HW1 Yuchen Wang

1.

(a) i. In order to prove that Q^T is orthogonal, we need to prove that $(Q^T)^T Q^T = I$.

Since $(Q^T)^T = Q$, we need $QQ^T = I$.

Since we know that Q is orthogonal, $QQ^T = I$ is given.

Therefore, Q^T is orthogonal

In order to prove that Q^{-1} is orthogonal, we need to prove that $(Q^{-1})^T \cdot Q^{-1} = I$.

Since Q is orthogonal, we know that $Q^T = Q^{-1}$

$(Q^{-1})^T \cdot Q^{-1} = (Q^T)^T \cdot Q^T = QQ^T = I$

Therefore, Q^{-1} is orthogonal.

ii. Let λ be the eigenvalue and v be the corresponding eigenvector.

So, we have $Qv = \lambda v \Rightarrow \|Qv\|^2 = \|\lambda v\|^2 = |\lambda|^2 \|v\|^2$

Since Q is orthogonal, we know that $\|Qv\| = \|v\| \Rightarrow \|Qv\|^2 = \|v\|^2$

So, we got $|\lambda|^2 \|v\|^2 = \|v\|^2 \Rightarrow |\lambda|^2 = 1$

Therefore, Q has eigenvalues with norm 1.

iii. Based on the fact that Q is orthogonal, we have $\det(Q) = \det(Q^T)$

We also know that the determinant of the Identity matrix is 1

Therefore, $1 = \det(I) = \det(QQ^T) = \det(Q) \cdot \det(Q^T) = \det(Q) \cdot \det(Q)$

Since we have $1 = (\det(Q))^2$, the determinant of Q is either $+1$ or -1.

iv. We need to prove that $\|Qv\|^2 = \|v\|^2$, where v is an n dimensional vector.

$\|Qv\|^2 = v^T Q^T Qv = v^T I v = \|v\|^2$

(since Q is orthogonal)

Therefore, we proved that Q defines a length preserving transformation.

(6) i. The singular value decomposition of A is $A = UDV^T$, where the columns of U are the left-singular vectors and the columns of V are the right-singular vectors

$AA^T = UDV^TVD^TU^T = UDD^TU^T \Rightarrow U$ are eigenvectors

So, the eigenvectors of AA^T are the left-singular vectors of A

$A^TA = VD^TU^TUDV^T = VD^TDV^T \Rightarrow V$ are eigenvectors

So, the eigenvectors of A^TA are the right-singular vectors of A.

ii. The singular values of A are the positive square root of the eigenvalues of AA^T or A^TA

(4) i. False

ii. False

iii. True

iv. True

v. True

2.

(a) i. $P(H50|T) = \frac{P(H50)P(T|H50)}{P(T)}$

$= \frac{P(H50)P(T|H50)}{P(T|H50)P(H50) + P(T|H60)P(H60)}$

$= \frac{\frac{1}{2} \times \frac{1}{2}}{\frac{1}{2} \times \frac{1}{2} + \frac{4}{10} \times \frac{1}{2}}$

≈ 0.556

The posterior probability that this is an H50 coin is 0.556

ii. $P(H50|THHHH) = \frac{P(H50)P(THHHH|H50)}{P(THHHH)}$

$= \frac{P(H50)P(THHHH|H50)}{P(THHHH|H60)P(H50) + P(THHHH|H60)P(H60)}$

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

$= \frac{0.03125}{0.03125 + 0.04352}$ 0.07445

≈ 0.420

The probability that the coin is type H50 is 0.42

iii. We use 9HIT to represent the event that there are 9 heads in 10 flips.

$P(9HIT) = P(9HIT|H50)P(H50) + P(9HIT|H55)P(55) + P(9HIT|H60)P(60)$

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

$= 0.00236001$

↓ continue

$$P(H50 \mid 9H1T) = \frac{P(H50)P(9H1T \mid H50)}{P(9H1T)}$$

$$= \frac{\frac{1}{3} \times 0.5^{10}}{0.0023\,6001}$$

$$\approx 0.138$$

$$P(H55 \mid 9H1T) = \frac{P(H55)P(9H1T \mid H55)}{P(9H1T)}$$

$$= \frac{\frac{1}{3} \times 0.55^{9} \times 0.45}{0.0023\,6001}$$

$$\approx 0.293$$

$$P(H60 \mid 9H1T) = \frac{P(H60)P(9H1T \mid H60)}{P(9H1T)}$$

$$= \frac{\frac{1}{3} \times 0.6^{9} \times 0.4}{0.0023\,6001}$$

$$\approx 0.570$$

The probability that the coin is type H50 is 0.138
The probability that the coin is type H55 is 0.293
The probability that the coin is type H60 is 0.570

(b) $P(\text{Sienna} \mid \text{Hiked}) = \dfrac{P(\text{Sienna})P(\text{Hiked} \mid \text{Sienna})}{P(\text{Hiked} \mid \text{Sienna})P(\text{Sienna}) + P(\text{Hiked} \mid \text{Health})P(\text{Health}) + P(\text{Hiked} \mid \text{LA})P(\text{LA}) + P(\text{Hiked} \mid \text{Engineer})P(\text{Engineer})}$

$$= \frac{15\% \times 90\%}{9\% \times 15\% + 21\% \times 18\% + 0\% \times 24\% + 10\% \times 4\%}$$

$$= \frac{0.135}{0.2138}$$

$$\approx 0.634$$

The conditional probability that the student is from Sienna is 0.634

(c) $P(\text{Pregnant}|\text{Positive}) = \dfrac{P(\text{Pregnant})P(\text{Positive}|\text{Pregnant})}{P(\text{Positive}|\text{Preg})P(\text{Preg}) + P(\text{Positive}|\text{NPreg})P(\text{NPreg})}$

$= \dfrac{1\% \times 99\%}{99\% \times 1\% + 10\% \times 99\%}$

≈ 0.091

The probability that a woman is pregnant given she received a positive test is 0.091. Although the probability of positive test is high for the pregnant female, the population of non-pregnant female is large. Also, there is 10% probability that test returns positive when a female is not pregnant. Therefore, $P(\text{Pregnant}|\text{Positive})$ is not that large.

(d) $\mathbb{E}(Ax+b) = \mathbb{E}(Ax) + \mathbb{E}(b) = A\mathbb{E}(x) + b$

(e) $\text{cov}(Ax+b) = \mathbb{E}[((Ax+b) - \mathbb{E}(Ax+b))((Ax+b) - \mathbb{E}(Ax+b))^T]$

$= \mathbb{E}[((Ax+b) - (A\mathbb{E}(x)+b))((Ax+b) - (A\mathbb{E}(x)+b))^T]$

$= \mathbb{E}[(Ax - A\mathbb{E}(x))(Ax - A\mathbb{E}(x))^T]$

$= \mathbb{E}[A(x - \mathbb{E}(x))A^T(x - \mathbb{E}(x))^T]$

$= AA^T\mathbb{E}[(x - \mathbb{E}(x))(x - \mathbb{E}(x))^T]$

$= AA^T\text{cov}(x)$

3. I use the derivatives in the matrix cookbook.

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

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

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

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

(e) $\nabla_A f = B^T$

(f) $\nabla_A f = \nabla_A [\mathrm{tr}(BA) + \mathrm{tr}(A^T B) + \mathrm{tr}(A^T B)]$ ↓ 100, 103, 107 in matrix cook
$= B^T + B + (AB + BA)^T$

(g) $\nabla_A f = \nabla_A [\mathrm{tr}((A + XB)^T (A + XB))]$
$= \nabla_A [\mathrm{tr}(A^T A + XA^T B + XB^T A + X^2 BB^T)]$
$= \nabla_A [\mathrm{tr}(A^T A) + \mathrm{tr}(XA^T B) + \mathrm{tr}(XB^T A) + \mathrm{tr}(X^2 BB^T)]$ ↓ 115, 103, 100 in matrix cook
$= 2A + XB + XB$
$= 2A + 2XB$

4. Since we want $\frac{1}{2}\sum_{i=1}^{n}\|y^{(i)} - Wx^{(i)}\|^2$ to be the minimum, we would like to calculate

$\nabla_W \frac{1}{2}\sum_{i=1}^{n}\|y^{(i)} - Wx^{(i)}\|^2 = 0$

$$\begin{aligned}
\frac{1}{2}\sum_{i=1}^{n}\|y^{(i)} - Wx^{(i)}\|^2 &= \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - Wx^{(i)})^T(y^{(i)} - Wx^{(i)}) \\
&= \frac{1}{2}tr(Y - WX)^T(Y - WX) \\
&= \frac{1}{2}tr(Y^T - W^TX^T)(Y - WX) \\
&= \frac{1}{2}tr(Y^TY - Y^TWX - YW^TX^T + W^TWXX^T)
\end{aligned}$$

$$\begin{aligned}
\nabla_W \frac{1}{2}\sum_{i=1}^{n}\|y^{(i)} - Wx^{(i)}\|^2 &= \nabla_W \frac{1}{2}tr(Y^TY - Y^TWX - YW^TX^T + W^TWXX^T) \\
&= \frac{1}{2}\nabla_W tr(Y^TY) - tr(Y^TWX) - tr(YW^TX^T) + tr(W^TWXX^T) \\
&= \frac{1}{2}\nabla_W[-tr(Y^TWX) - tr(YW^TX^T) + tr(W^TWXX^T)] \\
&= \frac{1}{2}[-X^TY - YX^T + WXX^T + WXX^T] \\
&= \frac{1}{2}(-2X^TY + 2WXX^T) \\
&= -X^TY + WXX^T
\end{aligned}$$

We want $(-X^TY + WXX^T) = 0 \Rightarrow WXX^T = X^TY$

$$W = X^TY(XX^T)^{-1}$$

Therefore, the optimal W is $X^TY(XX^T)^{-1}$

5.

$$\frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \theta^T x^{(i)})^2 + \frac{\lambda}{2}\|\theta\|_2^2 = \frac{1}{2}(X\theta - y)^T(X\theta - y) + \frac{\lambda}{2}\|\theta\|_2^2$$

$$= \frac{1}{2}(\theta^T X^T X\theta + \theta^T X^T y - y^T X\theta + y^T y) + \frac{\lambda}{2}\theta^T\theta$$

$$= \frac{1}{2}\theta^T X^T X\theta - \theta^T X^T y + \frac{1}{2}y^T y + \frac{\lambda}{2}\theta^T\theta$$

drop $y^T y$

$$\nabla_\theta\left(\frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \theta^T x^{(i)})^2 + \frac{\lambda}{2}\|\theta\|_2^2\right) = \nabla_\theta\left(\frac{1}{2}\theta^T X^T X\theta - \theta^T X^T y + \frac{1}{2}y^T y + \frac{\lambda}{2}\theta^T\theta\right)$$

$$= X^T X\theta - X^T y + \lambda\theta$$

$$= (X^T X + \lambda I)\theta - X^T y$$

$$= (X^T X + \lambda I)\theta - X^T y$$

$$(X^T X + \lambda I)\theta = X^T y$$

$$\theta = (X^T X + \lambda I)^{-1} X^T y$$

So, $\theta = (X^T X + \lambda I)^{-1} X^T y$

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 2023, Prof. J.C. Kao, TAs: T.M, P.L, R.G, K.K, N.V, S.R, S.P, M.E

In [45]:

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

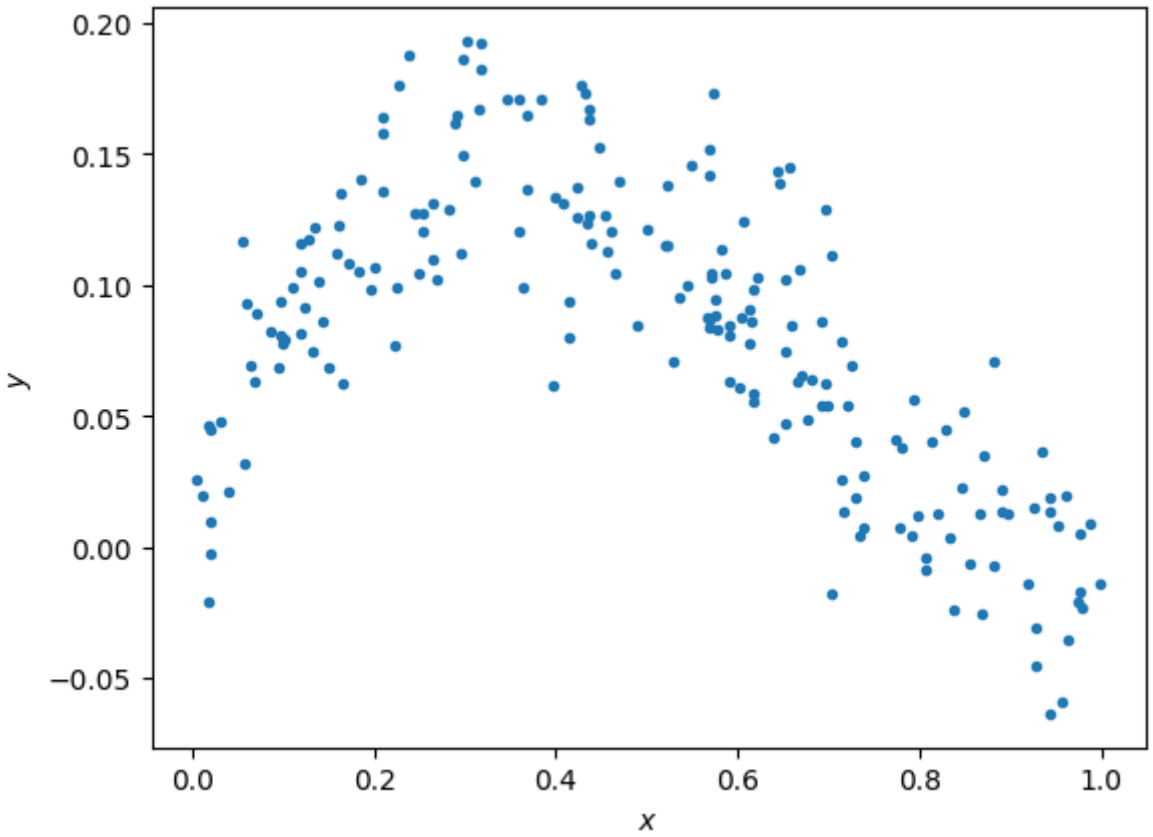
In [47]:

```
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[47]:

```
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 uniform since np.random.uniform is used.

(2) The generating distribution of the additive noise ϵ is normal(Gaussian) since np.random.normal is used. It has mean = 0 and sd = 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 [48]:

```
# 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))

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

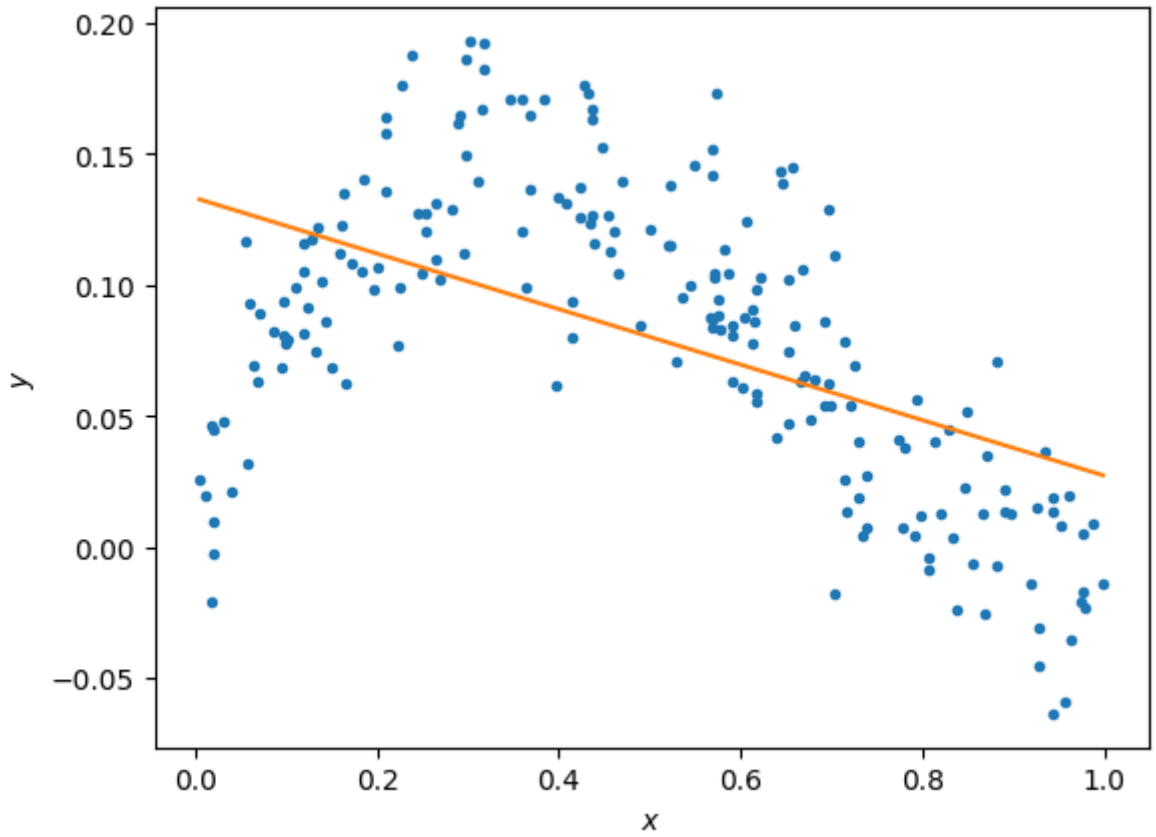
In [49]:

```
# 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[49]:

```
[<matplotlib.lines.Line2D at 0x7f78b95985e0>]
```



QUESTIONS

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

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

ANSWERS

(1) The linear model underfit the data.

(2) Make the model to be polynomial with higher orders.

Fitting data to the model (5 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 [50]:

```
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 c
#   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 np.arange(N):
    if i == 0:
        xhat = np.vstack((x, np.ones_like(x)))
    else:
        xhat = np.vstack((x**(i+1), xhat))
    thetas.append(np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y)))
    xhats.append(xhat)

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

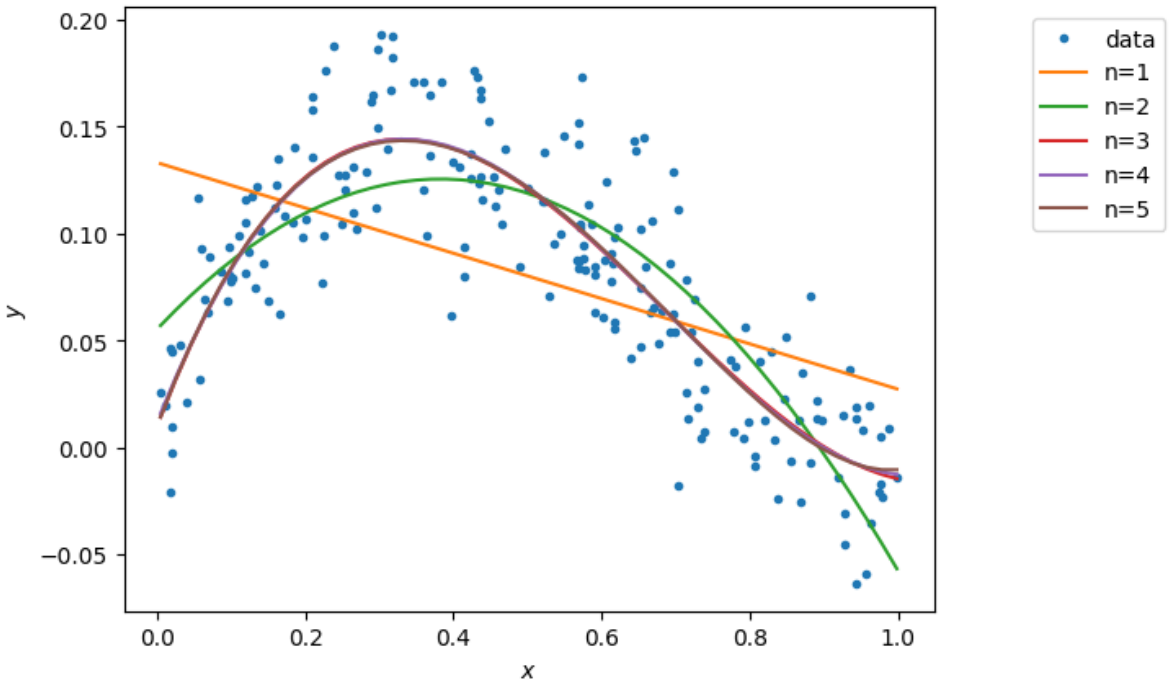
In [51]:

```
# 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 (5 points)

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

In [52]:

```
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 np.arange(N):
    training_errors.append(np.sqrt(np.sum((y - thetas[i].dot(xhats[i]))**2) / len(y)

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

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

```
Training errors are:
 [0.04878484486357111, 0.03305287008607351, 0.028582518785273934, 0.02
8575083088762804, 0.028568302706890546]
```

QUESTIONS

(1) What polynomial has the best training error?

(2) Why is this expected?

ANSWERS

(1) The polynomial with order 5 has the best training error.

(2) The higher order polynomial will give more curveness to the model. With the complexity of the model, the line could curve to the data points better. Therefore, the higher order polynomial has the best training error.

Generating new samples and testing error (5 points)

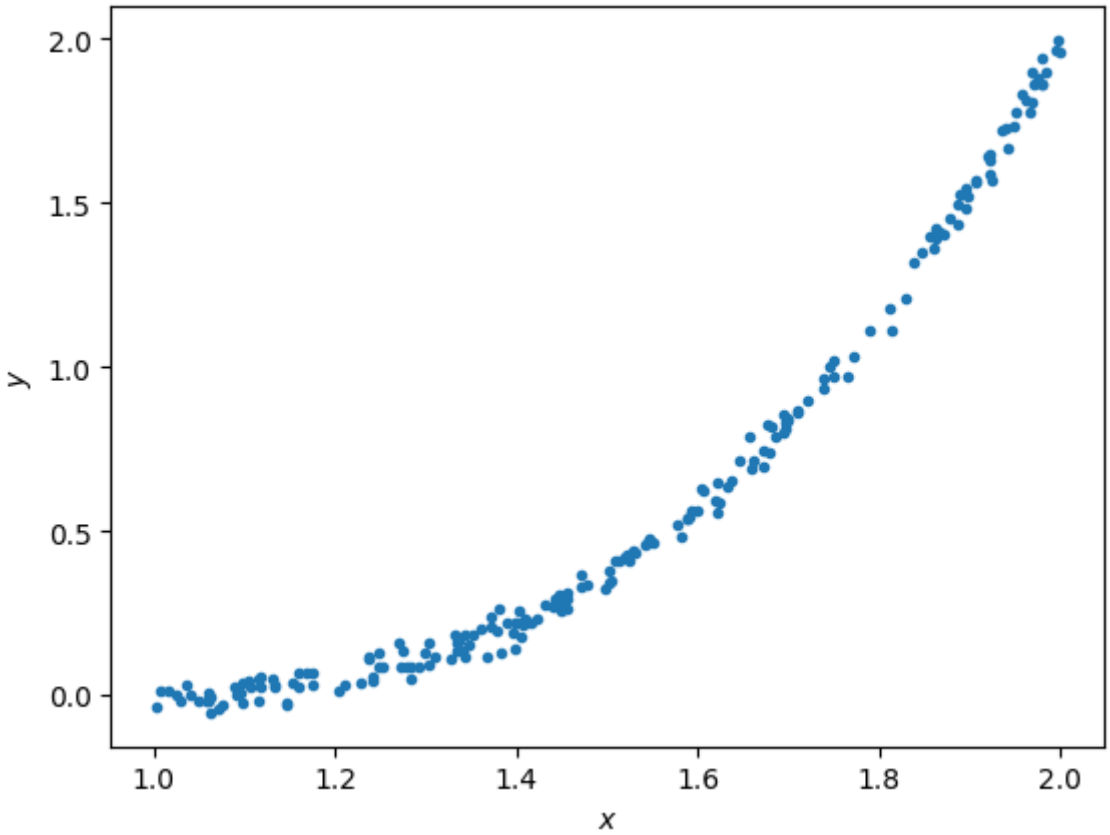
Here, we'll now generate new samples and calculate testing error of polynomial models of orders 1 to 5.

In [53]:

```
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[53]:

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



In [54]:

```
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)
```

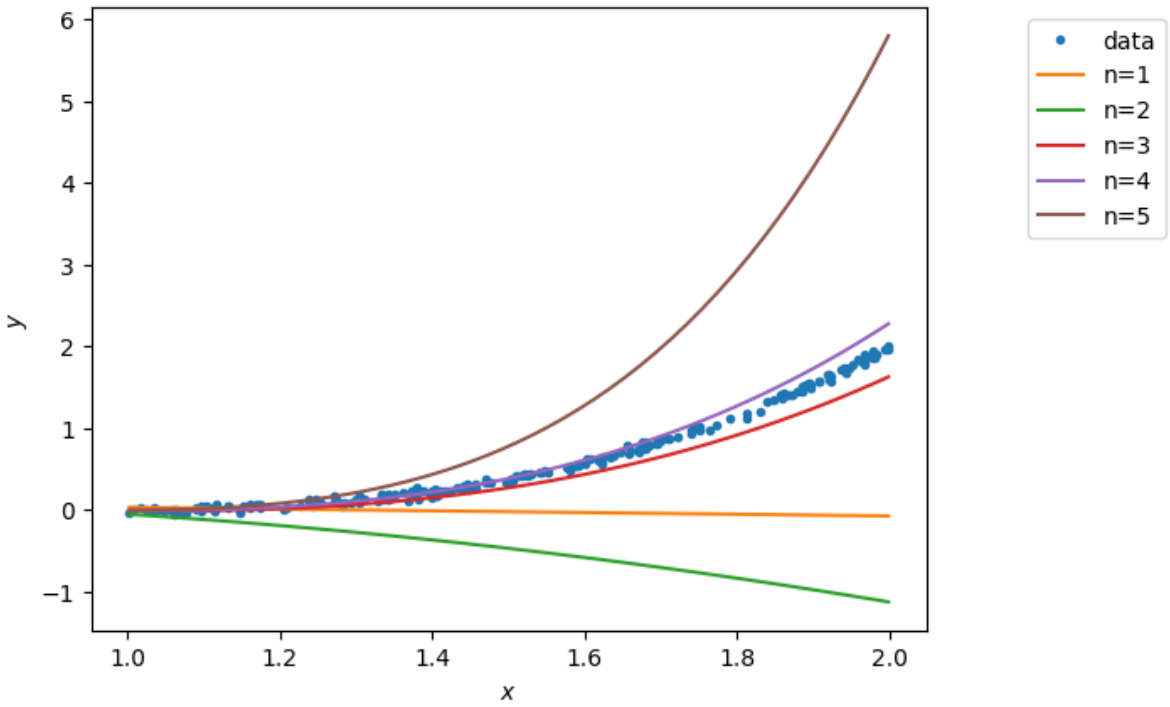
In [55]:

```
# 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 [56]:

```
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 np.arange(N):
    testing_errors.append(np.sqrt(np.sum((y - thetas[i].dot(xhats[i]))**2) / len(y))

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

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

```
Testing errors are:
 [0.8992310706681896, 1.4601093262169775, 0.17679641139679478, 0.10895
304124519173, 1.4659816443054745]
```

QUESTIONS

(1) What polynomial has the best testing error?

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

ANSWERS

(1) The polynomial with order 4 has the best testing error.

(2) Although the higher order polynomial will fit the training data well, it might cause overfit. This decreases the generalizability, which will cause large errors outside the original training data.