CS545 – Machine Learning for Signal Processing

Graphs in Signal Processing and Machine Learning

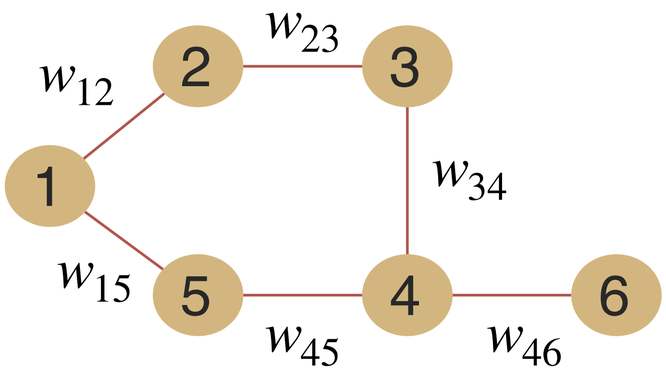
November 13 2023

Today's lecture

- Using graphs instead of vectors/matrices
- Graph signal processing basics
- Graph machine learning

A Graph

- A Graph is a set of vertices (nodes) and edges
- Each node has a feature or *message*
  - Messages move through nodes to distribute information
- Edges may have associated weights
  - Messages are weighted as they move through the edge
- Graphs can be directed or undirected
  - *Undirected*: 1→2 is equivalent to 2→1
  - *Directed*: 1→2 and 2→1 have different weights
- Good for representing pairwise relationships



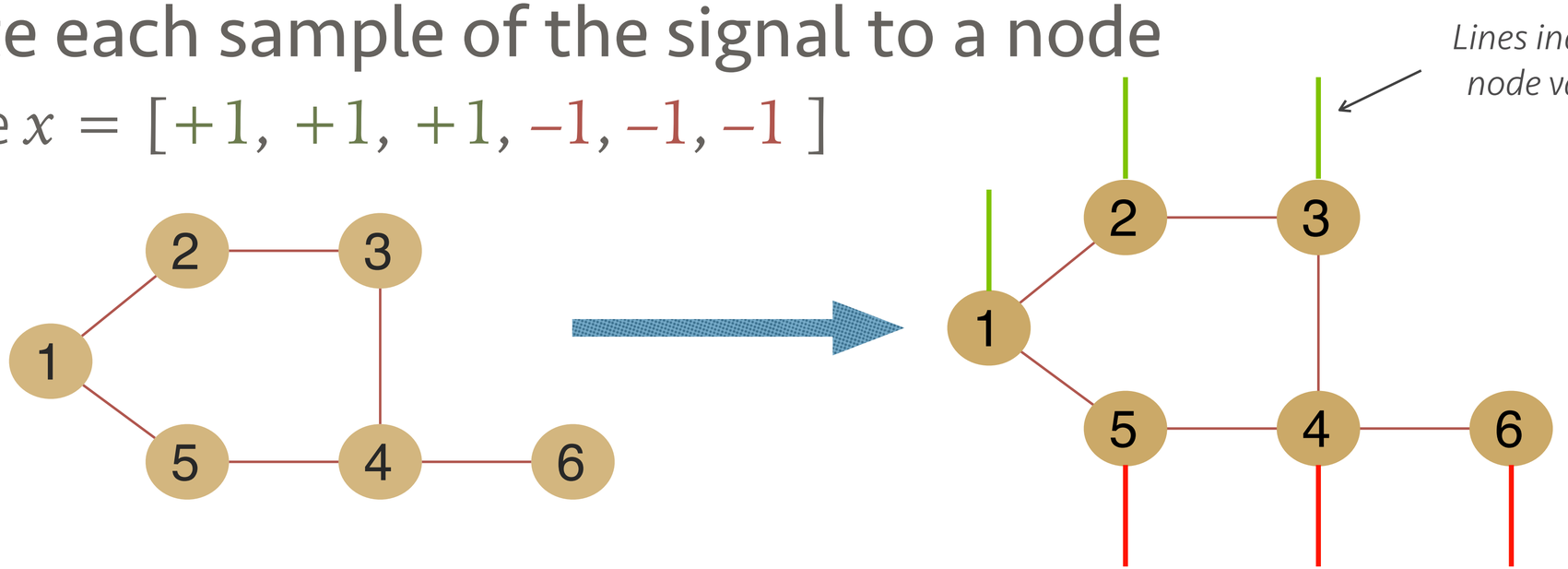
Graph ↓ $G = \{V, E\}$ ↓ *Edges*

↑

Vertices or Nodes

Representing a signal as a graph

- Associate each sample of the signal to a node
  - Example $x = [+1, +1, +1, -1, -1, -1]$



- The signal sample values become node values or features
  - The sample relationships come from the graph
- Graphs are a good representation of irregular signals
  - Edges give us structure, nodes hold samples

Graph Representations

- Symbolically, $G = \{V, E\}$ completely defines a graph
  - But it's an inconvenient representation for us
- Representing a graph as a familiar data structure
  - Use an *adjacency matrix* $\mathbf{W}$ of size $N \times N$
    - N is the number of vertices
    - The elements of $\mathbf{W}$ give the weights of the edges
    - Undirected graphs have a symmetric adjacency matrix
  - Pack all node values into an N-dimensional vector $\mathbf{v}$
    - i-th's node value is in v_i

Examples

Undirected Path Graph

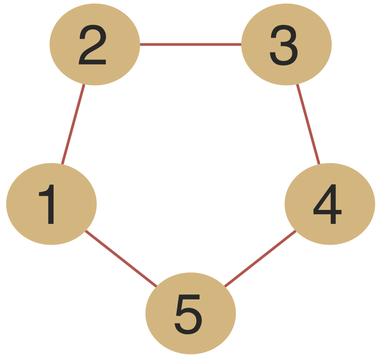
$$\mathbf{W} = \begin{matrix} \\ \textit{from} \end{matrix} \overset{\textit{to}}{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}}$$

Directed Path Graph

$$\mathbf{W} = \begin{matrix} \\ \textit{from} \end{matrix} \overset{\textit{to}}{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}$$

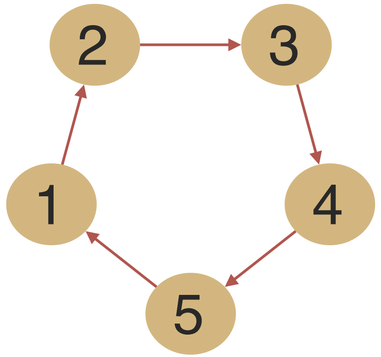
Examples

Undirected Ring Graph



$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

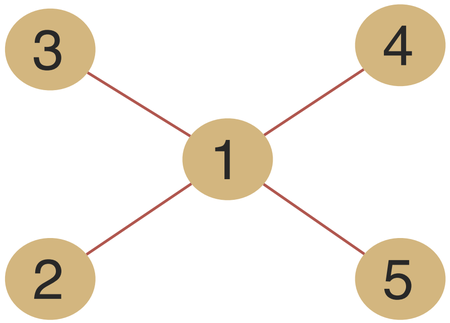
Directed Ring Graph



$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

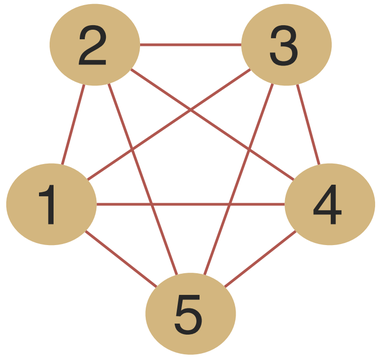
More Examples

Undirected Star Graph



$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Undirected Fully Connected Graph

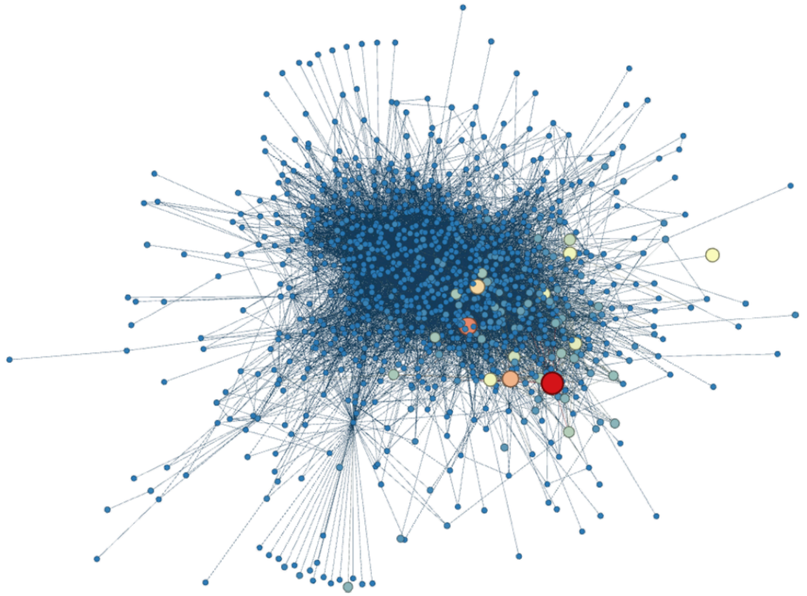


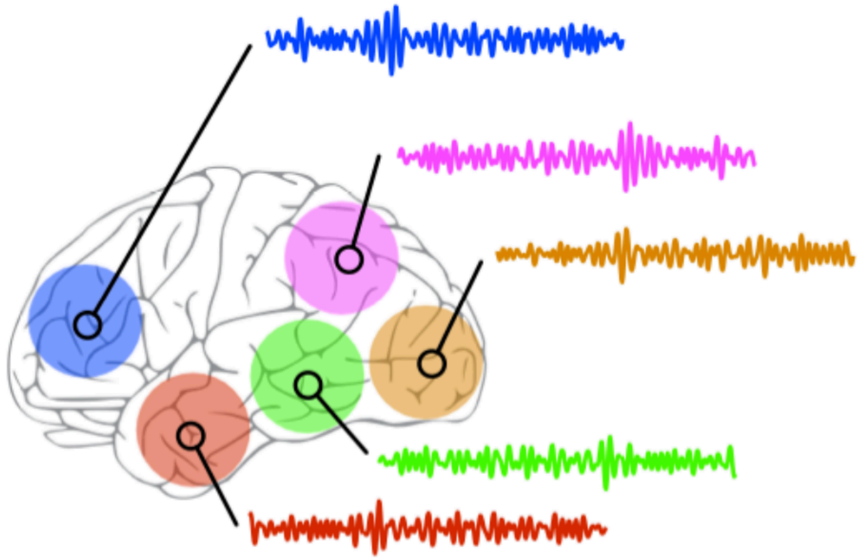
$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Real-world Examples

Social Network Graph

Vertices: 1,000 Twitter users
Edges: Relationship between users
Signal: Apple-related hash-tags in a 6 week period





Brain Imaging

Vertices: Different brain regions
Edges: Structural connectivity between regions
Signal: Blood-oxygen-level series

Graph Signal Processing

- How do we process such representations?
  - *Graph Signal Processing*
- Classical DSP assumes “regular” data
  - GSP generalizes classical DSP to graph signals
- GSP also helps define learning on graphs
  - Make signal processing parameters trainable

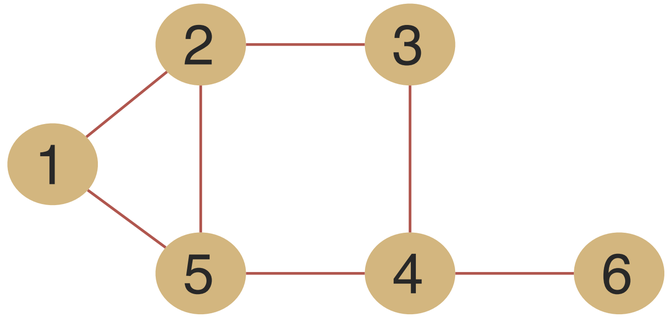
New domains to worry about

- Classical signal processing
  - Time domain
    - Natural representation (e.g. pixels)
    - Simple relationship between samples
  - Frequency domain
    - Fourier Transforms, Cosine Transforms etc.
    - Signal properties can be easily analyzed
- Graph signal processing
  - Vertex (spatial) domain
    - Described using the *adjacency matrix*
    - Arbitrary sample relations
  - Graph Spectral domain
    - *Graph Fourier Transform*
    - Important for analysis of signal properties

- How do we move from spatial to spectral domain?
  - We need the *Graph Laplacian matrix* $\mathbf{L}$

The Graph Laplacian

- Consider an undirected graph
  - Assume that all the weights are equal to 1



Graph Laplacian *Degree Matrix* *Graph Adjacency*

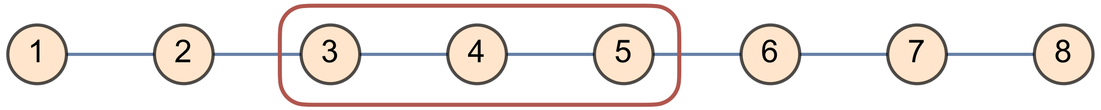
$$\begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{L} \qquad \mathbf{D} = \text{diag}(\mathbf{W} \cdot \mathbf{1}) \qquad \mathbf{W}$$

(diag of sum or $\mathbf{W}$*'s rows)*

For a simple graph

- Consider a very simple signal
  - 1d regularly sampled time series

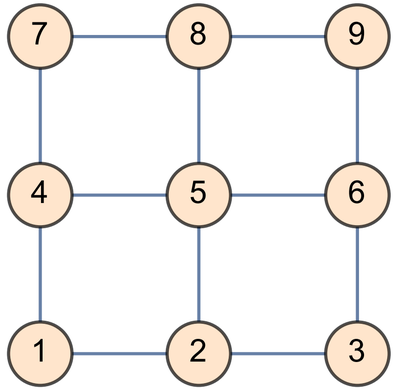


Graph Laplacian = *Degree Matrix* − *Graph Adjacency*

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Which generalizes

- For a regular 2D grid we see something similar
  - It's an edge filter!
  - Technically, the Laplace operator



Graph Laplacian

$$\begin{pmatrix} 2 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 3 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 3 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{pmatrix} =$$

Degree Matrix

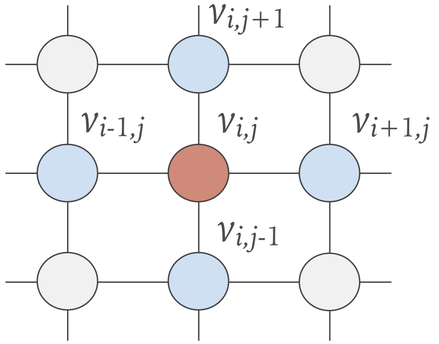
$$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix} -$$

Graph Adjacency

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

The Graph Laplacian: Properties

- Approximates the Laplace operator on the graph
  - Captures local geometry
- For undirected graphs, $\mathbf{L}$ is symmetric
- Off-diagonal entries are non-positive
- Rows add up to zero
  - This is due to the degree matrix $\mathbf{D}$
- Normalized Laplacian
  - We will stick to using $\mathbf{L}$



$$\begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{L}_{\text{norm}} = \mathbf{D}^{-\frac{1}{2}} \cdot (\mathbf{D} - \mathbf{W}) \cdot \mathbf{D}^{-\frac{1}{2}}$$

The Graph Fourier Transform

- Given a graph containing a signal $\mathbf{s}$
  - Get graph Laplacian $\mathbf{L}$
    - $\mathbf{L}$ has a complete set of orthonormal eigenvectors
      - Sorted by the magnitude of the eigenvalues

$$\mathbf{L} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^{\top}$$

Graph equivalent of Fourier bases — *Graph Fourier spectral coefficients*

- Transforming the graph signal

Signal Fourier coefficients $\longrightarrow \mathbf{S} = \mathbf{V}^{\top} \cdot \mathbf{s} \longleftarrow$ *Signal spatial coefficients*

Graph Fourier Transform

What does this mean?

- For the k-th eigenvalue/vector pair:

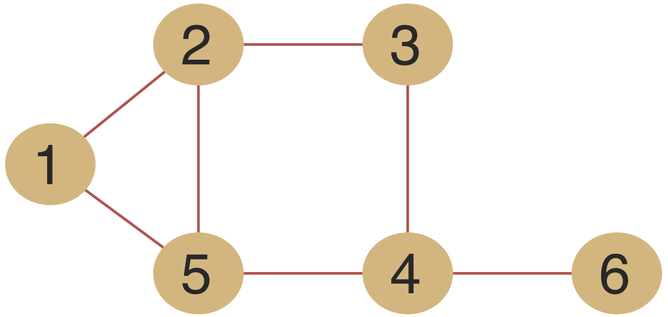
$$\lambda_k = \mathbf{u}_k^\top \cdot \underline{\mathbf{L} \cdot \mathbf{u}_k} \text{ with } \mathbf{u}_k^\top \cdot \mathbf{u}_k = 1$$

$\mathbf{L}$ *is a difference filter, this product measures local change in* $\mathbf{u}$

- Bases $\mathbf{u}$ capture "local variation"
  - "How much" does a signal change in a neighborhood
  - Lower eigenvalue $\longrightarrow$ smoother signal $\longrightarrow$ lower "frequency"
  - $\mathbf{u}$ ordering will be in terms of local variation
    - i.e. amount of high frequency

Example

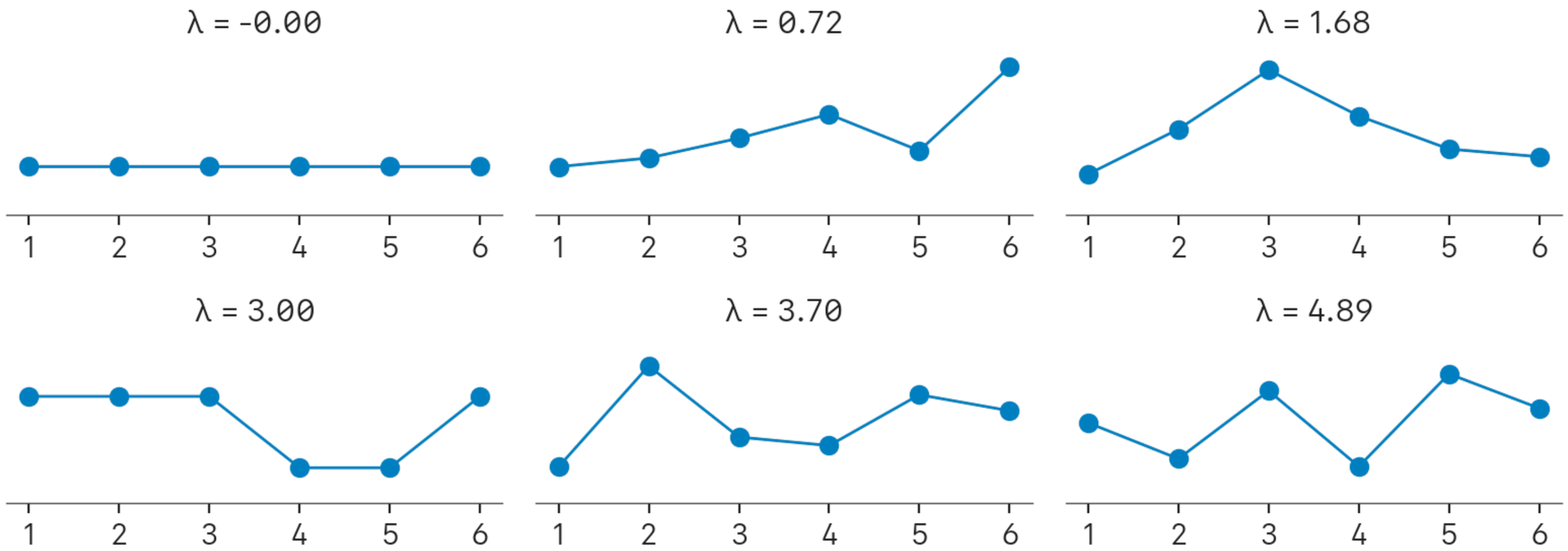
Graph



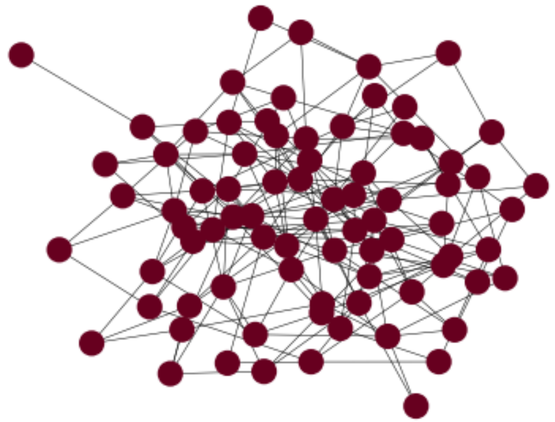
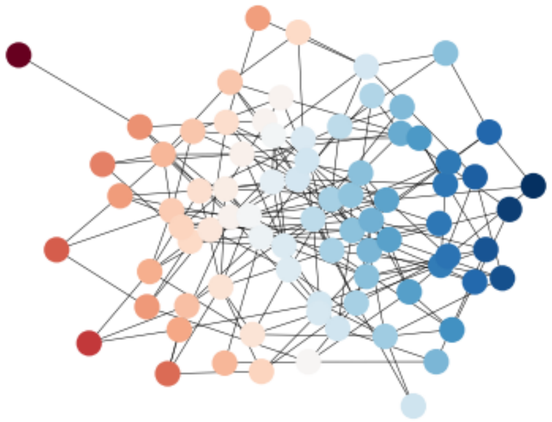
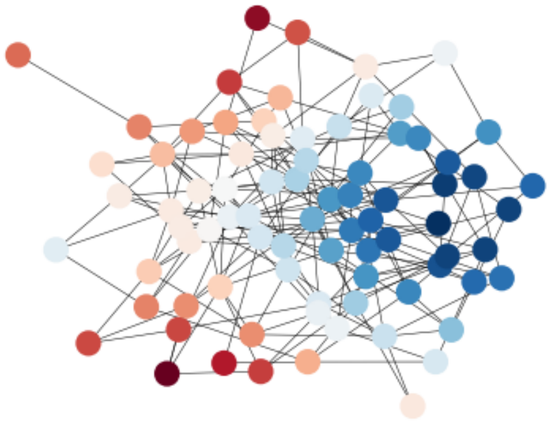
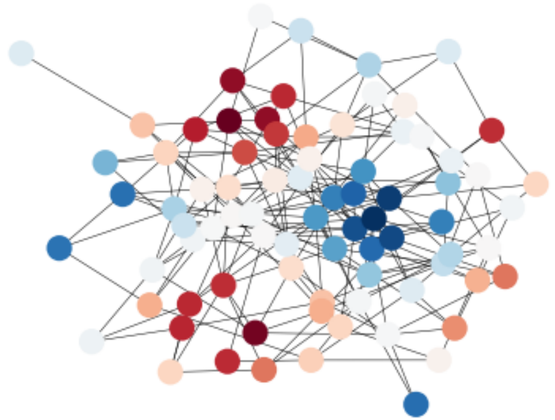
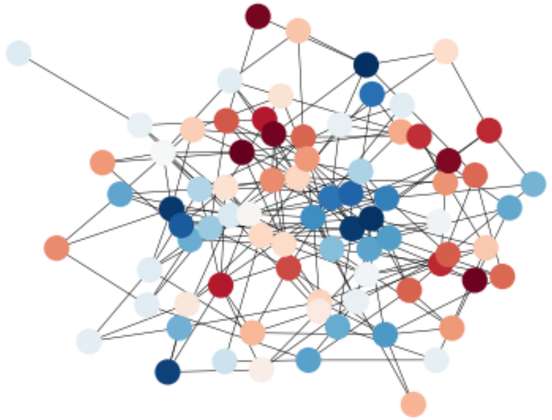
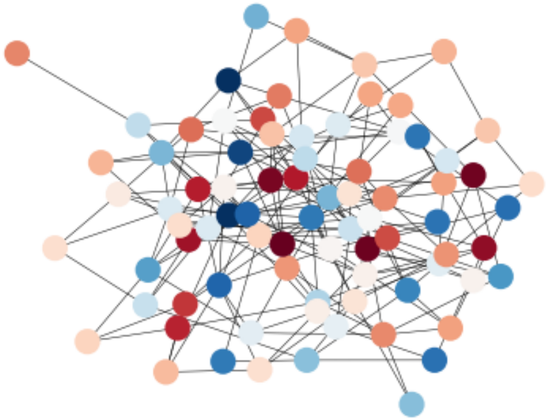
Laplacian

$$\mathbf{L} = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

Eigenvectors/eigenvalues

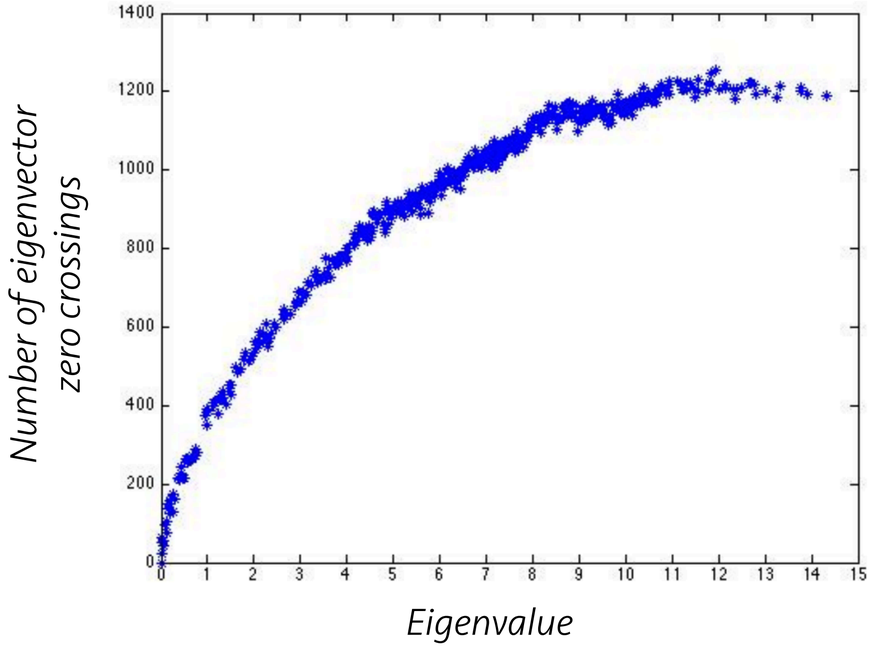


Low and high frequency graph signals

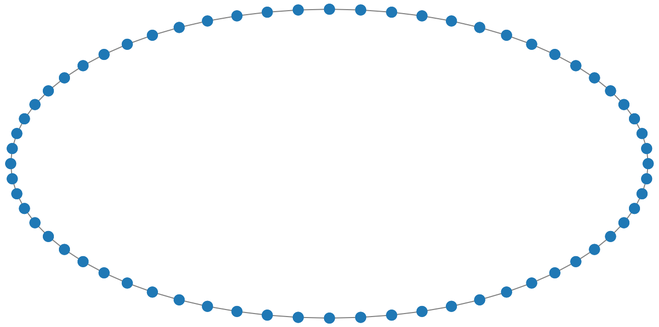
A new notion of "frequency"

- The eigenvalues measure graph "frequency" content
  - How much the corresponding samples change
- Eigenvectors are ranked by their eigenvalues
  - The initial eigenvectors are smooth and change slowly
  - Later eigenvectors change quickly
  - Similar to classical DSP frequency

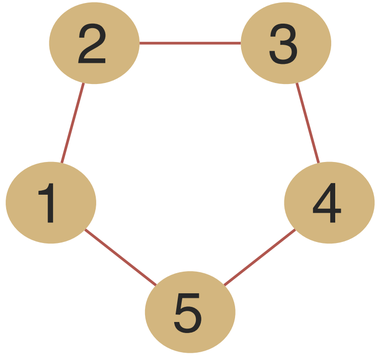


Special Case I: DFT

- Consider an undirected Ring graph
  - Assume all the weights are equal to unity



Ring graph
64 Vertices



Ring graph
5 Vertices

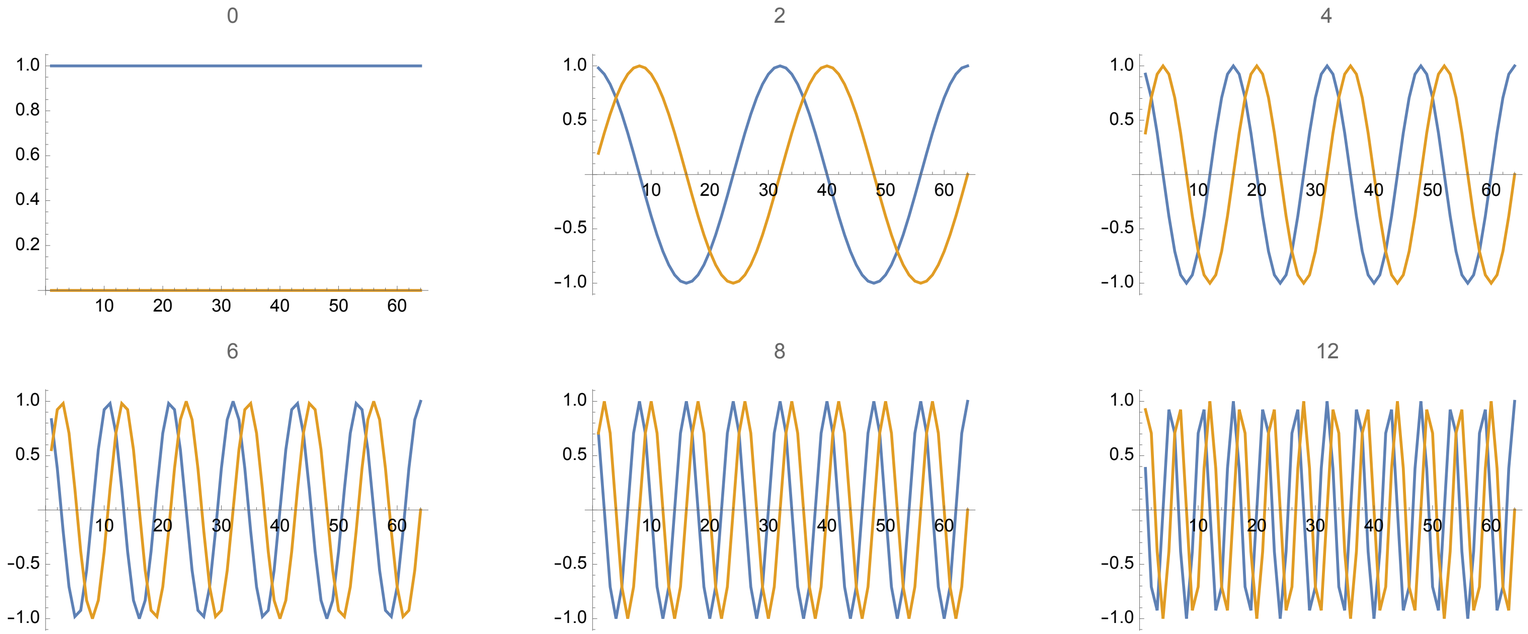
$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Adjacency
5 Vertices

- The adjacency matrix is *circulant*
  - And it's eigenvectors are DFT's complex exponentials

Special Case I: DFT

- DFT bases/eigenvectors: $u_k(n) = e^{i\frac{2\pi kn}{N}}$



- Paired sine and cosine terms
  - Similar local variation
  - Higher frequency for larger eigenvalues

Special Case II: DCT

- Consider an undirected Path graph
  - Assume all the weights are equal to unity

Path graph
5 Vertices

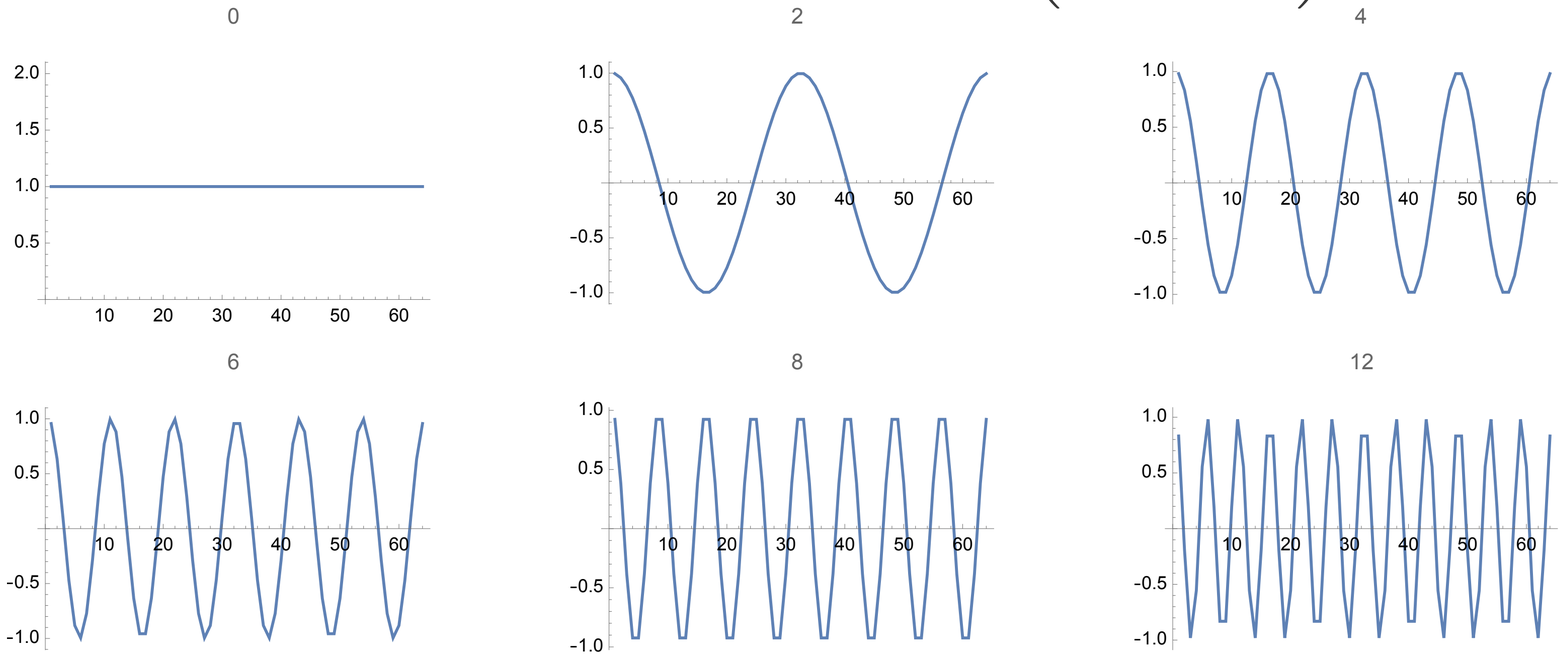
$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Adjacency
5 Vertices

- The adjacency matrix is *Toeplitz*
  - Eigenvectors will be sinusoids

Special Case II: DCT

- DCT-II bases/eigenvectors: $u_k(n) = \sqrt{\frac{2}{N}} \cos\left(\pi k \frac{n - .5}{N}\right)$



- Type 2 Discrete Cosine Transform
  - Real-valued alternative to Fourier Transforms
  - What PCA tends to give us!

Imposing a Graph

- So far
  - Graphs are a collection of nodes and edges
    - Graphs are defined by their adjacency matrices $\mathbf{W}$
  - The graph Laplacian $\mathbf{L}$ can be computed from $\mathbf{W}$
    - Used to compute the Graph Fourier Transform
    - Generalizes classical Fourier Transform theory

- We assumed that the data lived on an underlying graph
  - Can we learn a graph from the data itself?
  - Can audio / images be interpreted as a graph?

- Learning a Graph = Learning an adjacency matrix $\mathbf{W}$
  - Easy to learn adjacency matrices from the data
    - Use a suitable distance metric to get $\mathbf{W}$

Defining a graph from a signal

- Adjacency based on spatial location

$$w_t(i,j) = e^{-a|t_i - t_j|}$$

  - Can be time-indices or spatial coordinates

- Adjacency based on sample values

$$w_v(i,j) = e^{-\beta|v_i - v_j|}$$

  - Can be features, sample-values, image colors etc.

- Overall Adjacency matrix

$$\mathbf{W} = \gamma \mathbf{W}_t + (1-\gamma)\mathbf{W}_v$$

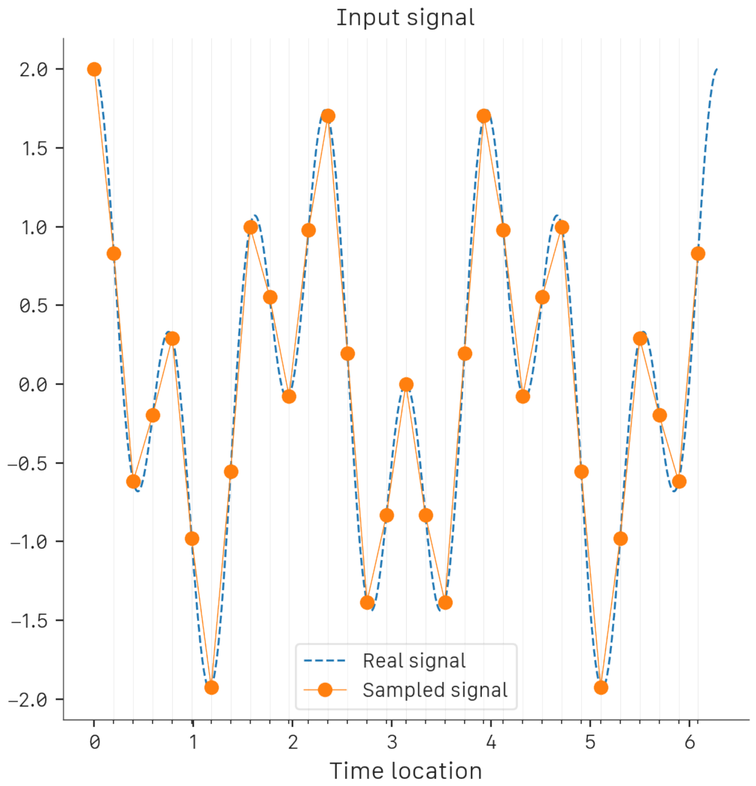
Classical filtering example

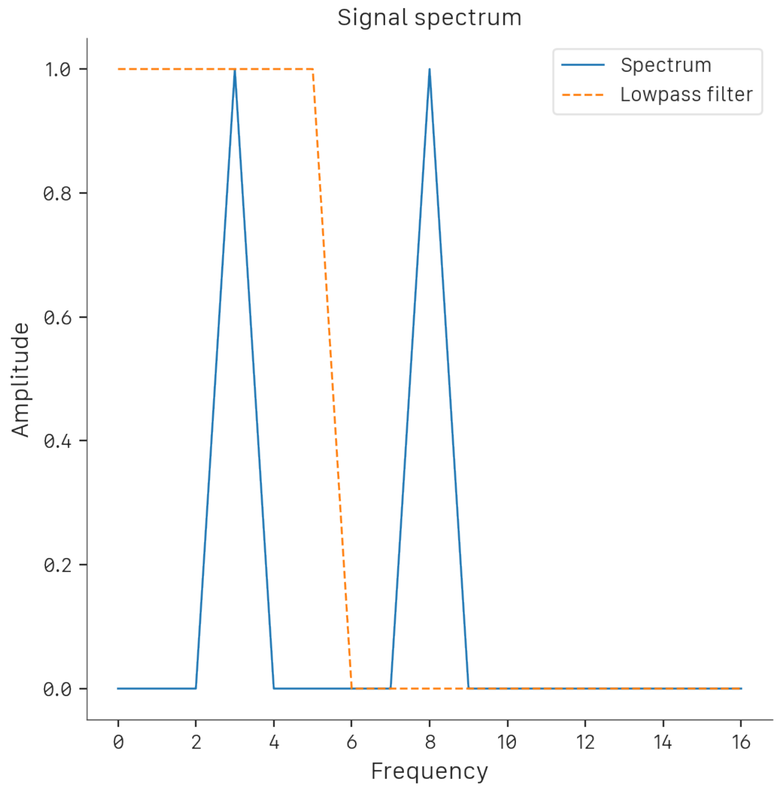
- Applying a filter in the Fourier domain

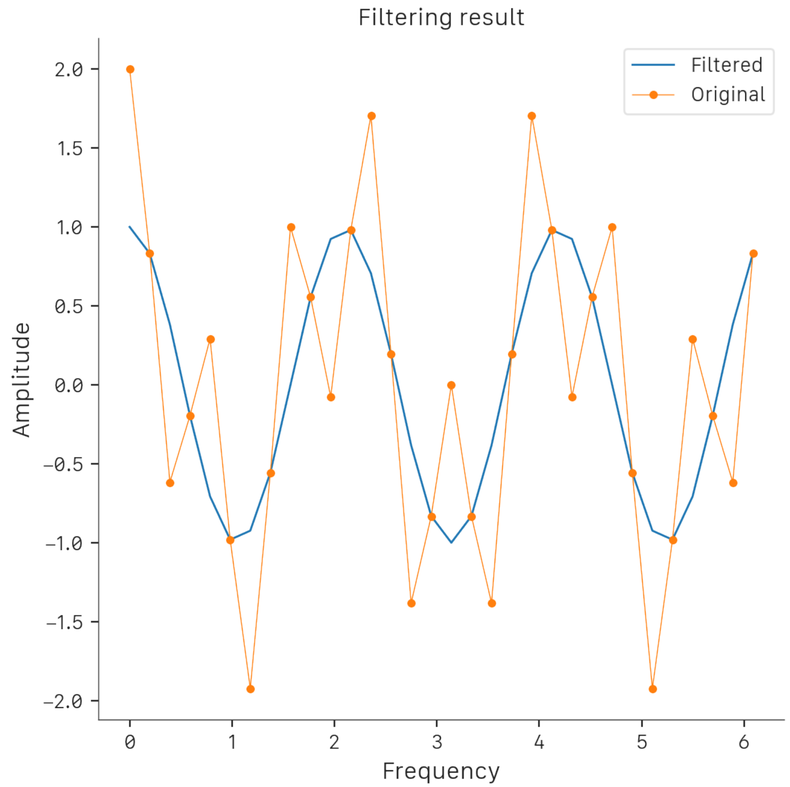
$$x(t) = \cos(3t) + \cos(8t)$$

Transform signal into frequency domain
Apply lowpass filter

Invert to time-domain

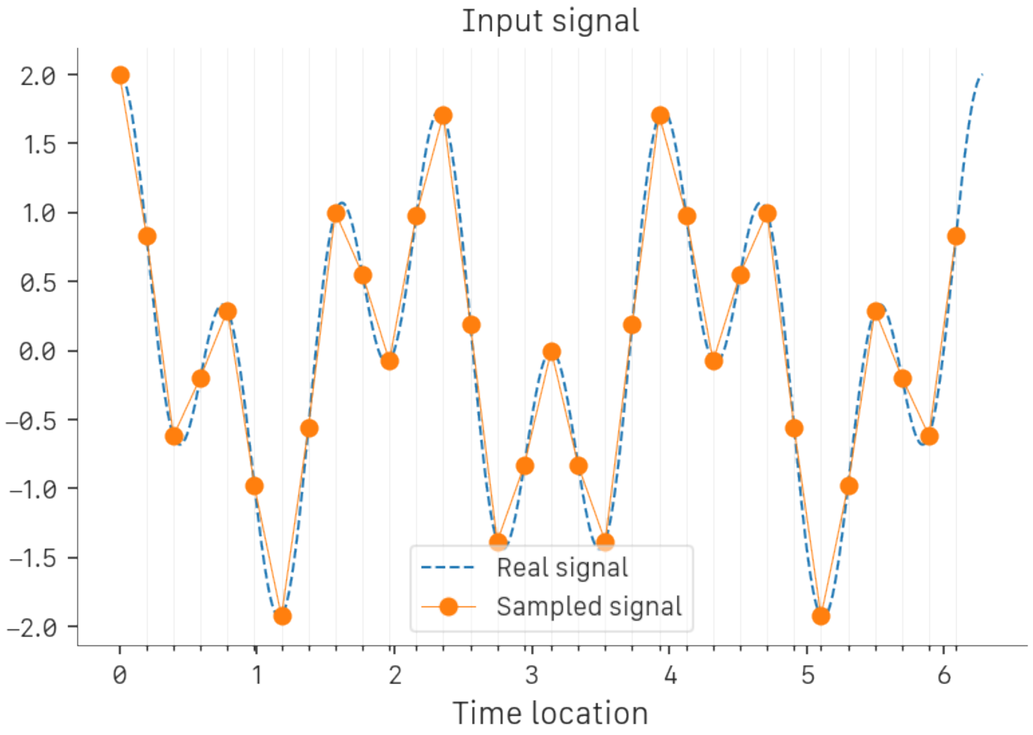


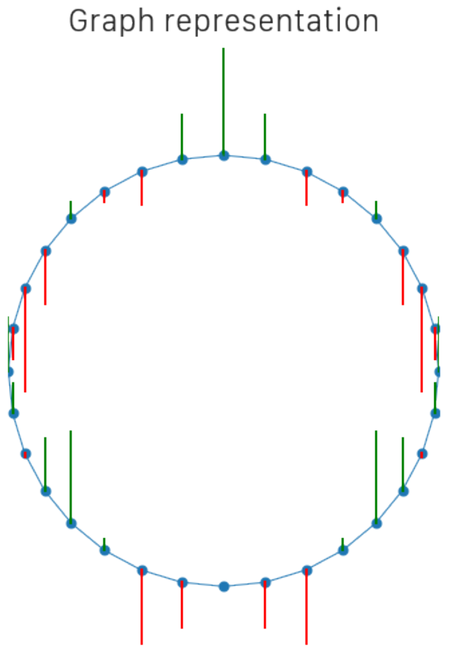


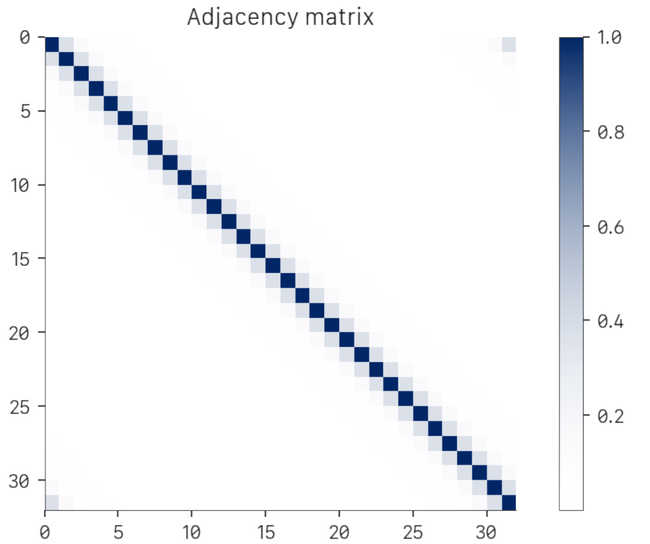


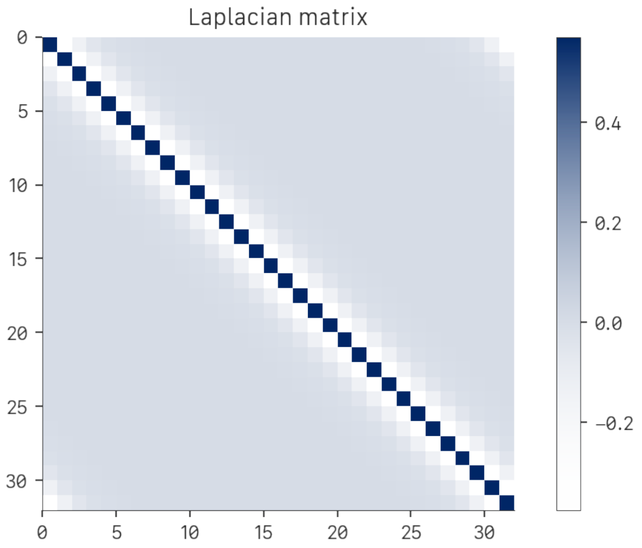
Filtering on Graphs

- Define a simple circular graph
  - Place sampled values in vertices



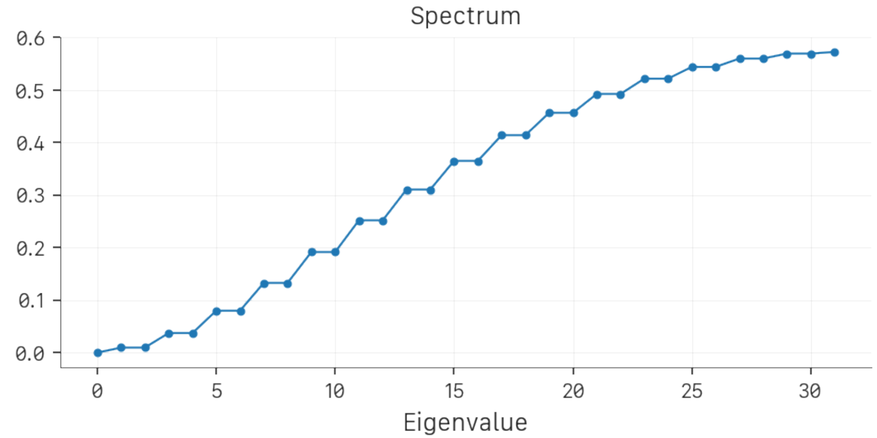


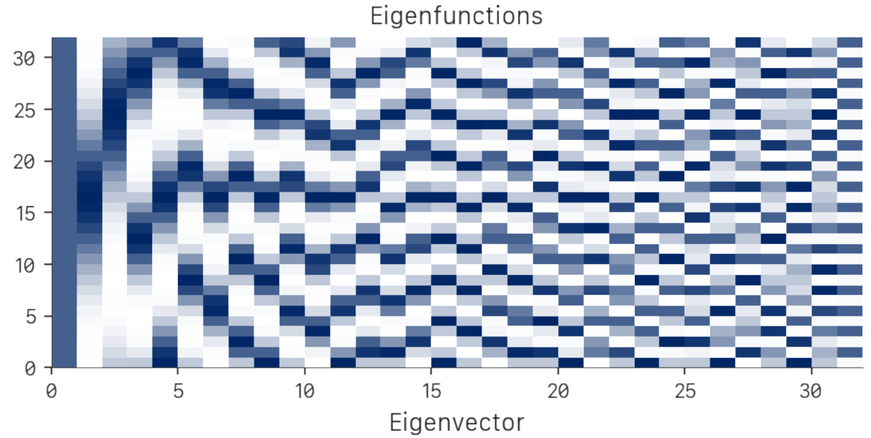


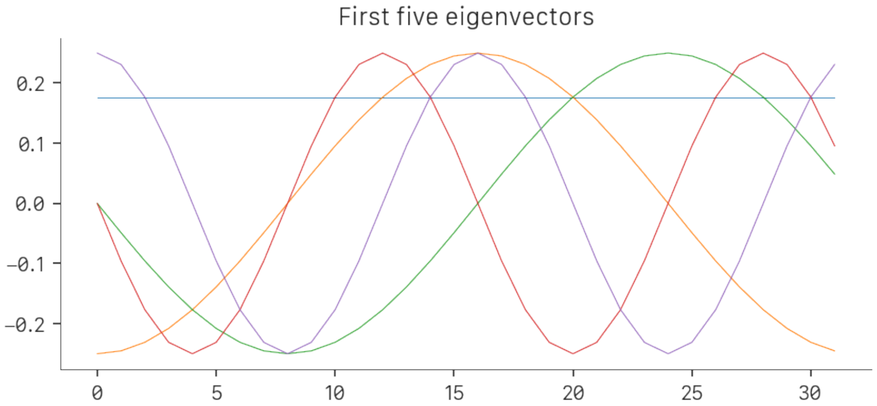


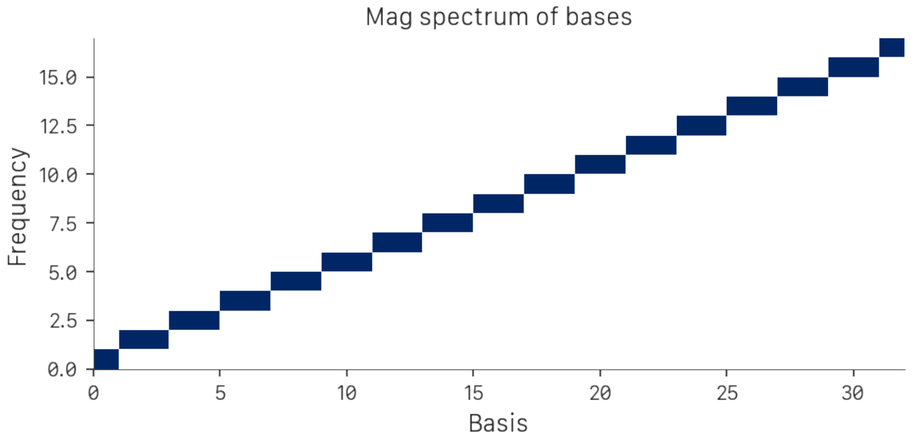
Step 1: Get frequency bases

- Eigendecomposition of the Graph Laplacian
  - Learn a suitable Graph Fourier transform $\mathbf{L} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^{\top}$









Step 2: Transform Data

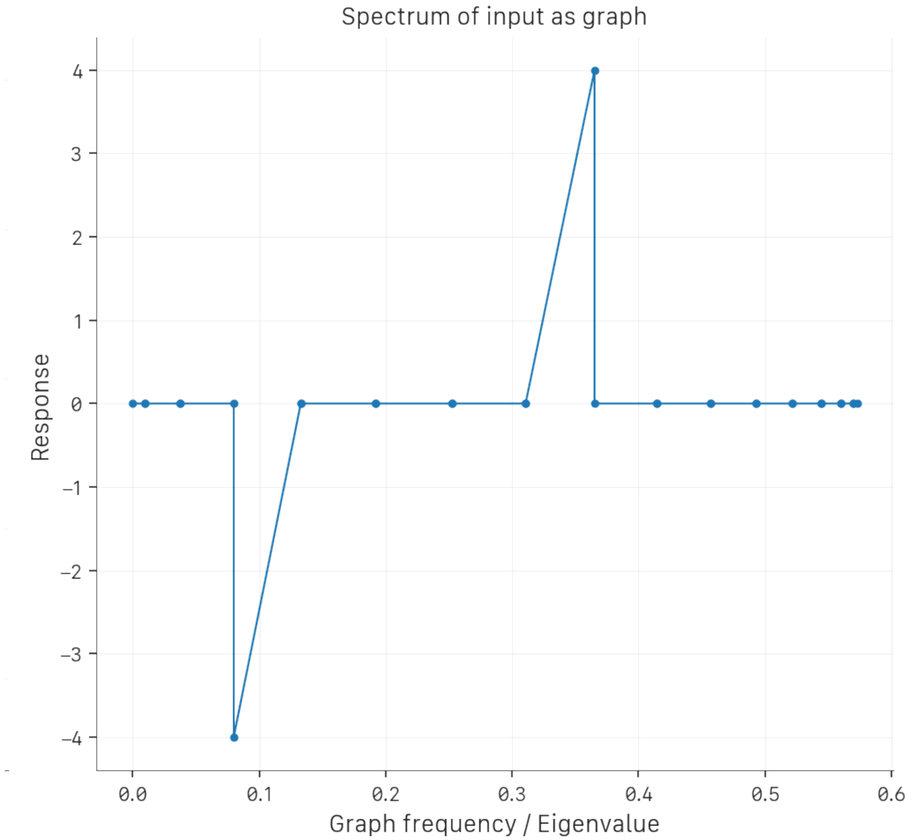
- Transform data into Fourier Domain
  - Use the eigenvectors learned in previous step

Signal Fourier coefficients

$$\mathbf{X} = \mathbf{V}^{\top} \cdot \mathbf{x}$$

Graph Fourier bases

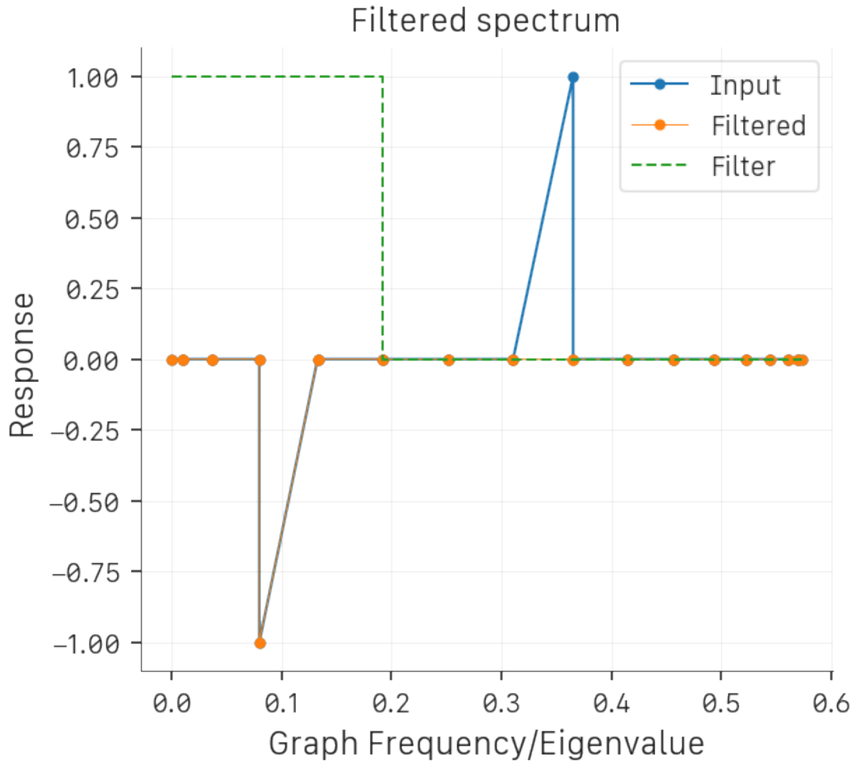
Graph Signal



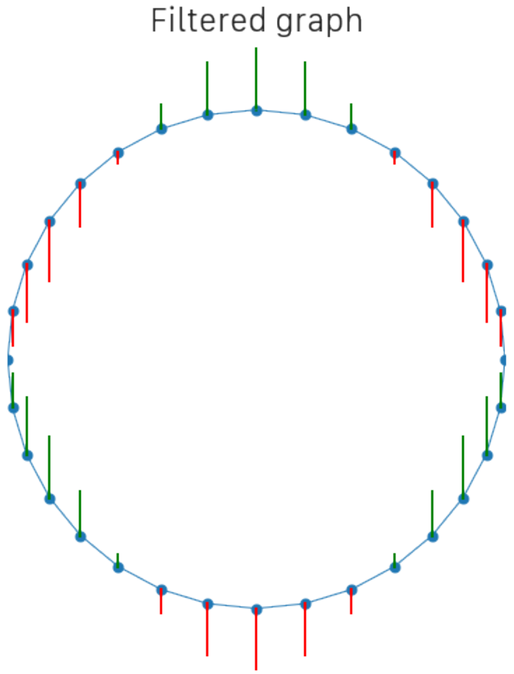
Step 3: Filter and invert

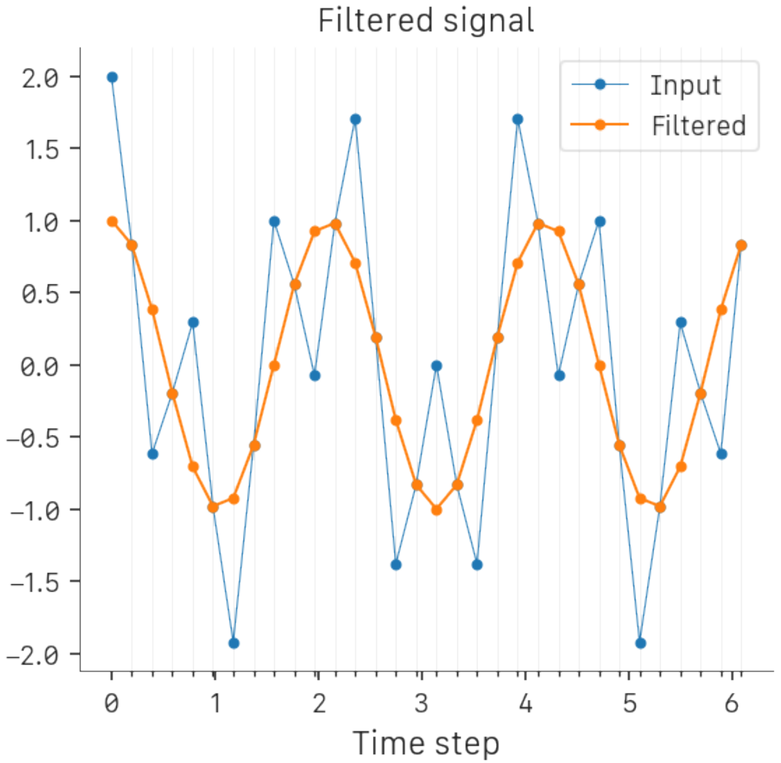
- Design and apply a lowpass filter
  - Use learned bases to invert back to time

$$\mathbf{Z} = \mathbf{F} \odot \mathbf{X}$$



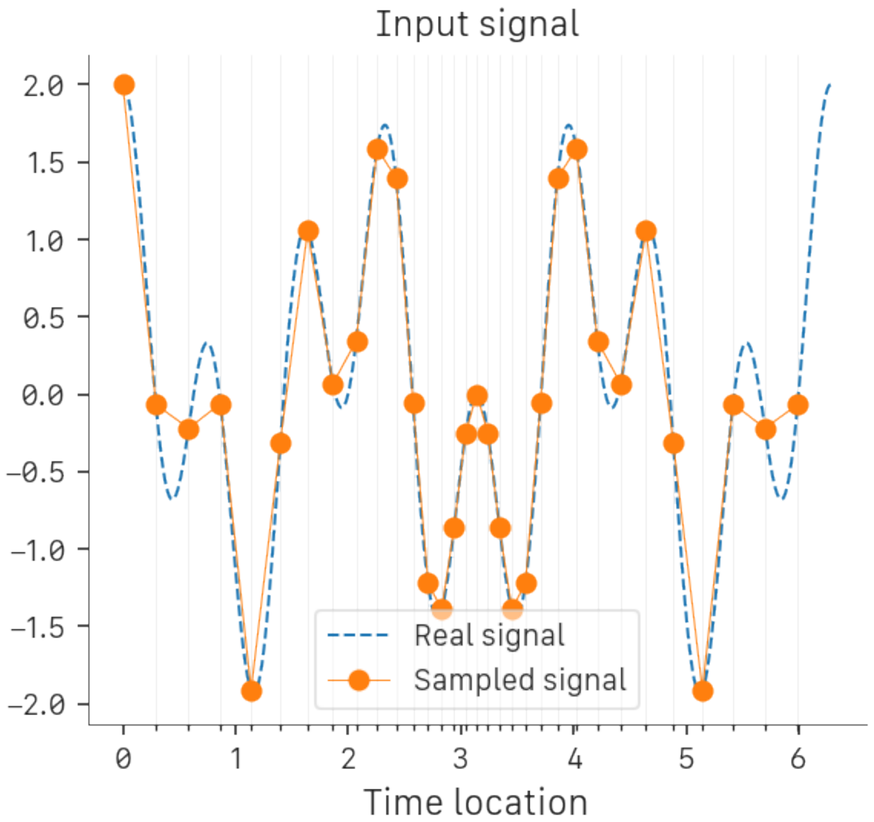
$$\mathbf{z} = \mathbf{V} \cdot \mathbf{Z}$$

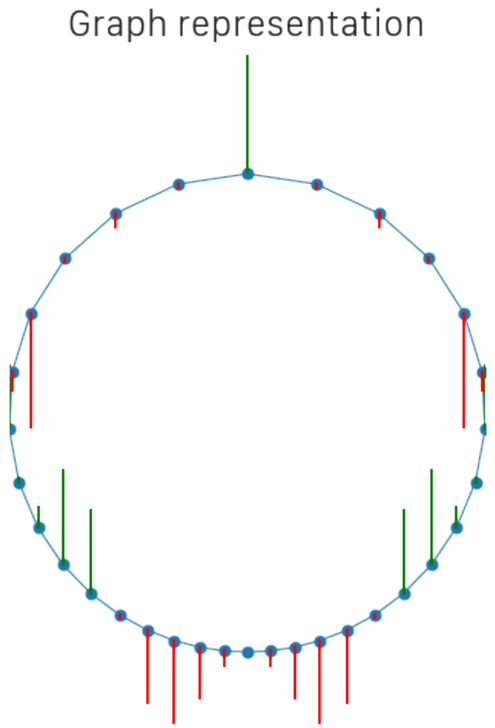




Nothing new so far, but …

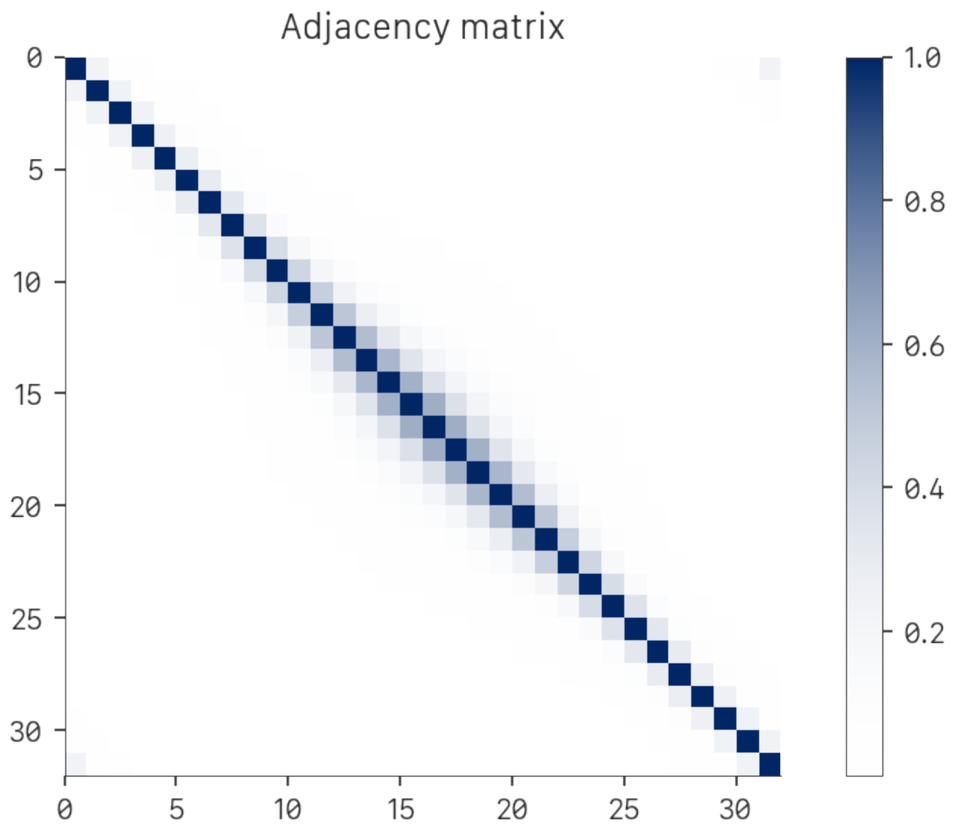
- What if my graph is not as regular?
  - What if the sample rate is variable?

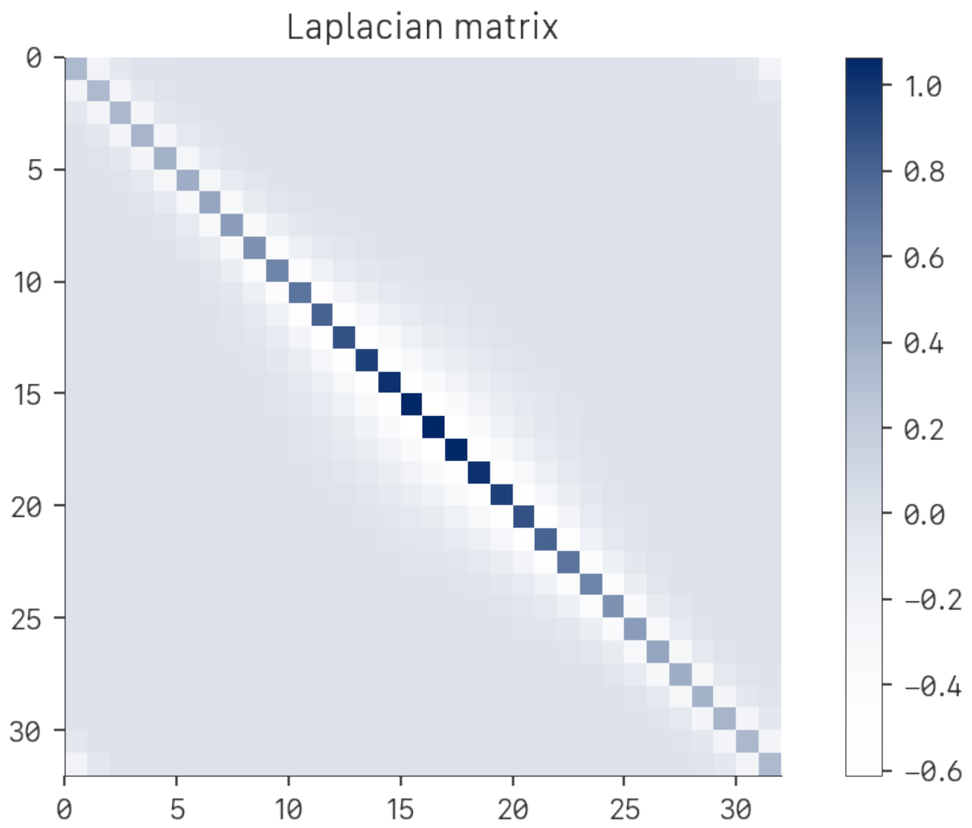




New time structure gets encoded in graph

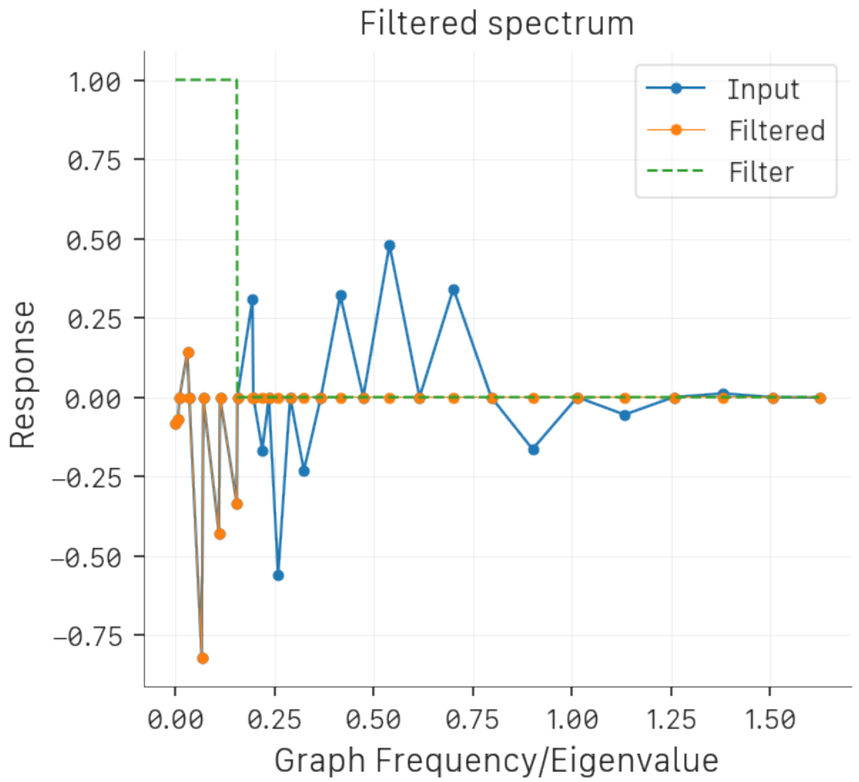
- Nodes closer in time have a stronger edge weight

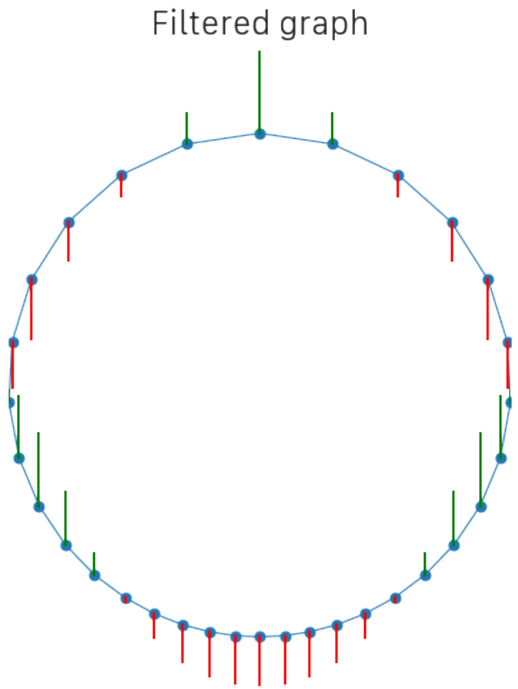


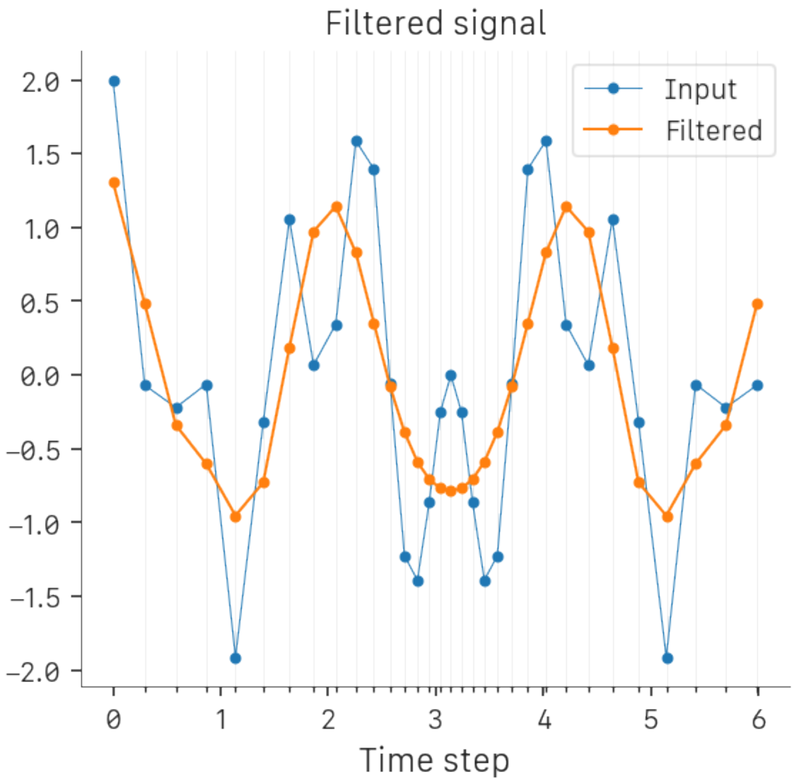


We can still apply a lowpass filter

- Using the same process as before
  - But now we can deal with the strange sample rate variation

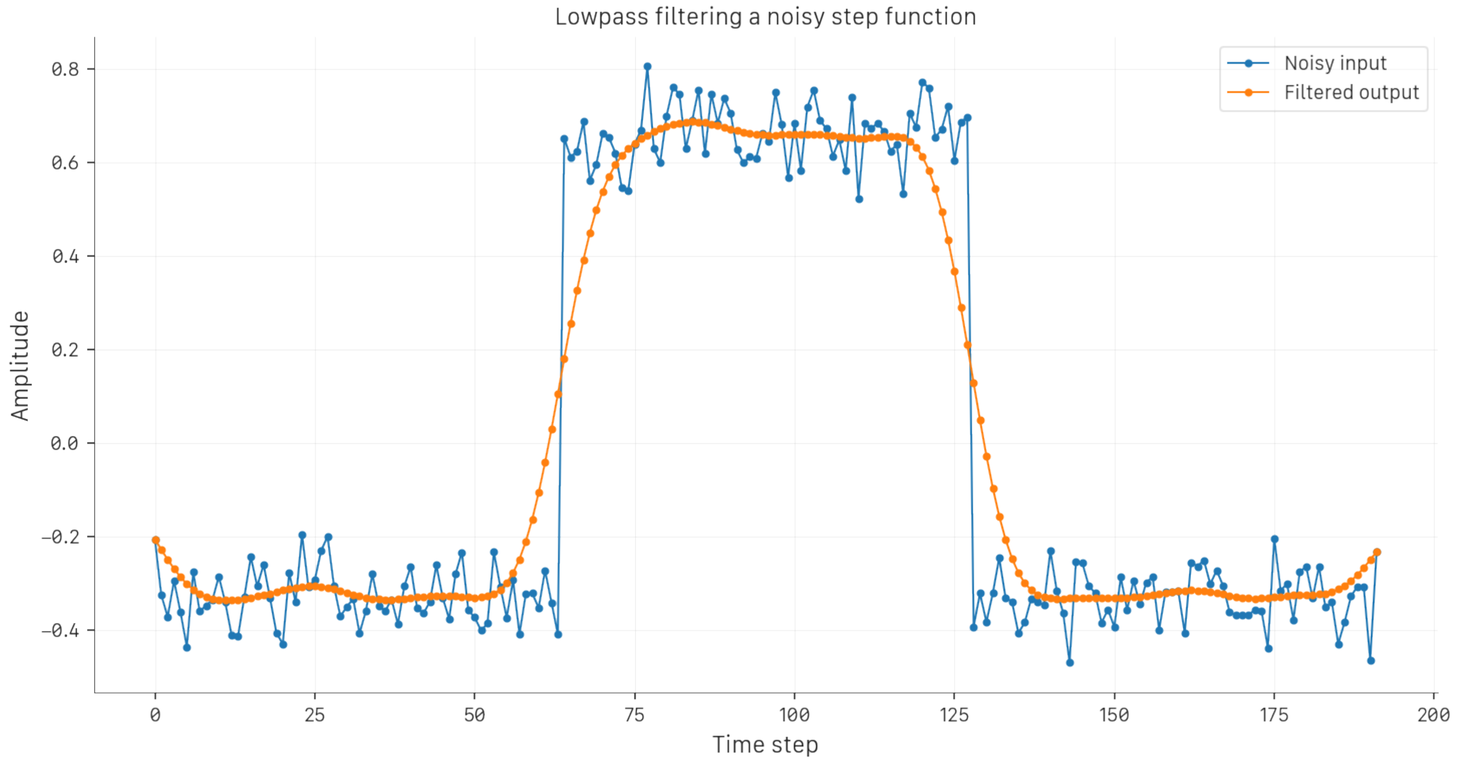






Encoding context from a signal

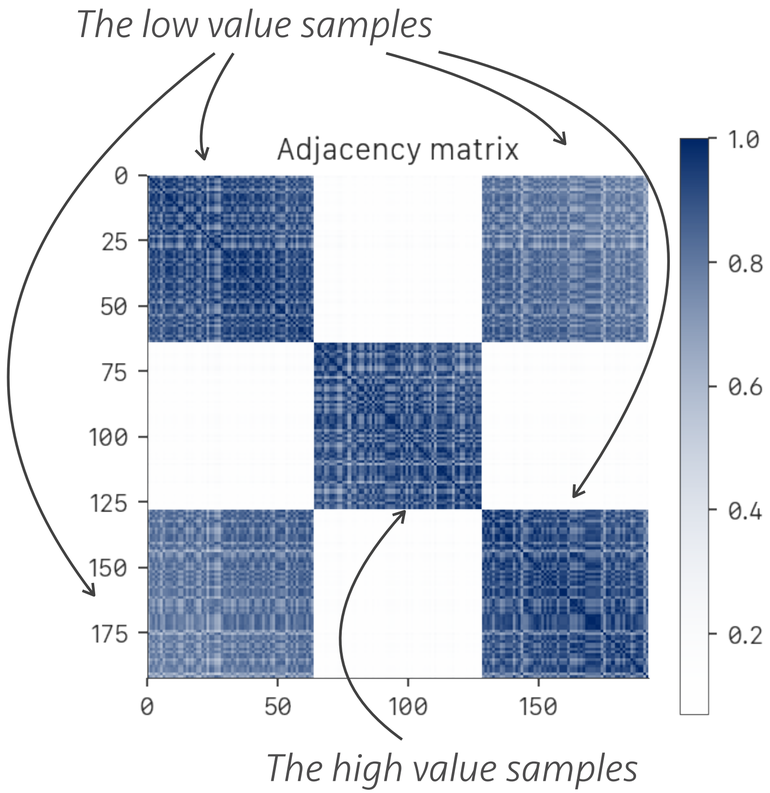
- Consider a noisy input with steep edges
  - The jumps are treated as high-frequency components
  - Filtering smooths out the jumps, altering the original signal

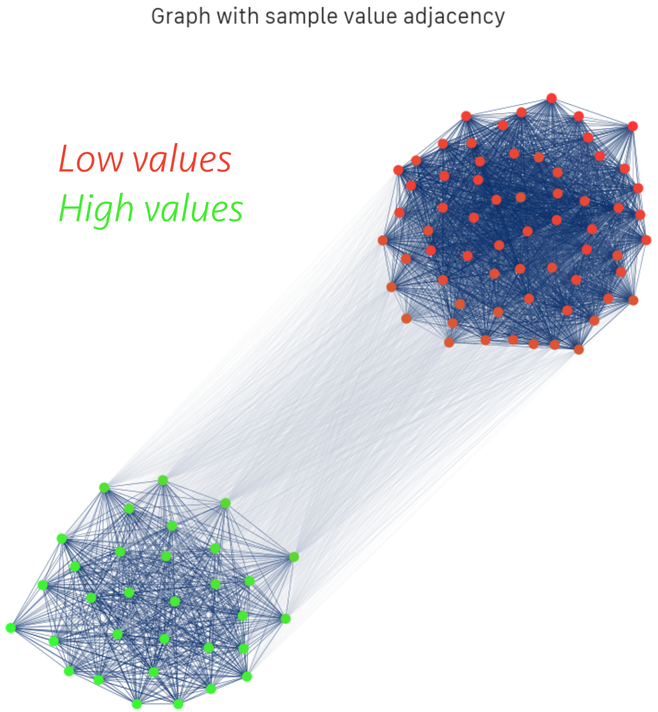


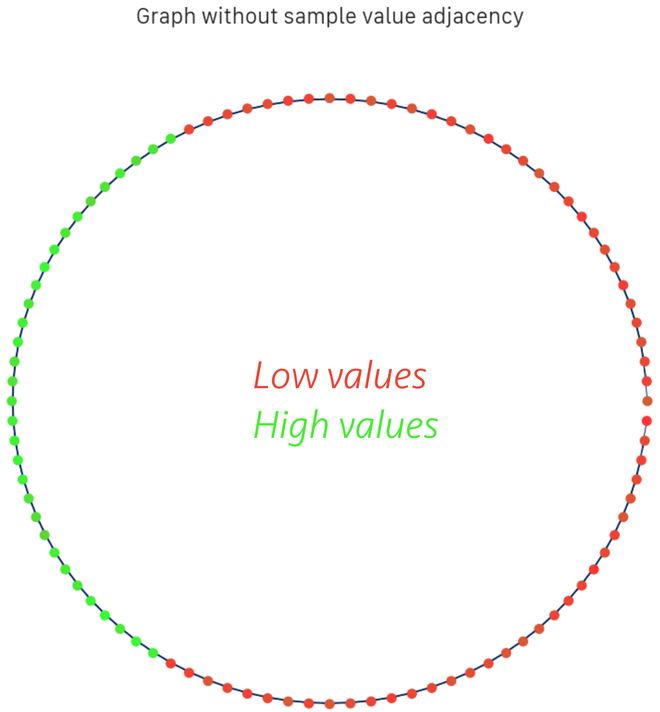
Using the signal in the adjacency

- We wish to convey that similar values are "closer"
  - Make an adjacency matrix that also uses the signal values

$$w_{i,j} = e^{-a|i-j|-\beta|x_i-x_j|}$$

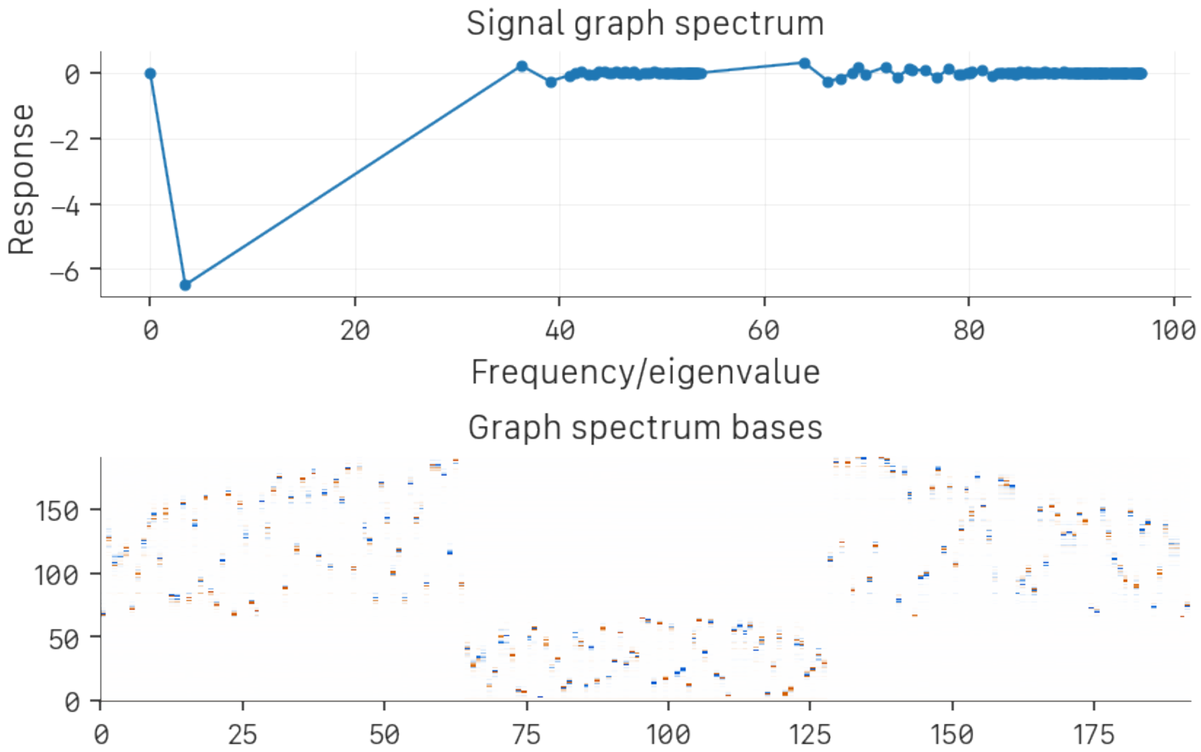


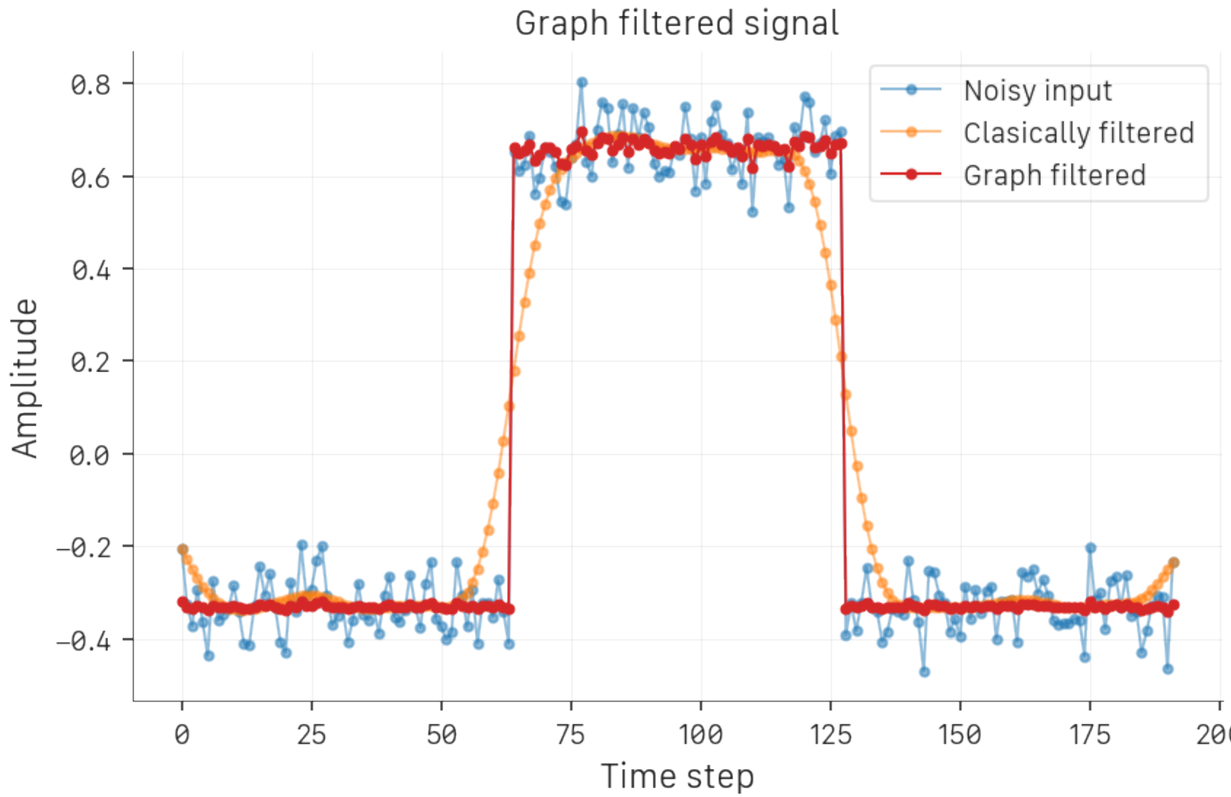




Edge-preserving filter

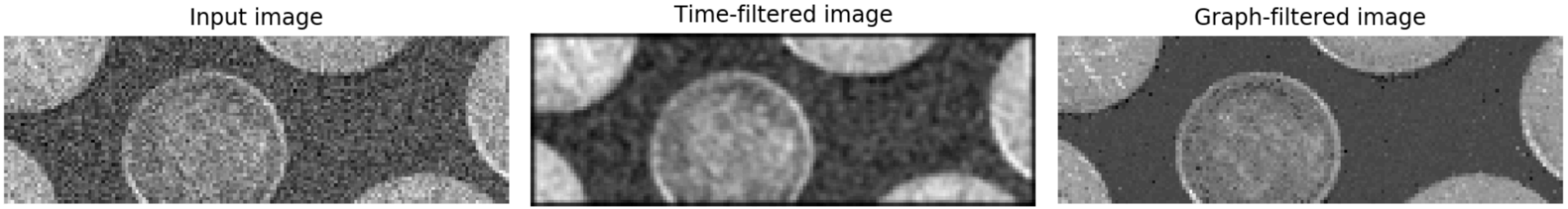
- Resulting filter doesn't smooth the large step
  - Effectively preserves edges in the signal





Edge preserving filters: 2D Example

- Consider a noisy image
  - Sample value is given by pixel intensity
  - Spatial location is given by pixel co-ordinates
- Vectorize the samples and spatial locations
  - Everything else stays the same as before

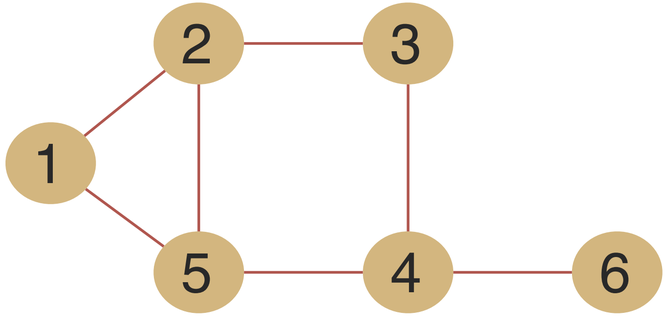


Defining graphs from data

- K-Nearest Neighbor graphs
  - Alternative way to impose a graph on data
  - Every vertex is connected to only *K* of its closest points
- Fourier Transforms on KNN graphs
  - Eigendecomposition on KNN adjacency matrix
  - We did that for manifold learning!

Filtering in the spatial domain

- Classical DSP: filtering in time vs frequency
  - Frequency: Multiplication ↔ Time: Convolution
  - Can we have such interpretations for graphs?

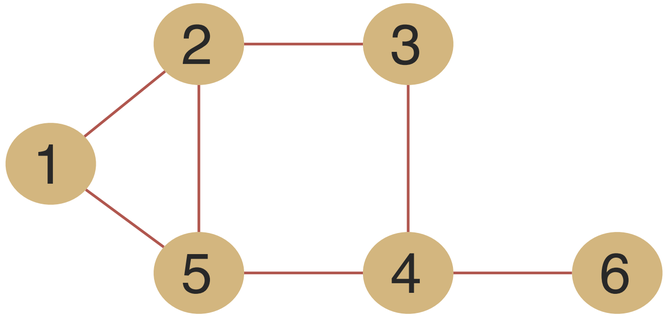


- Filter outputs are simply weighted sums of time-shifted inputs
  - We use a time shift operator to define filtering
    - Shifts a sequence by one time step
- The *graph-shift matrix*
  - Non-zero for connected edges, e.g. the Laplacian
    - Can (confusingly) otherwise be arbitrary!

$$S = \begin{bmatrix} S_{11} & S_{12} & 0 & 0 & S_{15} & 0 \\ S_{21} & S_{22} & S_{23} & 0 & S_{25} & 0 \\ 0 & S_{32} & S_{33} & S_{34} & 0 & 0 \\ 0 & 0 & S_{43} & S_{44} & S_{45} & S_{46} \\ S_{51} & S_{52} & 0 & S_{54} & S_{55} & 0 \\ 0 & 0 & 0 & S_{64} & 0 & S_{66} \end{bmatrix}$$

Filtering in the spatial domain

- The graph shift can be computed locally
  - $\mathbf{S}\cdot\mathbf{x}$ can be computed from current and 1-hop nodes
  - $\mathbf{S}^2\cdot\mathbf{x}$ uses up to 2-hop nodes
  - Used to define filters



- Example:

$$\mathbf{x} = \left[-1, 2, 0, 0, 0, 0\right]^\top$$

$$\mathbf{h} = \left[1, 1, 0.5\right]^\top$$

$$\mathbf{S} = \mathbf{I} + \mathbf{W} \quad \leftarrow \textit{Adjacency matrix}$$

$$\mathbf{y} = h_0\mathbf{x} + h_1\mathbf{S}\cdot\mathbf{x} + h_2\mathbf{S}^2\cdot\mathbf{x}$$

$$\mathbf{S}\cdot\mathbf{x} = \begin{bmatrix}1\\1\\2\\0\\1\\0\end{bmatrix}, \quad \mathbf{S}^2\cdot\mathbf{x} = \begin{bmatrix}3\\5\\3\\3\\3\\0\end{bmatrix}, \quad \mathbf{y} = h_1\mathbf{x} + h_2\mathbf{S}\cdot\mathbf{x} + h_3\mathbf{S}^2\cdot\mathbf{x} = \begin{bmatrix}1\\5\\2.5\\1.5\\2\\0\end{bmatrix}$$

Filtering

To summarize

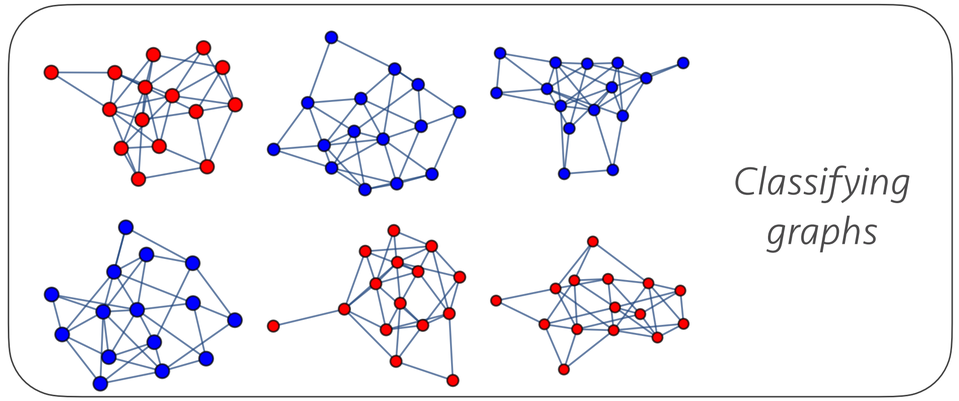
- Signals can be defined as graphs
  - Allows us to represent richer sample relationships
- We can define direct analogs to classical DSP
  - Time/frequency domain
  - Filtering operations
    - Not mentioned today: sampling, wavelets, etc …
- So now we can use these to do machine learning!

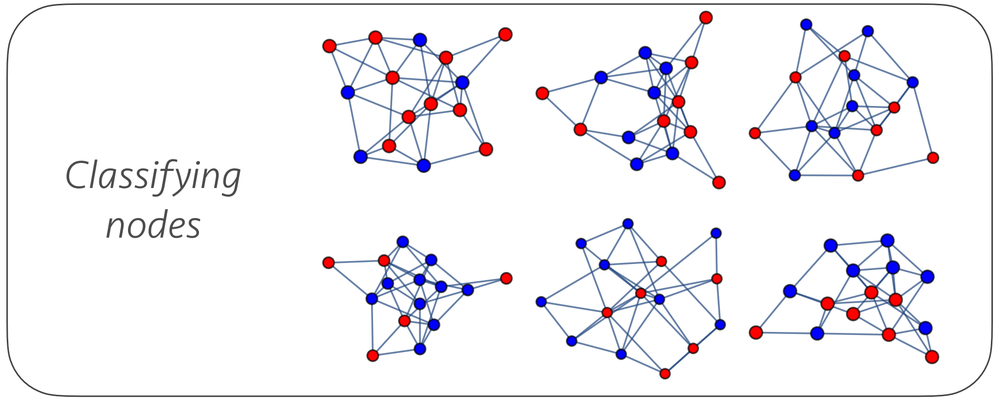
Graph Neural Networks

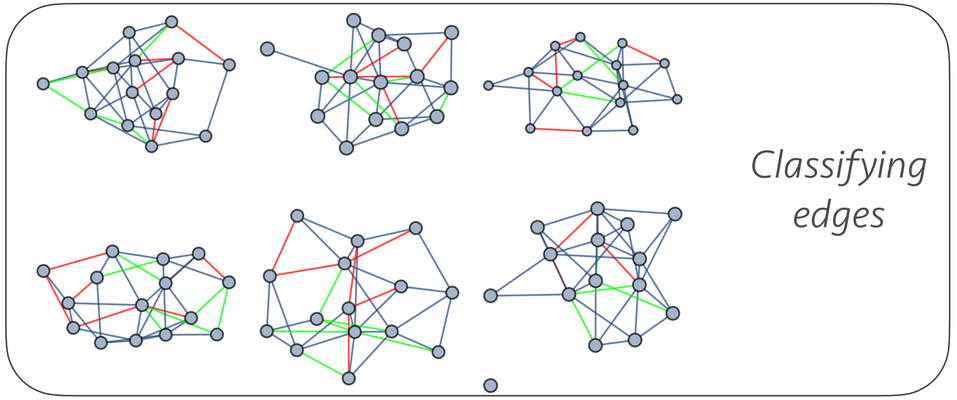
- Neural nets that operate on graphs
  - Most often rely on graph convolutions (CNN equivalents)
  - Using the graph-shift matrix we can also define graph RNNs
- Useful for non-conventionally sampled data
  - Molecules, social graphs, Mesh representations, LIDAR data, …
- One of the hot topics recently

Types of graph processing

- Graph-level processing
  - Labeling an entire graph
    - E.g. classify a molecule
- Node-level processing
  - Labelling each node
    - E.g. LIDAR pixel classification
- Edge-level processing
  - Labelling/predicting each edge
    - E.g. action prediction

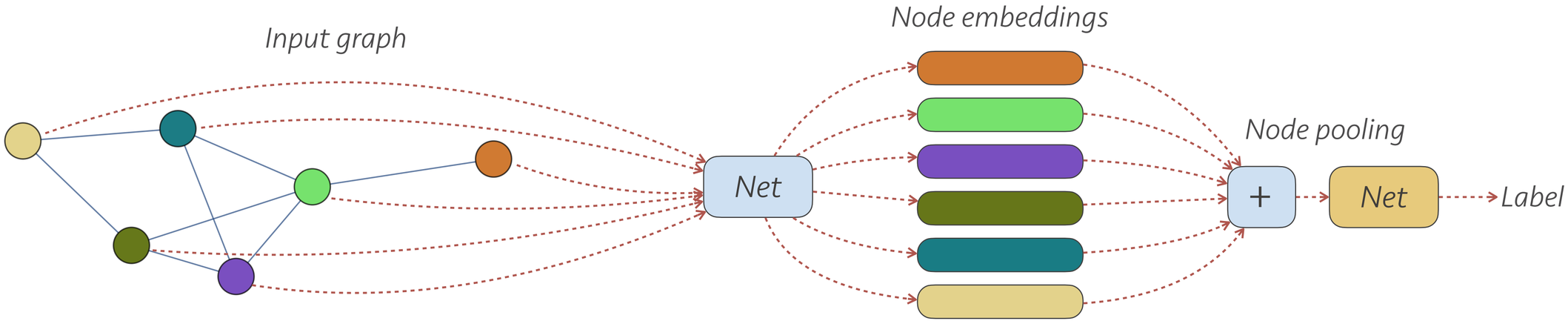






Towards a graph network

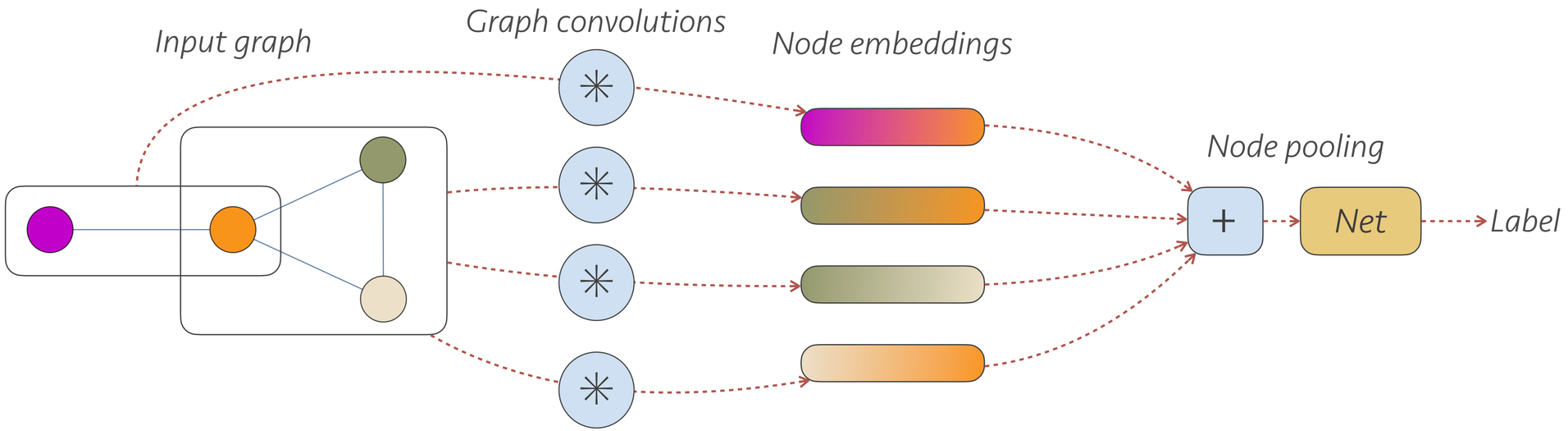
- Embedding the nodes only
  - Apply the same network on each node to obtain an embedding



- Depending on task
  - Label nodes ⟶ embedding is the label
  - Label graph ⟶ pool information by combining node embeddings
    - Usually take their sum and feed into a regular network

Adding the graph structure

- Use graph convolutions to combine neighboring nodes when computing node embeddings
  - Explicitly includes local structure of input graph
  - We can also add a "master node" to get global structure

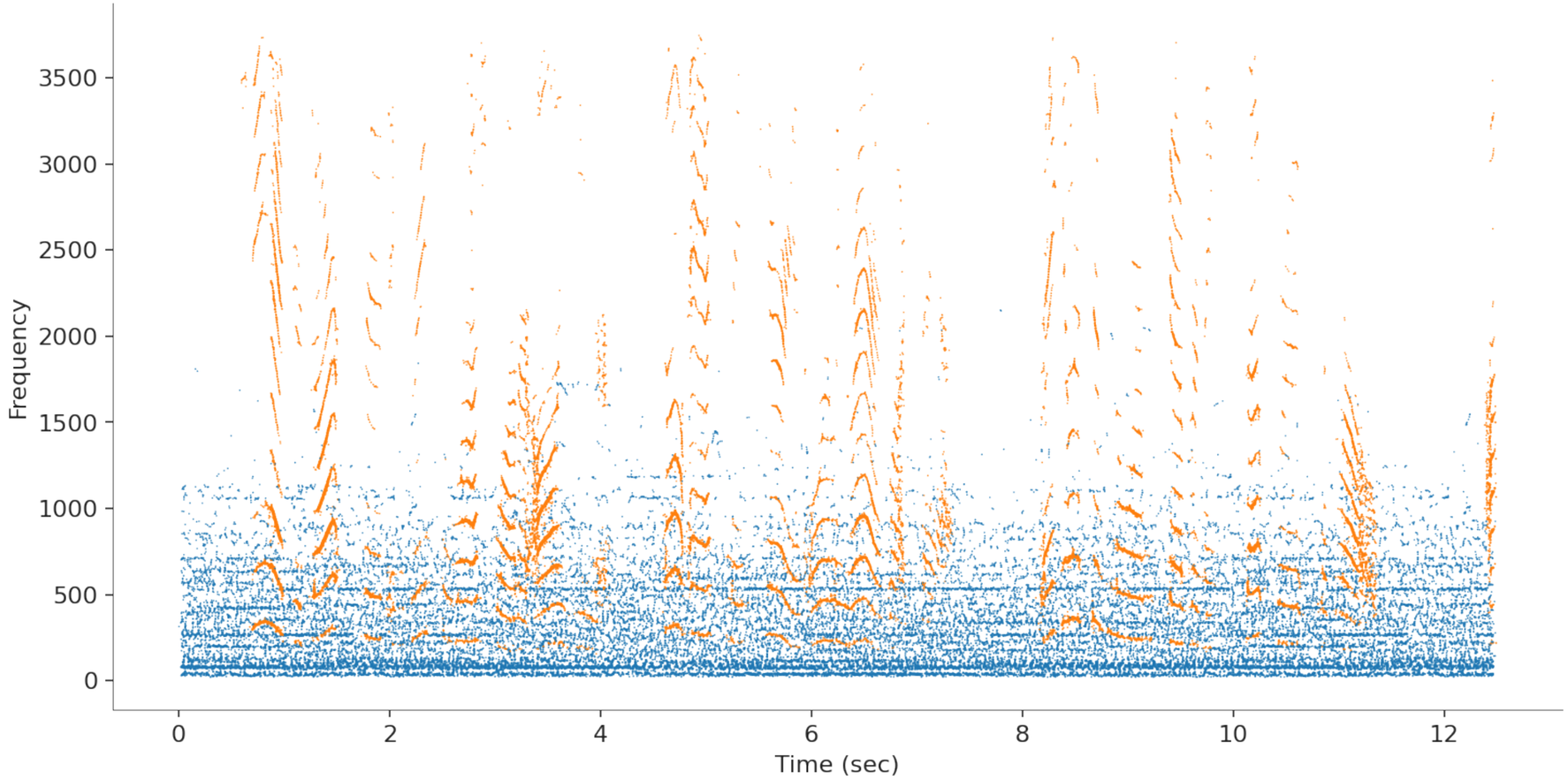


Lots of variations here

- Different sorts of Graph CNN types
  - Mostly least-publishable unit variations ☹
- Attention/transformer models
  - Instead of convolving, attend to relevant nodes
- Autoregressive models
  - GraphRNN, recursively passes graph messages

Example: Point cloud audio processing

- Using point values to describe spectral content
  - New representation that requires novel processing



The advantage of point cloud data

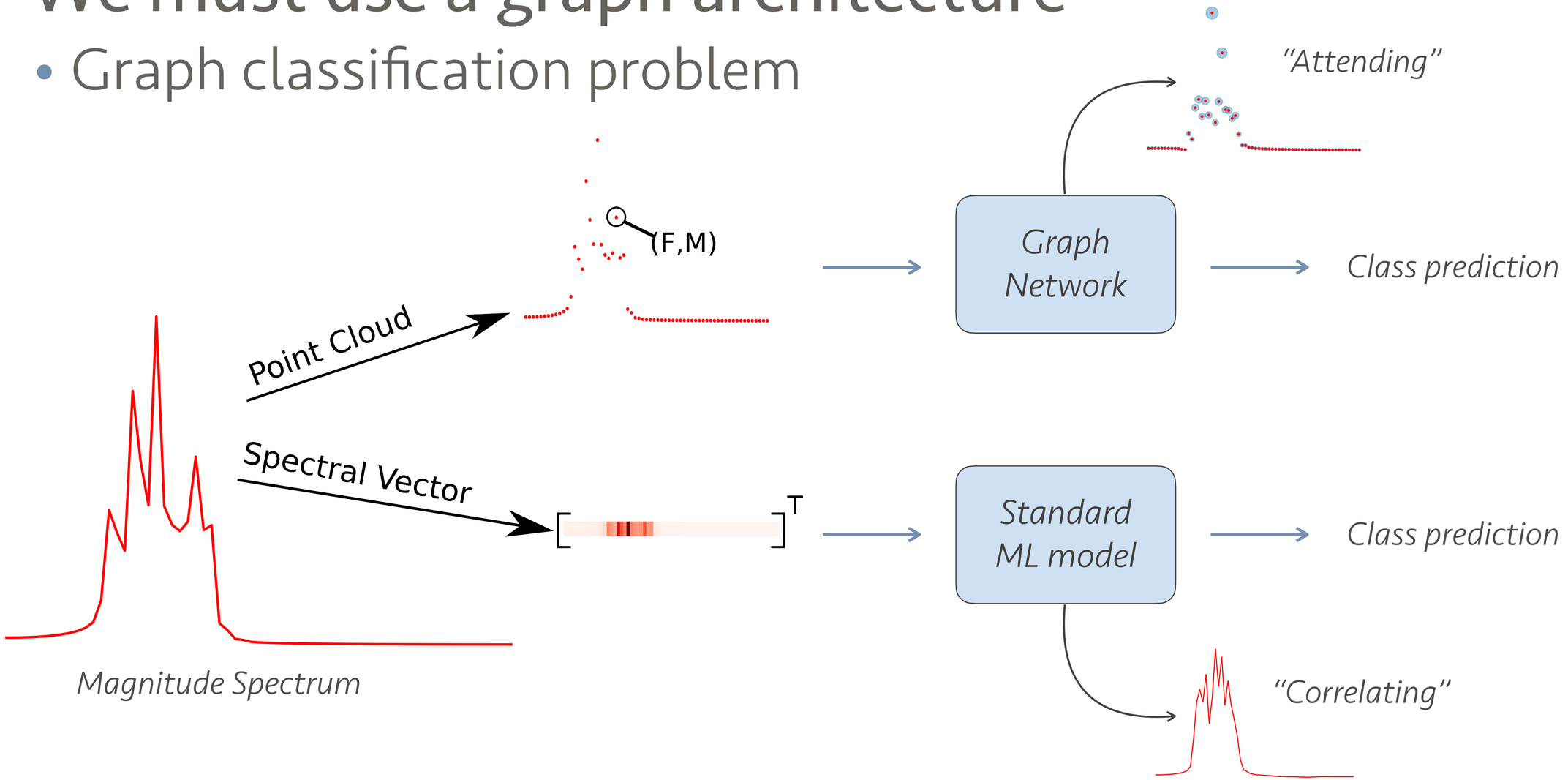
- With point clouds we only use as much data as we can
  - We can pick and choose depending on the hardware
  - Or we can sample as needed

- A different representation:
  - Instead of e.g. a spectral vector: $\mathbf{f} = [f_{\omega 1}, f_{\omega 2}, f_{\omega 3}, \ldots, f_{\omega N}]$
  - Use nodes: $\mathbf{t} = [(\omega_1, v), (\omega_4, v), (\omega_{10.2}, v), \ldots]$
    - Tuples $\mathbf{t}$ can be a subset of $\mathbf{f}$, or even resampled values

- Frees us from fixed sizes and sample rates (ω is in Hz)

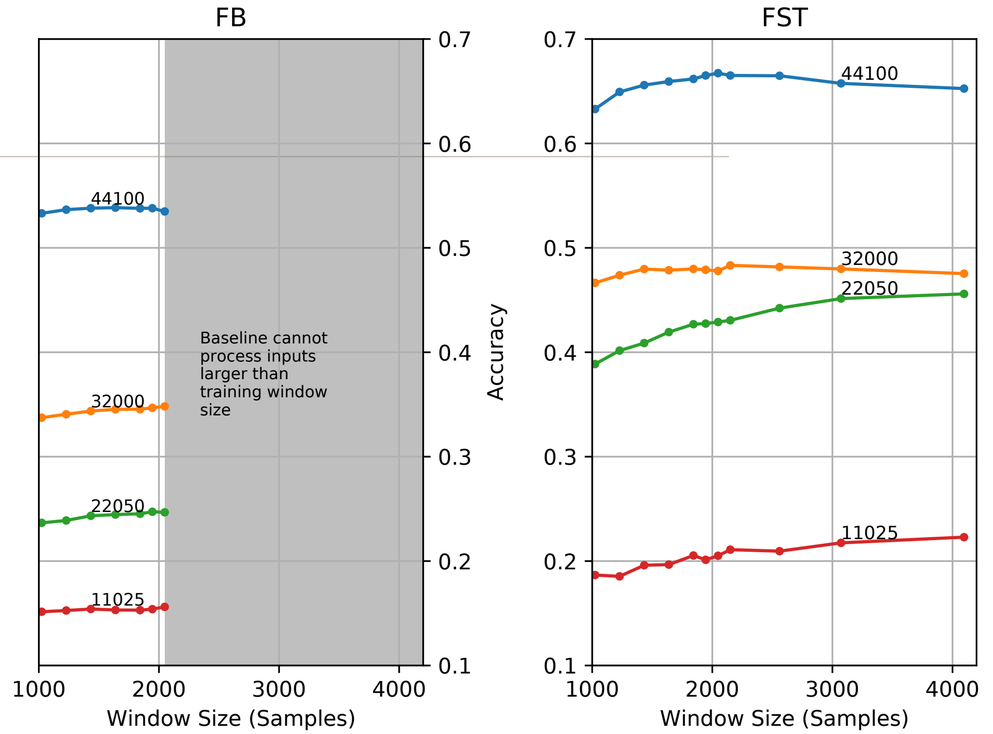
Processing point data

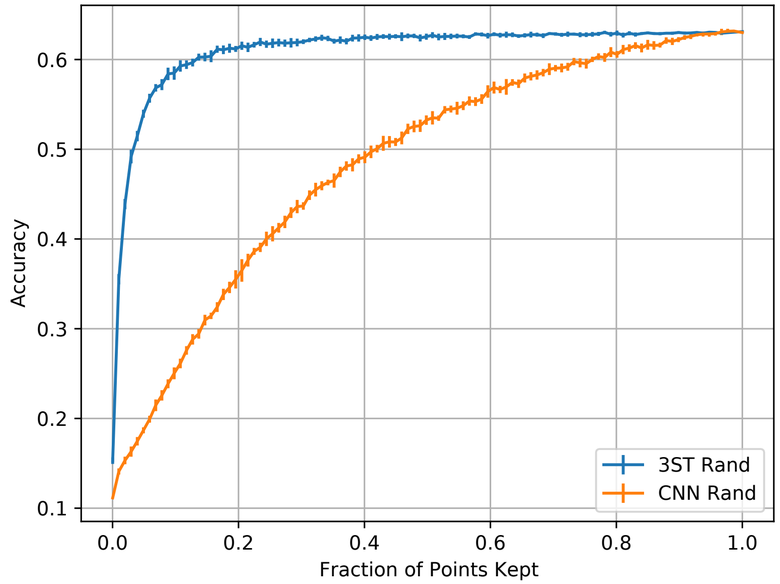
- We must use a graph architecture
  - Graph classification problem



Example application

- Sound recognition task (ESC-10)
  - Classifying from spectro-temporal data
- Win-win-win situation
  - Better performance overall
  - Largely invariant to sampling rate
  - Smaller model: 80K vs 160K parameters
  - Works well with only 20% of input!
  - Faster training
    - Using only 1/16th of training data when using subsampling of input representation





Recap

- Interpreting signals as a graph has benefits
  - Graphs can capture underlying structure
- Signal Processing on Graph signals
  - Generalizing classical DSP approaches to graph signals
    - The Graph Fourier Transform
  - Applications: Filtering and Denoising
    - E.g., Edge-preserving filters
- Graph networks
  - Generalize passage of information through graphs
    - Several novel applications now possible

Reading material

- Graph Signal Processing
  - https://arxiv.org/pdf/1712.00468.pdf
- Graph convolutions and graph nets:
  - https://distill.pub/2021/understanding-gnns/
  - https://distill.pub/2021/gnn-intro/
- Point cloud audio processing
  - https://arxiv.org/pdf/2105.02469.pdf

Upcoming schedule

- On travel Nov 27-30, next lecture will be our last
  - Secure Multiparty processing
  - Rest of generative models
  - We might have some extra time left
    - Send me ideas on subjects to discuss briefly
- Projects
  - We will hold a poster session on Dec 5th noon-2pm
    - This won't be your final report, fine if you have no results yet
      - Tell us what you are doing and how you plan to do it
  - We have 29 groups → ~4min/group!