PROBLEM 1

a) $X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 5 \end{bmatrix}$ $Y = \begin{bmatrix} 4 \\ 7 \\ 9 \\ 12 \\ 18 \end{bmatrix}$

$$X^T X = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 5 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 5 \end{bmatrix}$$

$$= \begin{bmatrix} 5 & 11 \\ 11 & 39 \end{bmatrix}$$

$$(X^T X)^{-1} = \begin{bmatrix} 39/74 & -11/74 \\ -11/74 & 5/74 \end{bmatrix}$$

$$(X^T X)^{-1} X^T = \begin{bmatrix} 39/74 & 14/37 & 17/74 & 3/37 & -8/37 \\ -11/74 & -3/37 & -1/74 & 2/37 & 7/37 \end{bmatrix}$$

$$\theta = (X^T X)^{-1} X^T Y = \begin{bmatrix} 289/74 \\ 205/74 \end{bmatrix}$$

$$= \begin{bmatrix} 3.9054 \\ 2.7702 \end{bmatrix}$$

$$\boxed{y = 3.9054 + 2.7702x}$$

PROBLEM 3

a) $f(x,\theta) = \frac{\theta}{x^2} e^{-\frac{\theta}{x}}$

(a1)

$$L(\theta) = \prod_i \frac{\theta}{x_i^2} e^{-\frac{\theta}{x_i}}$$

$$L(\theta) = \sum_i \left[\log\theta - 2\log x_i - \frac{\theta}{x_i} \right]$$

$$n\log\theta - 2\sum \log x_i - \theta \sum \frac{1}{x_i} = L(\theta)$$

Differentiate and equate to 0

$$\frac{n}{\theta} - \sum \frac{1}{x_i} = 0$$

~~$\theta = n / [1/$~~

$$\boxed{\theta = \frac{n}{\sum_i (1/x_i)}}$$

(a2) $f(5,\theta) = \frac{\theta}{25} e^{-\frac{\theta}{5}}$

$f(6,\theta) = \frac{\theta}{36} e^{-\frac{\theta}{6}}$

$f(4,\theta) = \frac{\theta}{16} e^{-\frac{\theta}{4}}$

$L(\theta) = \prod_i \frac{\theta}{25} \times \frac{\theta}{36} \times \frac{\theta}{16} e^{\left[-\frac{\theta}{5} - \frac{\theta}{6} - \frac{\theta}{4}\right]}$

$L(\theta) = 3 \log \theta - \log(25 \times 36 \times 16) - \theta\left[\frac{1}{5} + \frac{1}{6} + \frac{1}{4}\right]$

Differentiating & equation to 0

$\frac{3}{\theta} = \left[\frac{1}{5} + \frac{1}{6} + \frac{1}{4}\right]$

$\theta = \frac{3}{37/60} = 4.87$ ~~4.87~~ 4.864

Alternate method: use 3(a1)

$\theta = \frac{n}{\sum 1/x_i} = \frac{3}{\left(\frac{1}{5} + \frac{1}{6} + \frac{1}{4}\right)} = 4.864$

PROBLEM 3

b) b1) $P(y:\phi) = (1-\phi)^{y-1}\phi$

$= \exp\left(\log(1-\phi)^{y-1} \cdot \log\phi\right)$

$= \exp\left[y\log(1-\phi) - \log\left(\frac{1-\phi}{\phi}\right)\right]$

This is the exponential family $\longrightarrow$ ①

Here $b(y) = 1$

$\eta = \log(1-\phi)$

$T(y) = y$

$a(\eta) = \cancel{\log(1-\phi)}$ $\log\left(\frac{1-\phi}{\phi}\right) = \log\left(\frac{e^{\eta}}{1-e^{\eta}}\right)$

b2) Canonical response function

$g(\eta) = E(y|\phi)$ = expected value which is mea

$g(\eta) = \frac{1}{\phi} = \frac{1}{1-e^{\eta}}$

b3) From b)

$$P(y|\phi) = \exp\left(y \log(1-\phi) - \log\left(\frac{1-\phi}{\phi}\right)\right)$$

$$= \exp\left(y\eta - \log\left(\frac{e^{\eta}}{1-e^{\eta}}\right)\right)$$

$$L(\theta) = \log\left[\exp\left(y\eta - \log\left(\frac{e^{\eta}}{1-e^{\eta}}\right)\right)\right]$$

~~$= \log\left[\exp\left(\theta^T x_i \cdot \text{Put } y = \theta^T x^i \quad y = y^i\right)\right.$~~

By rules of GLM model, $\eta = \theta^T x^i \quad y = y^i$

$$L(\theta) = \theta^T x^i y^i + \log\left(\frac{1-e^{\theta^T x^i}}{e^{\theta^T x^i}}\right)$$

$$L(\theta) = \theta^T x^i y^i + \log\left(e^{-\theta^T x^i} - 1\right)$$

Differentiate this

$$\frac{\partial}{\partial \theta_j} L(\theta) = x_j^i y^i - \frac{1}{1-e^{-\theta^T x^i}} x_j^i$$

$$= \left(y^i - \frac{1}{1-e^{-\theta^T x^i}}\right) x_j^i$$

Stochastic gradient ascent is:

$$\theta_j := \theta_j + \alpha \frac{\partial L_i(\theta)}{\partial \theta_j}$$

$$\theta_j = \theta_j + \alpha \left(y^i - \frac{1}{1-e^{-\theta^T x^i}}\right) x_j^i$$

$$\frac{\partial}{\partial \theta_j} l(\theta) = y^i x^i + \frac{e^{-\theta^T x^i}}{e^{-\theta^T x^i} - 1} (-x^i)$$

$$= y^i x_j^i - \frac{x_j^i}{1 - e^{+\theta^T x^i}}$$

Stochastic gradient descend

$$\theta_j \rightarrow \theta_j + \alpha \frac{\partial L, \theta}{\partial \theta_j}$$

$$\boxed{\theta_j \rightarrow \theta_j + \alpha \left(y^i - \frac{1}{1 - e^{\theta^T x^i}} \right) x_j^i}$$

In [4]:
```
#Problem 1b)
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import inv

dataset=np.genfromtxt('https://canvas.cmu.edu/courses/6620/files/2805738/downloa
X1= np.array([0,1,2,3,5])
Y= np.array([4,7,9,12,18])
n = len(X1)
X0=np.ones((n))
X=np.column_stack((X0,X1))


#using normal equations
XT = np.transpose(X)
t = np.linalg.inv(XT.dot(X))
t = t.dot(XT)
t = t.dot(Y)

print("b0 = ",t[0], ",   b1 =",t[1])
print("y= %f + %fx" % (t[0],t[1]))
```

```
b0 =  3.9054054054054066 ,   b1 = 2.77027027027027
y= 3.905405 + 2.770270x
```

In [3]:
```
#Problem 1c)
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import inv

dataset=np.genfromtxt('https://canvas.cmu.edu/courses/6620/files/2805738/downloa
X1= (dataset[:,0])
Y= (dataset[:,1])
n = len(X1)
X0=np.ones((n))
X=np.column_stack((X0,X1))


#using normal equations
XT = np.transpose(X)
t = np.linalg.inv(XT.dot(X))
t = t.dot(XT)
t = t.dot(Y)

print("b0 = ",t[0], ",   b1 =",t[1])
print("y= %f + %fx" % (t[0],t[1]))
```

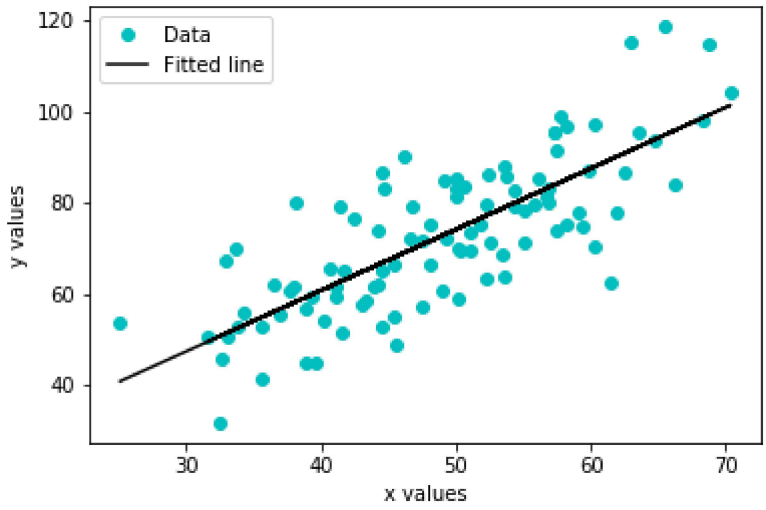
```
b0 =  7.99102098226934 ,   b1 = 1.3224310227553822
y= 7.991021 + 1.322431x
```

In [ ]:

In [52]:
```
#Problem 2a)
#import libraries
import numpy as np
import matplotlib.pyplot as plt

#create dataset
dataset=np.genfromtxt('https://canvas.cmu.edu/courses/6620/files/2805738/downloa
X1= (dataset[:,0])
Y= (dataset[:,1])
n = len(X1)
X0=np.ones((n))
X=np.column_stack((X0,X1))

#initial parameter setting
alpha=0.0001
b=7
m=1
iteration=200

#cost function
def cost_fun(b,m):
    error = 0.0
    for i in range(0,n):
        error += (Y[i] - (m*X1[i] +b))**2
    J= error / (n*2)
    return J

#define gradient descent
def gradient_descent(b,m,alpha,iteration):
    b_c=b
    m_c=m
    for j in range(iteration):
        g0=0
        g1=0
        for i in range(n):
            g0+=(Y[i] - (m_c*X1[i] +b_c))*(-1)
            g1+=X1[i]*(Y[i] - (m_c*X1[i] +b_c))*(-1)
        b_c=b_c-(alpha*g0)/n
        m_c=m_c-(alpha*g1)/n
    return[b_c,m_c]

for i in range (iteration):
    b,m=gradient_descent(b,m, alpha, iteration)
    error=cost_fun(b,m)
print("b0 = ",b, ",   b1 =",m)
print("y= %f + %fx" % (b,m))

plt.xlabel('x values')
plt.ylabel('y values')
plt.plot(X1, Y,'co', X1, b+m*X1,'k')
plt.legend(['Data','Fitted line'])
plt.show()
```

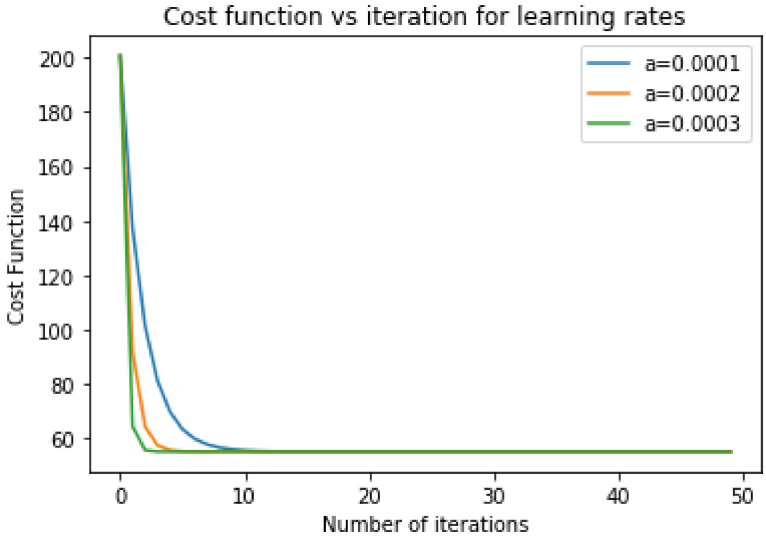
```
b0 =  7.144631959366427 ,   b1 = 1.3390665411156633
```

```
y= 7.144632 + 1.339067x
```



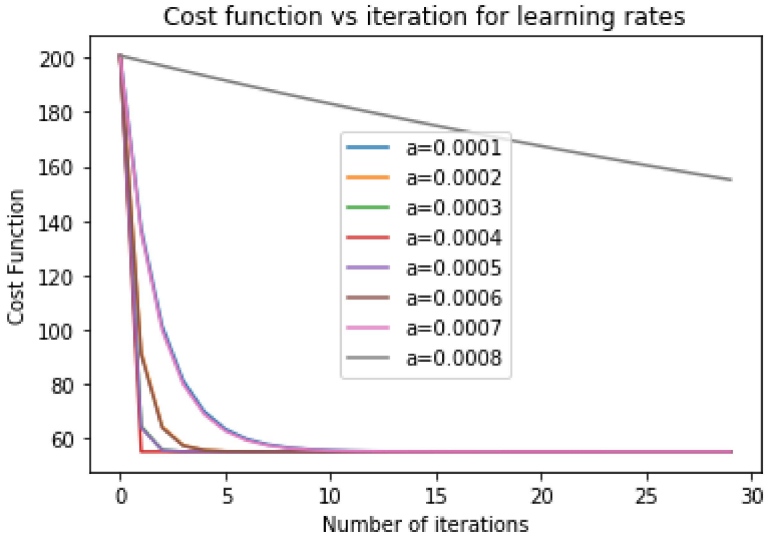
In [46]:

```
#Problem 2b)
import numpy as np
import matplotlib.pyplot as plt

dataset=np.genfromtxt('https://canvas.cmu.edu/courses/6620/files/2805738/downloa
X1= (dataset[:,0])
Y= (dataset[:,1])
n = len(X1)
X0=np.ones((n))
X=np.column_stack((X0,X1))

alpha=3
b=7
m=1
iteration=50
list=[]
e = np.zeros((iteration,alpha))

def cost_function(r,t):
    error = 0.0
    for i in range(0,n):
        error =error+ (Y[i] - (t*X1[i] +r))**2
    J= error / (n*2)
    return J

def gradient_descent(b,m,s,iteration):
    b_c=b
    m_c=m
    for j in range(iteration):
        g0=0
        g1=0
        for i in range(n):
            g0=g0+(Y[i] - (m_c*X1[i] +b_c))*(-1)
            g1=g1+X1[i]*(Y[i] - (m_c*X1[i] +b_c))*(-1)

        b_c=b_c-(((s+1)/10000)*g0)/n
        m_c=m_c-(((s+1)/10000)*g1)/n
    return[b_c,m_c]

for j in range(alpha):
    for k in range (iteration):
        if j==0:
            list.append(k)
        r,t=gradient_descent(b,m, j, list[k])

        e[k,j]=cost_function(r,t)

    plt.plot(list,e[:,j])#list,e[:,1],'c',list,e[:,2],'b')
   # print(list,e[:,j])
plt.xlabel('Number of iterations')
plt.ylabel('Cost Function')
plt.title('Cost function vs iteration for learning rates')
plt.legend(['a=0.0001','a=0.0002','a=0.0003'])

plt.show()
```


Cost function vs iteration for learning rates
a=0.0001
a=0.0002
a=0.0003
Cost Function
Number of iterations
200
180
160
140
120
100
80
60
0
10
20
30
40
50

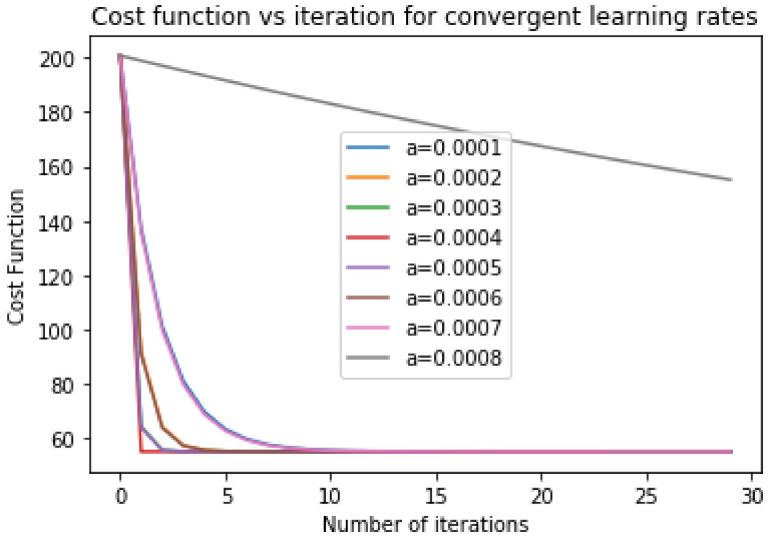
In [48]:
```
#Problem 2c)
import numpy as np
import matplotlib.pyplot as plt

dataset=np.genfromtxt('https://canvas.cmu.edu/courses/6620/files/2805738/downloa
X1= (dataset[:,0])
Y= (dataset[:,1])
n = len(X1)
X0=np.ones((n))
X=np.column_stack((X0,X1))

alpha=8
b=7
m=1
iteration=30
list=[]
e = np.zeros((iteration,alpha))

def cost_function(r,t):
    error = 0.0
    for i in range(0,n):
        error =error+ (Y[i] - (t*X1[i] +r))**2
    J= error / (n*2)
    return J

def gradient_descent(b,m,s,iteration):
    b_c=b
    m_c=m
    for j in range(iteration):
        g0=0
        g1=0
        for i in range(n):
            g0=g0+(Y[i] - (m_c*X1[i] +b_c))*(-1)
            g1=g1+X1[i]*(Y[i] - (m_c*X1[i] +b_c))*(-1)

        b_c=b_c-(((s+1)/10000)*g0)/n
        m_c=m_c-(((s+1)/10000)*g1)/n
    return[b_c,m_c]

for j in range(alpha):
    for k in range (iteration):
        if j==0:
            list.append(k)
        r,t=gradient_descent(b,m, (j), list[k])

        e[k,j]=cost_function(r,t)

    plt.plot(list,e[:,j])#list,e[:,1],'c',list,e[:,2],'b')
   # print(list,e[:,j])
plt.xlabel('Number of iterations')
plt.ylabel('Cost Function')
plt.title('Cost function vs iteration for learning rates')
plt.legend(['a=0.0001','a=0.0002','a=0.0003','a=0.0004','a=0.0005','a=0.0006','a
plt.show()
#From the graph above, learning rate 0.0004
#It reaches a minimum cost function the fastest.
```

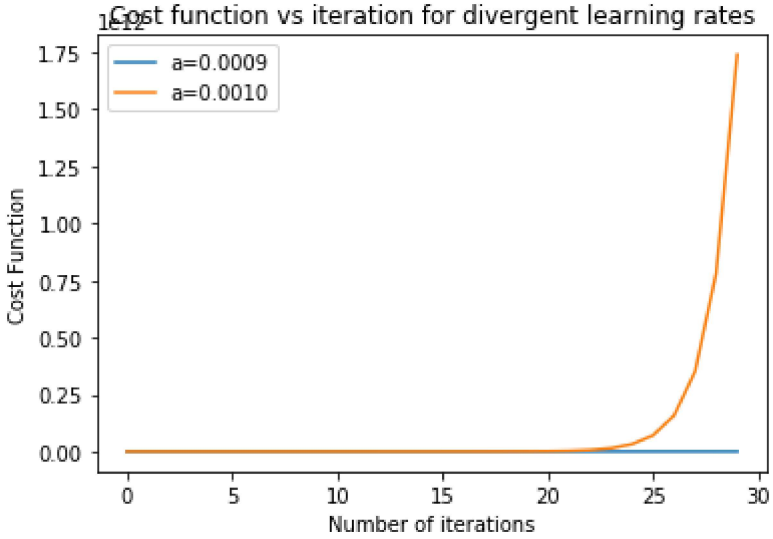
```
#Therefore, the optimum learning rate is 0.0004
```



In [49]:
```
#Problem 2d)
import numpy as np
import matplotlib.pyplot as plt

dataset=np.genfromtxt('https://canvas.cmu.edu/courses/6620/files/2805738/downloa
X1= (dataset[:,0])
Y= (dataset[:,1])
n = len(X1)
X0=np.ones((n))
X=np.column_stack((X0,X1))

alpha=10
b=7
m=1
iteration=30
list=[]
e = np.zeros((iteration,alpha))

def cost_function(r,t):
    error = 0.0
    for i in range(0,n):
        error =error+ (Y[i] - (t*X1[i] +r))**2
    J= error / (n*2)
    return J

def gradient_descent(b,m,s,iteration):
    b_c=b
    m_c=m
    for j in range(iteration):
        g0=0
        g1=0
        for i in range(n):
            g0=g0+(Y[i] - (m_c*X1[i] +b_c))*(-1)
            g1=g1+X1[i]*(Y[i] - (m_c*X1[i] +b_c))*(-1)

        b_c=b_c-(((s+1)/10000)*g0)/n
        m_c=m_c-(((s+1)/10000)*g1)/n
    return[b_c,m_c]

for j in range(alpha):
    for k in range (iteration):
        if j==0:
            list.append(k)
        r,t=gradient_descent(b,m, j, list[k])

        e[k,j]=cost_function(r,t)
    if j<8:
        plt.plot(list,e[:,j])#list,e[:,1],'c',list,e[:,2],'b')
plt.xlabel('Number of iterations')
plt.ylabel('Cost Function')
plt.title('Cost function vs iteration for convergent learning rates')
plt.legend(['a=0.0001','a=0.0002','a=0.0003','a=0.0004','a=0.0005','a=0.0006','a
plt.figure(2)
plt.plot(list,e[:,8], e[:,9])
plt.xlabel('Number of iterations')
plt.ylabel('Cost Function')
```

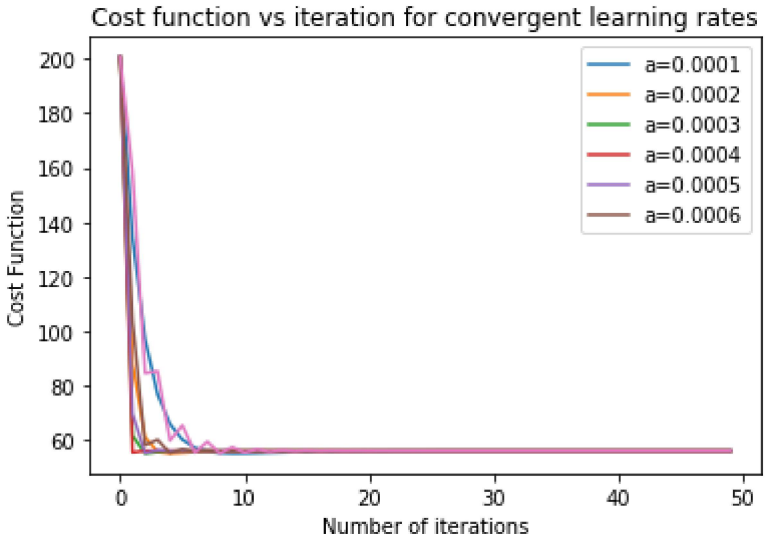
```
plt.title('Cost function vs iteration for divergent learning rates')
plt.legend(['a=0.0009','a=0.0010'])
plt.show()


#Approximate upper bound of learning rate is 0.0008 beyond which divergence
#occurs. Divergence occurs at all learning rates above 0.0008.
```





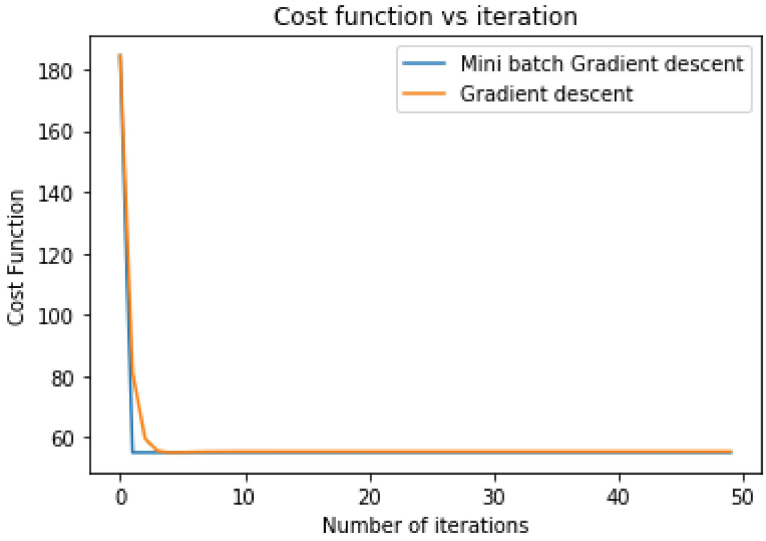
In [33]:
```
#Problem 2e)
import numpy as np
import matplotlib.pyplot as plt

dataset=np.genfromtxt('https://canvas.cmu.edu/courses/6620/files/2805738/downloa
X1= (dataset[:,0])
Y= (dataset[:,1])
n = len(X1)
X0=np.ones((n))
X=np.column_stack((X0,X1))

alpha=7
b=7
m=1
iteration=50

list=[]
e = np.zeros((iteration,alpha))
e1=[]
minibatch = np.random.choice(n, 20, replace=False)
def cost_function(r,t):
    error = 0.0
    for i in range(0,n):
        error =error+ (Y[i] - (t*X1[i] +r))**2
    J= error / (n*2)
    return J

def gradient_descent(b,m,s,iteration):
    b_c=b
    m_c=m
    for j in range(iteration):
        g0=0
        g1=0
        for i in range(n):
            g0=g0+(Y[i] - (m_c*X1[i] +b_c))*(-1)
            g1=g1+X1[i]*(Y[i] - (m_c*X1[i] +b_c))*(-1)

        b_c=b_c-(((s+1)/10000)*g0)/n
        m_c=m_c-(((s+1)/10000)*g1)/n
    return[b_c,m_c]

def minibatch_descent(b,m,s,iteration, minibatch):
    b_c=b
    m_c=m
    for j in range(iteration):
        g0=0
        g1=0
        for i in range(n):
            if i in minibatch:
                g0=g0+(Y[i] - (m_c*X1[i] +b_c))*(-1)
                g1=g1+X1[i]*(Y[i] - (m_c*X1[i] +b_c))*(-1)

        b_c=b_c-(((s+1)/10000)*g0)/20
        m_c=m_c-(((s+1)/10000)*g1)/20
    return[b_c,m_c]
```

```
for j in range(alpha):
    for k in range (iteration):
        if j==0:
            list.append(k)
        r,t=minibatch_descent(b,m, j, list[k], minibatch)

        e[k,j]=cost_function(r,t)

    plt.plot(list,e[:,j])#list,e[:,1],'c',list,e[:,2],'b')
plt.xlabel('Number of iterations')
plt.ylabel('Cost Function')
plt.title('Cost function vs iteration for convergent learning rates')
plt.legend(['a=0.0001','a=0.0002','a=0.0003','a=0.0004', 'a=0.0005','a=0.0006'])
plt.show()

#Optimum learning rate for mini batch gradient descent is 0.0004
```



In [42]:
```
#Problem 2e) continued
import numpy as np
import matplotlib.pyplot as plt

dataset=np.genfromtxt('https://canvas.cmu.edu/courses/6620/files/2805738/downloa
X1= (dataset[:,0])
Y= (dataset[:,1])
n = len(X1)
X0=np.ones((n))
X=np.column_stack((X0,X1))

alpha=3
b=8
m=1
iteration=50

list=[]
e = []
e1=[]
minibatch = np.random.choice(n, 20, replace=False)
def cost_function(r,t):
    error = 0.0
    for i in range(0,n):
        error =error+ (Y[i] - (t*X1[i] +r))**2
    J= error / (n*2)
    return J

def gradient_descent(b,m,s,iteration):
    b_c=b
    m_c=m
    for j in range(iteration):
        g0=0
        g1=0
        for i in range(n):
            g0=g0+(Y[i] - (m_c*X1[i] +b_c))*(-1)
            g1=g1+X1[i]*(Y[i] - (m_c*X1[i] +b_c))*(-1)

        b_c=b_c-(((s+1)/10000)*g0)/n
        m_c=m_c-(((s+1)/10000)*g1)/n
    return[b_c,m_c]

def minibatch_descent(b,m,s,iteration, minibatch):
    b_c=b
    m_c=m
    for j in range(iteration):
        g0=0
        g1=0
        for i in range(n):
            if i in minibatch:
                g0=g0+(Y[i] - (m_c*X1[i] +b_c))*(-1)
                g1=g1+X1[i]*(Y[i] - (m_c*X1[i] +b_c))*(-1)

        b_c=b_c-(((s+1)/10000)*g0)/20
        m_c=m_c-(((s+1)/10000)*g1)/20
    return[b_c,m_c]
```

```
for k in range (iteration):
    list.append(k)
    r,t=minibatch_descent(b,m,1, list[k], minibatch)
    e.append(cost_function(r,t))



for k in range (iteration):
    z1,z2=gradient_descent(b,m, 3, list[k])
    e1.append(cost_function(z1,z2))

plt.plot(list,e1,list,e)
plt.xlabel('Number of iterations')

plt.ylabel('Cost Function')
plt.title('Cost function vs iteration')
plt.legend(['Mini batch Gradient descent','Gradient descent'])
plt.show()
```



In [41]:
```
#Mini batch gradient descent takes 1 iteration to converge and
#attain minimum cost function.
#Gradient descent takes more iterations to
#converge at an optimum learning rate of 0.0004.
#Mini batch thus avoids arrival at local minima,
#it converges fastest to global minimum than gradient descent
#and is computationally faster
```

In [ ]:
```

```