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: $y = x - 2x^2 + x^3 + \epsilon$

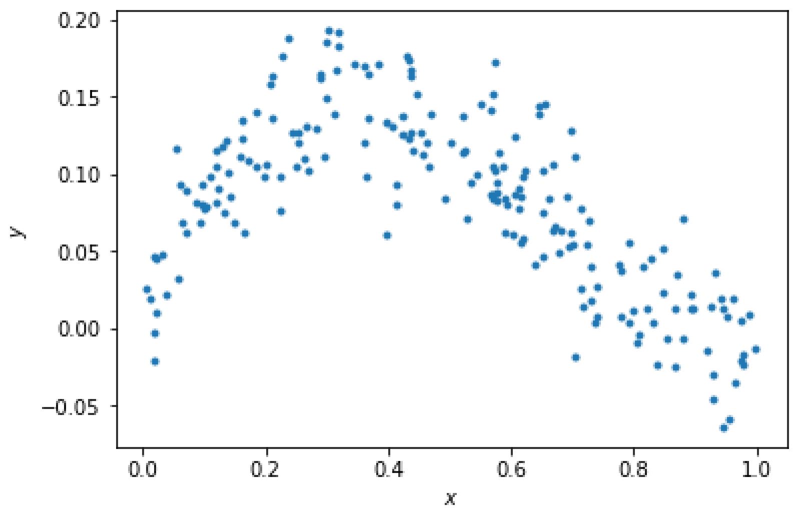
In [2]:

```
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, '$y$')
```



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 noise is a normal distribution with mean 0 and standard deviation 0.03.

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]

xhat_t = xhat.T
xhat_y = xhat_t.T.dot(y)
inv_mat = np.linalg.inv(xhat.dot(xhat_t))
theta = inv_mat.dot(xhat_y)# please modify this line

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

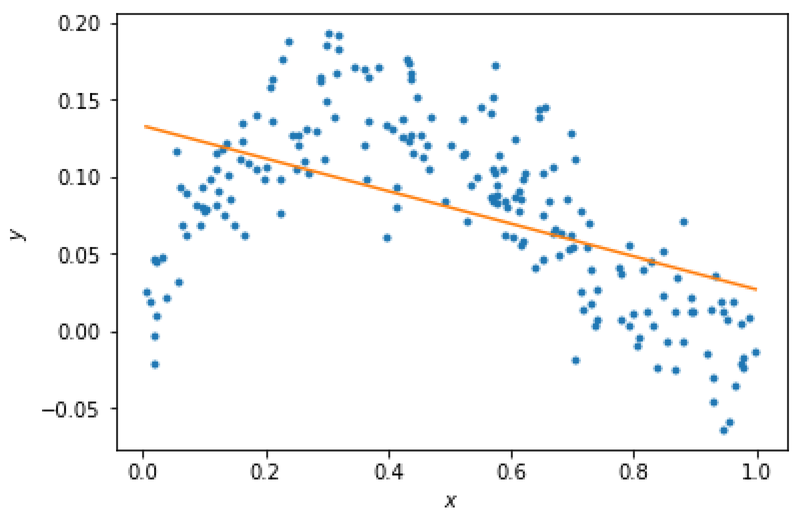
In [4]:

```
# 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 0x11e635e80>]
```



QUESTIONS

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

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

ANSWERS

(1) This model underfit the data.

(2) Add parameters for higher order of feature x, construct a higher degree polynomial model.

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 [5]:

```
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 fit o
#   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,
#   ... etc.
for i in range(1,N+1):
    xhat = np.ones_like(x)
    for j in range(1,i+1):
        xhat = np.vstack((x**j,xhat))

    xhat_t = xhat.T
    xhat_y = xhat_t.T.dot(y)
    inv_mat = np.linalg.inv(xhat.dot(xhat_t))
    theta = inv_mat.dot(xhat_y)

    xhats.append(xhat_t)
    thetas.append(theta)


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

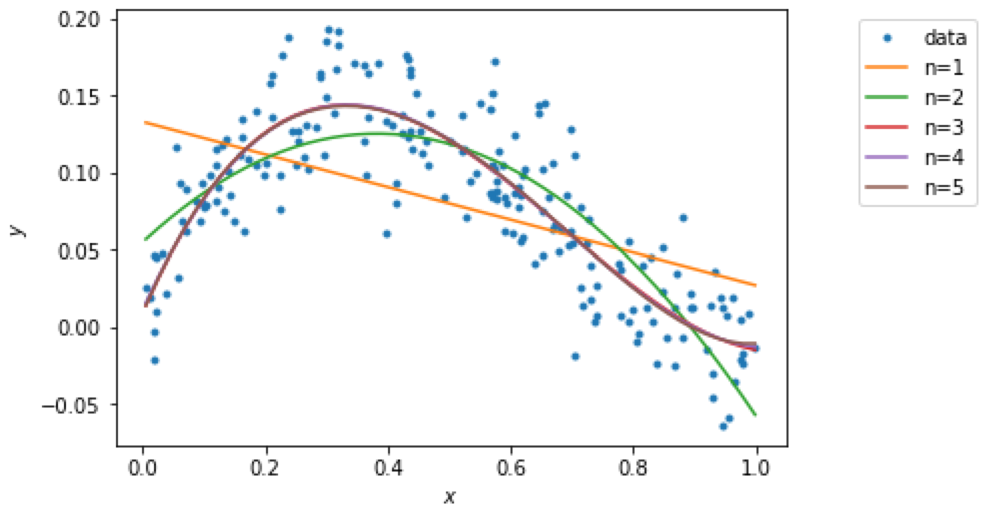
In [6]:

```
# 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:

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

In [7]:

```
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+

for i in range(N):
    error = y - xhats[i].dot(thetas[i])
    training_error = error.T.dot(error)/num_train
    training_errors.append(training_error)

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

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

```
Training errors are:
 [0.0023799610883627007, 0.0010924922209268528, 0.0008169603801105374,
0.0008165353735296982, 0.0008161479195525295]
```

QUESTIONS

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

(2) Why is this expected?

ANSWERS

(1) The polynomial model with order of 5.

(2) With more and higher orders, it gives the model more space to adjust parameters to change the look of the function. In this way, the curve we get can better fit the curve of given data.

Generating new samples and testing error (5 points)

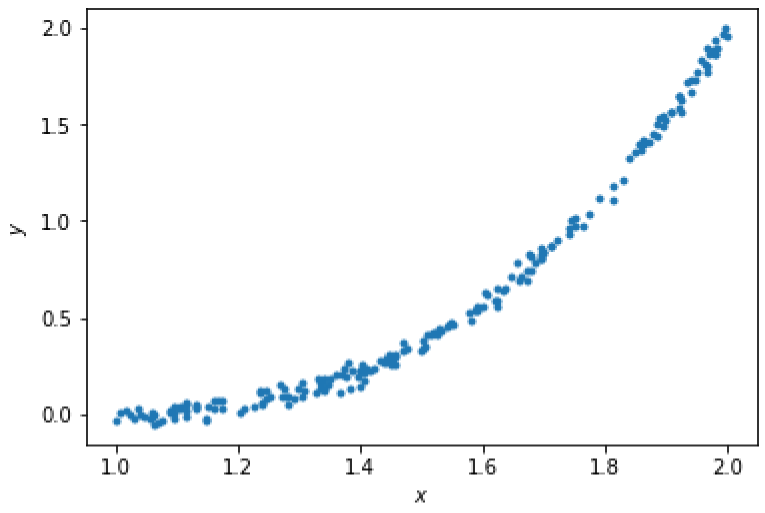
Here, we'll now generate new samples and calculate the testing error of polynomial models of orders 1 to 5.

In [8]:

```
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[8]:

```
Text(0, 0.5, '$y$')
```



In [9]:

```
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)
```

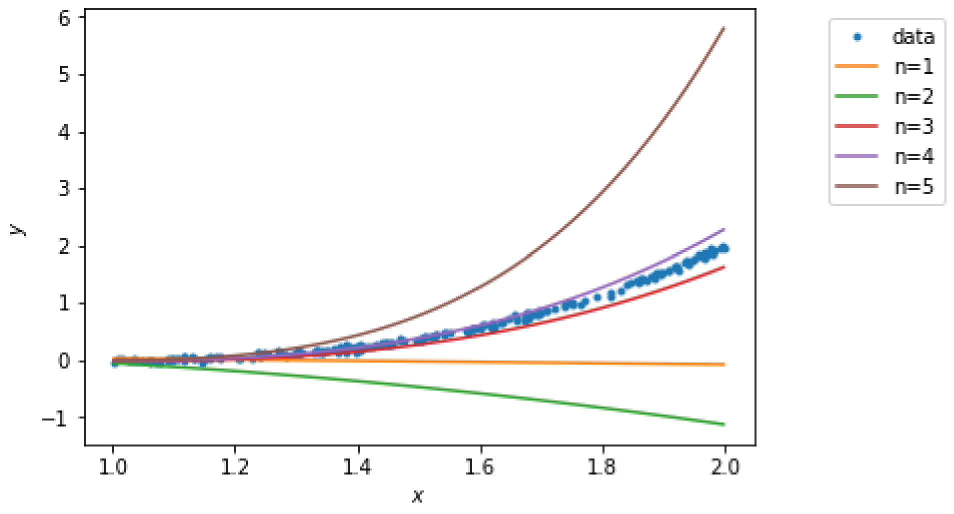
In [10]:

```
# 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 [11]:

```
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+1.

for i in range(N):
    error = y - xhats[i].T.dot(thetas[i])
    testing_error = error.T.dot(error) / num_train
    testing_errors.append(testing_error)

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

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

```
Testing errors are: 
 [0.8086165184550587, 2.1319192445057893, 0.031256971082763 93, 0.01187
0765189474703, 2.1491021817652625]
```

QUESTIONS

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

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

ANSWERS

(1) The model with the order of 4.

(2) The model with the order of 5 overfits the training data, which means it tries to fit the 200-sample traning data too much. As a result, it is influenced by the noise during learning process and predicts not well for the test set.