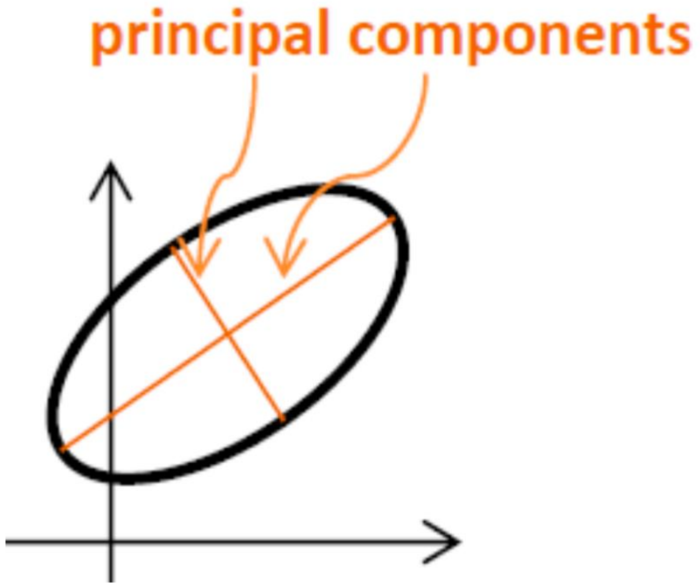
Principal component analysis

What happens under the hood

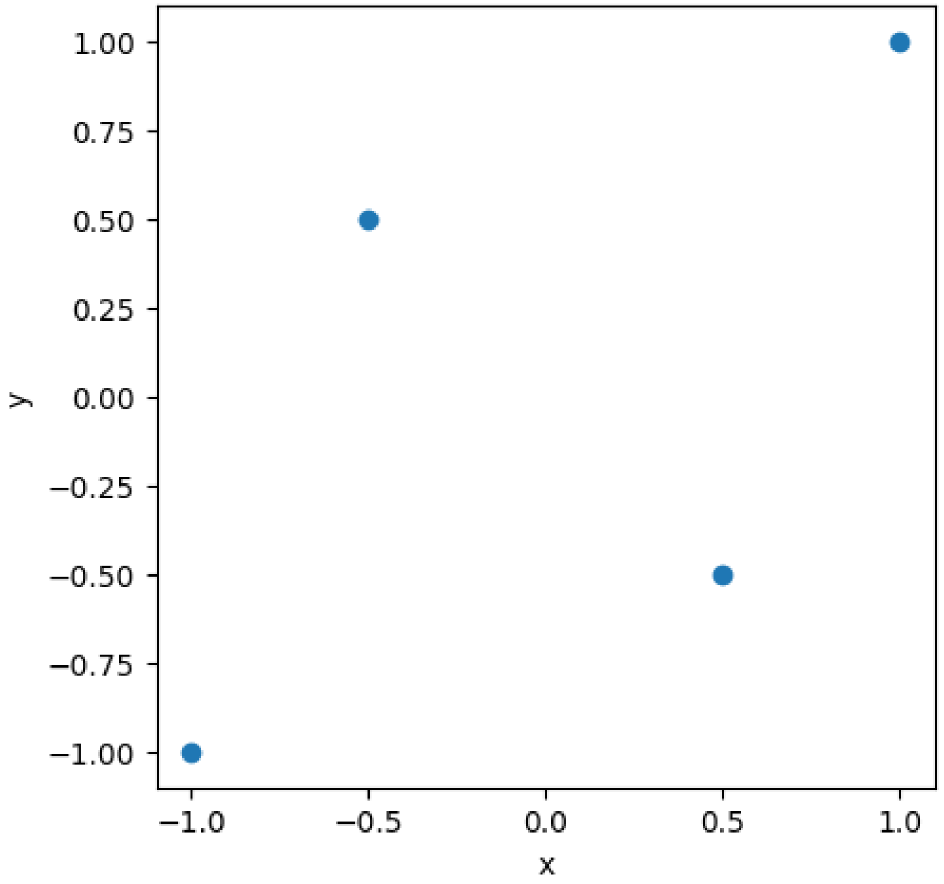
In the class…

- “Finding a coordinate transformation, where the covariance matrix is diagonal”
  - Eigenvalues
    = Variances in new coordinate system
  - Eigenvectors
    = New coordinate axes = Principal components
- Diagonalization / Eigendecomposition



Eigendecomposition

- Covariance matrix?
  - $S_{xx} = \sigma_x^2 = \frac{1}{4-1}\sum_{i=1}^{4}(x_i - \mu_x)^2$
    $= \frac{1}{3}(1^2 + 0.5^2 + (-0.5)^2 + (-1)^2)$
    $= 0.83$
  - $S_{yy} = \sigma_y^2 = \frac{1}{4-1}\sum_{i=1}^{4}\left(y_i - \mu_y\right)^2$
    $= \frac{1}{3}(1^2 + (-0.5)^2 + 0.5^2 + (-1)^2)$
    $= 0.83$

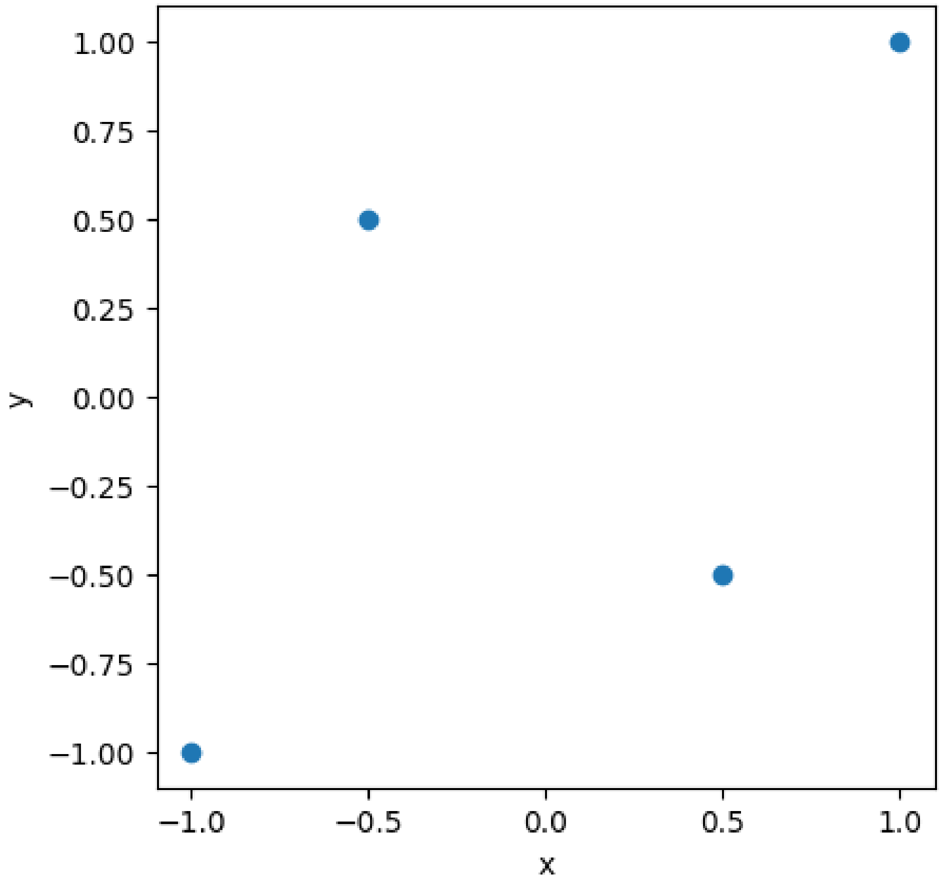


Eigendecomposition

- Covariance matrix?
  - $S_{xy} = S_{yx} = \frac{1}{4-1}\sum_{i=1}^{4}(x_i - \mu_x)(y_i - \mu_y)$

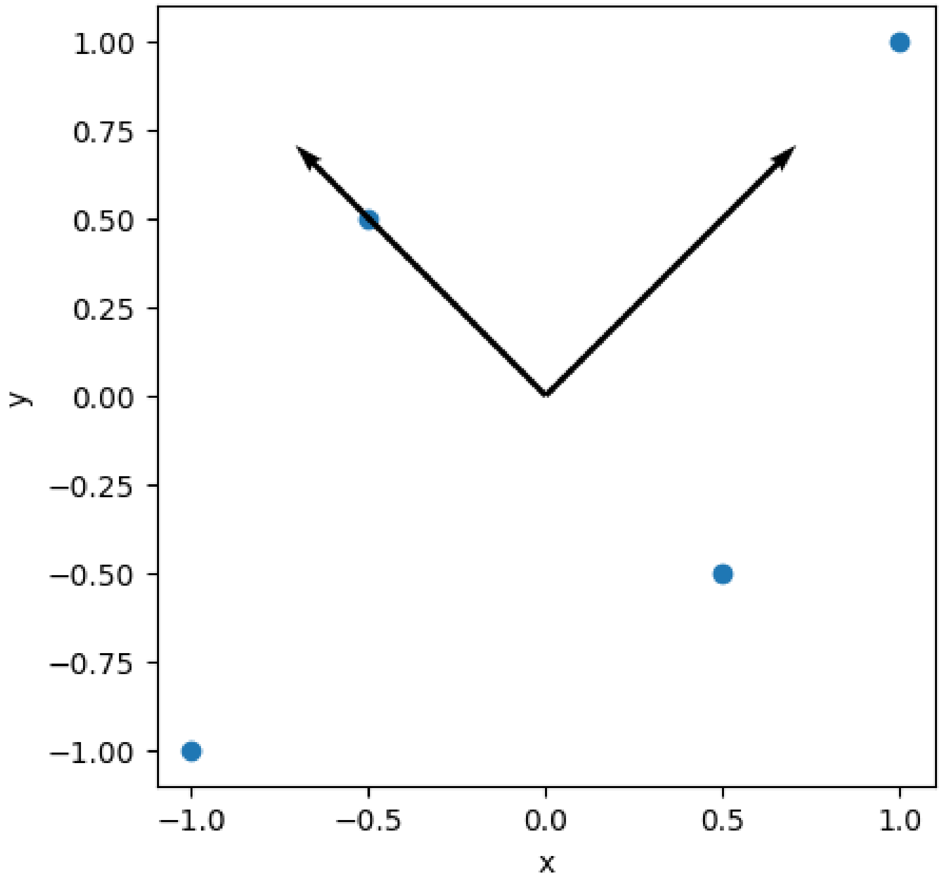
    $= \frac{1}{3}\begin{pmatrix} 1 \cdot 1 + 0.5 \cdot (-0.5) \\ +(-0.5) \cdot 0.5 + (-1) \cdot (-1) \end{pmatrix}$

    $= 0.5$



Eigendecomposition

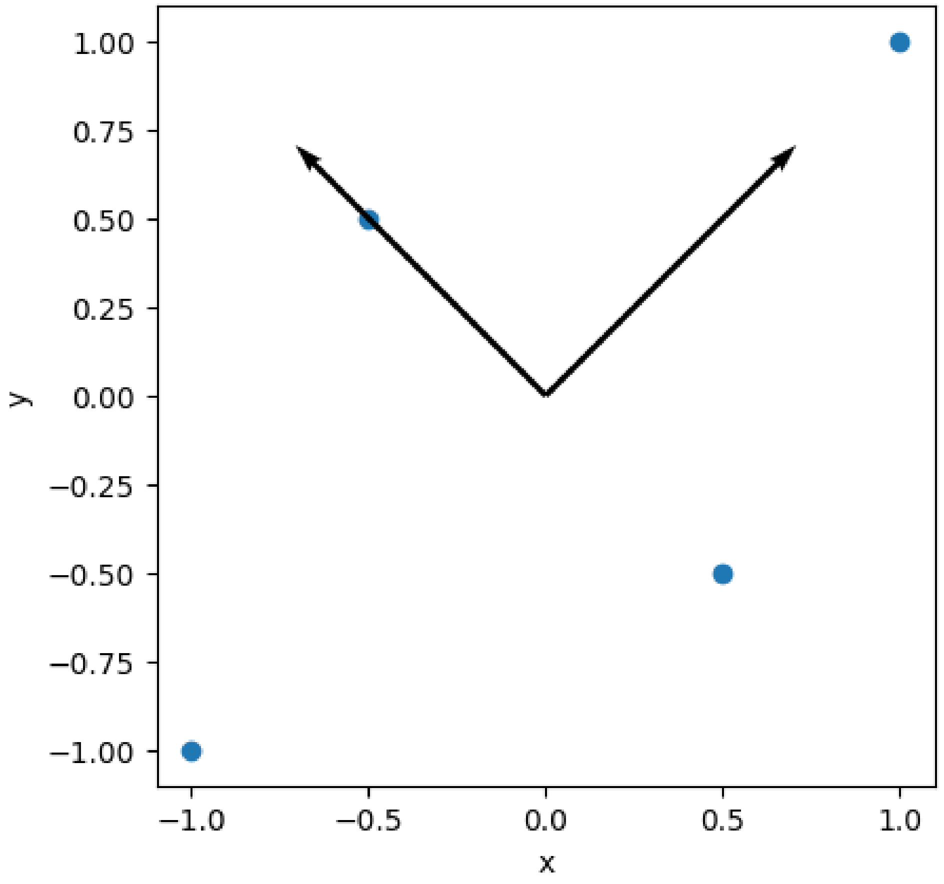
- Covariance matrix?
  - $S = \begin{bmatrix} 0.83 & 0.5 \\ 0.5 & 0.83 \end{bmatrix}$
  - Eigenvalues = 1.33, 0.33
  - Eigenvectors = [0.707, ±0.707]



Eigendecomposition

- Covariance matrix?
  - $S = \frac{1}{N-1}(X - \mu[X])^T(X - \mu[X])$

    N × N N × D D × N
  - N = 2 and D = 4 in this case



Exercise 1-1

- Implement your own PCA!
  - Try it on the toxicity data in the lecture
  - Check if you get the same results (eigenvalues etc.)
  - It's totally fine if your eigenvectors point to the opposite direction, why?

```
class MyPCA1:
    def fit(self, X):
        self.mean_ = np.mean(X, axis=0)
        self.n_samples_ = X.shape[0]
        X_centered = X - self.mean_
        cov_matrix = X_centered.T @ X_centered / (self.n_samples_ - 1)
        eigenvalues, eigenvectors = ...
        self.components_ = eigenvectors[:, ::-1].T
        self.explained_variance_ = eigenvalues[::-1]
        self.explained_variance_ratio_ = (
            self.explained_variance_ / np.sum(self.explained_variance_)
        )

    def transform(self, X):
        X_centered = X - self.mean_
        X_transformed = X_centered @ self.components_.T
        return X_transformed

pca = MyPCA1()
pca.fit(X_train)
print(pca.explained_variance_)
```

Calculate cov matrix

Run eigendecomposition of cov matrix with numpy

Confirm your results

In reality...

PCA

```
class sklearn.decomposition.PCA(n_components=None, *, copy=True, whiten=False, svd_solver='auto', tol=0.0, iterated_power='auto', n_oversamples=10, power_iteration_normalizer='auto', random_state=None)
```

Principal component analysis (PCA). ??? [source]

Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. The input data is centered but not scaled for each feature before applying the SVD.

Singular value decomposition

- What is eigendecomposition, again?
  - Finding $S = V\Lambda V^T$
  - Columns of V are normalized and orthogonal vectors
  - Λ is diagonal matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 16 & 0 \\ 0 & 0 & 9 \end{bmatrix}$$

- Issue: Covariant matrix is squared

$$S = \frac{1}{N-1}(X - \mu[X])^T(X - \mu[X])$$

- Alternative: Singular value decomposition
  - Finding $X = U\Sigma V^T$
  - Columns of U are normalized and orthogonal vectors
  - Columns of V are normalized and orthogonal vectors
  - Σ is rectangular diagonal matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \end{bmatrix}$$

Singular value decomposition

- Diagonalization
  - $S = \frac{1}{N-1}(X - \mu[X])^T(X - \mu[X]) = V\Lambda V^T$

N × N N × D D × N N × N N × N

N × N

- Singular value decomposition
  - $X - \mu[X] = U\Sigma V^T$

D × N D × D N × N

D × N

$$\left(S = \frac{1}{N-1} V\Sigma^{\mathrm{T}} U^T U\Sigma V^T = V\Lambda V^T\right)$$

$$\left(\Lambda = \frac{1}{N-1}\Sigma^T\Sigma\right)$$

Exercise 1-2

- (Re-)Implement your own PCA!
  - Try it on the toxicity data in the lecture
  - Check if you get the same results (eigenvalues etc.)

```python
class MyPCA2:
    def fit(self, X):
        self.mean_ = np.mean(X, axis=0)
        self.n_samples_ = X.shape[0]
        X_centered = X - self.mean_
        U, S, VT = ...
        self.components_ = VT
        self.explained_variance_ = (S ** 2) / (self.n_samples_ - 1)
        self.explained_variance_ratio_ = (
            self.explained_variance_ / np.sum(self.explained_variance_)
        )

    def transform(self, X):
        X_centered = X - self.mean_
        return X_centered @ self.components_.T

pca = MyPCA2()
pca.fit(X_train)
print(pca.explained_variance_)
```

Directly decompose X with numpy

Confirm your results

Linear regression

What happens under the hood

In the class…

$$\boxed{Y = X\beta + \varepsilon}$$

<u>fitting:</u> finding the best β in by minimizing the errors

$$(Y - X\beta)^T(Y - X\beta) = \sum_k {\varepsilon_k}^2$$

$$\frac{\partial}{\partial\beta}\sum_k {\varepsilon_k}^2 = 0 \longrightarrow \beta_{best} = \hat{\beta} = (X^T X)^{-1} X^T Y$$

Pseudoinversion

- We want to solve for

$$\underset{D \times 1}{Y} = \underset{D \times N}{X}\underset{N \times 1}{\beta}$$

- You may want to try

$$\underset{N \times D}{X^{-1}}\underset{D \times 1}{Y} = \underset{N \times 1}{\beta}$$

…but that only works if X is a square matrix

Pseudoinversion

- Make it square somehow?

$$\underset{N\times D}{X^T}\,\underset{D\times 1}{Y} = \underset{N\times N}{(X^T X)}\,\underset{N\times 1}{\beta}$$

- Now this is invertible

$$\underset{N\times N}{(X^T X)^{-1}}\,\underset{N\times D}{X^T}\,\underset{D\times 1}{Y} = \underset{N\times 1}{\beta}$$

which is what we saw in the class

Pseudoinversion

- We want to solve for

$$\underset{D \times 1}{Y} = \underset{D \times N}{X}\,\underset{N \times 1}{\beta}$$

- What we can do

$$\underbrace{(X^T X)^{-1} X^T}_{\substack{X^+ \\ N \times D}}\,\underset{D \times 1}{Y} = \underset{N \times 1}{\beta}$$

Exercise 2-1

- Implement your own OLS!
  - Try it on the toxicity data in the lecture
  - Check if you get the same results (toxicity score prediction etc.)

```python
class MyOLS1:
    def fit(self, X, y):
        X_pinv = ...
        self.coef_ = ...

    def predict(self, X):
        return X @ self.coef_

model = MyOLS1()
model.fit(X_train_const.to_numpy(), y_train)
y_test_pred = model.predict(X_test_const.to_numpy())
```

Calculate pseudoinversion

Calculate β

Confirm your results

In reality…

- We want to solve for

$$Y = X\beta$$

- What you can do

$$\underbrace{(X^T X)^{-1} X^T}_{X^+} Y = \beta$$

Singular value decomposition, again

- We want to solve for

$$\underset{\text{D × 1}}{Y} = \underset{\text{D × N}}{X}\underset{\text{N × 1}}{\beta}$$

- Let's do SVD on X

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \end{bmatrix}$$

$$\Sigma^+ = \begin{bmatrix} 1/\sigma_1 & 0 & 0 \\ 0 & 1/\sigma_2 & 0 \end{bmatrix}$$

$$Y = U\Sigma V^T\beta$$

$$\underset{\text{N × N}}{V}\underset{\text{N × D}}{\Sigma^+}\underset{\text{D × D}}{U^T}\underset{\text{D × 1}}{Y} = \underset{\text{N × 1}}{\beta}$$

Exercise 2-2

- (Re-)Implement your own OLS!
  - Try it on the toxicity data in the lecture
  - Check if you get the same results (toxicity score prediction etc.)

```python
class MyOLS2:
    def fit(self, X, y):
        U, S, VT = ...
        S_pinv = np.zeros_like(X.T)
        S_pinv[np.arange(min(X.shape)), np.arange(min(X.shape))] = ...
        self.coef_ = ...

    def predict(self, X):
        return X @ self.coef_

model = MyOLS2()
model.fit(X_train_const.to_numpy(), y_train)
y_test_pred = model.predict(X_test_const.to_numpy())
```

Solve SVD for X

Fill in Σ^+

Calculate β

Confirm your results

Questions?