CS598PS Machine Learning for Signal Processing

Features Part I – Principal Component Analysis

13 September 2017

Today's lecture

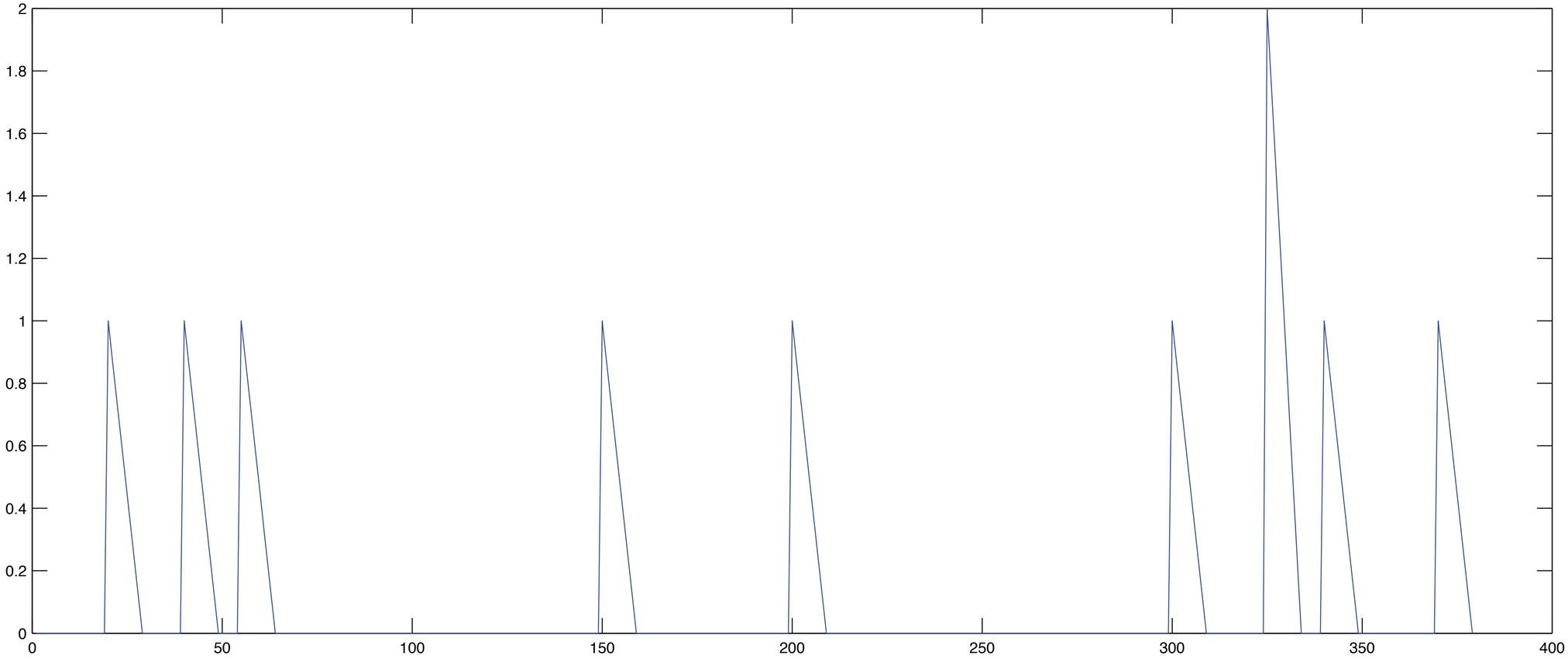
- Adaptive Feature Extraction
- Principal Component Analysis
  - How, why, when, which

A dual goal

- Find a good representation
  - The features part
- Reduce redundancy in the data
  - A side effect of “proper” features

Example case

- Describe this input



A “good feature”

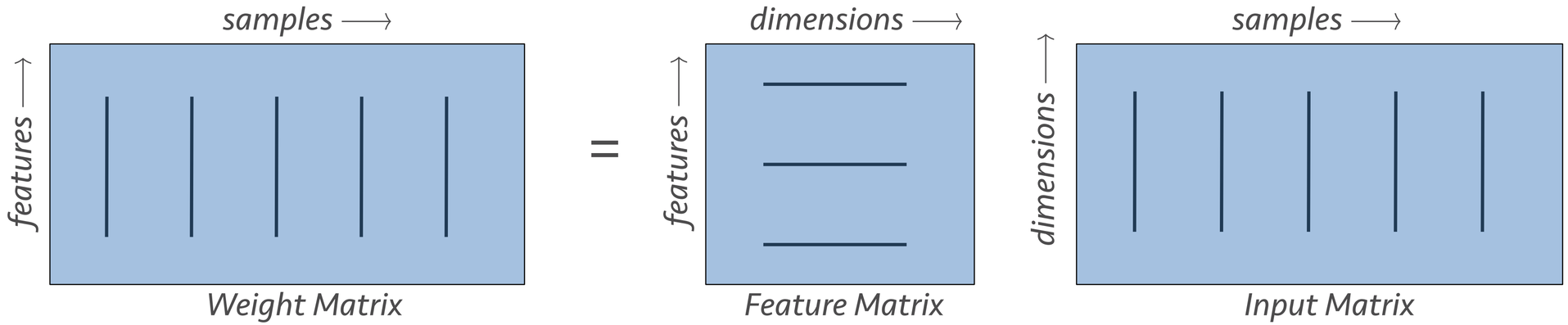
- “Simplify” the explanation of the input
  - Represent repeating patterns
  - Once defined, makes the input description simpler

- How do we define these abstract qualities?
  - On to the math …

Linear features

$$\mathbf{Z} = \mathbf{W} \cdot \mathbf{X}$$

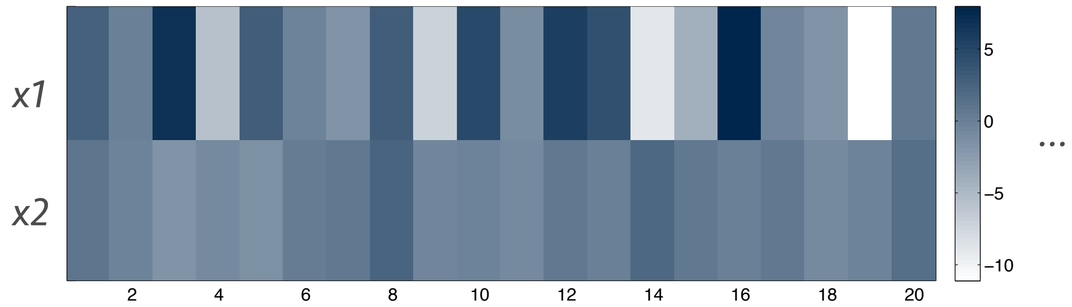
Weight Matrix *Feature Matrix* *Input Matrix*



A 2D case

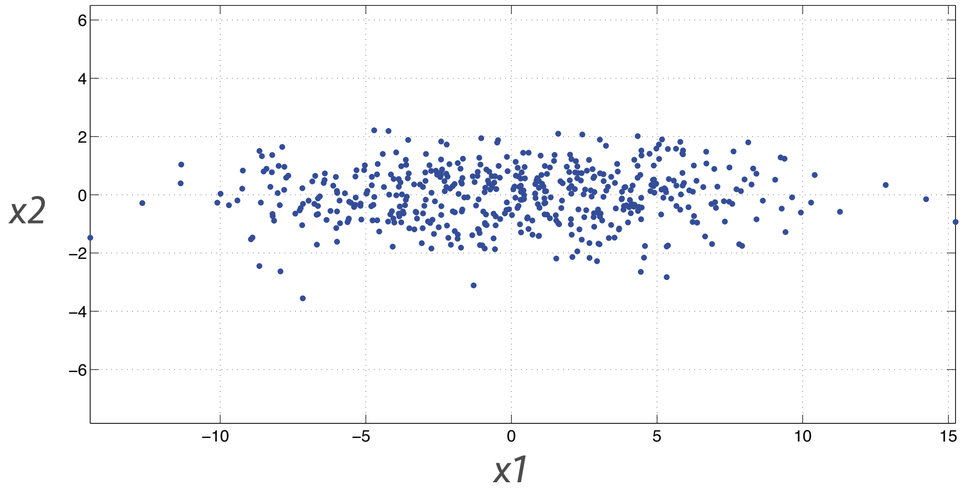
$$\mathbf{Z} = \mathbf{W} \cdot \mathbf{X} = \begin{bmatrix} \mathbf{z}^\top_1 \\ \mathbf{z}^\top_2 \end{bmatrix} = \begin{bmatrix} \mathbf{w}^\top_1 \\ \mathbf{w}^\top_2 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x}^\top_1 \\ \mathbf{x}^\top_2 \end{bmatrix}$$

Matrix representation of data



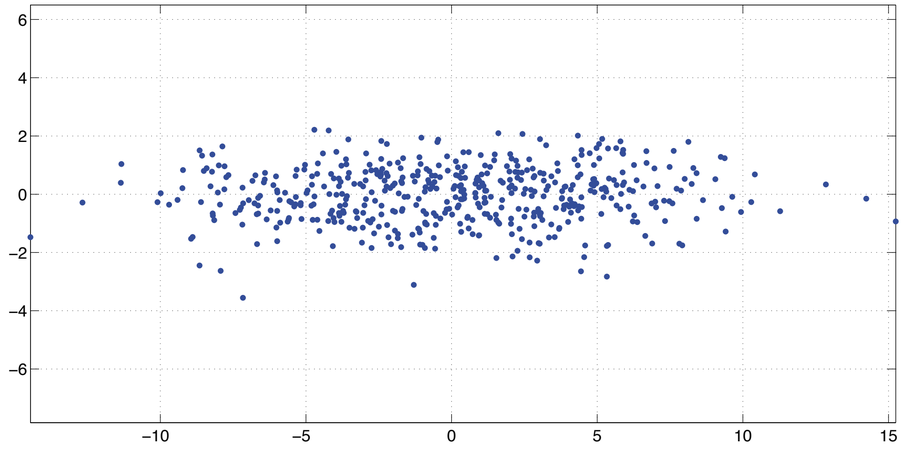
...

Scatter plot of same data



Defining a goal

- Desirable feature features
  - Give “simple” weights
  - Avoid feature similarity
- How do we define these?



One way to proceed

- "Simple weights"
  - Minimize similarity of the two weight dimensions
    - i.e. $\mathbf{z}_1$ and $\mathbf{z}_2$ should be "different"

- "Feature similarity"
  - Same thing for features!
    - $\mathbf{w}_1$ and $\mathbf{w}_2$ should be different

One way to proceed

- "Simple weights"
  - Minimize similarity of the two weight dimensions
    - *Decorrelate*: $\mathbf{z}_1^\top \cdot \mathbf{z}_2 = 0$
- "Feature similarity"
  - Same thing for features!
    - *Decorrelate*: $\mathbf{w}_1^\top \cdot \mathbf{w}_2 = 0$

Doing it in a single expression

- Use the covariance matrix

Technically, we divide by $N-1$ but we simplify for readability

$$\mathrm{Cov}\left(\mathbf{z}_1, \mathbf{z}_2\right) = \begin{bmatrix} \mathbf{z}_1^{\top} \cdot \mathbf{z}_1 & \mathbf{z}_1^{\top} \cdot \mathbf{z}_2 \\ \mathbf{z}_2^{\top} \cdot \mathbf{z}_1 & \mathbf{z}_2^{\top} \cdot \mathbf{z}_2 \end{bmatrix} / N = \mathbf{Z} \cdot \mathbf{Z}^{\top} / N$$

  - $\{i,j\}$ element is the dot product of $\{i,j\}$-dimension pair
    - Represents all dimension combinations at once

Diagonalizing the covariance

- Diagonalizing the covariance suppresses cross-dimensional co-activity
  - if $\mathbf{z}_1$ is high, $\mathbf{z}_2$ won't be, etc.

$$\text{Cov}\left(\mathbf{z}_1, \mathbf{z}_2\right) = \begin{bmatrix} \mathbf{z}_1^\top \cdot \mathbf{z}_1 & \mathbf{z}_1^\top \cdot \mathbf{z}_2 \\ \mathbf{z}_2^\top \cdot \mathbf{z}_1 & \mathbf{z}_2^\top \cdot \mathbf{z}_2 \end{bmatrix} / N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \mathbf{I}$$

Problem definition

- For a given input $\mathbf{X}$
- Find a feature matrix $\mathbf{W}$
- So that the weights *decorrelate*

$$\left(\mathbf{W}\cdot\mathbf{X}\right)\cdot\left(\mathbf{W}\cdot\mathbf{X}\right)^{\top}=N\mathbf{I}\Rightarrow\mathbf{Z}\cdot\mathbf{Z}^{\top}=N\mathbf{I}$$

How do we solve this?

- Any ideas?

$$\left(\mathbf{W}\cdot\mathbf{X}\right)\cdot\left(\mathbf{W}\cdot\mathbf{X}\right)^{\top}=N\mathbf{I}$$

Solving for diagonalization

$$\Big(\mathbf{W}\cdot\mathbf{X}\Big)\cdot\Big(\mathbf{W}\cdot\mathbf{X}\Big)^{\top} = N\mathbf{I} \Rightarrow$$

$$\Rightarrow \mathbf{W}\cdot\mathbf{X}\cdot\mathbf{X}^{\top}\cdot\mathbf{W}^{\top} = N\mathbf{I} \Rightarrow$$

$$\Rightarrow \mathbf{W}\cdot\mathrm{Cov}\Big(\mathbf{X}\Big)\cdot\mathbf{W}^{\top} = \mathbf{I}$$

Solving for diagonalization

- Covariance matrices are positive definite
  - And symmetric
    - have orthogonal eigenvectors and real eigenvalues
  - and are factorized by:

$$\mathbf{A} \cdot \mathbf{U} = \mathbf{U} \cdot \mathbf{\Lambda} \Rightarrow \mathbf{U}^{\top} \cdot \mathbf{A} \cdot \mathbf{U} = \mathbf{\Lambda}$$

  - Where $\mathbf{U}$ has the eigenvectors of $\mathbf{A}$ in its columns
  - $\mathbf{\Lambda} = \mathrm{diag}(\lambda_i)$, where λ_i are the eigenvalues of $\mathbf{A}$

Solving for diagonalization

Eigenanalysis *Problem to solve*

What we have:

$$\mathbf{U}^\top \cdot \mathbf{A} \cdot \mathbf{U} = \mathbf{\Lambda} \qquad \mathbf{W} \cdot \mathrm{Cov}\left(\mathbf{X}\right) \cdot \mathbf{W}^\top = \mathbf{I}$$

Substitute:

$$\mathbf{U}_{\mathrm{Cov}(\mathbf{X})}^\top \cdot \mathrm{Cov}\left(\mathbf{X}\right) \cdot \mathbf{U}_{\mathrm{Cov}(\mathbf{X})} = \mathbf{\Lambda}_{\mathrm{Cov}(\mathbf{X})}$$

Get the identity:

$$\left(\mathbf{\Lambda}_{\mathrm{Cov}(\mathbf{X})}\right)^{-1/2} \mathbf{U}_{\mathrm{Cov}(\mathbf{X})}^\top \cdot \mathrm{Cov}\left(\mathbf{X}\right) \cdot \mathbf{U}_{\mathrm{Cov}(\mathbf{X})} \left(\mathbf{\Lambda}_{\mathrm{Cov}(\mathbf{X})}\right)^{-1/2} = \mathbf{I}$$

Solve for $\mathbf{W}$ *:*

$$\mathbf{W} = \left(\mathbf{\Lambda}_{\mathrm{Cov}(\mathbf{X})}\right)^{-1/2} \mathbf{U}_{\mathrm{Cov}(\mathbf{X})}^\top = \begin{bmatrix} \sqrt{\lambda_1}_{\mathrm{Cov}(\mathbf{X})} & 0 \\ 0 & \sqrt{\lambda_2}_{\mathrm{Cov}(\mathbf{X})} \end{bmatrix}^{-1} \cdot \mathbf{U}_{\mathrm{Cov}(\mathbf{X})}^\top$$

Solving for diagonalization

- The solution is a product of the eigenvectors $\mathbf{U}$ and the square root of the eigenvalues $\mathbf{\Lambda}$ of $\mathrm{Cov}(\mathbf{X})$

$$\mathbf{W} \cdot \mathrm{Cov}\left(\mathbf{X}\right) \cdot \mathbf{W}^{\top} = \mathbf{I} \Rightarrow$$

$$\Rightarrow \mathbf{W} = \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{bmatrix}^{-1} \cdot \mathbf{U}^{\top}$$

Another solution

- That was not the only solution

- Consider this one:

$$\mathbf{W}\cdot\mathbf{X}\cdot\left(\mathbf{W}\cdot\mathbf{X}\right)^{\top}=N\mathbf{I}\Rightarrow$$

$$\mathbf{W}\cdot\mathbf{X}\cdot\mathbf{X}^{\top}\cdot\mathbf{W}^{\top}=N\mathbf{I}\Rightarrow$$

$$\mathbf{W}=\sqrt{N}\left(\mathbf{X}\cdot\mathbf{X}^{\top}\right)^{-1/2}$$

Another solution

- That was not the only solution
  - Consider this one: $\mathbf{W} \cdot \mathbf{X} \cdot \left(\mathbf{W} \cdot \mathbf{X} \right)^{\top} = N\mathbf{I} \Rightarrow$

$$\mathbf{W} \cdot \mathbf{X} \cdot \mathbf{X}^{\top} \cdot \mathbf{W}^{\top} = N\mathbf{I} \Rightarrow$$

$$\mathbf{W} = \sqrt{N} \left(\mathbf{X} \cdot \mathbf{X}^{\top} \right)^{-1/2}$$

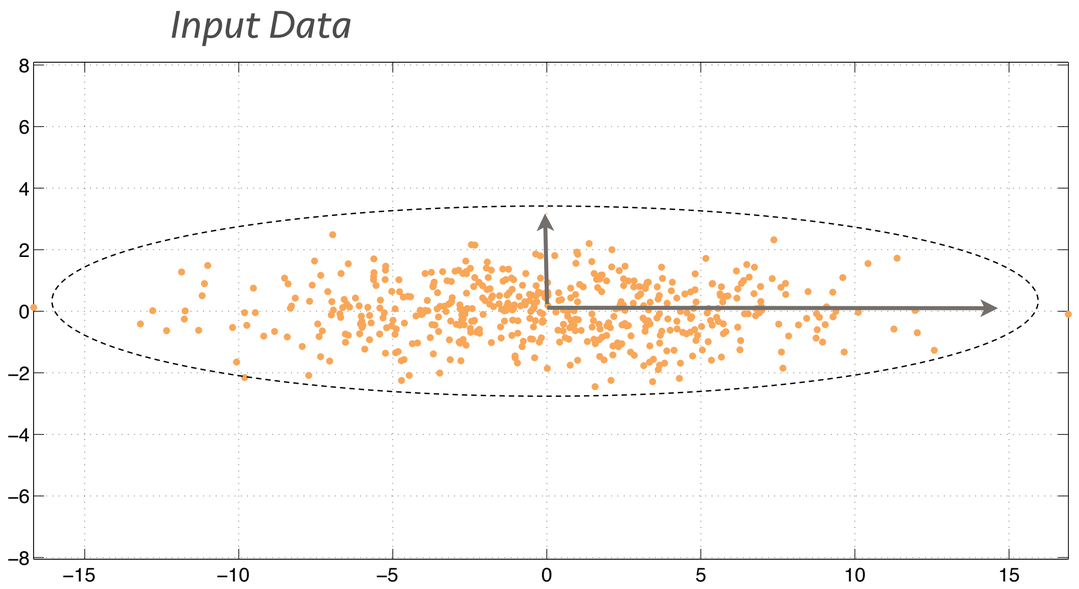
- How do you get the matrix square root?

$$\mathbf{W} = \mathbf{U} \cdot \mathbf{S}^{-1/2} \cdot \mathbf{V}^{\top}$$

$$\left[\mathbf{U}, \mathbf{S}, \mathbf{V} \right] = \mathrm{SVD} \left(\mathbf{X} \cdot \mathbf{X}^{\top} \right)$$

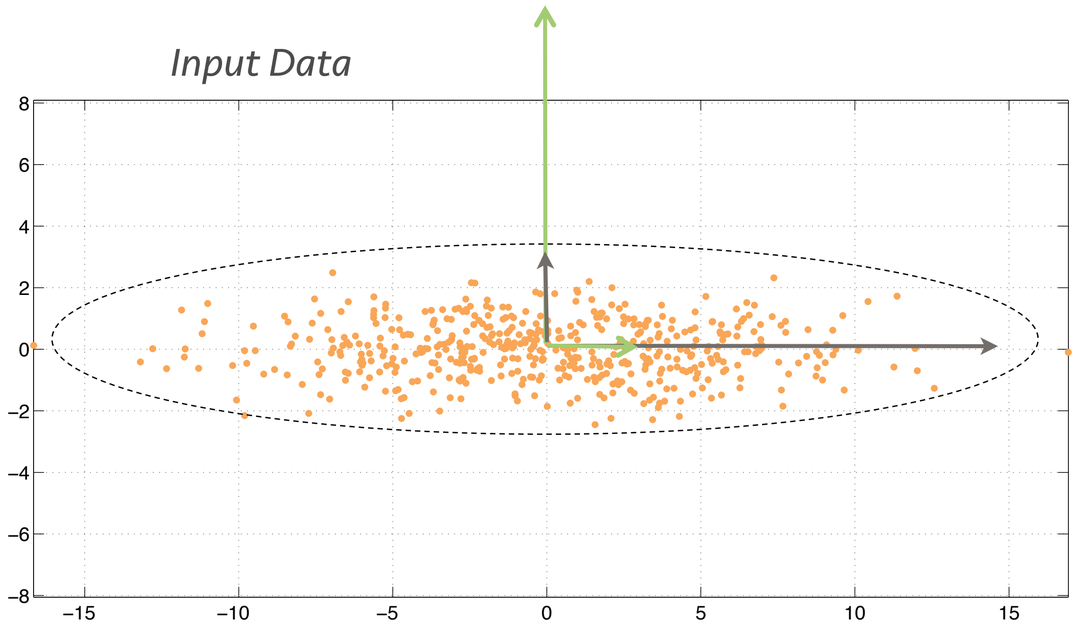
Decorrelation in pictures

- An implicit Gaussian assumption
  - N-dimensional data has N directions of variance



Undoing the variance

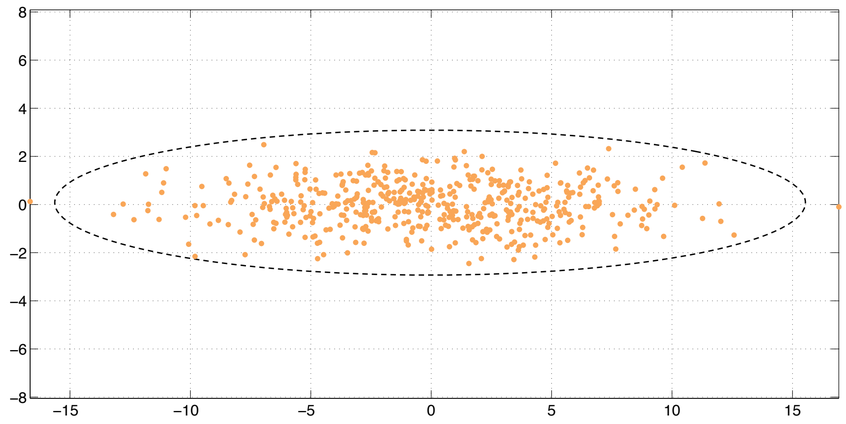
- The decorrelating matrix $\mathbf{W}$ contains two vectors that normalize the input's variance



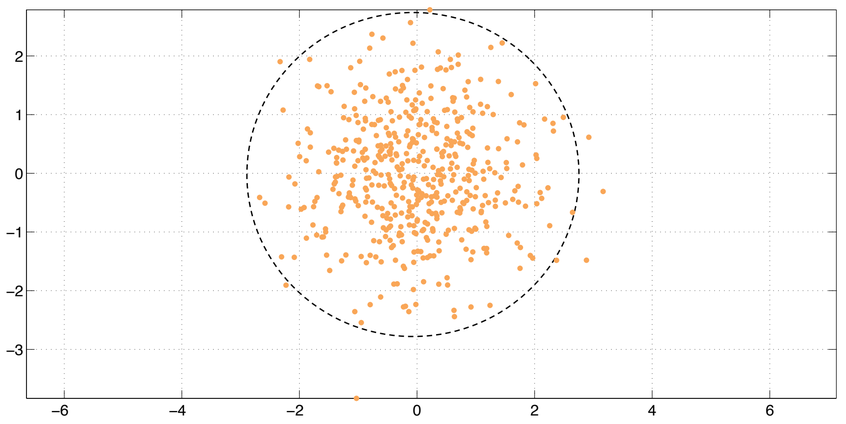
Resulting transform

- Input gets scaled to a well-behaved Gaussian with unit variance in all dimensions

Input Data

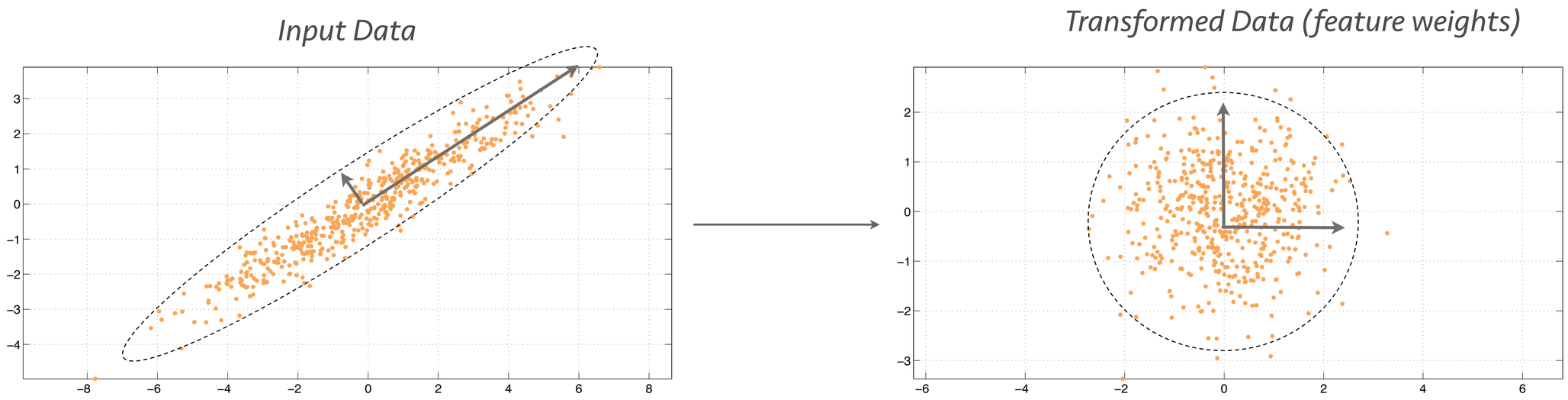


Transformed Data (feature weights)



A more complex case

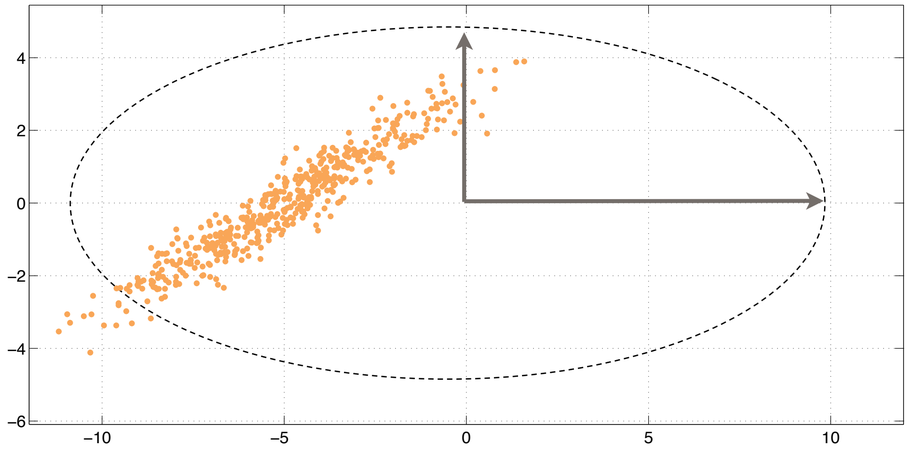
- Having correlation between two dimensions
  - We still find the directions of maximal variance
  - But we also rotate in addition to scaling



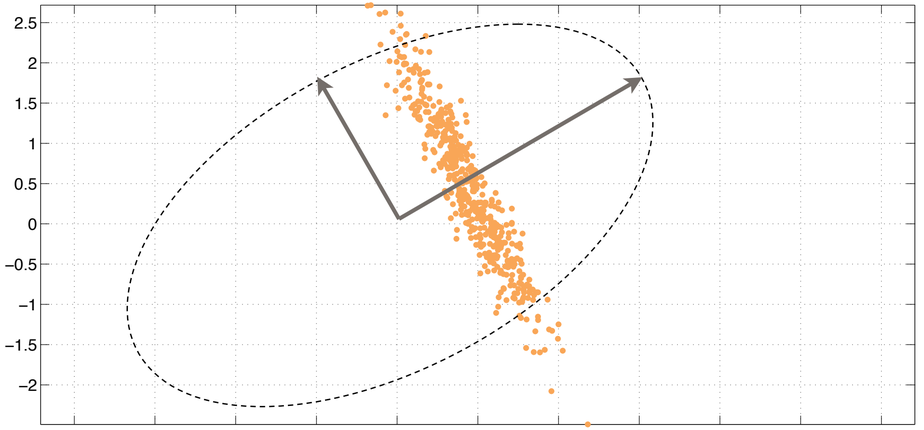
One more detail

- So far we considered zero-mean inputs
  - The transforming operation was a rotation
- If the input mean is not zero bad things happen!
  - Make sure that your data is zero-mean!

Input Data



Transformed Data (feature weights)



Principal Component Analysis

- This transform is known as PCA
  - The features are the principal components
    - They are orthogonal to each other
    - And produce orthogonal (white) weights
  - Major tool in statistics
    - Removes dependencies from multivariate data
- Also known as the KLT
  - Karhunen-Loève Transform

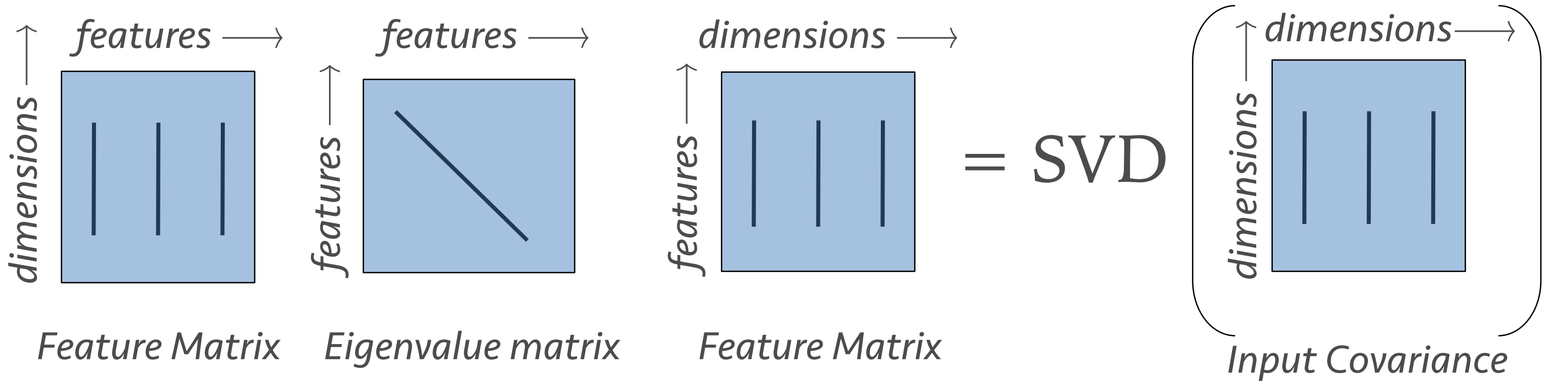
A better way to compute PCA

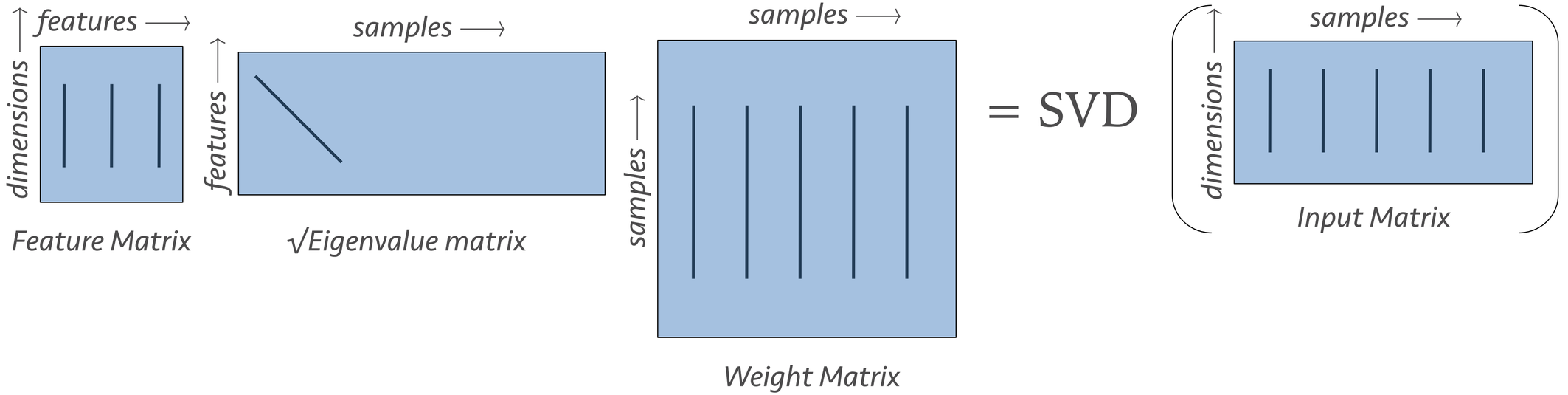
- The Singular Value Decomposition way

$$\left[\mathbf{U},\mathbf{S},\mathbf{V}\right]=\mathrm{SVD}\left(\mathbf{A}\right)\Rightarrow\mathbf{A}=\mathbf{U}\cdot\mathbf{S}\cdot\mathbf{V}^{\top}$$

- Relationship to eigendecomposition
  - In our case the input will be the $\mathbf{A}$ matrix
  - $\mathbf{U}$ and $\mathbf{S}$ will be like the eigenvectors/values of $\mathbf{A}$
- Why the SVD?
  - More stable, more robust, fancy extensions

PCA through the SVD





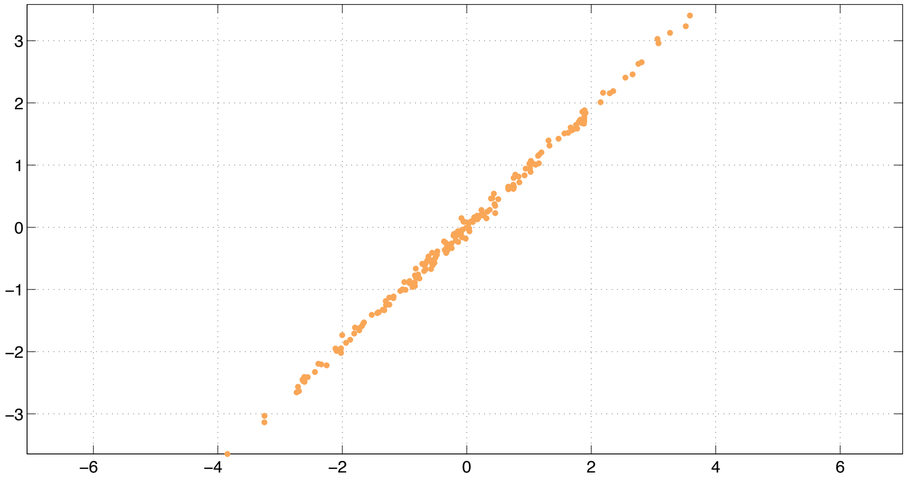
Dimensionality reduction

- PCA is great for high dimensional data
- Allows us to perform dimensionality reduction
  - Helps us find relevant structure in data
  - Helps us throw away things that won't matter

A simple example

- Two very correlated dimensions
  - e.g. size and weight of fruit
  - One effective variable
- Whitening matrix is:

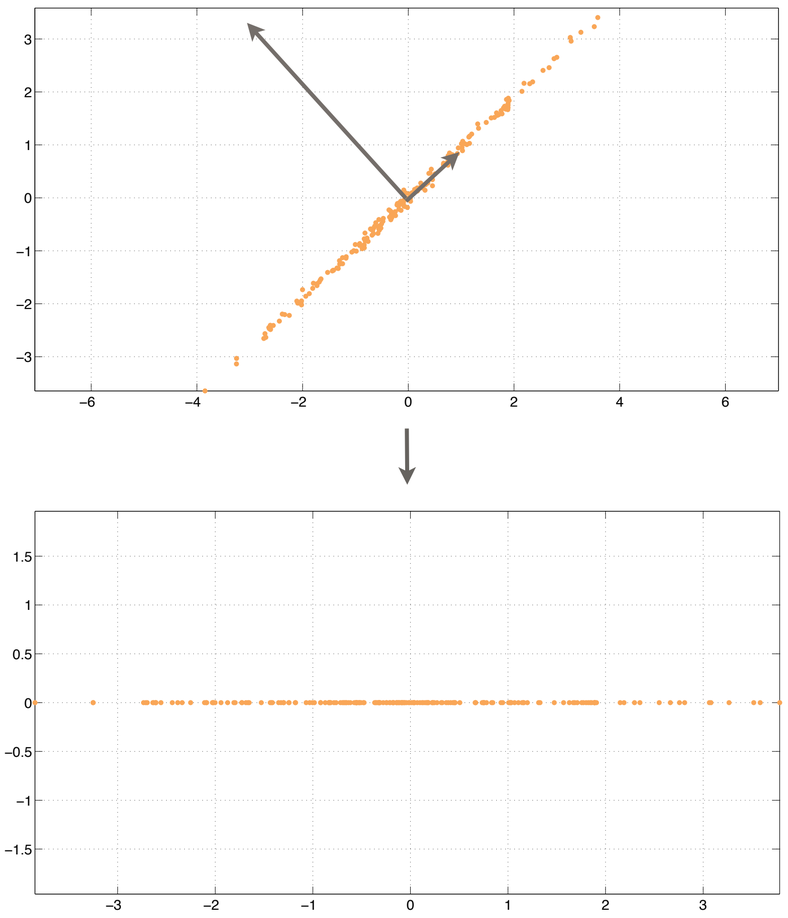
$$\mathbf{w} = \begin{bmatrix} -0.36 & -0.34 \\ -14.1 & 14.8 \end{bmatrix}$$



  - Large variance between the two components
    - Implies that only one dimension is useful

A simple example

- Second principal component needs to be super-boosted to whiten the weights
  - maybe is it useless?
- Keep only high variance
  - Throw away components with minor variance contributions



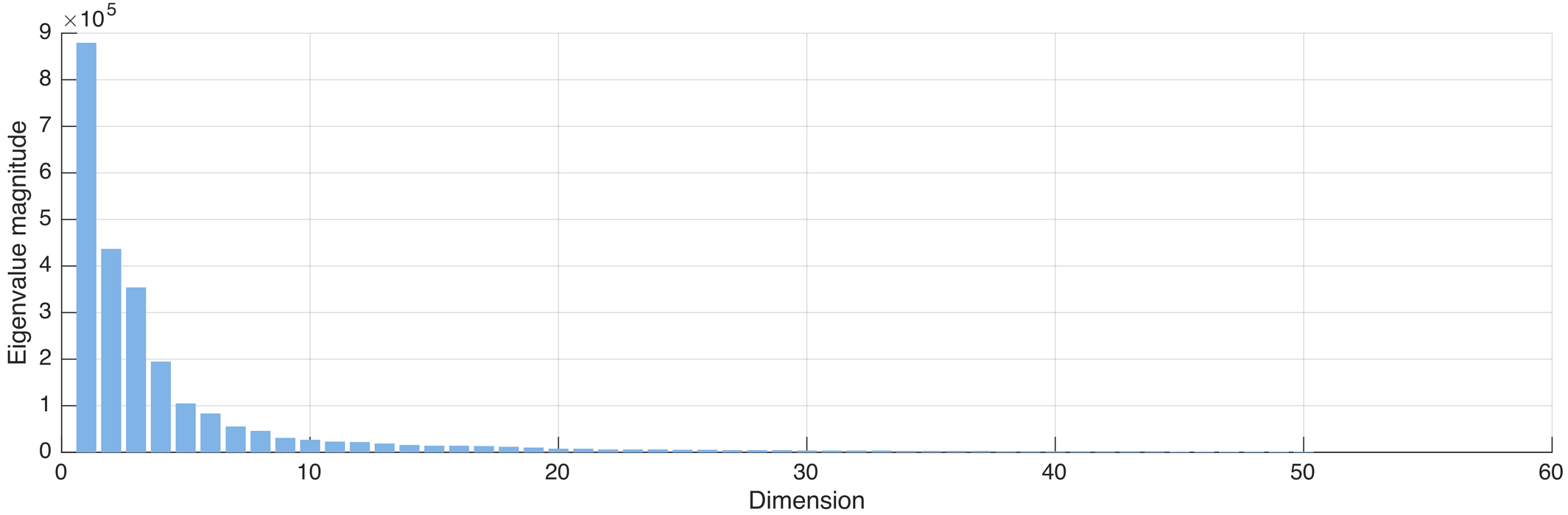
What is the number of dimensions?

- If the input was M dimensional, how many dimensions do we keep?
  - No solid answer (estimators exist but can be flaky)

- Look at the singular values/eigenvalues
  - They will show the variance of each component, at some point it will be small

Example

- Eigenvalues of 4800-dimensional video data
  - Little variance after component 20
  - We don't need to keep the rest of the data



...

So where are the features?

- We strayed off-subject
  - What happened to the features?
  - We only mentioned that they are orthogonal

- We talked about the weights so far, let's talk about the principal components
  - They should encapsulate structure
  - How do they look like?

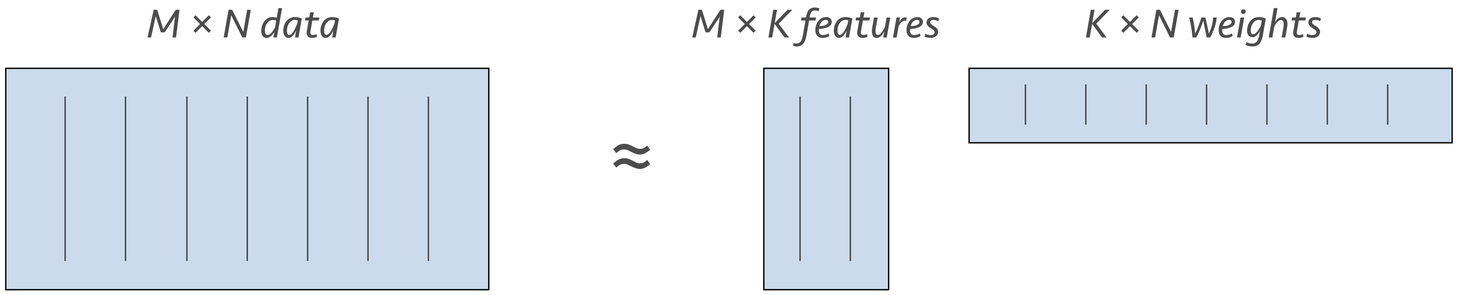
Face analysis

- Analysis of a face database
  - What are good features for faces?
- Is there anything special there?
  - What will PCA give us?
  - Any extra insight?
- Lets see some code in action

The Eigenfaces

Low-rank decompositions

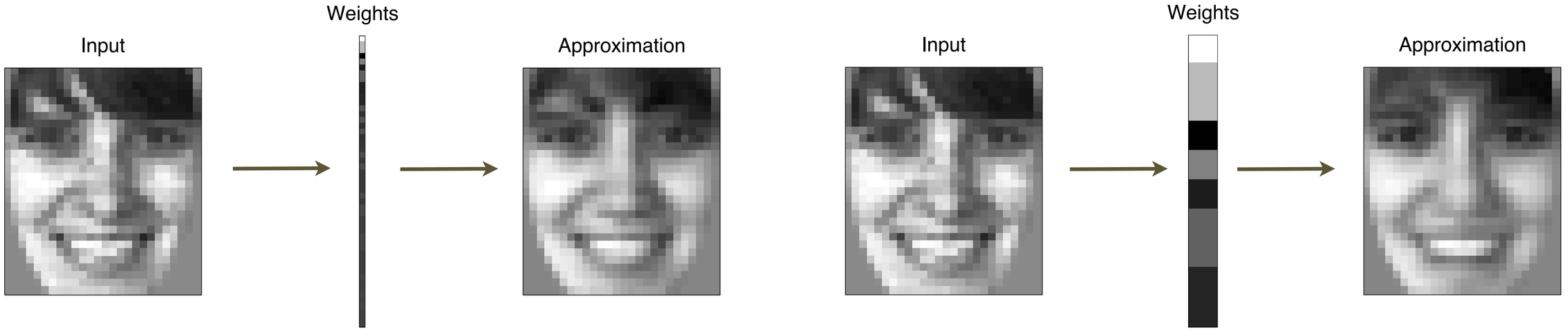
- Big data is painful to deal with
  - We want to reduce their quantity without losing information



- For many operations we will use the low-rank weights
  - More versions of that later

Low-rank advantage with face data

- Full data
  - 600 faces with 30 × 26 pixels = 600 × 780 = 468,000 values
- Low-rank version
  - K=50: 600 weights with 50 values + 50 × 780 bases = 69,000 values
  - K=10: 600 weights with 10 values + 10 × 780 bases = 6,780 values



PCA for large dimensionalities

- Sometimes the data is very high dimensional
  - e.g. 720p videos: $1280 \times 720 \times T = 921{,}600\text{-D} \times T$ frames
- You will not do an SVD that big!
  - Complexity is $O(4m^2n + 8mn^2 + 9n^3)$
- We can now use approximate methods
  - Faster, but sloppier

EM-PCA

- Alternate between successive approximations
  - Start with random $\mathbf{W}$ and loop over:

$$\mathbf{Z} = \mathbf{W}^{+} \cdot \mathbf{X}, \qquad \mathbf{W} = \mathbf{X} \cdot \mathbf{Z}^{+}$$

  - After convergence $\mathbf{W}$ spans the PCA space
  - For low rank $\mathbf{W}$ the computations are much faster!
    - More later when we cover EM

Approximate eig/SVD methods

- Rich literature on approximate methods
  - E.g. Lanczos, conjugate gradient, etc.
  - Much faster than regular eig/SVD
    - Especially for estimating a few components
- What to use:
  - MATLAB: `eigs()` and/or `svds()`
  - Python: `scipy.sparse.linalg.eigs()`, `scipy.sparse.linalg.svds()`
    - For symmetric inputs use `scipy.sparse.linalg.eigsh()`

PCA for online data

- Sometimes we have too many data samples
  - Irrespective of the dimensionality
  - e.g. long video recordings

- Incremental SVD algorithms
  - Update the $\mathbf{U}$, $\mathbf{S}$, $\mathbf{V}$ matrices using only a small data subset or even a single sample point
  - Very efficient updates

PCA for online data II

- "Neural net" algorithms
  - Algorithms that update $\mathbf{W}$ one sample at a time
  - Gradient methods

- E.g. Sanger's rule:

$$\Delta\mathbf{W} = \mathbf{W} \cdot \mathbf{x}_i \cdot \left(\mathbf{x}_i - \mathbf{W}^\top \cdot \mathbf{W} \cdot \mathbf{x}_i\right)^\top$$

$$\mathbf{W} = \mathbf{W} + \mu \Delta\mathbf{W}$$

Back to signals and human perception

- We already talked about perceptual features
  - And how they correlate with classical DSP

- Is there a connection to make with PCA?

An example

- Let's take a time series which is not "white"
  - Each sample is somewhat correlated with the previous one (Markov process)

$$\left[\; x(t), \quad \cdots, \quad x(t+T) \; \right]$$

- We'll make it multidimensional

$$\mathbf{X} = \begin{bmatrix} x(t) & x(t+1) & \\ \vdots & \vdots & \cdots \\ x(t+N) & x(t+1+N) & \end{bmatrix}$$

An example

- In this context, features will be repeating temporal patterns smaller than N

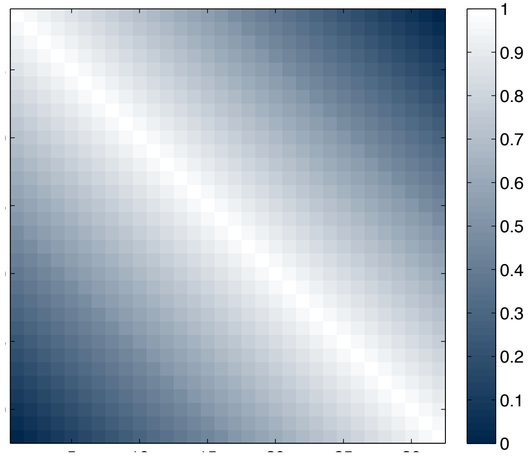
$$\mathbf{Z} = \mathbf{W} \cdot \begin{bmatrix} x(t) & x(t+1) & \\ \vdots & \vdots & \cdots \\ x(t+N) & x(t+1+N) & \end{bmatrix}$$

- If $\mathbf{W}$ is the Fourier matrix then we are performing a time/frequency analysis

PCA on the time series

- By our definition there is a correlation between successive samples:

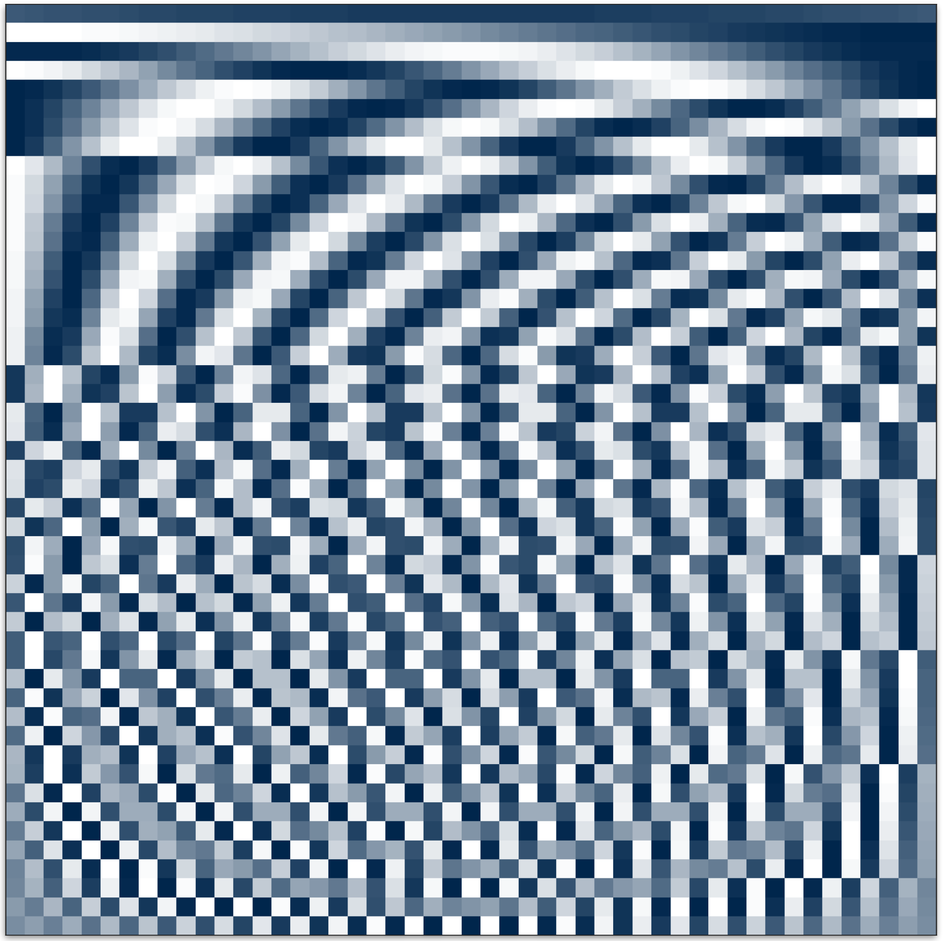
$$\mathrm{Cov}\left(\mathbf{X}\right) \approx \begin{bmatrix} 1 & 1-e & \cdots & 0 \\ 1-e & 1 & 1-e & \vdots \\ \vdots & 1-e & 1 & 1-e \\ 0 & \cdots & 1-e & 1 \end{bmatrix} =$$



- Resulting covariance matrix will be symmetric Toeplitz, tapering from one towards zero

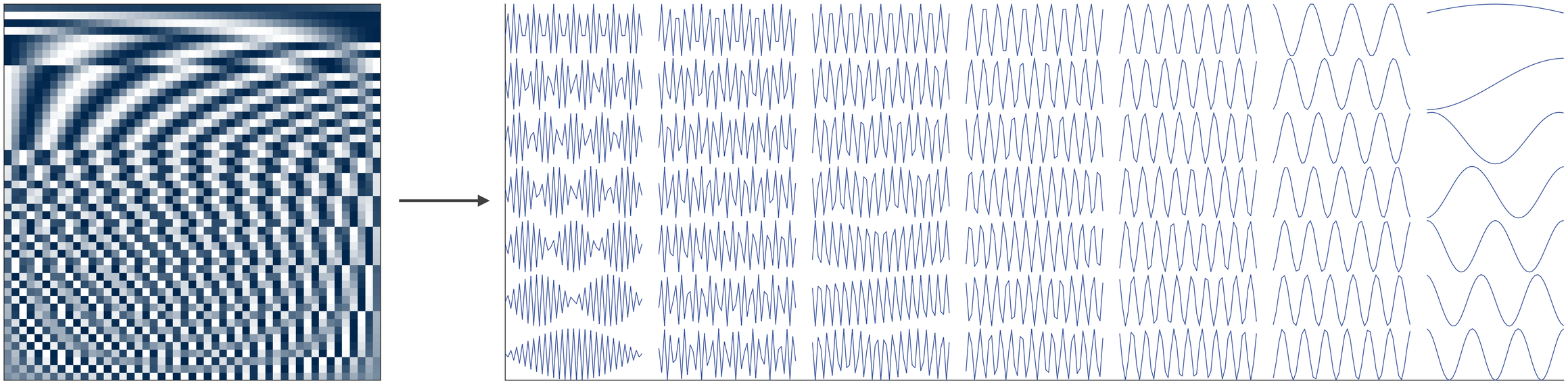
The resulting eigenvectors

- What does this look like? (approximately)



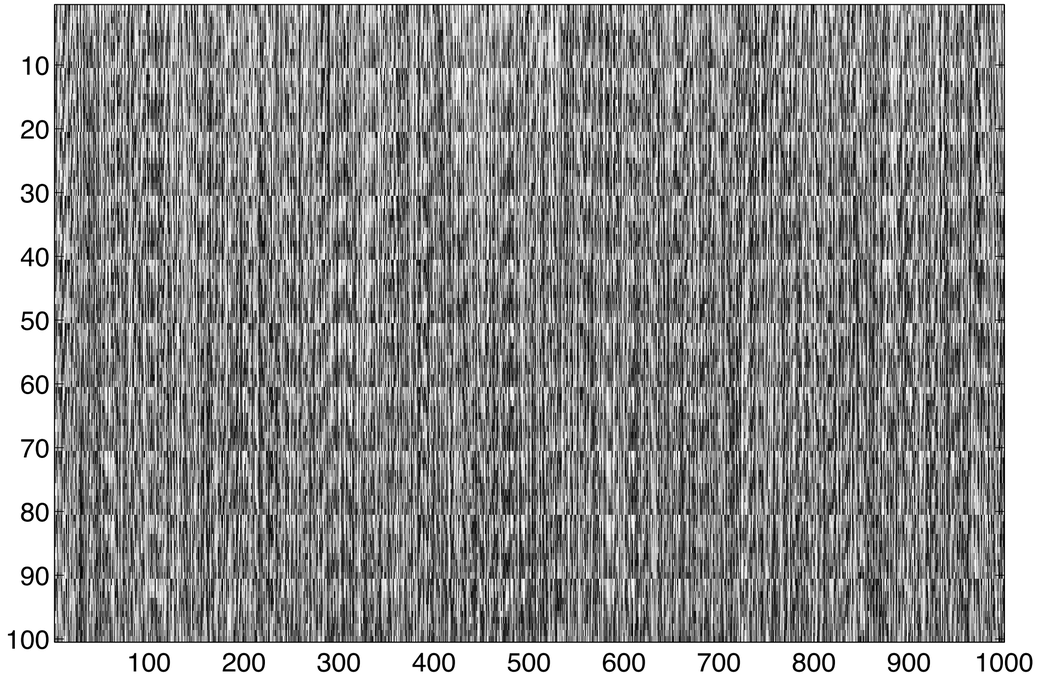
From PCA to a frequency transform

- The eigenvectors of Toeplitz matrices are (approximately) sinusoids of varying frequencies



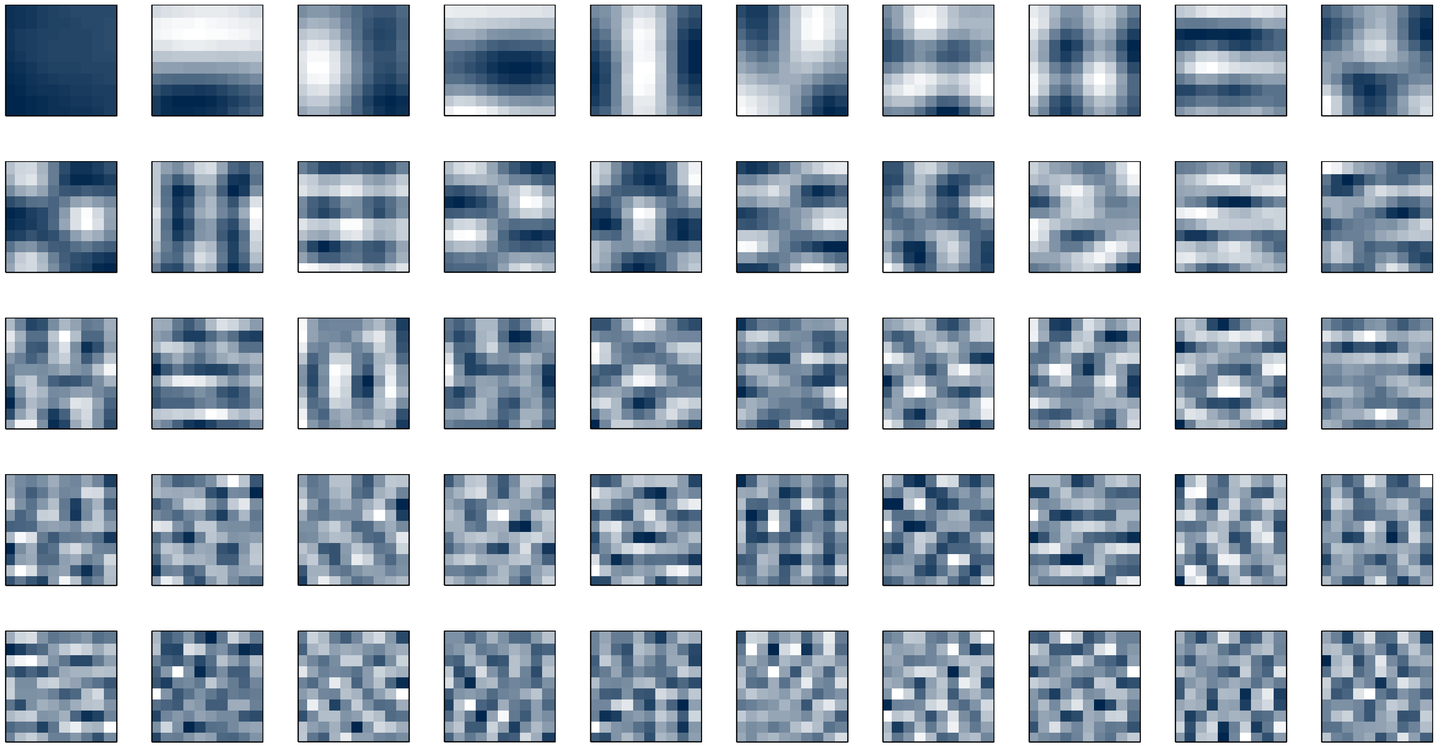
What about images?

- Take lots of pictures, pick random local patches
  - What are the features of this data set?



The components of image patches

- Do PCA on patch matrix and rewrap the components
  - What is this?



2D bases to 1D

- Remember the 2D Fourier transform?
  - We left/right multiply with the Fourier matrix

$$\mathbf{Y} = \mathbf{F} \cdot \mathbf{X} \cdot \mathbf{F}$$

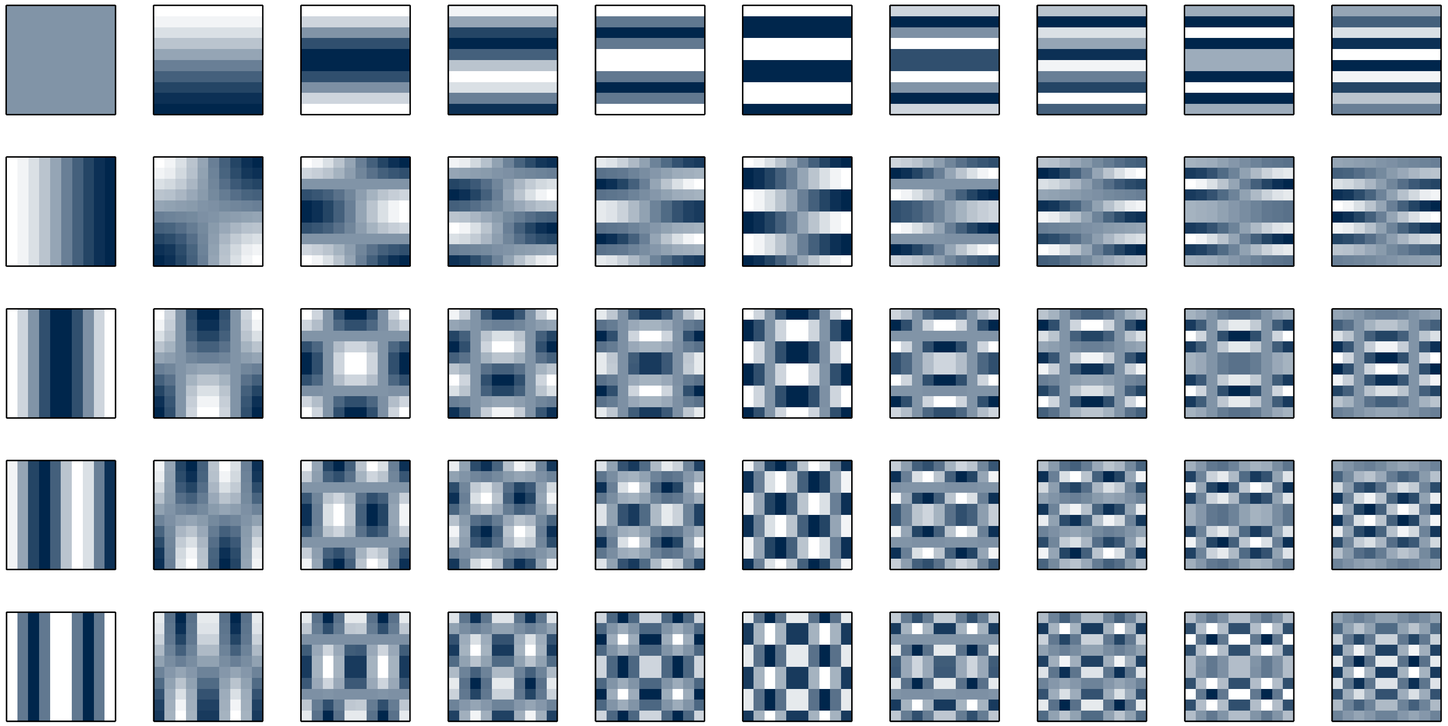
- Which in 1D translates to

$$\mathrm{vec}\left(\mathbf{Y}\right) = \mathrm{vec}\left(\mathbf{F} \cdot \mathbf{X} \cdot \mathbf{F}\right) = \left(\mathbf{F}^{\top} \otimes \mathbf{F}\right) \cdot \mathrm{vec}\left(\mathbf{X}\right)$$

- So what does that matrix look like?

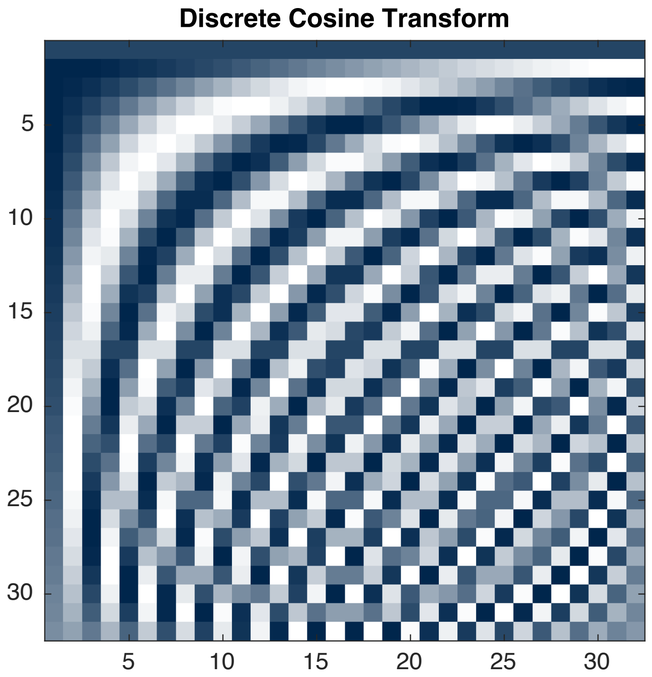
2D sinusoidal bases on 1D

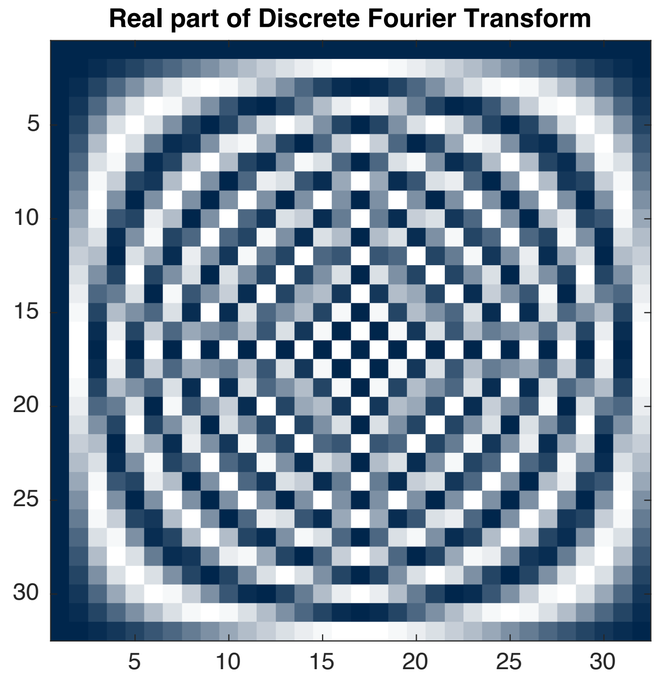
- Looks like the PCA components from images!



Not quite a Fourier transform

- Real-valued bases are sinusoids of various frequencies
  - Why not complex-valued? You get these with circulant covariance
- This is known as the Discrete Cosine Transform (DCT)





So now you know

- A sinusoidal transform is an “optimal” decomposition for time series
  - In fact, you will often not do PCA and do a DCT
- There is also a connection with our perceptual system
  - We employ similar filters in our ears and eyes (but we’ll make that connection later)

Comparing the DCT with PCA

- How to compress a picture
  - Chop into small pieces
  - Perform either:
    - DCT transform
    - PCA transform
  - Keep only large weights
  - Resynthesize using approximation

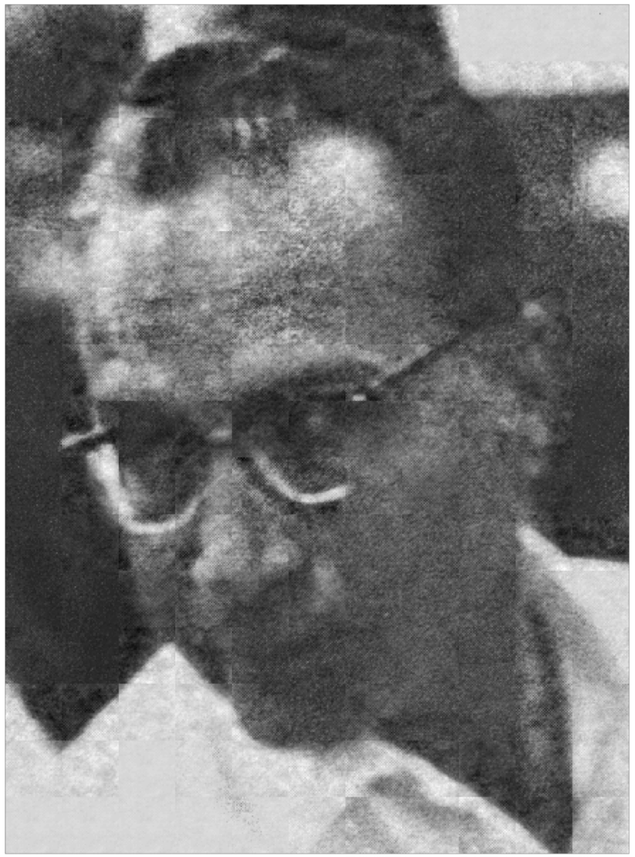
Comparing the DCT with PCA

Original Input

JPEG version

0.14 sec

PCA version



2.1 sec

Recap

- Principal Component Analysis
  - Get used to it!
  - Decorrelates multivariate data, finds useful components, reduces dimensionality
- Many ways to get to it
  - Knowing what to use with your data helps
- Interesting connection to Fourier transform

Next lecture

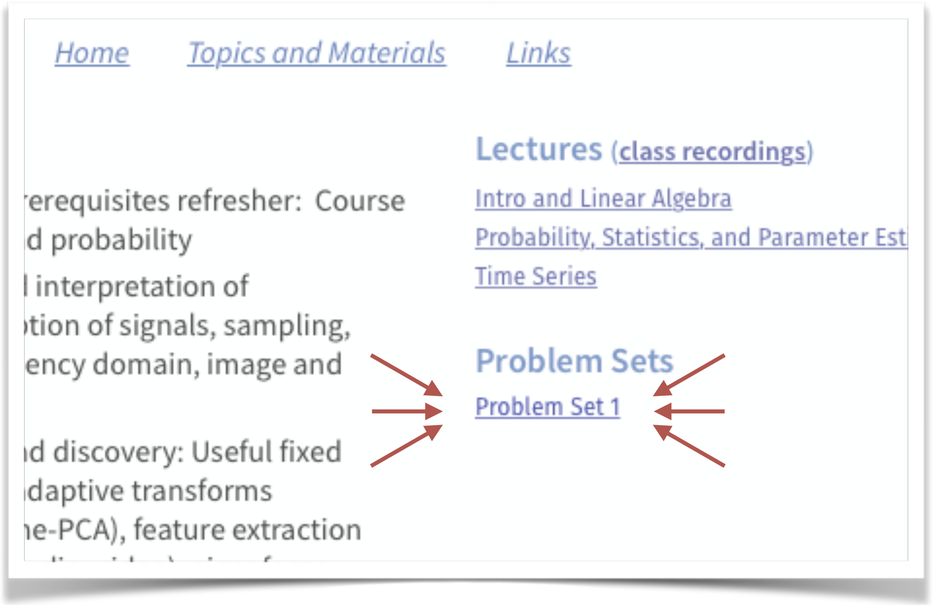
- Continuing with the features theme
- Independent Component Analysis
  - "PCA on steroids"
  - Strong statistical tool, very interesting properties and connections
- Other useful feature transforms

Reading

- Textbook sections 6.1-6.4
- Eigenfaces (optional)
  - http://en.wikipedia.org/wiki/Eigenface
  - http://www.cs.ucsb.edu/~mturk/Papers/mturk-CVPR91.pdf
  - http://www.cs.ucsb.edu/~mturk/Papers/jcn.pdf
- Incremental SVD (optional)
  - http://www.merl.com/publications/TR2002-024/
- EM-PCA (optional)
  - http://cs.nyu.edu/~roweis/papers/empca.ps.gz

Reminder

- There is homework sent out already
  - If you haven't received any class email let me know now!!



  - See your TA for questions early!