1. Linear algebra refresher.

(a) i

Because Q is orthogonal, $Q^T = Q^{-1}$. Also, $Q^T(Q^T)^T = Q^TQ = I$ (since Q is orthogonal)

$(Q^T)^T Q^T = QQ^T = I$ (since Q is orthogonal)

So Q^{-1} and Q^T are also orthogonal

ii

assume eigenvector is v

~~eigenvalue is λ~~

Then we have $Av = \lambda v$

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

$\|Av\|^2 = \overline{(Av)}^T (Av)$ by definition of the length

$= \bar{v}^T A^T A v$ because A is real

$= \bar{v}^T v$ because $A^TA = I$ as A is orthogonal

$= \|v\|^2$ by definition of the length

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

Since v is an eigenvector, it is non-zero, and hence $\|v\| \neq 0$

Canceling $\|v\|$, we have $|\lambda|^2 = 1$. Since the length is non-negative, we get $|\lambda| = 1$

iii.

Since Q is orthogonal, $QQ^T = I = Q^TQ$ by definition

Using the fact that $\det(AB) = \det(A)\det(B)$, we have

$\det(I) = 1 = \det(QQ^T) = \det(Q)\det(Q^T) = \det(Q)\det(Q) = [\det Q]^2$

Since we have $[\det(Q)]^2 = 1$, then $\det(Q) = \pm\sqrt{1} = \pm 1$

iv.

Say that Q is orthogonal. Take arbitrary $\vec{x} \in R^n$. We want to show $\|\vec{x}\| = \|Q(\vec{x})\|$. By definition $\|\vec{x}\| = (\vec{x} \cdot \vec{x})^{\frac{1}{2}}$ and $\|T(\vec{x})\| = (T(\vec{x}) \cdot T(\vec{x}))^{\frac{1}{2}}$

Since Q is orthogonal, we know that $\vec{x} \cdot \vec{x} = Q(\vec{x}) \cdot Q(\vec{x})$, so the results show that Q defines a length preserving transformation

(b) Assume we have n eigenvalues $[\lambda_1, \lambda_2, \dots \lambda_n]$, the corresponding eigenvectors $[x_1, x_2, \dots x_n]$. Then we have

$$\Sigma = \begin{bmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \ddots \\ & & & \lambda_n \end{bmatrix} \quad W = [x_1 \; x_2 \; \dots \; x_n]$$

Since $Ax = \lambda x$, we have $AW = W\Sigma \Rightarrow A = W\Sigma W^{-1}$

Since $\|x_i\|_2^2 = 1$, then $W^T = W^{-1}$, $A = W\Sigma W^T$

Since A's SVD decomposition is $A = UDV^T = U\begin{bmatrix}\Sigma \\ 0\end{bmatrix}V^T$

Thus, $A^T = VD^TU^T = V[\Sigma \;\; 0]U^T$

Then $A^TA = VD^TU^TUDV^T$

Since U is orthogonal: $A^TA = VD^TDV^T = V\Sigma^2V^T$

Thus, A's right singular vector V is the W built by A^TA's eigenvector.

and A^TA's eigenvalue is the square of the singular value of A

Same: $AA^T = UDV^TVD^TU^T = UDD^TU^T = U\Sigma^2U^T$

Thus, A's left singular vector is the W built by AA^T's eigenvector

and AA^T's eigenvalue is the square of the singular value of A

(c) i. False. At most n distinct eigenvalues

ii. False. If v_1 and v_2 are eigenvectors of A corresponding to eigenvalues λ_1 and λ_2 respectively, then the sum v_1+v_2 is not guaranteed to be an eigenvector.
In fact, unless v_1 and v_2 are scalar multiples of each other (i.e. $v_1=\lambda v_2$), v_1+v_2 will not satisfy the eigenvector equation $Av=\lambda v$ for any single eigenvalue λ

iii. Correct

iv. Correct

v. Correct

2 (a) since $n=1,2,3,4\ldots$ where n denotes the round in which the duel ends, partitions the sample space so by law of total probability.

i. $P(A \text{ not hit}) = \sum_{n=1}^{\infty} P(A \text{ not hit}, n)$

Now, if duel ends in n rounds and A isn't hit,

then $\underset{1}{\underline{BM}}\ \underset{2}{\underline{BM}}\ \underset{3}{\underline{BM}}\ \underset{4}{\underline{BM}}\ \cdots\ \underset{n-1}{\underline{BM}}\ \underset{n}{\underline{A}}$

So, $P(A \text{ not hit}, n) = (1-P_A)^{n-1}(1-P_B)^{n-1} P_A(1-P_B)$

$= P_A(1-P_A)^{n-1}(1-P_B)^{n}$

So $P(A \text{ not hit}) = P_1 \sum_{n=1}^{\infty}(1-P_A)^{n-1}(1-P_B)^{n}$

$= P_1 \sum_{n=1}^{\infty}(1-P_A)^{n-1}(1-P_B)^{n}$

Since $\sum_{n=1}^{\infty}(1-P_A)^{n-1}(1-P_B)^{n}$ is the sum of a geometric series with $a = 1-P_B$, $r = (1-P_A)(1-P_B)$

So, $P(A \text{ not hit}) = \dfrac{P_A(1-P_B)}{1-(1-P_A)(1-P_B)}$

ii. $P(\text{both duelists are hit}) = \sum_{n=1}^{\infty}(1-P_A)^{n-1}(1-P_B)^{n-1} P_A P_B$

$= P_A P_B \sum_{n=1}^{\infty}(1-P_A)^{n-1}(1-P_B)^{n-1}$

$= \dfrac{P_A P_B}{1-(1-P_A)(1-P_B)}$

iii. Since duel can end after n th round of shots in 3 possible ways: ① A hit ② B hit ③ Both hit

Then, by laws of total probability

P(duel ends after n^{th} round)

$$=(1-P_A)^{n-1}(1-P_B)^{n-1}(1-P_A)P_B+(1-P_A)^{n-1}(1-P_B)^{n-1}P_A(1-P_B)$$

$$+(1-P_A)^{n-1}(1-P_B)^{n-1}P_AP_B$$

$$=\left[(1-P_A)(1-P_B)\right]^{n-1}\left[1-(1-P_A)(1-P_B)\right]$$

iv. $P(\text{Duel ends after } n^{th} \text{ round} \mid A \text{ is not hit})$

$$= \frac{P(\text{Duel ends after } n^{th} \text{ round} + A \text{ is not hit})}{P(A \text{ is not hit})}$$

$$= \frac{[(1-P_A)(1-P_B)]^{n-1}[(1-P_B)P_A]}{[(1-P_B)\cdot P_A / 1-(1-P_B)(1-P_A)]}$$

$$= \frac{[(1-P_A)(1-P_B)]^{n-1}}{\frac{1}{1-(1-P_B)(1-P_A)}} = [(1-P_A)(1-P_B)]^{n-1}[1-(1-P_A)(1-P_B)]$$

v. $P(\text{end in the } n\text{-th} \mid \text{both are shot}) = \frac{P(\text{end in the } n\text{-th, both are shot})}{P(\text{both are shot})}$

From ii, we get $P(\text{both are shot}) = \frac{P_A P_B}{P_A + P_B - P_A P_B}$

$$P(\text{end in } n\text{th} \mid \text{both hit}) = \frac{[(1-P_A)(1-P_B)]^{n-1} P_A P_B}{\frac{P_A P_B}{P_A + P_B - P_A P_B}}$$

$$= [(1-P_A)(1-P_B)]^{n-1} \cdot (P_A + P_B - P_A P_B)$$

2(b)

i. suppose we define X_i as the bernoulli random variable

$$X_i = \begin{cases} 1, & \text{the ith faculty is isolated} \\ 0, & \text{the ith faculty is not isolated} \end{cases}$$

Then
$$X = \sum_{i=1}^{18} X_i$$

Now, $E[X_i] = P(X_i = 1)$

$P(X_i = 1) = P(\text{ith faculty is isolated})$

Define
E: ith facluty is ECE
C: ith faculty is CSE
M: ith faculty is Math

Since E, C, M partitions the sample space so by law of total probability,

$P(\text{ith faculty is isolated})$

$= P(i|E)P(E) + P(i|C)P(C) + P(i|M)P(M)$

$= \left(\frac{12}{17}\right)\cdot\left(\frac{11}{16}\right)\frac{1}{3}\cdot 3 = \frac{33}{68}$

Hence by Linearity of expectations

$E[X] = 18E[X_i] = 18\cdot\frac{33}{68} = 8.735$

ii. Define Y_i as the Bernoulli random variable

$$Y_i = \begin{cases} 1, & \text{ith faculty is semi-happy} \\ 0, & \text{ith faculty is not semi-happy} \end{cases}$$

$Y = \sum_{i=1}^{15} Y_i$ Now, $E[Y_i] = P(Y_i = 1)$

$P(Y_i = 1) = P(i\text{th faculty is semi-happy})$

Use the same event defined: E, C, M

$P(i\text{th faculty is semi-happy})$

$= P(i|E)P(E) + P(i|C)P(C) + P(i|M)P(M)$

$= \left[\frac{12}{17} \cdot \frac{5}{16} + \frac{5}{17} \cdot \frac{12}{16}\right] \cdot \frac{1}{3} \cdot 3 = \frac{30}{68}$

By linearity of expectations

$E[Y] = 15E[Y_i] = 18 \cdot \frac{30}{68} = 7.941$

ii. Define Z_i as the Bernoulli random variable

$$Z_i = \begin{cases} 1, & i\text{th faculty is joyous} \\ 0, & i\text{th faculty is not joyous} \end{cases}$$

Then $Z = \sum_{i=1}^{15} Z_i$ $E[Z_i] = P(Z_i = 1)$

$P(Z_i = 1) = P(i\text{th faculty is joyous})$

Use the same event defined before: E, C, M

$P(i\text{th faculty is joyous}) = P(i|E)P(E) + P(i|C)P(C) + P(i|M)P(M)$

$= \left[\frac{5}{17} \cdot \frac{4}{16}\right] \cdot \frac{1}{3} \cdot 3 = \frac{5}{68}$

$E[Z] = 15E[Z_i] = 18 \times \frac{5}{68} = 1.324$

2. (c) Let's define the following events:

D: man has a dangerous type of the disease

T: man has a positive LSA test

From the problem statement, we are given the following quantities

$$P(T \mid D) = 0.9 \quad P(T \mid D^c) = 0.01 \quad P(D) = 0.0005$$

i. By Bayes law,

$$P(D \mid T) = \frac{P(T \mid D) P(D)}{P(T \mid D) P(D) + P(T \mid D^c) P(D^c)}$$

$$= \frac{0.9 \times 0.0005}{0.9 \times 0.0005 + 0.01 \times 0.9995} = 0.043$$

ii. By Bayes Law,

$$P(D \mid T^c) = \frac{P(T^c \mid D) P(D)}{P(T^c \mid D) P(D) + P(T^c \mid D^c) P(D^c)} = \frac{0.1 \times 0.0005}{0.1 \times 0.0005 + 0.99 \times 0.9995}$$

$$= 0.00005052 8$$

(d) $E(x)$ is the expected value of vector x. $E(Ax+b)$ is the expectation of the vector $Ax+b$ where we have known the dimension of x is n. Then A and b are deterministic and the dimension of A is $m \times n$, and the dimension of b is m.

Using the linearity of expectation to break down the calcution:

$$E(Ax+b) = \mathbb{E}([Ax_1+b, Ax_2+b \ldots Ax_n+b]^T)$$
$$= [\mathbb{E}(Ax_1+b), \mathbb{E}(Ax_2+b), \ldots, \mathbb{E}(Ax_n+b)]^T$$

If A and b are deterministic, we can pull them out of the expectation:

$$E(Ax+b) = A[\mathbb{E}(x_1), \mathbb{E}(x_2), \ldots, \mathbb{E}(x_n)]^T + b$$

Since $x_1, x_2 \ldots x_n$ are identically distributed random variables, their expectations are the same. Let's denote this common expectation as μ.

$$\mathbb{E}(Ax+b) = A[\mathbb{E}(x_1), \mathbb{E}(x_2) \ldots \mathbb{E}(x_n)]^T + b = A[\mu, \mu, \ldots \mu]^T + b$$
$$= A\mu + b$$

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

If A is a deterministic matrix and $E(x)$ is a constant vector, we can further simplify.

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

Using the properties of covariance, we can rewrite the expession as:

$$\operatorname{cov}(Ax+b) = A \operatorname{cov}(x) A^T$$

3. (a) $\nabla_x x^T A y = Ay$

(b) $\nabla_y x^T A y = \nabla_y (Ay)^T x = \nabla_y y^T A^T x = A^T x$

(c) $\nabla_A x^T A y = x y^T$

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

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

(f) $f = tr(BA + A^T B + A^2 B)$

$\nabla_A f = \nabla_A [tr(BA) + tr(A^T B) + tr(A^2 B)]$

$= \nabla_A [tr(BA) + tr(A^T B) + tr(AAB)]$

$= \nabla_A [tr(BA) + tr(A^T B) + tr(BAIA)]$ I is the identity matrix $d\,tr(AXBX) = A^T X^T B^T + B^T X^T A^T$

$= B^T + B + B^T A^T + A^T B^T$

(g) $f = \|A + \lambda B\|^2$

$\nabla_A f = \nabla_A tr[(A+\lambda B)(A+\lambda B)^T]$

$= \nabla_A tr[(A+\lambda B)(A+\lambda B)^T]$

$= \nabla_A tr[(A+\lambda B)(A^T+\lambda B^T)]$

$= \nabla_A tr[AA^T + \lambda AB^T + \lambda BA^T + \lambda^2 BB^T]$

$= 2A + \lambda B + \lambda B^T$

4.

To find the optimal W, we take the derivate of the loss function

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

with respect to W and set the derivate to zero

Expand the loss function $L(W)$:

$$L(W) = \frac{1}{2}(y - Wx)^T (y - Wx) = \frac{1}{2}(y^T - x^T W^T)(y - Wx)$$

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

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

Take the derivative of the above formula with respect to W and make the derivative zero.

$$\frac{\partial L(W)}{\partial W} = \frac{1}{2}[0 - yx^T - yx^T + W(xx^T + xx^T)] = \frac{1}{2}(-2yx^T$$

$$+ 2Wxx^T) = -yx^T + Wxx^T = 0$$

Then $Wxx^T = yx^T$

Now we can find the optimal parameter W by solving the above equation

A common method for solving equations is to use Normal Equation:

$$W = YX^T(XX^T)^{-1}$$

5. $$J(\theta) = \frac{1}{2}\sum_{i=1}^{N}[(y^{(i)})^2 - 2y^{(i)}\theta^T \hat{x}^{(i)} + (\theta^T x^{(i)})^2) + \frac{\lambda}{2}\sum_{j=1}^{M}\theta_j^2$$

$$\nabla_\theta J(\theta) = -\sum_{i=1}^{N}(y^{(i)} - \theta^T x^{(i)})x^{(i)} + \lambda\theta$$

$$0 = -\sum_{i=1}^{N}(y^{(i)} - \theta^T x^{(i)})x^{(i)} + \lambda\theta$$

Matrix form:

$$-(X^T Y) - X^T X\theta) + \lambda\theta = 0$$

$$-X^T Y + X^T X\theta + \lambda\theta = 0$$

$$X^T Y = X^T X\theta + \lambda\theta$$

$$\theta = (X^T X + \lambda I)^{-1} X^T Y$$

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 2024, Prof. J.C. Kao, TAs: T.Monsoor, Y. Liu, S. Rajesh, L. Julakanti, K. Pang

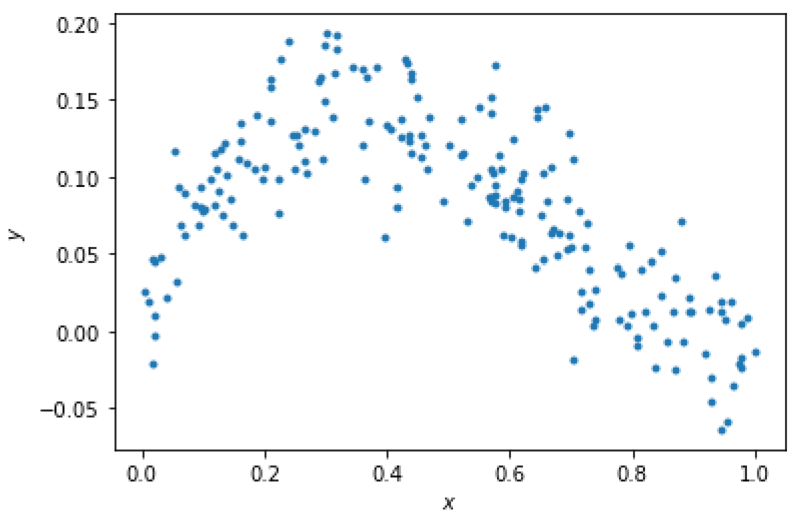
In [84]:
```
import numpy as np
import matplotlib.pyplot as plt

#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 [85]:
```
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[85]: `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 a random number uniformly distributed on the interval [0, 1). The values of x range from 0 (inclusive) to 1 (exclusive) and each value has an equal probability of being seen as uniformly distributed. The resulting y is calculated from the values of x, with some normally distributed noise with mean 0 and standard deviation 0.03 added.

(2) The generated noise is normally distributed with a mean of 0 and a standard deviation of 0.03 (also known as a Gaussian distribution).

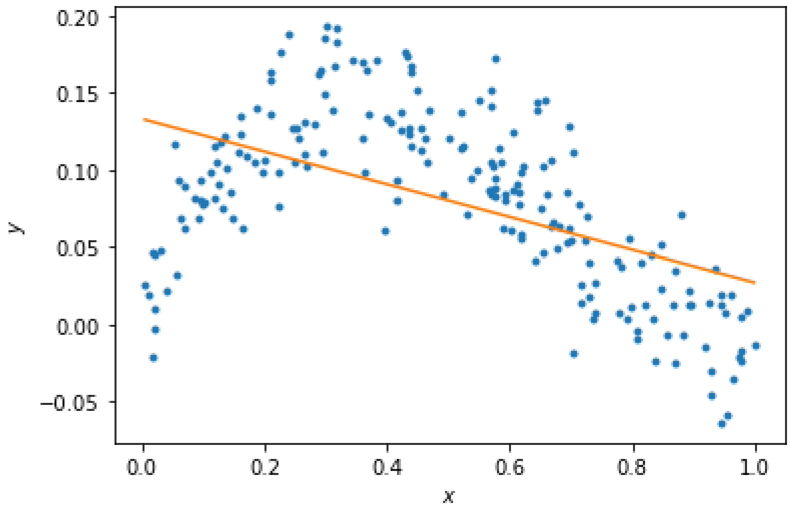
Fitting data to the model (5 points)

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

In [27]:
```
from sklearn.linear_model import LinearRegression
# 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.zeros(2) # please modify this line

X = x.reshape(-1, 1)
model = LinearRegression()
model.fit(X, y)
a = model.coef_[0]  # Slope
b = model.intercept_  # Intercept
theta = np.array([a, b])

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

In [28]:
```
# 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[28]:
```
[<matplotlib.lines.Line2D at 0x1b17b876100>]
```



QUESTIONS

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

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

ANSWERS

(1) under-overfit the data.

(2) If you want to fit y with a linear model, you need to extend the features by expanding the original feature x. Specifically, expand x into polynomial features such as x, x^2, x^3, and so on. These polynomial features are then used as inputs to construct a linear model to fit y.

Fitting data to the model (5 points)

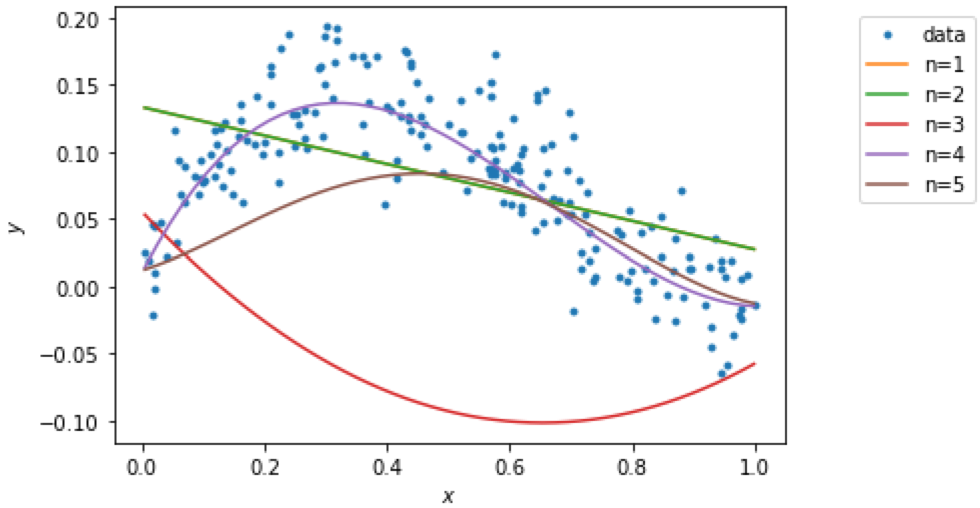
Here, we'll now do regression to polynomial models of orders 1 to 5. Note, the order 1 model is the linear model you prior fit.

In [72]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
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 f
#   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
#   ... etc.

thetas.append([model.coef_[0], model.intercept_])
```

```
for order in range(1, N):
    # Create polynomial features
    poly_features = PolynomialFeatures(degree=order)
    X_poly = poly_features.fit_transform(X)

    # Create a new linear regression model
    model_poly = LinearRegression()

    # Fit the model to the polynomial features
    model_poly.fit(X_poly, y)

    # Get the estimated coefficients
    theta_poly = np.append(model_poly.coef_, model_poly.intercept_)

    # Add the theta array to the list
    thetas.append(theta_poly)



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

In [74]:

```
# 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,:],  np.dot(np.array(thetas[i]),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 (5 points)

Here, we'll now calculate the training error of polynomial models of orders 1 to 5.

In [75]:
```
from sklearn.metrics import mean_squared_error

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
# Order 1: Linear model (already fitted)
y_pred = model.predict(X)
mse = mean_squared_error(y, y_pred)
training_errors.append(mse)
for order in range(2, 6):
    # Create polynomial features
    poly_features = PolynomialFeatures(degree=order)
    X_poly = poly_features.fit_transform(X)

    # Create a new linear regression model
    model_poly = LinearRegression()

    # Fit the model to the polynomial features
    model_poly.fit(X_poly, y)

    # Predict the target values
    y_pred_poly = model_poly.predict(X_poly)

    # Calculate the mean squared error
    mse_poly = mean_squared_error(y, y_pred_poly)

    # Add the training error to the list
    training_errors.append(mse_poly)


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

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

```
Training errors are: 
 [0.0023799610883627007, 0.001092492220926853, 0.0008169603801105373, 0.000816535
373529698, 0.0008161479195525291]
```

QUESTIONS

(1) What polynomial has the best training error?

(2) Why is this expected?

ANSWERS

(1) The 5th order polynomial training error is the best.

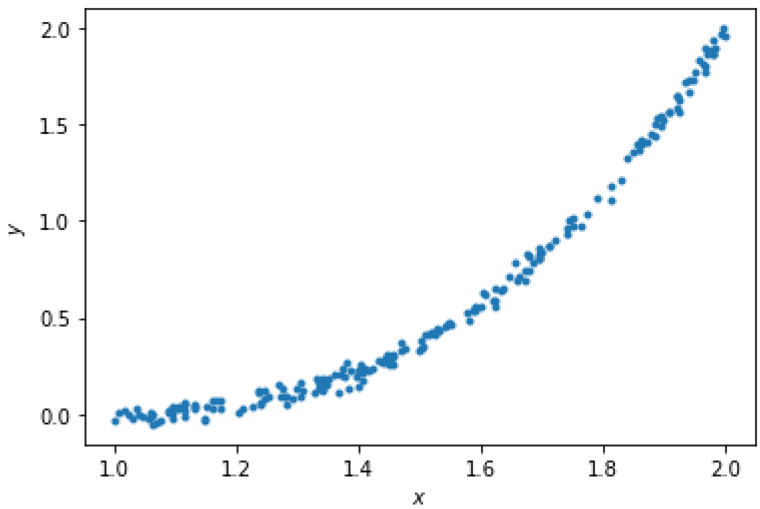
(2) Since the amount of data is small and there is an error in the data, if the factorial is too large, the error is reduced, but there is an overfitting problem.

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 [76]:
```
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('$y$')
```

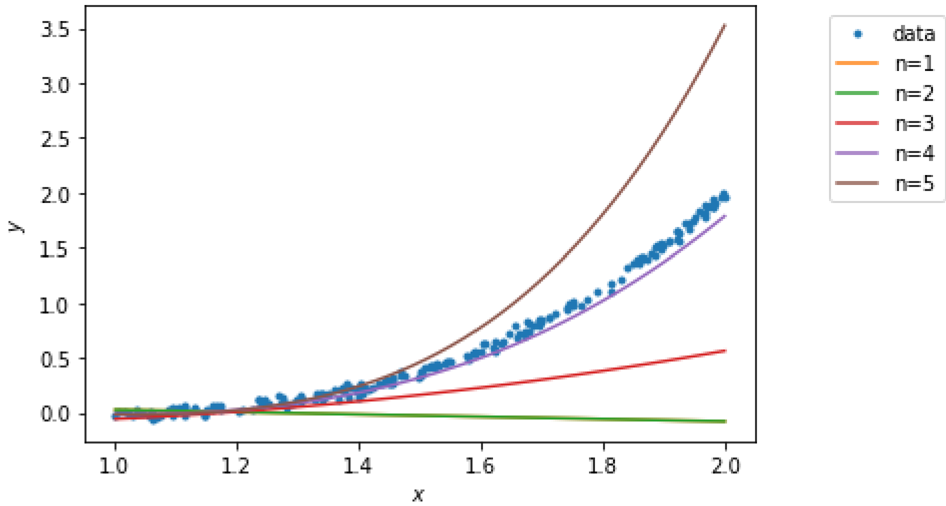
Out[76]: Text(0, 0.5, 'y')



In [77]:
```
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 [83]:
```
# 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,:], np.dot(thetas[i],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 [81]:
```
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
# pass
X = x.reshape(-1, 1)
y_pred = model.predict(X)
mse = mean_squared_error(y, y_pred)
training_errors.append(mse)
```

```
for order in range(0, N):
    # Create polynomial features
    poly_features = PolynomialFeatures(degree=order)
    X_poly = poly_features.fit_transform(X)

    # Create a new linear regression model
    model_poly = LinearRegression()

    # Fit the model to the polynomial features
    model_poly.fit(X_poly, y)

    # Predict the target values
    y_pred_poly = model_poly.predict(X_poly)

    # Calculate the mean squared error
    mse_poly = mean_squared_error(y, y_pred_poly)

    # Add the training error to the list
    testing_errors.append(mse_poly)


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

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

```
Testing errors are:
 [0.3755032551553172, 0.040027663041497165, 0.0013057361468590306, 0.000938745244
2745558, 0.0009306959438728502]
```

QUESTIONS

(1) What polynomial has the best testing error?

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

ANSWERS

(1) 5 degree polynomials

(2) It's overfitting.

```
In [ ]:
```