Programming for Business Computing

Applications in Economics

Ling-Chieh Kung

Department of Information Management
National Taiwan University

Applications in Economics

- Just like management and business, we may find a lot of things in economics that may be helped by computer programming.
- In this lecture, we will demonstrate how a **numerical solution** of an **equilibrium analysis** may be obtained through programming.
  - Especially when an analytical solution cannot (or is too hard to) be found.
- We will use some classic examples in **game theory** to do the demonstration.

Programming for Business Computing

Game Theory and Equilibrium Analysis

Ling-Chieh Kung

Department of Information Management
National Taiwan University

Prisoners' dilemma: story

- A and B broke into a grocery store and stole some money. Before police officers caught them, they hided those money carefully without leaving any evidence. However, a monitor got their images when they broke the window.
- They were kept in two separated rooms. Each of them were offered two choices: **Denial or confession**.
  - If both of them deny the fact of stealing money, they will both get one month in prison.
  - If one of them confesses while the other one denies, the former will be set free while the latter will get nine months in prison.
  - If both confesses, they will both get six months in prison.
- They **cannot communicate** and they must make their choices **simultaneously**.
- All they want is to be in prison as short as possible.
- What will they do?

Prisoners' dilemma: analysis

- We may use the following matrix to formulate this "game:"

		Player 2	
		Denial	Confession
Player 1	Denial	$-1, -1$	$-9, 0$
	Confession	$0, -9$	$-6, -6$

  – There are two **players**, each has two possible **actions**.
  – For each combination of actions, the two numbers are the **utilities** of the two players: the first for player 1 and the second for player 2.
- Prisoner 1 thinks: "If he denies, I should confess. "If he confesses, I should still confess. I should confess anyway!"
- For prisoner 2, the situation is the same.
- The **solution** (outcome) of this game is that both will confess.

Prisoners' dilemma: discussions

- This outcome can be "improved" if they can **cooperate**.
- **Lack of cooperation** can result in a **lose-lose** outcome.
  – Such a situation is **socially inefficient**.
- We will see more situations similar to the prisoners' dilemma.

Solutions for a game

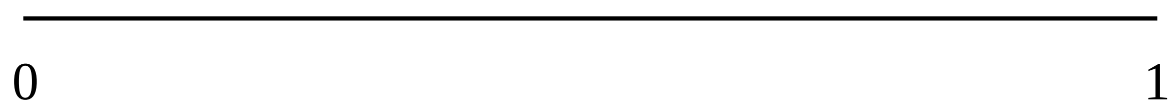
- In this game, confession is said to be a **dominant strategy**.
- Not all the games can be solved by finding dominant strategy. E.g.,

		Player 2		
		L	C	R
Player 1	T	0, 4	4, 0	5, 3
	M	4, 0	0, 4	5, 3
	B	3, 5	3, 5	6, 6

- We need a new solution concept: **equilibrium**!

An example: retail locations

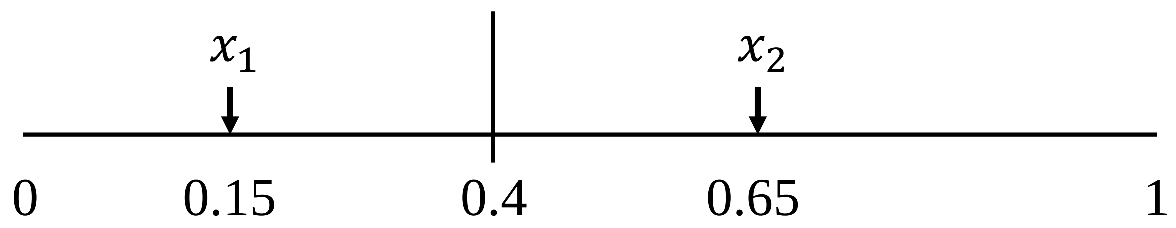
- Consider two retailers setting their locations along $[0, 1]$.
- Consumers spread **uniformly** along the line segment $[0, 1]$.



- The retail price is fixed. Therefore, each retailer wants to maximize her **market share**.
- All consumers go to **the closer store** and buy one unit.

An example: retail locations

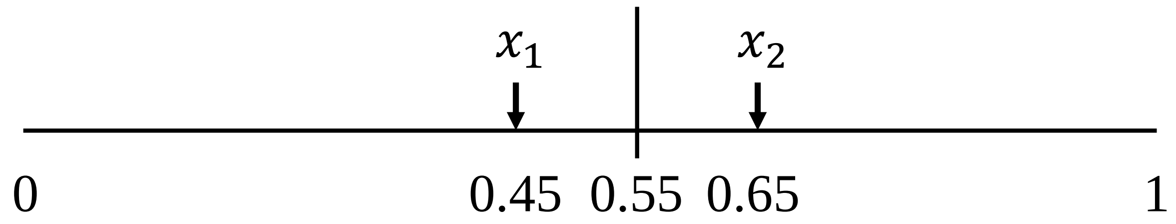
- Let x_1 and x_2 be the locations decided by retailers 1 and 2.
  - For example, suppose initially $x_1 = 0.15$ and $x_2 = 0.65$:



- Market shares?
  - Retailer 1's market share is 0.4 and retailer 2's is 0.6.
- If you are retailer 1, what will you do?

An example: retail locations

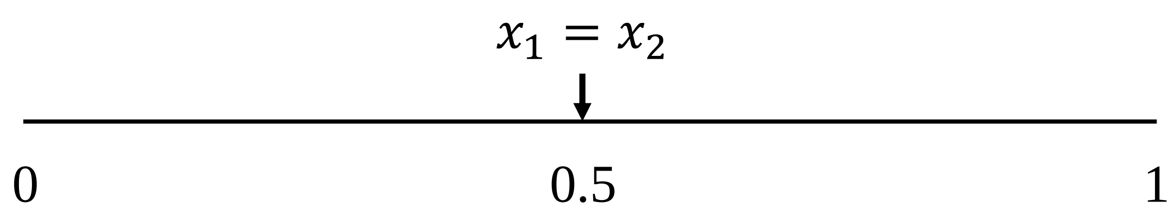
- \item Retailer 1 can move to the **right**, e.g., to $x_1 = 0.45$.



- Retailer 1's market share is now 0.55.
- If you are retailer 2, how will you react?

Equilibrium

- The unique **equilibrium** is $\boldsymbol{x_1 = x_2 = 0.5}$.



  - Both players' market share are 0.5.
- This is not the only case that they share the market equally.
  - However, this is the only case that no one wants to **unilaterally deviate**.
- The Nobel Laureate John Nash formalized the above idea as **Nash equilibrium**.
  - In a (Nash) equilibrium, no player wants to deviate while all others do not.

Programming for Business Computing

Cournot Competition with Identical Costs

Ling-Chieh Kung

Department of Information Management
National Taiwan University

Cournot Competition

- In 1838, Antoine Cournot introduced the following **quantity competition** between two firms.
- Let q_i be the production quantity of firm i, $i = 1, 2$.
  - $Q = q_1 + q_2$ is the aggregate supply.
- Let $P(Q) = a - Q$ be the market-clearing price.
- Unit production cost of both firms is $c < a$.
- Each firm wants to maximize its own profit.
- Our questions are:
  - In this environment, what will these two firms do?
  - Is the outcome satisfactory?
  - What is the difference between duopoly and monopoly (i.e., decentralization and integration)?

Cournot Competition

- Players: firms 1 and 2.
- Action spaces: $S_i = [0, \infty)$ for $i = 1, 2$.
- Firm 1's utility (profit) function is

$$u_1(q_1, q_2) = q_1[a - (q_1 + q_2) - c].$$

- Firm 2's utility (profit) function is

$$u_2(q_1, q_2) = q_2[a - (q_1 + q_2) - c].$$

- As for an outcome, we look for a equilibrium.

Best responses

- Given q_2, firm 1's **best response** q_1^* as a function of q_2 is

$$q_1^* = \frac{a - q_2 - c}{2}.$$

  - This is the unique solution to $\frac{d}{dq_1}\{q_1[a - (q_1 + q_2) - c]\} = 0$.
  - If he produces q_2 units, I should produce q_1^* units to maximize my profit.
- Similarly, given q_1, firm 2's **best response** q_2^* as a function of q_1 is

$$q_2^* = \frac{a - q_1 - c}{2}.$$

  - If he produces q_1 units, I should produce q_2^* units to maximize my profit.
- Everyday one of the two firms responds optimally to its competitor.
  - What will be the **equilibrium**? When will they stop moving?

Moving toward the equilibrium

- As an example, suppose that $a = 10$ and $c = 2$.
- Best responses: $q_1^* = \frac{10-q_2-2}{2}$ and $q_2^* = \frac{10-q_1-2}{2}$.
- Iteration 0:
  - Initially, firm 1 enters the market and produces $q_1 = \frac{10-2}{2} = 4$.
  - Then firm 2 enters the market and produces $q_2 = \frac{10-4-2}{2} = 2$.
- Iteration 1:
  - Firm 1 responds and produces $q_1 = \frac{10-2-2}{2} = 3$.
  - Firm 2 responds and produces $q_2 = \frac{10-3-2}{2} = 2.5$.
- And iterations 2, 3, 4, …
- Why not write a program to find an equilibrium?

Finding the equilibrium

- Best responses: $q_1^* = \frac{10-q_2-2}{2}$ and $q_2^* = \frac{10-q_1-2}{2}$.

```
def printEq(q1, q2, n):
  for i in range(n + 1):
    s = "(" + str(round(q1[i], 4))
    s += ", " + str(round(q2[i], 4))
    s += ")"
    print(s)

a = 10.0
c = 2.0
n = 10

q1, q2 = CournotEq(a, c, n)
printEq(q1, q2, n)
```

```
def CournotEq(a, c, n):
  # iteration 0: entering the market
  q1 = list()
  q1.append((a - c) / 2)
  q2 = list()
  q2.append((a - q1[0] - c) / 2)

  # in each iteration, respond once
  for i in range(n):
    q1Next = (a - q2[i] - c) / 2
    q1.append(q1Next)
    q2Next = (a - q1Next - c) / 2
    q2.append(q2Next)

  return q1, q2
```

Finding the equilibrium

- The outcome is:

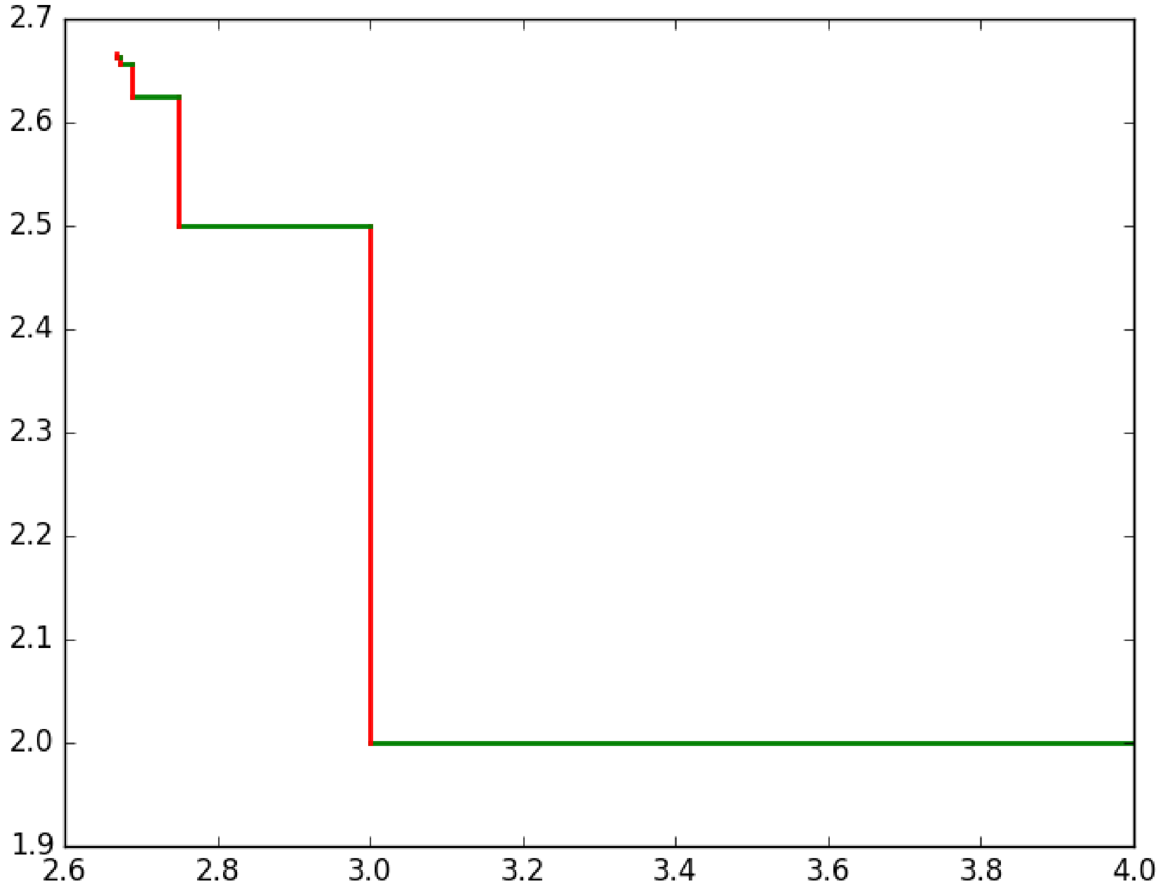
```
(4.0, 2.0)
(3.0, 2.5)
(2.75, 2.625)
(2.6875, 2.6563)
(2.6719, 2.6641)
(2.668, 2.666)
(2.667, 2.6665)
(2.6667, 2.6666)
(2.6667, 2.6667)
(2.6667, 2.6667)
(2.6667, 2.6667)
```

- Seems that no firm will unilaterally deviate from $(q_1^*, q_2^*) = \left(\frac{8}{3}, \frac{8}{3}\right)$.

Visualizing the search path (1.0)

```
import matplotlib.pyplot as plt

def plotEqUgly(q1, q2, a, c, n):
  for i in range(n): # for each iteration, draw two moves
    q1Cur = q1[i]
    q2Cur = q2[i]
    q1Next = q1[i + 1]
    q2Next = q2[i + 1]

    plt.plot([q1Cur, q1Next], [q2Cur, q2Cur], "g", linewidth = 2.0)
    plt.plot([q1Next, q1Next], [q2Cur, q2Next], "r", linewidth = 2.0)

  plt.show()
```

The search path



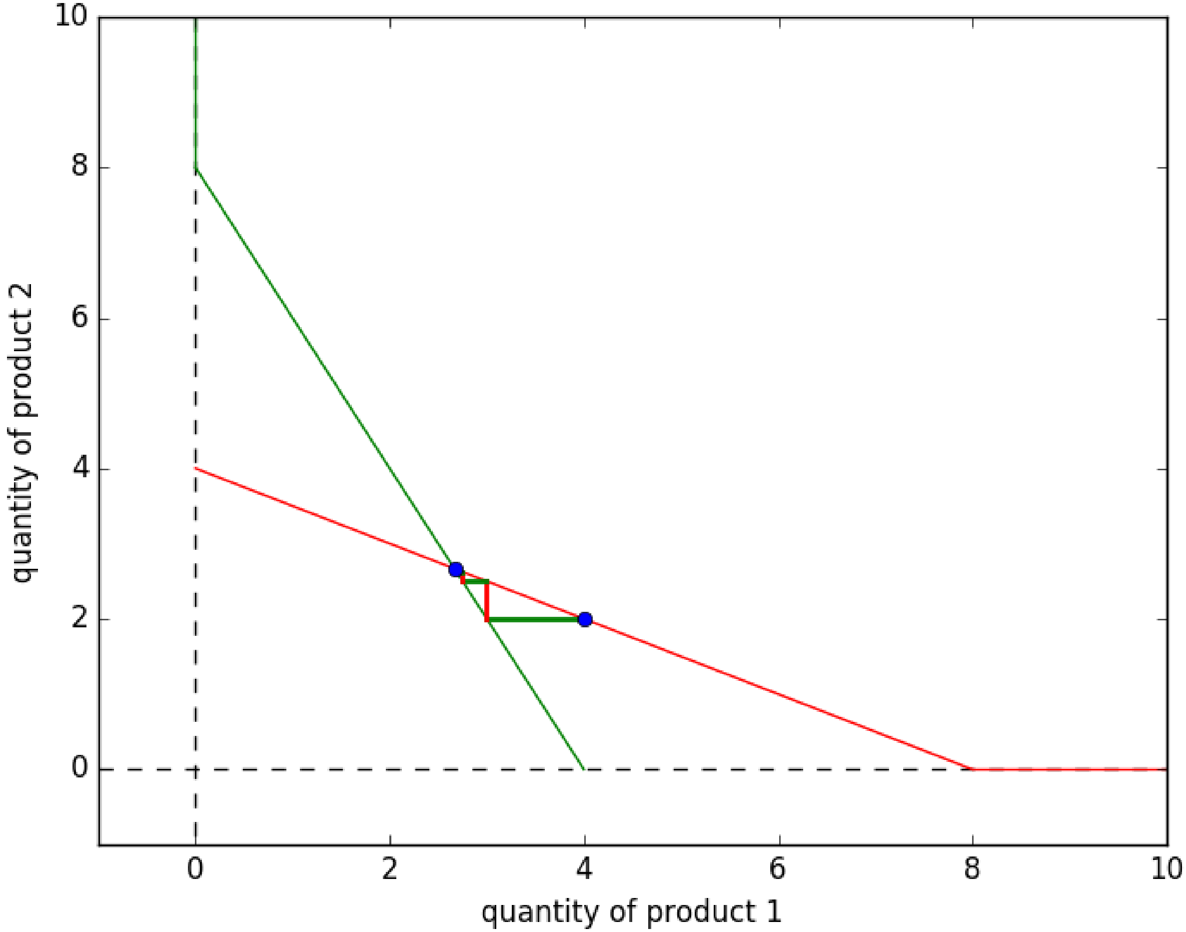
Visualizing the search path (2.0)

```
import matplotlib.pyplot as plt

def plotEq(q1, q2, a, c, n):
  ub1 = (a - c) / 2 * 2.5
  ub2 = (a - c) / 2 * 2.5
  ub = max(ub1, ub2)

  # axes
  plt.plot([0, 0], [ub, -1], "k--")
  plt.plot([-1, ub], [0, 0], "k--")

  # best response lines
  plt.plot([0, 0], [ub, a - c], "g")
  plt.plot([0, (a - c) / 2], [a - c, 0], "g")
  plt.plot([ub, a - c], [0, 0], "r")
  plt.plot([a - c, 0], [0, (a - c) / 2], "r")
```

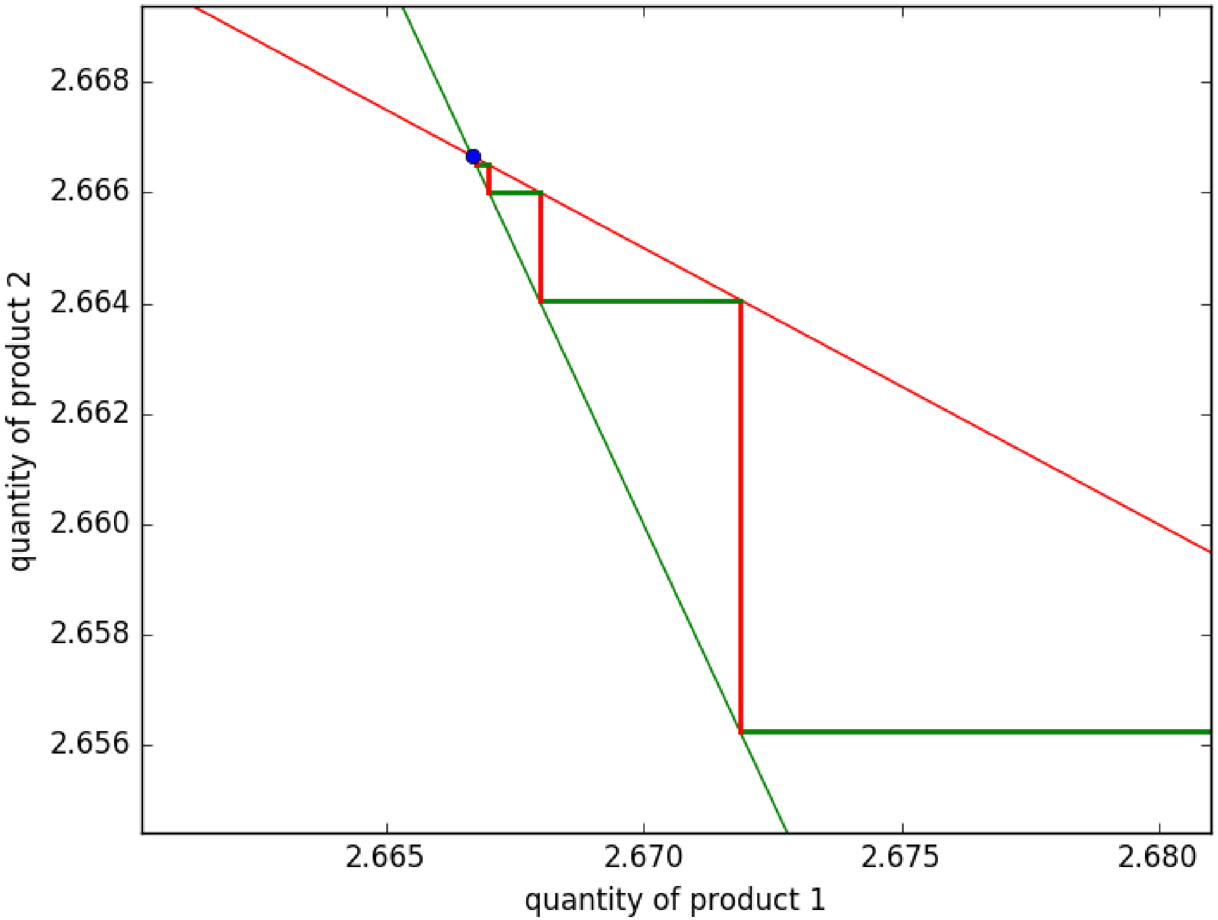
Visualizing the search path (2.0)

```
  for i in range(n): # for each iteration, draw two moves
    q1Cur = q1[i]
    q2Cur = q2[i]
    q1Next = q1[i + 1]
    q2Next = q2[i + 1]

    plt.plot([q1Cur, q1Next], [q2Cur, q2Cur], "g", linewidth = 2.0)
    plt.plot([q1Next, q1Next], [q2Cur, q2Next], "r", linewidth = 2.0)

# initial point and equilibrium point
q1Eq = (a + c - 2 * c) / 3
q2Eq = (a + c - 2 * c) / 3
plt.plot([q1[0]], [q2[0]], "bo")
plt.plot([q1Eq], [q2Eq], "bo")

plt.axis([-1, ub, -1, ub])
plt.xlabel('quantity of product 1')
plt.ylabel('quantity of product 2')
plt.show()
```

The search path



The search path (zooming in)



Solving for the equilibrium analytically

- In the previous example, the equilibrium is the **intersection of the best response functions**.
  - This is always true.
- Therefore, we may simply solve the linear system

$$q_1^* = \frac{10 - q_2^* - 2}{2} \quad \text{and} \quad q_2^* = \frac{10 - q_1^* - 2}{2}.$$

  - The unique solution is exactly $(q_1^*, q_2^*) = \left(\frac{8}{3}, \frac{8}{3}\right)$.
- In fact, even if we do not have the values of a and c:
  - The unique equilibrium is $(q_1^*, q_2^*) = \left(\frac{a-c}{3}, \frac{a-c}{3}\right)$.
  - This is how we **analytically** solve for an equilibrium.

Nash equilibrium in general

- The definition of Nash equilibrium implies that it locates at the **intersection of the best response functions**.
  - Otherwise, at least one player has the incentive to unilaterally deviate.
- To solve a game, all we need is to:
  - Find the best response functions (by solving each individual's utility maximization functions).
  - Find an intersection (or "the" intersection when it is unique).
- Keep mind that it is an outcome of **iterative best responses**.

Programming for Business Computing

Cournot Competition with Different Costs

Ling-Chieh Kung

Department of Information Management
National Taiwan University

The two approaches

- In the previous example (Cournot competition with identical costs), we applied two approaches to find the equilibrium.
  - Numerical: iterative best responses.
  - Analytical: intersection of best responses.
- The second way is simple and more informative.
  - An analytical solution $(q_1^*, q_2^*) = \left(\frac{a-c}{3}, \frac{a-c}{3}\right)$ provides managerial insights.
- However, in some cases an analytical solution is hard to obtain.
  - If not impossible.
  - A numerical solution is then helpful.

Cournot competition with different costs

- Consider Cournot competition again.
- Suppose that the production costs are now c_1 and c_2, which may be different.
- Firm 1's utility (profit) function is

$$u_1(q_1, q_2) = q_1[a - (q_1 + q_2) - c_1].$$

- Firm 2's utility (profit) function is

$$u_2(q_1, q_2) = q_2[a - (q_1 + q_2) - c_2].$$

Cournot competition with different costs

- Firm 1's best response becomes

$$q_1^* = \begin{cases} \dfrac{a - q_2 - c_1}{2} & \text{if } a - q_2 - c_1 > 0 \\ 0 & \text{otherwise} \end{cases}.$$

- Firm 2's best response becomes

$$q_2^* = \begin{cases} \dfrac{a - q_1 - c_2}{2} & \text{if } a - q_1 - c_2 > 0 \\ 0 & \text{otherwise} \end{cases}.$$

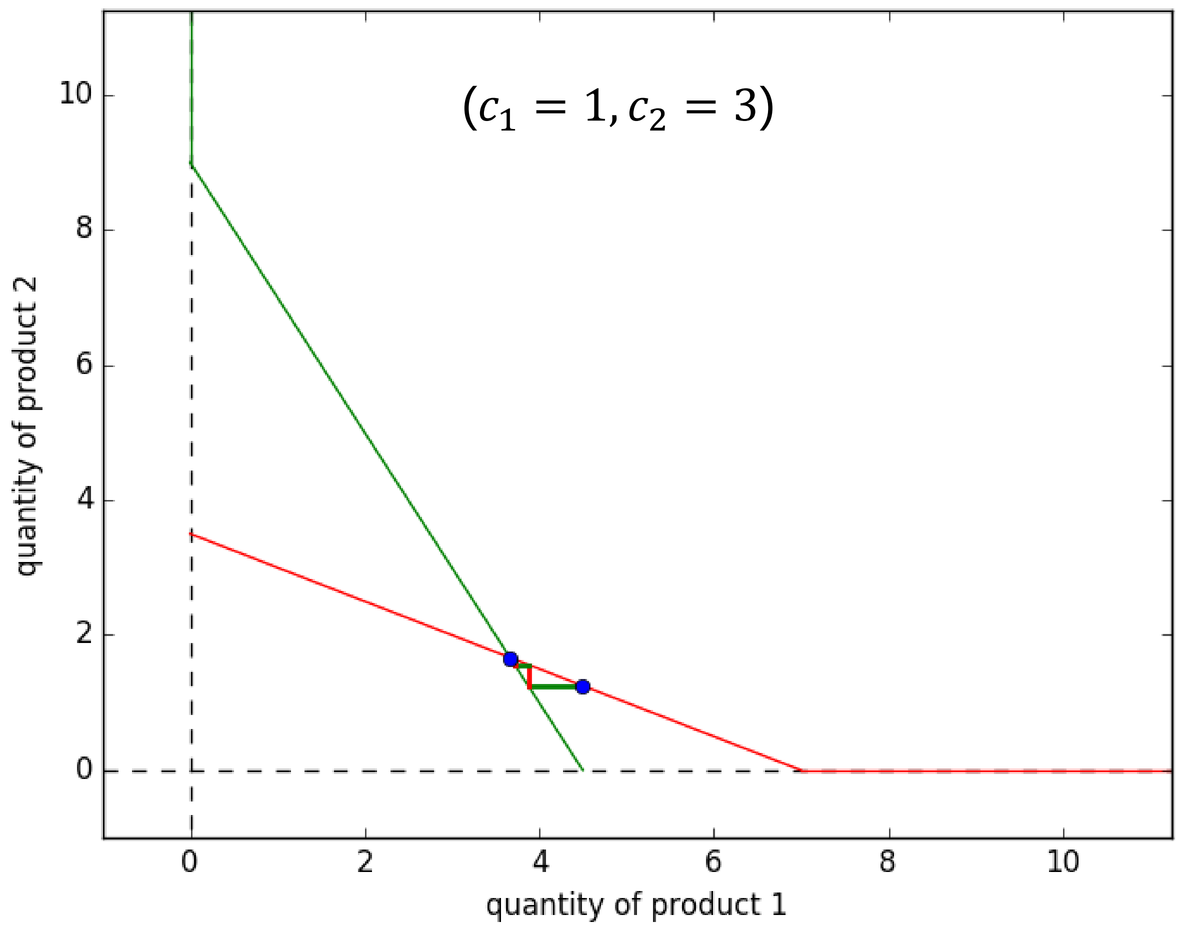
- How to analytically solve for an intersection?

Finding the equilibrium

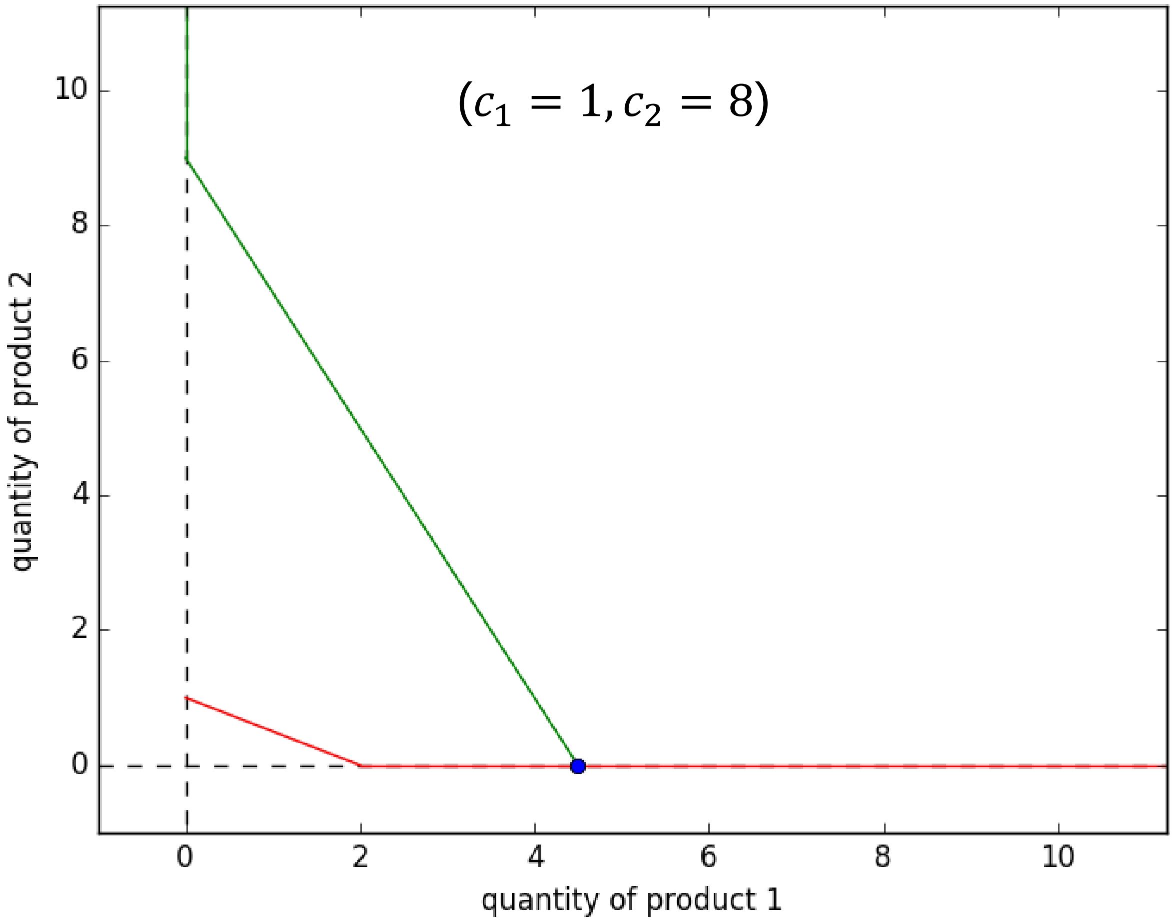
- Numerically getting an equilibrium is easy!

```
def CournotEq(a, c1, c2, n):
  # iteration 0: entering the market
  q1 = list()
  q1.append((a - c1) / 2)
  q2 = list()
  q2.append(max((a - q1[0] - c2) / 2, 0))

  # in each iteration, respond once
  for i in range(n):
    q1Next = max((a - q2[i] - c1) / 2, 0)
    q1.append(q1Next)
    q2Next = max((a - q1Next - c2) / 2, 0)
    q2.append(q2Next)

  return q1, q2
```

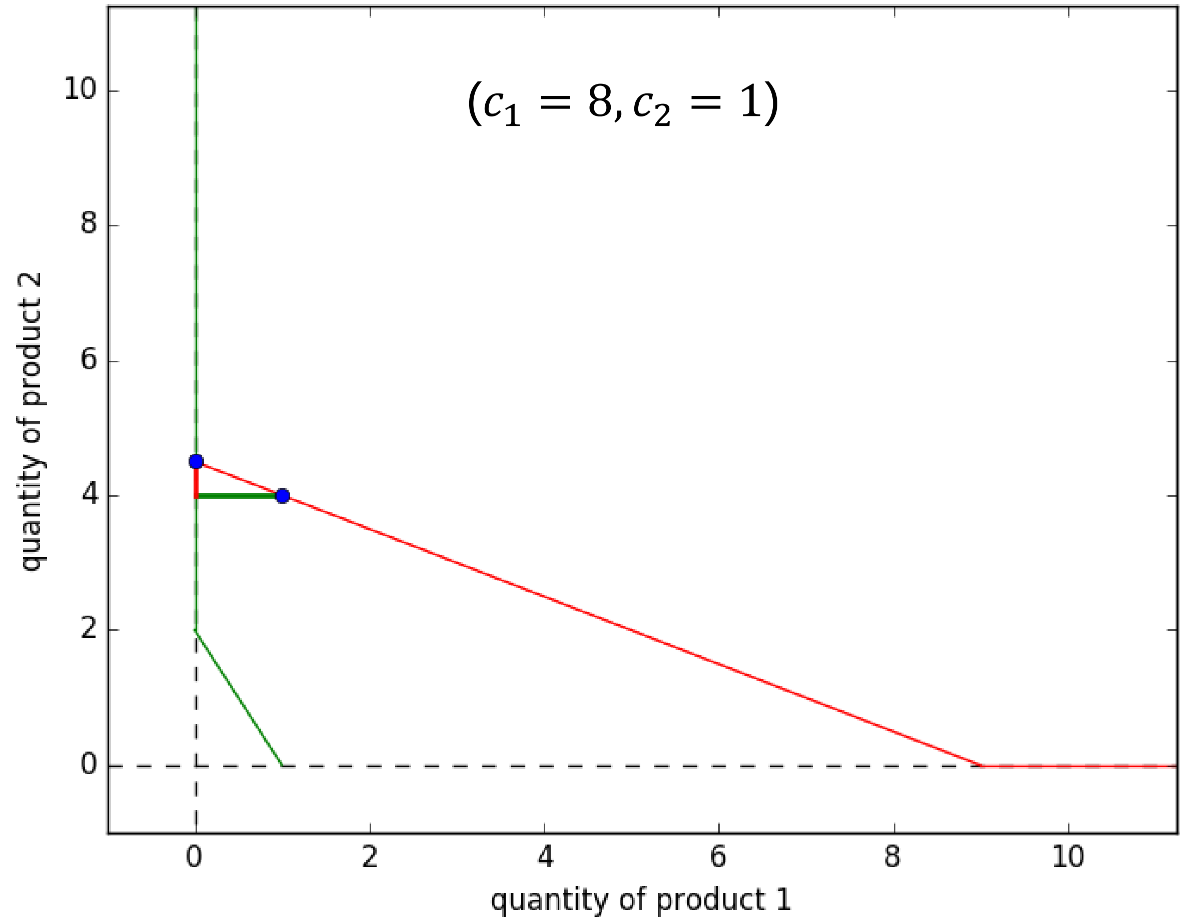
The search path



The search path



The search path



Impact of the costs

- How is the equilibrium affected by the costs?
- To do a demonstration, let's change c_2 from 1, 2, …, to 9 while fixing a to 10 and c_1 to 5.
  - For each pair of c_1 and c_2, we run 10 iterations to look for an equilibrium.
  - We then print out the nine equilibria to see the impact of c_2.

Impact of the costs

- The implementation and result:

```
1 (0.3333, 4.3333)
2 (0.6667, 3.6667)
3 (1.0, 3.0)
4 (1.3333, 2.3333)
5 (1.6667, 1.6667)
6 (2.0, 1.0)
7 (2.3333, 0.3333)
8 (2.5, 0)
9 (2.5, 0)
```

- When c_2 increases:
  - Firm 1 produces more.
  - Firm 2 produces less.

```
def CournotEqFinal_c2(a, c1, c2List, n):
  for c2 in c2List:
    # get the equilibrium
    q1, q2 = CournotEq(a, c1, c2, n)
    q1Eq = q1[n - 1]
    q2Eq = q2[n - 1]

    # print it out
    s = str(c2)
    s += " (" + str(round(q1Eq, 4))
    s += ", " + str(round(q2Eq, 4))
    s += ")"
    print(s)

a = 10.0
c1 = 5.0
n = 10
c2List = range(1, 10)
CournotEqFinal_c2(a, c1, c2List, n)
```

Summary

- Computer programming can help people solve problems in economics.
  – In particular, **analytically intractable** problems.
  – This is exactly the story of George Dantzig and the simplex method.
- This is also true in physics, chemistry, biology, etc.
- This is also true, of course, in business and management.