Creating a Visual Representation of Best Responses and Nash Equalibria in 2D - Toby Devlin, c1432293

This paper will look into ways of visually connecting methods of identifying best responses and understanding dominance in a 2 player game in the space $(A, B) \in \mathbb{R}^{2\times 2^2}$. We will be looking at a series of games that have certain characteristics to see how they differ in their best responses and visualisation. These characteristics include levels of dominant strategies and player rationality.

Using utility matrices $(A, B) \in \mathbb{R}^{2\times 2^2}$ we represent the strategy space of the row player as $\sigma_r = (x, 1-x)$ and $\sigma_c = (y, 1-y)$ for the column player. If we try to understand the game from the row players perspective, his score is dictated by the variables x, y and A; one of which is in our control. If he knows that the column player will have mixed strategy in the form $\sigma_c = (y, 1-y)$ how can he estimate the expected outcome utility of our strategy $(x, 1-x)$? Using the utility equation for the row player we can see:

$$u_r(\sigma_r, \sigma_c) = \sigma_r A \sigma_c^T = \begin{pmatrix} x & 1-x \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} y \\ 1-y \end{pmatrix} = xy(a_{11}-a_{12}-a_{21}+a_{22})+x(a_{12}-a_{22})+y(a_{21}-a_{22})+a_{22} = f(x,y,A)$$

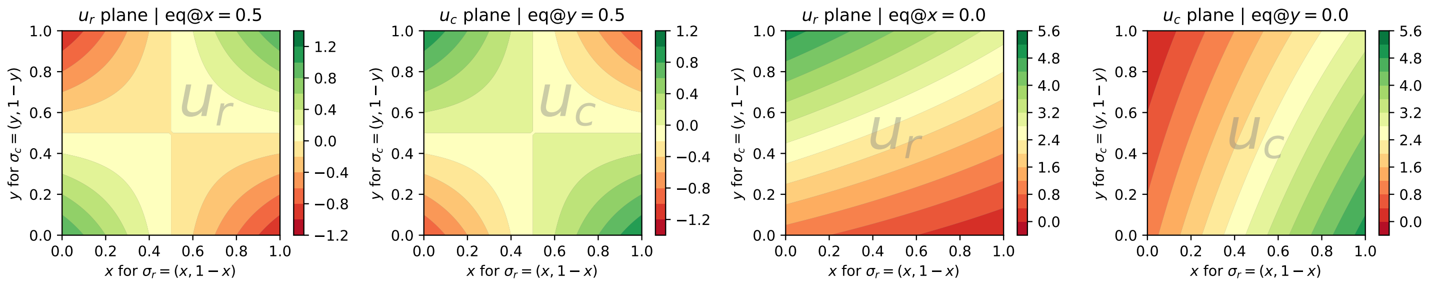
This is just a function of the two mixed strategy variables x and y and can be extended to calculate u_c by using the column players utility matrix B. The function below calculates this utility for any matrix M and any given x, y. We can leverage `numpy.meshgrid()` to evaluate this utility across a plane of $x, y \in [0, 1]$.

```
def get_utility_plane(x,y,M):
    '''Generates the utility plane using sympy and labdafy evaluation'''
    a, b = sym.symbols('a, b')
    A = sym.Matrix(M)
    # Define mixed strategies
    sigma_r, sigma_c = sym.Matrix([[a, 1-a]]), sym.Matrix([b, 1-b])
    # Use sympy to evaluate and lambdafy
    result = (sigma_r * A *  sigma_c)[0]
    return sym.lambdify((a,b), result)(x,y)
```

Using this method we plot the plane for the games, Matching Pennies(MP) & The Prisoners Dilemma(PD) below using `matplotlib.pyplot.contourf()`; the full plotting method is shown in the appendix.

$$\text{MP: } A = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \quad B = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \qquad | \qquad \text{PD: } A = \begin{pmatrix} 3 & 0 \\ 5 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 3 & 5 \\ 0 & 1 \end{pmatrix}$$

Here we see a filled contour plot of what the opponents are doing and the respective play off utility for a single turn of moves.



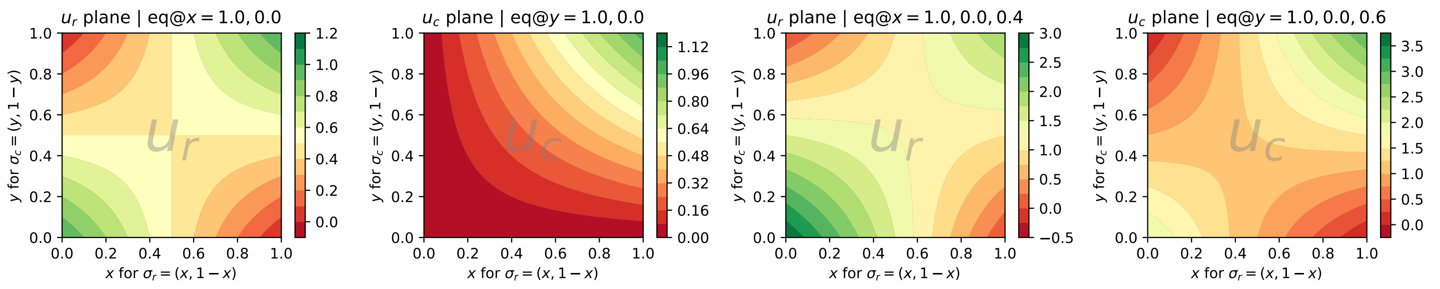
The axis represents how players can form a strategy space but only control 1 axis; the row player controls x and the column player controls y. For example, in the MP game (left pair) we can see that its best for players to play in the corners, however, the best utilities for players lie in opposite corners in the 2 utility planes; if a player was to try and play a corner (e.g $y = 0$) the other player could exploit this ($x = 1$). This is expected for zero-sum games because for one player to win in this type of game the other must lose.

The equilibria here can be seen to be $x = .5$, $y = .5$, as `nashpy` calculated, by observing the utility plane and asking, for both players, 'if I move can I score better?'. The answer is no. Consider moving y to 0, then x could also move to 0 and get an expected utility of 1 leaving $u_c = -1$.

The second game is The Prisoners Dilemma, a strongly dominated game. The Nash equilibria is less clear but by analysing the 2 utility matrices, and asking the question again, we can see that $x = 0$, $y = 0$ does make sense. The second strategy for both matrices is strongly dominant, i.e. playing anything other than $x = 0$ and $y = 0$ is irrational.

The process of asking 'if I set my variable differently can I score better?' in turn for each player will eventually end up with finding an equilibria. What were actually doing here is support enumeration, checking if positions across 2 positions are best responses to each other, this is more clear when we add more equilibria points to the examples. Below are made up games that have multiple equilibria points (the best responses are underlined):

$$\text{LEFT: } A = \begin{pmatrix} \underline{1} & 0 \\ 0 & \underline{1} \end{pmatrix} \quad B = \begin{pmatrix} \underline{1} & 0 \\ \underline{0} & \underline{0} \end{pmatrix} \quad | \quad \text{RIGHT: } A = \begin{pmatrix} \underline{2} & 0 \\ 0 & \underline{3} \end{pmatrix} \quad B = \begin{pmatrix} \underline{3} & 0 \\ 0 & \underline{2} \end{pmatrix}$$



Lets think of the support enumeration algorithm, the first step is to identify best responses support size 1. The above matrices have matching best responses when the position in both A and B are underlined and thus an equilibria. In the left game there are 2, at $\sigma_r = (1,0)$, $\sigma_c = (1,0)$ and $\sigma_r = (0,1)$, $\sigma_c = (0,1)$. This also holds for the right game too. Looking at the plane this is clear because both players will benefit from these positions and have no reason to move.

What is more difficult to see is the equilibria $\sigma_r = (0.4, 0.6)$, $\sigma_c = (0.6, 0.4)$ of the right game. In the support enumeration algorithm this corresponds to the solving of best response mixed strategies of support size 2:

$$\sum_{i \in I} \sigma_{r_i} B_{ij} = v \, \forall j \in J \quad \text{or} \quad \begin{pmatrix} \sigma_{r_1} & \sigma_{r_2} \end{pmatrix} \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} v & v \end{pmatrix} \Rightarrow 3\sigma_{r_1} = 2\sigma_{r_2} \Rightarrow 3x = 2(1-x) \Rightarrow x = 2/5 = 0.4$$

and

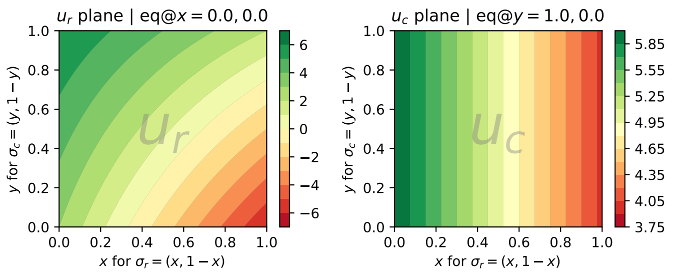
$$\sum_{j \in J} A_{ij} \sigma_{c_j}^T = u \, \forall i \in I \quad \text{or} \quad \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} \sigma_{c_1} \\ \sigma_{c_2} \end{pmatrix} = \begin{pmatrix} u \\ u \end{pmatrix} \Rightarrow 2\sigma_{c_1} = 3\sigma_{c_2} \Rightarrow 2y = 3(1-y) \Rightarrow y = 3/5 = 0.6$$

These equations represent providing an opponent with an indifferent choice of utility for their strategy. If, for both players, the calculated mixed strategy is the best response to their opponents calculated mixed strategy then a nash equilibria has been found. This result can be visually understood by identifying that the a players choice of mixed strategy sits on the inflection point of the utility plane of the their opponent. This means that each player must actively decide to play their opponents inflection point or be taken advantage of; i.e. they provide an indifferent choice of utility to their opponent while also being provided an indifferent utility choice.

The relationship between pairs of visual utility planes and the calculated best responses can easily describe an example. Imagine you, the row player, are busy at home with your new puppy and have σ_r = (ignore, pet) while the puppy has σ_c = (come to you, run away, play with a toy). This leads pay off matrices:

$$A = \begin{pmatrix} 2 & 2 & -6 \\ \underline{6} & \underline{6} & \underline{3} \end{pmatrix} \quad | \quad B = \begin{pmatrix} \underline{4} & 2 & \underline{4} \\ \underline{6} & -1 & \underline{6} \end{pmatrix}$$

Immediately it can be seen that this game reduces to a strongly dominant matrix A (always pet the puppy) vs the non dominant B (dogs love toys or cuddles). This leads to the outcome that the row player will always play his dominant strategy and the column player has no control over his own score. An interesting observation is that because of the strongly dominant nature of A the row player can play any $\sigma_c = (y, 0, 1-y) \quad \forall y \in [0,1]$. This leads us to an interesting outcome; there are infinite nash equilibria: $x = 0$, $y \in [0,1]$ the strategy $\sigma_r = (0,1)$ has any response by the column player as a best response, giving an example of a degenerate game.



```
>>> # SUPPORT ENUMERATION IN NASHPY
>>> A = np.array([[2, 2, -6], [6, 6, -3]])
>>> B = np.array([[4, 2, 4], [6, -1, 6]])
>>> g = nash.Game(A, B)
>>> for pairs in g.support_enumeration():
...     print('sigma_r:', pairs[0], '\t sigma_c:', pairs[1])

sigma_r: [0. 1.]          sigma_c: [1. 0. 0.]
sigma_r: [0. 1.]          sigma_c: [0. 0. 1.]
```

APPENDIX

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import nash
import sympy as sym
sym.init_printing()

def plot_matrix_pair(A, B, subplot_base=121, norm_max=1, norm_min=-1,
                     colour='grey', plot_poly=True, cfill=False,
                     save_str="", offset_watermark=False):
    '''Plots a game A vs B to the screen. Polytope Vertacies: 0:cyan, 1:magenta, 2:black, 3:blue'''
    # Create [0,1]^2 plane:
    x_range, y_range = [np.arange(0, 1.1, 0.01)] * 2
    x_grid, y_grid = np.meshgrid(x_range, y_range)
    polytope_eqns = get_polytope_vertex_eqns(A,B)
    game = nash.Game(A, B)
    eqlbria = list(game.support_enumeration())
    num_eqs = len(eqlbria)

    # Build the Data for row player:
    row_Z = get_utility_plane(x_grid, y_grid, A)
    # Build the Data for col player:
    col_Z = get_utility_plane(x_grid, y_grid, B)
    #PLOTS
    plt.rcParams['axes.facecolor'] = colour
    #Row: P = {x \in R | x>=0, xB<=1}
    plt.subplot(subplot_base)
    if cfill:
        plt.contourf(x_grid, y_grid, row_Z, 15, cmap='RdYlGn') # Norm is for the coloured lines
    else:
        plt.contour(x_grid, y_grid, row_Z, cmap='RdYlGn', norm=mpl.colors.Normalize(vmin=norm_min, vmax=norm_max)) # Norm is for the coloured lines
    if plot_poly:
        plt.plot(x_range,[1,1]+[0]*(len(x_range)-2),'c-',x_range,[0.01]*len(x_range),'m-')
        plt.plot(x_range,polytope_eqns[2](x_range),'k-',x_range,polytope_eqns[3](x_range),'b-')
    plt.colorbar()
    eqs = ",".join(["{:.1f}".format(eqlbria[n][0][0]) for n in range(num_eqs)])
    plt.title("$u_r$ plane | eq@$x="+eqs+"$")
    plt.xlabel("$x$ for $\sigma_r=(x,1-x)$")
    plt.ylabel("$y$ for $\sigma_c=(y,1-y)$")
    plt.xlim(0,1)
    plt.ylim(0,1)
    if offset_watermark:
        plt.text(0.66, 0.66,'$u_r$', horizontalalignment='center',verticalalignment='center', color='grey', fontsize=40, alpha=0.4)
    else:
        plt.text(0.5, 0.5,'$u_r$', horizontalalignment='center',verticalalignment='center', color='grey', fontsize=40, alpha=0.4)

    #Col: Q = {y \in R | Ay<=1, y>=0}
    plt.subplot(subplot_base+1)
    if cfill:
        plt.contourf(x_grid, y_grid, col_Z, 15, cmap='RdYlGn') # Norm is for the coloured lines
    else:
        plt.contour(x_grid, y_grid, col_Z, cmap='RdYlGn', norm=mpl.colors.Normalize(vmin=norm_min, vmax=norm_max)) # Norm is for the coloured lines
    if plot_poly:
        plt.plot(x_range,[1,1]+[0]*(len(x_range)-2),'k-',x_range,[0.01]*len(x_range),'b-')
        plt.plot(x_range,polytope_eqns[0](x_range),'c-',x_range,polytope_eqns[1](x_range),'m-')
    plt.colorbar()
    eqs = ",".join(["{:.1f}".format(eqlbria[n][1][0]) for n in range(num_eqs)])
    plt.title("$u_c$ plane | eq@$y="+eqs+"$")
    plt.xlabel("$x$ for $\sigma_r=(x,1-x)$")
    plt.ylabel("$y$ for $\sigma_c=(y,1-y)$")
    plt.xlim(0,1)
    plt.ylim(0,1)
    if offset_watermark:
        plt.text(0.66, 0.66,'$u_c$', horizontalalignment='center',verticalalignment='center', color='grey', fontsize=40, alpha=0.4)
    else:
        plt.text(0.5, 0.5,'$u_c$', horizontalalignment='center',verticalalignment='center', color='grey', fontsize=40, alpha=0.4)

    plt.tight_layout()
    if not save_str == "":
        plt.savefig("img/"+save_str)
```

Figure 1: Extended plotting code that was used to output figures.