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 2021, Prof. J.C. Kao, TAs: N. Evirgen, A. Ghosh, S. Mathur, T. Monsoor, G. Zhao

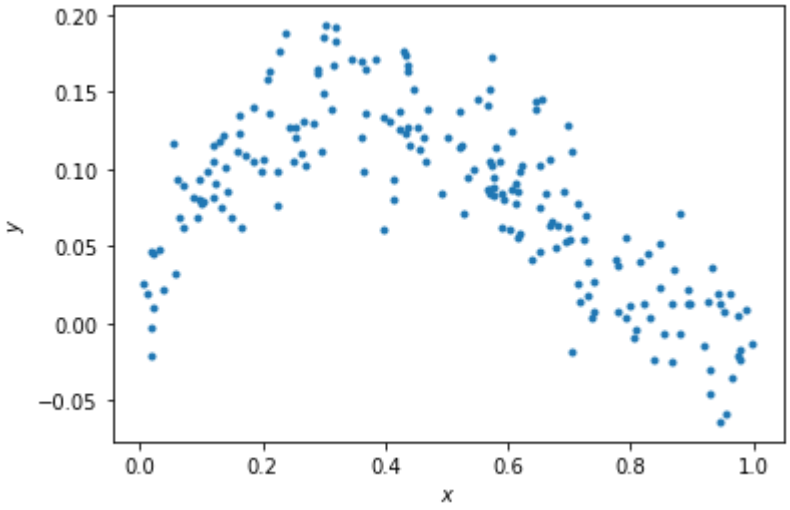
In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

#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 [3]:
```
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[3]: 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) X is uniformly distributed in (0, 1).

(2) ϵ is normally distributed with μ = 0 and σ = 0.03.

Fitting data to the model (5 points)

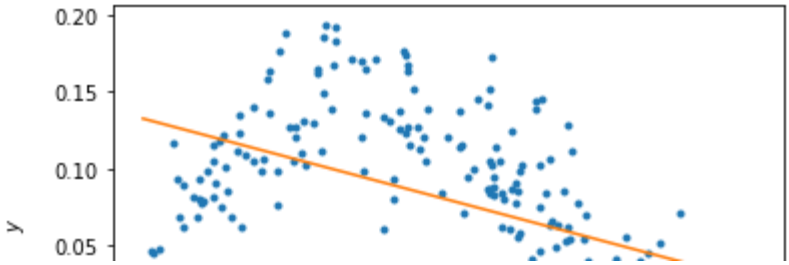
Here, we'll do linear regression to fit the parameters of a model $y = ax + b$.

In [26]:
```
# 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

print(xhat.shape)
print(xhat.T.shape)

#use normal equation to calculate theta
th = np.linalg.inv((xhat).dot(xhat.T)).dot(xhat.dot(y))
print(th)
print(th.shape)
```

```
(2, 200)
(200, 2)
[-0.10599633  0.13315817]
(2,)
```

In [32]:
```
# 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,:], th.dot(xs))
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x7f8186b53a00>]
```



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 under-fits the data.

(2) Include higher-order terms in the model by increasing the size of theta.

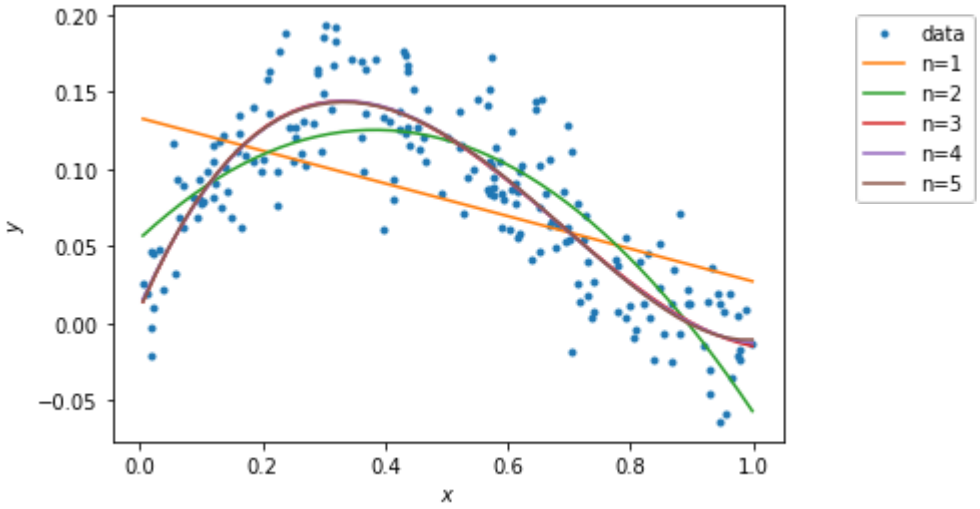
Fitting data to the model (10 points)

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 [35]:
```
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 polynomia
#   i.e., thetas[0] is equivalent to theta above.
#   i.e., thetas[1] should be a length 3 np.array with the coefficients of th
#   ... etc.

xhat = np.vstack((x, np.ones_like(x)))
theta = np.linalg.inv((xhat).dot(xhat.T)).dot(xhat.dot(y))

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

for i in range(1, N):
    x_row = x**(i+1)
    xhat = np.vstack((x_row, xhat))
    xhats.append(xhat)
    print("xhat shape: {}".format((xhat.shape)))
    theta = np.linalg.inv((xhat).dot(xhat.T)).dot(xhat.dot(y))
    thetas.append(theta)
    print("theta shape: {}".format((theta.shape)))


#pass

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

```
xhat shape: (3, 200)
theta shape: (3,)
xhat shape: (4, 200)
theta shape: (4,)
xhat shape: (5, 200)
theta shape: (5,)
xhat shape: (6, 200)
theta shape: (6,)
```

In [36]:
```
# 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 [38]:
```
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 or

for i in range(0, N):
    training_loss = ((np.linalg.norm(y - thetas[i].dot(xhats[i])))**2) * 0.5
    training_errors.append(training_loss)

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

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

```
Training errors are:
 [0.2379961088362701, 0.1092492220926853, 0.08169603801105373, 0.081653537352
96978, 0.08161479195525295]
```

QUESTIONS

(1) What polynomial has the best training error?

(2) Why is this expected?

ANSWERS

(1) The highest-order polynomial has the best training error.

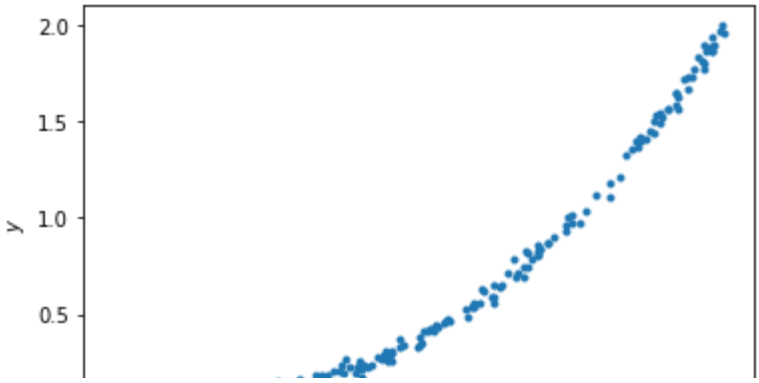
(2) It has the most degrees of freedom to fit each individual point in the training dataset, whether or not that reflects the noise in the dataset or the actual trends in the data.

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.

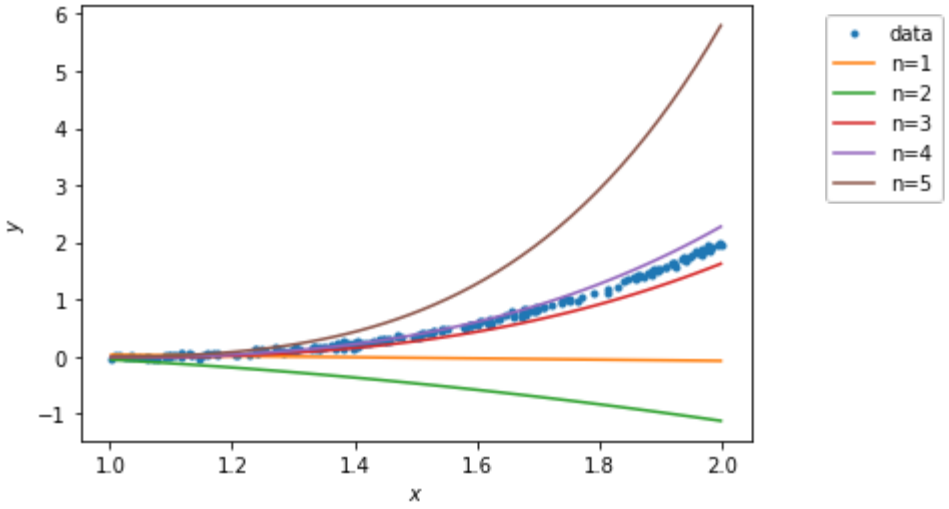
In [39]:
```
x = np.random.uniform(low=1, high=2, size=(num_train,))
y = x - 2*x**2 + x**3 + np.random.normal(loc=0, scale=0.03, size=(num_train,)
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

Out[39]:
```
Text(0, 0.5, '$y$')
```



In [40]:
```
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 [41]:
```
# 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 [44]:
```
testing_errors = []

# print(xhats[0].shape)

# ==================== #
# 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 orde


for i in range(0, N):
    test_error = ((np.linalg.norm(y - thetas[i].dot(xhats[i])))**2) * 0.5
    testing_errors.append(test_error)

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

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

```
Testing errors are:
 [80.86165184550585, 213.1919244505791, 3.125697108408374, 1.187076521149622
6, 214.91021747012792]
```

QUESTIONS

(1) What polynomial has the best testing error?

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

ANSWERS

(1) The 4th degree polynomial has the lowest testing error.

(2) The 5th degree polynomial doesn't generalize well because it overfits - it finds patterns in the training set which do not correspond to patterns in the unseen test data with different noise or range.

In [ ]: