Recursive Nash-in-Nash Bargaining Solution

Xiaowei Yu and Keith Waehrer

This code with user defined bargaining pairs and gross-profit functions will solve for the recursive Nash-in-Nash bargaining solution described in Yu and Waehrer (2023) “Recursive Nash-in-Nash bargaining solution” available https://waehrer.net/Yu_Waehrer_2023_RNnN.pdf. Please cite to our paper if you make use of this code or adapt the algorithm for use with a different language.

The default configuration of the code is set to solve for Example 3 in the paper, but these are easily changed to adapt the code to a bargaining problem of arbitrary structure. Besides the solution options described below, it is straightforward to change the set of bargaining pairs and the gross profit functions. Look below for headings that start with “User defined:”.

In footnote 14, two alternative definitions of recursive Nash-in-Nash are suggested. The allow for the endogenous inclusion of disagreements as a result of failures of the individual rationality condition. Suppose at a stage with bargaining pairs g’ that the component balance and fairness conditions would violate individual rationality for bargaining pairs f ⊆ g’, then the payoffs for that stage could be defined as

(1) $U_i^{g'} = U_i^{g'\setminus f}$

(2) $U_i^{g'} = (1/|f|)\sum_{j\in f} U_i^{g'\setminus j}$

To run the code without endogenizing disagreements, set the option variable below to 0 (zero). Otherwise, set to 1 or 2 depending on which of the two options is preferred.

```
In[ ]:= (* User defined: *) option = 0;
```

User defined: Bargaining pairs.

These should be based on a set of players that are numbered sequentially starting at 1.

```
In[ ]:= S = {{1, 3}, {1, 4}, {2, 3}, {2, 4}};

In[ ]:= (*Check that players are numbered sequentially*)
        If[Total[Union@Flatten[S]] ≠ Sum[i, {i, Max@Flatten[S]}], Beep[];
          Print[Style["***Not an appropriate set of bargaining pairs. Check that
               players are numbered sequentially starting at 1.***", 18, Red]]];
```

User defined: Gross profits.

The 1st argument is the index of the player and 2nd argument is a subset of S.

```
In[ ]:= gp[i_, s_] :=
         If[i < 3, -c[Count[s, {i, _}]], r[Count[s, {_, i}], (Length[s] - Count[s, {_, i}])]]

        (* Remove the commenting out of these commands to
         implement the structure applied to Example 3 in the appendix. *)

        (* c[_]=0;
        r[a_,b_]:=Max[3a-5/4 b,0]; *)
```

Define the bargaining pairs for each recursive bargaining problem (this is the set of subsets of S) and the set of players .

```
In[∘]:= SS = Subsets[S]; n = Union[Flatten[S]];
```

Define bargaining groups for each recursive stage

For each element of SS, the algorithm iteratively combines bargaining pairs into groups with non-empty intersection until the set of bargaining groups can no longer be combined.

```
In[∘]:= BG[SS_] := Module[{BGroups = {{}}, bgs, s, x, y, z, notz},
           mx[b_] := Table[If[IntersectingQ[b⟦j⟧, b⟦i⟧], 1, 0], {i, Length[b]}, {j, Length[b]}];
           Do[bgs = SS⟦i⟧; x = mx[bgs];
            While[x ≠ IdentityMatrix[Length[bgs]],
             y = PositionLargest[Table[Total[x⟦i⟧], {i, Length[bgs]}]];
             z = Flatten@Position[x⟦y⟦1⟧⟧, 1];
             notz = Flatten@Position[x⟦y⟦1⟧⟧, 0];
             bgs = Join[{Union[Flatten@bgs⟦z⟧]}, bgs⟦notz⟧];
             x = mx[bgs];
            ];
            BGroups = Append[BGroups, bgs],
            {i, 2, Length[SS]}]; BGroups];
        BGroups = BG[SS];
```

Define functions that solve for the stage associated with the i - th element of SS .

The function F uses recursively defined u[i, s], the payoff to player i at the stage with bargaining pairs s.

```
In[∘]:= gprofit[s_] := Table[gp[i, s], {i, Length[n]}];
        marginal[x_, s_] :=
          v[x⟦1⟧] - u[x⟦1⟧, DeleteCases[s, x]] == v[x⟦2⟧] - u[x⟦2⟧, DeleteCases[s, x]];
        eqs[s_] := Table[marginal[s⟦j⟧, s], {j, Length[s]}]; (* Fairness conditions*)
        u[y_, s_] := 0 /; FreeQ[Flatten[s], y];
        F[i_] := Module[{s, v0, v1},
          s = SS⟦i⟧;
          (* v0 and v1 define the component balance conditions *)
          v0 = Map[v[#] == 0 &, Complement[n, Union[Flatten[s]]]];
          v1 = Map[Total@Flatten@gprofit[s]⟦#⟧ == Total@Thread[v[#]] &, BGroups⟦i⟧];
          sol = Solve[Join[eqs[s], v0, v1], Array[v, Length[n]]];
           ir = Map[If[sol⟦1, All, 2⟧⟦#⟦1⟧⟧ ≥ u[#⟦1⟧, DeleteCases[s, #]], 0, 1] &, s];
          If[Total[ir] ≠ 0,
           x = Flatten@Position[ir, 1];
           Print["IR failure at stage ", i, " ir =", ir, " ", s];
           If[i == Length[SS], Print["DISAGREEMENT AT FINAL STAGE"]];
           If[option == 1, sol = Map[v[#] → u[#, Complement[s, s⟦x⟧]] &, n]];
           If[option == 2,
            sol = Table[v[j] → Mean[Map[u[j, Complement[s, {#}]] &, s⟦x⟧]], {j, Last[n]}]];
          ];
          Flatten@Values@sol
         ]
```

Recursively solve for payoff functions u[i, S]

If gross-profits are defined numerically, violations of the individual rationality condition will be reported. If the “option” variable is set to 1 or 2 and there are violations of the individual rationality condition, the calculation of the payoffs will be alternatively calculated as described above.

```
In[ ]:= Do[solution = F[i];
          Do[u[j, SS⟦i⟧] = solution⟦j⟧, {j, Length[n]}],
         {i, Length[SS]}]
```

Print solution

To see the solutions associated with recursive stages, replace S in the payoff function with the set of bargaining pairs for the desired stage, which should be an element of SS.

```
In[ ]:= Do[Print["u", i, " = ", Simplify@u[i, S]], {i, Length[n]}]
```

$$u1 = \frac{1}{6} (-2 c[1] - 2 c[2] - 2 r[1, 1] + r[2, 0] + 3 r[2, 2])$$

$$u2 = \frac{1}{6} (-2 c[1] - 2 c[2] - 2 r[1, 1] + r[2, 0] + 3 r[2, 2])$$

$$u3 = \frac{1}{6} (2 c[1] - 4 c[2] + 2 r[1, 1] - r[2, 0] + 3 r[2, 2])$$

$$u4 = \frac{1}{6} (2 c[1] - 4 c[2] + 2 r[1, 1] - r[2, 0] + 3 r[2, 2])$$