



ECOLE  
POLYTECHNIQUE  
DE BRUXELLES

# The proper orthogonal decomposition: a linear autoencoder

# Why do we need data compression

**Data compression** is the process of **reducing** the **dimension** of a **digital signal** (audio, photo, video, etc.).

# Why do we need data compression

**Data compression** is the process of **reducing** the **dimension** of a **digital signal** (audio, photo, video, etc.).

The compressed content can be **reconstructed** either:

- perfectly (**lossless** compression),
- with minor reconstruction error (**lossy** compression).

# Why do we need data compression

**Data compression** is the process of **reducing** the **dimension** of a **digital signal** (audio, photo, video, etc.).

The compressed content can be **reconstructed** either:

- perfectly (**lossless** compression),
- with minor reconstruction error (**lossy** compression).

Data compression is widely used in computer science to reduce the size digital images.

For example, an uncompressed 24-bit image in 4K resolution would take up around 30 MB.

# Compression is important in physics as well

Optical **diagnostics** produce high-resolution **images** of physical systems (PIV for example).

# Compression is important in physics as well

Optical **diagnostics** produce high-resolution **images** of physical systems (PIV for example).

Numerical **simulations** (CFD) approximate the **solution** of non-linear **PDE** on computational **grids** with millions or billions of cells.

# Compression is important in physics as well

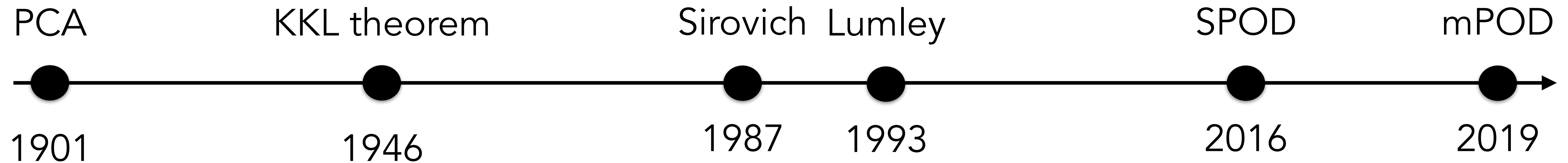
Optical **diagnostics** produce high-resolution **images** of physical systems (PIV for example).

Numerical **simulations** (CFD) approximate the **solution** of non-linear **PDE** on computational **grids** with millions or billions of cells.

**Data compression** can be used to:

- Find **physical insights**,
- **Accelerate** numerical **simulations**,
- Build **surrogate models**.

# Timeline



The mathematical theory is **old** and well **established**.

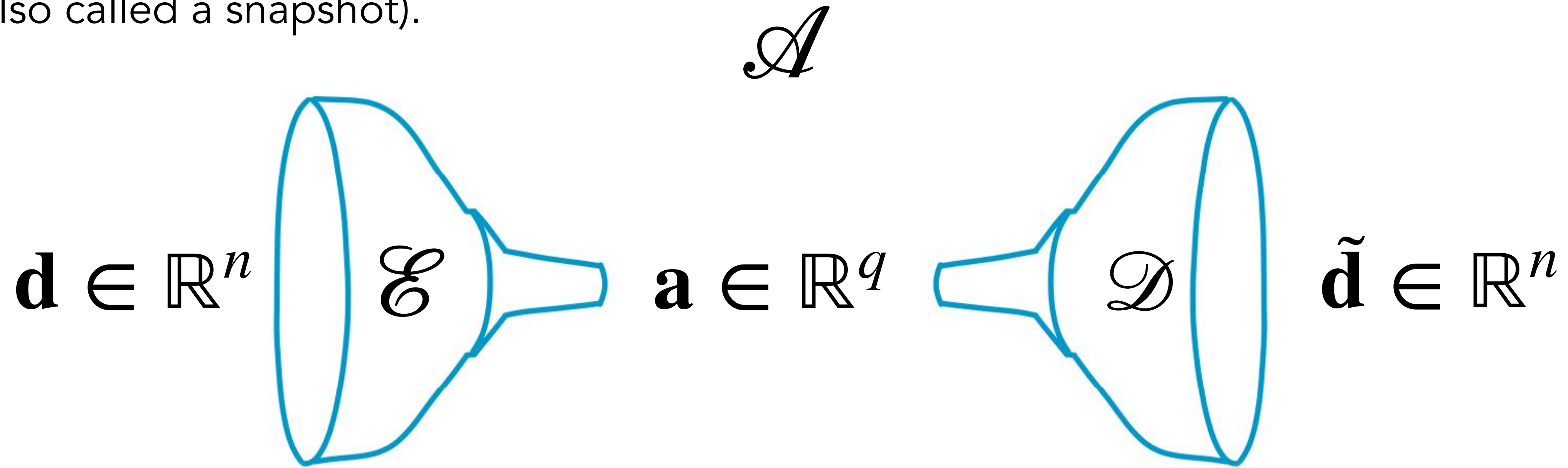
Practical **applications** were implemented when the **computational power** started to be **available**.

With the renewed interest in machine learning, these techniques have been extended/refined.



# We use an autoencoder to compress data

An autoencoder  $\mathcal{A}(\mathbf{d})$  is a mathematical machine that encodes and decodes a piece of data  $\mathbf{d}$  (this is also called a snapshot).



It is made of an encoding function  $\mathcal{E}(\mathbf{d})$  and a decoding function  $\mathcal{D}(\mathbf{a})$ .

The objective is to compress the dimensionality (reducing  $q$ ) as much as possible while losing as little information as possible ( $\tilde{\mathbf{d}} \approx \mathbf{d}$ ).

# Linear autoencoder

We want to find a linear autencoder, so our encoding and decoding functions will be matrices:

$$\mathcal{E}(\mathbf{d}) = \mathbf{E}\mathbf{d} \quad \mathcal{D}(\mathbf{a}) = \mathbf{\Phi}\mathbf{a}$$

$(q \times n) \qquad (n \times q)$

# Linear autoencoder

We want to find a linear autencoder, so our encoding and decoding functions will be matrices:

$$\mathcal{E}(\mathbf{d}) = \mathbf{E}\mathbf{d} \quad \mathcal{D}(\mathbf{a}) = \mathbf{\Phi}\mathbf{a}$$

$(q \times n) \qquad (n \times q)$

Implicitly, we are assuming that each vector  $\mathbf{d}$  can be written as a linear combination of  $q$  basis vectors  $[\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_q]$ :

$$\mathbf{d} = a_1\mathbf{b}_1 + a_2\mathbf{b}_2 + \dots = \sum_{i=1}^q a_i\mathbf{b}_i = \mathbf{B}\mathbf{a}$$

# Linear autoencoder

We want to find a linear autencoder, so our encoding and decoding functions will be matrices:

$$\mathcal{E}(\mathbf{d}) = \mathbf{E}\mathbf{d} \quad \mathcal{D}(\mathbf{a}) = \mathbf{\Phi}\mathbf{a}$$

$(q \times n) \qquad \qquad (n \times q)$

Implicitly, we are assuming that each vector  $\mathbf{d}$  can be written as a linear combination of  $q$  basis vectors  $[\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_q]$ :

$$\mathbf{d} = a_1\mathbf{b}_1 + a_2\mathbf{b}_2 + \dots = \sum_{i=1}^q a_i\mathbf{b}_i = \mathbf{B}\mathbf{a}$$

$$\mathbf{B} \in \mathbb{R}^{n \times q} = \begin{array}{c} \begin{array}{c} \updownarrow n \end{array} \left[ \begin{array}{ccc} | & | & | \\ \mathbf{b}_1 & \mathbf{b}_2 & \dots & \mathbf{b}_q \\ | & | & | \end{array} \right] \begin{array}{c} \leftarrow q \rightarrow \end{array} \end{array}$$

# The canonical basis

The canonical basis is composed of a set of vectors, each of whose components are all zero, except one that equals 1. It is the basis we use conventionally to represent vectors.

For example, in 3D space the canonical basis is represented as:

$$\mathbf{e}_x = (1,0,0) \quad \mathbf{e}_y = (0,1,0) \quad \mathbf{e}_z = (0,0,1)$$

Each 3D vector can be naturally represented as a linear combinations of the basis, for example:

$$(4,3,7) = 4\mathbf{e}_x + 3\mathbf{e}_y + 7\mathbf{e}_z$$

But this is not the only basis we can use (there is an infinite number of basis for any given space). The basis that we find with the POD is optimised for the dataset that we have.

# Relationship between decoder and encoder

We start from the **decoder**:

$$\tilde{\mathbf{d}} = \mathbf{B}\mathbf{a} = \Phi\mathbf{a}$$

# Relationship between decoder and encoder

We start from the **decoder**:

$$\tilde{\mathbf{d}} = \mathbf{B}\mathbf{a} = \Phi\mathbf{a}$$

The **encoder** will be the “**inverse**” of the **encoder** in a least squares sense

$$\mathbf{a} = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\tilde{\mathbf{d}} = \mathbf{E}\tilde{\mathbf{d}}$$

# Relationship between decoder and encoder

We start from the **decoder**:

$$\tilde{\mathbf{d}} = \mathbf{B}\mathbf{a} = \Phi\mathbf{a}$$

The **encoder** will be the “**inverse**” of the **encoder** in a least squares sense

$$\mathbf{a} = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\tilde{\mathbf{d}} = \mathbf{E}\tilde{\mathbf{d}}$$

we would like very much to **avoid** computing the **inverse** so we assume that  $\mathbf{B}$  is an **orthonormal** basis:

$$\mathbf{B}^T\mathbf{B} = \mathbf{I}$$



# Relationship between decoder and encoder

We start from the **decoder**:

$$\tilde{\mathbf{d}} = \mathbf{B}\mathbf{a} = \Phi\mathbf{a}$$

The **encoder** will be the “**inverse**” of the **encoder** in a least squares sense

$$\mathbf{a} = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{d} = \mathbf{E}\mathbf{d}$$

we would like very much to **avoid** computing the **inverse** so we assume that  $\mathbf{B}$  is an **orthonormal** basis:

$$\mathbf{B}^T\mathbf{B} = \mathbf{I}$$

The **autoencoder** is then:

$$\tilde{\mathbf{d}} = \mathbf{B}\mathbf{B}^T\mathbf{d}$$

# Collecting data

Our autoencoder will be optimized for our dataset, so we need to collect a set of samples  $[\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_m]$ , so that we can build our data matrix:

$$\mathbf{D} \in \mathbb{R}^{n \times m} = \begin{array}{c} \begin{array}{c} \updownarrow \\ n \end{array} \left[ \begin{array}{ccc} | & | & | \\ \mathbf{d}_1 & \mathbf{d}_2 & \dots & \mathbf{d}_m \\ | & | & | \end{array} \right] \begin{array}{c} \leftarrow m \rightarrow \end{array} \end{array}$$

Tensors (like images or 3D grids) have to be reshaped into 1D snapshots  $\mathbf{d}_j$ .

# Objective function

The **goal** is to have the **reconstructed** data  $\tilde{\mathbf{D}}$  as **close** as possible to the **original** data  $\mathbf{D}$  (in a least-squares sense):

$$\begin{aligned} \Phi = \operatorname{argmin} \quad & ||\mathbf{D} - \tilde{\mathbf{D}}||_2^2 = ||\mathbf{D} - \mathbf{B}\mathbf{B}^T\mathbf{D}||_2^2 \\ \text{s.t.} \quad & \mathbf{B}^T\mathbf{B} = \mathbf{I} \end{aligned}$$

# Objective function

The **goal** is to have the **reconstructed** data  $\tilde{\mathbf{D}}$  as **close** as possible to the **original** data  $\mathbf{D}$  (in a least-squares sense):

$$\begin{aligned}\Phi &= \operatorname{argmin} ||\mathbf{D} - \tilde{\mathbf{D}}||_2^2 = ||\mathbf{D} - \mathbf{B}\mathbf{B}^T\mathbf{D}||_2^2 \\ \text{s.t. } \mathbf{B}^T\mathbf{B} &= \mathbf{I}\end{aligned}$$

This optimization problem is **guaranteed** to have a single **optimum** point, which we can obtain by solving this eigenvalue problem:

$$\underset{(n \times n)}{\mathbf{C}} = \underset{(n \times n)}{\mathbf{D}\mathbf{D}^T} = \underset{(n \times n)}{\Phi\mathbf{L}\Phi^T}$$

where  $\mathbf{C} = \mathbf{D}\mathbf{D}^T$  is the **covariance** matrix between the **rows** of  $\mathbf{D}$  (often called spatial covariance matrix because the rows of  $\mathbf{D}$  represent points in space).

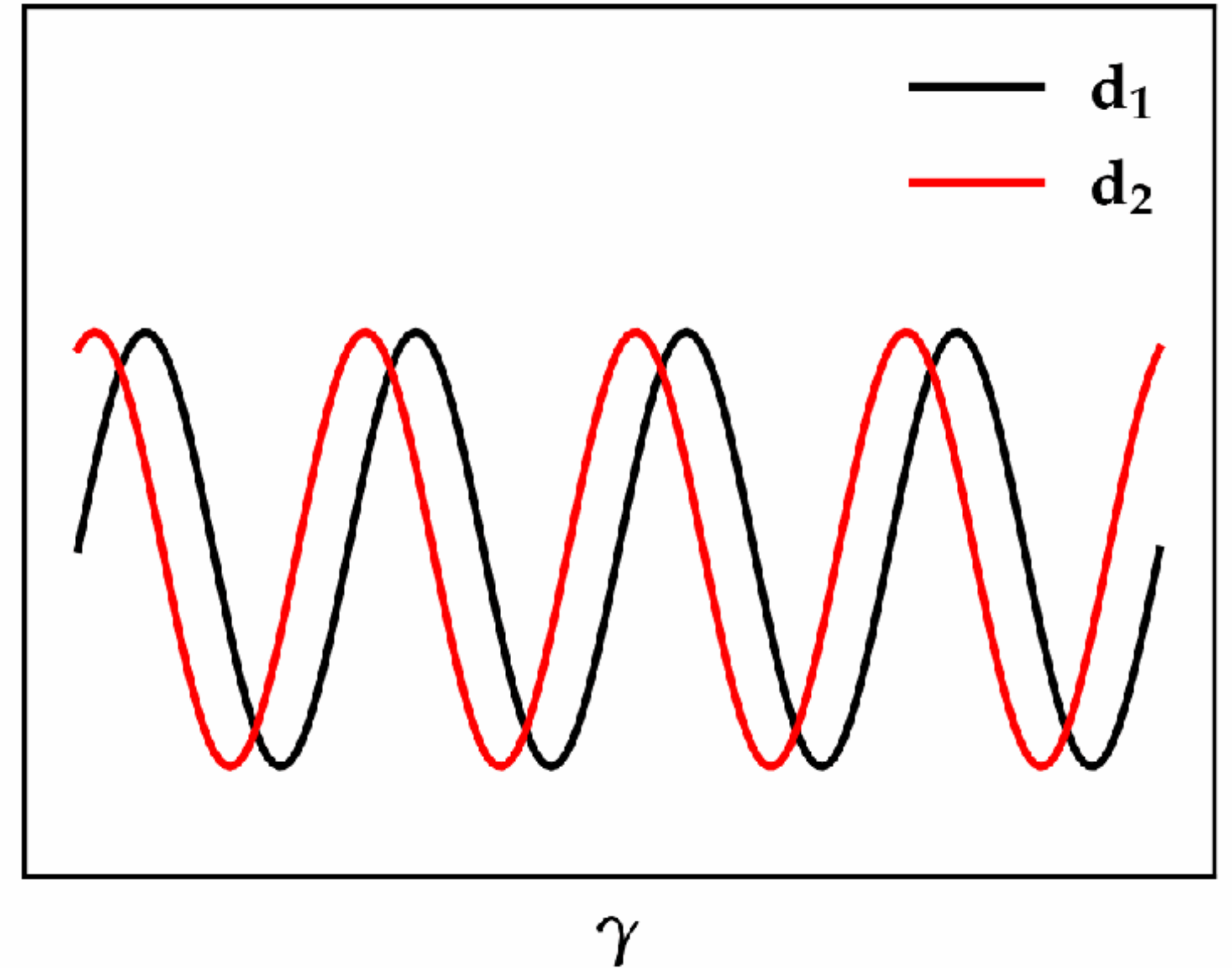
# Intuitive explanation

Why is the covariance matrix involved in the dimensionality reduction?

The **covariance** between two points represents a measure of **shared information**.

If two points share the same information, we can eliminate one without affecting the total information content.

$$c_{1,2} = \mathbf{d}_1 \mathbf{d}_2^T = 38$$



# Practical problem

The **dimensionality**  $n$  of  $\mathbf{D}$  is **big** (this is the reason why we are compressing in the first place), so this means that the **dimensions** of  $\mathbf{C} = \mathbf{D}\mathbf{D}^T$  are **huge**.

For example, for a set of low-res images  $n = 64 \times 64 = 4096$  the covariance matrix would contain  $\sim 17$  million elements.

We can tackle the problem from the other side considering the **column-wise covariance** matrix:

$$\underset{(m \times m)}{\mathbf{K}} = \underset{(m \times m)}{\mathbf{D}^T \mathbf{D}} = \mathbf{\Psi} \mathbf{L} \mathbf{\Psi}^T$$

Then, we compute the decoder as a linear combination of original snapshots:

$$\begin{aligned} \mathbf{Z} &= \mathbf{D}\mathbf{\Psi} \\ \mathbf{\Phi} &= \mathbf{Z}\mathbf{\Sigma}^{-1} \end{aligned}$$

# The singular value decomposition

We have that  $\mathbf{D} = \mathbf{Z}\mathbf{\Psi}^T$  and  $\mathbf{Z} = \mathbf{\Phi}\mathbf{\Sigma}$ , so putting everything together we obtain the **singular value decomposition**:

$$\mathbf{D} = \underset{(n \times m)}{\mathbf{\Phi}} \underset{(n \times m)}{\mathbf{\Sigma}} \underset{(m \times m)}{\mathbf{\Psi}^T}$$

where:

- $\mathbf{\Phi}$  is an **orthonormal basis** for the **row**-space of  $\mathbf{D}$  (spatial basis),
- $\mathbf{\Psi}$  is an **orthonormal basis** for the **column**-space of  $\mathbf{D}$  (temporal or parametric basis),
- $\mathbf{\Sigma}$  is a matrix containing the **singular values** (norm of the projections of  $\mathbf{D}$ ).

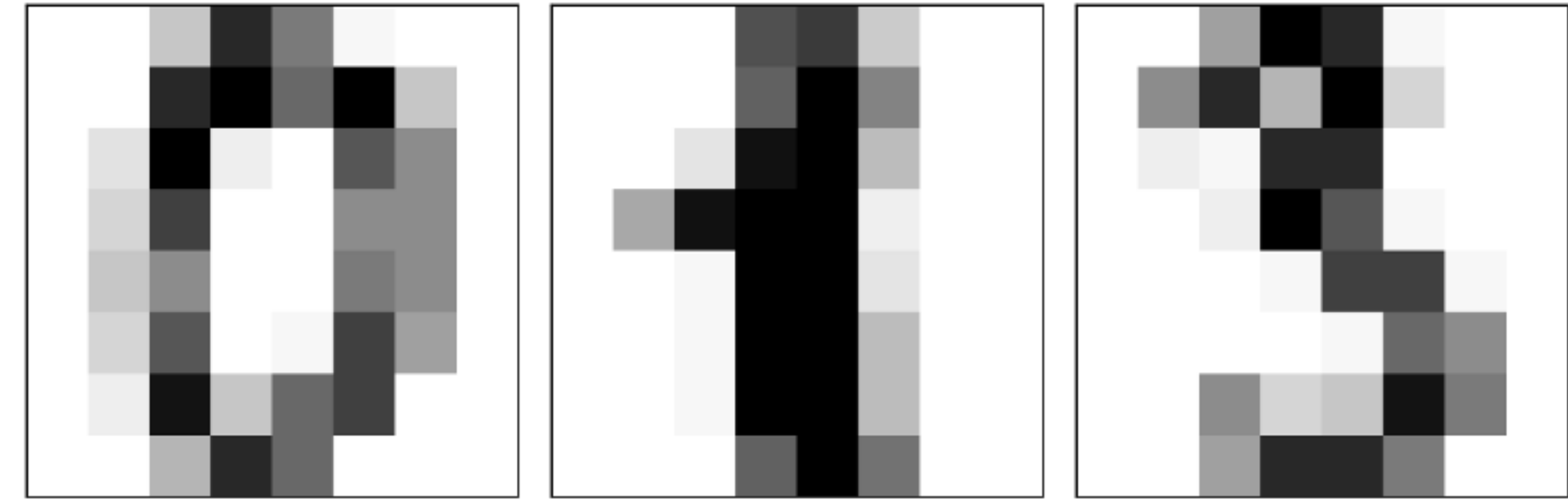
After the  $m$ -th row, the singular values are all zeros so we can write the compact SVD:

$$\mathbf{D} = \underset{(n \times m)}{\mathbf{\Phi}} \underset{(m \times m)}{\mathbf{\Sigma}} \underset{(m \times m)}{\mathbf{\Psi}^T}$$

# An example: compressing digits

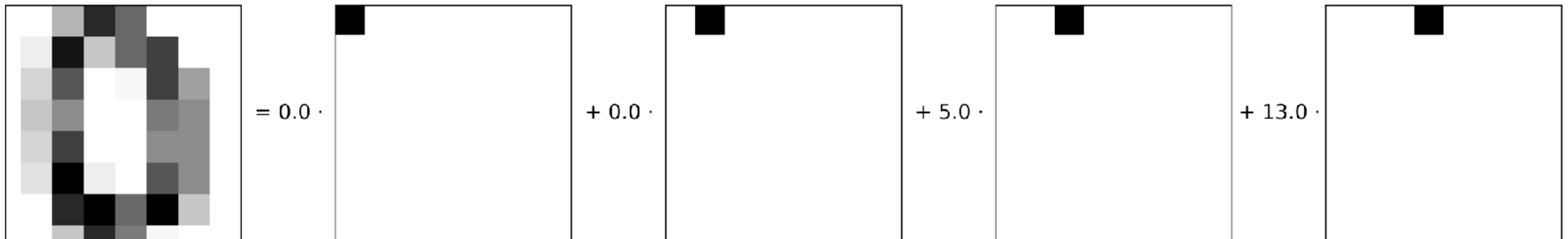
We have a set of pictures showing handwritten digits that we want to compress.

Each image has size  $8 \times 8$  and we have 1797 images.



In the canonical basis, the representation of each image is given by:

$$\mathbf{d} = d_1 \mathbf{e}_1 + d_2 \mathbf{e}_2 + d_3 \mathbf{e}_3 + \cdots = \sum_{i=1}^n d_i \mathbf{e}_i$$

A diagram illustrating the representation of a handwritten digit image as a linear combination of basis vectors. On the left is an 8x8 grayscale image of a handwritten digit. To its right is an equals sign followed by four terms, each consisting of a coefficient, a dot, and an 8x8 image. The first term is  $= 0.0 \cdot$  followed by an image with a single black pixel at the top-left corner. The second term is  $+ 0.0 \cdot$  followed by an image with a single black pixel at the top-left corner. The third term is  $+ 5.0 \cdot$  followed by an image with a single black pixel at the top-left corner. The fourth term is  $+ 13.0 \cdot$  followed by an image with a single black pixel at the top-left corner.



# An example: compressing digits

To apply the POD, first we assemble the data matrix, which will have dimensions  $64 \times 1797$ .

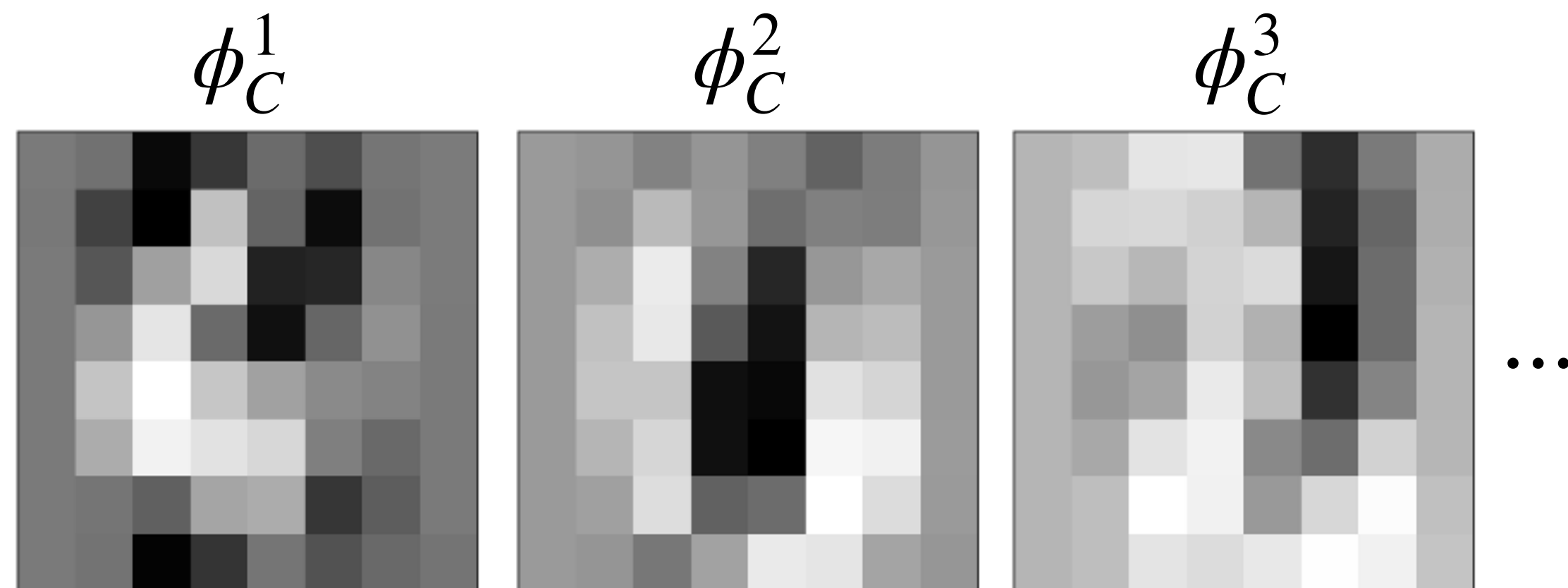
$$\mathbf{D} \in \mathbb{R}^{n \times m} = \begin{matrix} & \xleftarrow{m} & & \\ \begin{matrix} \uparrow \\ n \\ \downarrow \end{matrix} & \left[ \begin{array}{ccc|c} | & | & & | \\ \mathbf{d}_1 & \mathbf{d}_2 & \cdots & \mathbf{d}_m \\ | & | & & | \end{array} \right] & \end{matrix}$$

We also need to center the dataset, i.e. we need to subtract the average value across the rows:

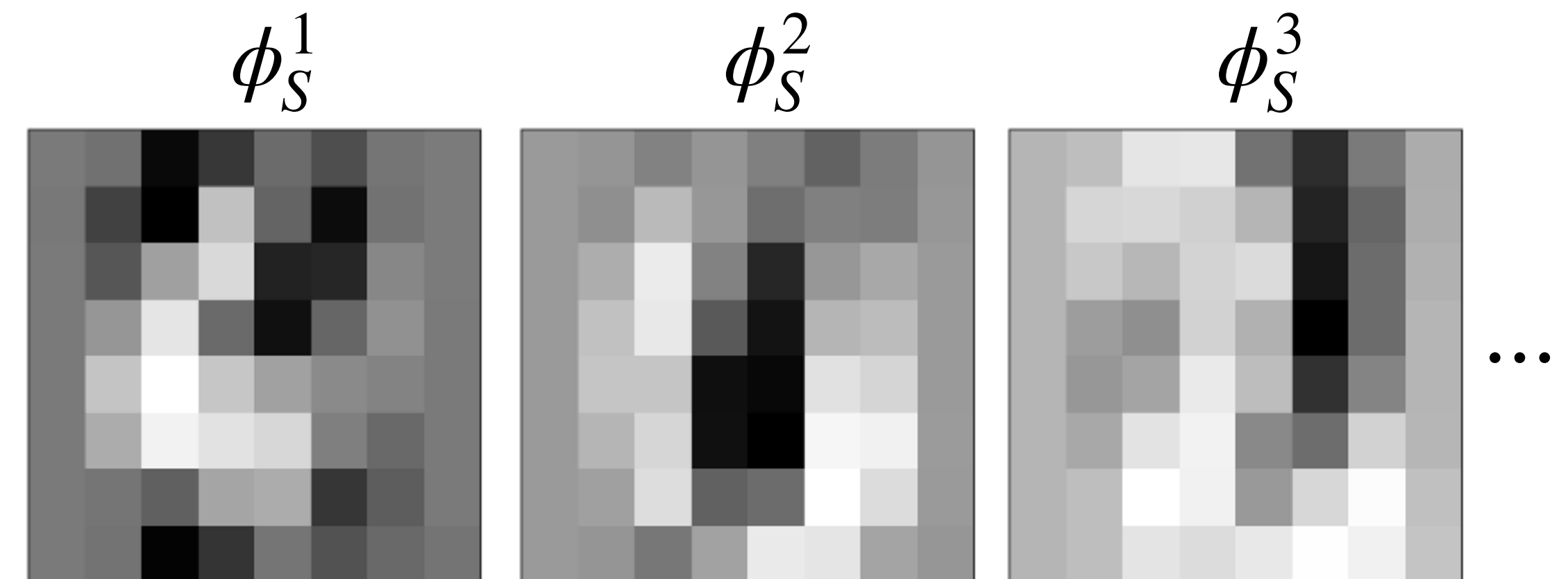
$$\mathbf{D}_0 = \mathbf{D} - \mathbf{D}_\mu$$

In this case we can compare the results that we obtain from the decomposition of the spatial covariance matrix and the singular value decomposition:

$$\mathbf{C} = \mathbf{D}_0 \mathbf{D}_0^T = \mathbf{\Phi}_C \mathbf{L} \mathbf{\Phi}_C^T$$



$$\mathbf{D}_0 = \mathbf{\Phi}_S \mathbf{\Sigma} \mathbf{\Psi}^T$$



# An example: compressing digits

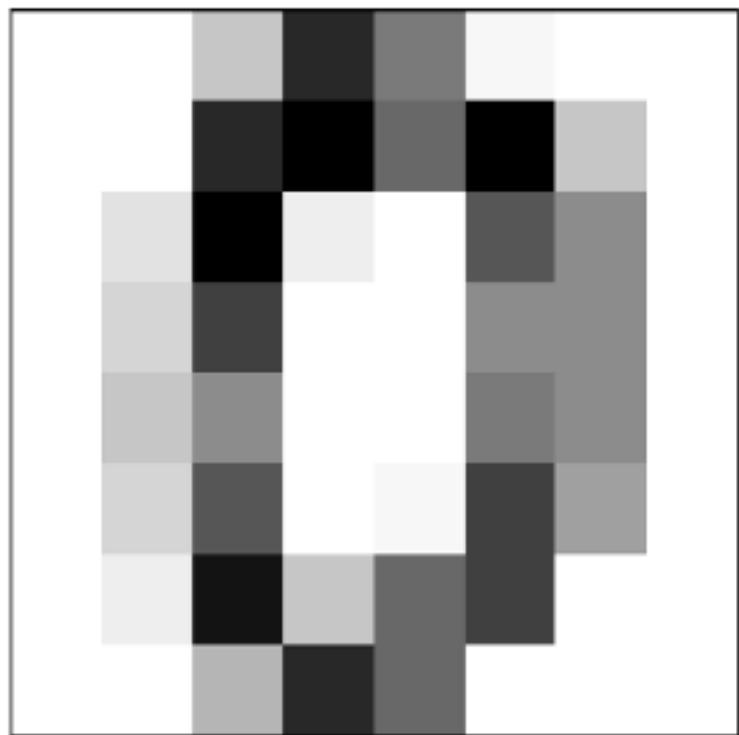
Now that we have a new basis we can write each snapshot as:

$$\mathbf{d} = \mathbf{d}_\mu + a_1\boldsymbol{\phi}_1 + a_2\boldsymbol{\phi}_2 + a_3\boldsymbol{\phi}_3 + \cdots = \sum_{i=1}^m a_i\boldsymbol{\phi}_i$$

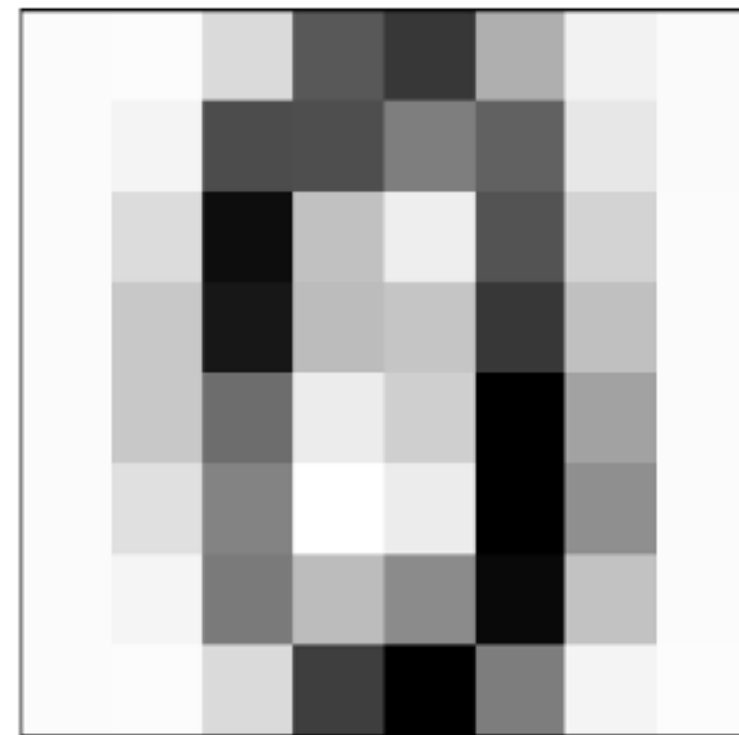
If we use all the basis the reconstruction is perfect (this is just a change of basis).

But if we use a fraction of those, we can still get a good reconstruction.

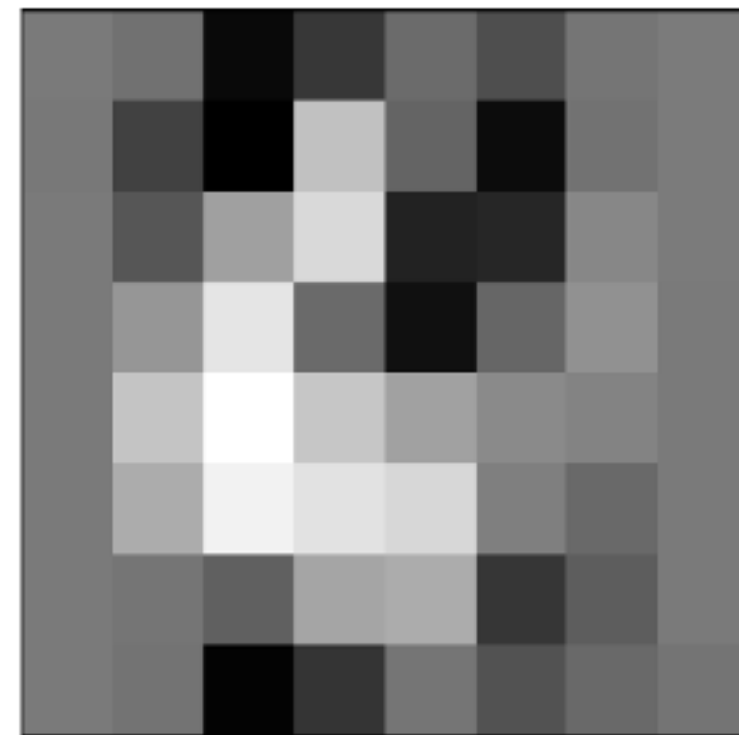
Original



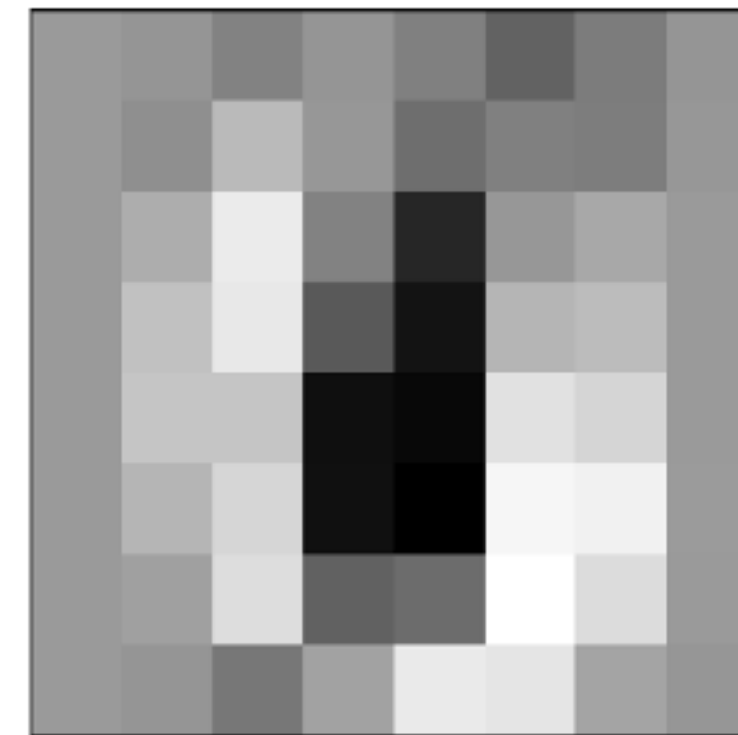
Reconstructed



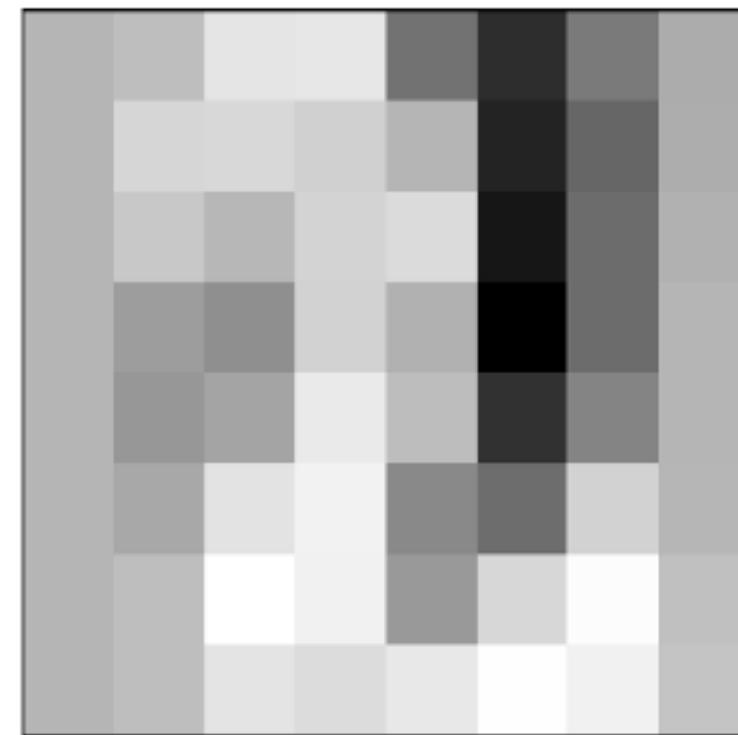
= 1.3



-21.3



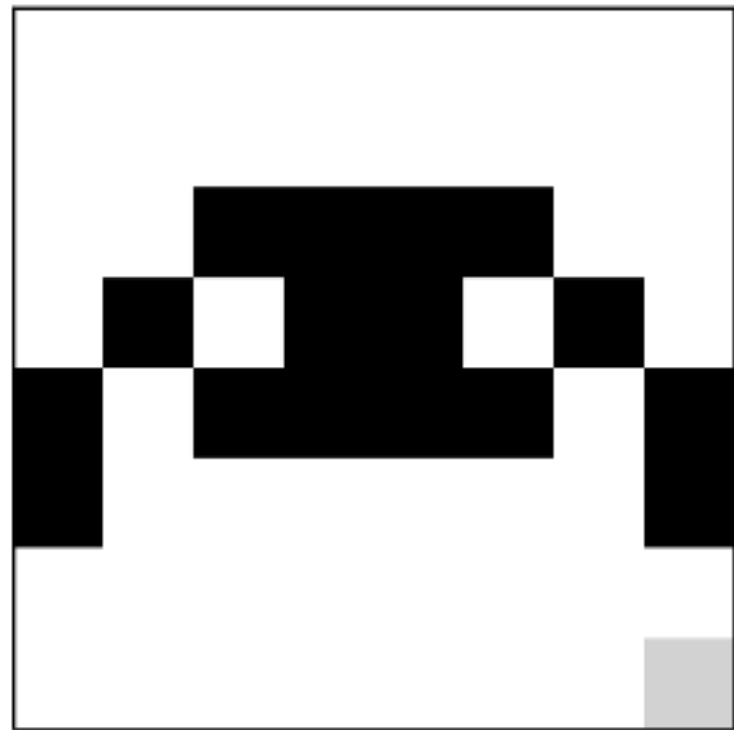
+9.5



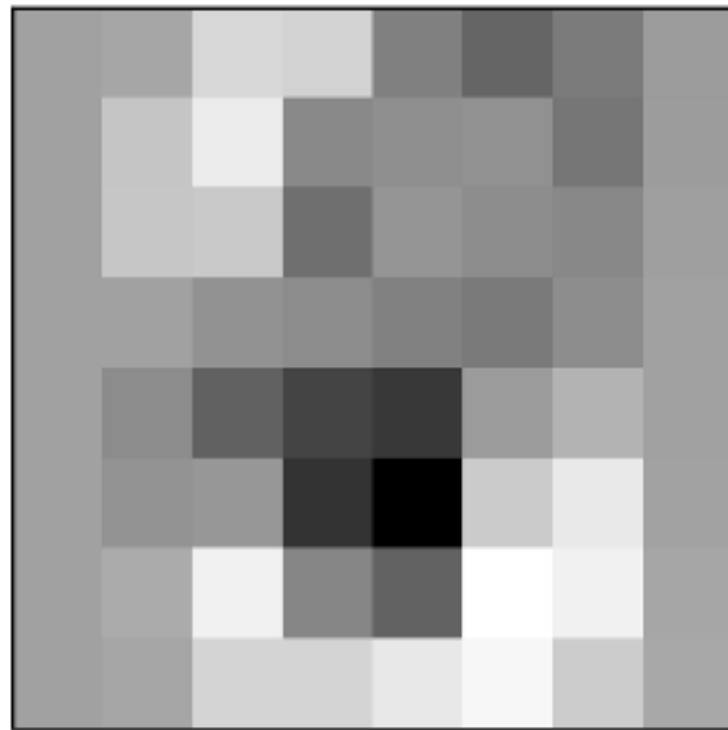
# An example: compressing digits

The **basis** that we have obtained is still a **universal** basis, i.e. we can use it to represent everything other than digits (in the 64D space).

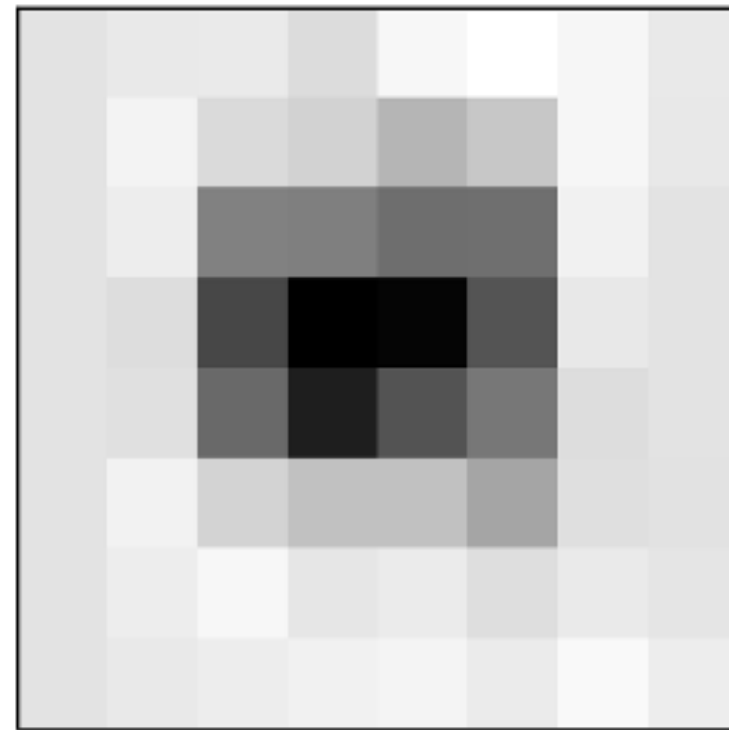
Original



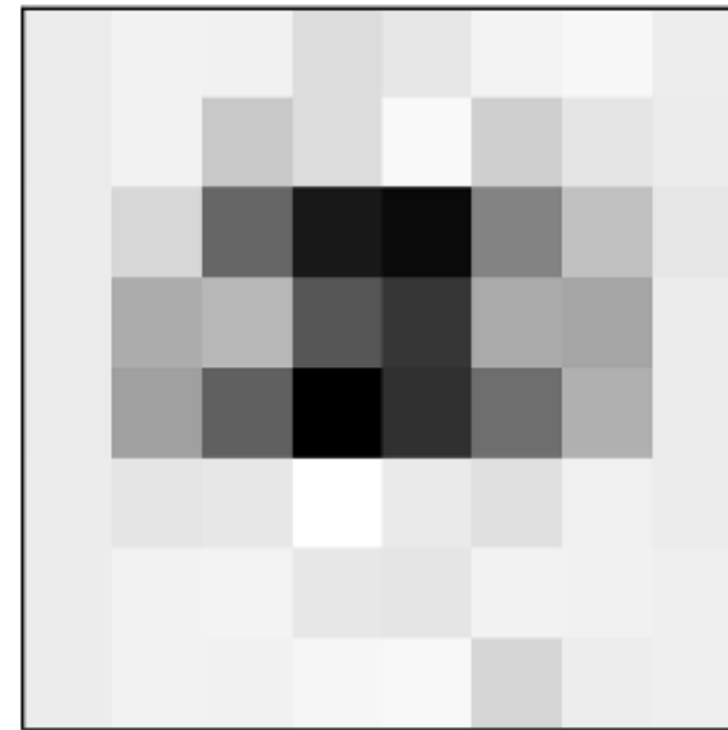
Rec. (3)



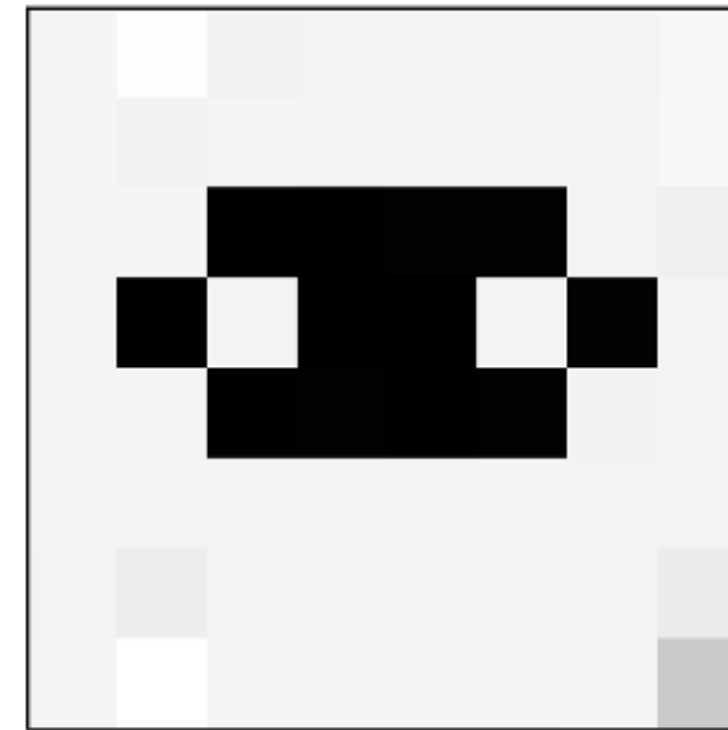
Rec. (16)



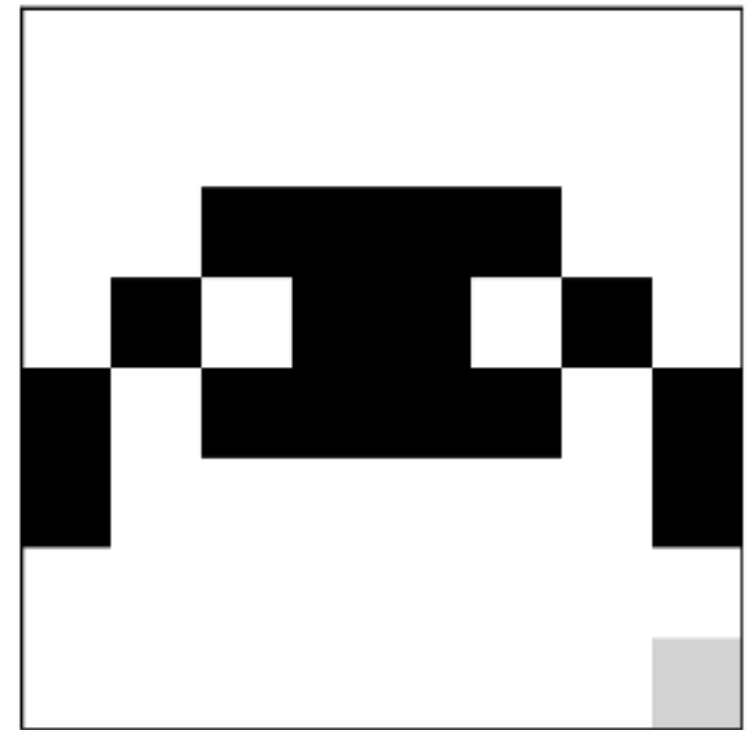
Rec. (32)



Rec. (48)



Rec. (64)



The **basis** is **optimized** for the data it has seen (digits) so it won't work as well for other kind of images.

# POD and PCA

POD and PCA use the same machinery (the SVD) but they differ in the way the data is assembled and the interpretation of the results.

$$\mathbf{D} = \begin{matrix} & \text{POD} \\ & \xleftrightarrow{m} \\ \begin{matrix} \updownarrow n \\ \left[ \begin{array}{c|c|c|c} | & | & \cdots & | \\ \mathbf{d}_1 & \mathbf{d}_2 & \cdots & \mathbf{d}_m \\ | & | & & | \end{array} \right] \end{matrix} \end{matrix}$$

The goal is to compress  $n$

$$\mathbf{D} = \mathbf{\Phi} \mathbf{A}^T$$

$\mathbf{A}$  is the low-dimensional projection

$$\mathbf{D} = \begin{matrix} & \text{PCA} \\ & \xleftrightarrow{m} \\ \begin{matrix} \updownarrow n \\ \left[ \begin{array}{c|c|c} \text{---} & \mathbf{d}_1 & \text{---} \\ \text{---} & \mathbf{d}_2 & \text{---} \\ & \vdots & \\ \text{---} & \mathbf{d}_m & \text{---} \end{array} \right] \end{matrix} \end{matrix}$$

The goal is to compress  $m$

$$\mathbf{D} = \mathbf{Z} \mathbf{\Psi}^T$$

$\mathbf{Z}$  is the low-dimensional projection

# Modal analysis of temporal evolving flow

**POD** is very often used to analyse the **dynamical behaviour** of temporally **evolving flow**.

The data matrix is arranged such that each column is a snapshot at the  $j$ -th timestep and each row is the time evolution of the  $i$ -th point.

$$\mathbf{D}(\mathbf{r}, t) = \begin{array}{c} \text{space} \downarrow \end{array} \begin{array}{c} \xrightarrow{\text{time}} \\ \left[ \begin{array}{ccc} | & | & | \\ \mathbf{d}(t_1) & \mathbf{d}(t_2) & \cdots \mathbf{d}(t_m) \\ | & | & | \end{array} \right] \end{array}$$

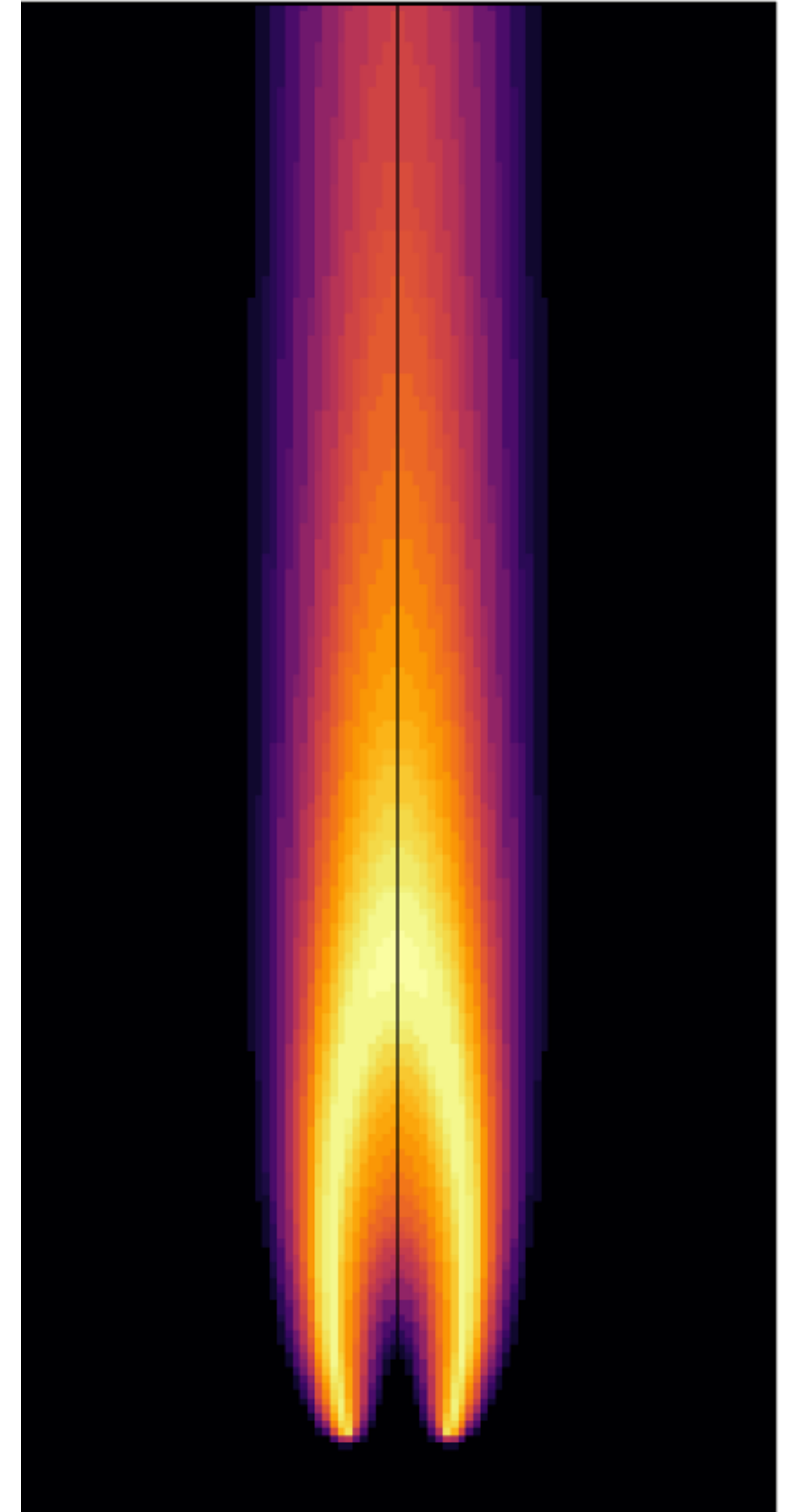
By applying the SVD we separate the spatial and temporal information:

$$\mathbf{D}(\mathbf{r}, t) = \mathbf{\Phi}(\mathbf{r})\mathbf{\Sigma}\mathbf{\Psi}(t)^T$$

The POD modes  $\mathbf{\Phi}(\mathbf{r})$  do not depend in time and they capture the most energetic structures in the flow.

# Example: pulsating flame

This is a laminar flame, where the fuel inlet is modulated at a frequency of 10 Hz.

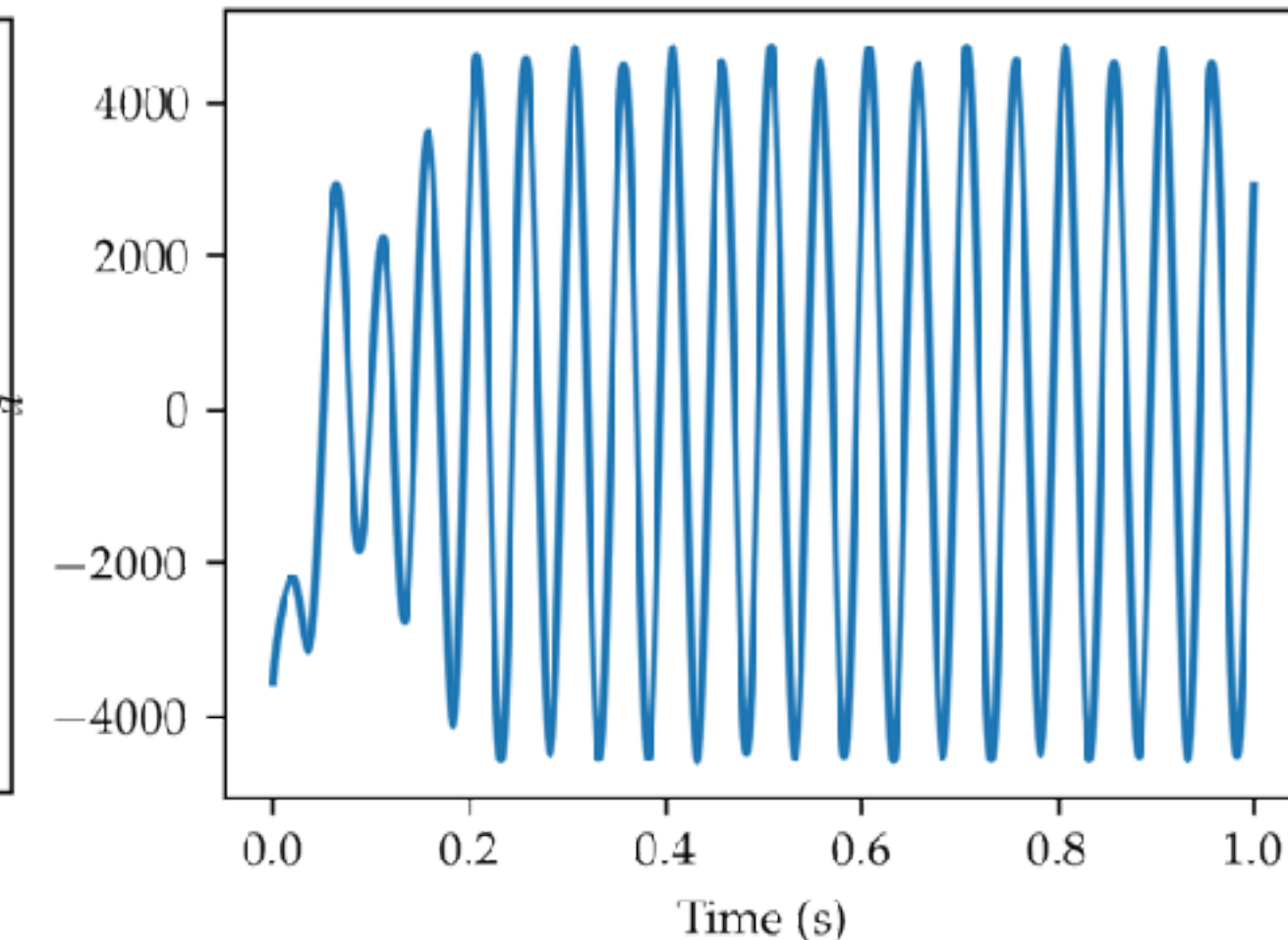
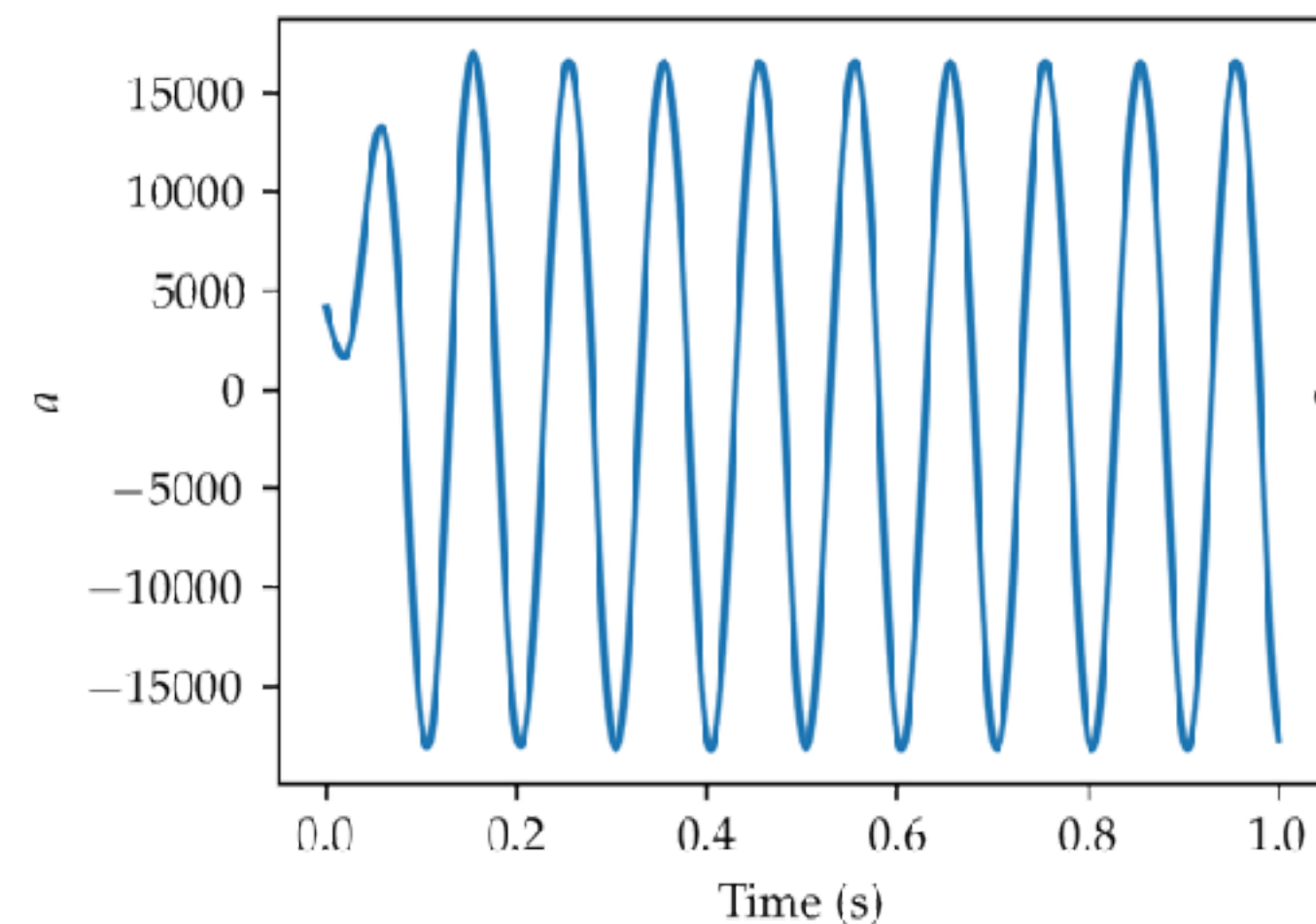
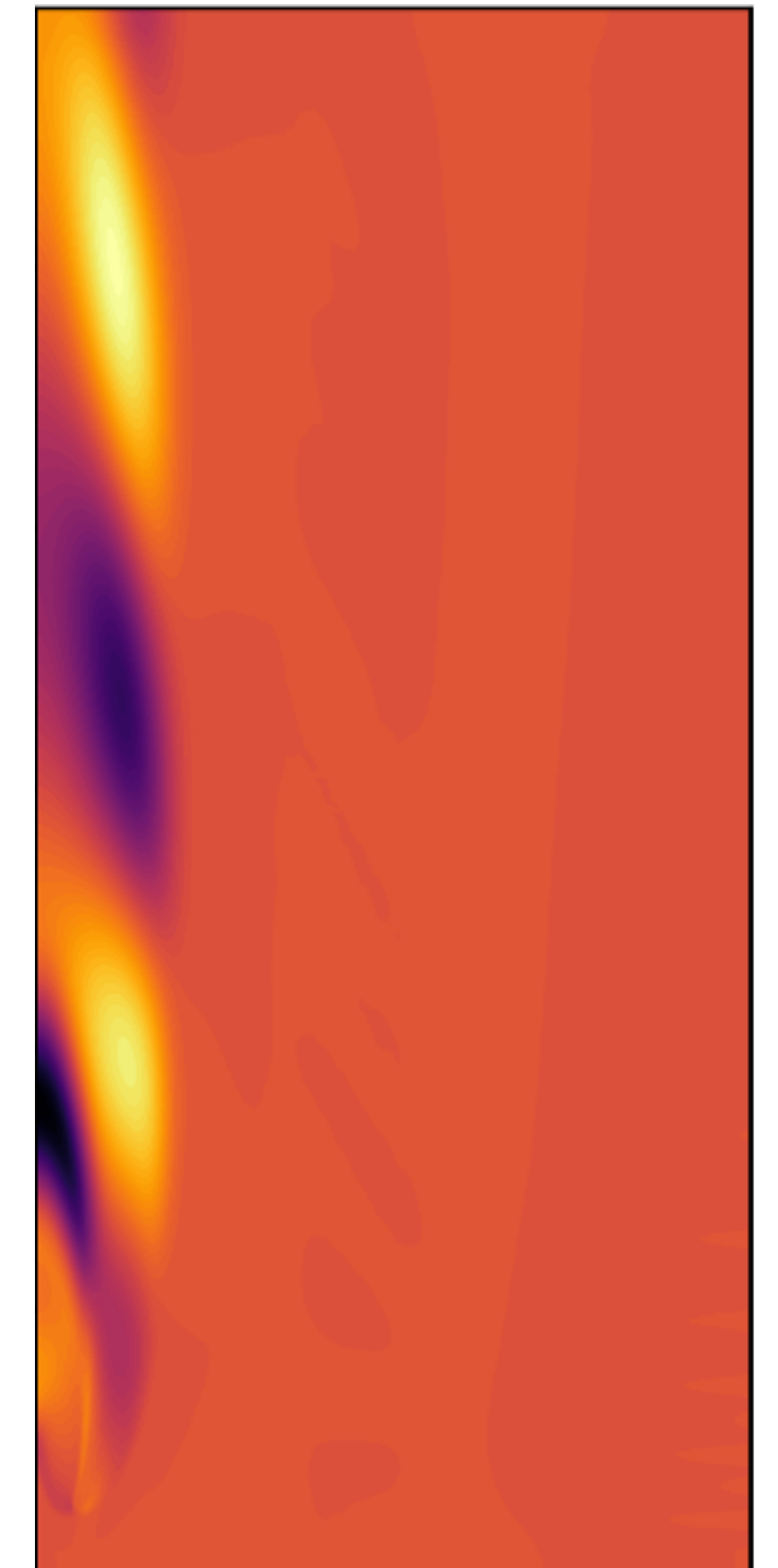


# Example: pulsating flame

This is a laminar flame, where the fuel inlet is modulated at a frequency of 10 Hz.

The POD **modes** represent the **coherent structures** in the dataset.

By analysing the **temporal** evolution and the **frequency** content of the **coefficients** we can infer the type of flow **structure**.



# Comparison with the DFT

The **DFT** is similar to the POD in the sense that it **separates** the **temporal** and **spatial** information.

However, the transforming matrix does **not depend** on the **data**, and the goal is to decompose the dataset into a sum of periodic functions.

$$\Psi_F \in \mathbb{C}^{m \times m} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega & \dots & \omega^{(m-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{(m-1)} & \dots & \omega^{(m-1)^2} \end{bmatrix}$$

$$\omega = \exp(2\pi i/m) \quad i = \sqrt{-1}$$

$$\mathbf{D}(\mathbf{r}, t) = \Phi_F(\mathbf{r}) \Sigma_F \Psi_F(t)^T$$

The DFT **modes**  $\Phi(\mathbf{r})$  are complex vectors (contain magnitude and phase) and are **not orthogonal**. The diagonal matrix  $\Sigma_F$  contains the amplitude of the sines and cosines.



# Galerkin projection

The POD properties are attractive to solve PDEs in an efficient way.

A general set of non-linear PDEs (Navier-Stokes for example) can be represented as:

$$\dot{\mathbf{d}}(\mathbf{r}, t) = \mathcal{N}(\mathbf{d}(\mathbf{r}, t))$$

# Galerkin projection

The POD properties are attractive to solve PDEs in an efficient way.

A general set of non-linear PDEs (Navier-Stokes for example) can be represented as:

$$\dot{\mathbf{d}}(\mathbf{r}, t) = \mathcal{N}(\mathbf{d}(\mathbf{r}, t))$$

Each snapshot can be represented as:  $\mathbf{d}(\mathbf{r}, t) = \sum_{i=1}^q a_i(t) \boldsymbol{\phi}_i(\mathbf{r})$

# Galerkin projection

The POD properties are attractive to solve PDEs in an efficient way.

A general set of non-linear PDEs (Navier-Stokes for example) can be represented as:

$$\dot{\mathbf{d}}(\mathbf{r}, t) = \mathcal{N}(\mathbf{d}(\mathbf{r}, t))$$

Each snapshot can be represented as:  $\mathbf{d}(\mathbf{r}, t) = \sum_{i=1}^q a_i(t) \boldsymbol{\phi}_i(\mathbf{r})$

The systems can be rewritten as  $\sum_{i=1}^q \dot{a}_i(t) \boldsymbol{\phi}_i(\mathbf{r}) = \mathcal{N}(\sum_{i=1}^q a_i(t) \boldsymbol{\phi}_i(\mathbf{r}))$

# Galerkin projection

The POD properties are attractive to solve PDEs in an efficient way.

A general set of non-linear PDEs (Navier-Stokes for example) can be represented as:

$$\dot{\mathbf{d}}(\mathbf{r}, t) = \mathcal{N}(\mathbf{d}(\mathbf{r}, t))$$

Each snapshot can be represented as:  $\mathbf{d}(\mathbf{r}, t) = \sum_{i=1}^q a_i(t) \boldsymbol{\phi}_i(\mathbf{r})$

The systems can be rewritten as  $\sum_{i=1}^q \dot{a}_i(t) \boldsymbol{\phi}_i(\mathbf{r}) = \mathcal{N}(\sum_{i=1}^q a_i(t) \boldsymbol{\phi}_i(\mathbf{r}))$

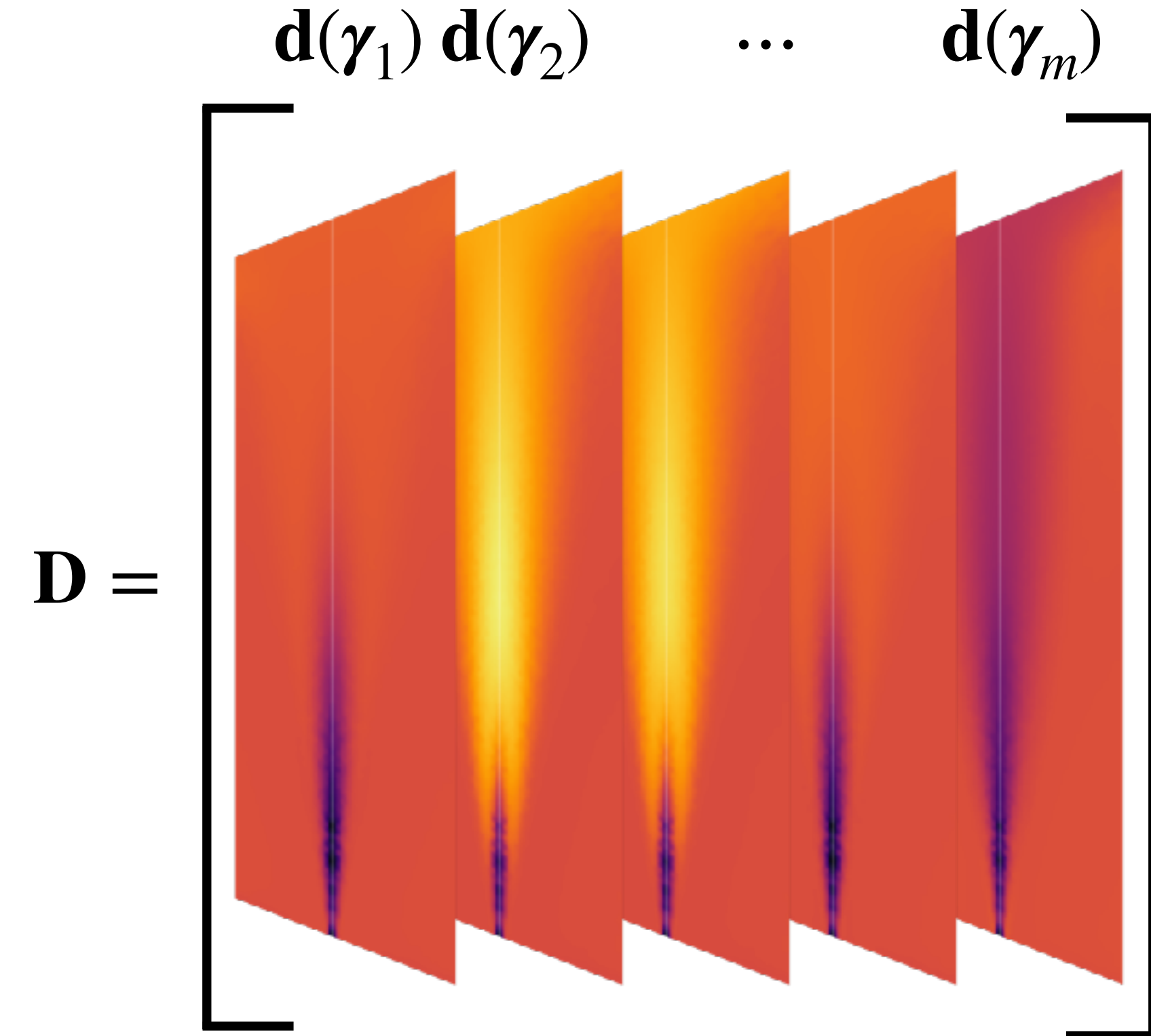
Then, by exploiting the fact that  $\langle \boldsymbol{\phi}_i, \boldsymbol{\phi}_j \rangle = 0$  if  $i \neq j$  we get a set of  $q$  ODEs:

$$\dot{a}_i(t) = \langle \mathcal{N}(\sum_{i=1}^q a_i(t) \boldsymbol{\phi}_i(\mathbf{r})), \boldsymbol{\phi}_i(\mathbf{r}) \rangle \quad i = 1, \dots, q$$

# Parametric reduced-order models

Let's consider a system controlled by a set of parameters  $\gamma$ .

We want to predict the system's state  $\mathbf{d}$  for every possible combination of  $\gamma$ .



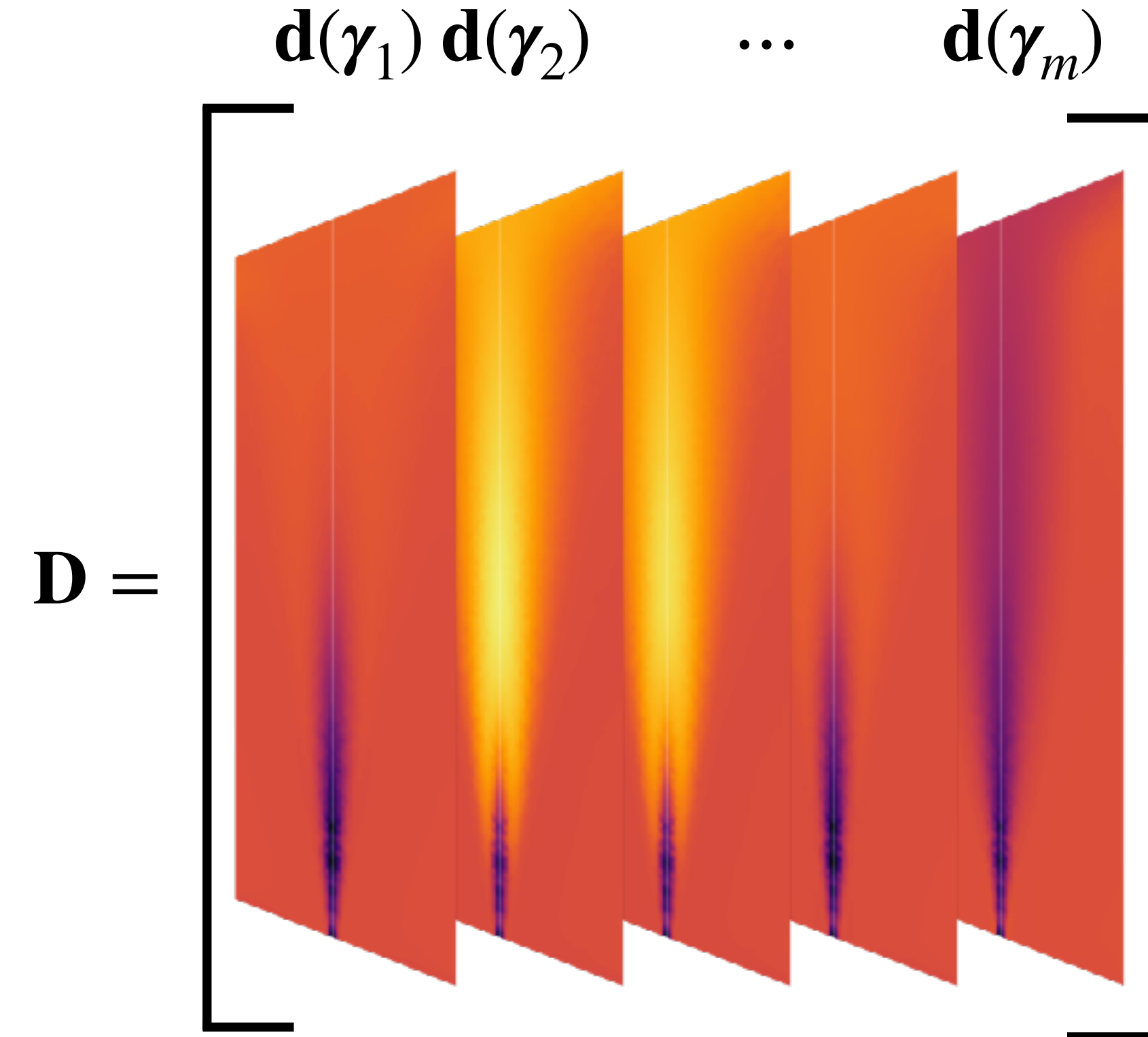
# Parametric reduced-order models

Let's consider a system controlled by a set of parameters  $\gamma$ .

We want to predict the system's state  $\mathbf{d}$  for every possible combination of  $\gamma$ .

The temptation could be to directly estimate this input-output relationship with a regression model:

$$\mathbf{d} = f(\gamma) + \epsilon_n \quad f: \mathbb{R}^d \mapsto \mathbb{R}^n$$



# Parametric reduced-order models

Let's consider a system controlled by a set of parameters  $\gamma$ .

We want to predict the system's state  $\mathbf{d}$  for every possible combination of  $\gamma$ .

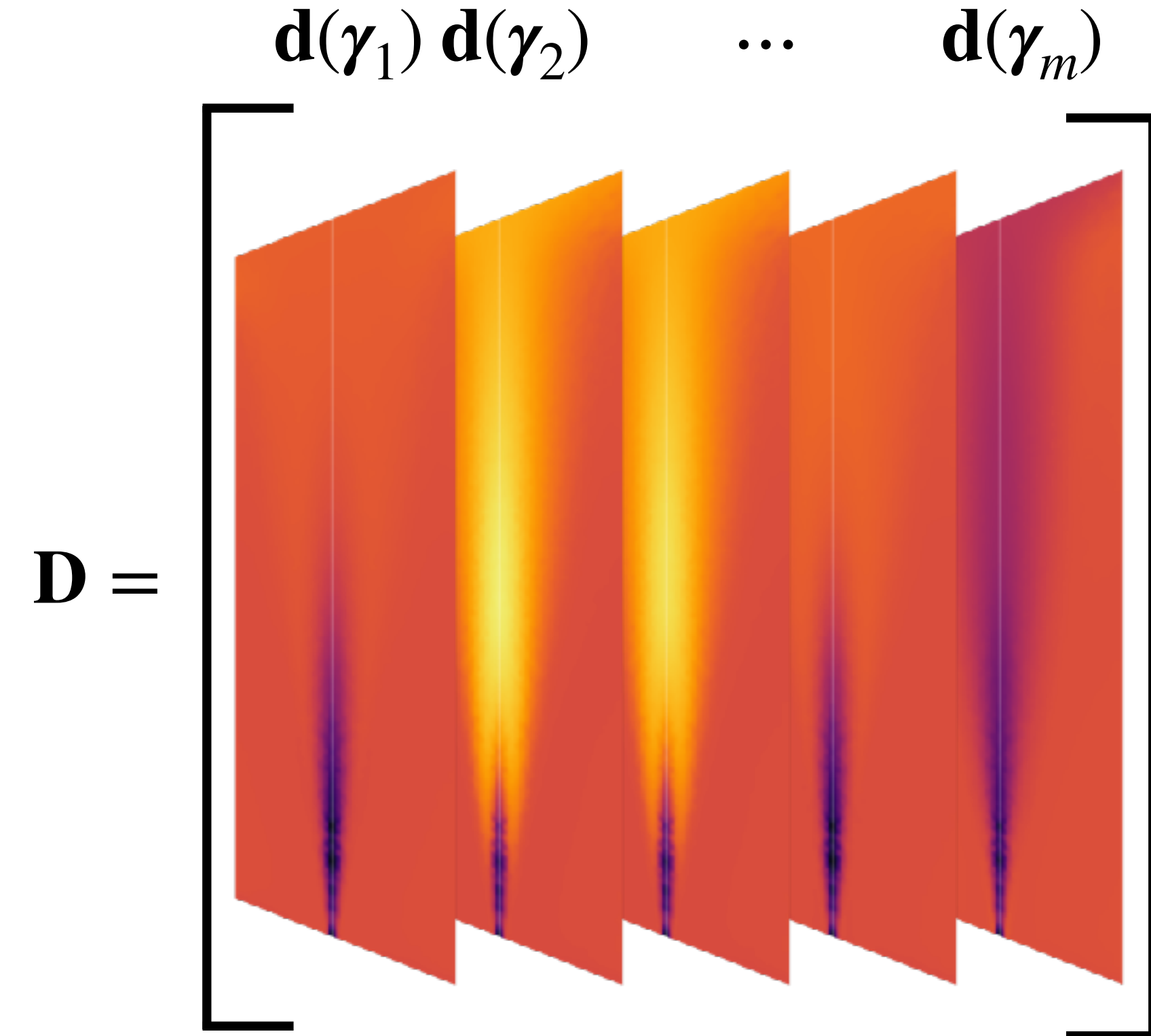
The temptation could be to directly estimate this input-output relationship with a regression model:

$$\mathbf{d} = f(\gamma) + \epsilon_n \quad f: \mathbb{R}^d \mapsto \mathbb{R}^n$$

It is more efficient to estimate the regression with the low-dimensional projection:

$$\mathbf{d}(\mathbf{r}, \gamma) = \sum_{i=1}^q a_i(\gamma) \phi_i(\mathbf{r})$$

$$a_i = r_i(\gamma) + \epsilon \quad i = 1 \dots, q \quad r_i: \mathbb{R}^d \mapsto \mathbb{R}$$



# References

A couple of good books on machine learning:

- Brunton, S. L., & Kutz, J. N. (2022). *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press.
- Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020). *Mathematics for machine learning*. Cambridge University Press.

If you are into POD and want to know more:

- Ghojogh, B., Crowley, M., Karray, F., & Ghodsi, A. (2023). *Elements of dimensionality reduction and manifold learning* (pp. 1-596). Berlin/Heidelberg, Germany: Springer.