(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.]],
                  "EU1(E, q)" : [EU1_E(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.
 ↪]]},
              index=[0, 0.25, 0.5, 0.75, 1.]).plot(grid=True,ax=axes[1,0])
s=1
pd.DataFrame({"EU1(C, q)" : [EU1_C(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.]],
                  "EU1(E, q)" : [EU1_E(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.
 ↪]]},
```

```
                index=[0, 0.25, 0.5, 0.75, 1.]).plot(grid=True,ax=axes[1,1])
s=1.5
pd.DataFrame({"EU1(C, q)" : [EU1_C(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.]],
                    "EU1(E, q)" : [EU1_E(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.
 ↪]]},
                index=[0, 0.25, 0.5, 0.75, 1.]).plot(grid=True,ax=axes[1,2])
s=2
pd.DataFrame({"EU1(C, q)" : [EU1_C(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.]],
                    "EU1(E, q)" : [EU1_E(p, s=s) for p in [0, 0.25, 0.5, 0.75, 1.
 ↪]]},
                index=[0, 0.25, 0.5, 0.75, 1.]).plot(grid=True,ax=axes[1,3])
```

[95]: <AxesSubplot:>



First of all, it is obvious that if $s < 1$, E remains a strictly dominant strategy $(\forall p, q)$ for both players, leading to (E, E) again.

Next, we can have a look at $s > 1$. If that is the case, there are no pure dominant strategies anymore. However, if p_1 wants to make p_2 indifferent, the following has to hold :

$$EU_2(C, p) = EU_2(E, p)$$

which yields

$$p(s) = \frac{1+s}{2s}$$

Equivalently

$$EU_1(C, q) = EU_1(E, q)$$

leads to

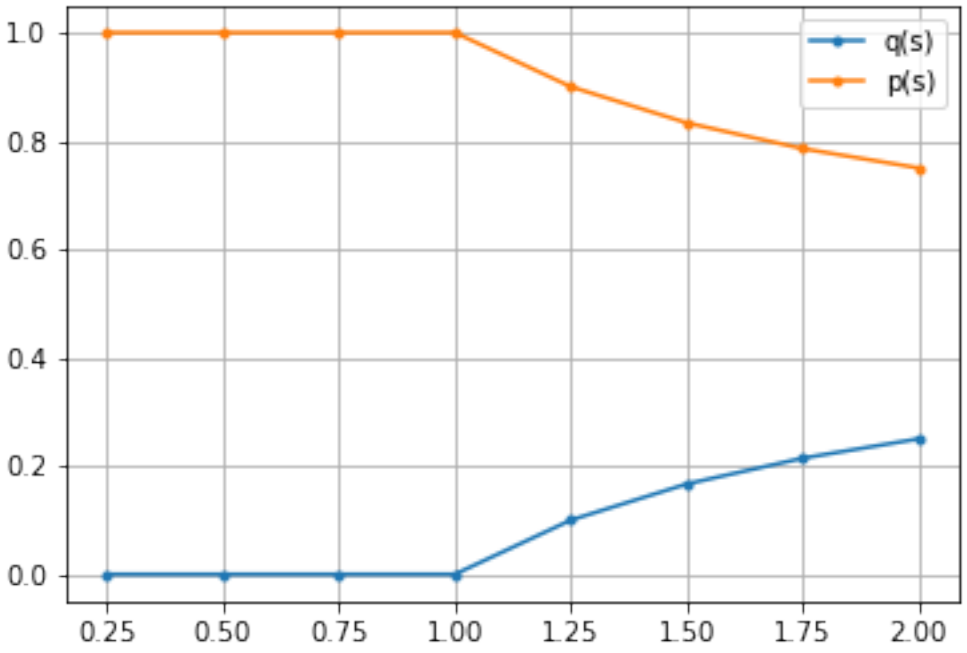
$$q(s) = \frac{s-1}{2s}$$

[120]:
```
def q(s) :
    res = (s-1)/(2*s)
    if res > 1 :
        return 1
    elif res < 0:
        return 0
    else:
        return res

def p(s) :
    res = (1+s)/(2*s)
    if res > 1 :
        return 1
    elif res < 0:
        return 0
    else:
        return res
```

[122]:
```
pd.DataFrame({"q(s)" : [ q(s)  for s in [0.25, 0.5, 0.75, 1., 1.25 ,1.5, 1.75,␣
↪2]],
                  "p(s)" : [p(s) for s in [0.25, 0.5, 0.75, 1., 1.25 ,1.5, 1.
↪75, 2]]},
             index=[0.25, 0.5, 0.75, 1., 1.25 ,1.5, 1.75, 2]).
↪plot(grid=True,style=".-",figsize=(6,4.1))
```

[122]: <AxesSubplot:>



For any $s > 1$, we will obtain mixed equilibria depending on s, because each player mixes with $p(s)$ ($q(s)$) **iff** $s > 1$.

More specifically, we can derive an interesting relationship between p and q. Using $p(s)$ and $q(s)$ (solve both for s and make them equal) we can derive

$$p = 1 - q$$

which is exactly what we see in the figure above. This is expected given the nature of the problem (coordination vs anti-coordination).

0.3.1 2.2.1

[137]:
```
A = ["H","D"]
pd.DataFrame([[("0.5*v-c", "0.5*v-c"),("v",0)],[(0,"v"),("0.5*v","0.5*v")]],␣
↪columns=A, index=A)
```

[137]:

	H	D
H	(0.5*v-c, 0.5*v-c)	(v, 0)
D	(0, v)	(0.5*v, 0.5*v)

0.3.2 2.2.2

$v > 0$

$v \geq c$

Obviously if $\frac{v}{2} - c > 0$ then $c < \frac{v}{2}$. In this case, H becomes a dominant strategy with game equilibrium (H, H).

If $\frac{v}{2} - c < 0$ then $c > \frac{v}{2}$. In this case two pure NEs will emerge at (H, D) and (D, H), indicating another mixed NE.

$$EU_2(H, p) = EU_2(D, p)$$

gives

$$p(v, c) = \frac{v}{2c}$$

Equivalently

$$EU_1(H, q) = EU_1(D, q)$$

leads to

$$q(v, c) = \frac{v}{2c}$$

which gives us the mixed NE $(\frac{v}{2c}, \frac{v}{2c})$, which also solves the special case in which $c = \frac{v}{2}$, resulting in (H, H) again.

0.3.3 2.3

$b_i(r_i, r_j) = 10 - r_i + \frac{r_j}{2}$.

$u_i(r_i, r_j) = b_i \times r_i - 4 \times r_i$.

0.3.4 2.3.1

F.O.C for BR_i

$$\frac{d\ u_i}{dr_i}(r_j) = 0$$

yields

$$BR_i(r_j) \leftarrow r_i^*(r_j) = 3 + \frac{r_j}{4}$$

we can now use

$$r_i^*(r_j) \leftarrow r_j(r_i^*)$$

to solve

$$r_j(r_i^*) = r_j^*(r_i)$$

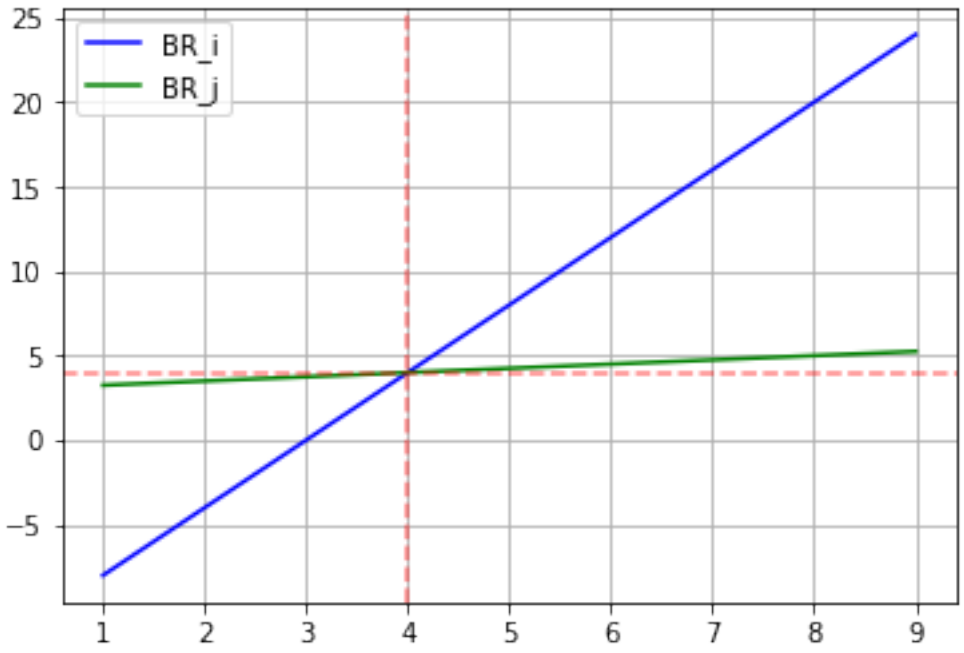
which leads to equilibrium effort of $q^{NE} = 4$.

0.3.5 2.3.2

[171]:
```
def BR_i(ri) : return 4*ri - 12
def BR_j(ri) : return 3 + ri/4
def BR_j2(ri) : return 0.5 + ri/4
```

[202]:
```
pd.DataFrame({"BR_i" : [ BR_i(_) for _ in range(1,10)],
              "BR_j" : [ BR_j(_) for _ in range(1,10)]},
             index=list(range(1,10))).plot(grid=True,style="-",figsize=(6,4.
 ↪1),ms=10,color=["blue", "green"])
plt.axvline(x=4, c="red",ls="--", alpha=.49)
plt.axhline(y=4, c="red",ls="--", alpha=.49)
```

[202]:
```
<matplotlib.lines.Line2D at 0x1239d9fd0>
```

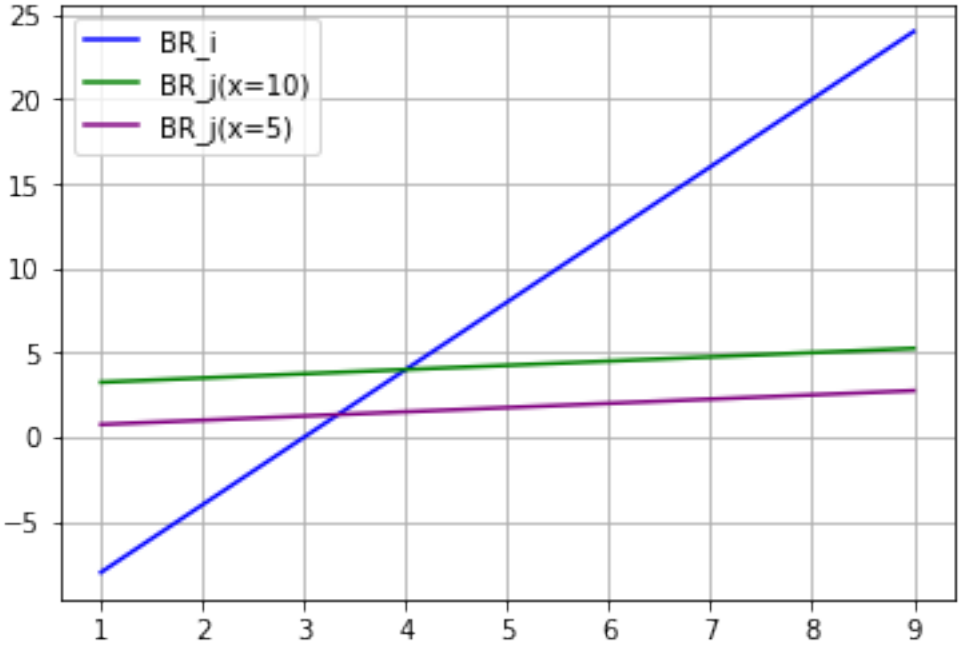


0.3.6 2.3.3

In the general form with $b_i(r_i, r_j) = x - r_i + \frac{r_j}{2}$ the F.O.C ensures that $BR_i = \frac{x-4}{2} + \frac{r_j}{2}$ s.t the intercept of the best response function is decreasing if x decreases, where x emphasises on the utlity a country gets from its own effort r.

[204]:
```
pd.DataFrame({"BR_i" : [ BR_i(_) for _ in range(1,10)],
              "BR_j(x=10)" : [ BR_j(_) for _ in range(1,10)],
             "BR_j(x=5)" : [ BR_j2(_) for _ in range(1,10)]},
             index=list(range(1,10))).plot(grid=True,style="-",figsize=(6,4.
 ↪1),ms=10,color=["blue", "green", "purple"])
```

[204]: <AxesSubplot:>



The new eqilibrium is smaller both in r_j and r_i and therefore socially inferior to the first one.

0.3.7 2.4.1

For $n = 2$, we have

$$U_1(x_1, x_2) = x_1(1 - x_1 - x_2)$$

$$U_1(x_1, x_2) = x_1 - x_1^2 - x_2 x_1$$

To get the best response, we have to satisfy F.O.C :

$$\frac{dU_1}{dx_1} = 0$$

which yields

$$x_1^*(x_2) = \frac{1}{2} - \frac{1}{2}x_2$$

Equivalently for p_2

$$x_2^*(x_1) = \frac{1}{2} - \frac{1}{2}x_1$$

Now let

$$x_2^*(x_1) \leftarrow x_1(x_2^*) = 1 - 2x_2^*$$

solve

$$x_1(x_2^*) = x_1^*(x_2)$$

in order to get

$$x^{NE} = \frac{1}{3}$$

0.3.8 2.4.2

The shape of U and the fact $U_i \leftarrow U \ \forall \ i$ guarantees non-optimal solutions if naive maximization (f.o.c) is applied.

For example :

$$2U(\frac{1}{3}, \frac{1}{3}) < U(\frac{1}{4}, \frac{1}{3}) + U(\frac{1}{3}, \frac{1}{4})$$

In order to get the social optimum, maximize W manually or its also possible to reach the optimum by introducing either additional environmental variables (Why are there no opportunity costs in this environment?!) or by creating more flexible utlity functions.

0.3.9 2.4.3

For the n player case we have

$$U_i(\vec{X}) = x_i(1 - \sum_{j=1}^{n} x_j) = x_i - x_i^2 - \sum_{j \neq i}^{n} x_j x_i$$

with

$$\frac{dU_1}{dx_1} = 1 - 2x_i - \sum_{j \neq i}^{n} x_j$$

F.O.C yields

$$x_i^*(x_{-i}) = \frac{1}{2} - \sum_{j \neq i}^{n} \frac{x_j}{2}$$

Trivially to derive the symmetric solution vector $\vec{X}^*$, we can assume $\vec{X}^* = [x_1^*, x_2^*, ..., x_n^*]$, s.t.

$$x_i^{NE} = x_i^*(\vec{X}^*) = \frac{1}{2} - \sum_{j \neq i}^{n} \frac{x_j^*}{2}$$

which is equivalent to

$$x_i^{NE} = x_i^*(\vec{X}^*) = \frac{1}{2} - \frac{(n-1)x_i^*}{2} = \frac{1}{1+n}$$

which is consistent with $n = 2$.

[ ]: