Midterm Exam

HSE 545: Machine Learning

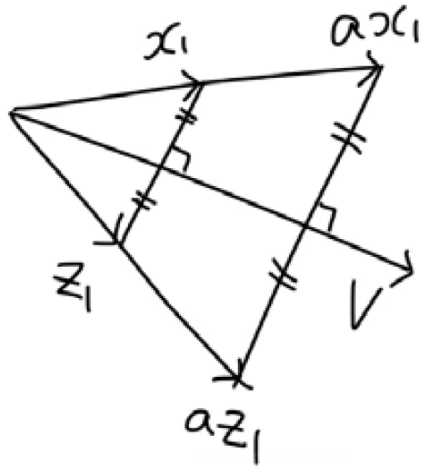
by Seungchul Lee

Problem 1

(a)

Let f be a reflection transformation. To prove that f is linear, we need to show

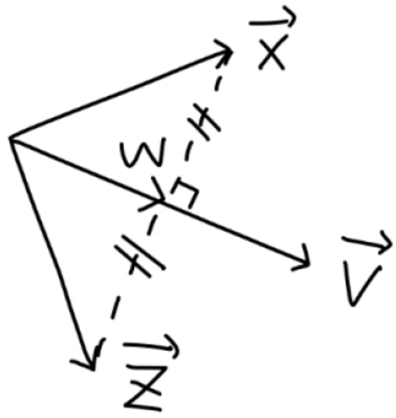
$$f(\alpha x_1) = \alpha\, f(x_1)$$
$$f(x_1 + x_2) = f(x_1) + f(x_2)$$



By the above figure, the linearity is obvious.

(b)

Suppose that ω is a projection vector of x onto v.



$$
\begin{aligned}
\omega &= \frac{v^T x}{v^T v} v = \frac{v v^T}{v^T v} x \\
z &= x - 2(\omega - x) = 2\omega - x \\
&= \left(2 \frac{v v^T}{v^T v} - I\right) x = Mx
\end{aligned}
$$

(c)

From geometirc interpretation, we know that ω and $x - \omega$ are eigenvectors of the reflection transformation and the corresponding eigenvalues are 1 and -1 repectively. Let P be a reflection matrix. Then,

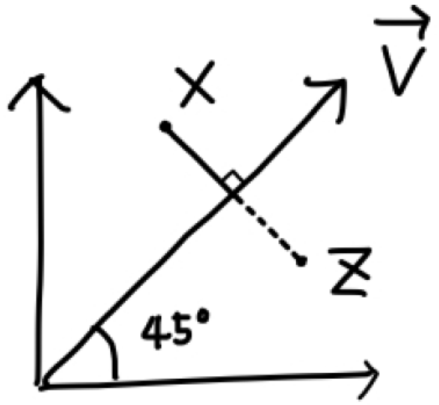
$$
\begin{aligned}
Px &= \lambda_1 \omega + \lambda_2 (x - \omega) \\
&= 1 \cdot \omega + (-1) \cdot (x - \omega) \\
&= 2\omega - x
\end{aligned}
$$

the rest of calculation is the sames as **(b)**.

(d)

Plug $v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ into the result of **(b)**, then it gives

$$M = 2\frac{\begin{bmatrix} 1 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 1 \end{bmatrix}}{\begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix}} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$



$$\text{if } x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \implies Z = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_1 \end{bmatrix}$$

$$Mx = \lambda x \implies (M - \lambda I)\, x = 0$$

$$\det \begin{bmatrix} -\lambda & 1 \\ 1 & -\lambda \end{bmatrix} = \lambda^2 - 1 = 0 \implies \lambda = \pm 1$$

$$\text{For } \lambda_1 = 1, x_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\text{For } \lambda_2 = -1, x_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Problem 2

(a)

A rotation does not change the magnitude of a vector. Hence,

$$\begin{aligned} \|Rx\|_2^2 = \|x\|_2^2 &\implies x^T R^T R x = x^T x,\ \forall x \in \mathbb{R}^n \\ &\implies x^T (R^T R - I) x = 0,\ \forall x \in \mathbb{R}^n \\ &\implies R^T R = I \end{aligned}$$

(b)

From **(a)** we can see that

$$R^T R = I \implies R^{-1} = R^T$$

Also, since an inverse function is unique (or from geometric intuition),

$$R^{-1}(\theta) = R(-\theta)$$

(c)

The result is obvious from **(a)**.

Problem 3

(a)

For an arbitrary vector $\mathbf{x}$, A_1 transforms it as

$$A_1\mathbf{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ x_1 \end{bmatrix}$$

which implies it permutes the elements of $\mathbf{x}$.

(b)

We need to show columns of a permutation matrix A are orthogonal to each other.

Let a_i and a_j be i-th and j-th columns of the A ($i \neq j$). Since 1) positions of one in each columns are different and 2) each column contains a single one,

$$a_i^T a_j = 0.$$

Hence, a permutation matrix is orthogonal.

(c)

By **(b)**, $g(f(x)) \implies A^TA = AA^T = I$. Hence g is an inverse function of f.

(d)

By **(a)**, ones in each row and column of matrix A have the following meanings:

- one in i-th column indicates where the element x_i to go
- one in j-th row indicates where the transformed vector $A\mathbf{x}$'s j-th element comes from.

Therefore, ones in each row and column of the transposed matrix A^T have the reversed meanings:

- one in j-th column indicates where the transformed vector $A\mathbf{x}$'s j-th element comes from.
- one in i-th row indicates where the element x_i to go.

which is equivalent explanation of inverse transformation.

Problem 4

(a)

Sum of 2-norms of multiple vectors can be expressed as a 2-norm of a single vector.

$$\left\|A\theta - y\right\|_2^2 + \left\|\sqrt{\lambda} I\theta - 0\right\|_2^2 = \left\| \begin{bmatrix} A \\ \sqrt{\lambda} I \end{bmatrix} \theta - \begin{bmatrix} y \\ 0 \end{bmatrix} \right\|_2^2$$

$$\hat{\theta} = \left(\begin{bmatrix} A \\ \sqrt{\lambda} I \end{bmatrix}^T \begin{bmatrix} A \\ \sqrt{\lambda} I \end{bmatrix} \right)^{-1} \begin{bmatrix} A \\ \sqrt{\lambda} I \end{bmatrix}^T \begin{bmatrix} y \\ 0 \end{bmatrix} = \left(A^T A + \lambda I_n\right)^{-1} A^T y$$

(b)

Let $J(\theta) = \|A\theta - y\|_2^2 + \lambda\|\theta\|_2^2$. A gradient descent algorithm is formulated as the following,

$$\theta \leftarrow \theta - \eta \frac{\partial J}{\partial \theta}.$$

Need to compute $\frac{\partial J}{\partial \theta}$.

$$\begin{aligned} g_{\text{projection}} &= \frac{\partial}{\partial \theta} \|A\theta - y\|_2^2 \\ &= \frac{\partial}{\partial \theta} (A\theta - y)^T (A\theta - y) \\ &= \frac{\partial}{\partial \theta} (\theta^T A^T A\theta - 2\theta^T A^T y - y^T y) \\ &= 2A^T A\theta - 2A^T y \end{aligned}$$

$$\begin{aligned} g_{\text{regularizer}} &= \frac{\partial}{\partial \theta} \|\theta\|_2^2 \\ &= \frac{\partial}{\partial \theta} \theta^T \theta \\ &= 2\theta. \end{aligned}$$

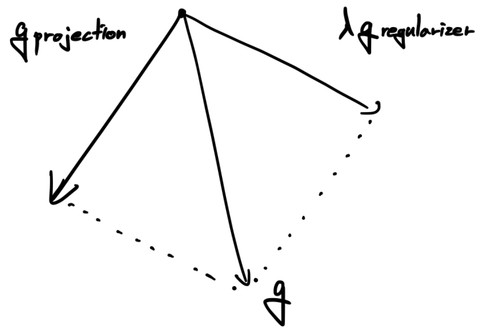
Therefore,

$$\begin{aligned} \frac{\partial J}{\partial \theta} &= g_{\text{projection}} + \lambda g_{\text{regularizer}} \\ &= 2A^T A\theta - 2A^T y + \lambda 2\theta. \end{aligned}$$

Then, the gradient descent algorithm is formulated as the following

$$\theta \leftarrow \theta - \eta(2A^T A\theta - 2A^T y + \lambda 2\theta).$$

(c)



$g_{\text{regularizer}}$ make θ converge to zero. Since $g = g_{\text{projection}} + \lambda g_{\text{regularizer}}$, we can see that regularizer underestimates the value of the projection.

(d)

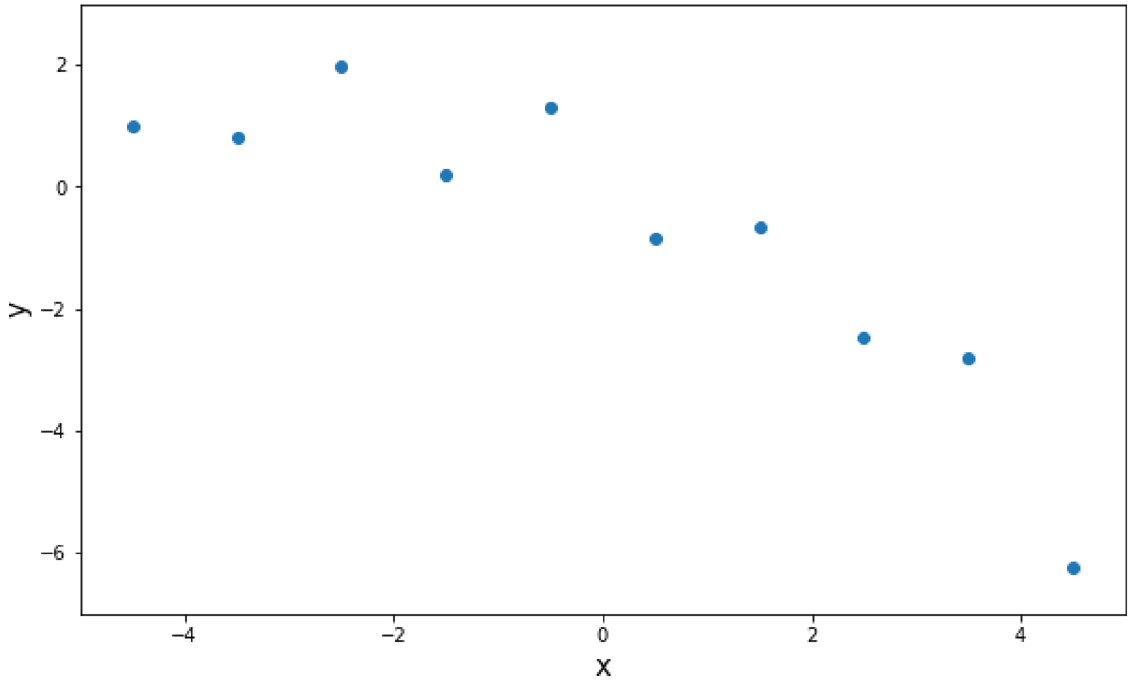
Likewise, **(a)** shows that the regularizer underestimates the value of projection by

$$(A^T A)^{-1} A^T y \to (A^T A + \lambda I_n)^{-1} A^T y.$$

(e)

In [1]:

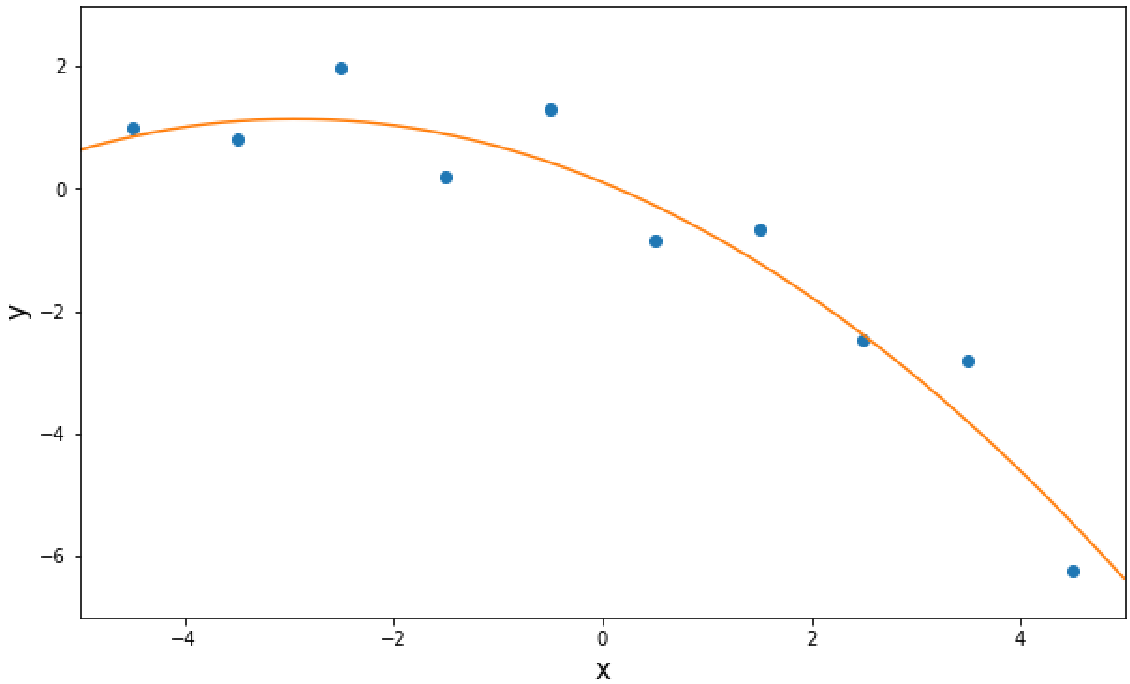
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-4.5, 4.5, 10)
y = np.array([0.9819, 0.7973, 1.9737, 0.1838, 1.3180, -0.8361, -0.6591, -2.4701, -2.8122, -6.251
2])

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'o')
plt.xlabel('x', fontsize=15)
plt.ylabel('y', fontsize=15)
plt.xlim(-5, 5)
plt.ylim(-7, 3)
plt.show()
```



In [2]:

```
x = np.array(x).reshape(-1, 1)
y = np.array(y).reshape(-1, 1)
A = np.hstack([x**0, x, x**2])
A = np.asmatrix(A)
theta = np.zeros([3, 1])
alpha = 0.000001
lamb = 1

for i in range(100000):
    g = A.T*A*theta - A.T*y + lamb*theta
    theta = theta - alpha*g

print(theta)

xp = np.arange(-5,5,0.01).reshape(-1,1)
yp = theta[0,0] + theta[1,0]*xp + theta[2,0]*xp**2

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'o')
plt.plot(xp[:,0], yp[:,0])
plt.xlabel('x', fontsize=15)
plt.ylabel('y', fontsize=15)
plt.xlim(-5, 5)
plt.ylim(-7, 3)
plt.show()
```

```
[[ 0.10833274]
 [-0.70202688]
 [-0.1193053 ]]
```



Problem 5

(a)

Let's denote $\mathbf{x}_{-j}$ the vector $\mathbf{x}$ without the j-th element x_j. $P(y \mid \mathbf{x}) = P(y \mid \mathbf{x}_{-j})$ indicates y is independent of x_j. Hence x_j is a useless feature.

(b)

Without loss of generality, let's assume j-th value of the sparse vector ω is zero. Then,

$$\begin{aligned} P(y \mid \mathbf{x}) &= f(x; \omega) \\ &= f(x; \omega_{-j}) \\ &= P(y \mid \mathbf{x}_{-j}) \end{aligned}$$

Therefore, Lasso selects meaningful features.

(c)

In [4]:

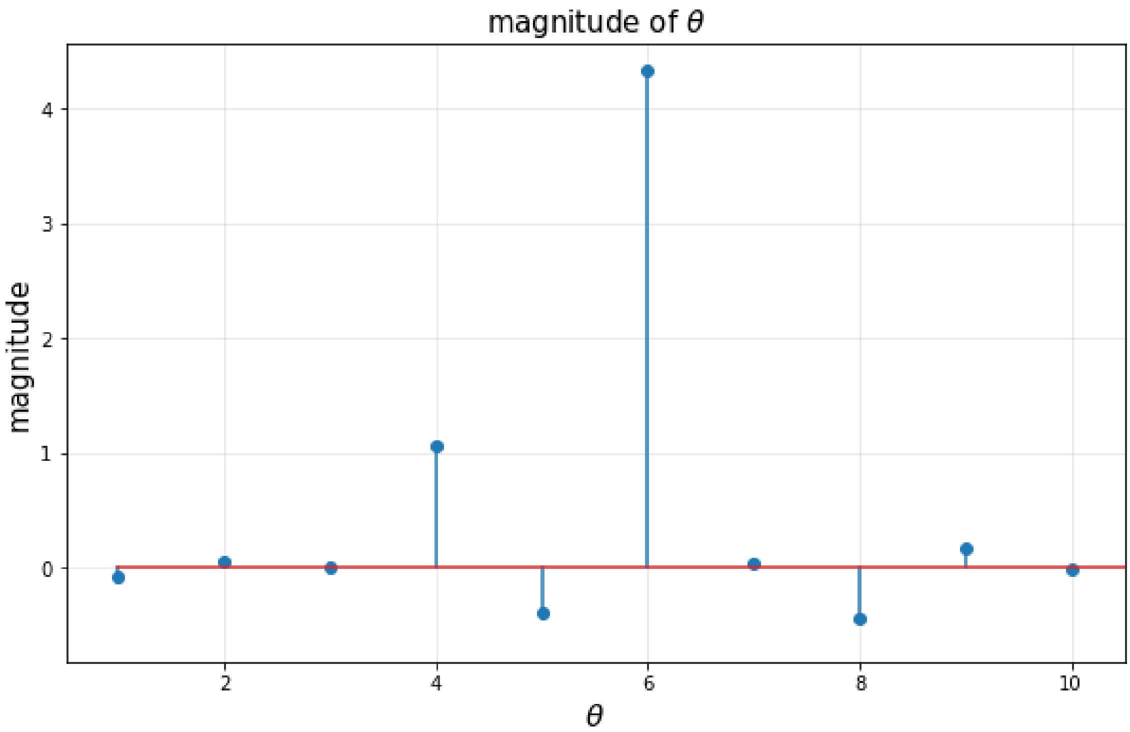
```
from six.moves import cPickle
x = cPickle.load(open('./data/data_input.pkl', 'rb'))
y = cPickle.load(open('./data/data_target.pkl', 'rb'))
```

In [5]:

```
# Write your code here
import cvxpy as cvx

y = y.reshape(506,1)
n_data = 506
d = 13

lamb = 2
theta = cvx.Variable(d, 1)
obj = cvx.Minimize(cvx.sum_squares(x*theta - y) + lamb*cvx.norm(theta, 1))
prob = cvx.Problem(obj).solve('SCS')
```

In [6]:

```
# Regulization ( = ridge nonlinear regression) encourages small weights, but not exactly 0
plt.figure(figsize=(10, 6))
plt.title(r'magnitude of $₩theta$', fontsize=15)
plt.xlabel(r'$₩theta$', fontsize=15)
plt.ylabel('magnitude', fontsize=15)
plt.stem(np.linspace(1, 13, 13).reshape(-1, 1), theta.value)
plt.xlim([0.5, 10.5])
plt.grid(alpha=0.3)
plt.show()
```



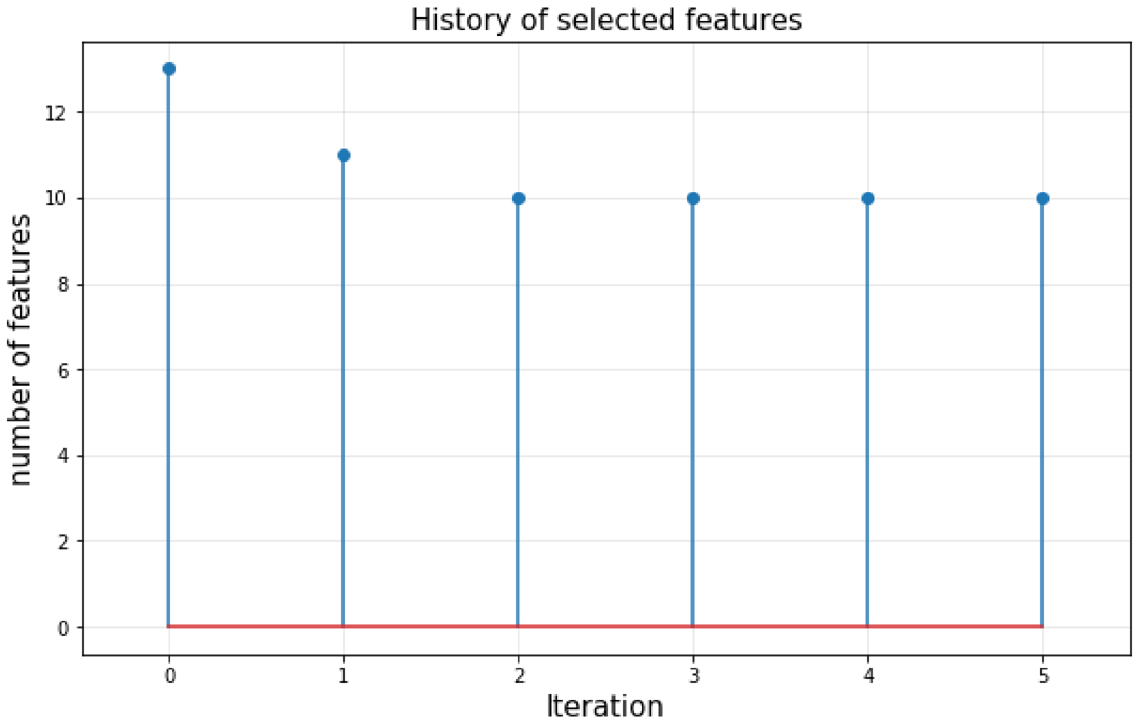
(d)

In [7]:

```
#Write your code here
d = 13
features = [13]
elim_idx = [False]*13
elim_num = 0
lamb = 2
x_selected = x.copy()

for i in range(5):
    theta = cvx.Variable(d, 1)
    obj = cvx.Minimize(cvx.sum_squares(x_selected*theta - y) + lamb*cvx.norm(theta, 1))
    prob = cvx.Problem(obj).solve('SCS')

    for j in range(d):
        if abs(theta.value[j]) < 0.01:
            if elim_idx[j] == False:
                elim_idx[j] = True
                elim_num += 1
                x_selected[:,j] = np.zeros(x_selected.shape[0])
    features.append(d - elim_num)
```

In [8]:

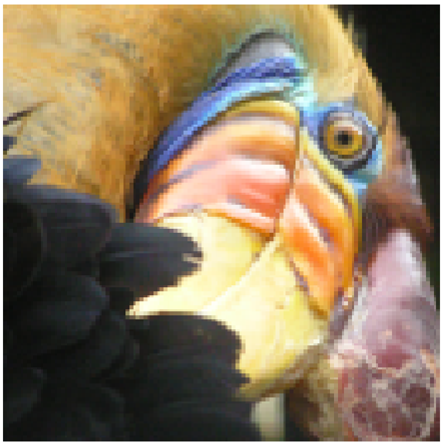
```
plt.figure(figsize=(10, 6))
plt.title(r'History of selected features', fontsize=15)
plt.xlabel('Iteration', fontsize=15)
plt.ylabel('number of features', fontsize=15)
plt.stem(np.linspace(0, 5, 6).reshape(-1, 1), features)
plt.xlim([-0.5,5.5])
plt.grid(alpha=0.3)
plt.show()
```



Problem6

In [9]:

```
from six.moves import cPickle
import matplotlib.pyplot as plt

A = cPickle.load(open('./data/bird.pkl', 'rb'))

plt.figure(figsize=(4, 4))
plt.imshow(A.astype('uint8'))
plt.axis('off')
plt.show()

print('Matrix shape: {}'.format(A.shape))
```



```
Matrix shape: (128, 128, 3)
```

In [10]:

```
# Write down your code here

k = 16
A_re = A.reshape(128*128,3)
centroid = np.array([np.linspace(5, 255, k), np.linspace(5, 255, k), np.linspace(5, 255, k)]).T
mu = centroid.copy()

y = np.empty([128*128, 1])
d = np.zeros([k,1])

for n_iter in range(10):
    for i in range(128*128):
        for j in range(k):
            d[j] = np.linalg.norm(A_re[i,:] - mu[j,:], 2)
        y[i] = np.argmin(d)

    err = 0
    for i in range(k):
        mu[i,:] = np.mean(A_re[np.where(y == i)[0]], axis=0)
        err += np.linalg.norm(centroid[i,:] - mu[i,:],2)

    centroid = mu.copy()
    print("Iteration : {0}/{1}, err : {2}".format(n_iter+1, 10, err))
    if err < 1e-10:
        print("Iteration:", n_iter)
        break

img = np.zeros([128*128, 3])

for i in range(16):
    img[np.where(y[:,0] == i)] = centroid[i]
```

```
Iteration : 1/10, err : 502.83625127080626
Iteration : 2/10, err : 130.27782022936955
Iteration : 3/10, err : 167.98678545461357
Iteration : 4/10, err : 64.65594727843731
Iteration : 5/10, err : 39.90913727379131
Iteration : 6/10, err : 39.08137279763004
Iteration : 7/10, err : 39.40752526191003
Iteration : 8/10, err : 40.014691012938016
Iteration : 9/10, err : 37.137936083585046
Iteration : 10/10, err : 28.949098937074094
```

In [12]:

```
plt.figure(figsize=(4, 4))

plt.title('After K-means')
plt.imshow(img.reshape(128,128,3).astype('uint8'))
plt.axis('off')
plt.show()
```

After K-means