ECE 247 HW #1 - Ashwin Ranade

① a) i) Let $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, then the characteristic polynomial is

$\det(A - \lambda I) = \det\begin{pmatrix} -\lambda & 1 \\ 1 & -\lambda \end{pmatrix} = \lambda^2 - 1$. Note $AA^T = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

Setting the characteristic polynomial to 0, we get $\lambda^2 - 1 = 0 \rightarrow \lambda = \pm 1$.

$\boxed{\text{Hence, the 2 eigenvalues are } \pm 1}$

Let $\vec{v}$ and $\vec{w}$ be the eigenvectors corresponding to 1, -1 respectively. Then,

$$(A - (1) I_2)\vec{v} = 0$$

$$\left(\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right)\vec{v} = 0$$

$$\begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}\vec{v} = 0 \rightarrow \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \rightarrow -v_1 + v_2 = 0 \rightarrow v_2 = v_1$$

Then, $\vec{v} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} v_1 \\ v_1 \end{pmatrix} = v_1\begin{pmatrix} 1 \\ 1 \end{pmatrix} = k\begin{pmatrix} 1 \\ 1 \end{pmatrix}, k \in \mathbb{R}$.

Similary, for $\lambda = -1$,

$$(A - (-1) I_2)\vec{w} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$w_1 + w_2 = 0 \rightarrow w_1 = -w_2$

$\vec{w} = \begin{pmatrix} w_1 \\ -w_1 \end{pmatrix} = w_1\begin{pmatrix} 1 \\ -1 \end{pmatrix}$.

$\boxed{\text{Hence, } \vec{v} = k_1\begin{pmatrix} 1 \\ 1 \end{pmatrix}, k_1 \in \mathbb{R}, \text{ and } \vec{w} = k_2\begin{pmatrix} 1 \\ -1 \end{pmatrix}, k_2 \in \mathbb{R}}$

$\boxed{\text{We notice that each eigenvalue has norm 1, and the eigenvectors are orthogonal}}$

ii) Since $AB = I \rightarrow BA = I$, $AA^T = I \rightarrow A^T A = I$

Let $\vec{v}$ be an eigenvector with eigenvalue λ such that $\lambda\vec{v} = A\vec{v}$. Then,

$$\vec{v}^T A^T A\vec{v} = (A\vec{v})^T (A\vec{v}) = (\lambda\vec{v})^T(\lambda\vec{v}) = (\lambda\vec{v})\cdot(\lambda\vec{v})$$
$$= \lambda^2\, \vec{v}\cdot\vec{v} = \lambda^2 |\vec{v}|^2.$$

Also, $\vec{v}^T A^T A\vec{v} = \vec{v}^T I \vec{v} = \vec{v}\cdot\vec{v} = |\vec{v}|^2$ since $A^T A = I$.

Hence, $\lambda^2|\vec{v}|^2 = |\vec{v}|^2$, or $\lambda^2 = 1 \rightarrow \lambda = \pm 1$.

$\boxed{\text{Hence all eigenvalues must be } \pm 1 \text{, so all eigenvalues must have norm 1.}}$

iii) From Piazza, we only need to focus on real eigenvalues, so we can ignore the adjoint case.

From ii), we know we can only have eigenvalues ± 1.

Let $\vec{x}$ be an eigenvector with eigenvalue λ_1, and $\vec{y}$ be an eigenvector with λ_2. Then $A\vec{x} = \lambda_1 \vec{x}$, $A\vec{y} = \lambda_2 \vec{y}$, with $\lambda_1 \neq \lambda_2$.

Then, $\vec{x} \cdot \vec{y} = \vec{x}^T \vec{y} = \vec{x}^T I \vec{y} = \vec{x}^T A^T A \vec{y} = (A\vec{x})^T (A\vec{y})$
$= (\lambda_1 \vec{x})^T (\lambda_2 \vec{y}) = \lambda_1 \lambda_2 (\vec{x} \cdot \vec{y})$.

Hence either $\lambda_1 \lambda_2 = 1$ or $\vec{x} \cdot \vec{y} = 0$. Since $\lambda_1 \cdot \lambda_2 = 1 \cdot -1 = -1 \neq 1$, $\vec{x} \cdot \vec{y} = 0$, and hence $\vec{x}$ and $\vec{y}$ are orthogonal, as needed.

iv) Since the eigenvalues are ± 1, the transformation does not scale the input in any way. Since length and angles must be preserved,
$A\vec{x}$ will simply be $\vec{x}$ after a rotation (around the origin), or a flip / reflection across some subspace.

If $\det(A) = 1$, we will not have a reflection, while if $\det(A) = -1$, we will (note we can still rotate before / after a rotation).

b) We utilize the Spectral Thereom, which tells us that if A is $n \times n$ and symmetric, $A = PDP^{-1} = PDP^T$, where P's columns are an orthonormal eigenbasis of $\mathbb{R}^n$, and D is a diagonal matrix with entries $\lambda_1, \dots, \lambda_n$.

Let $A = U \Sigma V^T$, where $A \in \mathbb{R}^{m \times n}$. Then,

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

Since V is orthogonal, $V^T V = I$, so

$AA^T = U \Sigma \Sigma^T U^T$. Since $(AA^T)^T = A^{T^T} A^T = AA^T$, AA^T is symmetric, so we can use spectral decomposition to conclude

$\boxed{\text{the left singular vectors of } A \text{ are the eigenvectors of } AA^T.}$

Similarly, since $A^T A = (U \Sigma V^T)^T (U \Sigma V^T) = V \Sigma^T U^T U \Sigma V^T$
$= V \Sigma^T \Sigma V^T$,

$\boxed{\text{the right singular vectors of } A \text{ are the eigenvectors of } A^T A.}$

ii) From i), we know $AA^T = U \Sigma \Sigma^T U^T$, so by the spectral thereom $\Sigma \Sigma^T$ is the eigenvalue matrix for AA^T.

If A is $m \times n$, then Σ is also $m \times n$, so $\Sigma \Sigma^T$ is $m \times m$. However, A only has $\min(m,n)$ singular values.

Hence, the first $\min(m,n)$ eigenvalues of AA^T correspond to the singular values of A, and any remaining eigenvalues (any remaining diagonal entries in $\Sigma \Sigma^T$) will be 0.

The eigenvalues of $A^T A$ are the same, except we are filling in $\Sigma^T \Sigma$ instead (so we might not need to pad with 0s depending on the dimension of $\Sigma^T \Sigma$).

c) i) False
ii) False
iii) True
iv) True
v) True

(2)

a) i) Using the probability chain rule, we get

$$P(H50 \mid Tails) = \frac{P(H50 \wedge Tails)}{P(Tails)} = \frac{0.5 \cdot 0.5}{P(Tails \mid H50)\, P(H50) + P(Tails \mid H60)\, P(H60)}$$

$$= \frac{0.25}{0.4 \cdot 0.5 + 0.5 \cdot 0.5} \boxed{= \frac{5}{9}}$$

ii) We want to find $P(H50 \mid THHH)$. Using Bayes' Thm, we can rewrite as

$$P(H50 \mid THHH) = \frac{P(THHH \mid H50)\, P(H50)}{P(THHH)}$$

Then using the law of probability we can rewrite $P(THHH)$ as

$$P(THHH) = P(THHH \mid H50)\, P(H50) + P(THHH \mid H60)\, P(H60)$$
$$= 0.5^4 \, 0.5 + 0.4 \cdot 0.6^3 \cdot 0.5$$

Hence, $P(H50 \mid THHH) = \dfrac{0.5^5}{0.5^5 + 0.5 \cdot 0.4 \cdot 0.6^3} \approx \boxed{42\%}$

iii) We first need to find

$$P(9 \text{ heads}) = P(H50)\, P(9 \text{ heads} \mid H50) + P(H55)\, P(9 \text{ heads} \mid H55) + P(60)\, P(9 \text{ heads} \mid H60).$$

We can do this with the binomial distribution. We use

$\binom{n}{k} p^k (1-p)^{n-k}$ with $n = 10$, $k = 9$, and $p = 0.5, 0.55$, and 0.6.

Then, $P(9 \text{ heads} \mid H50) = \binom{10}{9} 0.5^9 \, 0.5^1 = 10 \cdot 0.5^{10} = \frac{5}{512}$,

$P(9 \text{ heads} \mid H55) = \binom{10}{9} 0.55^9 \, 0.45^1 \approx 0.020724$, and

$P(9 \text{ heads} \mid H60) = \binom{10}{9} 0.6^9 \, 0.4^1 \approx 0.04031$

Hence, $P(9 \text{ heads}) = 0.0236$.

Then, $P(H50 \mid 9 \text{ heads}) = \dfrac{P(9 \text{ heads} \mid H50)\, P(H50)}{P(9 \text{ heads})} = \dfrac{5/512 \cdot \frac{1}{3}}{0.0236} \boxed{\approx 14\%}$

$P(H55 \mid 9 \text{ heads}) = \dfrac{0.020724 \cdot 1/3}{0.0236} \boxed{\approx 29\%}$

$P(H60 \mid 9 \text{ heads}) = \dfrac{0.04031 \cdot 1/3}{0.0236} \boxed{\approx 57\%}$

b) We once again use Bayes' Thm.

$$P(\text{pregnant} \mid \text{positive}) = \frac{P(\text{positive} \mid \text{pregnant})\, P(\text{pregnant})}{P(\text{positive})}, \text{ and then}$$

$$P(\text{positive}) = P(\text{positive} \mid \text{pregnant})\, P(\text{pregnant}) + P(\text{positive} \mid \text{not pregnant})\, P(\text{not pregnant})$$

$$P(\text{positive}) = 0.99\,(0.01) + 0.1\,(0.99) = 0.1089$$

Hence $P(\text{pregnant} \mid \text{positive}) = \frac{0.99\,(0.01)}{0.1089} \approx \boxed{9.1\%}$

The number is low due to the high rate of false positives; since only 1% of the population is pregnant, most positive tests will come from false positives instead of true positives.

c) Since expected value is a linear operator, we have

$\mathbb{E}(A\vec{x} + \vec{b}) = \mathbb{E}(A\vec{x}) + \mathbb{E}(\vec{b})$.

Then since $\vec{b}$ is deterministic (not random), $\mathbb{E}(\vec{b}) = \vec{b}$.

Finally, since A is deterministic, we have

$$\mathbb{E}(A\vec{x}) = \mathbb{E}\left[\begin{pmatrix} a_{11} & & a_{1n} \\ & \ddots & \\ a_{n1} & & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}\right] = \begin{pmatrix} a_{11} & & a_{1n} \\ & \ddots & \\ a_{n1} & & a_{nn} \end{pmatrix} \mathbb{E}\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

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

Hence $\boxed{\mathbb{E}(A\vec{x} + \vec{b}) = A\,\mathbb{E}(\vec{x}) + \vec{b}}$

d) We get $\operatorname{cov}(A\vec{x} + \vec{b}) = \mathbb{E}\left((A\vec{x} + \vec{b} - \mathbb{E}(A\vec{x} + \vec{b})) * (A\vec{x} + \vec{b} - \mathbb{E}(A\vec{x} + \vec{b}))^T \right)$

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

$= \mathbb{E}\left((A\vec{x} - A\mathbb{E}(\vec{x}))(A\vec{x} - A\mathbb{E}(\vec{x}))^T \right)$

$= \mathbb{E}\left(A(\vec{x} - \mathbb{E}(\vec{x}))(\vec{x} - \mathbb{E}(\vec{x}))^T A^T \right)$

$= A\,\mathbb{E}\left((\vec{x} - \mathbb{E}(\vec{x}))(\vec{x} - \mathbb{E}(\vec{x}))^T\right) A^T$

$\boxed{= A \operatorname{cov}(\vec{x}) A^T}$ we are able to cancel $\vec{b}$ and factor A since both are deterministic.

3 a) $\nabla_{\vec{x}} \vec{x}^T A\vec{y} = \nabla_{\vec{x}} \vec{x} \cdot A\vec{y}$. Let $\vec{d} = A\vec{y}$, then

$$\nabla_{\vec{x}} (\vec{x} \cdot \vec{d}) = \begin{pmatrix} \partial/\partial x_1 (\vec{x} \cdot \vec{d}) \\ \partial/\partial x_2 (\vec{x} \cdot \vec{d}) \\ \partial/\partial x_n (\vec{x} \cdot \vec{d}) \end{pmatrix} = \begin{pmatrix} \vec{d}_1 \\ \vdots \\ \vec{d}_n \end{pmatrix} = \vec{d} \boxed{= A\vec{y}}$$

b) $\nabla_y \; x^T A y = \nabla_y (A^T x)^T y$. Let $A^T x = d$, then

$$\nabla_y \; x^T A y = \nabla_y \; d^T y = \nabla_y (d_1 y_1 + \dots + d_m y_m) = d = \boxed{A^T x}$$

c) $\vec{x}^T A y = \sum_i^n \sum_j^m a_{ij} x_i y_j$ Then,

$$\nabla_A (\vec{x}^T A y) = \begin{bmatrix} \frac{\partial f}{\partial a_{11}} x_1 y_1 & & \\ & \ddots & \\ & & \frac{\partial f}{\partial a_{nm}} x_n y_m \end{bmatrix} = \begin{bmatrix} x_1 y_1 & & x_1 y_m \\ & \ddots & \\ x_n y_1 & & x_n y_m \end{bmatrix}$$

$$\boxed{= \vec{x} \vec{y}^T}$$

d) $\nabla_x f = \nabla_x (\vec{x}^T A \vec{x}) + \nabla_x (\vec{b}^T \vec{x})$.

From class, we know the gradient of the first term is $A\vec{x} + A^T\vec{x}$.

From (a), we know $\nabla_x (\vec{b}^T \vec{x}) = \vec{b}$. Since gradient is a linear operator,

$$\boxed{\nabla_x f = A\vec{x} + A^T \vec{x} + \vec{b}}$$

e) $tr(AB) = \sum_i \sum_j (ab)_{ij} = \sum_i \sum_j b_{ji} \, a_{ij}$.

Then $\frac{\partial f}{\partial a_{11}} = \frac{\partial}{\partial a_{11}} \left(a_{11} b_{11} + a_{12} b_{21} + \dots + a_{nm} b_{mn} \right) = b_{11}$,

$\frac{\partial f}{\partial a_{12}} = \frac{\partial}{a_{12}} \left(a_{11} b_{11} + a_{12} b_{21} + \dots + a_{nm} b_{mn} \right) = b_{21}$, and

$$\nabla_A f = \begin{bmatrix} b_{11} & b_{21} & b_{31} & \dots & b_{m1} \\ b_{12} & & & & \\ \vdots & & & & \\ b_{1m} & & & & b_{mn} \end{bmatrix}$$

these are swapped in B

As we can see, the matrix is flipped from B.

Hence, $\boxed{\nabla_A f = B^T}$

4 Since $\|\vec{x}\|^2 = tr(\vec{x}^T x)$,

$$f = \frac{1}{2} \sum_{i=1}^{n} \| y^i - W x^i \|^2 = \frac{1}{2} \sum_{i=1}^{n} tr\left((y^i - Wx^i)^T (y^i - Wx^i) \right).$$

Then, using FOIL /distributing,

$$f = \frac{1}{2} \sum_{i=1}^{n} tr\left(y^{iT} y^i - y^{iT} W x^i - (W x^i)^T y^i + (W x^i)^T (W x^i) \right).$$

Then, since $tr(A+B) = tr(A) + tr(B)$, and $tr(AB) = tr(BA)$,

$$f = \frac{1}{2} \sum_{i=1}^{n} \left(tr(y^{iT} y^i) - tr(y^{iT} W x^i) - tr((W x^i)^T y^i) + tr((Wx^i)^T (Wx^i)) \right).$$

Now we apply $\partial / \partial W$, noting $\frac{\partial}{\partial X} tr(AXB) = A^T B^T$ and $\frac{\partial}{\partial X} tr(AX^T B) = BA$.

Then $\frac{\partial f}{\partial W} = \frac{1}{2} \sum_{i=1}^{n} \left(0 - y^{iT^T} x^{iT} - y^i x^{iT} + \frac{\partial}{\partial W} tr\left((Wx^i)^T (Wx^i) \right) \right).$

Using the hint, we get $\frac{\partial}{\partial W} tr\left((Wx^i)^T (Wx^i) \right) = \frac{\partial}{\partial W} tr\left((Wx^i)(Wx^i)^T \right)$

$= \frac{\partial}{\partial W} tr(W x^i x^{iT} W) = W (x^i x^{iT})^T + W (x^i x^{iT})$

Putting it all together, we get

$$D = -2y^i x^{iT} + W(x^i x^{iT})^T + W(x^i x^{iT})$$

$$D = -2y^i x^{iT} + 2W x^i x^{iT}$$, where

$$\frac{\partial f}{\partial W} = \frac{1}{2} \sum_{i=1}^{n} D = 0$$

Now we solve for W. We have

$$\frac{1}{2}(-2y^i x^{iT} + 2W x^i x^{iT}) = 0$$

$$W x^i x^{iT} = y^i x^{iT}$$

$$W = y^i x^{iT} (x^i x^{iT})^{-1}$$ for $i \in [1, n]$, since $(x^i x^{iT})$ is symmetric so is invertible.

Since this holds for all $i \in [1, n]$, we can collapse the x^i and y^i to get:

$$\boxed{W = y x^T (x x^T)^{-1}}$$

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 C147/C247, Winter Quarter 2021, Prof. J.C. Kao, TAs: N. Evirgen, A. Ghosh, S. Mathur, T. Monsoor, G. Zhao

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt

#allows matlab plots to be generated in line
%matplotlib inline
```

Data generation

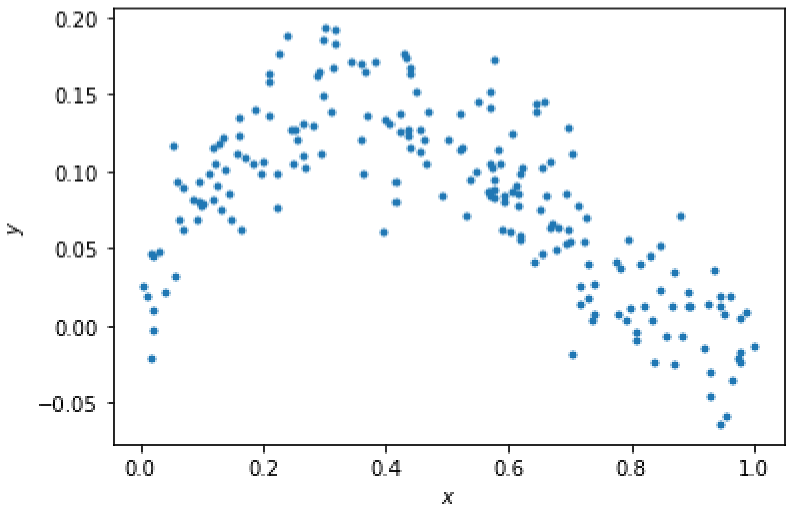
For any example, we first have to generate some appropriate data to use. The following cell generates data according to the model: \$y = x - 2x^2 + x^3 + \epsilon\$

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) We use `np.random.uniform(low=0, high=1)` to generate x values; hence, the generating distribution of x is a uniform distribution over [0, 1).

(2) We use `np.random.normal(loc=0, scale=0.03)` to generate ϵ values; hence, the distribution of ϵ is a normal (Gaussian) distribution with mean 0 and standard deviation 0.03.

Fitting data to the model (5 points)

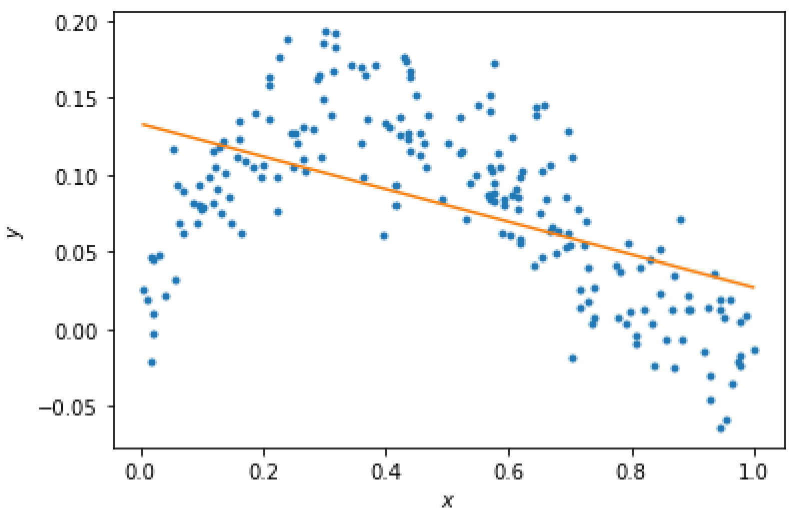
Here, we'll do linear regression to fit the parameters of a model $y = ax + b$.

In [3]:
```
# xhat = (x, 1)
xhat = np.vstack((x, np.ones_like(x)))

# ==================== #
```

```python
# START YOUR CODE HERE #
# ==================== #
# GOAL: create a variable theta; theta is a numpy array whose elements are [a, b]

#we use least squares formula discussed in lecture 3

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

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

In [4]:

```python
# 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 model underfits the data, since the data is clearly a negative quadratic distribution, not a linear distribution.

(2) We could improve the model by doing polynomial regression instead of linear regression.

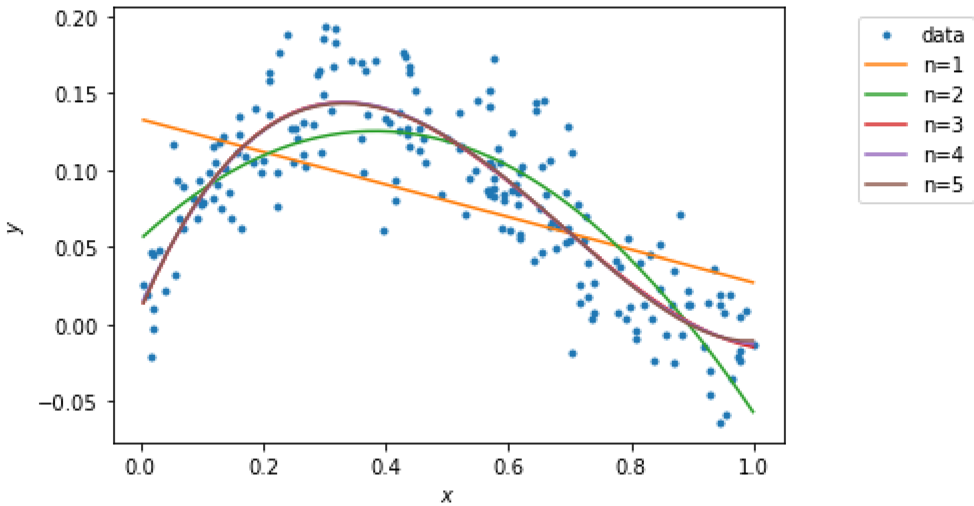
Fitting data to the model (10 points)

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 #
# ==================== #
xhats.append(xhat)
thetas.append(theta)

for i in range(2, 6):
    xhats.append(np.vstack((x**i, xhats[-1])))
    thetas.append(np.linalg.inv(xhats[-1].dot(xhats[-1].T)).dot(xhats[-1].dot(y)))


# 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 respectively.
#   ... etc.

pass

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

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)

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 #
# =================== #

for i in range(N):
    predicted_y = thetas[i].dot(xhats[i])
    error = 0.5 * np.sum((predicted_y - y)**2)
    training_errors.append(error)


# 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.
pass

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

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

```
Training errors are: 
 [0.2379961088362701, 0.10924922209268531, 0.08169603801105374, 0.08165353735296982, 0.08161479195525298]
```

QUESTIONS

(1) What polynomial has the best training error?

(2) Why is this expected?

ANSWERS

(1) The polynomial with degree 5 has the best/least training data with an error of 0.08161479195525298.

(2) This is expected since a higher degree polynomial will always fit the provided data no worse than a lower degree polynomial, since it can simulate a lower degree polynomial by setting its higher degree coefficients to 0. Looking at the graph, we see that n = 3, 4, and 5 are basically overlapping with one another.

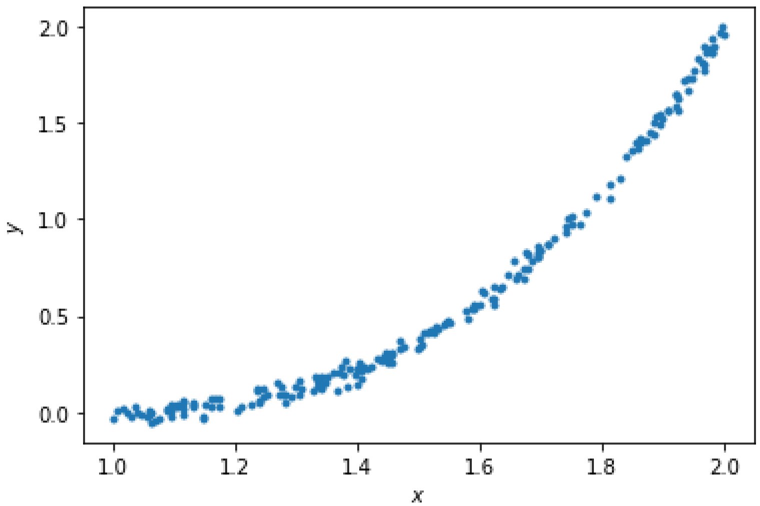
Generating new samples and testing error (5 points)

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

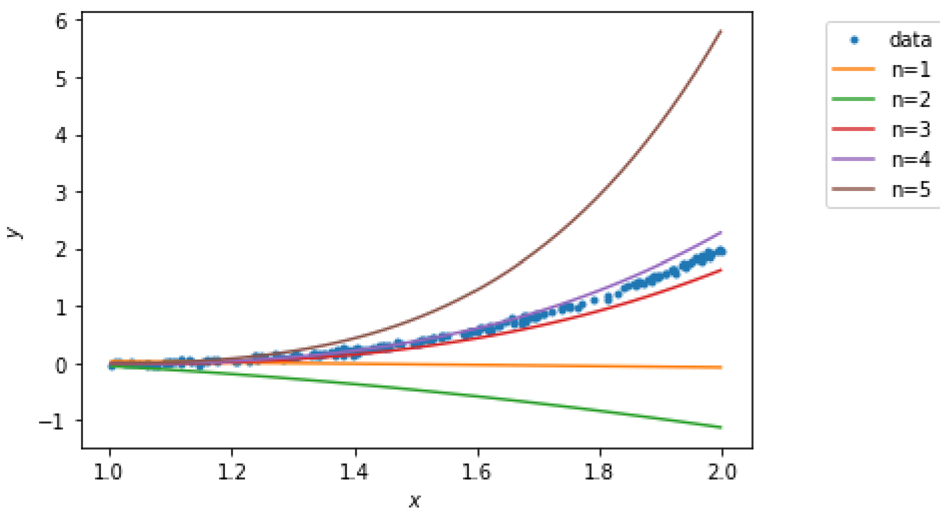
Out[8]: Text(0, 0.5, 'y')



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 #
# ==================== #

for i in range(N):
    predicted_y = thetas[i].dot(xhats[i])
    error = 0.5 * np.sum((predicted_y - y)**2)
    testing_errors.append(error)

# 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.
pass

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

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

```
Testing errors are: 
 [80.86165184550586, 213.19192445057894, 3.1256971082763925, 1.1870765189474703, 214.91021817652626]
```

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 degree 4 has the best testing error (only 1.1870765189474703), slightly edging the polynomial with degree 3.

(2) Due to overfitting, the polynomial with degree 5 tends to model the noise in the distribution, instead of the underlying trends. Hence, it doesn't generalize well to another data sample of the same distribution, such as the test data set.