High-performance sampling of Determinantal Point Processes

Jack Poulson

HODGE STAR
hodgestar.com

Numerical algorithms for high-perf. computational science
The Royal Society, London
April 08, 2019

Overview

- Will draw strong connection between techniques for efficiently **factoring matrices** and for **sampling structured subsets** of a ground set.
- The basic bridge: forming a **Schur complement** equates to forming a representation of a **conditional distribution**.
- One can import HPC techniques, such as **DAG-scheduled** dense and sparse-direct **blocked algorithms**, from factorizations to **Determinantal Point Processes** [Macchi-1975, Burton/Pemantle-1993, Benjamini/Lyons/Peres/Schramm-2001].
- Implementations are available in the permissively licensed, header-only C++14 package Catamari [P-2018] available at hodgestar.com/catamari.

Overview

- Will draw strong connection between techniques for efficiently **factoring matrices** and for **sampling structured subsets** of a ground set.
- The basic bridge: forming a **Schur complement** equates to forming a representation of a **conditional distribution**.
- One can import HPC techniques, such as **DAG-scheduled** dense and sparse-direct **blocked algorithms**, from factorizations to **Determinantal Point Processes** [Macchi-1975, Burton/Pemantle-1993, Benjamini/Lyons/Peres/Schramm-2001].
- Implementations are available in the permissively licensed, header-only C++14 package Catamari [P-2018] available at hodgestar.com/catamari.

Overview

- Will draw strong connection between techniques for efficiently **factoring matrices** and for **sampling structured subsets** of a ground set.
- The basic bridge: forming a **Schur complement** equates to forming a representation of a **conditional distribution**.
- One can import HPC techniques, such as **DAG-scheduled** dense and sparse-direct **blocked algorithms**, from factorizations to **Determinantal Point Processes** [Macchi-1975, Burton/Pemantle-1993, Benjamini/Lyons/Peres/Schramm-2001].
- Implementations are available in the permissively licensed, header-only C++14 package Catamari [P-2018] available at hodgestar.com/catamari.

Overview

- Will draw strong connection between techniques for efficiently **factoring matrices** and for **sampling structured subsets** of a ground set.
- The basic bridge: forming a **Schur complement** equates to forming a representation of a **conditional distribution**.
- One can import HPC techniques, such as **DAG-scheduled** dense and sparse-direct **blocked algorithms**, from factorizations to **Determinantal Point Processes** [Macchi-1975, Burton/Pemantle-1993, Benjamini/Lyons/Peres/Schramm-2001].
- Implementations are available in the permissively licensed, header-only C++14 package Catamari [P-2018] available at hodgestar.com/catamari.

Main idea: pivots as inclusion probabilities

Sampling a DPP can be reinterpreted as 'factoring' a class of matrices such that the j'th pivot is the probability of including the j'th item.

Flip a coin weighted by the pivot to determine inclusion:

- If the item is kept, proceed as in an LU/LDL factorization.
- If the item is dropped, take the pivot's complement in $[0, 1]$ and negate – i.e., subtract one – and proceed as normal.

The likelihood of the sample is thus the product of the absolute value of the diagonal of the 'factorization'.

Essentially all high-performance techniques for dense and sparse-direct factorizations therefore carry over.

Main idea: pivots as inclusion probabilities

Sampling a DPP can be reinterpreted as 'factoring' a class of matrices such that the j'th pivot is the probability of including the j'th item.

Flip a coin weighted by the pivot to determine inclusion:

- If the item is kept, proceed as in an LU/LDL factorization.
- If the item is dropped, take the pivot's complement in $[0, 1]$ and negate – i.e., subtract one – and proceed as normal.

The likelihood of the sample is thus the product of the absolute value of the diagonal of the 'factorization'.

Essentially all high-performance techniques for dense and sparse-direct factorizations therefore carry over.

Main idea: pivots as inclusion probabilities

Sampling a DPP can be reinterpreted as 'factoring' a class of matrices such that the j'th pivot is the probability of including the j'th item.

Flip a coin weighted by the pivot to determine inclusion:

- If the item is kept, proceed as in an LU/LDL factorization.
- If the item is dropped, take the pivot's complement in $[0, 1]$ and negate – i.e., subtract one – and proceed as normal.

The likelihood of the sample is thus the product of the absolute value of the diagonal of the 'factorization'.

Essentially all high-performance techniques for dense and sparse-direct factorizations therefore carry over.

Main idea: pivots as inclusion probabilities

Sampling a DPP can be reinterpreted as 'factoring' a class of matrices such that the j'th pivot is the probability of including the j'th item.

Flip a coin weighted by the pivot to determine inclusion:

- If the item is kept, proceed as in an LU/LDL factorization.
- If the item is dropped, take the pivot's complement in $[0, 1]$ and negate – i.e., subtract one – and proceed as normal.

The likelihood of the sample is thus the product of the absolute value of the diagonal of the 'factorization'.

Essentially all high-performance techniques for dense and sparse-direct factorizations therefore carry over.

Main idea: pivots as inclusion probabilities

Sampling a DPP can be reinterpreted as 'factoring' a class of matrices such that the j'th pivot is the probability of including the j'th item.

Flip a coin weighted by the pivot to determine inclusion:

- If the item is kept, proceed as in an LU/LDL factorization.
- If the item is dropped, take the pivot's complement in $[0,1]$ and negate – i.e., subtract one – and proceed as normal.

The likelihood of the sample is thus the product of the absolute value of the diagonal of the 'factorization'.

Essentially all high-performance techniques for dense and sparse-direct factorizations therefore carry over.

What is meant by a 'structured subset'?

The basic mechanism of a (finite) **Point Process** is to define a probability distribution over the power set of a ground set $[n] = [0, ..., n-1]$.

A **determinantal** point process sets the probability of a subset $J \subseteq [n]$ being in the sample equal to the J-minor of a fixed **marginal kernel matrix**.

The kernel matrix is often assumed Hermitian positive semi-definite – with spectrum in $[0, 1]$, but Hermiticity does not hold in some important cases.

Inadmissible combinations of members of the set can therefore be encoded through linear dependencies in the kernel matrix.

Before diving into the details, it will be instructive to describe some Hermitian and non-Hermitian standard DPPs.

What is meant by a 'structured subset'?

The basic mechanism of a (finite) **Point Process** is to define a probability distribution over the power set of a ground set $[n] = [0, ..., n-1]$.

A **determinantal** point process sets the probability of a subset $J \subseteq [n]$ being in the sample equal to the J-minor of a fixed **marginal kernel matrix**.

The kernel matrix is often assumed Hermitian positive semi-definite – with spectrum in $[0, 1]$, but Hermiticity does not hold in some important cases.

Inadmissible combinations of members of the set can therefore be encoded through linear dependencies in the kernel matrix.

Before diving into the details, it will be instructive to describe some Hermitian and non-Hermitian standard DPPs.

What is meant by a 'structured subset'?

The basic mechanism of a (finite) **Point Process** is to define a probability distribution over the power set of a ground set $[n] = [0, ..., n-1]$.

A **determinantal** point process sets the probability of a subset $J \subseteq [n]$ being in the sample equal to the J-minor of a fixed **marginal kernel matrix**.

The kernel matrix is often assumed Hermitian positive semi-definite – with spectrum in $[0, 1]$, but Hermiticity does not hold in some important cases.

Inadmissible combinations of members of the set can therefore be encoded through linear dependencies in the kernel matrix.

Before diving into the details, it will be instructive to describe some Hermitian and non-Hermitian standard DPPs.

What is meant by a 'structured subset'?

The basic mechanism of a (finite) **Point Process** is to define a probability distribution over the power set of a ground set $[n] = [0, ..., n-1]$.

A **determinantal** point process sets the probability of a subset $J \subseteq [n]$ being in the sample equal to the J-minor of a fixed **marginal kernel matrix**.

The kernel matrix is often assumed Hermitian positive semi-definite – with spectrum in $[0, 1]$, but Hermiticity does not hold in some important cases.

Inadmissible combinations of members of the set can therefore be encoded through linear dependencies in the kernel matrix.

Before diving into the details, it will be instructive to describe some Hermitian and non-Hermitian standard DPPs.

What is meant by a 'structured subset'?

The basic mechanism of a (finite) **Point Process** is to define a probability distribution over the power set of a ground set $[n] = [0, ..., n-1]$.

A **determinantal** point process sets the probability of a subset $J \subseteq [n]$ being in the sample equal to the J-minor of a fixed **marginal kernel matrix**.

The kernel matrix is often assumed Hermitian positive semi-definite – with spectrum in $[0, 1]$, but Hermiticity does not hold in some important cases.

Inadmissible combinations of members of the set can therefore be encoded through linear dependencies in the kernel matrix.

Before diving into the details, it will be instructive to describe some Hermitian and non-Hermitian standard DPPs.

What is meant by a 'structured subset'?

The basic mechanism of a (finite) **Point Process** is to define a probability distribution over the power set of a ground set $[n] = [0, ..., n-1]$.

A **determinantal** point process sets the probability of a subset $J \subseteq [n]$ being in the sample equal to the J-minor of a fixed **marginal kernel matrix**.

The kernel matrix is often assumed Hermitian positive semi-definite – with spectrum in $[0, 1]$, but Hermiticity does not hold in some important cases.

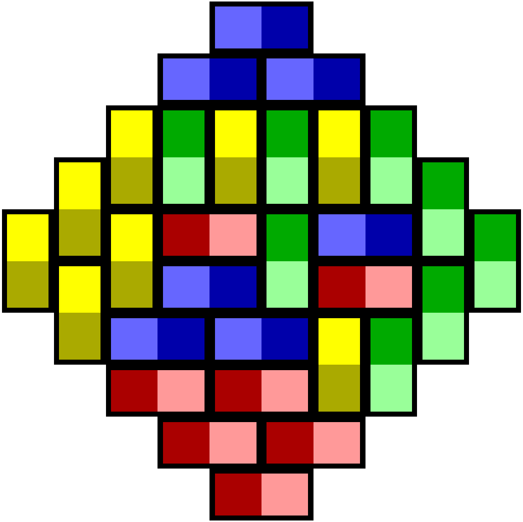
Inadmissible combinations of members of the set can therefore be encoded through linear dependencies in the kernel matrix.

Before diving into the details, it will be instructive to describe some Hermitian and non-Hermitian standard DPPs.

Aztec diamond: $d = 5$

```
$ ./aztec_diamond --diamond_size=5
```

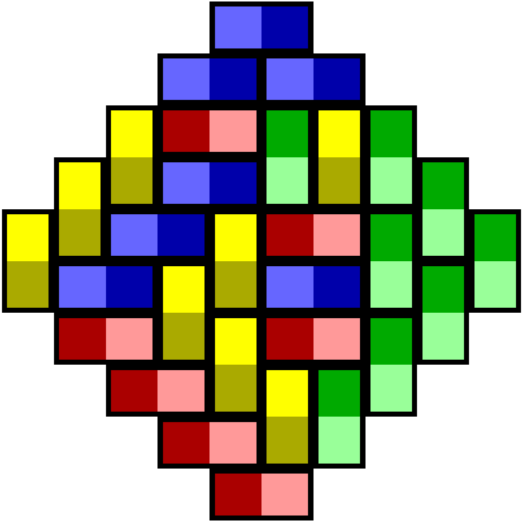
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-10.3972)$



Aztec diamond: $d = 5$

```
$ ./aztec_diamond --diamond_size=5
```

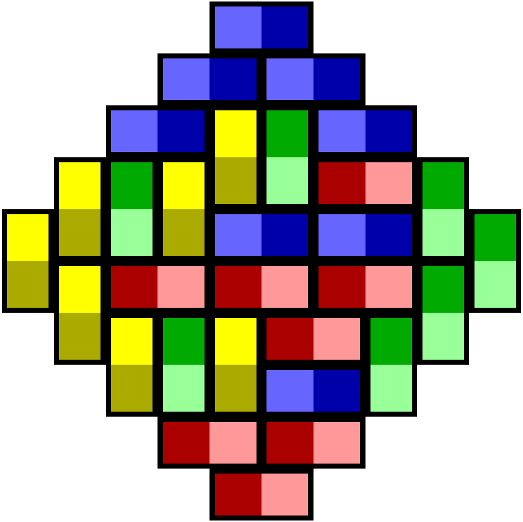
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-10.3972)$



Aztec diamond: $d = 5$

```
$ ./aztec_diamond --diamond_size=5
```

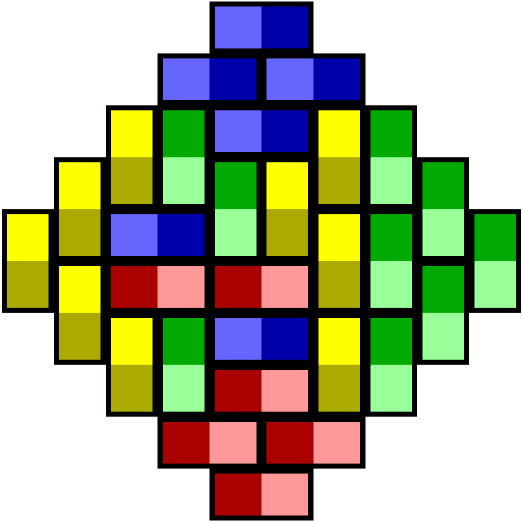
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-10.3972)$



Aztec diamond: $d = 5$

```
$ ./aztec_diamond --diamond_size=5
```

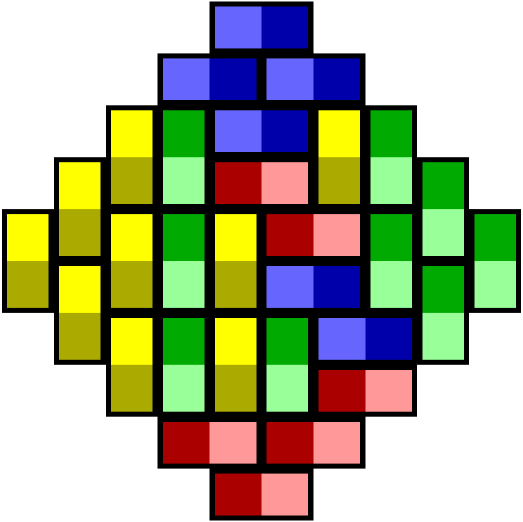
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-10.3972)$



Aztec diamond: $d = 5$

```
$ ./aztec_diamond --diamond_size=5
```

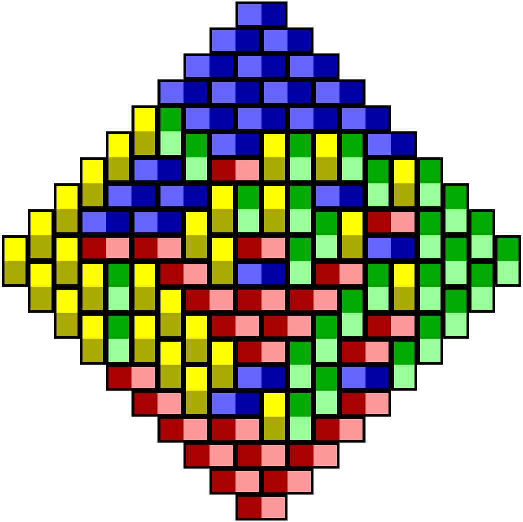
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-10.3972)$



Aztec diamond: $d = 10$

```
$ ./aztec_diamond --diamond_size=10
```

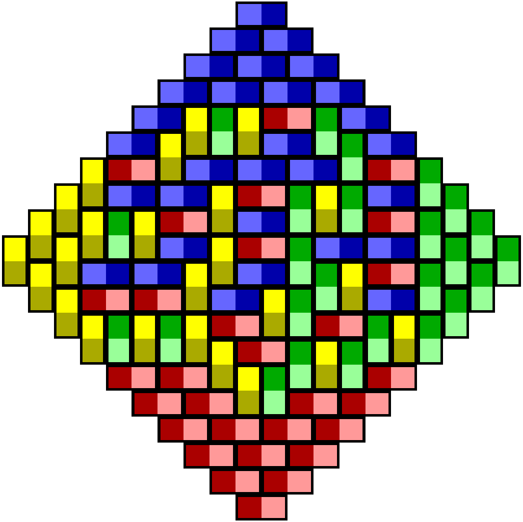
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-38.1231)$



Aztec diamond: $d = 10$

```
$ ./aztec_diamond --diamond_size=10
```

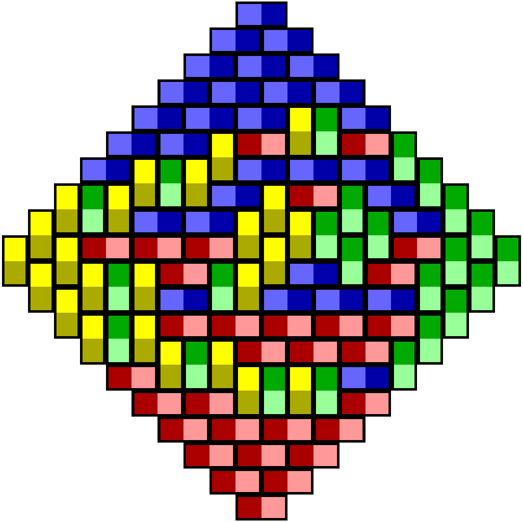
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-38.1231)$



Aztec diamond: $d = 10$

```
$ ./aztec_diamond --diamond_size=10
```

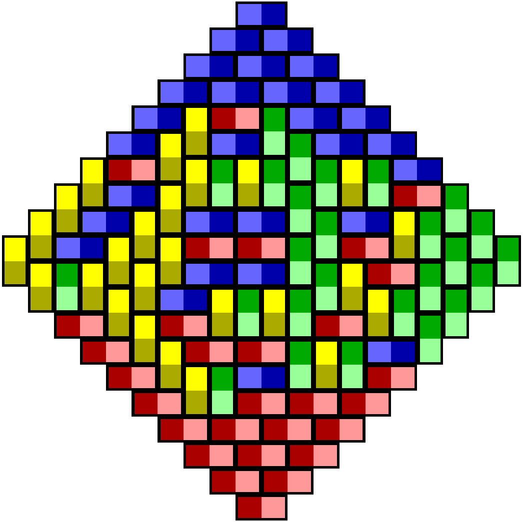
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-38.1231)$



Aztec diamond: $d = 10$

```
$ ./aztec_diamond --diamond_size=10
```

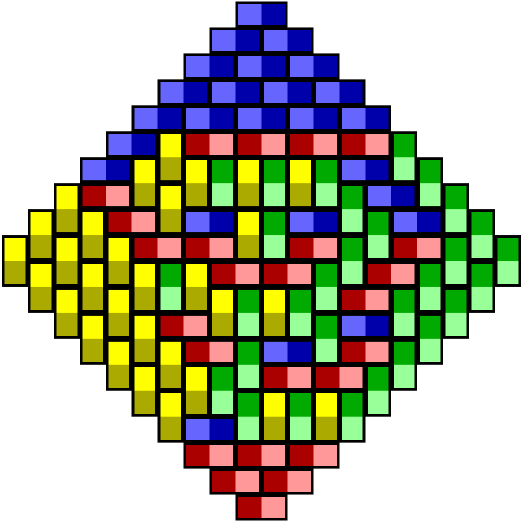
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-38.1231)$



Aztec diamond: $d = 10$

```
$ ./aztec_diamond --diamond_size=10
```

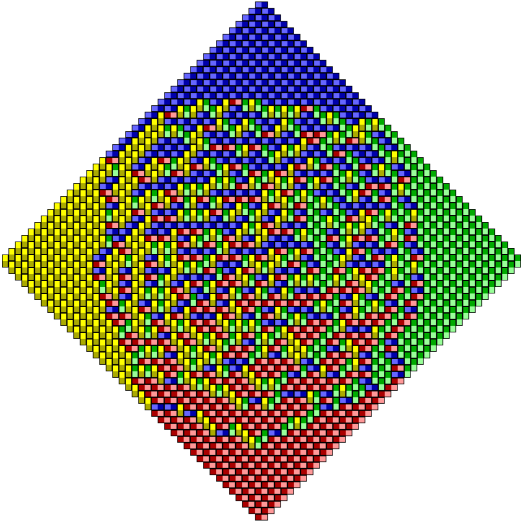
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-38.1231)$



Aztec diamond: $d = 40$

```
$ ./aztec_diamond --diamond_size=40
```

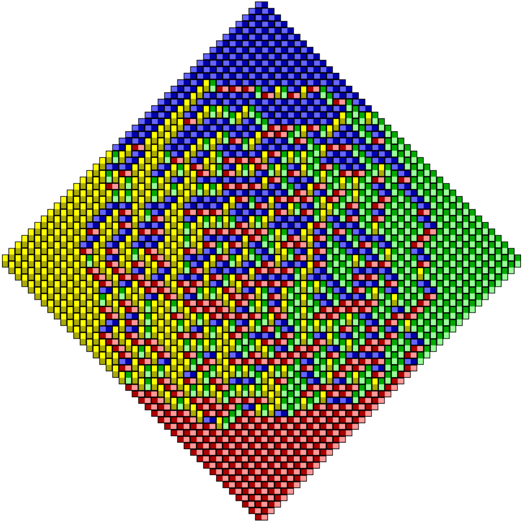
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-568.381)$



Aztec diamond: $d = 40$

```
$ ./aztec_diamond --diamond_size=40
```

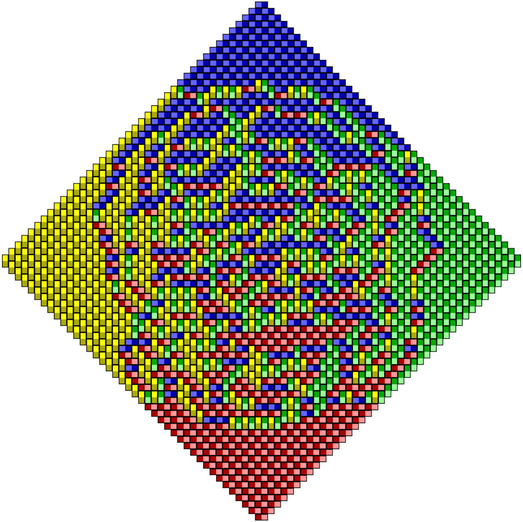
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-568.381)$



Aztec diamond: $d = 40$

```
$ ./aztec_diamond --diamond_size=40
```

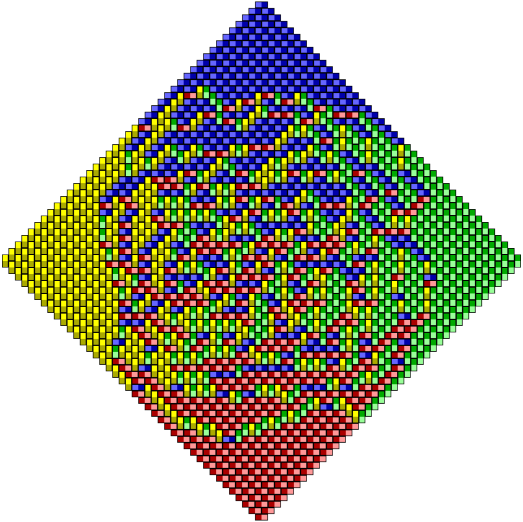
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-568.381)$



Aztec diamond: $d = 40$

```
$ ./aztec_diamond --diamond_size=40
```

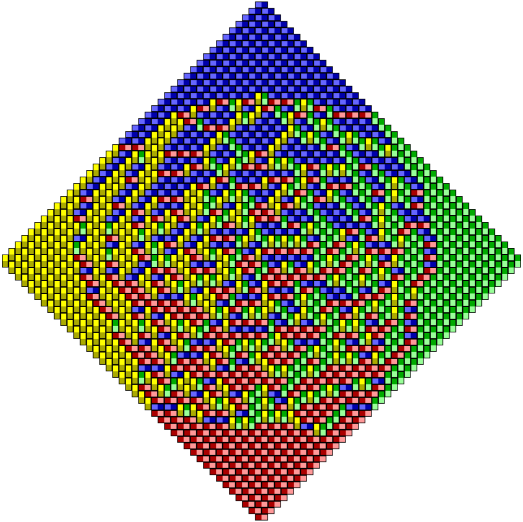
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-568.381)$



Aztec diamond: $d = 40$

```
$ ./aztec_diamond --diamond_size=40
```

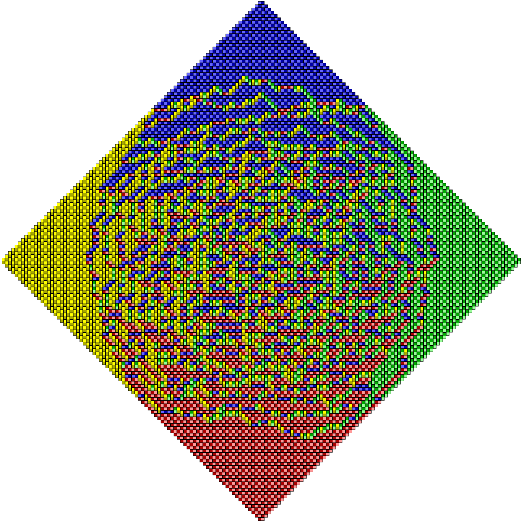
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-568.381)$



Aztec diamond: $d = 80$

```
$ ./aztec_diamond --diamond_size=80
```

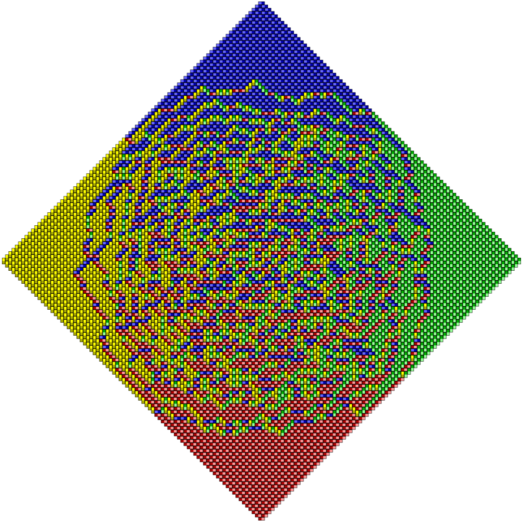
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-2245.8)$



Aztec diamond: $d = 80$

```
$ ./aztec_diamond --diamond_size=80
```

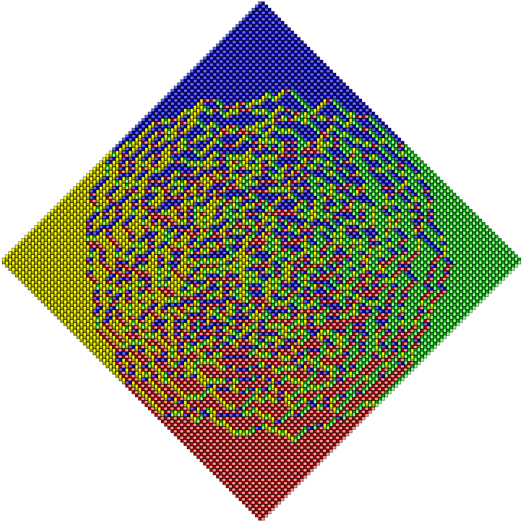
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-2245.8)$



Aztec diamond: $d = 80$

```
$ ./aztec_diamond --diamond_size=80
```

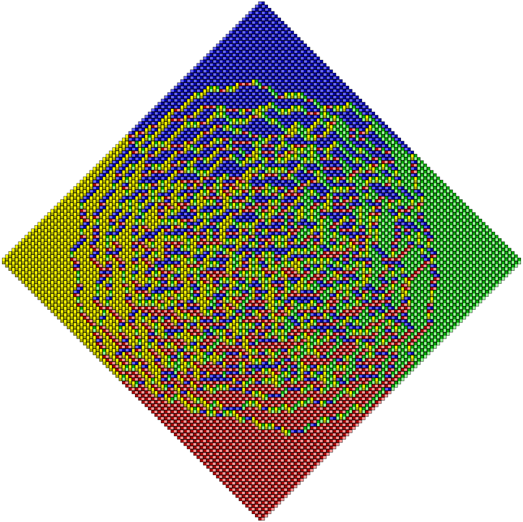
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-2245.8)$



Aztec diamond: $d = 80$

```
$ ./aztec_diamond --diamond_size=80
```

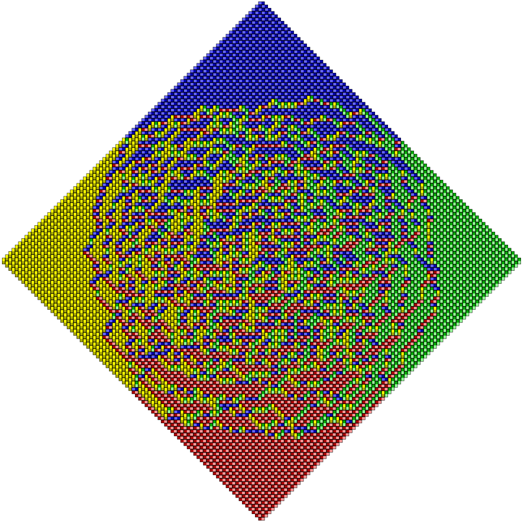
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-2245.8)$



Aztec diamond: $d = 80$

```
$ ./aztec_diamond --diamond_size=80
```

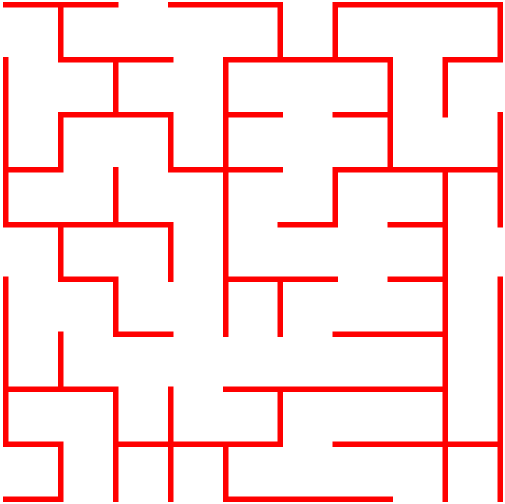
Complex non-Hermitian kernel; Sample likelihoods: $\exp(-2245.8)$



Uniform Spanning Tree in $\mathbb{Z}^2$ ($d = 10$)

```
$ ./uniform_spanning_tree --x_size=10 --y_size=10
```

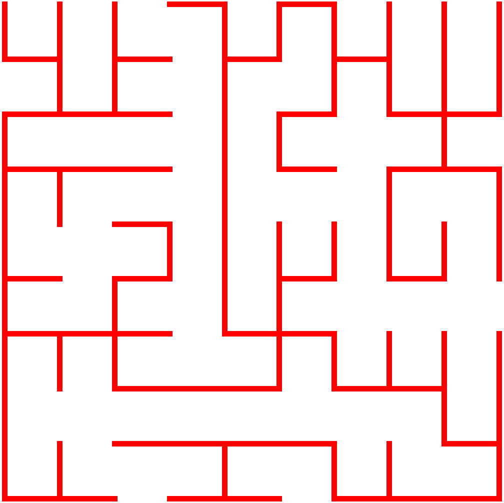
Real-symm' elementary kernel; Sample likelihoods: $\exp(-98.448)$



Uniform Spanning Tree in $\mathbb{Z}^2$ ($d = 10$)

```
$ ./uniform_spanning_tree --x_size=10 --y_size=10
```

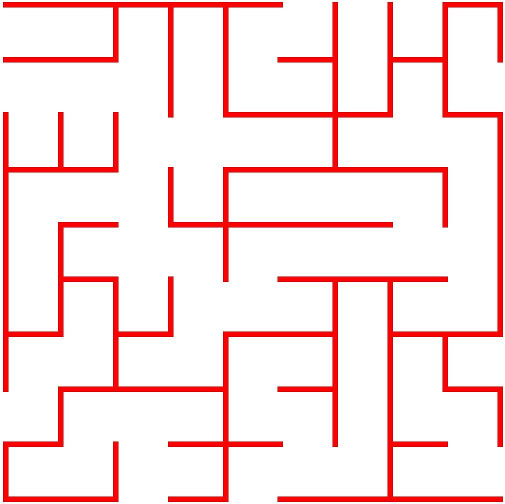
Real-symm' elementary kernel; Sample likelihoods: $\exp(-98.448)$



Uniform Spanning Tree in $\mathbb{Z}^2$ $(d = 10)$

```
$ ./uniform_spanning_tree --x_size=10 --y_size=10
```

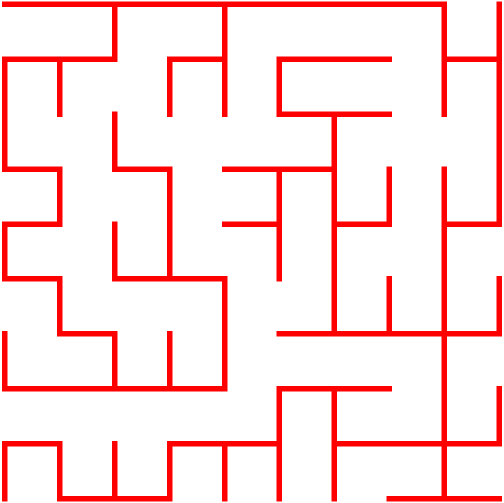
Real-symm' elementary kernel; Sample likelihoods: $\exp(-98.448)$



Uniform Spanning Tree in $\mathbb{Z}^2$ ($d = 10$)

```
$ ./uniform_spanning_tree --x_size=10 --y_size=10
```

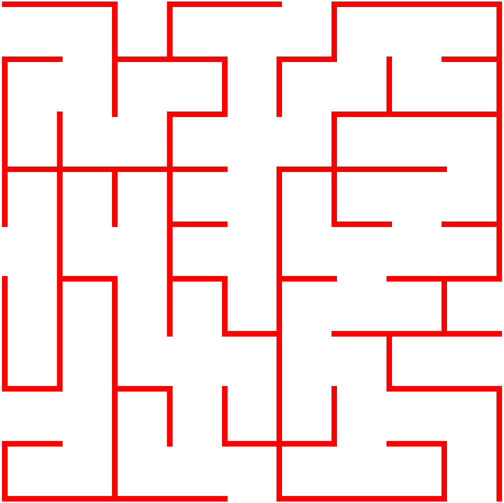
Real-symm' elementary kernel; Sample likelihoods: $\exp(-98.448)$



Uniform Spanning Tree in $\mathbb{Z}^2$ ($d = 10$)

```
$ ./uniform_spanning_tree --x_size=10 --y_size=10
```

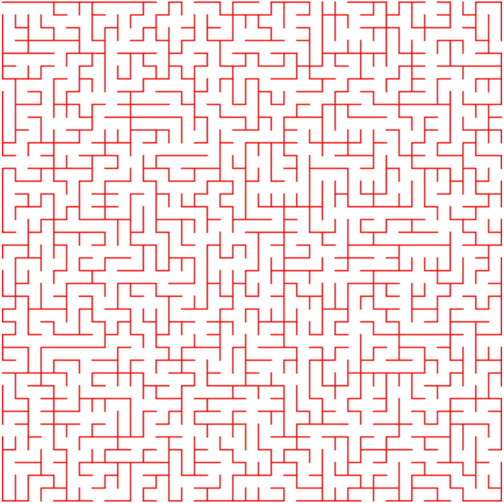
Real-symm' elementary kernel; Sample likelihoods: $\exp(-98.448)$



Uniform Spanning Tree in $\mathbb{Z}^2$ ($d = 40$)

```
$ ./uniform_spanning_tree --x_size=40 --y_size=40
```

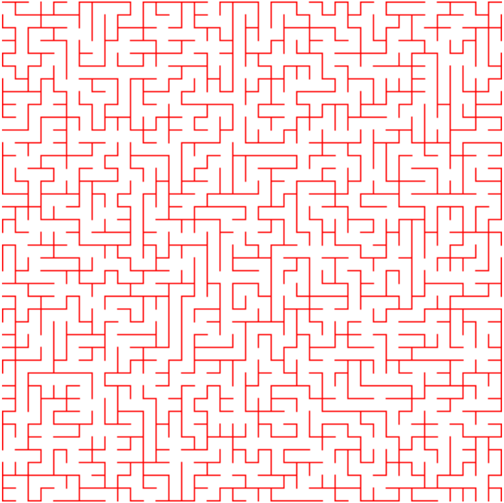
Real-symm' elementary kernel; Sample likelihoods: $\exp(-1794.24)$



Uniform Spanning Tree in $\mathbb{Z}^2$ ($d = 40$)

```
$ ./uniform_spanning_tree --x_size=40 --y_size=40
```

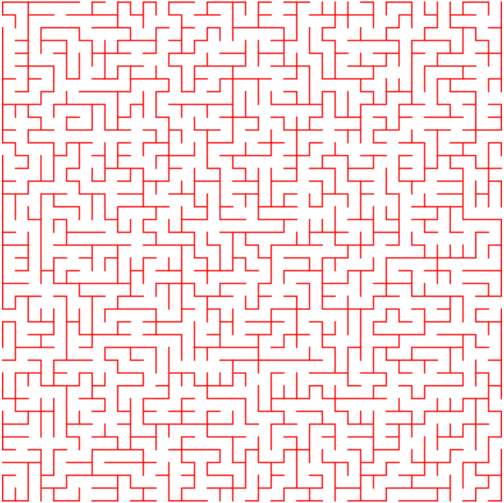
Real-symm' elementary kernel; Sample likelihoods: $\exp(-1794.24)$



Uniform Spanning Tree in $\mathbb{Z}^2$ ($d = 40$)

```
$ ./uniform_spanning_tree --x_size=40 --y_size=40
```

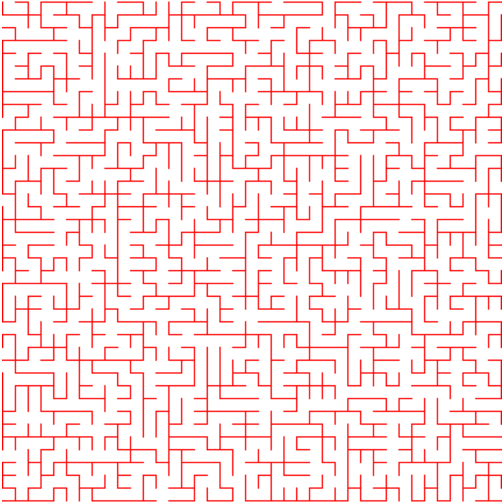
Real-symm' elementary kernel; Sample likelihoods: $\exp(-1794.24)$



Uniform Spanning Tree in $\mathbb{Z}^2$ $(d = 40)$

```
$ ./uniform_spanning_tree --x_size=40 --y_size=40
```

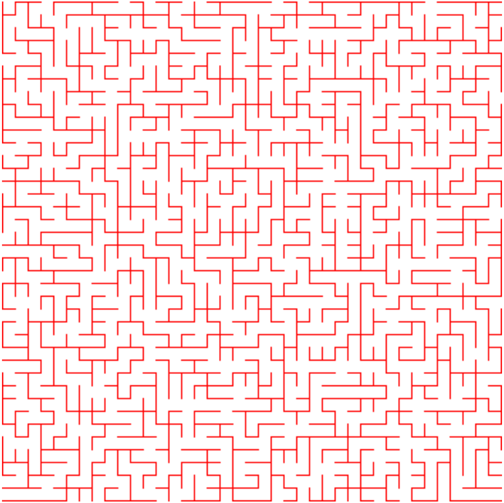
Real-symm' elementary kernel; Sample likelihoods: $\exp(-1794.24)$



Uniform Spanning Tree in $\mathbb{Z}^2$ ($d = 40$)

```
$ ./uniform_spanning_tree --x_size=40 --y_size=40
```

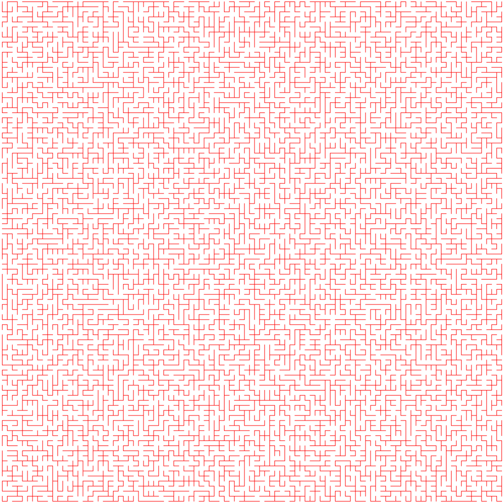
Real-symm' elementary kernel; Sample likelihoods: $\exp(-1794.24)$



Uniform Spanning Tree in $\mathbb{Z}^2$ ($d = 100$)

```
$ ./uniform_spanning_tree --x_size=100 --y_size=100
```

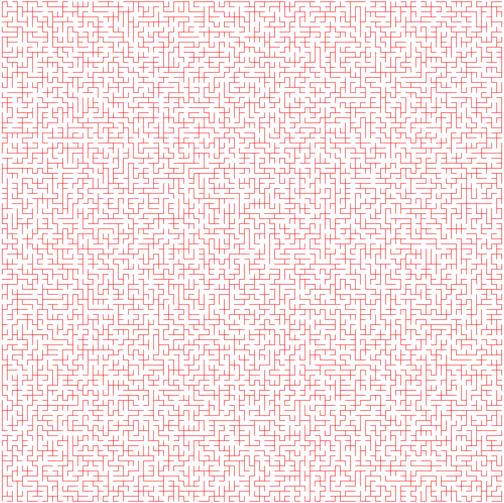
Real-symm' elementary kernel; Sample likelihoods: $\exp(-11,484.5)$



Uniform Spanning Tree in $\mathbb{Z}^2$ ($d = 100$)

```
$ ./uniform_spanning_tree --x_size=100 --y_size=100
```

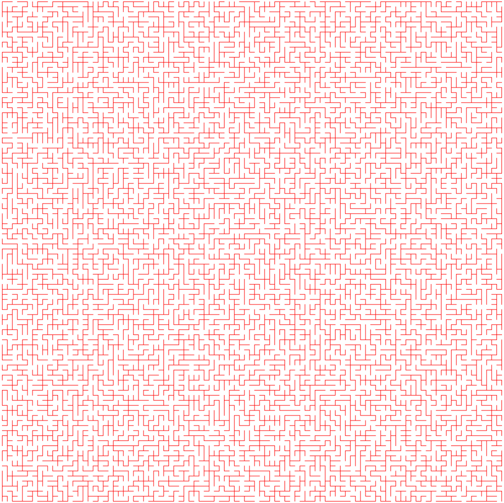
Real-symm' elementary kernel; Sample likelihoods: $\exp(-11,484.5)$



Uniform Spanning Tree in $\mathbb{Z}^2$ ($d = 100$)

```
$ ./uniform_spanning_tree --x_size=100 --y_size=100
```

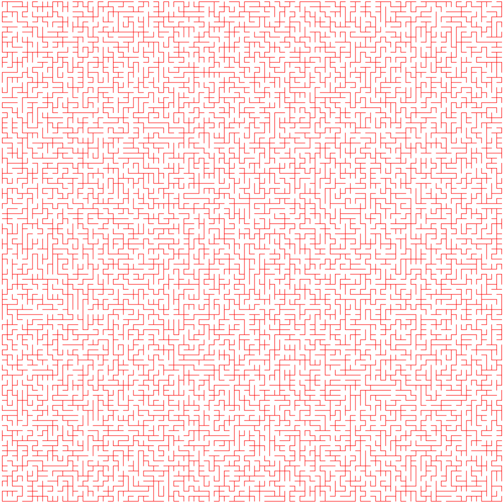
Real-symm' elementary kernel; Sample likelihoods: $\exp(-11,484.5)$



Uniform Spanning Tree in $\mathbb{Z}^2$ ($d = 100$)

```
$ ./uniform_spanning_tree --x_size=100 --y_size=100
```

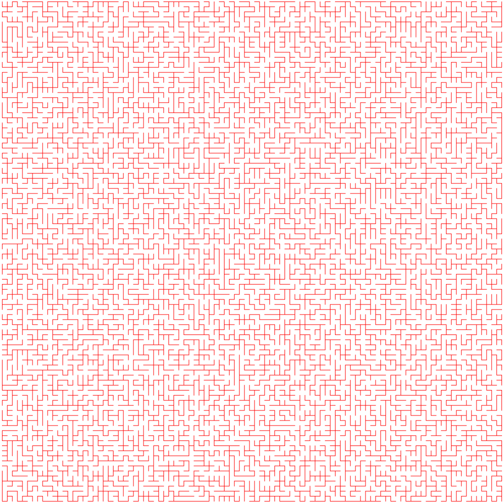
Real-symm' elementary kernel; Sample likelihoods: $\exp(-11,484.5)$



Uniform Spanning Tree in $\mathbb{Z}^2$ ($d = 100$)

```
$ ./uniform_spanning_tree --x_size=100 --y_size=100
```

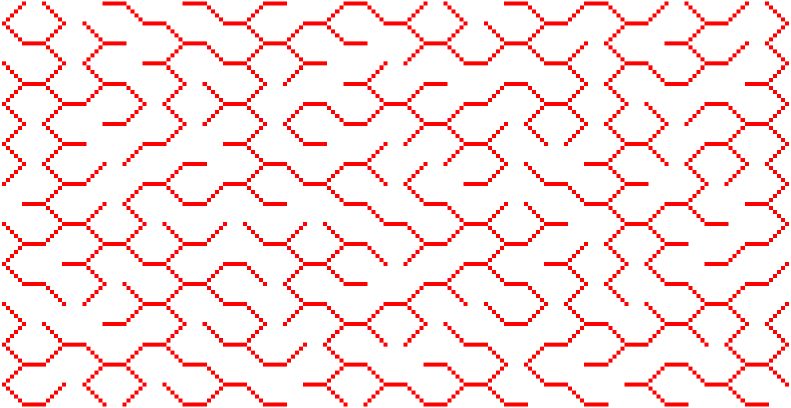
Real-symm' elementary kernel; Sample likelihoods: $\exp(-11,484.5)$



UST for hexagonal tiling of plane ($d = 10$)

```
$ ./uniform_spanning_tree --x_size=10 --y_size=10 --hexagonal=true
```

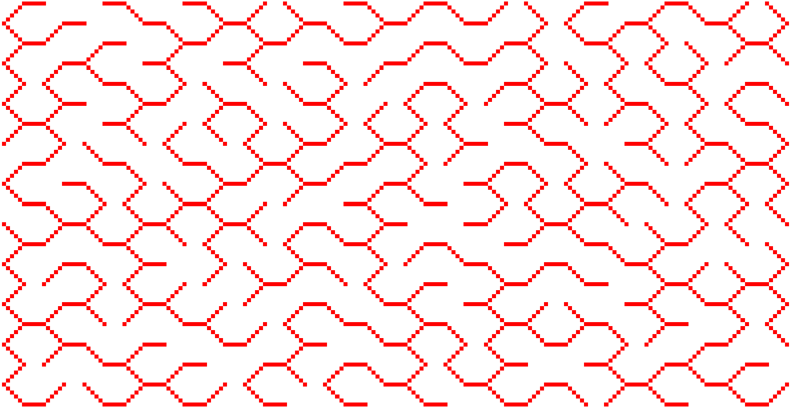
Real-symm' elementary kernel; Sample likelihoods: $\exp(-299.101)$



UST for hexagonal tiling of plane ($d = 10$)

```
$ ./uniform_spanning_tree --x_size=10 --y_size=10 --hexagonal=true
```

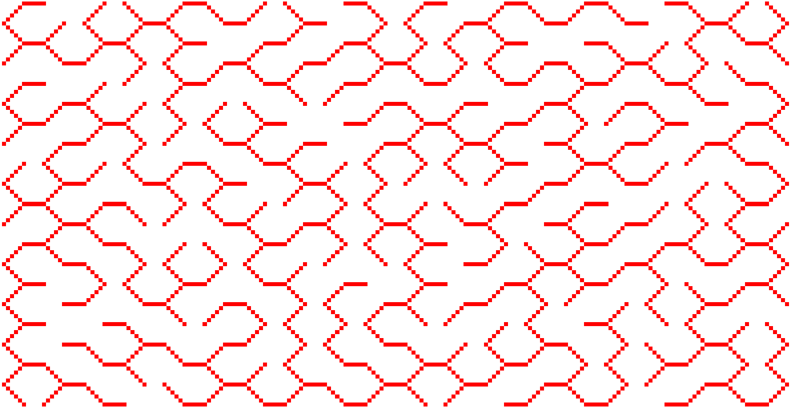
Real-symm' elementary kernel; Sample likelihoods: $\exp(-299.101)$



UST for hexagonal tiling of plane ($d = 10$)

```
$ ./uniform_spanning_tree --x_size=10 --y_size=10 --hexagonal=true
```

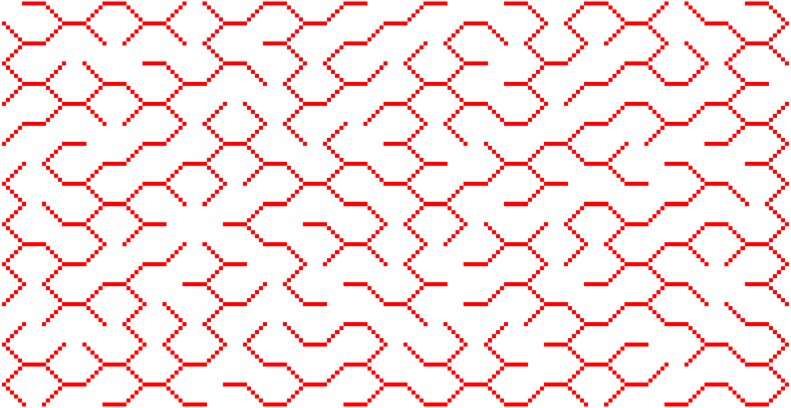
Real-symm' elementary kernel; Sample likelihoods: $\exp(-299.101)$



UST for hexagonal tiling of plane ($d = 10$)

```
$ ./uniform_spanning_tree --x_size=10 --y_size=10 --hexagonal=true
```

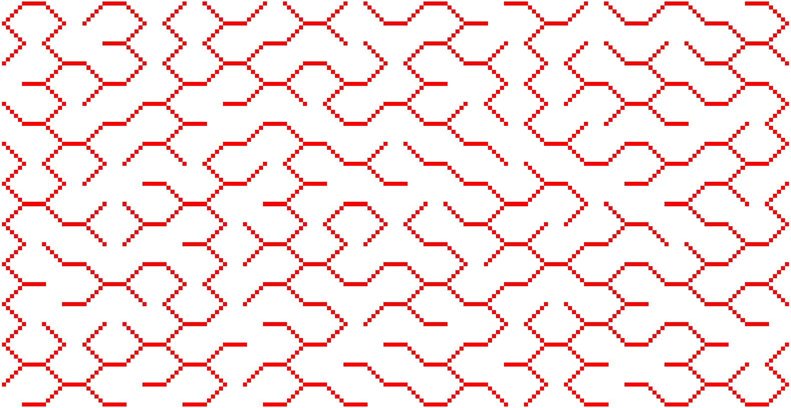
Real-symm' elementary kernel; Sample likelihoods: $\exp(-299.101)$



UST for hexagonal tiling of plane ($d = 10$)

```
$ ./uniform_spanning_tree --x_size=10 --y_size=10 --hexagonal=true
```

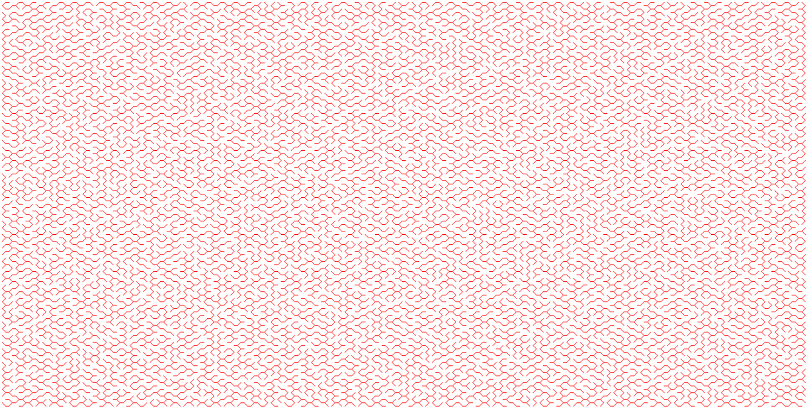
Real-symm' elementary kernel; Sample likelihoods: $\exp(-299.101)$



UST for hexagonal tiling of plane ($d = 60$)

```
$ ./uniform_spanning_tree --x_size=60 --y_size=60 --hexagonal=true
```

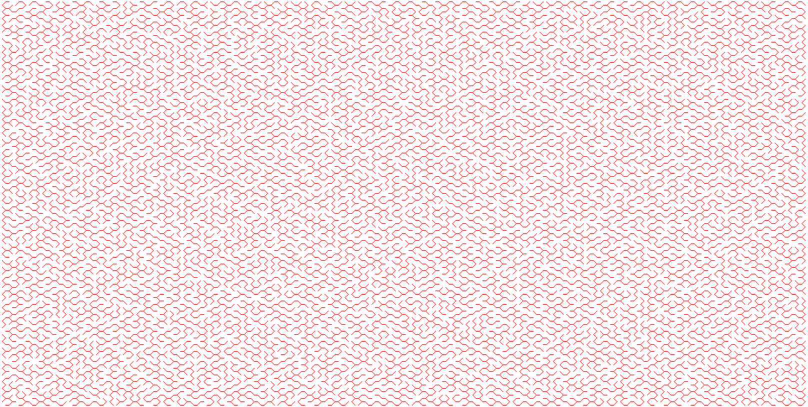
Real-symm' elementary kernel; Sample likelihoods: $\exp(-11,486.8)$



UST for hexagonal tiling of plane ($d = 60$)

```
$ ./uniform_spanning_tree --x_size=60 --y_size=60 --hexagonal=true
```

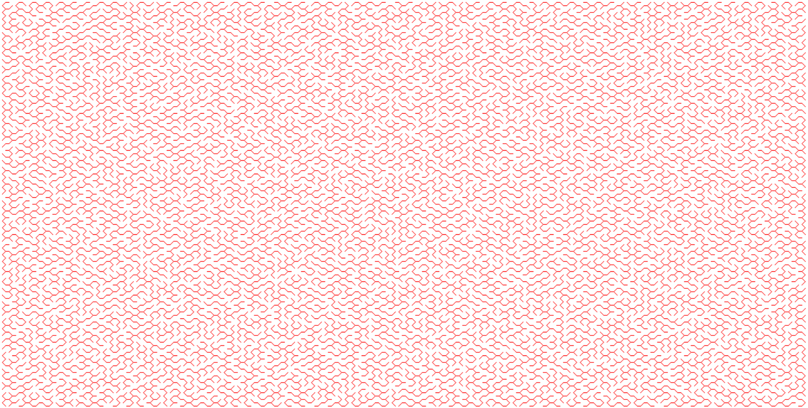
Real-symm' elementary kernel; Sample likelihoods: $\exp(-11,486.8)$



UST for hexagonal tiling of plane ($d = 60$)

```
$ ./uniform_spanning_tree --x_size=60 --y_size=60 --hexagonal=true
```

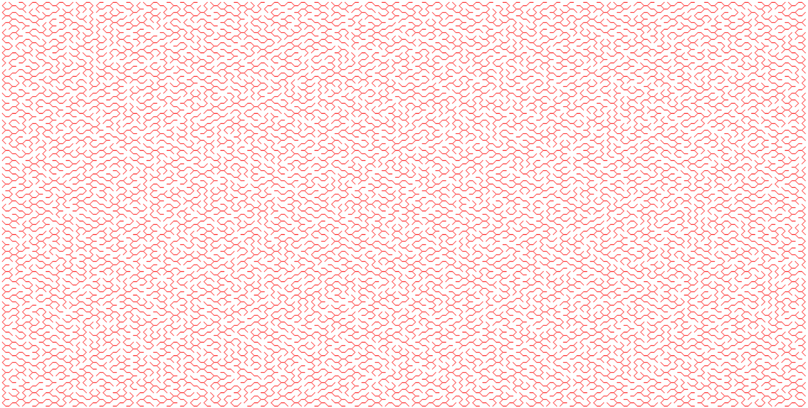
Real-symm' elementary kernel; Sample likelihoods: $\exp(-11,486.8)$



UST for hexagonal tiling of plane ($d = 60$)

```
$ ./uniform_spanning_tree --x_size=60 --y_size=60 --hexagonal=true
```

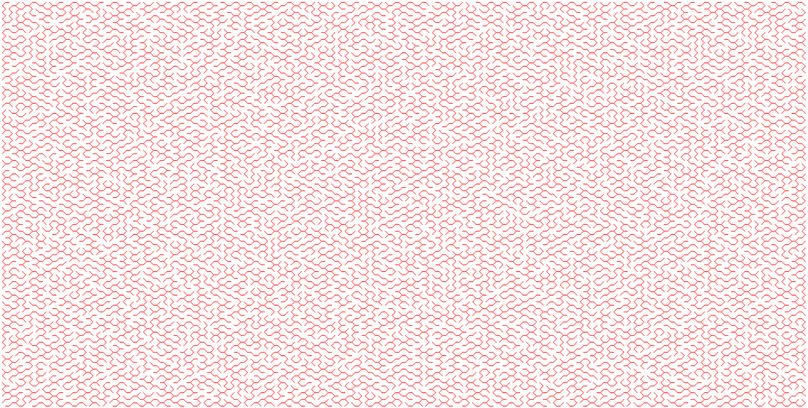
Real-symm' elementary kernel; Sample likelihoods: $\exp(-11,486.8)$



UST for hexagonal tiling of plane ($d = 60$)

```
$ ./uniform_spanning_tree --x_size=60 --y_size=60 --hexagonal=true
```

Real-symm' elementary kernel; Sample likelihoods: $\exp(-11,486.8)$



Hermitian Determinantal Point Processes

Definition 1. A **(Hermitian) marginal kernel matrix** is a (real or complex) Hermitian matrix whose eigenvalues live in $[0, 1]$.

Definition 2. A **(finite, Hermitian) Determinantal Point Process (DPP)** is a random variable $\mathbf{Y}$ over the power set of $\mathcal{Y} = \{0, ..., n-1\} = [n]$ generated by a $n \times n$ (Hermitian) marginal kernel matrix K via the rule

$$\mathbb{P}_K[Y \subseteq \mathbf{Y}] = \det(K_Y),$$

where K_Y is the $|Y| \times |Y|$ submatrix of K formed by restricting to the rows and columns in the index set Y.

Definition 3. A (Hermitian) DPP is called **elementary** if the eigenvalues of its marginal kernel matrix all lie in $\{0, 1\}$.

Hermitian Determinantal Point Processes

Definition 1. A **(Hermitian) marginal kernel matrix** is a (real or complex) Hermitian matrix whose eigenvalues live in $[0, 1]$.

Definition 2. A **(finite, Hermitian) Determinantal Point Process (DPP)** is a random variable $\mathbf{Y}$ over the power set of $\mathcal{Y} = \{0, ..., n-1\} = [n]$ generated by a $n \times n$ (Hermitian) marginal kernel matrix K via the rule

$$\mathbb{P}_K[Y \subseteq \mathbf{Y}] = \det(K_Y),$$

where K_Y is the $|Y| \times |Y|$ submatrix of K formed by restricting to the rows and columns in the index set Y.

Definition 3. A (Hermitian) DPP is called **elementary** if the eigenvalues of its marginal kernel matrix all lie in $\{0, 1\}$.

Hermitian Determinantal Point Processes

Definition 1. A **(Hermitian) marginal kernel matrix** is a (real or complex) Hermitian matrix whose eigenvalues live in $[0, 1]$.

Definition 2. A **(finite, Hermitian) Determinantal Point Process (DPP)** is a random variable $\mathbf{Y}$ over the power set of $\mathcal{Y} = \{0, ..., n-1\} = [n]$ generated by a $n \times n$ (Hermitian) marginal kernel matrix K via the rule

$$\mathbb{P}_K[Y \subseteq \mathbf{Y}] = \det(K_Y),$$

where K_Y is the $|Y| \times |Y|$ submatrix of K formed by restricting to the rows and columns in the index set Y.

Definition 3. A (Hermitian) DPP is called **elementary** if the eigenvalues of its marginal kernel matrix all lie in $\{0, 1\}$.

Non-Hermitian DPP kernels

Definition 4. A (finite) Determinantal Point Process is a random variable $\mathbf{Y}$ over the power set of $\mathcal{Y} = [n]$ generated by an **admissible** $K \in \mathbb{C}^{n \times n}$ that is consistent with the rule:

$$\mathbb{P}_K[Y \subseteq \mathbf{Y}] = \det(K_Y).$$

Proposition 1 (Brunel-2018). A matrix $K \in \mathbb{C}^{n \times n}$ is admissible as a DPP marginal kernel iff

$$(-1)^{|J|}\det(K - I_J) \geq 0, \quad \forall J \subseteq [n].$$

Non-Hermitian DPP kernels

Definition 4. A (finite) Determinantal Point Process is a random variable $\mathbf{Y}$ over the power set of $\mathcal{Y} = [n]$ generated by an **admissible** $K \in \mathbb{C}^{n \times n}$ that is consistent with the rule:

$$\mathbb{P}_K[Y \subseteq \mathbf{Y}] = \det(K_Y).$$

Proposition 1 (Brunel-2018). A matrix $K \in \mathbb{C}^{n \times n}$ is admissible as a DPP marginal kernel iff

$$(-1)^{|J|}\det(K - I_J) \geq 0, \;\; \forall J \subseteq [n].$$

Equivalence classes of DPP kernels

Proposition 2 (P-2019). The equivalence class of a structurally symmetric DPP kernel $K \in \mathbb{C}^{n \times n}$ is its orbit under the group of diagonal similarity transformations, i.e.,

$$\{D^{-1}KD \,:\, D = \mathrm{diag}(d), d \in (\mathbb{C}^{x})^n\}.$$

For complex Hermitian and real symmetric K, the entries of D must respectively lie in $U(1)$ and $O(1)$.

Proposition 3 (P-2019). The equivalence class of a structurally nonsymmetric DPP kernel K strictly contains its orbit under the group of diagonal similarity transformations.

Proof.

If structural symmetry is broken at a 2×2 submatrix, we need only observe that:

$$\mathrm{DPP}(\begin{pmatrix} \alpha & 0 \\ \beta & \gamma \end{pmatrix}) = \mathrm{DPP}(\begin{pmatrix} \alpha & 0 \\ 0 & \gamma \end{pmatrix}),$$

but neither is contained in the orbit of the other. □

Equivalence classes of DPP kernels

Proposition 2 (P-2019). The equivalence class of a structurally symmetric DPP kernel $K \in \mathbb{C}^{n \times n}$ is its orbit under the group of diagonal similarity transformations, i.e.,

$$\{D^{-1}KD \,:\, D = \mathrm{diag}(d), d \in (\mathbb{C}^{x})^{n}\}.$$

For complex Hermitian and real symmetric K, the entries of D must respectively lie in $U(1)$ and $O(1)$.

Proposition 3 (P-2019). The equivalence class of a structurally nonsymmetric DPP kernel K strictly contains its orbit under the group of diagonal similarity transformations.

Proof.

If structural symmetry is broken at a 2×2 submatrix, we need only observe that:

$$\mathrm{DPP}(\begin{pmatrix} \alpha & 0 \\ \beta & \gamma \end{pmatrix}) \equiv \mathrm{DPP}(\begin{pmatrix} \alpha & 0 \\ 0 & \gamma \end{pmatrix}),$$

but neither is contained in the orbit of the other. □

Conditioning and Schur complements

Proposition 4. Given disjoint subsets $A, B \subset \mathcal{Y}$,

$$\mathbb{P}[B \subseteq \mathbf{Y} | A \subseteq \mathbf{Y}] = \det(K_B - K_{B,A} K_A^{-1} K_{A,B}).$$

Proof.

$$\det(K_{A \cup B}) = \det(K_A) \det(K_B - K_{B,A} K_A^{-1} K_{A,B})$$

and

$$\mathbb{P}[B \subseteq \mathbf{Y} | A \subseteq \mathbf{Y}] = \frac{\det(K_{A \cup B})}{\det(K_A)}. \square$$

Conditioning and Schur complements

Proposition 4. Given disjoint subsets $A, B \subset \mathcal{Y}$,

$$\mathbb{P}[B \subseteq \mathbf{Y} | A \subseteq \mathbf{Y}] = \det(K_B - K_{B,A} K_A^{-1} K_{A,B}).$$

Proof.

$$\det(K_{A \cup B}) = \det(K_A) \det(K_B - K_{B,A} K_A^{-1} K_{A,B})$$

and

$$\mathbb{P}[B \subseteq \mathbf{Y} | A \subseteq \mathbf{Y}] = \frac{\det(K_{A \cup B})}{\det(K_A)}. \square$$

Conditioning and Schur complements

Proposition 5. Given disjoint subsets $A, B \subset \mathcal{Y}$,

$$\mathbb{P}[B \subseteq \mathbf{Y} | A \subseteq \mathbf{Y}^c] = \det(K_B - K_{B,A}(K_A - I)^{-1} K_{A,B}).$$

Proof.

The claim follows from repeated application of the case where A is a single element, a:

$$\begin{aligned}
\mathbb{P}[B \subseteq \mathbf{Y} | a \notin \mathbf{Y}] &= \frac{\mathbb{P}[a \notin \mathbf{Y} | B \subseteq \mathbf{Y}]\mathbb{P}[B \subseteq \mathbf{Y}]}{\mathbb{P}[a \notin \mathbf{Y}]} \\
&= \frac{(1 - \mathbb{P}[a \in \mathbf{Y} | B \subseteq \mathbf{Y}])\mathbb{P}[B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\mathbb{P}[B \subseteq \mathbf{Y}] - \mathbb{P}[a \in \mathbf{Y} \wedge B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\det(K_B) - \det(K_{a \cup B})}{1 - K_a} \\
&= \det(K_B)(1 - K_{a,B}\frac{{K_B}^{-1}}{K_a - 1}K_{B,a}) \\
&= \det(K_B - K_{B,a}(K_a - 1)^{-1}K_{a,B}). \quad \square
\end{aligned}$$

Conditioning and Schur complements

Proposition 5. Given disjoint subsets $A, B \subset \mathcal{Y}$,

$$\mathbb{P}[B \subseteq \mathbf{Y} | A \subseteq \mathbf{Y}^c] = \det(K_B - K_{B,A}(K_A - I)^{-1} K_{A,B}).$$

Proof.

The claim follows from repeated application of the case where A is a single element, a:

$$\begin{aligned}
\mathbb{P}[B \subseteq \mathbf{Y} | a \notin \mathbf{Y}] &= \frac{\mathbb{P}[a \notin \mathbf{Y} | B \subseteq \mathbf{Y}] \mathbb{P}[B \subseteq \mathbf{Y}]}{\mathbb{P}[a \notin \mathbf{Y}]} \\
&= \frac{(1 - \mathbb{P}[a \in \mathbf{Y} | B \subseteq \mathbf{Y}]) \mathbb{P}[B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\mathbb{P}[B \subseteq \mathbf{Y}] - \mathbb{P}[a \in \mathbf{Y} \wedge B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\det(K_B) - \det(K_{a \cup B})}{1 - K_a} \\
&= \det(K_B)(1 - K_{a,B} \frac{{K_B}^{-1}}{K_a - 1} K_{B,a}) \\
&= \det(K_B - K_{B,a}(K_a - 1)^{-1} K_{a,B}). \quad \square
\end{aligned}$$

Conditioning and Schur complements

Proposition 5. Given disjoint subsets $A, B \subset \mathcal{Y}$,

$$\mathbb{P}[B \subseteq \mathbf{Y} | A \subseteq \mathbf{Y}^c] = \det(K_B - K_{B,A}(K_A - I)^{-1} K_{A,B}).$$

Proof.
The claim follows from repeated application of the case where A is a single element, a:

$$\begin{aligned}
\mathbb{P}[B \subseteq \mathbf{Y} | a \notin \mathbf{Y}] &= \frac{\mathbb{P}[a \notin \mathbf{Y} | B \subseteq \mathbf{Y}]\mathbb{P}[B \subseteq \mathbf{Y}]}{\mathbb{P}[a \notin \mathbf{Y}]} \\
&= \frac{(1 - \mathbb{P}[a \in \mathbf{Y} | B \subseteq \mathbf{Y}])\mathbb{P}[B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\mathbb{P}[B \subseteq \mathbf{Y}] - \mathbb{P}[a \in \mathbf{Y} \wedge B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\det(K_B) - \det(K_{a \cup B})}{1 - K_a} \\
&= \det(K_B)(1 - K_{a,B} \frac{{K_B}^{-1}}{K_a - 1} K_{B,a}) \\
&= \det(K_B - K_{B,a}(K_a - 1)^{-1} K_{a,B}). \quad \square
\end{aligned}$$

Conditioning and Schur complements

Proposition 5. Given disjoint subsets $A, B \subset \mathcal{Y}$,

$$\mathbb{P}[B \subseteq \mathbf{Y} | A \subseteq \mathbf{Y}^c] = \det(K_B - K_{B,A}(K_A - I)^{-1} K_{A,B}).$$

Proof.

The claim follows from repeated application of the case where A is a single element, a:

$$\begin{aligned}
\mathbb{P}[B \subseteq \mathbf{Y} | a \notin \mathbf{Y}] &= \frac{\mathbb{P}[a \notin \mathbf{Y} | B \subseteq \mathbf{Y}]\mathbb{P}[B \subseteq \mathbf{Y}]}{\mathbb{P}[a \notin \mathbf{Y}]} \\
&= \frac{(1 - \mathbb{P}[a \in \mathbf{Y} | B \subseteq \mathbf{Y}])\mathbb{P}[B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\mathbb{P}[B \subseteq \mathbf{Y}] - \mathbb{P}[a \in \mathbf{Y} \wedge B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\det(K_B) - \det(K_{a \cup B})}{1 - K_a} \\
&= \det(K_B)(1 - K_{a,B} \frac{K_B{}^{-1}}{K_a - 1} K_{B,a}) \\
&= \det(K_B - K_{B,a}(K_a - 1)^{-1} K_{a,B}). \quad \square
\end{aligned}$$

Conditioning and Schur complements

Proposition 5. Given disjoint subsets $A, B \subset \mathcal{Y}$,

$$\mathbb{P}[B \subseteq \mathbf{Y} | A \subseteq \mathbf{Y}^c] = \det(K_B - K_{B,A}(K_A - I)^{-1} K_{A,B}).$$

Proof.

The claim follows from repeated application of the case where A is a single element, a:

$$\begin{aligned}
\mathbb{P}[B \subseteq \mathbf{Y} | a \notin \mathbf{Y}] &= \frac{\mathbb{P}[a \notin \mathbf{Y} | B \subseteq \mathbf{Y}]\mathbb{P}[B \subseteq \mathbf{Y}]}{\mathbb{P}[a \notin \mathbf{Y}]} \\
&= \frac{(1 - \mathbb{P}[a \in \mathbf{Y} | B \subseteq \mathbf{Y}])\mathbb{P}[B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\mathbb{P}[B \subseteq \mathbf{Y}] - \mathbb{P}[a \in \mathbf{Y} \wedge B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\det(K_B) - \det(K_{a \cup B})}{1 - K_a} \\
&= \det(K_B)(1 - K_{a,B} \frac{K_B{}^{-1}}{K_a - 1} K_{B,a}) \\
&= \det(K_B - K_{B,a}(K_a - 1)^{-1} K_{a,B}). \quad \square
\end{aligned}$$

Conditioning and Schur complements

Proposition 5. Given disjoint subsets $A, B \subset \mathcal{Y}$,

$$\mathbb{P}[B \subseteq \mathbf{Y} | A \subseteq \mathbf{Y}^c] = \det(K_B - K_{B,A}(K_A - I)^{-1} K_{A,B}).$$

Proof.

The claim follows from repeated application of the case where A is a single element, a:

$$\begin{aligned}
\mathbb{P}[B \subseteq \mathbf{Y} | a \notin \mathbf{Y}] &= \frac{\mathbb{P}[a \notin \mathbf{Y} | B \subseteq \mathbf{Y}] \mathbb{P}[B \subseteq \mathbf{Y}]}{\mathbb{P}[a \notin \mathbf{Y}]} \\
&= \frac{(1 - \mathbb{P}[a \in \mathbf{Y} | B \subseteq \mathbf{Y}]) \mathbb{P}[B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\mathbb{P}[B \subseteq \mathbf{Y}] - \mathbb{P}[a \in \mathbf{Y} \wedge B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\det(K_B) - \det(K_{a \cup B})}{1 - K_a} \\
&= \det(K_B)(1 - K_{a,B} \frac{{K_B}^{-1}}{K_a - 1} K_{B,a}) \\
&= \det(K_B - K_{B,a}(K_a - 1)^{-1} K_{a,B}). \quad \square
\end{aligned}$$

Conditioning and Schur complements

Proposition 5. Given disjoint subsets $A, B \subset \mathcal{Y}$,

$$\mathbb{P}[B \subseteq \mathbf{Y} | A \subseteq \mathbf{Y}^c] = \det(K_B - K_{B,A}(K_A - I)^{-1} K_{A,B}).$$

Proof.
The claim follows from repeated application of the case where A is a single element, a:

$$\begin{aligned}
\mathbb{P}[B \subseteq \mathbf{Y} | a \notin \mathbf{Y}] &= \frac{\mathbb{P}[a \notin \mathbf{Y} | B \subseteq \mathbf{Y}]\mathbb{P}[B \subseteq \mathbf{Y}]}{\mathbb{P}[a \notin \mathbf{Y}]} \\
&= \frac{(1 - \mathbb{P}[a \in \mathbf{Y} | B \subseteq \mathbf{Y}])\mathbb{P}[B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\mathbb{P}[B \subseteq \mathbf{Y}] - \mathbb{P}[a \in \mathbf{Y} \wedge B \subseteq \mathbf{Y}]}{1 - \mathbb{P}[a \in \mathbf{Y}]} \\
&= \frac{\det(K_B) - \det(K_{a \cup B})}{1 - K_a} \\
&= \det(K_B)(1 - K_{a,B} \frac{K_B{}^{-1}}{K_a - 1} K_{B,a}) \\
&= \det(K_B - K_{B,a}(K_a - 1)^{-1} K_{a,B}). \quad \square
\end{aligned}$$

Traditional Hermitian DPP sampling

Lemma 5 (Hough et al.-2006). Given any $\mathbf{Y} \sim \mathrm{DPP}(K)$, where K has spectral decomposition $Q \Lambda Q^*$, sampling from $\mathbf{Y}$ is equivalent to sampling from the random elementary DPP with kernel $P(Q_{\mathbf{Z}})$, where $P(U) \equiv UU^*$ and $Q_{\mathbf{Z}}$ consists of the columns of Q with indices from $\mathbf{Z} \sim \mathrm{DPP}(\Lambda)$.

"Alg. 1 runs in time $O(Nk^3)$, where k is the number of eigenvectors selected [...] the initial eigendecomposition of $[K]$ is often the computational bottleneck, requiring $O(N^3)$ time. Modern multi-core machines can compute eigendecompositions up to $N \approx 1,000$ at interactive speeds of a few seconds, or larger problems up to $N \approx 10,000$ in around ten minutes."
[Kulesza/Taskar-2012]

[Gillenwater-2014] reduced the factored elementary DPP sampling down to cubic complexity via what is equivalent to diagonally-pivoted Cholesky.[1]

[1] [Gillenwater-2014] Approximate inference for determinantal point processes

Traditional Hermitian DPP sampling

Lemma 5 (Hough et al.-2006). Given any $\mathbf{Y} \sim \text{DPP}(K)$, where K has spectral decomposition $Q \Lambda Q^*$, sampling from $\mathbf{Y}$ is equivalent to sampling from the random elementary DPP with kernel $P(Q_{\mathbf{Z}})$, where $P(U) \equiv UU^*$ and $Q_{\mathbf{Z}}$ consists of the columns of Q with indices from $\mathbf{Z} \sim \text{DPP}(\Lambda)$.

"Alg. 1 runs in time $O(Nk^3)$, where k is the number of eigenvectors selected [...] the initial eigendecomposition of $[K]$ is often the computational bottleneck, requiring $O(N^3)$ time. Modern multi-core machines can compute eigendecompositions up to $N \approx 1,000$ at interactive speeds of a few seconds, or larger problems up to $N \approx 10,000$ in around ten minutes."
[Kulesza/Taskar-2012]

[Gillenwater-2014] reduced the factored elementary DPP sampling down to cubic complexity via what is equivalent to diagonally-pivoted Cholesky.[1]

[1] [Gillenwater-2014] Approximate inference for determinantal point processes

Traditional Hermitian DPP sampling

Lemma 5 (Hough et al.-2006). Given any $\mathbf{Y} \sim \mathrm{DPP}(K)$, where K has spectral decomposition $Q\Lambda Q^*$, sampling from $\mathbf{Y}$ is equivalent to sampling from the random elementary DPP with kernel $P(Q_{\mathbf{Z}})$, where $P(U) \equiv UU^*$ and $Q_{\mathbf{Z}}$ consists of the columns of Q with indices from $\mathbf{Z} \sim \mathrm{DPP}(\Lambda)$.

"Alg. 1 runs in time $O(Nk^3)$, where k is the number of eigenvectors selected [...] the initial eigendecomposition of $[K]$ is often the computational bottleneck, requiring $O(N^3)$ time. Modern multi-core machines can compute eigendecompositions up to $N \approx 1,000$ at interactive speeds of a few seconds, or larger problems up to $N \approx 10,000$ in around ten minutes."
[Kulesza/Taskar-2012]

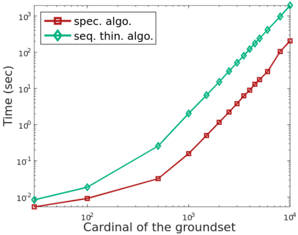
[Gillenwater-2014] reduced the factored elementary DPP sampling down to cubic complexity via what is equivalent to diagonally-pivoted Cholesky.[1]

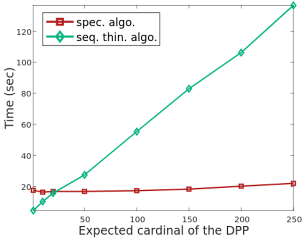
[1] [Gillenwater-2014] Approximate inference for determinantal point processes

Traditional Hermitian DPP sampling

Recently, authors are noticing connections to LDL^H factorizations.[2][3]

In [Launay et al.-2018], timings are provided for the spectrally-preprocessed and "sequentially thinned" algorithm for elementary real symmetric kernels of rank 20 and varying size (left) and varying rank and size 5000 (right):





This talk decreases runtimes by 100-1000x, for more general kernels, by importing dense factorization techniques. We then extend to non-Hermitian and sparse-direct analogues.

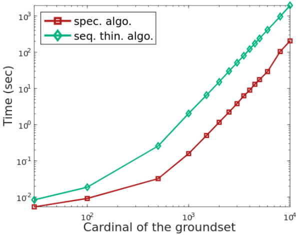
[2] [Chen et al.-2017] Fast Greedy MAP inference for Det' Point Processes

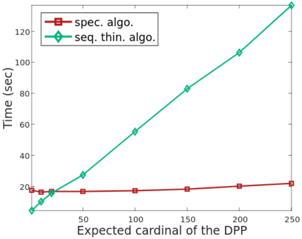
[3] [Launay et al.-2018] Exact sampling of determinantal point processes without eigendecomposition. arxiv.org/abs/1802.08429v3

Traditional Hermitian DPP sampling

Recently, authors are noticing connections to LDL^H factorizations.[2][3]

In [Launay et al.-2018], timings are provided for the spectrally-preprocessed and "sequentially thinned" algorithm for elementary real symmetric kernels of rank 20 and varying size (left) and varying rank and size 5000 (right):





This talk decreases runtimes by 100-1000x, for more general kernels, by importing dense factorization techniques. We then extend to non-Hermitian and sparse-direct analogues.

[2] [Chen et al.-2017] Fast Greedy MAP inference for Det' Point Processes

[3] [Launay et al.-2018] Exact sampling of determinantal point processes without eigendecomposition. arxiv.org/abs/1802.08429v3

Unblocked DPP sampling factorization

```
sample = []
for j in range(n):
  J2 = [j+1:n]
  if Bernoulli(K(j,j)):
    sample.append(j)
  else:
    K(j,j) -= 1
  K(J2,j) /= K(j,j)
  K(J2,J2) -= K(J2,j) * K(j,J2)
return sample
```

This is a small tweak of unblocked, unpivoted LU factorization – readily specializable to LDL^H and LDL^T for Hermitian and complex symmetric matrices.

The majority of the work is in rank-1 updates. And the standard optimizations apply (e.g., blocking and sparse-direct factorization)!

The likelihood of the sample is equal to the product of the absolute value of the diagonal of the result.

Unblocked DPP sampling factorization

```
sample = []
for j in range(n):
  J2 = [j+1:n]
  if Bernoulli(K(j,j)):
    sample.append(j)
  else:
    K(j,j) -= 1
  K(J2,j) /= K(j,j)
  K(J2,J2) -= K(J2,j) * K(j,J2)
return sample
```

This is a small tweak of unblocked, unpivoted LU factorization – readily specializable to LDL^H and LDL^T for Hermitian and complex symmetric matrices.

The majority of the work is in rank-1 updates. And the standard optimizations apply (e.g., blocking and sparse-direct factorization)!

The likelihood of the sample is equal to the product of the absolute value of the diagonal of the result.

Unblocked DPP sampling factorization

```
sample = []
for j in range(n):
  J2 = [j+1:n]
  if Bernoulli(K(j,j)):
    sample.append(j)
  else:
    K(j,j) -= 1
  K(J2,j) /= K(j,j)
  K(J2,J2) -= K(J2,j) * K(j,J2)
return sample
```

This is a small tweak of unblocked, unpivoted LU factorization – readily specializable to LDL^H and LDL^T for Hermitian and complex symmetric matrices.

The majority of the work is in rank-1 updates. And the standard optimizations apply (e.g., blocking and sparse-direct factorization)!

The likelihood of the sample is equal to the product of the absolute value of the diagonal of the result.

Unblocked DPP sampling factorization

```
sample = []
for j in range(n):
  J2 = [j+1:n]
  if Bernoulli(K(j,j)):
    sample.append(j)
  else:
    K(j,j) -= 1
  K(J2,j) /= K(j,j)
  K(J2,J2) -= K(J2,j) * K(j,J2)
return sample
```

This is a small tweak of unblocked, unpivoted LU factorization – readily specializable to LDL^H and LDL^T for Hermitian and complex symmetric matrices.

The majority of the work is in rank-1 updates. And the standard optimizations apply (e.g., blocking and sparse-direct factorization)!

The likelihood of the sample is equal to the product of the absolute value of the diagonal of the result.

Blocked DPP sampling factorization

```
sample = []
J1_beg = 0
while J1_beg < n:
  J1_end = min(n, J1_beg+blocksize)
  J1 = [J1_beg:J1_end]; J2 = [J1_end:n]
  subsample, K(J1,J1) = unblocked_dpp(K(J1,J1))
  sample.append(subsample + J1_beg)
  K(J2,J1) /= triu(K(J1,J1))
  K(J1,J2) \= unit_tril(K(J1,J1))
  K(J2,J2) -= K(J2,J1) * K(J1,J2)
  J1_beg = J1_end
return sample
```

OpenMP 4.0 tasks – say, with tile sizes of 128 – can be readily used to provide shared-memory, DAG-scheduled parallelism [Agullo/Langou/Luszczek-2010, Yarkhan et al.-2011, Chan et al.-2007].

Blocked DPP sampling factorization

```
sample = []
J1_beg = 0
while J1_beg < n:
  J1_end = min(n, J1_beg+blocksize)
  J1 = [J1_beg:J1_end]; J2 = [J1_end:n]
  subsample, K(J1,J1) = unblocked_dpp(K(J1,J1))
  sample.append(subsample + J1_beg)
  K(J2,J1) /= triu(K(J1,J1))
  K(J1,J2) \= unit_tril(K(J1,J1))
  K(J2,J2) -= K(J2,J1) * K(J1,J2)
  J1_beg = J1_end
return sample
```

OpenMP 4.0 tasks – say, with tile sizes of 128 – can be readily used to provide shared-memory, DAG-scheduled parallelism [Agullo/Langou/Luszczek-2010, Yarkhan et al.-2011, Chan et al.-2007].

Unblocked, greedy, MAP DPP sampling

```
sample = []
for j in range(n):
  J2 = [j+1:n]
  if K(j,j) >= 0.5:
    sample.append(j)
  else:
    K(j,j) -= 1
  K(J2,j) /= K(j,j)
  K(J2,J2) -= K(J2,j) * K(j,J2)
return sample
```

Greedy MAP sampling is a trivial tweak of the standard sampler, and the blocked extension is essentially identical.

Unblocked, greedy, MAP DPP sampling

```
sample = []
for j in range(n):
  J2 = [j+1:n]
  if K(j,j) >= 0.5:
    sample.append(j)
  else:
    K(j,j) -= 1
  K(J2,j) /= K(j,j)
  K(J2,J2) -= K(J2,j) * K(j,J2)
return sample
```

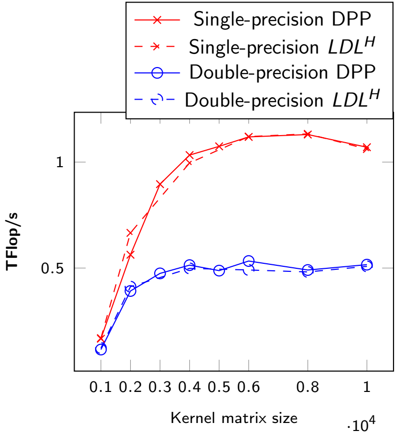
Greedy MAP sampling is a trivial tweak of the standard sampler, and the blocked extension is essentially identical.

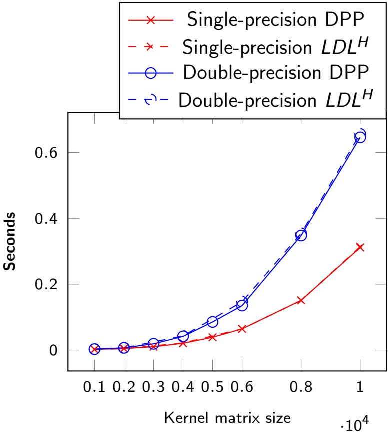
Full-rank real symmetric DPP on i9-7960x (16-core)

Dense, real LDL^H-based DPP sampler [P-2019].

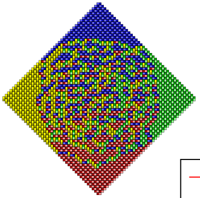
For comparison, stock DPPy v0.1.0 [Gautier-2019] takes 250 seconds for each sample when $n = 5000$.

```
$ OMP_NUM_THREADS=16 ./dense_dpp
```

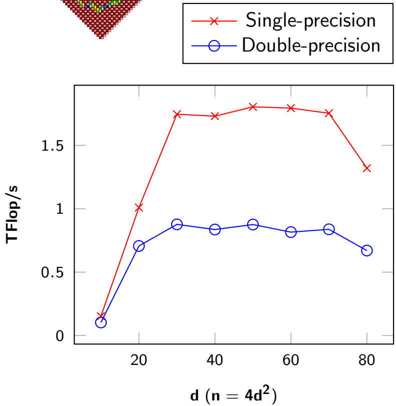


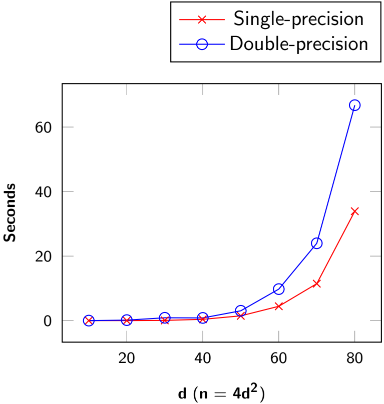


Aztec diamond DPP on i9-7960x (16-core)



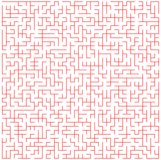
Dense, complex LU-based DPP sampler [P-2019].*



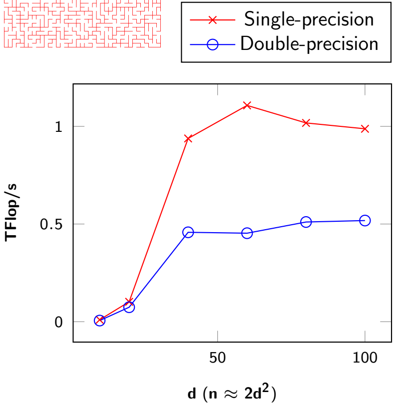


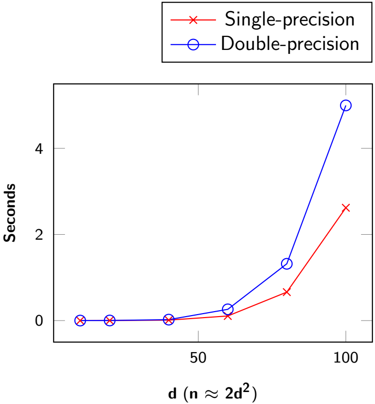
*Generated from the Kenyon formula over the Kasteleyn matrix [Chhita et al.-2015].

$\mathbb{Z}^2$ UST DPP on i9-7960x (16-core)



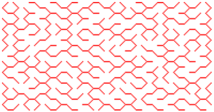
Dense, real LDL^H-based DPP sampler [P-2019].*



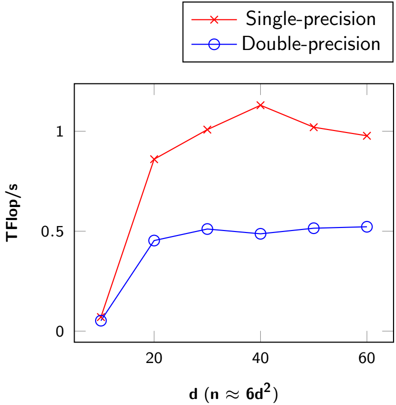


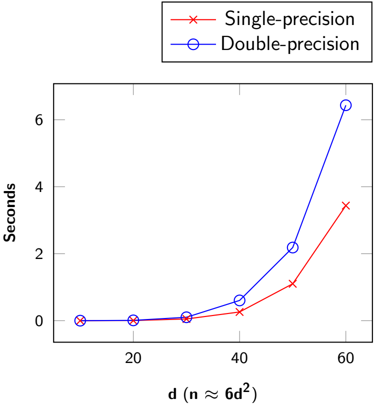
*Over the Gramian of the star space basis [Lyons/Peres-2016].

Hexagonal UST DPP on i9-7960x (16-core)



Dense, real LDL^H-based DPP sampler [P-2019].*

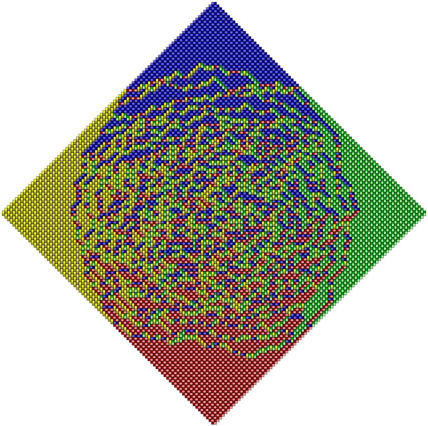




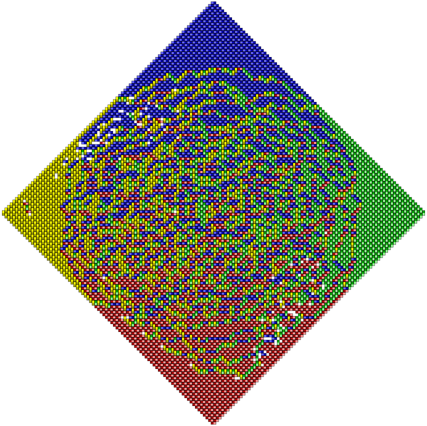
*Over the Gramian of the star space basis [Lyons/Peres-2016].

Low precision corrupting sampling

```
$ ./aztec_diamond --diamond_size=80
```



Double-precision sample



Single-precision sample (visibly erroneous)

Basic questions for DPP factorizations

Given the close connection between DPP sampling and dense factorization:

- One should be able to probabilistically generalize element growth and numerical stability bounds.
- Use maximum-entropy diagonal pivot selection? Minimizes worst case pivot.
- High-performance techniques for backpropagating through Cholesky are now known [Murray-2016].[4] Do these blocked algorithms extend to DPPs?

[4] [Murray-2016] Differentiation of the Cholesky decomposition. arxiv.org/abs/1602.07527

Basic questions for DPP factorizations

Given the close connection between DPP sampling and dense factorization:

- One should be able to probabilistically generalize element growth and numerical stability bounds.
- Use maximum-entropy diagonal pivot selection? Minimizes worst case pivot.
- High-performance techniques for backpropagating through Cholesky are now known [Murray-2016].[4] Do these blocked algorithms extend to DPPs?

[4] [Murray-2016] Differentiation of the Cholesky decomposition. arxiv.org/abs/1602.07527

Basic questions for DPP factorizations

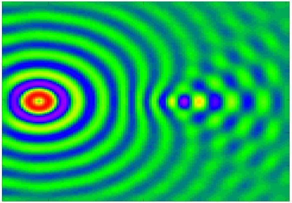
Given the close connection between DPP sampling and dense factorization:

- One should be able to probabilistically generalize element growth and numerical stability bounds.
- Use maximum-entropy diagonal pivot selection? Minimizes worst case pivot.
- High-performance techniques for backpropagating through Cholesky are now known [Murray-2016].[4] Do these blocked algorithms extend to DPPs?

[4] [Murray-2016] Differentiation of the Cholesky decomposition. arxiv.org/abs/1602.07527

Sparse-direct DPP factorizations

We have so-far discussed analogues of **dense** factorizations, and **sparse-direct** analogues are a natural extension.

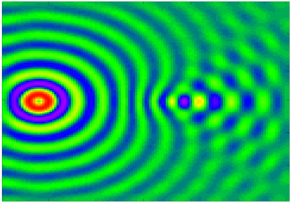


Catamari implements templated, real and complex, Cholesky / LDL^H / LDL^T – switching between DAG-scheduled, **right-looking supernodal** and **up-looking simplicial** based upon arithmetic intensity [Chen/Davis/Hager/Rajamanickam-2008].

A variant of the sparse-direct LDL^H is provided for sparse DPPs. (Unpivoted) sparse-direct LU and LDL^T DPP sampling is a straight-forward extension.

Sparse-direct DPP factorizations

We have so-far discussed analogues of **dense** factorizations, and **sparse-direct** analogues are a natural extension.

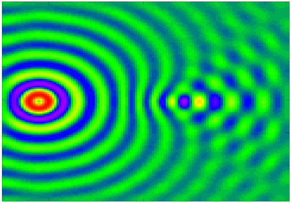


Catamari implements templated, real and complex, Cholesky / LDL^H / LDL^T – switching between DAG-scheduled, **right-looking supernodal** and **up-looking simplicial** based upon arithmetic intensity [Chen/Davis/Hager/Rajamanickam-2008].

A variant of the sparse-direct LDL^H is provided for sparse DPPs. (Unpivoted) sparse-direct LU and LDL^T DPP sampling is a straight-forward extension.

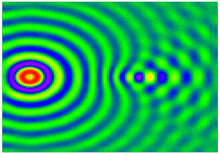
Sparse-direct DPP factorizations

We have so-far discussed analogues of **dense** factorizations, and **sparse-direct** analogues are a natural extension.



Catamari implements templated, real and complex, Cholesky / LDL^H / LDL^T – switching between DAG-scheduled, **right-looking supernodal** and **up-looking simplicial** based upon arithmetic intensity [Chen/Davis/Hager/Rajamanickam-2008].

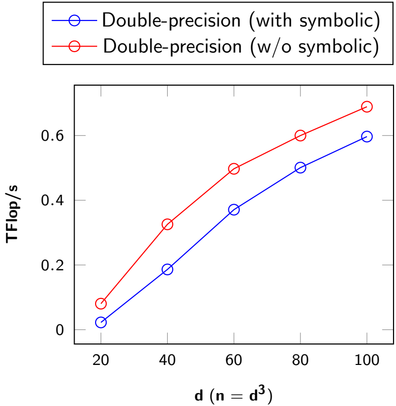
A variant of the sparse-direct LDL^H is provided for sparse DPPs. (Unpivoted) sparse-direct LU and LDL^T DPP sampling is a straight-forward extension.

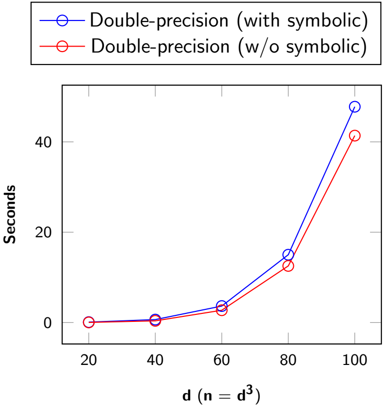


Complex sparse LDL^T on i9-7960x (16-core)

3D Helmholtz w/ PML and trilinear, hexahedral elements

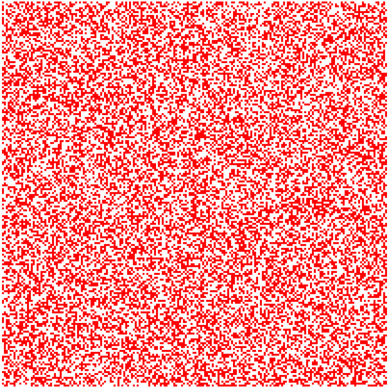
```
$ OMP_NUM_THREADS=16 ./helmholtz_3d_pml
```



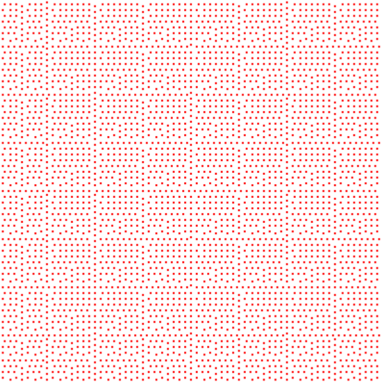


(MAP) Sampling from 2D $-\sigma\Delta$

```
$ ./dpp_shifted_2d_negative_laplacian \
    --x_size=200 --y_size=200 --scale=0.72
```



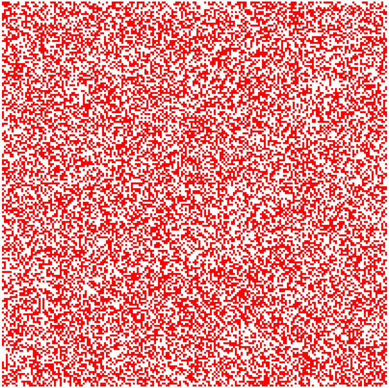
Log-likelihood: -27472.2
Sample time: 0.0107 seconds



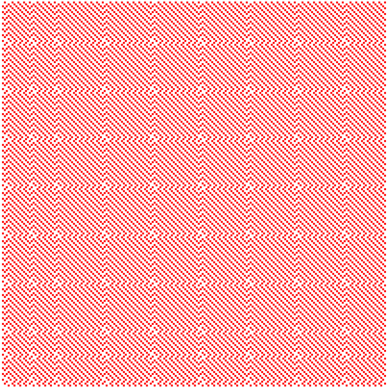
Log-likelihood: -26058
Sample time: 0.0112 seconds

(MAP) Sampling from 2D $-\sigma\Delta$

```
$ ./dpp_shifted_2d_negative_laplacian \
    --x_size=200 --y_size=200 --scale=0.75
```



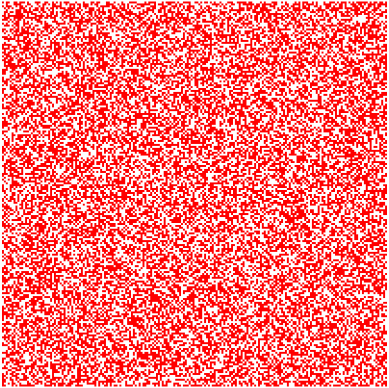
Log-likelihood: -27612.6
Sample time: 0.0124 seconds



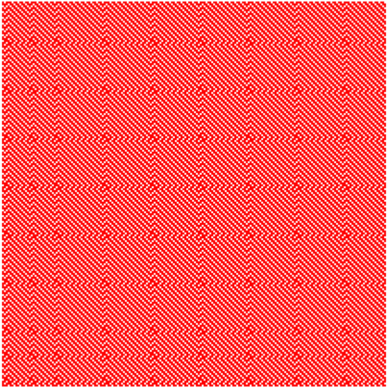
Log-likelihood: -26009
Sample time: 0.0114 seconds

(MAP) Sampling from 2D $-\sigma\Delta$

```
$ ./dpp_shifted_2d_negative_laplacian \
    --x_size=200 --y_size=200 --scale=0.85
```



Log-likelihood: -27581.7
Sample time: 0.0114 seconds



Log-likelihood: -25765
Sample time: 0.0118 seconds

Closing

Availability:
Quotient is available under the Mozilla Public License 2.0 at
hodgestar.com/quotient/ and gitlab.com/hodge_star/quotient.
This talk is based on version 0.2.

Catamari is available under the Mozilla Public License 2.0 at
hodgestar.com/catamari/ and gitlab.com/hodge_star/catamari.
This talk is based on version 0.2.3.

These slides are available at:
hodgestar.com/catamari/April8-2019-RoyalSociety.pdf

Acknowledgements:

- Alex Kulesza and Jenny Gillenwater:
  For answering my initial DPP sampling questions.

Questions/comments?