Mathematical Economics

Alpha Chiang

Chapter 3

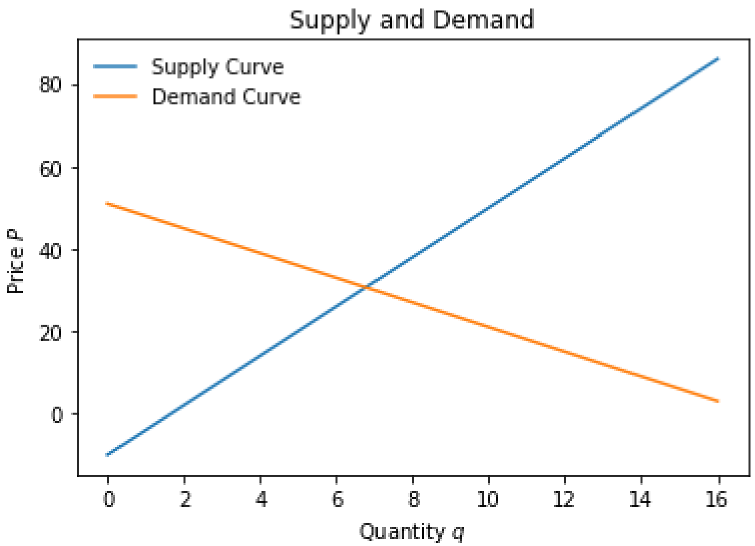
3.2 Partial Market Equilibrium—A Linear Model

In [23]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import sympy as sy
```

In [24]:
```
def S(P,c=10,d=6):
    return (-c + d*P)

def D(P,a=51,b=3):
    return (a - b*P)

P = np.linspace(0, 16, 1000)
```

In [25]:
```
plt.plot(P, S(P), label = "Supply Curve")
plt.plot(P, D(P), label = "Demand Curve")
plt.title("Supply and Demand")
plt.legend(frameon = False)
plt.xlabel("Quantity $q$")
plt.ylabel("Price $P$")
```

Out[25]:
```
Text(0, 0.5, 'Price $P$')
```



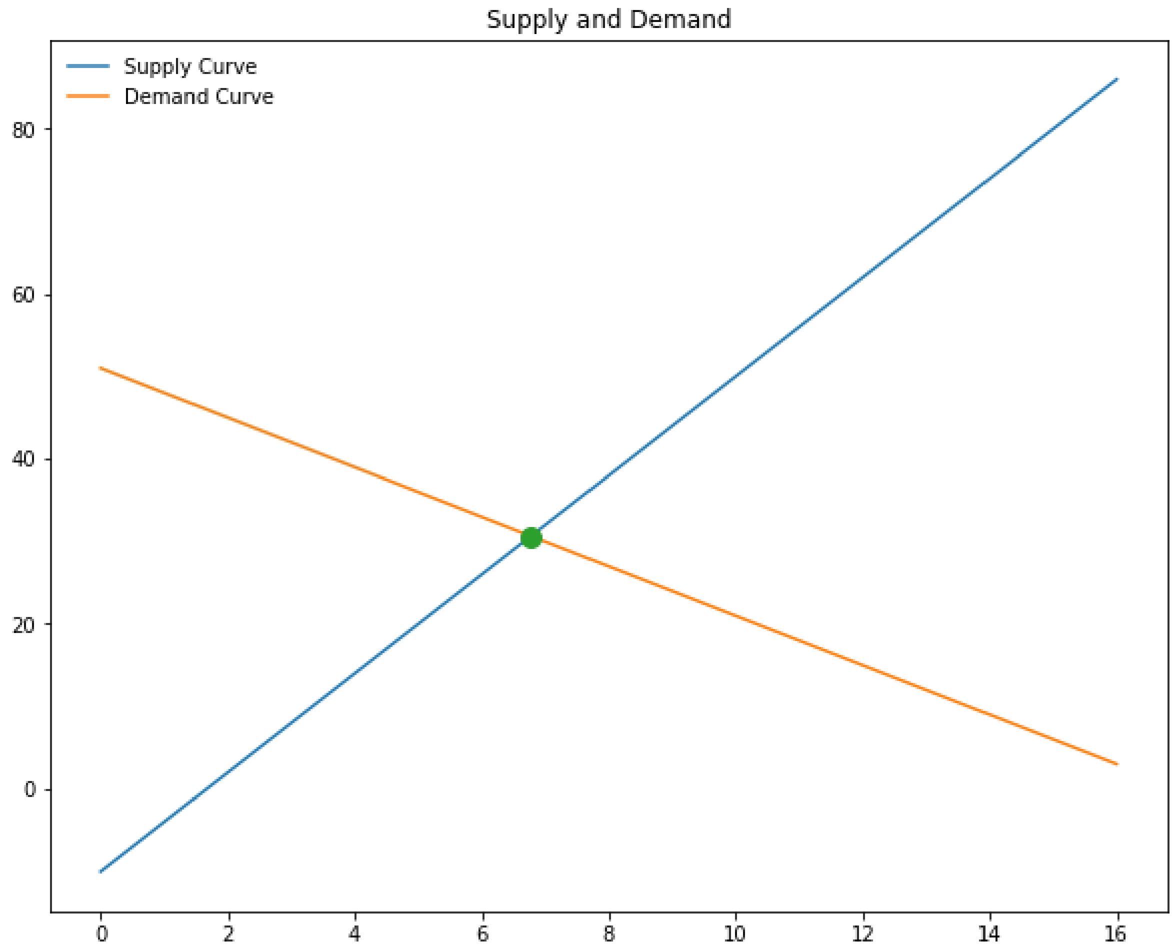
In [26]:
```
P = sy.Symbol('P')
eq = sy.Eq(S(P), D(P))
display(sy.solve(eq))
display(S(61/9))
```

```
[61/9]
30.666666666666664
```

By using https://calculus-notes.readthedocs.io/en/latest/0.8_consumer_surplus.html

In [27]:
```
plt.figure(figsize= (10, 8))
P = np.linspace(0, 16, 1000)


plt.plot(P, S(P), label = "Supply Curve")
plt.plot(P, D(P), label = "Demand Curve")
plt.plot(61/9, 30.666666666666664, 'o', markersize = 10)


plt.title("Supply and Demand")
plt.legend(frameon = False)



ax.annotate('Equilibrium at (61/9,30.666666666666664)',
    xy=(61/9,30.666666666666664),xytext=(61/9,30.666666666666664))
```

Out[27]: Text(6.777777777777778, 30.666666666666664, 'Equilibrium at (61/9,30.666666666666664)')



In [28]:
```
from sympy import symbols, Eq, solve
```

Solution by Elimination of Variables

In [10]:
```
P = sy.Symbol('P')
a = sy.Symbol('a')
b = sy.Symbol('b')
c = sy.Symbol('c')
d = sy.Symbol('d')
Q = sy.Symbol('Q')
eq1 = Eq(a + b*P,Q)
```

```
eq2 = Eq(-c + d*P,Q)
solve((eq1,eq2), (P,Q))
```

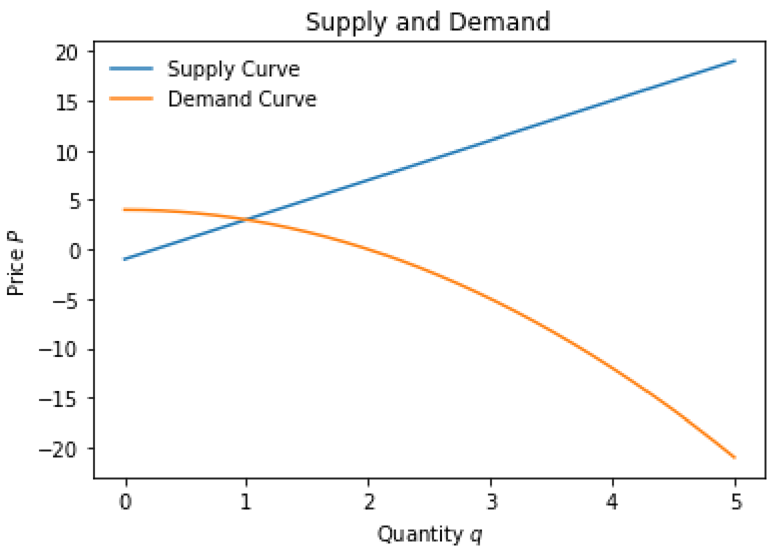
Out[10]: {Q: -(a*d + b*c)/(b - d), P: -(a + c)/(b - d)}

3.3 Partial Market Equilibrium—A Nonlinear ModeL

In [11]:
```
def S(P,c=1,d=4):
    return (-c + d*P)

def D(P,a=4,b=1):
    return (a - b*P**2)

P = np.linspace(0, 5, 1000)
```

In [12]:
```
plt.plot(P, S(P), label = "Supply Curve")
plt.plot(P, D(P), label = "Demand Curve")
plt.title("Supply and Demand")
plt.legend(frameon = False)
plt.xlabel("Quantity $q$")
plt.ylabel("Price $P$")
```

Out[12]: Text(0, 0.5, 'Price P')



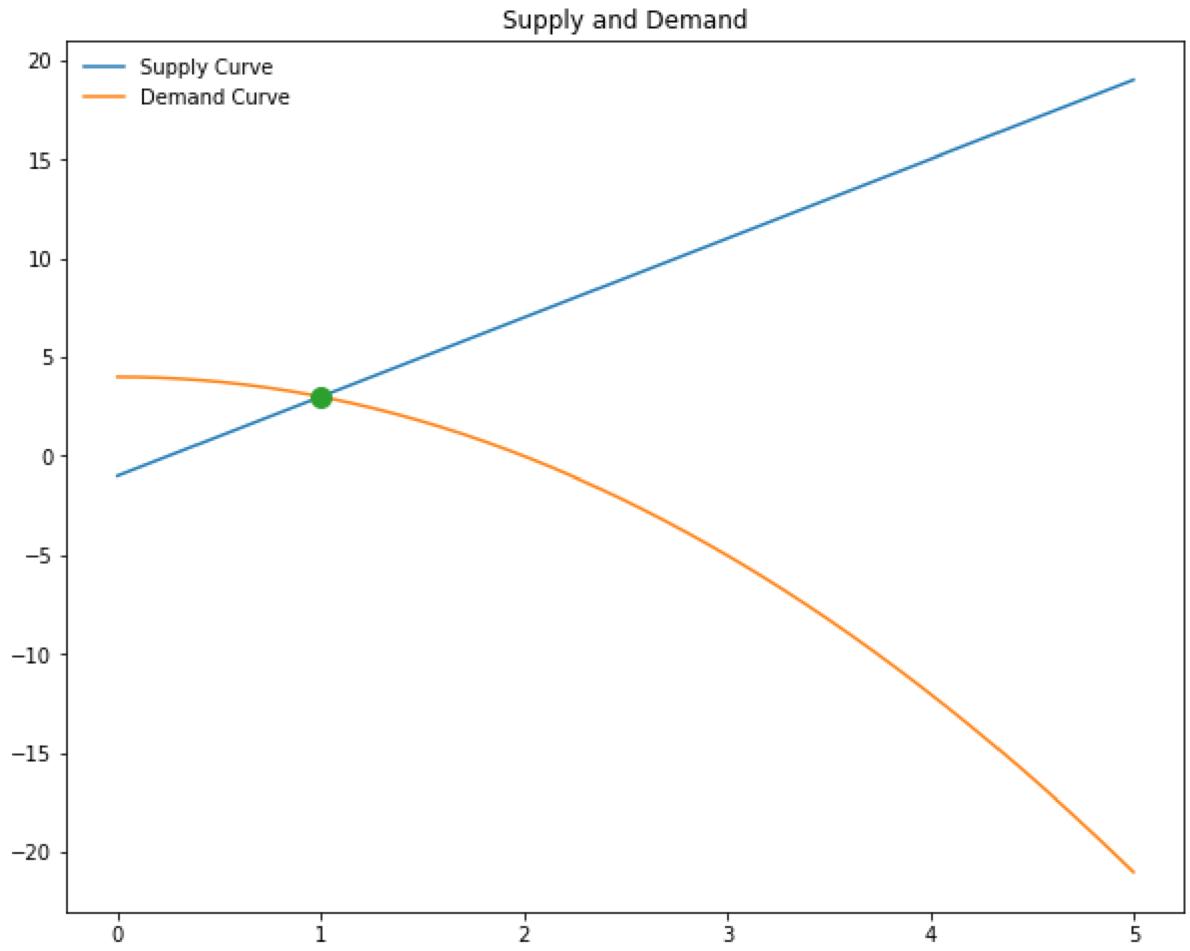
In [13]:
```
P = sy.Symbol('P')
```

```python
eq = sy.Eq(S(P), D(P))
display(sy.solve(eq))
display(S(1)) #We use 1 since price cannot be negative
```

```
[-5, 1]
3
```

In [14]:
```python
plt.figure(figsize= (10, 8))
P = np.linspace(0, 5, 1000)


plt.plot(P, S(P), label = "Supply Curve")
plt.plot(P, D(P), label = "Demand Curve")
plt.plot(1,3, 'o', markersize = 10)


plt.title("Supply and Demand")
plt.legend(frameon = False)




ax.annotate('Equilibrium at (61/9,30.666666666666664)',
            xy=(1,3),xytext=(1,3 ),  arrowprops=dict(facecolor='black'))
```

Out[14]:
```
Text(1, 3, 'Equilibrium at (61/9,30.666666666666664)')
```



Higher-Degree Polynomial Equations --- Roots

In [15]:
```
from scipy import optimize
import matplotlib.pyplot as plt
```

In [16]:
```
def f(x):
    return (x**3 - x**2 - 4*x + 4)
x = np.array([-3, 0 , 3]) # Define an array which is near to  possible roots
roots = optimize.newton(f, x)
roots
```

Out[16]:
```
array([-2.,  1.,  2.])
```

3.4 General Market Equilibrium

In [17]:
```
P1 = sy.Symbol('P_1')
P2 = sy.Symbol('P_2')

c0 = sy.Symbol('c_0')
c1 = sy.Symbol('c_1')
c2 = sy.Symbol('c_2')
gamma0 = sy.Symbol('\\gamma_0')
gamma1 = sy.Symbol('\\gamma_1')
gamma2 = sy.Symbol('\\gamma_2')
eq1 = Eq(c1*P1 + c2*P2,-c0)
eq2 = Eq(gamma1*P1 + gamma2*P2,-gamma0)
display(eq1)
display(eq2)
```

$$P_1 c_1 + P_2 c_2 = -c_0$$

$$P_1 \gamma_1 + P_2 \gamma_2 = -\gamma_0$$

In [18]:
```
from sympy import symbols, Eq, solve
solve((eq1,eq2), (P1,P2))
```

Out[18]:
```
{P_1: (-\gamma_0*c_2 + \gamma_2*c_0)/(\gamma_1*c_2 - \gamma_2*c_1),
 P_2: (\gamma_0*c_1 - \gamma_1*c_0)/(\gamma_1*c_2 - \gamma_2*c_1)}
```

Numerical Example

In [19]:
```
eq1 = Eq(-5*P1 + 1*P2,-12)
eq2 = Eq(1*P1 +  -3*P2,-16)
display(eq1)
display(eq2)
solve((eq1,eq2), (P1,P2))
```

$$-5P_1 + P_2 = -12$$

$$P_1 - 3P_2 = -16$$

Out[19]:
```
{P_1: 26/7, P_2: 46/7}
```

3.5 Equilibrium in National-Income Analysis

In [20]:
```
Y = sy.Symbol('Y')
```

```
C = sy.Symbol('C')

I0 = sy.Symbol('I_0')
G0 = sy.Symbol('G_0')
a = sy.Symbol('a')
b = sy.Symbol('b')
eq1 = Eq(C + I0 + G0,Y)
eq2 = Eq(a + b *Y,C)
display(eq1)
display(eq2)
solve((eq1,eq2), (Y,C))
```

$$C + G_0 + I_0 = Y$$

$$Yb + a = C$$

Out[20]:
```
{C: -(a + b*(G_0 + I_0))/(b - 1), Y: -(G_0 + I_0 + a)/(b - 1)}
```

EXERCISE 3.5 Q3

In [21]:
```
eq1 = Eq(C + 16 + 14,Y)
eq2 = Eq(25 + 6*Y**(1/2),C)
display(eq1)
display(eq2)
solve((eq1,eq2), (Y,C))
```

$$C + 30 = Y$$

$$6Y^{0.5} + 25 = C$$

Out[21]:
```
[(121.000000000000, 91.0000000000000)]
```

Furkan Zengin

Mathematical Economics

Alpha Chiang

Chapter 4-5

Chapter 4 Linear Models and Matrix Algebra Matrices as Arrays

In [1]:
```
from sympy import symbols,Matrix
x1, x2, x3 = symbols('x_1,x_2,x_3')
A = Matrix(([6, 3, 1], [1, 4, -2], [4, -1 , 5]))
A
```

Out[1]:
$$\begin{bmatrix} 6 & 3 & 1 \\ 1 & 4 & -2 \\ 4 & -1 & 5 \end{bmatrix}$$

In [2]:
```
x = Matrix((x1,x2,x3))
x
```

Out[2]:
$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

In [3]:
```
d = Matrix((22,12,10))
d
```

Out[3]:
$$\begin{bmatrix} 22 \\ 12 \\ 10 \end{bmatrix}$$

In [4]:
```
A * x
```

Out[4]:

$$\begin{bmatrix} 6x_1 + 3x_2 + x_3 \\ x_1 + 4x_2 - 2x_3 \\ 4x_1 - x_2 + 5x_3 \end{bmatrix}$$

In [5]:

```
import numpy as np
npA = np.array(([6, 3, 1], [1, 4, -2], [4, -1 , 5]))
# To be able to solve this system linearly, we need to use numpy arrays
npA
npd = np.array((22,12,10))
npd
x = np.linalg.solve(npA, npd)
x
```

Out[5]:

```
array([2., 3., 1.])
```

Basic Matrix Operations and other operations can be found the below websites

https://numpy.org/doc/stable/reference/generated/numpy.matrix.html

https://docs.sympy.org/latest/tutorial/matrices.html

Example 6 --PAGE 78--

In [7]:

```
from sympy import symbols, Eq, solve
Y, C, I0, G0, a, b = symbols('Y, C, I_0, G_0, a, b')
eq1 = Eq(Y, C + I0 + G0)
eq2 = Eq(C, a + b*Y)

result = solve([eq1, eq2],(Y,C))

print(result[Y])
print(result[C])
```

```
-(G_0 + I_0 + a)/(b - 1)
-(a + b*(G_0 + I_0))/(b - 1)
```

In [8]:

```
from sympy import symbols,Matrix
from sympy import Symbol, dsolve, Function, Derivative, Eq
Y, C, I0, G0, a, b = symbols('Y, C, I_0, G_0, a, b')
eq1 = Eq(Y, C + I0 + G0)
eq1
```

Out[8]:

$Y = C + G_0 + I_0$

In [9]:
```
eq2 = Eq(C, a + b*Y)
eq2
```

Out[9]: $C = Yb + a$

In [10]:
```
from sympy import *
Y, C, I0, G0, a, b = symbols('Y, C, I_0, G_0, a, b')
eqns = [ (Y - C - I0 - G0), (C - a - b*Y)]
linsolve(eqns, [Y ,C ])
```

Out[10]: $$\left\{\left(-\frac{G_0 + I_0 + a}{b - 1},\ -\frac{G_0 b + I_0 b + a}{b - 1}\right)\right\}$$

CHAPTER 5

Linear Models and Matrix Algebra (Continued)

Example 9 --PAGE 97--

In [12]:
```
from sympy import symbols,Matrix
x1, x2, x3 = symbols('x_1,x_2,x_3')
A = Matrix(([7, -3, -3], [2, 4, 1], [0, -2 , -1]))
A
```

Out[12]: $$\begin{bmatrix} 7 & -3 & -3 \\ 2 & 4 & 1 \\ 0 & -2 & -1 \end{bmatrix}$$

In [13]:
```
x = Matrix((x1,x2,x3))
x
```

Out[13]: $$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

In [14]:
```
A * x
```

Out[14]:

$$\begin{bmatrix} 7x_1 - 3x_2 - 3x_3 \\ 2x_1 + 4x_2 + x_3 \\ -2x_2 - x_3 \end{bmatrix}$$

In [15]:
```
import numpy as np
npA = np.array(([7, -3, -3], [2, 4, 1], [0, -2 , -1]))
D1 = np.linalg.det(npA)
D1
```

Out[15]:
```
-7.999999999999998
```

In [16]:
```
npd = np.array((7,0,2))
npd
x = np.linalg.solve(npA, npd)
x
```

Out[16]:
```
array([-0.5,  1.5, -5. ])
```

In [17]:
```
from numpy.linalg import matrix_rank
matrix_rank(npA) #Rank of a Matrix
```

Out[17]:
```
3
```

5.4 Finding the Inverse Matrix

Example 1

In [18]:
```
import numpy as np
npA = np.array(([4, 1, 2], [5, 2, 1], [1, 0 , 3]))
npA
```

Out[18]:
```
array([[4, 1, 2],
       [5, 2, 1],
       [1, 0, 3]])
```

In [19]:
```
from numpy.linalg import inv
inv(npA)
```

Out[19]:
```
array([[ 1.        , -0.5       , -0.5       ],
```

```
       [-2.33333333,  1.66666667,  1.        ],
       [-0.33333333,  0.16666667,  0.5       ]])
```

Example 2

In [20]:
```
npA = np.array(([3, 2], [1, 0]))
inv(npA)
```

Out[20]:
```
array([[ 0. ,  1. ],
       [ 0.5, -1.5]])
```

Example 3

In [21]:
```
npA = np.array(([4, 1, -1], [0, 3, 2], [3, 0 , 7]))
D2 = np.linalg.det(npA)
D2
```

Out[21]:
```
98.99999999999999
```

In [22]:
```
inv(npA)
```

Out[22]:
```
array([[ 0.21212121, -0.07070707,  0.05050505],
       [ 0.06060606,  0.31313131, -0.08080808],
       [-0.09090909,  0.03030303,  0.12121212]])
```

5.6 Application to Market and National-Income Models

Market Model

In [24]:
```
P1 = Symbol('P_1')
P2 = Symbol('P_2')
c0 = Symbol('c_0')
c1 = Symbol('c_1')
c2 = Symbol('c_2')
gamma0 = Symbol('\\gamma_0')
gamma1 = Symbol('\\gamma_1')
gamma2 = Symbol('\\gamma_2')
eq1 = Eq(c1*P1 + c2*P2,-c0)
eq2 = Eq(gamma1*P1 + gamma2*P2,-gamma0)
display(eq1,eq2)
```

$$P_1 c_1 + P_2 c_2 = -c_0$$

$$P_1 \gamma_1 + P_2 \gamma_2 = -\gamma_0$$

In [25]:
```
from sympy import symbols, Eq, solve
solve((eq1,eq2), (P1,P2))
```

Out[25]:
```
{P_1: (-\gamma_0*c_2 + \gamma_2*c_0)/(\gamma_1*c_2 - \gamma_2*c_1),
 P_2: (\gamma_0*c_1 - \gamma_1*c_0)/(\gamma_1*c_2 - \gamma_2*c_1)}
```

IS-LM Model: Closed Economy

In [26]:
```
Y = Symbol('Y')
C = Symbol('C')
I = Symbol('I')
G = Symbol('G')
a = Symbol('a')
b = Symbol('b')
t = Symbol('t')
d = Symbol('d')
e = Symbol('e')
i = Symbol('i')
G0 = Symbol('G_0')
M0 = Symbol('M_0')
Md = Symbol("M_d")
Ms = Symbol("M_s")
l = Symbol('l')
k = Symbol('k')
eq1 = Eq(Y, C + I + G)
eq2 = Eq(Md,Ms)
eq3 = Eq(C, a + b*(1 - t)*Y)
eq4 = Eq(I, d -e*i)
eq5 = Eq(G, G0)
eq6 = Eq(M0, k*Y -l*i)
```

In [28]:
```
display(eq1,eq2,eq3,eq4,eq5,eq6)
```

$$Y = C + G + I$$

$$M_d = M_s$$

$$C = Yb(1 - t) + a$$

$$I = d - ei$$

$$G = G_0$$

$M_0 = Yk - il$

In [29]:
```
from sympy import symbols,Matrix
A = Matrix(([1, -1, -1, 0], [b*(1-t), -1, 0, 0],
            [0, 0 , 1, e], [k,0,0,-l]))
A
```

Out[29]:
$$\begin{bmatrix} 1 & -1 & -1 & 0 \\ b(1-t) & -1 & 0 & 0 \\ 0 & 0 & 1 & e \\ k & 0 & 0 & -l \end{bmatrix}$$

In [30]:
```
x = Matrix((Y,C,I,i))
x
```

Out[30]:
$$\begin{bmatrix} Y \\ C \\ I \\ i \end{bmatrix}$$

In [31]:
```
d = Matrix((G0,-a,d,M0))
d
```

Out[31]:
$$\begin{bmatrix} G_0 \\ -a \\ d \\ M_0 \end{bmatrix}$$

In [32]:
```
A * x
```

Out[32]:
$$\begin{bmatrix} -C - I + Y \\ -C + Yb(1-t) \\ I + ei \\ Yk - il \end{bmatrix}$$

In [33]:
```
from sympy import *
Y, C, I, G, G0, a, b = symbols('Y, C, I, G, G_0, a, b')
```

```
t, d, e, i, M0 ,Md ,Ms ,l ,k = symbols("t, d, e, i,M_0,M_d,M_s,l,k")
eqns = [(Y - C - I - G0), (-C + a + (b*(1 - t)*Y)), (I - d + (e*i)) ,(-M0 + (k*Y) + (-l*i))]
eqns
```

Out[33]: [-C - G_0 - I + Y, -C + Y*b*(1 - t) + a, I - d + e*i, -M_0 + Y*k - i*l]

In [46]:
```
SOL = linsolve(eqns, [Y,C,i,I])
# Run the SOL
```

EXERCISE 5.6 --Q3--

In [38]:
```
from sympy import symbols,Matrix
A = Matrix(([0.3, 100], [0.25, -200]))
A
```

Out[38]:

$$\begin{bmatrix} 0.3 & 100 \\ 0.25 & -200 \end{bmatrix}$$

In [39]:
```
x = Matrix((Y,i))
x
```

Out[39]:

$$\begin{bmatrix} Y \\ i \end{bmatrix}$$

In [40]:
```
d = Matrix((252,176))
d
```

Out[40]:

$$\begin{bmatrix} 252 \\ 176 \end{bmatrix}$$

In [41]:
```
A * x
```

Out[41]:

$$\begin{bmatrix} 0.3Y + 100i \\ 0.25Y - 200i \end{bmatrix}$$

In [42]:
```
import numpy as np
npA = np.array(([0.3, 100], [0.25, -200]))
npA
npd = np.array((252,176))
npd
```

```python
x = np.linalg.solve(npA, npd)
x
```

Out[42]:

```
array([8.0e+02, 1.2e-01])
```

Furkan zengin

Mathematical Economics

Alpha Chiang

Chapter 7

Rules of Differentiation nand Their Use in Comparative Statics

In [26]:
```
from sympy import Symbol, dsolve, Function, Derivative, Eq
from sympy import exp, sin, sqrt, diff
```

7.1 Rules of Differentiation for a Function of One Variable

In [21]:
```
#  Example 1
y = Function("y")
x = Symbol('x')

display(Eq(y(x),x**3))
diff(x**3, x)
```

$$y(x) = x^3$$

Out[21]: $3x^2$

Example 4

In [4]:
```
display(Eq(y(x),1/x**3))
diff(1/x**3, x)
```

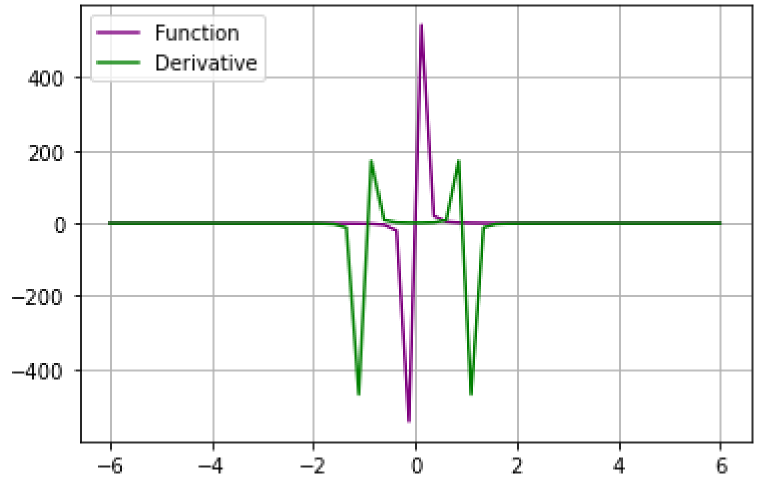
$$y(x) = \frac{1}{x^3}$$

Out[4]: $-\frac{3}{x^4}$

Using this way:

https://www.geeksforgeeks.org/how-to-calculate-and-plot-the-derivative-of-a-function-using-python-matplotlib/

In [5]:
```
import matplotlib.pyplot as plt
from scipy.misc import derivative
import numpy as np

# defining the function
def function(x):
    return 1/x**3

def deriv(x):
    return derivative(function, x)


y = np.linspace(-6, 6)


plt.plot(y, function(y), color='purple', label='Function')


plt.plot(y, deriv(y), color='green', label='Derivative')

plt.legend(loc='upper left')
plt.grid(True)
```



Example 5

Before running this code, we should run the first two code again !!!

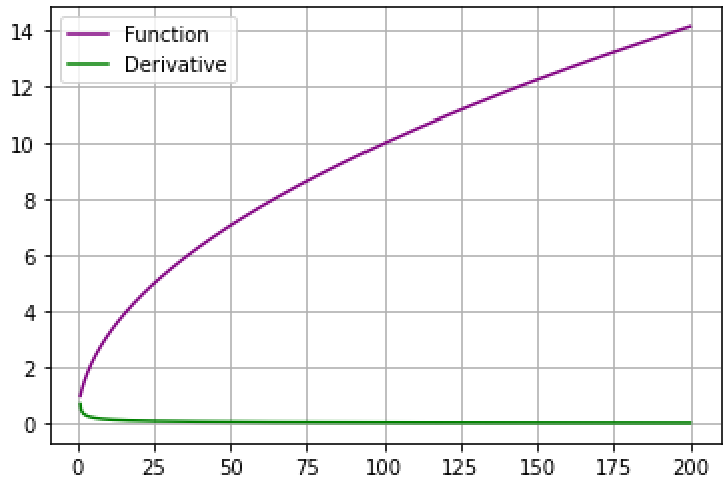
We should do this before taking diff every time !

In [13]:
```
display(Eq(y(x),sqrt(x)))
diff(sqrt(x), x)
```

$$y(x) = \sqrt{x}$$

Out[13]:
$$\frac{1}{2\sqrt{x}}$$

In [14]:
```
import matplotlib.pyplot as plt
from scipy.misc import derivative
import numpy as np

def function(x):
    return x**(1/2)

def deriv(x):
    return derivative(function, x)

y = np.linspace(1, 200,1000)

plt.plot(y, function(y), color='purple', label='Function')
plt.plot(y, deriv(y), color='green', label='Derivative')
plt.legend(loc='upper left')
plt.grid(True)
```



EXERCISE 7.1 -- Q3(b) --

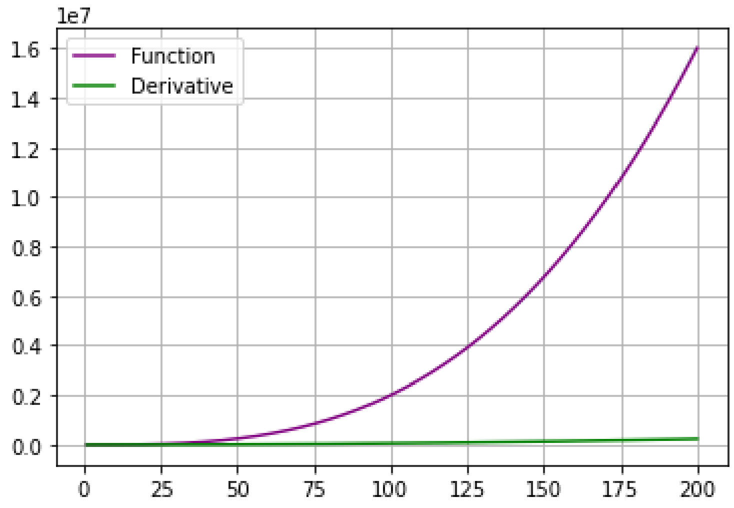
In [15]:
```
c = Symbol('c')
a = Symbol('a')
b = Symbol('b')
y = Function("y")
u = Symbol('u')
display(Eq(y(u), a*u**(b)))
diff(a*u**(b),u)
```

$$y(u) = au^b$$

Out[15]:
$$\frac{abu^b}{u}$$

In [16]:
```
import matplotlib.pyplot as plt
from scipy.misc import derivative
import numpy as np

def function(u,a = 2, b = 3):
    return a*u**(b)

def deriv(x):
    return derivative(function, x)

y = np.linspace(1, 200,1000)
```

```
plt.plot(y, function(y), color='purple', label='Function')
plt.plot(y, deriv(y), color='green', label='Derivative')
plt.legend(loc='upper left')
plt.grid(True)
```



7.2 Rules of Differentiation Involving Two or More Functions of the Same Variable

Example 2

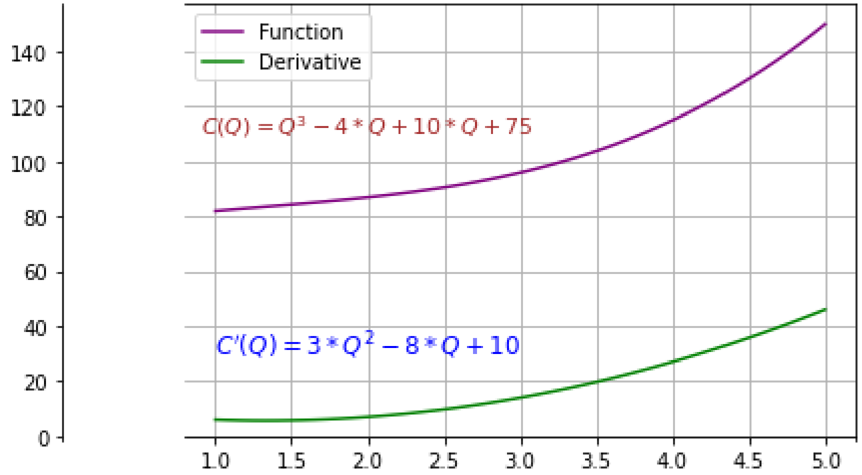
In [17]:
```
C = Function('C')
Q = Symbol('Q')
x = Symbol("x")

display(Eq(C(Q),Q**3 - 4*Q**2 + 10*Q + 75))
diff(Q**3 - 4*Q**2 + 10*Q + 75,Q)
```

$$C(Q) = Q^3 - 4Q^2 + 10Q + 75$$

Out[17]: $3Q^2 - 8Q + 10$

In [18]:
```
import matplotlib.pyplot as plt
from scipy.misc import derivative
import numpy as np

def function(Q):
    return Q**3 - 4*Q**2 + 10*Q + 75
```

```
def deriv(Q):
    return derivative(function, Q)

y = np.linspace(1, 5)

plt.plot(y, function(y), color='purple', label='Function')
plt.plot(y, deriv(y), color='green', label='Derivative')
plt.legend(loc='upper left')
plt.gca().spines['left'].set_position('zero',)
plt.gca().spines['bottom'].set_position('zero',)
plt.legend(loc='upper left')
plt.text(2, 30, r"$C'(Q)= 3*Q^2 -8*Q +10$",
         horizontalalignment='center',
         fontsize=12, color='blue')
plt.text(2, 110, r'$C(Q)=Q^3- 4*Q +10*Q +75$',
         horizontalalignment='center',
         fontsize=11, color='brown')
plt.grid(True)
```



Relationship Between Marginal-Cost and Average-Cost Functions

--Figure 7.3--

In [28]:
```
C = Symbol('C')
Q = Symbol('Q')
x = Symbol("x")
M = Symbol("M")
```

```
A = Symbol("A")
M_C = Symbol("MC")
display(Eq(M_C,Q**3 - 12*Q**2 + 60*Q))
display(diff(Q**3 - 12*Q**2 + 60*Q,Q))
AC = (Q**2 - 12*Q+ 60)
AC
```

$$MC = Q^3 - 12Q^2 + 60Q$$

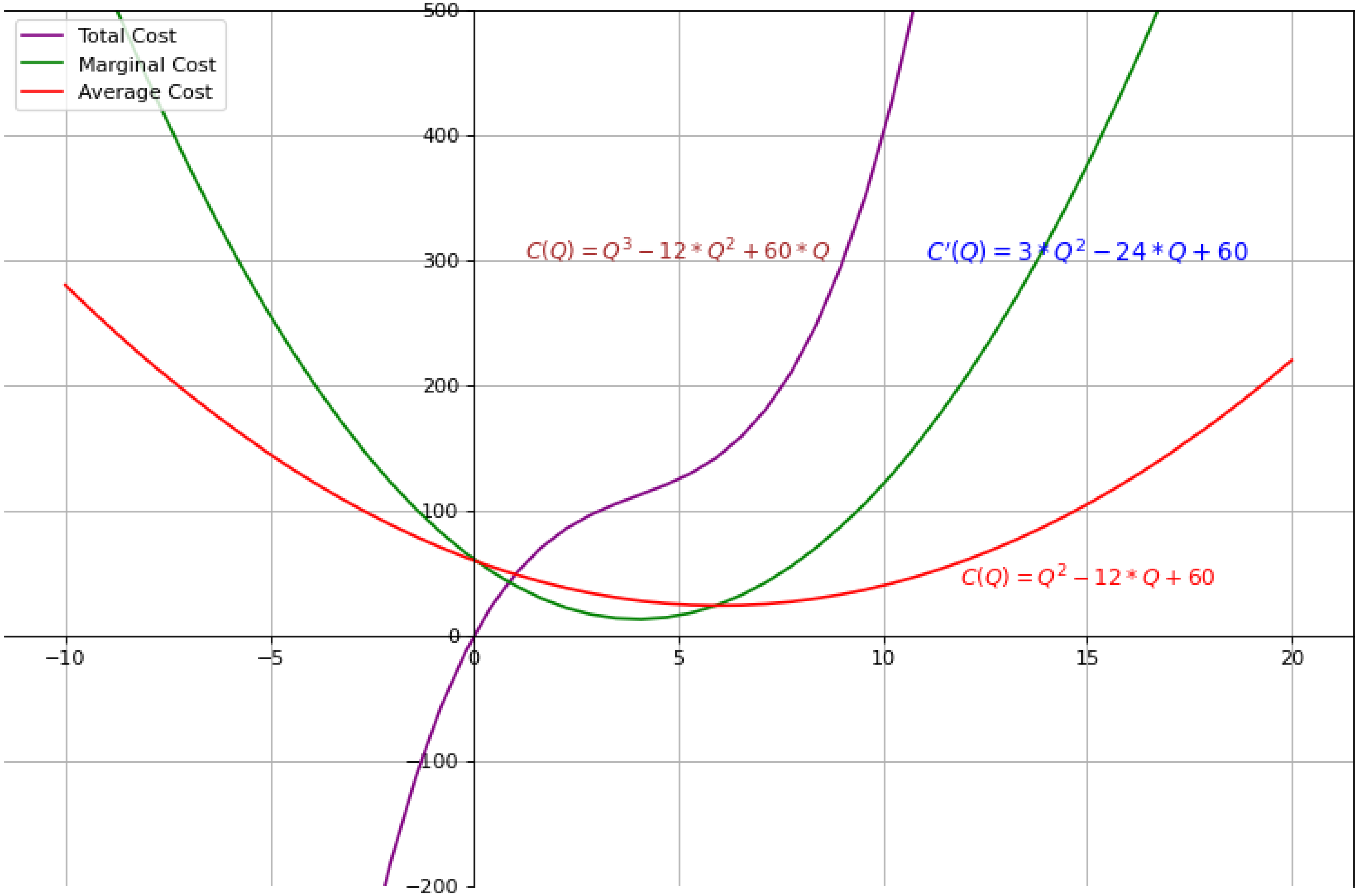
$$3Q^2 - 24Q + 60$$

Out[28]: $$Q^2 - 12Q + 60$$

In [29]:

```
import matplotlib.pyplot as plt
from scipy.misc import derivative
import numpy as np
from matplotlib.pyplot import figure

def function(Q):
    return Q**3 - 12*Q**2 + 60*Q

def deriv(Q):
    return derivative(function, Q)

def Avecost(Q):
    return (Q**2 -12*Q + 60)

figure(figsize=(12, 8), dpi=80)
y = np.linspace(-10,20)
plt.ylim((-200,500))
plt.plot(y, function(y), color='purple', label='Total Cost')
plt.plot(y, deriv(y), color='green', label='Marginal Cost')
plt.plot(y, Avecost(y), color='red', label='Average Cost')
plt.legend(loc='upper left')
plt.gca().spines['left'].set_position('zero',)

plt.gca().spines['bottom'].set_position('zero',)
plt.legend(loc='upper left')
plt.text(15, 300, r"$C'(Q)= 3*Q^2 -24*Q +60$",
         horizontalalignment='center',
         fontsize=12, color='blue')
plt.text(5, 300, r'$C(Q)=Q^3 -12*Q^2 +60*Q$',
         horizontalalignment='center',
         fontsize=11, color='brown')
```

```python
plt.text(15, 40, r'$C(Q)=Q^2 -12*Q +60$',
         horizontalalignment='center',
         fontsize=11, color='red')
plt.grid(True)
```



In [30]:
```python
import sympy as sy
eq1 = Eq(deriv(Q),Avecost(Q))
eq1
```

Out[30]: $-0.5(Q-1.0)^3 + 6.0(Q-1.0)^2 + 0.5(Q+1.0)^3 - 6.0(Q+1.0)^2 + 60.0 = Q^2 - 12Q + 60$

In [31]:
```
display(sy.solve(eq1))
display(deriv(5.91))
```

```
[0.0845240525773498, 5.91547594742265]
23.944299999999997
```

In [32]:
```
from sympy import Symbol, dsolve, Function, Derivative, Eq
from scipy.misc import derivative
y = Symbol("y")
x = Symbol('x')
f = Function("f")
f2 = Function("f2")

def f(y):
    return 3*y**2

def deriv1(y):
    return derivative(f, y)

def f2(x):
    return 2*x + 5

def deriv2(x):
    return derivative(f2, x)

Chain = deriv1(y)*deriv2(x)
Chain
```

Out[32]: $-3.0(y-1.0)^2+3.0(y+1.0)^2$

In [33]:
```
def f(y):
    return y - 3

def deriv1(y):
    return derivative(f, y)

def f2(x):
    return x**3

def deriv2(x):
    return derivative(f2, x)

Chain = deriv1(y)*deriv2(x)
Chain
```

Out[33]: $-0.5(x-1.0)^3+0.5(x+1.0)^3$

In [35]:
```
from sympy import symbols
x, y, z = symbols('x y z')
```

In [37]:
```
z = 3*y**2
y = 2*x + 5
diff(z, x)
```

Out[37]: $24x+60$

In [39]:
```
z = y - 3
y = x**3
diff(z, x)
```

Out[39]: $3x^2$

7.4 Partial Differentiation Techniques of Partial Differentiation

Example 1

In [40]:
```
from sympy import symbols, diff
x1, x2 = symbols('x_1 x_2')
f = Function("f")
f1 = 3*x1**2 + x1*x2 + 4*x2**2
eq1 = Eq(f(x1,x2),3*x1**2 + x1*x2 + 4*x2**2)
display(eq1)
display(diff(f1, x1))
display(diff(f1,x2))
```

$f(x_1,x_2)=3x_1^2+x_1x_2+4x_2^2$

$6x_1+x_2$

x_1+8x_2

In [41]:
```
from sympy import *
res1 = diff(f1, x1)
res1.subs({x1:1, x2:3})
```

Out[41]: 9

In [42]:
```
res2 = diff(f1, x2)
res2.subs({x1:1, x2:3})
```

Out[42]: 25

Example 3

In [43]:
```
from sympy import symbols, diff
u, v, y = symbols('u v y')
f = Function("f")
f2 = (3*u - v)/(u**2 + 3*v)
eq2 = Eq(y,(3*u - v)/(u**2 + 3*v))
display(eq2)
display(diff(f2, u))
display(diff(f2,v))
```

$$y = \frac{3u - v}{u^2 + 3v}$$

$$-\frac{2u\left(3u - v\right)}{\left(u^2 + 3v\right)^2} + \frac{3}{u^2 + 3v}$$

$$-\frac{3\left(3u - v\right)}{\left(u^2 + 3v\right)^2} - \frac{1}{u^2 + 3v}$$

In [44]:
```
res1 = diff(f2, u)
res1.subs({u:2, v:2})
```

Out[44]: $\frac{7}{50}$

In [45]:
```
res2 = diff(f2, v)
res2.subs({u:2, v:2})
```

Out[45]: $-\frac{11}{50}$

EXERCISE 7.4 -- Q5 --

In [46]:
```
from sympy import symbols, diff
x1, x2  = symbols('x_1 x_2')
U = Function("U")
f1 = (x1 + 2)**(2) * (x2 + 3 )**3
eq1 = Eq(U(x1,x2),(x1 + 2)**(2) * (x2 + 3 )**3)
display(eq1)
display(diff(f1, x1))
display(diff(f1,x2))
```

$$U(x_1, x_2) = (x_1 + 2)^2 (x_2 + 3)^3$$

$$(2x_1 + 4)(x_2 + 3)^3$$

$$3(x_1 + 2)^2 (x_2 + 3)^2$$

In [47]:
```
res1 = diff(f1, x1)
res1.subs({x1:3, x2:3})
```

Out[47]: 2160

In [48]:
```
res2 = diff(f1, x2)
res2.subs({x1:3, x2:3})
```

Out[48]: 2700

7.5 Applications to Comparative-Static Analysis National-Income Model

In [49]:
```
from sympy import symbols, diff
Y,alpha,beta,gamma,I0,G0,delta  = symbols('Y \\alpha \\beta \\gamma I_0 G_0 \\delta')
U = Function("U")
Y = (alpha - beta*gamma + I0 + G0)/(1 - beta + beta*delta)
eq1 = Eq(Y,(alpha - beta*gamma + I0 + G0)/(1 - beta + beta*delta))
display(eq1)
display(diff(Y, G0))
display(diff(Y, gamma))
display(diff(Y, delta))
```

True

$$\frac{1}{\beta\delta - \beta + 1}$$

$$-\frac{\beta}{\beta\delta - \beta + 1}$$

$$-\frac{\beta\left(G_0 + I_0 + \alpha - \beta\gamma\right)}{\left(\beta\delta - \beta + 1\right)^2}$$

In [50]:

```
from sympy import symbols,Matrix
x, y, z = symbols('x,y,z')
A = Matrix(([diff(Y, G0)], [diff(Y, gamma)],[diff(Y, delta)]))
A
```

Out[50]:

$$\left[\begin{matrix}\frac{1}{\beta\delta-\beta+1}\\-\frac{\beta}{\beta\delta-\beta+1}\\-\frac{\beta(G_0+I_0+\alpha-\beta\gamma)}{(\beta\delta-\beta+1)^2}\end{matrix}\right]$$

Furkan zengin

Mathematical Economics

Alpha Chiang

Chapter 8

Comparative-Static Analysis of General Function Models

Differentials and Point Elasticity

In [3]:
```
from sympy import Symbol, dsolve, Function, Derivative,symbols
from sympy import diff, Eq
Q = Function("Q")
P = Function("P")
a, d, epsilond = symbols("a d \\epsilon_d")
#Point elasticity of demand
eq1 = Eq(epsilond , (Derivative(Q(a),a,1)/Q(a)) / ((Derivative(P(a),a))/Q(a)))
eq1
```

Out[3]:

$$\epsilon_d = \frac{\frac{d}{da}Q(a)}{\frac{d}{da}P(a)}$$

Example 1

In [6]:
```
from scipy.misc import derivative
from sympy import simplify
x = Symbol("x")

def demand(x):
    "x = P"
    return 100-2*x

def deriv(x):
    return derivative(demand,x)

def avg(x):
```

```
    return demand(x)/x

def elasticity(x):
    return deriv(x)/avg(x)

E = elasticity(x)
simplify(E)
```

Out[6]:

$$\frac{1.0x}{x - 50}$$

Example 2

In [7]:

```
def demand(x):
    return x**2 + 7*x

def deriv(x):
    return derivative(demand,x)

def avg(x):
    return demand(x)/x

def elasticity(x):
    return deriv(x)/avg(x)

E = elasticity(x)
simplify(E)
```

Out[7]:

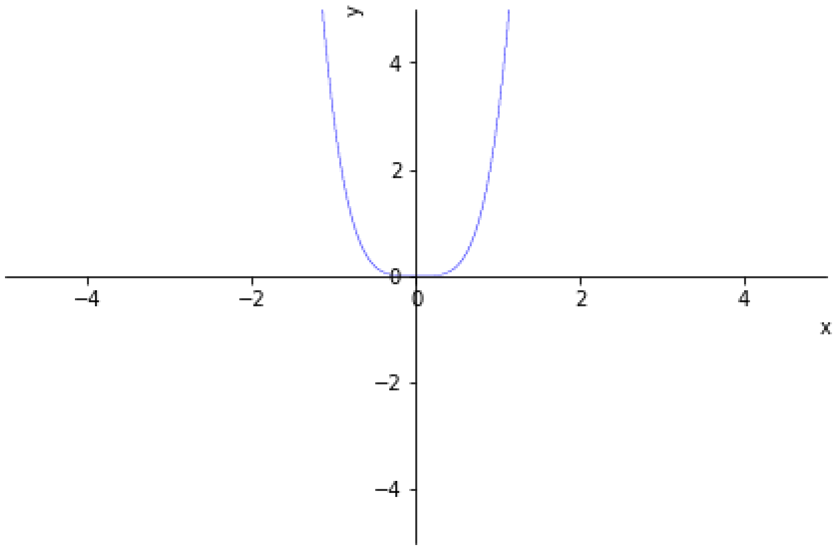
$$\frac{1.0\left(2.0x + 7.0\right)}{1.0x + 7.0}$$

8.5 Derivatives of Implicit Functions

Example 1

In [9]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import sympy as sy
x, y = symbols('x y')
eq1 = Eq(y - 3*x**4, 0)
sy.plot_implicit(eq1)
```



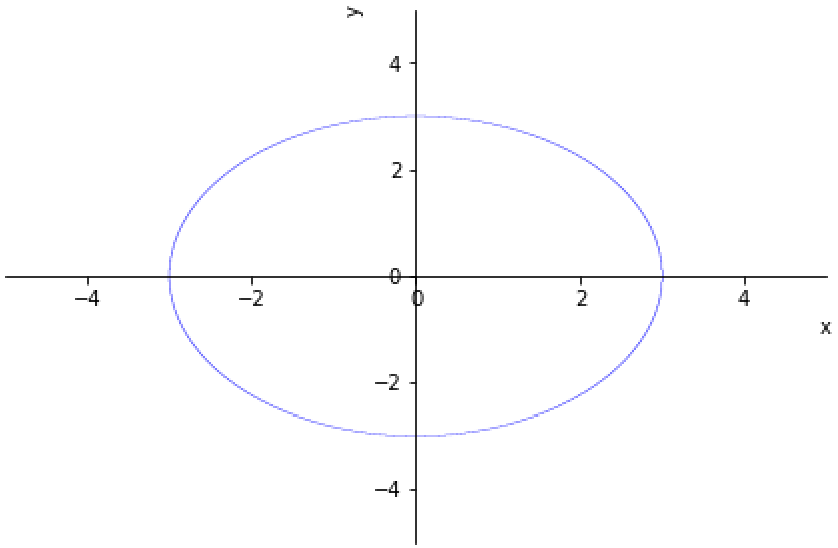
Out[9]: <sympy.plotting.plot.Plot at 0x2d4c3566df0>

In [10]:
```
eq1 = y - 3*x**4
deq1 = sy.idiff(eq1, y, x)
deq1
```

Out[10]: $12x^3$

Example 2

In [12]:
```
eq2 = Eq(x**2 + y**2 - 9, 0)
sy.plot_implicit(eq2)
```



Out[12]: <sympy.plotting.plot.Plot at 0x2d4c4676c10>

In [13]:
```
eq2 = x**2 + y**2 - 9
deq2 = sy.idiff(eq2, y, x)
deq2
```

Out[13]: $-\frac{x}{y}$

Example 3

In [14]:
```
x, y, w, z= symbols('x y w,z')
eq3 = (y**3 * x**2) + w**3 +y*x*w - 3
deq3 = sy.idiff(eq3, y, x)
display(deq3)
deq3.subs({x:1, y:1, w:1}) # At the point (1, 1, 1)
```

$$-\frac{y\left(w+2xy^2\right)}{x\left(w+3xy^2\right)}$$

Out[14]: $-\frac{3}{4}$

In [15]:
```
dx, dy, dw, dz = symbols('dx dy dw dz')
```

```python
def f(x, y, w):
    eq1 = x*y - w
    F1 = diff(eq1,x)
    F1_1 = diff(eq1,y)
    F1_2 = diff(eq1,w)
    return F1*dx + F1_1*dy + F1_2*dw


def f2(z, y, w):
    eq2 = y - w**3 - 3*z
    F2 = diff(eq2,z)
    F2_1 = diff(eq2,y)
    F2_2 = diff(eq2,w)
    return F2*dz + F2_1*dy + F2_2*dw

def f3(w,z):
    eq3 = w**3 + z**3 -2*w*z
    F3 = diff(eq3,z)
    F3_1 = diff(eq3,w)
    return F3*dz + F3_1*dw
display(f(x,y,w), f2(z,y,w), f3(w,z))
TotalD = [f(x,y,w), f2(z,y,w), f3(w,z)]
TotalD
```

$$-dw + dxy + dyx$$

$$-3dww^2 + dy - 3dz$$

$$dw\left(3w^2 - 2z\right) + dz\left(-2w + 3z^2\right)$$

Out[15]:
```
[-dw + dx*y + dy*x,
 -3*dw*w**2 + dy - 3*dz,
 dw*(3*w**2 - 2*z) + dz*(-2*w + 3*z**2)]
```

In [16]:
```python
import sympy as sp
M = sp.Matrix(([y,x,-1],[0,1,-3*w**2],[0,0,3*w**2 -2*z]))
display(M)
Det1 = sp.det(M)
display(Det1)
Det1.subs({y:4,w:1,z:1}) #the Jacobian determinant
```

$$\begin{bmatrix} y & x & -1 \\ 0 & 1 & -3w^2 \\ 0 & 0 & 3w^2 - 2z \end{bmatrix}$$

$$3w^2y - 2yz$$

Out[16]: 4

Example 6

In [17]:

```
from sympy import diff, Eq
Y,C,G0,I0,T,alpha = symbols('Y C G_0 I_0 T \\alpha')
beta , delta ,gamma = symbols('\\beta \\delta \\gamma')
eq1 =  Y - C - I0 - G0
eq2 = C - alpha - beta*(Y - T)
eq3 = T - gamma - delta*Y
M2 = sp.Matrix(([diff(eq1,Y),diff(eq1,C),diff(eq1,T)],
    [diff(eq2,Y),diff(eq2,C),diff(eq2,T)],
    [diff(eq3,Y),diff(eq3,C),diff(eq3,T)]))
display(M2)
Det2 = sp.det(M2)
display(Det2)
```

$$\begin{bmatrix} 1 & -1 & 0 \\ -\beta & 1 & \beta \\ -\delta & 0 & 1 \end{bmatrix}$$

$$\beta\delta - \beta + 1$$

Furkan zengin

Mathematical Economics

Alpha Chiang

Chapter 9

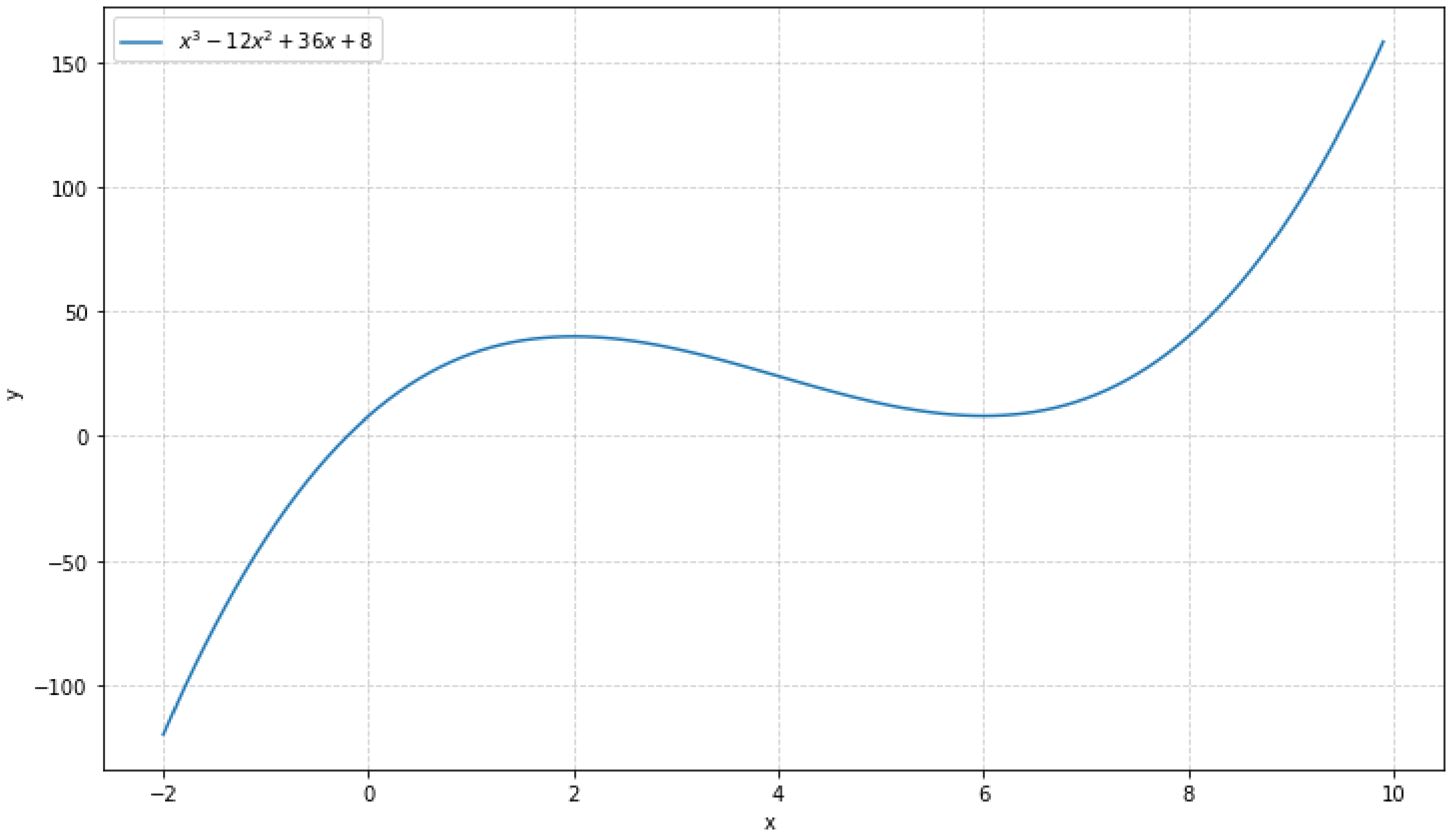
Optimization: A Special Variety of Equilibrium Analysis

9.2 Relative Maximum and Minimum: First-Derivative Test

Example 1

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np

def f(x):
    return x**3 - 12*x**2 + 36*x + 8

plt.figure(figsize = (12, 7))
x1 = np.arange(-2, 10, 0.1)
plt.plot(x1, f(x1),label ='$x^3 - 12x^2 + 36x + 8$')
plt.xlabel('x ')
plt.ylabel('y ')
plt.grid(alpha =.6, linestyle ='--')
plt.legend()
plt.show()
```



In [2]:
```
import numpy as np
from scipy import optimize
def f(x):
    return x**3 - 12*x**2 + 36*x + 8

grid = (-10, 10, 0.1)
xmin_global = optimize.brute(f, (grid, ))
print("Global minima found %s" % xmin_global)

# Constrain optimization
xmin_local = optimize.fminbound(f, -2, 10)
print("Local minimum found %s" % xmin_local)
```

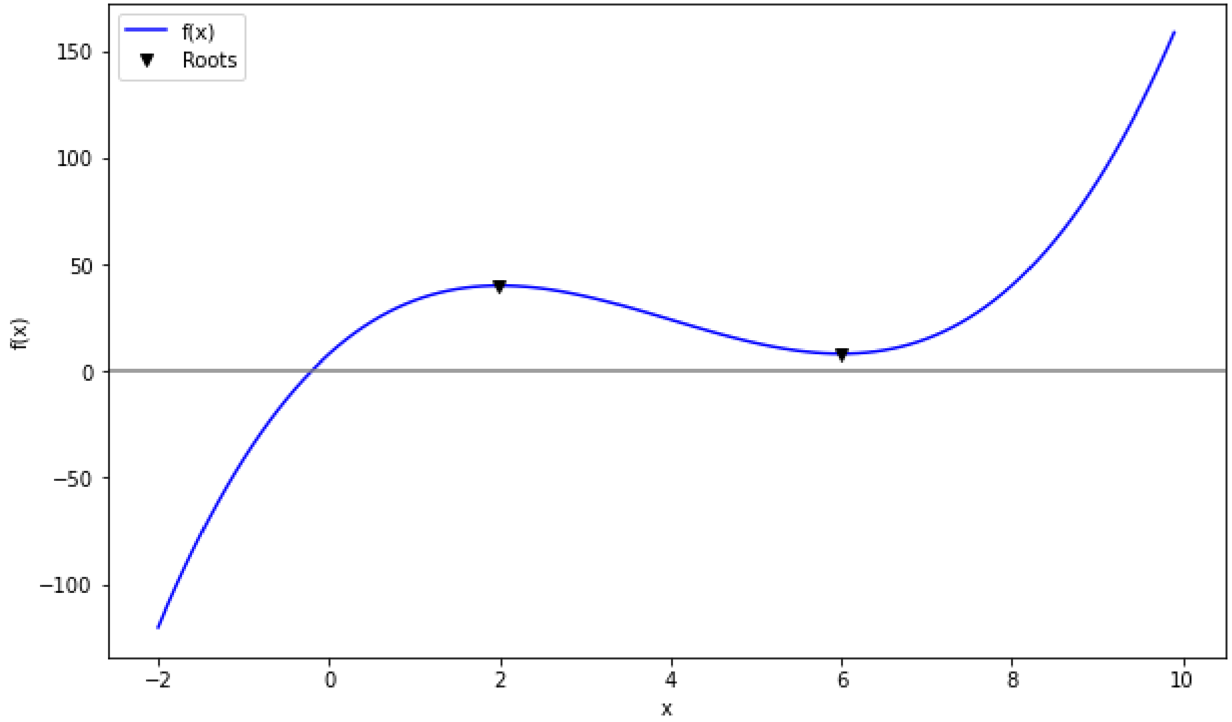
```
Global minima found [-6.338253e+29]
Local minimum found 6.000000017351318
```

In [3]:
```
root = optimize.root(f, 2)  # our initial guess is 1
```

```
print("First root found %s" % root.x)
root2 = optimize.root(f, 6)
print("Second root found %s" % root2.x)
```

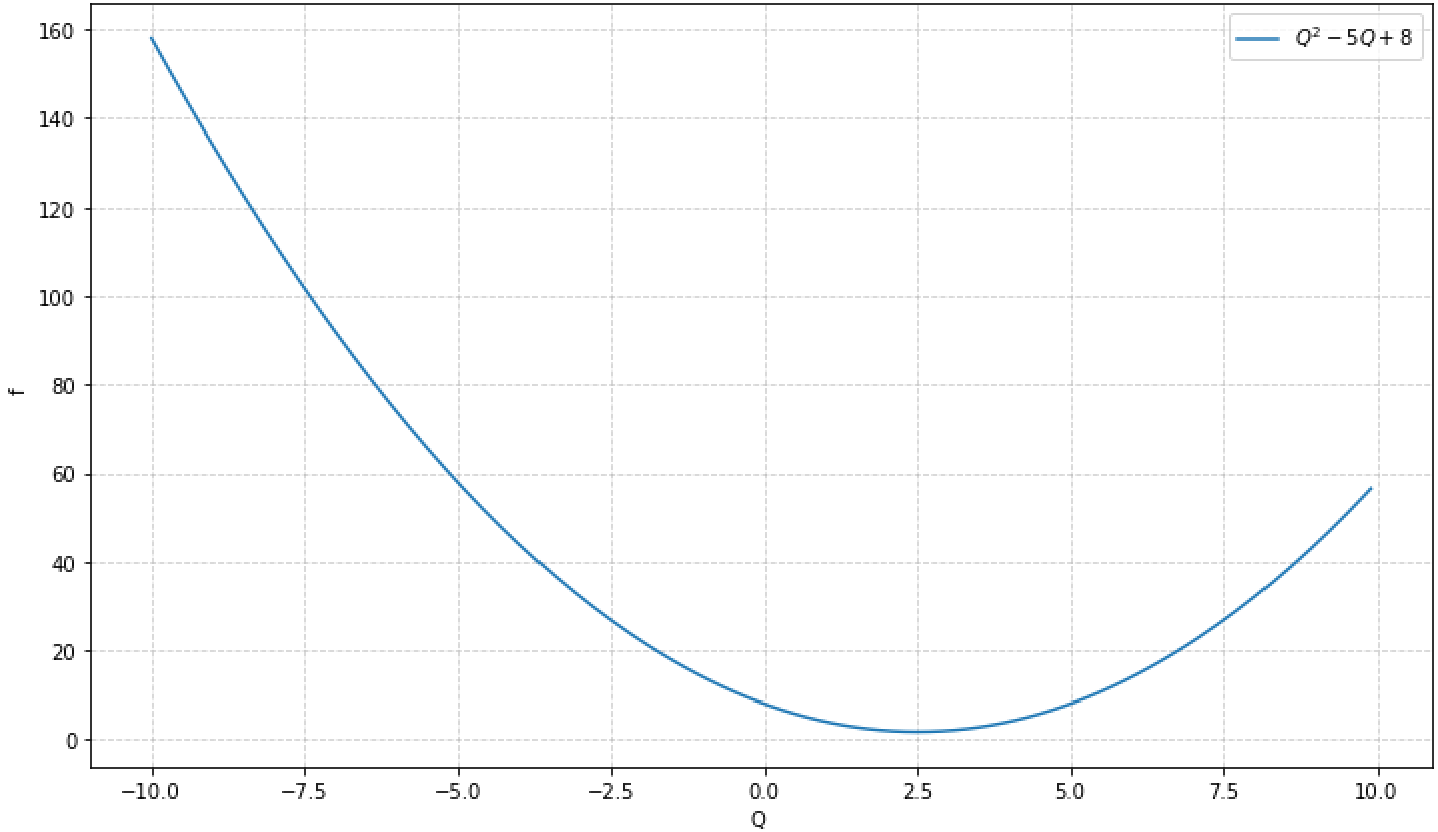
```
First root found [1.98350515]
Second root found [6.]
```

In [5]:
```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)

ax.plot(x1, f(x1), 'b-', label="f(x)")

roots = np.array([root.x, root2.x])
ax.plot(roots, f(roots), 'kv', label="Roots")

ax.legend(loc='best')
ax.set_xlabel('x')
ax.set_ylabel('f(x)')
ax.axhline(0, color='gray')
plt.show()
```



Example 2

In [6]:
```
def f(Q):
    return Q**2 -5*Q + 8

plt.figure(figsize = (12, 7))
Q1 = np.arange(-10, 10, 0.1)
plt.plot(Q1, f(Q1),label ='$Q^2 -5Q + 8$')
plt.xlabel('Q ')
plt.ylabel('f ')
plt.grid(alpha =.6, linestyle ='--')
plt.legend()
plt.show()
```



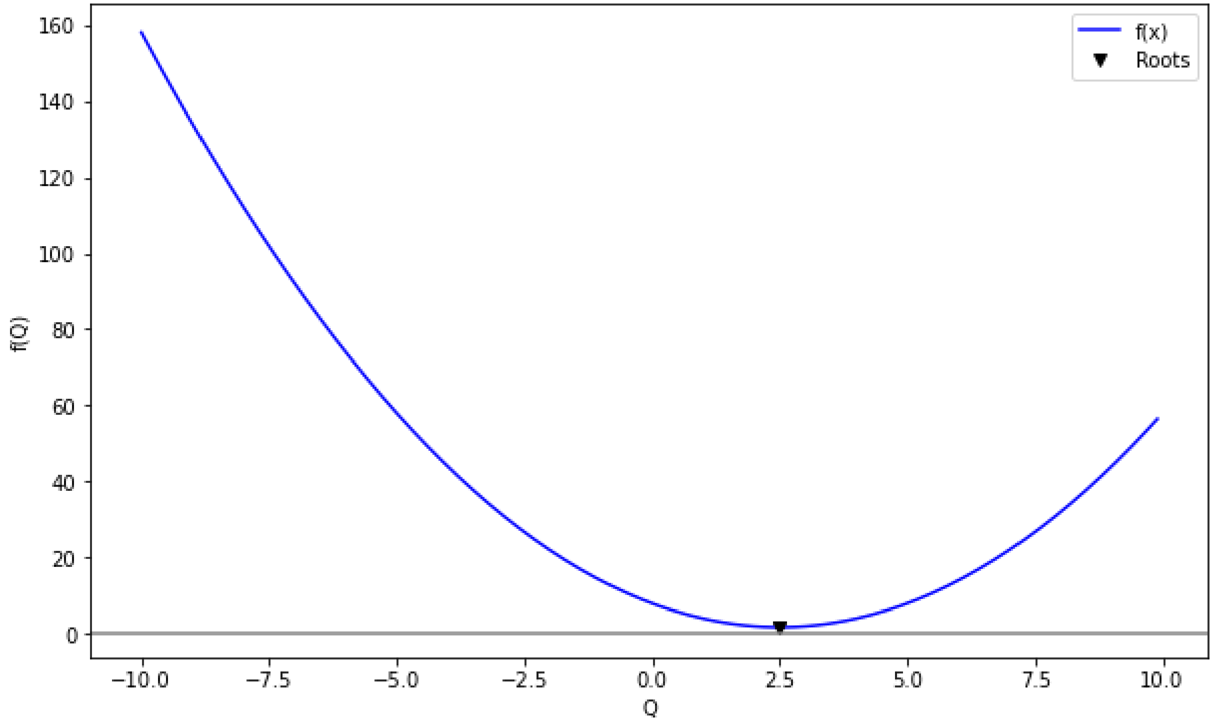
In [7]:
```
root = optimize.root(f, 2)
print("First root found %s" % root.x)
```

```python
root2 = optimize.root(f, 6)
print("Second root found %s" % root2.x)
```

```
First root found [2.50000001]
Second root found [2.49907395]
```

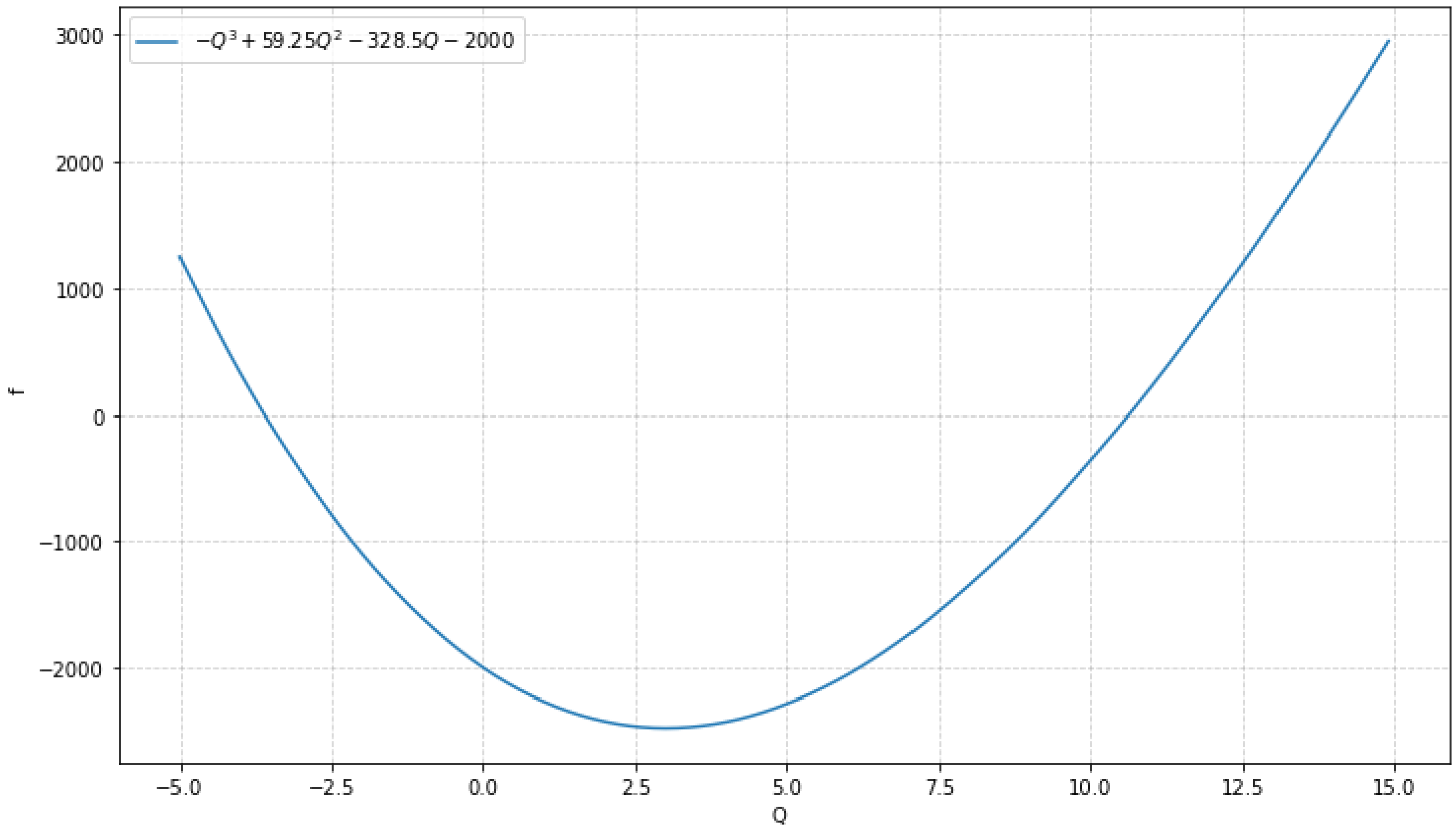
In [8]:

```python
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)


ax.plot(Q1, f(Q1), 'b-', label="f(x)")


roots = np.array([root.x, root2.x])
ax.plot(roots, f(roots), 'kv', label="Roots")


ax.legend(loc='best')
ax.set_xlabel('Q')
ax.set_ylabel('f(Q)')
ax.axhline(0, color='gray')
plt.show()
```



Example 3 --Page 238 --

In [14]:
```
def f(Q):
    return -Q**3 + 59.25*Q**2 - 328.5*Q - 2000

plt.figure(figsize = (12, 7))
Q1 = np.arange(-5, 15, 0.1)
plt.plot(Q1, f(Q1),label ='$-Q^3 + 59.25Q^2 - 328.5Q - 2000$')
plt.xlabel('Q ')
plt.ylabel('f ')
plt.grid(alpha =.6, linestyle ='--')
plt.legend()
plt.show()
```



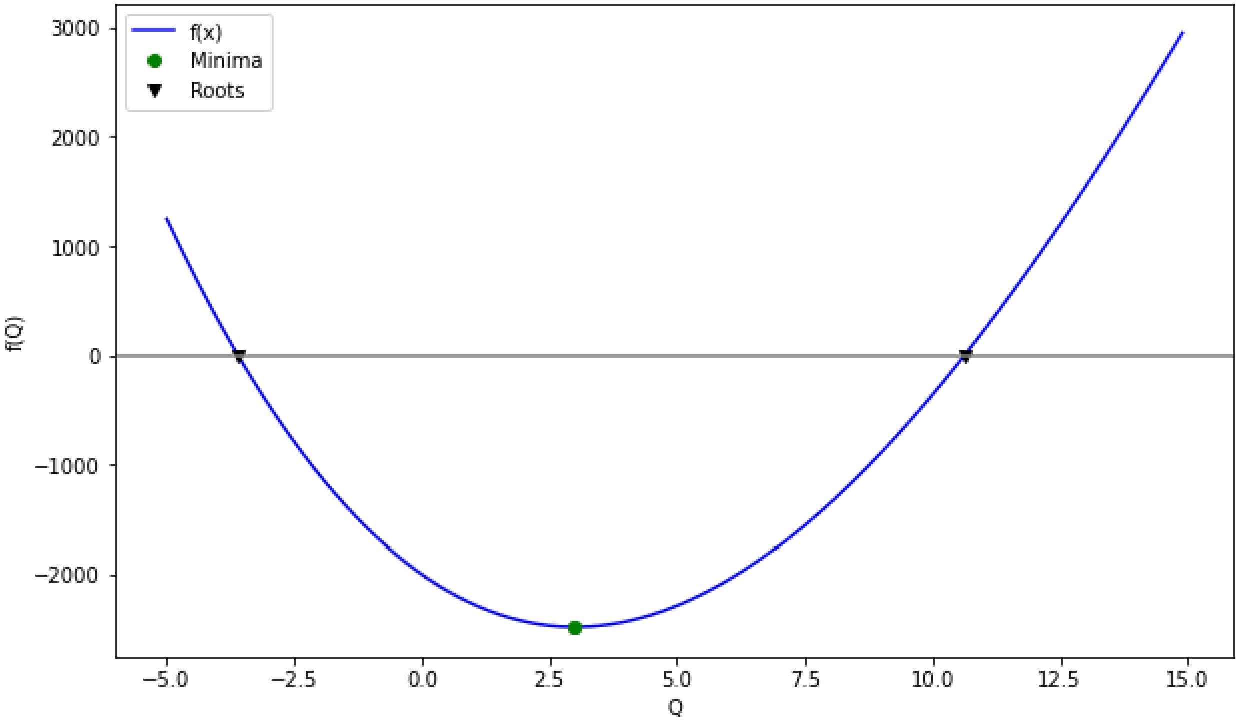
In [16]:

```
root = optimize.root(f, 2)

root2 = optimize.root(f, 6)
grid = (-10, 10, 0.1)
xmin_global = optimize.brute(f, (grid, ))


xmin_local = optimize.fminbound(f, 0, 10)
```

In [18]:

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)


ax.plot(Q1, f(Q1), 'b-', label="f(x)")
```

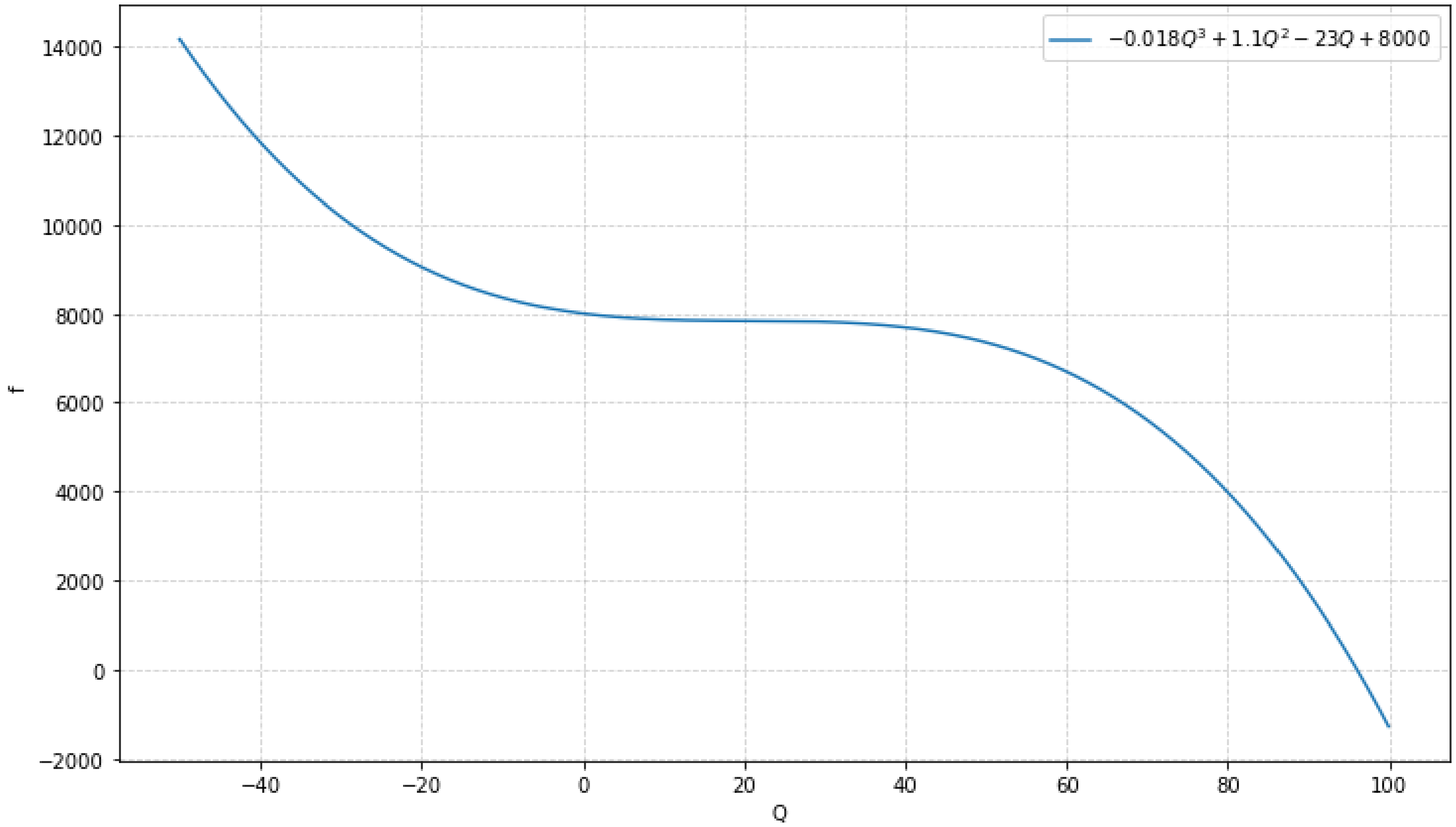
```
xmins = np.array([xmin_global[0], xmin_local])
ax.plot(xmins, f(xmins), 'go', label="Minima")


roots = np.array([root.x, root2.x])
ax.plot(roots, f(roots), 'kv', label="Roots")


ax.legend(loc='best')
ax.set_xlabel('Q')
ax.set_ylabel('f(Q)')
ax.axhline(0, color='gray')
plt.show()
```



Example 4

In [19]:
```
def f(Q):
    return -0.018*Q**3 + 1.1*Q**2 - 23*Q + 8000

plt.figure(figsize = (12, 7))
```

```
Q1 = np.arange(-50, 100, 0.1)
plt.plot(Q1, f(Q1),label ='$-0.018Q^3 + 1.1Q^2 - 23Q + 8000$')
plt.xlabel('Q ')
plt.ylabel('f ')
plt.grid(alpha =.6, linestyle ='--')
plt.legend()
plt.show()
```



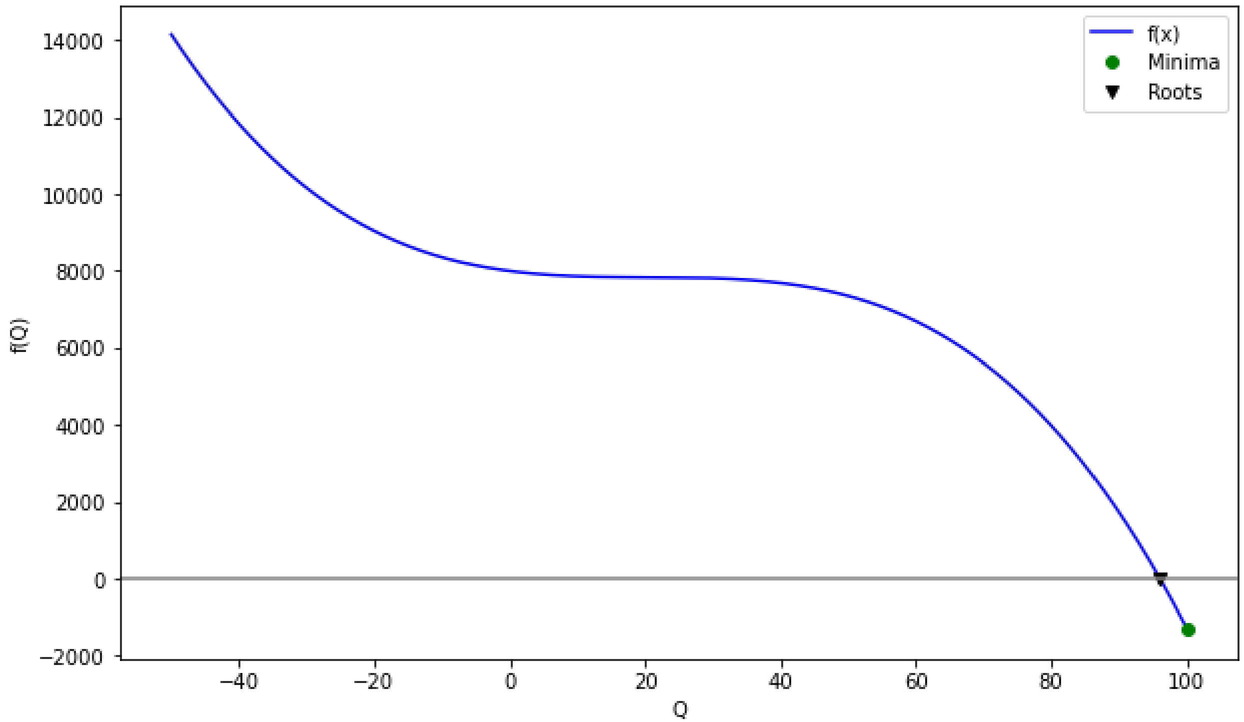
In [20]:

```
root = optimize.root(f, 2)
print("First root found %s" % root.x)
root2 = optimize.root(f, 6)
print("Second root found %s" % root2.x)
grid = (-50, 10, 0.1)


xmin_local = optimize.fminbound(f, 0, 100)
print("Local minimum found %s" % xmin_local)
```

```
First root found [96.01406042]
Second root found [96.01406042]
Local minimum found 99.9999921581193
```

In [21]:

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)

ax.plot(Q1, f(Q1), 'b-', label="f(x)")

xmins = np.array(xmin_local)
ax.plot(xmins, f(xmins), 'go', label="Minima")

roots = np.array([root.x, root2.x])
ax.plot(roots, f(roots), 'kv', label="Roots")

ax.legend(loc='best')
ax.set_xlabel('Q')
ax.set_ylabel('f(Q)')
ax.axhline(0, color='gray')
plt.show()
```



f(x)
Minima
Roots
14000
12000
10000
8000
6000
4000
2000
0
−2000
−40
−20
0
20
40
60
80
100
Q
f(Q)

In [ ]:

Mathematical Economics

Alpha Chiang

Chapter 10 - 11

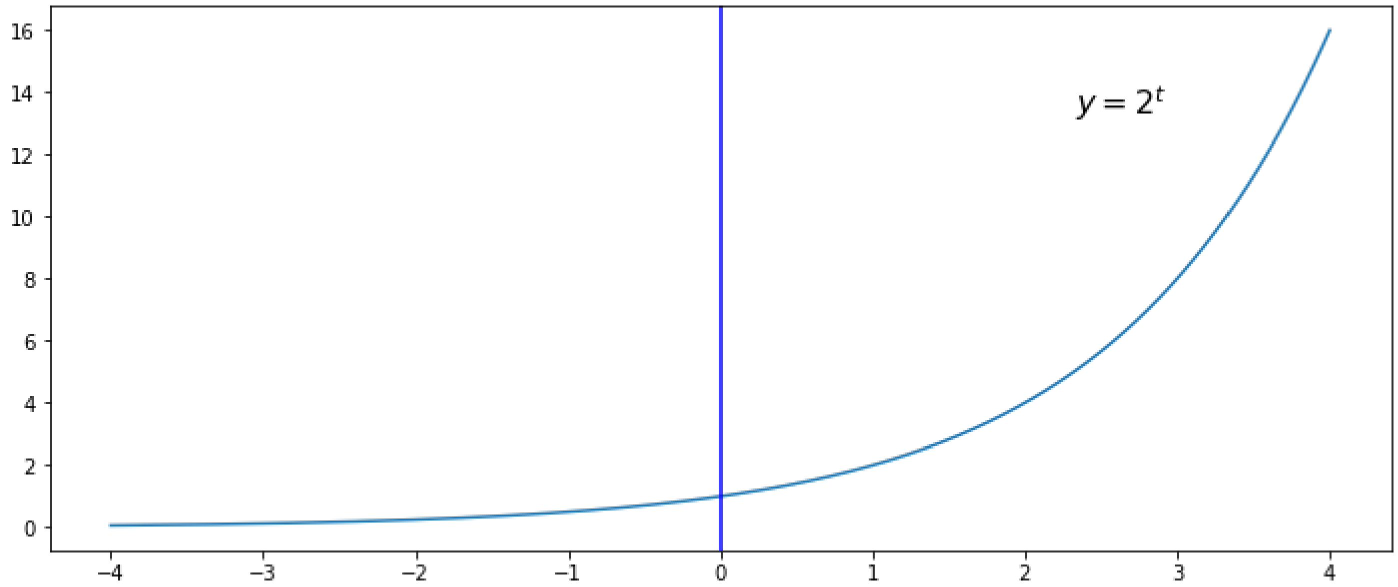
Exponential and Logarithmic Functions

10.1 The Nature of Exponential Functions

In [2]:
```
from sympy import Symbol, dsolve, Function, Derivative, Eq
f = Function("f")
t = Symbol('t')
b = Symbol('b')
eq1 = Eq(f(t), b**t)
eq1
```

Out[2]: $f(t) = b^t$

In [3]:
```
import matplotlib .pyplot as plt
import numpy as np

fig = plt.figure(figsize =(12,5))
ax = fig.add_subplot()
b = np.linspace(-4, 4, 1000)
func, = ax.plot(b, 2**b)
ax.annotate ('$y = 2^t$', xy =(0.8 , 0.8) , fontsize =16, xycoords='axes fraction',
ha='center')
ax.axvline(x = 0, color = 'b', label = 'axvline - full height')
plt.show()
```



Chapter 11

The Case of More than One Choice Variable

Example 1

In [4]:
```
from sympy import Symbol, dsolve, Function, Derivative, Eq
from sympy import exp, sin, sqrt, diff,cos,pi ,latex ,simplify
f = Function("f")
y = Symbol('y')
x = Symbol('x')
def z(x,y):
    return x**3 + 5*x*y - y**2

display(diff(z(x, y), x))
display(diff(z(x, y), y))
display(diff(z(x, y), x, 2))
display(diff(z(x, y), y, 2))
display(diff(z(x, y), x, y))
```

$3x^2 + 5y$

$5x - 2y$

$6x$

-2

5

Example 2

In [5]:
```
y = Symbol('y')
x = Symbol('x')
def z(x,y):
    return x**2*exp(-y)

display(diff(z(x, y), x))
display(diff(z(x, y), y))
display(diff(z(x, y), x, 2))
display(diff(z(x, y), y, 2))
display(diff(z(x, y), x, y))
```

$2xe^{-y}$

$-x^2e^{-y}$

$2e^{-y}$

x^2e^{-y}

$-2xe^{-y}$

Example 4

In [6]:
```
from scipy import optimize
def z(x,y):
    return 8*x**3 + 2*x*y -3*x**2 + y**2 + 1
def f(x,y):
    d1 = diff(z(x, y), x)
    d1_ = diff(z(x,y),x,2)
    d2 = diff(z(x, y), y)
    d2_ = diff(z(x,y),y,2)
    return d1,d1_,d2,d2_
f(x,y)
```

Out[6]: (24*x**2 - 6*x + 2*y, 6*(8*x - 1), 2*x + 2*y, 2)

In [7]:
```
from sympy import *

x, y = symbols('x, y')
eq1 = Eq(24*x**2 - 6*x + 2*y, 0)
eq2 = Eq(2*x + 2*y, 0)

sol = solve([eq1, eq2], [x, y])
sol
```

Out[7]: [(0, 0), (1/3, -1/3)]

Example 5

In [8]:
```
def z(x,y):
    return x + 2*exp(1)*y - exp(x) - exp(2*y)
def f(x,y):
    d1 = diff(z(x, y), x)
    d1_ = diff(z(x,y),x,2)
    d2 = diff(z(x, y), y)
    d2_ = diff(z(x,y),y,2)
    return d1,d1_,d2,d2_
f(x,y)
```

Out[8]: (1 - exp(x), -exp(x), -2*exp(2*y) + 2*E, -4*exp(2*y))

In [9]:
```
x, y = symbols('x, y')
eq1 = Eq(1 - exp(x) + 2*y, 0)
eq2 = Eq(-2*exp(2*y) + 2*exp(1), 0)

sol = solve([eq1, eq2], [x, y])
sol
```

Out[9]: [(log(2), 1/2)]

In [10]:
```
from sympy import *
r = Symbol('r')
def f(r):
    S = Matrix([[ 2 - r, 2],
            [ 2, -1 - r]])
```

```
    return S.det()
f(r)
```

Out[10]: $r^2 - r - 6$

In [11]:

```
roots = solve(r**2 - r - 6,r)
roots
```

Out[11]: [-2, 3]

In [12]:

```
x1 = Symbol('x_1')
x2 = Symbol('x_2')
M1 = Matrix([[-1, 2],
            [2, -4]])
M2 = Matrix((x1,x2))
M1*M2
```

Out[12]: $\begin{bmatrix} -x_1 + 2x_2 \\ 2x_1 - 4x_2 \end{bmatrix}$

11.4 Objective Functions with More than Two Variables

Example 1

In [15]:

```
x1 = Symbol('x_1')
x2 = Symbol('x_2')
x3 = Symbol("x_3")
def z(x1, x2, x3):
    return (2*x1**2 + x1*x2 + 4*x2**2 +
            x1*x3 + x3**2 + 2)

F11 = diff(z(x1,x2,x3), x1,2)
F12 = diff(z(x1,x2,x3), x1,x2)
F13 = diff(z(x1,x2,x3), x1,x3)

F21 = diff(z(x1,x2,x3), x2,x1)
F22 = diff(z(x1,x2,x3), x2,2)
F23 = diff(z(x1,x2,x3), x2,x3)

F31 = diff(z(x1,x2,x3), x3,x1)
F32 = diff(z(x1,x2,x3), x3,x2)
F33 = diff(z(x1,x2,x3), x3,2)
```

```
M1 = Matrix([[F11, F12, F13],
             [F21, F22, F23],
             [F31, F32, F33]])
M1
```

Out[15]:

$$\begin{bmatrix} 4 & 1 & 1 \\ 1 & 8 & 0 \\ 1 & 0 & 2 \end{bmatrix}$$

In [16]:

```
M1.det()
```

Out[16]: 54

In [14]:

```
# Another and easy way

from sympy import symbols, Matrix, Function, simplify, exp, hessian, solve, init_printing
init_printing()

x1, x2,x3 = symbols('x1 x2 x3')
f, g, h = symbols('f g h', cls=Function)

X = Matrix([x1,x2,x3])
f = Matrix([2*x1**2 + x1*x2 + 4*x2**2 +
x1*x3 + x3**2 + 2])
hessianf = simplify(hessian(f, X))
hessianf
```

Out[14]:

$$\begin{bmatrix} 4 & 1 & 1 \\ 1 & 8 & 0 \\ 1 & 0 & 2 \end{bmatrix}$$

11.6 Economic Applications

In [17]:

```
P1 = Symbol('P_1')
P2 = Symbol('P_2')
Q1 = Symbol("Q_1")
Q2 = Symbol("Q_2")
```

```python
def z(P1,P2,Q1,Q2):
    return (P1*Q1 + P2*Q2 - 2*Q1**2 -Q1*Q2
           - 2*Q2**2)
d1 = diff(z(P1,P2,Q1,Q2),Q1)
d2 = diff(z(P1,P2,Q1,Q2),Q2)
display(d1,d2)
```

$$P_1 - 4Q_1 - Q_2$$

$$P_2 - Q_1 - 4Q_2$$

In [18]:

```python
from sympy import symbols, Eq, solve

eq1 = Eq(P1, 4*Q1 + Q2)
eq2 = Eq(P2, Q1 + 4*Q2)
eq3 = Eq(12, 4*Q1 + Q2)
eq4 = Eq(18, Q1 + 4*Q2)

result2 = solve([eq3, eq4],(Q1, Q2))
result = solve([eq1, eq2],(Q1, Q2))
display(result,result2)
```

$$\left\{Q_1 : \frac{4P_1}{15} - \frac{P_2}{15},\ Q_2 : -\frac{P_1}{15} + \frac{4P_2}{15}\right\}$$

$$\{Q_1 : 2,\ Q_2 : 4\}$$

In [19]:

```python
F11 = diff(z(P1,P2,Q1,Q2),Q1,2)
F12 = diff(z(P1,P2,Q1,Q2),Q1,Q2)


F21 = diff(z(P1,P2,Q1,Q2),Q1,Q2)
F22 = diff(z(P1,P2,Q1,Q2),Q1,2)


M1 = Matrix([[F11, F12],
             [F21, F22]])
display(M1,M1.det())
```

$$\begin{bmatrix} -4 & -1 \\ -1 & -4 \end{bmatrix}$$

15

In [20]:
```
P1 = Symbol('P_1')
P2 = Symbol('P_2')
P3 = Symbol('P_3')
Q = Symbol("Q")
Q1 = Symbol("Q_1")
Q2 = Symbol("Q_2")
Q3 = Symbol('Q_2')

def z(Q1):
    R1 = (63*Q1 - 4*Q1**2)
    return R1

def z2(Q3):
    R3 = (75*Q3 - 6*Q3**2)
    return R3

def z3(Q2):
    R2 = (105*Q2 - 5*Q2**2)
    return R2

def z4(Q):
    C = 20 + 15*Q
    return C
d1 = diff(z(Q1),Q1)
d2 = diff(z2(Q3),Q3)
d3 = diff(z3(Q2),Q2)
d4 = diff(z4(Q),Q)
display(d1,d2,d3,d4)
```

$63 - 8Q_1$

$75 - 12Q_2$

$105 - 10Q_2$

15

In [21]:
```
a = np.array([[-8, 0, 0], [0, -10, 0], [0, 0, -12]])
b = np.array([-48, -90, -60])
x = np.linalg.solve(a, b)
x
```

Out[21]: array([6., 9., 5.])

Example 5

In [22]:
```
P = Symbol('P')
L = Symbol('L')
K = Symbol("K")
alpha = Symbol("\\alpha")
w = Symbol("w")
r = Symbol("r")


def p(P,L,K,w,r,alpha):
    return (P*L**(alpha) * K**(alpha) -
           w*L - r*K)

F11 = diff(p(P,L,K,w,r,alpha) ,L,2)
F12 = diff(p(P,L,K,w,r,alpha) ,L,K)


F21 = diff(p(P,L,K,w,r,alpha), K,L)
F22 = diff(p(P,L,K,w,r,alpha),K,2)


M1 = Matrix([[F11, F12],
             [F21, F22]])
display(M1,M1.det())
```

$$\begin{bmatrix} \frac{K^{\alpha}L^{\alpha}P\alpha(\alpha-1)}{L^2} & \frac{K^{\alpha}L^{\alpha}P\alpha^2}{KL} \\ \frac{K^{\alpha}L^{\alpha}P\alpha^2}{KL} & \frac{K^{\alpha}L^{\alpha}P\alpha(\alpha-1)}{K^2} \end{bmatrix}$$

$$-\frac{2K^{2\alpha}L^{2\alpha}P^2\alpha^3 - K^{2\alpha}L^{2\alpha}P^2\alpha^2}{K^2L^2}$$

In [23]:
```
P = Symbol('P')
L = Symbol('L')
K = Symbol("K")
alpha = Symbol("\\alpha")
w = Symbol("w")
r = Symbol("r")
```

```
def p(P,L,K,w,r,alpha):
    return (P*L**(alpha) * K**(alpha) -
           w*L - r*K)
```

In [24]:

```
d1 = diff(p(P,L,K,w,r,alpha) ,L)
d2 = diff(p(P,L,K,w,r,alpha),K)
display(d1,d2)
```

$$\frac{K^{\alpha}L^{\alpha}P\alpha}{L} - w$$

$$-r + \frac{K^{\alpha}L^{\alpha}P\alpha}{K}$$

In [25]:

```
from sympy import symbols, Eq, solve

eq1 = Eq(w, (K**(alpha) * L**(alpha)*P*alpha)/L)
eq2 = Eq(r, (K**(alpha) * L**(alpha)*P*alpha)/K)

result = solve([eq1, eq2],(K, L))
display(result)
```

$$\left[\left(\left(\frac{w\left(\left(\frac{1}{P\alpha}\right)^{\frac{1}{2\alpha-1}}\left(rw^{\frac{1-\alpha}{\alpha}}\right)^{\frac{\alpha}{2\alpha-1}}\right)^{1-\alpha}}{P\alpha}\right)^{\frac{1}{\alpha}}, \left(\frac{1}{P\alpha}\right)^{\frac{1}{2\alpha-1}}\left(rw^{\frac{1-\alpha}{\alpha}}\right)^{\frac{\alpha}{2\alpha-1}}\right)\right]$$

Furkan zengin

Mathematical Economics

Alpha Chiang

Chapter 12

Optimization with Equality Constraints

Lagrange-Multiplier Method

In [5]:

```
from sympy import *
import numpy as np
from scipy.linalg import cholesky, solve_triangular
from scipy.linalg import cho_solve, cho_factor
from scipy.linalg import solve
from scipy.optimize import minimize
from sympy import Symbol, dsolve, Function, Derivative, Eq
x1 = Symbol("x_1")
x2 = Symbol('x_2')
Z = Symbol("Z")
lamd = Symbol("\\lambda")
eq1 = Eq(Z, x1*x2 + 2*x1 + lamd*(60 - 4*x1 -2*x2))
display(eq1)




def f(x):
    return -(x[0]*x[1] + 2*x[0])

cons = ({'type': 'eq',
         'fun' : lambda x: np.array([4*x[0] + 2*x[1] - 60])})

x0 = np.array([1,1,4])
res = minimize(f, x0, constraints=cons)
res
```

$$Z = \lambda\left(-4x_1 - 2x_2 + 60\right) + x_1x_2 + 2x_1$$

Out[5]:

```
     fun: -127.99999999999983
     jac: array([-16.        , -7.99999809,   0.        ])
 message: 'Optimization terminated successfully'
    nfev: 16
     nit: 4
    njev: 4
  status: 0
 success: True
       x: array([ 7.9999997, 14.0000006,  4.       ])
```

In [8]:

```
res = minimize(f, x0, constraints=cons,method="trust-constr")
```

Example 1

In [9]:

```
y = Symbol('y')
x = Symbol('x')
eq1 = Eq(Z, x*y + lamd*(6 - x - y))
display(eq1)

def f(x):
    return -(x[0]*x[1])

cons = ({'type': 'eq',
         'fun' : lambda x: np.array([x[0] + x[1] - 6])})

x0 = np.array([1,1,3])
res = minimize(f, x0, constraints=cons)
res
```

$$Z = \lambda\left(-x - y + 6\right) + xy$$

Out[9]:

```
     fun: -8.999999999999998
     jac: array([-3., -3.,  0.])
 message: 'Optimization terminated successfully'
    nfev: 8
     nit: 2
    njev: 2
  status: 0
 success: True
       x: array([3., 3., 3.])
```

Example 2

In [10]:
```
x1 = Symbol("x_1")
x2 = Symbol('x_2')
Z = Symbol("Z")
lamd = Symbol("\\lambda")
eq1 = Eq(Z, x1**2 + x2**2   + lamd*(2 - x1  - 4*x2))
display(eq1)


def f(x):
    return (x[0]**2 + x[1]**2)

cons = ({'type': 'eq',
         'fun' : lambda x: np.array([x[0] + 4*x[1] - 2])})

x0 = np.array([1,1,1])
res = minimize(f, x0, constraints=cons)
res
```

$$Z = \lambda\left(-x_1 - 4x_2 + 2\right) + x_1^2 + x_2^2$$

Out[10]:
```
     fun: 0.2352941176470589
     jac: array([0.23529412, 0.94117649, 0.        ])
 message: 'Optimization terminated successfully'
    nfev: 16
     nit: 4
    njev: 4
  status: 0
 success: True
       x: array([0.11764705, 0.47058824, 1.        ])
```

Example 2 --Using derivatives and matrices--

In [11]:
```
x1 = Symbol("x_1")
x2 = Symbol('x_2')
Z = Symbol("Z")
lamd = Symbol("\\lambda")

def z(x1,x2,lamd):
    return x1**2 + x2**2   + lamd*(2 - x1  - 4*x2)
def Z(x1,x2,lamd):
    dZ1 = diff(z(x1,x2,lamd),x1)
    dZ2 = diff(z(x1,x2,lamd),x2)
    dZ3 = diff(z(x1,x2,lamd),lamd)
    return dZ1,dZ2,dZ3
Z(x1,x2,lamd)
```

Out[11]: (-\lambda + 2*x_1, -4*\lambda + 2*x_2, -x_1 - 4*x_2 + 2)

In [12]:
```
# We can build a matrix
A = np.array([
    [2, 0, -1],
    [0, 2, -4],
    [-1, -4, 0]
])
b = np.array([0, 0, -2])
solve(A, b)
```

Out[12]: array([0.11764706, 0.47058824, 0.23529412])

In [13]:
```
x1 = Symbol("x_1")
x2 = Symbol('x_2')
Z = Symbol("Z")
lamd = Symbol("\\lambda")

def z(x1,x2,lamd):
    return x1**2 + x2**2   + lamd*(2 - x1  - 4*x2)
def g(x1,x2):
    return x1 + 4*x2 - 2
def Z(x1,x2,lamd):
    dZ1 = diff(z(x1,x2,lamd),x1,2)
    dZ2 = diff(z(x1,x2,lamd),x2,2)
    dZ3 = diff(z(x1,x2,lamd),lamd,2)
    dZ4 = diff(g(x1,x2),x1)
    dZ5 = diff(g(x1,x2),x2)
    return dZ1,dZ2,dZ3,dZ4,dZ5
Z(x1,x2,lamd)
# By using these values we can build a hessian matrix
# and we can check maximum and minimum values
```

Out[13]: (2, 2, 0, 1, 4)

In [14]:
```
x1 = Symbol("x_1")
x2 = Symbol('x_2')
U = Symbol("U")
lamd = Symbol("\\lambda")
B = Symbol("B")
r = Symbol("r")
eq1 = Eq(U, x1*x2   + lamd*(B - x1 - (x2/(1+r)) ))
```

```python
display(eq1)


def u(x1,x2,lamd,B,r):
    return x1*x2 + lamd*(B - x1 - (x2/(1+r)))
def U(x1,x2,lamd,B,r):
    dU1 = diff(u(x1,x2,lamd,B,r),x1)
    dU2 = diff(u(x1,x2,lamd,B,r),x2)
    dU3 = diff(u(x1,x2,lamd,B,r),lamd)
    return dU1,dU2,dU3
display(U(x1,x2,lamd,B,r))
a = lamd/(lamd/(1+r))
display(a)
```

$$U = \lambda\left(B - x_1 - \frac{x_2}{r+1}\right) + x_1x_2$$

(-\lambda + x_2, -\lambda/(r + 1) + x_1, B - x_1 - x_2/(r + 1))

$r + 1$

In [15]:

```python
x1 = Symbol("x_1")
x2 = Symbol('x_2')
U = Symbol("U")
lamd = Symbol("\\lambda")
B = Symbol("B")
r = Symbol("r")
eq1 = Eq(U, x1*x2   + lamd*(B - x1 - (x2/(1+r)) ))
display(eq1)


def u(x1,x2,lamd,B,r):
    return x1*x2 + lamd*(B - x1 - (x2/(1+r)))

def g(x1,x2,B,r):
    return x1 + x2/(1+r) - B


def U(x1,x2,lamd,B,r):
    dU1 = diff(u(x1,x2,lamd,B,r),x1,2)
    dU2 = diff(u(x1,x2,lamd,B,r),x2,2)
    dU3 = diff(u(x1,x2,lamd,B,r),x1,x2)
    dU4 = diff(u(x1,x2,lamd,B,r),lamd,2)
    dg1 = diff(g(x1,x2,B,r),x1)
```

```
    dg2 = diff(g(x1,x2,B,r),x2)

    return dU1,dU2,dU3,dU4,dg1,dg2
display(U(x1,x2,lamd,B,r)) # can be used again to build the hessian
```

$$U = \lambda \left(B - x_1 - \frac{x_2}{r+1}\right) + x_1 x_2$$

(0, 0, 1, 0, 1, 1/(r + 1))

In [17]:

```
dU1 = diff(u(x1,x2,lamd,B,r),x1,2)
dU2 = diff(u(x1,x2,lamd,B,r),x2,2)
dU3 = diff(u(x1,x2,lamd,B,r),x1,x2)
dU4 = diff(u(x1,x2,lamd,B,r),lamd,2)
dg1 = diff(g(x1,x2,B,r),x1)
dg2 = diff(g(x1,x2,B,r),x2)
M1 = Matrix([[dU1, -dU3, -dg2],
 [-dU3, dU2, dU3],
 [-dg2, dU3, dU1]])
display(M1)
display(M1.det())
```

$$\begin{bmatrix} 0 & -1 & -\frac{1}{r+1} \\ -1 & 0 & 1 \\ -\frac{1}{r+1} & 1 & 0 \end{bmatrix}$$

$$\frac{2}{r+1}$$

Plots of Examples

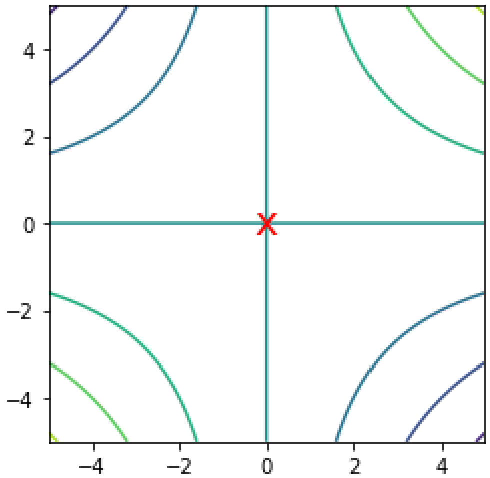
Using https://www2.hawaii.edu/~jonghyun/courses.html

Example 1

In [18]:

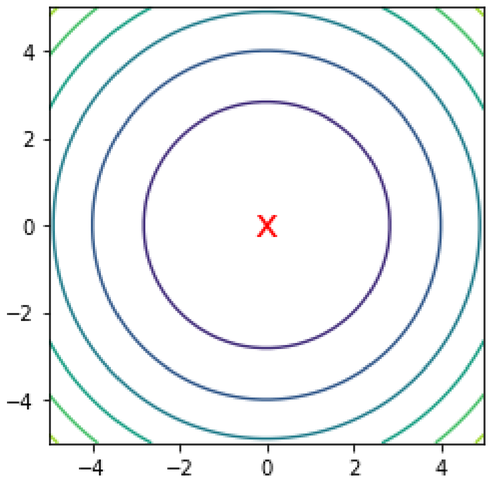
```
import scipy.optimize as opt
import numpy as np
import matplotlib.pyplot as plt
def func(x):
    return x[0]*x[1]
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
```

```
X,Y = np.meshgrid(x,y)
XY = np.vstack([X.ravel(), Y.ravel()])
Z = func(XY).reshape(50,50)
plt.contour(X, Y, Z)
plt.text(0, 0, 'x', va='center', ha='center',
color='red', fontsize=20)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```



In [19]:

```
import scipy.optimize as opt
import numpy as np
import matplotlib.pyplot as plt
def func(x):
    return x[0]**2 + x[1]**2
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X,Y = np.meshgrid(x,y)
XY = np.vstack([X.ravel(), Y.ravel()])
Z = func(XY).reshape(50,50)
plt.contour(X, Y, Z)
plt.text(0, 0, 'x', va='center', ha='center',
color='red', fontsize=20)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```



EXERCISE 12.5 --Q1--

In [20]:
```
x = Symbol("x")
y = Symbol('y')
U = Symbol("U")
lamd = Symbol("\\lambda")
eq1 = Eq(U, (x+2)*(y+1)   + lamd*(130 -4*x  - 6*y))
display(eq1)


def f(x):
    return -((x[0]+2) * (x[1]+1))

cons = ({'type': 'eq',
         'fun' : lambda x: np.array([4*x[0] + 6*x[1] - 130])})

x0 = np.array([1,1,3])
res = minimize(f, x0, constraints=cons)
res
```

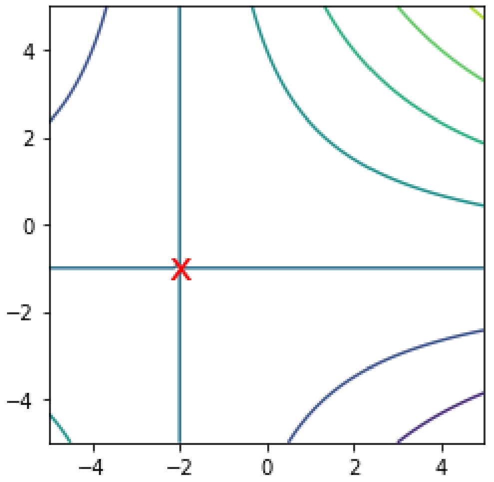
$$U = \lambda\left(-4x - 6y + 130\right) + \left(x + 2\right)\left(y + 1\right)$$

Out[20]:
```
     fun: -215.99999999999963
     jac: array([-12., -18.,   0.])
 message: 'Optimization terminated successfully'
    nfev: 16
     nit: 4
    njev: 4
```

```
 status: 0
success: True
      x: array([16.0000008 , 10.99999947,  3.        ])
```

In [21]:

```
def func(x):
    return (x[0] +2)*(x[1] + 1)
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X,Y = np.meshgrid(x,y)
XY = np.vstack([X.ravel(), Y.ravel()])
Z = func(XY).reshape(50,50)
plt.contour(X, Y, Z)
plt.text(-2, -1, 'x', va='center', ha='center',
color='red', fontsize=20)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```



Furkan Zengin

Mathematical Economics

Alpha Chiang

Chapter 13

Further Topics in Optimization

Example 1

In [2]:
```
from sympy import *
import numpy as np
from sympy import Symbol, dsolve, Function, Derivative, Eq
from scipy.optimize import minimize, rosen, rosen_der
x = Symbol("x")
y = Symbol('y')
Z = Symbol("Z")
lamd1 = Symbol("\\lambda_1")
lamd2 = Symbol("\\lambda_2")
eq1 = Eq(Z, x*y + lamd1*(100 - x - y) + lamd2*(40 - x))
display(eq1)

def f(x):
    return (x[0]*x[1])

cons = ({'type': 'ineq',
         'fun' : lambda x: np.array([x[0] + x[1] - 100, x[0] - 40])})
x0 = np.array([2,2,1])
res = minimize(f, x0, constraints=cons)
res
```

$$Z = \lambda_1 \left(-x - y + 100\right) + \lambda_2 \left(40 - x\right) + xy$$

Out[2]:
```
     fun: 2500.000000003316
     jac: array([50., 50.,  0.])
 message: 'Optimization terminated successfully'
    nfev: 8
     nit: 2
```

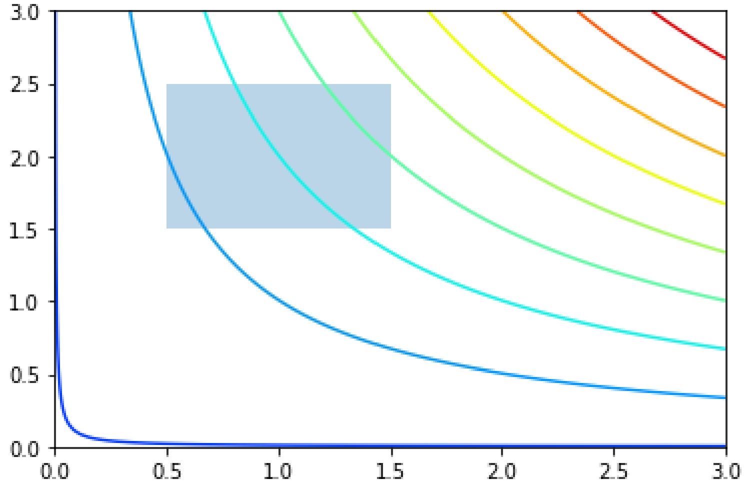
```
    njev: 2
  status: 0
 success: True
       x: array([50., 50.,  1.])
```

In [3]:

```
%matplotlib inline
import scipy.linalg as la
import numpy as np
import scipy.optimize as opt
import matplotlib.pyplot as plt
import pandas as pd
x = np.linspace(0, 3, 100)
y = np.linspace(0, 3, 100)
X, Y = np.meshgrid(x, y)
Z = f(np.vstack([X.ravel(), Y.ravel()])).reshape((100,100))
plt.contour(X, Y, Z, np.arange(-1.99,10, 1), cmap='jet');
plt.fill([0.5,0.5,1.5,1.5], [2.5,1.5,1.5,2.5], alpha=0.3)
plt.axis([0,3,0,3])
```

Out[3]:

```
(0.0, 3.0, 0.0, 3.0)
```



In [4]:

```
# Another way for example 1
def func(x, sign=1.0):
    return sign*(x[0]*x[1])
def func_deriv(x, sign=1.0):
    dfdx0 = sign*(x[1])
    dfdx1 = sign*(x[0])
```

```python
    return np.array([ dfdx0, dfdx1 ])
# take the derivative of objective function
```

In [5]:
```python
cons = ({'type': 'ineq',
         'fun' : lambda x: np.array([x[0] + x[1] - 100]),
         'jac' : lambda x: np.array([1,1])},
        {'type': 'ineq',
         'fun' : lambda x: np.array([x[0] - 40]),
         'jac' : lambda x: np.array([1, 0])})
# for jac we take derivatives of constraints
```

In [6]:
```python
res = minimize(func, [10,10],  jac=func_deriv,
               constraints=cons, method='SLSQP', options={'disp': True})

print(res.x)
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2500.000000003316
            Iterations: 2
            Function evaluations: 2
            Gradient evaluations: 2
[50. 50.]
```

Example 2

In [7]:
```python
x1 = Symbol("x_")
x2 = Symbol('x_2')
Z = Symbol("Z")
lamd1 = Symbol("\\lambda_1")
lamd2 = Symbol("\\lambda_2")
eq1 = Eq(Z, (x1-4)**2 + (x2-4)**2 +
         lamd1*(6 - 2*x1 - 3*x2) + lamd2*(-12 + 3*x1 + 2*x2))
display(eq1)

def f(x):
    return ((x[0] - 4)**2 + (x[1] - 4)**2)

cons = ({'type': 'ineq',
         'fun' : lambda x: np.array([2*x[0] + 3*x[1] - 6,
                                     -3*x[0] - 2*x[1] +12])})

x0 = np.array([2,2,1])
```

```
res = minimize(f, x0, constraints=cons)
res
```

$$Z = \lambda_1 \left(-2x - 3x_2 + 6\right) + \lambda_2 \left(3x + 2x_2 - 12\right) + \left(x - 4\right)^2 + \left(x_2 - 4\right)^2$$

Out[7]:

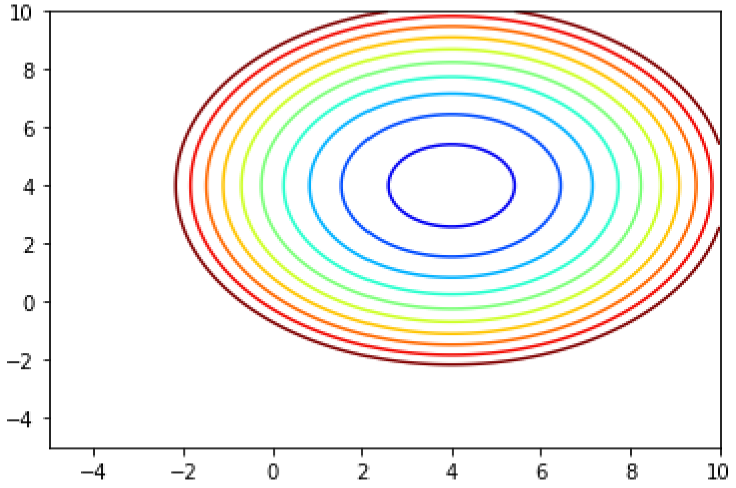
```
     fun: 4.92307692307701
     jac: array([-3.69230771, -2.46153849,  0.        ])
 message: 'Optimization terminated successfully'
    nfev: 16
     nit: 4
    njev: 4
  status: 0
 success: True
       x: array([2.15384616, 2.76923076, 1.        ])
```

In [8]:

```
x = np.linspace(-5, 10, 100)
y = np.linspace(-5, 10, 100)
X, Y = np.meshgrid(x, y)
Z = f(np.vstack([X.ravel(), Y.ravel()])).reshape((100,100))
plt.contour(X, Y, Z, np.arange(-1.99,40, 4), cmap='jet');
plt.axis([-5,10,-5,10])
```

Out[8]:

```
(-5.0, 10.0, -5.0, 10.0)
```



13.2 The Constraint Qualification

Example 3

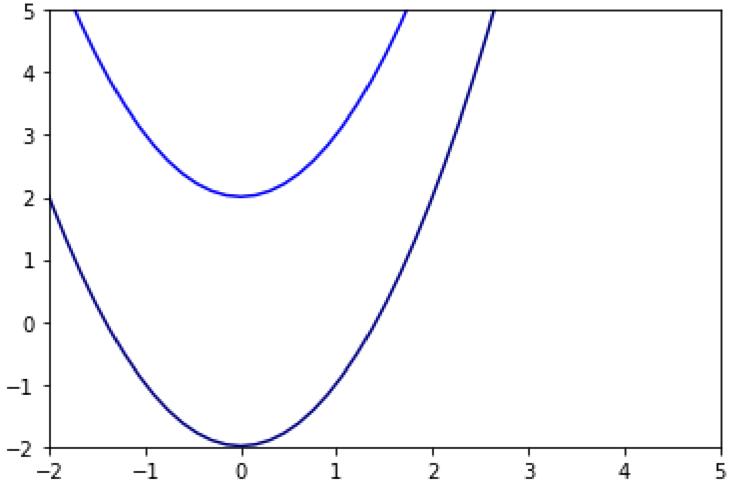
In [9]:
```python
x1 = Symbol("x_1")
x2 = Symbol('x_2')
Z = Symbol("Z")
lamd1 = Symbol("\\lambda_1")
lamd2 = Symbol("\\lambda_2")
eq1 = Eq(Z, x2 - x1**2 +
         lamd1*(10 - x1**2 - x2)**3 + lamd2*(-2 + x1))
display(eq1)

def f(x):
    return (x[1] - x[0]**2)

cons = ({'type': 'ineq',
         'fun' : lambda x: np.array([-(10 - x[0]**2 - x[1])**3,
                                    -x[0] + 2])})

x0 = np.array([2,2,1])

res = minimize(f, x0, constraints=cons)
res
```

$$Z = \lambda_1 \left(-x_1^2 - x_2 + 10\right)^3 + \lambda_2 \left(x_1 - 2\right) - x_1^2 + x_2$$

Out[9]:
```
     fun: 1.9999981835694722
     jac: array([-4.,  1.,  0.])
 message: 'Optimization terminated successfully'
    nfev: 145
     nit: 36
    njev: 36
  status: 0
 success: True
       x: array([2.        , 5.99999818, 1.        ])
```

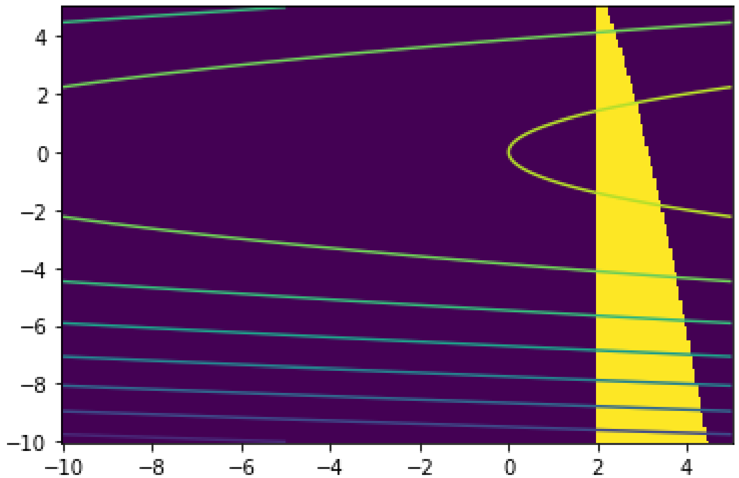
In [11]:
```python
x = np.linspace(-5, 10, 100)
y = np.linspace(-5, 10, 100)
X, Y = np.meshgrid(x, y)
Z = f(np.vstack([X.ravel(), Y.ravel()])).reshape((100,100))
plt.contour(X, Y, Z, np.arange(-1.99,40, 4), cmap='jet');
plt.axis([-2,5,-2,5])
```

Out[11]:
```
(-2.0, 5.0, -2.0, 5.0)
```



In [12]:

```
X, Y = np.meshgrid(np.linspace(-10, 5, 256)
                   , np.linspace(-10, 5, 256))

plt.figure()
plt.pcolormesh(X, Y, (-(10 - X**2 - Y)**3 <= 0) &
               (-X + 2 <= 0),shading='auto')
plt.contour(X, Y, X - Y**2)
plt.show()
```

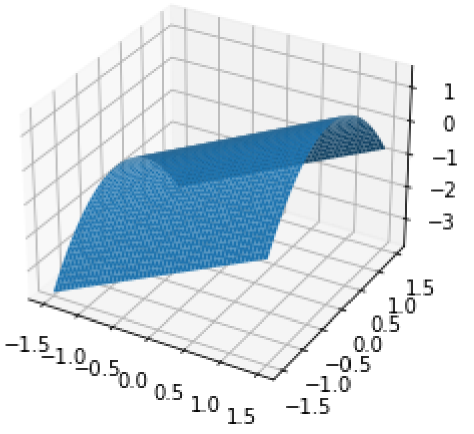


In [13]:

```
import numpy as np
```

```
x = np.linspace(-1.5, 1.5)

[X, Y] = np.meshgrid(x, x)

import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.gca(projection='3d')

ax.plot_surface(X, Y, X - Y**2)
```

Out[13]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x1a636a83190>



13.3 Economic Applications

Example 1

In [14]:

```
x = Symbol("x")
y = Symbol('y')
Z = Symbol("Z")
lamd1 = Symbol("\\lambda_1")
lamd2 = Symbol("\\lambda_2")
eq1 = Eq(Z, x*y**2 +
          lamd1*(100 - x -y) + lamd2*(120 - 2*x -y))
display(eq1)
```

```python
def f(x):
    return x[0]*(x[1]**2)

cons = ({'type': 'ineq',
         'fun' : lambda x: np.array([(x[0] + x[1] - 100)
                                    ,(2*x[0] + x[1] - 120)])})

x0 = np.array([10,20,0])

res = minimize(f, x0, constraints=cons)
res
# this cannot be solved by these method
# we should use derivatives and matrices below
```

$$Z = \lambda_1 \left(-x - y + 100\right) + \lambda_2 \left(-2x - y + 120\right) + xy^2$$

Out[14]:

```
     fun: 3.139129143754785e-07
     jac: array([ 2.93285041e-09, -1.15913628e-02,  0.00000000e+00])
 message: 'Optimization terminated successfully'
    nfev: 46
     nit: 10
    njev: 10
  status: 0
 success: True
       x: array([ 1.07033402e+02, -5.41557940e-05,  0.00000000e+00])
```

In [15]:

```python
x = Symbol("x")
y = Symbol('y')
Z = Symbol("Z")
lamd1 = Symbol("\\lambda_1")
lamd2 = Symbol("\\lambda_2")
def z(x,y,lamd1,lamd2):
    return x*y**2+lamd1*(100- x -y)+lamd2*(120 - 2*x-y)

def Z(x,y,lamd1,lamd2):
    dZ1 = diff(z(x,y,lamd1,lamd2),x)
    dZ2 = diff(z(x,y,lamd1,lamd2),y)
    dZ3 = diff(z(x,y,lamd1,lamd2),lamd1)
    dZ4 = diff(z(x,y,lamd1,lamd2),lamd2)
    return dZ1,dZ2,dZ3, dZ4
Z(x,y,lamd1,lamd2)
# Assume lamd1 = 0
```

Out[15]:

```
(-\lambda_1 - 2*\lambda_2 + y**2,
 -\lambda_1 - \lambda_2 + 2*x*y,
```

```
-x - y + 100,
-2*x - y + 120)
```

first install gekko from https://gekko.readthedocs.io/en/latest/

In [16]:

```
# first install gekko from https://gekko.readthedocs.io/en/latest/
# a1 = x, a2 = y, a3 = lambda2
from gekko import GEKKO
m = GEKKO()
a1,a2,a3 = [m.Var(1) for i in range(3)]
m.Equations([a2**2 - 2*a3==0,\
             2*a1*a2 - a3==0,\
             120 - 2*a1 - a2==0])
m.solve(disp=False)
print(a1.value,a2.value,a3.value)
```

```
[20.0] [79.999999999] [3199.9999999]
```

Furkan zengin

Mathematical Economics

Alpha Chiang

Chapter 14

In [4]:
```
from sympy import Symbol, exp, sin, sqrt, diff,sqrt,Function
x = Symbol('x')
y = Symbol('y')
from sympy import integrate
expr1 = exp(x*y)
display(expr1)
import sympy as sy
sy.init_printing()
I = sy.Integral(expr1,(x,0,5))
I
```

e^{xy}

Out[4]:

$$\int\limits_{0}^{5} e^{xy}\, dx$$

In [2]:
```
integrate(I,(x,0,5))
```

Out[2]:

$$\begin{cases} \frac{5e^{5y}}{y} - \frac{5}{y} & \text{for } y > -\infty \wedge y < \infty \wedge y \neq 0 \\ 25 & \text{otherwise} \end{cases}$$

In [9]:
```
H = Symbol('H')
t = Symbol('t')
expr2 = t**(-1/2)
display(expr2)
I2 = sy.Integral(expr2,(t,0,5))
I2
```

$t^{-0.5}$

Out[9]:

$$\int\limits_{0}^{5} t^{-0.5}\, dt$$

In [8]:

```
integrate(I2,(t,0,1))
```

Out[8]: 4.47213595499958

In [13]:

```
expr3 = sqrt(x**3)
display(expr3)
I3 = sy.Integral(expr3,(x,0,5))
I3
```

$\sqrt{x^3}$

Out[13]:

$$\int\limits_{0}^{5} \sqrt{x^3}\, dx$$

In [11]:

```
integrate(I3,(x,0,1))
```

Out[11]: $10\sqrt{5}$

In [16]:

```
expr4 = 2*exp(2*x) + (14*x/(7*x**2 + 5))
display(expr4)
I4= sy.Integral(expr4,(x,0,1))
I4
```

$$\frac{14x}{7x^2+5} + 2e^{2x}$$

Out[16]:

$$\int\limits_{0}^{1} \left(\frac{14x}{7x^2+5} + 2e^{2x}\right) dx$$

In [15]:

```
integrate(I4,(x,0,1))
```

Out[15]: $-\log(5) - 1 + \log(12) + e^2$

In [17]:
```
expr5 = exp(x)*x
expr5
I5= sy.Integral(expr5,(x,0,1))
I5
```

Out[17]: $$\int_0^1 xe^x\,dx$$

In [18]:
```
integrate(I5,(x,0,5))
```

Out[18]: 5

In [19]:
```
a = Symbol('a')
b = Symbol('b')
expr6 = (2*a*x + b)*(a*x**2 + b*x)**7
expr6
```

Out[19]: $(2ax + b)\left(ax^2 + bx\right)^7$

In [20]:
```
I6= sy.Integral(expr6,(x,0,1))
I6
```

Out[20]: $$\int_0^1 (2ax + b)\left(ax^2 + bx\right)^7\,dx$$

In [21]:
```
integrate(I6,(x,0,1))
```

Out[21]: $$\frac{a^8}{8} + a^7b + \frac{7a^6b^2}{2} + 7a^5b^3 + \frac{35a^4b^4}{4} + 7a^3b^5 + \frac{7a^2b^6}{2} + ab^7 + \frac{b^8}{8}$$

In [22]:
```
k = Symbol('k')
i = Symbol('i')
f = Function("f")
expr7 = f(x)
expr7
```

Out[22]: $f(x)$

In [23]:
```
from sympy.abc import i, k, m, n, x
from sympy import Sum, factorial, oo, IndexedBase, Function
```

In [24]:
```
I7= Sum(k, (k, 1, n))*sy.Integral(expr7,(x,0,n))
I7
```

Out[24]:

$$\left(\int\limits_{0}^{n} f(x)\, dx\right) \sum_{k=1}^{n} k$$

In [25]:
```
integrate(I7,(x,0,n))
```

Out[25]:

$$n \left(\int\limits_{0}^{n} f(x)\, dx\right) \sum_{k=1}^{n} k$$

In [26]:
```
expr8 = k*exp(x)
expr8
```

Out[26]: ke^{x}

In [27]:
```
I8= sy.Integral(expr8,(x,a,b))
I8
```

Out[27]:

$$\int\limits_{a}^{b} ke^{x}\, dx$$

In [28]:
```
integrate(I8,(x,a,b))
```

Out[28]: $-a\left(-ke^{a} + ke^{b}\right) + b\left(-ke^{a} + ke^{b}\right)$

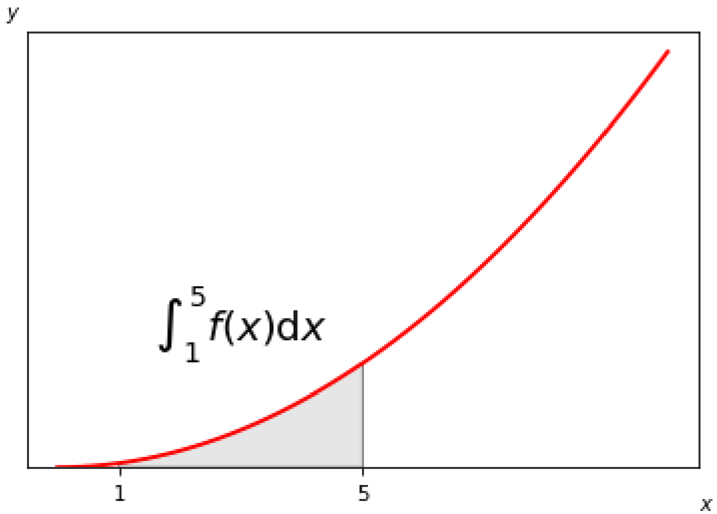
In [29]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
```

In [31]:

```
def f1(x):
    return x**2

a, b = 1, 5  # integral limits
x = np.linspace(0, 10)
y = f1(x)
```

In [32]:

```
fig, ax = plt.subplots()
ax.plot(x, y, 'r', linewidth=2)
ax.set_ylim(bottom=0)


ix = np.linspace(1, 5)
iy = f1(ix)
verts = [(1, 0), *zip(ix, iy), (5, 0)]
poly = Polygon(verts, facecolor='0.9', edgecolor='0.5')
ax.add_patch(poly)

ax.text(0.5 * (1 + 5), 30, r"$\int_1^5 f(x)\mathrm{d}x$",
        horizontalalignment='center', fontsize=20)

fig.text(0.9, 0.05, '$x$')
fig.text(0.1, 0.9, '$y$')

ax.xaxis.set_ticks_position('bottom')

ax.set_xticks((1, 5))
ax.set_xticklabels(('$1$', '$5$'))
ax.set_yticks([])

plt.show()
```



In [35]:
```
from sympy.abc import i, k, m, n, x
expr9 = 1/x**2
expr9
```

Out[35]: $\frac{1}{x^2}$

In [36]:
```
I9= sy.Integral(expr9,(x,1,oo))
I9
```

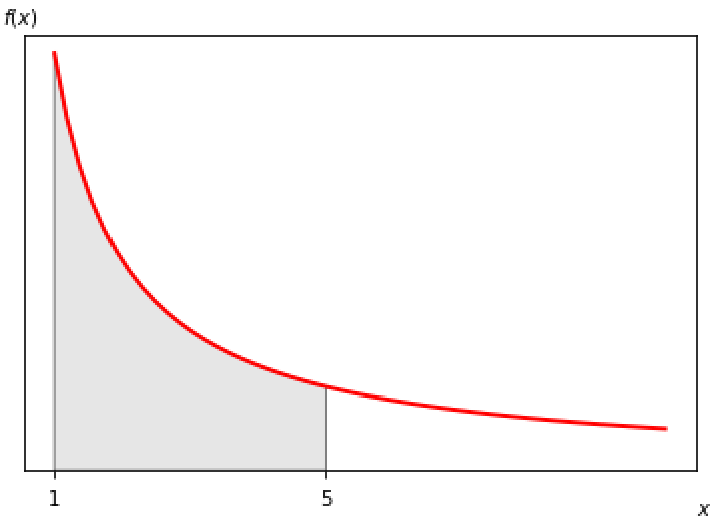
Out[36]: $\int_1^\infty \frac{1}{x^2}\,dx$

In [37]:
```
integrate(I9,(x,1,oo))
```

Out[37]: ∞

In [38]:
```
def f2(x):
    return 1/x

a, b = 1, 5  # integral limits
x = np.linspace(1, 10)
y = f2(x)
```

In [39]:
```
fig, ax = plt.subplots()
ax.plot(x, y, 'r', linewidth=2)
ax.set_ylim(bottom=0)

ix = np.linspace(1, 5)
iy = f2(ix)
verts = [(1, 0), *zip(ix, iy), (5, 0)]
poly = Polygon(verts, facecolor='0.9', edgecolor='0.5')
ax.add_patch(poly)

fig.text(0.9, 0.05, '$x$')
fig.text(0.1, 0.9, '$f(x)$')

ax.xaxis.set_ticks_position('bottom')

ax.set_xticks((1, 5))
ax.set_xticklabels(('$1$', '$5$'))
ax.set_yticks([])

plt.show()
```



In [40]:
```
from sympy.abc import i, k, m, n, x
from sympy import Sum, factorial, oo, IndexedBase, Function
```

In [41]:
```
R = Function("R")
```

```
t = Symbol("t")
r = Symbol("r")
D = Symbol("D")
from sympy import Product, oo
expr10 = R(t)*exp(-r*t)
expr10
```

Out[41]: $R(t)e^{-rt}$

In [42]:
```
I10= sy.Integral(expr10,(t,0,3))
I10
```

Out[42]:

$$\int_0^3 R(t)e^{-rt}\,dt$$

In [43]:
```
integrate(I10,(t,0,3))
```

Out[43]:

$$3\int_0^3 R(t)e^{-rt}\,dt$$

In [44]:
```
expr10 = D*exp(-r*t)
expr10
```

Out[44]: De^{-rt}

In [45]:
```
I10= sy.Integral(expr10,(t,0,3))
I10
```

Out[45]:

$$\int_0^3 De^{-rt}\,dt$$

In [46]:
```
integrate(I10,(t,0,3))
```

Out[46]:

$$\begin{cases} \frac{3D}{r} - \frac{3De^{-3r}}{r} & \text{for } r > -\infty \wedge r < \infty \wedge r \neq 0 \\ 9D & \text{otherwise} \end{cases}$$

```
In [ ]:
```

Mathematical Economics

Alpha Chiang

Chapter 15

First-Order Linear Differential Equations with Constant/Coefficient and Constant Term

In [2]:
```
from sympy import Symbol, dsolve, Function, Derivative, Eq

y = Function("y")
x = Symbol('x')
t = Symbol('t')
w = Function('w')              #15.1
u = Function('u')

d1 = Derivative(y(t),t)

Eq(d1 + u(t) * y(t),w(t))
```

Out[2]:
$$u(t)y(t) + \frac{d}{dt}y(t) = w(t)$$

The Homogeneous Case

In [3]:
```
a = Symbol('a')

d2 = Derivative(y(t),t)
Eq(d2 + a * y(t),0)
```

Out[3]:
$$ay(t) + \frac{d}{dt}y(t) = 0$$

In [4]:
```
A = Symbol('A')
e = Symbol('e')
Eq(A * e**(-a*t),y(t))#general solution
```

Out[4]: $Ae^{-at} = y(t)$

Example 1

In [5]:
```
d3 = Derivative(y(t),t)
display(Eq(d3 + 2* y(t),6))
dsolve(d3 + 2* y(t) - 6, y(t))
```

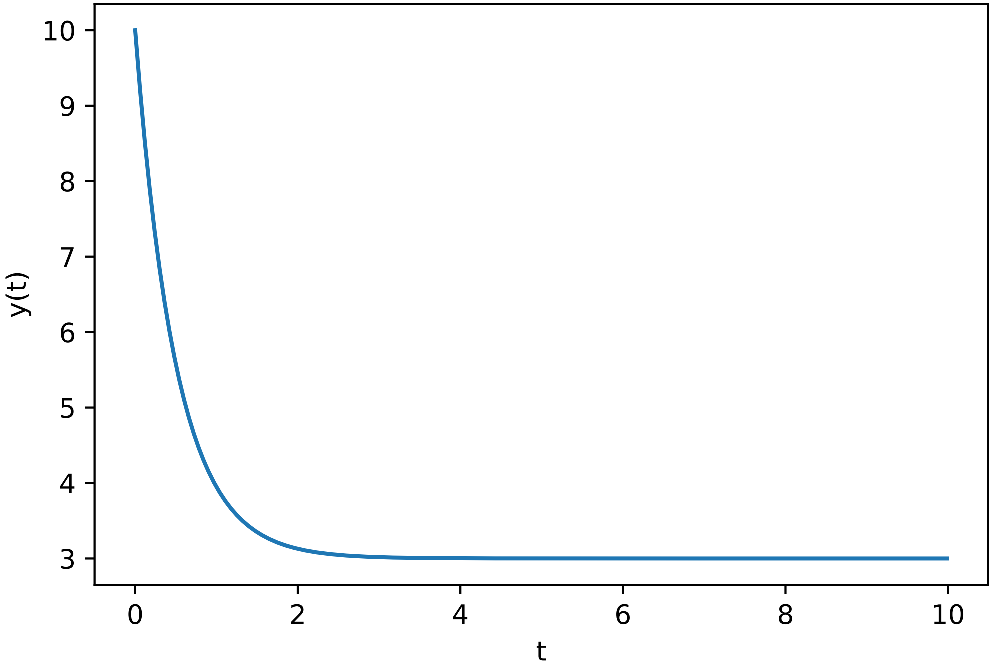
$$2y(t) + \frac{d}{dt}y(t) = 6$$

Out[5]: $y(t) = C_1e^{-2t} + 3$

In [6]:
```
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('svg', 'png')
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 400
```

In [8]:
```
%config InlineBackend.figure_format = 'svg'
from scipy.integrate import odeint
import numpy as np

def f(y, t):

    return -2 * y + 6

y0 = 10
a = 0
b = 10

t = np.arange(a, b, 0.01)
y = odeint(f, y0, t)

import pylab
pylab.plot(t, y)
pylab.xlabel('t'); pylab.ylabel('y(t)')
```

Out[8]:
```
Text(0, 0.5, 'y(t)')
```



Example 2

In [9]:
```
from sympy import Symbol, dsolve, Function, Derivative, Eq
y = Function("y")
x = Symbol('x')
t = Symbol('t')
w = Function('w')
u = Function('u')
d4 = Derivative(y(t),t)
display(Eq(d4 + 4*y(t),0))
dsolve(d3 + 4* y(t) , y(t))
```

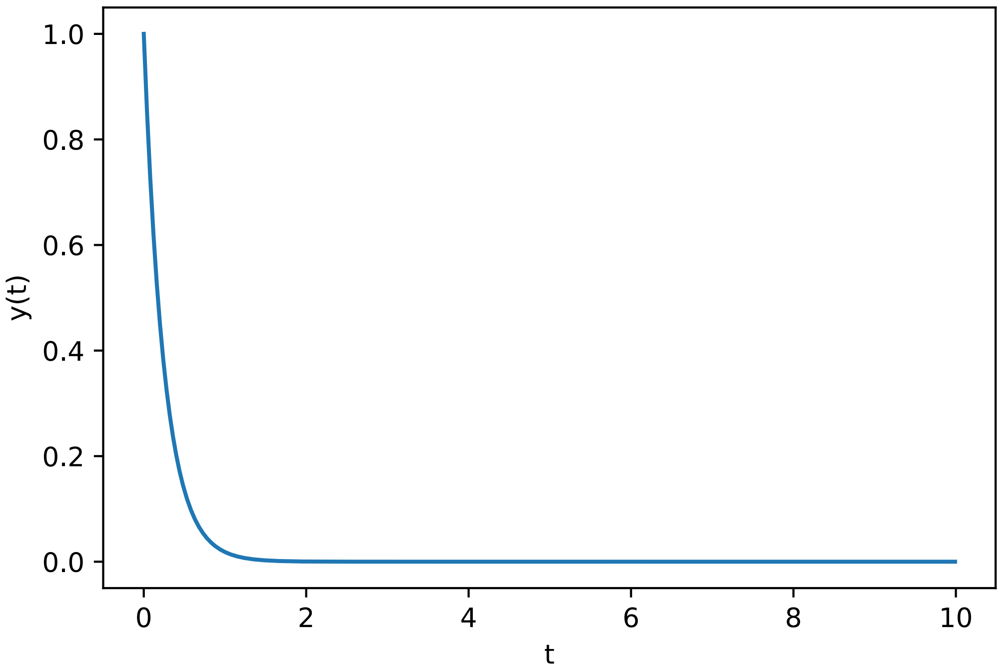
$$4y(t) + \frac{d}{dt}y(t) = 0$$

Out[9]: $y(t) = C_1 e^{-4t}$

In [10]:
```
def f(y, t):

    return -4 * y
```

```python
y0 = 1
a = 0
b = 10

t = np.arange(a, b, 0.01)

y = odeint(f, y0, t)

import pylab
pylab.plot(t, y)
pylab.xlabel('t'); pylab.ylabel('y(t)')
```

Out[10]: Text(0, 0.5, 'y(t)')



In [15]:
```python
from sympy import Symbol, dsolve, Function, Derivative, Eq
P = Function("P")
j = Symbol("j")
beta = Symbol("\\beta")
gamma = Symbol("\\gamma")
delta = Symbol("\\delta")
alpha = Symbol("\\alpha")
t = Symbol("t")
```

In [12]:
```
d5 = Derivative(P(t),t)
display(Eq(d5 + j * (beta + delta) * P(t),j*(alpha + gamma)))
```

$$j\left(\beta+\delta\right)P(t)+\frac{d}{dt}P(t)=j\left(\alpha+\gamma\right)$$

In [13]:
```
dsolve(d5 + j * (beta + delta) * P(t) - j*(alpha + gamma) , P(t))
```

Out[13]:

$$P(t)=\frac{\alpha+\gamma+e^{-C_1\beta-C_1\delta-\beta jt-\delta jt}}{\beta+\delta}$$

Variable Coefficient and Variable Term

Example 1

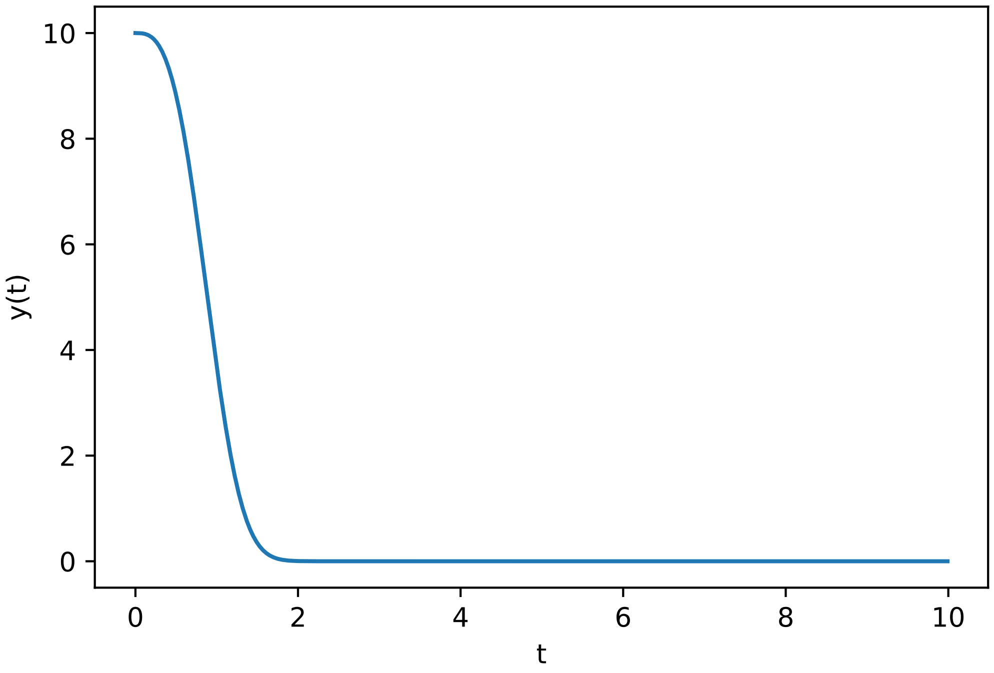
In [17]:
```
y = Function("y")
d6 = Derivative(y(t),t)
display(Eq(d6 + (3*t**2 * y(t)),0))
dsolve(d6 + 3*t**2*y(t), y(t))
```

$$3t^2y(t)+\frac{d}{dt}y(t)=0$$

Out[17]: $y(t)=C_1e^{-t^3}$

In [18]:
```
def f(y, t):

    return -3*y*t**2

y0 = 10
a = 0
b = 10

t = np.arange(a, b, 0.01)

y = odeint(f, y0, t)

import pylab
pylab.plot(t, y)
pylab.xlabel('t'); pylab.ylabel('y(t)')
```

Out[18]: Text(0, 0.5, 'y(t)')



Example 2

In [2]:
```
from sympy import Symbol, dsolve, Function, Derivative, Eq
y = Function("y")
t = Symbol("t")
d7 = Derivative(y(t),t)
display(Eq(d7 + (2*t * y(t)),t))
dsolve(d7 + t*2*y(t) - t, y(t))
```

$$2ty(t) + \frac{d}{dt}y(t) = t$$

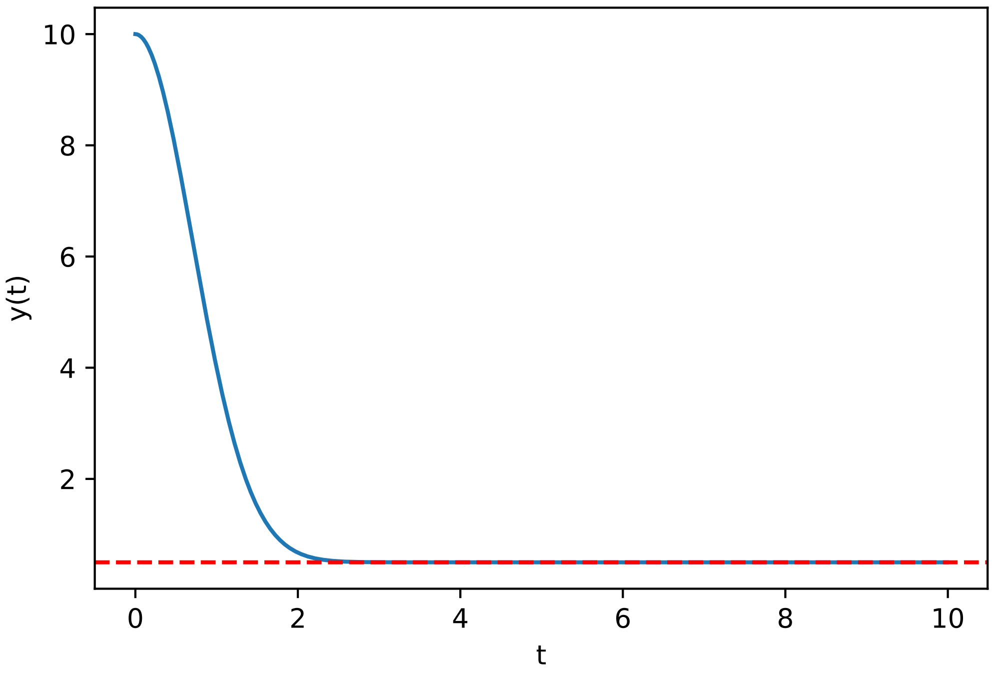
Out[2]:

$$y(t) = \frac{C_1 e^{-t^2}}{2} + \frac{1}{2}$$

In [3]:
```
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('svg', 'png')
import matplotlib as mpl
```

```
mpl.rcParams['figure.dpi'] = 400
%config InlineBackend.figure_format = 'svg'
from scipy.integrate import odeint
import numpy as np
def f(y, t):

    return -2*y*t + t

y0 = 10
a = 0
b = 10

t = np.arange(a, b, 0.01)
y = odeint(f, y0, t)
import pylab
pylab.plot(t, y)
pylab.axhline(y = 0.5, color = 'r', linestyle = "dashed") #equilibrium
pylab.xlabel('t'); pylab.ylabel('y(t)')
```

Out[3]: Text(0, 0.5, 'y(t)')



Solow Growth Model -- A Quantitative Illustration

In [4]:

```
from sympy import Symbol, dsolve, Function, Derivative, Eq
z = Function("z")
s = Symbol("s")


lambd = Symbol("\\lambda")
alpha = Symbol("\\alpha")
t = Symbol("t")
d8 = Derivative(z(t),t)
display(Eq(d8 + (1- alpha)* lambd * z(t) ,(1- alpha)*s))
dsolve(d8 + (1- alpha)* lambd * z(t) - (1- alpha)*s, z(t))
```

$$\lambda (1 - \alpha) z(t) + \frac{d}{dt} z(t) = s (1 - \alpha)$$

Out[4]:

$$z(t) = \frac{s + e^{\lambda(C_1 + \alpha t - t)}}{\lambda}$$

Furkan zengin

Mathematical Economics

Alpha Chiang

Chapter 16

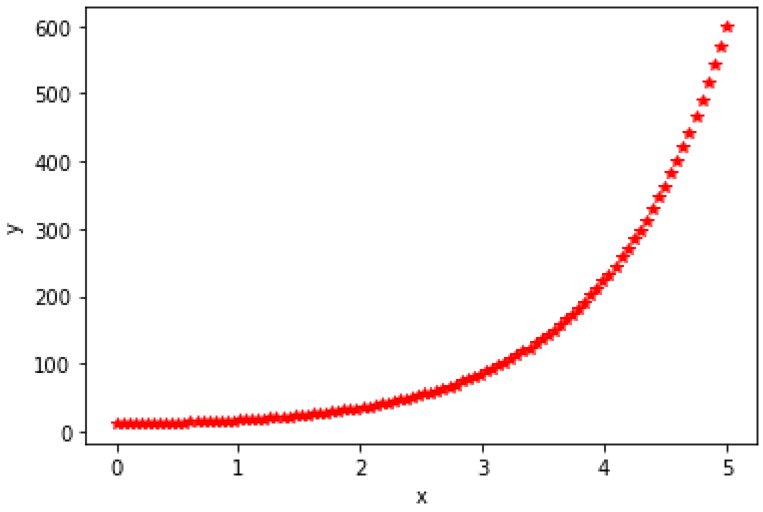
Higher-Order Differential Equations

In [2]:
```
from matplotlib import pyplot as plt
from scipy.integrate import odeint
import numpy as np
```

Example 1

In [3]:
```
def f(y,x):
    return (y[1], - y[1] + 2 * y[0] - 10)

y0 = [12,-2]
xs = np.linspace(0,5,100)
sol = odeint(f, y0, xs)
ys = sol[:,0]
ys2 = sol[:,1]
```

In [4]:
```
plt.plot(xs, ys,"r*")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



In [5]:
```
from sympy import Symbol, dsolve, Function, Derivative, Eq
y = Function("y")
t = Symbol('t')
dy2 = Derivative(y(t), t,2)
dy1 = Derivative(y(t),t)
eq1 = Eq(dy2 + dy1 - 2*y(t), -10)
eq1
```

Out[5]:

$$-2y(t) + \frac{d}{dt}y(t) + \frac{d^2}{dt^2}y(t) = -10$$

In [6]:
```
sol1 = dsolve(eq1, y(t))
sol1
```

Out[6]:

$$y(t) = C_1e^{-2t} + C_2e^{t} + 5$$

Example 5

In [7]:
```
y = Function("y")
t = Symbol('t')
dy2 = Derivative(y(t), t,2)
dy1 = 6 * Derivative(y(t),t)
eq1 = Eq(dy2 + dy1 +9 * y(t), 27)
eq1
```

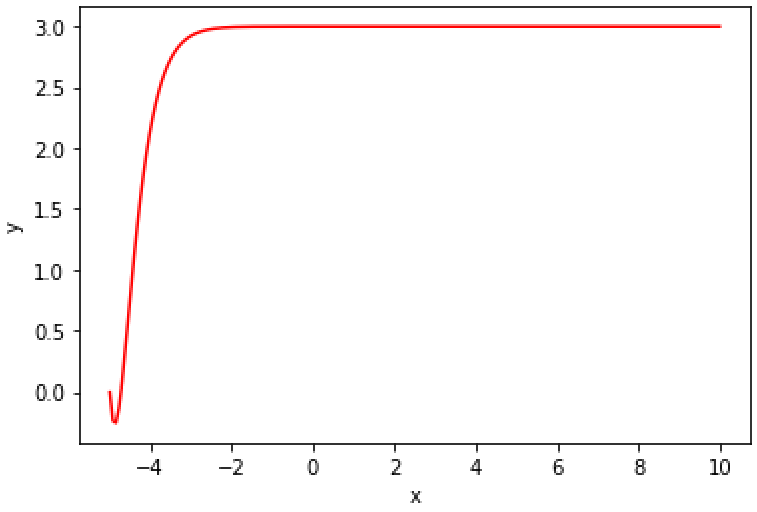
Out[7]: $$9y(t) + 6\frac{d}{dt}y(t) + \frac{d^2}{dt^2}y(t) = 27$$

In [8]:
```
sol2 = dsolve(eq1, y(t))
sol2
```

Out[8]: $$y(t) = (C_1 + C_2 t)\, e^{-3t} + 3$$

In [9]:
```
def f(y,x):
    return (y[1], - 6 * y[1] - 9 * y[0] + 27)

y0 = [0,-5]
xs = np.linspace(-5,10,200)
sol = odeint(f, y0, xs)
ys = sol[:,0]
ys2 = sol[:,1]
```

In [10]:
```
plt.plot(xs, ys,"r")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



Complex roots

In [11]:
```
y = Function("y")
```

```
t = Symbol('t')
dy2 = Derivative(y(t), t,2)
dy1 = 2 * Derivative(y(t),t)
eq1 = Eq(dy2 + dy1 + 17 * y(t), 34)
eq1
```

Out[11]:

$$17y(t) + 2\frac{d}{dt}y(t) + \frac{d^2}{dt^2}y(t) = 34$$

In [12]:

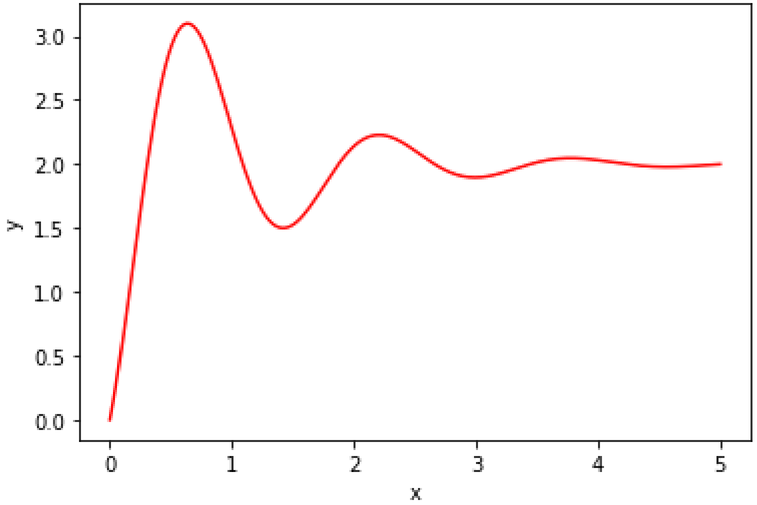
```
sol2 = dsolve(eq1, y(t))
sol2
```

Out[12]:

$$y(t) = (C_1 \sin(4t) + C_2 \cos(4t))\, e^{-t} + 2$$

In [14]:

```
def f(y,x):
    return (y[1], - 2 * y[1] - 17 * y[0] + 34)

y0 = [0,5]
xs = np.linspace(0,5,200)
sol = odeint(f, y0, xs)
ys = sol[:,0]
plt.plot(xs, ys,"r")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



In [15]:
```
y = Function("y")
t = Symbol('t')
dy2 = Derivative(y(t), t,2)
dy1 = -4 * Derivative(y(t),t)
eq1 = Eq(dy2 + dy1 + 8 * y(t), 0)
eq1
```

Out[15]:

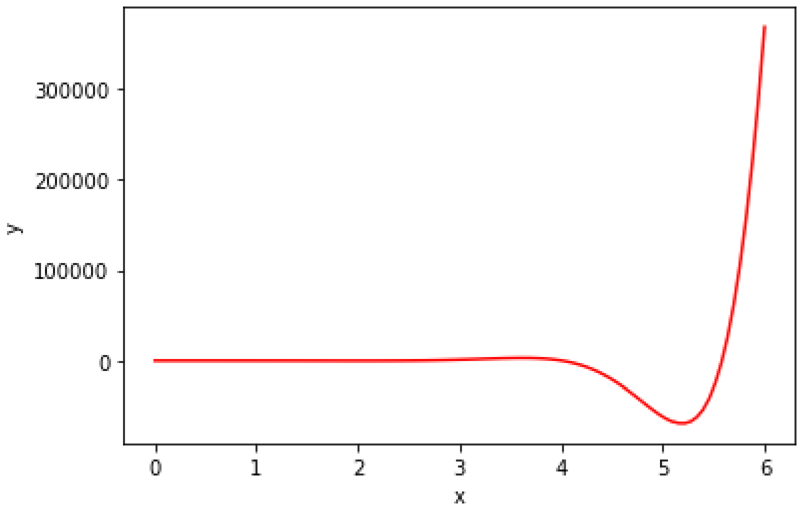
$$8y(t) - 4\frac{d}{dt}y(t) + \frac{d^2}{dt^2}y(t) = 0$$

In [16]:
```
sol2 = dsolve(eq1, y(t))
sol2
```

Out[16]:

$$y(t) = (C_1 \sin(2t) + C_2 \cos(2t))\, e^{2t}$$

In [17]:
```
def f(y,x):
    return (y[1], +4 * y[1] -8 * y[0] + 0)

y0 = [3,7]
xs = np.linspace(0,6,100)
sol = odeint(f, y0, xs)
ys = sol[:,0]

plt.plot(xs, ys,"r")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



EXERCISE 16.3

In [18]:
```
y = Function("y")
t = Symbol('t')
dy2 = 2 * Derivative(y(t), t,2)
dy1 = -12 * Derivative(y(t),t)
eq1 = Eq(dy2 + dy1 + 20 * y(t), 40)
eq1
```

Out[18]:

$$20y(t) - 12\frac{d}{dt}y(t) + 2\frac{d^2}{dt^2}y(t) = 40$$

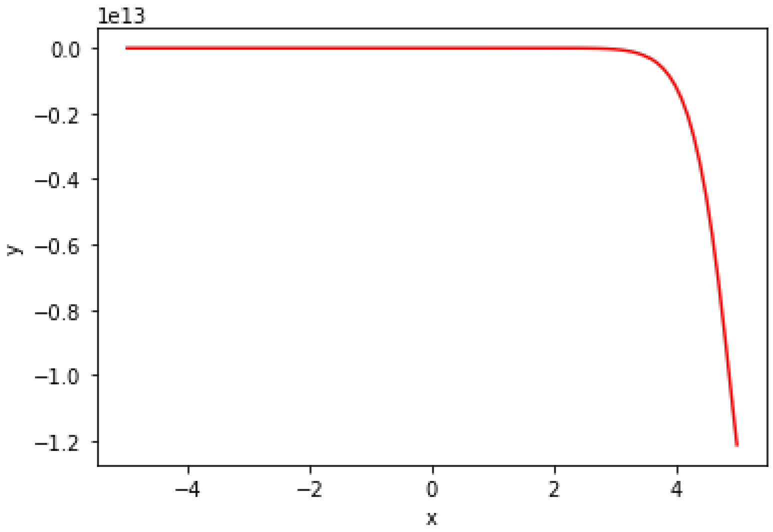
In [19]:
```
sol2 = dsolve(eq1, y(t))
sol2
```

Out[19]:

$$y(t) = (C_1 \sin(t) + C_2 \cos(t))\, e^{3t} + 2$$

In [20]:
```
def f(y,x):
    return (y[1], + 6 * y[1] -10 * y[0] + 20)

y0 = [4,5]
xs = np.linspace(-5,5,100)
sol = odeint(f, y0, xs)
ys = sol[:,0]
```

```
plt.plot(xs, ys,"r")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



A Market Model with Price Expectations

Example 1

In [21]:

```
P = Function("P")
Q = Symbol('Q')
t = Symbol("t")
dy2 = Derivative(P(t), t,2)
dy1 = -4 * Derivative(P(t), t)
eq1 = Eq(dy2 + dy1 - 12 * P(t), -48)
eq1
```

Out[21]:

$$-12P(t) - 4\frac{d}{dt}P(t) + \frac{d^2}{dt^2}P(t) = -48$$

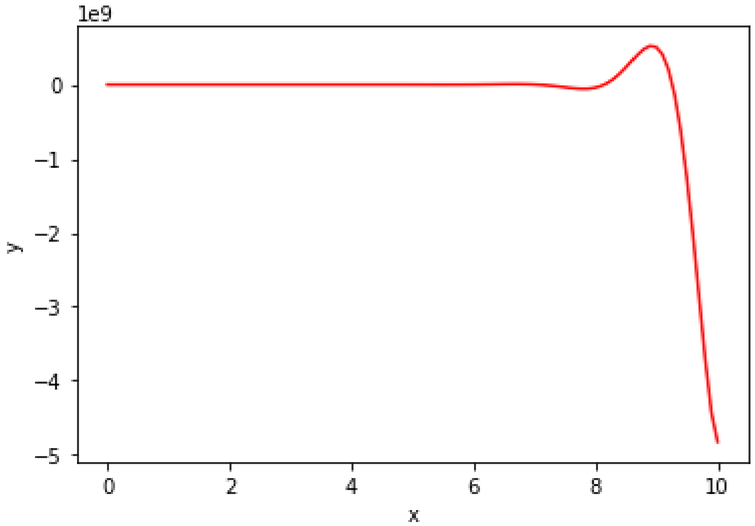
In [22]:

```
sol2 = dsolve(eq1, P(t))
sol2
```

Out[22]:

$$P(t) = C_1e^{-2t} + C_2e^{6t} + 4$$

In [23]:
```
def f(y,x):
    return (y[1], + 4*y[1] -12*y[0] - 48)

y0 = [6,4]
xs = np.linspace(0,10,100)
sol = odeint(f, y0, xs)
ys = sol[:,0]

plt.plot(xs, ys,"r")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



Example 2

In [24]:
```
P = Function("P")
Q = Symbol('Q')
t = Symbol("t")
dy2 = Derivative(P(t), t,2)
dy1 = 2 * Derivative(P(t), t)
eq1 = Eq(dy2 + dy1 +5* P(t), 45)
eq1
```

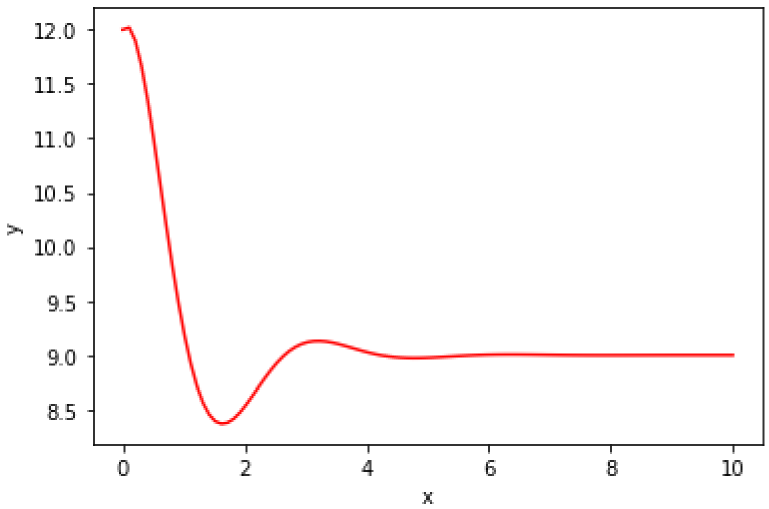
Out[24]:

$$5P(t) + 2\frac{d}{dt}P(t) + \frac{d^2}{dt^2}P(t) = 45$$

In [25]:
```python
sol2 = dsolve(eq1, P(t))
sol2
```

Out[25]: $P(t) = (C_1 \sin(2t) + C_2 \cos(2t))\, e^{-t} + 9$

In [26]:
```python
def f(y,x):
    return (y[1], -2*y[1] -5*y[0] + 45)

y0 = [12,1]
xs = np.linspace(0,10,100)
sol = odeint(f, y0, xs)
ys = sol[:,0]

plt.plot(xs, ys,"r")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



EXERCISE 16.4 Q/3

In [27]:
```python
from sympy import symbols, Eq, solve
P = Function("P")
Q = Symbol('Q')
Q_d = Symbol("Q_d")
Q_s = Symbol("Q_s")
t = Symbol("t")
```

```python
dy2 = 3 * Derivative(P(t), t,2)
dy1 = Derivative(P(t), t)
eq1 = Eq(dy2 + dy1 - P(t) + 9,Q_d)
display(eq1)


dy2_ = 5 * Derivative(P(t), t,2)
dy1_ = -Derivative(P(t), t)
eq2 = Eq(dy2_ + dy1_ +4* P(t) -1 ,Q_s)
display(eq2)
```

$$-P(t) + \frac{d}{dt}P(t) + 3\frac{d^2}{dt^2}P(t) + 9 = Q_d$$

$$4P(t) - \frac{d}{dt}P(t) + 5\frac{d^2}{dt^2}P(t) - 1 = Q_s$$

In [28]:

```python
dy3 = 2 * Derivative(P(t), t,2)
dy2 = -2* Derivative(P(t), t)
eq3 = Eq(dy3 + dy2 +5* P(t),10)
display(eq3)
```

$$5P(t) - 2\frac{d}{dt}P(t) + 2\frac{d^2}{dt^2}P(t) = 10$$

In [29]:

```python
eq1.lhs - eq2.lhs
```

Out[29]:

$$-5P(t) + 2\frac{d}{dt}P(t) - 2\frac{d^2}{dt^2}P(t) + 10$$

In [30]:

```python
sol3 = dsolve(eq3, P(t))
sol3
```

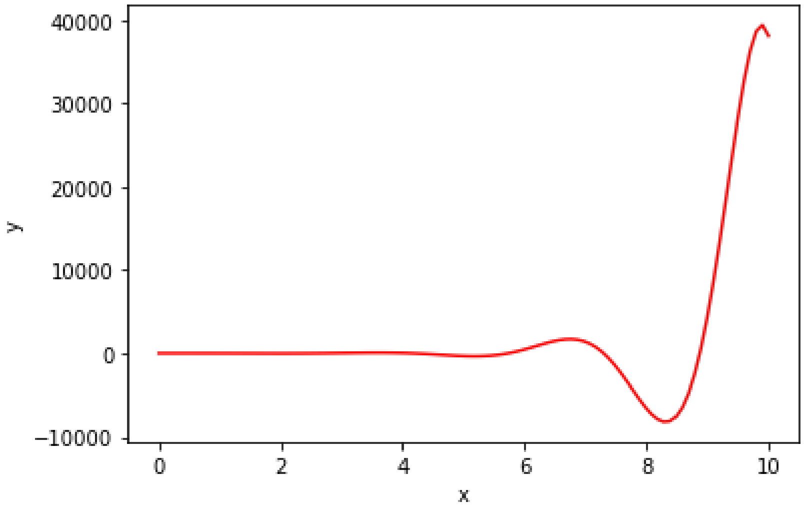
Out[30]:

$$P(t) = \left(C_1 \sin\left(\frac{3t}{2}\right) + C_2 \cos\left(\frac{3t}{2}\right)\right)e^{\frac{t}{2}} + 2$$

In [31]:

```python
def f(y,x):
    return (y[1], +2*y[1] -5*y[0] + 10)

y0 = [4,4]
xs = np.linspace(0,10,100)
```

```
sol = odeint(f, y0, xs)
ys = sol[:,0]

plt.plot(xs, ys,"r")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



16.5 The Interaction of Inflation and Unemployment

In [32]:

```
P = Function("P")
pi = Function("\u03C0")
beta = Symbol("\\beta")
k = Symbol("k")
j = Symbol("j")
g = Symbol("g")
t = Symbol("t")
m = Symbol("m")
dy2 = Derivative(pi(t), t,2)
dy1 = (beta * k + j*(1-g)) * Derivative(pi(t), t)
eq4 = Eq(dy2 + dy1 - (j*beta*k)*pi(t) ,(j*beta*k*m))
eq4
```

Out[32]:

$$-\beta jk\pi(t) + (\beta k + j(1-g))\frac{d}{dt}\pi(t) + \frac{d^2}{dt^2}\pi(t) = \beta jkm$$

In [33]:
```
sol4 = dsolve(eq4, pi(t))
sol4
```

Out[33]:

$$\pi(t) = C_1 e^{\frac{t\left(-\beta k + gj - j - \sqrt{\beta^2 k^2 - 2\beta gjk + 6\beta jk + g^2 j^2 - 2gj^2 + j^2}\right)}{2}} + C_2 e^{\frac{t\left(-\beta k + gj - j + \sqrt{\beta^2 k^2 - 2\beta gjk + 6\beta jk + g^2 j^2 - 2gj^2 + j^2}\right)}{2}} - m$$

16.6 Differential Equations with a Variable Term

In [34]:
```
y = Function("y")
t = Symbol('t')
dy2 = Derivative(y(t), t,2)
dy1 = 5 * Derivative(y(t),t)
eq1 = Eq(dy2 + dy1 + 3 * y(t), 6*t**2 - t - 1)
eq1
```

Out[34]:

$$3y(t) + 5\frac{d}{dt}y(t) + \frac{d^2}{dt^2}y(t) = 6t^2 - t - 1$$

In [35]:
```
sol1 = dsolve(eq1, y(t))
sol1
```

Out[35]:

$$y(t) = C_1 e^{\frac{t\left(-5-\sqrt{13}\right)}{2}} + C_2 e^{\frac{t\left(-5+\sqrt{13}\right)}{2}} + 2t^2 - 7t + 10$$

Furkan Zengin

Mathematical Economics

Alpha Chiang

Chapter 17

Discrete Time: First-Order Difference Equations

17.2 Solving a First-Order Difference Equation

Example 3

Example 4

In [2]:
```
from sympy import Symbol, dsolve, Function, Derivative, Eq

from sympy import Function, rsolve
from sympy.abc import t,m,n
y = Function("y");
y0 = Symbol("y_0")
f = m*y(t+1) - n*y(t)  ;
sol = rsolve(f, y(t), {y(0):y0});
sol
```

Out[2]:

$$y_0\left(\frac{n}{m}\right)^t$$

Example 4

In [3]:
```
yt1 = Symbol("y_t+1")
yt = Symbol('y_t')
eq1 = Eq(yt1 - 5*yt,1)
eq1
```

Out[3]:

$$-5y_t + y_{t+1} = 1$$

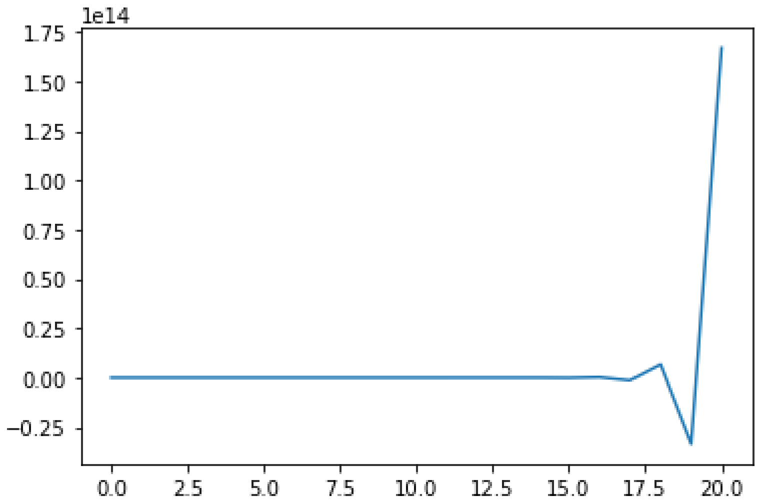
In [5]:
```python
from sympy import Function, rsolve
from sympy.abc import t
y = Function("y");
f = y(t+1) - 5*y(t) - 1 ;
sol = rsolve(f, y(t), {y(0):7/4});
print("y_t = ${}".format(sol))
display(sol)
```

```
y_t = $2.0*5**t - 1/4
```

$$2.0 \cdot 5^{t} - \frac{1}{4}$$

In [6]:
```python
import numpy as np
import matplotlib.pyplot as plt
N = 20
index_set = range(N+1)
x = np.zeros(len(index_set))
x[0] = 7/4
for n in index_set[1:]:
    x[n] = -5*x[n-1]
plt.plot(index_set, x)
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x1fc951660a0>]
```



In [7]:
```python
t = Symbol("t")
yt1 = Symbol("y_t+1")
yt = Symbol('y_t')
```

```
eq1 = Eq(yt, 2*(-4/5)**t + 9)
eq1
```

Out[7]: $y_t = 2(-0.8)^t + 9$

In [9]:
```
from sympy import Function, rsolve
from sympy.abc import t
y = Function("y")
f = y(t) - 2*(-4/5)**t - 9
sol = rsolve(f, y(t), {y(0):1})
```

EXERCISE 17.3 --Q3/c--

In [10]:
```
t = Symbol("t")
yt1 = Symbol("y_t+1")
yt = Symbol('y_t')
eq1 = Eq(yt1 + 1/4*yt, 5)
eq1
```

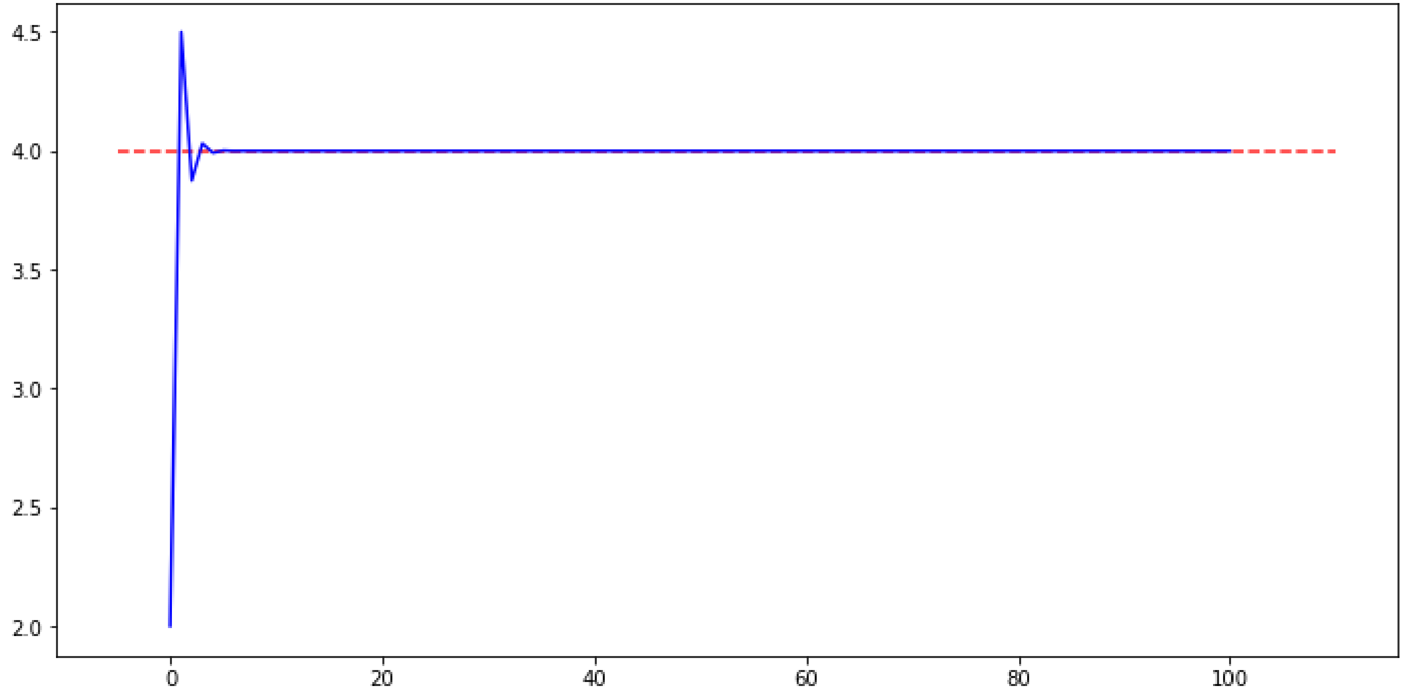
Out[10]: $0.25y_t + y_{t+1} = 5$

In [11]:
```
from sympy import Function, rsolve
from sympy.abc import t
y = Function("y")
f = y(t+1) + 1/4*y(t) - 5
sol = rsolve(f, y(t), {y(0):2})
sol
```

Out[11]: $4.0 - 2.0(-0.25)^t$

In [12]:
```
N = 100
index_set = range(N+1)
x = np.zeros(len(index_set))
x[0] = 2
for t in index_set[1:]:
    x[t] = -1/4 * x[t-1] + 5

plt.figure(figsize = (12, 6))
plt.hlines(4,-5, 110, linestyle='--', alpha=0.9,color='red')
# Eq value
plt.plot(index_set, x,'b')
```

Out[12]: [<matplotlib.lines.Line2D at 0x1fc954c82e0>]



17.4 The Cobweb Model

In [13]:
```
t = Symbol("t")
Pt1 = Symbol("P_t+1")
Pt = Symbol('P_t')
beta = Symbol('\\beta')
alpha = Symbol('\\alpha')
gamma = Symbol('\\gamma')
delta = Symbol('\\delta')
eq1 = Eq(Pt1 + (delta/beta)*Pt, (alpha+gamma)/beta)
eq1
```

Out[13]:
$$\frac{P_t\delta}{\beta} + P_{t+1} = \frac{\alpha + \gamma}{\beta}$$

In [14]:
```
from sympy import Function, rsolve
from sympy.abc import t
y = Function("y")
P0 = Symbol("P_0")
```

```
f = y(t+1) + (delta/beta)*y(t) - (alpha+gamma)/beta
sol = rsolve(f, y(t), {y(0):P0})
sol
# For the visulation of Cobweb model
# https://dongminkim0220.github.io/posts/cobweb/
```

Out[14]:

$$\frac{\left(-\frac{\delta}{\beta}\right)^{t}\left(P_{0}\beta + P_{0}\delta - \alpha - \gamma\right)}{\beta + \delta} + \frac{\alpha + \gamma}{\beta + \delta}$$

In [16]:

```
from sympy import symbols, Eq, solve
t = Symbol("t")
Pt1 = Symbol("P_t+1")
Pt = Symbol('P_t')
Pt1_ = Symbol("P_t-1")
Qd = Symbol("Q_d")
Qs = Symbol("Q_s")

eq1 = Eq(18 - 3*Pt,Qd)
display(eq1)



eq2 = Eq(-3 + 4*Pt1_,Qs)
display(eq2)
eq1.lhs - eq2.lhs
```

$$18 - 3P_{t} = Q_{d}$$

$$4P_{t-1} - 3 = Q_{s}$$

Out[16]: $-3P_{t} - 4P_{t-1} + 21$

In [17]:

```
eq3 = Eq(-3*Pt -4*Pt1_, -21)
eq3
```

Out[17]: $-3P_{t} - 4P_{t-1} = -21$

In [18]:

```
from sympy import Function, rsolve
from sympy.abc import t
y = Function("y")
P0 = Symbol("P_0")
```

```python
f = -3*y(t) + -4*y(t-1) + 21
sol = rsolve(f, y(t), {y(0):1})
sol
```

Out[18]:

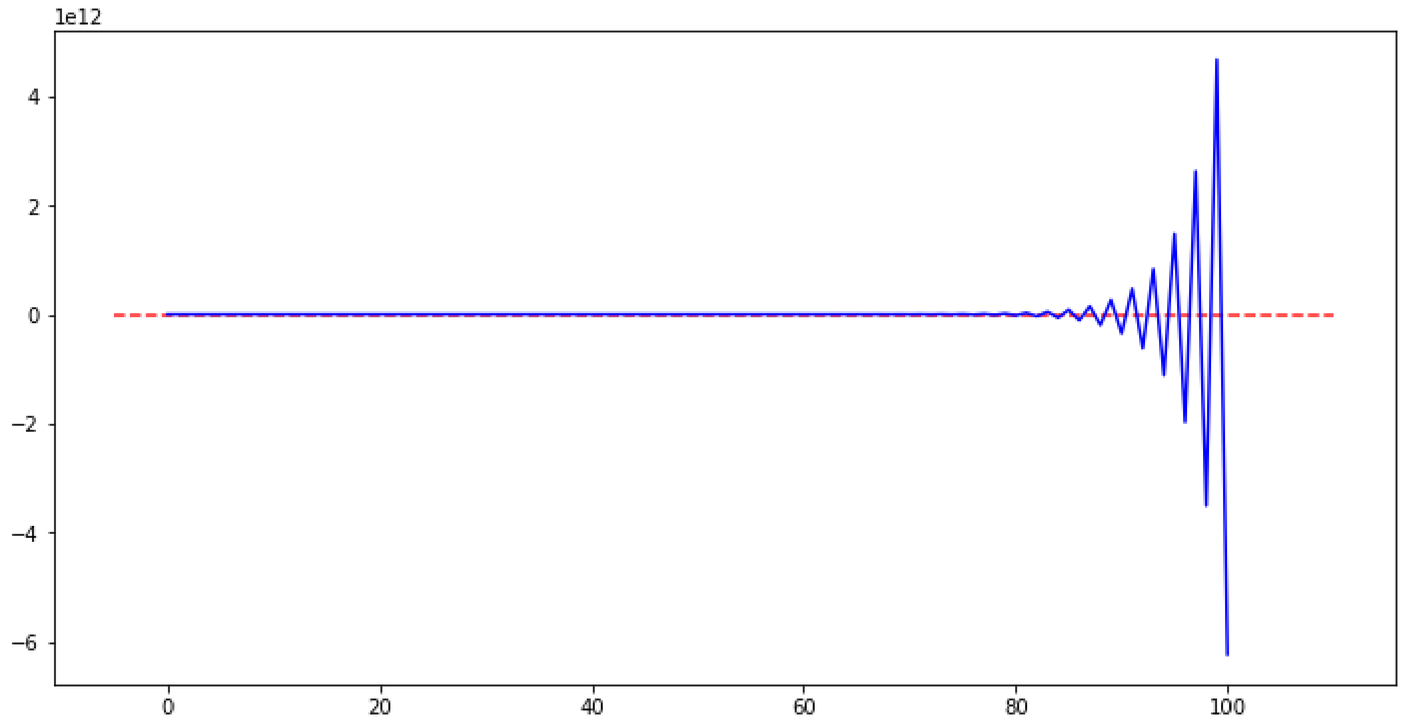
$$3 - 2\left(-\frac{4}{3}\right)^{t}$$

In [19]:

```python
N = 100
index_set = range(N+1)
x = np.zeros(len(index_set))
x[0] = 1
for t in index_set[1:]:
    x[t] = - 4/3 * x[t-1] + 21/3

plt.figure(figsize = (12, 6))
plt.hlines(4,-5, 110, linestyle='--', alpha=0.9,color='red')
# Eq value
plt.plot(index_set, x,'b')
```

Out[19]:

```
[<matplotlib.lines.Line2D at 0x1fc9554d8e0>]
```



17.5 A Market Model with Inventory

In [20]:

```
t = Symbol("t")
Pt1 = Symbol("P_t+1")
Pt = Symbol('P_t')
beta = Symbol('\\beta')
alpha = Symbol('\\alpha')
gamma = Symbol('\\gamma')
delta = Symbol('\\delta')
sigma = Symbol('\\sigma')
eq1 = Eq(Pt1 - (1 - sigma*(beta + delta))*Pt,
(alpha+gamma)*sigma)
eq1
```

Out[20]: $-P_t\left(-\sigma\left(\beta+\delta\right)+1\right)+P_{t+1}=\sigma\left(\alpha+\gamma\right)$

In [21]:

```
from sympy import Function, rsolve
from sympy.abc import t
y = Function("y")
```

```
P0 = Symbol("P_0")
f = y(t+1)-(1-sigma*(beta + delta))*y(t)-(alpha+gamma)*sigma
sol = rsolve(f, y(t), {y(0):P0})
sol
```

Out[21]:

$$\frac{(-\beta\sigma - \delta\sigma + 1)^{t}\left(P_0\beta + P_0\delta - \alpha - \gamma\right)}{\beta + \delta} + \frac{\alpha\sigma + \gamma\sigma}{\sigma\left(\beta + \delta\right)}$$

Mathematical Economics

Alpha Chiang

Chapter 18

Higher-Order Difference Equations

In [1]:
```
from IPython.display import display, Math, Latex
Math(r'\Delta^2 y_{t+1} = \Delta(\Delta y_t) = \Delta(y_{t+1}- y_t)')
```

Out[1]: $\Delta^2 y_{t+1} = \Delta(\Delta y_t) = \Delta(y_{t+1} - y_t)$

18.1 Second-Order Linear Difference Equations with Constant Coefficients and Constant Term

Particular Solution

In [2]:
```
from sympy import Symbol, dsolve, Function, Derivative, Eq

from sympy import Function, rsolve
from sympy.abc import t,c
y = Function("y");
y0 = Symbol("y_0")
a1 = Symbol("a_1")
a2 = Symbol("a_2")

f = y(t+2) + a1*y(t+1) + a2*y(t) - c
sol = rsolve(f, y(t), {y(0):y0});
sol
```

Out[2]:

$$C_0\left(-\frac{a_1}{2} - \frac{\sqrt{a_1^2 - 4a_2}}{2}\right)^t + \frac{c}{a_1 + a_2 + 1} + \frac{\left(-\frac{a_1}{2} + \frac{\sqrt{a_1^2-4a_2}}{2}\right)^t\left(-C_0\left(a_1 + a_2 + 1\right) + a_1 y_0 + a_2 y_0 - c + y_0\right)}{a_1 + a_2 + 1}$$

Example 1

In [3]:
```
yt2 = Symbol("y_t+2")
yt1 = Symbol('y_t+1')
yt  = Symbol('y_t')
eq1 = Eq(yt2 - 3*yt1 +4*yt,6)
display(eq1)

from sympy.abc import t,c,k
y = Function("y");


f = y(t+2) - 3*y(t+1) + 4*y(t)-6;
sol = rsolve(f, y(t), {y(0):1, y(1):2});
sol
```

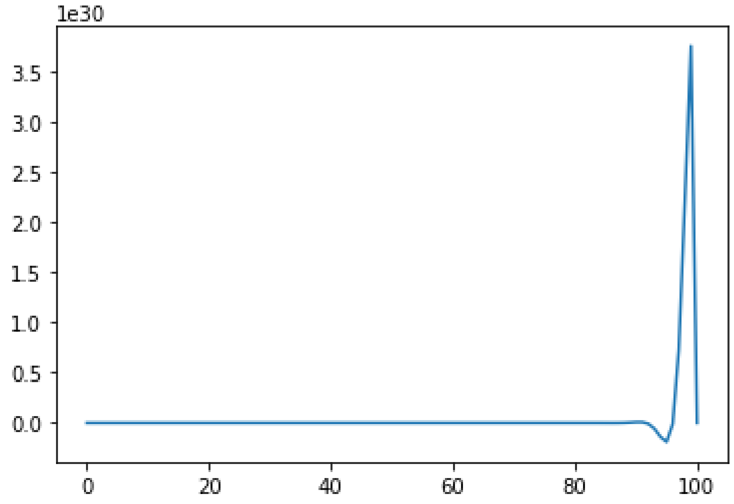
$$4y_t - 3y_{t+1} + y_{t+2} = 6$$

Out[3]:

$$\left(-1 + \frac{2\sqrt{7}i}{7}\right)\left(\frac{3}{2} - \frac{\sqrt{7}i}{2}\right)^t + \left(-1 - \frac{2\sqrt{7}i}{7}\right)\left(\frac{3}{2} + \frac{\sqrt{7}i}{2}\right)^t + 3$$

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt
N = 100
index_set = range(N+1)
x = np.zeros(len(index_set))
x[0] = 1
x[1] = 1
for t in index_set[1:N]:
    x[t] = 3*x[t-1] - 4*x[t-2] + 6
plt.plot(index_set, x)
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x214dc5ae070>]
```



Example 2

In [5]:
```
yt2 = Symbol("y_t+2")
yt1 = Symbol('y_t+1')
yt  = Symbol('y_t')
eq1 = Eq(yt2 + yt1 - 2*yt, 12)
display(eq1)



from sympy.abc import t,c,k
y = Function("y");


f = y(t+2) + y(t+1) - 2*y(t)- 12;
sol = rsolve(f, y(t), {y(0):1, y(1):2});
sol
```

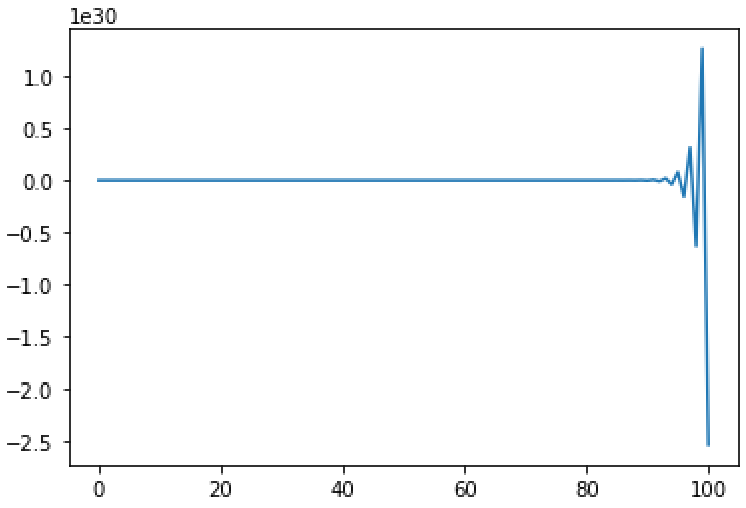
$$-2y_t + y_{t+1} + y_{t+2} = 12$$

Out[5]: $(-2)^t + 4t$

In [6]:
```
N = 100
index_set = range(N+1)
x = np.zeros(len(index_set))
x[0] = 1
```

```
x[1] = 1
for t in index_set[1:]:
    x[t] = -x[t-1] + 2*x[t-2] + 12
plt.plot(index_set, x)
```

Out[6]: [<matplotlib.lines.Line2D at 0x214dd661df0>]



Example 3

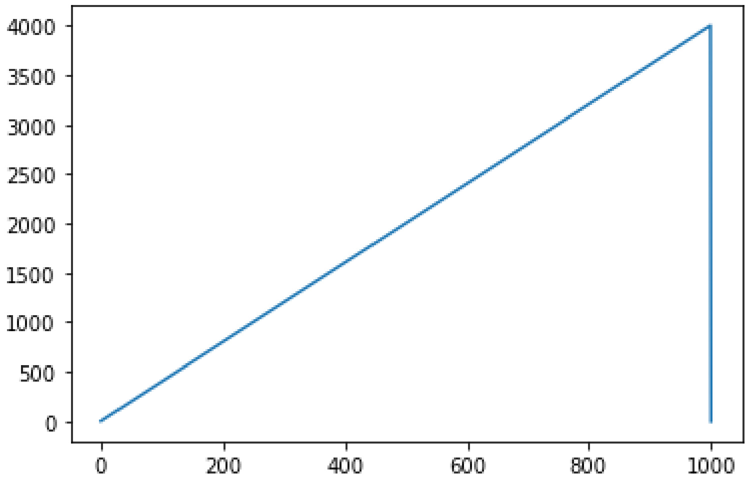
In [7]:
```
yt2 = Symbol("y_t+2")
yt1 = Symbol('y_t+1')
yt  = Symbol('y_t')
eq1 = Eq(yt2 + yt1 - 2*yt, 12)
display(eq1)

from sympy.abc import t,c,k
y = Function("y");


f = y(t+2) + y(t+1) - 2*y(t)- 12;
sol = rsolve(f, y(t), {y(0):4, y(1):5});
sol
```

$$-2y_t + y_{t+1} + y_{t+2} = 12$$

Out[7]: $(-2)^t + 4t + 3$

In [8]:
```
import numpy as np
import matplotlib.pyplot as plt
N = 1000
index_set = range(N+1)
x = np.zeros(len(index_set))
x[0] = 4
x[1] = 5
for t in index_set[1:N]:
    x[t] = -x[t-1] + 2*x[t-2] + 12

plt.plot(index_set, x)
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x214dd6cc6a0>]
```



Example 4

In [9]:
```
yt2 = Symbol("y_t+2")
yt1 = Symbol('y_t+1')
yt  = Symbol('y_t')
eq1 = Eq(yt2 + 6*yt1 + 9*yt, 4)
display(eq1)

from sympy.abc import t,c,k
y = Function("y");


f = y(t+2) + 6*y(t+1) + 9*y(t) - 4;
```

```
sol = rsolve(f, y(t), {y(0):1, y(1):1});
sol
```

$$9y_t + 6y_{t+1} + y_{t+2} = 4$$

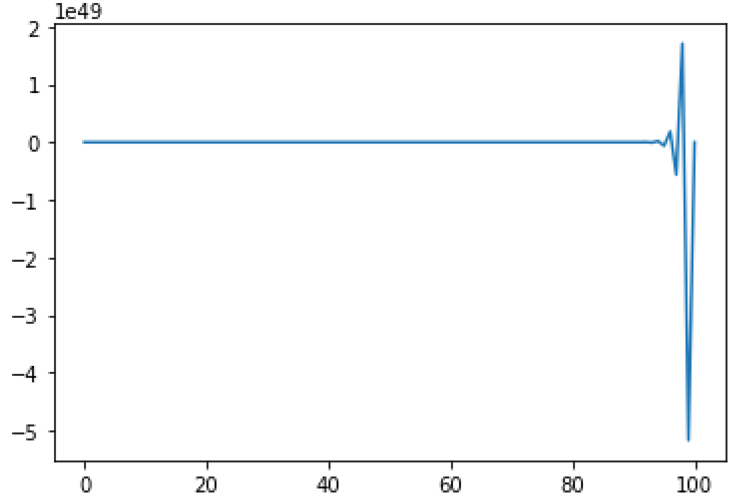
Out[9]:

$$(-3)^t \left(\frac{3}{4} - t\right) + \frac{1}{4}$$

In [10]:

```
N = 100
index_set = range(N+1)
x = np.zeros(len(index_set))
x[0] = 4
x[1] = 5
for t in index_set[1:N]:
    x[t] = -6*x[t-1] - 9*x[t-2] + 4

plt.plot(index_set, x)
```

Out[10]: [<matplotlib.lines.Line2D at 0x214dd7540a0>]



Example 5

In [11]:

```
yt2 = Symbol("y_t+2")
yt1 = Symbol('y_t+1')
yt  = Symbol('y_t')
eq1 = Eq(yt2 +  1/4*yt, 5)
display(eq1)
```

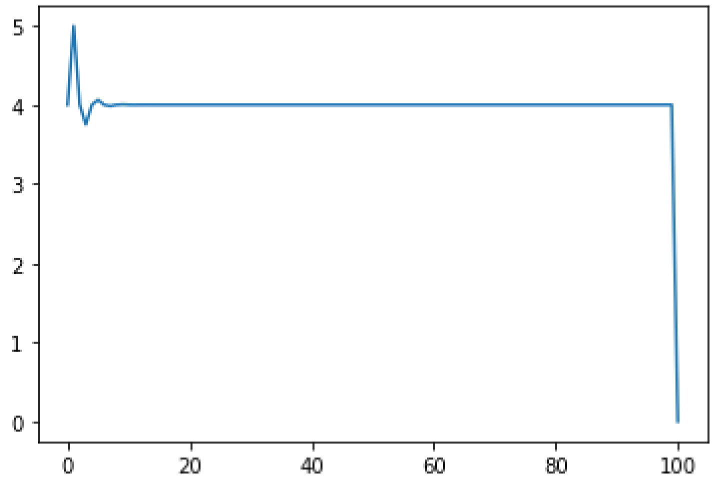
```
from sympy.abc import t,c,k
y = Function("y")
f = y(t+2) + 0*y(t+1) + 1/4*y(t) - 5
sol = rsolve(f, y(t))
sol
```

$$0.25y_t + y_{t+2} = 5$$

Out[11]: $$C_0(0.5i)^t + C_1(-0.5i)^t + 4.0$$

In [12]:
```
N = 100
index_set = range(N+1)
x = np.zeros(len(index_set))
x[0] = 4
x[1] = 5
for t in index_set[1:N]:
    x[t] =  -1/4*x[t-2] + 5

plt.plot(index_set, x)
```

Out[12]: [<matplotlib.lines.Line2D at 0x214dd7c8970>]



18.2 Samuelson Multiplier-Acceleration Interaction Model

In [13]:
```
Yt2 = Symbol("Y_t+2")
Yt1 = Symbol('Y_t+1')
```

```
Yt  = Symbol('Y_t')
alpha= Symbol("\\alpha")
gamma= Symbol("\\gamma")
G0 = Symbol('G_0')


eq1 = Eq(Yt2-gamma*(1+alpha)*Yt1 +  alpha*gamma*Yt, G0)
display(eq1)

from sympy.abc import t,c,k
y = Function("y")
f = y(t+2)-gamma*(1+alpha)*y(t+1) + alpha*gamma*y(t) - G0
sol = rsolve(f, y(t))
sol
```

$$Y_t\alpha\gamma - Y_{t+1}\gamma(\alpha+1) + Y_{t+2} = G_0$$

Out[13]:

$$C_0\left(\frac{\gamma(\alpha+1)}{2} - \frac{\sqrt{\gamma(\alpha^2\gamma + 2\alpha\gamma - 4\alpha + \gamma)}}{2}\right)^t + C_1\left(\frac{\gamma(\alpha+1)}{2} + \frac{\sqrt{\gamma(\alpha^2\gamma + 2\alpha\gamma - 4\alpha + \gamma)}}{2}\right)^t - \frac{G_0}{\gamma - 1}$$

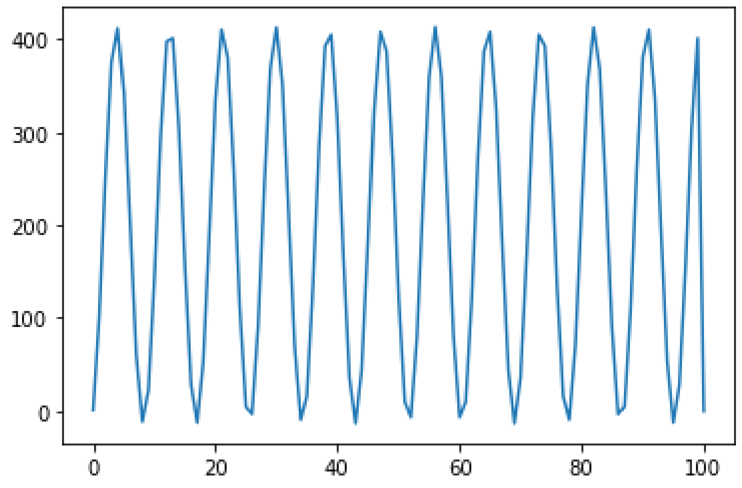
Example 2

In [15]:

```
N = 100
index_set = range(N+1)
x = np.zeros(len(index_set))
x[0] = 1
x[1] = 1
for t in index_set[1:N]:
    x[t] =  3/2*x[t-1] - x[t-2] + 100

plt.plot(index_set, x)
```

Out[15]:

```
[<matplotlib.lines.Line2D at 0x214dd8be190>]
```



8.3 Inflation and Unemployment in Discrete Time

In [16]:
```
pt2 = Symbol("p_t+2")
pt1 = Symbol('p_t+1')
pt  = Symbol('p_t')
beta= Symbol("\\beta")
k= Symbol("k")
j = Symbol('j')
m = Symbol('m')
g = Symbol("g")


eq1 = Eq( pt2 - ((1+g*j+(1-j)*(1+beta*k)) / (1+beta*k))*pt1 +
         ((1-j*(1-g)))*pt/((1+beta*k)), (j*beta*k*m)/(1+beta*k))
display(eq1)

from sympy import simplify
from sympy.abc import t,c,k
y = Function("y")
f = y(t+2)-((1+g*j+(1-j)*(1+beta*k)))*y(t+1)/(1+beta*k) +(1/(1+beta*k))*((1-j*(1-g)))*y(t) - (j*beta*k*m)/(1+beta*k)
# we just write the 18.24
sol = rsolve(f, y(t))
simplify(sol)
```

$$\frac{p_t\left(-j\left(1-g\right)+1\right)}{\beta k+1}-\frac{p_{t+1}\left(gj+\left(1-j\right)\left(\beta k+1\right)+1\right)}{\beta k+1}+p_{t+2}=\frac{\beta jkm}{\beta k+1}$$

Out[16]:

$$C_0\left(\frac{-\beta jk+\beta k+gj-j-\sqrt{\beta^2j^2k^2-2\beta^2jk^2+\beta^2k^2-2\beta gj^2k-2\beta gjk+2\beta j^2k-2\beta jk+g^2j^2-2gj^2+j^2}+2}{2(\beta k+1)}\right)^t + C_1\left(\frac{-\beta jk+\beta k+gj-j+\sqrt{\beta^2j^2k^2-2\beta^2jk^2+\beta^2k^2-2\beta gj^2k-2\beta gjk+2\beta j^2k-2\beta jk+g^2j^2-2gj^2+j^2}+2}{2(\beta k+1)}\right)^t + m$$

18.4 Generalizations to Variable-Term and Higher-Order Equations

In [17]:

```
yt2 = Symbol("y_t+2")
yt1 = Symbol('y_t+1')
yt  = Symbol('y_t')
t  = Symbol('t')
eq1 = Eq(yt2 + yt1 -3*yt, 7**t)
display(eq1)

from sympy.abc import t,c,k
y = Function("y")
f = y(t+2) + y(t+1) -3*y(t) - 7**t
sol = rsolve(f, y(t))
sol
```

$$-3y_t + y_{t+1} + y_{t+2} = 7^t$$

Out[17]:

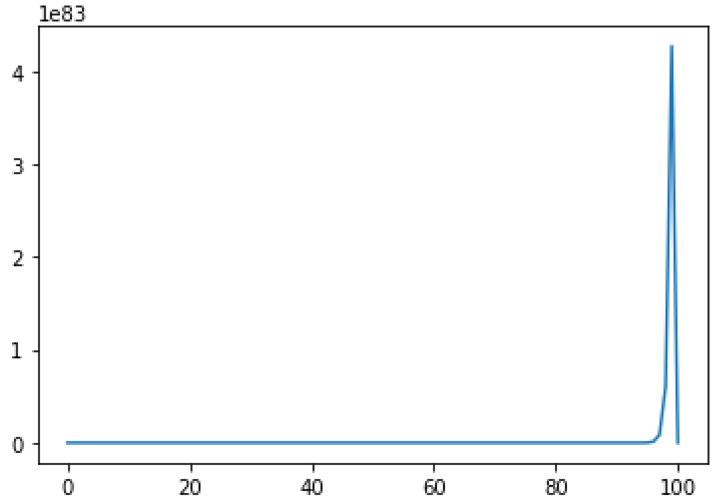
$$\frac{7^t}{53} + C_0\left(-\frac{1}{2}+\frac{\sqrt{13}}{2}\right)^t + C_1\left(-\frac{\sqrt{13}}{2}-\frac{1}{2}\right)^t$$

In [18]:

```
N = 100
index_set = range(N+1)
x = np.zeros(len(index_set))
x[0] = 1
x[1] = 1
for t in index_set[1:N]:
    x[t] =  -x[t-1] + 3*x[t-2] + 7**t

plt.plot(index_set, x)
```

Out[18]:

```
[<matplotlib.lines.Line2D at 0x214dd99e610>]
```



Furkan zengin

Mathematical Economics

Alpha Chiang

Chapter 19

Simultaneous Differential Equations

In [2]:

```
from sympy import *

t = symbols('t')
x = Function('x')
y = Function('y')
dydt =  61 - x(t) - 4*y(t)
eqs = [
        Eq(x(t).diff(t) + 2*dydt + 2*x(t)+ 5*y(t) -77,0),
        Eq(y(t).diff(t) +x(t) + 4*y(t) -61,0)
    ]

pprint(eqs[0])
pprint(eqs[1])
```

```
           d
-3⋅y(t) + ──(x(t)) + 45 = 0
          dt
                 d
x(t) + 4⋅y(t) + ──(y(t)) - 61 = 0
                dt
```

In [3]:

```
ics = {x(0): 6, y(0): 12}
DD = dsolve(eqs, [x(t), y(t)],ics = ics)
print(DD)
```

```
[Eq(x(t), 1 + 3*exp(-t) + 2*exp(-3*t)), Eq(y(t), 15 - exp(-t) - 2*exp(-3*t))]
```

Using https://www.researchgate.net/profile/Stephen-Mason-8

In [8]:

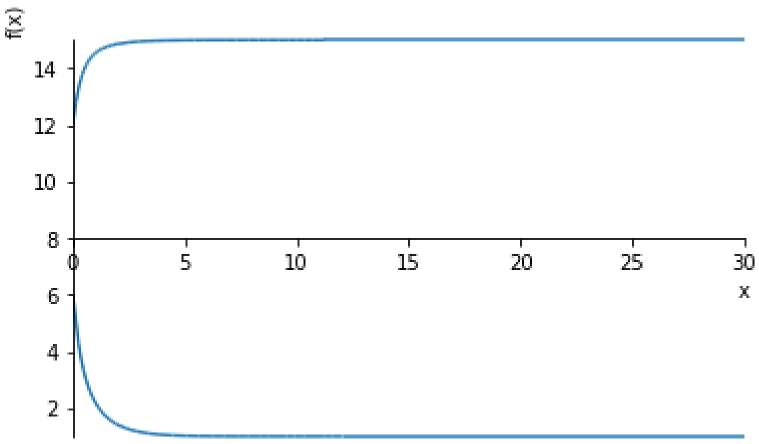
```python
import numpy as np
import matplotlib.pyplot as plt
from sympy import init_printing
init_printing()

from sympy import Function, Indexed, Tuple, sqrt, dsolve, solve, Eq, Derivative, sin, cos, symbols
from sympy.abc import k, t
from sympy import solve, Poly, Eq, Function, exp

from sympy import Indexed, IndexedBase, Tuple, sqrt
from IPython.display import display
from sympy import *
from sympy.abc import *
from sympy.plotting import plot
init_printing()
t, C1, C2 = symbols("t C1 C2")
x, y = symbols("x y", cls = Function, Function = True)


dydt =  61 - x(t) - 4*y(t)
eqs = [
        Eq(x(t).diff(t) + 2*dydt + 2*x(t)+ 5*y(t) -77,0),
        Eq(y(t).diff(t) +x(t) + 4*y(t) -61,0)
    ]
ics = {x(0): 6, y(0): 12}
soln = dsolve(eqs, [x(t), y(t)],ics = ics)

constants = solve((soln[0].subs(t,0).subs(x(0),1), soln[1].subs(t,0).subs(y(0),2)),{C1,C2})

xsoln = expand(soln[0].rhs.subs(constants))
display(xsoln)
print(xsoln)
ysoln = soln[1].rhs.subs(constants)
display(ysoln)
print(ysoln)

plot((xsoln, (t, 0, 30)), (ysoln, (t, 0, 30)))
```

$$1 + 3e^{-t} + 2e^{-3t}$$

```
1 + 3*exp(-t) + 2*exp(-3*t)
```

$$15 - e^{-t} - 2e^{-3t}$$

```
15 - exp(-t) - 2*exp(-3*t)
```



Out[8]: <sympy.plotting.plot.Plot at 0x7f74d0f35510>

EXERCISE 19.2 --Q/4/a--

In [9]:

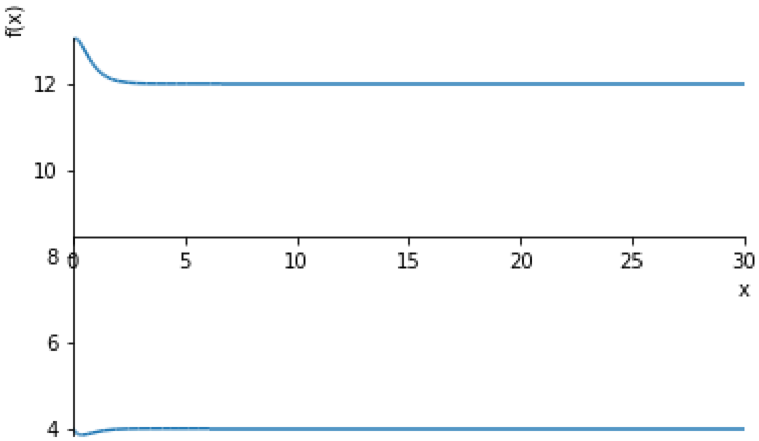
```
t, C1, C2 = symbols("t C1 C2")
x, y = symbols("x y", cls = Function, Function = True)


eqs = [
        Eq(x(t).diff(t) - x(t) - 12*y(t) + 60 ,0),
        Eq(y(t).diff(t) +x(t) + 6*y(t) - 36 ,0)
    ]
ics = {x(0): 13, y(0): 4}
soln = dsolve(eqs, [x(t), y(t)],ics = ics)

constants = solve((soln[0].subs(t,0).subs(x(0),1), soln[1].subs(t,0).subs(y(0),2)),{C1,C2})

xsoln = expand(soln[0].rhs.subs(constants))
display(xsoln)
print(xsoln)
ysoln = soln[1].rhs.subs(constants)
display(ysoln)
print(ysoln)
plot((xsoln, (t, 0, 30)), (ysoln, (t, 0, 30)))
```

$$12 + 4e^{-2t} - 3e^{-3t}$$

12 + 4*exp(-2*t) - 3*exp(-3*t)

$$4 - e^{-2t} + e^{-3t}$$

4 - exp(-2*t) + exp(-3*t)



Out[9]: <sympy.plotting.plot.Plot at 0x7f74d04f9710>

EXERCISE 19.2 --Q/4/b--

In [10]:

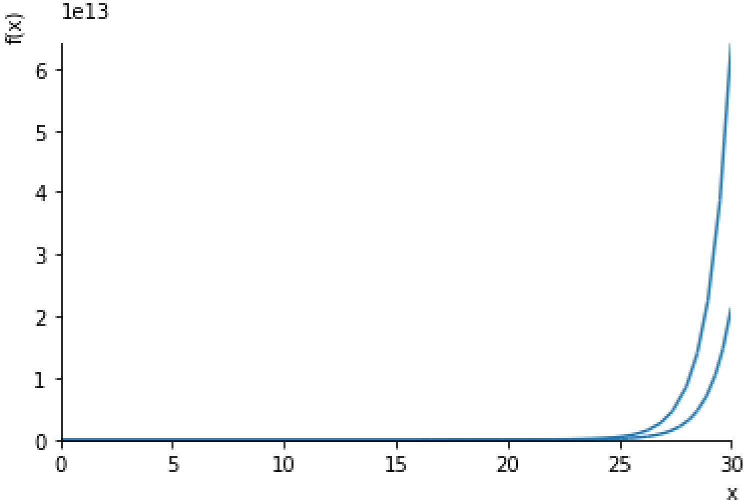
```
t, C1, C2 = symbols("t C1 C2")
x, y = symbols("x y", cls = Function, Function = True)


eqs = [
        Eq(x(t).diff(t) - 2*x(t) + 3*y(t) - 10 ,0),
        Eq(y(t).diff(t) - x(t) + 2*y(t) - 9 ,0)
     ]
ics = {x(0): 8, y(0): 5}
soln = dsolve(eqs, [x(t), y(t)],ics = ics)

constants = solve((soln[0].subs(t,0).subs(x(0),1), soln[1].subs(t,0).subs(y(0),2)),{C1,C2})

xsoln = expand(soln[0].rhs.subs(constants))
display(xsoln)
print(xsoln)
ysoln = soln[1].rhs.subs(constants)
display(ysoln)
print(ysoln)
plot((xsoln, (t, 0, 30)), (ysoln, (t, 0, 30)))
```

$$6e^{t} + 7 - 5e^{-t}$$

```
6*exp(t) + 7 - 5*exp(-t)
```

$$2e^{t} + 8 - 5e^{-t}$$

```
2*exp(t) + 8 - 5*exp(-t)
```



Out[10]: <sympy.plotting.plot.Plot at 0x7f74d0386350>

Simultaneous Difference Equations

In [11]:

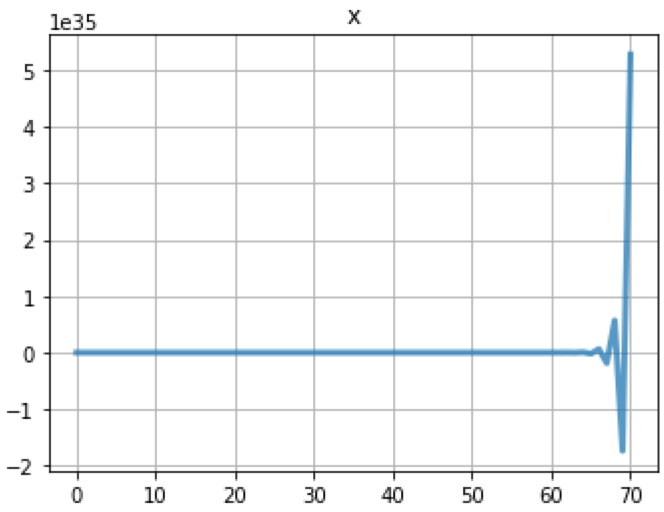
```
from sympy import Symbol, dsolve, Function, Derivative, Eq

from sympy import Function, rsolve
from sympy.abc import t,c
y = Function("y");
y0 = Symbol("y_0")
a1 = Symbol("a_1")
a2 = Symbol("a_2")

f = y(t+1) + 6*y(t) + 9*y(t-1) - 4
sol = rsolve(f, y(t), {y(0):1});
sol
```

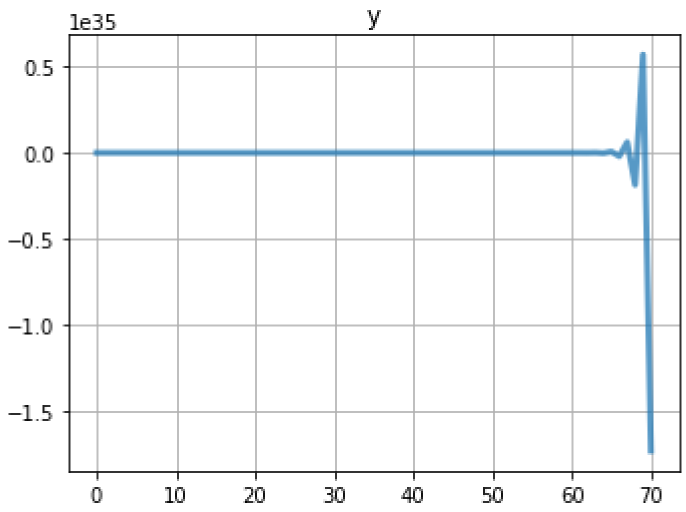
Out[11]:

$$(-3)^{t}\left(C_{1}t + \frac{3}{4}\right) + \frac{1}{4}$$

In [12]:

```
T = 71
x = np.zeros(T)
x[0] = 1

y = np.zeros(T)

y[0] = 1

for t in range(T-1):
    x[t+1] = -6*x[t] - 9*y[t] + 4
    y[t+1] = x[t]



fig = plt.figure(figsize=(12,4))

ax = fig.add_subplot(1,2,1)
ax.plot(x,lw=3,alpha=0.75)
ax.set_title('x')
ax.grid()

ax = fig.add_subplot(1,2,2)
ax.plot(y,lw=3,alpha=0.75)
ax.set_title('y')
ax.grid()
```





The Inflation-Unemployment Model Once More

In [13]:

```
from sympy import *

C1, C2 = symbols("C1 C2")
k,j, g, beta,alpha,T,mu = symbols("k j g \\beta \\alpha T \\mu")
t = symbols('t')
x = Function('x')
y = Function('y')

eqs = [
        Eq(x(t).diff(t) - j*(1-g)*x(t) + (j*beta)*y(t)- j*(alpha-T) ,0),
        Eq(y(t).diff(t) + k*g*x(t) + k*beta*y(t) - k*(alpha-T-mu) ,0)
    ]

pprint(eqs[0])
pprint(eqs[1])
```

```
                                                         d
\beta⋅j⋅y(t) - j⋅(1 - g)⋅x(t) - j⋅(-T + \alpha) + ──(x(t)) = 0
                                                         dt
                                                         d
\beta⋅k⋅y(t) + g⋅k⋅x(t) - k⋅(-T + \alpha - \mu) + ──(y(t)) = 0
                                                         dt
```

In [14]:

```
DD = dsolve(eqs, [x(t), y(t)])
```

In [15]:

```
constants = solve((DD[0].subs(t,0).subs(x(0),1), DD[1].subs(t,0).subs(y(0),2)),{C1,C2})

xsoln = expand(DD[0].rhs.subs(constants))
ysoln = DD[1].rhs.subs(constants)
```

In [16]:

```
C1, C2 = symbols("C1 C2")
k,j, g, beta,alpha,T,mu = symbols("k j g \\beta \\alpha T \\mu")
t = symbols('t')
x = Function('x')
y = Function('y')

eqs = [
        Eq(x(t).diff(t) - 3/4*(1-1)*x(t) + (3/4*3)*y(t)- 3/4*(1/6) ,0),
        Eq(y(t).diff(t) + 1/2*1*x(t) + 1/2*3*y(t) - 1/2*(1/6-mu) ,0)
    ]
```

```
pprint(eqs[0])
pprint(eqs[1])
```

```
                d
2.25⋅y(t) + ──(x(t)) - 0.125 = 0
                dt
                                          d
0.5⋅\mu + 0.5⋅x(t) + 1.5⋅y(t) + ──(y(t)) - 0.0833333333333333 = 0
                                          dt
```

In [17]:

```
DD = dsolve(eqs, [x(t), y(t)])
```

In [18]:

```
constants = solve((DD[0].subs(t,0).subs(x(0),1), DD[1].subs(t,0).subs(y(0),2)),{C1,C2})

xsoln = expand(DD[0].rhs.subs(constants))
ysoln = DD[1].rhs.subs(constants)
```

Furkan zengin

In [ ]:

```

```