# Using Reduce to Compute Nash Equilibria

## Classroom Tools for Game Theory

**Sérgio O. Parreiras**

The Karush–Kuhn–Tucker equations (under suitable conditions) provide necessary and sufficient conditions for the solution of the problem of maximizing (minimizing) a concave (convex) function. This article corrects the program in [1], which computes the solution of Karush–Kuhn–Tucker equations. Our main goal, however, is to provide a program to compute the set of all Nash equilibria of a bimatrix game. The program works well for “small” games (i.e. 4×4 or smaller games); thus, in particular, it is suitable for constructing classroom examples and as an additional tool to empower students in classes using game theory.

## ■ Karush–Kuhn–Tucker Equations

The Karush–Kuhn–Tucker equations (under suitable assumptions) provide necessary and sufficient conditions for the solution of the problem of maximizing (minimizing) a concave (convex) function.

For an excellent reference, see the tutorial in [2]. Here we modify the code of [1] by correcting minor typos, simplifying, and letting the user specify restrictions on the exogenous parameters of the model.

```
KT[obj_, cons_, vars_, paramcons_: True] :=
Module[{stdcons, eqcons, ineqcons, lambdas, mus,
   eqs1, eqs2, eqs3},
  stdcons =
   cons /.{x_ ≥ y_ → y - x ≤ 0, x_ > y_ → y - x < 0,
     x_ == y_ → x - y == 0, x_ ≤ y_ → x - y ≤ 0};
  eqcons = Cases[stdcons, x_ == 0 → x];
  ineqcons = Cases[stdcons, x_ ≤ 0 → x];

  lambdas = Array[λ, Length[ineqcons]];
  mus = Array[μ, Length[eqcons]];
```

```
    eqs1 = D[(* Lagrangian: *)
      obj + mus.eqcons - lambdas.ineqcons == 0, {vars}];
    eqs2 = Thread[lambdas ≥ 0];
    eqs3 = Table[lambdas[[i]] ineqcons[[i]] == 0,
      {i, Length[ineqcons]}];

    Assuming[
     paramcons,
     Refine[Reduce[Join[eqs1, eqs2, eqs3, cons],
       Join[vars, lambdas, mus], Reals,
       Backsubstitution → True]]
    ]
   ]
```

## □ An Example from Consumer Choice

The inputs of `KT` are the objective function to be maximized, the list of constraints, and the list of choice variables. Here is an example from consumer choice theory: maximize a utility function, subject to a budget constraint.

```
KT[Log[x] + Log[y], {px * x + py * y ≤ income}, {x, y}]
```

$$\left(py < 0 \;\&\&\; px < 0 \;\&\&\; income > 0 \;\&\&\; x == \frac{income}{2\,px} \;\&\&\; y == \frac{income}{2\,py} \;\&\&\; \lambda[1] == \frac{2}{income}\right) \;||\; \left(py < 0 \;\&\&\; px > 0 \;\&\&\; income > 0 \;\&\&\; x == \frac{income}{2\,px} \;\&\&\; y == \frac{income}{2\,py} \;\&\&\; \lambda[1] == \frac{2}{income}\right) \;||\; \left(py > 0 \;\&\&\; px < 0 \;\&\&\; income > 0 \;\&\&\; x == \frac{income}{2\,px} \;\&\&\; y == \frac{income}{2\,py} \;\&\&\; \lambda[1] == \frac{2}{income}\right) \;||\; \left(py > 0 \;\&\&\; px > 0 \;\&\&\; income > 0 \;\&\&\; x == \frac{income}{2\,px} \;\&\&\; y == \frac{income}{2\,py} \;\&\&\; \lambda[1] == \frac{2}{income}\right)$$

Several of the solutions do not make economic sense, because they do not use the fact that the income, price of good $x$, and price of good $y$ are all positive. However, KT lets the user specify restrictions on the exogenous parameters of the model.

```
KT[Log[x] + Log[y], {px x + py y ≤ income}, {x, y},
 {income > 0, px > 0, py > 0}]
```

$$x == \frac{\text{income}}{2\,\text{px}} \;\&\&\; y == \frac{\text{income}}{2\,\text{py}} \;\&\&\; \lambda[1] == \frac{2}{\text{income}}$$

An important advantage of KT over other optimization functions (such as Maximize or Minimize) is that KT returns the value of the Kuhn–Tucker multipliers. These multipliers have an important economic interpretation: they are shadow prices for the constrained resources. In the above example, for instance, the value of $\lambda[1]$ is the "infinitesimal" increment in the utility function of the consumer that is generated when the budget constraint is relaxed by increasing the consumer's income by an "infinitesimal amount."

## ■ Computing Nash Equilibrium of Bimatrix Games

Nash equilibrium is the main solution concept in game theory. It is a crucial tool for economics and political science models. Essentially, a Nash equilibrium is a profile of strategies (one strategy for each player), such that if a player takes the choices of the others as given (i.e. as parameters), then the player's strategy must maximize his or her payoff.

The function Nash takes as input the payoff function of player 1, the payoff function of player 2, and the actions available to players 1 and 2. It returns the entire set of Nash equilibria.

```
Nash[obj1_, obj2_, vars1_, vars2_] := Module[
{n1, n2, lambdas1, lambdas2, eqs1, eqs2, eqs3, eqs4,
   eqs5, eqcons1, eqcons2},

  n1 = Length[vars1];
  n2 = Length[vars2];

  lambdas1 = Array[λ1, n1];
  lambdas2 = Array[λ2, n2];

  eqs1 = D[(* Lagrangian 1: *)
    obj1 + μ1 Sum[vars1[[k]], {k, n1}] + lambdas1.vars1 == 0,
    {vars1}];
  eqs2 = D[(* Lagrangian 2: *)
    obj2 + μ2 Sum[vars2[[k]], {k, n2}] + lambdas2.vars2 == 0,
    {vars2}];

  eqs3 = Join[
    Thread[lambdas1 ≥ 0],
    Thread[lambdas2 ≥ 0]
   ];
  eqs4 = Table[lambdas1[[i]] vars1[[i]] == 0, {i, n1}];
  eqs5 = Table[lambdas2[[i]] vars2[[i]] == 0, {i, n2}];

  eqcons1 = {Sum[vars1[[i]], {i, n1}] - 1 == 0};
  eqcons2 = {Sum[vars2[[i]], {i, n2}] - 1 == 0};

  Reduce[
    Join[eqs1, eqs2, eqs3, eqs4, eqs5, eqcons1, eqcons2,
     Thread[-vars1 ≤ 0], Thread[-vars2 ≤ 0]],
    Join[vars1, vars2, lambdas1, lambdas2, {μ1, μ2}],
    Reals, Backsubstitution → True
   ] /. Join[
    {μ1 == x_ → True, μ2 == x_ → True},
    Inner[Rule, Thread[lambdas1 == x_],
      {Table[True, {i, n1}]}ᵀ, List][[1]],
    Inner[Rule, Thread[lambdas2 == x_],
      {Table[True, {i, n2}]}ᵀ, List][[1]]
   ]
  ]
```

## □ A Colonel Blotto's Game

There are many versions of Colonel Blotto's game; this is a simple one taken from [3]. General A (row player) has three divisions to defend a city; she has to choose how many divisions to place at the north road and how many divisions at the south road. General B (column player) has two divisions to try to invade the city; he also has to choose how many divisions to be assigned to the north road and how many to the south road. If General A has at least as many divisions as General B at a given road, General A wins the battle there (defense is favored in the case of a tie). To win the game, however, A must defeat B on both battlefields. Thus, A has four possible strategies and B has three strategies. The table below summarizes the players' strategies and payoffs (victory = 1, defeat = 0 for the whole campaign). For example, in the first row and first column the entry is $1, 0$, which means A won and B lost; A chose three divisions for the north road and none for the south road; B chose two for the north and none for the south. Because $3 \geq 2$ and $0 \geq 0$, A won both battles.

| | | General B | | |
|---|---|---|---|---|
| | | (2, 0) | (1, 1) | (0, 2) |
| General A | (3, 0) | 1, 0 | 0, 1 | 0, 1 |
| | (2, 1) | 1, 0 | 1, 0 | 0, 1 |
| | (1, 2) | 0, 1 | 1, 0 | 1, 0 |
| | (0, 3) | 0, 1 | 0, 1 | 1, 0 |

▲ **Game 1.** Colonel Blotto.

A Nash equilibrium for this game is a probability distribution over strategies; use `P` for the probabilities chosen by General A and `Q` for the probabilities chosen by General B.

```
Module[
 {P, Q, u},
 P = Table[p[i], {i, 4}];
 Q = Table[q[i], {i, 3}];
 u = P.{{1, 0, 0}, {1, 1, 0}, {0, 1, 1}, {0, 0, 1}}.Q;
 Nash[u, 1 - u, P, Q]
]
```

$$\left(p[1] == 0\ \&\&\ p[2] == \frac{1}{2}\ \&\&\ 0 \le p[3] \le \frac{1}{2}\ \&\&\ p[4] == \frac{1}{2}(1 - 2\,p[3])\ \&\&\ q[1] == \frac{1}{2}\ \&\&\ q[2] == 0\ \&\&\ q[3] == \frac{1}{2}\right)\ ||$$

$$\left(0 < p[1] < \frac{1}{2}\ \&\&\ p[2] == \frac{1}{2}(1 - 2\,p[1])\ \&\&\ p[1] \le p[3] \le \frac{1}{2}\ \&\&\ p[4] == \frac{1}{2}(1 - 2\,p[3])\ \&\&\ q[1] == \frac{1}{2}\ \&\&\ q[2] == 0\ \&\&\ q[3] == \frac{1}{2}\right)\ ||$$

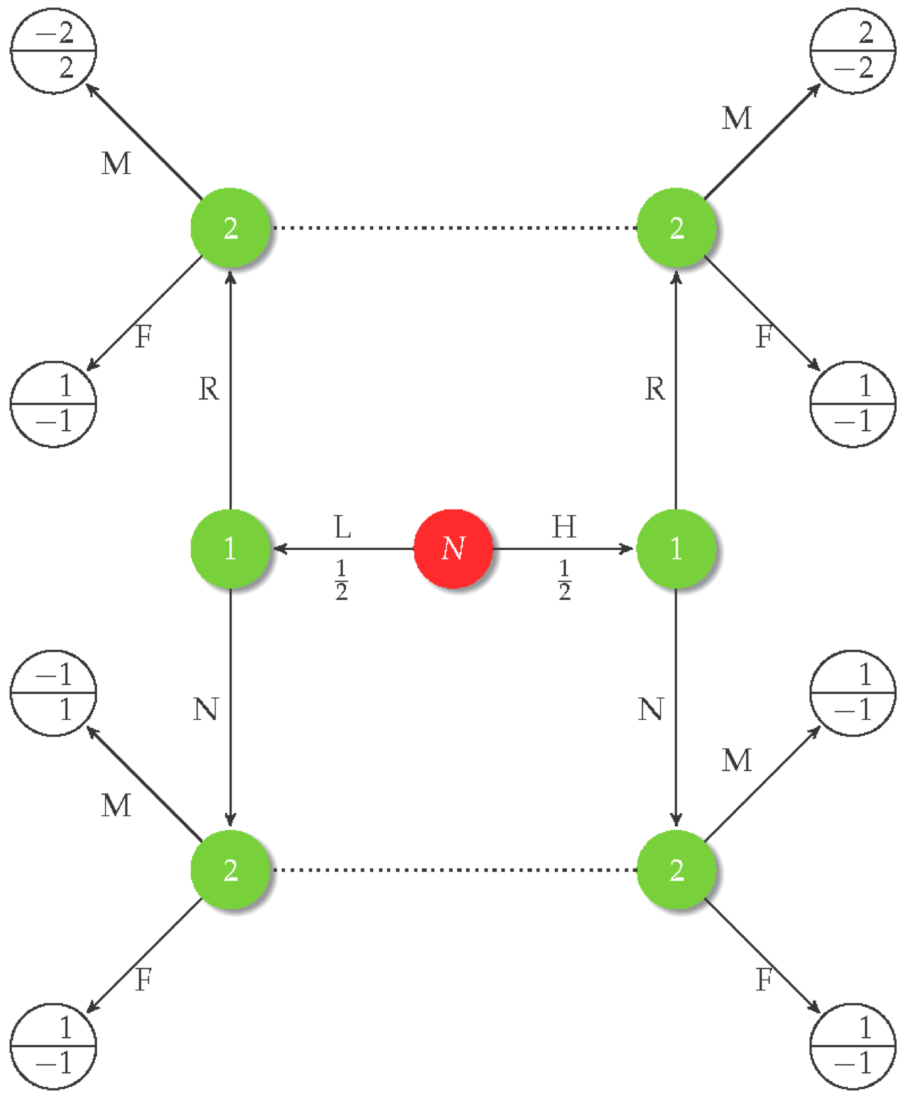
$$\left(p[1] == \frac{1}{2}\ \&\&\ p[2] == 0\ \&\&\ p[3] == \frac{1}{2}\ \&\&\ p[4] == 0\ \&\&\ q[1] == \frac{1}{2}\ \&\&\ q[2] == 0\ \&\&\ q[3] == \frac{1}{2}\right)$$

The game has many Nash equilibria, but we still can make predictions: General B is never going to spread his forces evenly (the probability of his second strategy is zero in any equilibrium, `q[2] == 0`); with probability $\frac{1}{2}$, B's two divisions are placed at the north road (`q[1] ==` $\frac{1}{2}$) and with probability $\frac{1}{2}$, they are placed at the south road (`q[3] ==` $\frac{1}{2}$). As for General A, the probability that she places all of her three divisions on one front is less than half (i.e. `p[1] ≤` $\frac{1}{2}$ and `p[4] ≤` $\frac{1}{2}$). Also, the probability that General A places two or more divisions at the north (or south) is always equal to half (i.e. `p[1] + p[2] ==` $\frac{1}{2}$ and `p[3] + p[4] ==` $\frac{1}{2}$).

## □ A Card Game

This game is also borrowed from [3]. A deck has two cards, one high and one low. Each player places one dollar into the pot. Player 1 gets one card from the deck. Player 2 does not see Player 1's card. Player 1 decides whether to raise (by placing another dollar in the pot) or not raise. Player 2 observes 1's action and then has to decide whether to match the bet or fold. If Player 2 folds, then Player 1 wins the contents of the pot. However, if Player 2 matches, Player 2 places another dollar into the pot if Player 1 had previously raised. Player 1 reveals her card. If it is the high card, Player 1 wins the pot; otherwise, Player 2 wins it.

See Figure 1 for the corresponding game tree. We introduce a fictitious player, Nature, who randomly decides if the card is high or low. We depict the bimatrix representation of the game. Player 1 has four strategies: always raise (RR), always not raise (NN), raise if the card is high and not otherwise (RN), and not raise if the card is high and raise otherwise (NR). Player 2 also has four strategies: always match (MM), always fold (FF), match only if Player 1 raised (MF), and fold only if Player 1 raised (FM). For simplicity, in the bimatrix representation, we write the expected payoffs of Player 1 and omit Player 2's payoffs (this is without loss of generality in zero-sum games).



| | MM | MF | FM | FF |
|---|---|---|---|---|
| RR | 0 | 0 | 1 | 1 |
| RN | 1 / 2 | 3 / 2 | 0 | 1 |
| NR | −1 / 2 | −1 / 2 | 1 | 1 |
| NN | 0 | 1 | 0 | 1 |

▲ **Figure 1.** Game tree of the card game.

```
Module[
 {P, Q, u},
 P = Table[p[i], {i, 4}];
 Q = Table[q[i], {i, 4}];
 u = P.{{0, 0, 1, 1}, {1 / 2, 3 / 2, 0, 1}, {-1 / 2, -1 / 2, 1, 1},
    {0, 1, 0, 1}}.Q ; (* Player 1's expected payoff;
 Player 2's is 1-u. *)
 Nash[u, 1 - u, P, Q]
]
```

$$p[1] == \frac{1}{3} \&\& p[2] == \frac{2}{3} \&\& p[3] == 0 \&\& p[4] == 0 \&\&$$
$$q[1] == \frac{2}{3} \&\& q[2] == 0 \&\& q[3] == \frac{1}{3} \&\& q[4] == 0$$

In this case, the Nash equilibrium delivers a sharp prediction. When Player 1 has the high card, she always raises (`p[1] + p[2] == 1`), but when she has the low card, she bluffs with probability $\frac{1}{3}$ (the probability of RR is `p[1] ==` $\frac{1}{3}$). When Player 1 does not raise, Player 2 always matches (`q[1] + q[3] == 1`). If Player 1 raises, Player 2 still may match, but with probability $\frac{2}{3}$ (the probability of always matching MM is `q[1] ==` $\frac{2}{3}$).

## ■ Summary

We extended the code of [1] to solve for Kuhn–Tucker conditions with additional assumptions on parameters and, more importantly, using the Kuhn–Tucker equations we provide a program to compute all the Nash equilibria of finite bimatrix games.

## ■ Conclusion

We presented a program to compute the set of all Nash equilibria in finite bimatrix games. Its intended goal is as a classroom tool for students and instructors. Needless to say, the code is not efficient. For larger inputs (say bimatrix games with five or more actions per player), `Reduce` often fails to solve the system of Kuhn–Tucker equations. For optimizing algorithms, we suggest [4]. Nevertheless, with continuous improvement of hardware and algorithms for solving semialgebraic systems (see [5]), these methods may become useful for research applications sooner than we think. Finally, as algorithmic game theory courses become more popular in computer science departments, it seems that the time to bring computational methods and algorithms to economics departments is already overdue.

## ■ References

[1] F. J. Kampas, "Tricks of Using Reduce to Solve Khun–Tucker Equations," *The Mathematica Journal*, **9**(4), 2005 pp. 686–689. www.mathematica-journal.com/issue/v9i4/contents/Tricks9-4/Tricks9-4_2.html.

[2] M. J. Osborne. "Optimization: The Kuhn–Tucker Conditions for Problems with Inequality Constraints," from *Mathematical Methods for Economic Theory: A Tutorial.* (Jan 8, 2014) www.economics.utoronto.ca/osborne/MathTutorial/MOIF.HTM.

[3] M. Osborne, "An Introduction to Game Theory," New York: Oxford University Press, 2004.

[4] R. D. McKelvey, A. M. McLennan, and T. L. Turocy. "Gambit: Software Tools for Game Theory." (Jan 8, 2014) www.gambit-project.org.

[5] Wolfram Research, "Real Polynomial Systems" from Wolfram *Mathematica* Documentation Center—A Wolfram Web Resource. reference.wolfram.com/mathematica/tutorial/RealPolynomialSystems.html.

S. O. Parreiras, "Using `Reduce` to Compute Nash Equilibria," *The Mathematica Journal*, 2014. dx.doi.org/doi:10.3888/tmj.16-3.

### About the Author

Sérgio O. Parreiras is an associate professor at the UNC-Chapel Hill Department of Economics. His research focus is on game theory and its applications to auctions, mechanism design, and contests. He is also interested in computational economics, general equilibrium theory, algorithmic game theory, and evolutionary anthropology.

**Sérgio O. Parreiras**
*UNC, Department of Economics*
*Gardner Hall, 200B*
*Chapel Hill, N.C. 27599-3305*
*sergiop@unc.edu*