Linear regression workbook

This workbook will walk you through a linear regression example. It will provide familiarity with Jupyter Notebook and Python. Please print (to pdf) a completed version of this workbook for submission with HW #1.

ECE 239AS, Winter Quarter 2018, Prof. J.C. Kao, TAs C. Zhang and T. Xing

In [7]:

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

#allows matlab plots to be generated in line
%matplotlib inline
```

Data generation

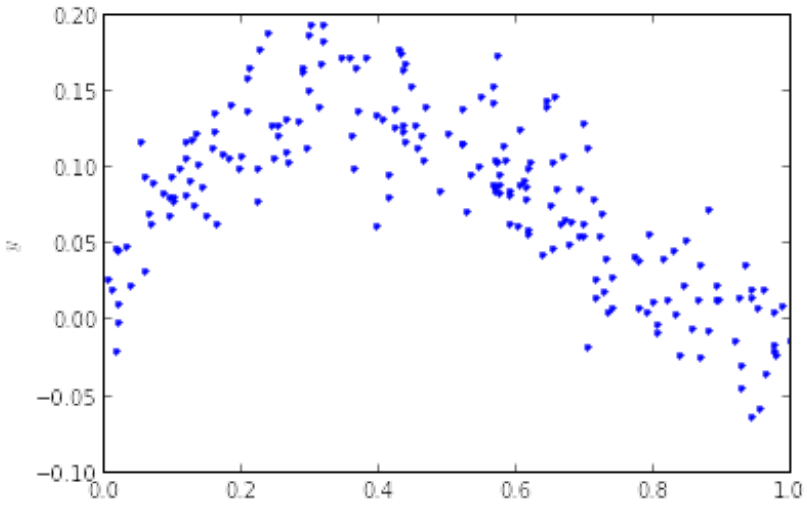
For any example, we first have to generate some appropriate data to use. The following cell generates data according to the model: $y = x - 2x^2 + x^3 + \epsilon$

In [8]:

```
np.random.seed(0)   # Sets the random seed.
num_train = 200      # Number of training data points

# Generate the training data
x = np.random.uniform(low=0, high=1, size=(num_train,))
y = x - 2*x**2 + x**3 + np.random.normal(loc=0, scale=0.03, size=(num_train,
))
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

Out[8]:

```
<matplotlib.text.Text at 0x1157a0d90>
```



QUESTIONS:

Write your answers in the markdown cell below this one:

(1) What is the generating distribution of x?

(2) What is the distribution of the additive noise ϵ?

ANSWERS:

(1) The generating distribution of x is a uniform distribution from 0 to 1.

(2) The distribution of the additive noise is a normal distribution.

Fitting data to the model (5 points)

Here, we'll do linear regression to fit the parameters of a model $y = ax + b$.

In [5]:

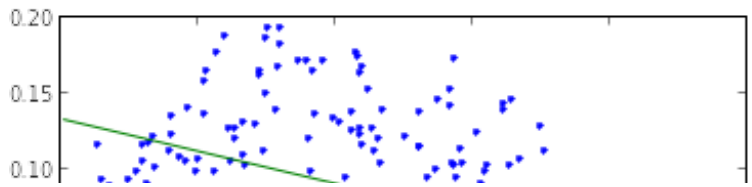
```
# xhat = (x, 1)
xhat = np.vstack((x, np.ones_like(x)))

# GOAL: create a variable theta; theta is a numpy array whose elements are
[a, b]

xhat_orig = np.transpose(xhat)
xhat_transposed = xhat
inv_mat = np.linalg.inv(xhat_transposed.dot(xhat_orig))
xhat_y = xhat_transposed.dot(y)
theta = inv_mat.dot(xhat_y)
```

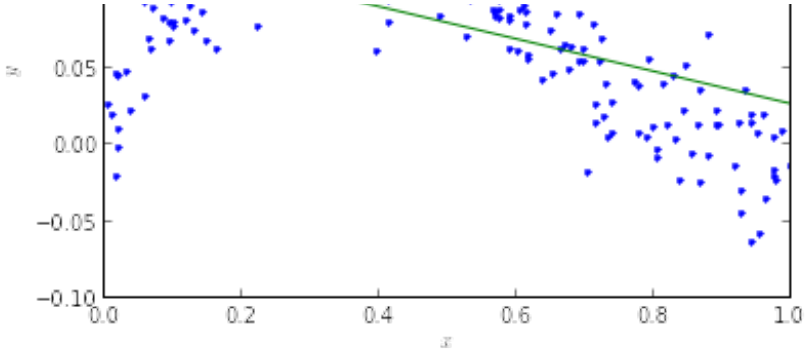
In [6]:

```
# Plot the data and your model fit.
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')

# Plot the regression line
xs = np.linspace(min(x), max(x),50)
xs = np.vstack((xs, np.ones_like(xs)))
plt.plot(xs[0,:], theta.dot(xs))
```

Out[6]:

```
[<matplotlib.lines.Line2D at 0x11577df10>]
```





QUESTIONS

(1) Does the linear model under- or overfit the data?

(2) How to change the model to improve the fitting?

ANSWERS

(1) The model underfits the data.

(2) To improve the fit, use a model of higher polynomial.

Fitting data to the model (10 points)

Here, we'll now do regression to polynomial models of orders 1 to 5. Note, the order 1 model is the linear model you prior fit.

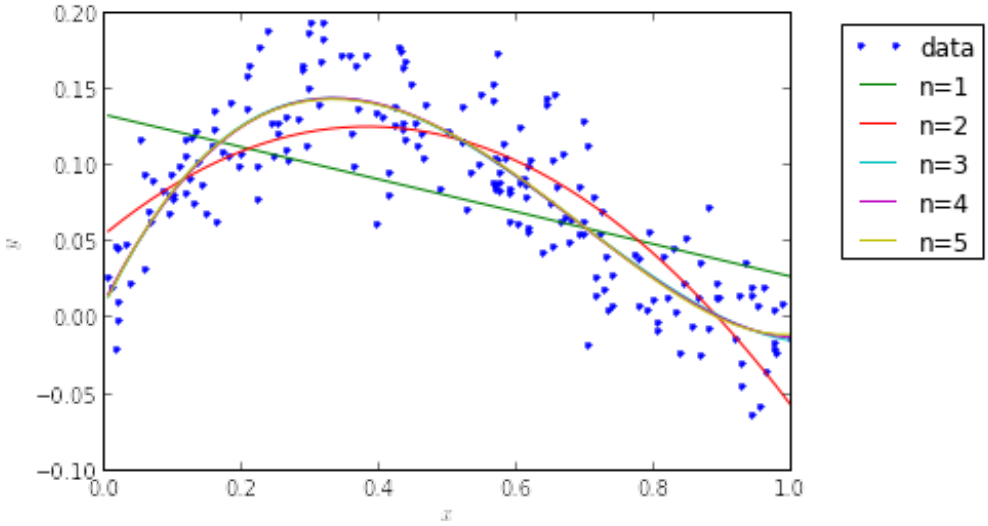
In [15]:

```
N = 5
xhats = []
thetas = []

# ==================== #
# START YOUR CODE HERE #
# ==================== #

# GOAL: create a variable thetas.
# thetas is a list, where theta[i] are the model parameters for the polynom
ial fit of order i+1.
#   i.e., thetas[0] is equivalent to theta above.
#   i.e., thetas[1] should be a length 3 np.array with the coefficients of
the x^2, x, and 1 respectively.
#   ... etc.

xhat = np.ones_like(x)
for i in range(0,N):
    if i == 0:
        xhat = np.vstack((x,xhat))
    else:
        xhat = np.vstack((x**(i+1),xhats[i-1]))
    xhats.append(xhat)

for i in range(0,N):
    xhats_orig = np.transpose(xhats[i])
```

```
    xhats_orig = np.transpose(xhats[i])
    xhats_transpose = xhats[i]
    inv_mat = np.linalg.inv(xhats_transpose.dot(xhats_orig))
    xhats_y = xhats_transpose.dot(y)
    theta = inv_mat.dot(xhats_y)
    thetas.append(theta)

# ================== #
# END YOUR CODE HERE #
# ================== #
```

In [16]:

```
# Plot the data
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')

# Plot the regression lines
plot_xs = []
for i in np.arange(N):
    if i == 0:
        plot_x = np.vstack((np.linspace(min(x), max(x),50), np.ones(50)))
    else:
        plot_x = np.vstack((plot_x[-2]**(i+1), plot_x))

    plot_xs.append(plot_x)

for i in np.arange(N):
    ax.plot(plot_xs[i][-2,:], thetas[i].dot(plot_xs[i]))

labels = ['data']
[labels.append('n={}'.format(i+1)) for i in np.arange(N)]
bbox_to_anchor=(1.3, 1)
lgd = ax.legend(labels, bbox_to_anchor=bbox_to_anchor)
```



Calculating the training error (10 points)

Here, we'll now calculate the training error of polynomial models of orders 1 to 5.

In [17]:

In [17]:

```
training_errors = []

# ==================== #
# START YOUR CODE HERE #
# ==================== #

# GOAL: create a variable training_errors, a list of 5 elements,
# where training_errors[i] are the training loss for the polynomial fit of
order i+1.

for i in range(0,N):
    yhats = thetas[i].dot(xhats[i])
    errors = (y - yhats)**2
    lse = sum(errors)
    training_errors.append(lse)

# ================== #
# END YOUR CODE HERE #
# ================== #

print ('Training errors are: \n', training_errors)
```

```
('Training errors are: \n', [0.47599221767254024, 0.21849844418537076, 0.16
339207602210748, 0.16330707470593966, 0.16322958391050585])
```

QUESTIONS

(1) What polynomial has the best training error?

(2) Why is this expected?

ANSWERS

(1) The highest polynomial, 5, has the best training error

(2) This is expected because higher degree polynomials have more freedom to fit the points of the training set

Generating new samples and testing error (5 points)

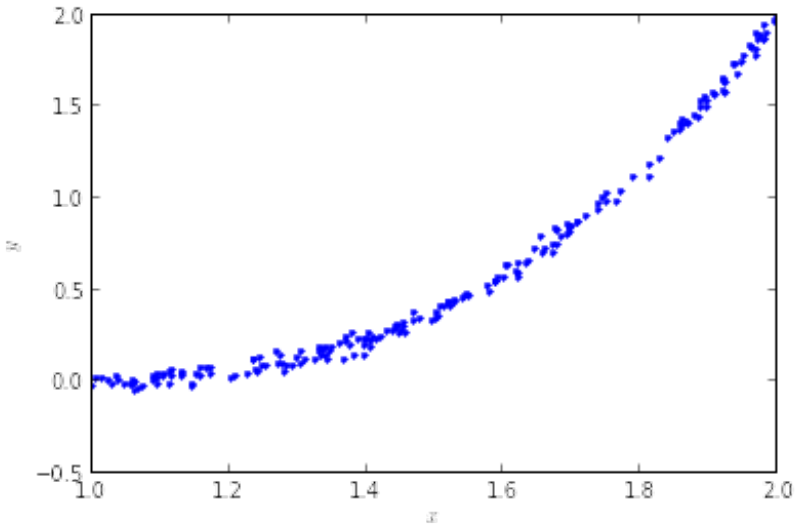
Here, we'll now generate new samples and calculate testing error of polynomial models of orders 1 to 5.

In [18]:

```
x = np.random.uniform(low=1, high=2, size=(num_train,))
y = x - 2*x**2 + x**3 + np.random.normal(loc=0, scale=0.03, size=(num_train,
))
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

Out[18]:

<matplotlib.text.Text at 0x11364cbd0>

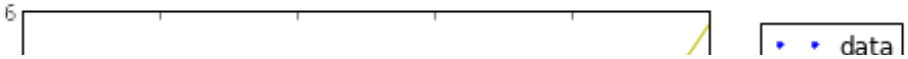


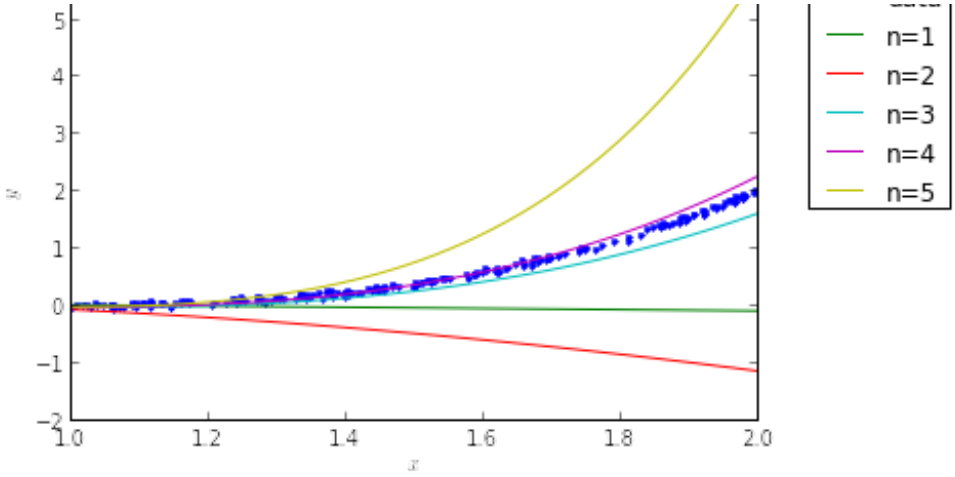
In [19]:

```
xhats = []
for i in np.arange(N):
    if i == 0:
        xhat = np.vstack((x, np.ones_like(x)))
        plot_x = np.vstack((np.linspace(min(x), max(x),50), np.ones(50)))
    else:
        xhat = np.vstack((x**(i+1), xhat))
        plot_x = np.vstack((plot_x[-2]**(i+1), plot_x))

    xhats.append(xhat)
```

In [20]:

```
# Plot the data
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')

# Plot the regression lines
plot_xs = []
for i in np.arange(N):
    if i == 0:
        plot_x = np.vstack((np.linspace(min(x), max(x),50), np.ones(50)))
    else:
        plot_x = np.vstack((plot_x[-2]**(i+1), plot_x))

    plot_xs.append(plot_x)

for i in np.arange(N):
    ax.plot(plot_xs[i][-2,:], thetas[i].dot(plot_xs[i]))

labels = ['data']
[labels.append('n={}'.format(i+1)) for i in np.arange(N)]
bbox_to_anchor=(1.3, 1)
lgd = ax.legend(labels, bbox_to_anchor=bbox_to_anchor)
```





In [22]:

```
testing_errors = []

# ==================== #
# START YOUR CODE HERE #
# ==================== #

# GOAL: create a variable testing_errors, a list of 5 elements,
# where testing_errors[i] are the testing loss for the polynomial fit of or
der i+1.

for i in range(0,N):
    yhats = thetas[i].dot(xhats[i])
    errors = (y - yhats)**2
    lse = sum(errors)
    testing_errors.append(lse)

# ================== #
# END YOUR CODE HERE #
# ================== #

print ('Testing errors are: \n', testing_errors)
```

```
('Testing errors are: \n', [161.72330369101161, 426.38384890115913, 6.25139
42167985785, 2.3741530385763361, 429.82043649440118])
```

QUESTIONS

(1) What polynomial has the best testing error?

(2) Why polynomial models of orders 5 does not generalize well?

ANSWERS

(1) The polynomial of degree 4 has the best testing error

(2) Polynomial models of orders 5 don't generalize well because they overfit the data, which leads to an eventual decrease in performance compared to polynomial models that generalize well.