1.

(a) i. $A = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$

$AA^T = I$

$A - \lambda I$

$= \begin{pmatrix} \frac{1}{\sqrt{2}} - \lambda & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} - \lambda \end{pmatrix}$

$\det(A - \lambda I) = 0$

$\Rightarrow \lambda_1 = \frac{1+j}{\sqrt{2}}$

$\lambda_2 = \frac{1-j}{\sqrt{2}}$

$A - \lambda_1 I$

$= \begin{pmatrix} -\frac{j}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{j}{\sqrt{2}} \end{pmatrix}$

$\Rightarrow$ eigenvector $v_1 = \begin{pmatrix} -\frac{j}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$

$A - \lambda_2 I$

$= \begin{pmatrix} \frac{j}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{j}{\sqrt{2}} \end{pmatrix}$

$\Rightarrow$ eigenvector $v_2 = \begin{pmatrix} \frac{j}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$

ii. $Av = \lambda v$

$\Rightarrow \|Av\|^2 = \|\lambda v\|^2 = |\lambda|^2 \|v\|^2$

$\|Av\|^2$

$= (Av)^T (Av)$

$= (v^T A^T)(Av)$

$= v^T A^T A v$

$AA^T = I$

$\Rightarrow AA^T A = A$

$$\left.\begin{array}{l} \Rightarrow A(A^T A - I) = 0 \\ A \neq 0 \end{array}\right\}$$

$\Rightarrow A^T A = I$

$\Rightarrow \|Av\|^2$

$= v^T v = \|v\|^2$

$\Rightarrow |\lambda|^2 = 1$

iii. Let λ_1, λ_2 be distinct eigenvalues of A correpanding to e-vects v_1, v_2

$$\left.\begin{array}{l} Av_1 = \lambda_1 v_1 \\ Av_2 = \lambda v_2 \end{array}\right\} \Rightarrow (Av_1)^T A v_2 = (\lambda v_1)^T (\lambda v_2)$$

$\Rightarrow v_1^T A^T A v_2 = \lambda_1 \lambda_2 v_1^T v_2$

$\Rightarrow v_1^T v_2 = \lambda_1 \lambda_2 v_1^T v_2$

$\Rightarrow (\lambda_1 \lambda_2 - 1) v_1^T v_2 = 0$

since ~~$v_1 \neq v_2$~~ $\lambda_1 \neq \lambda_2$ and $|\lambda| = 1$, $\lambda_1 \lambda_2 \neq 1$

So $v_1^T v_2 = 0$, namely orthogonal

iv. Its norm would not change. However, it will be rotated or flipped.

(b) i. Denote SVD of A is $A = U \Sigma V^T$
$= U_1 S V_1^T$

AA^T

$= U \Sigma V^T V \Sigma^T U^T$

$= U_1 S S^T U_1^T$

$A^T A$

$= V \Sigma^T U^T U \Sigma V^T$

$= V_1 S^T S V_1^T$

The left singular vectors of A are e-vects of AA^T. The right singular vectors of A are e-vects of $A^T A$.

ii. The singular value of A is the square root of e-vals of AA^T and $A^T A$.

(c) i. False. Zero linear operator 0 has ~~only~~ one eigenvalue~~s~~ 0. Identity linear operator has only one distinct eigenvalue~~s~~, 1.

ii. False. Sum would change direction of e-vect.

iii. True.

iiii. False. The argument is false when counting repeated e-vals.

v. ~~True~~. False.

$A = \begin{bmatrix} \frac{3}{2} & -\frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 \end{bmatrix}$

with two e-vals 1, 1 corresponding to e-vect

$\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix}, \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \\ 0 \end{bmatrix}$

However, $\begin{bmatrix} \sqrt{2} \\ 0 \\ 0 \end{bmatrix}$ is not an e-vect.

2.

(a) i. $p(H50 \mid T)$

$= \frac{p(H50, T)}{p(T)}$

~~$= \frac{p(T \mid H50)\, p(H50)}{\frac{1}{2}(p(H50, T) + p(H60, T))}$~~

~~$= \frac{0.5 \times 0.5}{\frac{1}{2}(0.5 + 0.6)}$~~

$= \frac{p(T \mid H50)\, p(H50)}{p(H50, T) + p(H60, T)}$

$= \frac{p(T \mid H50)\, p(H50)}{p(T \mid H50)\, p(H50) + p(T \mid H60)\, p(H60)}$

$= \frac{0.5 \times 0.5}{0.5 \times 0.5 + 0.4 \times 0.5}$

$= \frac{5}{9}$

ii. $p(H50 \mid THHH)$

$= \frac{p(H50, THHH)}{p(H50, THHH) + p(H60, THHH)}$

$= \frac{p(THHH \mid H50)\, p(H50)}{p(THHH \mid H50)\, p(H50) + p(THHH, H60)\, p(H60)}$

$= \frac{(0.5 \times 0.5 \times 0.5 \times 0.5) \times 0.5}{0.5^4 \times 0.5 + (0.4 \times 0.6^3) \times 0.5}$

$= \frac{625}{1489}$

$= 0.4197$

iii. $p(H50 \mid \geq 9H)$

$= \frac{p(\geq 9H \mid H50)\, p(H50)}{p(\geq 9H \mid H50)\, p(H50) + p(\geq 9H \mid H55)\, p(H55) + p(\geq 9H \mid H60)}$

$= \frac{(0.5^{10} \times 11) \times \frac{1}{3}}{(0.5^{10} \times 11) \times \frac{1}{3} + (0.55^{10} + 10 \times 0.45 \times 0.55^9) \times \frac{1}{3} + (0.6^{10} + 10 \times 0.4 \times 0.6^9) \times \frac{1}{3}}$

$= 0.1337$

Similarly,

$p(H55 \mid \geq 9H)$

$= \frac{(0.55^{10} + 10 \times 0.45 \times 0.55^9) \times \frac{1}{3}}{(0.5^{10} \times 11) \times \frac{1}{3} + (0.55^{10} + 10 \times 0.45 \times 0.55^9) \times \frac{1}{3} + (0.6^{10} + 10 \times 0.4 \times 0.6^9) \times \frac{1}{3}}$

$= 0.2894$

$p(H60 \mid \geq 9H)$

$= \frac{(0.6^{10} + 10 \times 0.4 \times 0.6^9) \times \frac{1}{3}}{(0.5^{10} \times 11) \times \frac{1}{3} + (0.55^{10} + 10 \times 0.45 \times 0.55^9) \times \frac{1}{3} + (0.6^{10} + 10 \times 0.4 \times 0.6^9) \times \frac{1}{3}}$

$= 0.5769$

(b) $p(preg \mid pos)$ ~~$p(pos \mid preg)$~~

$$= \frac{p(pos, preg)}{\cancel{p(preg)}\, p(pos)}$$

$$= \frac{p(preg)\, p(pos \mid preg)}{p(pos \mid preg)\, p(preg) + p(pos \mid not\ preg)\, p(not\ preg)}$$

$$= \frac{0.01 \times 0.99}{0.99 \times 0.01 + 0.1 \times 0.99}$$

$$= 0.0909$$

That makes sense because fall-out ratio is too high (10%).

(c) $E(Ax_i)$

$$= E\left(\sum_{j=1}^{n} A_{i,j} x_j\right)$$

$$= \left(\sum_{j=1}^{n} A_{ij} E(x_j)\right)$$

$$= \left(\sum_{j=1}^{n} A_{ij} E(x)_j\right)$$

$$= [A \cdot E(x)]_i$$

$$\Rightarrow E(Ax) = AE(x)$$

$$\Rightarrow E(Ax + b) = E(Ax) + b$$
$$= AE(x) + b$$

(d) $cov(Ax + b)$

$$= E\left((Ax + b - AE(x) - b)(Ax + b - (AE(x) + b))^T\right)$$

$$= E\left((Ax - AE(x))(Ax - (AE(x))^T)\right)$$

$$= E\left(A(x - E(x))((x - E(x)))^T A^T\right)$$

$$= AE\left((x - E(x))((x - E(x)))^T\right) A^T$$

$$= A\, cov(x) A^T$$

3.

(a) $\nabla_x x^T A y = Ay$

(b) $\nabla_y x^T A y = A^T x$

(c) $\nabla_A x^T A y$

$$= \begin{bmatrix} \frac{\partial x^T A y}{\partial a_{11}} & \cdots & \frac{\partial x^T A y}{\partial a_{1m}} \\ \vdots & & \vdots \\ \frac{\partial x^T A y}{\partial a_{n1}} & \cdots & \frac{\partial x^T A y}{\partial a_{nm}} \end{bmatrix}$$

$$= xy^T$$

(d) $\nabla_x (x^T A x + b^T x)$

$$= \nabla_x (x^T A x) + \nabla_x (b^T x)$$

$$= Ax + A^T x + b$$

(e) Suppose $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times q}$

$tr(AB)$

$$= tr\left(\begin{bmatrix} a_1 \\ \vdots \\ a_m \end{bmatrix} [b_1 \cdots b_q]\right)$$

$$= tr\left(\begin{bmatrix} a_1 b_1 & a_1 b_2 & \cdots & a_1 b_q \\ \vdots & & & \vdots \\ a_m b_1 & \cdots & & a_m b_q \end{bmatrix}\right)$$

$$= \sum_{i=1}^{n} a_{1i} b_{i1} + \sum_{i=1}^{n} a_{2i} b_{i2} + \cdots$$

$$\Rightarrow \frac{\partial tr(AB)}{\partial A}$$

$$= \begin{bmatrix} \frac{\partial tr(AB)}{\partial a_{11}} & \cdots & \frac{\partial tr(AB)}{\partial a_{1n}} \\ \vdots & & \vdots \\ \frac{\partial tr(AB)}{\partial a_{m1}} & \cdots & \frac{\partial tr(AB)}{\partial a_{mn}} \end{bmatrix}$$

$$\underset{m=q}{=} \begin{cases} \begin{bmatrix} b_{11} & b_{21} & \cdots & b_{n1} \\ \vdots & & & \vdots \\ b_{1n} & \cdots & & b_{nn} \end{bmatrix} & \text{if } n = m \\ \begin{bmatrix} b_{11} & \cdots & b_{n1} \\ \vdots & & \vdots \\ b_{1n} & \cdots & b_{nn} \\ 0 & \cdots & 0 \end{bmatrix} & \text{if } n < m \ \text{~~otherwise~~} \\ \begin{bmatrix} b_{11} & \cdots & b_{m1} & 0 \\ \vdots & & \vdots & \vdots \\ b_{1m} & \cdots & b_{mm} & 0 \end{bmatrix} & \text{otherwise} \end{cases}$$

4. $f(W) = \frac{1}{2} \sum_{i=1}^{n} \| y^{(i)} - Wx^{(i)} \|^2$

$$= \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - Wx^{(i)})^T (y^{(i)} - Wx^{(i)})$$

$$= \frac{1}{2} \sum_{i=1}^{n} (-2y^{(i)T} W x^{(i)} + x^{(i)T} W^T W x^{(i)})$$

$$= \sum_{i=1}^{n} \left[-tr(y^{(i)T} W x^{(i)}) + \frac{1}{2} tr(x^{(i)T} W^T W x^{(i)})\right]$$

$$= \sum_{i=1}^{n} \left[-tr(W x^{(i)} y^{(i)T}) + \frac{1}{2} tr(W x^{(i)} x^{(i)T} W^T)\right]$$

$$= -tr\left(W \sum_{i=1}^{n} x^{(i)} y^{(i)T}\right) + \frac{1}{2} tr\left(W \sum_{i=1}^{n} (x^{(i)} x^{(i)T}) W^T\right)$$

$$= -tr(WXY^T) + \frac{1}{2} tr(WXX^T W^T)$$

$$\frac{\partial f}{\partial W} = -YX^T + \frac{1}{2}(WXX^T + WXX^T)$$

$$= -YX^T + WXX^T$$

$$\frac{\partial f}{\partial W} = 0 \Rightarrow W = YX^T (XX^T)^{-1}$$

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

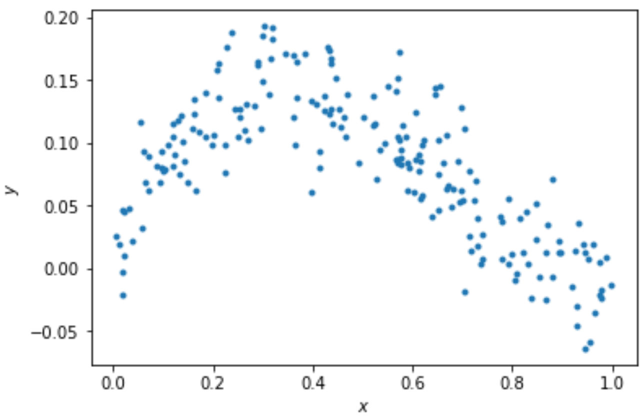
In [2]:

```
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_tr
ain,))
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) Uniform distribution.

(2) 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 [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 a
        re [a, b]

        theta = np.zeros(2) # please modify this line
        step = 0.05
        # Gradient Descent
        for i in range(1000):
            dloss = -xhat.dot(y) + xhat.dot(xhat.T).dot(theta)
            theta = theta - (dloss/abs(dloss))*(step/np.sqrt(pow(dloss,2)+1))
            if i == 200:
                step = step / 5
            elif i == 500:
                step = step / 5
        # ================== #
        # END YOUR CODE HERE #
        # ================== #
```

```
In [4]: theta
```

```
Out[4]: array([-0.10800943,  0.13464018])
```

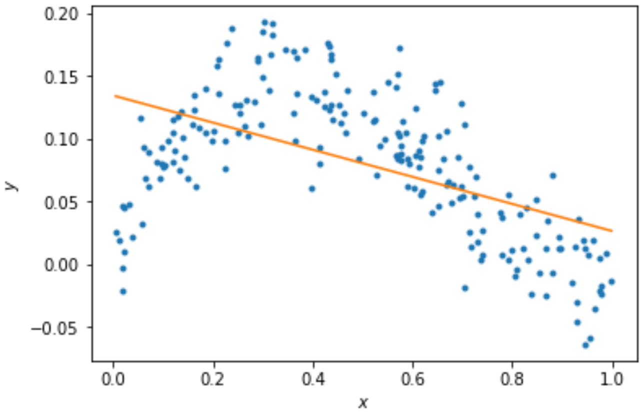
```
In [5]: theta_lin = np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y))
        theta_lin
```

```
Out[5]: array([-0.10599633,  0.13315817])
```

```
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 0x7f815aa60828>]
```



QUESTIONS

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

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

ANSWERS

(1) No. It's fine consider that we are doing linear regression.

(2) We could do regression to polynomial models.

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 [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 poly
nomial 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.
# Linear
xhats.append(xhat)
thetas.append(theta)
# pow 2
xhats.append(np.vstack((pow(x,2),x,np.ones(x.size))))
theta_tmp = np.zeros(3)
step = 0.05
# Gradient Descent
for i in range(1000):
    dloss = -xhats[-1].dot(y) + xhats[-1].dot(xhats[-1].T).dot(theta_tm
p)
    theta_tmp = theta_tmp - (dloss/abs(dloss))*(step/np.sqrt(pow(dloss,
2)+1))
    if i == 200:
        step = step / 5
    elif i == 500:
        step = step / 5
thetas.append(theta_tmp)
# pow 3
xhats.append(np.vstack((pow(x,3),pow(x,2),x,np.ones(x.size))))
theta_tmp = np.zeros(4)
step = 0.05
# Gradient Descent
for i in range(1000):
    dloss = -xhats[-1].dot(y) + xhats[-1].dot(xhats[-1].T).dot(theta_tm
p)
    theta_tmp = theta_tmp - (dloss/abs(dloss))*(step/np.sqrt(pow(dloss,
2)+1))
    if i == 200:
        step = step / 5
    elif i == 500:
        step = step / 5
thetas.append(theta_tmp)
# pow 4
xhats.append(np.vstack((pow(x,4),pow(x,3),pow(x,2),x,np.ones(x.size))))
theta_tmp = np.zeros(5)
step = 0.05
# Gradient Descent
for i in range(1000):
    dloss = -xhats[-1].dot(y) + xhats[-1].dot(xhats[-1].T).dot(theta_tm
p)
    theta_tmp = theta_tmp - (dloss/abs(dloss))*(step/np.sqrt(pow(dloss,
2)+1))
    if i == 200:
        step = step / 5
    elif i == 500:
        step = step / 5
thetas.append(theta_tmp)
# pow 5
xhats.append(np.vstack((pow(x,5),pow(x,4),pow(x,3),pow(x,2),x,np.ones(x.
size))))
theta_tmp = np.zeros(6)
step = 0.05
```

```
In [8]: thetas
```

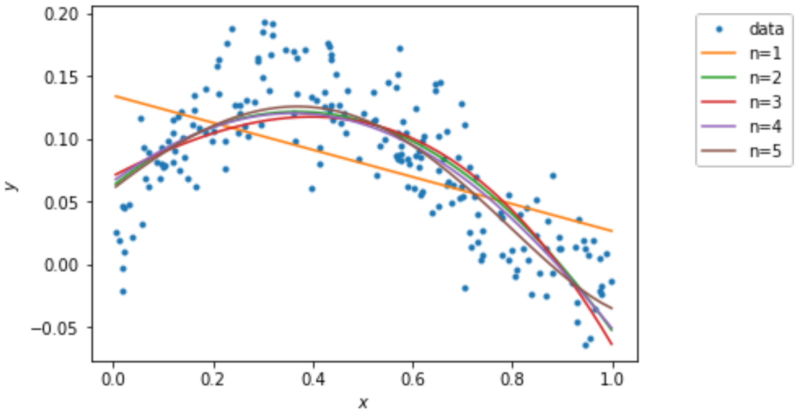
```
Out[8]: [array([-0.10800943,  0.13464018]),
          array([-0.43682215,  0.32109683,  0.062921  ]),
          array([-0.20165278, -0.13866618,  0.20538297,  0.07073462]),
          array([ 0.11693129, -0.20611328, -0.312629  ,  0.28407817,  0.0665852
         5]),
          array([ 0.3102328 , -0.11760083, -0.3841783 , -0.21215671,  0.30802978,
                  0.06060368])]
```

```
In [9]: # 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(5
        0)))
            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 [10]: 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.
         training_errors.append(0.5*(y.T.dot(y)-2*y.T.dot(xhats[0].T).dot(thetas
         [0])+thetas[0].T.dot(xhats[0]).dot(xhats[0].T).dot(thetas[0])))
         training_errors.append(0.5*(y.T.dot(y)-2*y.T.dot(xhats[1].T).dot(thetas
         [1])+thetas[1].T.dot(xhats[1]).dot(xhats[1].T).dot(thetas[1])))
         training_errors.append(0.5*(y.T.dot(y)-2*y.T.dot(xhats[2].T).dot(thetas
         [2])+thetas[2].T.dot(xhats[2]).dot(xhats[2].T).dot(thetas[2])))
         training_errors.append(0.5*(y.T.dot(y)-2*y.T.dot(xhats[3].T).dot(thetas
         [3])+thetas[3].T.dot(xhats[3]).dot(xhats[3].T).dot(thetas[3])))
         training_errors.append(0.5*(y.T.dot(y)-2*y.T.dot(xhats[4].T).dot(thetas
         [4])+thetas[4].T.dot(xhats[4]).dot(xhats[4].T).dot(thetas[4])))
         pass

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

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

```
Training errors are:
 [0.23805129841210315, 0.1105261723492853, 0.12439276615592265, 0.1108956
9272844269, 0.10007705323623883]
```

QUESTIONS

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

(2) Why is this expected?

ANSWERS

(1) 5-order polynomial model has the lowest training error.

(2) Because a higher order polynomial model can always do the same or better than a low one since the higher one includes all pows of the lower one.

Generating new samples and testing error (5 points)

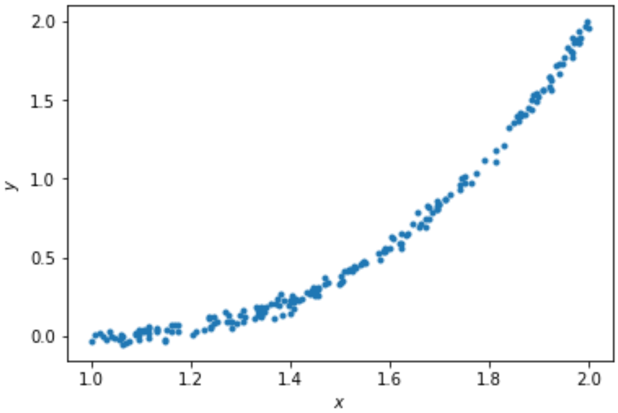
Here, we'll now generate new samples and calculate the testing error of polynomial models of orders 1 to 5.

In [11]:

```
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_tr
ain,))
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

Out[11]:

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



In [12]:

```
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(5
0)))
    else:
        xhat = np.vstack((x**(i+1), xhat))
        plot_x = np.vstack((plot_x[-2]**(i+1), plot_x))

    xhats.append(xhat)
```

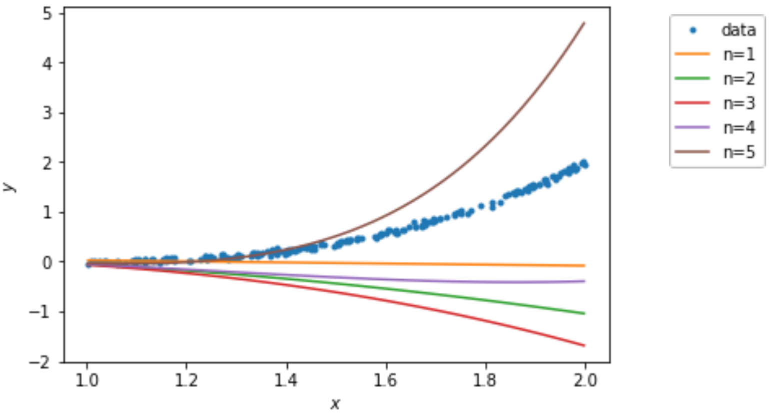
In [13]:

```
# 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(5
0)))
    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 [14]:

```
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.
testing_errors.append(0.5*(y.T.dot(y)-2*y.T.dot(xhats[0].T).dot(thetas
[0])+thetas[0].T.dot(xhats[0]).dot(xhats[0].T).dot(thetas[0])))
testing_errors.append(0.5*(y.T.dot(y)-2*y.T.dot(xhats[1].T).dot(thetas
[1])+thetas[1].T.dot(xhats[1]).dot(xhats[1].T).dot(thetas[1])))
testing_errors.append(0.5*(y.T.dot(y)-2*y.T.dot(xhats[2].T).dot(thetas
[2])+thetas[2].T.dot(xhats[2]).dot(xhats[2].T).dot(thetas[2])))
testing_errors.append(0.5*(y.T.dot(y)-2*y.T.dot(xhats[3].T).dot(thetas
[3])+thetas[3].T.dot(xhats[3]).dot(xhats[3].T).dot(thetas[3])))
testing_errors.append(0.5*(y.T.dot(y)-2*y.T.dot(xhats[4].T).dot(thetas
[4])+thetas[4].T.dot(xhats[4]).dot(xhats[4].T).dot(thetas[4])))
pass

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

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

```
Testing errors are:
 [81.129479259598455, 199.70224888594782, 294.05510960446202, 129.4231450
8486794, 105.33118027312869]
```

QUESTIONS

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

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

ANSWERS

(1) Shockingly, linear regression model has the lowest testing error. That is mainly because the testing data have different distribution as training data. And lower order polynomial model is less sensitive about testing dataset.

(2) A high order polynomial model is largely fitted to training data. It is fitted so much so that it often learns the character of the training dataset, which causes overfit. Once it is given a dataset that it has never seen before, it would not make a performance as good as the training dataset. A lower order polynomial model, however, usually has higher training error since it does not learn much about the character of the training dataset. But it has comparable testing error.