T2.1 Algebra lineal: reducción de la dimensión y cálculo de derivadas

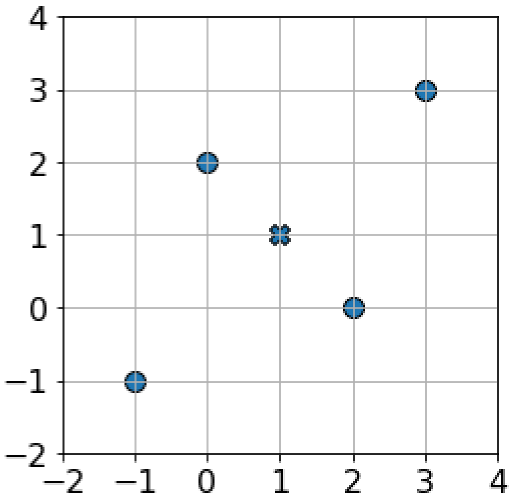
Índice

1 Matrices de datos

Sea $\mathbf{X} \in \mathbb{R}^{N \times D}$ una matriz de N datos D-dimensionales.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
X = np.array([ [-1, -1], [0, 2], [2, 0], [3, 3] ]).astype(float)
m = np.mean(X, axis=0)
fig, ax = plt.subplots(); ax.set_aspect("equal"); plt.grid(True)
plt.axis([-2, 4, -2, 4]); plt.xticks(fontsize=16); plt.yticks(fontsize=16)
plt.scatter(m[0], m[1], facecolor='C0', edgecolor='k', s=100, marker="X")
plt.scatter(X[:,0], X[:,1], facecolor='C0', edgecolor='k', s=100);
```



1.1 Suma de trozos de la matriz

Suma de filas: $\mathbf{1}_N^t\mathbf{X} = (\sum_n x_{n1}, \cdots, \sum_n x_{nD})$

Media de los datos: $\bar{\boldsymbol{x}}^t = \frac{1}{N}\mathbf{1}_N^t\mathbf{X}$

Suma de columnas: $\mathbf{X}\mathbf{1}_D = \begin{pmatrix} \sum_d x_{1d} \\ \vdots \\ \sum_d x_{Nd} \end{pmatrix}$

Suma de todas las entradas: $\mathbf{1}_N^t\mathbf{X}\mathbf{1}_D = \sum_{ij} x_{ij}$

Media global de los datos: $\bar{x} = \frac{1}{ND}\mathbf{1}_N^t\mathbf{X}\mathbf{1}_D$

In [2]:

```python
import numpy as np
X = np.array([ [-1, -1], [0, 2], [2, 0], [3, 3] ]).astype(float); N, D = X.shape
print("Suma de filas: ", np.ones(N) @ X, np.sum(X, axis=0))
print("Media de los datos: ", 1/N * np.ones(N) @ X, np.mean(X, axis=0))
print("Suma de columnas: ", X @ np.ones(D), np.sum(X, axis=1))
print("Suma de todas las entradas: ", np.ones(N) @ X @ np.ones(D), np.sum(X))
print("Media global de los datos: ", 1/(N*D) * np.ones(N) @ X @ np.ones(D), np.mean(X))
```

```
Suma de filas:  [4. 4.] [4. 4.]
Media de los datos:  [1. 1.] [1. 1.]
Suma de columnas:  [-2.  2.  2.  6.] [-2.  2.  2.  6.]
Suma de todas las entradas:  8.0 8.0
```

```
Media global de los datos:  1.0 1.0
```

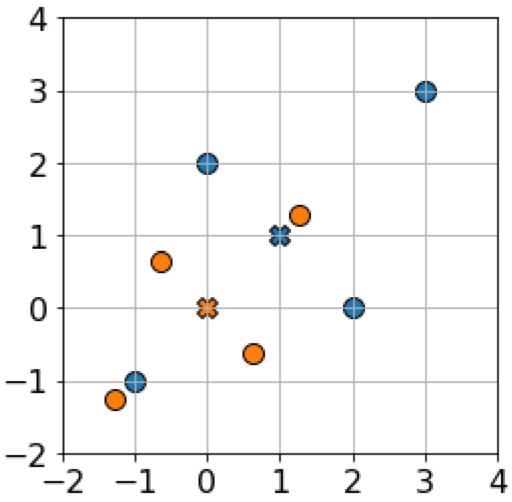
1.2 Escalado de filas y columnas de una matriz

Escalado de filas con $\mathbf{S}=\operatorname{diag}(\boldsymbol{s})$: $\quad\mathbf{SX}=\begin{bmatrix}s_1&\cdots&0\\&\ddots&\\0&\cdots&s_N\end{bmatrix}\begin{bmatrix}x_{11}&\cdots&x_{1D}\\&\ddots&\\x_{N1}&\cdots&x_{ND}\end{bmatrix}=\begin{bmatrix}s_1x_{11}&\cdots&s_1x_{1D}\\&\ddots&\\s_Nx_{N1}&\cdots&s_Nx_{ND}\end{bmatrix}$

Escalado de columnas con $\mathbf{S}=\operatorname{diag}(\boldsymbol{s})$: $\quad\mathbf{XS}=\begin{bmatrix}x_{11}&\cdots&x_{1D}\\&\ddots&\\x_{N1}&\cdots&x_{ND}\end{bmatrix}\begin{bmatrix}s_1&\cdots&0\\&\ddots&\\0&\cdots&s_D\end{bmatrix}=\begin{bmatrix}s_1x_{11}&\cdots&s_Dx_{1D}\\&\ddots&\\s_1x_{N1}&\cdots&s_Dx_{ND}\end{bmatrix}$

Estandarización: $\quad\operatorname{standardize}(\mathbf{X})=(\mathbf{X}-\mathbf{1}_N\boldsymbol{\mu}^t)\operatorname{diag}(\boldsymbol{\sigma})^{-1}$

In [3]:
```python
import numpy as np
import matplotlib.pyplot as plt
X = np.array([ [-1, -1], [0, 2], [2, 0], [3, 3] ]).astype(float)
m = np.mean(X, axis=0); sigma = np.std(X, axis=0)
Xstd = (X - m) / sigma
fig, ax = plt.subplots(); ax.set_aspect("equal"); plt.grid(True)
plt.axis([-2, 4, -2, 4]); plt.xticks(fontsize=16); plt.yticks(fontsize=16)
plt.scatter(m[0], m[1], facecolor='C0', edgecolor='k', s=100, marker="X")
plt.scatter(X[:,0], X[:,1], facecolor='C0', edgecolor='k', s=100);
plt.scatter(0, 0, facecolor='C1', edgecolor='k', s=100, marker="X")
plt.scatter(Xstd[:,0], Xstd[:,1], facecolor='C1', edgecolor='k', s=100);
```



1.3 Suma de cuadrados, centrado y matriz de dispersión

Suma de cuadrados: $\mathbf{S}_0 = \mathbf{X}^t\mathbf{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N] \begin{bmatrix} \boldsymbol{x}_1^t \\ \boldsymbol{x}_2^t \\ \vdots \\ \boldsymbol{x}_N^t \end{bmatrix} = \sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^t = \sum_{n=1}^{N} \begin{pmatrix} x_{n1}^2 & \cdots & x_{n1}x_{nD} \\ & \ddots & \\ x_{nD}x_{n1} & \cdots & x_{nD}^2 \end{pmatrix}$

Matriz de centrado: $\mathbf{C}_N = \mathbf{I}_N - \frac{1}{N}\mathbf{1}_N\mathbf{1}_N^t$, simétrica e idempotente ($\mathbf{C}_N^k = \mathbf{C}_N, k \geq 1$), centra los datos

$$\tilde{\mathbf{X}} = \mathbf{X} - \mathbf{1}_N\bar{\boldsymbol{x}}^t = \mathbf{X} - \frac{1}{N}\mathbf{1}_N\mathbf{1}_N^t\mathbf{X} = \mathbf{C}_N\mathbf{X}$$

Matriz de dispersión: $\mathbf{S}_{\bar{\boldsymbol{x}}} = \sum_{n=1}^{N}(\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^t = \tilde{\mathbf{X}}^t\tilde{\mathbf{X}} = \mathbf{X}^t\mathbf{C}_N^t\mathbf{C}_N\mathbf{X} = \mathbf{X}^t\mathbf{C}_N\mathbf{X}$

In [4]:
```
import numpy as np
X = np.array([ [-1, -1], [0, 2], [2, 0], [3, 3] ]).astype(float); N, D = X.shape
print("Suma de cuadrados:\n", X.T @ X)
C = np.eye(N) - np.ones((N, N))/N; print("Matriz de centrado:\n", C)
print("Matriz de dispersión:\n", X.T @ C @ X, "\n", N * np.cov(X, rowvar=False, bias=True))
```

```
Suma de cuadrados:
 [[14. 10.]
 [10. 14.]]
Matriz de centrado:
 [[ 0.75 -0.25 -0.25 -0.25]
 [-0.25  0.75 -0.25 -0.25]
 [-0.25 -0.25  0.75 -0.25]
 [-0.25 -0.25 -0.25  0.75]]
Matriz de dispersión:
 [[10.  6.]
 [ 6. 10.]]
 [[10.  6.]
 [ 6. 10.]]
```

1.4 Matriz de Gram

Matriz de Gram: $\mathbf{K} = \mathbf{X}\mathbf{X}^t = \begin{bmatrix} \boldsymbol{x}_1^t\boldsymbol{x}_1 & \cdots & \boldsymbol{x}_1^t\boldsymbol{x}_N \\ & \vdots & \\ \boldsymbol{x}_N^t\boldsymbol{x}_1 & \cdots & \boldsymbol{x}_N^t\boldsymbol{x}_N \end{bmatrix}$ (matriz de productos escalares)

Gram para datos centrados a partir de Gram: **truco del doble centrado**

$$\tilde{\mathbf{K}} = \tilde{\mathbf{X}}\tilde{\mathbf{X}}^t = \mathbf{C}_N\mathbf{X}(\mathbf{C}_N\mathbf{X})^t = \mathbf{C}_N\mathbf{K}\mathbf{C}_N$$

In [5]:
```
import numpy as np
X = np.array([ [-1, -1], [0, 2], [2, 0], [3, 3] ]).astype(float); N, D = X.shape
K = X @ X.T; print("Gram:\n", K)
Xc = X - np.mean(X, axis=0)
Kc = Xc @ Xc.T; print("Gram centrados:\n", Kc)
C = np.eye(N) - np.ones((N, N))/N
Kc_K = C @ K @ C; print("Gram centrados con truco:\n", Kc_K)
```

```
Gram:
 [[ 2. -2. -2. -6.]
 [-2.  4.  0.  6.]
 [-2.  0.  4.  6.]
 [-6.  6.  6. 18.]]
Gram centrados:
 [[ 8.  0.  0. -8.]
 [ 0.  2. -2.  0.]
 [ 0. -2.  2.  0.]
 [-8.  0.  0.  8.]]
Gram centrados con truco:
 [[ 8.  0.  0. -8.]
 [ 0.  2. -2.  0.]
 [ 0. -2.  2.  0.]
 [-8.  0.  0.  8.]]
```

1.5 Matriz de distancias

Distancia (Euclídea) al cuadrado entre $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^D$: $\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 = (\boldsymbol{x}-\boldsymbol{y})^t(\boldsymbol{x}-\boldsymbol{y}) = \|\boldsymbol{x}\|_2^2 - 2\boldsymbol{x}^t\boldsymbol{y} + \|\boldsymbol{y}\|_2^2$

Extensión a matrices de datos, $\mathbf{X} \in \mathbb{R}^{N\times D}$, $\mathbf{Y} \in \mathbb{R}^{M\times D}$: $\mathbf{D} \in \mathbb{R}^{N\times M}$

$$\mathbf{D} = \operatorname{diag}(\mathbf{X}\mathbf{X}^t)\mathbf{1}_M^t - 2\mathbf{X}\mathbf{Y}^t + \mathbf{1}_N \operatorname{diag}(\mathbf{Y}\mathbf{Y}^t)^t$$

Caso $\mathbf{Y} = \mathbf{X}$: $\mathbf{D} = \mathbf{H} - 2\mathbf{K} + \mathbf{H}^t$ con $\mathbf{H} = \operatorname{diag}(\mathbf{K})\mathbf{1}_N^t$ y $\mathbf{K} = \mathbf{X}\mathbf{X}^t$ (Gram)

In [6]:
```
import numpy as np
X = np.array([ [-1, -1], [0, 2], [2, 0], [3, 3] ]).astype(float)
K = X @ X.T; H = np.diag(K).reshape(-1, 1); print(H - 2*K + H.T)
```

```
[[ 0. 10. 10. 32.]
 [10.  0.  8. 10.]
 [10.  8.  0. 10.]
 [32. 10. 10.  0.]]
```

In [7]:
```
import numpy as np
import scipy.spatial.distance as dist
X = np.array([ [-1, -1], [0, 2], [2, 0], [3, 3] ]).astype(float)
V = dist.pdist(X, 'sqeuclidean')
print(V, "\n", dist.squareform(V))
```

```
[10. 10. 32.  8. 10. 10.]
 [[ 0. 10. 10. 32.]
 [10.  0.  8. 10.]
 [10.  8.  0. 10.]
 [32. 10. 10.  0.]]
```

2 Diagonalización

2.1 Conceptos básicos

Sea $\mathbf{A} \in \mathbb{R}^{n \times n}$ una matriz cuadrada.

2.1.1 Valor y vector propio

Valor y vector propio: $\lambda \in \mathbb{R}$ es un **valor propio** de $\mathbf{A}$ y $\boldsymbol{u} \in \mathbf{R}^n$ "el" correspondiente **vector propio** asociado si:

$$\mathbf{A}\boldsymbol{u} = \lambda\boldsymbol{u}, \qquad \boldsymbol{u} \neq \mathbf{0}$$

No unicidad del vector propio: si $\boldsymbol{u}$ es vector propio asociado a λ, todo $c\boldsymbol{u}$ con $c \in \mathbf{R}$ no nulo tamién lo es

$$\mathbf{A}(c\boldsymbol{u}) = c\mathbf{A}\boldsymbol{u} = c\lambda\boldsymbol{u} = \lambda(c\boldsymbol{u})$$

Asunción de normalidad: por simplicidad, se asume que "el" vector propio asociado a λ está normalizado, $\|\boldsymbol{u}\|_2 = 1$; el entrecomillado "el" se debe a que $-\boldsymbol{u}$ también está normalizado, por lo que tenemos dos opciones

2.1.2 Ecuación característica

Ecuación característica: permite hallar los n valores propios de $\mathbf{A}$ y sus vectores propios asociados

$$|\lambda\mathbf{I}_n - \mathbf{A}| = 0$$

2.1.3 Propiedades

Traza de $\mathbf{A}$: $\operatorname{tr}(\mathbf{A}) = \sum_{i=1}^{n} \lambda_i$

Determinante de $\mathbf{A}$: $|\mathbf{A}| = \prod_{i=1}^{n} \lambda_i$

Rango de $\mathbf{A}$: $\operatorname{range}(\mathbf{A}) = \sum_{i=1}^{n} \mathbb{I}(\lambda_i > 0)$

Valor y vector propio de $\mathbf{A}^{-1}$: si $\mathbf{A}$ es no singular y $(\lambda_i, \boldsymbol{u}_i)$ es un par valor-vector propio de $\mathbf{A}$, entonces $\mathbf{A}^{-1}$ existe y $(1/\lambda_i, \boldsymbol{u}_i)$ es un par valor-vector propio de $\mathbf{A}^{-1}$

2.2 Diagonalización

Sea $\mathbf{A} \in \mathbb{R}^{n \times n}$ una matriz cuadrada.

2.2.1 Ecuación global

Ecuación global: $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ y $\mathbf{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)$ son matrices de valores y vectores propios de $\mathbf{A}$ si:

$$\mathbf{A}\mathbf{U} = [\mathbf{A}\boldsymbol{u}_1, \mathbf{A}\boldsymbol{u}_2, \ldots, \mathbf{A}\boldsymbol{u}_n] = [\lambda_1\boldsymbol{u}_1, \lambda_2\boldsymbol{u}_2, \ldots, \lambda_n\boldsymbol{u}_n] = \mathbf{U}\boldsymbol{\Lambda}$$

2.2.2 Descomposición propia

Si los vectores propios son linealmente independientes, $\mathbf{U}$ es invertible y $\mathbf{A}$ **diagonalizable**, con **descomposición propia:**

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{-1}$$

3 Matrices reales y simétricas

3.1 Valores y vectores propios de matrices reales y simétricas

Sea $\mathbf{A} \in \mathbb{R}^{n \times n}$ una matriz **real y simétrica.**

3.1.1 Descomposición propia de una matriz real y simétrica

3.1.1 Descomposición propia de una matriz real y simétrica

Valores propios de $\mathbf{A}$: reales, $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, $\lambda_i \in \mathbb{R}$

Vectores propios de $\mathbf{A}$: ortogonales dos a dos y normalizados, $\boldsymbol{u}_i^t \boldsymbol{u}_j = \mathbb{I}(i = j)$, por lo que $\mathbf{U}$ es ortogonal, $\mathbf{U}^t\mathbf{U} = \mathbf{U}\mathbf{U}^t = \mathbf{I}$

Descomposición propia de $\mathbf{A}$: $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^t = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)\, \mathrm{diag}(\lambda_1, \ldots, \lambda_n)(\boldsymbol{u}_1^t; \ldots; \boldsymbol{u}_n^t) = \sum_{i=1}^n \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^t$

Inversa (si existe): $\mathbf{A}^{-1} = \mathbf{U}\mathbf{\Lambda}^{-1}\mathbf{U}^t = \sum_{i=1}^n \frac{1}{\lambda_i} \boldsymbol{u}_i \boldsymbol{u}_i^t$

Ejercicio: Halla la descomposición propia de $\mathbf{\Sigma} = \begin{bmatrix} \frac{5}{2} & \frac{3}{2} \\ \frac{3}{2} & \frac{5}{2} \end{bmatrix}$

$$|\lambda \mathbf{I} - \mathbf{\Sigma}| = 0 \to \left| \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} \frac{5}{2} & \frac{3}{2} \\ \frac{3}{2} & \frac{5}{2} \end{bmatrix} \right| = \begin{vmatrix} \lambda - \frac{5}{2} & -\frac{3}{2} \\ -\frac{3}{2} & \lambda - \frac{5}{2} \end{vmatrix} = 0$$

$$\to \left(\lambda - \frac{5}{2}\right)^2 - \frac{9}{4} = 0 \to \lambda^2 - 5\lambda + 4 = 0 \to \begin{cases} \lambda_1 = 4 \\ \lambda_2 = 1 \end{cases}$$

$$\mathbf{\Sigma}\boldsymbol{e}_1 = \lambda_1 \boldsymbol{e}_1 \to \boldsymbol{e}_1 = \begin{pmatrix} \alpha \\ \alpha \end{pmatrix} \xrightarrow{\|\boldsymbol{e}_1\|=1} \boldsymbol{e}_1 = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix}$$

$$\mathbf{\Sigma}\boldsymbol{e}_2 = \lambda_2 \boldsymbol{e}_2 \to \boldsymbol{e}_2 = \begin{pmatrix} -\alpha \\ \alpha \end{pmatrix} \xrightarrow{\|\boldsymbol{e}_1\|=1} \boldsymbol{e}_2 = \begin{bmatrix} -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} -0.7071 \\ 0.7071 \end{bmatrix}$$

$$\mathbf{\Sigma} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^t \qquad \mathbf{U} = [\boldsymbol{e}_1, \boldsymbol{e}_2] \qquad \mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2)$$

Ejercicio (cont.): invierte $\mathbf{\Sigma}$

$$\begin{aligned} \mathbf{\Sigma}^{-1} &= [\boldsymbol{u}_1, \boldsymbol{u}_2]\, \mathrm{diag}(1/\lambda_1, 1/\lambda_2) \begin{bmatrix} \boldsymbol{u}_1^t \\ \boldsymbol{u}_2^t \end{bmatrix} \\ &= \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} 1/4 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{8} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{8} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \\ &= \begin{bmatrix} \frac{2}{16} + \frac{2}{4} & \frac{2}{16} - \frac{2}{4} \\ \frac{2}{16} - \frac{2}{4} & \frac{2}{16} + \frac{2}{4} \end{bmatrix} = \begin{bmatrix} \frac{5}{8} & -\frac{3}{8} \\ -\frac{3}{8} & \frac{5}{8} \end{bmatrix} = \begin{bmatrix} 0.625 & -0.375 \\ -0.375 & 0.625 \end{bmatrix} \end{aligned}$$

In [1]:
```python
import numpy as np
S = np.array([[5/2, 3/2], [3/2, 5/2]])
La, U = np.linalg.eigh(S)
i = La.argsort()[::-1]; La = La[i]; U = U[:,i]
I = U @ np.diag(1/La) @ U.T
print(La, "\n", U, "\n", I)
```

```
[4. 1.]
 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
 [[ 0.625 -0.375]
 [-0.375  0.625]]
```

3.1.2 Comprobación de definición positiva

Consideremos la forma cuadrática asociada a $\mathbf{A}$:

$$f(\boldsymbol{x})=\boldsymbol{x}^t\mathbf{A}\boldsymbol{x}=\boldsymbol{x}^t\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^t\boldsymbol{x}\overset{\boldsymbol{y}=\mathbf{U}^t\boldsymbol{x}}{=}\boldsymbol{y}^t\boldsymbol{\Lambda}\boldsymbol{y}=\sum_{i=1}^n\lambda_i y_i^2$$

Como $y_i^2\geq 0$, el signo solo depende de los λ_i:

- $\mathbf{A}$ es **definida positiva** sii $\lambda_i>0$ para todo i
- $\mathbf{A}$ es **semidefinida positiva** sii $\lambda_i\geq 0$ para todo i
- $\mathbf{A}$ es **definida negativa** sii $\lambda_i<0$ para todo i
- $\mathbf{A}$ es **semidefinida negativa** sii $\lambda_i\leq 0$ para todo i
- $\mathbf{A}$ es **indefinida** sii tiene λ_i positivos y negativos

3.2 Geometría de las formas cuadráticas

Matriz de covarianzas: $\boldsymbol{\Sigma}$ real, simétrica y semi-definida positiva sii $\boldsymbol{\Sigma}$ es matriz de covarianzas

Valores propios de $\boldsymbol{\Sigma}$: no negativos

Valores propios de $\boldsymbol{\Sigma}$ definida positiva: positivos, ninguno nulo; $\boldsymbol{\Sigma}^{-1}$ existe y es definida positiva

Forma cuadrática asociada a $\boldsymbol{\Sigma}^{-1}$ definida positiva: distancia de Mahalanobis (al cuadrado)

$$f(\boldsymbol{x}) = \boldsymbol{x}^t\boldsymbol{\Sigma}^{-1}\boldsymbol{x} = \boldsymbol{x}^t\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^t\boldsymbol{x} = \boldsymbol{y}^t\boldsymbol{\Lambda}\boldsymbol{y} = \sum_{i=1}\lambda_i y_i^2$$

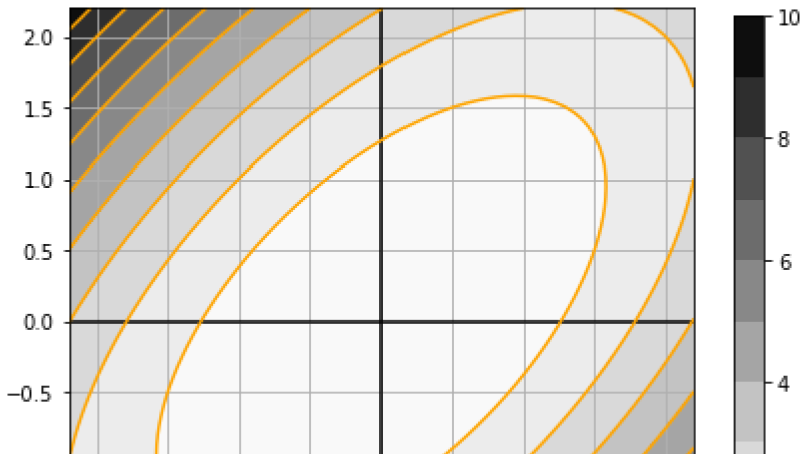
Los conjuntos de nivel de $f(\boldsymbol{x})$ son hiperelipsoides; elipses en 2d.

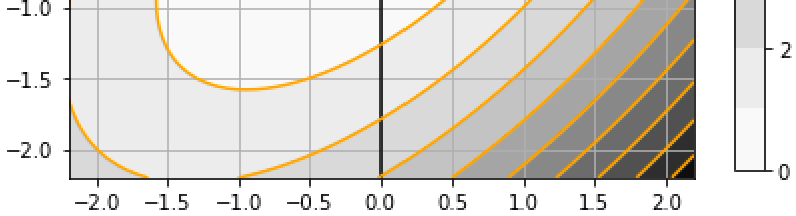
Ejemplo: distancia de Mahalanobis al origen (al cuadrado) con $\boldsymbol{\Sigma} = \begin{bmatrix} \frac{5}{2} & \frac{3}{2} \\ \frac{3}{2} & \frac{5}{2} \end{bmatrix}$

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
S = np.array([[5/2, 3/2], [3/2, 5/2]])
La, U = np.linalg.eigh(S)
i = La.argsort()[::-1]; La = La[i]; U = U[:,i]
I = U @ np.diag(1/La) @ U.T
print(La, "\n", U, "\n", I)
x_min = y_min = -2.2; x_max = y_max = 2.2
X, Y = np.meshgrid(np.linspace(x_min, x_max, num=64), np.linspace(y_min, y_max, num=64))
XY = np.c_[np.ravel(X), np.ravel(Y)]
d = lambda xy: xy.T @ I @ xy
D = np.apply_along_axis(d, 1, XY)
fig, ax = plt.subplots(figsize=(7, 7))
ax.set(aspect='equal', xlim=(x_min, x_max), ylim=(y_min, y_max))
ax.grid(); ax.axhline(0, color='black'); ax.axvline(0, color='black')
ax.contour(X, Y, D.reshape(X.shape), 10, colors='orange', linestyles='solid')
cp = ax.contourf(X, Y, D.reshape(X.shape), 10, cmap="Greys")
plt.colorbar(cp, ax=ax, shrink=0.8);
```

```
[4. 1.]
 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
 [[ 0.625 -0.375]
 [-0.375  0.625]]
```





4 Reducción de la dimensión

4.1 Análisis de componentes principales (PCA)

Maldición de la dimensionalidad: muchas técnicas de aprendizaje automático empeoran sensiblemente con datos de alta dimensión

Reducción de la dimensión: dada una matriz de N datos en un espacio de alta dimensión, $\mathbb{R}^D$, queremos aprender (no supervisadamente) una transformación de $\mathbb{R}^D$ en un espacio de dimensión reducida, $\mathbb{R}^K, K \ll D$, que produzca una "buena aproximación" de los datos originales

Codificación: operación de reducción de la dimensión, $\operatorname{encode}(\boldsymbol{x}) = \boldsymbol{z}$, $\boldsymbol{x} \in \mathbb{R}^D$ y $\boldsymbol{z} \in \mathbb{R}^K$

Decodificación: operación de reconstrucción del dato original, $\operatorname{decode}(\boldsymbol{z}) = \hat{\boldsymbol{x}}$, $\hat{\boldsymbol{x}} \in \mathbb{R}^D$ y $\boldsymbol{z} \in \mathbb{R}^K$

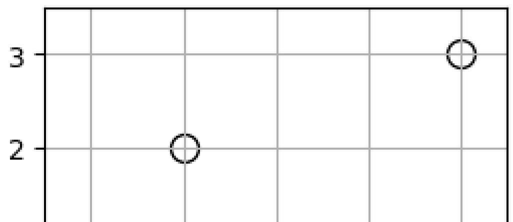
PCA: escoge una proyección lineal ortogonal $\mathbf{W} \in \mathbb{R}^{D\times K}$ de mínima **distorsión o error de reconstrucción**

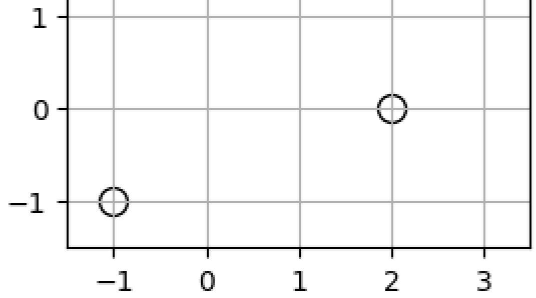
$$\mathcal{L}(\mathbf{W}) = \frac{1}{N}\sum_n \|\boldsymbol{x}_n - \operatorname{decode}(\operatorname{encode}(\boldsymbol{x}_n))\|_2^2 \qquad \text{con} \quad \operatorname{encode}(\boldsymbol{x}) = \mathbf{W}^t\boldsymbol{x} \quad \text{y} \quad \operatorname{decode}(\boldsymbol{z}) = \mathbf{W}\boldsymbol{z}$$

Ejemplo: conjunto de $N = 4$ bidimensionales que queremos reducir a unidimensionales con mínima distorsión

In [1]:

```python
import numpy as np; import matplotlib.pyplot as plt
X = np.array([ [-1, -1], [0, 2], [2, 0], [3, 3] ]); N = len(X); fig, ax = plt.subplots(figsize=(3, 3))
ax.set_aspect("equal"); plt.axis([-1.5, 3.5, -1.5, 3.5]); plt.grid(True)
plt.scatter(*X.T, facecolor='white', edgecolor='k', s=100);
```





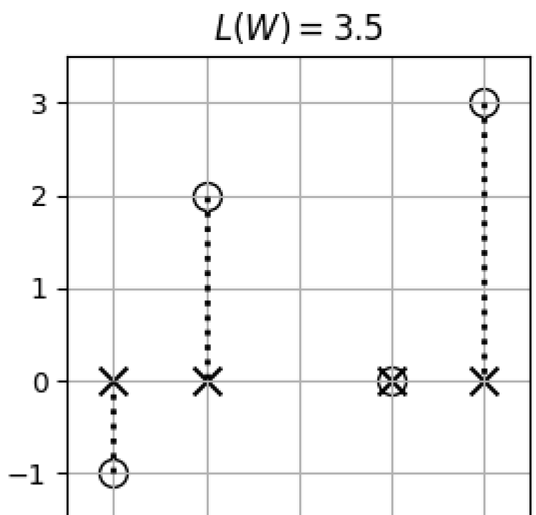
Ejercicio: codifica, decodifica y halla la distorsión de los datos del ejemplo con $\mathbf{W}=(1,0)^t$

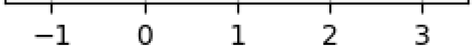
$$\begin{array}{llll}
\boldsymbol{x}_1=(-1,-1)^t & \boldsymbol{x}_2=(0,2)^t & \boldsymbol{x}_3=(2,0)^t & \boldsymbol{x}_4=(3,3)^t\\
z_1=\mathbf{W}^t\boldsymbol{x}_1=-1 & z_2=0 & z_3=2 & z_4=3\\
\hat{\boldsymbol{x}}_1=\mathbf{W}z_1=(-1,0)^t & \hat{\boldsymbol{x}}_2=(0,0)^t & \hat{\boldsymbol{x}}_3=(2,0)^t & \hat{\boldsymbol{x}}_4=(3,0)^t
\end{array}$$

$$\mathcal{L}(\mathbf{W})=\frac{1}{4}\left(\|\boldsymbol{x}_1-\hat{\boldsymbol{x}}_1\|_2^2+\|\boldsymbol{x}_2-\hat{\boldsymbol{x}}_2\|_2^2+\|\boldsymbol{x}_3-\hat{\boldsymbol{x}}_3\|_2^2+\|\boldsymbol{x}_4-\hat{\boldsymbol{x}}_4\|_2^2\right)=\frac{1}{4}(1+4+0+9)=\frac{14}{4}=3.5$$

In [2]:

```
import numpy as np; import matplotlib.pyplot as plt; from matplotlib.collections import LineCollection
X = np.array([ [-1, -1], [0, 2], [2, 0], [3, 3] ]); N = len(X); fig, ax = plt.subplots(figsize=(3, 3))
ax.set_aspect("equal"); plt.axis([-1.5, 3.5, -1.5, 3.5]); plt.grid(True)
plt.scatter(*X.T, facecolor='white', edgecolor='k', s=100)
K = 1; W = np.array([1, 0]).reshape(-1, K); Z = (X @ W).reshape(-1, K); hX = Z @ W.T
L = np.square(X - hX).sum(axis=1).mean(); ax.set_title(f'$L(W)={L}$')
plt.scatter(*hX.T, facecolor='black', s=100, marker='x')
lines = np.zeros((N, 2, 2)); lines[:, 0, :] = X; lines[:, 1, :] = hX
ax.add_collection(LineCollection(lines, linewidths=2, colors='black', linestyle='dotted'));
```





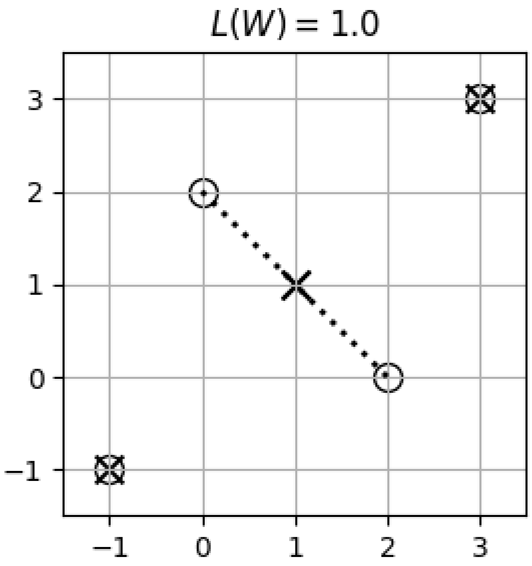
Ejercicio: codifica, decodifica y halla la distorsión de los datos del ejemplo con $\mathbf{W}=(\sqrt{2}/2,\sqrt{2}/2)^t$

$$\begin{array}{llll}
\boldsymbol{x}_1=(-1,-1)^t & \boldsymbol{x}_2=(0,2)^t & \boldsymbol{x}_3=(2,0)^t & \boldsymbol{x}_4=(3,3)^t\\
z_1=\mathbf{W}^t\boldsymbol{x}_1=-\sqrt{2} & z_2=\sqrt{2} & z_3=\sqrt{2} & z_4=3\sqrt{2}\\
\hat{\boldsymbol{x}}_1=\mathbf{W}z_1=(-1,-1)^t & \hat{\boldsymbol{x}}_2=(1,1)^t & \hat{\boldsymbol{x}}_3=(1,1)^t & \hat{\boldsymbol{x}}_4=(3,3)^t
\end{array}$$

$$\mathcal{L}(\mathbf{W})=\frac{1}{4}\left(\|\boldsymbol{x}_1-\hat{\boldsymbol{x}}_1\|_2^2+\|\boldsymbol{x}_2-\hat{\boldsymbol{x}}_2\|_2^2+\|\boldsymbol{x}_3-\hat{\boldsymbol{x}}_3\|_2^2+\|\boldsymbol{x}_4-\hat{\boldsymbol{x}}_4\|_2^2\right)=\frac{1}{4}(0+2+2+0)=1$$

In [3]:

```
import numpy as np; import matplotlib.pyplot as plt; from matplotlib.collections import LineCollection
X = np.array([ [-1, -1], [0, 2], [2, 0], [3, 3] ]); N = len(X); fig, ax = plt.subplots(figsize=(3, 3))
ax.set_aspect("equal"); plt.axis([-1.5, 3.5, -1.5, 3.5]); plt.grid(True)
plt.scatter(*X.T, facecolor='white', edgecolor='k', s=100)
K = 1; W = np.array([np.sqrt(2)/2, np.sqrt(2)/2]).reshape(-1, K); Z = (X @ W).reshape(-1, K); hX = Z @ W.T
L = np.square(X - hX).sum(axis=1).mean(); ax.set_title(f'$L(W)={L}$')
plt.scatter(*hX.T, facecolor='black', s=100, marker='x')
lines = np.zeros((N, 2, 2)); lines[:, 0, :] = X; lines[:, 1, :] = hX
ax.add_collection(LineCollection(lines, linewidths=2, colors='black', linestyle='dotted'));
```



Cálculo de componentes principales: columnas de la proyección lineal ortogonal $\mathbf{W}\in\mathbb{R}^{D\times K}$ de mínima distorsión

- **Descomposición propia de la matriz de covarianzas empírica ordenada por λs en orden no creciente**

$$\mathbf{\Sigma} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^t \quad \text{donde} \quad \mathbf{U} = (\boldsymbol{u}_1, \dots, \boldsymbol{u}_D) \quad \text{y} \quad \mathbf{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_n) \quad \text{con} \quad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D$$

- **Minimizar la distorsión equivale a maximizar la varianza (de los datos proyectados)**
  - $\boldsymbol{u}_1$ es una dirección de proyección óptima para maximizar la varianza de los datos proyectados, siendo λ_1 dicha varianza
  - Entre todas las direcciones ortonormales a $\boldsymbol{u}_1$, $\boldsymbol{u}_2$ y λ_2 son dirección óptima de proyección y varianza correspondiente
  - Y así, sucesivamente, hasta $\boldsymbol{u}_K$ y λ_K
- **Proyección lineal ortogonal en $\mathbb{R}^K$ de mínima distorsión o máxima varianza retenida**

$$\mathbf{W}_{\text{pca}} = (\boldsymbol{u}_1, \dots, \boldsymbol{u}_K) \in \mathbb{R}^{D\times K}$$

Ejercicio: calcula $\mathbf{W}_{\text{pca}}$ con los datos del ejemplo

- **Matriz de covarianzas empírica:** normalización de la matriz de dispersión hallada en 7.2.4, $\mathbf{\Sigma} = \begin{bmatrix} \frac{5}{2} & \frac{3}{2} \\ \frac{3}{2} & \frac{5}{2} \end{bmatrix}$
- **Primera componente principal:** hallada en 7.4.3, $\mathbf{W}_{\text{pca}} = \boldsymbol{u}_1 = (\sqrt{2}/2, \sqrt{2}/2)^t$

In [4]:
```
import numpy as np; X = np.array([ [-1, -1], [0, 2], [2, 0], [3, 3] ]); S = np.cov(X.T, bias=True)
La, U = np.linalg.eigh(S); i = La.argsort()[::-1]; La = La[i]; U = U[:,i]; print(U[:, 0])
```

```
[0.70710678 0.70710678]
```

Centrado previo de datos: se suele hacer, aunque PCA es invariante a traslaciones de los datos (ya que $\mathbf{\Sigma}$ lo es)

Elección de K: si se tiene $\mathbf{\Lambda}$, puede escogerse el menor K que explique un cierto porcentaje (p.e. 90%) de la varianza total al menos

$$q_K = \frac{1}{\operatorname{tr}(\mathbf{\Sigma})} \sum_{k=1}^{K} \lambda_k \quad \text{con} \quad \operatorname{tr}(\mathbf{\Sigma}) = \sum_{d=1}^{D} \lambda_d$$

Ejercicio: halla la calidad de la proyección óptima del ejemplo

- **Valores propios de $\mathbf{\Sigma}$:** $\lambda_1 = 4$ y $\lambda_2 = 1$ (ver 7.4.3)
- **Calidad de la proyección:** $q_1 = 4/5 = 80\%$

5 Descomposición en valores singulares (SVD)

5.1 Conceptos básicos

Singular value decomposition: generaliza la EVD a matrices rectangulares, del tipo $\mathbf{A} \in \mathbf{R}^{m\times n}$

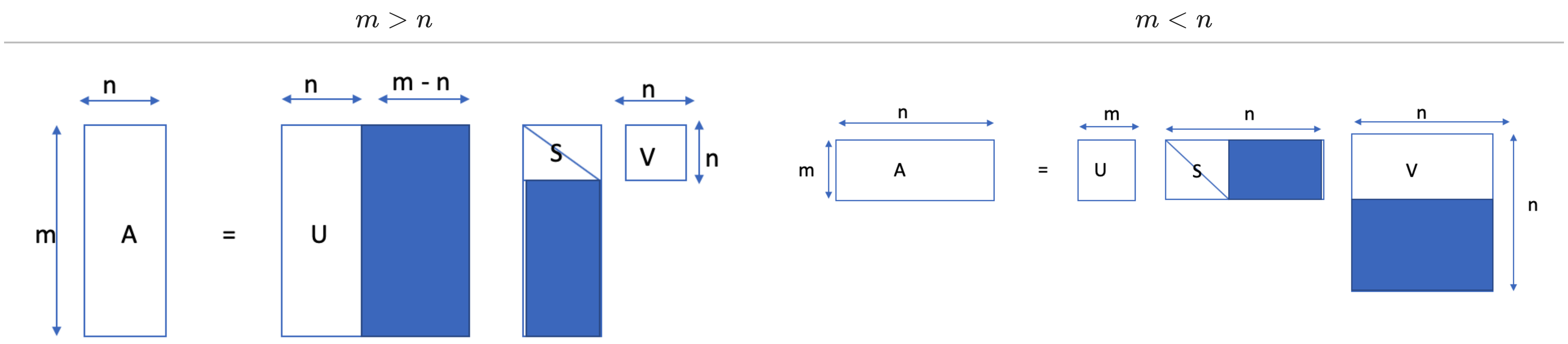
$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^t = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^t + \cdots + \sigma_r \boldsymbol{u}_r \boldsymbol{v}_r^t$$

Vectores singulares izquierdos: $\mathbf{U} \in \mathbf{R}^{m\times m}$, de columnas ortonormales, $\mathbf{U}^t\mathbf{U} = \mathbf{I}$

Valores singulares: $\mathbf{S} \in \mathbf{R}^{m\times n}$ con $r = \min(m, n)$ valores $\sigma_i \geq 0$ en su diagonal; ceros fuera

Vectores singulares derechos: $\mathbf{V} \in \mathbf{R}^{n\times n}$, de filas y columnas ortonormales, $\mathbf{V}^t\mathbf{V} = \mathbf{V}\mathbf{V}^t = \mathbf{I}$

Economy sized o thin SVD: ignora partes sombreadas



5.2 Conexión entre la SVD y la EVD

5.2.1 Matriz cuadrada real, simétrica y definida positiva

Sea $\mathbf{A} \in \mathbb{R}^{n\times n}$ real, simétrica y definida positiva, con EVD $\mathbf{A} = \mathbf{E}\mathbf{\Lambda}\mathbf{E}^t$ y SVD $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^t$

Valores singulares igual a propios: $\mathbf{S} = \mathbf{\Lambda}$

Vectores singulares izquierdos y derechos igual a propios (salvo cambios de signo): $\mathbf{U} = \mathbf{V} = \mathbf{E}$

In [1]:

```
import numpy as np; X = np.array([ [-1, -1], [0, 2], [2, 0], [3, 3] ]); A = np.cov(X.T, bias=True)
L, E = np.linalg.eigh(A); i = L.argsort()[::-1]; L = L[i]; E = E[:,i]; print("EVD:\n", L, "\n", E, "\n")
U, S, Vt = np.linalg.svd(A) # np devuelve valores singulares en orden no creciente
print("SVD:\n", U, "\n", S, "\n", Vt)
```

```
EVD:
```

```
 [4. 1.]
 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]

SVD:
 [[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]
 [4. 1.]
 [[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]
```

5.2.2 Matriz real arbitraria

Sea $\mathbf{A}\in\mathbb{R}^{m\times n}$ real con SVD $\;\mathbf{A}=\mathbf{U}\mathbf{S}\mathbf{V}^t$

Vectores y valores propios de $\mathbf{A}^t\mathbf{A}$: $\;\mathbf{V}\;$ y $\;\mathbf{D}_n=\mathbf{S}^t\mathbf{S}$

$$\mathbf{A}^t\mathbf{A}=\mathbf{V}\mathbf{S}^t\mathbf{U}^t\mathbf{U}\mathbf{S}\mathbf{V}^t=\mathbf{V}(\mathbf{S}^t\mathbf{S})\mathbf{V}^t\;\Rightarrow\;(\mathbf{A}^t\mathbf{A})\mathbf{V}=\mathbf{V}\mathbf{D}_n$$

Vectores y valores propios de $\mathbf{A}\mathbf{A}^t$: $\;\mathbf{U}\;$ y $\;\mathbf{D}_m=\mathbf{S}\mathbf{S}^t$

$$\mathbf{A}\mathbf{A}^t=\mathbf{U}\mathbf{S}\mathbf{V}^t\mathbf{V}\mathbf{S}^t\mathbf{U}^t=\mathbf{U}(\mathbf{S}\mathbf{S}^t)\mathbf{U}^t\;\Rightarrow\;(\mathbf{A}\mathbf{A}^t)\mathbf{U}=\mathbf{U}\mathbf{D}_m$$

Thin SVD: $\;\mathbf{S}\in\mathbb{R}^{r\times r}\;$ con $r=\min(m,n),\;$ por lo que $\;\mathbf{D}=\mathbf{S}^2=\mathbf{S}^t\mathbf{S}=\mathbf{S}\mathbf{S}^t$

Existencia de la SVD: la EVD no siempre existe (incluso con $\mathbf{A}$ cuadrada); la SVD sí

5.3 SVD truncada

5.3.1 SVD truncada

SVD K-truncada: $\;\hat{\mathbf{A}}_K=\mathbf{U}_K\mathbf{S}_K\mathbf{V}_K^t,\;$ aproximación de rango K de $\mathbf{A}\in\mathbb{R}^{m\times n}$ construida a partir de sus K mayores valores singulares, junto con sus K vectores singulares asociados, izquierdos y derechos

Optimalidad de la SVD K-truncada: $\;\hat{\mathbf{A}}_K\;$ es la mejor aproximación de $\mathbf{A}$ en norma Frobenius (al cuadrado), $\;\|\mathbf{A}-\hat{\mathbf{A}}_K\|_F^2$

5.3.2 PCA con la SVD

Sea $\mathbf{X}\in\mathbb{R}^{N\times D}$ una matriz de datos **centrada** de matriz de covarianzas empírica $\boldsymbol{\Sigma}=\frac{1}{N}\mathbf{X}^t\mathbf{X}\;$ (simétrica y semi-definida positiva)

SVD K-truncada de $\mathbf{X}$: $\quad\hat{\mathbf{X}}_K=\mathbf{U}_K\mathbf{S}_K\mathbf{V}_K^t$

SVD K-truncada de $\mathbf{\Sigma}$: $\hat{\mathbf{\Sigma}}_K = \mathbf{E}_K\mathbf{\Lambda}_K\mathbf{E}_K^t$ con $\mathbf{E}_K = \mathbf{V}_K$ y $\mathbf{\Lambda}_K = \frac{1}{N}\mathbf{S}_K^2$

PCA de $\mathbf{X}$ con la SVD: $\operatorname{PCA}(\mathbf{X}) = \mathbf{X}\mathbf{E}_K \approx \mathbf{U}_K\mathbf{S}_K\mathbf{V}_K^t\mathbf{V}_K = \mathbf{U}_K\mathbf{S}_K$ no requiere $\mathbf{\Sigma}$

Reconstrucción de $\mathbf{X}$ tras PCA: $\hat{\mathbf{X}}_K = \operatorname{PCA}(\mathbf{X})\mathbf{V}_K^t$

In [1]:
```
import numpy as np; X = np.array([ [-1, -1], [0, 2], [2, 0], [3, 3] ]); X = X - np.mean(X, axis=0)
U, S, Vt = np.linalg.svd(X); Xr = U[:, 0] * S[0]
print("Datos centrados:\n", X, "\nComponente principal 1:\n", Vt[:, 0], "\nDatos reducidos:\n", Xr)
print("Datos reconstruidos:\n", Xr.reshape(-1, 1) @ Vt[0, :].reshape(1, -1))
```

```
Datos centrados:
 [[-2. -2.]
 [-1.  1.]
 [ 1. -1.]
 [ 2.  2.]]
Componente principal 1:
 [0.70710678 0.70710678]
Datos reducidos:
 [-2.82842712e+00  4.31373875e-16 -3.41684101e-17  2.82842712e+00]
Datos reconstruidos:
 [[-2.00000000e+00 -2.00000000e+00]
 [ 3.05027392e-16  3.05027392e-16]
 [-2.41607145e-17 -2.41607145e-17]
 [ 2.00000000e+00  2.00000000e+00]]
```

6 Cálculo matricial

6.1 Preliminares

Cálculo vectorial: análisis real multivariable de vectores en dos o más dimensiones

Cálculo matricial: notación especializada para manejar derivadas parciales de una única función con respecto a muchas variables, o de una función multivariable con respecto a una única variable

Referencia: cálculo matricial en Wikipedia

6.1.1 Formato numerador o Jacobiana

Formato numerador o Jacobiana: primero el formato del numerador y luego el del denominador transpuesto, **si puede acomodarse**

Formato numerador o Jacobiana: primero el formato del numerador y luego el del denominador transpuesto, **si puede acodomarse en una matriz**

Tipos	Escalar $y \in \mathbb{R}^{1\times 1}$	Vector $\boldsymbol{y} \in \mathbb{R}^{m\times 1}$	Matriz $\mathbf{Y} \in \mathbb{R}^{m\times n}$
$x \in \mathbb{R}^{1\times 1}$	$\dfrac{\partial y}{\partial x}$	$\dfrac{\partial \boldsymbol{y}}{\partial x} = \begin{bmatrix} \dfrac{\partial y_1}{\partial x} \\ \vdots \\ \dfrac{\partial y_m}{\partial x} \end{bmatrix}$	$\dfrac{\partial \mathbf{Y}}{\partial x} = \begin{bmatrix} \dfrac{\partial y_{11}}{\partial x} & \cdots & \dfrac{\partial y_{1n}}{\partial x} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial y_{m1}}{\partial x} & \cdots & \dfrac{\partial y_{mn}}{\partial x} \end{bmatrix}$
$\boldsymbol{x} \in \mathbb{R}^{n\times 1}$	$\dfrac{\partial y}{\partial \boldsymbol{x}} = \begin{bmatrix} \dfrac{\partial y}{\partial x_1} & \cdots & \dfrac{\partial y}{\partial x_n} \end{bmatrix}$	$\dfrac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \begin{bmatrix} \dfrac{\partial y_1}{\partial \boldsymbol{x}} \\ \vdots \\ \dfrac{\partial y_m}{\partial \boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial y_1}{\partial x_1} & \cdots & \dfrac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial y_m}{\partial x_1} & \cdots & \dfrac{\partial y_m}{\partial x_n} \end{bmatrix}$	$\dfrac{\partial \mathbf{Y}}{\partial \boldsymbol{x}} = \begin{bmatrix} \dfrac{\partial y_{11}}{\partial \boldsymbol{x}} & \cdots & \dfrac{\partial y_{1n}}{\partial \boldsymbol{x}} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial y_{m1}}{\partial \boldsymbol{x}} & \cdots & \dfrac{\partial y_{mn}}{\partial \boldsymbol{x}} \end{bmatrix}$ 3-tensor
$\mathbf{X} \in \mathbb{R}^{p\times q}$	$\dfrac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \dfrac{\partial y}{\partial x_{11}} & \cdots & \dfrac{\partial y}{\partial x_{p1}} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial y}{\partial x_{1q}} & \cdots & \dfrac{\partial y}{\partial x_{pq}} \end{bmatrix}$	$\dfrac{\partial \boldsymbol{y}}{\partial \mathbf{X}} = \begin{bmatrix} \dfrac{\partial y_1}{\partial \mathbf{X}} \\ \vdots \\ \dfrac{\partial y_m}{\partial \mathbf{X}} \end{bmatrix}$ 3-tensor	$\dfrac{\partial \mathbf{Y}}{\partial \mathbf{X}} = \begin{bmatrix} \dfrac{\partial y_{11}}{\partial \mathbf{X}} & \cdots & \dfrac{\partial y_{1n}}{\partial \mathbf{X}} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial y_{m1}}{\partial \mathbf{X}} & \cdots & \dfrac{\partial y_{mn}}{\partial \mathbf{X}} \end{bmatrix}$ 4-tensor

6.1.2 Otros formatos

Formato denominador, gradiente o Hessiana: primero el formato del denominador y luego el del numerador transpuesto, si puede acodomarse en una matriz

- El resultado es el mismo que en formato numerador, aunque transpuesto

Origen de los dos formatos: se halla, sobre todo, en el formato que se da a $\dfrac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$

Formato mixto: consiste en escribir $\dfrac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}^t}$ y seguir el formato numerador

6.2 Identidades básicas

Identidades para escalares: a constante; u y v funciones reales de $x \in \mathbb{R}^{1\times 1}$, $\boldsymbol{x} \in \mathbb{R}^{n\times 1}$ o $\mathbf{X} \in \mathbb{R}^{p\times q}$

Identidades para escalares: a constante; u y v funciones reales de $x \in \mathbb{R}^{\dots}$, $\boldsymbol{x} \in \mathbb{R}^{\dots}$ o $\mathbf{X} \in \mathbb{R}^{\dots}$

$y \in \mathbb{R}^{1\times 1}$	$\frac{\partial y}{\partial x} \in \mathbb{R}^{1\times 1}$	$\frac{\partial y}{\partial \boldsymbol{x}} \in \mathbb{R}^{1\times n}$	$\frac{\partial y}{\partial \mathbf{X}} \in \mathbb{R}^{q\times p}$
a	0	$\mathbf{0}$	$\mathbf{0}$
au	$a\frac{\partial u}{\partial x}$	$a\frac{\partial u}{\partial \boldsymbol{x}}$	$a\frac{\partial u}{\partial \mathbf{X}}$
$u+v$	$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial x}$	$\frac{\partial u}{\partial \boldsymbol{x}} + \frac{\partial v}{\partial \boldsymbol{x}}$	$\frac{\partial u}{\partial \mathbf{X}} + \frac{\partial v}{\partial \mathbf{X}}$
uv	$u\frac{\partial v}{\partial x} + v\frac{\partial u}{\partial x}$	$u\frac{\partial v}{\partial \boldsymbol{x}} + v\frac{\partial u}{\partial \boldsymbol{x}}$	$u\frac{\partial v}{\partial \mathbf{X}} + v\frac{\partial u}{\partial \mathbf{X}}$
$g(u)$	$\frac{\partial g(u)}{\partial u}\frac{\partial u}{\partial x}$	$\frac{\partial g(u)}{\partial u}\frac{\partial u}{\partial \boldsymbol{x}}$	$\frac{\partial g(u)}{\partial u}\frac{\partial u}{\partial \mathbf{X}}$
$f(g(u))$	$\frac{\partial f(g)}{\partial g}\frac{\partial g(u)}{\partial u}\frac{\partial u}{\partial x}$	$\frac{\partial f(g)}{\partial g}\frac{\partial g(u)}{\partial u}\frac{\partial u}{\partial \boldsymbol{x}}$	$\frac{\partial f(g)}{\partial g}\frac{\partial g(u)}{\partial u}\frac{\partial u}{\partial \mathbf{X}}$

Identidades para vectores: $\boldsymbol{a}$ constante; $\boldsymbol{u}$ y $\boldsymbol{v}$ funciones vectoriales de $x \in \mathbb{R}^{1\times 1}$ o $\boldsymbol{x} \in \mathbb{R}^{n\times 1}$

$\boldsymbol{y} \in \mathbb{R}^{m\times 1}$	$\frac{\partial \boldsymbol{y}}{\partial x} \in \mathbb{R}^{m\times 1}$	$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \in \mathbb{R}^{m\times n}$
$\boldsymbol{a}$	$\mathbf{0}$	$\mathbf{0}$
$a\boldsymbol{u}$	$a\frac{\partial \boldsymbol{u}}{\partial x}$	$a\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}}$
$\boldsymbol{u}+\boldsymbol{v}$	$\frac{\partial \boldsymbol{u}}{\partial x} + \frac{\partial \boldsymbol{v}}{\partial x}$	$\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{x}}$
$\boldsymbol{g}(\boldsymbol{u})$	$\frac{\partial \boldsymbol{g}(\boldsymbol{u})}{\partial \boldsymbol{u}}\frac{\partial \boldsymbol{u}}{\partial x}$	$\frac{\partial \boldsymbol{g}(\boldsymbol{u})}{\partial \boldsymbol{u}}\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}}$
$\boldsymbol{f}(\boldsymbol{g}(\boldsymbol{u}))$	$\frac{\partial \boldsymbol{f}(\boldsymbol{g})}{\partial \boldsymbol{g}}\frac{\partial \boldsymbol{g}(\boldsymbol{u})}{\partial \boldsymbol{u}}\frac{\partial \boldsymbol{u}}{\partial x}$	$\frac{\partial \boldsymbol{f}(\boldsymbol{g})}{\partial \boldsymbol{g}}\frac{\partial \boldsymbol{g}(\boldsymbol{u})}{\partial \boldsymbol{u}}\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}}$

Identidades para matrices: a y $\mathbf{A}$ constantes; $\mathbf{U}$ y $\mathbf{V}$ funciones matriciales de x

$\mathbf{Y} \in \mathbb{R}^{m\times n}$	$\frac{\partial \mathbf{Y}}{\partial x} \in \mathbb{R}^{m\times n}$
$\mathbf{A}$	$\mathbf{0}$

$$
\begin{array}{cc}
a\mathbf{U} & a\dfrac{\partial \mathbf{U}}{\partial x} \\[2ex]
\mathbf{U}+\mathbf{V} & \dfrac{\partial \mathbf{U}}{\partial x}+\dfrac{\partial \mathbf{V}}{\partial x} \\[2ex]
\mathbf{U}\mathbf{V} & \mathbf{U}\dfrac{\partial \mathbf{V}}{\partial x}+\dfrac{\partial \mathbf{U}}{\partial x}\mathbf{V}
\end{array}
$$

6.3 Derivadas básicas

Sigmoide: $\sigma(x) = \dfrac{1}{1+e^{-x}}$

$$\frac{\partial \sigma(x)}{\partial x} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}-1}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}}\left(1-\frac{1}{1+e^{-x}}\right) = \sigma(x)(1-\sigma(x)) = \sigma(x)\sigma(-x)$$

Pérdida cuadrática: $\ell_2(y,\hat{y}) = \frac{1}{2}(y-\hat{y})^2$

$$\frac{\partial \ell_2(y,\hat{y})}{\partial \hat{y}} = \hat{y} - y$$

Log-pérdida: $\ell(\boldsymbol{y},\hat{\boldsymbol{y}}) = -\sum_c y_c \log \hat{y}_c$

$$\frac{\partial \ell(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} = \left(\frac{\partial \ell(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \hat{y}_1},\dots,\frac{\partial \ell(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \hat{y}_C}\right) = \left(-\frac{y_1}{\hat{y}_1},\dots,-\frac{y_C}{\hat{y}_C}\right)$$

6.4 Softmax

Softmax: $\boldsymbol{\mu} = \mathcal{S}(\boldsymbol{a}) = \left(\dfrac{e^{a_1}}{\sum_{\tilde{c}} e^{a_{\tilde{c}}}},\dots,\dfrac{e^{a_C}}{\sum_{\tilde{c}} e^{a_{\tilde{c}}}}\right)^t$

$$\frac{\partial \mu_i}{\partial a_j} = e^{a_i}\frac{\partial(\sum_{\tilde{c}} e^{a_{\tilde{c}}})^{-1}}{\partial a_j} + \frac{1}{\sum_{\tilde{c}} e^{a_{\tilde{c}}}}\frac{\partial e^{a_i}}{\partial a_j} = -\frac{e^{a_i}e^{a_j}}{(\sum_{\tilde{c}} e^{a_{\tilde{c}}})^2} + \frac{e^{a_i}\mathbb{I}(i=j)}{\sum_{\tilde{c}} e^{a_{\tilde{c}}}} = \mu_i(\mathbb{I}(i=j)-\mu_j)$$

$$\frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{a}} = \begin{pmatrix} \frac{\partial \mu_1}{\partial \boldsymbol{a}} \\ \frac{\partial \mu_2}{\partial \boldsymbol{a}} \\ \vdots \end{pmatrix} = \begin{pmatrix} \frac{\partial \mu_1}{\partial a_1} & \frac{\partial \mu_1}{\partial a_2} & \cdots & \frac{\partial \mu_1}{\partial a_C} \\ \frac{\partial \mu_2}{\partial a_1} & \frac{\partial \mu_2}{\partial a_2} & \cdots & \frac{\partial \mu_2}{\partial a_C} \\ \vdots & \vdots & \ddots & \vdots \end{pmatrix} = \begin{pmatrix} \mu_1(1-\mu_1) & -\mu_1\mu_2 & \cdots & -\mu_1\mu_C \\ -\mu_2\mu_1 & \mu_2(1-\mu_2) & \cdots & -\mu_2\mu_C \\ \vdots & \vdots & \ddots & \vdots \end{pmatrix} = \operatorname{diag}(\boldsymbol{\mu}) - \boldsymbol{\mu}^t\boldsymbol{\mu}$$

$$\frac{\partial \mu_C}{\partial \boldsymbol{a}} \quad \frac{\partial \mu_C}{\partial a_1} \quad \frac{\partial \mu_C}{\partial a_2} \quad \cdots \quad \frac{\partial \mu_C}{\partial a_C} \quad -\mu_C\mu_1 \quad -\mu_C\mu_2 \quad \cdots \quad \mu_C(1-\mu_C)$$

$$\frac{\partial \log \mu_i}{\partial a_j} = \frac{\partial \log \mu_i}{\partial \mu_i}\frac{\partial \mu_i}{\partial a_j} = \frac{1}{\mu_i}\mu_i(\mathbb{I}(i=j)-\mu_j) = \mathbb{I}(i=j)-\mu_j$$

$$\frac{\partial \log \boldsymbol{\mu}}{\partial \boldsymbol{a}} = \begin{pmatrix} \frac{\partial \log \mu_1}{\partial \boldsymbol{a}} \\ \frac{\partial \log \mu_2}{\partial \boldsymbol{a}} \\ \vdots \\ \frac{\partial \log \mu_C}{\partial \boldsymbol{a}} \end{pmatrix} = \begin{pmatrix} \frac{\partial \log \mu_1}{\partial a_1} & \frac{\partial \log \mu_1}{\partial a_2} & \cdots & \frac{\partial \log \mu_1}{\partial a_C} \\ \frac{\partial \log \mu_2}{\partial a_1} & \frac{\partial \log \mu_2}{\partial a_2} & \cdots & \frac{\partial \log \mu_2}{\partial a_C} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \log \mu_C}{\partial a_1} & \frac{\partial \log \mu_C}{\partial a_2} & \cdots & \frac{\partial \log \mu_C}{\partial a_C} \end{pmatrix} = \begin{pmatrix} 1-\mu_1 & -\mu_2 & \cdots & -\mu_C \\ -\mu_1 & 1-\mu_2 & \cdots & -\mu_C \\ \vdots & \vdots & \ddots & \vdots \\ -\mu_1 & -\mu_2 & \cdots & 1-\mu_C \end{pmatrix} = \operatorname{diag}(\mathbf{1}_C) - \mathbf{1}_C\boldsymbol{\mu}^t$$

6.5 Transformaciones lineales

Transformaciones lineales: $\boldsymbol{x} \in \mathbb{R}^n,\ \boldsymbol{y} = \mathbf{W}\boldsymbol{x},\ \mathbf{W} \in \mathbb{R}^{m\times n},\ \boldsymbol{u} \in \mathbb{R}^m$ constante

Derivada de $\boldsymbol{y}$ con respecto a $\boldsymbol{x}$:

$$\frac{\partial y_i}{\partial x_j} = \frac{\partial}{\partial x_j}\sum_k W_{ik}x_k = \sum_k W_{ik}\frac{\partial x_k}{\partial x_j} = W_{ij}$$

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \begin{pmatrix} \frac{\partial y_1}{\partial \boldsymbol{x}} \\ \vdots \\ \frac{\partial y_m}{\partial \boldsymbol{x}} \end{pmatrix} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix} = \mathbf{W}$$

Derivadas de $\boldsymbol{y}$ con respecto a $\mathbf{W}$:

$$\frac{\partial y_k}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}}\sum_l W_{kl}x_l = \sum_l x_l\frac{\partial W_{kl}}{\partial W_{ij}} = x_j\mathbb{I}(k=i)$$

$$\frac{\partial \boldsymbol{y}}{\partial W_{ij}} = \begin{pmatrix} \frac{\partial y_1}{\partial W_{ij}} \\ \vdots \\ \frac{\partial y_m}{\partial W_{ij}} \end{pmatrix} = x_j\,\text{one-hot}(i)$$

$$\frac{\partial y_k}{\partial \mathbf{W}} = \begin{pmatrix} \frac{\partial y_k}{\partial W_{11}} & \frac{\partial y_k}{\partial W_{21}} & \cdots & \frac{\partial y_k}{\partial W_{m1}} \\ \frac{\partial y_k}{\partial W_{12}} & \frac{\partial y_k}{\partial W_{22}} & \cdots & \frac{\partial y_k}{\partial W_{m2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_k}{\partial W_{1n}} & \frac{\partial y_k}{\partial W_{2n}} & \cdots & \frac{\partial y_k}{\partial W_{mn}} \end{pmatrix} = \begin{pmatrix} x_1\mathbb{I}(k=1) & x_1\mathbb{I}(k=2) & \cdots & x_1\mathbb{I}(k=m) \\ x_2\mathbb{I}(k=1) & x_2\mathbb{I}(k=2) & \cdots & x_2\mathbb{I}(k=m) \\ \vdots & \vdots & \ddots & \vdots \\ x_n\mathbb{I}(k=1) & x_n\mathbb{I}(k=2) & \cdots & x_n\mathbb{I}(k=m) \end{pmatrix} = \boldsymbol{x}\,\text{one-hot}(k)^t \in \mathbb{R}^{n\times m}$$

$$\boldsymbol{u}^t\frac{\partial \boldsymbol{y}}{\partial \mathbf{W}} = \frac{\partial \boldsymbol{u}^t\boldsymbol{y}}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}}\sum_{k=1}^m u_k y_k = \sum_{k=1}^m u_k\frac{\partial y_k}{\partial \mathbf{W}} = \sum_{k=1}^m u_k\boldsymbol{x}\,\text{one-hot}(k)^t = \boldsymbol{x}\sum_{k=1}^m u_k\,\text{one-hot}(k)^t = \boldsymbol{x}\boldsymbol{u}^t$$

6.6 Regresión logística

Datos: $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}, \quad \boldsymbol{x}_n \in \mathbb{R}^D, \quad \boldsymbol{y}_n \in \{0,1\}^C$ one-hot

Objetivo a derivar:

$$\mathcal{L}(\mathbf{W}) = \frac{1}{N}\sum_{n=1}^N \ell(\boldsymbol{y}_n, \boldsymbol{\mu}_n), \quad \ell(\boldsymbol{y}_n, \boldsymbol{\mu}_n) = -\sum_c y_{nc}\log\mu_{nc}, \quad \boldsymbol{\mu}_n = \mathcal{S}(\boldsymbol{a}_n), \quad \boldsymbol{a}_n = \mathbf{W}^t\boldsymbol{x}_n, \quad \mathbf{W} \in \mathbb{R}^{D\times C}$$

Derivada del objetivo respecto a $\mathbf{W}$:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{W}} &= \frac{1}{N}\sum_{n=1}^N \frac{\partial \ell(\boldsymbol{y}_n, \boldsymbol{\mu}_n)}{\partial \mathbf{W}} = \frac{1}{N}\sum_{n=1}^N \frac{\partial \ell(\boldsymbol{y}_n, \boldsymbol{\mu}_n)}{\partial \log\boldsymbol{\mu}_n}\frac{\partial \log\boldsymbol{\mu}_n}{\partial \boldsymbol{a}_n}\frac{\partial \boldsymbol{a}_n}{\partial \mathbf{W}} \\
\frac{\partial \ell(\boldsymbol{y}_n, \boldsymbol{\mu}_n)}{\partial \log\boldsymbol{\mu}_n} &= \left(\frac{\partial \ell(\boldsymbol{y}_n, \boldsymbol{\mu}_n)}{\partial \log\mu_{n1}}, \dots, \frac{\partial \ell(\boldsymbol{y}_n, \boldsymbol{\mu}_n)}{\partial \log\mu_{nC}}\right) = -\boldsymbol{y}_n^t \\
\frac{\partial \log\boldsymbol{\mu}_n}{\partial \boldsymbol{a}_n} &= \operatorname{diag}(\mathbf{1}_C) - \mathbf{1}_C\boldsymbol{\mu}_n^t \\
\frac{\partial \ell(\boldsymbol{y}_n, \boldsymbol{\mu}_n)}{\partial \log\boldsymbol{\mu}_n}\frac{\partial \log\boldsymbol{\mu}_n}{\partial \boldsymbol{a}_n} &= \boldsymbol{y}_n^t(\mathbf{1}_C\boldsymbol{\mu}_n^t - \operatorname{diag}(\mathbf{1}_C)) \overset{\boldsymbol{y}_n\text{ one-hot}}{=} (\boldsymbol{\mu}_n - \boldsymbol{y}_n)^t \overset{\text{def}}{=} \boldsymbol{u}_n^t \\
\frac{\partial \mathcal{L}}{\partial \mathbf{W}} &= \frac{1}{N}\sum_{n=1}^N \boldsymbol{u}_n^t\frac{\partial \boldsymbol{a}_n}{\partial \mathbf{W}} = \frac{1}{N}\sum_{n=1}^N \boldsymbol{x}_n\boldsymbol{u}_n^t = \frac{1}{N}\sum_{n=1}^N \boldsymbol{x}_n(\boldsymbol{\mu}_n - \boldsymbol{y}_n)^t
\end{aligned}$$