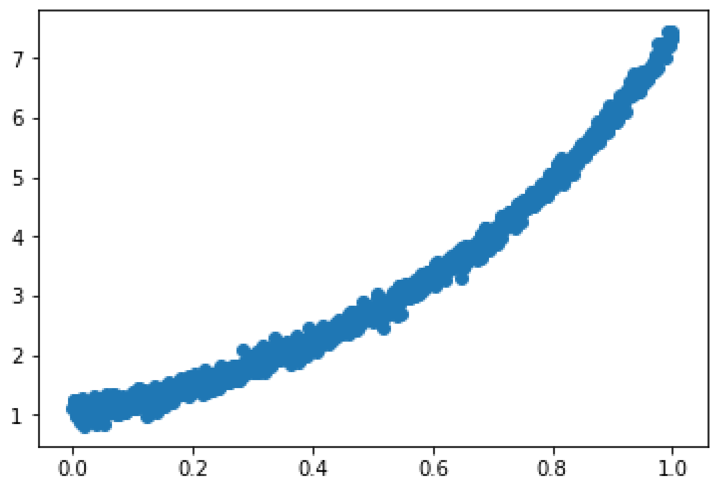
In [1]:

```python
# import basic packages
import scipy.io
import matplotlib.pyplot as plt
import numpy as np
import cv2, os, random
```

No. 2 - 1

In [2]:

```python
# 第二題之一 ： 將 mat 檔中的 x, y Plot 出來
# read mat files and plot
mat = scipy.io.loadmat('data.mat')
x, y = mat['x'].reshape(-1), mat['y'].reshape(-1)

# find maximum and minimum value of x and y
max_x = np.max(x)
min_x = np.min(x)
max_y = np.max(y)
min_y = np.min(y)

# plot and show
plt.scatter(mat['x'], mat['y'])
plt.show()
```



No. 2 - 2

In [3]:

```python
# 第二題之二：算出 y = θ₀ + θ₁x
def linear_regression(x,y):
    x = np.concatenate((np.ones((x.shape[0],1)),x[:,np.newaxis]),axis=1)
    y = y[:,np.newaxis]
    beta = np.matmul(np.matmul(np.linalg.inv(np.matmul(x.T,x)),x.T),y)
    return beta
```

In [4]:

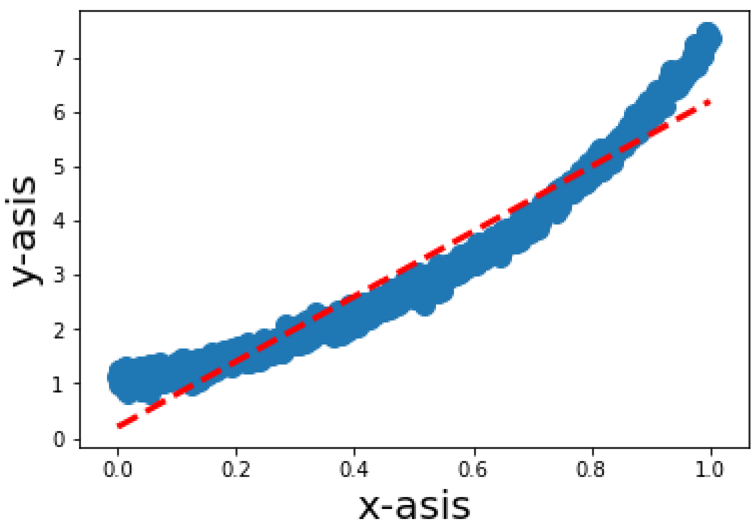
```python
# 將x,y帶入
parameters_line =linear_regression(x,y)
```

In [5]:

```
# 任意建立新的點
x_data = np.linspace(0,max_x,1000)
y_predict = parameters_line[0] + parameters_line[1]*x_data

# 將原本的資料與預測的 f(x) 畫出來
plt.scatter(x,y,s=100)
plt.plot(x_data,y_predict,'r--',linewidth = 3)
plt.xlabel('x-asis',fontsize=20)
plt.ylabel('y-asis',fontsize=20)
```

Out[5]:

```
Text(0, 0.5, 'y-asis')
```



No. 3

In [6]:

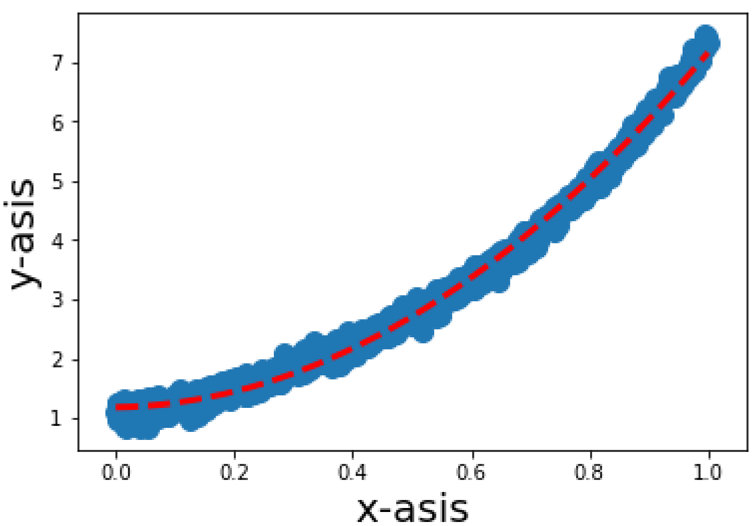
```
# 第三題：如上面，將一次線性式改為多項式 ( second order polynomial  )
# 第二題之二：算出 y = θ₀ + θ₁x + θ₂x²
def second_order_polynomial(x,y):
    x_tmp = np.concatenate((np.ones((x.shape[0],1)),x[:,np.newaxis]),axi
    x = np.concatenate((x_tmp,x[:,np.newaxis]**2),axis=1)
    y = y[:,np.newaxis]

    print("X is : \n{}".format(x))
    print("Y is : \n{}".format(y))
    beta = np.matmul(np.matmul(np.linalg.inv(np.matmul(x.T,x)),x.T),y)
    return beta
```

In [7]:
```
parameters_polynomial = second_order_polynomial(x,y)
```

```
X is :
[[1.00000e+00 0.00000e+00 0.00000e+00]
 [1.00000e+00 1.00000e-03 1.00000e-06]
 [1.00000e+00 2.00000e-03 4.00000e-06]
 ...
 [1.00000e+00 9.98000e-01 9.96004e-01]
 [1.00000e+00 9.99000e-01 9.98001e-01]
 [1.00000e+00 1.00000e+00 1.00000e+00]]
Y is :
[[1.11889485]
 [1.24080293]
 [1.2305603 ]
 ...
 [7.44260211]
 [7.36491736]
 [7.34314766]]
```

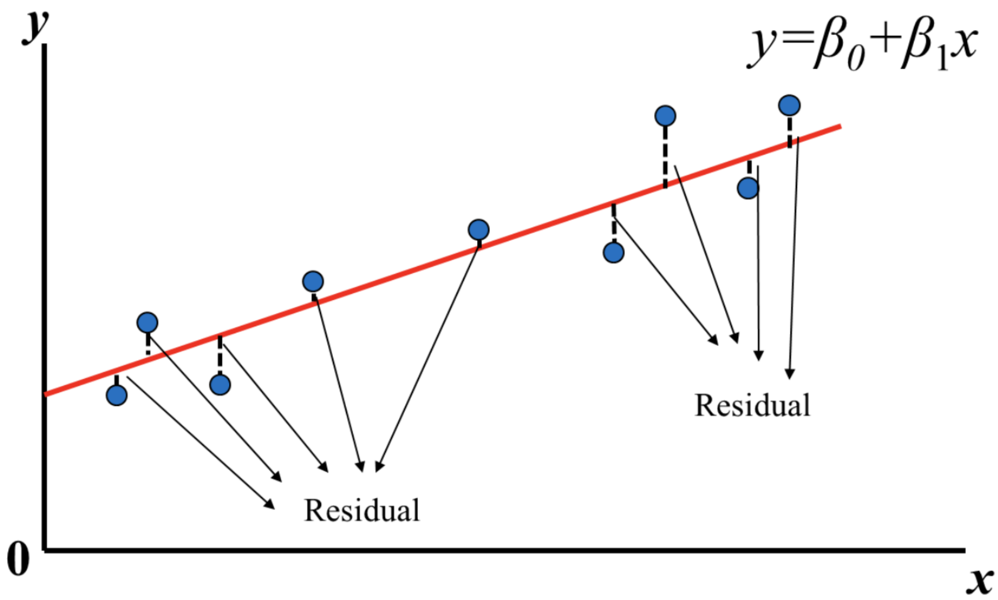
In [8]:
```
# 任意建立新的點
x_data = np.linspace(0,max_x,1000)
y_predict = parameters_polynomial[0] + parameters_polynomial[1]*x_data +

# 將原本的資料與預測的 f(x) 畫出來
plt.scatter(x,y,s=100)
plt.plot(x_data,y_predict,'r--',linewidth = 3)
plt.xlabel('x-asis',fontsize=20)
plt.ylabel('y-asis',fontsize=20)
```

Out[8]:
```
Text(0, 0.5, 'y-asis')
```



第二題及第三題作法說明及討論：[介紹 (https://medium.com/@chih.sheng.huang821/%E7%B7%9A% linear-regression-3a271a7453e)](https://medium.com/@chih.sheng.huang821/%E7%B7%9A%linear-regression-3a271a7453e)

首先以下圖來說：



我們可以用兩個角度切入：

- 線性代數角度
  - 以線性角度出發，我們將每個點（每筆資料）都設回一向量(vector)
  - 找到一個向量（線性方程或多項方程）使得各筆資料的投影（投影在該向量上）與其（各筆資料）距離越短越好
- 垂直 (y,y*) 之間的距離最小，如圖之紅線
  - 但若資料為多維度，則不易用「 二維平面 」看出來
  - 顯而易見，我們希望找到一條線（線性或是多項式的線）來代表整體資料，亦即這條線與數筆資料的直線距離越短越好
  - First Step, we make（考慮兩個變數：θ_0、θ_1） ：

$$\text{Loss}(\hat{\beta}_0, \hat{\beta}_1) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}\left(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)\right)^2$$

  - 接著利用微分概念得出：

> *為了推估β0，對Loss(β0, β1)做β0偏微分等於0*

$$
\begin{aligned}
&\frac{\partial \mathrm{Loss}(\hat{\beta}_0,\hat{\beta}_1)}{\partial \beta_0} = \frac{\partial \sum_{i=1}^{n}(y_i-\hat{\beta}_0-\hat{\beta}_1 x_i)^2}{\partial \beta_0} = 0 \\
&\Rightarrow -2\sum_{i=1}^{n}(y_i-\hat{\beta}_0-\hat{\beta}_1 x_i) = 0 \\
&\Rightarrow \hat{\beta}_0 = \frac{1}{n}\sum_{i=1}^{n}\left(y_i-\hat{\beta}_1 x_i\right) \\
&\Rightarrow \hat{\beta}_0 = \bar{y}-\hat{\beta}_1\bar{x}
\end{aligned}
$$

> *為了推估β1，對Loss(β0, β1)做β1偏微分等於0*

$$
\begin{aligned}
&\frac{\partial \mathrm{Loss}(\hat{\beta}_0,\hat{\beta}_1)}{\partial \beta_1} = \frac{\partial \sum_{i=1}^{n}(y_i-\hat{\beta}_0-\hat{\beta}_1 x_i)^2}{\partial \beta_1} = 0 \\
&\Rightarrow -2\sum_{i=1}^{n}(y_i-\hat{\beta}_0-\hat{\beta}_1 x_i)x_i = 0 \\
&\Rightarrow \sum_{i=1}^{n}(y_i-\hat{\beta}_0-\hat{\beta}_1 x_i)x_i = 0 \\
&\Rightarrow \sum_{i=1}^{n}y_i x_i - \sum_{i=1}^{n}\hat{\beta}_1 x_i^2 - \sum_{i=1}^{n}\hat{\beta}_0 x_i = 0 \\
&\Rightarrow \sum_{i=1}^{n}y_i x_i - \sum_{i=1}^{n}\left(\bar{y}-\hat{\beta}_1\bar{x}\right)x_i - \hat{\beta}_1\sum_{i=1}^{n}x_i^2 = 0 \\
&\Rightarrow \sum_{i=1}^{n}y_i x_i - \sum_{i=1}^{n}\bar{y}x_i + \sum_{i=1}^{n}\hat{\beta}_1\bar{x}x_i - \hat{\beta}_1\sum_{i=1}^{n}x_i^2 = 0 \\
&\Rightarrow \hat{\beta}_1\sum_{i=1}^{n}(x_i-\bar{x})x_i = \sum_{i=1}^{n}(y_i-\bar{y})x_i \\
&\Rightarrow \hat{\beta}_1 = \frac{\sum_{i=1}^{n}(y_i-\bar{y})(x_i-\bar{x})}{\sum_{i=1}^{n}(x_i-\bar{x})^2}
\end{aligned}
$$

而這兩題我皆以 「 *線性代數角度* 」 ，如講義所說： （如下兩圖）

Linear One

$$Loss\ function: f(\boldsymbol{\theta}) = \sum_{i=0}^{3}[y_i - \underbrace{(\boldsymbol{\theta_0} + x_i\boldsymbol{\theta_1})}_{\tilde{y}}]^2$$

$$\boldsymbol{X} = \begin{pmatrix} 1\ x_0 \\ 1\ x_1 \\ 1\ x_2 \\ 1\ x_3 \end{pmatrix} \quad \boldsymbol{y} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

$$f(\boldsymbol{\theta}) = \|\boldsymbol{X\theta} - \boldsymbol{y}\|^2 = (\boldsymbol{X\theta} - \boldsymbol{y})^{\boldsymbol{T}}(\boldsymbol{X\theta} - \boldsymbol{y})$$

$$\boldsymbol{\theta}^* = \mathrm{argmin}_\theta\ f(\boldsymbol{\theta}) = \mathrm{argmin}_\theta\ \|\boldsymbol{X\theta} - \boldsymbol{y}\|^2$$

Polynomial One

$$Loss\ function: f(\boldsymbol{\theta}) = \sum_{i=0}^{3}[y_i - \sum_{j=0}^{d}\theta_j \boldsymbol{x}_i^{\boldsymbol{j}}]^2$$

$$\boldsymbol{X} = \begin{pmatrix} 1\ x_0\ x_0^2 & & x_0^d \\ 1\ x_1\ x_1^2 & \dots & x_1^d \\ 1\ x_2\ x_2^2 & & x_2^d \\ 1\ x_3\ x_3^2 & & x_3^d \end{pmatrix} \qquad \boldsymbol{\theta} = \begin{pmatrix} \boldsymbol{\theta_0} \\ \boldsymbol{\theta_1} \\ \dots \\ \boldsymbol{\theta_d} \end{pmatrix}$$

大致推導步驟：

1. 首先我們一樣衡量 `loss function` ：

Loss function 定義為

$$\begin{aligned} Loss(\boldsymbol{\beta}) &= (\mathbf{Y}-\hat{\mathbf{Y}})^T(\mathbf{Y}-\hat{\mathbf{Y}}) \\ &= (\mathbf{Y}-\hat{\boldsymbol{\beta}}^T\mathbf{X})^T(\mathbf{Y}-\hat{\boldsymbol{\beta}}^T\mathbf{X}) \\ &= \mathbf{Y}^T\mathbf{Y}+\mathbf{X}^T\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T\mathbf{X}-2\mathbf{X}^T\hat{\boldsymbol{\beta}}\mathbf{Y} \end{aligned}$$

2. 接著，我們利用對 β 微分，找出相對應係數：

$$\begin{aligned} \frac{\partial Loss(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}} &= \partial\frac{\mathbf{Y}^T\mathbf{Y}+\mathbf{X}^T\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T\mathbf{X}-2\mathbf{X}^T\hat{\boldsymbol{\beta}}\mathbf{Y}}{\partial\boldsymbol{\beta}} = 0 \\ &\Rightarrow 2\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}}-2\mathbf{X}^T\mathbf{Y}=0 \\ &\Rightarrow \mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}}=\mathbf{X}^T\mathbf{Y} \\ &\Rightarrow \hat{\boldsymbol{\beta}}=(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \end{aligned}$$

3. 最後找到：

$$\Rightarrow \hat{\boldsymbol{\beta}}=(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

4. 而回到最初 $y = \theta_0 + \theta_1 x + \theta_2 x^2$，我們可以將 β 和 x 帶入，以矩陣、向量寫的話，怎可以表達為：

$$y_i = \boldsymbol{\beta}^T\mathbf{x}_i = \begin{bmatrix}\beta_0\\ \beta_1\\ \vdots\\ \beta_d\end{bmatrix}^T \begin{bmatrix}1\\ x_1^{(i)}\\ \vdots\\ x_d^{(i)}\end{bmatrix} = \beta_0+\beta_1 x_1^{(i)}+\ldots+\beta_d x_d^{(i)}$$

No. 4 - 1

In [9]:

```
# 第四題：處理 MNIST 資料集
from __future__ import print_function
import keras
from keras.datasets import mnist

# input image dimensions 28x28
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()


x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

```
Using TensorFlow backend.
```

In [10]:

```
# get some info of the data
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

No. 4 - 2

In [11]:

```
# save imagesfrom either the training or the testing dataset to form yo
# save those digit images's file with the name using that exact number
# create data folder
data_root = "data"
data_train = "train"
data_test = "test"
train_sample = np.zeros((10,100,28,28))
train_sample_answer = np.zeros((10,100))
test_sample = np.zeros((10,100,28,28))
test_sample_answer = np.zeros((10,100))

if not os.path.exists(data_root):
    os.makedirs(data_root)
if not os.path.exists(os.path.join(data_root, data_train)):
    os.makedirs(os.path.join(data_root, data_train))
if not os.path.exists(os.path.join(data_root, data_test)):
    os.makedirs(os.path.join(data_root, data_test))

# renew the count_dict
count_dict = {
    "0" : 0,
    "1" : 0,
    "2" : 0,
    "3" : 0,
    "4" : 0,
    "5" : 0,
    "6" : 0,
    "7" : 0,
    "8" : 0,
    "9" : 0
}

print("Saving 100 samples images for each digit in testing data.... ")

# save for train images
for test_index in range(x_test.shape[0]) :
    if y_test[test_index] == 1 and count_dict["1"] < 99:
        if not os.path.exists(os.path.join(data_root, data_test,"1")):
            os.makedirs(os.path.join(data_root, data_test,"1"))
        x_test[test_index] = x_test[test_index]*255
        test_sample[0][count_dict["1"]] = x_test[test_index].astype(np
        test_sample_answer[0][count_dict["1"]] = 1
        cv2.imwrite(os.path.join(data_root, data_test,"1/",str(count_di
        count_dict["1"] += 1
    elif y_test[test_index] == 2 and count_dict["2"] < 99:
        if not os.path.exists(os.path.join(data_root, data_test,"2")):
            os.makedirs(os.path.join(data_root, data_test,"2"))
        x_test[test_index] = x_test[test_index]*255
        test_sample[1][count_dict["2"]] = x_test[test_index].astype(np
        test_sample_answer[1][count_dict["2"]] = 2
        cv2.imwrite(os.path.join(data_root, data_test,"2/",str(count_di
        count_dict["2"] += 1
    elif y_test[test_index] == 3 and count_dict["3"] < 99:
        if not os.path.exists(os.path.join(data_root, data_test,"3")):
            os.makedirs(os.path.join(data_root, data_test,"3"))
        x_test[test_index] = x_test[test_index]*255
```

```
            test_sample[2][count_dict["3"]] = x_test[test_index].astype(np
            test_sample_answer[2][count_dict["3"]] = 3
            cv2.imwrite(os.path.join(data_root, data_test,"3/",str(count_d
            count_dict["3"] += 1
        elif y_test[test_index] == 4 and count_dict["4"] < 99:
            if not os.path.exists(os.path.join(data_root, data_test,"4")):
                os.makedirs(os.path.join(data_root, data_test,"4"))
            x_test[test_index] = x_test[test_index]*255
            test_sample[3][count_dict["4"]] = x_test[test_index].astype(np
            test_sample_answer[3][count_dict["4"]] = 4
            cv2.imwrite(os.path.join(data_root, data_test,"4/",str(count_d
            count_dict["4"] += 1
        elif y_test[test_index] == 5 and count_dict["5"] < 99:
            if not os.path.exists(os.path.join(data_root, data_test,"5")):
                os.makedirs(os.path.join(data_root, data_test,"5"))
            x_test[test_index] = x_test[test_index]*255
            test_sample[4][count_dict["5"]] = x_test[test_index].astype(np
            test_sample_answer[4][count_dict["5"]] = 5
            cv2.imwrite(os.path.join(data_root, data_test,"5/",str(count_d
            count_dict["5"] += 1
        elif y_test[test_index] == 6 and count_dict["6"] < 99:
            if not os.path.exists(os.path.join(data_root, data_test,"6")):
                os.makedirs(os.path.join(data_root, data_test,"6"))
            x_test[test_index] = x_test[test_index]*255
            test_sample[5][count_dict["6"]] = x_test[test_index].astype(np
            test_sample_answer[5][count_dict["6"]] = 6
            cv2.imwrite(os.path.join(data_root, data_test,"6/",str(count_d
            count_dict["6"] += 1
        elif y_test[test_index] == 7 and count_dict["7"] < 99:
            if not os.path.exists(os.path.join(data_root, data_test,"7")):
                os.makedirs(os.path.join(data_root, data_test,"7"))
            x_test[test_index] = x_test[test_index]*255
            test_sample[6][count_dict["7"]] = x_test[test_index].astype(np
            test_sample_answer[6][count_dict["7"]] = 7
            cv2.imwrite(os.path.join(data_root, data_test,"7/",str(count_d
            count_dict["7"] += 1
        elif y_test[test_index] == 8 and count_dict["8"] < 99:
            if not os.path.exists(os.path.join(data_root, data_test,"8")):
                os.makedirs(os.path.join(data_root, data_test,"8"))
            x_test[test_index] = x_test[test_index]*255
            test_sample[7][count_dict["8"]] = x_test[test_index].astype(np
            test_sample_answer[7][count_dict["8"]] = 8
            cv2.imwrite(os.path.join(data_root, data_test,"8/",str(count_d
            count_dict["8"] += 1
        elif y_test[test_index] == 9 and count_dict["9"] < 99:
            if not os.path.exists(os.path.join(data_root, data_test,"9")):
                os.makedirs(os.path.join(data_root, data_test,"9"))
            x_test[test_index] = x_test[test_index]*255
            test_sample[8][count_dict["9"]] = x_test[test_index].astype(np
            test_sample_answer[8][count_dict["9"]] = 9
            cv2.imwrite(os.path.join(data_root, data_test,"9/",str(count_d
            count_dict["9"] += 1
        elif y_test[test_index] == 0 and count_dict["0"] < 99:
            if not os.path.exists(os.path.join(data_root, data_test,"0")):
                os.makedirs(os.path.join(data_root, data_test,"0"))
            x_test[test_index] = x_test[test_index]*255
            test_sample[9][count_dict["0"]] = x_test[test_index].astype(np
```

```
        test_sample_answer[9][count_dict["0"]] = 0
        cv2.imwrite(os.path.join(data_root, data_test,"0/",str(count_d
        count_dict["0"] += 1
    else :
        pass

# =====================================================================

count_dict = {
    "0" : 0,
    "1" : 0,
    "2" : 0,
    "3" : 0,
    "4" : 0,
    "5" : 0,
    "6" : 0,
    "7" : 0,
    "8" : 0,
    "9" : 0
}

print("Saving 100 samples images for each digit in training data.... "

# save for train images
for train_index in range(x_train.shape[0]) :

    if y_train[train_index] == 1 and count_dict["1"] < 99:
        if not os.path.exists(os.path.join(data_root, data_train,"1"))
            os.makedirs(os.path.join(data_root, data_train,"1"))
        x_train[train_index] = x_train[train_index]*255
        train_sample[0][count_dict["1"]] = x_train[train_index].astype
        train_sample_answer[0][count_dict["1"]] = 1
        cv2.imwrite(os.path.join(data_root, data_train,"1/",str(count_
        count_dict["1"] += 1
    elif y_train[train_index] == 2 and count_dict["2"] < 99:
        if not os.path.exists(os.path.join(data_root, data_train,"2"))
            os.makedirs(os.path.join(data_root, data_train,"2"))
        x_train[train_index] = x_train[train_index]*255
        train_sample[1][count_dict["2"]] = x_train[train_index].astype
        train_sample_answer[1][count_dict["2"]] = 2
        cv2.imwrite(os.path.join(data_root, data_train,"2/",str(count_
        count_dict["2"] += 1
    elif y_train[train_index] == 3 and count_dict["3"] < 99:
        if not os.path.exists(os.path.join(data_root, data_train,"3"))
            os.makedirs(os.path.join(data_root, data_train,"3"))
        x_train[train_index] = x_train[train_index]*255
        train_sample[2][count_dict["3"]] = x_train[train_index].astype
        train_sample_answer[2][count_dict["3"]] = 3
        cv2.imwrite(os.path.join(data_root, data_train,"3/",str(count_
        count_dict["3"] += 1
    elif y_train[train_index] == 4 and count_dict["4"] < 99:
        if not os.path.exists(os.path.join(data_root, data_train,"4"))
            os.makedirs(os.path.join(data_root, data_train,"4"))
        x_train[train_index] = x_train[train_index]*255
        train_sample[3][count_dict["4"]] = x_train[train_index].astype
        train_sample_answer[3][count_dict["4"]] = 4
        cv2.imwrite(os.path.join(data_root, data_train,"4/",str(count_
```

```
        count_dict["4"] += 1
    elif y_train[train_index] == 5 and count_dict["5"] < 99:
        if not os.path.exists(os.path.join(data_root, data_train,"5"))
            os.makedirs(os.path.join(data_root, data_train,"5"))
        x_train[train_index] = x_train[train_index]*255
        train_sample[4][count_dict["5"]] = x_train[train_index].astype
        train_sample_answer[4][count_dict["5"]] = 5
        cv2.imwrite(os.path.join(data_root, data_train,"5/",str(count_
        count_dict["5"] += 1
    elif y_train[train_index] == 6 and count_dict["6"] < 99:
        if not os.path.exists(os.path.join(data_root, data_train,"6"))
            os.makedirs(os.path.join(data_root, data_train,"6"))
        x_train[train_index] = x_train[train_index]*255
        train_sample[5][count_dict["6"]] = x_train[train_index].astype
        train_sample_answer[5][count_dict["6"]] = 6
        cv2.imwrite(os.path.join(data_root, data_train,"6/",str(count_
        count_dict["6"] += 1
    elif y_train[train_index] == 7 and count_dict["7"] < 99:
        if not os.path.exists(os.path.join(data_root, data_train,"7"))
            os.makedirs(os.path.join(data_root, data_train,"7"))
        x_train[train_index] = x_train[train_index]*255
        train_sample[6][count_dict["7"]] = x_train[train_index].astype
        train_sample_answer[6][count_dict["7"]] = 7
        cv2.imwrite(os.path.join(data_root, data_train,"7/",str(count_
        count_dict["7"] += 1
    elif y_train[train_index] == 8 and count_dict["8"] < 99:
        if not os.path.exists(os.path.join(data_root, data_train,"8"))
            os.makedirs(os.path.join(data_root, data_train,"8"))
        x_train[train_index] = x_train[train_index]*255
        train_sample[7][count_dict["8"]] = x_train[train_index].astype
        train_sample_answer[7][count_dict["8"]] = 8
        cv2.imwrite(os.path.join(data_root, data_train,"8/",str(count_
        count_dict["8"] += 1
    elif y_train[train_index] == 9 and count_dict["9"] < 99:
        if not os.path.exists(os.path.join(data_root, data_train,"9"))
            os.makedirs(os.path.join(data_root, data_train,"9"))
        x_train[train_index] = x_train[train_index]*255
        train_sample[8][count_dict["9"]] = x_train[train_index].astype
        train_sample_answer[8][count_dict["9"]] = 9
        cv2.imwrite(os.path.join(data_root, data_train,"9/",str(count_
        count_dict["9"] += 1
    elif y_train[train_index] == 0 and count_dict["0"] < 99:
        if not os.path.exists(os.path.join(data_root, data_train,"0"))
            os.makedirs(os.path.join(data_root, data_train,"0"))
        x_train[train_index] = x_train[train_index]*255
        train_sample[9][count_dict["0"]] = x_train[train_index].astype
        train_sample_answer[9][count_dict["0"]] = 0
        cv2.imwrite(os.path.join(data_root, data_train,"0/",str(count_
        count_dict["0"] += 1
    else :
        pass

print("Finished !!!!!!!!")
# 上面是慢慢刻出來的，也可以直接用 numpy 的方法及特性，用 a [條件式] 來完成
```

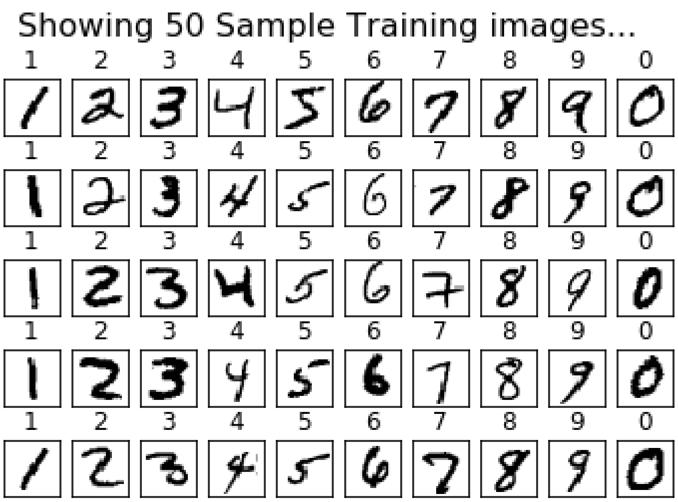
```
Saving 100 samples images for each digit in testing data....
Saving 100 samples images for each digit in training data....
```

```
Finished !!!!!!!!
```

No. 4 - 3

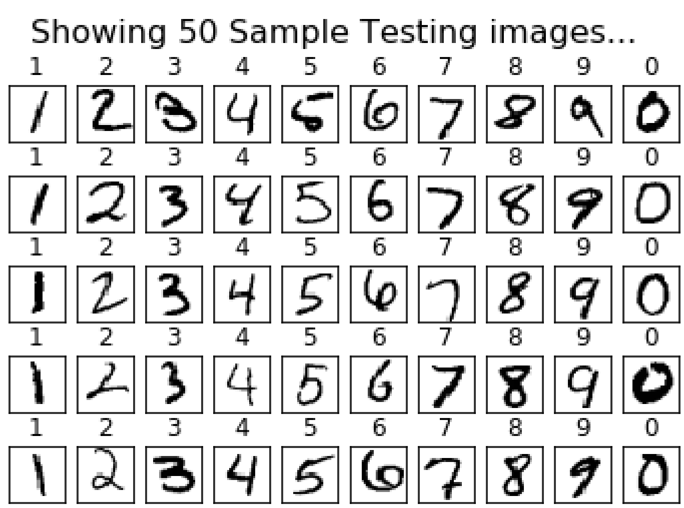
In [12]:

```python
# show training sample images
amount= 50
lines = 5
columns = 10
sample_train_data = np.zeros((50,28,28))
fig = plt.figure()
fig.suptitle('Showing 50 Sample Training images...', fontsize=16)

for i in range(amount):

    current_number_index = i%columns
    current_number_image = i//columns
    sample_train_data[i] = train_sample[current_number_index,current_nu
    ax = fig.add_subplot(lines, columns, 1 + i)
    ax.set_title(int(train_sample_answer[current_number_index,current_n
    ax.imshow(train_sample[current_number_index,current_number_image],
    ax.set_xticks([], [])
    ax.set_yticks([], [])

plt.show()
```



In [13]:

```python
# show training sample images
amount= 50
lines = 5
columns = 10
fig = plt.figure()
sample_test_data = np.zeros((50,28,28))
fig.suptitle('Showing 50 Sample Testing images...', fontsize=16)

for i in range(amount):

    current_number_index = i%columns
    current_number_image = i//columns
    ax = fig.add_subplot(lines, columns, 1 + i)
    sample_test_data[i] = test_sample[current_number_index,current_numb
    ax.set_title(int(test_sample_answer[current_number_index,current_nu
    ax.imshow(test_sample[current_number_index,current_number_image], c
    ax.set_xticks([], [])
    ax.set_yticks([], [])

plt.show()
```



In [14]:

```
# # 第四題之三 : Normailze the training image, we then choice on images
train_sample = train_sample/255

print("We now normalize the Training Images ...\n")
train_sample = train_sample.reshape(1000,784)
normalize_mean = np.nanmean(train_sample, axis = 0)
normalize_std = np.nanstd(train_sample, axis = 0)

# handle zero std
normalize_std[normalize_std==0] = 1

print("mean is : {}, and standard variance is {} !".format(normalize_me

print("Then we normalize the training sample dataset ...")
train_sample_normalize = ((train_sample - normalize_mean) / (normalize_
print("Normalized Data is :\n{}".format((train_sample_normalize)))
```

```
We now normalize the Training Images ...

mean is : [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 7.84313725e-05 4.86274510e-04
 0.00000000e+00 3.56862745e-04 1.34901961e-03 2.03529412e-03
 2.04705882e-03 1.25098039e-03 5.45098039e-04 7.41176471e-04
 8.54901961e-04 5.09803922e-05 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 8.50980392e-04 2.68627451e-03
 3.25490196e-03 3.80784314e-03 7.63529412e-03 9.40392157e-03
 1.82274510e-02 2.06352941e-02 1.86862745e-02 1.52941176e-02
 6.37254902e-03 3.39215686e-03 1.79607843e-03 1.80392157e-04
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 1.49019608e-04 8.15686275e-04
 3.05882353e-03 4.92156863e-03 7.87450980e-03 1.24235294e-02
 1.58470588e-02 2.06980392e-02 3.15529412e-02 4.71058824e-02
 5.80313725e-02 5.73019608e-02 4.61960784e-02 3.71137255e-02
 2.09529412e-02 1.14666667e-02 4.04313725e-03 6.94117647e-04
 4.58823529e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 2.39215686e-04 1.47058824e-03 2.58823529e-03 4.92549020e-03
 1.01490196e-02 1.80666667e-02 3.22352941e-02 4.57411765e-02
 7.50901961e-02 1.03266667e-01 1.40384314e-01 1.73231373e-01
 1.89062745e-01 1.86678431e-01 1.63211765e-01 1.27760784e-01
 9.33058824e-02 5.31215686e-02 2.45960784e-02 9.30196078e-03
 2.05882353e-03 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 9.80392157e-05
 2.29803922e-03 6.00000000e-03 1.32117647e-02 2.09333333e-02
```

```
3.16823529e-02 5.40980392e-02 8.79294118e-02 1.31780392e-01
1.83898039e-01 2.46662745e-01 3.10117647e-01 3.47329412e-01
3.58043137e-01 3.48627451e-01 3.09654902e-01 2.49188235e-01
1.87694118e-01 1.20576471e-01 6.25843137e-02 3.72117647e-02
1.61372549e-02 3.25098039e-03 5.29411765e-04 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 1.63137255e-03
5.56078431e-03 1.28705882e-02 2.71254902e-02 4.01333333e-02
6.42509804e-02 1.05807843e-01 1.61356863e-01 2.22129412e-01
2.89149020e-01 3.76807843e-01 4.53654902e-01 4.82803922e-01
4.77509804e-01 4.49227451e-01 4.07149020e-01 3.47349020e-01
2.61482353e-01 1.81196078e-01 1.03133333e-01 5.35568627e-02
2.56549020e-02 5.71372549e-03 1.62745098e-03 1.19607843e-03
0.00000000e+00 0.00000000e+00 5.76470588e-04 5.24313725e-03
1.10000000e-02 2.21607843e-02 4.04823529e-02 6.43215686e-02
9.81411765e-02 1.63545098e-01 2.34776471e-01 3.03619608e-01
3.86223529e-01 4.74168627e-01 5.20278431e-01 5.18121569e-01
5.18219608e-01 4.84576471e-01 4.38658824e-01 3.93215686e-01
3.22513725e-01 2.27521569e-01 1.33705882e-01 6.84862745e-02
2.80392157e-02 1.19647059e-02 4.67843137e-03 9.33333333e-04
0.00000000e+00 0.00000000e+00 4.50980392e-04 6.54509804e-03
1.69843137e-02 3.13568627e-02 5.25294118e-02 8.61843137e-02
1.34709804e-01 2.05737255e-01 2.88384314e-01 3.75203922e-01
4.53945098e-01 4.92223529e-01 4.85105882e-01 4.56074510e-01
4.43627451e-01 4.44807843e-01 4.34184314e-01 4.14886275e-01
3.57431373e-01 2.62800000e-01 1.58396078e-01 7.92784314e-02
3.20509804e-02 1.65529412e-02 2.45490196e-03 0.00000000e+00
0.00000000e+00 3.05882353e-04 2.50980392e-04 5.46666667e-03
1.83882353e-02 3.79019608e-02 6.14392157e-02 9.93372549e-02
1.58141176e-01 2.40380392e-01 3.33580392e-01 4.13411765e-01
4.46250980e-01 4.31125490e-01 3.84733333e-01 3.45866667e-01
3.56250980e-01 3.98466667e-01 4.18098039e-01 4.16349020e-01
3.60494118e-01 2.64215686e-01 1.56011765e-01 6.94784314e-02
2.73098039e-02 8.98039216e-03 5.33333333e-04 0.00000000e+00
0.00000000e+00 3.52941176e-05 1.17647059e-05 6.66274510e-03
1.76078431e-02 3.84000000e-02 6.32745098e-02 1.02313725e-01
1.69537255e-01 2.54490196e-01 3.55149020e-01 4.11396078e-01
3.90588235e-01 3.50035294e-01 2.90576471e-01 2.81490196e-01
3.18984314e-01 3.68207843e-01 4.05121569e-01 3.99588235e-01
3.40384314e-01 2.34211765e-01 1.20964706e-01 4.44313725e-02
1.43450980e-02 2.45490196e-03 4.43137255e-04 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 5.58431373e-03
1.51058824e-02 3.51843137e-02 6.14823529e-02 1.07768627e-01
1.79631373e-01 2.80333333e-01 3.71956863e-01 3.98333333e-01
3.61470588e-01 3.15090196e-01 2.84823529e-01 2.94800000e-01
3.37717647e-01 3.90839216e-01 4.04250980e-01 3.87462745e-01
3.00360784e-01 1.93227451e-01 9.23411765e-02 3.45411765e-02
1.05176471e-02 5.52941176e-04 8.50980392e-04 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 3.57254902e-03
1.47568627e-02 3.63843137e-02 7.16980392e-02 1.28654902e-01
2.08552941e-01 3.22635294e-01 3.97215686e-01 3.95768627e-01
3.45803922e-01 3.20768627e-01 3.40360784e-01 3.75729412e-01
4.28254902e-01 4.49870588e-01 4.31164706e-01 3.63223529e-01
2.64062745e-01 1.60682353e-01 8.19058824e-02 3.31843137e-02
1.08196078e-02 8.66666667e-04 6.78431373e-04 0.00000000e+00
0.00000000e+00 0.00000000e+00 1.92156863e-04 1.30588235e-03
1.30078431e-02 3.67764706e-02 7.75215686e-02 1.47192157e-01
2.40345098e-01 3.40180392e-01 3.93286275e-01 3.77458824e-01
```

```
3.52035294e-01 3.77549020e-01 4.36592157e-01 4.89329412e-01
5.21917647e-01 5.03627451e-01 4.40721569e-01 3.41662745e-01
2.42039216e-01 1.48184314e-01 8.42705882e-02 3.72823529e-02
1.11725490e-02 1.22745098e-03 7.84313725e-05 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 1.28235294e-03
1.00392157e-02 3.87921569e-02 8.70156863e-02 1.66223529e-01
2.64886275e-01 3.50670588e-01 3.80321569e-01 3.67054902e-01
3.87635294e-01 4.58172549e-01 5.03705882e-01 5.42254902e-01
5.40117647e-01 5.15400000e-01 4.35321569e-01 3.35490196e-01
2.34796078e-01 1.48380392e-01 8.19568627e-02 3.88784314e-02
1.32745098e-02 1.17254902e-03 5.09803922e-05 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 1.48235294e-03
9.15294118e-03 4.67058824e-02 1.02968627e-01 1.85564706e-01
2.82913725e-01 3.53235294e-01 3.70917647e-01 3.65168627e-01
4.16003922e-01 4.73435294e-01 5.03541176e-01 5.20560784e-01
5.08435294e-01 4.87239216e-01 4.17321569e-01 3.25305882e-01
2.34745098e-01 1.47607843e-01 8.24901961e-02 4.15176471e-02
1.60313725e-02 3.01960784e-03 1.01568627e-03 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 1.36078431e-03
1.54078431e-02 6.23176471e-02 1.24188235e-01 2.04309804e-01
2.80937255e-01 3.28858824e-01 3.32254902e-01 3.40929412e-01
3.81454902e-01 4.21823529e-01 4.65078431e-01 4.80470588e-01
4.78600000e-01 4.57458824e-01 4.02670588e-01 3.17760784e-01
2.24866667e-01 1.41596078e-01 7.80509804e-02 3.74666667e-02
1.78705882e-02 2.81568627e-03 4.35294118e-04 0.00000000e+00
0.00000000e+00 0.00000000e+00 2.15686275e-04 3.28235294e-03
2.73529412e-02 8.15803922e-02 1.36988235e-01 2.05964706e-01
2.57741176e-01 2.82843137e-01 2.81078431e-01 3.00823529e-01
3.23274510e-01 3.70160784e-01 4.26650980e-01 4.53607843e-01
4.57698039e-01 4.31050980e-01 3.78368627e-01 2.98603922e-01
2.10701961e-01 1.30129412e-01 7.22156863e-02 4.13137255e-02
2.27450980e-02 4.99607843e-03 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 6.70588235e-04 6.14901961e-03
3.69333333e-02 9.61058824e-02 1.52666667e-01 2.14215686e-01
2.35380392e-01 2.55690196e-01 2.72564706e-01 2.79741176e-01
3.06200000e-01 3.61925490e-01 4.23403922e-01 4.61250980e-01
4.44105882e-01 4.00945098e-01 3.46850980e-01 2.64552941e-01
1.77117647e-01 1.14729412e-01 7.02000000e-02 3.58196078e-02
1.72117647e-02 2.32941176e-03 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 6.90196078e-04 1.12666667e-02
5.08470588e-02 1.12690196e-01 1.80988235e-01 2.26435294e-01
2.50364706e-01 2.81580392e-01 3.11435294e-01 3.21752941e-01
3.58686275e-01 4.13105882e-01 4.69639216e-01 4.78501961e-01
4.41603922e-01 3.84317647e-01 3.20305882e-01 2.34258824e-01
1.45552941e-01 9.76941176e-02 5.28666667e-02 2.68078431e-02
8.62352941e-03 1.39215686e-03 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 1.53725490e-03 1.53215686e-02
5.89686275e-02 1.18796078e-01 1.86023529e-01 2.34560784e-01
2.83015686e-01 3.32439216e-01 3.67831373e-01 3.93486275e-01
4.35466667e-01 4.89901961e-01 5.18815686e-01 4.93392157e-01
4.19494118e-01 3.44149020e-01 2.69690196e-01 1.83784314e-01
1.22450980e-01 6.64470588e-02 3.76313725e-02 1.97568627e-02
7.01568627e-03 1.75686275e-03 5.64705882e-04 0.00000000e+00
0.00000000e+00 0.00000000e+00 2.06666667e-03 1.36666667e-02
5.11960784e-02 1.09203922e-01 1.71525490e-01 2.41462745e-01
3.18200000e-01 3.88223529e-01 4.28611765e-01 4.61070588e-01
4.99262745e-01 5.23964706e-01 5.03964706e-01 4.44235294e-01
```

```
 3.63709804e-01 2.79662745e-01 1.96545098e-01 1.37725490e-01
 8.91843137e-02 4.51098039e-02 2.59568627e-02 1.17607843e-02
 4.91764706e-03 9.92156863e-04 1.53333333e-03 0.00000000e+00
 0.00000000e+00 0.00000000e+00 7.96078431e-04 7.30980392e-03
 3.00431373e-02 7.14549020e-02 1.25196078e-01 1.96560784e-01
 2.85976471e-01 3.55486275e-01 4.15082353e-01 4.59741176e-01
 4.68345098e-01 4.60231373e-01 4.20070588e-01 3.44313725e-01
 2.61223529e-01 1.88749020e-01 1.37945098e-01 9.32823529e-02
 5.24352941e-02 2.81254902e-02 1.28470588e-02 4.67450980e-03
 1.35294118e-03 2.43137255e-04 5.56862745e-04 0.00000000e+00
 0.00000000e+00 0.00000000e+00 1.33333333e-04 3.21176471e-03
 1.43490196e-02 3.49568627e-02 6.80901961e-02 1.29266667e-01
 1.99556863e-01 2.71592157e-01 3.21941176e-01 3.53619608e-01
 3.52160784e-01 3.26188235e-01 2.83345098e-01 2.11243137e-01
 1.58992157e-01 1.18474510e-01 8.27725490e-02 4.84352941e-02
 2.73529412e-02 1.42039216e-02 6.02745098e-03 4.00000000e-03
 5.80392157e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 2.38431373e-03
 8.46274510e-03 1.68235294e-02 2.87254902e-02 5.69921569e-02
 8.52117647e-02 1.22937255e-01 1.54737255e-01 1.65545098e-01
 1.62031373e-01 1.48635294e-01 1.33909804e-01 1.00149020e-01
 8.47333333e-02 6.26980392e-02 3.89176471e-02 2.33607843e-02
 1.38313725e-02 5.17254902e-03 2.54117647e-03 2.00000000e-03
 4.54901961e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 1.49019608e-04
 2.29019608e-03 7.34901961e-03 1.53215686e-02 2.54862745e-02
 3.67215686e-02 4.98705882e-02 6.20627451e-02 6.01882353e-02
 5.52431373e-02 5.01176471e-02 4.23215686e-02 3.45960784e-02
 3.35294118e-02 2.10666667e-02 1.38941176e-02 7.78823529e-03
 4.54509804e-03 7.37254902e-04 8.35294118e-04 9.49019608e-04
 2.07843137e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 1.49019608e-04
 1.65490196e-03 3.24313725e-03 9.49019608e-03 1.30941176e-02
 1.71529412e-02 2.10196078e-02 2.24431373e-02 1.69254902e-02
 1.38313725e-02 1.48431373e-02 1.40431373e-02 1.47607843e-02
 1.45764706e-02 8.88235294e-03 4.38823529e-03 2.17647059e-03
 5.52941176e-04 1.45098039e-04 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 6.94117647e-04 1.61176471e-03 1.96470588e-03
 1.15686275e-03 0.00000000e+00 4.70588235e-05 7.96078431e-04
 1.42352941e-03 1.84705882e-03 6.03921569e-04 6.23529412e-04
 1.63529412e-03 1.40392157e-03 1.24705882e-03 4.31372549e-04
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00], and standa
rd variance is [1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+
00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 2.47897735e-03 1.53696596e-02
 1.00000000e+00 1.12793470e-02 3.14845217e-02 3.61362791e-02
```

```
 4.02035413e-02 3.25795366e-02 1.72288926e-02 2.34263360e-02
 2.70208532e-02 1.61133528e-03 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.94305360e-02 4.65223960e-02
 4.91598152e-02 4.83556857e-02 7.57403232e-02 7.73946776e-02
 1.20883337e-01 1.23423257e-01 1.19338802e-01 1.12402446e-01
 6.23063551e-02 5.16202344e-02 3.84570215e-02 5.70164791e-03
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 4.71005697e-03 1.57804701e-02
 4.39299012e-02 6.44448657e-02 7.73841793e-02 9.91274331e-02
 1.12282209e-01 1.20915686e-01 1.52137248e-01 1.83433269e-01
 2.10921385e-01 2.04245273e-01 1.87770898e-01 1.70077337e-01
 1.23823326e-01 9.64511557e-02 5.19038592e-02 1.79441960e-02
 1.45020175e-02 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 5.91790433e-03 2.38469988e-02 3.63921569e-02 5.25912128e-02
 9.07527795e-02 1.12041113e-01 1.52591838e-01 1.73031016e-01
 2.26424937e-01 2.60273896e-01 2.97219337e-01 3.32988142e-01
 3.50261516e-01 3.48050591e-01 3.29397148e-01 2.93712237e-01
 2.52495980e-01 1.92485269e-01 1.29671042e-01 7.56215934e-02
 3.39185933e-02 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 2.19178502e-03
 3.72244672e-02 6.88343213e-02 9.87655322e-02 1.27885459e-01
 1.54075170e-01 1.96331483e-01 2.52538969e-01 2.98743831e-01
 3.44480391e-01 3.83326319e-01 4.14108690e-01 4.22801467e-01
 4.28516506e-01 4.29960040e-01 4.18413373e-01 3.87076388e-01
 3.50808523e-01 2.85648765e-01 2.06978197e-01 1.65773843e-01
 1.08349597e-01 4.28481864e-02 1.20964819e-02 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 2.70971437e-02
 6.79893331e-02 9.66798683e-02 1.45540930e-01 1.75286276e-01
 2.18007705e-01 2.74124958e-01 3.29143231e-01 3.71629899e-01
 4.01965829e-01 4.23860267e-01 4.35960431e-01 4.43186042e-01
 4.36234945e-01 4.39469590e-01 4.40065314e-01 4.24197667e-01
 3.93499277e-01 3.40648656e-01 2.63081891e-01 1.97555898e-01
 1.38649289e-01 5.80924704e-02 2.80968214e-02 2.69363454e-02
 1.00000000e+00 1.00000000e+00 1.82204835e-02 6.44142484e-02
 9.03637806e-02 1.28009707e-01 1.71700439e-01 2.17868688e-01
 2.63231254e-01 3.22717946e-01 3.77590788e-01 3.97164616e-01
 4.13643516e-01 4.27715432e-01 4.31018763e-01 4.32213056e-01
 4.36630288e-01 4.30552688e-01 4.26582277e-01 4.29842296e-01
 4.15218474e-01 3.72060726e-01 2.90676337e-01 2.17574788e-01
 1.36705427e-01 9.37675186e-02 5.72637317e-02 2.48403920e-02
 1.00000000e+00 1.00000000e+00 1.42541198e-02 6.37724037e-02
 1.15471215e-01 1.56595851e-01 1.96827548e-01 2.52046708e-01
 3.01202761e-01 3.61821498e-01 4.00480230e-01 4.23978423e-01
 4.33497910e-01 4.40457239e-01 4.34291252e-01 4.30720213e-01
 4.28544336e-01 4.34449145e-01 4.35707433e-01 4.35186356e-01
 4.27513469e-01 3.93682753e-01 3.21211426e-01 2.36886226e-01
 1.55417908e-01 1.12733616e-01 3.40590265e-02 1.00000000e+00
 1.00000000e+00 9.66801168e-03 7.33857839e-03 6.24235343e-02
 1.17208351e-01 1.71124727e-01 2.12669386e-01 2.66006722e-01
 3.29507589e-01 3.83725496e-01 4.13676366e-01 4.30184524e-01
 4.30586513e-01 4.35453513e-01 4.17902224e-01 4.08270093e-01
 4.18980650e-01 4.27887996e-01 4.38513809e-01 4.35262858e-01
```

```
4.32707036e-01 4.01145771e-01 3.18860908e-01 2.20295314e-01
1.37750517e-01 7.22333248e-02 1.33997791e-02 1.00000000e+00
1.00000000e+00 1.11553981e-03 3.71846603e-04 7.31069589e-02
1.14951733e-01 1.67444677e-01 2.18018999e-01 2.66892865e-01
3.33918291e-01 3.80617640e-01 4.15583784e-01 4.24677767e-01
4.31513667e-01 4.16978346e-01 3.88896201e-01 3.87887458e-01
4.12412203e-01 4.17039078e-01 4.23725270e-01 4.32266097e-01
4.27252899e-01 3.81991218e-01 2.83332785e-01 1.70144616e-01
1.01211807e-01 3.72107321e-02 1.40062220e-02 1.00000000e+00
1.00000000e+00 1.00000000e+00 1.00000000e+00 6.12285128e-02
1.09260567e-01 1.65581204e-01 2.16512085e-01 2.77043479e-01
3.42867129e-01 3.97563741e-01 4.26422187e-01 4.35228866e-01
4.20920877e-01 4.00583598e-01 3.97223488e-01 4.00236296e-01
4.12670602e-01 4.18245086e-01 4.21640431e-01 4.33942119e-01
4.09307919e-01 3.54091469e-01 2.47822343e-01 1.56235074e-01
8.76356861e-02 1.28643704e-02 2.68969043e-02 1.00000000e+00
1.00000000e+00 1.00000000e+00 1.00000000e+00 4.51547945e-02
1.11573199e-01 1.67071284e-01 2.31035944e-01 3.01858647e-01
3.60325971e-01 4.14864412e-01 4.35230872e-01 4.38475615e-01
4.19564880e-01 4.01455529e-01 4.20869986e-01 4.29768341e-01
4.32143855e-01 4.28977826e-01 4.35249038e-01 4.27746203e-01
3.89917076e-01 3.26218918e-01 2.39976899e-01 1.56193994e-01
8.80991472e-02 1.93616541e-02 2.14431541e-02 1.00000000e+00
1.00000000e+00 1.00000000e+00 6.07349452e-03 2.36357437e-02
9.47040970e-02 1.62798154e-01 2.40318710e-01 3.13093993e-01
3.83591539e-01 4.18830386e-01 4.35396120e-01 4.26642490e-01
4.11749423e-01 4.18801853e-01 4.47773310e-01 4.43654429e-01
4.29872599e-01 4.32662179e-01 4.39262507e-01 4.17958434e-01
3.77880644e-01 3.13045283e-01 2.50242170e-01 1.61764300e-01
9.08982218e-02 2.96557283e-02 2.47897735e-03 1.00000000e+00
1.00000000e+00 1.00000000e+00 1.00000000e+00 2.86564642e-02
8.93335055e-02 1.69077481e-01 2.48218296e-01 3.32602179e-01
3.95961541e-01 4.25704324e-01 4.35570548e-01 4.12731157e-01
4.11460982e-01 4.38913559e-01 4.46292273e-01 4.26422713e-01
4.23246076e-01 4.40679959e-01 4.36973178e-01 4.19572832e-01
3.77689385e-01 3.17082491e-01 2.42327519e-01 1.68463879e-01
9.07216380e-02 2.17445938e-02 1.61133528e-03 1.00000000e+00
1.00000000e+00 1.00000000e+00 1.00000000e+00 2.75212311e-02
7.57626805e-02 1.81938804e-01 2.68646026e-01 3.49409503e-01
4.08324123e-01 4.33732677e-01 4.34873581e-01 4.13539073e-01
4.32180108e-01 4.43789784e-01 4.42245547e-01 4.23496566e-01
4.35473858e-01 4.41710657e-01 4.34952085e-01 4.12667894e-01
3.78739779e-01 3.14159122e-01 2.39592175e-01 1.76410728e-01
1.07788810e-01 4.76787759e-02 2.74895860e-02 1.00000000e+00
1.00000000e+00 1.00000000e+00 1.00000000e+00 2.70425046e-02
9.66642373e-02 2.08950182e-01 2.93751505e-01 3.64639404e-01
4.07133181e-01 4.25948197e-01 4.14568408e-01 4.13491599e-01
4.33481007e-01 4.41010475e-01 4.35002640e-01 4.36302482e-01
4.38848992e-01 4.36976274e-01 4.31549184e-01 4.11366010e-01
3.74914035e-01 3.12018877e-01 2.33386107e-01 1.60922422e-01
1.15134928e-01 4.19883501e-02 1.37583243e-02 1.00000000e+00
1.00000000e+00 1.00000000e+00 6.81718772e-03 3.88991996e-02
1.39925634e-01 2.41628468e-01 3.09355828e-01 3.60869461e-01
3.94443019e-01 4.02722404e-01 3.92531417e-01 4.04347613e-01
4.17754562e-01 4.24729022e-01 4.34363506e-01 4.42127204e-01
4.32799398e-01 4.29867236e-01 4.28122368e-01 4.02881800e-01
3.65222985e-01 2.97662494e-01 2.24478096e-01 1.78120512e-01
```

```
1.31266683e-01 6.23266592e-02 1.00000000e+00 1.00000000e+00
1.00000000e+00 1.00000000e+00 1.66543248e-02 6.18245432e-02
1.62289429e-01 2.63094762e-01 3.24703351e-01 3.67566232e-01
3.78764895e-01 3.86960287e-01 3.94886889e-01 3.98005339e-01
4.10628806e-01 4.20811254e-01 4.34185569e-01 4.28569033e-01
4.30387538e-01 4.28728204e-01 4.24913193e-01 3.92837341e-01
3.36123771e-01 2.89987537e-01 2.27048433e-01 1.64619400e-01
1.10943642e-01 3.59646894e-02 1.00000000e+00 1.00000000e+00
1.00000000e+00 1.00000000e+00 1.32145947e-02 9.33337346e-02
1.92150334e-01 2.83847415e-01 3.52712582e-01 3.77740626e-01
3.88062674e-01 4.05417451e-01 4.14128248e-01 4.17458464e-01
4.25861170e-01 4.31754715e-01 4.36126239e-01 4.34336797e-01
4.28983798e-01 4.29444515e-01 4.15571725e-01 3.75112631e-01
3.14620257e-01 2.68217585e-01 1.98776173e-01 1.41096770e-01
7.39112594e-02 3.23540047e-02 1.00000000e+00 1.00000000e+00
1.00000000e+00 1.00000000e+00 3.47863996e-02 1.05887026e-01
2.10203966e-01 2.93255656e-01 3.53925119e-01 3.81379354e-01
4.01847032e-01 4.19112774e-01 4.33457879e-01 4.31421639e-01
4.27148590e-01 4.32589787e-01 4.37515568e-01 4.35675452e-01
4.30552771e-01 4.21906430e-01 3.96344204e-01 3.48840376e-01
2.94195660e-01 2.18031064e-01 1.72963174e-01 1.19636805e-01
6.56588739e-02 3.44993959e-02 1.78486369e-02 1.00000000e+00
1.00000000e+00 1.00000000e+00 3.61715063e-02 1.00683995e-01
1.91695564e-01 2.86934281e-01 3.48424495e-01 3.86963031e-01
4.20285235e-01 4.34698674e-01 4.34702272e-01 4.30172389e-01
4.34290251e-01 4.34264383e-01 4.37362264e-01 4.38227909e-01
4.24100260e-01 4.00484355e-01 3.57256771e-01 3.11416839e-01
2.51655386e-01 1.86107995e-01 1.42524061e-01 9.21821408e-02
6.19852873e-02 2.02988025e-02 3.56082143e-02 1.00000000e+00
1.00000000e+00 1.00000000e+00 1.57644088e-02 7.08953563e-02
1.43703745e-01 2.27735527e-01 3.00609540e-01 3.57365986e-01
4.13513043e-01 4.33517303e-01 4.43593593e-01 4.45163565e-01
4.42278203e-01 4.39697007e-01 4.38685571e-01 4.18945345e-01
3.88456492e-01 3.46085215e-01 3.12220878e-01 2.57841024e-01
1.94107798e-01 1.44912887e-01 9.41427927e-02 5.37519809e-02
2.75125290e-02 7.68482980e-03 1.76007392e-02 1.00000000e+00
1.00000000e+00 1.00000000e+00 4.21426150e-03 4.30012542e-02
9.28086023e-02 1.53924271e-01 2.15637258e-01 2.89422648e-01
3.54883809e-01 3.96131941e-01 4.17656485e-01 4.28416199e-01
4.22130431e-01 4.08905636e-01 3.94505574e-01 3.53867354e-01
3.27081349e-01 2.86561352e-01 2.41678150e-01 1.89294967e-01
1.42797253e-01 1.00997799e-01 6.38058174e-02 5.53588274e-02
1.49112953e-02 1.00000000e+00 1.00000000e+00 1.00000000e+00
1.00000000e+00 1.00000000e+00 1.00000000e+00 3.21421143e-02
8.06643919e-02 1.16033106e-01 1.49094170e-01 2.01544628e-01
2.46412274e-01 2.93858218e-01 3.25152299e-01 3.31545893e-01
3.28351255e-01 3.17770840e-01 2.96901424e-01 2.62991940e-01
2.50282380e-01 2.10441394e-01 1.65911100e-01 1.36559423e-01
1.01292020e-01 5.63366111e-02 4.21739311e-02 3.89232034e-02
1.43780687e-02 1.00000000e+00 1.00000000e+00 1.00000000e+00
1.00000000e+00 1.00000000e+00 1.00000000e+00 4.71005697e-03
3.63182448e-02 7.46910205e-02 1.07978834e-01 1.40358305e-01
1.64853412e-01 1.95505788e-01 2.22466607e-01 2.13988286e-01
2.05510156e-01 1.93334988e-01 1.77023740e-01 1.63003448e-01
1.64132311e-01 1.30191685e-01 9.56745534e-02 8.12802031e-02
5.97747559e-02 1.41810428e-02 2.25793934e-02 2.99956260e-02
6.56928999e-03 1.00000000e+00 1.00000000e+00 1.00000000e+00
```

```
 1.00000000e+00 1.00000000e+00 1.00000000e+00 4.71005697e-03
 3.37044887e-02 4.64961576e-02 8.30424543e-02 1.04534050e-01
 1.12339090e-01 1.26668148e-01 1.34824206e-01 1.14736557e-01
 9.58816128e-02 1.04360654e-01 9.67735907e-02 1.03582006e-01
 1.06806491e-01 8.66944126e-02 4.95951838e-02 4.08673692e-02
 1.74767903e-02 4.58610810e-03 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 2.19389496e-02 3.20965669e-02 4.23300590e-02
 3.16813176e-02 1.00000000e+00 1.48738641e-03 2.51616201e-02
 3.41823212e-02 4.13875373e-02 1.53002924e-02 1.47982558e-02
 3.66735049e-02 3.32838932e-02 3.00333983e-02 1.36343754e-02
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
 1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00] !
Then we normalize the training sample dataset ...
Normalized Data is :
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

No. 4 - 3

In [15]:

```
print("Then we can also compute the Covariance Matrix ...")
train_cov = np.matmul(train_sample_normalize.T, train_sample_normalize)
print("Covariance of the sample data is : \n{}".format(train_cov))
```

```
Then we can also compute the Covariance Matrix ...
Covariance of the sample data is :
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

No. 4 - 4

In [16]:

```
#  第四題之四 :  Compute eigenpairs for the covariance
# reshape train sample
train_sample = train_sample.reshape(10,100,784)

# eigenvectors and eigenvalues for the from the covariance matrix
eig_val_cov, eig_vec_cov = np.linalg.eig(train_cov)

# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_val_cov[i]), eig_vec_cov[:,i]) for i in range(

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort(key=lambda x: x[0], reverse=True)

eig_pairs_500 = eig_pairs[:500]
eig_pairs_300 = eig_pairs[:300]
eig_pairs_100 = eig_pairs[:100]
eig_pairs_50 = eig_pairs[:50]
eig_pairs_10 = eig_pairs[:10]
eig_pairs_2 = eig_pairs[:2]
eig_pairs_biggest = eig_pairs[:1]
```

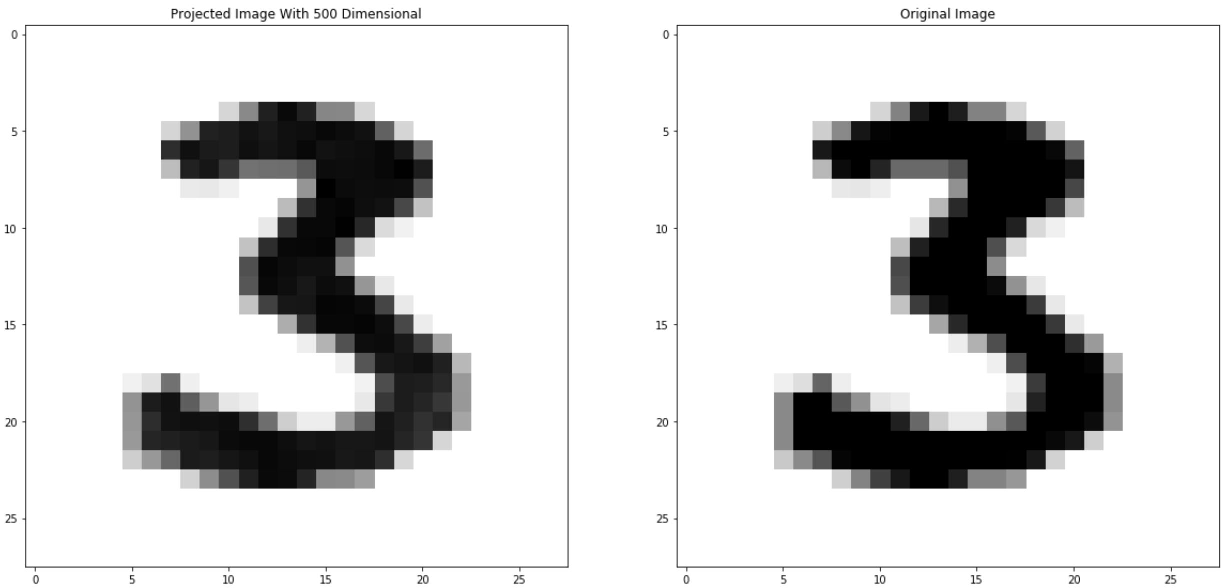
No. 4 - 5

In [17]:
```
# plot 前 500 大的 eigen vectors 所畫出來的圖
# first we random choice the digit
# then radom choice image
random_digit = random.choice(train_sample)
random_image = random.choice(random_digit)
vectors = np.zeros((500,784))

projected_result = np.zeros((784))

count = 0
for each_value, each_vector in eig_pairs_500 :
    tmp = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result += (tmp - np.mean(tmp))/ np.std(tmp)
    vectors[count] = each_vector
    count += 0

print("The New Data is : \n{}".format(np.matmul(vectors , train_sample.

fig = plt.figure(figsize=(20,10))
fig.suptitle('Showing Comparaison between Decoding (Eigen vectors of To

ax = fig.add_subplot(1, 2, 1)
ax.set_title("Projected Image With 500 Dimensional")
ax.imshow(projected_result.reshape(28,28), cmap='binary')
ax1 = fig.add_subplot(1, 2, 2)
ax1.set_title("Original Image")
ax1.imshow(random_image.reshape(28,28), cmap='binary')
```

```
The New Data is :
[[ 0.02086029  0.0060809   0.01243795 ...  0.0584147  -0.04757836
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 ...
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]]

/Users/wangboren/.pyenv/versions/3.6.4/lib/python3.6/site-packages/ipyker
nel_launcher.py:14: ComplexWarning: Casting complex values to real discar
ds the imaginary part
```

Out[17]:
```
<matplotlib.image.AxesImage at 0x143df68d0>
```

Showing Comparaison between Decoding (Eigen vectors of Top 500 eigen vlaues) and Original Result

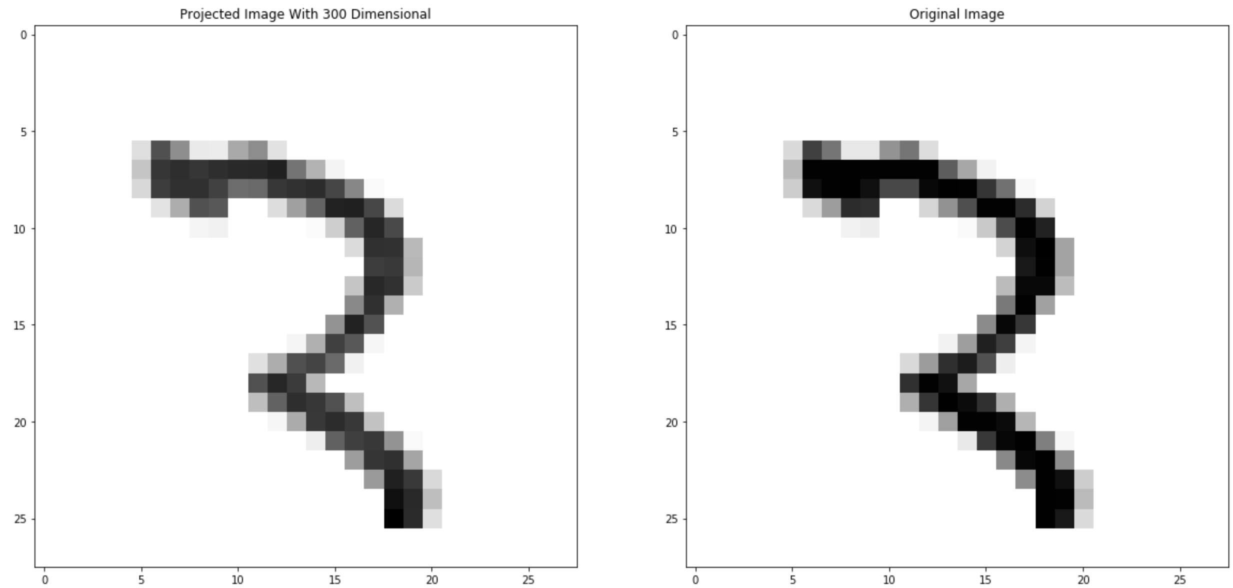


In [18]:
```
# plot 前 300 大的 eigen vectors 所畫出來的圖
# first we random choice the digit
# then radom choice image
random_digit = random.choice(train_sample)
random_image = random.choice(random_digit)
vectors = np.zeros((300,784))

projected_result = np.zeros((784))

count = 0
for each_value, each_vector in eig_pairs_300 :
    tmp = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result += (tmp - np.mean(tmp))/ np.std(tmp)
    vectors[count] = each_vector
    count += 0

print("The New Data is : \n{}".format(np.matmul(vectors , train_sample.resh

fig = plt.figure(figsize=(20,10))
fig.suptitle('Showing Comparaison between Decoding (Eigen vectors of Top 30

ax2 = fig.add_subplot(1, 2, 1)
ax3.set_title("Projected Image With 300 Dimensional")
ax4.imshow(projected_result.reshape(28,28), cmap='binary')
ax1 = fig.add_subplot(1, 2, 2)
ax1.set_title("Original Image")
ax1.imshow(random_image.reshape(28,28), cmap='binary')
```

```
The New Data is :
[[-0.048741   -0.08137146  0.05134475 ... -0.12016946 -0.03222935
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 ...
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]]

/Users/wangboren/.pyenv/versions/3.6.4/lib/python3.6/site-packages/ipyker
nel_launcher.py:14: ComplexWarning: Casting complex values to real discar
ds the imaginary part
```

Out[18]: <matplotlib.image.AxesImage at 0x1439d74a8>

Showing Comparaison between Decoding (Eigen vectors of Top 300 eigen vlaues) and Original Result



In [19]:

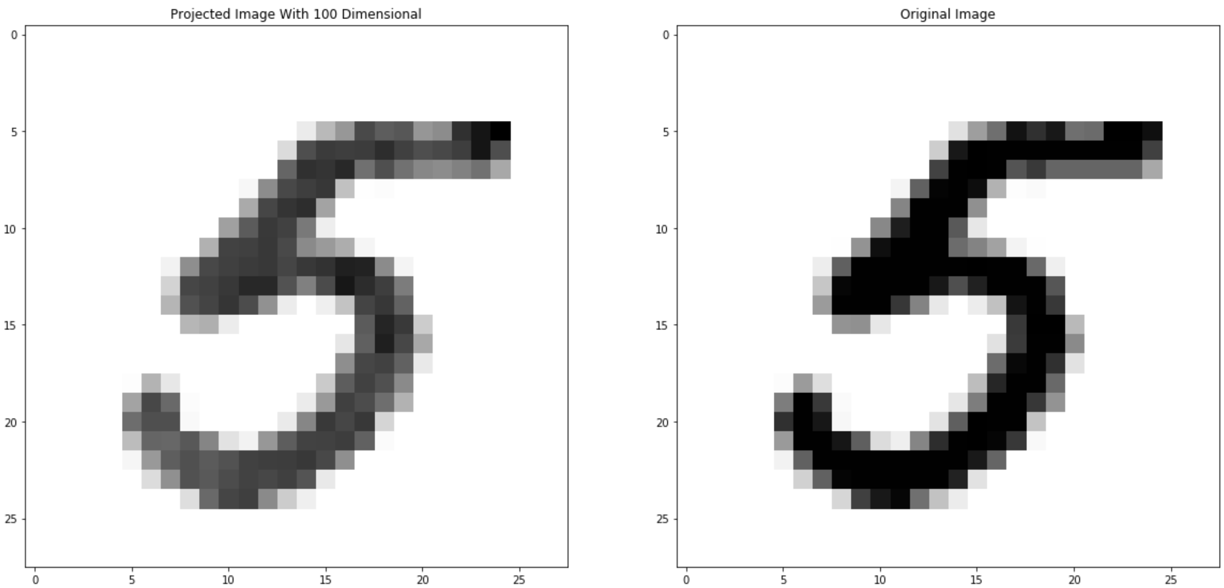
```python
# plot 前 100 大的 eigen vectors 所畫出來的圖
# first we random choice the digit
# then radom choice image
random_digit = random.choice(train_sample)
random_image = random.choice(random_digit)
vectors = np.zeros((100,784))

projected_result = np.zeros((784))

count = 0
for each_value, each_vector in eig_pairs_100 :
    tmp = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result += (tmp - np.mean(tmp))/ np.std(tmp)
    vectors[count] = each_vector
    count += 0

print("The New Data is : \n{}".format(np.matmul(vectors , train_sample.

fig = plt.figure(figsize=(20,10))
fig.suptitle('Showing Comparaison between Decoding (Eigen vectors of To

ax = fig.add_subplot(1, 2, 1)
ax.set_title("Projected Image With 100 Dimensional")
ax.imshow(projected_result.reshape(28,28), cmap='binary')
ax1 = fig.add_subplot(1, 2, 2)
ax1.set_title("Original Image")
ax1.imshow(random_image.reshape(28,28), cmap='binary')
```

```
The New Data is :
[[-0.08048171 -0.5481282  -0.31608903 ... -0.67696241 -0.60345038
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 ...
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]]

/Users/wangboren/.pyenv/versions/3.6.4/lib/python3.6/site-packages/ipyker
nel_launcher.py:14: ComplexWarning: Casting complex values to real discar
ds the imaginary part
```

Out[19]:

```
<matplotlib.image.AxesImage at 0x143d9c6d8>
```

Showing Comparaison between Decoding (Eigen vectors of Top 100 eigen vlaues) and Original Result



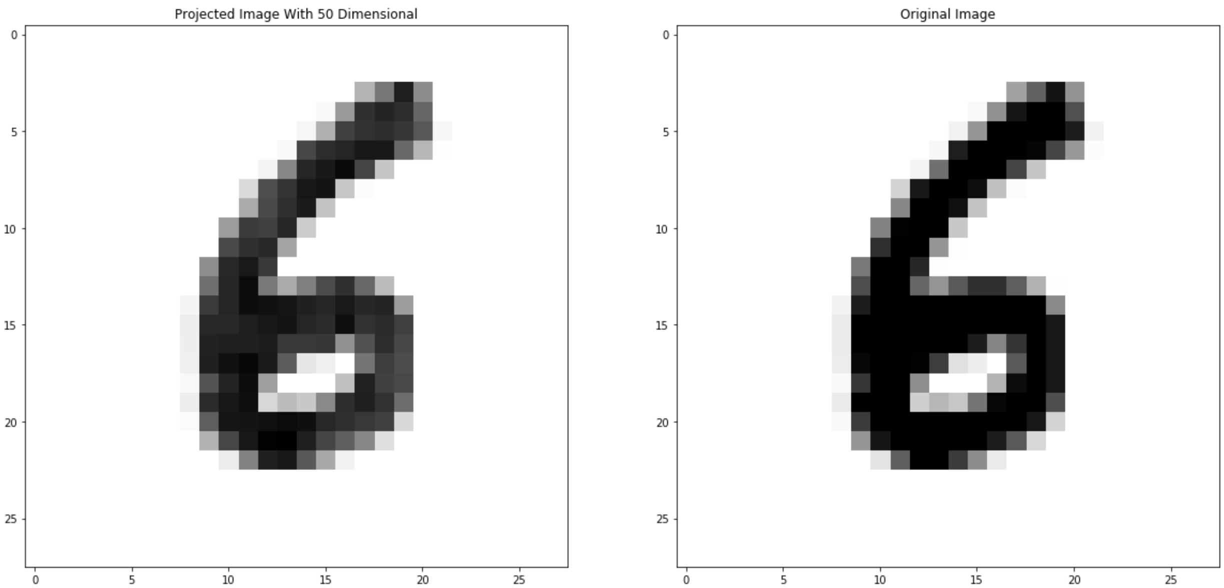
In [20]:

```python
# plot 前 50 大的 eigen vectors 所畫出來的圖
# first we random choice the digit
# then radom choice image
random_digit = random.choice(train_sample)
random_image = random.choice(random_digit)
vectors = np.zeros((50,784))

projected_result = np.zeros((784))

count = 0
for each_value, each_vector in eig_pairs_50 :
    tmp = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result += (tmp - np.mean(tmp))/ np.std(tmp)
    vectors[count] = each_vector
    count += 0

print("The New Data is : \n{}".format(np.matmul(vectors , train_sample.

fig = plt.figure(figsize=(20,10))
fig.suptitle('Showing Comparaison between Decoding (Eigen vectors of To

ax = fig.add_subplot(1, 2, 1)
ax.set_title("Projected Image With 50 Dimensional")
ax.imshow(projected_result.reshape(28,28), cmap='binary')
ax1 = fig.add_subplot(1, 2, 2)
ax1.set_title("Original Image")
ax1.imshow(random_image.reshape(28,28), cmap='binary')
```

```
The New Data is :
[[0.52669256 0.58678696 0.71077567 ... 0.77692768 0.26963672 0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]

/Users/wangboren/.pyenv/versions/3.6.4/lib/python3.6/site-packages/ipyker
nel_launcher.py:14: ComplexWarning: Casting complex values to real discar
ds the imaginary part
```

Out[20]: <matplotlib.image.AxesImage at 0x143981908>

Showing Comparaison between Decoding (Eigen vectors of Top 50 eigen vlaues) and Original Result

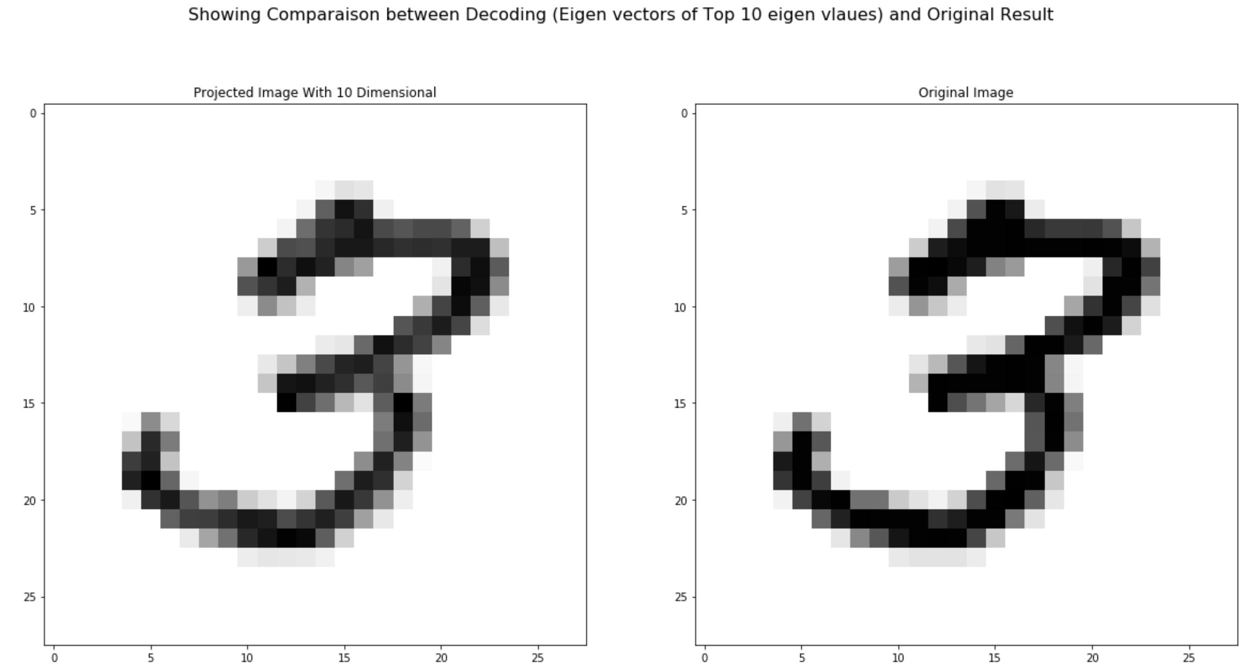


In [21]:

```
# plot 前 10 大的 eigen vectors 所畫出來的圖
# first we random choice the digit
# then radom choice image
random_digit = random.choice(train_sample)
random_image = random.choice(random_digit)
vectors = np.zeros((10,784))

projected_result = np.zeros((784))

count = 0
for each_value, each_vector in eig_pairs_10 :
    tmp = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result += (tmp - np.mean(tmp))/ np.std(tmp)
    vectors[count] = each_vector
    count += 0

print("The New Data is : \n{}".format(np.matmul(vectors , train_sample.
fig = plt.figure(figsize=(20,10))
fig.suptitle('Showing Comparaison between Decoding (Eigen vectors of To

ax = fig.add_subplot(1, 2, 1)
ax.set_title("Projected Image With 10 Dimensional")
ax.imshow(projected_result.reshape(28,28), cmap='binary')
ax1 = fig.add_subplot(1, 2, 2)
ax1.set_title("Original Image")
ax1.imshow(random_image.reshape(28,28), cmap='binary')
```

```
The New Data is :
[[ 1.5865072   0.6157811   0.37166823 ... -0.11523196 -0.73602805
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 ...
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]]

/Users/wangboren/.pyenv/versions/3.6.4/lib/python3.6/site-packages/ipyker
nel_launcher.py:14: ComplexWarning: Casting complex values to real discar
ds the imaginary part
```

Out[21]: <matplotlib.image.AxesImage at 0x10db21b38>


Showing Comparaison between Decoding (Eigen vectors of Top 10 eigen vlaues) and Original Result
Projected Image With 10 Dimensional
Original Image
0
5
10
15
20
25

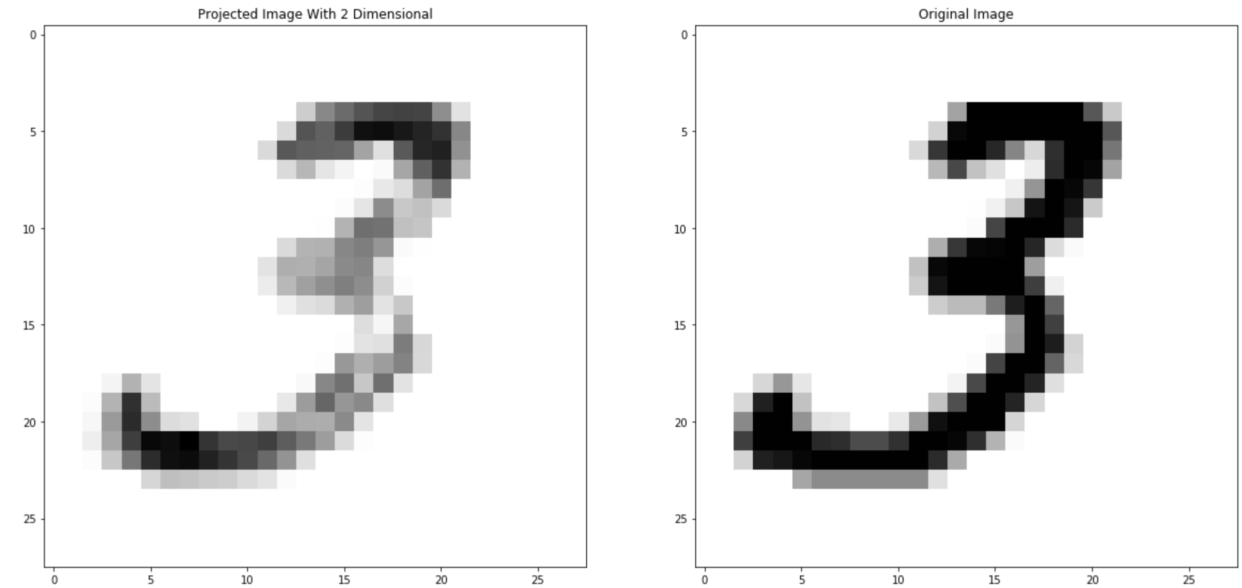
In [22]:

```python
# plot 前 2 大的 eigen vectors 所畫出來的圖
# first we random choice the digit
# then radom choice image
random_digit = random.choice(train_sample)
random_image = random.choice(random_digit)
vectors = np.zeros((2,784))

projected_result = np.zeros((784))

count = 0
for each_value, each_vector in eig_pairs_2 :
    tmp = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result += (tmp - np.mean(tmp))/ np.std(tmp)
    vectors[count] = each_vector
    count += 0

print("The New Data is : \n{}".format(np.matmul(vectors , train_sample.

fig = plt.figure(figsize=(20,10))
fig.suptitle('Showing Comparaison between Decoding (Eigen vectors of To

ax = fig.add_subplot(1, 2, 1)
ax.set_title("Projected Image With 2 Dimensional")
ax.imshow(projected_result.reshape(28,28), cmap='binary')
ax1 = fig.add_subplot(1, 2, 2)
ax1.set_title("Original Image")
ax1.imshow(random_image.reshape(28,28), cmap='binary')
```

```
The New Data is :
[[-1.52921289 -0.31946832 -0.36104658 ... -2.19082761 -4.23549517
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]]

/Users/wangboren/.pyenv/versions/3.6.4/lib/python3.6/site-packages/ipyker
nel_launcher.py:14: ComplexWarning: Casting complex values to real discar
ds the imaginary part
```

Out[22]: <matplotlib.image.AxesImage at 0x10dbe4d68>

Showing Comparaison between Decoding (Eigen vectors of Top 2 eigen vlaues) and Original Result



In [23]:

```
# plot 前 1 大的 eigen vectors 所畫出來的圖
# first we random choice the digit
# then radom choice image
random_digit = random.choice(train_sample)
random_image = random.choice(random_digit)
vectors = np.zeros((1,784))

projected_result = np.zeros((784))

count = 0
for each_value, each_vector in eig_pairs_biggest :
    tmp = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result += (tmp - np.mean(tmp))/ np.std(tmp)
    vectors[count] = each_vector
    count += 0

print("The New Data is : \n{}".format(np.matmul(vectors , train_sample.

fig = plt.figure(figsize=(20,10))
fig.suptitle('Showing Comparaison between Decoding (Eigen vectors of Bi

ax = fig.add_subplot(1, 2, 1)
ax.set_title("Projected Image With Biggest Dimensional")
ax.imshow(projected_result.reshape(28,28), cmap='binary')
ax1 = fig.add_subplot(1, 2, 2)
ax1.set_title("Original Image")
ax1.imshow(random_image.reshape(28,28), cmap='binary')
```

```
The New Data is :
[[-5.87548696e-01  1.13263370e-01 -3.85826422e-01 -3.22809676e-01
  -6.28629174e-01  1.21978670e+00 -6.65365684e-01 -3.66026543e-01
  -6.58843700e-01  1.67620625e+00 -4.23648026e-01 -6.34063253e-01
  -8.03149267e-01 -7.27381229e-01 -2.94386744e-01  3.36819963e-02
  -5.23457486e-01 -6.65036872e-01 -3.75742670e-01 -4.72484763e-01
  -5.40990284e-01  2.09294074e+00 -2.56917593e-01 -5.71421712e-01
  -8.89599875e-01 -1.14693967e-01  3.25907651e-02 -5.10005866e-01
  -3.61754801e-01 -7.34991710e-01 -6.38190062e-01  1.61052352e+00
  -5.50602591e-01  5.65570483e-01 -8.48859944e-01 -4.60520955e-01
  -2.74417988e-01 -5.86320562e-01 -6.20731318e-01 -3.86553230e-01
  -4.93972094e-01 -7.00285833e-01 -6.73338583e-01 -6.19184123e-01
  -5.20725695e-01 -8.64561754e-01 -7.72120478e-02 -4.06630572e-01
  -2.41561707e-01  4.40311281e-01 -4.85687841e-01 -1.43954658e-01
   2.68236449e-01 -5.69491255e-01 -1.09816166e+00 -4.39247664e-01
  -3.86769439e-01 -2.73088603e-01 -5.73726649e-01 -7.96170411e-01
  -5.30500072e-01  4.11016330e-01 -4.73612516e-01 -2.35357057e-01
  -2.12951420e-02 -7.11414010e-01 -5.68600027e-01 -5.23736794e-01
  -3.72710987e-01 -1.08741707e-02  1.69324382e-01 -4.32391172e-01
```

In [24]:

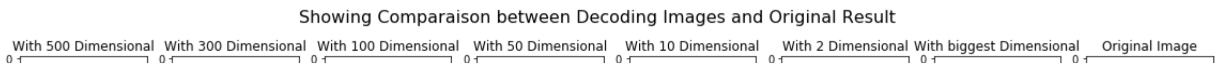
```
# combine all result for all decoded images
random_digit = random.choice(train_sample)
random_image = random.choice(random_digit)

projected_result_500 = np.zeros((784))
projected_result_300 = np.zeros((784))
projected_result_100 = np.zeros((784))
projected_result_50 = np.zeros((784))
projected_result_10 = np.zeros((784))
projected_result_2 = np.zeros((784))
projected_result_biggest = np.zeros((784))

vectors_500 = np.zeros((500,784))
vectors_300 = np.zeros((300,784))
vectors_100 = np.zeros((100,784))
vectors_50 = np.zeros((50,784))
vectors_10 = np.zeros((10,784))
vectors_2 = np.zeros((2,784))
vectors_biggest = np.zeros((1,784))

count_500 = 0
count_300 = 0
count_100 = 0
count_50 = 0
count_10 = 0
count_2 = 0
count_biggest = 0

for each_value, each_vector in eig_pairs_500 :
    tmp_500 = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result_500 += (tmp_500 - np.mean(tmp_500))/ np.std(tmp_5(
    vectors_500[count_500] = each_vector
    count_500 += 0

for each_value, each_vector in eig_pairs_300 :
    tmp_300 = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result_300 += (tmp_300 - np.mean(tmp_300))/ np.std(tmp_3(
    vectors_300[count_300] = each_vector
    count_300 += 0

for each_value, each_vector in eig_pairs_100 :
    tmp_100 = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result_100 += (tmp_100 - np.mean(tmp_100))/ np.std(tmp_1(
    vectors_100[count_100] = each_vector
    count_100 += 0

for each_value, each_vector in eig_pairs_50 :
    tmp_50 = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result_50 += (tmp_50 - np.mean(tmp_50))/ np.std(tmp_50)
    vectors_50[count_50] = each_vector
    count_50 += 0

for each_value, each_vector in eig_pairs_10 :
    tmp_10 = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result_10 += (tmp_10 - np.mean(tmp_10))/ np.std(tmp_10)
    vectors_10[count_10] = each_vector
```

```
    count_10 += 0

for each_value, each_vector in eig_pairs_10 :
    tmp_2 = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result_2 += (tmp_2 - np.mean(tmp_2))/ np.std(tmp_2)
    vectors_2[count_2] = each_vector
    count_2 += 0

for each_value, each_vector in eig_pairs_biggest :
    tmp_biggest = np.abs(random_image.reshape(-1).T * each_vector)
    projected_result_biggest += (tmp_biggest - np.mean(tmp_biggest))/ n
    vectors_biggest[count_biggest] = each_vector
    count_biggest += 0


print("The New Data with 500 dimensional is : \n{}".format(np.matmul(ve
print("The New Data with 300 dimensional is : \n{}".format(np.matmul(ve
print("The New Data with 100 dimensional is : \n{}".format(np.matmul(ve
print("The New Data with 50 dimensional is : \n{}".format(np.matmul(vec
print("The New Data with 10 dimensional is : \n{}".format(np.matmul(vec
print("The New Data with 2 dimensional is : \n{}".format(np.matmul(vect
print("The New Data with biggest dimensional is : \n{}".format(np.matmu

fig = plt.figure(figsize=(20,3))
fig.suptitle('Showing Comparaison between Decoding Images and Original

ax = fig.add_subplot(1, 8, 1)
ax.set_title("With 500 Dimensional")
ax.imshow(projected_result_500.reshape(28,28), cmap='binary')
ax = fig.add_subplot(1, 8, 2)
ax.set_title("With 300 Dimensional")
ax.imshow(projected_result_300.reshape(28,28), cmap='binary')
ax = fig.add_subplot(1, 8, 3)
ax.set_title("With 100 Dimensional")
ax.imshow(projected_result_100.reshape(28,28), cmap='binary')
ax = fig.add_subplot(1, 8, 4)
ax.set_title("With 50 Dimensional")
ax.imshow(projected_result_50.reshape(28,28), cmap='binary')
ax = fig.add_subplot(1, 8, 5)
ax.set_title("With 10 Dimensional")
ax.imshow(projected_result_10.reshape(28,28), cmap='binary')
ax = fig.add_subplot(1, 8, 6)
ax.set_title("With 2 Dimensional")
ax.imshow(projected_result_2.reshape(28,28), cmap='binary')
ax = fig.add_subplot(1, 8, 7)
ax.set_title("With biggest Dimensional")
ax.imshow(projected_result_biggest.reshape(28,28), cmap='binary')
ax = fig.add_subplot(1, 8, 8)
ax.set_title("Original Image")
ax.imshow(random_image.reshape(28,28), cmap='binary')
```

```
nel_launcher.py:50: ComplexWarning: Casting complex values to real discar
ds the imaginary part
/Users/wangboren/.pyenv/versions/3.6.4/lib/python3.6/site-packages/ipyker
nel_launcher.py:56: ComplexWarning: Casting complex values to real discar
ds the imaginary part
/Users/wangboren/.pyenv/versions/3.6.4/lib/python3.6/site-packages/ipyker
nel_launcher.py:62: ComplexWarning: Casting complex values to real discar
ds the imaginary part
```

```
ds the imaginary part
/Users/wangboren/.pyenv/versions/3.6.4/lib/python3.6/site-packages/ipyker
nel_launcher.py:68: ComplexWarning: Casting complex values to real discar
ds the imaginary part
```

Out[24]: <matplotlib.image.AxesImage at 0x146cf8b70>



接著介紹第四題解法及過程，和相關概念：

首先第一題：

- 我們分別對 train / test 兩類資料依序走訪
- 每次走訪都會檢查這筆圖案是哪類(0~9)，接著放入該類別所建立的 numpy 之中，並且存在 /data/train 與 /data/test 中
- 此外，我也限制每個數字(Label)的資料不會超過 100 筆資料！

接著第二題：

- 我用老師的 Sample 將上述所儲存的資料，用 for-loop 將每張圖片畫出來，分別畫出 train 與 test

接著第三題：

- 接著我們為了讓數據以統一座標點為出法，利用全數據的平均值以及標準差，以零為基礎，正規化數據！

最後第四題：

- 我先用 'np.matmul(train_sample_normalize.T, train_sample_normalize)' 算出 covariance，也就是

    cov = $X^T \cdot X$

- 接著再用 numpy linear algebra 的 eig function 找出 eigen vectors / eigen values
- 接著依照其 eigen value 的大小，將前 500,300,100,50,10,2,1 大的 eigen vector 找出來
- 接著再依序用這些 vector ，利用 $X^T \cdot$vector ，算出距離後正規化，並相互疊加，重建圖形，並且與原圖比較

> 這題我們發現，利用前 N 大 eigen value 的 eigen vector 所重新建立的圖形，當 N 越多的時候，越能接近原圖。
> 但以最大的 eigen vector 重建時，也發現因為其影響力較大，所以重建時就已經很貼近原圖了
> 這也意味著，只需要前 N 大的 eigen vector，就能保留圖形的資訊了，達到降維效果！如上面所 print 出的 New Data

In [ ]:

```
 
```