Python for Economists

Natalia Irena Gust-Bardon

June 2019

1. The Definite Integral. Consumers' Surplus

Example 1.1: Given the demand function $P = 46 - 3Q - Q^2$, and the equilibrium price $P_0 = 6$. Evaluate the consumers' surplus.

Step 1: Solve the equation to find Q_0:

$$46 - 3Q - Q^2 = 6$$

```
>>> from sympy import Symbol
>>> import sympy as sp
>>> from sympy import solve
>>> Q = Symbol('Q')
>>> demand_function = 46 - 3*Q - Q**2
>>> P0 = 6
>>> equilibrium=demand_function - P0
>>> sols=sp.solve(equilibrum, dict=True)
```

Step 2: Out of two roots consider only the positive one (P_0):

```
>>> for x in sols:
>>>   if x[Q] > 0:
>>>         res = x[Q]
```

Step 3: Evaluate the consumers' surplus accordingly:

Consumers' surplus: $\int_0^{Q_0} f(Q)dQ - Q_0P_0$

```
>>> consumers_surplus = sp.integrate(demand_function, (Q, 0, res)) - res * P0
```

Step 4: Print consumers' surplus

```
>>> print('Consumers\' surplus equals:', float(consumers_surplus))
```

Example 1.2: Based on Example 1.1 prepare a plot:

Step 1: Plot the demand function and two break red lines of P_0 and Q_0

```
>>> from pylab import plot, show
>>> from sympy.utilities.lambdify import lambdify
>>> from sympy import Symbol
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> Q = Symbol('Q')
>>> demand_function = 46 - 3*Q - Q**2
>>> evaldemand_function = lambdify(Q, demand_function, modules=['numpy'])
>>> plt.plot((0, 5.5), (P0, P0), color='r', linestyle='--')
>>> plt.plot((res, res), (15, 0), color='r', linestyle='--')
>>> t = np.linspace(0, P0, 100)
>>> plt.plot(t, evaldemand_function(t), color='b', linestyle='-',
label='demand function f(Q)')
```

Step 2: Fill the area between: $f(Q), P_0, Q_0$

```
>>> ht = np.linspace(0, 5, 100)
```

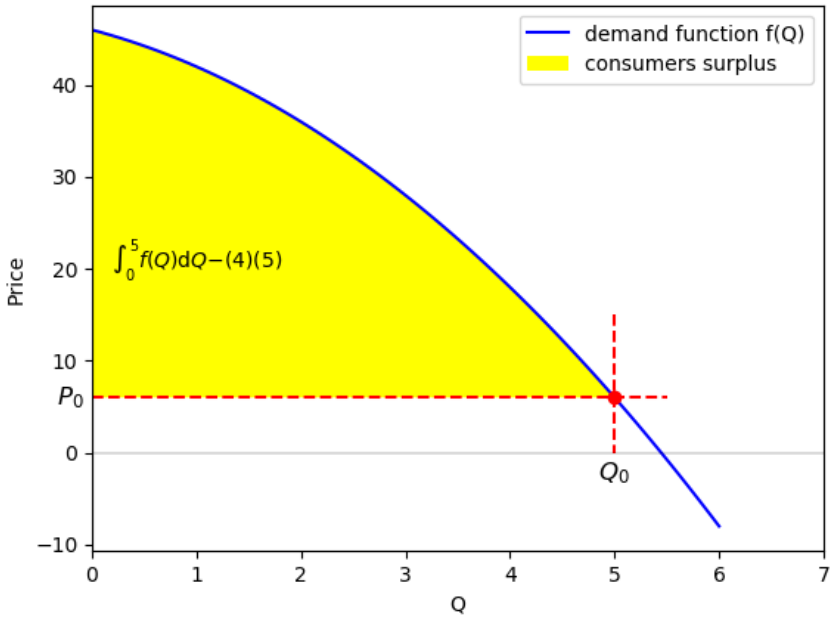
```
>>> plt.fill_between(ht, 6, evaldemand_function(ht), facecolor='yellow',
label='consumers surplus')
```

Step 3: Add additional elements

```
>>> plt.text(1, 20, r"$\int_0^5 f(Q)\mathrm{d}Q$$-(4)(5)$",
          horizontalalignment='center', fontsize=10)
>>> plt.text(-0.2, 6, r"$P_0$", horizontalalignment='center',
verticalalignment='center', fontsize=12)
>>> plt.text(5, -2.2, r"$Q_0$",
horizontalalignment='center',verticalalignment='center', fontsize=12)
>>> plt.legend(loc='best')
>>> plt.axhline(y=0, color='k', linewidth=0.2)
>>> plt.axvline(x=0, color='k')
>>> plt.xlim(0, 7)
>>> data_x= [5]
>>> data_y= [6]
>>> plt.plot(data_x, data_y, 'or')
>>> plt.xlabel('Q')
>>> plt.ylabel('Price')
```

Step 4: Print the plot

```
>>> plt.show()
```



2. The Definite Integral. Producers' Surplus

Example 2.1: Given the supply function $P = Q^2 + Q + 10$, and the equilibrium price $P_0 = 30$. Evaluate the producers' surplus.

Step 1: Solve the equation to find Q_0:

$$Q^2 + Q + 10 = 30$$

```
>>> from sympy import Symbol
>>> import sympy as sp
>>> from sympy import solve
>>> Q = Symbol('Q')
>>> supply_function = Q**2+Q+10
>>> P0 = 30
>>> equilibrium=supply_function - P0
>>> sols=sp.solve(equilibrum, dict=True)
```

Step 2: Out of two roots consider only the positive one (P_0):

```
>>> for x in sols:
>>>   if x[Q] > 0:
>>>         res = x[Q]
>>> print('Q0 equals:', float(res))
```

Step 3: Evaluate the producers' surplus accordingly:

Producers' surplus: $Q_0 P_0 - \int_0^{Q_0} f(Q)dQ$

```
>>> producers_surplus = res * P0 - sp.integrate(supply_function, (Q, 0, res))
```

Step 4: Print consumers' surplus

```
>>> print('Producers\' surplus equals:', float(producers_surplus))
```

Example 2.2: Based on Example 2.1 prepare a plot:

Step 1: Plot the demand function and two break red lines of P_0 and Q_0

```
>>> from pylab import plot, show
>>> from sympy.utilities.lambdify import lambdify
>>> from sympy import Symbol
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> Q = Symbol('Q')
>>> supply_function = Q**2+Q+10
>>> evalsupply_function = lambdify(Q, supply_function, modules=['numpy'])
>>> plt.plot((0, 4.5), (P0, P0), color='r', linestyle='--')
>>> plt.plot((res, res), (35, 0), color='r', linestyle='--')
>>> t = np.linspace(0, P0, 100)
>>> plt.plot(t, evalsupply_function(t), color='b', linestyle='-',
label='demand function f(Q)')
```

Step 2: Fill the area between: $f(Q), P_0, Q_0$

```
>>> ht = np.linspace(0, 4, 100)
```

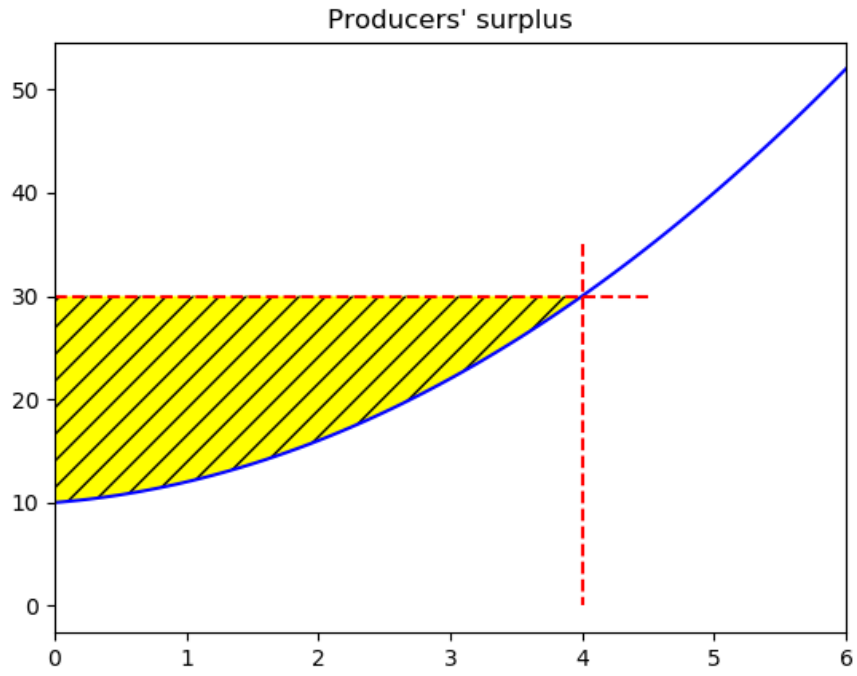
```
>>> plt.fill_between(ht, P0, evalsupply_function(ht), facecolor='yellow',
label='producers surplus’,  hatch=“//")
```

Step 3: Add additional elements

```
>>> from pylab import title
>>> title('Producers\' surplus')
>>> plt.xlim(0, 6)
>>> plt.ylim(0, 55)
```

Step 4: Print the plot

```
>>> plt.show()
```



3. The Definite Integral. Cost, revenue and profit functions

Example 3.1: Given the total cost function $C = Q^3 - 10Q^2 + 30Q + 3550$, and the total revenue function $R = 600Q$. Evaluate the producers' profit.

Step 1: Solve the equation to find Q_1 and Q_2:

$$Profit = R - C$$

$$Profit = 600Q - Q^3 + 10Q^2 - 30Q - 3550$$

```
>>> from sympy import Symbol
>>> import sympy as sp
>>> from sympy import solve, re
>>> Q = Symbol('Q')
>>> total_cost_function = 3550 + 30*Q - 10*Q**2 + Q**3
>>> total_revenue_function = 600*Q
>>> profit = total_cost_function - total_revenue_function
>>> sols=sp.solve(profit)
```

Step 2: Consider only the positive roots:

```
>>> endpoints = [ ]
>>> for x in sols:
>>>   endpoint = float(re(x))
>>>   if endpoint >=0:
>>>         endpoints.append(endpoint)
>>> ep1, ep2 = sorted(endpoints)
```

Step 3: Evaluate the profit:

```
>>>profit = sp.integrate(total_revenue_function, (Q, ep1, ep2)) -
sp.integrate(total_cost_function, (Q, ep1, ep2))
>>> print('Profit equals:', '%.2f' % (profit))
```

Step 4: Maximize the profit:

Example 3.2: Based on Example 3.1 prepare a plot:

Step 1: Plot the cost and revenue functions:

```
>>> from pylab import plot, show
>>> from sympy.utilities.lambdify import lambdify
>>> from sympy import Symbol
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> Q = Symbol('Q')
>>> total_cost_function = 3550 + 30*Q - 10*Q**2 + Q**3
>>> evaltotal_cost_function = lambdify(Q, total_cost_function,
modules=['numpy'])
>>> evaltotal_revenue_function = lambdify(Q, total_revenue_function,
modules=[‘numpy’])
>>> t = np.linspace(0, 30, 100)
>>> plt.plot(t, evaltotal_cost_function(t), color='r', label='Total cost')
>>> plt.plot(t, evaltotal_revenue_function(t), color='b', label='Total
revenue')
```

Step 2: Fill the area between the revenue and cost functions:

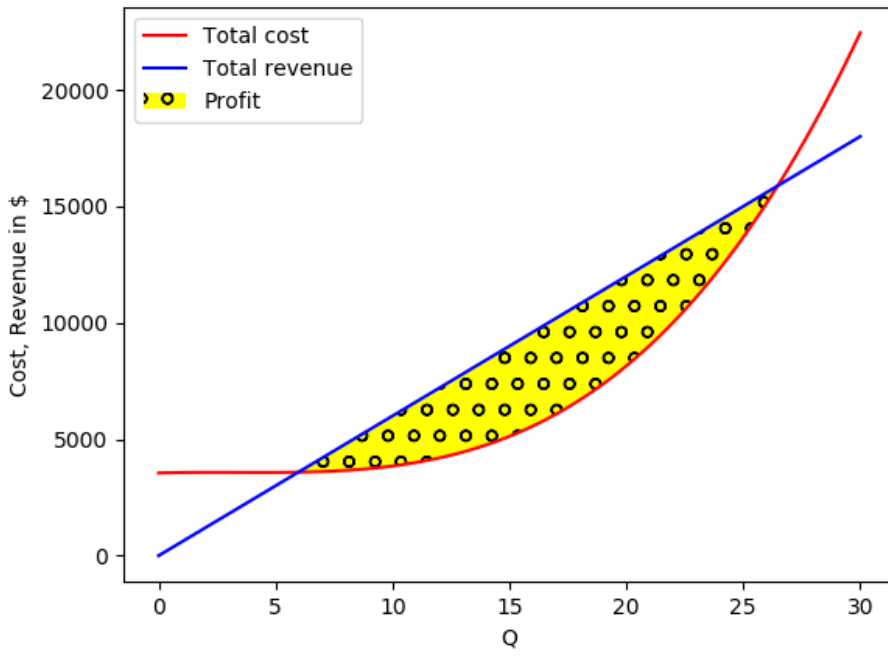
```
>>> plt.fill_between( t, evaltotal_cost_function(t),
evaltotal_revenue_function(t),
    where=evaltotal_cost_function(t) < evaltotal_revenue_function(t),
facecolor='yellow', hatch="o", label='Profit')
```

Step 3: Add labels and the legend:

```
>>> plt.xlabel('Q')
>>> plt.ylabel('Cost, Revenue in $')
>>> plt.legend(loc='best')
```

Step 4: Print the plot:

```
>>> plt.show()
```



4. The Derivative. Optimizing Economic Functions

Example 4.1: Given the total cost and revenue functions from Example 3.1, maximize the profit for a firm.

Step 1: Set up the profit function and take the first derivative:

$$Profit = R - C$$

$$Profit = 600Q - Q^3 + 10Q^2 - 30Q - 3550$$

```
>>> from sympy import Symbol
>>> import sympy as sp
>>> Q = Symbol('Q')
>>> total_cost_function = 3550 + 30*Q - 10*Q**2 + Q**3
>>> total_revenue_function = 600*Q
>>> profit = total_revenue_function - total_cost_function
>>> expr1=sp.diff(profit,Q)
```

Step 2: Take the first derivative of the profit function and find the critical points. Take only the positive critical point:

$$Profit' : -3Q^2 + 20Q + 570 = 0$$

```
>>> sols = sp.solve(expr1)
>>> for x in sols:
>>>    if x > 0:
>>>        res = x
>>> k = profit.subs(Q, res)
>>> print('Profit is maximized at Q =', '%.2f' % float(res), 'where profit
equals', '%.2f' % float(k))
```

Example 4.2: To the plot presented in Example 3.2 add tangent to TC that is parallel to TR:

Note: Geometrically, the maximized profit happens when the difference between the total revenue function (TR) and the total cost function (TC) is maximized —it occurs when a tangent to TR and TC curve are parallel.

The equation of the tangent line is given by: $y = a(Q - Q_0) + y_0$.

Step 1: Find a — slope of the tangent. Since we want the tangent line to be parallel to the line of total revenue, we need for it to have the same slope. Hence, a is 600.

```
>>> from sympy import Poly
>>> p = Poly(total_revenue_function, Q)
>>> a = p.coeff_monomial(Q)
```

Step 2: Q_0 is 'res' from Example 4.1., To find y_0, solve TC(Q_0):

```
>>> y = total_cost_function.subs(Q, res)
```

Step 3: Create the tangent line and add additional elements:

```
>>> tangent = a*Q - a*res + y
>>> evaltotal_cost_function = lambdify(Q, total_cost_function,
modules=['numpy'])
```

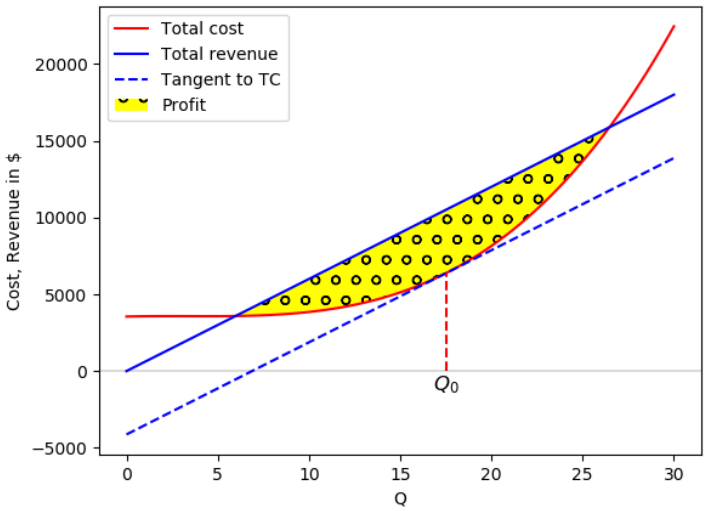
```
>>> evaltotal_revenue_function = lambdify(Q, total_revenue_function,
modules=[‘numpy'])
>>> evaltangent = lambdify(Q, tangent, modules=['numpy'])
>>> t = np.linspace(0, 30, 100)
>>> plt.plot(t, evaltotal_cost_function(t), color='r', label='Total
cost')
>>> plt.plot(t, evaltotal_revenue_function(t), color='b', label='Total
revenue')
>>> plt.plot(t, evaltangent(t), color='b', linestyle='--',
label='Tangent to TC')
>>> plt.fill_between(

t, evaltotal_cost_function(t), evaltotal_revenue_function(t),
where=evaltotal_cost_function(t) < evaltotal_revenue_function(t),
facecolor='yellow', hatch="o", label='Profit')

>>> plt.plot((res, res), (0, y), color='r', linestyle='--')
>>> plt.axhline(y=0, color='k', linewidth=0.2)
>>> plt.text(res, -900, r"$Q_0$", horizontalalignment='center',
verticalalignment='center', fontsize=12)

>>> plt.xlabel('Q') plt.ylabel('Cost, Revenue in $')
plt.legend(loc='best')

>>> plt.show()
```



5. Matrices. Constrained Optimization

Example 4.1: What combination of goods x and y should a firm produce to minimize costs when the joint cost function is $c = 5x^2 + 8y^2 - xy + 50$, and the firm has a production quota of $x + y = 28$?

Step 1: Prepare the Lagrangian function and find the first-order partial derivatives accordingly:

$$c = 5x^2 + 8y^2 - xy + 50 + \lambda(28 - x - y)$$

$$c_x = 10x - y - \lambda$$

$$c_y = x + 16y - \lambda$$

$$c_\lambda = -x - y - 28$$

```
>>> from sympy import Poly
>>> from sympy.abc import x, y, A
>>> import sympy as sp
>>> import numpy as np
>>> lagrangian_function=5*x**2 + 8*y**2 - x*y + 50 + A*28 - A*x - A*y
>>> expr1=sp.diff(lagrangian_function, x)
>>> expr2=sp.diff(lagrangian_function, y)
>>> expr3=sp.diff(lagrangian_function, A)
```

Step 2: Find matrix C and matrix B form of the equations, according to the formula:

$$CX = B$$

$$\begin{bmatrix} 10 & -1 & -1 \\ 1 & 16 & -1 \\ -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -28 \end{bmatrix}$$

And solve $X = C^{-1}B$.

a) Identify coefficients for matrix C and matrix B

```
>>> p = Poly(expr1, x, y, A)
>>> k = Poly(expr2, x, y, A)
>>> s = Poly(expr3, x, y, A)
>>> expr1_x = p.coeff_monomial(x)
>>> expr1_y = p.coeff_monomial(y)
>>> expr1_A = p.coeff_monomial(A)
>>> expr2_x = k.coeff_monomial(x)
>>> expr2_y = k.coeff_monomial(y)
>>> expr2_A = k.coeff_monomial(A)
>>> expr3_x = s.coeff_monomial(x)
>>> expr3_y = s.coeff_monomial(y)
>>> expr3_A = s.coeff_monomial(A)
```

a) Assign coefficients to matrix C and matrix B

```
>>> C = np.array([
```

```
                    [float(expr1_x), float(expr1_y), float(expr1_A)],
                    [float(expr2_x), float(expr2_y), float(expr2_A)],
                    [float(expr3_x), float(expr3_y), float(expr3_A)]
                                                            ])
>>> B = np.array([-float(p.nth(0, 0, 0)), -float(k.nth(0, 0, 0)), -
float(s.nth(0, 0, 0))])
```

Step 3: Solve the equations:

```
>>> sols = np.linalg.solve(C, B)
```

The solution will be: $\bar{x} = 17, \bar{y} = 11, \lambda = 159$

Step 4: Evaluate the cost, while $\bar{x} = 17, \bar{y} = 11$:

$$c(17,11) = 5x^2 + 8y^2 - xy + 50$$

```
>>> x_good, y_good, A_lambda = sols
>>> cost_function=5*x**2 + 8*y**2 + x*y + 50
>>> cost_min=cost_function.subs([(x, x_good), (y, y_good)])
```

Step 5: Print the solution:

```
>>> print('The following combination of goods minimizes the costs: x=',
x_good, 'y=', y_good)
>>> print('This combination of goods will minimize costs to the level of:',
'%.2f' % cost_min)
>>> print('Interpretation of lambda: a decrease in the production quota will
lead to a cost reduction of', A_lambda)
```

6. Polynomial inequalities

Example 6.1: Given the total cost function $C = Q^3 - 10Q^2 + 30Q + 3550$, a firm wants to assess how many products can be produced not exceeding the total cost of 10,000 USD. At the same time, the company is willing to invest at least 6,000 USD. Find the range of the production within the constraint:

$$6{,}600 < C < 10{,}000$$

Step 1: Solve two inequalities: $6{,}600 < C$, $C < 10{,}000$:

```
>>>from sympy import Poly, Symbol, solve_poly_inequality
>>> Q=Symbol('Q')
>>> cost=Q**3-10*Q**2+30*Q+3550
>>> constraint_max=10000
>>> constraint_min=6000
>>> cost_max = cost - constraint_max
>>> cost_min = cost - constraint_min
>>> p = Poly(cost_max, Q)
>>> z = Poly(cost_min, Q)
>>> k = solve_poly_inequality(p, '<')
>>> s = solve_poly_inequality(z, '>')
```

Step 2: Find the intersection of the solution of the inequalities:

```
>>> x_1 = s[0].evalf()
>>> x_2 = k[0].evalf()
>>> production = x_1.intersect(x_2)
>>> x, y = production.boundary
```

Step 3: Print the solution:

```
>>> print('Within the given constraints, the production should be in the
following range:', '%.2f' % x, '< Q <', '%.2f'% y)
```

Example 6.2: Based on Example 6.1 prepare a plot:

Step 1: Plot the cost function

```
>>> from sympy.utilities.lambdify import lambdify
>>> from sympy import Symbol
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> Q=Symbol('Q')
>>> cost=Q**3-10*Q**2+30*Q+3550
>>> evalcost_function = lambdify(Q, cost, modules=['numpy'])
>>> t = np.linspace(0, 25, 100)
>>> plt.plot(t, evalcost_function(t), color='r', label='Cost function')
```

Step 2: Plot the other lines

```
>>> plt.axhline(y=constraint_min, color='k', linewidth=0.05)
>>> plt.axhline(y=constraint_max, color='k', linewidth=0.05)
>>> plt.axhline(y=3000, color='k', linewidth=0.2)
>>> plt.plot((x, x), (3000, constraint_min), color='k', linewidth=0.5,
linestyle='--')
```

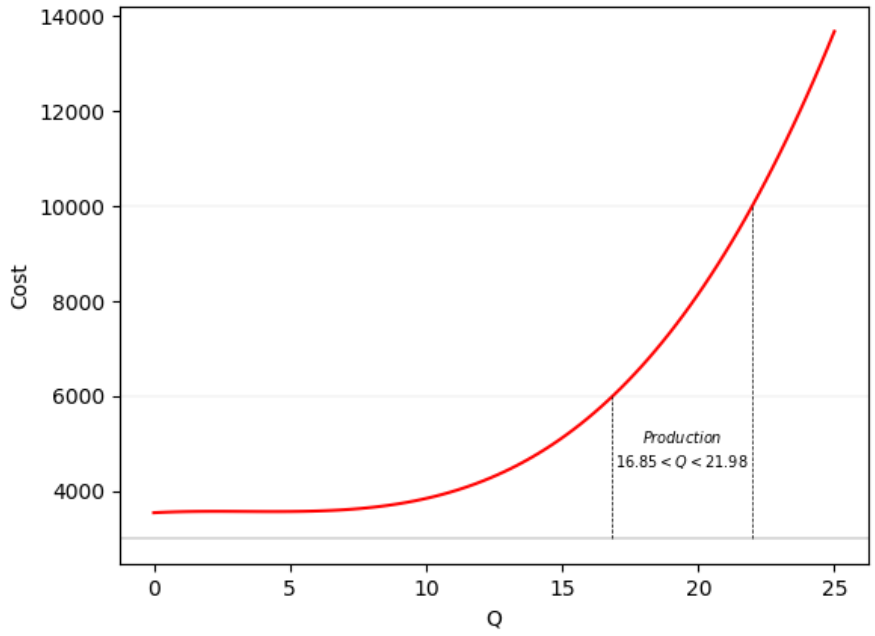
```
>>> plt.plot((y, y), (3000, constraint_max), color='k', linewidth=0.5,
linestyle='--')
```

Step 3: Add additional elements:

```
>>> plt.text(19.4, 4500, r"$16.85<Q<21.98$", horizontalalignment='center',
fontsize=7)
>>> plt.text(19.4, 5000, r"$Production$", horizontalalignment='center',
fontsize=7)
>>> plt.xlabel('Q')
>>> plt.ylabel('Cost')
```

Step 4: Print the plot

```
>>> plt.show()
```



7. Matrices. Input-output analysis

Example 7.1: A town has a merchant, a baker and a framer. To produce $1 of output the merchant requires $0.10 worth of its own goods, $0.20 worth of baked goods and $0.30 worth of the farmer's products. To produce $1 of output the baker requires $0.20 worth of goods from the merchant, $0.10 worth of its own goods and $0.20 worth of goods from the farmer. To produce $1 of output the farmer requires $0.10 worth of goods from the merchant, $0.50 worth products from the backer and $0.30 worth of its own goods. How much should each produce to meet consumer demand of $10,000 worth of output from the merchant, $15,000 worth of goods from the backer and $13,000 worth of goods from the farmer?

Step 1: Prepare the matrix of technical coefficients (M) and the demand matrix (D) , according to:

$$X = 0.1x + 0.2y + 0.3z + 10000$$
$$Y = 0.2x + 0.1y + 0.2z + 15000$$
$$Z = 0.1x + 0.5y + 0.3z + 13000$$

$$\text{M} = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.2 & 0.1 & 0.2 \\ 0.1 & 0.5 & 0.3 \end{bmatrix} \quad \text{D} = \begin{bmatrix} 10000 \\ 15000 \\ 13000 \end{bmatrix}$$

```
>>> from sympy import Poly
>>> from sympy.abc import x, y, z
>>> import numpy as np
>>> X_function = 0.1*x+0.2*y+0.3*z+10000
>>> Y_function = 0.2*x +0.1*y+0.2*z+15000
>>> Z_function = 0.1*x+0.5*y+0.3*z+13000
>>> p = Poly(X_function, x, y, z)
>>> k = Poly(Y_function, x, y, z)
>>> s = Poly(Z_function, x, y, z)
>>> X_function_x = p.coeff_monomial(x)
>>> X_function_y = p.coeff_monomial(y)
>>> X_function_z = p.coeff_monomial(z)
>>> Y_function_x = k.coeff_monomial(x)
>>> Y_function_y = k.coeff_monomial(y)
>>> Y_function_z = k.coeff_monomial(z)
>>> Z_function_x = s.coeff_monomial(x)
>>> Z_function_y = s.coeff_monomial(y)
>>> Z_function_z = s.coeff_monomial(z)
>>> M = np.array([
    [float(X_function_x), float(X_function_y), float(X_function_z)],
    [float(Y_function_x), float(Y_function_y), float(Y_function_z)],
    [float(Z_function_x), float(Z_function_y), float(Z_function_z)]
    ])
>>> D = np.array([float(p.nth(0, 0, 0)), float(k.nth(0, 0, 0)),
float(s.nth(0, 0, 0))])
```

Step 2: Solve the equations, according to: $X = MX + D$ => $(I - M)X = D$.

a) solve: $I - M$

```
>>> I = np.identity(3)
>>> matrix = I - M
```

b) solve the equation using `np.linalg.solve`:

```
>>> sols = np.linalg.solve(matrix, D)
>>> print('To meet both the internal and consumer demands, the
merchant must have an output of $', sols[0], 'the backer must have an
output of $', sols[1], 'and the farmer must have an output of $',
sols[2])
```