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 C147/C247 Winter Quarter 2020, Prof. J.C. Kao, TAs W. Feng, J. Lee, K. Liang, M. Kleinman, C. Zheng

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

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

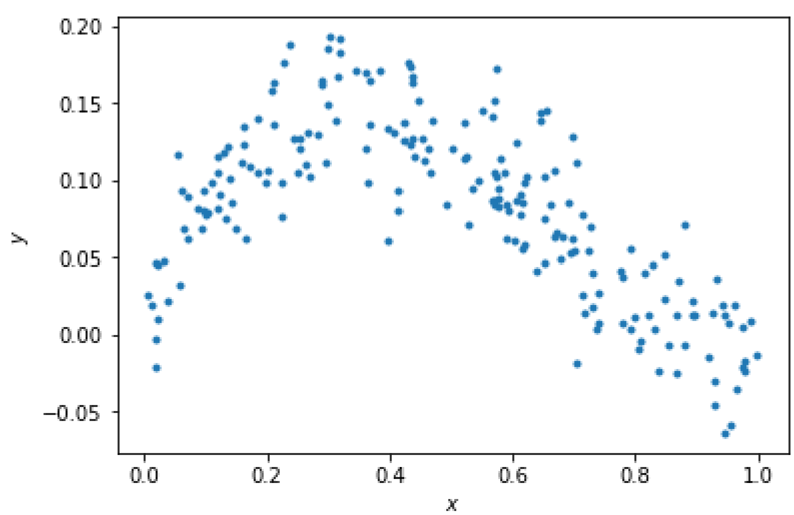
Data generation

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 [2]:
```
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[2]:
```
Text(0, 0.5, '$y$')
```



Typesetting math: 100%

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 distribution of X is a uniform RV (random variable) in between 0 and 1.

(2) the distribution of the additive noise ϵ is a guassian RV (random variable) with 0 mean and 0.03 standard deviation

Fitting data to the model (5 points)

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

In [3]:
```
# xhat = (x, 1)
xhat = np.vstack((x, np.ones_like(x)))

# ==================== #
# START YOUR CODE HERE #
# ==================== #
# GOAL: create a variable theta; theta is a numpy array whose elements are [a, b]

theta = np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y))
print(theta)

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

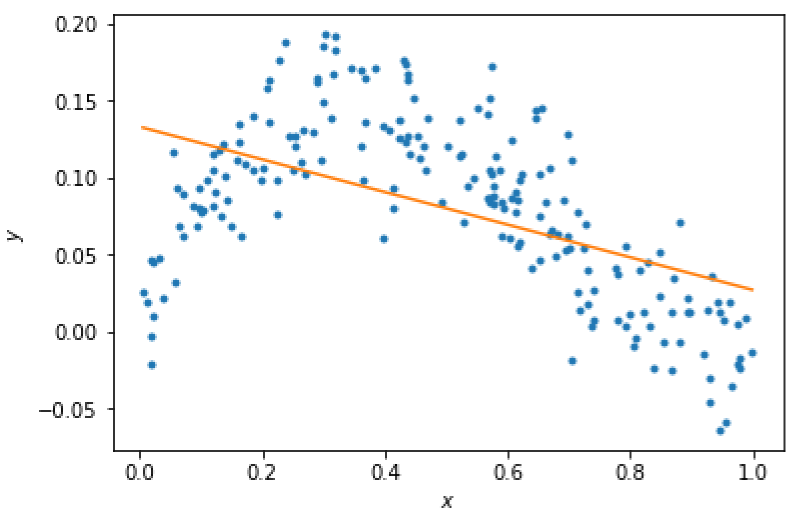
```
[-0.10599633  0.13315817]
```

Typesetting math: 100%

In [4]:
```
# 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[4]:
```
[<matplotlib.lines.Line2D at 0x2a657f28f28>]
```



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 definitely underfits the data.

(2) By looking at the graph, increasing the order of the model would help reduce underfitting.

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.

Typesetting math: 100%

In [7]:

```
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 polynomial fi
#   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^
#   ... etc.

xhat = np.vstack((x, np.ones_like(x)))
theta = np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y)) #Copypasta'd from the ab

xhats.append(xhat)
thetas.append(theta)

for i in range(1, N):
    alpha = np.power(x, i+1)
    xhat = np.vstack((alpha, xhat))
    theta = np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y))
    xhats.append(xhat)
    thetas.append(theta)

pass

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

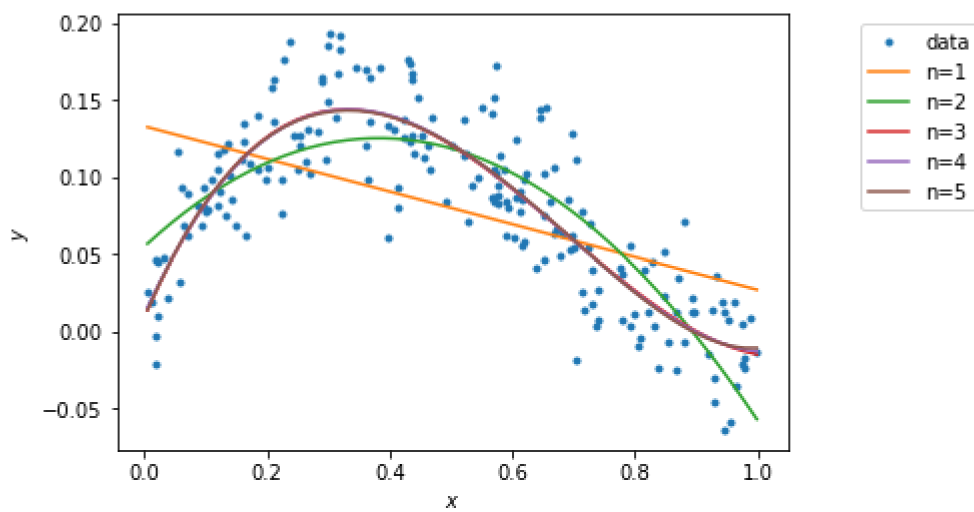
Typesetting math: 100%

In [8]:
```
# 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:

$$L(\theta) = \frac{1}{2} \sum_j (\hat{y}_j - y_j)^2$$

Typesetting math: 100%

In [11]:
```
training_errors = []

# ==================== #
# START YOUR CODE HERE #
# ==================== #
yhats = []
for i in range(0, N):
    yhats.append(thetas[i].dot(xhats[i]))
    training_errors.append(0.5*np.sum(np.power((y-yhats[i]), 2)))
# 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
pass

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

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

```
Training errors are: 
 [0.23799610883627012, 0.10924922209268528, 0.08169603801105371, 0.081653537352
96982, 0.0816147919552529]
```

QUESTIONS

(1) Which polynomial model has the best training error?

(2) Why is this expected?

ANSWERS

(1) The 5th order polynomial has the best (eg lower) training error.

(2) This is since the model has more freedom to, eg it could potentially overfit. A higher order polynomial has the ability to be a lower model by shifting its weights of its higher order terms (eg coefficients close to 0 for terms of x^2, x^3 ...)

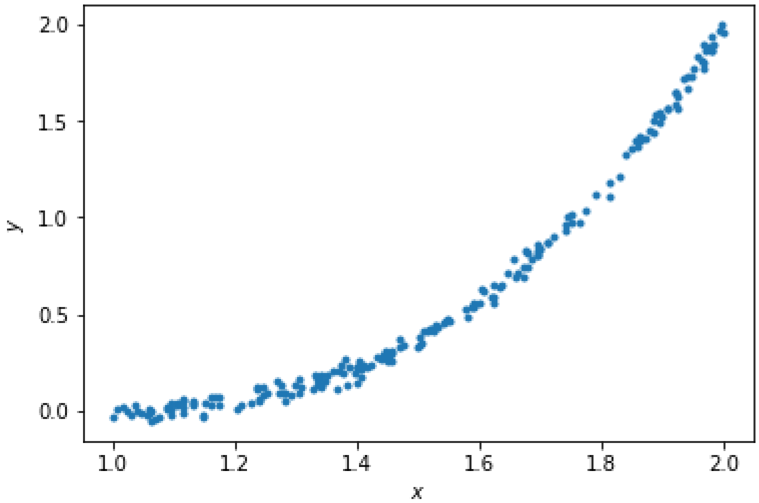
Generating new samples and testing error (5 points)

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

Typesetting math: 100%

In [12]:
```
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[12]: Text(0, 0.5, 'y')



In [13]:
```
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)
```

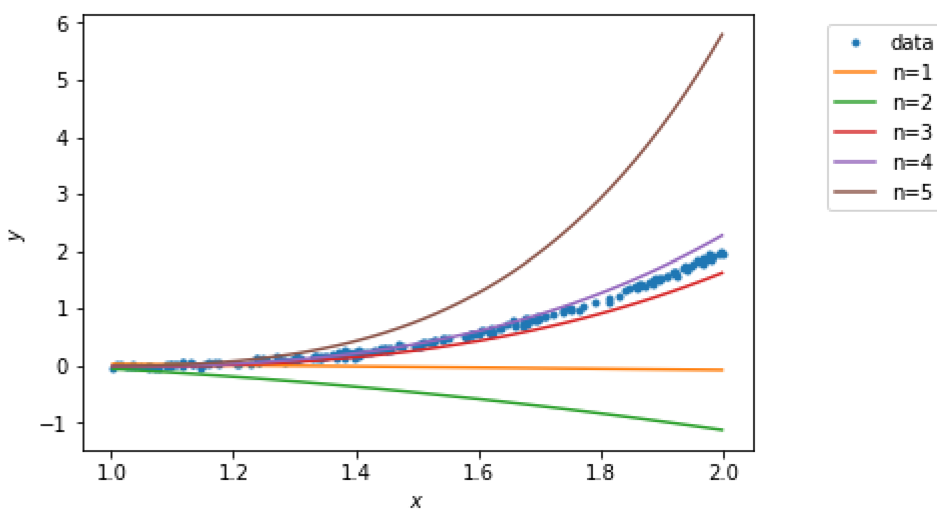
Typesetting math: 100%

In [14]:
```
# 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)
```



Typesetting math: 100%

In [18]:
```
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 order i+

yhats = []
for i in range(0, N):
    #Do the same function
    yhats.append(thetas[i].dot(xhats[i]))
    testing_errors.append(0.5*np.sum(np.power((y-yhats[i]), 2)))
pass

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

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

```
Testing errors are:
 [80.86165184550595, 213.19192445058434, 3.125697108312602, 1.187076519510533,
214.9102182117732]
```

QUESTIONS

(1) Which polynomial model has the best testing error?

(2) Why does the order-5 polynomial model not generalize well?

ANSWERS

(1) The graph n = 4 (order 4) most resembles / is closest to the distribution that we just created, and has the best (lowest) testing error.

(2) The order-5 polynomial is overfitted to the training data, so when test-data is given (from a different distribution), it fails miserably.

Typesetting math: 100%