1)

a) Suppose $\alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 = 0$.

Then, we have the following:

(1) $\alpha_1 + \alpha_2 = 0$

(2) $2\alpha_1 + \alpha_2 + \alpha_3 = 0$

(3) $\alpha_3 + \alpha_4 = 0$

(4) $-\alpha_1 + \alpha_4 = 0$

(3), (4): $\alpha_3 = -\alpha_1$

(2): $2\alpha_1 + \alpha_2 + (-\alpha_1) = 0$

$\Rightarrow \alpha_1 + \alpha_2 = 0$, same as (1)

Let $\alpha_1 = 1$, $\alpha_2 = -1$. Then $\alpha_3 = -1$, $\alpha_4 = 1$.

$$\alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 = x_1 - x_2 - x_3 + x_4 = \begin{pmatrix} 1 \\ 2 \\ 0 \\ -1 \\ 0 \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Therefore, there exists $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ other than $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 0$ such that $\alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 = 0$, so the columns of X are not linearly independent. So, the largest set of linearly independent vectors is at most 3. So, if we find a set of 3 columns that are linearly independent, the largest set is 3.

Take x_1, x_2, x_3 and suppose $\alpha x_1 + \alpha x_2 + \alpha x_3 = 0$.

Then, we know:

(1) $\alpha_1 + \alpha_2 = 0$

(2) $2\alpha_1 + \alpha_2 + \alpha_3 = 0$

(3) $\alpha_3 = 0$

(4) $-\alpha_1 = 0 \Rightarrow \alpha_1 = 0$

(1): Since $\alpha_1 = 0$, $\alpha_2 = 0$.

Thus, since $\alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 = 0$ requires $\alpha_1 = \alpha_2 = \alpha_3 = 0$, the first 3 columns are linearly independent.

Therefore, a largest set of linearly independent columns of X is $\boxed{\{x_1, x_2, x_3\}}$.

b) Take x_2, x_3, x_4 and suppose $\alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 = 0$.

(1) $\alpha_2 = 0$

(2) $\alpha_2 + \alpha_3 = 0$

(3) $\alpha_3 + \alpha_4 = 0$

(4) $\alpha_4 = 0$

(2),(3): $\alpha_2 = \alpha_4 = 0 \Rightarrow \alpha_3 = 0$

So, $\alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 = 0$ only if $\alpha_2 = \alpha_3 = \alpha_4 = 0$.

Thus, $\{x_2, x_3, x_4\}$ are linearly independent so the answer in part (a) is $\boxed{\text{not unique}}$.

c) $\boxed{\text{rank}(X) = 3}$ because the largest set of linearly independent columns of X has 3 vectors.

d) Claim: $\text{rank}(X^T X) = \text{rank}(X)$

Proof:

1) Let $v \in \text{Null}(X)$, or $Xv = 0$.

Then, $(X^T X)v = X^T(Xv) = X^T 0 = 0$, so $v \in \text{Null}(X^T X)$.

Since $v \in \text{Null}(X) \Rightarrow v \in \text{Null}(X^T X)$, $\text{Null}(X) \subset \text{Null}(X^T X)$.

(2) Let $v \in \text{Null}(X^T X)$, or $(X^T X)v = 0$.

Then, $v^T (X^T X) v = v^T 0 = 0$.

Assume for contradiction that $v \notin \text{Null}(X)$.

Then, $Xv = w$ where $w \neq 0$:

$$(Xv)^T = w^T$$
$$v^T X^T = w^T$$
$$v^T X^T (Xv) = w^T (Xv)$$
$$0 = w^T (Xv) = w^T w$$

Since $w^T w = w_1^2 + \ldots + w_k = 0$, then $w_1 = \ldots = w_k = 0$, or $w = 0$.

This means $Xv = w = 0$, so $v \in \text{Null}(X)$, which is a contradiction.

Since $v \in \text{Null}(X^T X) \Rightarrow v \in \text{Null}(X)$, $\text{Null}(X^T X) \subset \text{Null}(X)$.

Combining these, $\text{Null}(X^T X) = \text{Null}(X)$.

By rank-nullity, $\dim(\text{Null}(X^T X)) + \text{Rank}(X^T X) = k$ and $\dim(\text{Null}(X)) + \text{Rank}(X) = k$.

$\text{Null}(X^T X) = \text{Null}(X) \Rightarrow \dim(\text{Null}(X^T X)) = \dim(\text{Null}(X))$

Thus, $\text{rank}(X^T X) = \text{rank}(X)$.

Therefore, $\boxed{\text{rank}(X^T X) = 3}$.

2)

a) $X = \begin{pmatrix} 0.63 & -0.63 \\ -0.63 & 0.63 \\ 0.63 & 0.63 \\ -0.63 & -0.63 \end{pmatrix}$ yes, $\boxed{\text{linearly independent}}$ because columns X_1 and X_2 are not scalar multiples of each other.

b) $X = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & -1 & 0 \end{pmatrix}$

The columns of X are linearly independent if $w=0$ is the only vector that satisfies $Xw=0$.

Let $w = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$. Then, $Xw=0$: $\begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 0 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$

This gives:

$w_1 - w_2 + w_3 = 0 \quad (1)$

$w_1 + w_2 - w_3 = 0 \quad (2)$

$w_1 - w_2 = 0 \quad (3)$

$(3) \Rightarrow (1)$: $(w_1 - w_2) + w_3 = 0$

$0 + w_3 = 0 \Rightarrow w_3 = 0$

(1): $w_1 + w_2 - w_3 = w_1 + w_2 = 0$

(3): $w_1 - w_2 = 0$

$(1), (3) \Rightarrow w_1 = w_2 = 0$

Thus, the only w that satisfies $Xw=0$ is $w=0$, so the columns of X are $\boxed{\text{linearly independent}}$.

c) $X = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 5 & 1 \\ 8 & 13 & 3 \end{pmatrix}$ Let X_j denote the jth column of X

$X_3 = 2X_1 - X_2$, so X_3 is a linear combination of X_1 and X_2, so the columns of X are $\boxed{\text{not linearly independent}}$

d) $X = \begin{pmatrix} 2 & 4 \\ -8 & 12 \\ 4 & 8 \end{pmatrix}$ columns X_1 and X_2 are not scalar multiples of each other, so the columns of X are linearly independent, making $\boxed{\text{rank}(X) = 2}$

3)

a) $f(w) = w^T(3x) + x^Tw$

x^Tw is the inner product of x and w, so $x^Tw = x_1w_1 + \ldots + x_dw_d = w^Tx$.

$$f(w) = 3w^Tx + x^Tw = 3x^Tw + x^Tw = 4x^Tw$$

$\nabla_w x^Tw = x$, so $\nabla f w = \boxed{4x}$.

b) $f(w) = (w-3x)^T(2w-x)$

$$= w^T(2w) - w^Tx - 3x^T(2w) + 3x^Tx$$

$$= 2w^Tw - w^Tx - 6x^Tw + 3x^Tx$$

$w^Tx = x^Tw \Rightarrow f(w) = 2w^Tw - 7w^Tx + 3x^Tx$

$w^Tw = w_1^2 + \ldots + w_d^2 \Rightarrow \nabla_w w^Tw = 2w$

$\nabla_w w^Tx = x$

$\nabla_w x^Tx = 0$

$\nabla_w f(w) = 2(2w) - 7x + 0 \Rightarrow \nabla_w f(w) = \boxed{4w - 7x}$

c) $f(w) = x^T\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}w$ Let $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$. Then $f(w) = x^TAw$.

x^TA is a 1×2 matrix, so $\nabla_w f(w)$ is a 2×1 matrix where the first element is the first element of x^TA and the second element is the second element of x^TA.

Thus, $\nabla_w f(w) = (x^TA)^T = A^Tx$.

$$\nabla_w f(w) = \boxed{\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} x}$$

d) $f(w) = w^T \begin{pmatrix} 1 & 3 \\ -3 & 1 \end{pmatrix} w$ Let $A = \begin{pmatrix} 1 & 3 \\ -3 & 1 \end{pmatrix}$. Then $f(w) = w^T A w$.

From class, $\nabla_w f(w) = Aw + A^T w$ because $A \neq A^T$.

$$\nabla_w f(w) = \begin{pmatrix} 1 & 3 \\ -3 & 1 \end{pmatrix} w + \begin{pmatrix} 1 & -3 \\ 3 & 1 \end{pmatrix} w = \boxed{\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} w = 2w}$$

e) $f(w) = w^T \begin{pmatrix} 1 & 3 \\ 3 & 9 \end{pmatrix} w$ Let $A = \begin{pmatrix} 1 & 3 \\ 3 & 9 \end{pmatrix}$. Then $f(w) = w^T A w$.

From class, $\nabla_w f(w) = 2Aw$ because $A = A^T$.

$$\nabla_w f(w) = 2 \begin{pmatrix} 1 & 3 \\ 3 & 9 \end{pmatrix} w = \boxed{\begin{pmatrix} 2 & 6 \\ 6 & 18 \end{pmatrix} w}$$

In [1]:
```python
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt
import math
```

4a

In [2]:
```python
# Load in training data and labels
# File available on Canvas

face_data_dict = np.load("face_emotion_data.npz")
face_features = face_data_dict["X"]
face_labels = face_data_dict["y"]
n, p = face_features.shape

# Solve the least - squares solution. weights is the array of weight coefficients
weights = la.inv(face_features.T @ face_features) @ face_features.T @ face_labels

# TODO : find weights
print(f"Part 4a. Found weights:\n{weights}")
```

```
Part 4a. Found weights:
[[ 0.94366942]
 [ 0.21373778]
 [ 0.26641775]
 [-0.39221373]
 [-0.00538552]
 [-0.01764687]
 [-0.16632809]
 [-0.0822838 ]
 [-0.16644364]]
```

4b

Suppose the feature points for the new face are given as a vector $x_i = [x_{i1}, \ldots, x_{i9}]$. We can compute the prediction for the new face by computing $\hat{y}_i = x_i^T \cdot w$ where w is the weights vector found earlier. More specifically, we calculate the inner product of x_i and w by finding $x_{i1} \cdot w_1 + \ldots + x_{i9} \cdot w_9$. Then, we compute $sign(\hat{y}_i)$ to get the prediction for the new face, smiling if positive and not smiling if negative.

4c

In [3]:
```python
def lstsq_cv_err(
    features: np.ndarray, labels: np.ndarray, subset_count: int = 8
) -> float:
    """Estimate the error of a least-squares classifier
        using cross-validation . Use subset_count different
        train / test splits with each subset acting as the
        holdout set once.

    Parameters:
        features (np.ndarray): dataset features as a 2D
            array with shape (sample_count, feature_count)
        labels (np.ndarray): dataset class labels (+1/-1)
            as a 1D array with length (sample_count)
        subset_count (int): number of subsets to divide the
            dataset into
            Note: assumes that subset_count divides the
                dataset evenly

    Returns:
        cls_err (float): estimated classification error
            rate of least-squares method
    """

    sample_count, feature_count = features.shape
    subset_size = sample_count // subset_count

    # Reshape arrays for easier subset-level manipulation
    features = features.reshape(subset_count, subset_size, feature_count)
    labels = labels.reshape(subset_count, subset_size)

    subset_idcs = np.arange(subset_count)
    train_set_size = (subset_count - 1) * subset_size
    subset_err_counts = np.zeros(subset_count)

    for i in range(subset_count):
        # TODO: select relevant dataset,
        # fit and evaluate a linear model,
```

```
        # then store errors in subset_err_counts[i]
        current_features = np.zeros((0, feature_count))
        current_labels = np.zeros((0,))
        for j in range(subset_count):
            if j != i:
                current_features = np.concatenate((current_features, features[j]))
                current_labels = np.concatenate((current_labels, labels[j]))
        hold_out = features[i]
        current_weights = (
            la.inv(current_features.T @ current_features)
            @ current_features.T
            @ current_labels
        )
        predictions = hold_out @ current_weights

        for k in range(subset_size):
            if np.sign(predictions[k]) != np.sign(labels[i][k]):
                subset_err_counts[i] += 1

    # Average over the entire dataset to find the classification error
    cls_err = np.sum(subset_err_counts) / (subset_count * subset_size)
    return cls_err

# Run on the dataset with all features included
full_feat_cv_err = lstsq_cv_err(face_features, face_labels)
print(full_feat_cv_err)
print(f"Error estimate: {full_feat_cv_err*100:.3f}%")
```

```
0.046875
Error estimate: 4.688%
```

4d

We find the weights of all 9 features in the least squares estimation. We can let our heuristic be that features with lower weight magnitudes (absolute value of weight) are less important and therefore should be removed first. There are some limitations to this method because a feature with a smaller weight might not necessarily be less important, more that it is scaled appropriately to match the magnitudes of the values for that feature. For example, if feature 1 has small numbers, it could still have high feature importance even with a low weight since the low weight is to account for small inputs, not necessarily for lower importance. Nonetheless, we implement this heuristic since from a glance it appears that most of the data across features have similar sized values.

4e

In [4]:

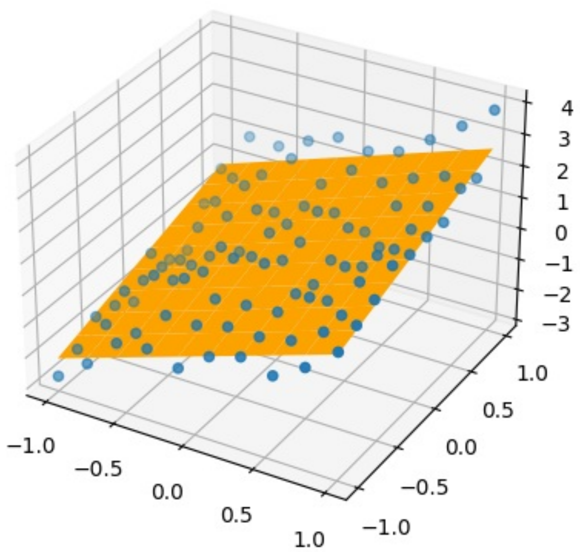
```
features_by_weights = [[abs(weights[i, 0]), i] for i in range(p)]
features_by_weights.sort(reverse=True)

current_features = [x[1] for x in features_by_weights]
current_error = lstsq_cv_err(face_features, face_labels)

for i in range(p):
    print('features:', current_features)
    print('error:', current_error * 100, '\n')
    current_features.pop()
    current_error = lstsq_cv_err(face_features[:, current_features], face_labels)
```

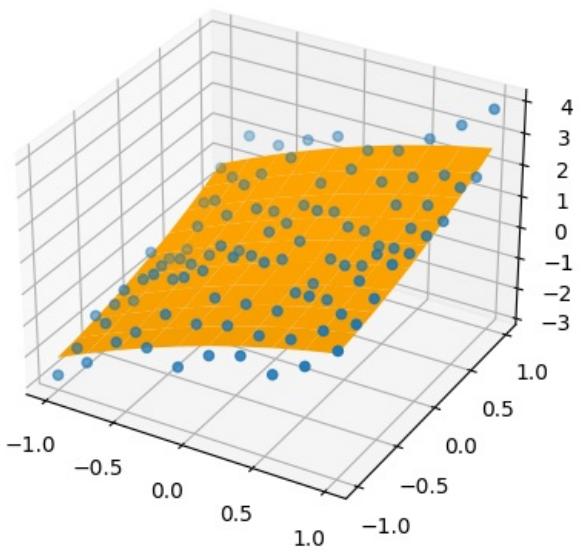
```
features: [0, 3, 2, 1, 8, 6, 7, 5, 4]
error: 4.6875

features: [0, 3, 2, 1, 8, 6, 7, 5]
error: 4.6875

features: [0, 3, 2, 1, 8, 6, 7]
error: 4.6875

features: [0, 3, 2, 1, 8, 6]
error: 4.6875

features: [0, 3, 2, 1, 8]
error: 5.46875

features: [0, 3, 2, 1]
error: 7.03125

features: [0, 3, 2]
error: 7.8125

features: [0, 3]
error: 8.59375

features: [0]
error: 7.03125
```

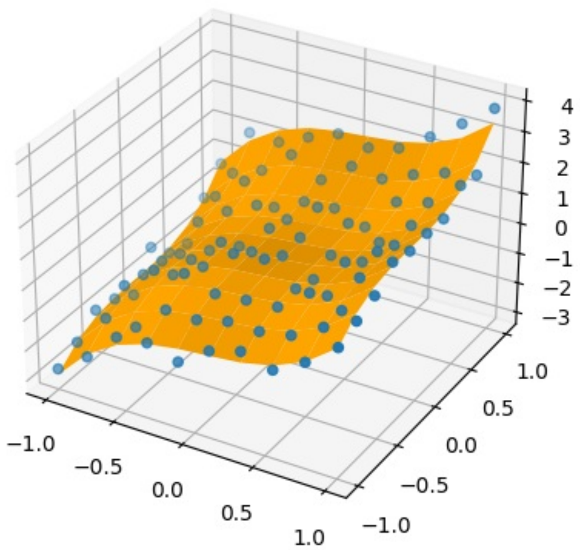
Based on the results above, it seems that we should stop removing features once the remaining features are 0, 3, 2, 1, and 8, which gives an error of 5.46875%. Otherwise, if we continue removing features (in this case, the next feature to be removed is feature 8), then our CV accuracy goes above 6%, which we do not want.

In [5]:

```
import numpy as np

# File available on Canvas
data = np.load("polydata_2D.npz")
x1 = np.ravel(data["x1"])
x2 = np.ravel(data["x2"])
y = data["y"]

N = x1.size
p = np.zeros((3, N))

for d in [1, 2, 3]:
    # Generate the X matrix for this d
    # Find the least-squares weight matrix w_d
    # Evaluate the best-fit polynomial at each point (x1, x2)
    # and store the result in the corresponding column of p
    X = np.zeros((N, 2 * d + 1))
    for i in range(N):
        X[i, 0] = 1
        for j in range(1, d + 1):
            X[i, j] = math.pow(x1[i], j)
            X[i, j + d] = math.pow(x2[i], j)
    w_d = la.inv(X.T @ X) @ X.T @ y
    p[d - 1] = X @ w_d

# Plot the degree 1 surface
Z1 = p[0, :].reshape(data["x1"].shape)
ax = plt.axes(projection="3d")
ax.scatter(data["x1"], data["x2"], y)
ax.plot_surface(data["x1"], data["x2"], Z1, color="orange")
plt.show()

# Plot the degree 2 surface
Z2 = p[1, :].reshape(data["x1"].shape)
ax = plt.axes(projection="3d")
ax.scatter(data["x1"], data["x2"], y)
ax.plot_surface(data["x1"], data["x2"], Z2, color="orange")
plt.show()

# Plot the degree 3 surface
Z3 = p[2, :].reshape(data["x1"].shape)
ax = plt.axes(projection="3d")
ax.scatter(data["x1"], data["x2"], y)
ax.plot_surface(data["x1"], data["x2"], Z3, color="orange")
plt.show()
```







Processing math: 100%