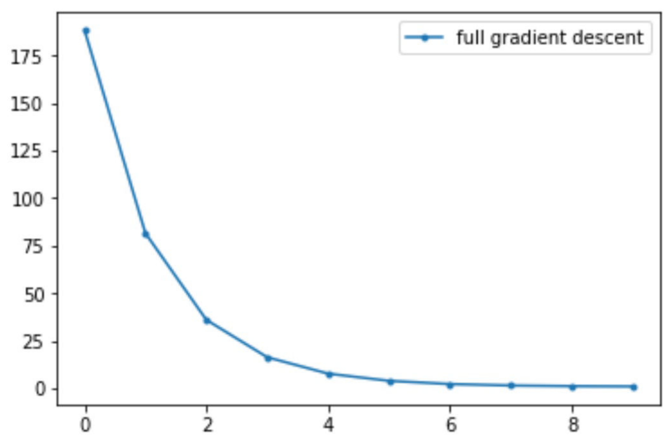
In [67]:

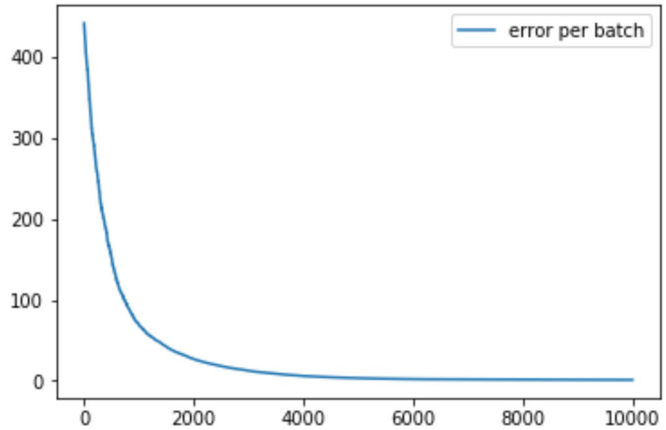
```
import numpy as np
import numpy.linalg as npla
import matplotlib.pyplot as plt
import scipy.io as scio
import random

DATAFILE = "gradient_descent_data.mat"

data = scio.loadmat(DATAFILE)
A = data['x']     # (1000, 2)
y = data['y']     # (1000, 1)
n, d = A.shape     # n = 1000 number of samples, d = 2 number of features
```

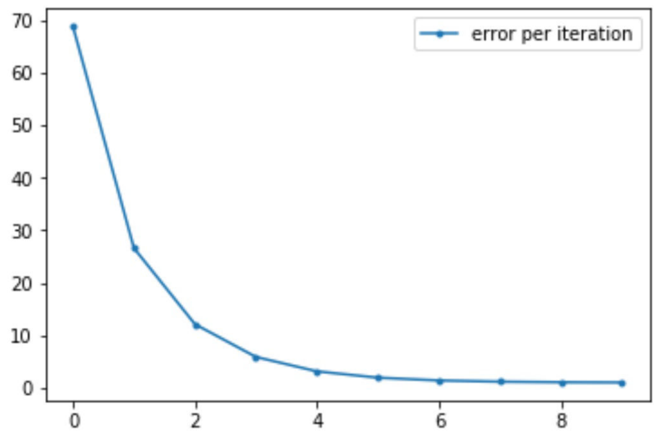
In [103]:

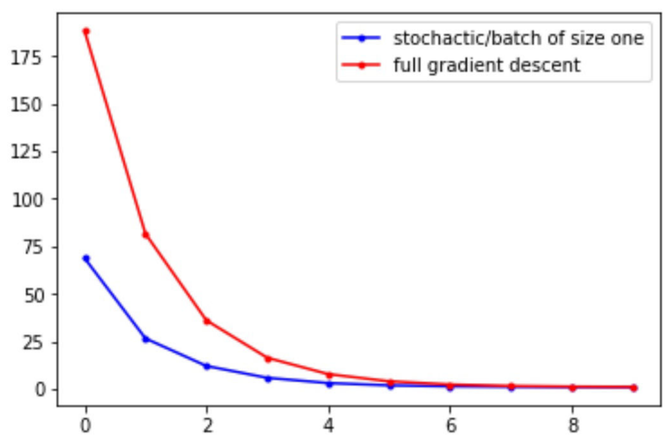
```
# Problem 3.a

alpha = 0.05
w = np.zeros( (d, 1), "float" )
errors_3a = np.zeros( (10, 1), "float" )
for i in range(10):
    grad_loss = 2 * A.T.dot(A).dot(w) - 2 * A.T.dot(y)
    w = w - alpha * grad_loss / (2 * n)
    errors_3a[i] = np.mean((A.dot(w) - y) ** 2)

plt.figure()
plt.plot(errors_3a, ".-", label="full gradient descent")
plt.legend(loc="best")
plt.show()
```



In [104]:
```
# Problem 3.b

alpha = 0.05
w = np.zeros( (d, 1), "float" )
errors_per_iter = np.zeros( (10, 1), "float" )
errors_per_batch = np.zeros( (10, n), "float" )
for i in range(10):
    for j in range(n):
        grad_loss = 2 * A[[j],:].T.dot(A[[j],:]).dot(w) - 2 * A[[j],:].T.dot(y[[
j]])
        w = w - alpha * grad_loss / (2 * n)
        errors_per_batch[i,j] = np.mean((A.dot(w) - y) ** 2)
    errors_per_iter[i] = np.mean((A.dot(w) - y) ** 2)

plt.figure()
plt.plot(errors_per_batch.flatten(), label="error per batch")
plt.legend(loc="best")
plt.figure()
plt.plot(errors_per_iter, ".-", label="error per iteration")
plt.legend(loc="best")
plt.figure()
plt.plot(errors_per_iter, "b.-", label="stochactic/batch of size one")
plt.plot(errors_3a, "r.-", label="full gradient descent")
plt.legend(loc="best")
plt.show()
```

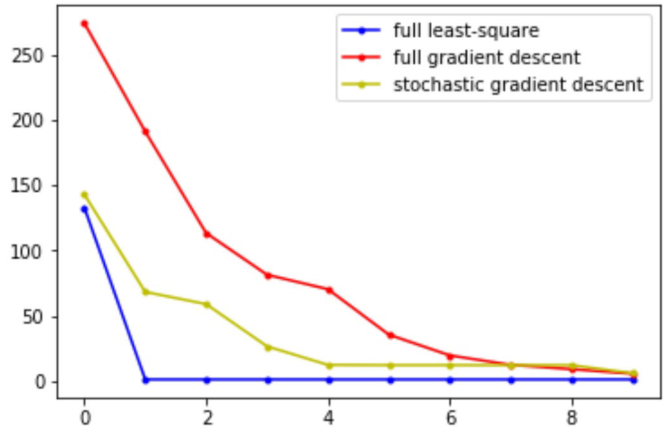

error per batch
0
100
200
300
400
0
2000
4000
6000
8000
10000

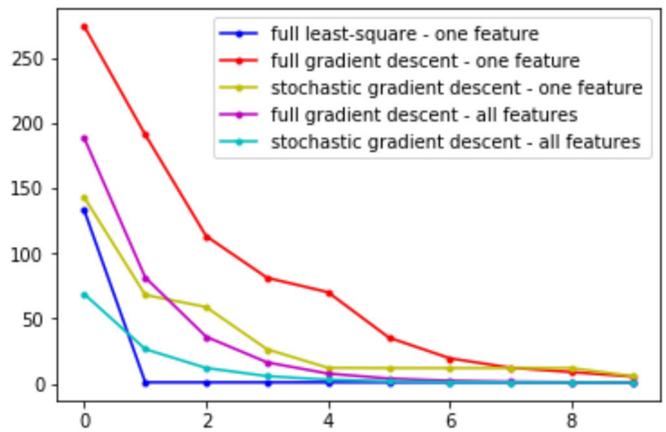

error per iteration
0
10
20
30
40
50
60
70
0
2
4
6
8


stochactic/batch of size one
full gradient descent
0
25
50
75
100
125
150
175
0
2
4
6
8

In [81]:

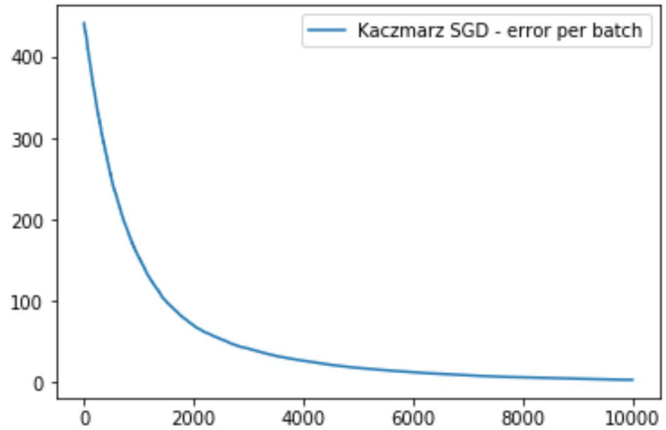
```
# Problem 3.c

alpha = 0.05
w1 = np.zeros( (d, 1), "float" )
w2 = np.zeros( (d, 1), "float" )
w3 = np.zeros( (d, 1), "float" )
errors1 = np.zeros( (10, 1), "float" )
errors2 = np.zeros( (10, 1), "float" )
errors3 = np.zeros( (10, 1), "float" )

for i in range(10):
    r = random.random()
    k = 0 if r < 0.5 else 1
    # full least-square
    w1[k] = npla.inv( A[:,[k]].T.dot(A[:,[k]]) ).dot( A[:,[k]].T.dot(y) )
    errors1[i] = np.mean((A.dot(w1) - y) ** 2)
    # full gradient descent
    grad_loss = 2 * A.T.dot(A).dot(w2) - 2 * A.T.dot(y)
    w2[k] = w2[k] - alpha * grad_loss[k] / (2 * n)
    errors2[i] = np.mean((A.dot(w2) - y) ** 2)
    # stochastic gradient descent
    for j in range(n):
        grad_loss = 2 * A[[j],:].T.dot(A[[j],:]).dot(w3) - 2 * A[[j],:].T.dot(y[
[j]])
        w3[k] = w3[k] - alpha * grad_loss[k] / (2 * n)
    errors3[i] = np.mean((A.dot(w3) - y) ** 2)

plt.figure()
plt.plot(errors1, "b.-", label="full least-square")
plt.plot(errors2, "r.-", label="full gradient descent")
plt.plot(errors3, "y.-", label="stochastic gradient descent")
plt.legend(loc="best")
plt.figure()
plt.plot(errors1, "b.-", label="full least-square - one feature")
plt.plot(errors2, "r.-", label="full gradient descent - one feature")
plt.plot(errors3, "y.-", label="stochastic gradient descent - one feature")
plt.plot(errors_3a, "m.-", label="full gradient descent - all features")
plt.plot(errors_per_iter, "c.-", label="stochastic gradient descent - all featur
es")
plt.legend(loc="best")
plt.show()
```

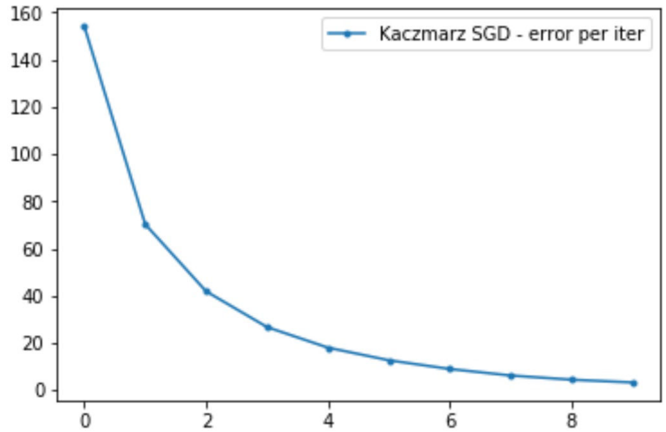

full least-square
full gradient descent
stochastic gradient descent
0
50
100
150
200
250
0
2
4
6
8

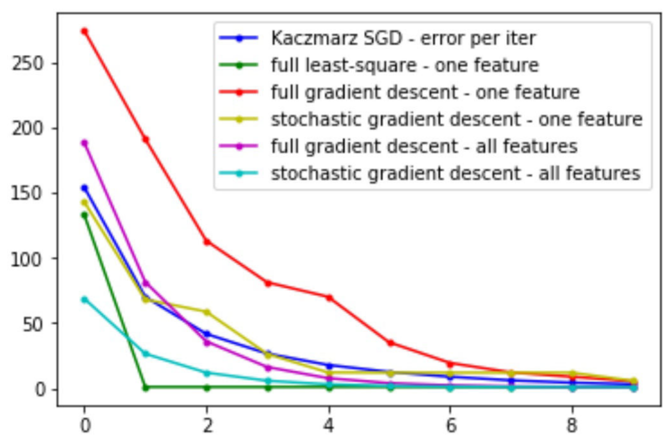

full least-square - one feature
full gradient descent - one feature
stochastic gradient descent - one feature
full gradient descent - all features
stochastic gradient descent - all features
0
50
100
150
200
250
0
2
4
6
8

In [82]:

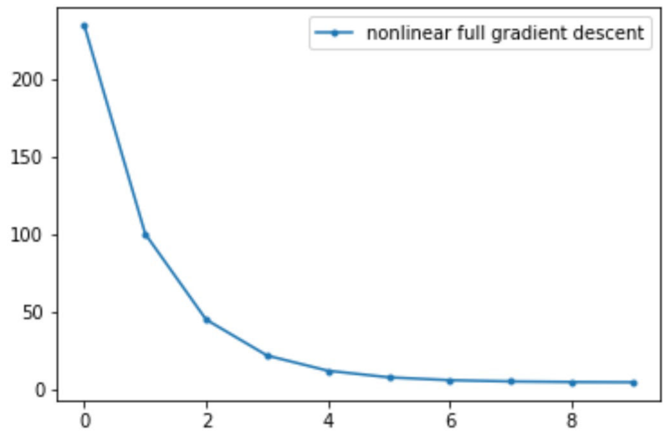
```
# Problem 3.d

w = np.zeros( (d, 1), "float" )
errors_per_iter_ aczmarz = np.zeros( (10, 1), "float" )
errors_per_batch_ aczmarz = np.zeros( (10, n), "float" )

alpha_list = []
prob_list = []
cummulative_prob_list = []
A_Froberius = np.sum(A ** 2)
for j in range(n):
    Aj_Froberius = np.sum( A[j, :] ** 2 )
    alpha_list.append(1 / Aj_Froberius)
    prob_list.append(Aj_Froberius / A_Froberius)
    if j == 0:
        cummulative_prob_list.append(Aj_Froberius / A_Froberius)
    else:
        cummulative_prob_list.append(cummulative_prob_list[j - 1]   Aj_Froberius
/ A_Froberius)

for i in range(10):
    for j in range(n):
        r = random.random()
        index = 0
          ile cummulative_prob_list[index] < r:
            index  = 1
        grad_loss = 2 * A[[index],:].T.dot(A[[index],:]).dot(w) - 2 * A[[index],
:].T.dot(y[[index]])
        w = w - alpha_list[index] * grad_loss / (2 * n)
        errors_per_batch_ aczmarz[i,j] = np.mean((A.dot(w) - y) ** 2)
    errors_per_iter_ aczmarz[i] = np.mean((A.dot(w) - y) ** 2)

plt.figure()
plt.plot(errors_per_batch_ aczmarz.flatten(), label=" aczmarz   D - error per ba
tch")
plt.legend(loc="best")
plt.figure()
plt.plot(errors_per_iter_ aczmarz, ".-", label=" aczmarz   D - error per iter")
plt.legend(loc="best")
plt.figure()
plt.plot(errors_per_iter_ aczmarz, "b.-", label=" aczmarz   D - error per iter")
plt.plot(errors1, "g.-", label="full least-square - one feature")
plt.plot(errors2, "r.-", label="full gradient descent - one feature")
plt.plot(errors3, "y.-", label="stochastic gradient descent - one feature")
plt.plot(errors_3a, "m.-", label="full gradient descent - all features")
plt.plot(errors_per_iter, "c.-", label="stochastic gradient descent - all featur
es")
plt.legend(loc="best")
plt.show()
```


Kaczmarz SGD - error per batch
0
100
200
300
400
0
2000
4000
6000
8000
10000


Kaczmarz SGD - error per iter
0
20
40
60
80
100
120
140
160
0
2
4
6
8

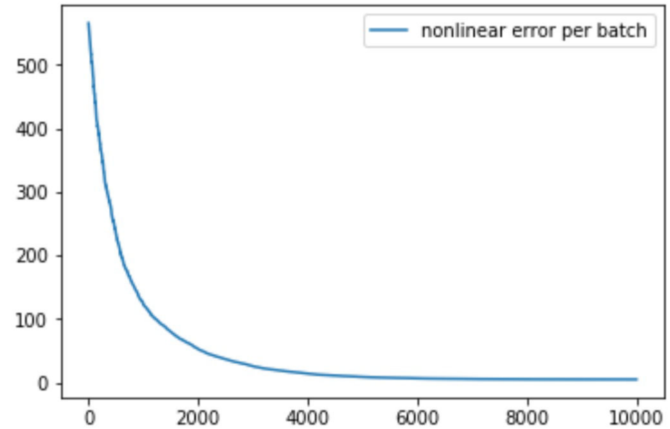

Kaczmarz SGD - error per iter
full least-square - one feature
full gradient descent - one feature
stochastic gradient descent - one feature
full gradient descent - all features
stochastic gradient descent - all features
0
50
100
150
200
250
0
2
4
6
8

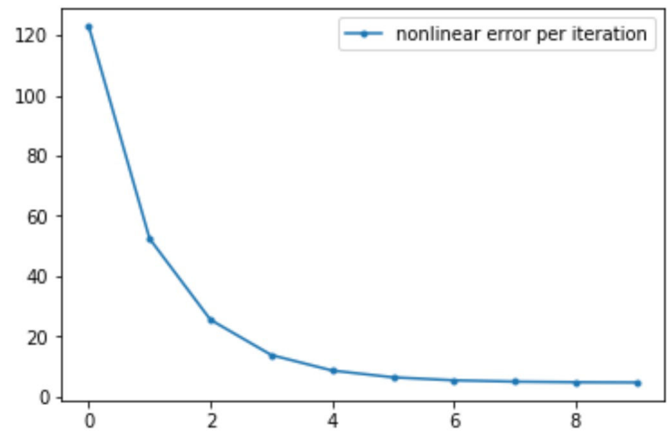
In [96]:
```
# Problem 3.e

y_nonlinear = y + 0.1 * A[:, [1]] ** 3

# Repeat Problem 3.a

alpha = 0.05
w = np.zeros( (d, 1), "float" )
errors_3a_nonlinear = np.zeros( (10, 1), "float" )
for i in range(10):
    grad_loss = 2 * A.T.dot(A).dot(w) - 2 * A.T.dot(y_nonlinear)
    w = w - alpha * grad_loss / (2 * n)
    errors_3a_nonlinear[i] = np.mean((A.dot(w) - y_nonlinear) ** 2)

plt.figure()
plt.plot(errors_3a_nonlinear, ".-", label="nonlinear full gradient descent")
plt.legend(loc="best")
plt.show()
```

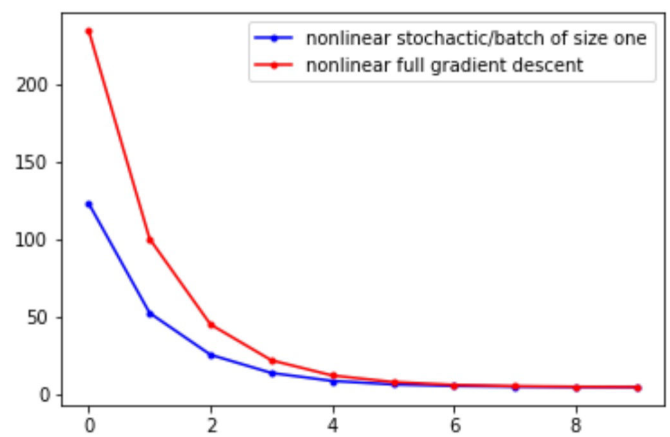


In [98]:

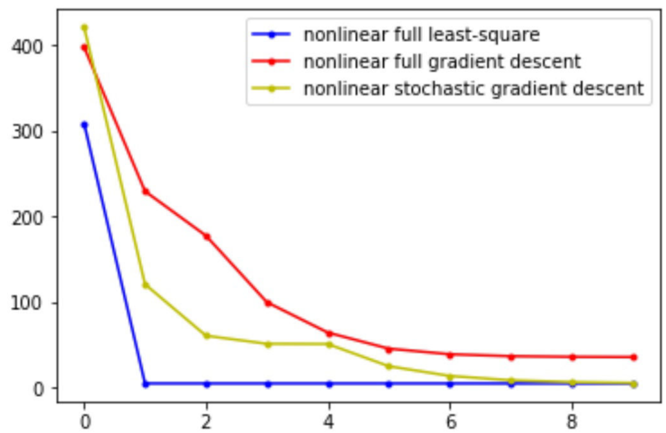
```
# Repeat Problem 3.b

alpha = 0.05
w = np.zeros( (d, 1), "float" )
errors_per_iter_nonlinear = np.zeros( (10, 1), "float" )
errors_per_batch_nonlinear = np.zeros( (10, n), "float" )
for i in range(10):
    for j in range(n):
        grad_loss = 2 * A[[j],:].T.dot(A[[j],:]).dot(w) - 2 * A[[j],:].T.dot(y_n
onlinear[[j]])
        w = w - alpha * grad_loss / (2 * n)
        errors_per_batch_nonlinear[i,j] = np.mean((A.dot(w) - y_nonlinear) ** 2)
    errors_per_iter_nonlinear[i] = np.mean((A.dot(w) - y_nonlinear) ** 2)

plt.figure()
plt.plot(errors_per_batch_nonlinear.flatten(), label="nonlinear error per batch"
)
plt.legend(loc="best")
plt.figure()
plt.plot(errors_per_iter_nonlinear, ".-", label="nonlinear error per iteration")
plt.legend(loc="best")
plt.figure()
plt.plot(errors_per_iter_nonlinear, "b.-", label="nonlinear stochactic/batch of
size one")
plt.plot(errors_3a_nonlinear, "r.-", label="nonlinear full gradient descent")
plt.legend(loc="best")
plt.show()
```


nonlinear error per batch
500
400
300
200
100
0
0
2000
4000
6000
8000
10000

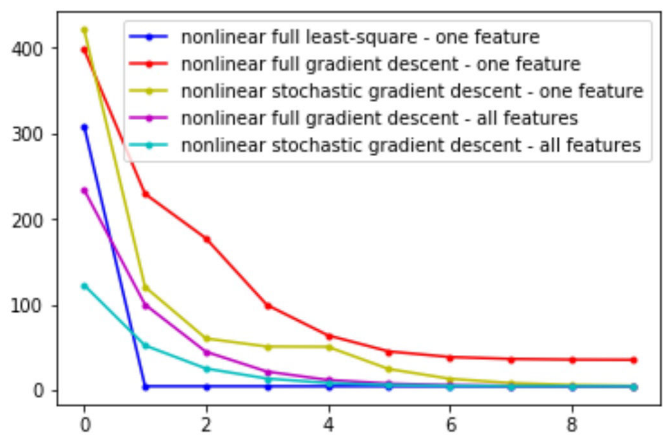

nonlinear error per iteration
120
100
80
60
40
20
0
0
2
4
6
8


nonlinear stochactic/batch of size one
nonlinear full gradient descent
200
150
100
50
0
0
2
4
6
8

In [99]:
```
# Repeat Problem 3.c

alpha = 0.05
w1 = np.zeros( (d, 1), "float" )
w2 = np.zeros( (d, 1), "float" )
w3 = np.zeros( (d, 1), "float" )
errors1_nonlinear = np.zeros( (10, 1), "float" )
errors2_nonlinear = np.zeros( (10, 1), "float" )
errors3_nonlinear = np.zeros( (10, 1), "float" )

for i in range(10):
    r = random.random()
    k = 0 if r < 0.5 else 1
    # full least-square
    w1[k] = npla.inv( A[:,[k]].T.dot(A[:,[k]]) ).dot( A[:,[k]].T.dot(y_nonlinear
) )
    errors1_nonlinear[i] = np.mean((A.dot(w1) - y_nonlinear) ** 2)
    # full gradient descent
    grad_loss = 2 * A.T.dot(A).dot(w2) - 2 * A.T.dot(y_nonlinear)
    w2[k] = w2[k] - alpha * grad_loss[k] / (2 * n)
    errors2_nonlinear[i] = np.mean((A.dot(w2) - y_nonlinear) ** 2)
    # stochastic gradient descent
    for j in range(n):
        grad_loss = 2 * A[[j],:].T.dot(A[[j],:]).dot(w3) - 2 * A[[j],:].T.dot(y_
nonlinear[[j]])
        w3[k] = w3[k] - alpha * grad_loss[k] / (2 * n)
    errors3_nonlinear[i] = np.mean((A.dot(w3) - y_nonlinear) ** 2)

plt.figure()
plt.plot(errors1_nonlinear, "b.-", label="nonlinear full least-square")
plt.plot(errors2_nonlinear, "r.-", label="nonlinear full gradient descent")
plt.plot(errors3_nonlinear, "y.-", label="nonlinear stochastic gradient descent"
)
plt.legend(loc="best")
plt.figure()
plt.plot(errors1_nonlinear, "b.-", label="nonlinear full least-square - one feat
ure")
plt.plot(errors2_nonlinear, "r.-", label="nonlinear full gradient descent - one
feature")
plt.plot(errors3_nonlinear, "y.-", label="nonlinear stochastic gradient descent
- one feature")
plt.plot(errors_3a_nonlinear, "m.-", label="nonlinear full gradient descent - al
l features")
plt.plot(errors_per_iter_nonlinear, "c.-", label="nonlinear stochastic gradient
descent - all features")
plt.legend(loc="best")
plt.show()
```

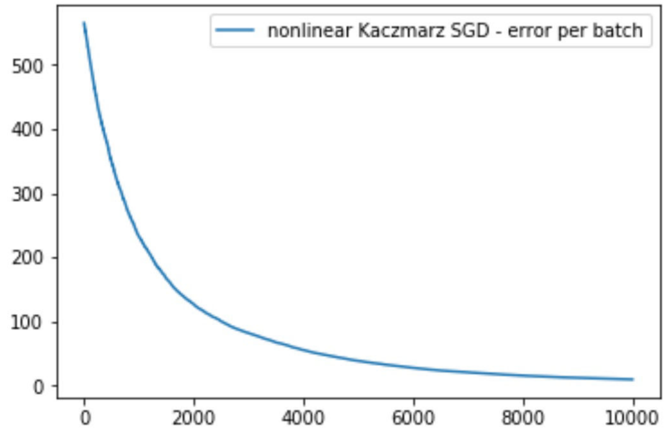


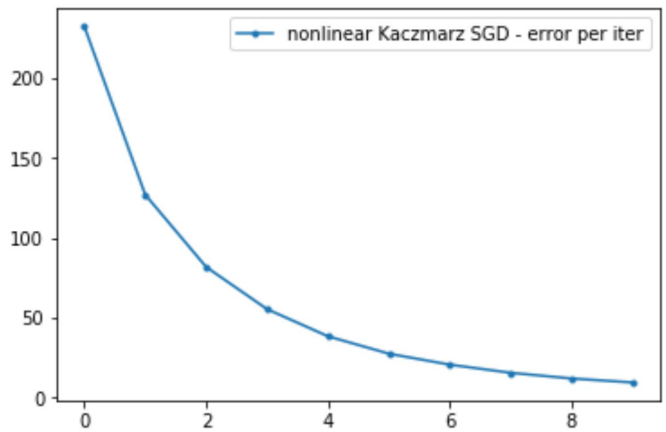
nonlinear full least-square
nonlinear full gradient descent
nonlinear stochastic gradient descent
0
100
200
300
400
0
2
4
6
8

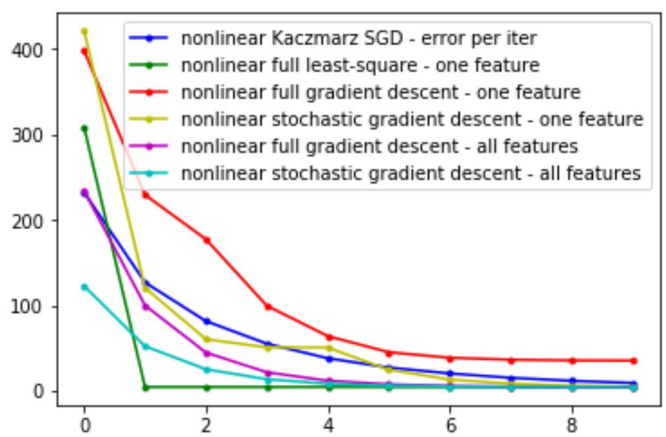


nonlinear full least-square - one feature
nonlinear full gradient descent - one feature
nonlinear stochastic gradient descent - one feature
nonlinear full gradient descent - all features
nonlinear stochastic gradient descent - all features
0
100
200
300
400
0
2
4
6
8

In [100]:
```
# Repeat Problem 3.d

w = np.zeros( (d, 1), "float" )
errors_per_iter_Kaczmarz_nonlinear = np.zeros( (10, 1), "float" )
errors_per_batch_Kaczmarz_nonlinear = np.zeros( (10, n), "float" )

alpha_list = []
prob_list = []
cummulative_prob_list = []
A_Froberius = np.sum(A ** 2)
for j in range(n):
    Aj_Froberius = np.sum( A[j, :] ** 2 )
    alpha_list.append(1 / Aj_Froberius)
    prob_list.append(Aj_Froberius / A_Froberius)
    if j == 0:
        cummulative_prob_list.append(Aj_Froberius / A_Froberius)
    else:
        cummulative_prob_list.append(cummulative_prob_list[j - 1] + Aj_Froberius
/ A_Froberius)

for i in range(10):
    for j in range(n):
        r = random.random()
        index = 0
        while cummulative_prob_list[index] < r:
            index += 1
        grad_loss = 2 * A[[index],:].T.dot(A[[index],:]).dot(w) - 2 * A[[index],
:].T.dot(y_nonlinear[[index]])
        w = w - alpha_list[index] * grad_loss / (2 * n)
        errors_per_batch_Kaczmarz_nonlinear[i,j] = np.mean((A.dot(w) - y_nonline
ar) ** 2)
    errors_per_iter_Kaczmarz_nonlinear[i] = np.mean((A.dot(w) - y_nonlinear) **
2)

plt.figure()
plt.plot(errors_per_batch_Kaczmarz_nonlinear.flatten(), label="nonlinear Kaczmar
z SGD - error per batch")
plt.legend(loc="best")
plt.figure()
plt.plot(errors_per_iter_Kaczmarz_nonlinear, ".-", label="nonlinear Kaczmarz SGD
- error per iter")
plt.legend(loc="best")
plt.figure()
plt.plot(errors_per_iter_Kaczmarz_nonlinear, "b.-", label="nonlinear Kaczmarz SG
D - error per iter")
plt.plot(errors1_nonlinear, "g.-", label="nonlinear full least-square - one feat
ure")
plt.plot(errors2_nonlinear, "r.-", label="nonlinear full gradient descent - one
feature")
plt.plot(errors3_nonlinear, "y.-", label="nonlinear stochastic gradient descent
- one feature")
plt.plot(errors_3a_nonlinear, "m.-", label="nonlinear full gradient descent - al
l features")
plt.plot(errors_per_iter_nonlinear, "c.-", label="nonlinear stochastic gradient
descent - all features")
plt.legend(loc="best")
plt.show()
```


nonlinear Kaczmarz SGD - error per batch
500
400
300
200
100
0
0
2000
4000
6000
8000
10000


nonlinear Kaczmarz SGD - error per iter
200
150
100
50
0
0
2
4
6
8


nonlinear Kaczmarz SGD - error per iter
nonlinear full least-square - one feature
nonlinear full gradient descent - one feature
nonlinear stochastic gradient descent - one feature
nonlinear full gradient descent - all features
nonlinear stochastic gradient descent - all features
400
300
200
100
0
0
2
4
6
8