COMPUTATIONAL ECONOMICS

An introduction by Example

Revised Edition

David A. Kendrick
Ruben P. Mercado
Hans M. Amman

Cournot Duopoly in Mathematica

1 Introduction

Students of economics are introduced first to the market structures of pure competition and monopoly. However, most real world examples are in the domain of oligopolies, which lie between these two extremes.

What distinguishes oligopolistic markets from either purely competitive markets or monopoly markets is that in an oligopoly market there is an interdependency of actions among firms. By interdependency, we mean that the choice of a given firm affects and is affected by the choices of other firms, an issue that does not exist in purely competitive or monopolistic markets.

In a purely competitive industry, it is assumed that individual firms are too small to influence the market price, and therefore the action of one firm has no way of influencing (or being influenced by) the actions of another. Alternatively, a monopolist has tremendous influence over the market price, but as a monopolist, it has no other firms whose actions it can influence. In an oligopolistic industry, two or more firms compete in the market, each of which has the ability to influence the market price and thus the choices of its competitors.

Problems involving interdependency of actions among multiple players are called games, and game theory is the study of these multiplayer decision problems.[1] We use the Mathematica programming language to solve several alternative game theoretic models of oligopoly market structure. All of these models are called *quantity* games as the strategic choice of the firms is quantity. Alternative models of price competition, such as the Bertrand model, in which each player chooses a price, are discussed briefly, but not modeled. We focus here on two-firm oligopoly models, called *duopolies*, but the models can easily be extended to incorporate a larger number of firms (an extension the student is encouraged to undertake in the Experiments section).

Although all of the problems discussed here can be solved with pen and paper, the use of Mathematica opens the door to the solution of substantially more complex models. The three topics covered here are game theory, oligopoly market structure, and the Mathematica programming language. Expertise in any of these individual areas should enhance learning of the other two, but the material covered in all three is kept at an introductory level and no previous knowledge is required. We begin with a short introduction to game theory as a means

[1] For a more comprehensive introduction to game theory see Gibbons (1992). Some of the examples used in the following are drawn from that source.

of introducing the tools and terminology that are required for our oligopoly models. Then we examine two popular models of oligopoly market behavior using Mathematica to derive their results and discuss the intuition of the solutions. We discuss the Cournot model in this chapter and the Stackelberg model in Chapter 10.

2 Game Theory

By identifying our oligopoly markets as games, we have already gone further than you might think toward modeling and solving these models. After all, game theory tells us how we can represent a game as well as how we should approach solving a game. In addition to introducing some basic concepts of game theory, this section discusses a very simple but popular game called the Prisoners' Dilemma, an extremely valuable tool because there is a direct parallel between this simple game and the oligopoly games we plan to solve.

There are many types of games, but it is useful to distinguish among a few of them. To begin with, games can be either *simultaneous move* or *sequential move*. In a simultaneous move game, all of the players choose their actions simultaneously, each without observing the actions of the others. In a sequential move game there is an order to the play. More precisely, a sequential move game is dynamic in that one player chooses an action, and then a second player chooses an action only after observing the first player's action.

A second classification is *complete information* versus *incomplete information* games. All of the games presented in this chapter are complete information games or games in which no firm has private information about itself that other firms cannot access. Finally, it is helpful to distinguish between games with *discrete* strategy choices and those with *continuous* strategy choices.

The first step in approaching a game is representing it, and there are two ways to do this: the normal form (usually a two-way table, appropriate for simultaneous move games) and the extensive form (usually a game tree, more appropriate for sequential move games). Here we adopt the normal-form representation of a simultaneous play game. The three elements that constitute normal-form representation are: (4) the players participating in the game, (2) the strategies (or actions) that are available to each of the players, and (3) the payoffs that each player would receive for each possible combination of strategies. For two-player games in which the strategy choices are discrete, normal-form games can

be represented in table format as the Prisoners' Dilemma game in Table 9.1.[2]

Player 2

Table 1: The Prisoners' Dilemma Game

		Mum	Fink
Player 1	Mum	-1, -1	-9, 0
	Fink	0, -9	-6, -6

Note that Table 9.1 completely represents the Prisoners' Dilemma game according to our definition of a normal-form representation. First there are two players involved in the game—Player 1 and Player 2. Second, both players can choose between the strategies Mum and Fink. Finally, the payoffs from each of the possible combinations of Player 1/Player 2 strategies are represented by the table's payoff matrix. For example, if Player 1 plays Mum and Player 2 plays Fink, then Player 1 receives a payoff of –9 (or 9 years in jail) and Player 2 receives a more favorable payoff of zero.

The story of the Prisoners' Dilemma game is as follows: Two suspects in a crime are detained by the authorities and interrogated separately. Each player can choose to offer no information (Mum) or to blame the crime on the other player (Fink). Furthermore, prisoners must choose their strategies without observing each other's choice. If both players choose Mum each spends only 1 year in jail (one unit of negative utility). If each blames the other (Fink, Fink) then each spends 6 years in jail. If one player chooses Fink and the other chooses Mum, then the player who finks goes free and the player who chose Mum spends 9 years in jail.

In order to solve the Prisoners' Dilemma, we adopt the notion of *Nash equilibrium strategies*. If each player's chosen strategy is the best response to the strategies played by all of the other players, then the strategies of a game's players constitute a pure strategy Nash equilibrium (called Nash equilibrium hereafter).[3] In other words, a Nash equilibrium occurs when every player chooses his strategy optimally given his opponents' chosen strategies.

Applying the concept of Nash equilibrium strategies to the Prisoners' Dilemma,

[2]See Gibbons (1992), p. 3.

[3]In game theory there is a distinction between pure strategy Nash equilibria and mixed strategy equilibria. This is a distinction that is beyond the scope of this chapter except to note that by Nash equilibria we mean pure strategy Nash equilibria.

we can find the Nash equilibrium. Begin by considering how Player 2 would best respond to Player 1 using the strategy Mum. In this event, Player 2 could also play Mum and spend a year in jail or Player 2 could play Fink and go free. So Player 2's best response to Player 1 choosing Mum is Fink. Therefore, the strategy (Mum, Mum) does not constitute a Nash equilibrium.

Does Player 1 choosing Mum and Player 2 choosing Fink (Mum, Fink) constitute a Nash equilibrium? The answer is no, because while Fink is Player 2's best response to Player 1 playing Mum, in order to be a Nash equilibrium it must also be the case that Mum is Player 1's best response to Player 2 playing Fink. However, we see that if Player 2 Finks, Player 1's best response is to Fink as well. Continuing with this logic you find that the only pure strategy Nash equilibrium for this game is for both players to fink on each other (Fink, Fink).

We have barely scratched the surface of game theory through this example, but we have addressed a few of the basics that allow us to better understand oligopoly market structure. First, we know that the three elements that constitute a simultaneous game are the players of the game, the strategies (or choices) available to each of these players, and each player's payoff for every possible combination of players' strategies. A fourth element, which we will see is important in sequential move games, is determining the order of play. These four are necessary for characterizing and solving any game. Finally, our simple example illustrated the solution concept that we employ in our oligopoly problems—the pure strategy Nash equilibrium. The intuition of a Nash equilibrium is that each player is choosing his best response or reaction to all of the other players' choices.

3 Static Models of Oligopoly Markets

Models of oligopoly markets depend on how firms interact, characteristics of the environment in which they interact, and potentially many other factors, so there is no single model of oligopoly market structure. The correct model depends on the characteristics of the industry being modeled. We focus here on two *one-shot* quantity games that are the foundations for many more sophisticated models of oligopoly markets. These models are one-shot in the sense that the game is played only a single time, not repeatedly every period. They are referred to as quantity games because the strategic choices of the firms' are their respective outputs (or quantities). As we will see, quantity games have an interesting characteristic that we exploit: If the quantity choices are assumed to be continuous, then the payoffs are also continuous.

Natural alternatives to quantity games are pricing games. Models of price competition have a winner-take-all aspect, according to which the firm that has the lowest price captures the entire market (and the market is split in the event of a tie). Therefore, whereas the strategic variable price is continuous, the payoffs (profits) are discontinuous. We do not consider pricing models except to note that the solutions to such games vary significantly from the quantity games considered here.

4 Cournot Competition

The first model of oligopoly market structure that we study is the Cournot quantity competition, named for the French mathematician Augustin Cournot, and first described in his book *Researches into the Mathematical Principles of the Theory of Wealth*, published in 1838, some 112 years before John Nash formalized the concept of Nash equilibrium strategies.

To make the problem more tractable, but without loss of generality, we assume that the industry is a duopoly. Our story of a Cournot duopoly market is as follows: There is a market consisting of two firms, each producing a homogeneous good at a constant (not necessarily the same) marginal cost. We assume that each firm knows its own cost as well as its competitor's cost and that they both know the market's demand function,

$$Q = f(p) \tag{1}$$

where
Q = quantity
p = price
and that they can derive the inverse demand function

$$p = g(Q) \tag{2}$$

The problem faced by these two firms is that each must choose the quantity that it supplies to the market without observing its competitor's output choice and let the market determine the price. This requires each firm to anticipate, when choosing its own quantity, how the other firm will behave.

The game described here is a one-shot simultaneous move game and as such can be represented as a normal form game. To do so we have to identify the

three necessary ingredients. First, we have the players, the two firms, which we denote as Firm 1 and Firm 2. Next, we must identify the firms' strategic choice. As already noted, that choice is quantity, which is assumed to be a continuous nonnegative variable. Finally, the payoff to each of the players in the game is simply the profit that this firm earns given its choice of quantity and the quantity chosen by the other firm.

To solve this game we apply the same solution concept we used to solve the Prisoners' Dilemma game. A pure strategy Nash equilibrium for this Cournot game is a set of quantities

$$(Q_1^*, Q_2^*) \tag{3}$$

in which each of the firms chooses its profit-maximizing output given its forecasted output choice of the other firm, and each firm's forecast of the other's output is correct. Recall that in the discrete strategy Prisoners' Dilemma game, finding the Nash equilibrium required us to consider each possible combination of strategies. However, in the Cournot game each firm has a continuum of possible strategy choices, so there are an infinite number of possible combinations of players' strategies. Fortunately, in the continuous Cournot model, we can generalize the strategic behavior of the firms by deriving what we call a reaction (or alternatively a best response) function. As we will see, calculus permits us to do this because our firms' payoffs (their profits) are continuous functions of their own quantity choice as well as the other firm's quantity. To see this more clearly, we begin with a Mathematica program (react.nb), which is available on the book web site. The program illustrates strategic behavior and the solution to the Cournot model graphically. This experiment is designed to familiarize the reader with the concept of a reaction function and to illustrate its connection to determining Nash equilibria. The instructions for running Mathematica are in Appendix 7. Turn to that appendix and run the react.nb file if you prefer to follow the Mathematica code while you are reading the rest of this chapter.

The first step in building the graphical model of the Cournot game is to model the industry characteristics. Because the market is a duopoly, the total market quantity Q is the sum of Firm 1's output choice, Q_1, and Firm 2's output choice, Q_2, that is,

$$Q = Q_1 + Q_2 \tag{4}$$

This simple assignment is made in Mathematica with the following input statement:

IN[]:= Q = Q1 + Q2;

The symbols *IN[]:*= are the Mathematica prompt for input and the expression Q = Q1 + Q2; is the user's input. It is important to note that the equal sign in the input is used in Mathematica for assigning names to expressions (or assigning values to variables). It does not define a formula or equation. To create an equation, one must use two equal signs (= =). Also note that the semicolon at the end of the foregoing command is used to suppress the display of output created by Mathematica for each input statement.

The other general market characteristic that we must specify at this time is the functional form for the inverse market demand faced by the duopolists. In all of our models we assume that the inverse demand curve is linear, that is,

(2) Price $= a - bQ$

or in Mathematica

IN[]:= Price = a – b*Q

As the Cournot game is one of simultaneous choice, it does not matter which firm's optimization problem we consider first, so we start with Firm 2's problem.

Firm 2's profits are equal to the difference between the revenue from selling the quantity Q_2 and the cost of selling this quantity. Revenue is Firm 2's own quantity (Q_2) multiplied by the market price, that is,

(3) $\text{Profit}_2 = Q_2\,(\text{Price} - c_2)$

where c_2 is Firm 2's constant unit cost. This can be written in Mathematica as

IN[]:= eqPr2 = Profit2 == Q2(P[Q1,Q2] - c2) ;

Note that the price P is a function of the quantity decisions of the two firms, that is, Q1 and Q2. The above statement creates a formula named eqPr2 that defines Profit2 (Firm 2's profits) to be equal to Q2 multiplied by the difference between the market price and the firm's marginal cost. We have used the string

P[Q1,Q2]

here to indicate that the price is a function of both Q1 and Q2; however, Mathematica does not recognize the functional dependency. Rather it just treats P[Q1,Q2] as a string of characters.

This is the general representation of Firm 2's profits. However, because we have specified that our industry is subject to a linear demand curve, we want to replace the general form of the price function with the linear demand specified in the earlier Mathematica statement,

Price = a – b*Q

While the practice of defining a generalized profit function and then substituting in a specific functional form may seem cumbersome, it is good programming practice because it allows us to change the functional form of our market demand by editing a single Mathematica statement.

Mathematically we obtain an expression for the profit of the second firm by substitution of equation (2) into equation (3) to obtain

(4) $\text{Profit}_2 = Q_2 \left[a - bQ - c_2\right]$

and then substituting equation (4) into equation (4) to get

(5) $\text{Profit}_2 = Q_2 \left[a - b\left(Q_1 + Q_2\right) - c_2\right]$

These steps are accomplished in Mathematica in the following way. First the substitution of the specific linear demand function for the general form is done with the Mathematica statement

IN[]:= eqPr2 = Expand[% /. P[Q1,Q2] -> Price]

In Mathematica, % refers to the last result generated (and %% refers to the second to last result generated, and so on) and /. is the replacement identifier. So the foregoing statement takes the original profit equation (named eqPr2) and replaces the general form of the demand equation P[Q1,Q2] with our explicit linear inverse demand expression Price specified earlier as (Price = a – b*Q). Since there is no semicolon at the end , the output statement gives the result of

this substitution:

OUT[]:= Profit2 == Q2 (a - c2 - b (Q1 + Q2))

This output is the equation for Firm 2's profits or payoff as a function of the two firms' quantities.

To find Firm 2's profit-maximizing behavior, we take the derivative of its profit function, equation (5), with respect to its choice variable Q_2 and set the expression equal to zero, that is

$$frac\partial \text{Profit}_2 \partial Q_2 = a - c_2 - b\,(Q_1 + Q_2) + Q_2\,(-b) = 0 \qquad (5)$$

To accomplish this in our Mathematica program we define a new equation, named focPr2 (first-order condition for profit of Firm 2), which is the derivative, D[ ], of Firm 2's profit (payoff) function with respect to its choice of its own quantity Q2:

IN[]:= focPr2 = D[eqPr2, Q2]

This produces the following output, which is the derivative of equation eqPr2 (after the substitution) with respect to Q2:

OUT[]:= 0 == a - c2 - b Q2 - b (Q1 + Q2)

This first-order condition implicitly describes Firm 2's optimal behavior, but we want to find an explicit solution.

This is done by solving the first-order condition in equation (6) for Q_2, that is,

(7) $2bQ_2 = a - c_2 - bQ_1$

or

(8) $Q_2 = \frac{a - c_2 - bQ_1}{2b}$

We accomplish this in Mathematica by using the Solve statement to solve the first-order condition for Q2 and then naming the output temp2. It is helpful to note that the output from a Solve statement is a list of solutions, so that temp2 is the name of that list. In the case below, the list temp2 has only one solution

and therefore a single element:

IN[]:= temp2 = Solve[focPr2, Q2]

OUT[]:= Q2 -> ()

It is clear from the first-order condition above that Firm 2's optimal choice of Q2 depends on Firm 1's optimal choice Q1. This is the key to game theoretic problems; each party must consider what the other parties will do. Because Firm 2's choice of Q2 is a function of Firm 1's choice of Q1, we call the expression above Firm 2's *best response* or *reaction function*. As its name implies, this function dictates how Firm 2 chooses Q2 as a best response (or in reaction) to Firm 1's choice of Q1. In the simple linear case, we can solve for Firm 2's best response quantity (R2[Q1]) explicitly.

In the next line of code, we create an expression called React2 (Firm 2's reaction function), which represents Firm 2's optimal response (R2[Q1]) to Firm 1's quantity choice Q1:

IN[]:= React2 = R2[Q1] == Q2 /. temp2[[1]]

OUT[]:= R2[Q1] == ()

The right-hand side of this expression is simply the solution for Q2 that we found earlier. In other words, R2[Q1] is equal to Q2, where Q2 is replaced (/.) by the first solution ([[1]]) from the output list temp2, and R2[Q1] is just another name for Q2 that reflects the fact that this quantity is chosen in response to Q1.

We conclude our examination of Firm 2's behavior in the Cournot duopoly model by graphing the reaction function for Firm 2 that has been derived from the foregoing model:

IN[]:=

```
reactPlot = Plot[Q2 /. Solve[focPr2 /. a -> 1, b -> 1, c2 -> .5,Q2][[1]],
    Q1, 0, .55,
    PlotRange -> 0, .55,
    PlotStyle->RGBColor[1,0,0],Thickness[0.010],
    AxesLabel->"Q1","Q2",
```

PlotLabel->"Reaction Curve"]

This rather messy-looking Mathematica command creates a plot that we name reactPlot using the Plot command. The syntax for the Plot command is

Plot[f, x, xmin, xmax, option -> value]

where f is the expression to be plotted, the list x, xmin, xmax specifies the minimum and maximum values taken by the variable in the expression, and option -> value statements are used to set any display attributes of the graph.

Starting from the second line of the code above following the *IN[]:*= statement, the Plot[] command is used to create the plot. The function f that we want to plot is an expression that represents the values that Q2 takes—expressed in terms of the variable Q1 and the model's parameters, that is,

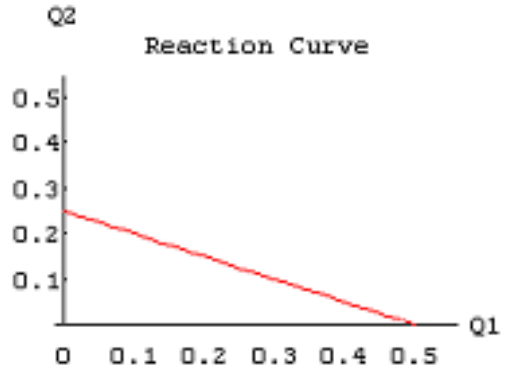
Q2 /. Solve[focPr2 /. a -> 1, b -> 1, c2 -> .5,Q2][[1]]

Our identification of Firm 2's reaction equation has shown that in the linear demand case we can find such an expression by solving the first-order condition, focPr2, for Q2. Therefore, this line of code tells Mathematica that we want to plot the values of the variable Q2, where an expression for Q2 is found from solving the first-order condition of Firm 2's profit function for the variable Q2, that is,

Solve[focPr2, Q2]

Within this Solve statement is a replacement command (/.) followed by a list (,) of replacements that specify the numerical values the model's parameters are assumed to take in this example.

The next line of this Plot command, Q1,0,.55, specifies the range of values for the variable Q1 in the expression for Q2. The remaining lines of code specify display options for the Mathematica plot. Options are used to control the plot color and thickness, axes labels, and plot labels. The resulting plot (Fig. 9.1) shows quantity Q2, that is, Firm 2's best response to any given quantity of Firm 1 (Q1). Thus if Firm 1 chooses Q1 = 0.3, then Firm 2's optimal reaction is to choose Q2 = 0.1.



Next we turn to the optimization problem for Firm 1. It solves a problem that is identical to Firm 2's except for the fact that Firm 1 solves for its own quantity Q1 and has a marginal cost of c1. The solution to Firm 1's reaction function is

OUT[]:= R1[Q2] == ()

We can plot this relationship for Firm 1 assuming the same parameter values and over the same interval of values as we did for Firm 2:

IN[]:=

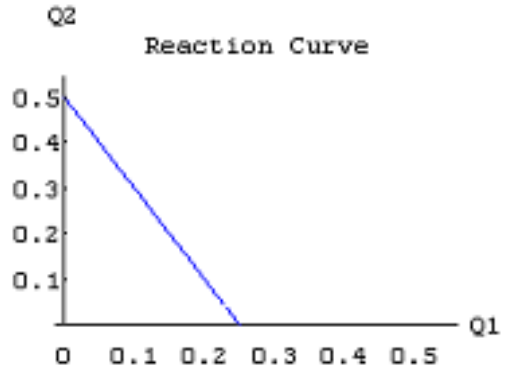
reactPlot =

```
Plot[Q2 /. Solve[focPr1 /. a -> 1, b -> 1, c1 -> .5, Q2][[1]],
    Q1, 0, .55,
    PlotRange -> 0, .55,
    PlotStyle-> RGBColor[0,0,1],Thickness[0.001],
    AxesLabel-> "Q1","Q2",
    PlotLabel->"Reaction Curve"]
```

In the graphical illustration of the Cournot solution shown in Figure 9.2 we are assuming that the firms have identical cost structures (c1 = c2 = 0.5). This plot shows what

quantity Q1 is Firm 1's best response to any given quantity choice by Firm 2 (Q2).

At this point we have plots showing how each firm should best respond to its competitor's various choices of output. According to our definition of Nash



equilibria, a Nash equilibrium of this game is a set of strategies in which each firm chooses an output that is a best response to the other's output choice. Or more succinctly, at a Nash equilibrium, both players are on their reaction functions. Before looking at this graphically, however, it is useful to see how the costs affect the solutions to our simple Cournot model. The simplest and most intuitive way to investigate this consideration is with another Mathematica plot:

IN[]:=

```
reactPlot = Plot[Q2 /. Solve[focPr1 /. a -> 1, b -> 1, c1 -> .5, Q2][[1]],
    Q2 /. Solve[focPr1 /. a -> 1, b -> 1, c1 -> .6, Q2][[1]],
    Q1, 0, .55,
    PlotRange -> 0, .55,
    PlotStyle->
    RGBColor[0,0,1],Thickness[0.001], ,
    RGBColor[0,0,1],Thickness[0.001],Dashing[.03,.02,
    AxesLabel-> "Q1","Q2",
    PlotLabel->"Reaction Curves"]
```

Note the syntax of the foregoing Mathematica statement. As we are plotting two reaction functions, the first element in the Plot[] command becomes a list of expressions f1, f2, where the first element in the list is Firm 1's reaction function when its marginal cost is 0.5 and the second is its reaction plot when its own marginal cost is 0.6. In the above,

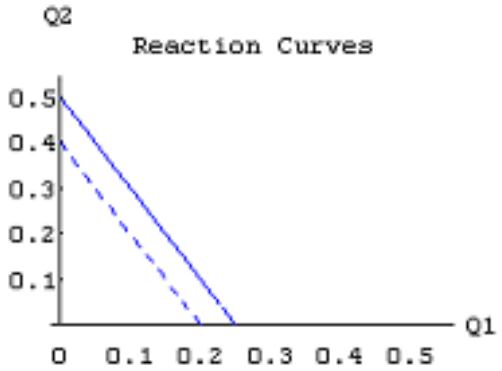
RGBColor[0,0,1], Thickness[0.001]

indicates that the first plot will be a solid blue line,[4] which is 0.001 thick, and

[4]Syntax: RGBColor [*red, green, blue*], where color intensities range from 0 to 1.

RGBColor[0,0,1], Dashing[.03,.02]

indicates that the second line will be blue and have dashes of length 0.03 and spacing of 0.02. With this statement we create a graph (Fig. 9.3) that plots Firm 1's reaction curve with the original parameter specifications and a marginal cost of 0.5 and also contains a second dashed plot that shows its reaction curve with a slightly higher marginal cost of 0.6. The color will show when the program is run but is not shown in Figure 4.



From this graphical sensitivity analysis of Firm 1's reaction functions we can see that higher unit costs, in the dashed line, shift a firm's reaction function downward. That is, for a given output choice by Firm 2, Firm 1's best response quantity is decreased for higher values of its own marginal cost. It is easy to see that Firm 2's marginal cost, c2, has no effect on the Firm 1's reaction function since the Mathematica variable c2 does not appear in Firm 1's reaction function. Similarly, Firm 1's marginal cost has no effect on Firm 2's reaction function.

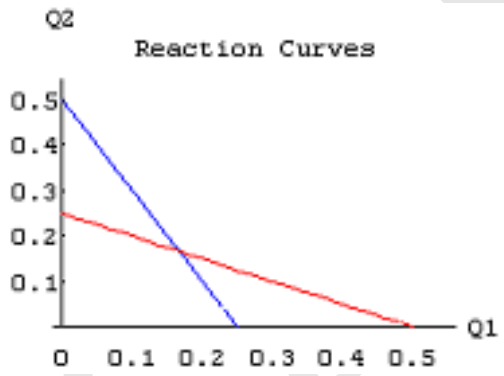
Our plots of the firms' reaction functions illustrate how each firm chooses its optimal quantity in response to the quantity choice of the other firm. If we knew Firm 1's choice of Q1 we could solve Firm 2's reaction function for its optimal choice of Q2. Similarly, if we knew Firm 2's choice Q2 we could solve Firm 1's reaction function for its optimal choice of Q1. The difficulty with the Cournot game is that the players choose their respective quantities simultaneously. Therefore, the solution to the simultaneous move Cournot game is found by solving both firms' reaction functions simultaneously for the quantities Q1 and Q2. Intuitively, the Nash equilibrium solution to the Cournot game has each firm choosing its best response quantity in reaction to the hypothesized quantity of the other firm.[5] To show the graphical solution to the Cournot model

[5]See Gibbons (1992), p. 62.

we plot both firms' reaction functions in a graph.

IN[]:=

```
reactPlot = Plot[Q2 /. Solve[focPr1 /.a -> 1, b -> 1, c1 -> .5, Q2][[1]],
    Q2 /. Solve[focPr2 /. a -> 1, b -> 1, c2 -> .5, Q2][[1]],
    Q1, 0, .55,
    PlotRange -> 0, .55,
    PlotStyle->
    RGBColor[0,0,1],Thickness[0.001],
    RGBColor[1,0,0],Thickness[0.010],
    AxesLabel->"Q1","Q2",
    PlotLabel->"Reaction Curves"]
```



The foregoing Mathematica statement produces a graph showing the equilibrium strategies. At the point where the two firms' reaction functions intersect in Figure 9.4, each firm is choosing its respective profit-maximizing output given their belief about the other firm's output choice and each of the firm's beliefs about the other is correct. This is the definition of a pure strategy Nash equilibrium.

This completes our use of the react.nb Mathematica file. While the graphical model in Figure 9.4 illustrates the behavior of the Cournot game's players in an intuitive way and clearly demonstrates how the Nash equilibrium is determined, we want more from our model than intuition. The next Mathematica program (cournot.nb) is a model of the same Cournot duopoly, but in addition to a graphical solution, we derive the analytic solution to this problem. Solving for the Cournot-Nash quantities permits us to determine the market supply and

consequently the market price. In addition, with the Nash equilibrium quantities and the corresponding market price we can derive the two firms' profit levels. All of this information is useful if we want to make comparisons among models of different market structures.

In the file cournot.nb we derive each firm's reaction function just as in the previous model. However, after identifying the two firms' reaction functions, rather than using a plot to find the Cournot-Nash solution, we solve for the optimal quantities directly. Recall from our graphical example that the Nash equilibrium strategy set was defined by the intersection of the two firms' reaction functions. Also remember that our definition of a Nash equilibrium requires that all the players simultaneously give their best response to each other's choices. Clearly, the Nash equilibrium strategies can be found by simultaneously solving the set of reaction functions for the models' strategic output choices.

To solve for the optimal quantities we add the following Mathematica statement:

IN[]:=

```
cournotQ =
    Simplify[Solve[React1 /. R1[Q2] -> Q1,
    React2 /. R2[Q1] -> Q2, Q1, Q2]]
```

This command renames each of the firms' reaction quantities R*i*[Q*j*] with the firm's actual chosen quantity Q*i* and then solves the two equations simultaneously for the choice variables Q1 and Q2. The resulting output from the command is the Nash equilibrium strategy:

OUT[]:= Q1 -> (), Q2 -> ()

In order to save these results we create two new Mathematica variables, Q1c and Q2c for each of the respective firms' Cournot quantities:

IN[]:= Q1c = Q1 /.%[[1]] ;

IN[]:= Q2c = Q2 /.%%[[1]] ;

The interpretation of these statements is: Q1c is defined as the variable Q1 where Q1 is replaced with the values from the first solution of the previous

Mathematica output, and similarly for Q2c.

In a similar manner we derive and store the Cournot market output, the Cournot market price, and Firm 1's profits and Firm 2's profits, respectively:

IN[]:= Qcour = Q /. Q1 -> Q1c, Q2 -> Q2c

OUT[]:= ()+ ()

In this statement, the Cournot market output, Qcour, is the market output, Q, which was defined to be Q1 + Q2, where we replace Q1 with Firm 1's Cournot quantity Q1c and Q2 with Firm 2's Cournot quantity Q2c.

The Cournot market price, Pcour, is found by substituting the firms' Cournot outputs, Q1c and Q2c, in place of Q1 and Q2 into the inverse demand function Price, which was defined at the start of the program, that is,

IN[]:= Pcour = Simplify[Price /. Q1 -> Q1c, Q2 -> Q2c]

OUT[]:= ()

The Mathematica Simplify command is used in the foregoing input statement to provide a simplified output expression.

With the market price determined, calculating the two firms' profits is straightforward. Firm 1's Cournot profit, designated pie1c to represent the profit (the Greek letter π) of the first firm in the Cournot solution, is the firm's Cournot output Q1c multiplied by the difference between the price and Firm 1's unit cost, that is, (Pcour – c1):

IN[]:= pie1c = Simplify[Q1c*(Pcour - c1)]

OUT[]:= (()())

A similar expression calculates Firm 2's Cournot level of profits, pie2c:

IN[]:= pie2c = Simplify[Q2c*(Pcour - c2)]

OUT[]:= (()())

5 Experiments

In this chapter we develop a series of experiments that cover many of the aspects of the models presented. However, one set of experiments to consider is the use of alternative cost functions and another is to examine modeling alternative market structures.

6 Further Reading

For an introduction to Mathematica see Wolfram (2003). For a more comprehensive introduction to game theory see Gibbons (1992). Some of the examples used in this chapter are drawn from that book so the reader will find continuity between this chapter and the Gibbons volume. For an introduction to the use of Mathematica in game theory see Dickhaut and Kaplan (1993). For a study on the use of Mathematica to simulate the effects of mergers among noncooperative oligopolists see Froeb and Werden (1996).

We turn in the next chapter to a different approach to solving the oligopoly problem, namely the Stackelberg Leadership model.

References

Dickhaut, J. and Kaplan, T.: 1993, A program for finding nash equilibrium, *in* H. R. Varian (ed.), *Economic and Financial Modeling with Mathematica*, Springer-Verlag, New York, USA, chapter 7, pp. 148–166.

Froeb, L. and Werden, G. J.: 1996, Simulating the effects of mergers among noncooperative oligopolists, *in* H. R. Varian (ed.), *Computational Economics and Finance: Modeling and Analysis with Mathematica*, Springer-Verlag, Santa Clara, California, USA, chapter 8, pp. 177–195.

Gibbons, R.: 1992, *Game Theory for Applied Economists*, Princeton University Press, Princeton, New Jersey, USA.

Wolfram, S.: 2003, *The Mathematica Book*, 5th edn, Wolfram Media and Cambridge University Press, Cambridge, United Kingdom.

Appendix

7 Running Mathematica

Mathematica is a widely available commercial software system. A web site for information about it is [http://www.wolfram.com].

Choose Programs from the Start menu and then choose Mathematica. Wait for a few seconds until a new window with a menu bar in its upper part appears. The content of this window is a white sheet called Notebook, which is like a document in a standard word processor. If you specified a file when opening Mathematica, this file is displayed.

Select Getting Started from the Help menu (located in the upper-right corner of the Mathematica window) and read the information that appears in the Info dialog box. To begin practicing with Mathematica, perform the calculations suggested in the section Doing Calculations. While doing this, you will appreciate the basic way of working with Mathematica, that is, your Notebook successively displays your inputs and the corresponding outputs. You may also notice that on the right side of your Notebook, a hierarchy of brackets appears. Each of them defines a cell (or a group of cells) that is the basic unit of organization in a Notebook. As you will quickly realize, cells can be hierarchically arranged (as in set and subsets). There are different kinds of cells: they can contain text, Mathematica input, Mathematica output, or graphs, and so on. Different small characters within each bracket identify the kind of cell. Here are some basic things you can do with cells:

- To edit a cell (i.e., to be able to work with it) just click on its bracket;
- To edit a group of cells, just click and drag on their brackets;
- To find out or change the kind of cell, edit the cell, select Style in the main menu and choose your option.
- To divide or merge cells, take the cursor to the division/insertion point of your choice, select Cell in the main menu and choose your options.
- To run a portion of a program contained in a cell of group of cells, just edit the cells and select Action-Evaluate in the main menu (or just press Shift-Enter)
- Finally, to save your Notebook, select File in the main menu and choose your options.

- Go to the book web site at [http://www.eco.utexas.edu/compeco] and then to the *Input Files for Chapters in the Book* section of the web site. Right click on the Leontief.nb file name and select the Save Target As ... option in order to save the file in your preferred directory.
- Go to the File menu and click Open to open the file**.**
- To run an input command or a cell containing a series of commands, click on the bracket on the right of it and hit Shift + Return (hold the shift key and hit Return at the same time). The output is displayed following the input, *unless* there is a **;** at the end of the input command line (**;** suppresses the output). You can run multiple cells by highlighting the corresponding brackets with your mouse and hitting Shift + Return once.
- Modify commands and rerun them sequentially, cell after cell, so that you can see the changes in the corresponding outputs.
- If you either select the outermost bracket and press Shift-Enter or go to the Kernel menu, choose Evaluation**,** and Evaluate Notebook**,** you rerun the complete program. If your program is large this may take a few minutes and it may be difficult for you to track down the results of your modifications. On the other hand, sometimes your modifications may require an updating of previous results, a clearing of previous values, or a change of attributes (and the Clear command or the SetAttributes command are usually at the beginning of the program). In these cases you may need to rerun the complete program to avoid errors or spurious results.