EE239AS
HW1
Yutong Lu
005027184

1. Linear algebra refresher

(a) i. Let $A = \begin{bmatrix} \frac{4}{5} & \frac{3}{5} \\ -\frac{3}{5} & \frac{4}{5} \end{bmatrix}$ satisfying $AA^T = I$.

$Av = \lambda v \Rightarrow (A - \lambda I)v = 0 \Rightarrow \det(A - \lambda I) = 0$

$\Rightarrow \det \begin{bmatrix} \frac{4}{5} - \lambda & \frac{3}{5} \\ -\frac{3}{5} & \frac{4}{5} - \lambda \end{bmatrix} = 0 \Rightarrow (\frac{4}{5} - \lambda)^2 - \frac{3}{5}(-\frac{3}{5}) = 0$

$\Rightarrow \frac{16}{25} + \lambda^2 - \frac{8}{5}\lambda + \frac{9}{25} = 0 \Rightarrow \lambda^2 - \frac{8}{5}\lambda + 1 = 0 \Rightarrow \lambda_1 = 0.8 + 0.6i,\ \lambda_2 = 0.8 - 0.6i$

When $\lambda_1 = 0.8 + 0.6i$,

$\begin{bmatrix} \frac{4}{5} - \frac{4}{5} - \frac{3}{5}i & \frac{3}{5} \\ -\frac{3}{5} & \frac{4}{5} - \frac{4}{5} - \frac{3}{5}i \end{bmatrix} v_1 = 0 \Rightarrow \begin{bmatrix} -\frac{3}{5}i & \frac{3}{5} \\ -\frac{3}{5} & -\frac{3}{5}i \end{bmatrix} v_1 = 0 \Rightarrow v_1 = \begin{bmatrix} -\frac{\sqrt{2}}{2}i \\ \frac{\sqrt{2}}{2} \end{bmatrix}$

when $\lambda_2 = 0.8 - 0.6i$,

$\begin{bmatrix} \frac{4}{5} - \frac{4}{5} + \frac{3}{5}i & \frac{3}{5} \\ -\frac{3}{5} & \frac{4}{5} - \frac{4}{5} + \frac{3}{5}i \end{bmatrix} v_2 = 0 \Rightarrow \begin{bmatrix} \frac{3}{5}i & \frac{3}{5} \\ -\frac{3}{5} & \frac{3}{5}i \end{bmatrix} v_2 = 0 \Rightarrow v_2 = \begin{bmatrix} \frac{\sqrt{2}}{2}i \\ \frac{\sqrt{2}}{2} \end{bmatrix}$

Eigenvalues can involve complex numbers rather than real numbers, and in this case, the two eigenvalues are complex conjugate to each other. and both of them have a norm of 1.
As for eigenvectors, they are also complex conjugate to each other, and $v_1^T v_2 = 0$, they are orthogonal.

ii. $AA^T = I$,

$AV = \lambda V \Rightarrow V^T A^T = \lambda^T V^T \Rightarrow V^T A^T A V = (\lambda V)(\lambda^T V^T)$

$\Rightarrow V^T (A^T A) V = |\lambda|^2 V V^T$

$\Rightarrow V^T V = |\lambda|^2 V V^T \Rightarrow \|V\|^2 = |\lambda|^2 \|V\|^2 \Rightarrow$ A has eigenvalues with norm 1.

iii. Assume u and v are eigenvectors corresponding to eigenvalues λ_u and λ_v of the matrix A.

$$\begin{cases} Au = \lambda_u \cdot u \Rightarrow u^T A = \lambda_u \cdot u^T & ① \\ Av = \lambda_v \cdot v & ② \end{cases}$$

By multiplying ① with ②, $u^T \underbrace{\cancel{A}\, A}_{I} v = \lambda_u \cdot u^T \lambda_v \cdot v \Rightarrow u^T v = \lambda_u \cdot \lambda_v \cdot u^T v$

$\Rightarrow u^T v (1 - \lambda_u \cdot \lambda_v) = 0$, then $u^T v = 0$ and $\lambda_u \cdot \lambda_v = 1$ due to arbitrary

Therefore, distinct eigenvectors are orthogonal.

iv. Before multiplying by A, we have unit eigenvectors on an unit circle. By plotting A, it can be observed that A distorts the unit circle, it scales space in direction $v^{(i)}$ by λ_i.

(b) i. The left singular vectors of A are the eigenvectors of AA^T. The right singular vectors of A are the eigenvectors of $A^T A$.

ii. The non zero singular values of A are the square roots of the eigenvalues of ATA and it is also true for A^TA.

(c) i. FALSE. There are linear operators with no eigenvalues, and actually, it should be that there are at most n distinct eigenvalues.

ii. FALSE. For example, $A=\begin{bmatrix}1&0\\0&2\end{bmatrix}$ and $\begin{bmatrix}1\\0\end{bmatrix}$ is the eigenvector of eigenvalue 1, and $\begin{bmatrix}0\\1\end{bmatrix}$ is the eigenvector of eigenvalue 2. However, $\begin{bmatrix}1\\1\end{bmatrix}$ is not an eigenvector of A.

iii. TRUE. $x^TAx = x^TQ\Lambda Q^Tx = (Q^Tx)^T\Lambda(Q^Tx) = \sum_{i=1}^{n}\lambda_i(q_i^Tx)^2 \geq \lambda_n\sum_{i=1}^{n}(q_i^Tx)^2 = \lambda_n\|x\|^2$

Then, $\lambda_n\|x\|^2 \geq 0$, and this ensures its eigenvalues are non-negative.

iv. TRUE. Suppose $L: \mathbb{R}^n \to \mathbb{R}^m$, then $\text{rank}(L) (= \dim R(L)) + \text{nullity}(L) = m$, then the no. of non-zero eigenvalues must be less or equal to the rank of a matrix.

v. FALSE Since these two eigenvectors are corresponding to one eigenvalue, then they can be linearly independent so that their sum may not be an eigenvector.

2. (a) i. $p(\text{head}|\text{"H50"}) = 0.5$, $p(\text{head}|\text{"H60"}) = 0.6$, find $p(\text{"H50"}|\text{tail})$?

$$p(\text{"H50"}|\text{tail}) = \frac{p(\text{tail}|\text{"H50"})\,p(\text{"H50"})}{p(\text{tail})} = \frac{p(\text{tail}|\text{"H50"})\,p(\text{"H50"})}{\sum_H p(\text{tail}|\text{"H"})}$$

$$= \frac{p(\text{tail}|\text{"H50"})\,p(\text{"H50"})}{p(\text{tail}|\text{"H50"})\,p(\text{"H50"}) + p(\text{tail}|\text{"H60"})\,p(\text{"H60"})}$$

$$= \frac{0.5\times0.5}{0.5\times0.5+(1-0.6)\times0.5} = \frac{5}{9}$$

ii. If the coin is of type H50, the probability of THHH is $(0.5)^4$

If the coin is of type H60, the probability of THHH is $0.4\times(0.6)^3$

The probability of it will be H50 is:

$$\frac{(0.5)^4}{(0.5)^4+0.4\times(0.6)^3} = 0.42$$

iii If the coin is of type H50, the probability of 9 heads out of ten flips is $C_{10}^{9}(0.5)^9C_{10}^{1}(0.5)$.

If the coin is of type H55, the probability of 9 heads out of ten flips is $C_{10}^{9}(0.55)^9C_{10}^{1}(1-0.55)$

If the coin is of type H60, the probability of 9 heads out of ten flips is $C_{10}^{9}(0.6)^9C_{10}^{1}(1-0.6)$

Then, the probability of coin being H50 is $\dfrac{C_{10}^{9}(0.5)^9C_{10}^{1}(0.5)}{C_{10}^{9}(0.5)^9C_{10}^{1}(0.5)+C_{10}^{9}(0.55)^9C_{10}^{1}(0.45)+C_{10}^{9}(0.6)^9C_{10}^{1}(0.4)}$ by X.

$$= \frac{(0.5)^{10}}{(0.5)^{10}+(0.55)^9(0.45)+(0.6)^9(0.4)} = 0.138$$

Scanned by CamScanner

The probability of the coin being H55 is $\frac{(0.55)^9(0.45)}{(0.5)^{10}+(0.55)^9(0.45)+(0.6)^9(0.4)} = 0.293$

The probability of the coin being H60 is $\frac{(0.6)^9(0.4)}{(0.5)^{10}+(0.55)^9(0.45)+(0.6)^9(0.4)} = 0.569$

(b) Find $p(\text{pregnant}|+ve)$

$$p(\text{pregnant}|+ve) = \frac{p(+ve|\text{pregnant})\,p(\text{pregnant})}{p(+ve|\text{pregnant})\,p(\text{pregnant}) + p(+ve|\text{not})\,p(\text{not})}$$

$$= \frac{99\% \times 1\%}{99\% \times 1\% + 10\% \times 99\%} = 9.09\%$$

Since 99% of the woman population is not pregnant at any time point, even the test indicates positive, it may just be a false positive, and this can also be inferred by high false positive rate (=10%) given in the question.

(c) $E[Ax+b] = \sum_x (Ax+b)p(x) = \sum_x Axp(x) + \sum_x bp(x) = A\sum_x xp(x) + b\sum_x p(x)$

$= AE[x] + b$

(d) $\text{cov}(x) = E((x-Ex)(x-Ex)^T)$, Find $\text{cov}(Ax+b)$

Let $Y = Ax+b$, then $\text{cov}(Y) = E[[Y-EY][Y-EY]^T]$

$Y - E[Y] = Ax+b - E[Ax+b] = Ax+b - AE[x] - b = Ax - AE[x] = A(x-E[x])$

$\Rightarrow \text{cov}(Y) = E\left(A(x-E[x])(x-E[x])^T A^T\right) = AE((x-E(x))(x-E(x))^T)A^T = A\,\text{cov}(x)\,A^T$

3. (a) $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, $A \in \mathbb{R}^{n\times m}$, find $\nabla_x x^T A y$

Let $z = x^T A y$

$$\frac{\partial z}{\partial x_1} = \frac{\partial \sum_i \sum_j x_i A_{ij} y_j}{\partial x_1} = \frac{\partial(\sum x_1 A_{1j} y_j + \cdots + \sum x_n A_{nj} y_j)}{\partial x_1} = \frac{\partial \sum x_1 A_{1j} y_j}{\partial x_1}$$

$= \sum_j A_{1j} y_j$

$\frac{\partial z}{\partial x} = Ay \quad \Rightarrow \nabla_x x^T A y = Ay$

(b) $\nabla_y x^T A y$ Let $z = x^T A y$. $\frac{\partial z}{\partial y_1} = \frac{\partial \sum_i \sum_j x_i A_{ij} y_j}{\partial y_1} = \sum_i x_i A_{i1} \Rightarrow \frac{\partial z}{\partial y} = x^T A$

(c) $\nabla_A x^T A y$, Let $z = x^T A y = \sum_i \sum_j x_i A_{ij} y_j$

$\frac{\partial z}{\partial A_{11}} = \frac{\partial \sum_i \sum_j x_i A_{ij} y_j}{\partial A_{11}} = x_1 y_1$

$\frac{\partial z}{\partial A_{12}} = \frac{\partial \sum_i \sum_j x_i A_{ij} y_j}{\partial A_{12}} = x_1 y_2$

$\vdots$

$$\Rightarrow \nabla_A x^T A y = \begin{bmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_m \\ \vdots & & \ddots & \vdots \\ x_n y_1 & & \cdots & x_n y_m \end{bmatrix} = xy^T$$

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

$\nabla_x f = (A+A^T)x + b$ since $\nabla_x(x^T A x) = \frac{\partial x^T A x}{\partial x} = \frac{\partial \sum a_{ij} x_i x_j}{\partial x}$

$\frac{\partial x^T A x}{\partial x_1} = \sum_{j=1}^n a_{1j} x_j + \sum_{i=1}^n a_{i1} x_i$

$= (Ax)_1 + (A^T x)_1$

$\Rightarrow \frac{\partial x^T A x}{\partial x} = (A+A^T)x$

for $b^T x$ part

$b^T x = b_1 x_1 + \cdots + b_n x_n$

$\Rightarrow \frac{\partial b^T x}{\partial x} = b$

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

(e) $f = tr(AB)$, find $\nabla_A f$

$$tr(AB) = tr\left[\begin{matrix} — a_1 — \\ — a_2 — \\ \vdots \\ — a_n — \end{matrix}\right]\left[\begin{matrix} | & | & & | \\ b_1 & b_2 & \cdots & b_n \\ | & | & & | \end{matrix}\right]$$

$$= tr\begin{bmatrix} a_1b_1 & a_1b_2 & \cdots & a_1b_n \\ a_2b_1 & a_2b_2 & & a_2b_n \\ \vdots & & \ddots & \vdots \\ a_nb_1 & \cdots & & a_nb_n \end{bmatrix}$$

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

$$\Rightarrow \frac{\partial\, tr(AB)}{\partial a_{ij}} = b_{ji}$$

$$\Rightarrow \nabla_A f = \frac{\partial\, tr(AB)}{\partial A} = B^T$$

4. $L = \min_W \frac{1}{2} \sum_{i=1}^{n} \|y^{(i)} - Wx^{(i)}\|_F^2 = \min_W \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - Wx^{(i)})^T (y^{(i)} - Wx^{(i)})$

$$= \min_W Tr\left(\frac{1}{2}(Y - XW)^T (Y - XW)\right)$$

$$\Leftrightarrow \min_W Tr\left[(Y - XW)^T (Y - XW)\right)$$

$$= \min_W Tr(Y^TY - Y^TXW - W^TX^TY + W^TX^TXW)$$

$$= \min_W Tr(Y^TY - 2Y^TXW + W^TX^TXW)$$

$$\nabla_W L = 0 \Rightarrow \frac{\partial Tr(Y^TY - 2Y^TXW + W^TX^TXW)}{\partial W} = 0$$

$$\Rightarrow 0 - 2X^TY + 2X^TXW = 0$$

$$\Rightarrow W = (X^TX)^{-1}X^TY \text{ is the optimal } W$$

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 2018, Prof. J.C. Kao, TAs C. Zhang and T. Xing

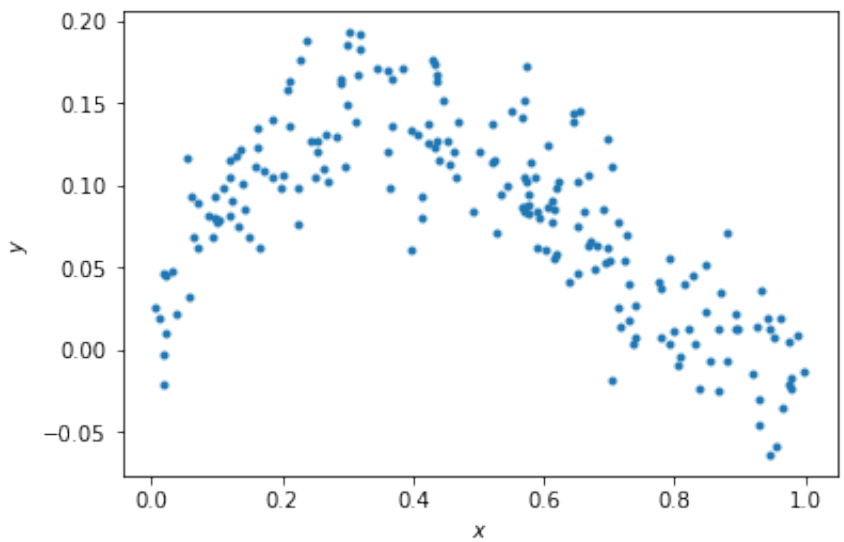
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

# x = np.arange(0,3*np.pi, 0.1)
# y = np.sin(x)

# plt.plot(x, y, label = 'Sine')
# plt.xlabel('x axis label')
# plt.ylabel('y axis label')
# plt.title('Sine')
# plt.legend()
# plt.grid()
# plt.show()

#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_tra
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(Gaussian) distribution with a mean of 0, standard deviation of 0.03, and the output shape of num_train samples.

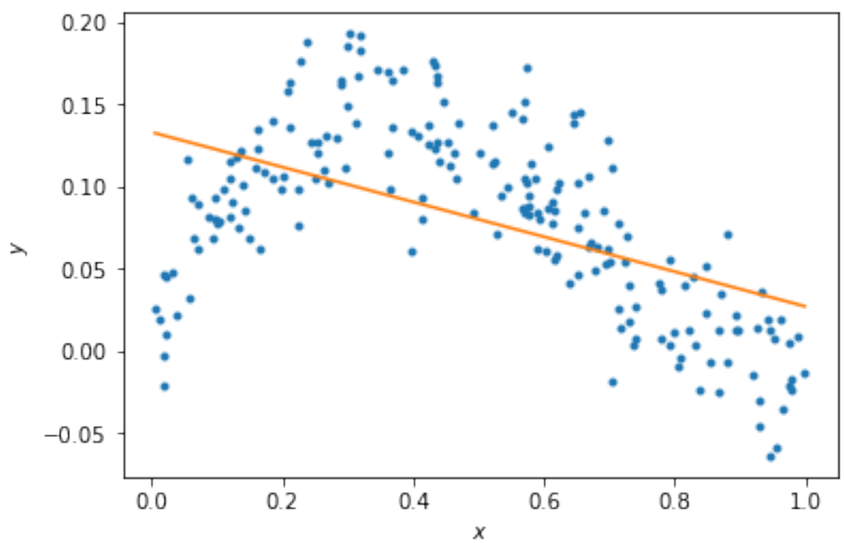
Fitting data to the model (5 points)

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)))
# print(xhat.T)
# ==================== #
# START YOUR CODE HERE #
# ==================== #
# GOAL: create a variable theta; theta is a numpy array whose elements ar

theta = np.zeros(2) # please modify this line
theta = np.linalg.lstsq(xhat.T, y)[0]
print(theta)


# theta = (a, b)
# ================== #
# END YOUR CODE HERE #
# ================== #
```

```
[-0.10599633  0.13315817]
```

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))

plt.show()
```



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 is underfitting the data.

(2) Need to change the model to a polynomial model, eg. a quadratic function.

Fitting data to the model (10 points)

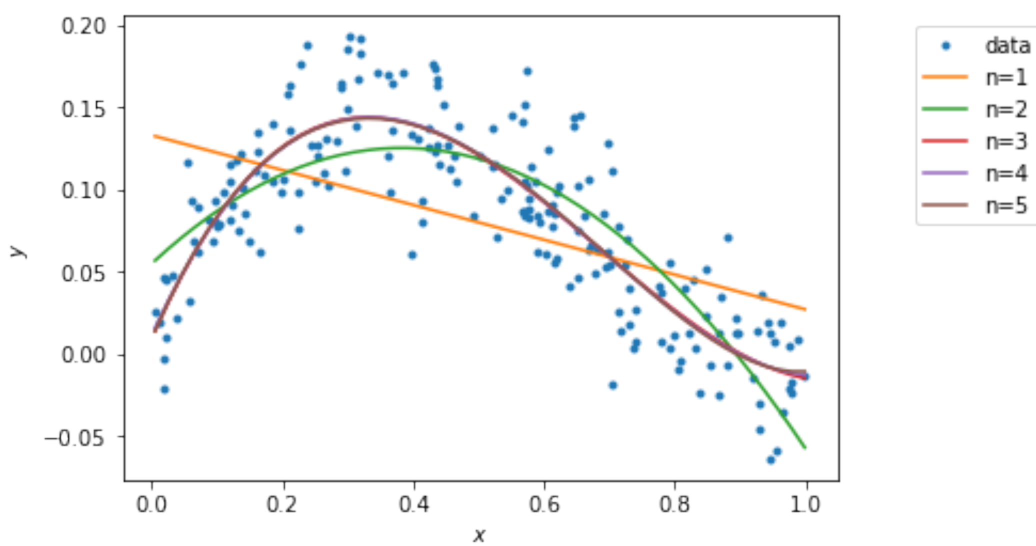
Here, we'll now do regression to polynomial models of orders 1 to 5. Note, the order 1 model is the linear model you prior fit.

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 polyn
#   i.e., thetas[0] is equivalent to theta above.
#   i.e., thetas[1] should be a length 3 np.array with the coefficients c
#   ... etc.


thetas.append(theta)
xhat = np.vstack((x, np.ones_like(x)))
# xhats.append(xhat)
for degree in range(2,N+1):
    xhat = np.vstack((x**degree, xhat))
    new_theta = np.linalg.lstsq(xhat.T, y)[0]
    thetas.append(new_theta)
#     xhats.append(xhat)
print(thetas)
pass


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

```
[array([-0.10599633,  0.13315817]), array([-0.48023061,  0.36743967,
0.05521084]), array([ 0.8843808 , -1.82077417,  0.91178032,  0.0097906
8]), array([ 0.14080037,  0.60466289, -1.64250929,  0.87250485,  0.011
75321]), array([ 0.52432591, -1.164568  ,  1.76052438, -2.07430275,  0
.93373916,
        0.009716  ])]
```

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)
# print(plot_xs)
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 [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 c

for i in np.arange(N):
    theta_ = thetas[i]
    error = sum((np.dot(theta_, xhat[-(i+2):, :])-y)**2)/x.size
    training_errors.append(error)
pass

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

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

```
Training errors are:
 [0.0023799610883627016, 0.0010924922209268528, 0.00081696038011053683
, 0.00081653537352969 78, 0.00081614791955252996]
```

QUESTIONS

(1) What polynomial has the best training error?

(2) Why is this expected?

ANSWERS

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

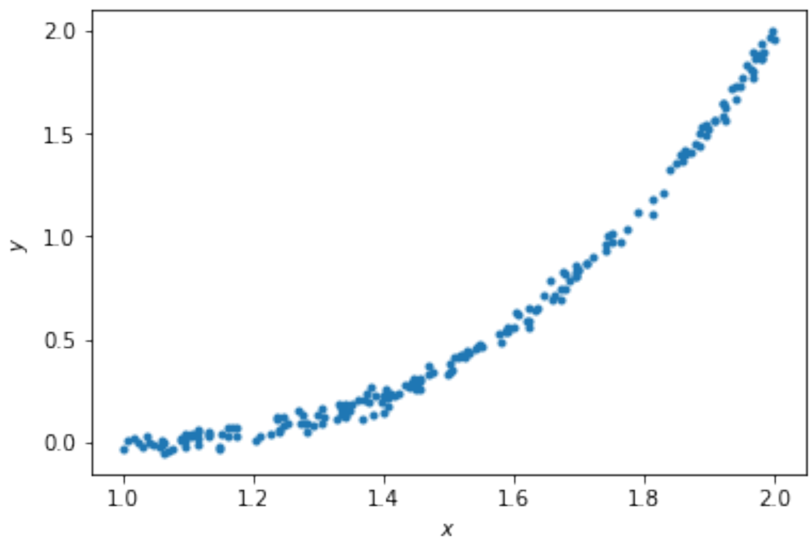
(2) It is because the model with high capacity can solve complex tasks, so that the regression line will be more approaching to true points. Therefore the training error will be decreasing as the order increases.

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

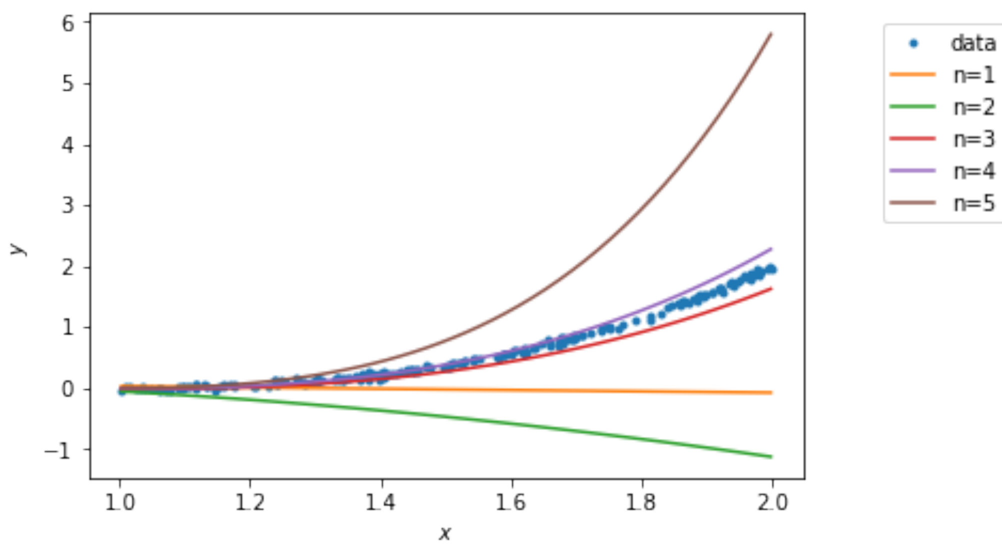
Out[8]:
```
Text(0,0.5,'$y$')
```



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
for i in np.arange(N):
    theta_ = thetas[i]
    error = sum((np.dot(theta_, xhat[-(i+2):, :])-y)**2)/x.size
    testing_errors.append(error)
pass

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

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

```
Testing errors are:
 [0.80861651845505844, 2.1319192445058217, 0.031256971083404202, 0.011
870765198475226, 2.1491021807250208]
```

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 the order 4 has the best tesing error.

(2) A polynomial model of degree 5 suffers from overfitting. It has too many parameters for the actual structure so that it focuses too much on the training data themselves but not generalization.

In [ ]: