Graph Sampling with Determiantal Point Processes

Quentin CHAN-WAI-NAM
MVA

Guillaume GAUTIER (tutor)
INRIA

Abstract

A great variety of data is originally structured as networks, or graphs. Data defined on the nodes of a given graph is called a graph signal. Here, we are interested in some results of the sampling theory, that state it is possible under some assumptions to reconstruct a graph signal given its values on only a node sample of size much smaller than the entire graph. We focus here on graph sampling using determinantal point processes, which are very useful tools to choose the nodes on which to sample the signal. This project is mainly focused on implementation and experimentation using results from [1].

1 Theory

1.1 Frequencies on graphs

The goal of the sampling theory on graphs [2] is similar to the goal of the Fourier sampling theory on classical signals: we try to decompose a given signal on some frequencies and use this decomposition to build a much compressed representation of the signal. Given a graph with N nodes, we define the Laplacian matrix as:

$$L = D - W$$

where $W \in \mathbb{R}^{N \times N}$ is the matrix of the edges of the graph such that $W_{i,j} \geq 0$ is the weight of the edge between nodes i and j, and $D \in \mathbb{R}^{N \times N}$ is diagonal with $D_i = \sum_j W_{i,j}$. Let (Λ, U) be the eigendecomposition of L such that $\Lambda = (\lambda_1, \cdots, \lambda_N)^\top$, $0 = \lambda_1 \leq \cdots \leq \lambda_N$ and $U = (u_1, \cdots, u_N)$ are the associated eigenvectors. We say that $U_k = (u_1, \cdots, u_k)$ are the k first low-frequency eigenmodes and any linear combination of $u_1, \cdots, u_k$ is called a k-bandlimited signal. Alternatively, x is a k-bandlimited signal iff

$$\exists\, \alpha \in \mathbb{R}^k, \qquad x = U_k \alpha. \tag{1}$$

1.2 Sampling on graphs and signal reconstruction

Sampling consists in choosing m nodes among N and measuring the signal on these nodes. If $\mathcal{A} = (\omega_1, \cdots, \omega_m)$ is the subset of the chosen nodes, $M \in R^{m \times N}$ is the measurement matrix such that $M_{i,j} = \delta_{j=\omega_i}$. Then a measurement of the signal x on the sample $\mathcal{A}$ writes

$$y = Mx + \mathcal{N} \tag{2}$$

where $\mathcal{N} \in \mathbb{R}^m$ is a measurement noise. In the following, we assume $\mathcal{N}$ is gaussian: $\mathcal{N}_i \sim \mathcal{N}(0, \sigma)$.

One shows that if x is a k-bandlimited signal, then one can hope to reconstruct x only if $m \geq k$. The goal of the sampling theory is, given a measurement budget m, to find $\mathcal{A}$ so that $|\mathcal{A}| = m$ and we can invert (2) so as to reconstruct exactly x from y. A sufficient condition for this to be true if $\mathcal{N} \equiv 0$ is that

$$MU_k \text{ has its smallest singular value } \sigma_1 > 0. \tag{3}$$

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

1.3 Reconstruction with determinantal point processes on graphs

Determinantal point processes (DPP) [3] are tools that are very useful in this situation, since they sample points in $\{1, \cdots, N\}$ with some negative correlation between points that are "close" to each other, that is "close" points will be less likely to be sampled together. In other terms, sampling from a DPP ensures the diversity of the sample.

In our case, if $\{1, \cdots, N\}$ are the nodes of the graph, the random variable $\mathcal{A}$ from a DPP of *marginal kernel* $K \in \mathbb{R}^{N \times N}$ verifies

$$\forall \mathcal{S} \subset \mathcal{A}, \qquad \mathbb{P}(\mathcal{S} \subset \mathcal{A}) = \det(K_{\mathcal{S}}) \tag{4}$$

where $K_{\mathcal{S}}$ is the submatrix of K with lines and columns indices in $\mathcal{S}$.

When sampling the measurement nodes, if x_1 and x_2 are two graph signals, we want to ensure that $\|x_1 - x_2\| > 0 \implies \|y_1 - y_2\| > 0$. One way to achieve this result is to reweight the measurement with a matrix such that the norm of the reweighted measurement is close to the norm of the original vector. If $\mathcal{A} = (\omega_1, \cdots, \omega_m)$, we define $P = \mathrm{diag}(\pi_{\omega_1}, \cdots, \pi_{\omega_m})$, and we have:

$$\mathbb{E}_{\mathcal{A}}\left(\left\|P^{-\frac{1}{2}} M x\right\|^2\right) = \|x\|^2 . \tag{5}$$

Thus, if we know the spanning eigenspace U_k, we can write the reconstruction problem as:

$$\begin{aligned} x_{\text{rec}} &= \mathrm{argmin}_{z \in \mathrm{span}(U_k)} \left\|P^{-\frac{1}{2}} (Mz - y)\right\|^2 \\ &= U_k \ \mathrm{argmin}_{\alpha \in \mathbb{R}^m} \left\|P^{-\frac{1}{2}} (M U_k \alpha - y)\right\|^2 \end{aligned}$$

$$\boxed{x_{\text{rec}} = U_k (U_k^\top M^\top P^{-1} M U_k)^{-1} U_k^\top M^\top P^{-1} y}. \tag{6}$$

In order to use the last formula, we must compute the inverse of an $N \times N$ matrix, which might be unfeasible if N is large. One could then use other methods, such as gradient descent.

The next step is then to find some marginal kernel K so that sampling $\mathcal{A}$ from a DPP of marginal kernel K (and defining the corresponding measurement matrix M accordingly) ensures $x_{\text{rec}} = x$ with (6). We can show that if we use the kernel

$$\boxed{K_k = U_k U_k^\top}, \tag{7}$$

then any sample $\mathcal{A}$ from this DPP verifies $|\mathcal{A}| = k$ a.s., (3) and perfect reconstruction can be achieved if there is no noise.

1.4 Reconstruction with unknown U_k using DPPs

In the preceding sections, we assumed U_k was known. It is possible that if N is very large, the computation of U_k becomes infeasible ; in this situation, we can not use the DPP given by K_k. A possible substitute is the kernel:

$$\boxed{K_q = U g_q(\Lambda) U^\top} \tag{8}$$

where $q > 0$ and $g_q(\lambda) = \frac{q}{q+\lambda}$. This is in fact an approximation of the kernel $K_k = U_k U_k^\top = U h_k(\Lambda) U^\top$ where $h_k(\lambda) = \mathbb{1}_{\lambda \leq \lambda_k}(\lambda)$. Moreover, there exists an Algorithm that enables sampling from the DPP of marginal kernel K_q without having to compute explicitely K_q, which makes this DPP particularly interesting (see Subsection 2.4).

If we do not know explicitely U_k, we can not use the direct reconstruction formula (6). Instead, we can use a regularized version of this formula that penalizes high frequencies:

$$\boxed{x_{\text{rec}} = \mathrm{argmin}_{z \in \mathbb{R}^N} \left\|P^{-\frac{1}{2}} (Mz - y)\right\|^2 + \gamma z^\top L^r z} \tag{9}$$

where $\gamma > 0$, $r > 0$ are regularization parameters.

2 Implementation details

This project was focused on implementation and experimentation. We used Python to implement the main algorithms. The code can be found here:

`https://github.com/14chanwa/graphsmlProject`

Our work decomposes in the following parts:

1. Generation of some graphs for benchmarking. We chose to work with community-structured graphs, as in [1].
2. Generation of k-bandlimited signals as in (1).
3. Sampling DPPs with marginal kernel K_k as in (7).
4. Sampling DPPs with marginal kernel K_q as in (8).
5. Reconstruct x from y as in (2) using the formula with known U_k (6).
6. Reconstruct x from y as in (2) using the formula without U_k (9).

Since L and W are sparse, we used functions from `scipy.sparse` as much as possible, for our implementation to scale with N.

2.1 Random graph generation

Function `generate_graph_from_stochastic_block_model`

For benchmarks, we chose to use community-structured graphs using the Stochastic Block Model (SBM) as in [1]. Consider a graph with k communities of cardinal N/k. An edge has a probability q_1 to be drawn between nodes i and j if they belong to the same community, and q_2 else. We can then parameterize the SBM with N, k, q_1, q_2, or alternatively with N, k, ϵ, c where $\epsilon = \frac{q_2}{q_1}$ and $c = q_1 \left(\frac{N}{k} - 1\right) + q_2 \left(N - \frac{N}{k}\right)$ is the average degree of a node in the graph. One can also show that there is a critical value of ϵ, $\epsilon_c = (c - \sqrt{c})/(c + \sqrt{c}(k-1))$ above which the community structure becomes undetectable for large N's, so we can also use the ratio ϵ/ϵ_c as a parameter.

We used the Python library NetworkX [4] to generate L, W and provide the necessary plot functions.

2.2 Generation of k-bandlimited signals

Function `generate_k_bandlimited_signal`

In order to generate a k-bandlimited signal, we must compute U_k and take a linear combination of these vectors as in (1). This is equivalent to compute the k lowest eigenmodes of the Laplacian matrix L. Even though we could use the function `numpy.linalg.eigh` directly, we chose to use functions that scale better with N. The function `scipy.sparse.eigh` uses a routine that enables to compute some eigenmodes from a sparse matrix. It is more efficient to compute the largest eigenmodes than the lowest ; we make use of the *shift-inverse* mode[1] of this function in order to compute the lowest eigenmodes of L by computing the largest eigenmodes of a dual problem.

2.3 Sampling DPPs with marginal kernel K_k

Function `sample_from_DPP`

We suppose U_k known ; then we can compute $K_k = U_k U_k^\top$. We then have to build an algorithm that samples a DPP from a given marginal kernel K ; we used the algorithm given in [1, 3] in order to do so.

Our version of this algorithm is presented in Algorithm 1. Its principle is, given the eigendecomposition of K, to first sample some of its eigenvectors with probabilities given by their corresponding eigenvalues. Then, we select recursively the nodes of the sample with probabilities given by the selected eigenvectors.

[1] See: `https://docs.scipy.org/doc/scipy/reference/tutorial/arpack.html`

Algorithm 1 Sampling a DPP with marginal kernel K

Input: Eigendecomposition of K: (Λ, V)
$J \leftarrow \emptyset$
for $n = 1..N$ **do**
  $J \leftarrow J \cup \{n\}$ with probability λ_n
end for
$V \leftarrow \{v_n\}_{n \in J}$
$Y \leftarrow \emptyset$
while $|V| > 0$ **do**
  $P \leftarrow (\|v_i\|^2)^\top_{i=1..|V|} / |V|$ (vector of size N)
  $i \leftarrow$ choice of i in $1..N$ with probability P
  $Y \leftarrow Y \cup \{i\}$
  if $|V| > 1$ **then**
    $j \leftarrow$ index of a v_j such that $\langle v_j, e_i \rangle \neq 0$.
    Remove a linear combination of v_j from $v_{j'}, j' \neq j$ such that $\forall j',\ \langle v_{j'}, e_i \rangle = 0$.
    Delete v_j from V.
    Use Gram-Schmidt algorithm to orthonormalize V.
  else
    break
  end if
end while
Return: Y

In order to use this algorithm on K_k, we must be able to compute at least the k first eigenmodes of K_k (since the other eigenvalues of K_k are null, we do not use the corresponding eigenmodes) ; we could use the same technique as in Subsection 2.2, but in this case the matrix is dense, so we might as well use `numpy`.

2.4 Sampling DPPs with marginal kernel K_q (Wilson's algorithm)

Functions `wilson_algorithm`, `generate_maze`

Test file `test_generate_maze.py`

This is the main contribution of [1] ; in order to avoid computing K_k (and thus U_k), the author uses an approximation K_q and gives an algorithm capable of sampling from the DPP of marginal kernel K_q without explicitely computing the kernel. This algorithm is based on a modified version of Propp-Wilson's algorithm [5].

Originally, this algorithm samples a random spanning tree on a given directed graph. A recreational application of this algorithm is random maze generation: consider a graph composed of nodes on a grid, plus a node that represents the border of the maze. The walls will consist in lines connecting these nodes: consider then the appropriate edges in this graph (two nodes are connected if they are adjacent, the border node is connected to all nodes in the border). Then when sampling a random spanning tree on this graph, one samples the walls of a maze such that there is one and only one way from one corridor to the other that do not imply turning back. We implemented this random maze generation in order to make sure our algorithm works, as presented in Figure 1.

The original algorithm works using loop-erased random walks in the graph until all nodes have been visited. In [1], the author modifies the algorithm so that it samples nodes from the graph by adding a sink node. At each step of the random walks, the walker has some probability depending on a parameter q to go to the sink: the last visited node is then part of the sample. This algorithm is presented in Algorithm 2.

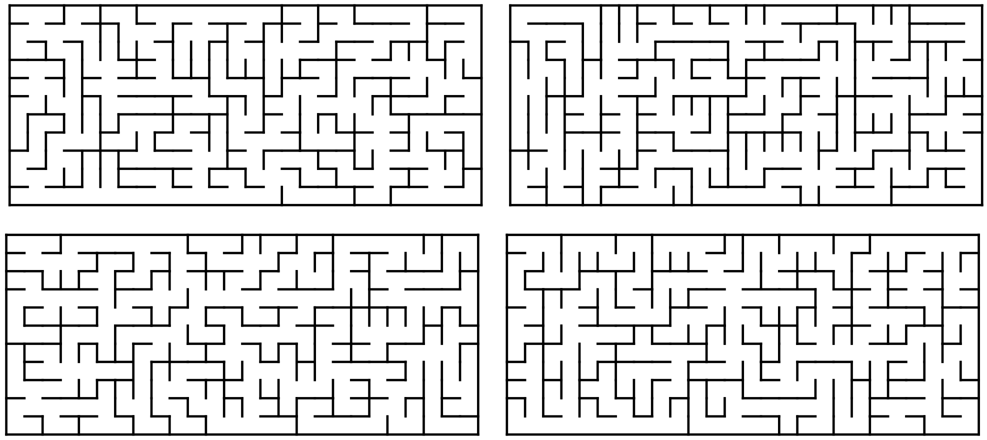


Figure 1: Examples of random mazes generated with random spanning trees on a grid of 25×10 wall nodes plus the border node.

Algorithm 2 Wilson's algorithm

Input: Adjacency matrix W, sink weight $q \geq 0$
$Y, \nu \leftarrow \emptyset$
while $\nu \neq \{1, \cdots, N\}$ **do**
  Begin a random walk S from a node $i \notin \nu$.
  if it reaches itself **then**
    Erase the loop in the walk.
  else if it reaches some node in ν or the sink Δ **then**
    $\nu \leftarrow \nu \cup S$
    if the last node is Δ **then**
      If ℓ is the last visited node before Δ, then $Y \leftarrow Y \cup \{\ell\}$.
    end if
  end if
end while
Return: Y

Note: our implementation also saves the sampled spanning tree. In the case $q = 0$, there is no sink and we have the original Propp-Wilson's algorithm.

3 Results

3.1 Sampling from K_k

Test file `test_sample_from_Kk.py`

In order to make sure we sample the right DPP with marginal kernel K_k with our Algorithm 1, we generate a graph from the SBM (as in Subsection 2.1) and sample from our algorithm a number of times n. If the algorithm is correct:

- Every sample is of size k (this is verified in practice).
- For any singleton i, $p(i \in \mathcal{A}) = K_{i,i}$ (4).
- For any pair i, j, $i < j$, $p(i \in \mathcal{A}$ and $j \in \mathcal{A}) = \det \begin{pmatrix} K_{i,i} & K_{i,j} \\ K_{j,i} & K_{j,j} \end{pmatrix}$ (4).
- For a given k-bandlimited signal x, $\mathbb{E}_{\mathcal{A}} \left(\left\| P^{-\frac{1}{2}} M x \right\|^2 \right) = \|x\|^2$ (5).

We choose two singletons and two pairs, and we compute the empirical probabilities and expectations over the $n = 50000$ samples, with $N = 100$, $k = 2$, $c = 16$, $\epsilon = 0.5 \times \epsilon_c$. The results are presented in Table 1.

Table 1: Verification of the correctness of the sampling algorithm for K_k.

Test	Theoretical proba	Empirical proba
Singleton 1	0.01895	0.01824
Singleton 2	0.01078	0.01006
Pair 1	0.00015	0.00016
Pair 2	0.00025	0.00020

Test	$\|x\|_2^2$	$\left\|P^{-1/2}Mx\right\|_2^2$
Norm	1.0	1.00074

We thus confirm that our algorithm samples the right DPP.

3.2 Sampling from K_q

Test file `test_sample_from_Kq.py`

We proceed the same way with our Algorithm 2 that samples from a DPP with marginal kernel K_q, with the difference that the size of the sample is not fixed. Rather, we should have:

$$\mathbb{E}\left(|\mathcal{A}|\right) = \sum_{i=1}^{N} \frac{q}{q+\lambda_i}. \tag{10}$$

Since this sampling algorithm is much slower than the preceding one, we chose to test the following parameters: $n = 10000$ samples, with $N = 100$, $k = 2$, $c = 16$, $\epsilon = 0.5 \times \epsilon_c$ and $q = 1.0$. The results are presented in Table 2.

Table 2: Verification of the correctness of the sampling algorithm for K_q.

Test	Theoretical proba	Empirical proba
Singleton 1	0.06752	0.06570
Singleton 2	0.05920	0.05590
Pair 1	0.00418	0.00450
Pair 2	0.00470	0.00490

Test	$\|x\|_2^2$	$\left\|P^{-1/2}Mx\right\|_2^2$
Norm	1.0	1.00290

Test	Theoretical	Empirical
Cardinal	6.95801	6.96870

We thus confirm that our algorithm samples the right DPP.

3.3 Signal reconstruction with K_k and known U_k

Test file `test_recovery_with_Kk.py`

In the following we choose $N = 100$, $k = 2$, $c = 16$, $\epsilon = 0.1 \times \epsilon_c$. We use the DPP of marginal kernel K_k to sample k measurement nodes, compute exactly P by using the diagonal of K_k and use the perfect reconstruction formula (6). The article presents the influence of ϵ on the reconstruction error ; we chose to compute the influence of the noise standard deviation σ of the noise $\mathcal{N}$ on the reconstruction error, where

$$y = Mx + \mathcal{N}.$$

We generate 100 graphs using the preceding parameters, and on each graph we generate 100 k-bandlimited signals. On each graph, we sample k measurement nodes and on each signal, we add $\mathcal{N}$ with a given σ. We then compute the reconstruction error on the signals (using the same DPP sample).

When $\mathcal{N} \equiv 0$, our algorithm reconstructs the signal perfectly, up to the machine precision (the mean error is $\sim 10^{-15}$, the max error is $\sim 10^{-8}$). When we increase σ, the reconstruction error increases. The results are presented in Figure 2.

We note that there are rare cases in which our reconstruction algorithm fails spectacularly. The max reconstruction errors are presented in Table 3.

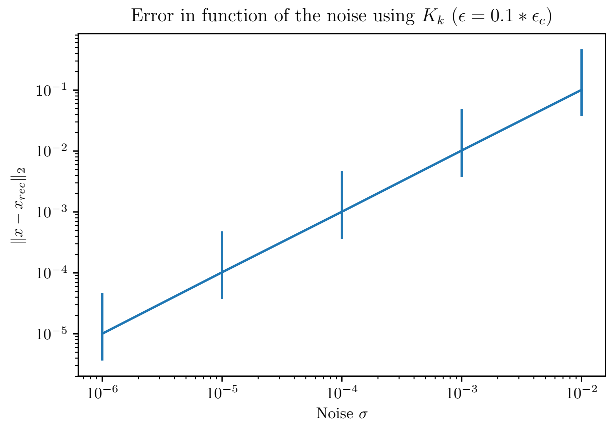


Figure 2: Influence of σ on the reconstruction error (DPP of marginal kernel K_k, reconstruction using (6)). The error bars are the 10^{th} and 90^{th} percentiles.

Table 3: Max reconstruction error (DPP of marginal kernel K_k, reconstruction using (6))

Noise σ	Max reconstruction error
10^{-6}	0.16
10^{-5}	1.46
10^{-4}	5.88
10^{-3}	64
10^{-2}	675

3.4 Signal reconstruction with K_q and known U_k

Test file `test_recovery_with_Kq.py`

In this section, we still assume U_k is known. The kernel K_q is build so as to be an approximation of K_k. This enables us to sample from K_q without having to explicitely compute K_q. We thus use the sampling algorithm with K_q to select the measurement nodes and reconstruct the signal using (6) just as in the preceding Subsection. We use the parameters $N = 100$, $k = 2$, $c = 16$, $\epsilon = 0.1 \times \epsilon_c$ and sample 100 graphs and 100 signals on each graph, then add the corresponding noise levels.

We can not set a specific cardinal for the samples of our DPP with kernel K_q ; it depends on a parameter q. We chose to fix and initial $q_{\text{init}} = 0.2$ and a desired cardinal m_{des} and increase q if $m - \sqrt{m} > m_{\text{des}}$ or decrease q if $m + \sqrt{m} < m_{\text{des}}$ with the algorithm suggested in [6]. In our tests, the mean cardinal of our sample is 3.11 (with $m_{\text{des}} = 2$).

The results are presented in Figure 3. We see that the reconstruction noise is comparable to the algorithm sampling from K_k. This means that K_q is effectively a good approximation of K_k. We could argue that this comparison might be unfair since the mean cardinal of our sample is $3.11 > k = 2$; yet the algorithm K_k can not produce samples of size $\neq k$, whereas in Wilson's algorithm we can set the expected sample size with the parameter q, that makes this algorithm more adaptable.

Similarly, there are cases in which the algorithm fails spectacularly, as presented in Table 4.

3.5 Signal reconstruction with K_q and unknown U_k

Test file `test_recovery_with_Kq_unknown_Uk.py`

For now, we only used (6), which requires the knowledge of $P = \text{diag}(\pi_{\omega_1}, \cdots, \pi_{\omega_1})$ and U_k. Yet in order to compute P, one has to compute the marginal kernel associated to the DPP used to sample the nodes and U_k. This is exactly what we wanted to avoid in the first place.

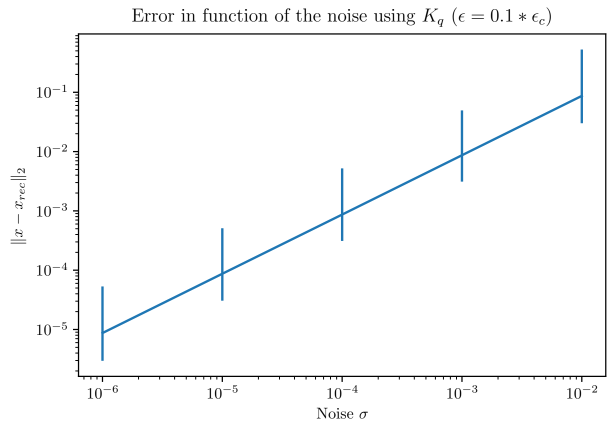


Figure 3: Influence of σ on the reconstruction error (DPP of marginal kernel K_q, mean cardinal 3.11, reconstruction using (6)). The error bars are the 10^{th} and 90^{th} percentiles.

Table 4: Max reconstruction error (DPP of marginal kernel K_q, mean cardinal 3.11, reconstruction using (6))

Noise σ	Max reconstruction error
10^{-6}	0.06
10^{-5}	1.56
10^{-4}	12.9
10^{-3}	187
10^{-2}	1091

In [1], the authors present an algorithm based on fast filtering on graphs to estimate P, and use a regularized reconstruction formula (9) to penalize high frequencies:

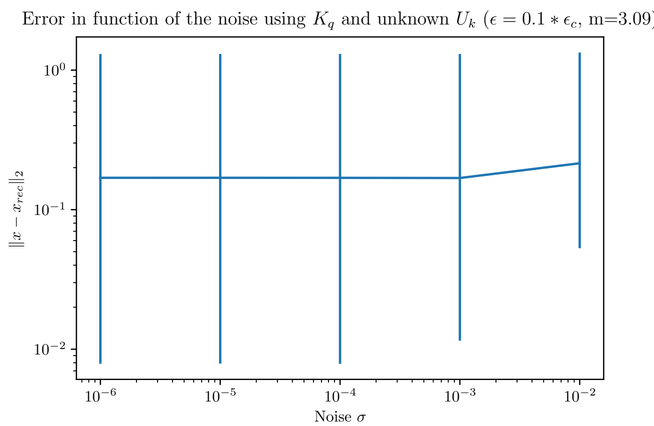
$$x_{\text{rec}} = \text{argmin}_{z \in \mathbb{R}^N} \left\| P^{-\frac{1}{2}} \left(Mz - y \right) \right\|^2 + \gamma z^\top L^r z$$

$$\boxed{x_{\text{rec}} = (M^\top P^{-1} M + \gamma L^r)^{-1} M^\top P^{-1} y}. \tag{11}$$

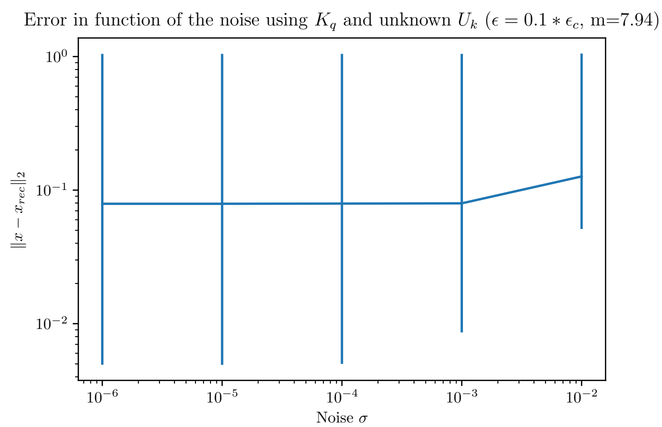
We chose not to focus on the algorithm that estimates P without computing K. We assume here that P is known ; this assumption does not hold for large N, since the knowledge of P requires the computation of K.

As before, using the direct formula (11) implies a $N \times N$ matrix inversion, which might be intractable. One could use optimization algorithms such as gradient descent instead.

We benchmark the reconstruction performance of the algorithm using measurement nodes sampled from K_q and (11). We use parameters $N = 100$, $k = 2$, $c = 16$, $\epsilon = 0.1 \times \epsilon_c$, $\gamma = 10^{-5}$, $r = 4$ and sample 100 graphs and 100 signals on each graph. As in the preceding Subsection, we use an initial parameter $q_{\text{init}} = 0.2$ and a desired cardinal m_{des} and increase q if $m - \sqrt{m} > m_{\text{des}}$ or decrease q if $m + \sqrt{m} < m_{\text{des}}$. This leads to a mean cardinal of $m = 3.09$ (for $m_{\text{des}} = 2$). We also run a test increasing the number of measures to a mean of $m = 7.94$ (for $m_{\text{des}} = 10$). We measure the influence of the noise level on the reconstruction results. The results are presented in Figure 4. Contrary to the preceding Subsections, we do not achieve perfect reconstruction when the noise is null – in this case the results are similar to those with $\sigma = 10^{-6}$. As expected, the results are not as good as the preceding Subsections, an error of $\sim 10^0$ being close to the error made by sampling a random signal of norm 1, but the noise level do not seem to influence greatly the reconstruction performance. We explain this as the fact that the regularization cancels the high frequencies, that make it less sensitive to noise. As before, there are cases in which the reconstruction fails as shown in Table 5. Overall, the reconstruction results are not very consistent as the dispersion of the error is quite large.



a.



b.

Figure 4: Influence of σ on the reconstruction error (DPP of marginal kernel K_q, mean cardinal **a.** 3.09 and **b.** 7.94, reconstruction using (11)). The error bars are the 10^{th} and 90^{th} percentiles.

Table 5: Max reconstruction error (DPP of marginal kernel K_q, mean cardinal 3.09 and 7.94, reconstruction using (11))

Noise σ	Max reconstruction error ($m = 3.09$)	Max reconstruction error ($m = 7.94$)
10^{-6}	19.19	10.63
10^{-5}	19.19	10.63
10^{-4}	19.18	10.63
10^{-3}	18.90	10.61
10^{-2}	19.09	10.10

4 Conclusion

In this project, we implemented algorithms from [1] and showed that indeed, when U_k is known, sampling from a DPP of marginal kernel K_k enables a good signal reconstruction relatively robust to noise. We also showed that K_q is a suitable approximation of K_k, and implemented a random walk algorithm that enables to sample from K_q without having to compute explicitely the kernel. Finally, we implemented signal reconstruction without known U_k using a regularized criterion to penalize high frequencies in the reconstructed signal, and showed that noise do not strongly affect the results, that seem relatively inconsistent nonetheless.

References

[1] Nicolas Tremblay, Pierre-Olivier Amblard, and Simon Barthelmé. Graph sampling with determinantal processes. *EUSIPCO*, August 2017.

[2] David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. Signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular data domains. *CoRR*, abs/1211.0053, 2012.

[3] Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning*, 5(2-3):123–286, July 2012.

[4] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[5] James Gary Propp and David Bruce Wilson. How to get a perfectly random sample from a generic markov chain and generate a random spanning tree of a directed graph. *J. Algorithms*, 27(2):170–217, May 1998.

[6] Luca Avena and Alexandre Gaudillière. On some random forests with determinantal roots. *arXiv:1310.1723v4*, October 2013.