1 Linear Algebra Refresher

(a) i.

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

$$AA^T = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}\begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} \left(\frac{\sqrt{2}}{2}\right)\left(\frac{\sqrt{2}}{2}\right) + \left(\frac{\sqrt{2}}{2}\right)\left(\frac{\sqrt{2}}{2}\right) & \left(\frac{\sqrt{2}}{2}\right)\left(\frac{\sqrt{2}}{2}\right) + \left(\frac{\sqrt{2}}{2}\right)\left(-\frac{\sqrt{2}}{2}\right) \\ \left(\frac{\sqrt{2}}{2}\right)\left(\frac{\sqrt{2}}{2}\right) + \left(\frac{\sqrt{2}}{2}\right)\left(-\frac{\sqrt{2}}{2}\right) & \left(\frac{\sqrt{2}}{2}\right)\left(\frac{\sqrt{2}}{2}\right) + \left(-\frac{\sqrt{2}}{2}\right)\left(-\frac{\sqrt{2}}{2}\right) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_2 \quad \checkmark$$

Find eigenvalues & eigenvectors:

$$0 = \det(A - \lambda I_2) = \begin{bmatrix} \frac{\sqrt{2}}{2} - \lambda & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} - \lambda \end{bmatrix} = \left(\frac{\sqrt{2}}{2} - \lambda\right)\left(-\frac{\sqrt{2}}{2} - \lambda\right) - \left(\frac{\sqrt{2}}{2}\right)^2 = \lambda^2 - 1$$
$$= (\lambda - 1)(\lambda + 1)$$

$\lambda_1 = -1$

$$A\vec{v}_1 = \lambda_1 \vec{v}_1$$
$$\vec{0} = (A - \lambda_1)\vec{v}_1$$
$$\vec{0} = \begin{bmatrix} \frac{\sqrt{2}}{2} + 1 & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} + 1 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$
$$\vec{0} = \begin{bmatrix} \frac{2+\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{2-\sqrt{2}}{2} \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$
$$\vec{0} = \begin{bmatrix} 1 & \sqrt{2}-1 \\ \frac{\sqrt{2}}{2} & \frac{2-\sqrt{2}}{2} \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$
$$\vec{0} = \begin{bmatrix} 1 & -1+\sqrt{2} \\ 1 & -1+\sqrt{2} \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$
$$\vec{0} = \begin{bmatrix} 1 & -1+\sqrt{2} \\ 0 & 0 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$v_1 + (-1+\sqrt{2})v_2 = 0$
$v_1 = (1-\sqrt{2})v_2$ → $v_2 = t$, $v_1 = (1-\sqrt{2})t$

$$\vec{v}_1 = \begin{bmatrix} 1-\sqrt{2} \\ 1 \end{bmatrix}$$

$\lambda_2 = 1$

$$A\vec{v}_2 = \lambda_2 \vec{v}_2$$
$$\vec{0} = (A - \lambda_2)\vec{v}_2$$
$$\vec{0} = \begin{bmatrix} \frac{\sqrt{2}}{2} - 1 & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} - 1 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$
$$\vec{0} = \begin{bmatrix} \frac{-2+\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{-2-\sqrt{2}}{2} \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$
$$\vec{0} = \begin{bmatrix} -1 & 1+\sqrt{2} \\ \frac{\sqrt{2}}{2} & \frac{-2-\sqrt{2}}{2} \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$
$$\vec{0} = \begin{bmatrix} -1 & 1+\sqrt{2} \\ 1 & -1-\sqrt{2} \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$
$$\vec{0} = \begin{bmatrix} 1 & -1-\sqrt{2} \\ 0 & 0 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$v_1 + (-1-\sqrt{2})v_2 = 0$
$v_1 = (-1-\sqrt{2})v_2$ → $v_2 = t$, $v_1 = (-1-\sqrt{2})t$

$$\vec{v}_2 = \begin{bmatrix} -1-\sqrt{2} \\ 1 \end{bmatrix}$$

The eigenvalues are ± 1, as the eigenvalues of an orthogonal matrix are ± 1.
The eigenvectors are orthogonal as $\begin{bmatrix} 1-\sqrt{2} \\ 1 \end{bmatrix} \cdot \begin{bmatrix} -1-\sqrt{2} \\ 1 \end{bmatrix} = -1 - \sqrt{2} + \sqrt{2} + 2 + 1 = 0$

ii. Show that an orthogonal matrix has eigenvalues with norm 1.

$$A^TA = I$$
$$\vec{v}^TA^TA\vec{v} = \vec{v}^T\vec{v}$$
$$(A\vec{v})^TA\vec{v} = \vec{v}^T\vec{v}$$

($A\vec{v} = \lambda\vec{v}$)

$$(\lambda\vec{v})^T\lambda\vec{v} = \vec{v}^T\vec{v}$$

($\lambda^T\lambda = \lambda^2$ b/c λ is scalar)

$$\lambda^2\vec{v}^T\vec{v} = \vec{v}^T\vec{v}$$
$$0 = \vec{v}^T\vec{v} - \lambda^2\vec{v}^T\vec{v}$$
$$0 = \vec{v}^T\vec{v}(1-\lambda^2)$$

Since $\vec{v}^T\vec{v} \neq 0$, $1-\lambda^2=0 \rightarrow 0=(1+\lambda)(1-\lambda)$

$$\lambda = \pm 1$$
$$\|\lambda\| = 1$$

iii. Show that eigenvectors of an orthogonal matrix corresponding to distinct eigenvalues are orthogonal.

Let λ_1 & λ_2 be distinct eigenvalues for matrix A.

$A\vec{v}_1 = \lambda_1\vec{v}_1$ & $A\vec{v}_2 = \lambda_2\vec{v}_2$

$$(A\vec{v}_1)^TA\vec{v}_2 = (A\vec{v}_1)^T\lambda_2\vec{v}_2$$
$$(A\vec{v}_1)^TA\vec{v}_2 = (\lambda_1\vec{v}_1)^T\lambda_2\vec{v}_2$$
$$\vec{v}_1^TA^TA\vec{v}_2 = \vec{v}_1^T\lambda_1^T\lambda_2\vec{v}_2$$

($\lambda_1^T\lambda_2 = \lambda_1\lambda_2$ b/c λ_1 & λ_2 are scalars)

($A^TA = I_n$)

$$\vec{v}_1^T\vec{v}_2 = \lambda_1\lambda_2\vec{v}_1^T\vec{v}_2$$
$$0 = \vec{v}_1^T\vec{v}_2(1-\lambda_1\lambda_2)$$

Since $\lambda_1\lambda_2=1$ b/c this means $\lambda_1=\lambda_2=1$ or $\lambda_1=\lambda_2=-1$, $\vec{v}_1^T\vec{v}_2=0$ so $\vec{v}_1$ & $\vec{v}_2$ are orthogonal.

iv. Under $A\vec{x}$, $\vec{x}$ is rotated and/or reflected while its length is preserved.

(b) i. As in Single value decomposition, we see $A = UDV^T$.

Left-singular vectors of A are the eigenvectors of AA^T.
↳ i.e. column vectors of U.

Right-singular vectors of A are the eigenvectors of A^TA.
↳ i.e. column vectors of V.

we call σ a singular value for A iff ① $A\vec{v} = \sigma\vec{u}$ and ② $A^T\vec{u} = \sigma\vec{v}$ s.t. $\vec{u}$ is a left-singular vector and $\vec{v}$ is a right-singular vector of A. From these, we see for AA^T and A^TA

$$A^T\vec{u} = \sigma\vec{v} \quad ②$$
$$AA^T\vec{u} = \sigma A\vec{v}$$
$$AA^T\vec{u} = \sigma(\sigma\vec{u}) \quad \text{② } A\vec{v} = \sigma\vec{u}$$
$$AA^T\vec{u} = \sigma^2\vec{u}$$

$$A\vec{v} = \sigma\vec{u} \quad ①$$
$$A^TA\vec{v} = \sigma A^T\vec{u}$$
$$A^TA\vec{v} = \sigma(\sigma\vec{v}) \quad \text{① } A^T\vec{u} = \sigma\vec{v}$$
$$A^TA\vec{v} = \sigma^2\vec{v}$$

This agrees with the definition that the singular values (σ) of matrix A are the square roots of the non-zero eigen values of AA^T and A^TA (their eigenvalues are equal). On the left, σ^2 is eigenvalue of AA^T and σ is its square root. On the right, σ^2 is eigenvalue of A^TA and σ is its square root.

ii. the singular values of A are the square roots of the nonzero eigenvalues of A^TA as well as the square roots of the nonzero eigenvalues of AA^T, because the nonzero eigenvalues of A^TA and AA^T are the same.

(c) i. False.

Some eigenvalues can be equal, yielding fewer than n distinct eigenvalues. For example, take a linear operator L defined by

$L(\vec{x}) = A\vec{x}$ where $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$. We find its eigenvalues:

$$0 = \det\begin{bmatrix} 1-\lambda & 1 & 1 \\ 1 & 1-\lambda & 1 \\ 1 & 1 & 1-\lambda \end{bmatrix} = (1-\lambda)(1-2\lambda+\lambda^2-1) - (1-\lambda-1) + (1-(1-\lambda))$$
$$= \lambda^2 - 2\lambda - \lambda^3 + 2\lambda^2 + \lambda + \lambda$$
$$= \lambda^2(\lambda-3) \rightarrow \lambda_1 = 0 \text{ w/ algebraic multiplicity } = 2, \quad \lambda_2 = 3 \text{ w/ algebraic multiplicity } = 1$$

→ Matrix A in 3D vector space has only 2 distinct eigenvalues.

ii. **False.** Counterexample of matrix w/ linearly independent eigenvectors:

$A = \begin{bmatrix} 2 & -2 & 1 \\ -1 & 3 & -1 \\ -2 & -4 & 3 \end{bmatrix}$ has eigenvalues w/ eigenvectors: $\lambda_1 = 1$, $\vec{v}_1 = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$; $\lambda_2 = 2$, $\vec{v}_2 = \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix}$; $\lambda_3 = 5$, $\vec{v}_3 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$

We form the matrix $[\vec{v}_1\ \vec{v}_2\ \vec{v}_3]$ and find $\det \begin{bmatrix} 0 & -1 & -1 \\ 1 & 1 & 1 \\ 2 & 2 & -1 \end{bmatrix} = 1(-1) - 2(1) - (2-2) = -3 \neq 0$

→ Eigenvectors are linearly independent, so it is not possible in this case that a non-zero sum of two eigenvectors of A is another eigenvector.

iii. **True.**

By definition positive semidefinite matrices have all non-negative eigenvalues and guarantee that $\forall \vec{x}$, $\vec{x}^T A \vec{x} \geq 0$.

iv. **True.**

I assume that the statement means: distinct non-zero eigenvalues.

Let $A = I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow 0 = \det \begin{bmatrix} 1-\lambda & 0 & 0 \\ 0 & 1-\lambda & 0 \\ 0 & 0 & 1-\lambda \end{bmatrix} = (1-\lambda)^3 \rightarrow \lambda_1 = 1$ w/ algebraic multiplicity = 3.

The rank of A is 3, which is greater than its one distinct eigenvalue.

v. **True**

$A\vec{v}_1 = \lambda\vec{v}_1 \qquad A\vec{v}_2 = \lambda\vec{v}_2$

$A\vec{v}_1 + A\vec{v}_2 = \lambda\vec{v}_1 + \lambda\vec{v}_2$

$A(\vec{v}_1 + \vec{v}_2) = \lambda(\vec{v}_1 + \vec{v}_2)$

→ $\vec{v}_1 + \vec{v}_2$ is also an eigenvector of A.

2. Probability Refresher

(a) i. Let A = this is H50 coin
Let B = coin lands tails

$$P(A|B) = \frac{P(A) \times P(B|A)}{P(B)} = \frac{0.5 \times 0.5}{0.95} = 0.556$$

probability coin lands tails $= P(B) = P(A)P(B|A) + P(\text{not } A)P(B|\text{not } A)$ (this is H60 coin)
$= 0.5 \times 0.5 + 0.5 \times 0.4$
$= 0.95$

ii. Let A = this is H50 coin
Let H = coin lands heads
Let T = coin lands tails

$$P(A|T,H,H,H) = \frac{P(A) \times P(T,H,H,H|A)}{P(T,H,H,H)}$$ ← T & H are conditionally independent given A

$$= \frac{P(A) \times P(T|A)P(H|A)P(H|A)P(H|A)}{P(T,H,H,H)}$$

$$= \frac{0.5 \times (0.5 \times 0.5 \times 0.5 \times 0.5)}{0.5(0.5)^1(0.5)^3 + 0.5(0.4)^1(0.6)^3}$$

$$= 0.9197$$

to explain this step

If X is a sequence w/ k tails & n-k heads, and p = prob that H50 lands tails and q = prob that H60 lands tails: $P(X) = \frac{1}{2}p^k(1-p)^{n-k} + \frac{1}{2}q^k(1-q)^{n-k}$
this can be explained: let A = event that H50 is used, so $\bar{A}$ = event that H60 is used. Thus, $P(X|R) = p^k(1-p)^{n-k}$ and $P(X|\bar{R}) = q^k(1-q)^{n-k}$.

$$P(X) = P(X,R) + P(X,\bar{R})$$
$$= P(X|R)P(R) + P(X|\bar{R})P(\bar{R})$$ ($P(R) = P(\bar{R}) = \frac{1}{2}$)
$$= \frac{1}{2}P(X|R) + \frac{1}{2}P(X|\bar{R})$$
$$= \frac{1}{2}p^k(1-p)^{n-k} + \frac{1}{2}q^k(1-q)^{n-k}$$

iii. Let A = this is H50 coin H = coin lands heads
B = this is H55 coin T = coin lands tails
C = this is H60 coin X = get sequence w/ 9 heads, 1 tail

$$P(X) = \frac{1}{3}(0.5)^9(0.5)^1 + \frac{1}{3}(0.55)^9(0.45)^1 + \frac{1}{3}(0.60)^9(0.40)^1 = 0.00236$$
(heads: 0.5, tails: 0.5)

$$P(A|X) = \frac{P(A)\times P(X|A)}{P(X)} = \frac{P(A)\times P(H|A)^9 P(T|A)^1}{P(X)} = \frac{\frac{1}{3}\times(0.5)^9(0.5)^1}{0.00236}$$

$$P\left(\text{Coin is H50} \,\middle|\, \text{9 heads, 1 tail}\right) = 0.138$$

$$P(B|X) = \frac{P(B)\times P(X|B)}{P(X)} = \frac{P(B)\times P(H|B)^9 P(T|B)^1}{P(X)} = \frac{\frac{1}{3}\times(0.55)^9(0.45)^1}{0.00236}$$

$$P\left(\text{Coin is H55} \,\middle|\, \text{9 heads, 1 tail}\right) = 0.293$$

$$P(C|X) = \frac{P(C)\times P(X|C)}{P(X)} = \frac{P(C)\times P(H|C)^9 P(T|C)^1}{P(X)} = \frac{\frac{1}{3}\times(0.60)^9(0.40)^1}{0.00236}$$

$$P\left(\text{Coin is H60} \,\middle|\, \text{9 heads, 1 tail}\right) = 0.569$$

(b) Based on the given information, I construct this table w/ total population of 10,000 women.

	Test returns "positive"	Test returns "Negative"	1% pregnant
Pregnant	99	1	100
Not pregnant	990	8910	9900
	1089	8911	10,000

In this sample, 1089 women receive a positive test, out of which 99 are actually positive, so P(pregnant | positive test) = 99/1089

= 0.0909

Another approach:

Let Y = positive test, B = pregnant. ($\bar{B}$: NOT pregnant)

We know that $P(Y|B) = 0.99$, $P(Y|\bar{B}) = 0.1$, $P(B) = 0.01$, $P(\bar{B}) = 0.99$

$$P(B|Y) = \frac{P(Y|B)P(B)}{P(Y)} = \frac{P(Y|B)P(B)}{P(Y|B)P(B) + P(Y|\bar{B})P(\bar{B})} = \frac{0.99\times 0.01}{0.99(0.01) + 0.1(0.99)} = 0.0909$$

This result makes sense because this value represents the true positive rate. The false positive rate is very high, with 990 of the 1089 positive results being false, which makes sense since most of the female population is not pregnant (99%).

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

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

For random variable X & constant a:

$\mathbb{E}[aX] = a\mathbb{E}[X]$

$\mathbb{E}[a] = a$

(d) $\text{cov}(x) = \mathbb{E}\left((x-\mathbb{E}x)(x-\mathbb{E}x)^T\right)$

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

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

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

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

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

$= A\,\text{cov}(x)\,A^T$

3. Multivariate derivatives

Identities explaining each step are written in blue and are taken from "The Matrix Cookbook" by Peterson & Pederson.

(a) $\nabla_x x^T A y = \frac{\partial}{\partial x}(x^T A y)$ ↳ $\frac{\partial x^T b}{\partial x} = \frac{\partial b^T x}{\partial x}$

$= \frac{\partial}{\partial x}(Ay)^T x$

$= Ay$ ↳ $\frac{\partial b^T x}{\partial x} = b$

$$x^T A y = [x_1 \dots x_n] \begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \\ A_{n1} & & A_{nm} \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

$$= [x_1 \dots x_n] \begin{bmatrix} A_{11}y_1 + \dots + A_{1m}y_m \\ \vdots \\ A_{n1}y_1 + \dots + A_{nm}y_m \end{bmatrix}$$

$$= x_1(A_{11}y_1 + \dots A_{m1}y_1) + \dots + x_n(A_{11}y_1 + \dots A_{m1}y_1)$$

$$\nabla_x x^T A y = \begin{bmatrix} \frac{\partial x^T A y}{\partial x_1} \\ \vdots \\ \frac{\partial x^T A y}{\partial x_n} \end{bmatrix} = Ay$$

(b) $\nabla_y x^T A y = \frac{\partial}{\partial y} x^T A y$ ↳ $\frac{\partial b^T x}{\partial x} = b$

$= (x^T A)^T$

$= A^T x$

(c) $\nabla_A x^T A y = \frac{\partial}{\partial A} x^T A y$ ↳ $\frac{\partial a^T X b}{\partial X} = ab^T$

$= xy^T$

$$x^T A y = [x_1 \dots x_n] \begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \\ A_{n1} & & A_{nm} \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

$$\nabla_A x^T A y = \begin{bmatrix} \frac{\partial x^T A y}{\partial A_{11}} & \cdots & \frac{\partial x^T A y}{\partial A_{1m}} \\ \vdots & \ddots & \\ \frac{\partial x^T A y}{\partial A_{n1}} & & \frac{\partial x^T A y}{\partial A_{nm}} \end{bmatrix} = \begin{bmatrix} x_1 y_1 & \cdots & x_1 y_m \\ \vdots & \ddots & \\ x_n y_1 & & x_n y_m \end{bmatrix} = x y^T$$

(d) $f = x^T A x + b^T x$

$\nabla_x f = \frac{\partial f}{\partial x}$

$\frac{\partial x^T B x}{\partial x} = (B + B^T) x$ 和 $\frac{\partial b^T x}{\partial x} = b$

$= (A + A^T) x + b$

$$x^T A x = [x_1 \dots x_n] \begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \\ A_{n1} & & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$= x_1 (A_{11} x_1 + \dots + A_{1n} x_n) + \dots + x_n (A_{n1} x_1 + \dots + A_{nn} x_n)$$

$$\nabla_x x^T A x = \frac{\partial}{\partial x_k} \left(\sum_{i=1}^{n} a_{ik} x_i x_k + \sum_{j=1}^{n} a_{kj} x_k x_j \right)$$

$$= \sum_{i=1}^{n} a_{ik} x_i + \sum_{j=1}^{n} a_{kj} x_j$$

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

(e) $f = \mathrm{tr}(AB)$

$\nabla_A f = \frac{\partial}{\partial A} \mathrm{tr}(AB)$

$\frac{\partial}{\partial X} \mathrm{Tr}(XB) = B^T$

$= B^T$

$$AB = \underset{n\times m}{\begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \\ A_{n1} & & A_{nm} \end{bmatrix}} \underset{m\times n}{\begin{bmatrix} B_{11} & \cdots & B_{1n} \\ \vdots & \ddots & \\ B_{m1} & & B_{mn} \end{bmatrix}}$$

$$= \begin{bmatrix} (A_{11}B_{11} + \cdots + A_{1m}B_{m1}) & \cdots & (A_{11}B_{1n} + \cdots + A_{1m}B_{mn}) \\ \vdots & \ddots & \\ (A_{n1}B_{11} + \cdots + A_{nm}B_{m1}) & & (A_{n1}B_{1n} + \cdots A_{nm}B_{mn}) \end{bmatrix}$$

$$\nabla_A f = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1m}} \\ \vdots & \ddots & \\ \frac{\partial f}{\partial A_{n1}} & & \frac{\partial f}{\partial A_{nm}} \end{bmatrix} = \begin{bmatrix} B_{11} & \cdots & B_{1m} \\ \vdots & \ddots & \\ B_{n1} & & B_{nm} \end{bmatrix} = B^T$$

4. Deriving least-squares w/ matrix derivatives

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

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

$A^TB = B^TA$ b/c is scalar → dot product is commutative: $A \cdot B = B \cdot A$

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

vectorization as done in class

$$= \frac{1}{2}(Y - XW)^T(Y - XW)$$

$$= \frac{1}{2}\left[Y^TY - Y^TXW - W^TX^TY + W^TX^TXW\right]$$

scalar values → $c^T = c$, so $Y^TXW = (Y^TXW)^T = W^TX^TY$

$$= \frac{1}{2}(W^TX^TXW - 2Y^TXW + Y^TY)$$

$$\nabla_W \mathcal{L}(W) = \frac{1}{2}\nabla_W (W^TX^TXW - 2Y^TXW + Y^TY)$$

$\nabla_W \mathcal{L}(W) = \nabla_W \text{tr}(\mathcal{L}(W))$ b/c $\mathcal{L}(W)$ is a scalar

$$= \frac{1}{2}\nabla_W \text{tr}(W^TX^TXW - 2Y^TXW + Y^TY)$$

$$= \frac{1}{2}\nabla_W \left[\text{tr}(W^TX^TXW) - 2\,\text{tr}(Y^TXW) + \text{tr}(Y^TY)\right]$$

$\text{tr}(Y^TY)$: not function of W

$\frac{\partial \text{tr}(WAW^T)}{\partial W} = WA^T + WA$

$\frac{\partial}{\partial X}\text{tr}(XB) = B^T$

$\frac{\partial \text{tr}(WA)}{\partial W} = A^T$

$$= \frac{1}{2}\left((W^T(X^TX))^T + (W^TX^TX) - 2(Y^TX)^T\right)$$

$$= \frac{1}{2}(2X^TXW - 2X^TY)$$

$$= X^TXW - X^TY$$

Set $\nabla_W L(W) = 0$ and solve for W, the optimal W.

$$0 = X^T X W - X^T Y$$

$$X^T X W = X^T Y$$

$$(X^T X)^{-1} X^T X W = (X^T X)^{-1} X^T Y$$

$$W = (X^T X)^{-1} X^T Y$$

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 2021, Prof. J.C. Kao, TAs: N. Evirgen, A. Ghosh, S. Mathur, T. Monsoor, G. Zhao

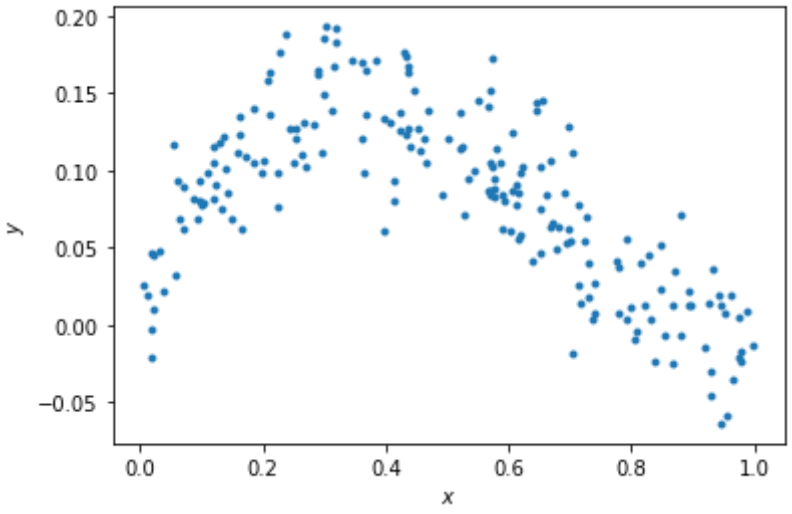
In [65]:
```
import numpy as np
import matplotlib.pyplot as plt

#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 [66]:
```
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[66]: 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) `np.random.uniform` is what draws `num_train` samples from a uniform distribution over the interval from 0 (the `low` parameter) to 1 (the `high` parameter).

(2) The noise is added by `np.random.normal` which draws random samples from a normal Gaussian distribution, with a mean of 0.3 (the `loc` parameter) and standard deviation of 0.03 (the `scale` parameter).

Fitting data to the model (5 points)

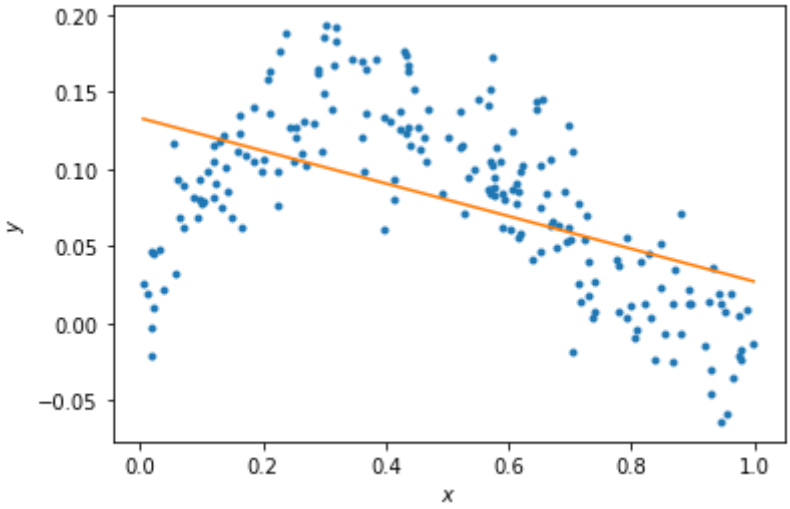
Here, we'll do linear regression to fit the parameters of a model $y = ax + b$.

In [67]:
```
# 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

# W = (X_T * X)^(-1) * (X_T) * Y
theta = np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y))
print(theta)

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

```
[-0.10599633  0.13315817]
```

In [68]:
```
# 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[68]:
```
[<matplotlib.lines.Line2D at 0x11a311410>]
```



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 whose shape resembles more that of a quadratic equation.

(2) To better fit the data, we can use a higher degree polynomial model, like a quadratic, to increase expressiveness.

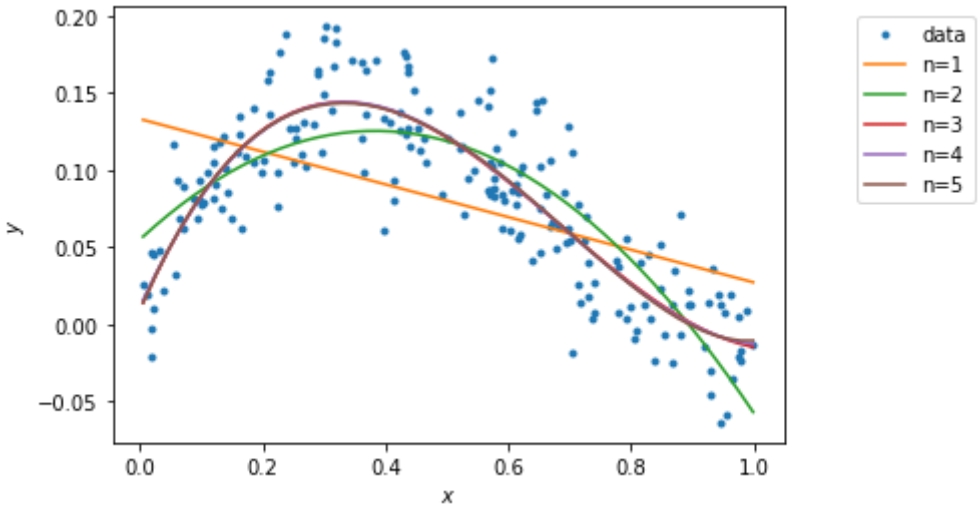
Fitting data to the model (10 points)

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 [69]:

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

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

for i in range(1, N):
    xhat_i = np.vstack((x**(i+1), xhats[-1]))
    xhats.append(xhat_i)

    theta_i = np.linalg.inv(xhat_i.dot(xhat_i.T)).dot(xhat_i.dot(y))
    thetas.append(theta_i)

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

In [70]:
```
# 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 [71]:
```
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

# L = (1/2) * summation [(y - theta * x_hat)^2]
for i in range(N):
    error = (1/2) * np.sum(np.square(y - (thetas[i]).dot(xhats[i])))
    training_errors.append(error)

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

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

```
Training errors are:
 [0.2379961088362701, 0.1092492220926853, 0.08169603801105374, 0.08165353
735296976, 0.08161479195525295]
```

QUESTIONS

(1) What polynomial has the best training error?

(2) Why is this expected?

ANSWERS

(1) The fifth-order polynomial (the highest order of these 5) has the lowest training error.

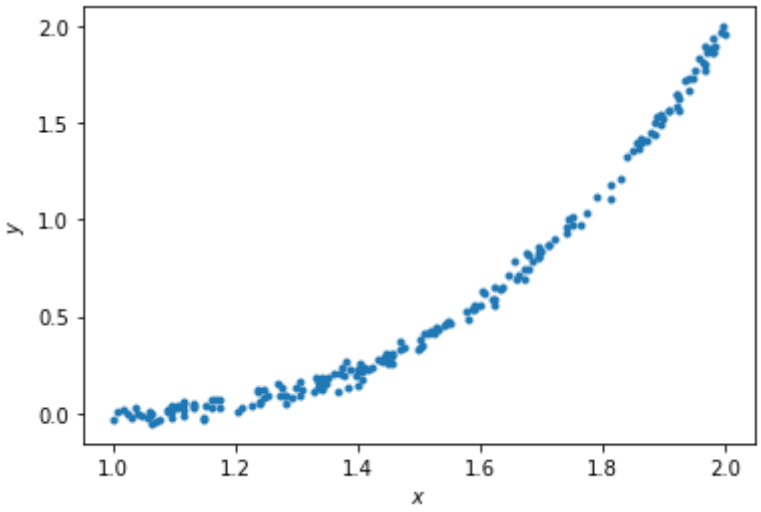
(2) As covered in class, a higher degree polynomial always fits the provided data no worse than a lower degree polynomial, because a higher degree polynomial can turn into a lower degree polynomial by getting rid of higher order terms by making their coefficient 0. Higher complexity better fits the training data since there are more functions to choose from.

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.

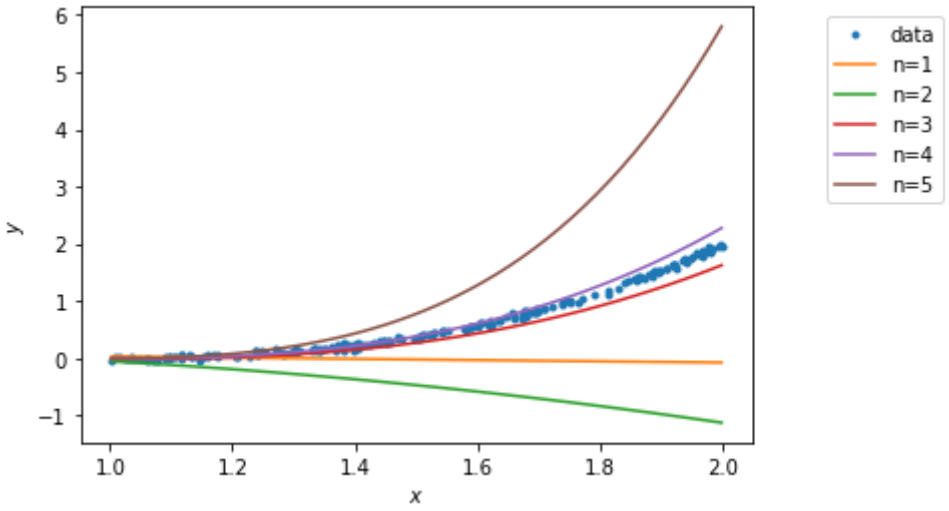
In [72]:
```
x = np.random.uniform(low=1, high=2, size=(num_train,))
y = x - 2*x**2 + x**3 + np.random.normal(loc=0, scale=0.03, size=(num_train
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

Out[72]: Text(0, 0.5, 'y')



In [73]:
```
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 [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,:], 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 [75]:
```
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 or
for i in range(N):
    error = (1/2) * np.sum(np.square(y - (thetas[i]).dot(xhats[i])))
    testing_errors.append(error)

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

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

```
Testing errors are:
 [80.86165184550586, 213.19192445057908, 3.1256971084083736, 1.1870765211
496224, 214.91021747012798]
```

QUESTIONS

(1) What polynomial has the best testing error?

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

ANSWERS

(1) Fourth-order polynomial has the lowest testing error of 1.1870765211496224.

(2) The fifth-order polynomial (the higest order model) does not generalize well because it overfits the data. This level of complexity fits the provided data (training data) too well, picking up the training data's inaccuracies in noise or random fluctuations, picking up patterns that decrease the model's generalizability. The model does not capture the underlying trend and cannot make good predictions on data it has not yet seen. In this case, the training error is low and testing error is high.

In [ ]: