# Uniform quantizer, vector quantization, Discrete cosine transform, Predictive coding

Coding Technology

Illés Horváth

2025/11/21

# Outline

1. Quantization and entropy
2. Uniform quantizer
3. Vector quantization
   - Lloyd-Max algorithm in higher dimension
   - Discrete cosine transform
   - Predictive coding

# Reminder: quantization

Irreversible data compression. Distortion vs compression ratio.

Quantization: rounding values from a large set to a small set. Quantizer, levels, domains.

Nearest neighbour rule, Center of gravity rule, Lloyd-Max property.

Optimal quantizer. Lloyd-Max algorithm.

# Quantization and entropy

For an $X$ discrete random variable, the entropy of $X$ is

$$H(X) = -\sum_k P(X = k) \log_2(P(X = k)).$$

# Quantization and entropy

For an $X$ discrete random variable, the entropy of $X$ is

$$H(X) = -\sum_k P(X = k) \log_2(P(X = k)).$$

**Theorem**

*For any function $Q$,*

$$H(Q(X)) \leq H(X).$$

# Quantization and entropy

For an $X$ discrete random variable, the entropy of $X$ is

$$H(X) = -\sum_k P(X = k) \log_2(P(X = k)).$$

**Theorem**

*For any function $Q$,*

$$H(Q(X)) \leq H(X).$$

(No proof.)

# Quantization and entropy

For an $X$ discrete random variable, the entropy of $X$ is

$$H(X) = -\sum_k P(X = k) \log_2(P(X = k)).$$

**Theorem**

*For any function $Q$,*

$$H(Q(X)) \leq H(X).$$

(No proof.)

Basically, by quantization, we lose information.

# Quantization and entropy

For an $X$ discrete random variable, the entropy of $X$ is

$$H(X) = -\sum_k P(X = k) \log_2(P(X = k)).$$

**Theorem**

*For any function $Q$,*

$$H(Q(X)) \leq H(X).$$

(No proof.)

Basically, by quantization, we lose information.

Note that the theorem only applies for $X$ discrete.

# Uniform quantizer

Let $X$ be a continuous random variable with probability density function $f(x)$.

Assume $f(x)$ is continuous on $[0, A]$ and 0 outside it (so $X$ is concentrated on $[0, A]$).

The uniform $N$-level quantizer $Q_N$ for the interval $[0, A]$ is the following:

$$\text{domains: } B_i = \left[(i-1)\frac{A}{N}, i\frac{A}{N}\right], \quad i = 1, \ldots, N,$$

$$\text{levels: } x_i = (2i-1)\frac{A}{2N}, \quad i = 1, \ldots, N.$$

The uniform quantizer is a simple quantizer (but not optimal except if $X$ has uniform distribution on $[0, A]$).

# Uniform quantizer

## Theorem

*For the distortion of $Q_N$, we have*

$$\lim_{N\to\infty} N^2 D(Q_N) = \frac{A^2}{12}$$

# Uniform quantizer

## Theorem

*For the distortion of $Q_N$, we have*

$$\lim_{N\to\infty} N^2 D(Q_N) = \frac{A^2}{12}$$

Essentially, the theorem says that for large $N$,

$$D(Q_N) \approx \frac{A^2}{12N^2}$$

# Uniform quantizer

## Theorem

*For the distortion of $Q_N$, we have*

$$\lim_{N\to\infty} N^2 D(Q_N) = \frac{A^2}{12}$$

Essentially, the theorem says that for large $N$,

$$D(Q_N) \approx \frac{A^2}{12N^2}$$

Proof (sketch). For large $N$, the conditional distribution of $X$ assuming $X \in B_i$ is close to the uniform distribution on $B_i$ since $f$ is continuous. The uniform distribution on an interval of length $L$ has variance $L^2/12$, so

$$D(Q_N) \approx \sum_{i=1}^{N} \int_{B_i} f(x) \frac{(A/N)^2}{12} \mathrm{d}x = \frac{A^2}{12N^2} \int_{-A}^{A} f(x) \mathrm{d}x = \frac{A^2}{12N^2}$$

# Differential entropy

If $X$ is a continuous random variable with probability density function $f(x)$, its *differential entropy* is

$$H(f) = -\int_{\mathbb{R}} f(x) \log_2(x) \mathrm{d}x$$

# Differential entropy

If $X$ is a continuous random variable with probability density function $f(x)$, its *differential entropy* is

$$H(f) = -\int_{\mathbb{R}} f(x)\log_2(x)\mathrm{d}x$$

Differential entropy is similar to entropy, but it lacks some of the nice properties of entropy. For example, it can be negative.

# Differential entropy

If $X$ is a continuous random variable with probability density function $f(x)$, its *differential entropy* is

$$H(f) = -\int_{\mathbb{R}} f(x)\log_2(x)\mathrm{d}x$$

Differential entropy is similar to entropy, but it lacks some of the nice properties of entropy. For example, it can be negative.

## Theorem

$$\lim_{N\to\infty}(H(Q_N(X)) + \log_2(A/N)) = H(f)$$

(No proof.)

# Differential entropy

The previous theorem essentially says that

$$H(Q_N(X)) \approx H(f) - \log_2(A) + \log_2 N$$

# Differential entropy

The previous theorem essentially says that

$$H(Q_N(X)) \approx H(f) - \log_2(A) + \log_2 N$$

Example. Consider the uniform distribution on $[0, 1]$. For the uniform quantization with levels $N = 2^k$, we basically get the value of $X$ with $k$ bits precision. The bits are independent since the distribution is uniform.

# Differential entropy

The previous theorem essentially says that

$$H(Q_N(X)) \approx H(f) - \log_2(A) + \log_2 N$$

Example. Consider the uniform distribution on $[0, 1]$. For the uniform quantization with levels $N = 2^k$, we basically get the value of $X$ with $k$ bits precision. The bits are independent since the distribution is uniform.

Accordingly, the above formula gives

$$\begin{aligned} H(f) &= \int_{-1/2}^{1/2} 1 \cdot 0 \mathrm{d}x = 0 \\ \log_2(A) &= \log_2(1) = 0 \\ \log_2(N) &= k \end{aligned}$$

and $H(Q_N(X)) = k$, which is indeed the entropy of $k$ bits.

# Lloyd-Max algorithm

Reminder: Lloyd-Max algorithm.

1. Initialize the levels.
2. Update the domains using the nearest neighbour property.
3. Compute $D(Q)$, if the decrease is below a given threshold, stop.
4. Update the levels using the center of gravity rule, repeat from step 2.

# Lloyd-Max algorithm

Does the Lloyd-Max algorithm always find the optimal quantizer?

# Lloyd-Max algorithm

Does the Lloyd-Max algorithm always find the optimal quantizer?

It always finds a locally optimal quantizer, but when there are more than one local optima, it can converge to either one.

# Lloyd-Max algorithm

Does the Lloyd-Max algorithm always find the optimal quantizer?

It always finds a locally optimal quantizer, but when there are more than one local optima, it can converge to either one.

For unimodal signal distributions (where the probability density function has a single local maximum), typically there is only one locally optimal quantizer, and the Lloyd-Max algorithm will find it.

# Lloyd-Max algorithm

Does the Lloyd-Max algorithm always find the optimal quantizer?

It always finds a locally optimal quantizer, but when there are more than one local optima, it can converge to either one.

For unimodal signal distributions (where the probability density function has a single local maximum), typically there is only one locally optimal quantizer, and the Lloyd-Max algorithm will find it.

What to do when there are multiple local optima?

- Start from a reasonably good initial condition.

# Lloyd-Max algorithm

Does the Lloyd-Max algorithm always find the optimal quantizer?

It always finds a locally optimal quantizer, but when there are more than one local optima, it can converge to either one.

For unimodal signal distributions (where the probability density function has a single local maximum), typically there is only one locally optimal quantizer, and the Lloyd-Max algorithm will find it.

What to do when there are multiple local optima?

- Start from a reasonably good initial condition.
- Start from multiple initial conditions, and select the best obtained quantizer.

# Vector quantization

Vector quantization is relevant when we need to quantize a multi-dimensional source with correlated coordinates.

# Vector quantization

Vector quantization is relevant when we need to quantize a multi-dimensional source with correlated coordinates.

Example. The height and weight of people in a population is typically dependent. If we quantize them jointly, the quantizer can utilize the joint distribution.

# Vector quantization

Vector quantization is relevant when we need to quantize a multi-dimensional source with correlated coordinates.

Example. The height and weight of people in a population is typically dependent. If we quantize them jointly, the quantizer can utilize the joint distribution.

If 50kg and 100kg have the same relative frequency within the population, they will typically be quantized with similar distortion.

# Vector quantization

Vector quantization is relevant when we need to quantize a multi-dimensional source with correlated coordinates.

Example. The height and weight of people in a population is typically dependent. If we quantize them jointly, the quantizer can utilize the joint distribution.

If 50kg and 100kg have the same relative frequency within the population, they will typically be quantized with similar distortion.

However, the pair (200cm, 50kg) is less frequent than (200cm, 100kg), so it can be quantized with higher distortion.

# Vector quantization

Vector quantization is relevant when we need to quantize a multi-dimensional source with correlated coordinates.

Example. The height and weight of people in a population is typically dependent. If we quantize them jointly, the quantizer can utilize the joint distribution.

If 50kg and 100kg have the same relative frequency within the population, they will typically be quantized with similar distortion.

However, the pair (200cm, 50kg) is less frequent than (200cm, 100kg), so it can be quantized with higher distortion.

Quantizing height and weight jointly can do that. Quantizing them separately cannot do that.

# Vector quantization

Similar to the 1-dimensional case, a vector quantizer $Q : \mathbb{R}^d \to \mathbb{R}^d$ is described by the collection of output vectors $x^{(i)} \in \mathbb{R}^d$ and domains $B^{(i)} \subseteq \mathbb{R}^d$ such that

$$Q(x) = x^{(i)} \quad \forall x \in B^{(i)}$$

# Vector quantization

Similar to the 1-dimensional case, a vector quantizer $Q : \mathbb{R}^d \to \mathbb{R}^d$ is described by the collection of output vectors $x^{(i)} \in \mathbb{R}^d$ and domains $B^{(i)} \subseteq \mathbb{R}^d$ such that

$$Q(x) = x^{(i)} \quad \forall x \in B^{(i)}$$

We use the Euclidean distance between points $x, y \in \mathbb{R}^d$:

$$d(x, y) = \|x - y\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_d - y_d)^2}$$

For coordinates of different units (e.g. kg vs cm), we usually apply a linear scaling to bring all coordinates to the same order of magnitude first!

# Vector quantization

Similar to the 1-dimensional case, a vector quantizer $Q : \mathbb{R}^d \to \mathbb{R}^d$ is described by the collection of output vectors $x^{(i)} \in \mathbb{R}^d$ and domains $B^{(i)} \subseteq \mathbb{R}^d$ such that

$$Q(x) = x^{(i)} \quad \forall x \in B^{(i)}$$

We use the Euclidean distance between points $x, y \in \mathbb{R}^d$:

$$d(x, y) = \|x - y\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_d - y_d)^2}$$

For coordinates of different units (e.g. kg vs cm), we usually apply a linear scaling to bring all coordinates to the same order of magnitude first!

The distortion can now be defined as

$$D(Q) = \mathbb{E}(\|X - Q(X)\|^2) = \sum_{i=1}^{N} \int_{B^{(i)}} \|x - x^{(i)}\|^2 f(x) \mathrm{d}x,$$

where $f(x)$ is the $d$-dimensional density function of $X$.

# Vector quantization

The nearest neighbour rule and the center of gravity rule must hold for any optimal quantizer also in higher dimensions.

# Vector quantization

The nearest neighbour rule and the center of gravity rule must hold for any optimal quantizer also in higher dimensions.

Nearest neighbour rule:

$$B_i = \{x : \|x - x_i\| \leq \|x - x_j\| \text{ for any } j \neq i\}.$$

# Vector quantization

The nearest neighbour rule and the center of gravity rule must hold for any optimal quantizer also in higher dimensions.

Nearest neighbour rule:

$$B_i = \{x : \|x - x_i\| \leq \|x - x_j\| \text{ for any } j \neq i\}.$$

Center of gravity rule:

$$x_i = \mathbb{E}(X|X \in B_i) = \frac{\int_{B_i} xf(x)\mathrm{d}x}{\int_{B_i} f(x)\mathrm{d}x}.$$

# Vector quantization

The nearest neighbour rule and the center of gravity rule must hold for any optimal quantizer also in higher dimensions.

Nearest neighbour rule:

$$B_i = \{x : \|x - x_i\| \leq \|x - x_j\| \text{ for any } j \neq i\}.$$

Center of gravity rule:

$$x_i = \mathbb{E}(X|X \in B_i) = \frac{\int_{B_i} xf(x)\mathrm{d}x}{\int_{B_i} f(x)\mathrm{d}x}.$$

A quantizer satisfies the Lloyd-Max property if it satisfies both the nearest neighbour rule and the center of gravity rule.

# Optimal quantizer

A quantizer is optimal if it has minimal distortion.

# Optimal quantizer

A quantizer is optimal if it has minimal distortion.

Finding the optimal quantizer explicitly in higher dimension is generally a difficult problem.

# Optimal quantizer

A quantizer is optimal if it has minimal distortion.

Finding the optimal quantizer explicitly in higher dimension is generally a difficult problem.

Example. How does the optimal quantizer for $N = 4$ output vectors look like over a square domain with uniform density?

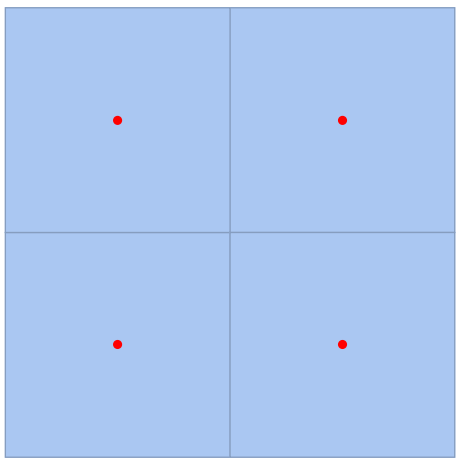
# Optimal quantizer

A quantizer is optimal if it has minimal distortion.

Finding the optimal quantizer explicitly in higher dimension is generally a difficult problem.

Example. How does the optimal quantizer for $N = 4$ output vectors look like over a square domain with uniform density?

This is a good candidate:



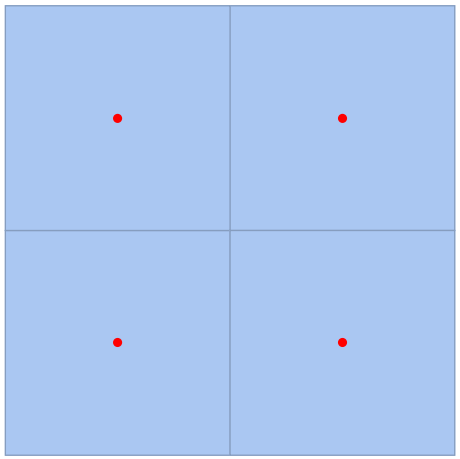
# Optimal quantizer

A quantizer is optimal if it has minimal distortion.

Finding the optimal quantizer explicitly in higher dimension is generally a difficult problem.

Example. How does the optimal quantizer for $N = 4$ output vectors look like over a square domain with uniform density?

This is a good candidate:



How could we find an optimal quantizer numerically?

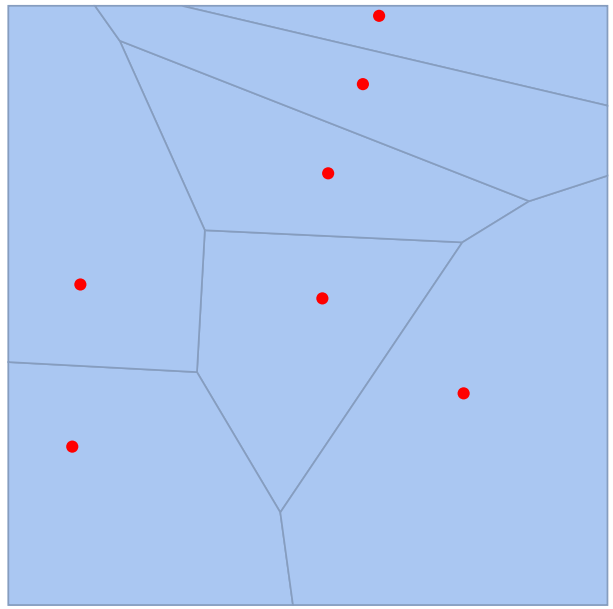
# Lloyd-Max algorithm in higher dimensions

The Lloyd-Max algorithm works the same for higher dimensions.

1. Initialize the levels.
2. Compute the domains for the output vectors using the nearest neighbour property.
3. Compute $D(Q)$, compare it to the old value of $D(Q)$ from the previous iteration, and if the difference is below a given threshold, stop.
4. Otherwise, from the domains, compute new values for the output vectors according to the center of gravity rule, then repeat from step 2.

# Lloyd-Max algorithm in higher dimensions

The nearest neighbor rule leads to the so-called Voronoi cells:



In $d = 2$ dimensions, the Voronoi cells are polygon-shaped.

# Lloyd-Max algorithm in higher dimensions

The Lloyd-Max algorithm will converge to a locally optimal quantizer.

In higher dimensions, there are possibly many different local optima even for simple distributions such as a uniform distribution over a square.

# Lloyd-Max algorithm in higher dimensions

The Lloyd-Max algorithm will converge to a locally optimal quantizer.

In higher dimensions, there are possibly many different local optima even for simple distributions such as a uniform distribution over a square.

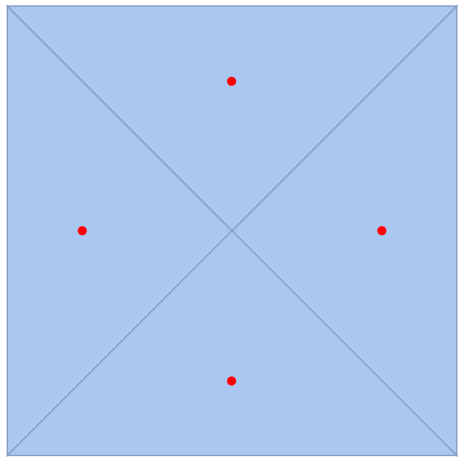
Also, in higher dimensions, the Lloyd-Max algorithm may have *unstable fixed points*; in practice, the algorithm will eventually move away from such fixed points, but it might take many iterations.

# Lloyd-Max algorithm in higher dimensions

The Lloyd-Max algorithm will converge to a locally optimal quantizer.

In higher dimensions, there are possibly many different local optima even for simple distributions such as a uniform distribution over a square.

Also, in higher dimensions, the Lloyd-Max algorithm may have *unstable fixed points*; in practice, the algorithm will eventually move away from such fixed points, but it might take many iterations.



# Lloyd-Max algorithm in higher dimensions

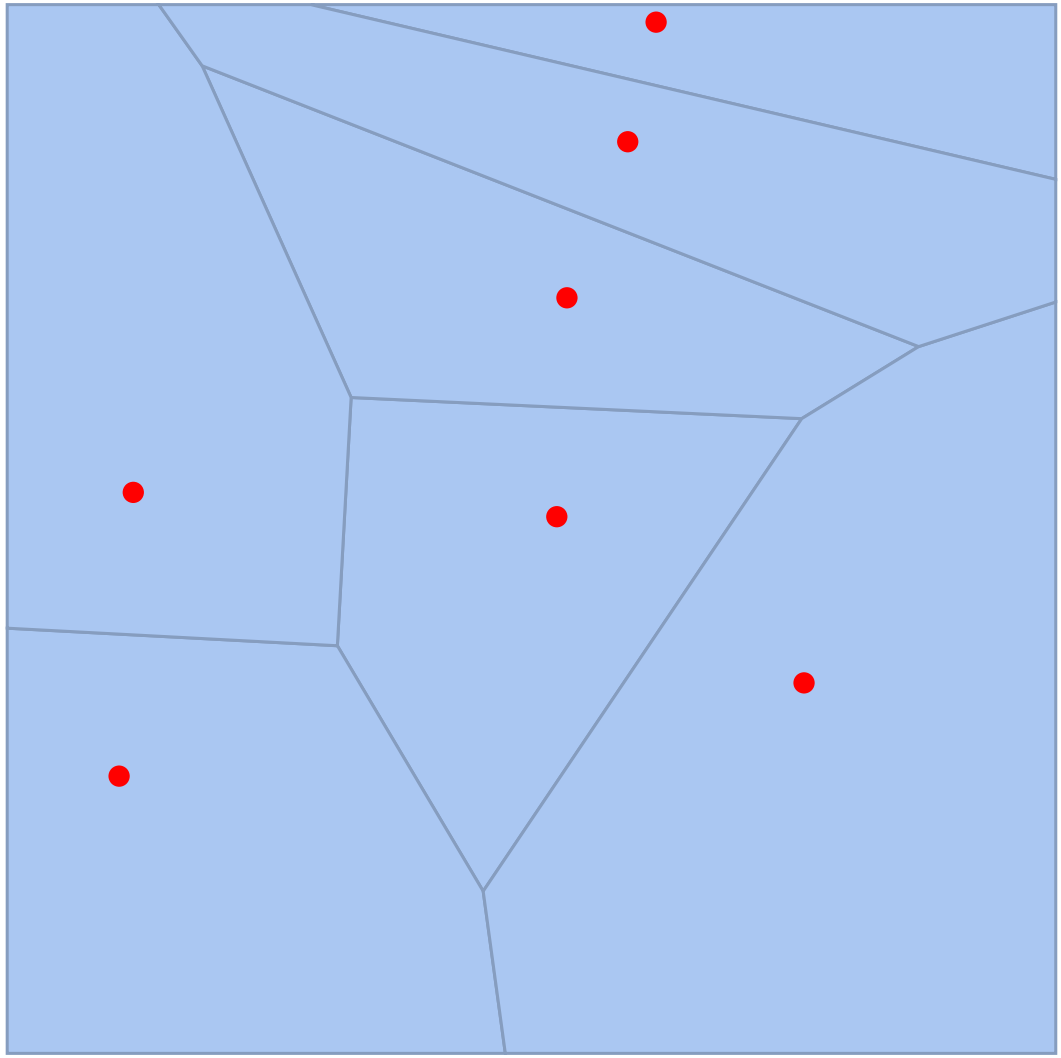
How could the optimal quantizer with $N = 7$ output vector look like for a uniform distribution over a square?

# Lloyd-Max algorithm in higher dimensions

How could the optimal quantizer with $N = 7$ output vector look like for a uniform distribution over a square?

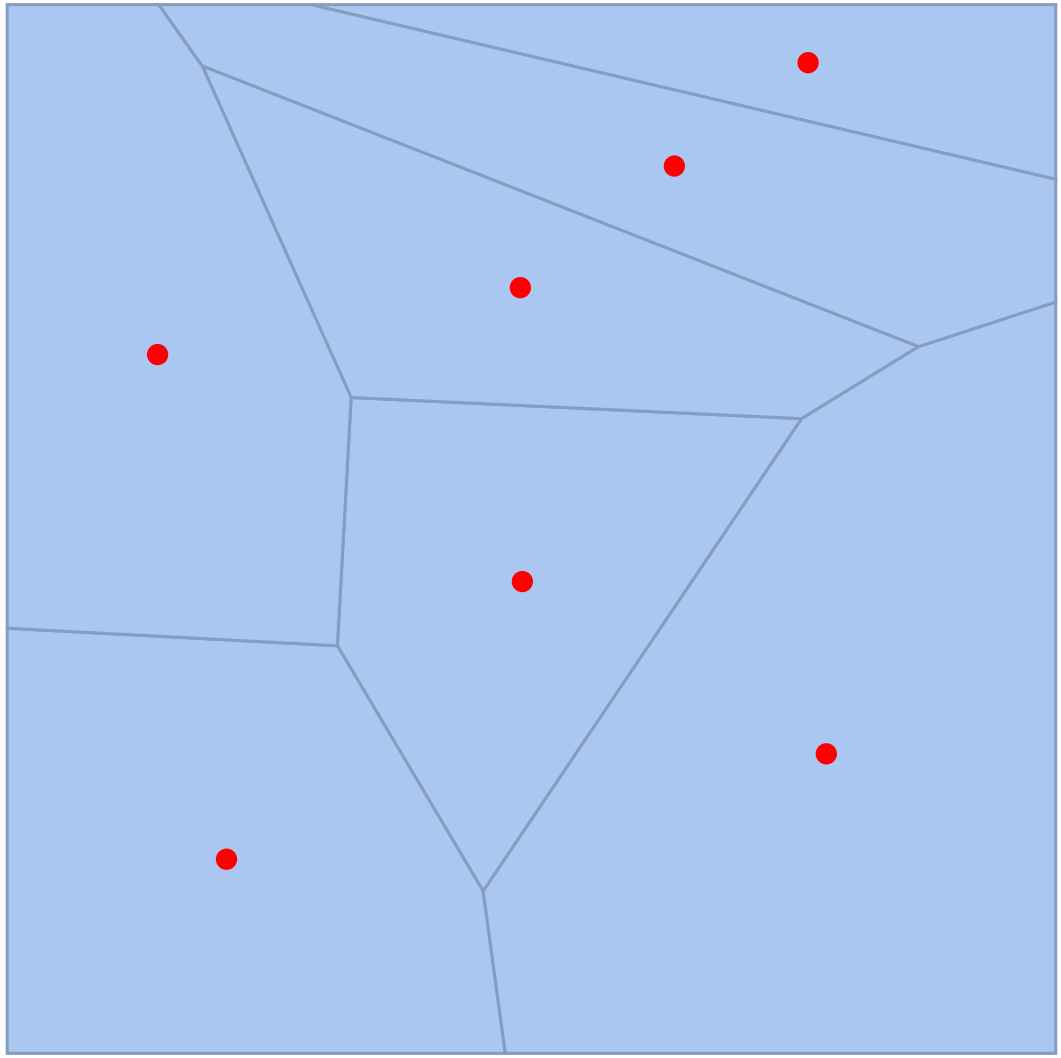
Let's see what the Lloyd-Max algorithm does! (We will test several different initializations to see if they converge to the same quantizer.)

# 2D Lloyd-Max algorithm



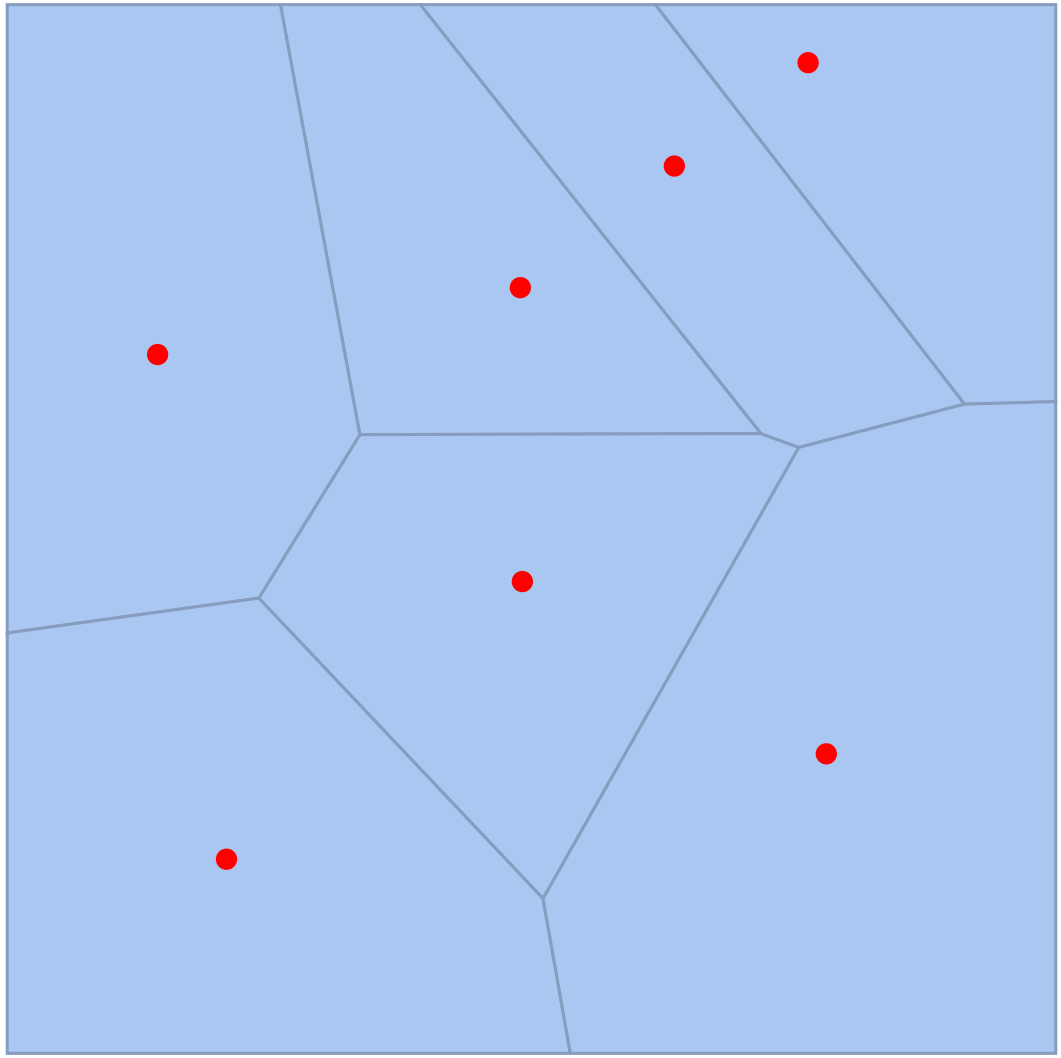
Initial points, domains from nearest neighbor rule

# 2D Lloyd-Max algorithm



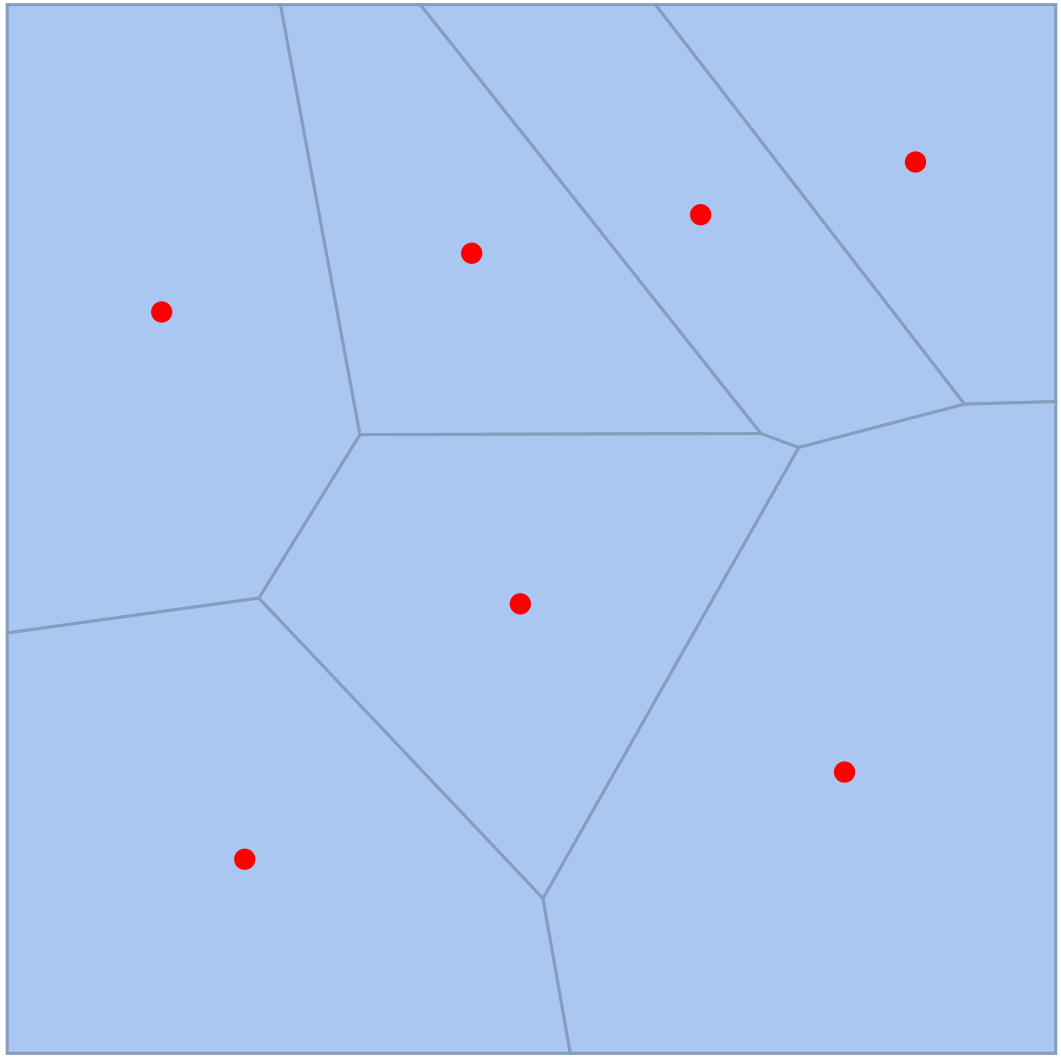
Points updated 1×

# 2D Lloyd-Max algorithm



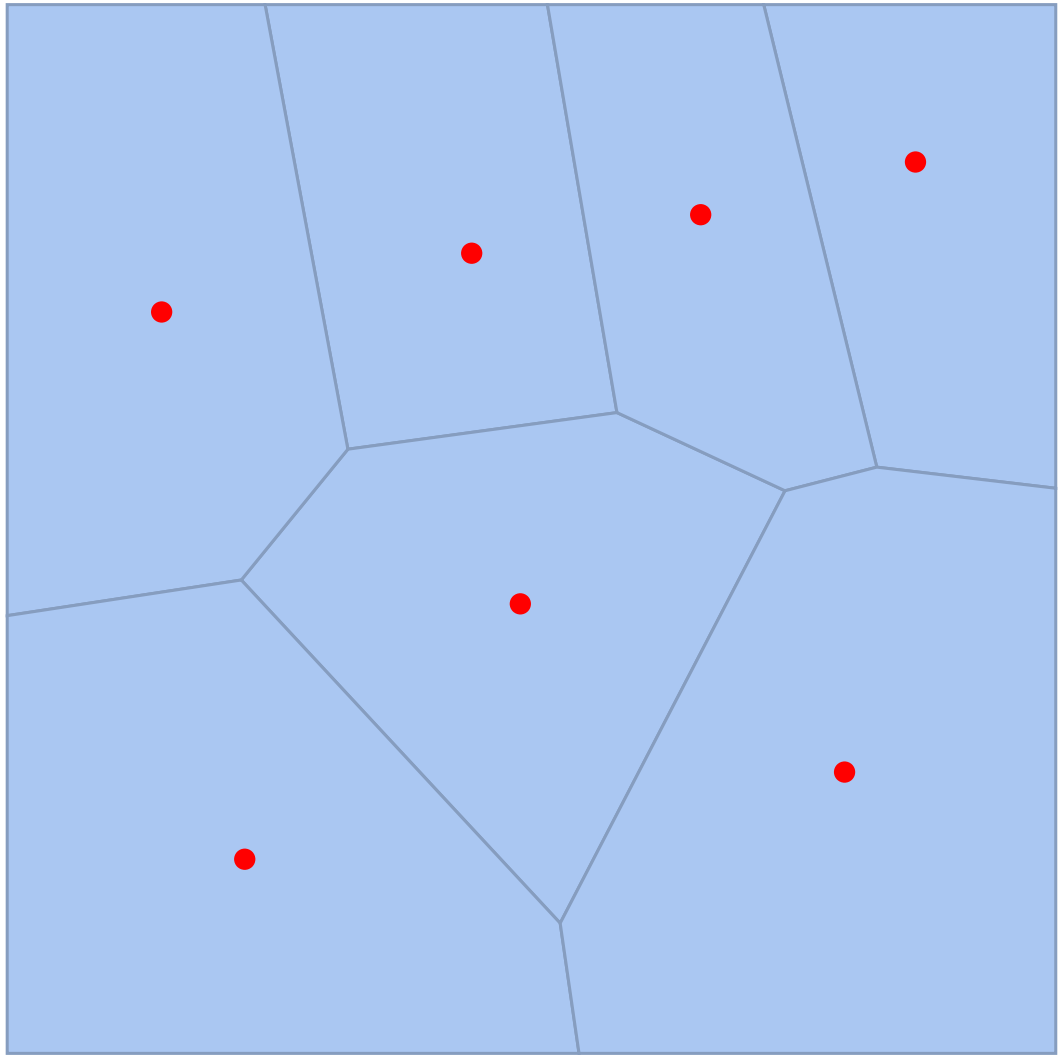
Points updated 1×, domains updated 1×

# 2D Lloyd-Max algorithm



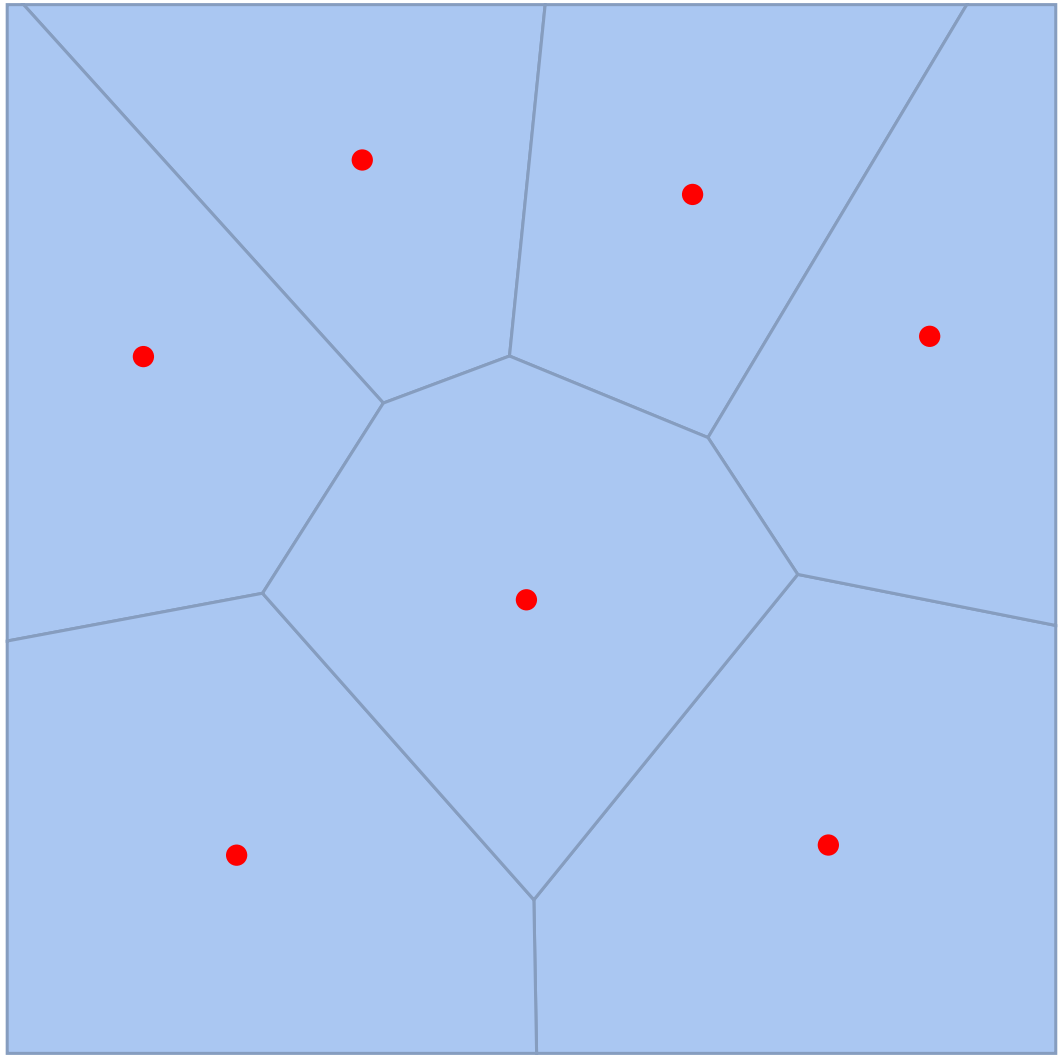
Points updated 2×, domains updated 1×

# 2D Lloyd-Max algorithm



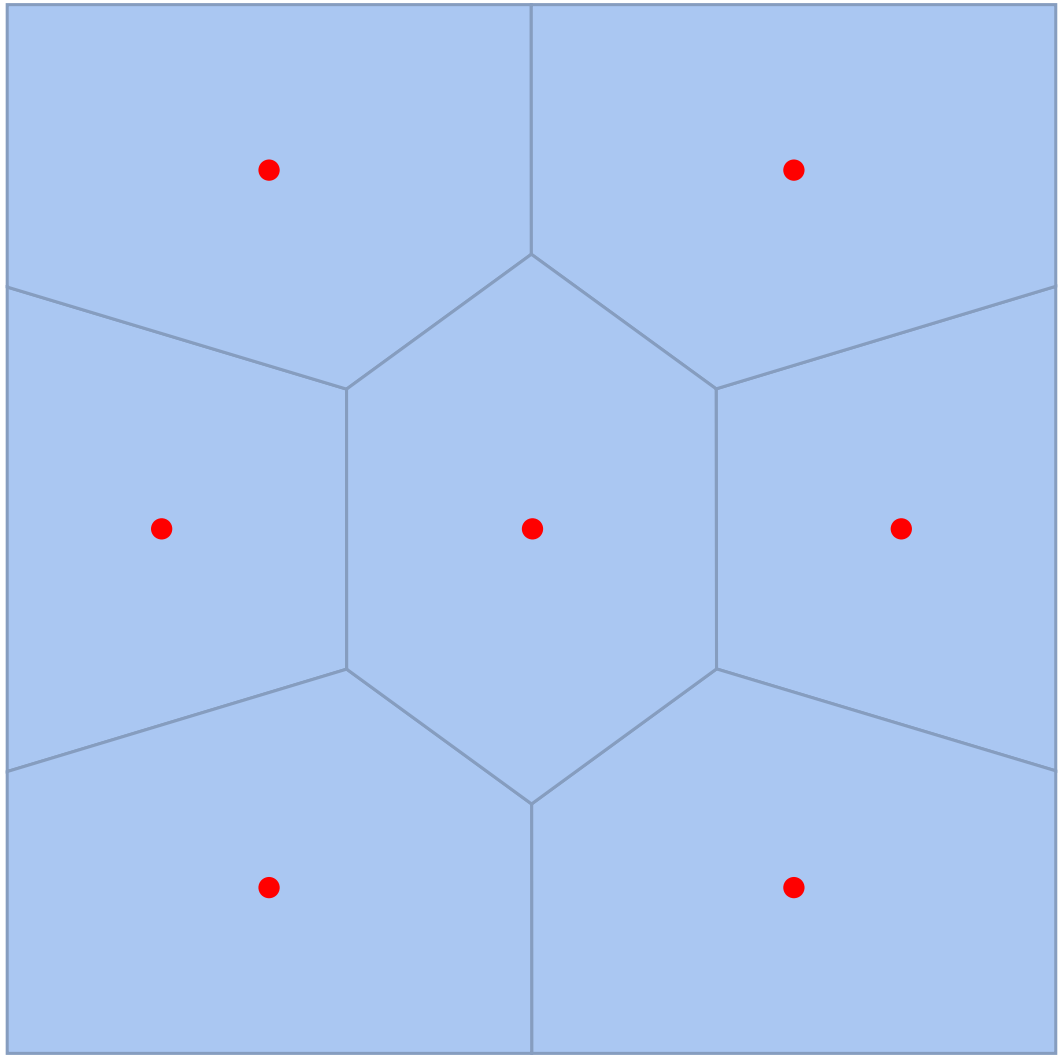
Points updated 2×, domains updated 2×

# 2D Lloyd-Max algorithm



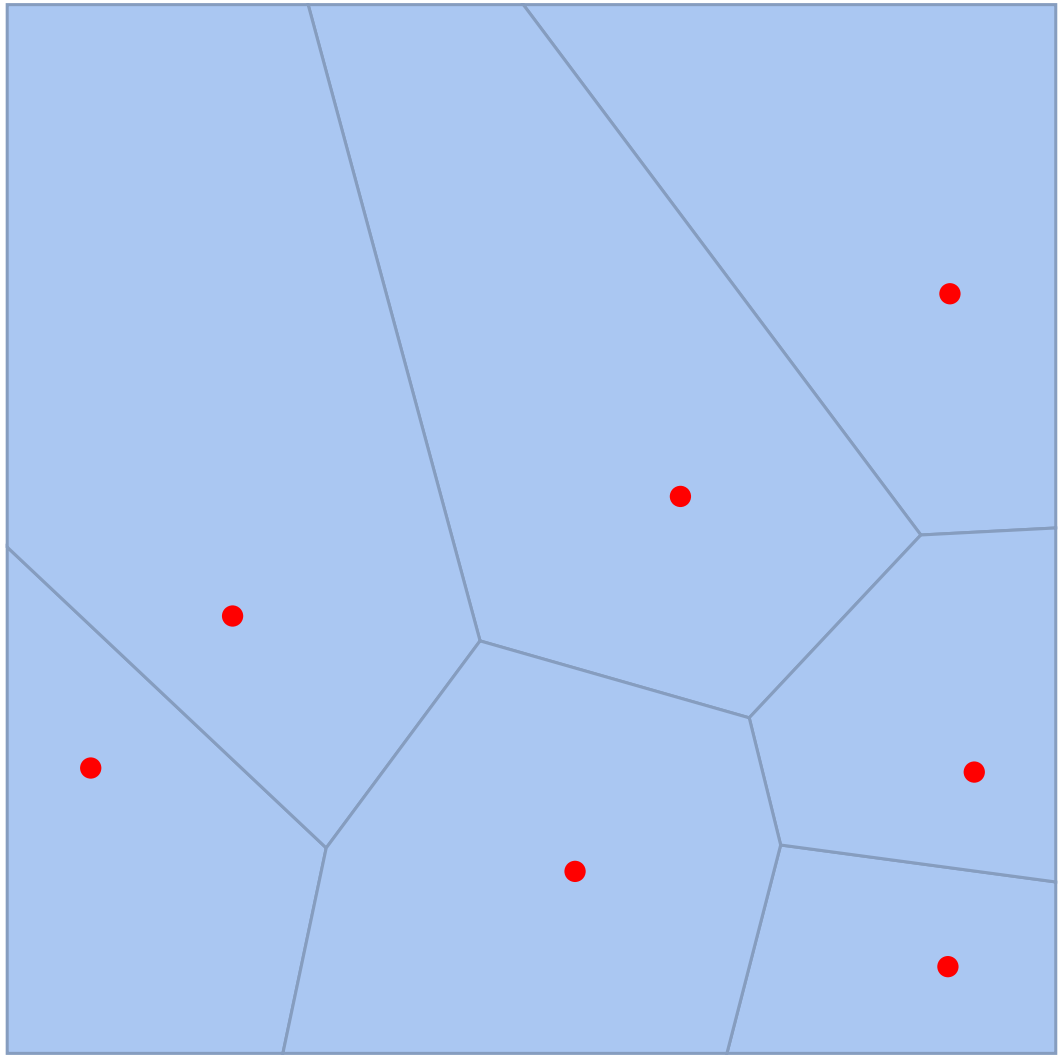
Points updated 10×, domains updated 10×

# 2D Lloyd-Max algorithm



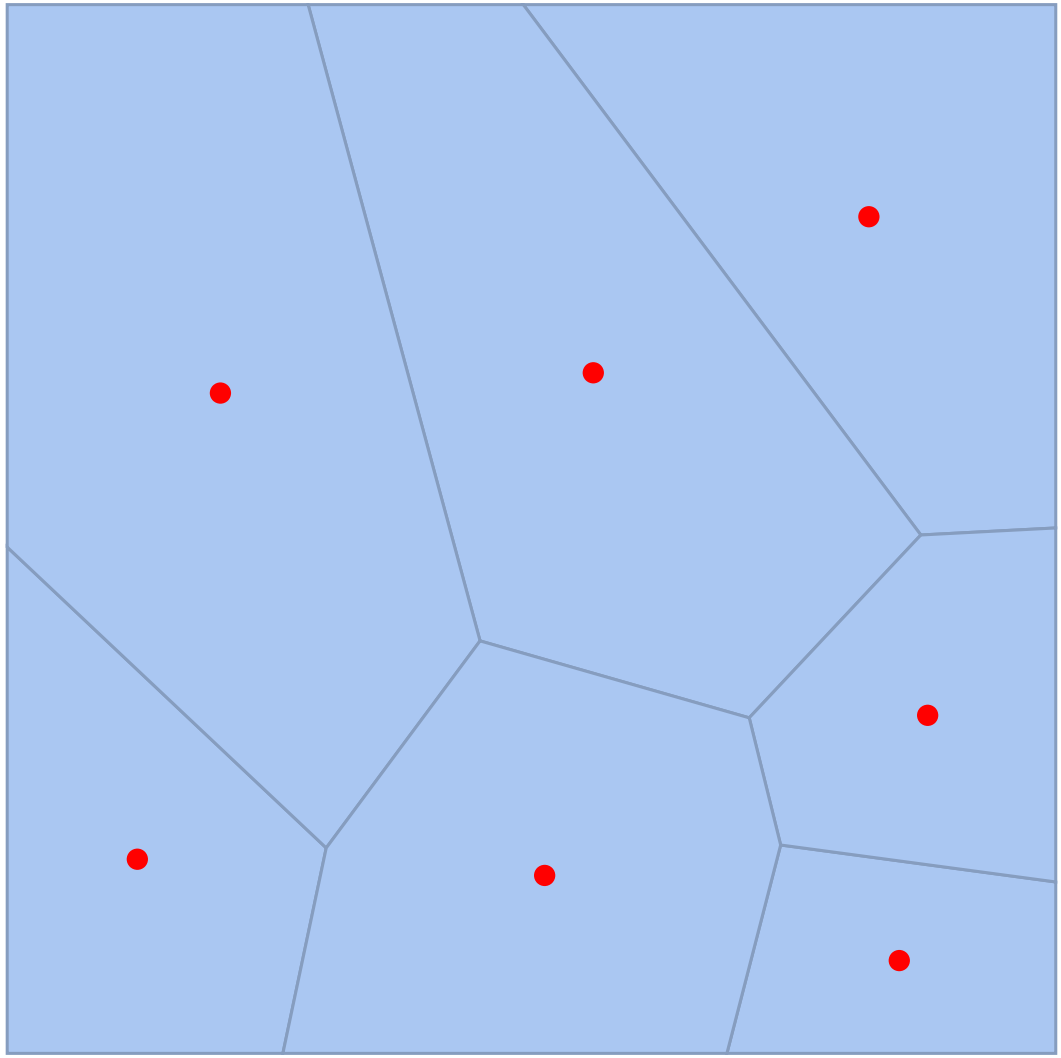
Points updated 50×, domains updated 50×

# 2D Lloyd-Max algorithm



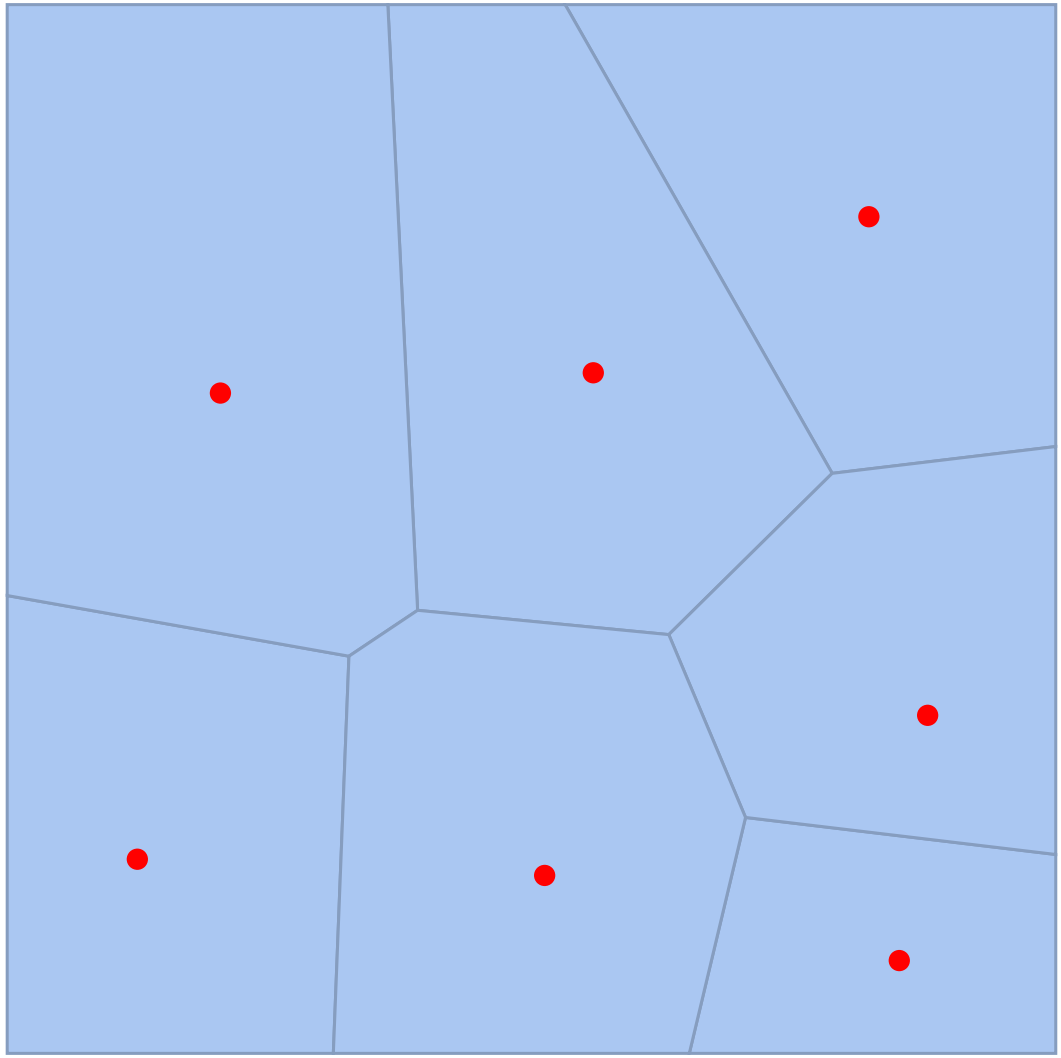
Initial points, domains from nearest neighbor rule

# 2D Lloyd-Max algorithm



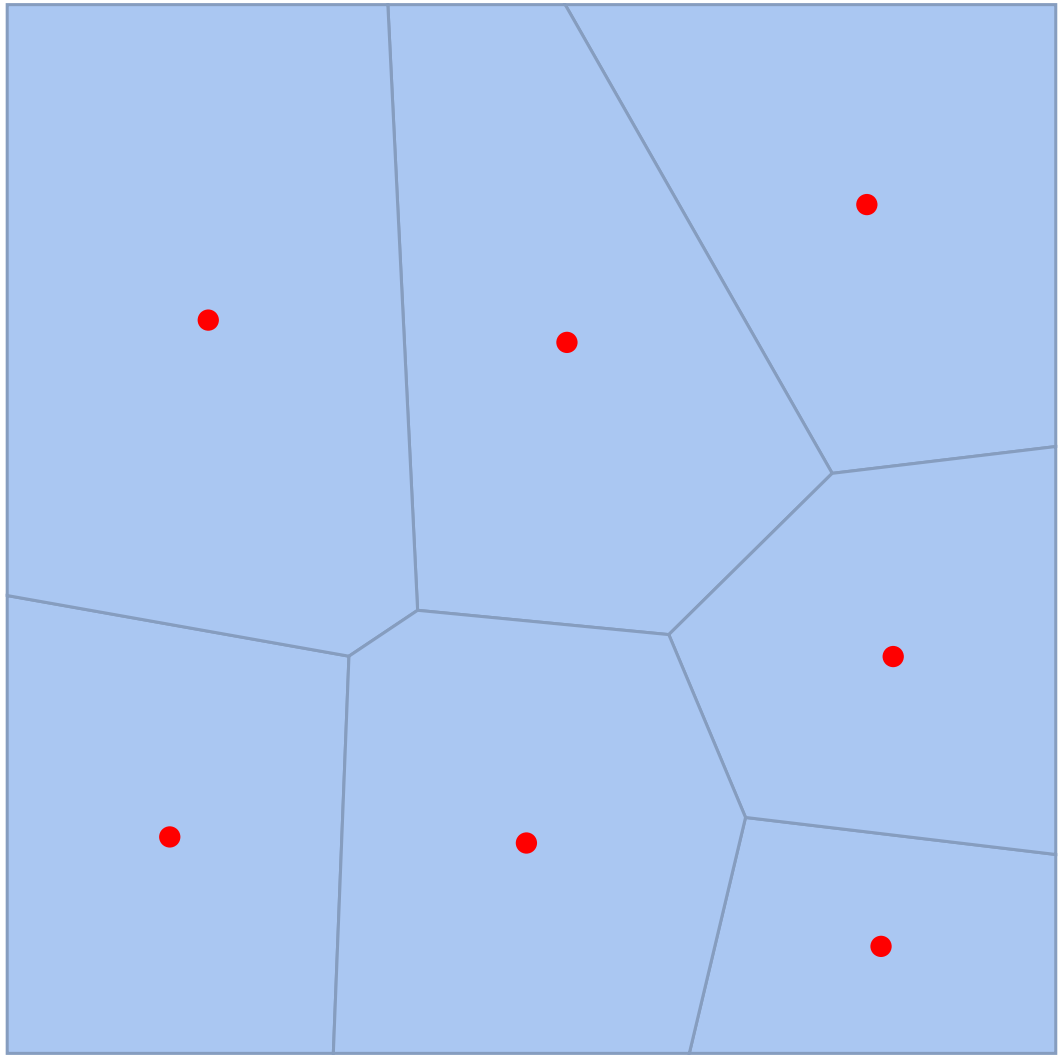
Points updated 1×

# 2D Lloyd-Max algorithm



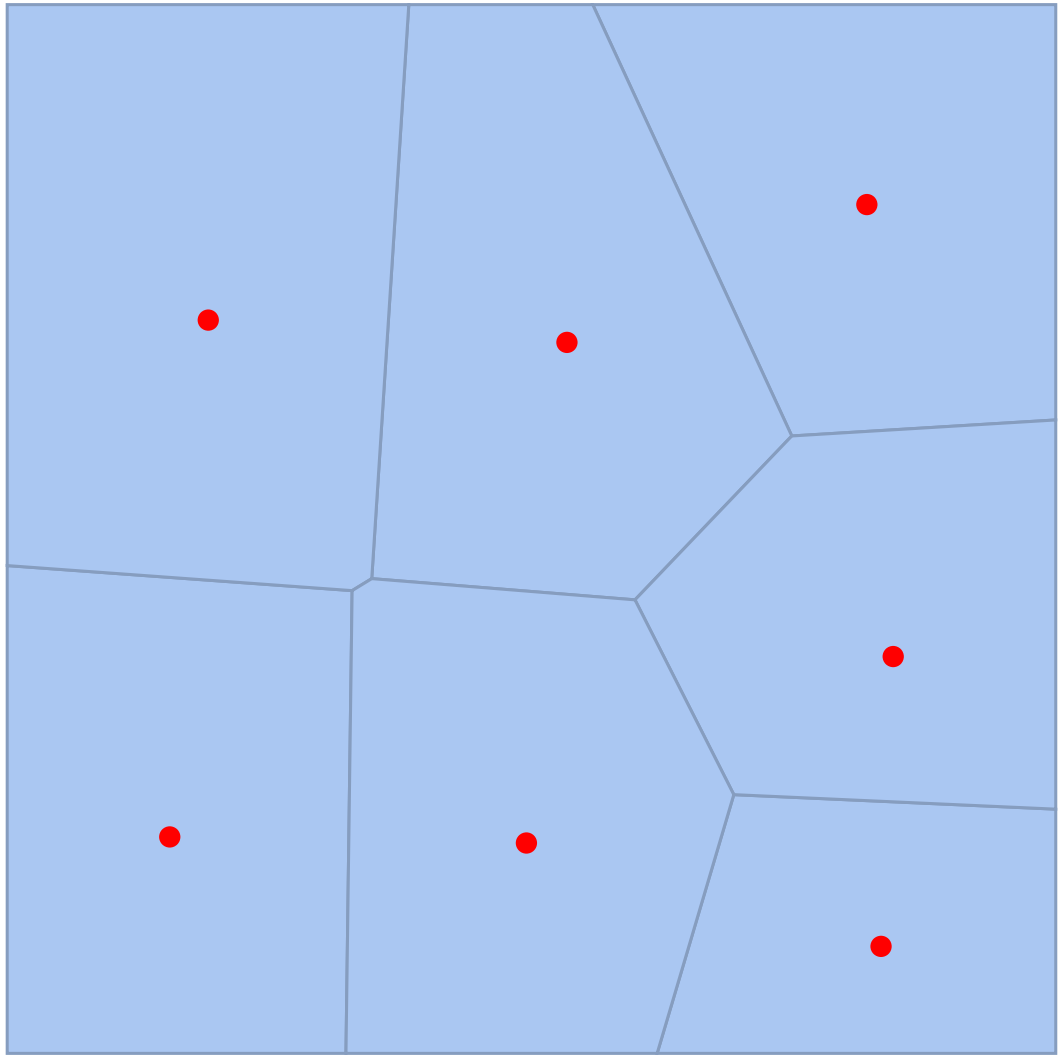
Points updated 1×, domains updated 1×

# 2D Lloyd-Max algorithm



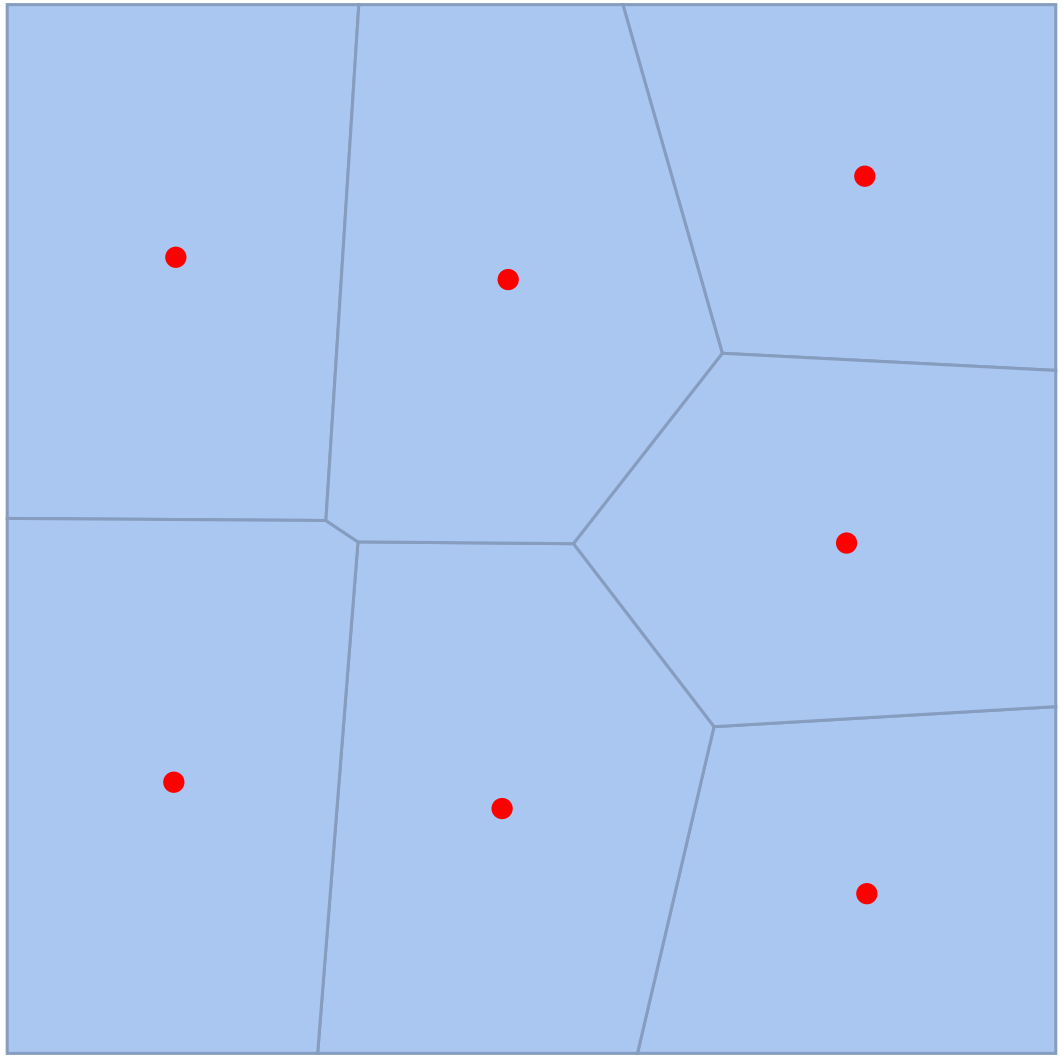
Points updated 2×, domains updated 1×

# 2D Lloyd-Max algorithm



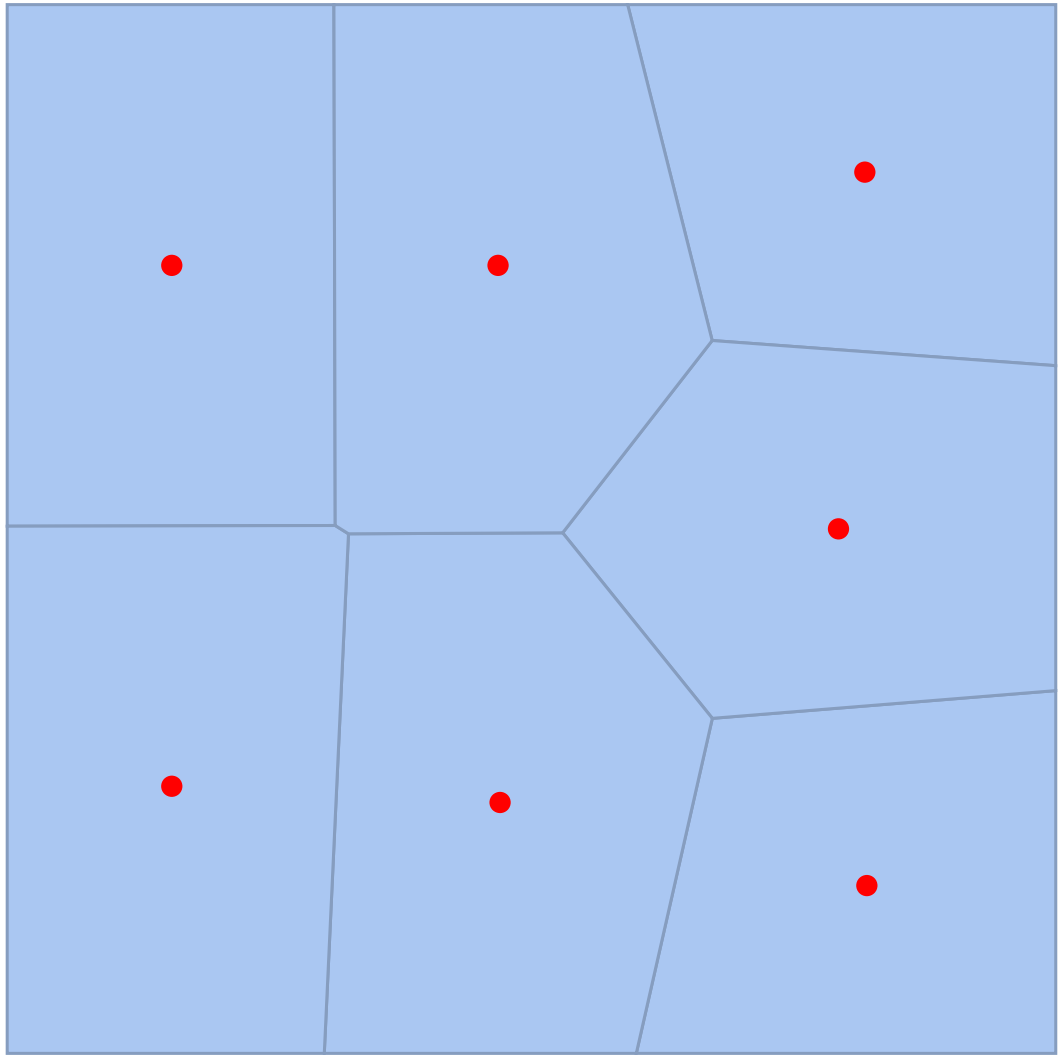
Points updated 2×, domains updated 2×

# 2D Lloyd-Max algorithm



Points updated 10×, domains updated 10×

# 2D Lloyd-Max algorithm



Points updated 50×, domains updated 50×

# Lloyd-Max algorithm in higher dimensions

In general, the result of the Lloyd-Max algorithm in higher dimensions depends highly on the initialization.

# Lloyd-Max algorithm in higher dimensions

In general, the result of the Lloyd-Max algorithm in higher dimensions depends highly on the initialization.

Similar to the 1-dimensional case, this can be addressed by running the algorithm from many different initializations, then selecting the best result.

# Lloyd-Max algorithm in higher dimensions

In general, the result of the Lloyd-Max algorithm in higher dimensions depends highly on the initialization.

Similar to the 1-dimensional case, this can be addressed by running the algorithm from many different initializations, then selecting the best result.

Depending on the application, we might also settle for a locally optimal quantizer.

# Outlook: structured quantizers

The Lloyd-Max algorithm converges to a locally optimal quantizer, which can often be relatively random-looking.

# Outlook: structured quantizers

The Lloyd-Max algorithm converges to a locally optimal quantizer, which can often be relatively random-looking.

Structured quantizers, on the other hand, search for a quantizer in a specific subset of quantizers where optimization is easier. This will result in finding a quantizer that is typically not optimal, but reasonably close, and has some specific structure.

# Outlook: structured quantizers

The Lloyd-Max algorithm converges to a locally optimal quantizer, which can often be relatively random-looking.

Structured quantizers, on the other hand, search for a quantizer in a specific subset of quantizers where optimization is easier. This will result in finding a quantizer that is typically not optimal, but reasonably close, and has some specific structure.

Tree-structured vector quantizers. First, only one coordinate is quantized, then for each domain, the next coordinate is quantized (possibly in a different manner), and so on.

# Outlook: structured quantizers

The Lloyd-Max algorithm converges to a locally optimal quantizer, which can often be relatively random-looking.

Structured quantizers, on the other hand, search for a quantizer in a specific subset of quantizers where optimization is easier. This will result in finding a quantizer that is typically not optimal, but reasonably close, and has some specific structure.

Tree-structured vector quantizers. First, only one coordinate is quantized, then for each domain, the next coordinate is quantized (possibly in a different manner), and so on.

Energy-form quantizers: in voice or music compression, usually there is a natural energy (volume) of the signal. The energy and the normalized signal (form) are quantized separately.

# Outlook: finite samples

In many applications, the $d$-dimensional probability density function is not available; instead a finite sample of signals is available.

# Outlook: finite samples

In many applications, the $d$-dimensional probability density function is not available; instead a finite sample of signals is available.

The Lloyd-Max algorithm and other optimizers all have versions adapted to such a finite sample instead of a theoretical density function.

# Outlook: finite samples

In many applications, the $d$-dimensional probability density function is not available; instead a finite sample of signals is available.

The Lloyd-Max algorithm and other optimizers all have versions adapted to such a finite sample instead of a theoretical density function.

We do not chase this direction.

# Transform coding

The general concept of transform coding is similar to text transforms: a transform is a reversible mapping of a signal such that the transformed signal is easier to compress than the original signal.

# Transform coding

The general concept of transform coding is similar to text transforms: a transform is a reversible mapping of a signal such that the transformed signal is easier to compress than the original signal.

The most widely used such transform is the Discrete cosine transform (DCT), designed for image and video processing (accordingly, it is the basis of JPEG and MPEG), but also used in many other applications.

# Transform coding

The general concept of transform coding is similar to text transforms: a transform is a reversible mapping of a signal such that the transformed signal is easier to compress than the original signal.

The most widely used such transform is the Discrete cosine transform (DCT), designed for image and video processing (accordingly, it is the basis of JPEG and MPEG), but also used in many other applications.

The main idea of DCT is to transform a block of $k$ values from the signal in a way that the first few values of the transform capture the main defining properties of the original signal block, and subsequent elements of the transform correspond to finer and finer details (which can then be compressed with higher distortion).

# DCT

Here we will discuss the version used for image processing.

DCT transforms a block of $k \times k$ pixels of an image jointly. In the JPEG standard, $k = 8$ (the general algorithm works for any choice of $k$, but is easier to compute when $k$ is a power of 2).

# DCT

Here we will discuss the version used for image processing.

DCT transforms a block of $k \times k$ pixels of an image jointly. In the JPEG standard, $k = 8$ (the general algorithm works for any choice of $k$, but is easier to compute when $k$ is a power of 2).

First, the pixels are converted into a $k \times k$ array so that every pixel is represented by a single real value.

# DCT

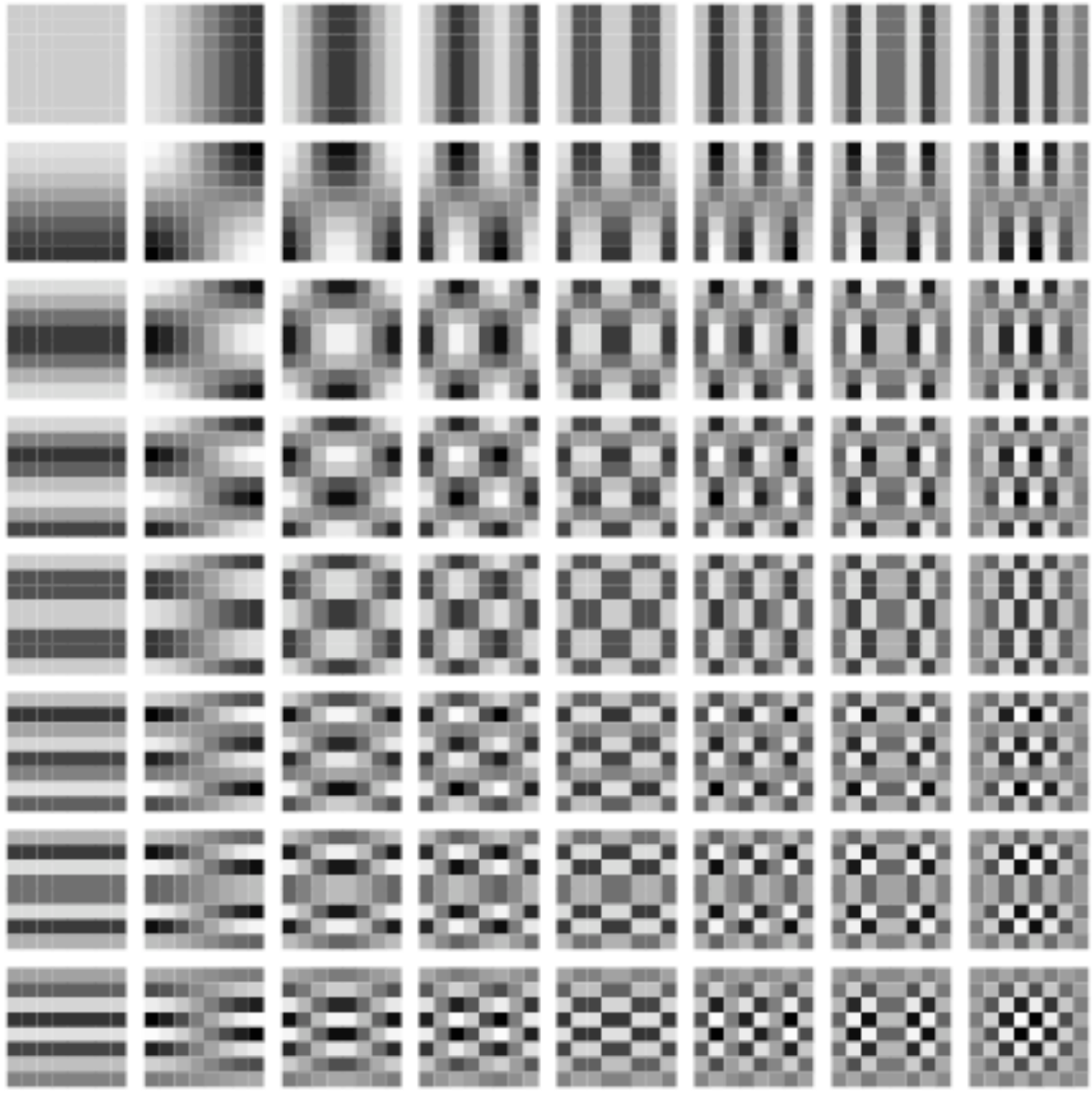
Here we will discuss the version used for image processing.

DCT transforms a block of $k \times k$ pixels of an image jointly. In the JPEG standard, $k = 8$ (the general algorithm works for any choice of $k$, but is easier to compute when $k$ is a power of 2).

First, the pixels are converted into a $k \times k$ array so that every pixel is represented by a single real value.

DCT then transforms the $k \times k$ values into another array of $k \times k$ values called *frequency coefficients*. The first of these coefficients (the *DC coefficient* or *main coefficient*) represents the average value of the original block, and subsequent coefficients represent deviations from the average in sinusoidal patterns with varying frequency.

# DCT



# DCT

Mathematically, the formula for the DCT is the following.

$X$ denotes the values of the original block, arranged in a $k \times k$ matrix. Then the DCT of $X$ is

$$Y = AXA^T,$$

where the matrix $A$ is the following:

- the first row of $A$ is $\begin{bmatrix} \frac{1}{\sqrt{k}} & \frac{1}{\sqrt{k}} & \cdots & \frac{1}{\sqrt{k}} \end{bmatrix}$
- for other rows,

$$a_{ij} = \sqrt{\frac{2}{k}} \cos\left(\frac{(2j-1)(i-1)\pi}{2k}\right)$$

# DCT

$A$ is an orthonormal matrix:

$$A^{-1} = A^T,$$

so the inverse transform is also simple:

$$Y = AXA^T \qquad \rightarrow \qquad X = A^T YA$$

# DCT

$A$ is an orthonormal matrix:

$$A^{-1} = A^T,$$

so the inverse transform is also simple:

$$Y = AXA^T \qquad \rightarrow \qquad X = A^T YA$$

The typical structure of $Y$ is such that the top left value (the DC coefficient) has high variance, and the other values are gradually decreasing in order away from the top left corner $\rightarrow$ they can be compressed to fewer bits.

# DCT

$A$ is an orthonormal matrix:

$$A^{-1} = A^T,$$

so the inverse transform is also simple:

$$Y = AXA^T \qquad \rightarrow \qquad X = A^T YA$$

The typical structure of $Y$ is such that the top left value (the DC coefficient) has high variance, and the other values are gradually decreasing in order away from the top left corner $\rightarrow$ they can be compressed to fewer bits.

Technically, the DCT does not compress; actual compression is done *after* applying DCT by quantizing the frequency coefficients.

# DCT

There are several variants of DCT, differing only in slight details. There are 16 variations based on how the sinusoidal patterns are arranged (8 called DCT-I to DCT-VIII and 8 called Discrete sine transforms, DST).

# DCT

There are several variants of DCT, differing only in slight details. There are 16 variations based on how the sinusoidal patterns are arranged (8 called DCT-I to DCT-VIII and 8 called Discrete sine transforms, DST).

The version presented above is DCT-II, and it is the most widely used variant.

# DCT

There are several variants of DCT, differing only in slight details. There are 16 variations based on how the sinusoidal patterns are arranged (8 called DCT-I to DCT-VIII and 8 called Discrete sine transforms, DST).

The version presented above is DCT-II, and it is the most widely used variant.

DCT is closely related to Discrete Fourier transform; the main difference is that DCT uses real values only, making the computation of DCT slightly easier.

# DCT

There are several variants of DCT, differing only in slight details. There are 16 variations based on how the sinusoidal patterns are arranged (8 called DCT-I to DCT-VIII and 8 called Discrete sine transforms, DST).

The version presented above is DCT-II, and it is the most widely used variant.

DCT is closely related to Discrete Fourier transform; the main difference is that DCT uses real values only, making the computation of DCT slightly easier.

Algorithmic complexity. Normally, computing the matrix product $AXA^T$ would take $O(k^4)$ operations, but this can be reduced to $O(k^2 \log(k))$ due to the special structure of $A$ using *Fast cosine transform* (FCT).

# DCT

There are several variants of DCT, differing only in slight details. There are 16 variations based on how the sinusoidal patterns are arranged (8 called DCT-I to DCT-VIII and 8 called Discrete sine transforms, DST).

The version presented above is DCT-II, and it is the most widely used variant.

DCT is closely related to Discrete Fourier transform; the main difference is that DCT uses real values only, making the computation of DCT slightly easier.

Algorithmic complexity. Normally, computing the matrix product $AXA^T$ would take $O(k^4)$ operations, but this can be reduced to $O(k^2 \log(k))$ due to the special structure of $A$ using *Fast cosine transform* (FCT).

FCT utilizes a factorization of $A$ as a product of simple matrices (similar to Fast Fourier transform (FFT)). We do not go into more details here.

# Predictive coding

Another class of compression methods is called Predictive coding.

# Predictive coding

Another class of compression methods is called Predictive coding.

The general idea of Predictive coding is the following: if a data source is such that each signal can be predicted from the previous signal (or several signals) with high accuracy, then the difference of the predicted signal and the actual signal is smaller order than the original signal, so fewer precise bits are sufficient when compressing it, compared to the original signal.

# Predictive coding

Another class of compression methods is called Predictive coding.

The general idea of Predictive coding is the following: if a data source is such that each signal can be predicted from the previous signal (or several signals) with high accuracy, then the difference of the predicted signal and the actual signal is smaller order than the original signal, so fewer precise bits are sufficient when compressing it, compared to the original signal.

Example. Consider the following sequence of bytes:

$$147, 145, 141, 146, 149, 147, 143, 145$$

# Predictive coding

Another class of compression methods is called Predictive coding.

The general idea of Predictive coding is the following: if a data source is such that each signal can be predicted from the previous signal (or several signals) with high accuracy, then the difference of the predicted signal and the actual signal is smaller order than the original signal, so fewer precise bits are sufficient when compressing it, compared to the original signal.

Example. Consider the following sequence of bytes:

$$147, 145, 141, 146, 149, 147, 143, 145$$

Here, it makes sense to use each signal as a predictor for the next signal, and take the difference, so the transformed sequence is

$$147, -2, -4, 5, 3, -2, -4, 2$$

# Predictive coding

$$147, -2, -4, 5, 3, -2, -4, 2$$

All elements of the sequence except the first can be coded using 4 bits (1 sign bit and 3 value bits).

# Predictive coding

$$147, -2, -4, 5, 3, -2, -4, 2$$

All elements of the sequence except the first can be coded using 4 bits (1 sign bit and 3 value bits).

- the first signal is coded using 8 bits;
- we also need to transmit the information of how many bits are used for subsequent values (4);
- further signals are coded using 4 bits each.

# Predictive coding

$$147, -2, -4, 5, 3, -2, -4, 2$$

All elements of the sequence except the first can be coded using 4 bits (1 sign bit and 3 value bits).

- the first signal is coded using 8 bits;
- we also need to transmit the information of how many bits are used for subsequent values (4);
- further signals are coded using 4 bits each.

More general versions of this idea may use predictors other than the previous signal; in this case, the difference between the predictor and the actual value are used for later signals.

# Predictive coding

Predictive coding can be executed in a lossless manner, or the signals can be quantized.

# Predictive coding

Predictive coding can be executed in a lossless manner, or the signals can be quantized.

When using quantizing, the order of quantization and prediction is important. When using the intuitive version

$$d_n = x_n - x_{n-1}, \qquad d_n \to \hat{d}_n,$$

(where $\hat{d}$ denotes quantization), the quantization errors may accumulate during reconstruction.

# Predictive coding

Predictive coding can be executed in a lossless manner, or the signals can be quantized.

When using quantizing, the order of quantization and prediction is important. When using the intuitive version

$$d_n = x_n - x_{n-1}, \qquad d_n \to \hat{d}_n,$$

(where $\hat{d}$ denotes quantization), the quantization errors may accumulate during reconstruction.

It is actually better to use

$$d_n = x_n - \hat{x}_{n-1}, \qquad d_n \to \hat{d}_n,$$

where $\hat{x}_{n-1}$ denotes the reconstructed version of $x_{n-1}$ based on $\hat{x}_0, \hat{d}_1, \ldots, \hat{d}_{n-1}$. In this order, the quantization error does not accumulate. (Essentially, the same information is used for compression and reconstruction.)