In [1]:

```
%matplotlib inline

import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
```

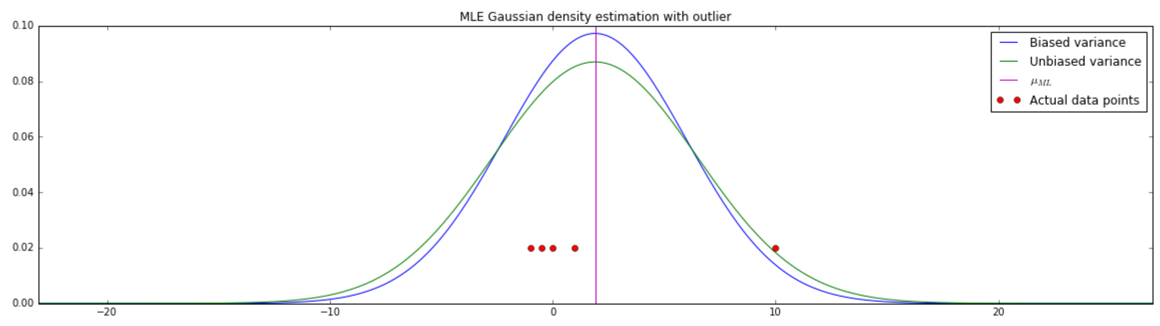
Problem 1

Maximum Likelihood Estimation

In [2]:

```
data_points = np.array([-1, 1, 10, -0.5, 0])

mu_ML = np.sum(data_points)/data_points.shape[0]
sigma2_ML = (np.sum(np.power(np.subtract(data_points, mu_ML),2)))/data_points.shape[0]
S2_ML = (np.sum(np.power(np.subtract(data_points, mu_ML),2)))/(data_points.shape[0]-1)

print "mean: " + str(mu_ML)
print "biased variance^2: " + str(sigma2_ML)
print "unbiased variance^2: " + str(S2_ML)

x = np.linspace(mu_ML-25, mu_ML+25, 300)

fig = plt.figure(figsize=(20,5))
plt.plot(x, mlab.normpdf(x, mu_ML, np.sqrt(sigma2_ML)), label="Biased variance")
plt.plot(x, mlab.normpdf(x, mu_ML, np.sqrt(S2_ML)), label="Unbiased variance")
plt.axvline(x=mu_ML, color="m", label="$\mu_{ML}$")

y_for_data = [np.average(mlab.normpdf(x, mu_ML, np.sqrt(S2_ML)))]*data_points.shape[0]
plt.plot(data_points, y_for_data, "ro", label = "Actual data points")
plt.legend()
plt.xlim(mu_ML-25, mu_ML+25)

plt.title("MLE Gaussian density estimation with outlier")
plt.show()
```

```
mean: 1.9
biased variance^2: 16.84
unbiased variance^2: 21.05
```

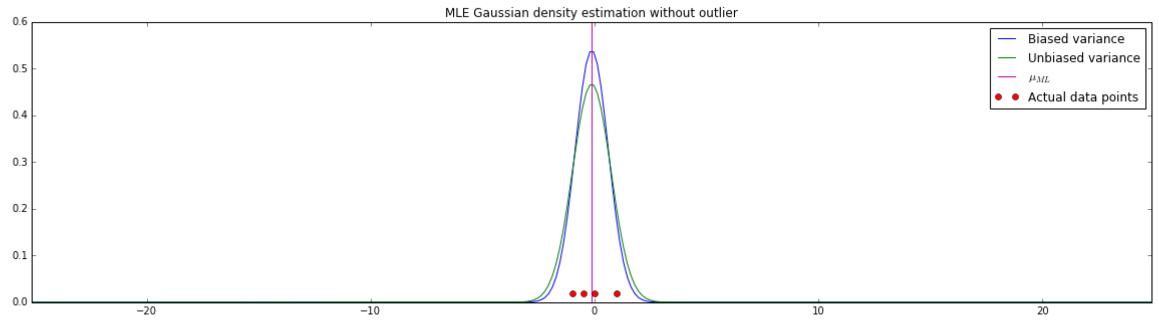


Let's remove the outlier now

In [3]:

```
data_points = np.array([-1, 1, -0.5, 0])

mu_ML = np.sum(data_points)/data_points.shape[0]
sigma2_ML = (np.sum(np.power(np.subtract(data_points, mu_ML),2)))/data_points.shape[0]
S2_ML = (np.sum(np.power(np.subtract(data_points, mu_ML),2)))/(data_points.shape[0]-1)

print "mean: " + str(mu_ML)
print "biased variance^2: " + str(sigma2_ML)
print "unbiased variance^2: " + str(S2_ML)

x = np.linspace(mu_ML-25, mu_ML+25, 300)

fig = plt.figure(figsize=(20,5))
plt.plot(x, mlab.normpdf(x, mu_ML, np.sqrt(sigma2_ML)), label="Biased variance")
plt.plot(x, mlab.normpdf(x, mu_ML, np.sqrt(S2_ML)), label="Unbiased variance")
plt.axvline(x=mu_ML, color="m", label="$\mu_{ML}$")

y_for_data = [np.average(mlab.normpdf(x, mu_ML, S2_ML))]*data_points.shape[0]
plt.plot(data_points, y_for_data, "ro", label = "Actual data points")
plt.legend()
plt.xlim(mu_ML-25, mu_ML+25)

plt.title("MLE Gaussian density estimation without outlier")
plt.show()
```

```
mean: -0.125
biased variance^2: 0.546875
unbiased variance^2: 0.729166666667
```

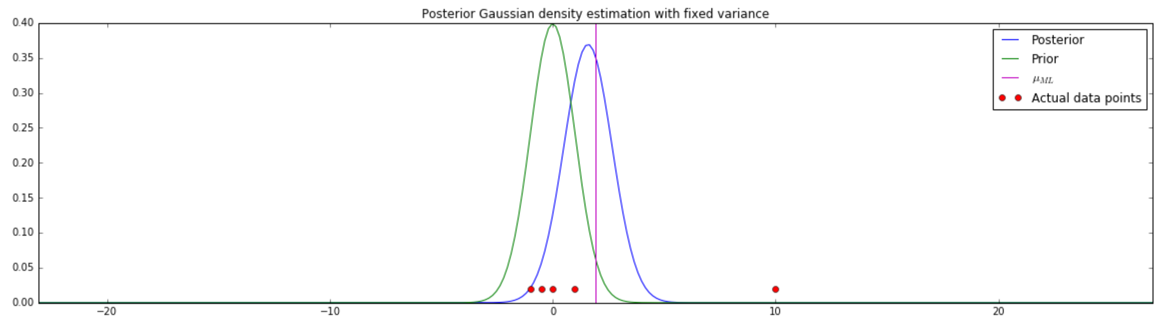


Now, fix the variance and slap a prior on the mean

In [4]:

```
data_points = np.array([-1, 1, 10, -0.5, 0])

N = data_points.shape[0]
mu_ML = np.sum(data_points)/N


sigma2_data = 1.
sigma2_prior = 1.
mu_prior = 0.

sigma2_n = np.divide(1., np.divide(N, sigma2_data) + np.divide(1., sigma2_prior))
mu_n = np.multiply(sigma2_n, np.divide(mu_prior, sigma2_prior) + np.divide(np.multiply(N
 mu_ML), sigma2_data))

print "mean: " + str(mu_n)
print "variance^2: " + str(sigma2_n)

x = np.linspace(mu_ML-25, mu_ML+25, 300)

fig = plt.figure(figsize=(20,5))
plt.plot(x, mlab.normpdf(x, mu_n, np.sqrt(sigma2_n + sigma2_data)), label="Posterior")
plt.plot(x, mlab.normpdf(x, mu_prior, np.sqrt(sigma2_prior)), label="Prior")
plt.axvline(x=mu_ML, color="m", label="$\mu_{ML}$")

y_for_data = [np.average(mlab.normpdf(x, mu_ML, S2_ML))]*data_points.shape[0]
plt.plot(data_points, y_for_data, "ro", label = "Actual data points")
plt.legend()
plt.xlim(mu_ML-25, mu_ML+25)

plt.title("Posterior Gaussian density estimation with fixed variance")
plt.show()
```

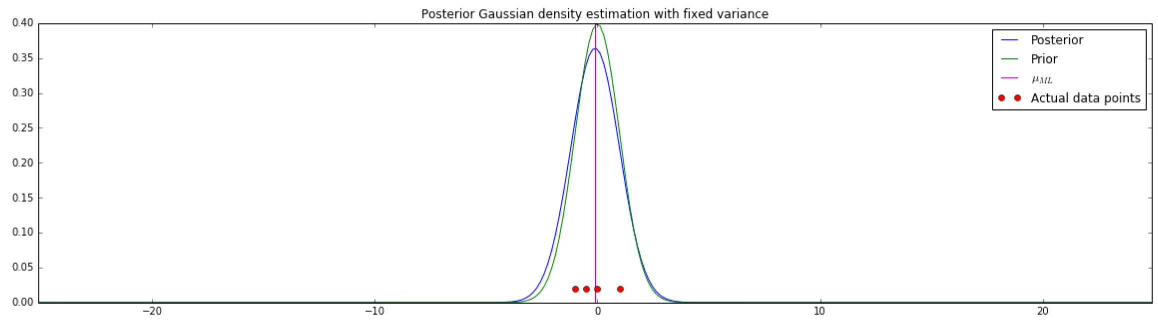
```
mean: 1.58333333333
variance^2: 0.166666666667
```



```
In [5]:
```

```python
data_points = np.array([-1, 1, -0.5, 0])

N = data_points.shape[0]
mu_ML = np.sum(data_points)/N


sigma2_data = 1.
sigma2_prior = 1.
mu_prior = 0.

sigma2_n = np.divide(1., np.divide(N, sigma2_data) + np.divide(1., sigma2_prior))
mu_n = np.multiply(sigma2_n, np.divide(mu_prior, sigma2_prior) + np.divide(np.multiply(N
 mu_ML), sigma2_data))

print "mean: " + str(mu_n)
print "variance^2: " + str(sigma2_n)

x = np.linspace(mu_ML-25, mu_ML+25, 300)

fig = plt.figure(figsize=(20,5))
plt.plot(x, mlab.normpdf(x, mu_n, np.sqrt(sigma2_n + sigma2_data)), label="Posterior")
plt.plot(x, mlab.normpdf(x, mu_prior, np.sqrt(sigma2_prior)), label="Prior")
plt.axvline(x=mu_ML, color="m", label="$\mu_{ML}$")

y_for_data = [np.average(mlab.normpdf(x, mu_ML, S2_ML))]*data_points.shape[0]
plt.plot(data_points, y_for_data, "ro", label = "Actual data points")
plt.legend()
plt.xlim(mu_ML-25, mu_ML+25)

plt.title("Posterior Gaussian density estimation with fixed variance")
plt.show()
```

```
mean: -0.1
variance^2: 0.2
```

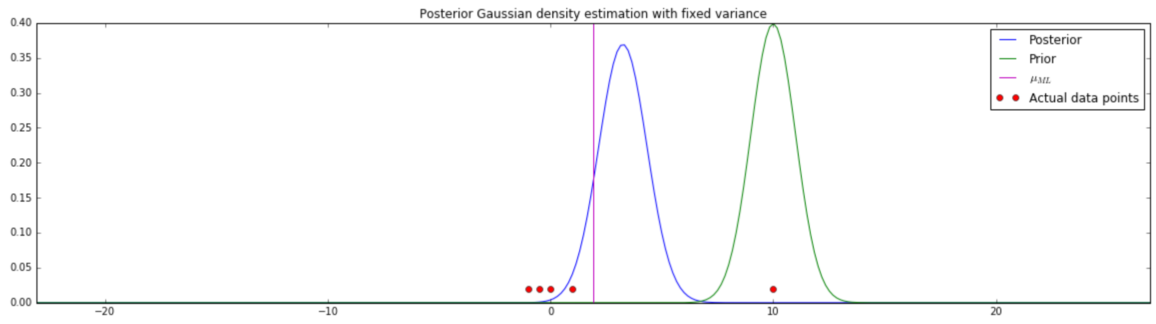


What if we choose the prior as $\mathcal{N}(10, 1)$?

In [6]:

```
data_points = np.array([-1, 1, 10, -0.5, 0])

N = data_points.shape[0]
mu_ML = np.sum(data_points)/N


sigma2_data = 1.
sigma2_prior = 1.
mu_prior = 10.

sigma2_n = np.divide(1., np.divide(N, sigma2_data) + np.divide(1., sigma2_prior))
mu_n = np.multiply(sigma2_n, np.divide(mu_prior, sigma2_prior) + np.divide(np.multiply(N
 mu_ML), sigma2_data))

print "mean: " + str(mu_n)
print "variance^2: " + str(sigma2_n)

x = np.linspace(mu_ML-25, mu_ML+25, 300)

fig = plt.figure(figsize=(20,5))
plt.plot(x, mlab.normpdf(x, mu_n, np.sqrt(sigma2_n + sigma2_data)), label="Posterior")
plt.plot(x, mlab.normpdf(x, mu_prior, np.sqrt(sigma2_prior)), label="Prior")
plt.axvline(x=mu_ML, color="m", label="$\mu_{ML}$")

y_for_data = [np.average(mlab.normpdf(x, mu_ML, S2_ML))]*data_points.shape[0]
plt.plot(data_points, y_for_data, "ro", label = "Actual data points")
plt.legend()
plt.xlim(mu_ML-25, mu_ML+25)

plt.title("Posterior Gaussian density estimation with fixed variance")
plt.show()
```

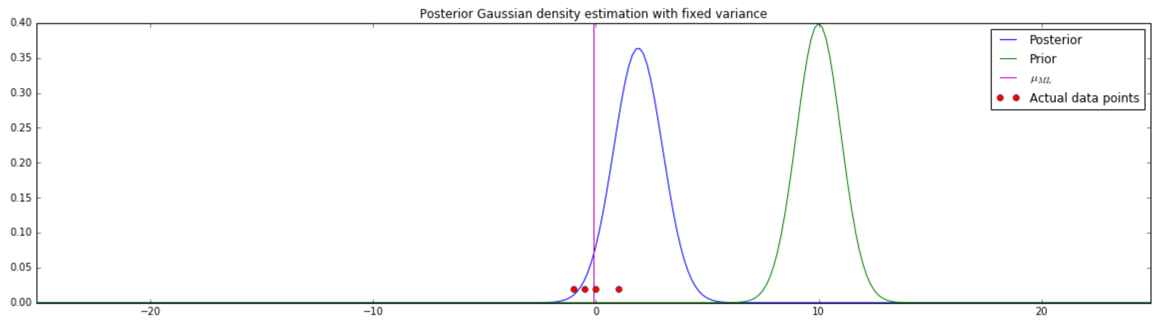
```
mean: 3.25
variance^2: 0.166666666667
```



In [7]:

```
data_points = np.array([-1, 1, -0.5, 0])

N = data_points.shape[0]
mu_ML = np.sum(data_points)/N


sigma2_data = 1.
sigma2_prior = 1.
mu_prior = 10.

sigma2_n = np.divide(1., np.divide(N, sigma2_data) + np.divide(1., sigma2_prior))
mu_n = np.multiply(sigma2_n, np.divide(mu_prior, sigma2_prior) + np.divide(np.multiply(N
 mu_ML), sigma2_data))

print "mean: " + str(mu_n)
print "variance^2: " + str(sigma2_n)

x = np.linspace(mu_ML-25, mu_ML+25, 300)

fig = plt.figure(figsize=(20,5))
plt.plot(x, mlab.normpdf(x, mu_n, np.sqrt(sigma2_n + sigma2_data)), label="Posterior")
plt.plot(x, mlab.normpdf(x, mu_prior, np.sqrt(sigma2_prior)), label="Prior")
plt.axvline(x=mu_ML, color="m", label="$\mu_{ML}$")

y_for_data = [np.average(mlab.normpdf(x, mu_ML, S2_ML))]*data_points.shape[0]
plt.plot(data_points, y_for_data, "ro", label = "Actual data points")
plt.legend()
plt.xlim(mu_ML-25, mu_ML+25)

plt.title("Posterior Gaussian density estimation with fixed variance")
plt.show()
```

```
mean: 1.9
variance^2: 0.2
```

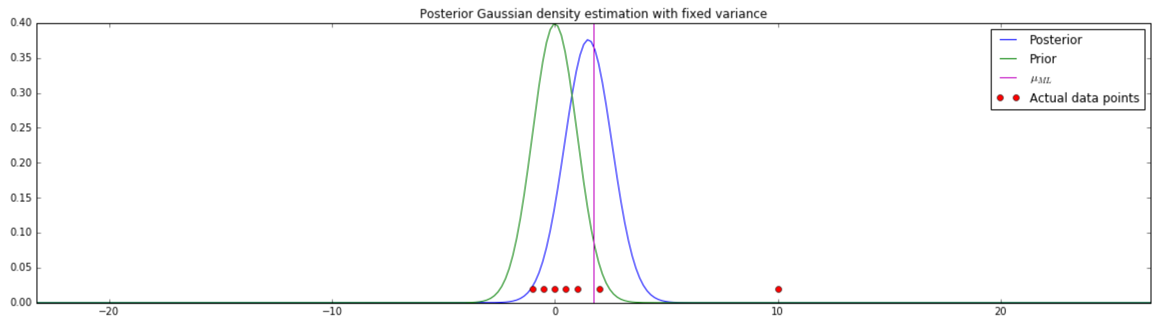


Ok, so that's that. What happens when we add two more datapoints?

In [8]:

```
data_points = np.array([-1, 1, 10, -0.5, 0, 2, 0.5])

N = data_points.shape[0]
mu_ML = np.sum(data_points)/N


sigma2_data = 1.
sigma2_prior = 1.
mu_prior = 0.

sigma2_n = np.divide(1., np.divide(N, sigma2_data) + np.divide(1., sigma2_prior))
mu_n = np.multiply(sigma2_n, np.divide(mu_prior, sigma2_prior) + np.divide(np.multiply(N
 mu_ML), sigma2_data))

print "mean: " + str(mu_n)
print "variance^2: " + str(sigma2_n)

x = np.linspace(mu_ML-25, mu_ML+25, 300)

fig = plt.figure(figsize=(20,5))
plt.plot(x, mlab.normpdf(x, mu_n, np.sqrt(sigma2_n + sigma2_data)), label="Posterior")
plt.plot(x, mlab.normpdf(x, mu_prior, np.sqrt(sigma2_prior)), label="Prior")
plt.axvline(x=mu_ML, color="m", label="$\mu_{ML}$")

y_for_data = [np.average(mlab.normpdf(x, mu_ML, S2_ML))]*data_points.shape[0]
plt.plot(data_points, y_for_data, "ro", label = "Actual data points")
plt.legend()
plt.xlim(mu_ML-25, mu_ML+25)

plt.title("Posterior Gaussian density estimation with fixed variance")
plt.show()
```

```
mean: 1.5
variance^2: 0.125
```



Problem 2

Maximum Likelihood Estimation

In this exercise, we perform maximum likelihood estimation (MLE) to find the parameters that fit the given polynomial (w) and the precision parameter for it (β). Refer to the slides on Bayesian Curve Fitting (https://users.soe.ucsc.edu/~vishy/fall2016/notes/CurveFitting.pdf)) for details on this.

The main thing to keep in mind is that we make use of an optimization package (here we use `curve_fit` from `scipy.optimize`) to find the $\mathbf{w}_{ML}$. The error function we need to minimize is given by (again, this is given in the slides):

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$

Then, β_{ML} can be then found by using the following expression:

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}_{ML}) - t_n\}^2$$

In [9]:

```
# first, we set our random seed and draw the x points at random from
np.random.seed(25)

# set number of data points
N = 10000

# we now extract the 1D data points from the uniform distribution
data = np.random.uniform(low=-100, high=100, size=N)

# the polynomial that generates the label for each point
f = lambda x: 0.1 + 2*x + x**2 + 3*x**3

# and finally, we populate the label array
label = np.array([])
for elem in data:
    label = np.append(label, np.random.normal(loc=f(elem), scale=1., size=1))

# the degree-3 polynomial we use to fit the data generated using f()
y_deg3 = lambda x, w0, w1, w2, w3: w0 + w1*x + w2*x**2 + w3*x**3

# minimize E(w) and obtain w_ML
w_ML, var = curve_fit(y_deg3, data, label)

# obtain beta_ML by plugging in w_ML into the equation give above
res_ML = y_deg3(data, w_ML[0], w_ML[1], w_ML[2], w_ML[3]) - label
beta_ML = float(N) / sum(res_ML**2)
print('N: %d' % N)
print('w_true: ' + str(np.array([0.1, 2., 1., 3.])))
print('w_ML:' + str(w_ML))
print('\nbeta_true: 1')
print('beta_ML: ' + str(beta_ML))
```

```
N: 10000
w_true: [ 0.1  2.   1.   3. ]
w_ML:[ 0.11169554  1.99990122  0.99999668  3.00000007]

beta_true: 1
beta_ML: 0.996677487071
```

Now, we use a higher-order polynomial form. Rest of the curve-fitting code remains the same as above.

In [10]:

```
# the degree-5 polynomial we use to fit the data generated using f()
y_deg5 = lambda x, w0, w1, w2, w3, w4, w5: w0 + w1*x + w2*x**2 + w3*x**3 + w4*x**4 + w5*
**5

# minimize E(w) and obtain w_ML
w_ML, var = curve_fit(y_deg5, data, label)

# obtain beta_ML by plugging in w_ML into the equation give above
res_ML = y_deg5(data, w_ML[0], w_ML[1], w_ML[2], w_ML[3], w_ML[4], w_ML[5]) - label
beta_ML = float(N) / sum(res_ML**2)
print('N: %d' % N)
print('w_true: ' + str(np.array([0.1, 2., 1., 3.])))
print('w_ML:' + str(w_ML))
print('\nbeta_true: 1')
print('beta_ML: ' + str(beta_ML))
```

```
N: 10000
w_true: [ 0.1  2.   1.   3. ]
w_ML:[  1.11446716e-01   1.99887661e+00   9.99997002e-01   3.00000055e+00
  -3.74965965e-11  -4.32771776e-11]

beta_true: 1
beta_ML: 0.996945980194
```

Bayesian Regression

For this exercise, we do the same thing as before. We first generate the data, and the corresponding labels. Then we use this data to compute the variance and mean for data points in the $[0, 1]$ interval, and then we plot our results.

In the current implementation, we take advantage of the dot product function in numpy and we avoid computing dot products in a sum formalism.

In [11]:

```
# first, we set our random seed and draw the x points at random from
np.random.seed(25)

# we now extract the 1D data points from the uniform distribution
data = np.random.uniform(low=-100, high=100, size=100)

# the polynomial that generates the label for each point
f = lambda x: 0.1 + 2*x + x**2 + 3*x**3

# and finally, we populate the label array
label = np.array([])
for elem in data:
    label = np.append(label, np.random.normal(loc=f(elem), scale=1., size=1))
```

Next up, we define the function that generates the polynomial for us i.e. ϕ where $\phi_i(x) = x^i$.

In [12]:

```python
def phi(x, poly_degree):
    """ generate the polynomial vector """
    this_phi = np.zeros((poly_degree+1, x.shape[0]))
    for k in range(poly_degree+1):
        this_phi[k] = x**k
    return this_phi
```

In [13]:

```python
def gimme_the_S(alpha, beta, poly_degree, data):
    """ generate the matrix S """
    # initialize S as alpha*I
    S = np.multiply(alpha,
                    np.identity(poly_degree+1))

    # compute the kernelized data matrix Phi
    my_phi = phi(data, poly_degree)

    # compute the final version of S
    S = np.add(S, np.multiply(beta,
                              # this dot product is equivalent to
                              # the sum over all phi from the data
                              np.dot(my_phi,
                                     my_phi.transpose())))

    return np.linalg.inv(S)
```

In [14]:

```python
def gimme_s(beta, S, new_data):
    """ compute the variance s for regressing on a new data point """
    current_degree = S.shape[0] - 1
    new_phi = phi(new_data, current_degree)

    # again, instead of doing the sum notation,
    # we just do dot products
    s2 = 1./beta + np.dot(new_phi.transpose(),
                          np.dot(S,
                                 new_phi))

    return np.sqrt(s2)
```

In order to get a bit more clarity on the computational side, when computing the mean of the Gaussian for the new point that we do regression on, we break the process down into the following parts

$$m(x) = \beta\phi(x)^\top \mathbf{S} \sum_{n=1}^{N} \phi(x_n) t_n$$

which in matrix form becomes

$$m(x) = \beta \underbrace{\Phi^\top \underbrace{\mathbf{S} \underbrace{\Phi \mathbf{t}}_{a}}_{b}}_{c}$$

Note how, since we have a reasonable implementation of the function `phi(x, poly_degree)`, this function works both for a single data point by giving ϕ, as well as for a data matrix by giving Φ.

In [15]:

```
def gimme_m(beta, new_data_point, old_data, old_labels, S):
    """ compute the mean m for regressing on a new data point """
    transformed_point = phi(new_data_point,
                            S.shape[0]-1)

    a = np.dot(phi(old_data, S.shape[0]-1), old_labels)

    b = np.dot(S, a)

    c = np.dot(transformed_point.transpose(), b)

    return beta*c
```

Ok, so what did we get? Let's run this on our data and plot the results.

In [16]:

```
S=gimme_the_S(alpha=1, beta=1, poly_degree=3, data=data)
```

In [17]:

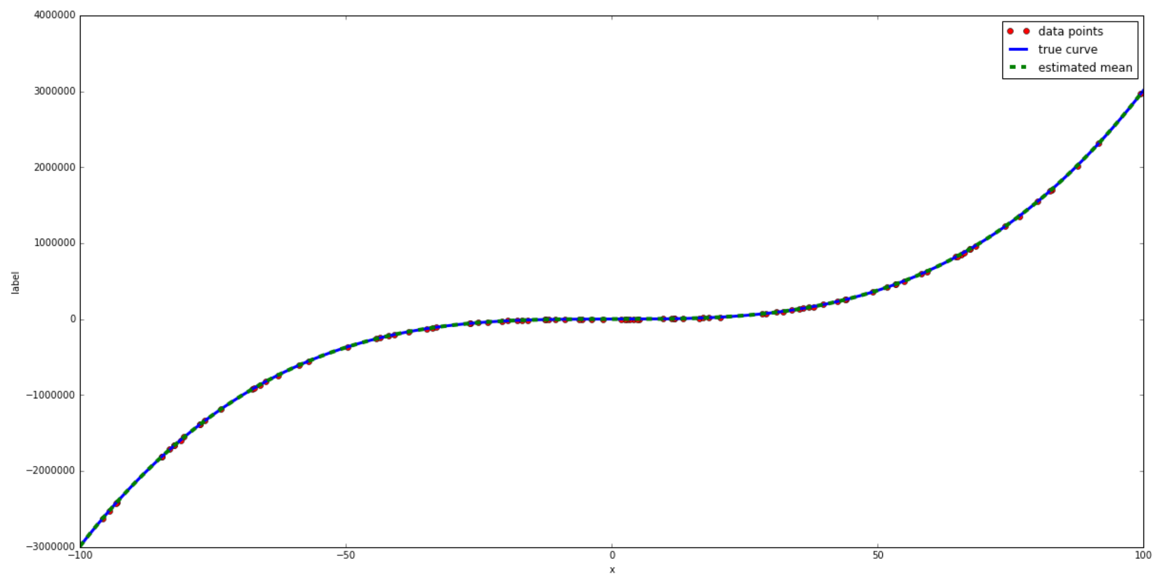
```
s = gimme_s(1, S, np.array([0]))

gimme_m(beta=1, new_data_point=np.array([0]), old_data = data, old_labels = label, S=S)
```

Out[17]:

```
array([-0.09016862])
```

In [18]:

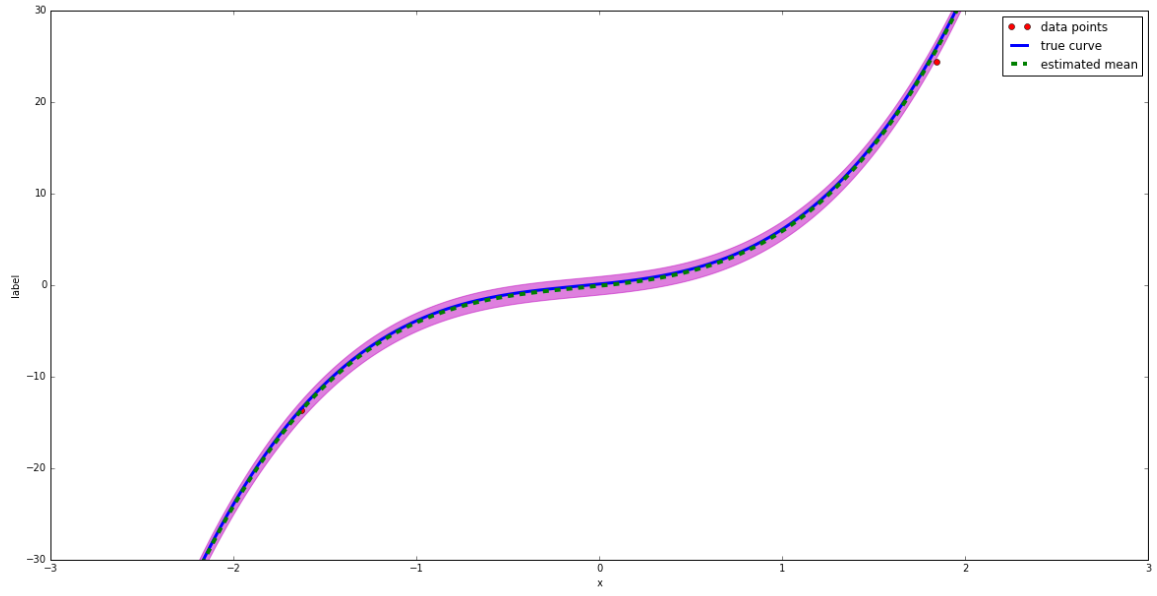
```
plotting_data = np.linspace(-100, 100, 1000)
true_labels = f(plotting_data)

estimated_mean, estimated_variance = np.array([]), np.array([])
for point in plotting_data:
    estimated_mean = np.append(estimated_mean,
                               gimme_m(beta=1,
                                       new_data_point=np.array([point]),
                                       old_data=data,
                                       old_labels=label,
                                       S=S))
    estimated_variance = np.append(estimated_variance,
                                   gimme_s(beta=1,
                                           S=S,
                                           new_data=np.array([point])))

# and now we plot
plt.figure(figsize=(20,10))
plt.plot(data, label, "ro", label="data points")
plt.plot(plotting_data, true_labels, "b", label="true curve", linewidth=3)
plt.plot(plotting_data, estimated_mean, "g",
         label="estimated mean", linestyle="--", linewidth=4)
plt.fill_between(plotting_data,
                 np.subtract(estimated_mean, estimated_variance),
                 np.add(estimated_mean, estimated_variance),
                 color="m", alpha=0.5)

plt.xlabel("x")
plt.ylabel("label")
plt.legend()
plt.show()
```



Well, we can't see much there. What if we just focused on the $[-10, 10]$ interval?

In [19]:

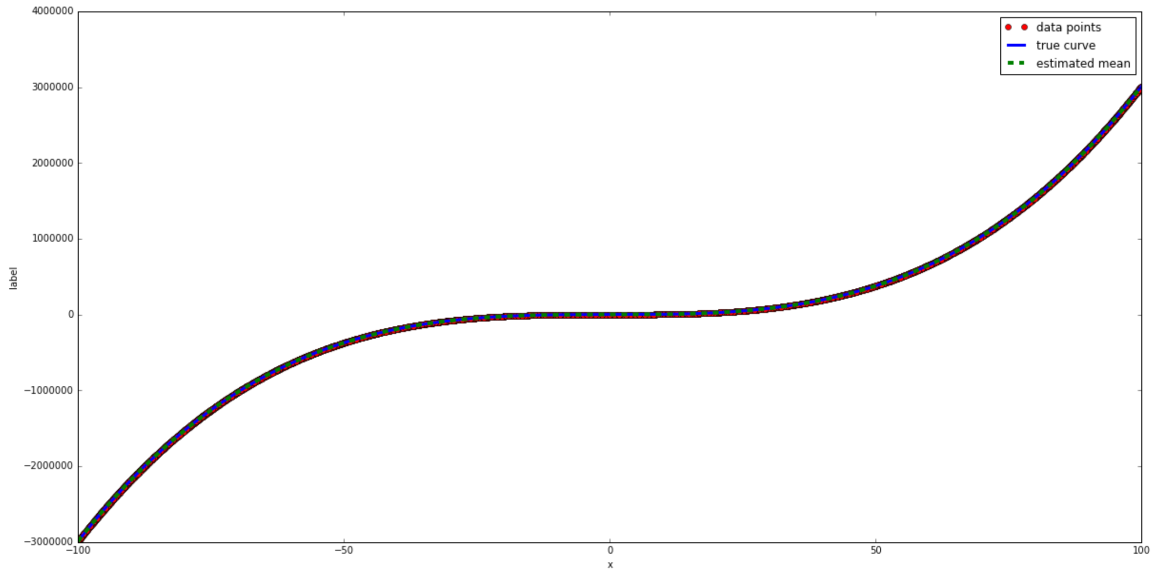
```
plotting_data = np.linspace(-10, 10, 1000)
true_labels = f(plotting_data)

estimated_mean, estimated_variance = np.array([]), np.array([])
for point in plotting_data:
    estimated_mean = np.append(estimated_mean,
                               gimme_m(beta=1,
                                       new_data_point=np.array([point]),
                                       old_data=data,
                                       old_labels=label,
                                       S=S))
    estimated_variance = np.append(estimated_variance,
                                   gimme_s(beta=1,
                                           S=S,
                                           new_data=np.array([point])))

# and now we plot
plt.figure(25, figsize=(20,10))
plt.plot(data, label, "ro", label="data points")
plt.plot(plotting_data, true_labels, "b", label="true curve",
         linewidth=3)
plt.plot(plotting_data, estimated_mean, "g",
         label="estimated mean", linestyle="--",
         linewidth=4)
plt.fill_between(plotting_data,
                 np.subtract(estimated_mean, estimated_variance),
                 np.add(estimated_mean, estimated_variance),
                 color="m", alpha=0.5)
plt.xlim(-3,3)
plt.ylim(-30,30)
plt.xlabel("x")
plt.ylabel("label")
plt.legend()
plt.show()
```



Let's do this exact same thing, but this time, let's try with 10000 data points, instead of just 100.

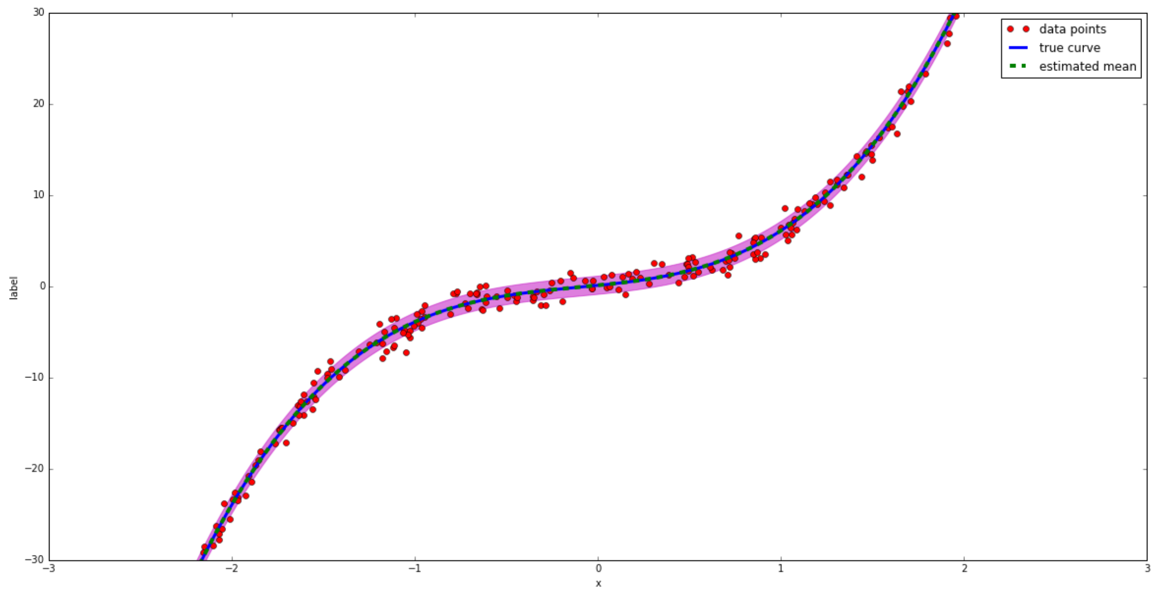
In [20]:

```
# generate new data
data = np.random.uniform(low=-100, high=100, size=10000)
label = np.array([])
for elem in data:
    label = np.append(label, np.random.normal(loc=f(elem), scale=1., size=1))

# We recompute the S matrix
S=gimme_the_S(alpha=1, beta=1, poly_degree=3, data=data)

# generate the plotting data
plotting_data = np.linspace(-100, 100, 1000)
true_labels = f(plotting_data)

estimated_mean, estimated_variance = np.array([]), np.array([])
for point in plotting_data:
    estimated_mean = np.append(estimated_mean,
                               gimme_m(beta=1,
                                       new_data_point=np.array([point]),
                                       old_data=data,
                                       old_labels=label,
                                       S=S))
    estimated_variance = np.append(estimated_variance,
                                   gimme_s(beta=1,
                                           S=S,
                                           new_data=np.array([point])))

# and now we plot
plt.figure(figsize=(20,10))
plt.plot(data, label, "ro", label="data points")
plt.plot(plotting_data, true_labels, "b", label="true curve", linewidth=3)
plt.plot(plotting_data, estimated_mean, "g",
         label="estimated mean", linestyle="--", linewidth=4)
plt.fill_between(plotting_data,
                 np.subtract(estimated_mean, estimated_variance),
                 np.add(estimated_mean, estimated_variance),
                 color="m", alpha=0.5)

plt.xlabel("x")
plt.ylabel("label")
plt.legend()
plt.show()
```



In [21]:

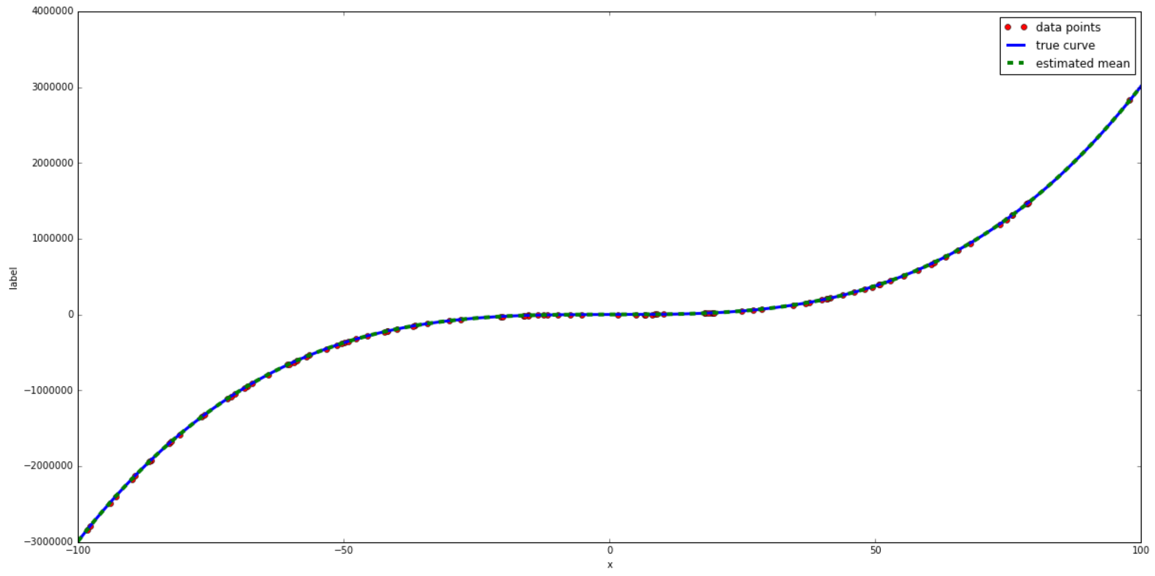
```
plotting_data = np.linspace(-10, 10, 10000)
true_labels = f(plotting_data)

estimated_mean, estimated_variance = np.array([]), np.array([])
for point in plotting_data:
    estimated_mean = np.append(estimated_mean,
                               gimme_m(beta=1,
                                       new_data_point=np.array([point]),
                                       old_data=data,
                                       old_labels=label,
                                       S=S))
    estimated_variance = np.append(estimated_variance,
                                   gimme_s(beta=1,
                                           S=S,
                                           new_data=np.array([point])))

# and now we plot
plt.figure(25, figsize=(20,10))
plt.plot(data, label, "ro", label="data points")
plt.plot(plotting_data, true_labels, "b", label="true curve",
         linewidth=3)
plt.plot(plotting_data, estimated_mean, "g",
         label="estimated mean", linestyle="--",
         linewidth=4)
plt.fill_between(plotting_data,
                 np.subtract(estimated_mean, estimated_variance),
                 np.add(estimated_mean, estimated_variance),
                 color="m", alpha=0.5)
plt.xlim(-3,3)
plt.ylim(-30,30)
plt.xlabel("x")
plt.ylabel("label")
plt.legend()
plt.show()
```



What happens when we set $\alpha = 100$?

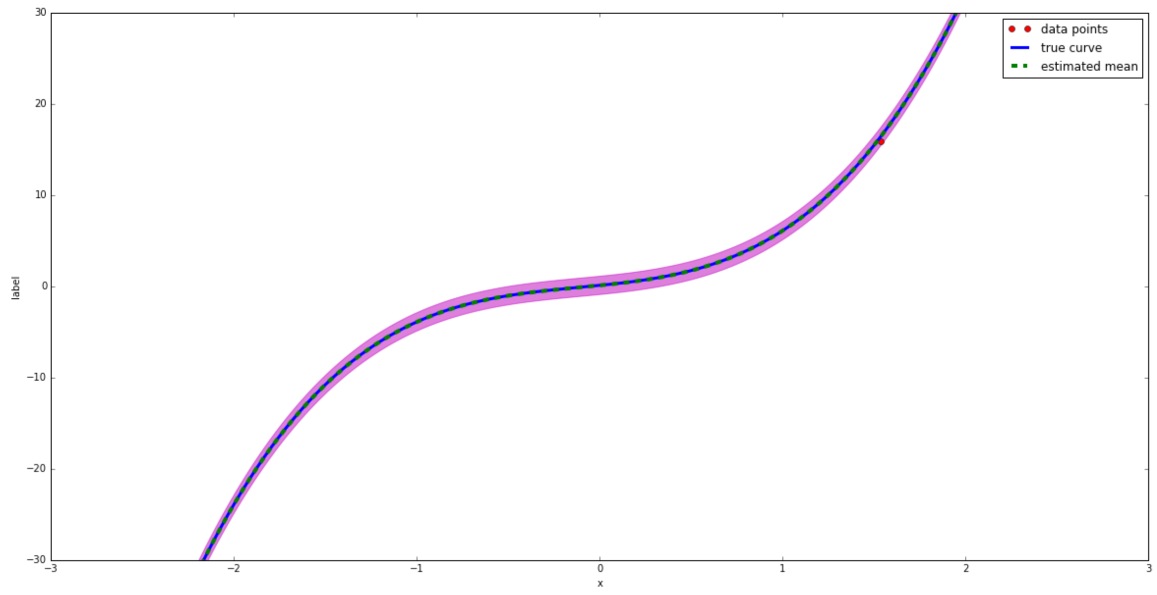
In [22]:

```
# generate new data
data = np.random.uniform(low=-100, high=100, size=100)
label = np.array([])
for elem in data:
    label = np.append(label, np.random.normal(loc=f(elem), scale=1., size=1))

# We recompute the S matrix
S=gimme_the_S(alpha=100, beta=1, poly_degree=3, data=data)

# generate the plotting data
plotting_data = np.linspace(-100, 100, 1000)
true_labels = f(plotting_data)

estimated_mean, estimated_variance = np.array([]), np.array([])
for point in plotting_data:
    estimated_mean = np.append(estimated_mean,
                               gimme_m(beta=1,
                                       new_data_point=np.array([point]),
                                       old_data=data,
                                       old_labels=label,
                                       S=S))
    estimated_variance = np.append(estimated_variance,
                                   gimme_s(beta=1,
                                           S=S,
                                           new_data=np.array([point])))

# and now we plot
plt.figure(figsize=(20,10))
plt.plot(data, label, "ro", label="data points")
plt.plot(plotting_data, true_labels, "b", label="true curve", linewidth=3)
plt.plot(plotting_data, estimated_mean, "g",
         label="estimated mean", linestyle="--", linewidth=4)
plt.fill_between(plotting_data,
                 np.subtract(estimated_mean, estimated_variance),
                 np.add(estimated_mean, estimated_variance),
                 color="m", alpha=0.5)

plt.xlabel("x")
plt.ylabel("label")
plt.legend()
plt.show()
```



In [23]:

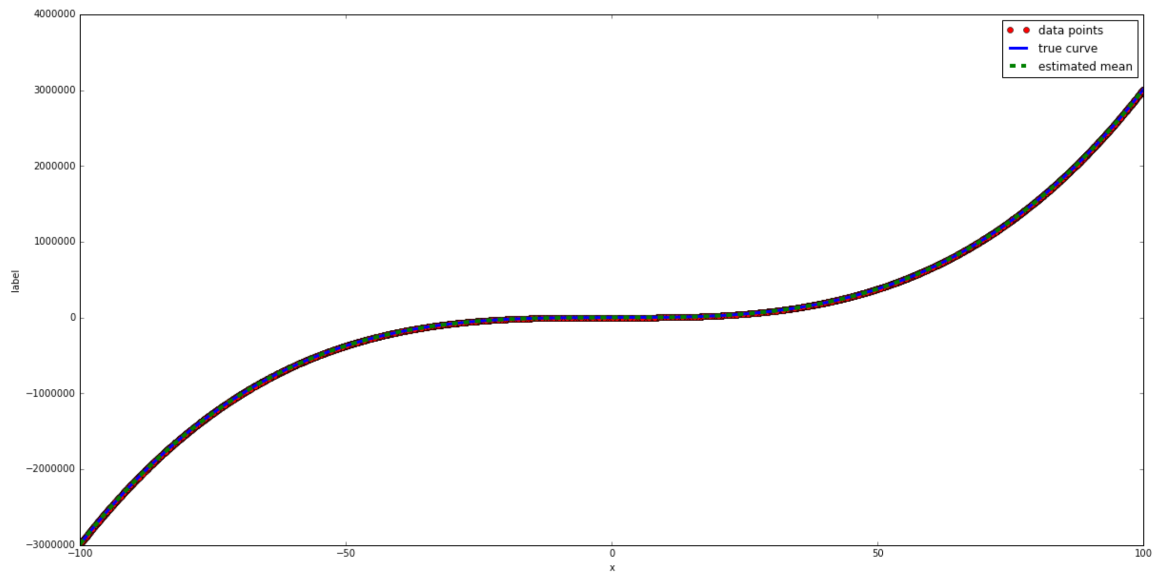
```
plotting_data = np.linspace(-10, 10, 1000)
true_labels = f(plotting_data)

estimated_mean, estimated_variance = np.array([]), np.array([])
for point in plotting_data:
    estimated_mean = np.append(estimated_mean,
                               gimme_m(beta=1,
                                       new_data_point=np.array([point]),
                                       old_data=data,
                                       old_labels=label,
                                       S=S))
    estimated_variance = np.append(estimated_variance,
                                   gimme_s(beta=1,
                                           S=S,
                                           new_data=np.array([point])))

# and now we plot
plt.figure(25, figsize=(20,10))
plt.plot(data, label, "ro", label="data points")
plt.plot(plotting_data, true_labels, "b", label="true curve",
         linewidth=3)
plt.plot(plotting_data, estimated_mean, "g",
         label="estimated mean", linestyle="--",
         linewidth=4)
plt.fill_between(plotting_data,
                 np.subtract(estimated_mean, estimated_variance),
                 np.add(estimated_mean, estimated_variance),
                 color="m", alpha=0.5)
plt.xlim(-3,3)
plt.ylim(-30,30)
plt.xlabel("x")
plt.ylabel("label")
plt.legend()
plt.show()
```



And now, let's increase the size of the sample to 10000.

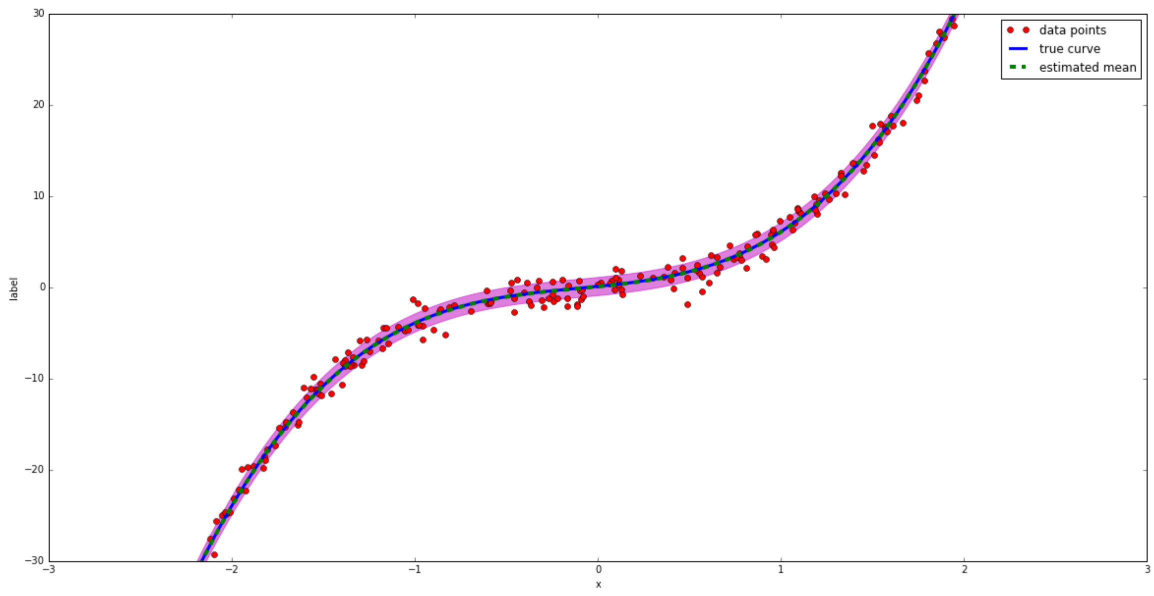
In [24]:

```
# generate new data
data = np.random.uniform(low=-100, high=100, size=10000)
label = np.array([])
for elem in data:
    label = np.append(label, np.random.normal(loc=f(elem), scale=1., size=1))

# We recompute the S matrix
S=gimme_the_S(alpha=100, beta=1, poly_degree=3, data=data)

# generate the plotting data
plotting_data = np.linspace(-100, 100, 1000)
true_labels = f(plotting_data)

estimated_mean, estimated_variance = np.array([]), np.array([])
for point in plotting_data:
    estimated_mean = np.append(estimated_mean,
                               gimme_m(beta=1,
                                       new_data_point=np.array([point]),
                                       old_data=data,
                                       old_labels=label,
                                       S=S))
    estimated_variance = np.append(estimated_variance,
                                   gimme_s(beta=1,
                                           S=S,
                                           new_data=np.array([point])))

# and now we plot
plt.figure(figsize=(20,10))
plt.plot(data, label, "ro", label="data points")
plt.plot(plotting_data, true_labels, "b", label="true curve", linewidth=3)
plt.plot(plotting_data, estimated_mean, "g",
         label="estimated mean", linestyle="--", linewidth=4)
plt.fill_between(plotting_data,
                 np.subtract(estimated_mean, estimated_variance),
                 np.add(estimated_mean, estimated_variance),
                 color="m", alpha=0.5)

plt.xlabel("x")
plt.ylabel("label")
plt.legend()
plt.show()
```



In [25]:

```
plotting_data = np.linspace(-10, 10, 10000)
true_labels = f(plotting_data)

estimated_mean, estimated_variance = np.array([]), np.array([])
for point in plotting_data:
    estimated_mean = np.append(estimated_mean,
                               gimme_m(beta=1,
                                       new_data_point=np.array([point]),
                                       old_data=data,
                                       old_labels=label,
                                       S=S))
    estimated_variance = np.append(estimated_variance,
                                   gimme_s(beta=1,
                                           S=S,
                                           new_data=np.array([point])))

# and now we plot
plt.figure(25, figsize=(20,10))
plt.plot(data, label, "ro", label="data points")
plt.plot(plotting_data, true_labels, "b", label="true curve",
         linewidth=3)
plt.plot(plotting_data, estimated_mean, "g",
         label="estimated mean", linestyle="--",
         linewidth=4)
plt.fill_between(plotting_data,
                 np.subtract(estimated_mean, estimated_variance),
                 np.add(estimated_mean, estimated_variance),
                 color="m", alpha=0.5)
plt.xlim(-3,3)
plt.ylim(-30,30)
plt.xlabel("x")
plt.ylabel("label")
plt.legend()
plt.show()
```


data points
true curve
estimated mean
label
x