ECE C247 HW 1 Solution

Ashish Kumar Singh (UID:105479019)

January 16, 2022

Problem 1. Linear Algebra Refresher

Sub-Problem 1(a)(i). Example of A for $AA^T = I$?

Solution 1(a)(i). Let

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$AA^T = \begin{bmatrix} a^2 + b^2 & ac + bd \\ ac + bd & c^2 + d^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

To satisfy diagonal terms, we can take $a = \sin\alpha$, $b = \cos\alpha$, $c = \sin\beta$, $d = \cos\beta$
Non-diagonal term becomes,

$$ac + bd = \sin\alpha\sin\beta + \cos\alpha\cos\beta$$

$$0 = \cos(\alpha - \beta)$$

$$\alpha = \beta \pm \frac{(2n+1)\pi}{2}$$

As an example we can take, $\alpha = 30°$ and $\beta = 120°$, we get,

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

For eigenvalues and eigenvectors, we set $(A - \lambda I)x = 0$,

$$(A - \lambda I) = \begin{bmatrix} \frac{1}{2} - \lambda & \frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{-1}{2} - \lambda \end{bmatrix}$$

Setting determinant to zero,

$$\tfrac{(2\lambda-1)(2\lambda+1)}{4} - \tfrac{3}{4} = 0$$

$$4\lambda^2 = 4$$

$$\lambda = \pm 1$$

For $\lambda = 1$, we can find its corresponding eigenvalue by $(A - I)x = 0$,

$$(A - I)x = \begin{bmatrix} \frac{1}{2} - 1 & \frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{-1}{2} - 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

Solving with additional constraint $x_1^2 + x_2^2 = 1$, we get $x_1 = \frac{\sqrt{3}}{2}$ and $x_2 = \frac{1}{2}$, thus eigenvector corresponding to $\lambda = 1$ is:

$$x = \begin{bmatrix} 0.866 \\ 0.5 \end{bmatrix}$$

Similarly, the eigenvector corresponding to $\lambda = -1$ is:

$$x = \begin{bmatrix} 0.5 \\ -0.866 \end{bmatrix}$$

We notice that eigenvalues are unit norm and eigenvectors are orthogonal.

Sub-Problem 1(a)(ii). show eigenvalues are unit norm

Solution 1(a)(ii). If λ is the eigenvalue of A then $Ax = \lambda x$,

$$(Ax)^T(Ax) = (\lambda x)^T(\lambda x)$$

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

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

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

Sub-Problem 1(a)(iii). show eigenvectors are orthogonal

Solution 1(a)(iii). If λ_1 and λ_2 are two distinct eigenvalues of A then $Ax_1 = \lambda_1 x_1$, $Ax_2 = \lambda_2 x_2$, and $\lambda_1 \neq \lambda_2$

$$(Ax_1)^T(Ax_2) = (\lambda_1 x_1)^T(\lambda_2 x_2)$$

$$x_1^T A^T A x_2 = \lambda_1 \lambda_2 x_1^T x_2$$

$$x_1^T x_2(\lambda_1 \lambda_2 - 1) = 0$$

We know $|\lambda_1| = 1$, $|\lambda_2| = 1$ and $\lambda_1 \neq \lambda_2$,so $\lambda_1 \lambda_2 \neq 1$ (assuming real eigenvalues), which leaves us with the only option,

$$x_1^T x_2 = 0$$

which means eigenvectors corresponding to distinct eigenvalues of A are orthogonal.

Sub-Problem 1(a)(iv). transformation Ax

Solution 1(a)(iv). Since the eigenvalues are unit norm, the vector x magnitude remains constant, only the direction changes. In other words, vector x is only rotated, its length remain constant.

Sub-Problem 1(b)(i). Relationship between singular vectors and eigenvectors

Solution 1(b)(i). By Singular vector decomposition, we can write, $A = U\Sigma V^T$, where U is the left singular vectors and V is the right singular vectors of A, $UU^T = I$, $V^TV = I$, Σ is a diagonal matrix,

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

If we multiply AA^T by U, we get,

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

We can see that the U is the eigenvectors of AA^T, and $\Sigma\Sigma^T$ have the eigenvalues of AA^T. Similarly, for A^TA,

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

If we multiply A^TA by V, we get,

$$A^TAV = V\Sigma^T\Sigma V^T V = V\Sigma^T\Sigma$$

We can see that the V is the eigenvectors of A^TA, and $\Sigma^T\Sigma$ have the eigenvalues of A^TA.

Hence, the left singular vectors of A are the eigenvectors of AA^T and the right singular vectors of A are the eigenvectors of A^TA.

Sub-Problem 1(b)(ii). Relationship between singular values and eigenvalues

Solution 1(b)(ii). Σ is a diagonal matrix, therefore $\Sigma^T\Sigma = \Sigma\Sigma^T =$ square of the diagonal terms of Σ.
Hence, from previous derivation, eigenvalues of AA^T and A^TA is the square of the singular value of A. In other words, singular values of A is square root of eigenvalues of AA^T and A^TA.

Sub-Problem 1(c). True or False

Solution 1(c)(i). False
There can be at most n distinct eigenvalues.

Solution 1(c)(ii). False
If eigenvalues are different then it will not be an eigenvector.

Solution 1(c)(iii). True
for eigenvectors, $x^TAx = x^T\lambda x \geq 0$, $\lambda \geq 0$

Solution 1(c)(iv). True
There can be at most n distinct eigenvalues.

Solution 1(c)(v). True
$A(x+y) = Ax + Ay = \lambda x + \lambda y = \lambda(x+y)$

Problem 2. Probability Refresher

Sub-Problem 2(a)(i). Find posterior $P(H50|T)$?

Solution 2(a)(i).

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

$$P(H50|T) = \frac{0.5 * 0.5}{0.5 * 0.5 + 0.4 * 0.5}$$

$$P(H50|T) = \frac{5}{9}$$

Sub-Problem 2(a)(ii). Find posterior $P(H50|THHH)$?

Solution 2(a)(ii).

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

$$P(H50|THHH) = \frac{0.5 * 0.5 * 0.5 * 0.5 * 0.5}{0.5 * 0.5 * 0.5 * 0.5 * 0.5 + 0.4 * 0.6 * 0.6 * 0.6 * 0.5}$$

$$P(H50|THHH) = \frac{0.5 * 0.5 * 0.5 * 0.5 * 0.5}{0.5 * 0.5 * 0.5 * 0.5 * 0.5 + 0.4 * 0.6 * 0.6 * 0.6 * 0.5}$$

$$P(H50|THHH) = \frac{0.03125}{0.03125 + 0.0432} \approx 0.41974$$

Sub-Problem 2(a)(iii). Find posterior for H50,H55,H60 for given data 9 head out of 10 tosses?

Solution 2(a)(iii). Let D denote the data with 9 head out of 10 tosses,

$$P(H50|D) = \frac{P(D|H50)P(H50)}{P(D|H50)P(H50) + P(D|H55)P(H55) + P(D|H60)P(H60)}$$

$$P(H50|D) = \frac{(1/3) * (0.5)^{10}}{(1/3) * (0.5)^{10} + (1/3) * (0.55)^9 * (0.45) + (1/3) * (0.6)^9 * (0.4)}$$

$$P(H50|D) \approx 0.13793$$

$$P(H55|D) = \frac{P(D|H55)P(H55)}{P(D|H50)P(H50) + P(D|H55)P(H55) + P(D|H60)P(H60)}$$

$$P(H55|D) = \frac{(1/3) * (0.55)^9 * (0.45)}{(1/3) * (0.5)^{10} + (1/3) * (0.55)^9 * (0.45) + (1/3) * (0.6)^9 * (0.4)}$$

$$P(H55|D) \approx 0.29271$$

$$P(H60|D) = \frac{P(D|H60)P(H60)}{P(D|H50)P(H50) + P(D|H55)P(H55) + P(D|H60)P(H60)}$$

$$P(H60|D) = \frac{(1/3) * (0.6)^9 * (0.4)}{(1/3) * (0.5)^{10} + (1/3) * (0.55)^9 * (0.45) + (1/3) * (0.6)^9 * (0.4)}$$

$$P(H60|D) \approx 0.56936$$

Sub-Problem 2(b). Find $P(pregnant|positive)$?

Solution 2(b). We can infer following probabilities:
$P(positive|pregnant) = 0.99$, $P(positive|notpregnant) = 0.1$,
$P(pregnant) = 0.01$, $P(notpregnant) = 0.1$

$$P(pregnant|positive) = \frac{P(positive, pregnant)}{P(positive, pregnant) + P(positive, notpregnant)}$$

$$P(pregnant|positive) = \frac{0.99 * 0.01}{0.99 * 0.01 + 0.99 * 0.1}$$

$$P(pregnant|positive) = \frac{1}{11} \approx 0.091$$

The answer shows that the test is very bad. It makes sense because the test is giving 10% false positive on a 99% of non-pregnant female population. So, e.g. for a female population of 1000, 990 are not pregnant and only 10 are pregnant but test gives 99 female as false positive. Thus, very low $P(pregnant|positive)$ makes sense, the test is very bad.

Sub-Problem 2(c). Find $\mathbb{E}(Ax + b)$?

Solution 2(c). Lets find the i^{th} term of $\mathbb{E}(Ax)$,

$$\mathbb{E}(Ax)_i = \mathbb{E}(\sum A_{i,j} x_j)$$

$$\mathbb{E}(Ax)_i = \sum A_{i,j}\, \mathbb{E}(x)_j$$

$$\mathbb{E}(Ax)_i = (A\, \mathbb{E}(x))_i$$

The above is true for every i, so we get,

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

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

Sub-Problem 2(d). Find $cov(Ax + b)$?

Solution 2(d).

$$cov(Ax+b) = \mathbb{E}\left(((Ax+b) - \mathbb{E}(Ax+b))((Ax+b) - \mathbb{E}(Ax+b))^T\right)$$

$$cov(Ax+b) = \mathbb{E}\left((A(x - \mathbb{E}(x)))(A(x - \mathbb{E}(x)))^T\right)$$

$$cov(Ax+b) = \mathbb{E}\left(A(x - \mathbb{E}(x))(x - \mathbb{E}(x))^T A^T\right)$$

$$cov(Ax+b) = A\,\mathbb{E}\left((x - \mathbb{E}(x))(x - \mathbb{E}(x))^T\right)A^T$$

$$cov(Ax+b) = Acov(x)A^T$$

Problem 3. Multivariate derivatives

Solution 3(a). First finding derivative wrt to each x_i, then combining

$$\nabla_{x_i} x^T Ay = \nabla_{x_i}\left(\sum_{i=1}^{n}\sum_{j=1}^{m} x_i A_{ij} y_j\right)$$

$$\nabla_{x_i} x^T Ay = \sum_{j=1}^{m} A_{ij} y_j$$

$$\nabla_x x^T Ay = \begin{bmatrix} \sum_{j=1}^{m} A_{1j} y_j \\ \vdots \\ \sum_{j=1}^{m} A_{nj} y_j \end{bmatrix}$$

$$\nabla_x x^T Ay = Ay$$

Solution 3(b). First finding derivative wrt to each y_j, then combining

$$\nabla_{y_j} x^T Ay = \nabla_{y_j}\left(\sum_{i=1}^{n}\sum_{j=1}^{m} x_i A_{ij} y_j\right)$$

$$\nabla_{y_j} x^T Ay = \sum_{i=1}^{n} x_i A_{ij}$$

$$\nabla_{y_j} x^T Ay = \sum_{i=1}^{n} A^T_{ji} x_i$$

$$\nabla_y x^T Ay = \begin{bmatrix} \sum_{i=1}^{n} A^T_{1i} x_i \\ \vdots \\ \sum_{i=1}^{n} A^T_{mi} x_i \end{bmatrix}$$

$$\nabla_y x^T Ay = A^T x$$

Solution 3(c). First finding derivative wrt to each A_{ij}, then combining

$$\nabla_{A_{ij}} x^T A y = \nabla_{A_{ij}} \left(\sum_{i=1}^{n} \sum_{j=1}^{m} x_i A_{ij} y_j \right)$$

$$\nabla_{A_{ij}} x^T A y = x_i y_j$$

$$\nabla_A x^T A y = \begin{bmatrix} x_1 y_1 & \dots & x_1 y_m \\ \vdots & \vdots & \vdots \\ x_n y_1 & \dots & x_n y_m \end{bmatrix}$$

$$\nabla_A x^T A y = x y^T$$

Solution 3(d). First finding derivative wrt to each x_k, then combining

$$\nabla_{x_k} \left(x^T A x + b^T x \right) = \nabla_{x_k} \left(\sum_{i=1}^{n} \sum_{j=1}^{m} A_{ij} x_i x_j + \sum_{i=1}^{n} b_i x_i \right)$$

$$\nabla_{x_k} \left(x^T A x + b^T x \right) = \sum_{j=1}^{m} A_{kj} x_j + \sum_{i=1}^{n} A_{ik} x_i + b_i$$

$$\nabla_{x_k} \left(x^T A x + b^T x \right) = \sum_{j=1}^{m} A_{kj} x_j + \sum_{i=1}^{n} A^T_{ki} x_i + b_i$$

$$\nabla_x \left(x^T A x + b^T x \right) = \begin{bmatrix} \sum_{j=1}^{m} A_{1j} x_j + \sum_{i=1}^{n} A^T_{1i} x_i + b_1 \\ \vdots \\ \sum_{j=1}^{m} A_{nj} x_j + \sum_{i=1}^{n} A^T_{ni} x_i + b_n \end{bmatrix}$$

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

Solution 3(e). Lets assume, dimension of A and B are (nxm) and (mxn) respectively. First finding derivative wrt to each A_{ij}, then combining

$$\nabla_{A_{ij}} tr(AB) = \nabla_{A_{ij}} \left(\sum_{i=1}^{n} (AB)_{ii} \right)$$

$$\nabla_{A_{ij}} tr(AB) = \nabla_{A_{ij}} \left(\sum_{i=1}^{n} \sum_{j=1}^{m} A_{ij} B_{ji} \right)$$

$$\nabla_{A_{ij}} tr(AB) = B_{ji} = B^T_{ij}$$

$$\nabla_{A_{ij}} tr(AB) = \begin{bmatrix} B^T_{11} & \dots & B^T_{1m} \\ \vdots & \vdots & \vdots \\ B^T_{n1} & \dots & B^T_{nm} \end{bmatrix}$$

$$\nabla_A tr(AB) = B^T$$

Problem 4. Least squares derivation

Solution 4. We can stack all the $y^{(i)}$ as column to form Y, similarly for $x^{(i)}$ to form X, then loss can be written as:

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

$$L(W) = \frac{1}{2}tr\Big((Y - WX)^T(Y - WX)\Big)$$

$$L(W) = \frac{1}{2}tr\Big(Y^TY - Y^TWX - X^TW^TY + X^TW^TWX\Big)$$

Using trace properties: $tr(A) = tr(A^T)$, $tr(A + B) = tr(A) + tr(B)$ and $tr(ABC) = tr(BCA) = tr(CAB)$, we get,

$$L(W) = \frac{1}{2}\Big(tr(Y^TY) - tr(Y^TWX) - tr(X^TW^TY) + tr(X^TW^TWX)\Big)$$

$$L(W) = \frac{1}{2}\Big(tr(Y^TY) - tr(WXY^T) - tr(WXY^T) + tr(WXX^TW^T)\Big)$$

Differentiating wrt to W, and setting it to zero to get W_{opt},

$$\nabla_W L(W) = \frac{1}{2}\Big(0 - YX^T - YX^T + WXX^T + WXX^T\Big) = 0$$

$$-YX^T + W_{opt}XX^T = 0$$
$$W_{opt}XX^T = YX^T$$
$$W_{opt} = YX^T(XX^T)^{-1}$$

Problem 5. Hello World in Jupyter

Solution 5. code is attached below

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 2022, Prof. J.C. Kao, TAs Y. Li, P. Lu, T. Monsoor, T. wang

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

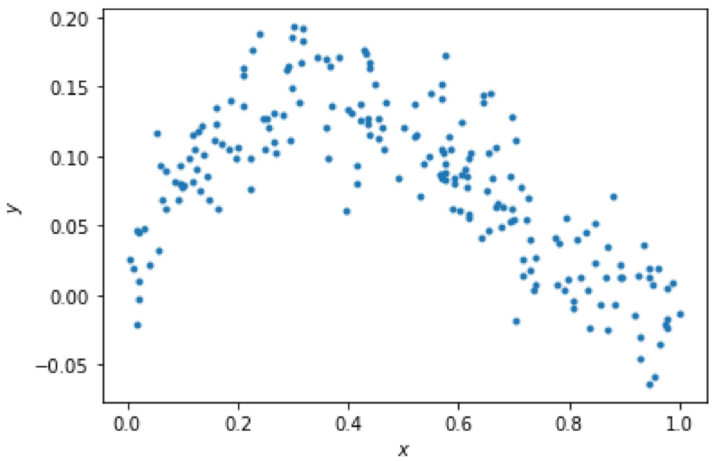
In [2]:

```
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_train,))
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) x is generated from a **uniform** distribution in range 0 to 1.

(2) ϵ noise is generated from a **gaussian** distribution with mean 0 and standard deviation 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 [4]:

```
# 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))
#print(theta)

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

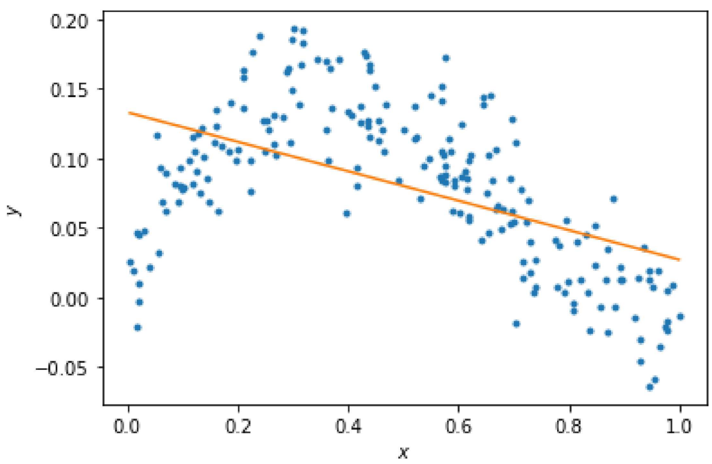
In [5]:

```
# 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[5]:

```
[<matplotlib.lines.Line2D at 0x1c0a44720a0>]
```



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 because there is non-linearity in data which the linear model cannot fit.

(2) Instead of linear model we can fit higher degree 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 [6]:

```
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 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 re
#   ... etc.

for i in range(N):
    if i==0:
        xhats.append(xhat)
    else:
        xhats.append(np.vstack((x**(i+1),xhats[i-1])))
    thetas.append(np.linalg.inv(xhats[i].dot(xhats[i].T)).dot(xhats[i].dot(y)))
#print(thetas)
# ================== #
# END YOUR CODE HERE #
# ================== #
```

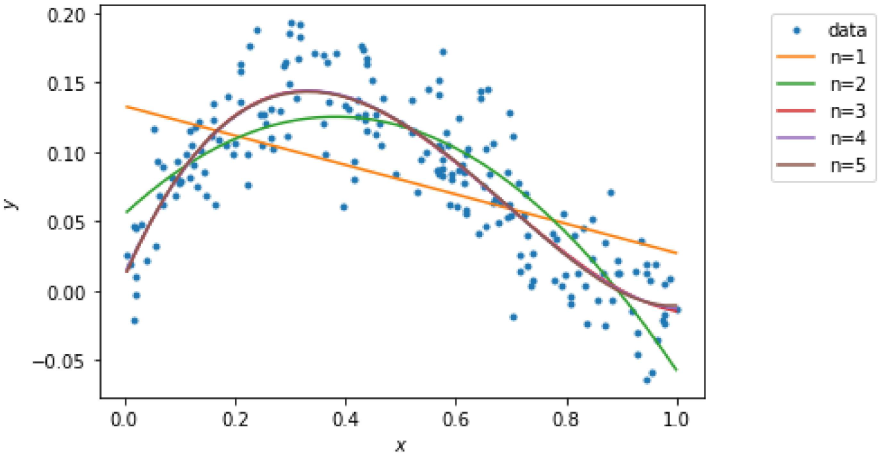
In [7]:

```
# 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 [8]:

```
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 range(N):
    training_errors.append(np.sum((y - thetas[i].dot(xhats[i]))**2/2))
# ================== #
# END YOUR CODE HERE #
# ================== #

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

```
Training errors are:
 [0.2379961088362701, 0.10924922209268531, 0.08169603801105374, 0.08165353735296982, 0.08161479
195525298]
```

QUESTIONS

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

(2) Why is this expected?

ANSWERS

(1) Polynomial model of **order 5** has the best training error.

(2) This is expected because polynomial of higher degree will try to **overfit** the data.

Generating new samples and validation error (5 points)

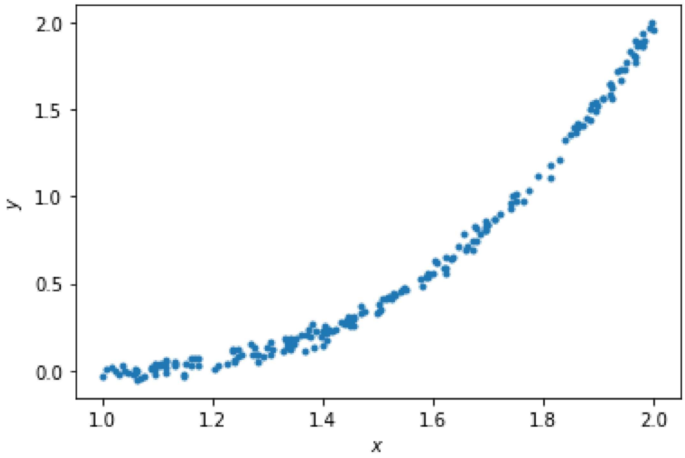
Here, we'll now generate new samples and calculate the validation error of polynomial models of orders 1 to 5.

In [9]:

```
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[9]:

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



In [10]:

```
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)
```

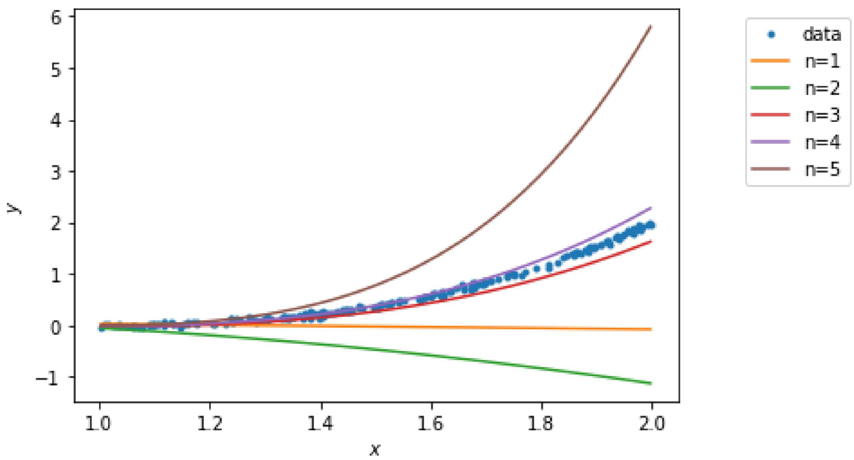
In [11]:

```
# 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 [ ]:
```
validation_errors = []

# ==================== #
# START YOUR CODE HERE #
# ==================== #

# GOAL: create a variable validation_errors, a list of 5 elements,
# where validation_errors[i] are the validation loss for the polynomial fit of order i+1.
for i in range(N):
    validation_errors.append(np.sum((y - thetas[i].dot(xhats[i]))**2/2))

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

print ('Validation errors are: \n', validation_errors)
```

QUESTIONS

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

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

ANSWERS

(1) Polynomial model of **order 4** has the best validation error.

(2) order-5 polynomial model didn't generalize well because it overfitted the training data in range 0 to 1, so when it was given data outside this range, it gave high errors.