ECE C147, Winter 2020
Homework 1
Submitted by Guy Ohayon

1 Linear algebra refresher

(a) i. A would be a matrix with orthogonal rows, such that the norm of each row is 1. This way, performing dot product between two rows would result in a scalar which is either 1 (if the rows are the same) or 0 (different rows are orthogonal). So, for example, such A could be:

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

Indeed, A holds the given property:

$$AA^T = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$$

Lets find the eigenvalues and eigenvectors of the matrix:

$$Av = \lambda v \Rightarrow |A - \lambda I| = 0 \Rightarrow \begin{vmatrix} \frac{1}{\sqrt{2}} - \lambda & \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} - \lambda \end{vmatrix} = 0$$

$$\Rightarrow (\frac{1}{\sqrt{2}} - \lambda)^2 + \frac{1}{2} = 0 \Rightarrow \frac{1}{2} - \sqrt{2}\lambda + \lambda^2 + \frac{1}{2} = 0$$

$$\Rightarrow 1 - \sqrt{2}\lambda + \lambda^2 = 0 \Rightarrow (\lambda - \frac{1+i}{\sqrt{2}})(\lambda - \frac{1-i}{\sqrt{2}}) = 0$$

$$\Rightarrow \boxed{\lambda_1 = \frac{1-i}{\sqrt{2}},\ \lambda_2 = \frac{1+i}{\sqrt{2}}}$$

$$Av_1 = \lambda_1 v_1 \Rightarrow \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} a_{v_1} \\ b_{v_1} \end{pmatrix} = \frac{1-i}{\sqrt{2}} \begin{pmatrix} a_{v_1} \\ b_{v_1} \end{pmatrix}$$

$$\Rightarrow a_{v_1} + b_{v_1} = (1-i)a_{v_1} \Rightarrow b_{v_1} = -ia_{v_1} \Rightarrow \boxed{v_1 = \begin{pmatrix} 1 \\ -i \end{pmatrix}}$$

$$Av_2 = \lambda_2 v_2 \Rightarrow \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} a_{v_2} \\ b_{v_2} \end{pmatrix} = \frac{1+i}{\sqrt{2}} \begin{pmatrix} a_{v_2} \\ b_{v_2} \end{pmatrix}$$

$$\Rightarrow a_{v_2} + b_{v_2} = (1+i)a_{v_2} \Rightarrow b_{v_2} = ia_{v_2} \Rightarrow \boxed{v_2 = \begin{pmatrix} 1 \\ i \end{pmatrix}}$$

We can notice that the both eigenvalues are composed by the components of the respective eigenvectors (normalized):

$$\lambda_1 = \frac{a_{v_1} + b_{v_1}}{\sqrt{2}},\ \lambda_2 = \frac{a_{v_2} + b_{v_2}}{\sqrt{2}}$$

The norm of both eigenvalues is 1, and the eigenvectors are orthogonal.

ii. From the given property, we get:

$$AA^T = I \Rightarrow A^T A = A^{-1} A A^T A = A^{-1} I A = A^{-1} A = I$$

.

Thus, $A^T A = I$. In addition, the eigenvalues hold:

$$Av = \lambda v \Rightarrow (Av)^T Av = (\lambda v)^T \lambda v \Rightarrow v^T \underbrace{A^T A}_{I} v = \lambda^2 v^T v$$

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

iii. Let v_1, v_2 be two eigenvectors of A, with distinct eigenvalues $\lambda_1 \neq \lambda_2$ respectively. We will use the following property:

$$Av = \lambda v \Rightarrow \frac{1}{\lambda} Av = v$$

Also, we will assume that A is real. Note that even though A is real, its eigenvectors can be complex (as we have seen in (i)). By using these properties, we get:

$$\overline{\lambda_2} v_1^T \overline{v_2} = v_1^T \overline{Av_2} = (A^T v_1)^T \overline{v_2} = (A^T \frac{1}{\lambda_1} Av_1)^T \overline{v_2} = \frac{1}{\lambda_1} v_1^T \overline{v_2}$$

Therefore:

$$(\overline{\lambda_2} - \frac{1}{\lambda_1}) v_1^T \overline{v_2} = 0$$

We know that $|\lambda_1| = |\lambda_2| = 1$. Thus, $\frac{1}{\lambda_1} = \overline{\lambda_1} \Rightarrow (\overline{\lambda_2} - \overline{\lambda_1}) v_1^T \overline{v_2} = 0$. Also, we know that $\lambda_1 \neq \lambda_2$ and so $\overline{\lambda_1} \neq \overline{\lambda_2}$. Thus: $v_1^T \overline{v_2} = 0 \Rightarrow v_1, v_2$ are orthogonal.

iv. The transformation $\boldsymbol{Ax}$ rotates or reflects the vector $\boldsymbol{x}$, while keeping the vector's norm the same.

(b) Let $U\Sigma V^T$ be the singular value decomposition of A (SVD can be performed on any matrix). The columns of U and V are the left and right singular vectors of A, respectively. Σ is a diagonal matrix with the singular values of A in the diagonal.

i. $AA^T = U\Sigma V^T(U\Sigma V^T)^T = U\Sigma V^T V\Sigma^T U^T$. We know that U and V are orthogonal matrices, so $V^T V = I$ and $UU^T = I$. Therefore, we get:

$$AA^T = U\Sigma\Sigma^T U^T$$

which is an eigendecomposition of AA^T. So, **the left singular vectors of A are the eigenvectors of AA^T.**
Similarly:

$$A^T A = (U\Sigma V^T)^T U\Sigma V^T = V\Sigma^T U^T U\Sigma V^T = V\Sigma^T\Sigma V^T$$

which is an eigendecomposition of $A^T A$. Therefore, **the right singular vectors of A are the eigenvectors of $A^T A$.**

ii. We have seen that the eigendecomposition of both $AA^T = U\Sigma\Sigma^T U^T$ and $A^T A = V\Sigma^T\Sigma V^T$. Therefore, it is immediate to conclude that the eigenvalues of both AA^T and $A^T A$ are the diagonal of $\Sigma\Sigma^T$, which is the square of the singular values of A: $\lambda_i = \sigma_i^2$ where σ_i is the ith singular value of A.

(c) i. False. For example, the matrix $I_{n\times n}$ is a linear operator in an n-dimensional vector space, but the matrix has n eigenvalues which are all the same.

ii. False. For example, consider the matrix $A = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$. The eigenvectors of the matrix are $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$. The sum of these eigenvectors results in the vector $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, but this is not an eigenvector of A.

iii. True. Let λ be an eigenvalue of A, and let v be the eigenvector corresponding to λ. Since A is positive semidefinite:

$$v^T Av \geq 0 \Rightarrow v^T \lambda v \geq 0 \Rightarrow ||v||^2\lambda \geq 0 \Rightarrow \lambda \geq 0$$

iv. True. For example, consider the matrix $A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. It's obvious that $rank(A) = 2$, but A has 2 eigenvectors: $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$, both with the same eigenvalue $\lambda = 1$. Therefore, we found a matrix with a rank that exceeds the number of distinct non-zero eigenvalues (we only have 1 distinct non-zero eigenvalue in that case).

v. True. Let v_1 and v_2 be two eigenvectors of a matrix A corresponding to the same eigenvalue λ. Therefore: $Av_1 = \lambda v_1,\ Av_2 = \lambda v_2$. We then get: $A(\alpha v_1 + \beta v_2) = \alpha A v_1 + \beta A v_2 = \alpha\lambda v_1 + \beta\lambda v_2 = \lambda(\alpha v_1 + \beta v_2)$. Therefore, $\alpha v_1 + \beta v_2$ is an eigenvector corresponding to the eigenvalue λ (given that the vector is not 0).

2 Probability refresher

(a) i. $P(H50|tails) = \frac{P(tails|H50)P(H50)}{P(tails)} = \frac{0.5 \cdot 0.5}{P(tails|H50)P(H50)+P(tails|H60)P(H60)} = \frac{0.5 \cdot 0.5}{0.5 \cdot 0.5 + 0.4 \cdot 0.5} = \boxed{\frac{5}{9}}$

ii. $P(H50|tails, heads, heads, heads) = \frac{P(tails,heads,heads,heads|H50)P(H50)}{P(tails,heads,heads,heads)} = \frac{0.5^5}{P(tails,heads,heads,heads|H50)P(H50)+P(tails,heads,heads,heads|H60)P(H60)} = \frac{0.5^5}{0.5^5 + 0.4 \cdot 0.6^3 \cdot 0.5} = \boxed{\frac{625}{1489}}$

iii. Let A be the event in which the coin lands on heads 9 out of 10 times.

$$P(A) = P(A|H50)P(H50)+P(A|H55)P(H55)+P(A|H60)P(H60)$$

$$= \binom{10}{9} 0.5^9 \cdot 0.5 \cdot \frac{1}{3} + \binom{10}{9} 0.55^9 \cdot 0.45 \cdot \frac{1}{3} + \binom{10}{9} 0.6^9 \cdot 0.4 \cdot \frac{1}{3} = 0.0236$$

$$P(H50|A) = \frac{P(A|H50)P(H50)}{P(A)} = \frac{\binom{10}{9} 0.5^9 \cdot 0.5 \cdot \frac{1}{3}}{0.0236} = \boxed{0.1379}$$

$$P(H55|A) = \frac{P(A|H55)P(H55)}{P(A)} = \frac{\binom{10}{9} 0.55^9 \cdot 0.45 \cdot \frac{1}{3}}{0.0236} = \boxed{0.2927}$$

$$P(H60|A) = \frac{P(A|H60)P(H60)}{P(A)} = \frac{\binom{10}{9} 0.6^9 \cdot 0.4 \cdot \frac{1}{3}}{0.0236} = \boxed{0.5693}$$

(b) It is given that:

$$P(Pos|Preg) = 0.99$$
$$P(Pos|\neg Preg) = 0.1$$
$$P(\neg Preg) = 0.99$$

Lets calculate the desired probability:

$$P(Preg|Pos) = \frac{P(Pos|Preg)P(Preg)}{P(Pos)} =$$
$$\frac{0.99 \cdot 0.01}{P(Pos|Preg)P(Preg) + P(Pos|\neg Preg)P(\neg Preg)} =$$
$$\frac{0.0099}{0.99 \cdot 0.01 + 0.1 \cdot 0.99} = \boxed{\frac{1}{11}}$$

A female's pregnancy is a strong indicator for a pregnancy test to come out positive. Yet, a positive test is not a strong indicator for a woman to be pregnant because 99% of the women are not pregnant, and there's still a notable chance for a test to come out positive even if the woman is not pregnant.

(c)

$$\mathbb{E}[Ax+b] = \mathbb{E}[\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}] + \mathbb{E}[b] = \mathbb{E}[\begin{pmatrix} x_1 a_{11} + \ldots + x_n a_{1n} \\ \vdots \\ x_1 a_{n1} + \ldots + x_n a_{nn} \end{pmatrix}] + b$$

$$= \boxed{A\,\mathbb{E}[x] + b}$$

(d)

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

3 Multivariate derivatives

(a) Ay is just a vector in $\mathbb{R}^n$. Lets call it b. $\nabla_x x^T b = \nabla_x = \nabla_x(x_1 b_1 + \ldots + x_n b_n) = b = \boxed{Ay}$

(b) $x^T A$ is a vector in $\mathbb{R}^{1\times m}$. Lets call it b^T. $\nabla_y b^T y = \nabla_y(b_1 y_1 + \ldots + b_n y_n) = b = \boxed{A^T x}$

(c) $\nabla_A x^T A y = \nabla_A(x^T \begin{pmatrix} \sum_{i=1}^m A_{1i} y_i \\ \vdots \\ \sum_{i=1}^m A_{ni} y_i \end{pmatrix}) = \nabla_A \sum_{j=1}^n \sum_{i=1}^n x_j y_i A_{ji} = \begin{pmatrix} x_1 y_1 & x_1 y_2 & \ldots & x_1 y_n \\ \vdots & & & \\ x_n y_1 & x_n y_2 & \ldots & x_n y_n \end{pmatrix} = \boxed{xy^T}$

(d) We have seen in clas: $\nabla_x x^T A x = (A + A^T)x$. Therefore: $\nabla_x f = \nabla_x(x^T A x + b^T x) = \boxed{(A + A^T)x + b}$

(e) $\nabla_A f = \nabla_A tr(AB) = \nabla_A(\sum_{i=1}^n \sum_{j=1}^m A_{ij} B_{ji}) = \begin{pmatrix} B_{11} & B_{21} & \ldots & B_{n1} \\ \vdots & & & \\ B_{1n} & B_{2n} & \ldots & B_{mn} \end{pmatrix} = \boxed{B^T}$

4 Deriving least-squares with matrix derivatives

Let $\mathcal{L}(W) = \frac{1}{2}\sum_{i=1}^n ||y^{(i)} - Wx^{(i)}||^2 = \frac{1}{2}\sum_{i=1}^n (y^{(i)} - Wx^{(i)})^T (y^{(i)} - Wx^{(i)})$ and let $A^T = \begin{pmatrix} —x^{(1)}— \\ —x^{(2)}— \\ \vdots \\ —x^{(n)}— \end{pmatrix}, B^T = \begin{pmatrix} —y^{(1)}— \\ —y^{(2)}— \\ \vdots \\ —y^{(n)}— \end{pmatrix}$. Therefore:

$$\mathcal{L}(W) = \frac{1}{2} tr((B - WA)^T (B - WA)) = \frac{1}{2} tr(B^T B - B^T WA - A^T W^T B + A^T W^T WA)$$

By using some trace properties, we get:

$$\mathcal{L}(W) = \frac{1}{2}(tr(B^T B) - tr(WAB^T) - tr(W^T BA^T) + tr(WAA^T W^T))$$

$$= \frac{1}{2}(tr(B^T B) - 2tr(WAB^T) + tr(WAA^T W^T))$$

To find the optimal W, we need to find W that satisfies $\frac{\mathcal{L}(W)}{\partial W} = 0$ because we the least-squares function is convex. Lets derive our loss function:

$$\frac{\mathcal{L}(W)}{\partial W} = \frac{1}{2}(-2BA^T + 2WAA^T) = WAA^T - BA^T = 0$$

$$\Rightarrow \boxed{W = BA^T(AA^T)^{-1}}$$

5 Hello World in Jupyter

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

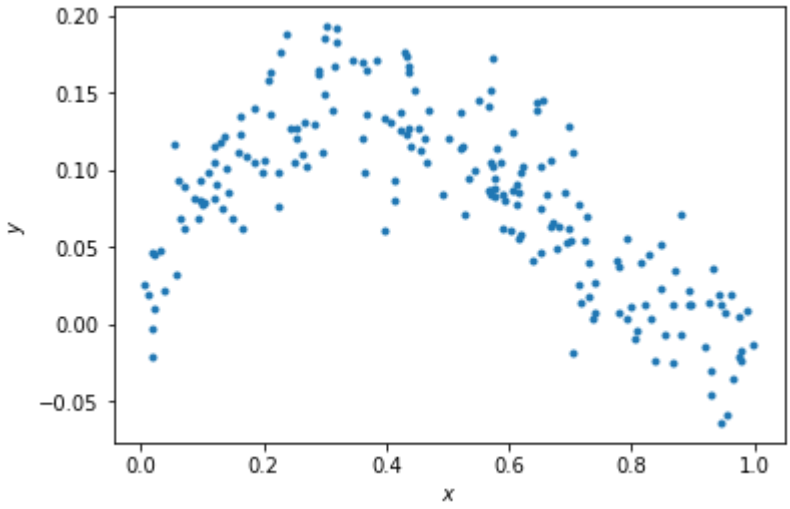
In [88]:
```
import numpy as np
import matplotlib.pyplot as plt

#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 [89]:
```
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[89]: 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) $x \sim Uniform[0, 1]$

(2) $\epsilon \sim \mathcal{N}(0, 0.03^2)$

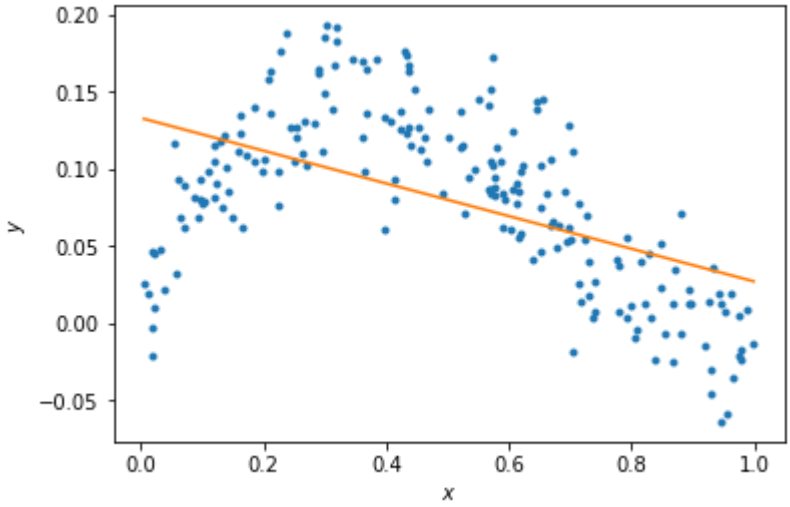
Fitting data to the model (5 points)

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

In [90]:
```
# 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(np.transpose(xhat))).dot(xhat).dot(y)

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

In [91]:
```
# 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[91]: [<matplotlib.lines.Line2D at 0x7fe4a27c7c50>]



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 underfits the data.

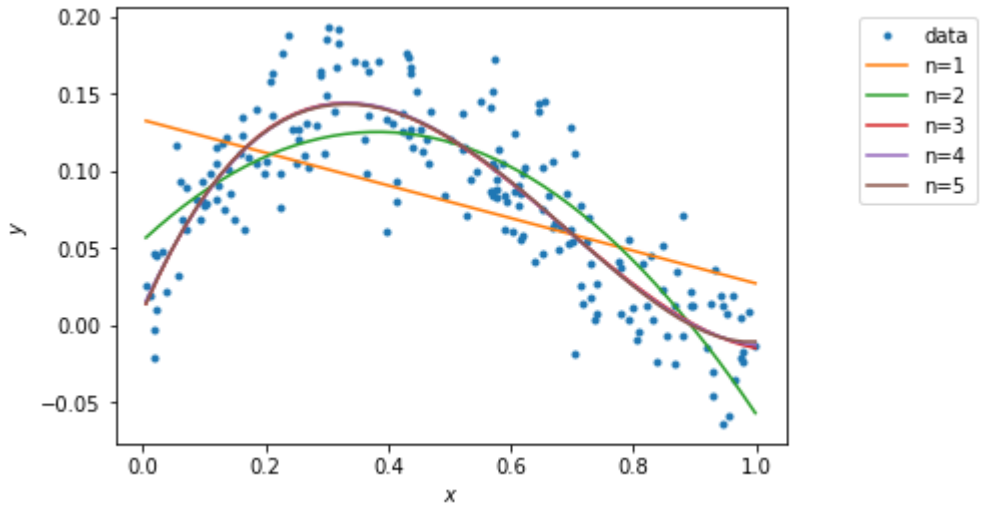
(2) We can make the model a polynomial of degree 3. This way the model would be more flexible and could have a local maximum around $x = 0.3$ and a local minimum at $x = 0.9$.

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 [92]:
```
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 po
lynomial 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 coefficient
s of the x^2, x, and 1 respectively.
#   ... etc.
features = [np.ones_like(x)]
for i in range(1, N + 1):
    features = [x**i] + features
    xhats.append(np.vstack(features))

for i in range(N):
    thetas.append(np.linalg.inv(xhats[i].dot(np.transpose(xhats[i]))).
dot(xhats[i]).dot(y))

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

In [93]:
```
# 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:

$L(\theta) = \frac{1}{2} \sum_j (\hat{y}_j - y_j)^2$

In [94]:
```
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 fi
t of order i+1.

for i in range(N):
    training_errors.append((1/2) * (y.T.dot(y)- 2 * y.T.dot(xhats[i].T
).dot(thetas[i]) + \
    thetas[i].T.dot(xhats[i]).dot(xhats[i].T).dot(thetas[i])))

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

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

```
Training errors are:
 [0.23799610883627054, 0.10924922209268595, 0.08169603801105374, 0.081
653537352969763, 0.081614791955252897]
```

QUESTIONS

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

(2) Why is this expected?

ANSWERS

(1) The polynomial model of degree 5.

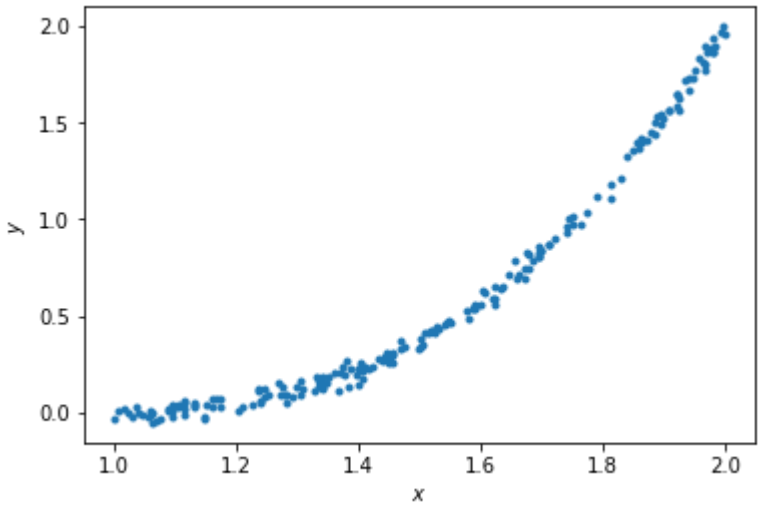
(2) A polynomial of higher degree is more flexible to fit data, because it could potentially have more local optimas (more degrees of freedom). Therefore, we expect a polynomial of degree 5 to be at least as good or even better than any other polynomial of degree 1, 2, 3, or 4.

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.

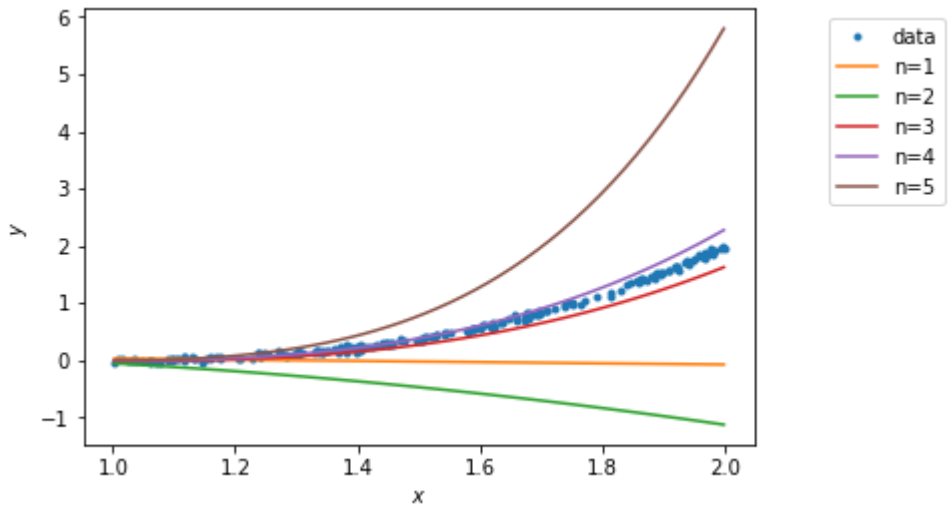
In [95]:
```
x = np.random.uniform(low=1, high=2, size=(num_train,))
y = x - 2*x**2 + x**3 + np.random.normal(loc=0, scale=0.03, size=(num_
train,))
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

Out[95]: Text(0,0.5,'y')



In [96]:
```
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 [97]:
```
# 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 [98]:
```
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 range(N):
    testing_errors.append((1/2) * (y.T.dot(y)- 2 * y.T.dot(xhats[i].T)
.dot(thetas[i]) + \
    thetas[i].T.dot(xhats[i]).dot(xhats[i].T).dot(thetas[i])))

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

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

```
Testing errors are:
 [80.861651845505946, 213.19192445058206, 3.1256971083047063, 1.187076
5193960438, 214.91021837340656]
```

QUESTIONS

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

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

ANSWERS

(1) The polynomial model of degree 4.

(2) This happens because the order-5 polynomial overfits the data (the model is too complex to generalize given our data distribution).