In [1]:
```
# Use gammaln for stability
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from scipy.io import loadmat
from scipy.special import digamma, gammaln
```

In [2]:
```
data_1 = loadmat('hw3_data_mat/data1.mat')
data_2 = loadmat('hw3_data_mat/data2.mat')
data_3 = loadmat('hw3_data_mat/data3.mat')
```

In [3]:
```
# Set prior parameters
a_0 = b_0 = 10e-16
e_0 = f_0 = 1
```

In [4]:
```
# Update q(lambda)
def update_q_lambda(n, e_0, f_0, mu, sigma, Y_T_Y, Y_T_X_T, X_X_T):
    e_p = 0.5 * n + e_0
    E_w_w_T = sigma + np.dot(mu, mu.T)
    f_p = 0.5 * (Y_T_Y - 2 * np.dot(Y_T_X_T, mu) + np.trace(np.dot(E_w_w
_T, X_X_T))) + f_0
    return e_p, float(f_p)
```

In [5]:
```
# Update q(alpha_k)
def update_q_alpha_k(a_0, b_0, mu, sigma, k):
    a_kp = 0.5 + a_0
    E_w_k_2 = mu[k]**2 + sigma[k, k]
    b_kp = 0.5 * E_w_k_2 + b_0
    return a_kp, b_kp
```

In [6]:
```
# Update q(w)
def update_q_w(e_t, f_t, a_t, b_t, X_X_T, X, Y):
    A_p = a_t / b_t
    E_lambda = e_t/float(f_t)
    M_p = E_lambda * X_X_T + np.diag(A_p)
    sigma_p = np.linalg.inv(M_p)
    mu_p = E_lambda * np.dot(np.dot(sigma_p, X.T), Y)
    return mu_p, sigma_p
```

In [7]:
```
# Compute L_1
def L_1(dim, a_t, b_t, mu_t, sigma_t):
    A_t = a_t / b_t
    E_A = np.diag(A_t)
    E_w_T_w = sigma_t + np.dot(mu_t, mu_t.T)
    E_ln_alpha = map(lambda a_b: digamma(a_b[0]) - np.log(a_b[1]), zip(a
_t, b_t))
    return - 0.5 * dim * np.log(2 * np.pi) + 0.5 * sum(E_ln_alpha) - 0.5
 * np.trace(np.dot(E_w_T_w, E_A))
```

```
In [8]:  # Compute L_2
         def L_2(e_0, f_0, e_t, f_t):
             return e_0 * np.log(f_0) - gammaln(e_0) + (e_0 -1) * (digamma(e_t) -
          np.log(f_t)) - f_0 * e_t/float(f_t)
```

```
In [9]:  # Compute L_3
         def L_3(a_0, b_0, a_t, b_t):
             return sum(map(lambda a_b: a_0 * np.log(b_0) - gammaln(a_0) + (a_0 -
          1) * (digamma(a_b[0]) - np.log(a_b[1])) \
                              - b_0 * a_b[0]/float(a_b[1]), zip(a_t, b_t)))
```

```
In [10]: # Compute L_4
         def L_4(dim, sigma_t):
             # Compute in logspace to prevent overflow
             sign, logdet_sigma_t = np.linalg.slogdet(sigma_t)
             # As 2 * np.pi * np.exp(1))** dim) is a constant and causes overflo
         w, remove
             return - 0.5 * (sign * logdet_sigma_t)
```

```
In [11]: # Compute L_5
         def L_5(e_t, f_t):
             return np.log(f_t) - gammaln(e_t) + (e_t - 1) * digamma(e_t) - e_t
```

```
In [12]: # Compute L_6
         def L_6(a_t, b_t):
             return sum(map(lambda a_b: np.log(a_b[1]) - gammaln(a_b[0]) +
         (a_b[0] - 1) * digamma(a_b[0]) - a_b[0], zip(a_t, b_t)))
```

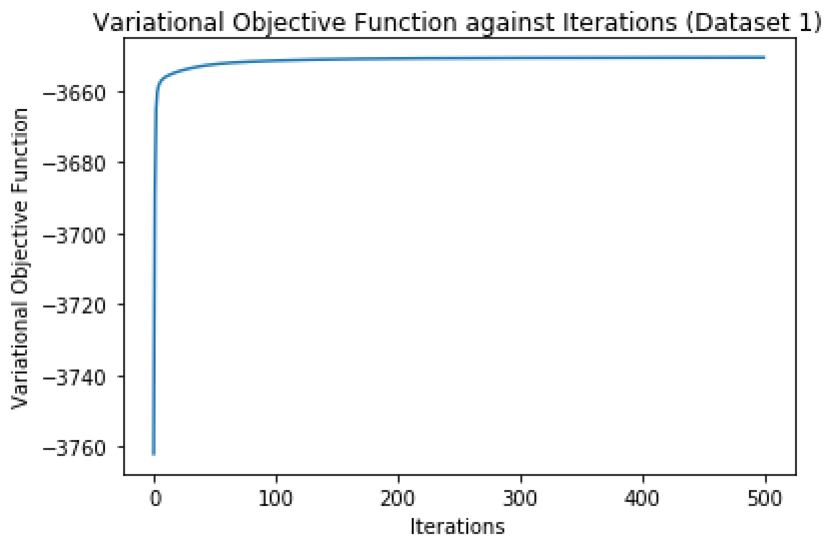
```
In [13]: # Compute L_7
         def L_7(n, e_t, f_t, mu_t, sigma_t, Y_T_Y, Y_T_X_T, X_X_T):
             E_ln_lambda = digamma(e_t) - np.log(f_t)
             E_lambda = e_t / float(f_t)
             E_w_T_w = np.dot(mu_t, mu_t.T) + sigma_t
             return 0.5 * n * (E_ln_lambda - np.log(2 * np.pi)) \
                    - 0.5 * E_lambda * (Y_T_Y - 2 * np.dot(Y_T_X_T, mu_t) + np.tr
         ace(np.dot(E_w_T_w, X_X_T)))
```

In [14]:
```
def variational_inference(X, y):
    # Get dimensions of X
    dim = X.shape[1]
    n = X.shape[0]

    # Calculate variables
    Y_T_Y = np.dot(y.T, y)[0][0]
    Y_T_X_T = np.dot(y.T, X)
    X_X_T = np.dot(X.T, X)

    # Initialise variables
    a_t = np.array([a_0] * dim, dtype='float64')
    b_t = np.array([b_0] * dim, dtype='float64')
    e_t = e_0
    f_t = f_0
    mu_t = np.zeros(dim, dtype='float64')
    sigma_t = np.diag(np.ones(dim, dtype='float64'))
    L = []
    L_1_t = L_2_t = L_3_t = L_4_t = L_5_t = L_6_t = L_7_t = 0

    # Run VI algorithm
    for t in range(500):
        # print t
        e_t, f_t = update_q_lambda(n, e_0, f_0, mu_t, sigma_t, Y_T_Y, Y_
T_X_T, X_X_T)
        a_p = []
        b_p = []
        for k in range(dim):
            a_kp, b_kp = update_q_alpha_k(a_0, b_0, mu_t, sigma_t, k)
            a_p.append(a_kp)
            b_p.append(b_kp)
        a_t = np.array(a_p)
        b_t = np.array(b_p).flatten()
        mu_t, sigma_t = update_q_w(e_t, f_t, a_t, b_t, X_X_T, X, y)
        L_1_t = L_1(dim, a_t, b_t, mu_t, sigma_t)
        L_2_t = L_2(e_0, f_0, e_t, f_t)
        L_3_t = L_3(a_0, b_0, a_t, b_t)
        L_4_t = L_4(dim, sigma_t)
        L_5_t = L_5(e_t, f_t)
        L_6_t = L_6(a_t, b_t)
        L_7_t = L_7(n, e_t, f_t, mu_t, sigma_t, Y_T_Y, Y_T_X_T, X_X_T)
        # print L_1_t, L_2_t, L_3_t, L_4_t, L_5_t, L_6_t, L_7_t
        L.append((L_1_t + L_2_t + L_3_t + L_7_t - L_4_t - L_5_t - L_6_t)
[0][0])

    return L, a_t, b_t, e_t, f_t, mu_t, sigma_t, dim
```

In [15]:
```
LL_1, a_1, b_1, e_1, f_1, mu_1, sigma_1, dim_1 = variational_inference(d
ata_1['X'], data_1['y'])
```

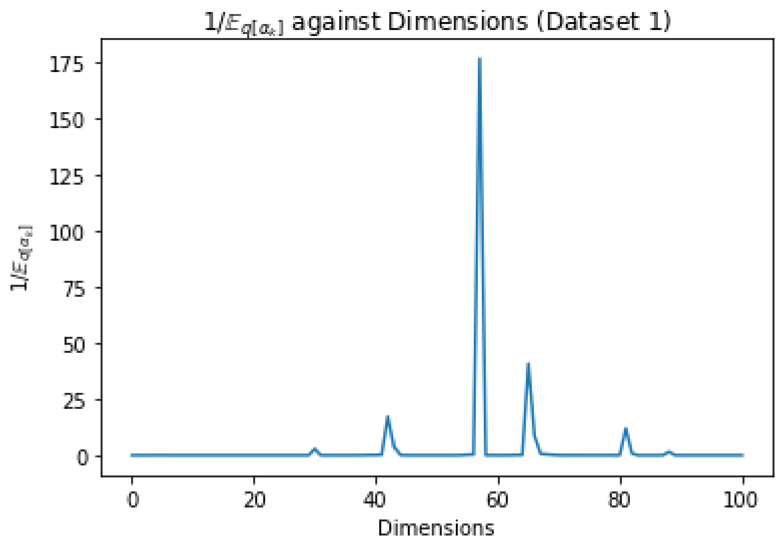
```
In [16]: plt.plot(range(500), LL_1)
         plt.xlabel('Iterations')
         plt.ylabel('Variational Objective Function')
         plt.title('Variational Objective Function against Iterations (Dataset
          1)')
```

```
Out[16]: Text(0.5,1,u'Variational Objective Function against Iterations (Dataset
          1)')
```



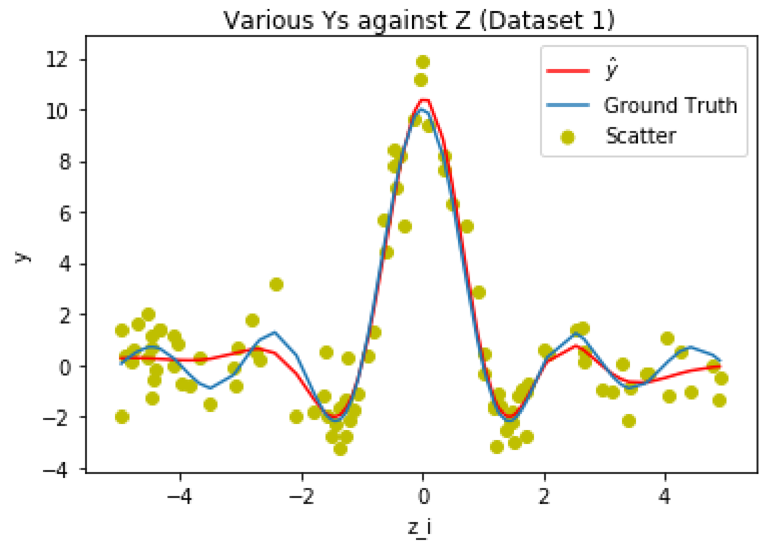
```
In [17]: plt.plot(range(dim_1), 1/(a_1/b_1))
         plt.xlabel('Dimensions')
         plt.ylabel(r'$1/\mathbb{E}_{q[\alpha_k]}$')
         plt.title(r'$1/\mathbb{E}_{q[\alpha_k]}$ against Dimensions (Dataset
         1)')
```

```
Out[17]: Text(0.5,1,u'$1/\\mathbb{E}_{q[\\alpha_k]}$ against Dimensions (Dataset
          1)')
```

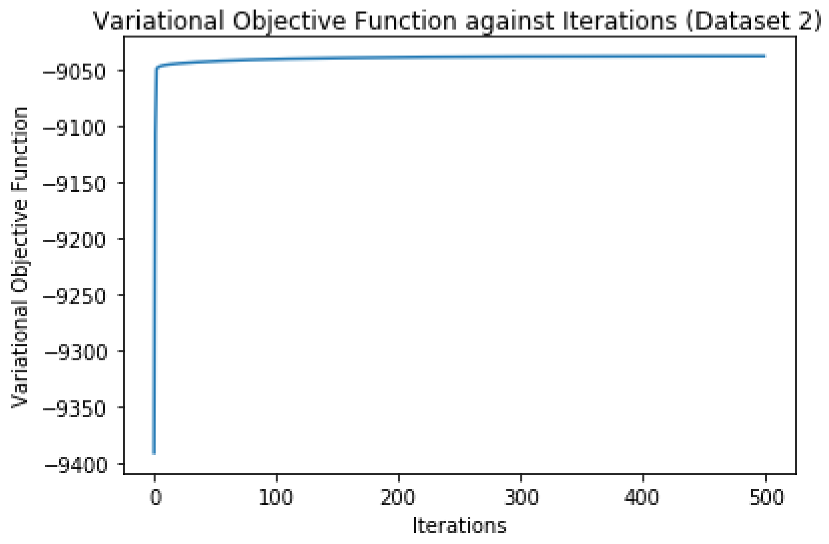


```
In [18]: print '1/E_q[lambda] (Dataset 1) =', f_1/e_1

         1/E_q[lambda] (Dataset 1) = 1.08004906537
```

```
In [19]: y_hat = np.dot(data_1['X'], mu_1)
         plt.plot(data_1['z'], y_hat, 'r', label=r'$\hat{y}$')
         plt.scatter(data_1['z'], data_1['y'], c='y', label='Scatter')
         plt.plot(data_1['z'], 10 * np.sinc(data_1['z']), label='Ground Truth')
         plt.legend()
         plt.xlabel('z_i')
         plt.ylabel('y')
         plt.title('Various Ys against Z (Dataset 1)')

Out[19]: Text(0.5,1,u'Various Ys against Z (Dataset 1)')
```

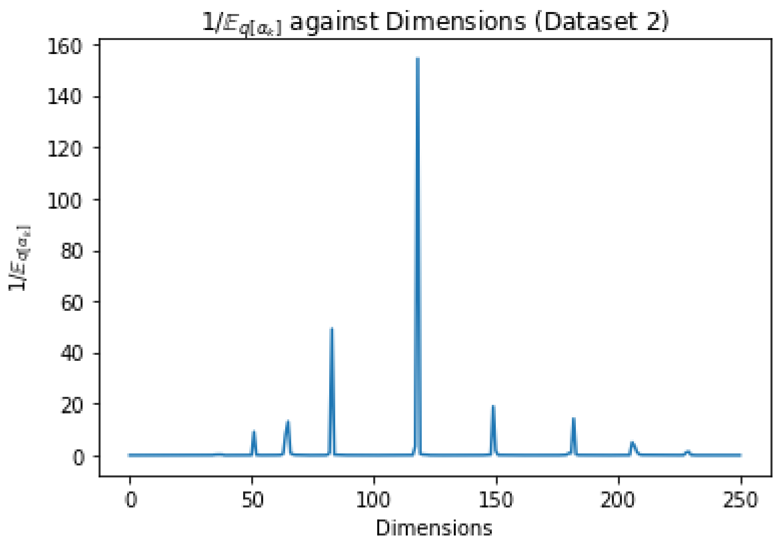


```
In [20]: LL_2, a_2, b_2, e_2, f_2, mu_2, sigma_2, dim_2 = variational_inference(d
         ata_2['X'].astype('float64'), data_2['y'].astype('float64'))

In [21]: plt.plot(range(500), LL_2)
         plt.xlabel('Iterations')
         plt.ylabel('Variational Objective Function')
         plt.title('Variational Objective Function against Iterations (Dataset
          2)')

Out[21]: Text(0.5,1,u'Variational Objective Function against Iterations (Dataset
          2)')
```



```
In [22]: plt.plot(range(dim_2), 1/(a_2/b_2))
         plt.xlabel('Dimensions')
         plt.ylabel(r'$1/\mathbb{E}_{q[\alpha_k]}$')
         plt.title(r'$1/\mathbb{E}_{q[\alpha_k]}$ against Dimensions (Dataset
         2)')
```

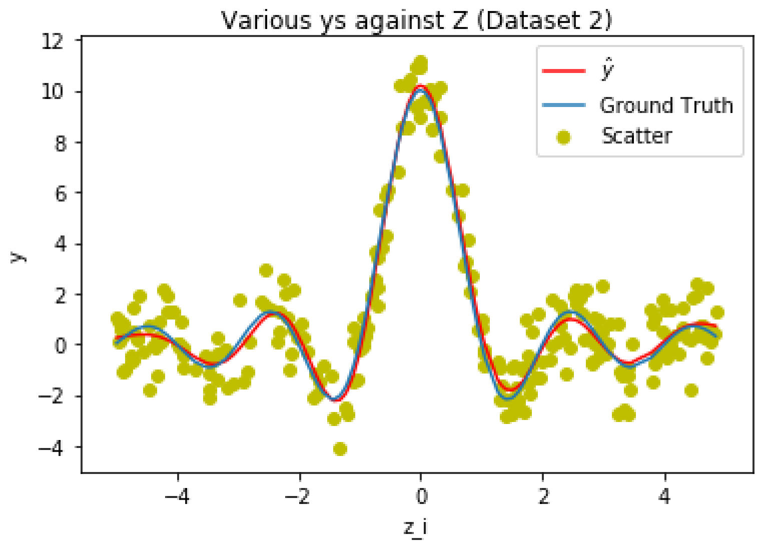
```
Out[22]: Text(0.5,1,u'$1/\\mathbb{E}_{q[\\alpha_k]}$ against Dimensions (Dataset
          2)')
```



```
In [23]: print '1/E_q[lambda] (Dataset 2) =', f_2/e_2
```

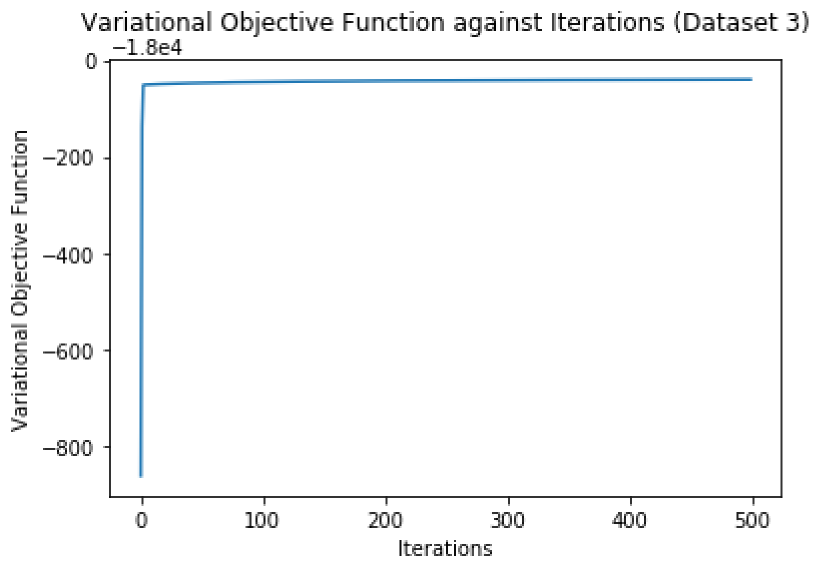
```
1/E_q[lambda] (Dataset 2) = 0.89944378672
```

```
In [24]: y_hat = np.dot(data_2['X'], mu_2)
         plt.plot(data_2['z'], y_hat, 'r', label=r'$\hat{y}$')
         plt.scatter(data_2['z'], data_2['y'], c='y', label='Scatter')
         plt.plot(data_2['z'], 10 * np.sinc(data_2['z']), label='Ground Truth')
         plt.legend()
         plt.xlabel('z_i')
         plt.ylabel('y')
         plt.title('Various ys against Z (Dataset 2)')

Out[24]: Text(0.5,1,u'Various ys against Z (Dataset 2)')
```

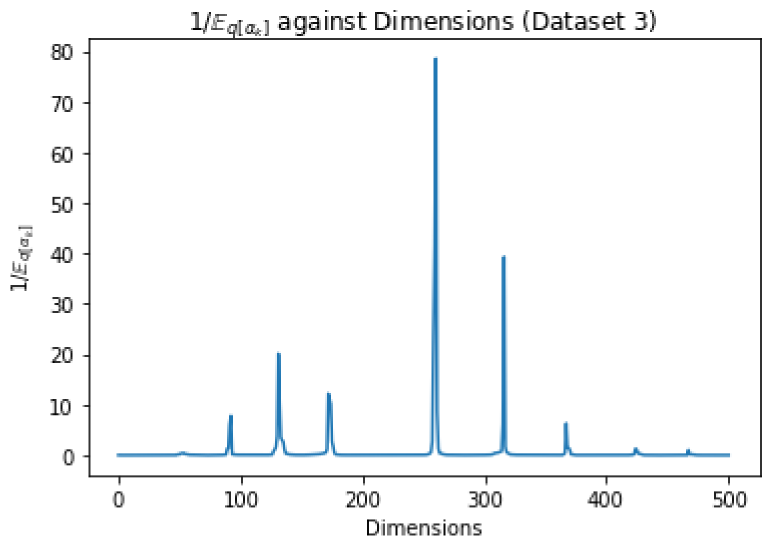


```
In [25]: LL_3, a_3, b_3, e_3, f_3, mu_3, sigma_3, dim_3 = variational_inference(d
         ata_3['X'].astype('float64'), data_3['y'].astype('float64'))
```

```
In [26]: plt.plot(range(500), LL_3)
         plt.xlabel('Iterations')
         plt.ylabel('Variational Objective Function')
         plt.title('Variational Objective Function against Iterations (Dataset
          3)', y=1.05)

Out[26]: Text(0.5,1.05,u'Variational Objective Function against Iterations (Data
         set 3)')
```



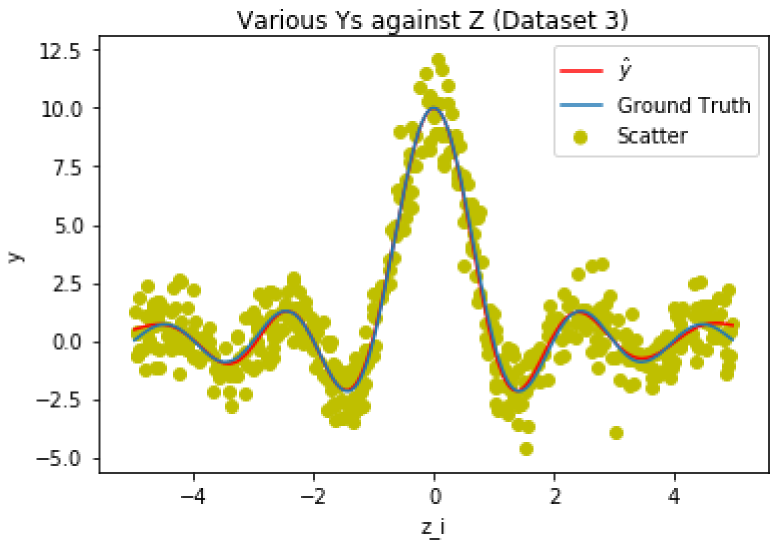
```
In [27]: plt.plot(range(dim_3), 1/(a_3/b_3))
         plt.xlabel('Dimensions')
         plt.ylabel(r'$1/\mathbb{E}_{q[\alpha_k]}$')
         plt.title(r'$1/\mathbb{E}_{q[\alpha_k]}$ against Dimensions (Dataset
         3)')

Out[27]: Text(0.5,1,u'$1/\\mathbb{E}_{q[\\alpha_k]}$ against Dimensions (Dataset
          3)')
```



```
In [28]: print '1/E_q[lambda] (Dataset 3) =', f_3/e_3

         1/E_q[lambda] (Dataset 3) = 0.978150754147
```

```
In [29]: y_hat = np.dot(data_3['X'], mu_3)
         plt.plot(data_3['z'], y_hat, 'r', label=r'$\hat{y}$')
         plt.scatter(data_3['z'], data_3['y'], c='y', label='Scatter')
         plt.plot(data_3['z'], 10 * np.sinc(data_3['z']), label='Ground Truth')
         plt.legend()
         plt.xlabel('z_i')
         plt.ylabel('y')
         plt.title('Various Ys against Z (Dataset 3)')
```

```
Out[29]: Text(0.5,1,u'Various Ys against Z (Dataset 3)')
```


Various Ys against Z (Dataset 3)
$\hat{y}$
Ground Truth
Scatter
y
z_i