1) a) i) $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

$A \cdot A^T = I$ ✓

Eigenvalue

$$\det \begin{bmatrix} -\lambda & 1 \\ 1 & -\lambda \end{bmatrix} = \lambda^2 - 1 = (\lambda+1)(\lambda-1) = 0$$

$\lambda = -1, 1$

Eigenvector:

$\lambda_1 = -1$ $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\vec{x}_{-1} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

$\lambda_2 = 1$ $\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\vec{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

I notice that $\vec{x}_{-1}$ and $\vec{x}_1$ are orthogonal to each other and that $\lambda_1 = -\lambda_2$

ii) $Av = \lambda v$ by def of eigenvector, eigenvalue

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

$(Av)^T(Av) = \lambda^2 |v|^2$

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

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

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

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

iii) Since A is orthogonal, we have $A^T = A^{-1}$ and thus $A^{-1}A = I = AA^{-1}$

$\Rightarrow A^TA = AA^T$

Thus A is normal

Since A is normal, $A = UDU^*$ where D is diagonal and the columns are eigenvectors

Thus the eigenvectors are orthogonal

iv) Vector x will be rotated or flipped

b) i) Left singular vectors of A are orthonormal eigenvectors of AA^T

Right singular vectors of A are orthonormal eigenvectors of A^TA

ii) Nonzero singular values of A are the square roots of non-zero eigenvalues of both AA^T and A^TA

c) i) False, consider I_2

ii) False, not true in all cases - $A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$

iii) True

iv) True

v) True

2) a) i) $P(\text{coin} = \text{HSO} \mid \text{lands} = \text{tails}) = \dfrac{P(C = \text{HSO} \cap l = \text{tails})}{P(l = \text{tails})}$

$= P(l = \text{tails} \mid C = \text{HSO}) P(C = \text{HSO})$

$$P(l=t,t,t,t \mid C=HSO)P(C=HSO) + P(l=t,t,t,t \mid C=HBO)P(C=HBO)$$

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

$$= 0.555$$

ii) $P(C=HSO \mid l1=T \cap l2=H \cap l3=H \cap l4=H)$

$$= \frac{P(C=HSO \cap l1=T \cap l2=H \cap l3=H \cap l4=H)}{P(l1=T \cap l2=H \cap l3=H \cap l4=H)}$$

$$= \frac{P(l1=T \mid C=HSO)P(l2=H \mid C=HSO)P(l3=H \mid C=HSO)P(l4=H \mid C=HSO)P(C=HSO)}{P(l1=T \mid C=HSO)P(l2=H \mid C=HSO)P(l3=H \mid C=HSO)P(l4=H \mid C=HSO)P(C=HSO) + P(l1=T \mid C=HBO)P(l2=H \mid C=HBO)P(l3=H \mid C=HBO)P(l4=H \mid C=HBO)P(C=HBO)}$$

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

$$= 0.419$$

iii) $P(l5=9H/10 \mid C=HSO) = 0.5^{10}$

$P(l5=9H/10 \mid C=HSS) = 0.55^9 \cdot 0.45$

$P(l5=9H/10 \mid C=HBO) = 0.60^9 \cdot 0.40$

$$P(C=HSO \mid l5=9H/10) = \frac{0.5^{10} \cdot 1/3}{0.5^{10} \cdot 1/3 + 0.55^9 \cdot 0.45 \cdot 1/3 + 0.60^9 \cdot 0.40 \cdot 1/3}$$

$$= 0.137$$

$$P(C=HSS \mid LS=9H/10) = \frac{0.5^{10} \cdot 1/3}{0.5^{10} \cdot 1/3 + 0.55^9 \cdot 0.45 \cdot 1/3 + 0.60^9 \cdot 0.40 \cdot 1/3}$$

$$= 0.292$$

$$P(C=HOD \mid LS=9H/10) = \frac{0.5^{10} \cdot 1/3}{0.5^{10} \cdot 1/3 + 0.55^9 \cdot 0.45 \cdot 1/3 + 0.60^9 \cdot 0.40 \cdot 1/3}$$

$$= 0.569$$

b) $$P(P=T \mid T=T) = \frac{P(P=T \cap T=T)}{P(T=T)}$$

$$= \frac{P(T=T \mid P=T) \cdot P(P=T)}{P(T=T \mid P=T)P(P=T) + P(T=T \mid P=F)P(P=F)}$$

$$= \frac{(0.99)(0.01)}{(0.99)(0.01) + (0.10)(0.99)}$$

$$= 0.0909$$

Not that likely b/c false positive is relatively high - if false pos. were lower, could be more confident

c) $E[Ax+b) = A\,E[x] + b$ b/c linearity

d) $Cov(Ax+b) = E[(Ax+b - E[Ax+b])(Ax+b-E[Ax+b])^T]$

$= E[(Ax+b-AE[x]-b)\cdot(Ax+b-AE[x]-b)^T]$

$= E[(A(x-E[x]))(A(x-E[x]))^T$

$= E[A(x-E[x])(x-E[x])^T A^T]$

$= A A^T Cov(x)$

3) a) $$x^T A y = [x_1, \ldots, x_n] \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

$$= x_1(a_{11}y_1 + \dots + a_{1m}y_m) +$$
$$x_2(a_{21}y_2 + \dots + a_{2m}y_m) +$$
$$\dots +$$
$$x_n(a_{n1}y_1 + \dots + a_{nm}y_m)$$

$$\nabla_x x^T A y = \begin{bmatrix} \frac{\partial x^T A y}{\partial x_1} \\ \vdots \\ \frac{\partial x^T A y}{\partial x_n} \end{bmatrix} = Ay$$

b) $$x^T A y = [x_1, \dots, x_n] \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

$$= y_1(x_1 a_{11} + \dots + x_n a_{n1}) +$$
$$y_2(x_1 a_{12} + \dots + x_n a_{n2}) +$$
$$\dots +$$
$$y_m(x_1 a_{1m} + \dots + x_n a_{nm})$$

$$\nabla_y x^T A y = \begin{bmatrix} \frac{\partial x^T A y}{\partial y_1} \\ \vdots \\ \frac{\partial x^T A y}{\partial y_m} \end{bmatrix} = x^T A$$

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

$$= \begin{bmatrix} x_1 y_1 & \cdots & x_1 y_m \\ \vdots & \ddots & \vdots \\ x_n y_1 & \cdots & x_n y_m \end{bmatrix}$$

$$= x y^T$$

d) $$f = x^T A x + b^T x$$

$$= \begin{bmatrix} x_1, \dots, x_n \end{bmatrix} \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$+ \begin{bmatrix} b_1, \dots, b_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$= x_1 \left(x_1 a_{11} + \dots + x_n a_{n1} \right) + x_2 \left(x_1 a_{12} + \dots + x_n a_{n2} \right)$$

$$+ \dots + x_n \left(x_1 a_{1n} + \dots + x_n a_{nn} \right)$$

$$+ b_1 x_1 + \dots + b_n x_n$$

$$\nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} = \begin{bmatrix} 2a_{11}x_1 + a_{12}x_2 + \dots + x_n a_{n1} + \dots + b_1 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + 2x_n a_{nn} + \dots + b_n \end{bmatrix}$$

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

e) $f = tr(AB)$

$$= tr\left(\begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{bmatrix} \right)$$

$$= tr\left(\begin{bmatrix} a_{11}b_{11} + \dots + a_{1m}b_{m1} & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & a_{n1}b_{1n} + \dots + a_{nm}b_{mn} \end{bmatrix} \right)$$

$$\nabla_A f = \begin{bmatrix} \frac{\partial f}{\partial a_{11}} & \cdots & \frac{\partial f}{\partial a_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial a_{n1}} & \cdots & \frac{\partial f}{\partial a_{nm}} \end{bmatrix}$$

$$= \begin{bmatrix} b_{11} & \cdots & b_{m1} \\ \vdots & \ddots & \vdots \end{bmatrix} = B^T$$

$\begin{bmatrix} b_{in} & \cdots & b_{mn} \end{bmatrix}$

4) Derive optimal w for $\min_w \frac{1}{2} \sum_{i=1}^{n} \left| y^{(i)} - wx^{(i)} \right|^2$

$\frac{1}{2} \sum_{i=1}^{n} \left| y^{(i)} - wx^{(i)} \right|^2$

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

$= \frac{1}{2} tr \left(\left(Y^T - X^T w^T \right) \left(Y - wX \right) \right)$

$y^{(i)} \in \mathbb{R}^{m \times 1}$

$w \in \mathbb{R}^{m \times c}$

$x^{(i)} \in \mathbb{R}^{c \times 1}$

$Y^T \in \mathbb{R}^{n \times m}$

$w^T \in \mathbb{R}^{c \times m}$

$X^T \in \mathbb{R}^{n \times c}$

where $Y^T = \begin{bmatrix} — & y^{(1)T} & — \\ & \vdots & \\ — & y^{(n)T} & — \end{bmatrix}$

$X^T = \begin{bmatrix} — & x^{(1)T} & — \\ & \vdots & \\ — & x^{(n)T} & — \end{bmatrix}$

$\frac{\partial f}{\partial w} = 0 = \frac{\partial}{\partial w} \frac{1}{2} tr \left(\left(Y^T - X^T w^T \right) \left(Y - wX \right) \right)$

$= \frac{1}{2} \frac{\partial}{\partial w} tr \left(Y^T Y - Y^T wX - X^T w^T Y + X^T w^T wX \right)$

$= \frac{1}{2} \frac{\partial}{\partial w} tr \left(Y^T Y \right) - tr \left(Y^T wX \right) - tr \left(X^T w^T Y \right) + tr \left(Xw^T wX \right)$

$= \frac{1}{2} \frac{\partial}{\partial w} - tr \left(Y^T wX \right) - tr \left(X^T w^T Y \right) + tr \left(Xw^T wX \right)$

$= \frac{1}{2} \left(-2YX^T + 2wXX^T \right)$

$= wXX^T - YX^T$

$wXX^T \left(XX^T \right)^{-1} = YX^T \left(XX^T \right)^{-1}$

$w = YX^T \left(XX^T \right)^{-1}$

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 239AS, Winter Quarter 2019, Prof. J.C. Kao, TAs M. Kleinman and A. Wickstrom and K. Liang and W. Chuang

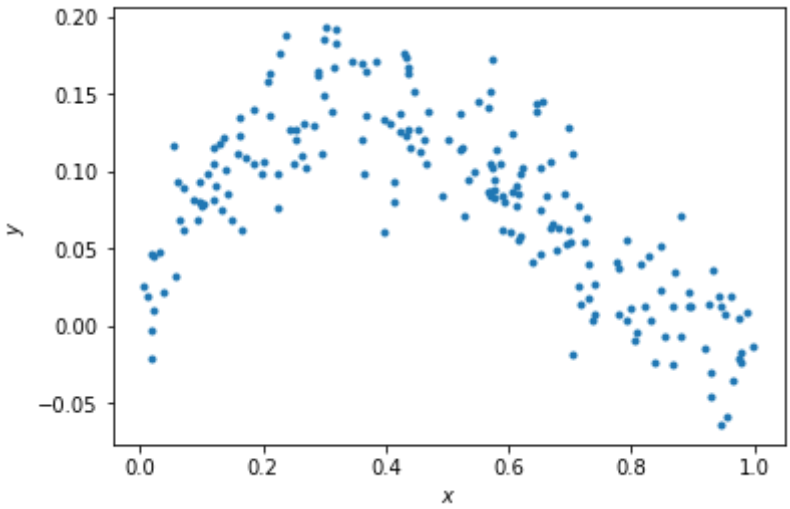
In [26]:
```
import numpy as np
import matplotlib.pyplot as plt

#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 [27]:
```
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[27]:
```
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 from a uniform distribution between 0 and 1

(2) ϵ is from a normal distribution with mean 0 and stddev 0.03

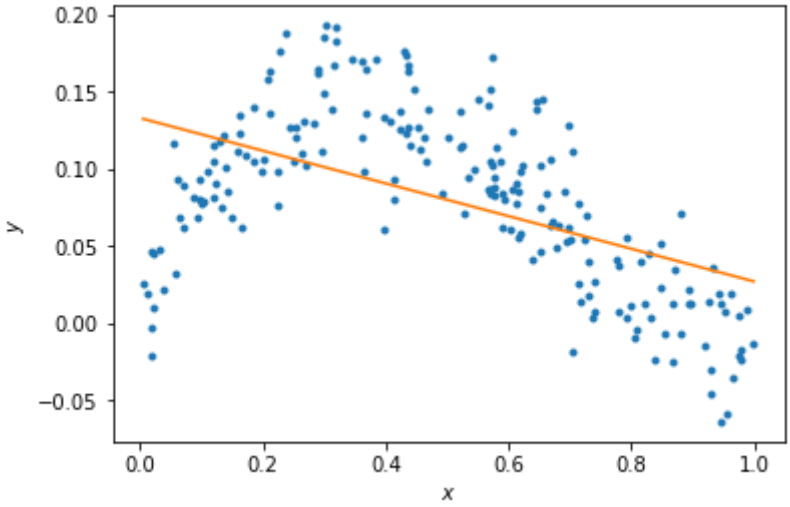
Fitting data to the model (5 points)

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

In [28]:
```
# 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

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

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

In [29]:
```
# 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[29]:
```
[<matplotlib.lines.Line2D at 0x113054710>]
```



QUESTIONS

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

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

```
### ANSWERS

(1) Underfits the data

(2) Increase complexity of the model (add x^2 and x^3 terms)
```

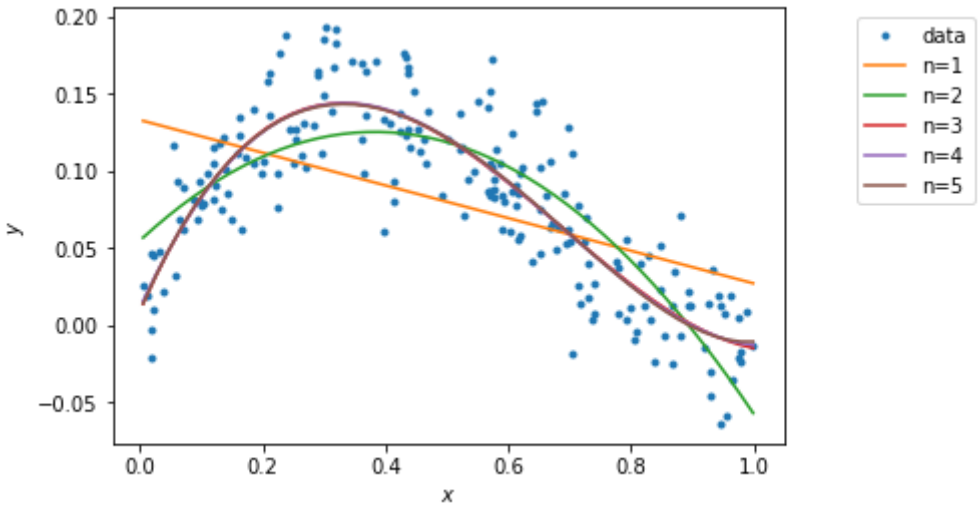
Fitting data to the model (10 points)

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 [30]:

```
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 polynom
#   i.e., thetas[0] is equivalent to theta above.
#   i.e., thetas[1] should be a length 3 np.array with the coefficients of
#   ... etc.
for i in np.arange(N):
    xhats.append(xhat if i == 0 else np.vstack((x**(i+1), xhats[i-1])))
    thetas.append(np.linalg.inv((xhats[i]).dot(xhats[i].T)).dot(xhats[i].dc

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

In [31]:
```
# 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 [32]:
```
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
for i in np.arange(N):
    training_errors.append(0.5*np.sum(np.square(y - (xhats[i].T).dot(thetas

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

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

```
Training errors are:
 [0.2379961088362701, 0.10924922209268531, 0.08169603801105374, 0.0816535
3735296982, 0.08161479195525295]
```

QUESTIONS

(1) What polynomial has the best training error?

(2) Why is this expected?

ANSWERS

(1) Polynomial with degree 5

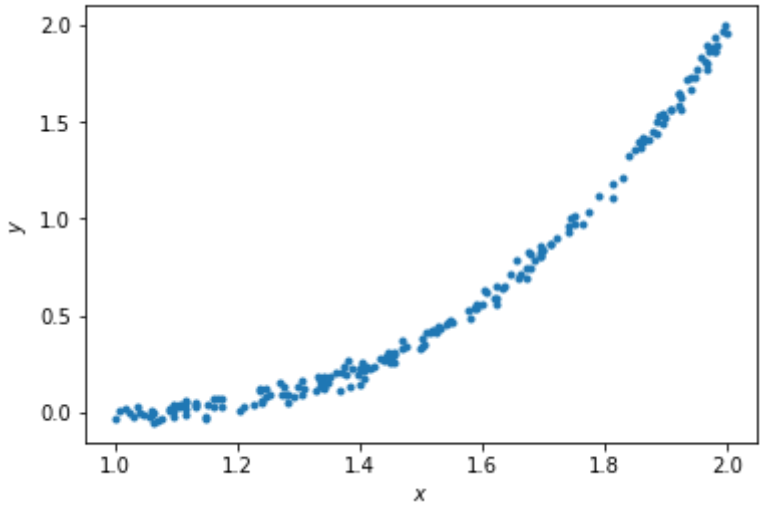
(2) There are more than 5 datapoints, and so training error is strictly decreasing as polynomial degree increases (because polynomial degree 5 encapsulates polynomial degree 4, and so on, thus, it can use the x^5 to decrease training error even further)

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.

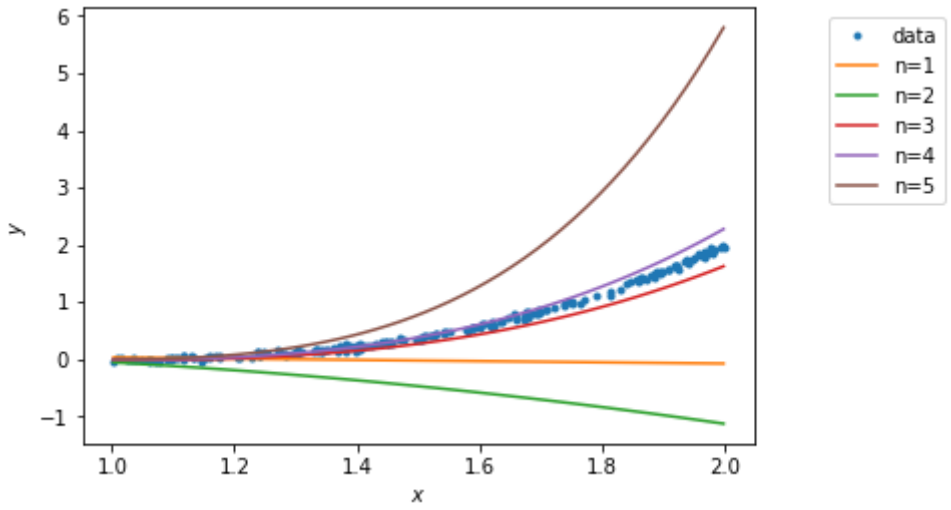
In [33]:
```
x = np.random.uniform(low=1, high=2, size=(num_train,))
y = x - 2*x**2 + x**3 + np.random.normal(loc=0, scale=0.03, size=(num_train
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

Out[33]:
```
Text(0, 0.5, '$y$')
```



In [34]:
```
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 [35]:
```
# 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 [36]:
```
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 or
for i in np.arange(N):
    testing_errors.append(0.5*np.sum(np.square(y - (xhats[i].T).dot(thetas[

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

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

```
Testing errors are:
 [80.86165184550586, 213.19192445057894, 3.1256971082763925, 1.1870765189
474703, 214.91021817652626]
```

QUESTIONS

(1) What polynomial has the best testing error?

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

ANSWERS

(1) Polynomial degree 4

(2) Polynomial degree 5 had overfit the training data which came from an X uniformly distributed over 0 and 1, whereas the testing data has an X uniformly distributed over 1 and 2. The x^5 term then begins to cause very high error, because the testing data seems unlike the training data, despite the underlying function being identical.