2.1

(a)

$$\frac{d}{d\omega}(X\omega - Y)^2 = 2(X\omega - Y)X = 0 \rightarrow \omega = (X^T X)^{-1} X^T Y$$

(b)

Q1: what does that tell us about the data?

At least d observations to make sure that the inverse exists

The row and column vectors are linear-independent.

Q2: What difference does the presence or absence of the origin in this subspace make?

If the subspace spanned by all data contains the origin, we need at least d+ 1 observations.

Q3: To what extent are these limitations enough to guarantee invertibility?

- At least d observations to make sure that the inverse exists
- If the subspace spanned by all data contains the origin, we need at least d+ 1 observations.
- The row and column vectors are linear-independent.

2.2

(a)

done

A solution can be found by considering the gradient and equating that to 0. Additionally, one needs to realize (though this may not be that trivial) that the function we are minimizing is convex. Maybe we would need a more formal argument for the latter as well...

(b)

If the data spans the space, we know that $(Z^TZ)^{-1}$ exists and vice versa.

(c)

```
X = np.array([[2,1], [2,3]])
XX = np.dot(X.T, X)
XX_inv = np.linalg.inv(XX)

```

```python
print("X:", "\n", str(X))
print("XX:", "\n", str(XX))
print("XX_inv:", "\n", str(XX_inv))

bias = np.ones((2,1))
Z = np.hstack((XX, bias))
ZZ = np.dot(Z.T, Z)

print("Z:", "\n", str(Z))
print("ZZ:", "\n", str(ZZ))
print("ZZ_inv", "\n", "singular martrix")
```

2.3

(a)

All straight lines but the vertical one that pass through the point (π, e) will minimize the objective function. E.g,

$$y = \frac{\pi e}{\pi^2 + 1} x + \frac{e}{\pi^2 + 1}$$

(b)

$$\{y | y = \frac{e - b}{\pi} x + b, b \in \Re\}$$

(c)

```python
X = np.array([[1,2], [-2,1]])
bias = np.ones((2,1))
X = np.hstack((X,bias))
Y = np.array([[-1], [1]])

XX_inv = np.dot(X.T, X)
XY = np.dot(X.T, Y)

w = np.dot(XX_inv, XY)

print("w:", '\n', str(w))
```

(d)

```python
sqres = np.sum(np.square(np.dot(X, w) - Y))
print(sqres)
```

(e)

$$w = (w_1, w_2), \{w_1, w_2 | w_1 + w_2 = \pi\}$$
$$Y = Xw, X \in \Re^{1\times 2}; w \in \Re^{2\times 1}$$

(f)

Of course, there is no unique answer to this question. Nevertheless, an argument one can make is that one does not want the solution to be unnecessarily tilted, increasing, or decreasing.

Interestingly, following this argument, we prefer to pick the solution for which the norm of the (non-intercept) weights is smallest.

2.4

(a)

```python
X = np.array([-2, -1, 0, 3]).T
Y = np.array([1, 1, 2, 3]).T
XX = np.dot(X.T, X)
XX_inv = 1./XX
XY = np.dot(X.T, Y)

w = XX_inv * XY
print("w:", '\n', str(w))

sqres = np.sum(np.square(np.dot(X, w) - Y))
print(sqres)
```

(b)

```python
X = np.array([-2, -1, 0, 3]).reshape(4,1)
bias = np.ones((4,1))
X = np.hstack((X,bias))

XX = np.dot(X.T, X)
XX_inv = np.linalg.inv(XX)
XY = np.dot(X.T, Y)

```

```python
w = np.dot(XX_inv, XY)
print("w:", '\n', str(w))

sqres = np.sum(np.square(np.dot(X, w) - Y))
print(sqres)
```

(c)

Degree = 3 with intercept or degree = 4 without intercept

Note that this last part is a bit tricky: the answer assumes that we do not need an explicit intercept anymore as it is automatically modelled by the 0th degree monomial anyway. If, however, we mean that such 0th degree are not at all allowed, we need an additional degree of freedom to fit four points, which means we need polynomials of degree 4.

2.5

done

2.6

Encode months and days into int number. $X = [x_1, x_2]$, where x_1 represents the number of moths and x_2 represents the number of days. Normalize the input X if necessary.

Encode weathers of a year into $\{-1,1\}^n$, where -1 represents rainy day and 1 represents not-rainy day. This problem becomes two-categories classification.

My quick-and-dirty solution would be two-fold. First, the encoding into months and days is not nice and I would like a more linear kind of timescale. So, I propose to first transform that into a 1D representation t by something like $t = 30(x_1 - 1) + (x_2 - 1)$*. This makes t linear over one year with a minimum of 0 and a maximum of 360. Now, I would expect some periodicity in the signal. So, I want to move away from the linear t. I would expect one max and one min temperature in the year, so a first order approximation with a (co)sinus should be a good first attempt. Based on this, I would use as final 2D input: (sin(t), cos(t)).*

2.7

Done

(d)

The issue is that linear polynomial does not take into account any **cross-terms**

2.8

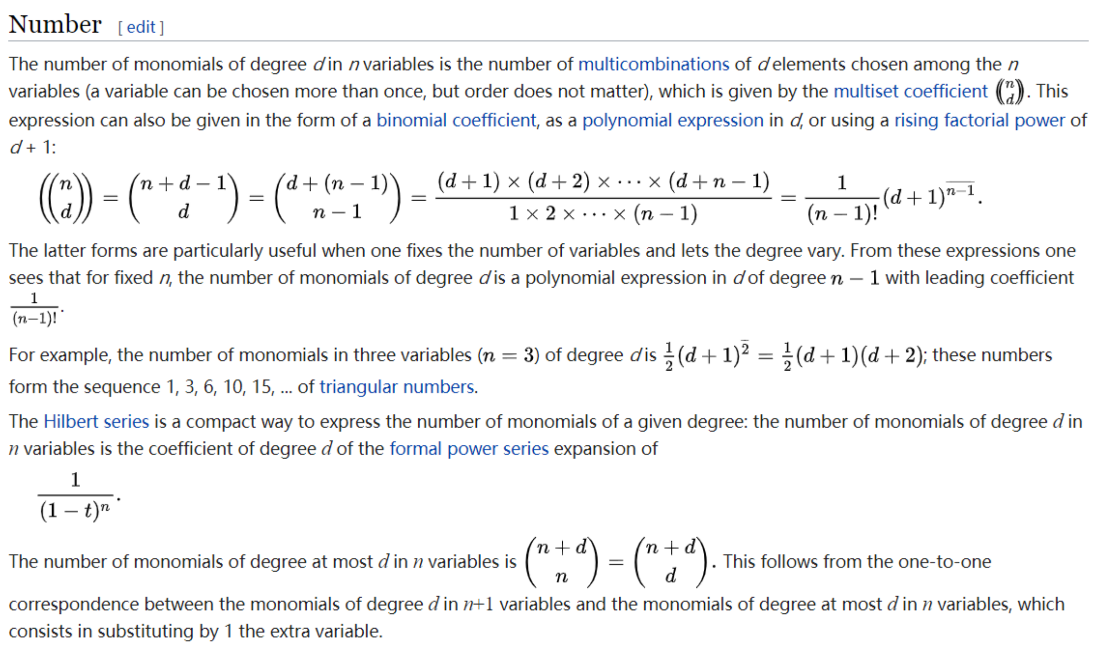
(a)

d = 3 and m = 3

$$\begin{aligned}
x_1^3 &\longleftrightarrow \star\star\star\,|\,| \\
x_1^2x_2 &\longleftrightarrow \star\star\,|\star| \\
x_1^2x_3 &\longleftrightarrow \star\star\,|\,|\star \\
x_1x_2^2 &\longleftrightarrow \star\,|\star\star| \\
x_1x_2x_3 &\longleftrightarrow \star\,|\star\,|\star \\
x_1x_3^2 &\longleftrightarrow \star\,|\,|\star\star \\
x_2^3 &\longleftrightarrow |\star\star\star| \\
x_2^2x_3 &\longleftrightarrow |\star\star|\star \\
x_2x_3^2 &\longleftrightarrow |\star\,|\star\star \\
x_3^3 &\longleftrightarrow |\,|\star\star\star
\end{aligned}$$

$$\frac{C_4^1 C_5^1}{A_2^2} = \frac{4\times5}{2} = 10$$

$$\frac{C_{m+1}^1 \times C_{m+2}^1 \times \ldots \times C_{m+d-1}^1}{A_{d-1}^{d-1}} = \frac{\frac{(m+d-1)!}{m!}}{(d-1)!} = \frac{(m+d-1)!}{(d-1)!\,m!}$$



Number [edit]

The number of monomials of degree d in n variables is the number of multicombinations of d elements chosen among the n variables (a variable can be chosen more than once, but order does not matter), which is given by the multiset coefficient $\left(\!\binom{n}{d}\!\right)$. This expression can also be given in the form of a binomial coefficient, as a polynomial expression in d, or using a rising factorial power of $d + 1$:

$$\left(\!\!\binom{n}{d}\!\!\right) = \binom{n+d-1}{d} = \binom{d+(n-1)}{n-1} = \frac{(d+1)\times(d+2)\times\cdots\times(d+n-1)}{1\times2\times\cdots\times(n-1)} = \frac{1}{(n-1)!}(d+1)^{\overline{n-1}}.$$

The latter forms are particularly useful when one fixes the number of variables and lets the degree vary. From these expressions one sees that for fixed n, the number of monomials of degree d is a polynomial expression in d of degree $n-1$ with leading coefficient $\frac{1}{(n-1)!}$.

For example, the number of monomials in three variables ($n = 3$) of degree d is $\frac{1}{2}(d+1)^{\overline{2}} = \frac{1}{2}(d+1)(d+2)$; these numbers form the sequence 1, 3, 6, 10, 15, ... of triangular numbers.

The Hilbert series is a compact way to express the number of monomials of a given degree: the number of monomials of degree d in n variables is the coefficient of degree d of the formal power series expansion of

$$\frac{1}{(1-t)^n}.$$

The number of monomials of degree at most d in n variables is $\binom{n+d}{n} = \binom{n+d}{d}$. This follows from the one-to-one correspondence between the monomials of degree d in $n+1$ variables and the monomials of degree at most d in n variables, which consists in substituting by 1 the extra variable.

(b)

If d > 1 then yes. The order of polynomial growth for the number of features equals the dimensionality d.

(c)

In bioinformatics one is dealing with gene expression data in which easily d 10, 000 and so m = 3 becomes unmanageable already. Worse even, are image classification problems in which in which one cannot afford to subsample the image. Nowadays one easily has d > 1, 000, 000 and so m = 2 already becomes infeasible.

2.9

(a)

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

$$T = \frac{1}{n}\sum_{i}^{n}(x_i - \overline{x})(x_i - \overline{x})^T$$

$$m_+ = \frac{1}{2n}\sum_{i}^{n} x_i \frac{(y_i + 1)}{2}$$

$$m_- = \frac{1}{2n}\sum_{i}^{n} x_i \frac{(y_i - 1)}{-2}$$

$$2T^{-1}(m_+ - m_-) = 2n[(X-\mu)^T(X-\mu)]^{-1} \times \frac{1}{2n} X^T Y = [(X-\mu)^T(X-\mu)]^{-1} X^T Y$$

If the x is standard normal distribution $\rightarrow\ \mu = 0$

$$2T^{-1}(m_+ - m_-) = (X^T X)^{-1} X^T Y$$

https://stats.stackexchange.com/questions/287144/variance-of-a-sample-covariance-for-normal-variables

(b)

```
#[x1, x2, x3] -> [x1-x2, 2*x2, 2*x1+x3]
linearTrans = np.array([[1,-1,0], [0,2,0], [2,0,1]]).T
testX = np.array([3,6,9]).reshape(1,3)
transX = np.dot(testX, linearTrans)
print("transX: ", transX)

transFeature = np.dot(feature, linearTrans)
transY_hat = forward(transFeature, w)
acc = np.sum(y_hat[:,0] == label)
print(acc)
plt.scatter(transFeature[:,0], transFeature[:,1], c=label)
transData = pr.prdataset(transFeature, targets=label)
decisionBoundary(transData, w)
plt.show()
```

(c)

Done

2.10

(a)

$$y|x \sim N(xw + w_0, \sigma), and\ x \sim N(v, \tau)$$

$$p(x, y) = p(y|x)p(x) = \frac{1}{\sigma\tau 2\pi} e^{-[\frac{\left(y-(xw+w_0)\right)^2}{2\sigma^2} + \frac{(x-v)^2}{2\tau^2}]}$$

(b)

After simplification,

$$\min_{w,w_0} \left(y - (xw + w_0)\right)^2$$

$$w = (X^T X)^{-1} X^T Y, w_0\ included$$

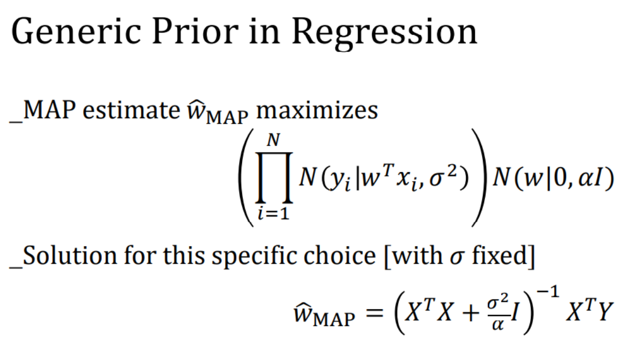
(c)

When do the partial derivation, θ disappears.

(d)

$$\theta = \frac{1}{n}\sqrt{\sum [y_i - (wx_i + w_0)][y_i - (wx_i + w_0)]^T}$$

(e)



Generic Prior in Regression

_MAP estimate $\hat{w}_{\text{MAP}}$ maximizes

$$\left(\prod_{i=1}^{N} N(y_i|w^T x_i, \sigma^2)\right) N(w|0, \alpha I)$$

_Solution for this specific choice [with σ fixed]

$$\hat{w}_{\text{MAP}} = \left(X^T X + \frac{\sigma^2}{\alpha} I\right)^{-1} X^T Y$$

2.11

(a)

Done

(b)

Done

(c)

$a \in (-\infty, 1) \cap [5,6)$ Error = 0.4

(d)

$a \in [2, \pi)$ Error = 0.2

2.12

Not done

2.13

Not done

2.14

X = [[0,0,1], [0,1,1], [1,0,1], [1,1,1]]

Y = [-1, -1, 1, 1]

$w^{(0)} = [0, 0, 0]$

Round 1:

$a^{(1)} = w^{(0)}X = [0, 0, 0, 0]$

$\hat{y}^{(1)} = [1, 1, 1, 1]$

$w^{(1)} = w^{(0)} + 1 \times (-[0,0,1] - [0,1,1]) = [0, -1, -2]$

Round 2:

$a^{(2)} = w^{(1)}X = [-2, -3, -2, -3]$

$\hat{y}^{(2)} = [-1, -1, -1, -1]$

$w^{(2)} = w^{(1)} + 1 \times ([1,0,1] + [1,1,1]) = [2, 0, 0]$

Round 3:

$a^{(3)} = w^{(2)}X = [0, 0, 2, 2]$

$\hat{y}^{(3)} = [1, 1, 1, 1]$

$w^{(3)} = w^{(2)} + 1 \times (-[0,0,1] - [0,1,1]) = [2, -1, -2]$

Round 4:

$a^{(4)} = w^{(3)}X = [-2, -3, 0, -1]$

$\hat{y}^{(4)} = [-1, -1, 1, -1]$

$w^{(4)} = w^{(3)} + 1 \times ([1,1,1]) = [3, 0, -1]$

{test: Round 5:

$a^{(5)} = w^{(4)}X = [-1, -1, 2, 2]$

$\hat{y}^{(4)} = [-1, -1, 1, 1]$

$done\}$

w = [3, 0, -1]

2.15

(a)

Done

(b)

Done

(c)

Done

(d)

If the samples are not linearly separable, the algorithm should be adjusted to tolerate the misclassification cases. We need to set a threshold to stop the optimizing process.

2.16

Done