Efficient Gaussian Process on Graphs

Matthew Zhang (mz473@cam.ac.uk)
Supervised by Professor Rich Turner & Isaac Reid

Division F, Department of Engineering
University of Cambridge

Outline

1. Background: Gaussian Processes (GP) on Graphs.
2. Previous Work and Motivation.
3. Key Algorithm: General Graph Random Features (GRF).
4. Current Progress.
5. Next Steps.

Extending Gaussian Processes (GP) to Graphs

- **GPs** are powerful in Euclidean spaces.
- Real-world data (e.g., traffic networks) are often better represented with **Graphs**.

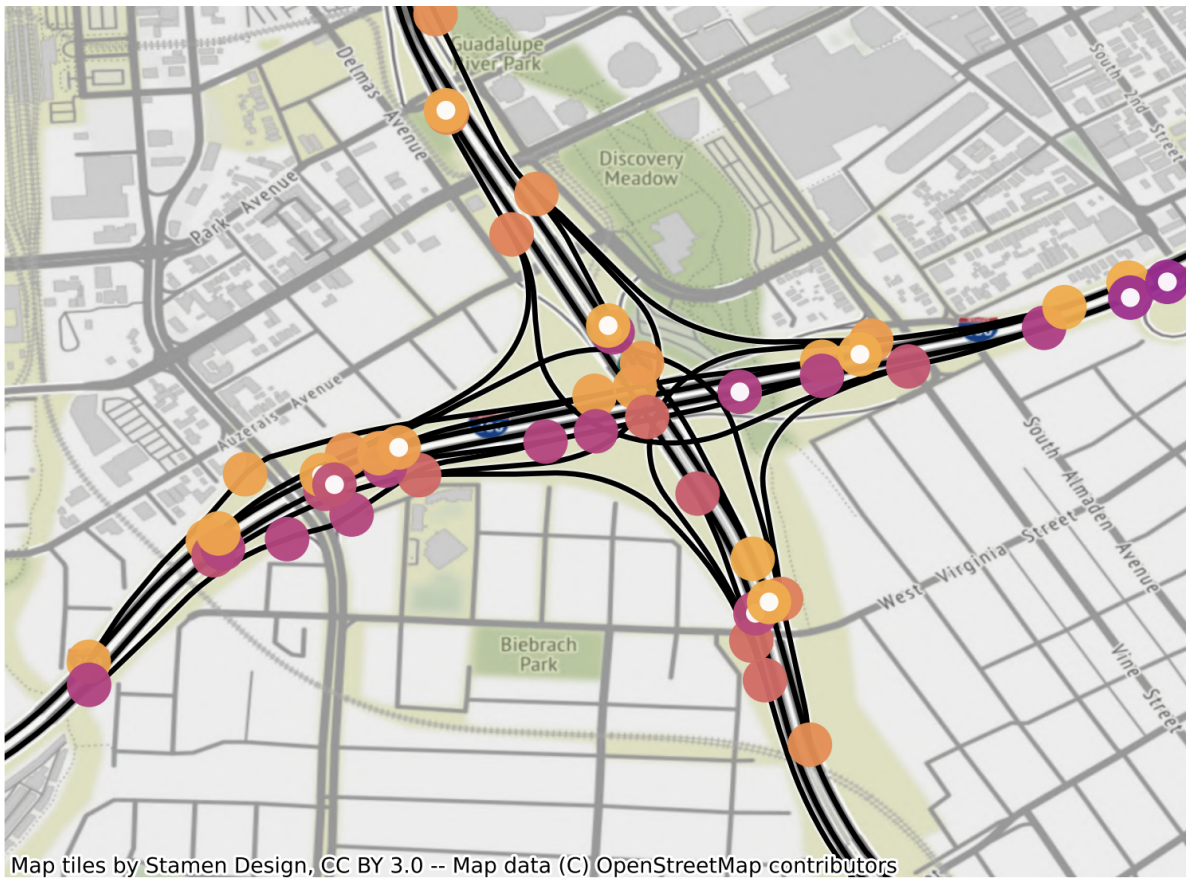


Figure: Traffic network as a graph.

How to Represent a Graph

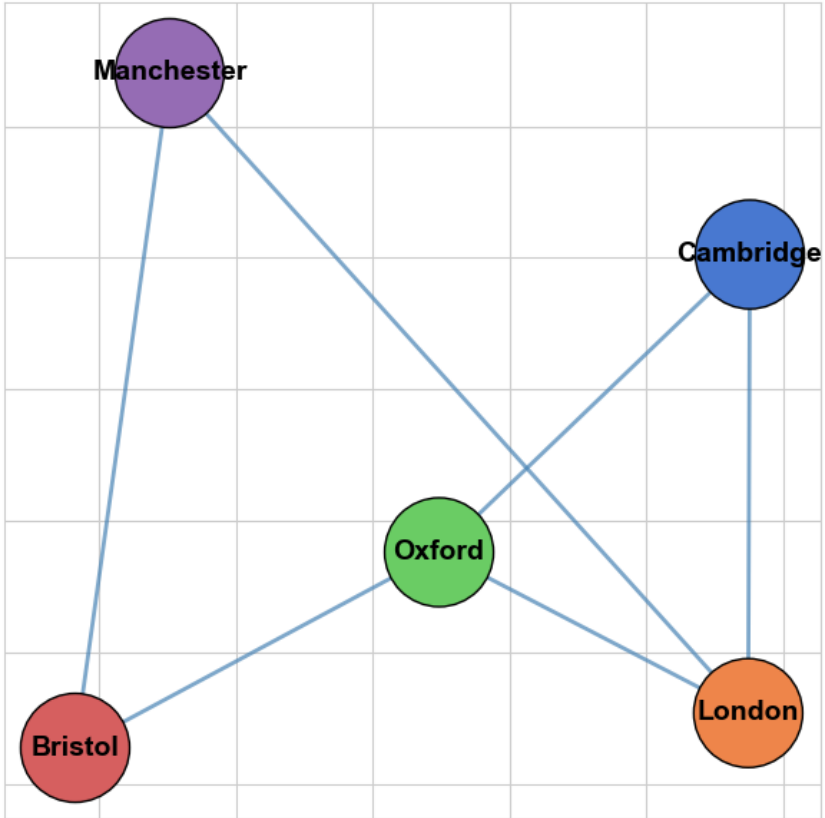


Figure: Example graph.

Adjacency matrix:

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

$$W_{i,j} = \begin{cases} 1, & \text{if connected} \\ 0, & \text{otherwise} \end{cases}$$

Generalizing Kernel Functions to Graphs

Graph Kernels:

- GP needs the covariance matrix, measuring "distance" between nodes.
- Using function of adjacency matrix: e.g., $K(W) = W^{-1}$

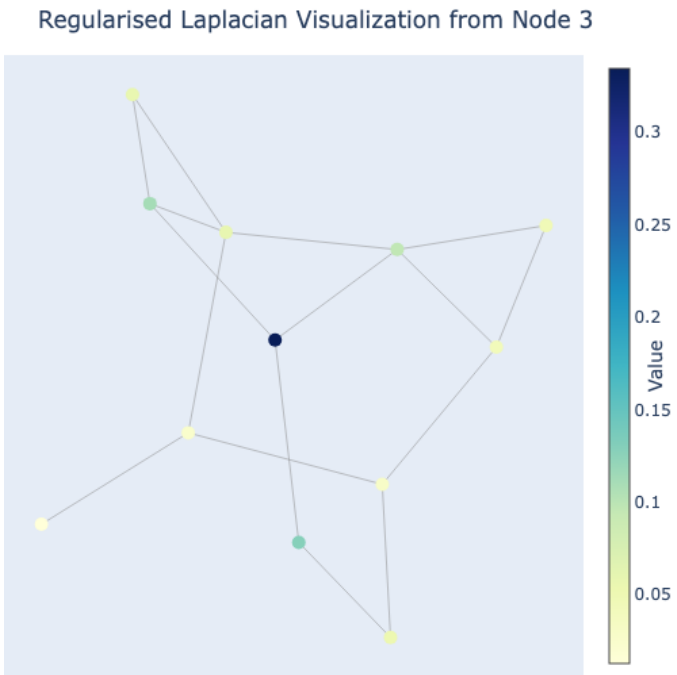


Figure: Kernel using Regularized Laplacian.

Previous Work: GP on Graphs

- V. Borovitskiy et al. (2020): GPs can do inference really well on graph datasets!

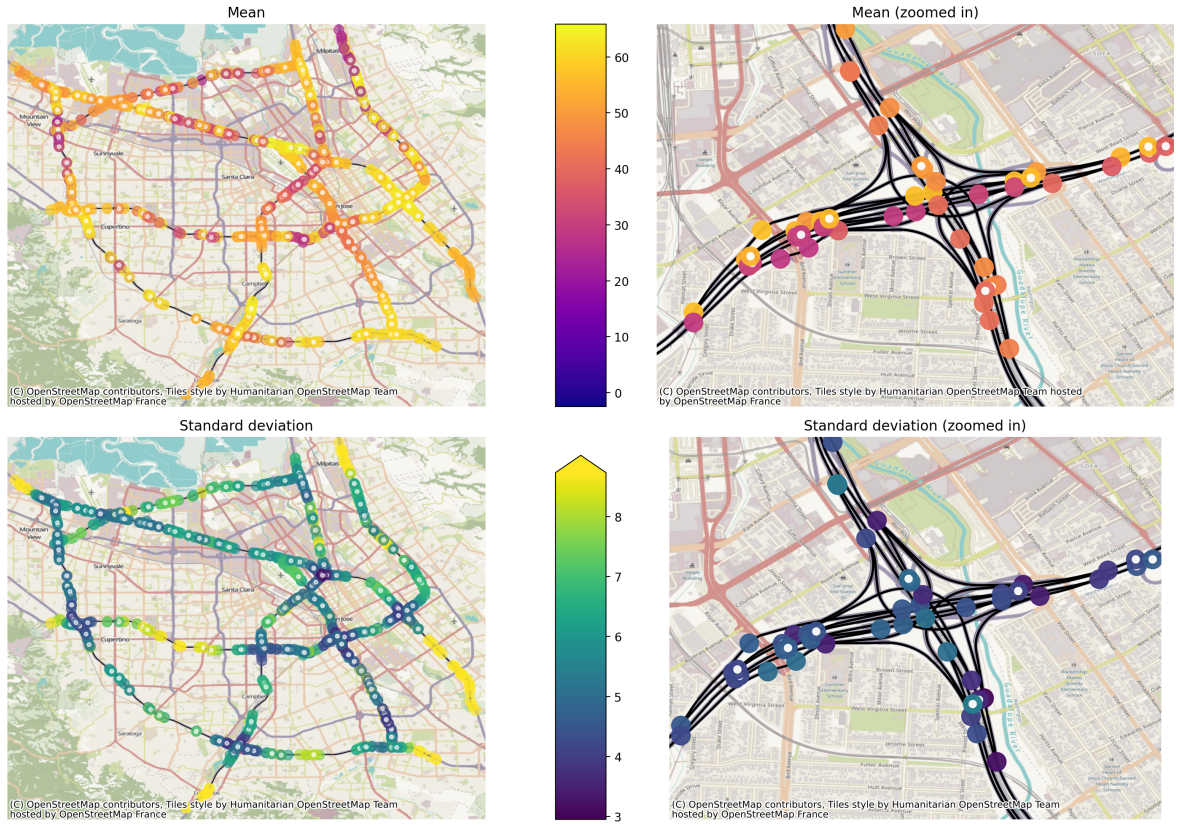


Figure: Gaussian Processes applied to a traffic network dataset.

Challenge: Evaluating Kernel is Expensive

Challenge: Computing the exact graph kernel involves inverting large matrices, resulting in high complexity $\sim O(n^3)$.

Project Objective: Can we find a way to apply GP on graph efficiently?

The Key Algorithm: Graph Random Features (GRF)[1]

- **Key Idea:** Approximate graph kernels by leveraging **random walks** on the graph.
- **Advantages:** Significantly reduces kernel computation complexity from $O(n^3)$ to sub-quadratic time.
- **Tools:** random walks, random features.

[1] Isaac et al., 2023

GRF Explained: Random Walk

What is a Random Walk?

1. Start at a randomly selected node.
2. Select a random neighbor of the current node.
3. Move to the selected neighbor and repeat.

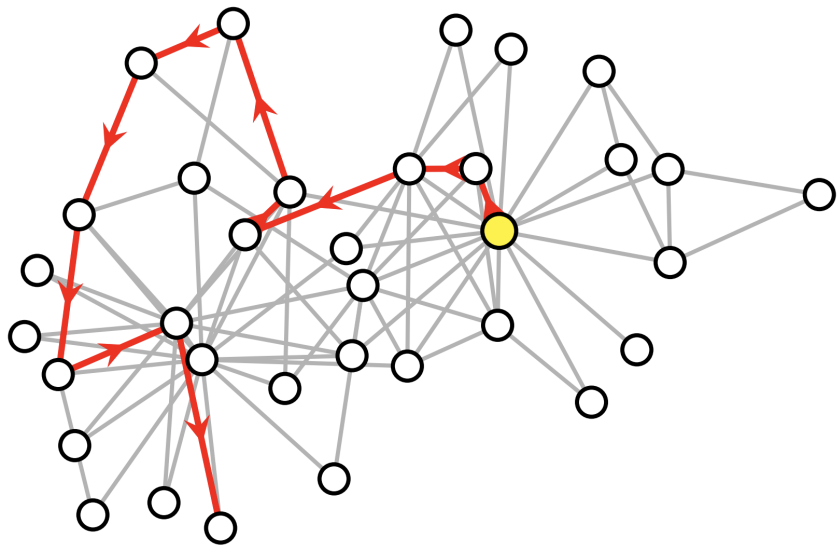


Figure: Illustration of random walks on a graph.

GRF Explained: Random Features

Random Features:

- Assign a signature vector ϕ_i for each node $i = 1, 2, \ldots, N$, such that:

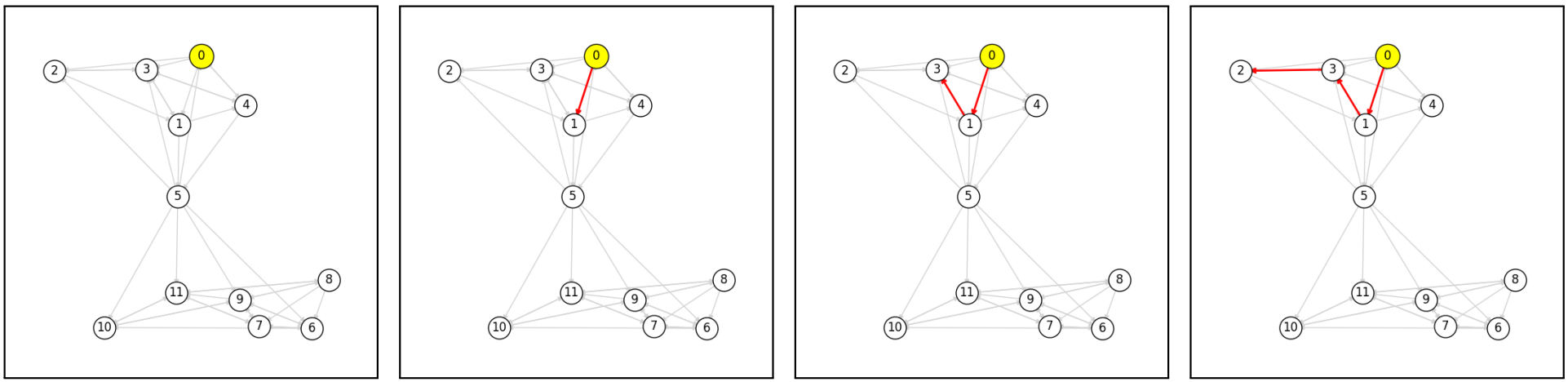
$$\phi_i^T \phi_j \approx \text{dis}(\text{node}_i, \text{node}_j) = K(W)_{i,j}.$$

Naiive Algorithm:

1. Select a starting node i in the graph.
2. Initialize $\phi_i = [0, 0, \ldots, 0]$.
3. Perform a random walk of m steps.
4. For each visited node j, increment the j-th entry of ϕ_i:

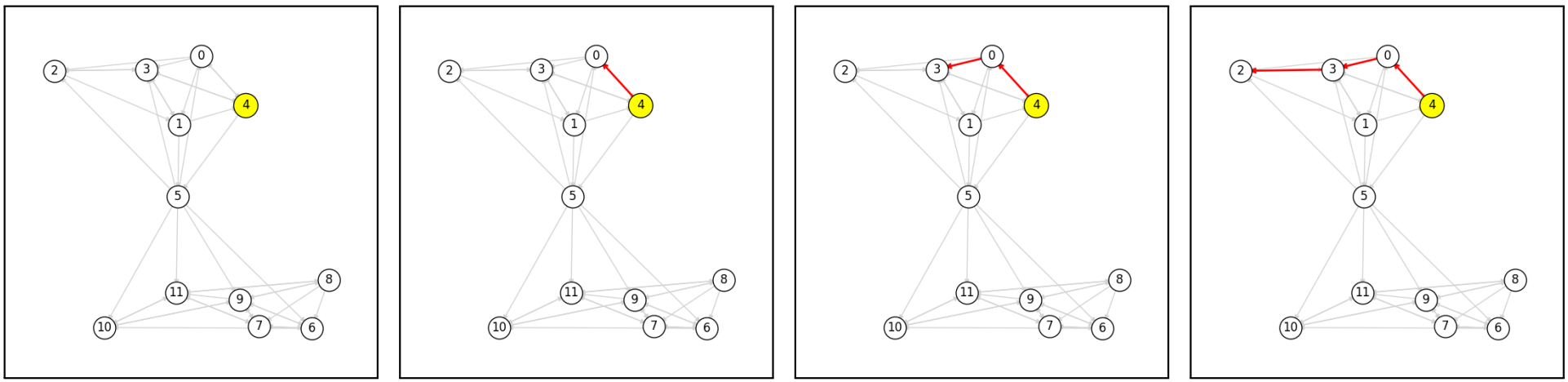
$$\phi_i[j] \leftarrow \phi_i[j] + 1.$$

GRF Explained: Calculate Random Features (1)



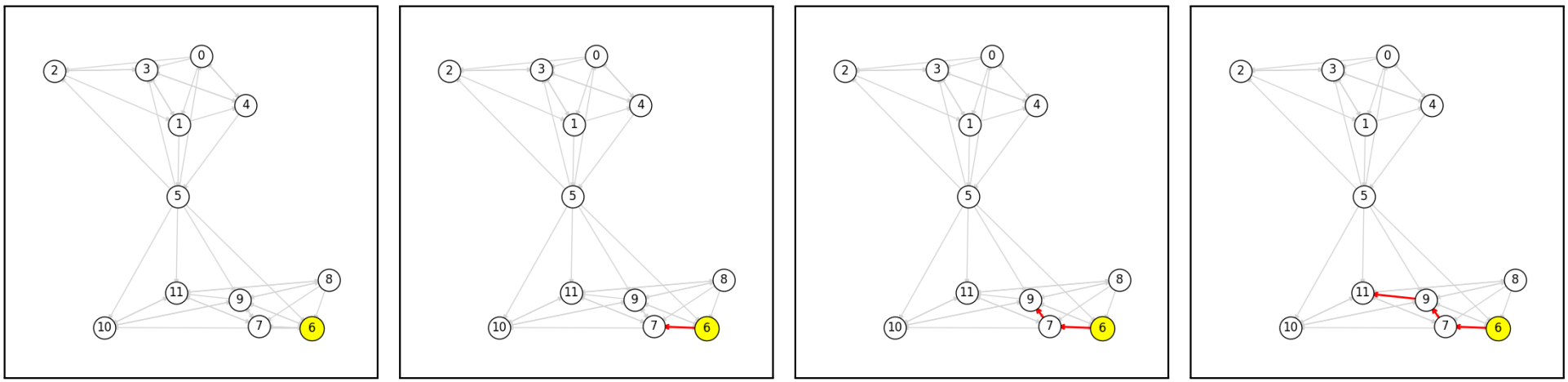
$$\Rightarrow \quad \phi_0^T = [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]$$

GRF Explained: Calculate Random Features (2)



$$\Rightarrow \quad \phi_4^T = [1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]$$

GRF Explained: Calculate Random Features (3)



$$\Rightarrow \quad \phi_6^T = [0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0]$$

GRF Explained: Properties of Random Features

Observations:

- $\phi_0^T \phi_4 = 3$ (closer nodes have larger dot products)
- $\phi_0^T \phi_6 = 0$ (distant nodes have smaller dot products)
- This is an okay approximation to the Graph Kernel.

In Practice:

- Weighting random walks with 'modulation functions'.
- Unbiased estimator:

$$\mathbb{E}[\phi_i \cdot \phi_j] = K(W)_{i,j}.$$

GRF Explained: The Actual Algorithm

Algorithm 1 Constructing a random feature vector $\phi_f(i) \in \mathbb{R}^N$ to approximate $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})$

Input: weighted adjacency matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$, vector of unweighted node degrees (no. neighbours) $\boldsymbol{d} \in \mathbb{R}^N$, modulation function $f : (\mathbb{N} \cup \{0\}) \to \mathbb{R}$, termination probability $p_{\text{halt}} \in (0, 1)$, node $i \in \mathcal{N}$, number of random walks to sample $m \in \mathbb{N}$.
Output: random feature vector $\phi_f(i) \in \mathbb{R}^N$

initialise: $\phi_f(i) \leftarrow \mathbf{0}$
for $w = 1, ..., m$ **do**
    initialise: $\texttt{load} \leftarrow 1$
    initialise: $\texttt{current\_node} \leftarrow i$
    initialise: $\texttt{terminated} \leftarrow \text{False}$
    initialise: $\texttt{walk\_length} \leftarrow 0$
    while $\texttt{terminated} = \text{False}$ **do**
        $\phi_f(i)[\texttt{current\_node}] \leftarrow \phi_f(i)[\texttt{current\_node}] + \texttt{load} \times f(\texttt{walk\_length})$
        $\texttt{walk\_length} \leftarrow \texttt{walk\_length} + 1$
        $\texttt{new\_node} \leftarrow \text{Unif}[\mathcal{N}(\texttt{current\_node})]$ ▷ assign to one of neighbours
        $\texttt{load} \leftarrow \texttt{load} \times \frac{d[\texttt{current\_node}]}{1 - p_{\text{halt}}} \times \mathbf{W}[\texttt{current\_node}, \texttt{new\_node}]$ ▷ update load
        $\texttt{current\_node} \leftarrow \texttt{new\_node}$
        $\texttt{terminated} \leftarrow (t \sim \text{Unif}(0, 1) < p_{\text{halt}})$ ▷ draw RV t to decide on termination
    end while
end for
normalise: $\phi_f(i) \leftarrow \phi_f(i)/m$

Figure: The actual GRF algorithm gives an extra weight to each step in the random walk.

The Big Picture & Current Progress

The project consists of three key components:

1. **GRF Algorithm:** ✓
2. **Graph GP Inference System:** ✓
3. **Integration:** □

Progress: GRF Experiments

Experiments: GRF was applied to approximate various graph kernels. **Results:** Quick Convergence to Ground Truth Kernel

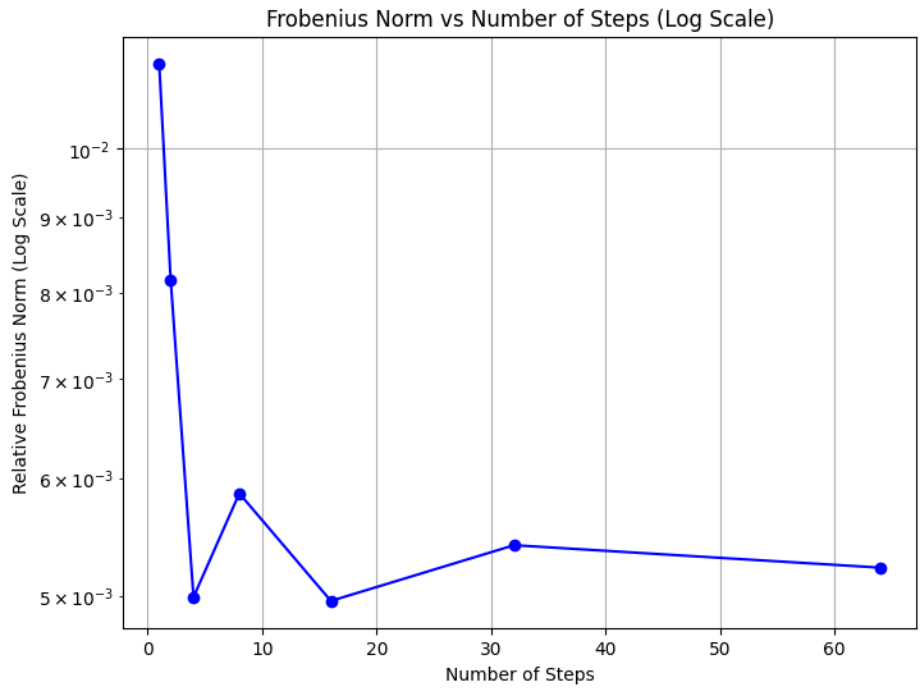


Figure: Convergence of Diffusion (Heat) Kernels Using GRF.

Progress: GP Inference on Graphs

Example:

- Sampled data on a graph covariance matrix, with hyperparameter $\beta = 2.7$.
- Fitted a GP, learned β: 2.66.

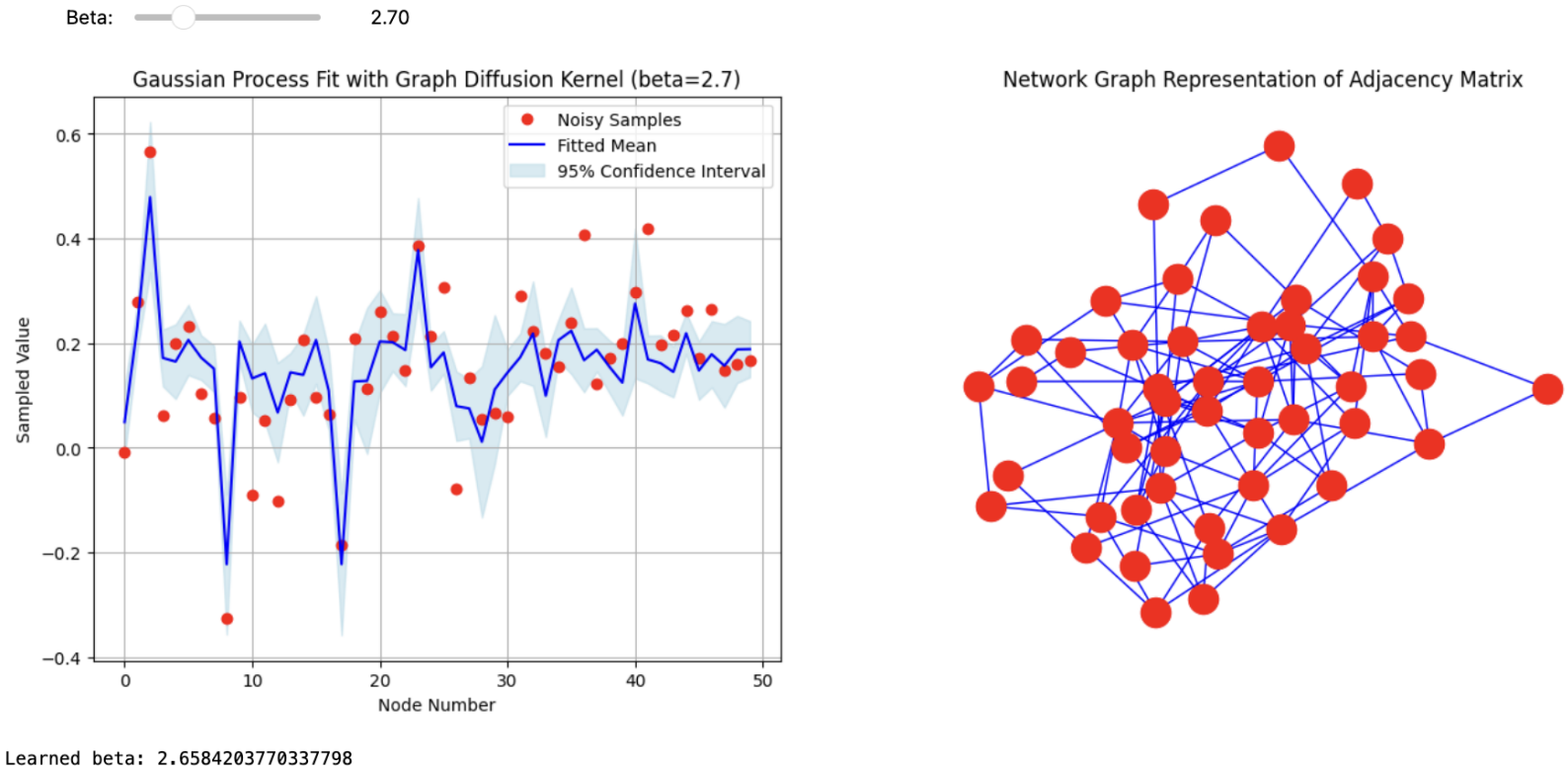


Figure: Left: Fitted GP on the graph; Right: Random graph.

Next Steps

- **Integrate GRF with Graph GP Inference**
- **Test on Real-World Data**
- **Quantify Efficiency Improvements & Accuracy Trade-off**