Linear-time Training of Nonlinear Low-Dimensional Embeddings using *N*-Body Approximation Algorithms

Max Vladymyrov
EECS, School of Engineering
University of California, Merced

EECS, UC Merced
March 14, 2014

In collaboration with **Miguel Á. Carreira-Perpiñán**

Linear-time Training of Nonlinear Low-Dimensional Embeddings

Winner of
2014 Student Poster Competition

N-Body Approxim[illegible]ms

Max Vladymyrov
EECS, School of Engineering
University of California, Merced

EECS, UC Merced
March 14, 2014

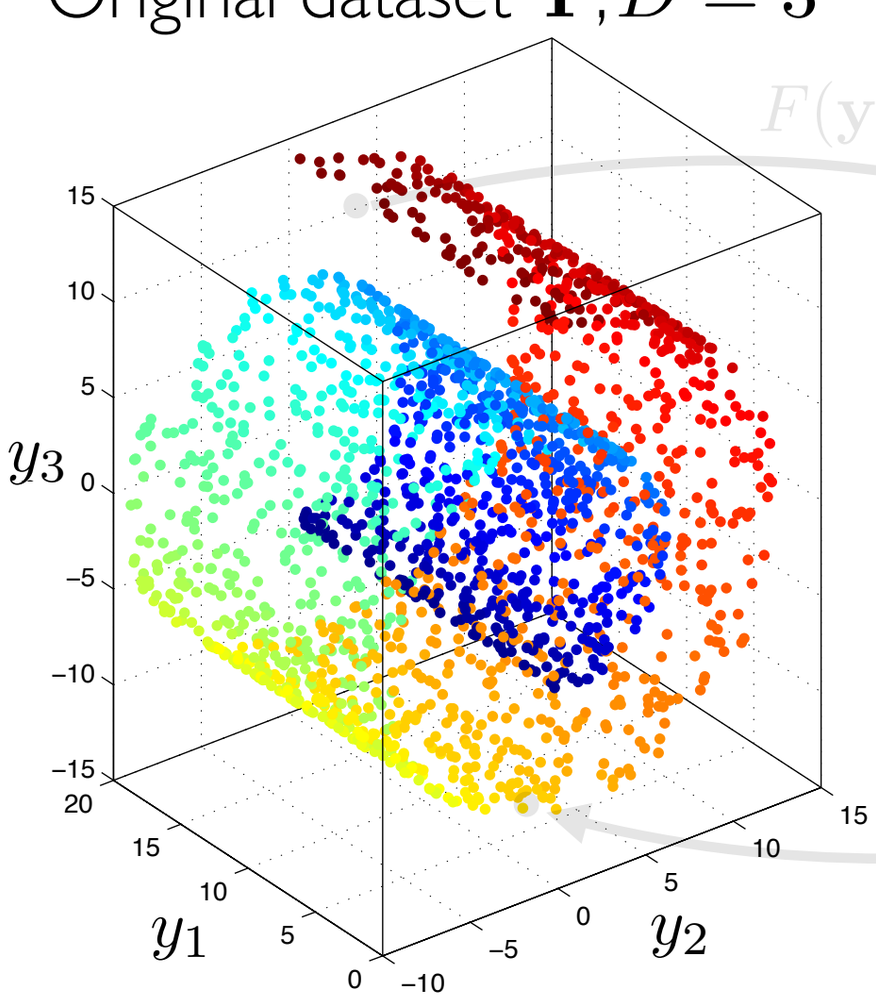
In collaboration with **Miguel Á. Carreira-Perpiñán**

Dimensionality reduction

Given a high-dimensional dataset $\mathbf{Y}_{D\times N} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$ find

- projection points $\mathbf{X}_{d\times N} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$,
- reduction mapping $\mathbf{x} = F(\mathbf{y})$,
- reconstruction mapping $\mathbf{y} = f(\mathbf{x})$,
- joint probability density $p(\mathbf{x}, \mathbf{y})$,
- estimate intrinsic dimensionality d

Original dataset $\mathbf{Y}, D = 3$



$F(\mathbf{y})$

Low-dimensional embedding $\mathbf{X}, d = 2$

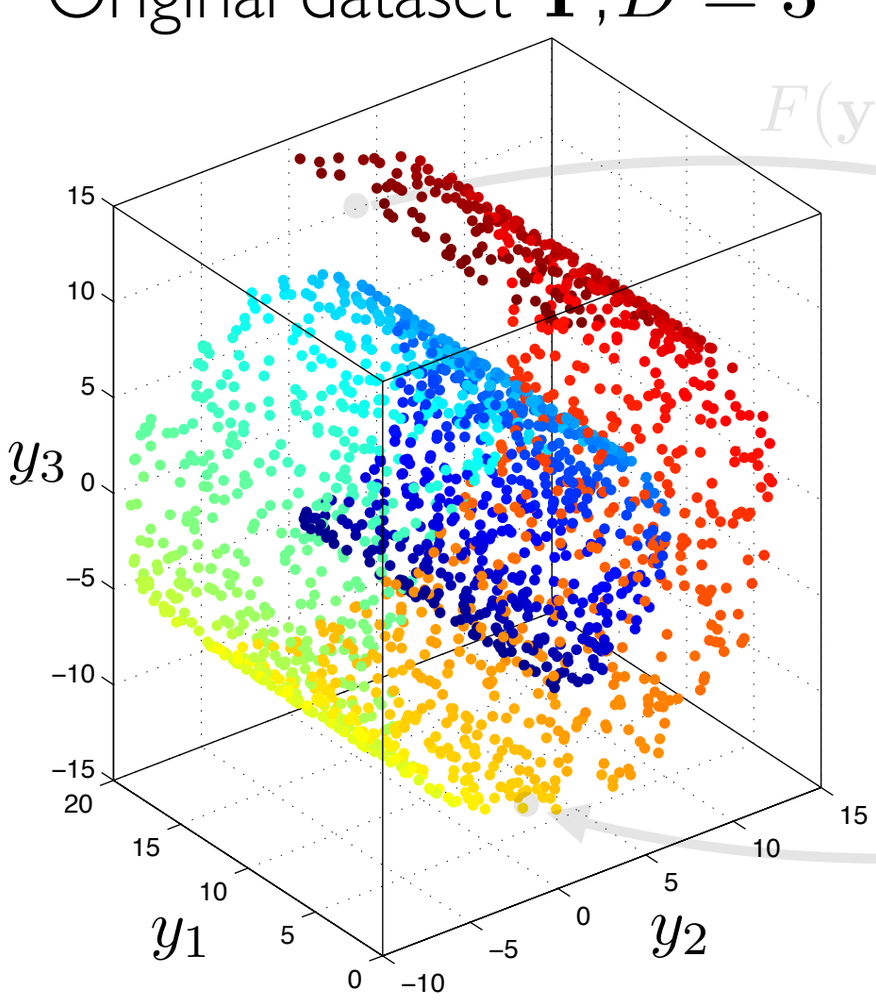
$f(\mathbf{x})$

Dimensionality reduction

Given a high-dimensional dataset $\mathbf{Y}_{D\times N} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$ find

- projection points $\mathbf{X}_{d\times N} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$,
- reduction mapping $\mathbf{x} = F(\mathbf{y})$,
- reconstruction mapping $\mathbf{y} = f(\mathbf{x})$,
- joint probability density $p(\mathbf{x}, \mathbf{y})$,
- estimate intrinsic dimensionality d

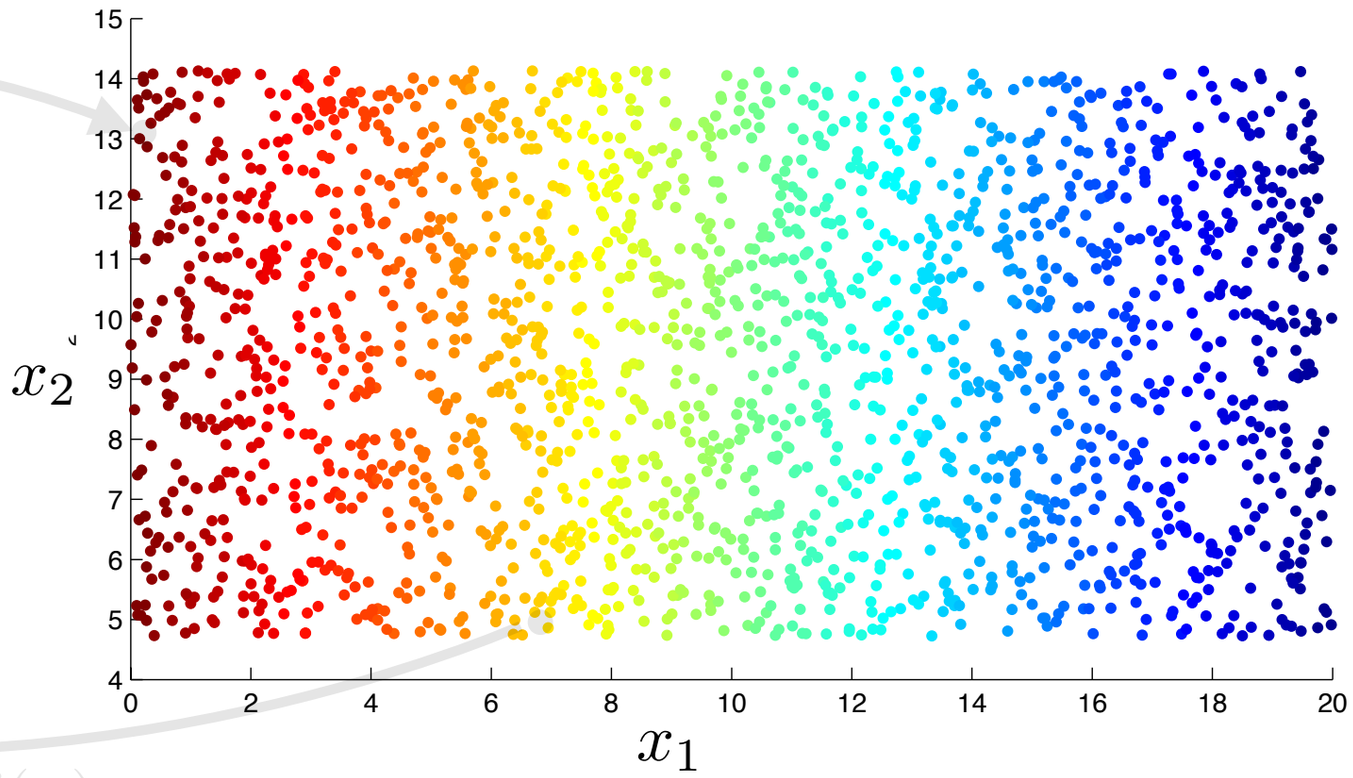
Original dataset $\mathbf{Y}, D = 3$



$F(\mathbf{y})$

$f(\mathbf{x})$

Low-dimensional embedding $\mathbf{X}, d = 2$



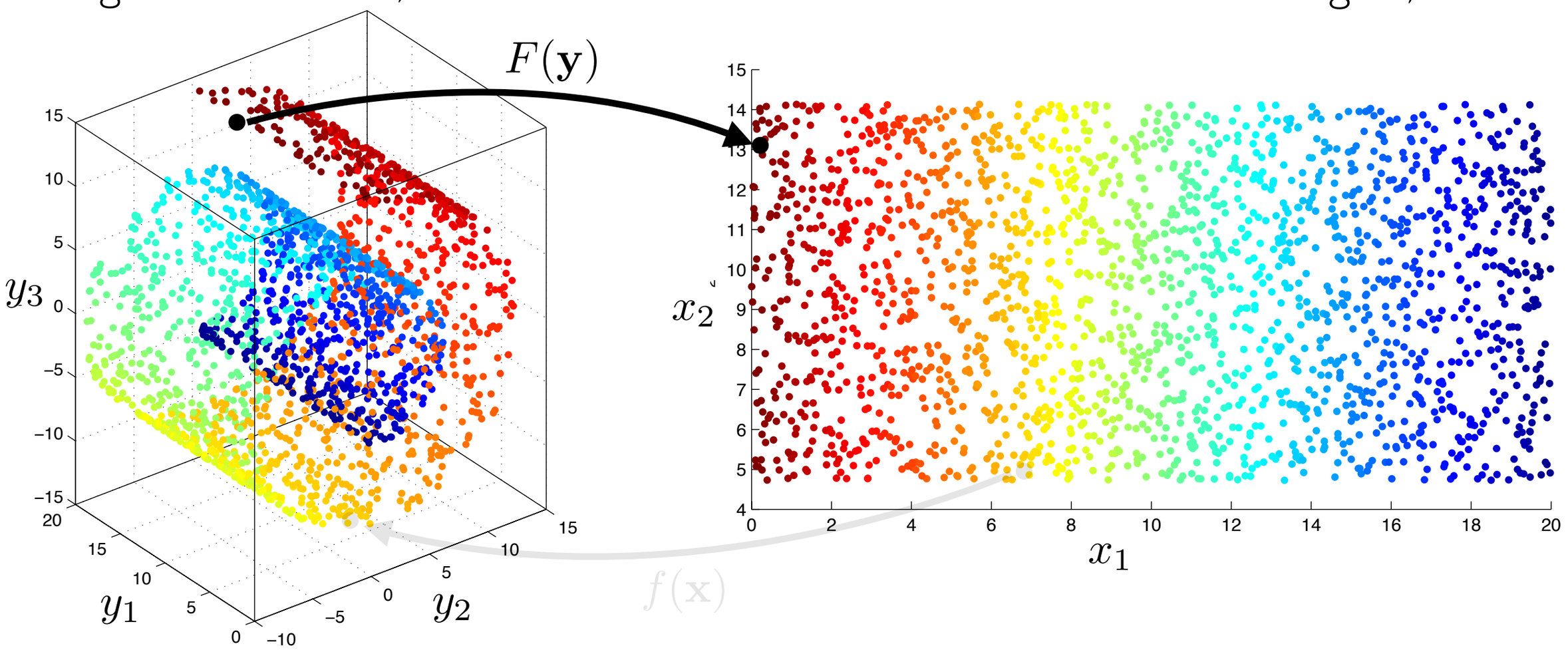
Dimensionality reduction

Given a high-dimensional dataset $\mathbf{Y}_{D\times N} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$ find

- projection points $\mathbf{X}_{d\times N} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$,
- reduction mapping $\mathbf{x} = F(\mathbf{y})$,
- reconstruction mapping $\mathbf{y} = f(\mathbf{x})$,
- joint probability density $p(\mathbf{x}, \mathbf{y})$,
- estimate intrinsic dimensionality d

Original dataset $\mathbf{Y}, D = 3$

Low-dimensional embedding $\mathbf{X}$, $d = 2$



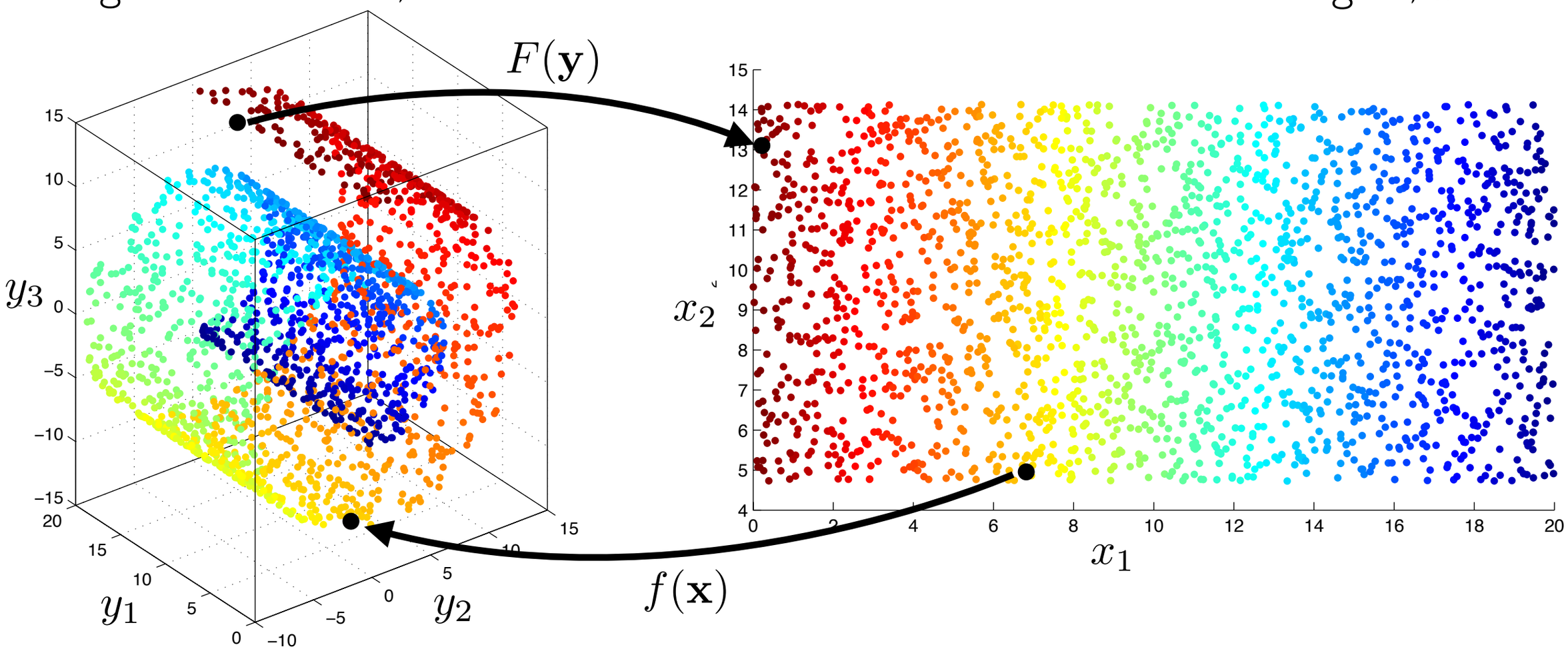
Dimensionality reduction

Given a high-dimensional dataset $\mathbf{Y}_{D\times N} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$ find

- projection points $\mathbf{X}_{d\times N} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$,
- reduction mapping $\mathbf{x} = F(\mathbf{y})$,
- reconstruction mapping $\mathbf{y} = f(\mathbf{x})$,
- joint probability density $p(\mathbf{x}, \mathbf{y})$,
- estimate intrinsic dimensionality d

Original dataset $\mathbf{Y}, D = 3$

Low-dimensional embedding $\mathbf{X}$, $d = 2$



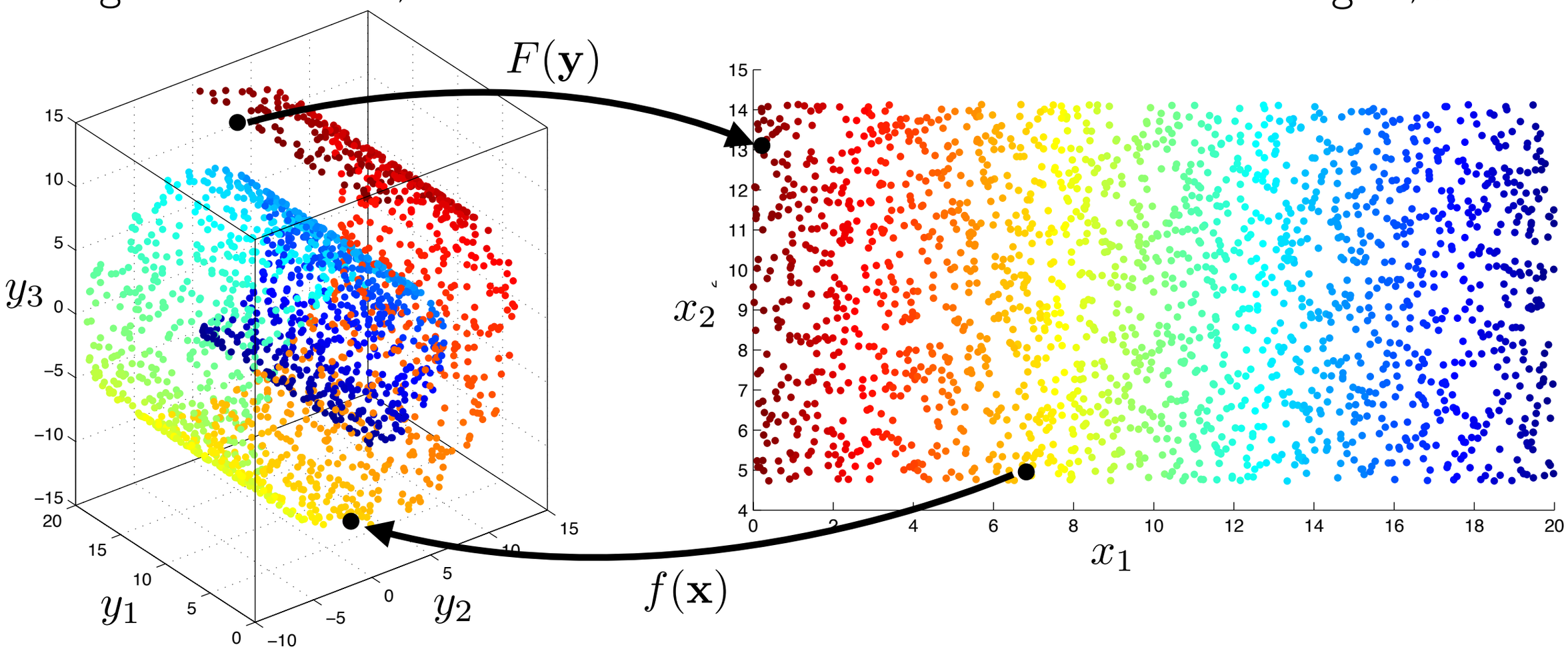
Dimensionality reduction

Given a high-dimensional dataset $\mathbf{Y}_{D\times N} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$ find

- projection points $\mathbf{X}_{d\times N} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$,
- reduction mapping $\mathbf{x} = F(\mathbf{y})$,
- reconstruction mapping $\mathbf{y} = f(\mathbf{x})$,
- joint probability density $p(\mathbf{x}, \mathbf{y})$,
- estimate intrinsic dimensionality d

Original dataset $\mathbf{Y}, D = 3$

Low-dimensional embedding $\mathbf{X}$, $d = 2$



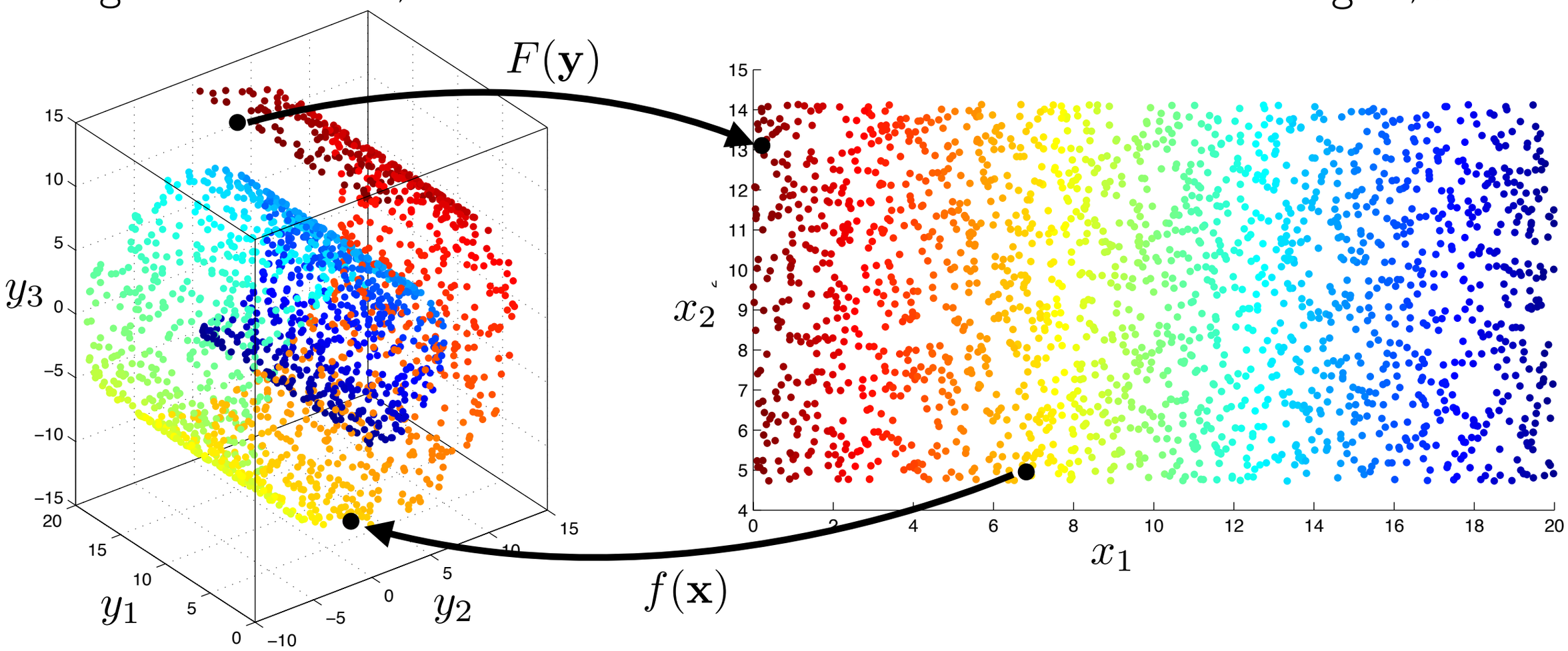
Dimensionality reduction

Given a high-dimensional dataset $\mathbf{Y}_{D\times N} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$ find

- projection points $\mathbf{X}_{d\times N} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$,
- reduction mapping $\mathbf{x} = F(\mathbf{y})$,
- reconstruction mapping $\mathbf{y} = f(\mathbf{x})$,
- joint probability density $p(\mathbf{x}, \mathbf{y})$,
- estimate intrinsic dimensionality d.

Original dataset $\mathbf{Y}, D = 3$

Low-dimensional embedding $\mathbf{X}$, $d = 2$



Dimensionality reduction

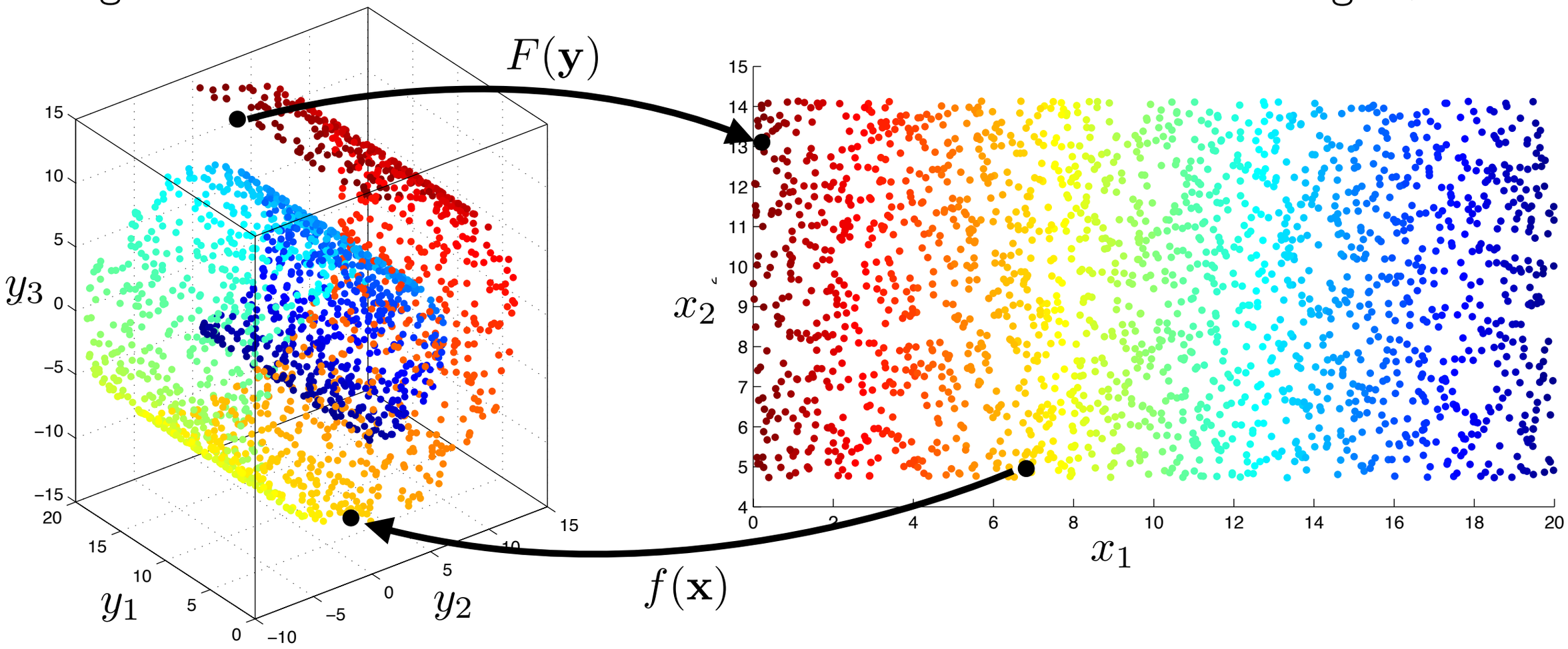
Given a high-dimensional dataset $\mathbf{Y}_{D\times N} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$ find

- projection points $\mathbf{X}_{d\times N} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$,
- reduction mapping $\mathbf{x} = F(\mathbf{y})$,
- reconstruction mapping $\mathbf{y} = f(\mathbf{x})$,
- joint probability density $p(\mathbf{x}, \mathbf{y})$,
- estimate intrinsic dimensionality d

this talk

Original dataset $\mathbf{Y}, D = 3$

Low-dimensional embedding $\mathbf{X}$, $d = 2$



Dimensionality reduction

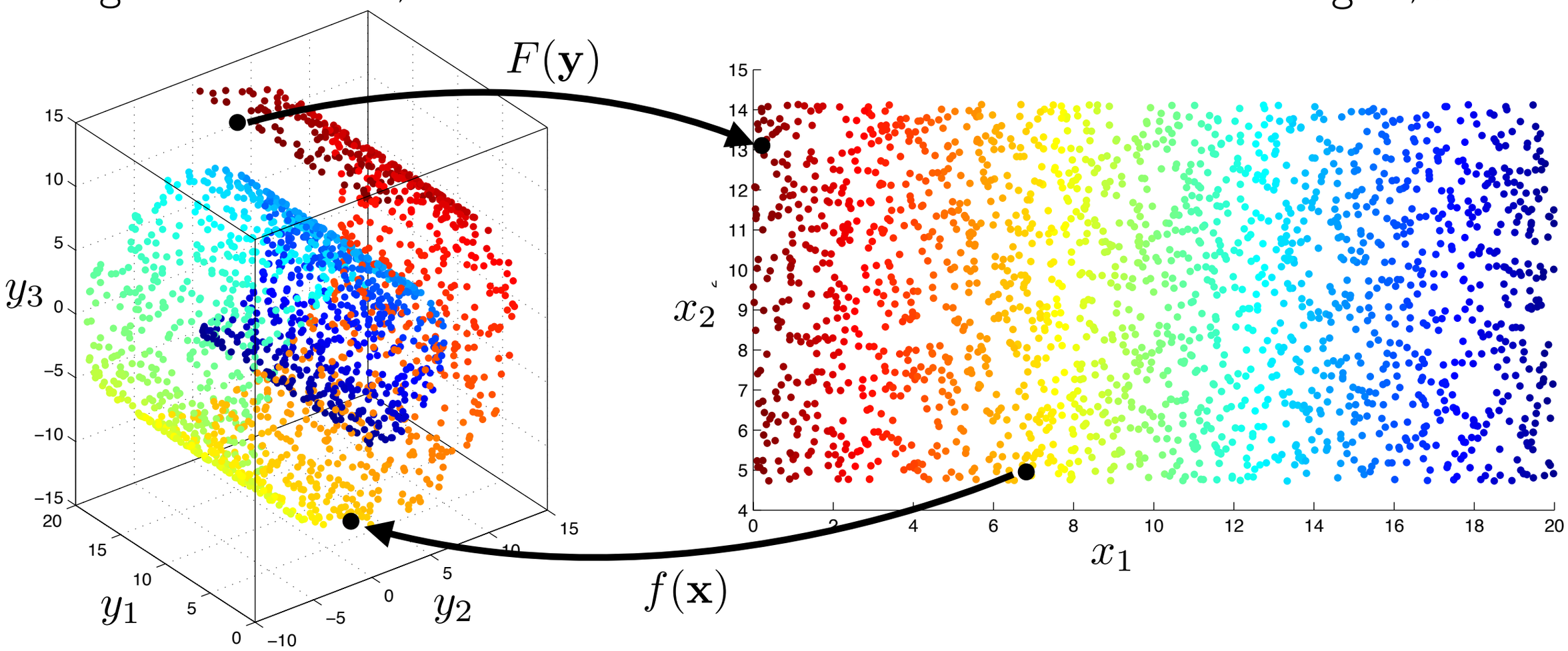
Given a high-dimensional dataset $\mathbf{Y}_{D\times N} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$ find

- projection points $\mathbf{X}_{d\times N} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$,
- reduction mapping $\mathbf{x} = F(\mathbf{y})$,
- reconstruction mapping $\mathbf{y} = f(\mathbf{x})$,
- joint probability density $p(\mathbf{x}, \mathbf{y})$,
- estimate intrinsic dimensionality d

this talk

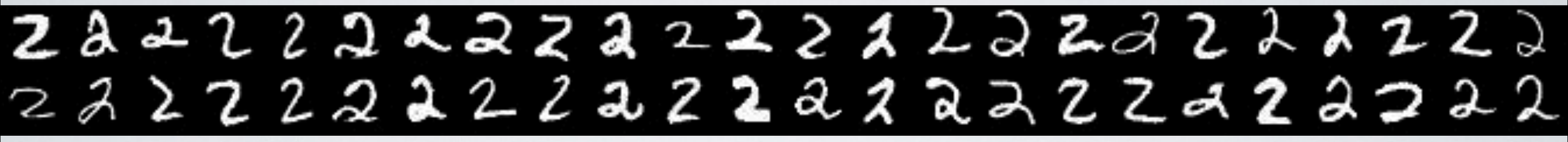
Original dataset $\mathbf{Y}, D = 3$

Low-dimensional embedding $\mathbf{X}$, $d = 2$



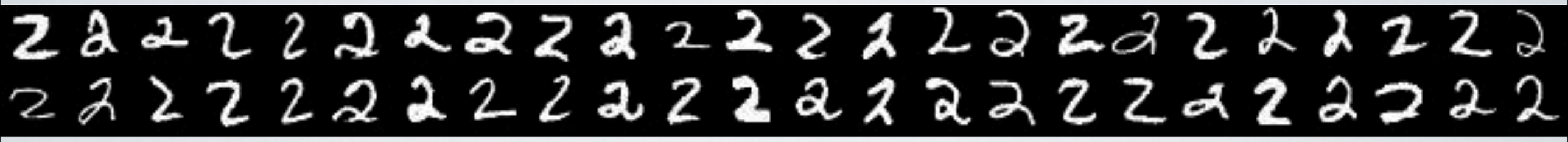
MNIST Handwritten digits

Consider a dataset with $1\,000$ handwritten digits 2 :



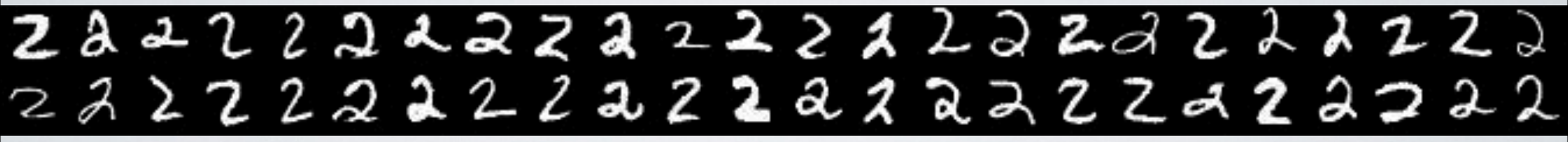
MNIST Handwritten digits

Consider a dataset with $1\,000$ handwritten digits 2 :



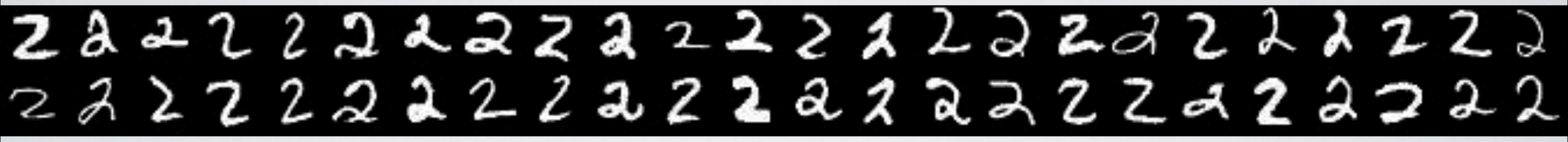
MNIST Handwritten digits

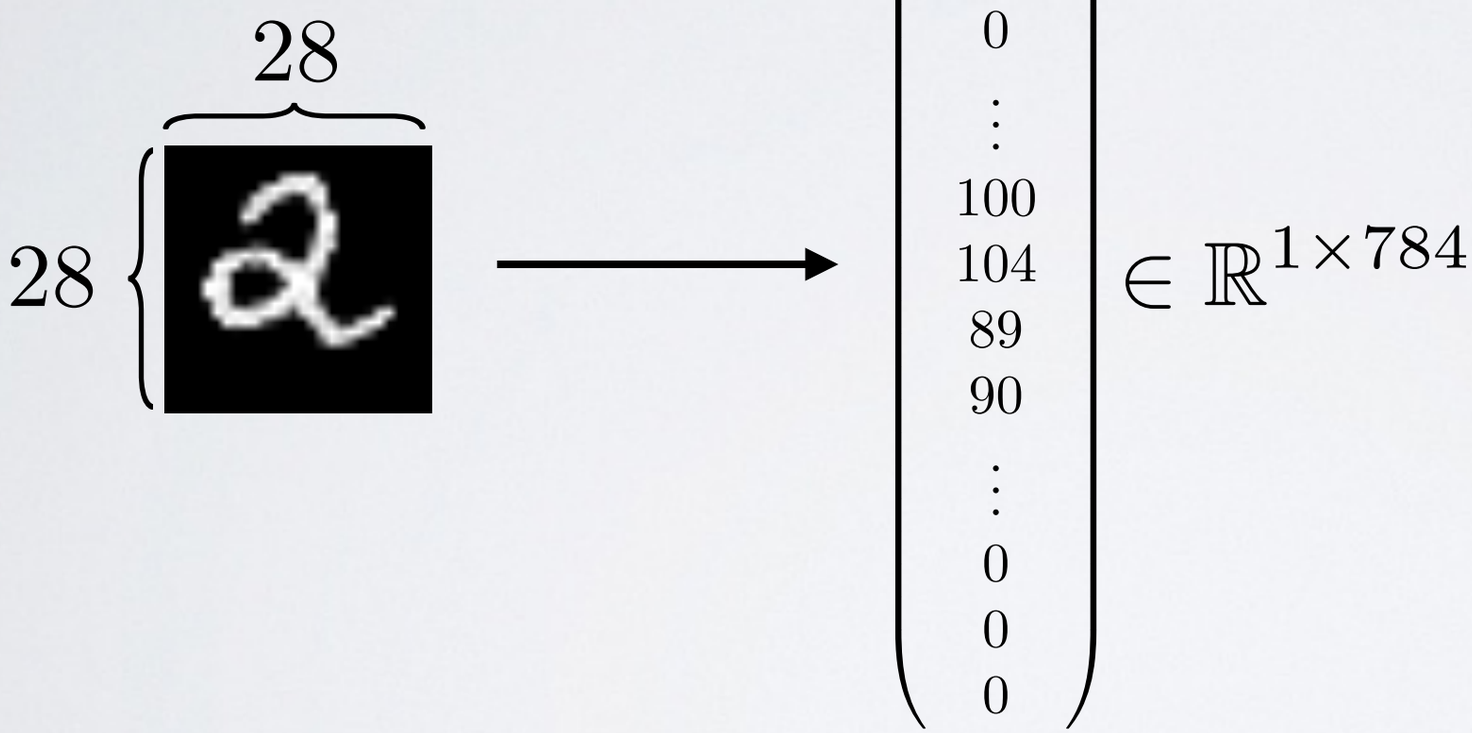
Consider a dataset with $1\,000$ handwritten digits 2 :



MNIST Handwritten digits

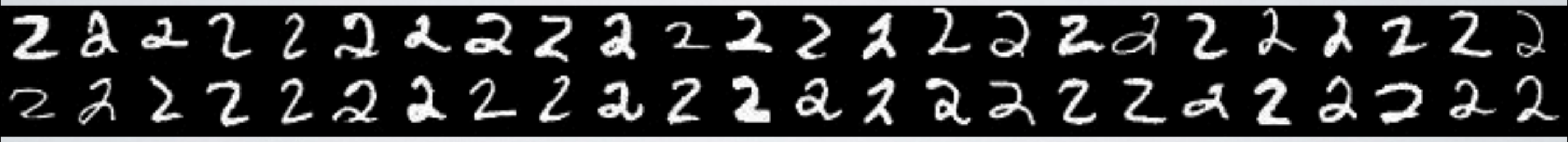
Consider a dataset with $1\,000$ handwritten digits 2 :

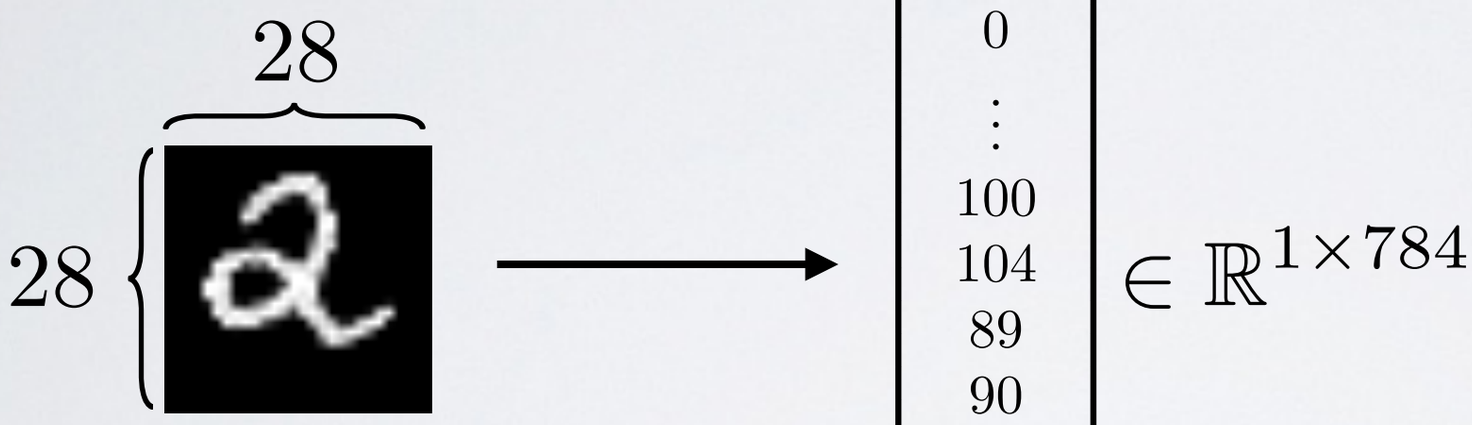




MNIST Handwritten digits

Consider a dataset with $1\,000$ handwritten digits 2 :





$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 100 \\ 104 \\ 89 \\ 90 \\ \vdots \\ 0 \\ 0 \\ 0 \end{pmatrix} \in \mathbb{R}^{1\times 784}$$

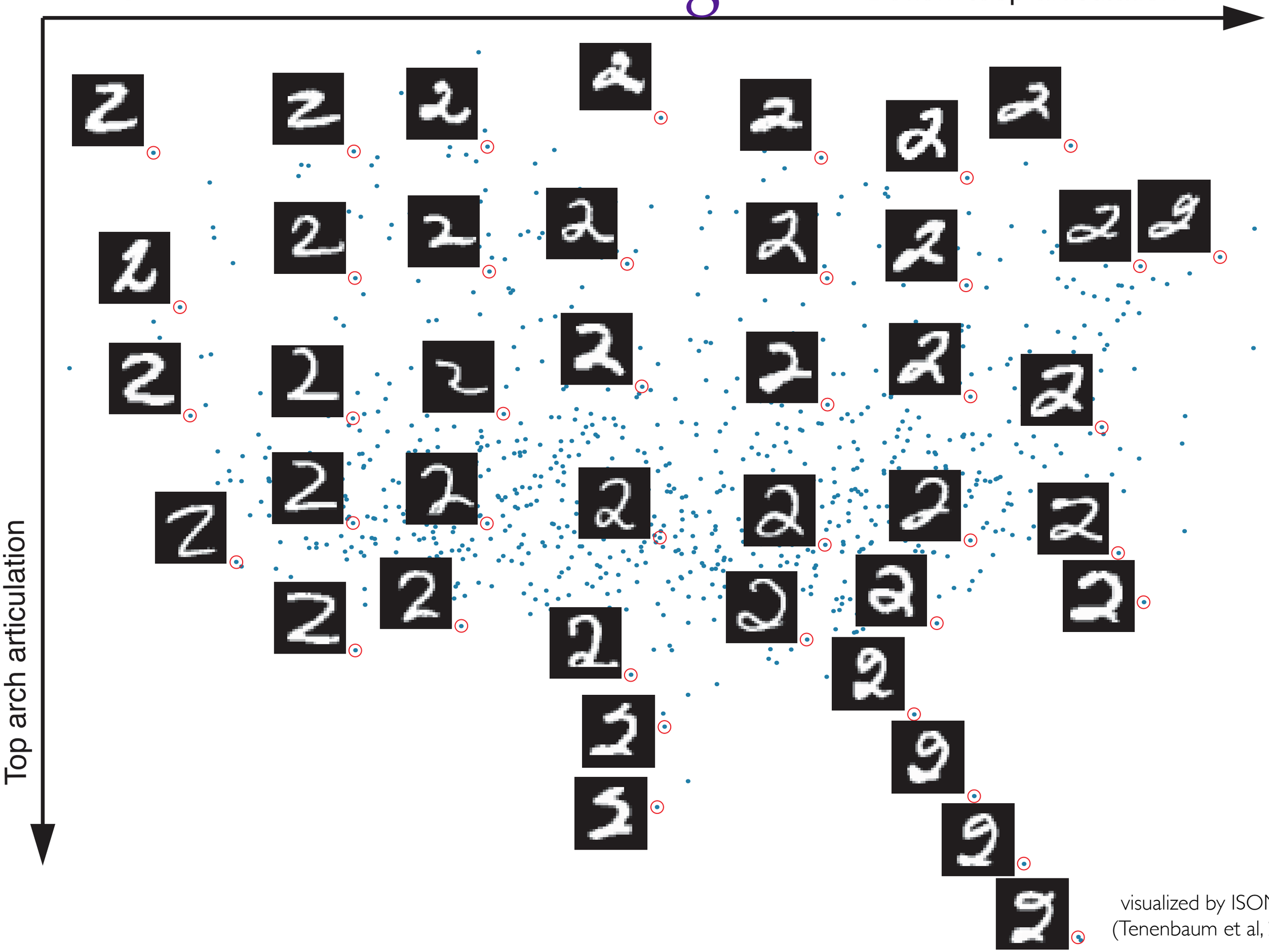
High-dimensional dataset: $\mathbf{Y} \in \mathbb{R}^{1\,000\times 784}$

Number of points: $N = 1\,000$

Number of dimensions: $D = 784$

Reduction space: $d = 2$

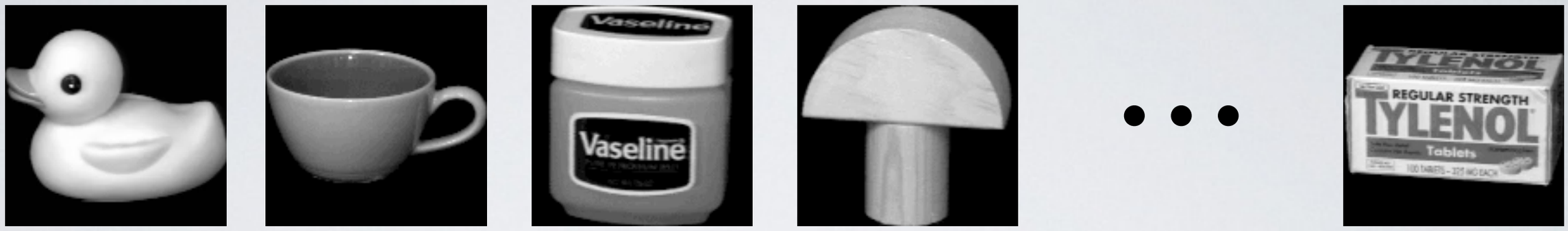
MNIST Handwritten digits



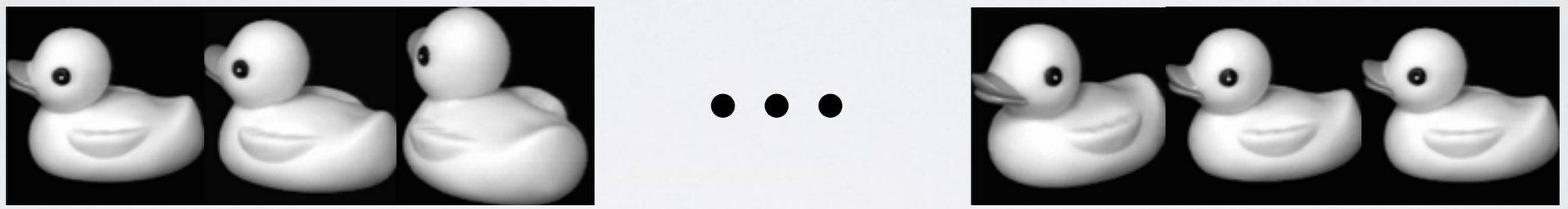
visualized by ISOMAP
(Tenenbaum et al, 2000)

COIL-20 Rotational sequences

10 objects:



72 images per object:



128

128

$\downarrow$

$\mathbb{R}^{1\times 16\,384}$

High-dimensional dataset: $\mathbf{Y} \in \mathbb{R}^{720\times 16\,384}$

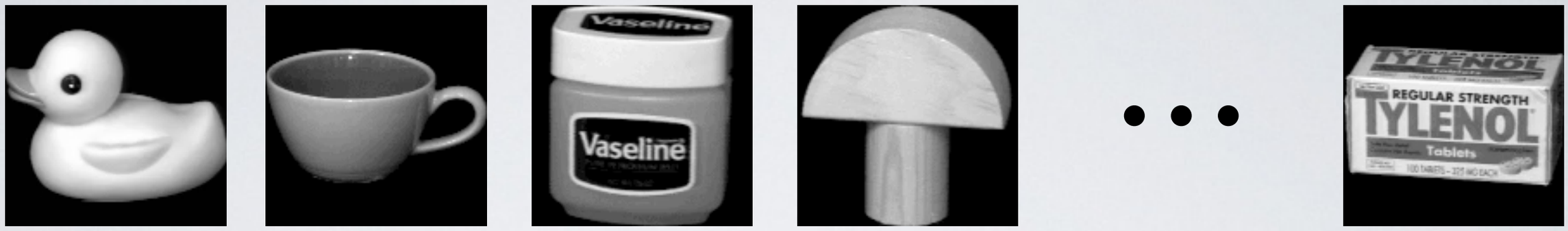
Number of points: $N = 720$

Number of dimensions: $D = 16\,384$

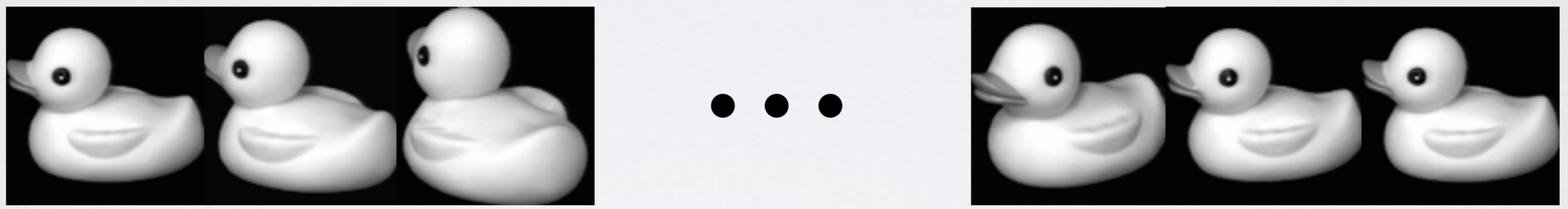
Reduction space: $d = 2$

COIL-20 Rotational sequences

10 objects:



72 images per object:



128

128

$\downarrow$

$\mathbb{R}^{1\times 16\,384}$

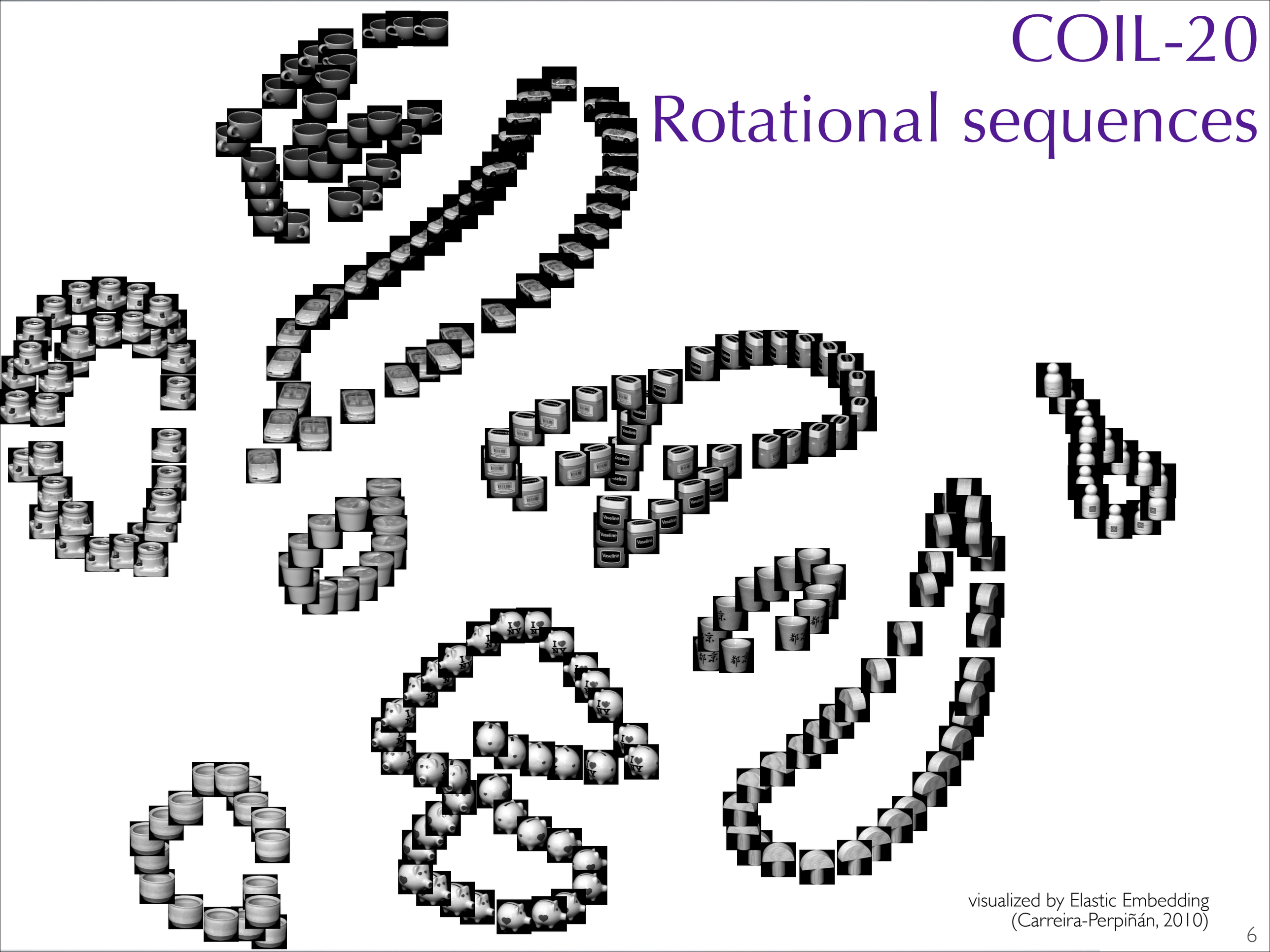
High-dimensional dataset: $\mathbf{Y} \in \mathbb{R}^{720\times 16\,384}$

Number of points: $N = 720$

Number of dimensions: $D = 16\,384$

Reduction space: $d = 2$

COIL-20
Rotational sequences



visualized by Elastic Embedding
(Carreira-Perpiñán, 2010)

Other use of dimensionality reduction

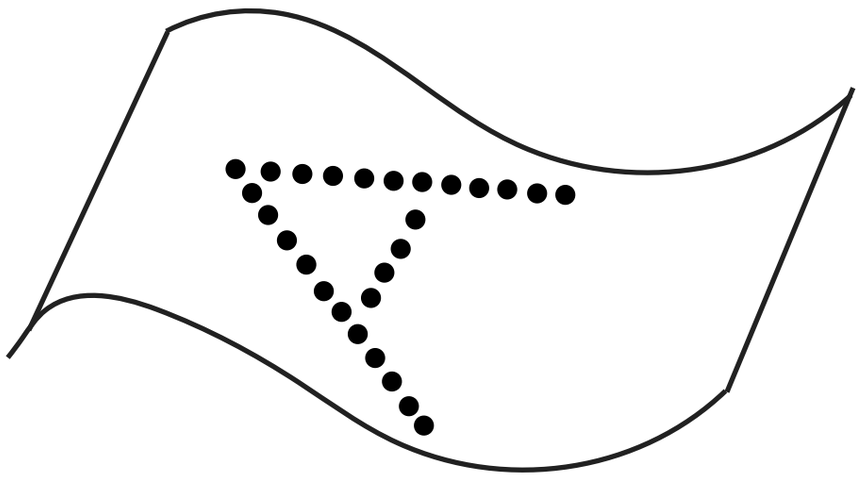
- Preprocessing before other task e.g. classification or regression:
  - denoising,
  - decreasing the complexity.
- Extracting latent structure of the data:
  - feature learning,
  - cluster information,
  - deep networks with autoencoders.

Dimensionality reduction

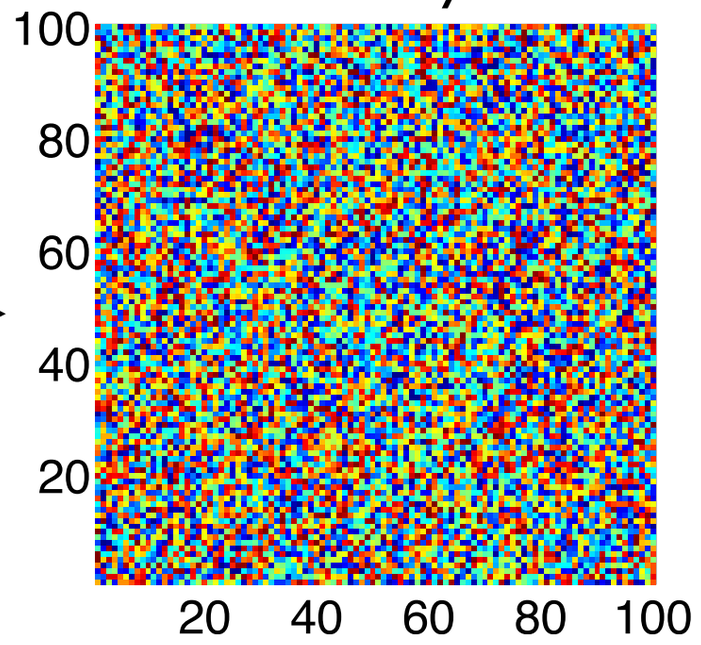
Given high-dimensional data points $\mathbf{Y}_{D\times N} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$.

1. Convert data points to a $N \times N$ *affinity* matrix $\mathbf{A}$.
2. Find low-dimensional coordinates $\mathbf{X}_{d\times N} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$, so that their similarity is as close as possible to $\mathbf{A}$.

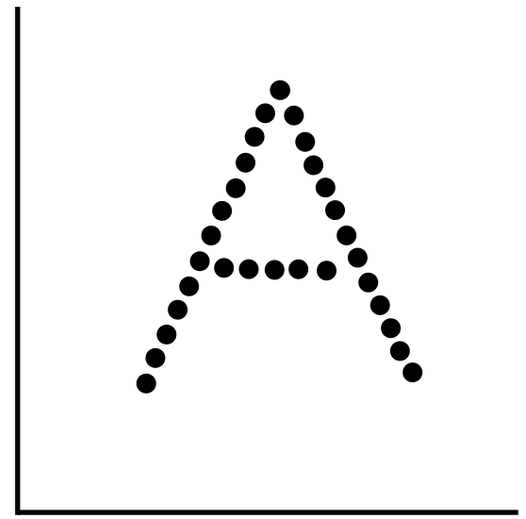
High-dimensional input $\mathbf{Y}$



Affinity $\mathbf{A}$



Low-dimensional output $\mathbf{X}$

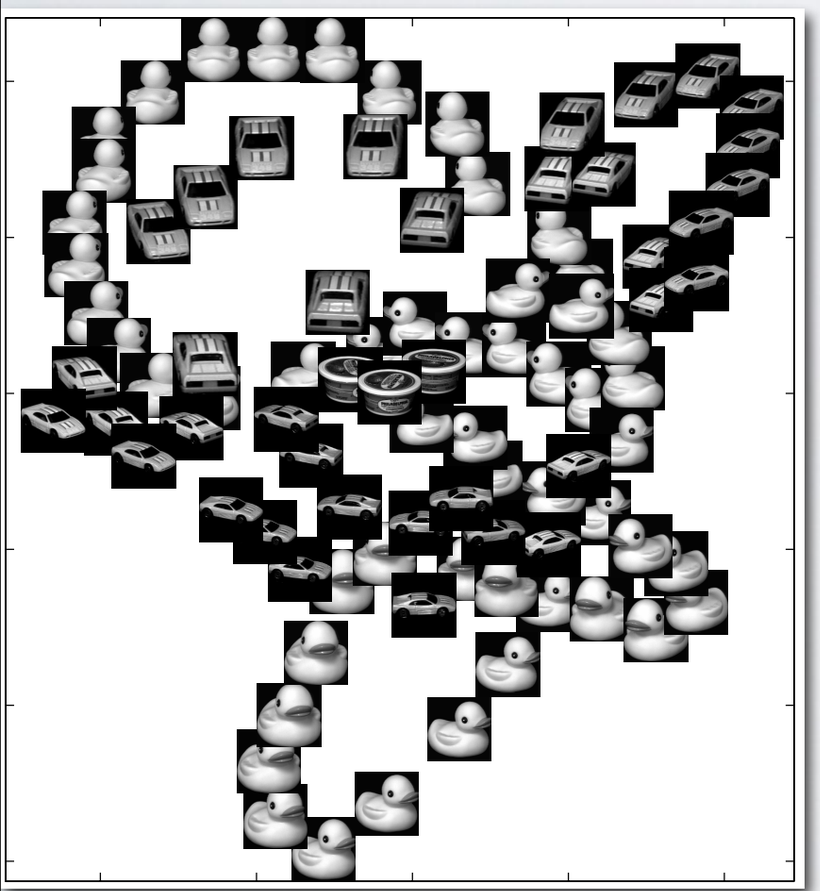


Classification of dimensionality reduction

- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.

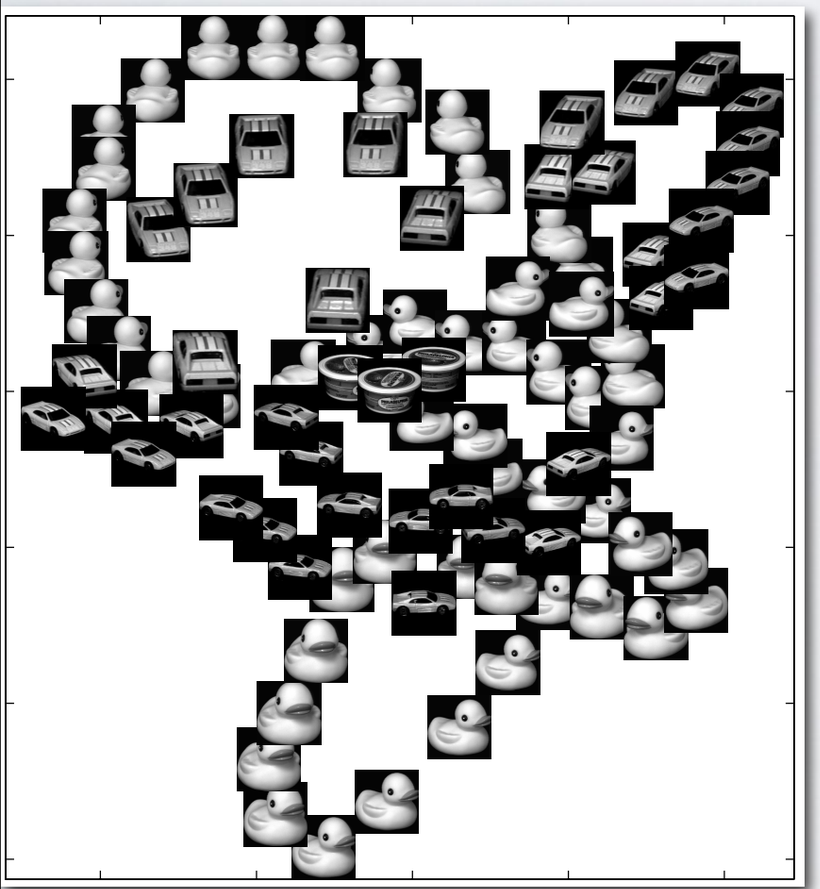
Classification of dimensionality reduction

- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.



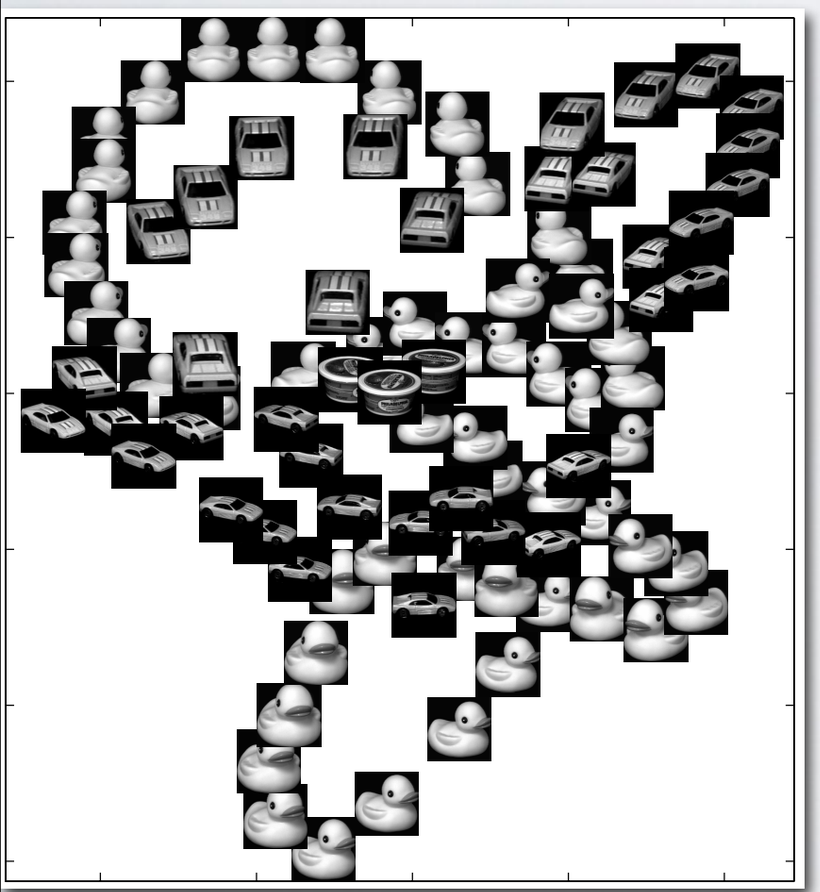
Classification of dimensionality reduction

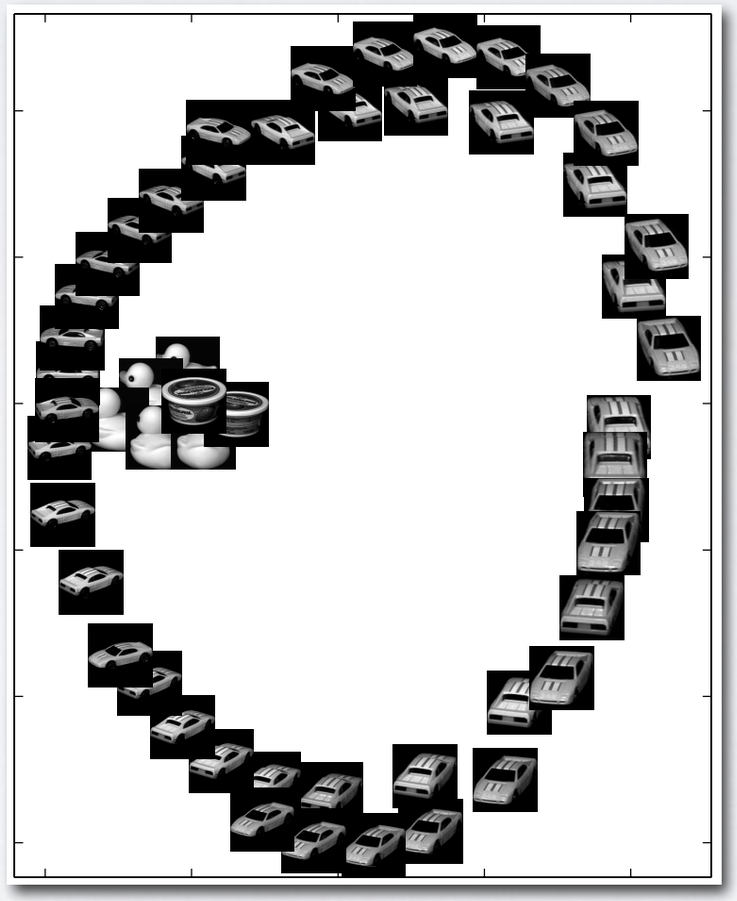
- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.
- Spectral methods
  - Laplacian Eigenmaps,
  - ISOMAP,
  - Locally Linear Embedding (LLE),
  - etc.



Classification of dimensionality reduction

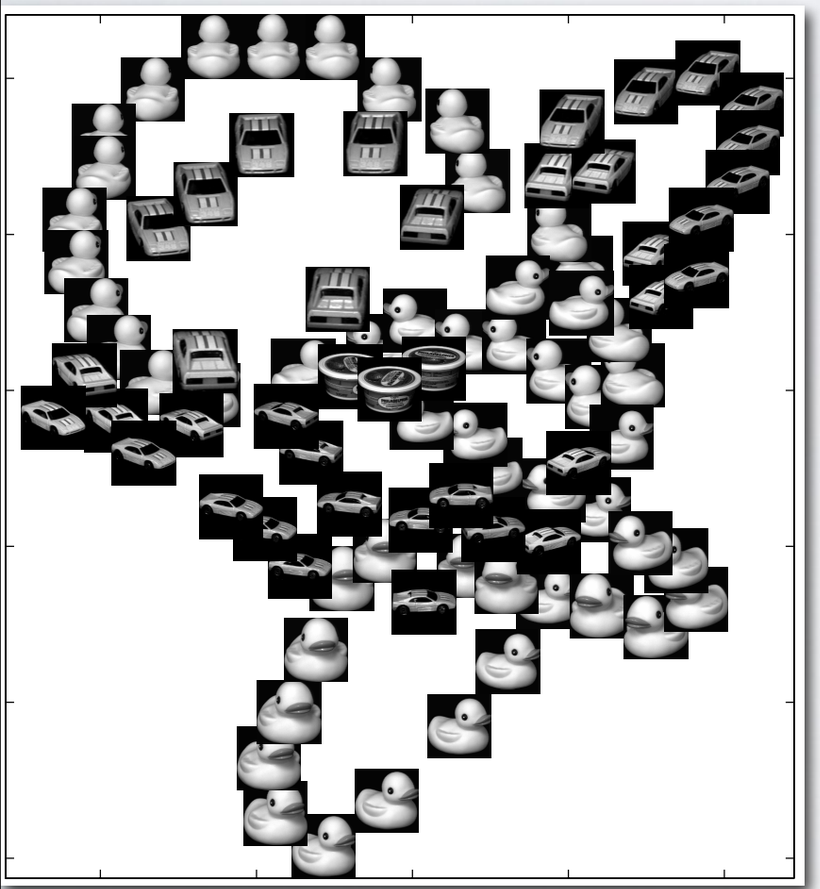
- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.
- Spectral methods
  - Laplacian Eigenmaps,
  - ISOMAP,
  - Locally Linear Embedding (LLE),
  - etc.

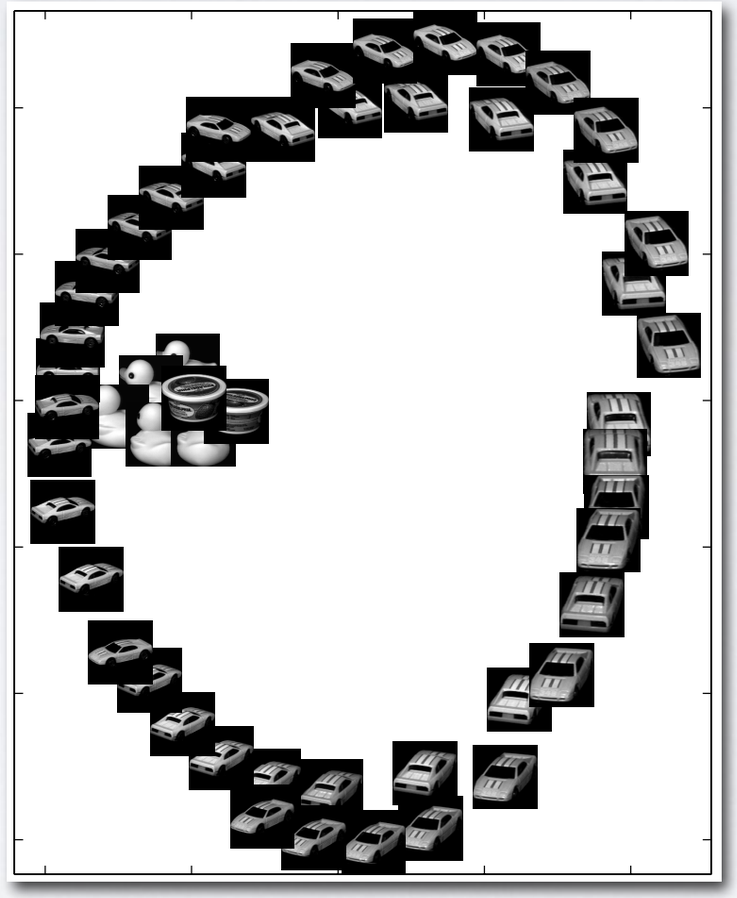




Classification of dimensionality reduction

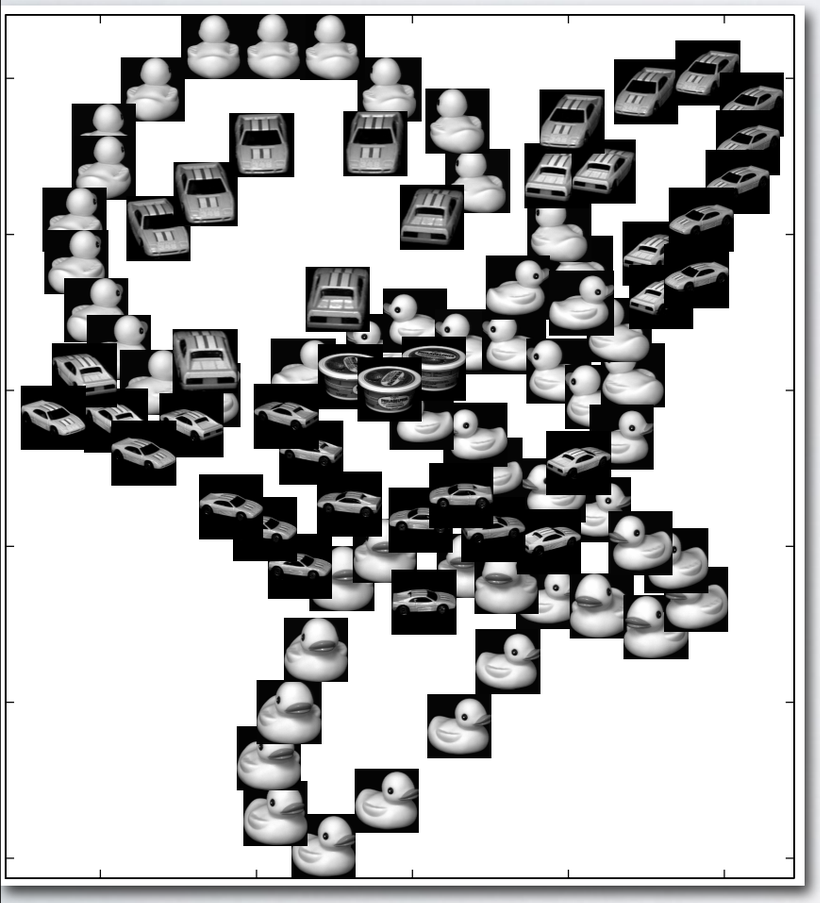
- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.
- Spectral methods
  - Laplacian Eigenmaps,
  - ISOMAP,
  - Locally Linear Embedding (LLE),
  - etc.
- Nonlinear embedding methods
  - Stochastic Neighbor Embedding,
  - *t*-SNE,
  - The Elastic Embedding (EE),
  - etc.

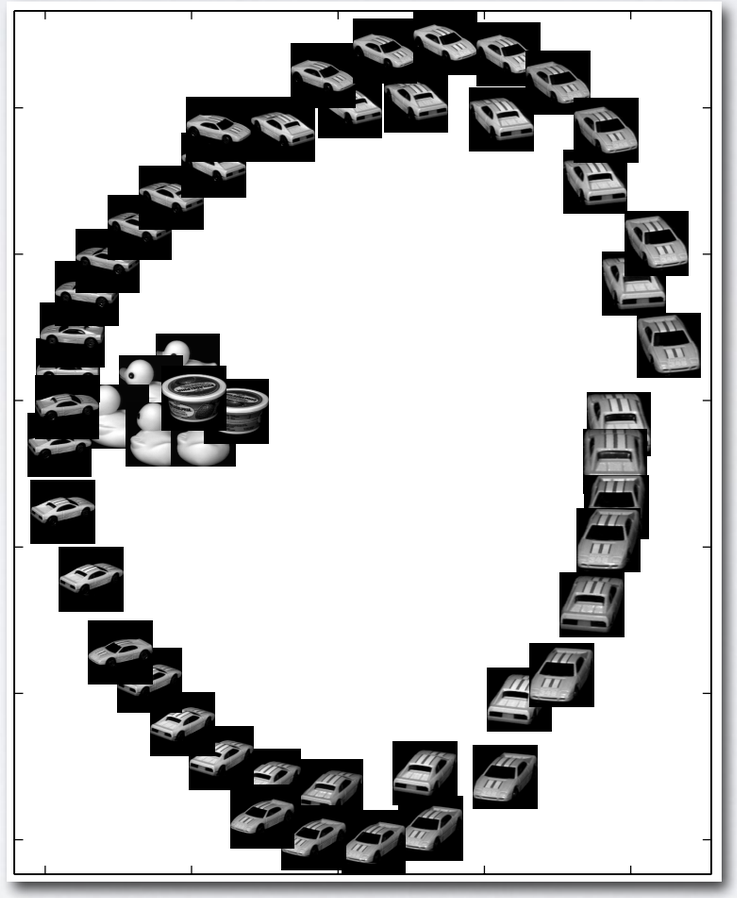


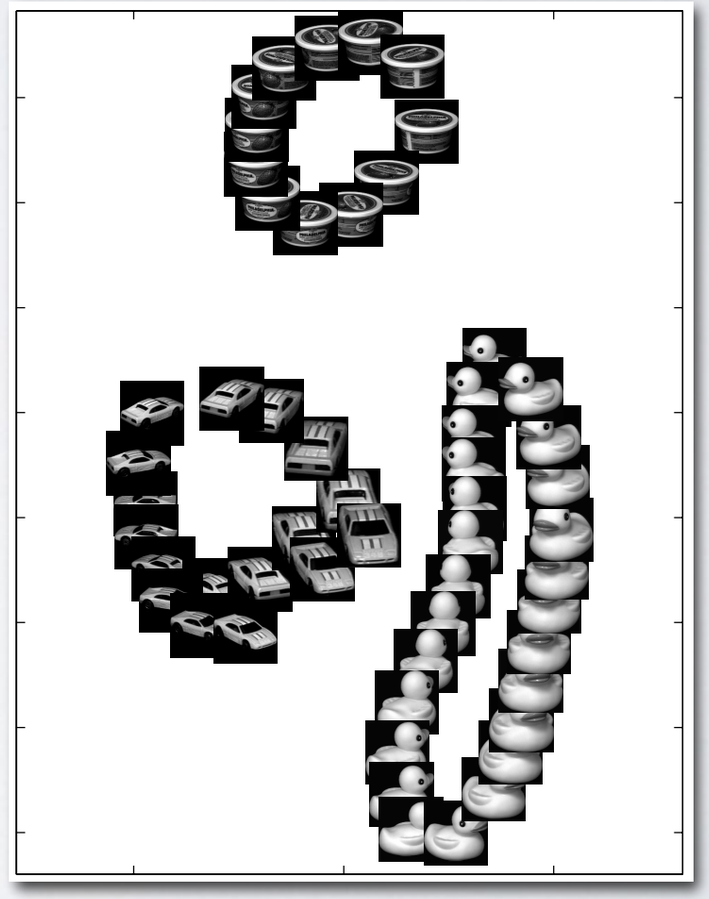


Classification of dimensionality reduction

- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.
- Spectral methods
  - Laplacian Eigenmaps,
  - ISOMAP,
  - Locally Linear Embedding (LLE),
  - etc.
- Nonlinear embedding methods
  - Stochastic Neighbor Embedding,
  - *t*-SNE,
  - The Elastic Embedding (EE),
  - etc.

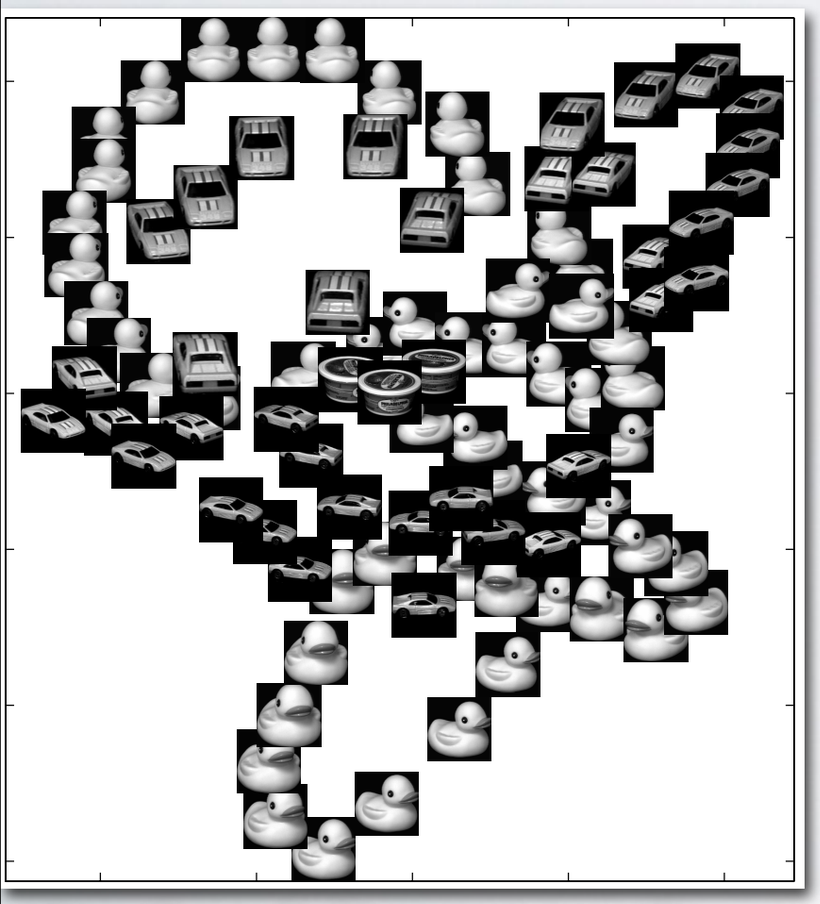


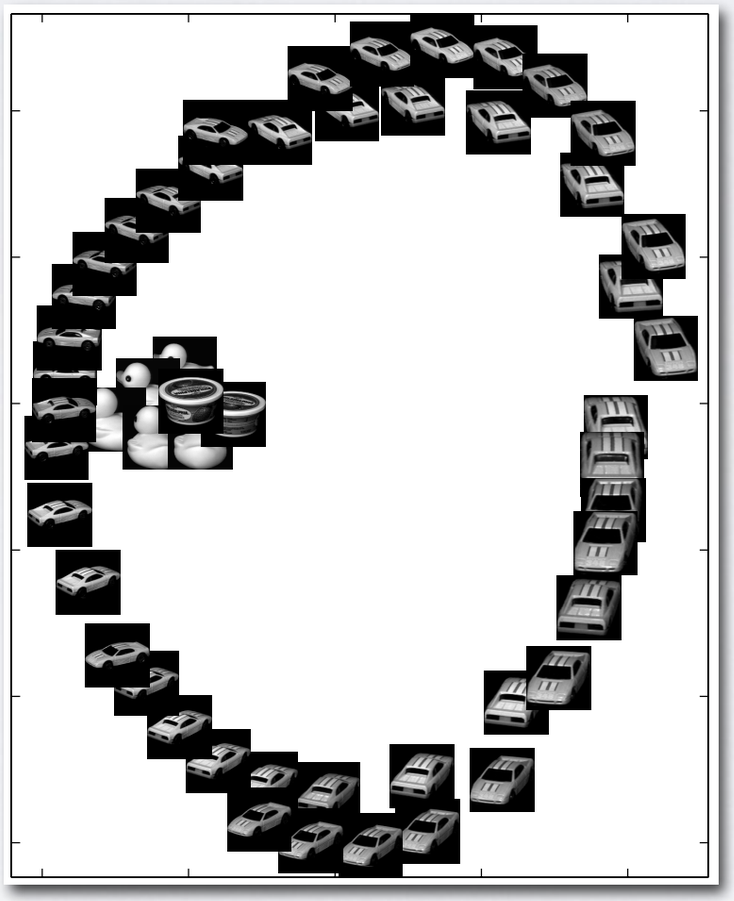


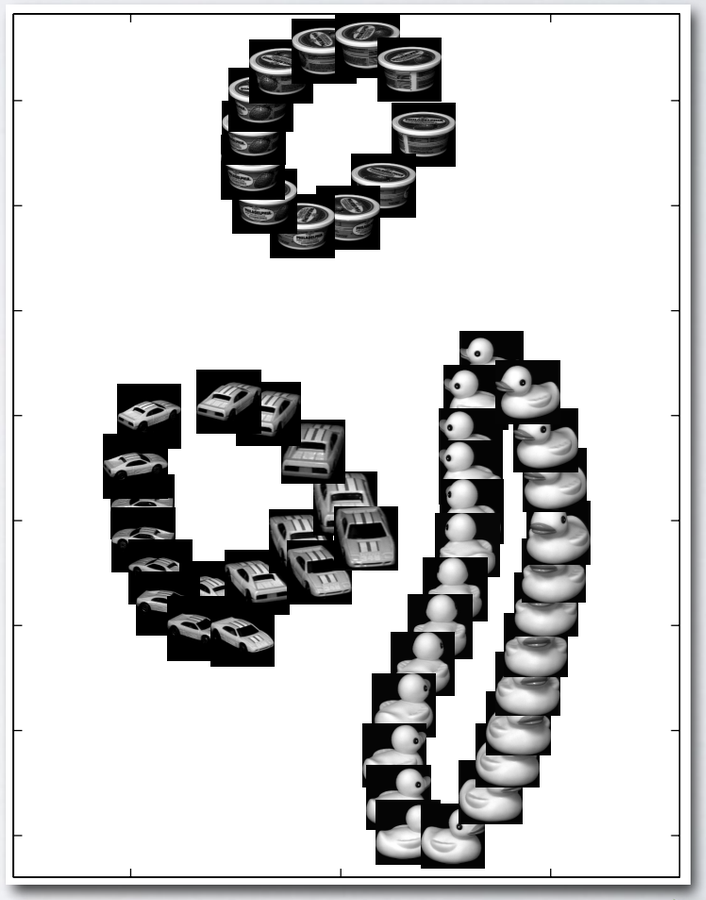


Classification of dimensionality reduction

- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.
- Spectral methods
  - Laplacian Eigenmaps,
  - ISOMAP,
  - Locally Linear Embedding (LLE),
  - etc.
- Nonlinear embedding methods
  - Stochastic Neighbor Embedding,
  - *t*-SNE,
  - The Elastic Embedding (EE),
  - etc.



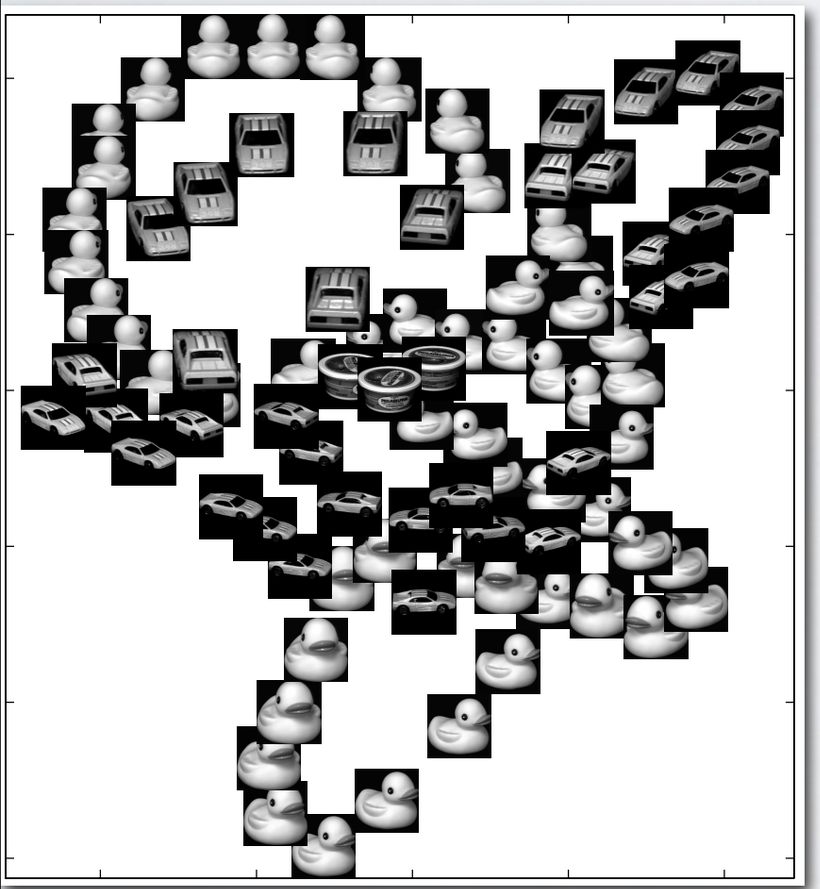


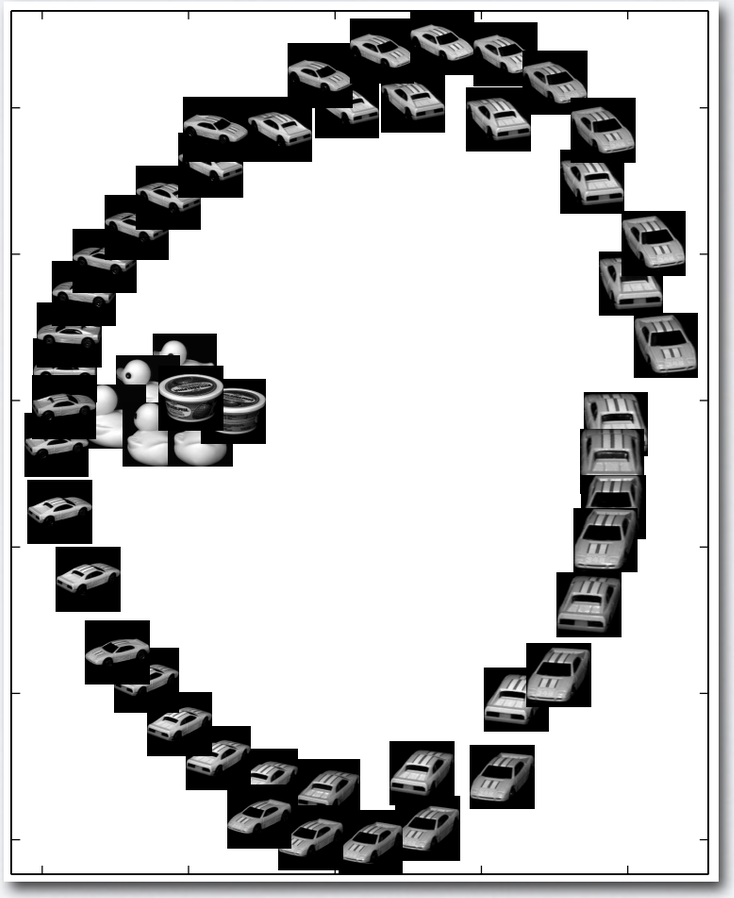


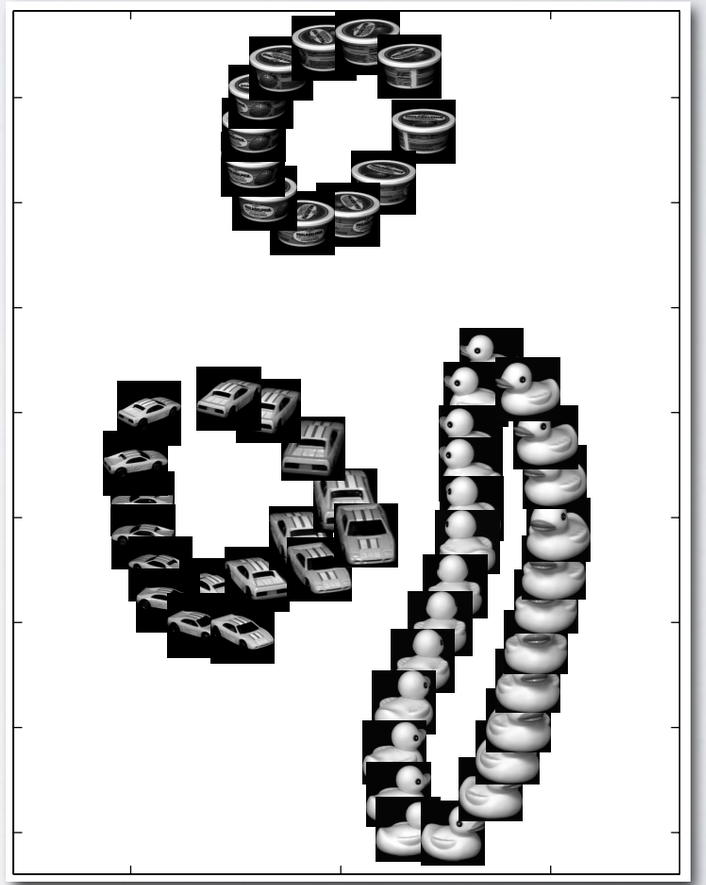
Embedding quality

Classification of dimensionality reduction

- Linear methods
  - principal component analysis (PCA),
  - classical multidimensional scaling (MDS).
  - etc.
- Spectral methods
  - Laplacian Eigenmaps,
  - ISOMAP,
  - Locally Linear Embedding (LLE),
  - etc.
- Nonlinear embedding methods
  - Stochastic Neighbor Embedding,
  - *t*-SNE,
  - The Elastic Embedding (EE),
  - etc.







Embedding quality

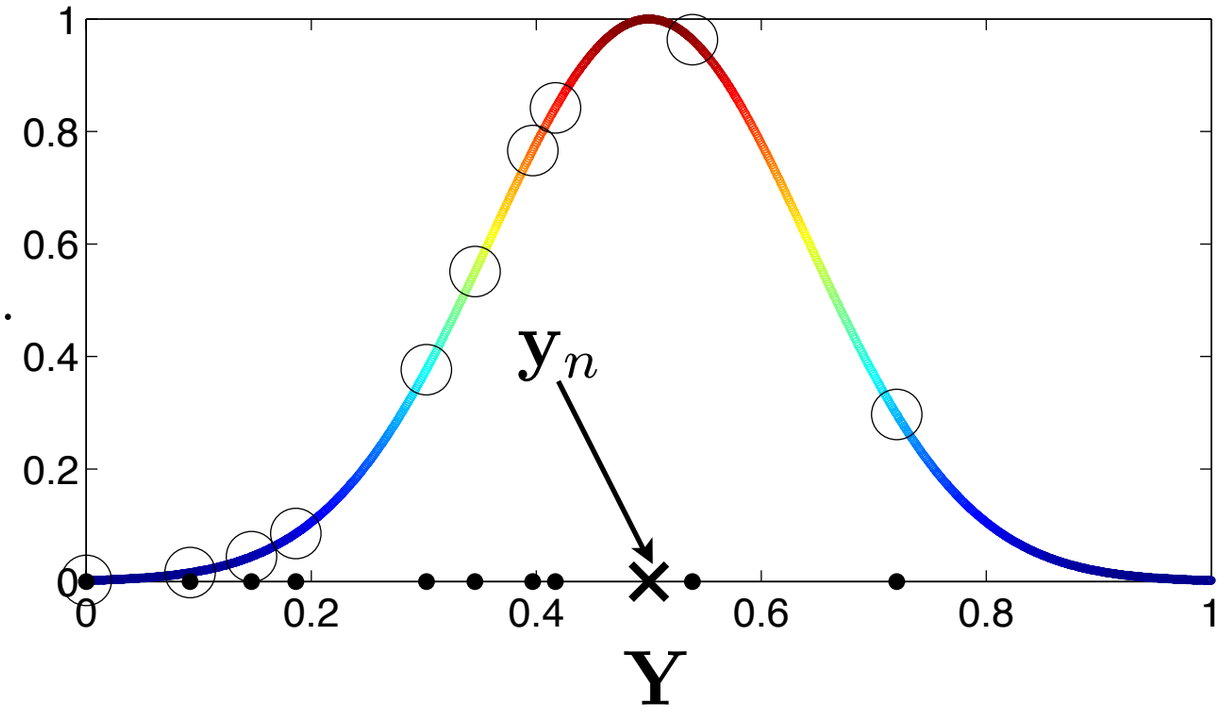
Runtime

Affinity matrix

- Affinity matrix $W \in \mathbb{R}^{N \times N}$ represents the similarities between points int he dataset. The higher the affinity value, the more similar are the points to each other.

- Intuition:
  - high weight to nearby points,
  - low weight to far away points.

w_n.



- Property:
  - affinity matrix enforces *locality* of the data.

- For example, Gaussian affinities are given by:

$$w_{nm} = \exp(-\frac{1}{2} \left\| (\mathbf{y}_n - \mathbf{y}_m)/\sigma \right\|^2)$$

Nonlinear Embedding (NLE) methods

Many of well-known methods can be written in the form:

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$

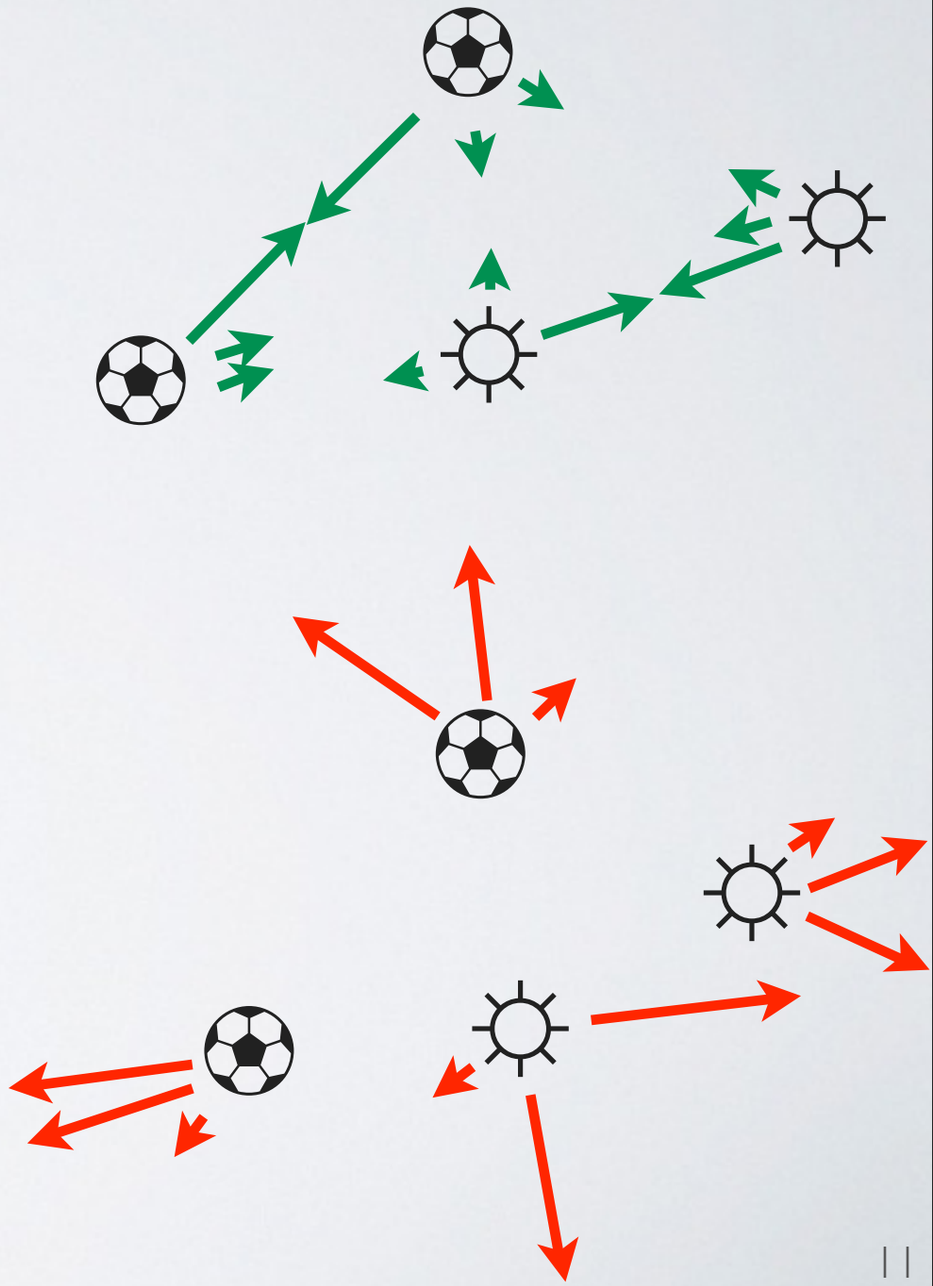
$E^+(\mathbf{X})$ is an *attractive* term:

- often quadratic,
- minimal with coincident points,
- defined usually on the sparse affinity (not all interactions are computed).

$E^-(\mathbf{X})$ is a *repulsive* term:

- often very nonlinear,
- minimal with points separated infinitely,
- all interactions should be computed.

Optimal embedding balances both forces.



NLE: Examples

- Laplacian Eigenmaps: (Belkin and Niyogi, '03)

$$E_{LE}(\mathbf{X}) = \sum_{n,m=1}^{N} w_{nm} \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2 \quad \text{s.t. translation and scale constraints}$$

- Stochastic neighbor embedding: (Hinton and Roweis, '03)

$$E_{SNE}(\mathbf{X}) = \sum_{n,m=1}^{N} p_{nm} \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2 + \sum_{n=1}^{N} \log \sum_{m \neq n}^{N} \exp(-\left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2)$$

- Symmetric stochastic neighbor embedding: (Cook et al, '07)

$$E_{s\text{-}SNE}(\mathbf{X}) = \sum_{n,m=1}^{N} p_{nm} \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2 + \log \sum_{n,m=1}^{N} \exp(-\left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2)$$

- t-SNE: (van der Maaten and Hinton '08)

$$E_{t\text{-}SNE}(\mathbf{X}) = \sum_{n,m=1}^{N} p_{nm} \log(1 + \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2) + \sum_{n,m=1}^{N} (1 + \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2)^{-1}$$

- The Elastic Embedding: (Carreira-Perpiñán, '10)

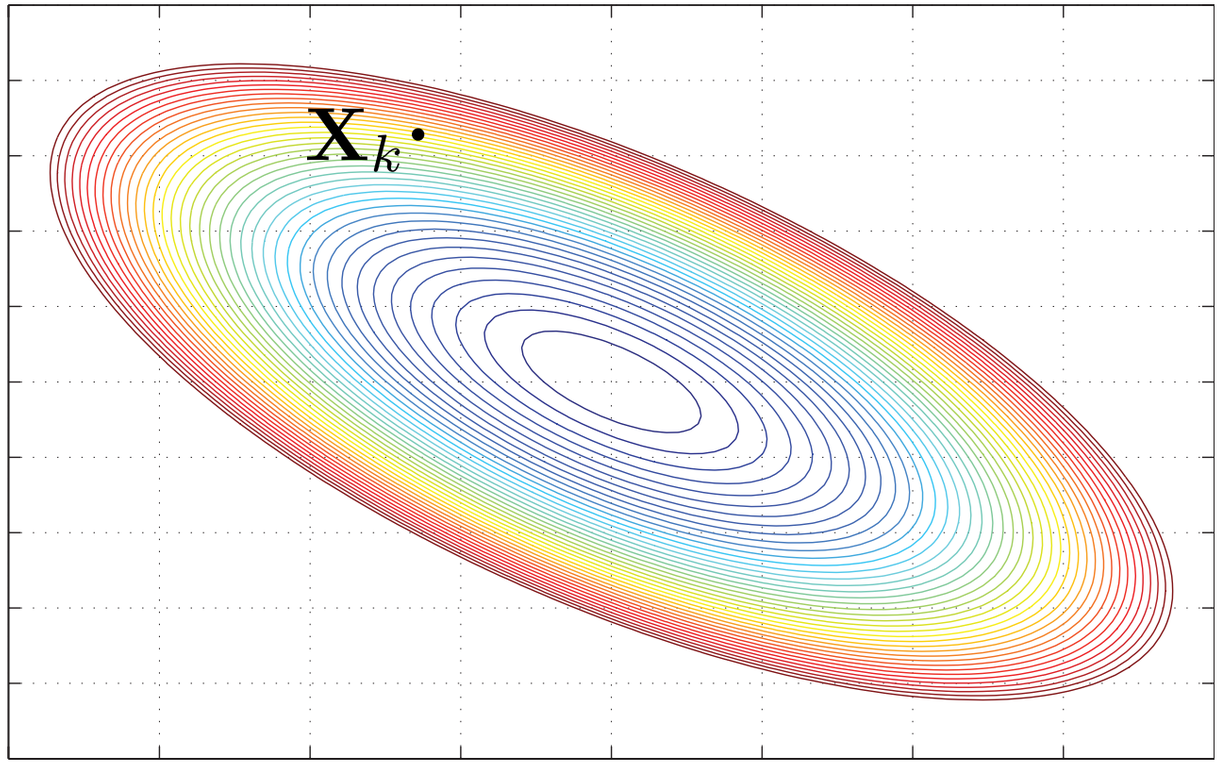
$$E_{EE}(\mathbf{X}) = \sum_{n,m=1}^{N} w_{nm}^{+} \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2 + \lambda \sum_{n,m=1}^{N} w_{nm}^{-} \exp(-\left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2)$$

NLE: Optimization

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^{+}(\mathbf{X}) + \lambda E^{-}(\mathbf{X}) \quad \lambda \geq 0$$

- Optimization:
  - compute the gradient
    $\mathbf{G}_k = 4\mathbf{X}_k(\mathbf{L}^{+} - \lambda\widetilde{\mathbf{L}})$
  - compute the direction. For example, gradient descent:
    $\mathbf{P}_k = -\mathbf{G}_k$
  - compute new iteration $\mathbf{X}_{k+1}$ using a line search:
    $\mathbf{X}_{k+1} = \mathbf{X}_k + \eta\mathbf{P}_k$
  - repeat till convergence.



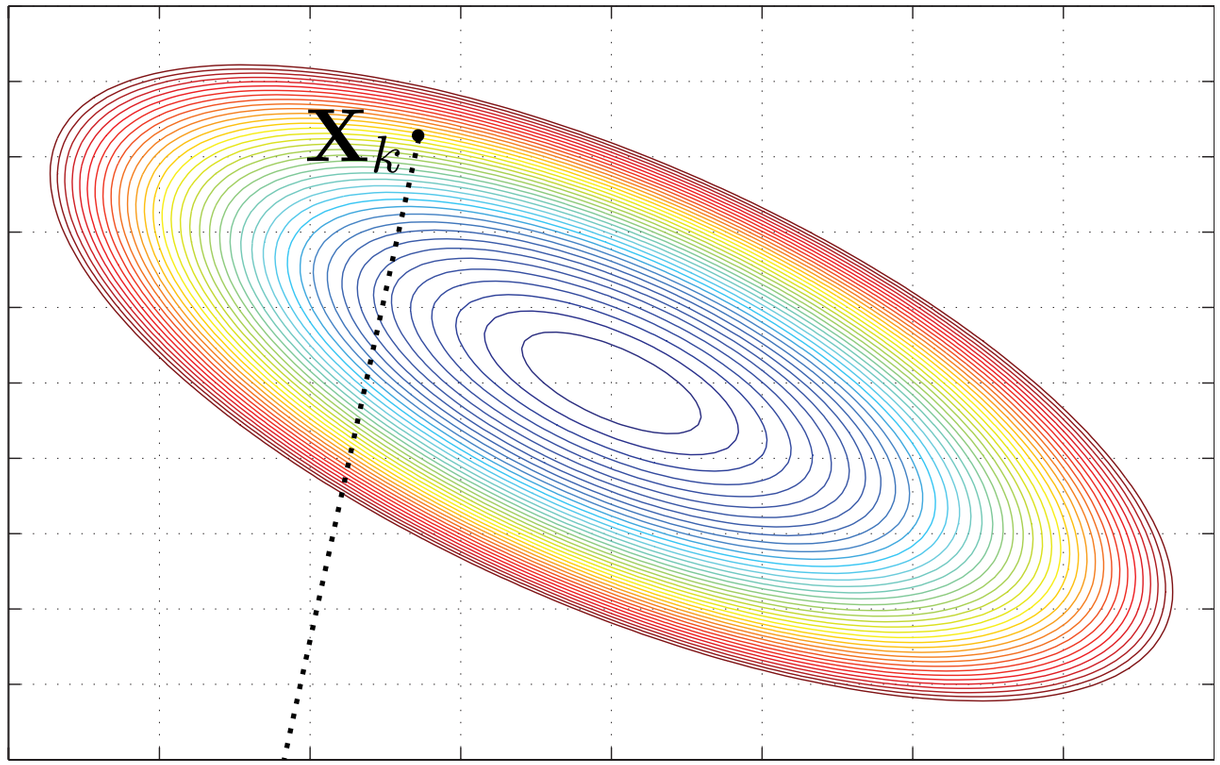
- Other gradient-based optimization methods are applicable: L-BFGS, Conjugate Gradient, etc..

NLE: Optimization

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^{+}(\mathbf{X}) + \lambda E^{-}(\mathbf{X}) \quad \lambda \geq 0$$

- Optimization:
  - compute the gradient
    $\mathbf{G}_k = 4\mathbf{X}_k(\mathbf{L}^{+} - \lambda\widetilde{\mathbf{L}})$
  - compute the direction. For example, gradient descent:
    $\mathbf{P}_k = -\mathbf{G}_k$
  - compute new iteration $\mathbf{X}_{k+1}$ using a line search:
    $\mathbf{X}_{k+1} = \mathbf{X}_k + \eta\mathbf{P}_k$
  - repeat till convergence.



- Other gradient-based optimization methods are applicable: L-BFGS, Conjugate Gradient, etc..

NLE: Optimization

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^{+}(\mathbf{X}) + \lambda E^{-}(\mathbf{X}) \quad \lambda \geq 0$$

- Optimization:
  - compute the gradient

$$\mathbf{G}_k = 4\mathbf{X}_k(\mathbf{L}^{+} - \lambda\widetilde{\mathbf{L}})$$

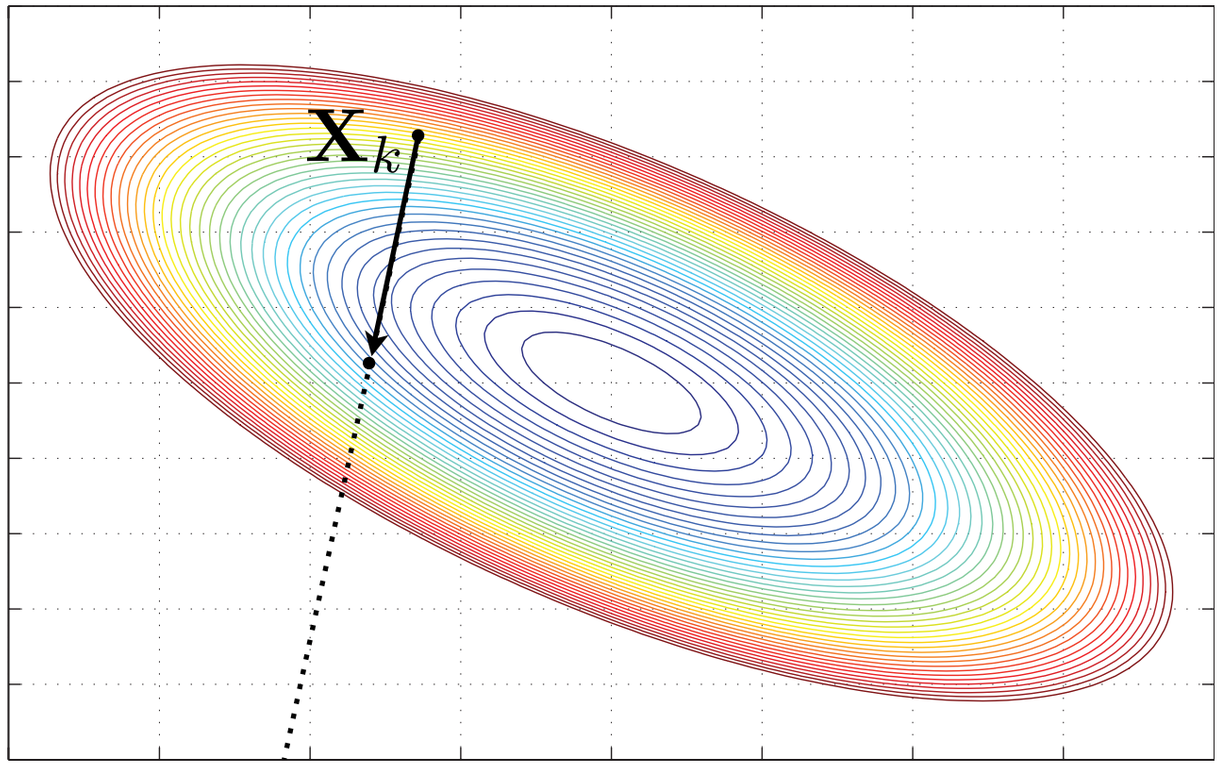
  - compute the direction. For example, gradient descent:

$$\mathbf{P}_k = -\mathbf{G}_k$$

  - compute new iteration $\mathbf{X}_{k+1}$ using a line search:

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \eta\mathbf{P}_k$$

  - repeat till convergence.



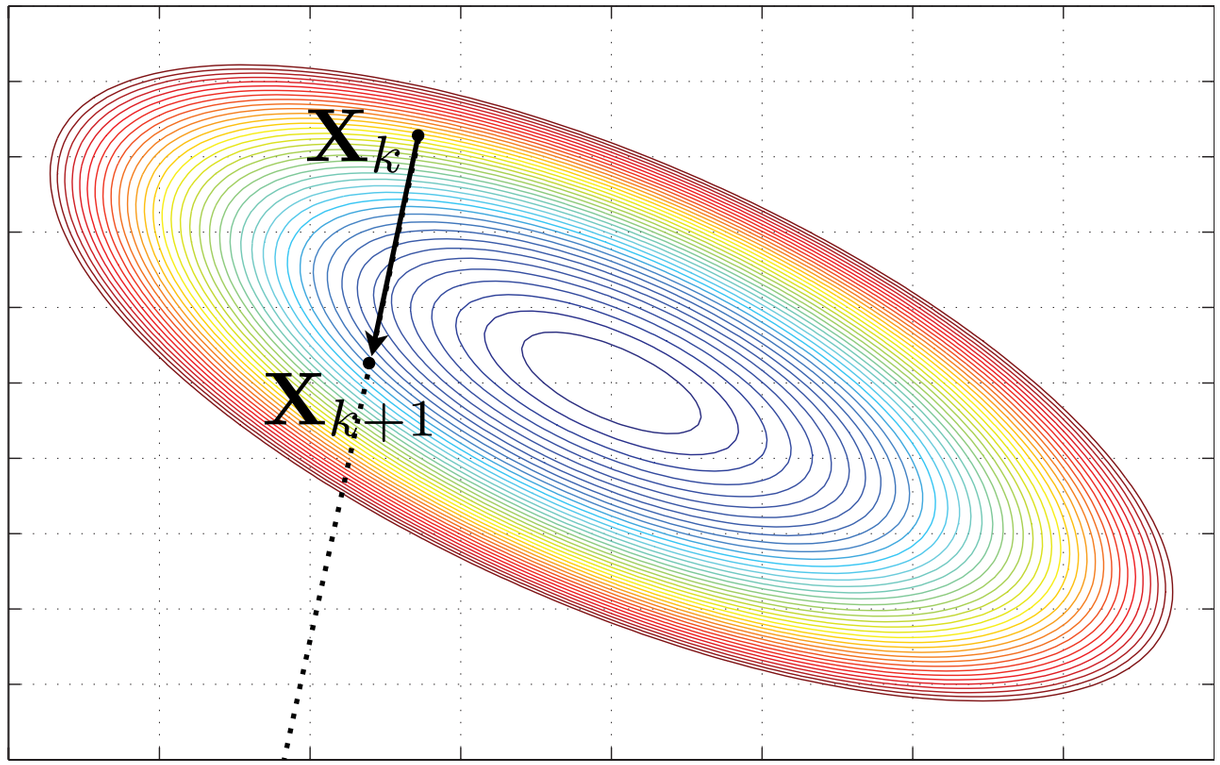
- Other gradient-based optimization methods are applicable: L-BFGS, Conjugate Gradient, etc..

NLE: Optimization

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^{+}(\mathbf{X}) + \lambda E^{-}(\mathbf{X}) \quad \lambda \geq 0$$

- Optimization:
  - compute the gradient
    $\mathbf{G}_k = 4\mathbf{X}_k(\mathbf{L}^{+} - \lambda\widetilde{\mathbf{L}})$
  - compute the direction. For example, gradient descent:
    $\mathbf{P}_k = -\mathbf{G}_k$
  - compute new iteration $\mathbf{X}_{k+1}$ using a line search:
    $\mathbf{X}_{k+1} = \mathbf{X}_k + \eta\mathbf{P}_k$
  - repeat till convergence.



- Other gradient-based optimization methods are applicable: L-BFGS, Conjugate Gradient, etc..

NLE: Optimization

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^{+}(\mathbf{X}) + \lambda E^{-}(\mathbf{X}) \quad \lambda \geq 0$$

- Optimization:
  - compute the gradient

$$\mathbf{G}_k = 4\mathbf{X}_k(\mathbf{L}^{+} - \lambda\widetilde{\mathbf{L}})$$

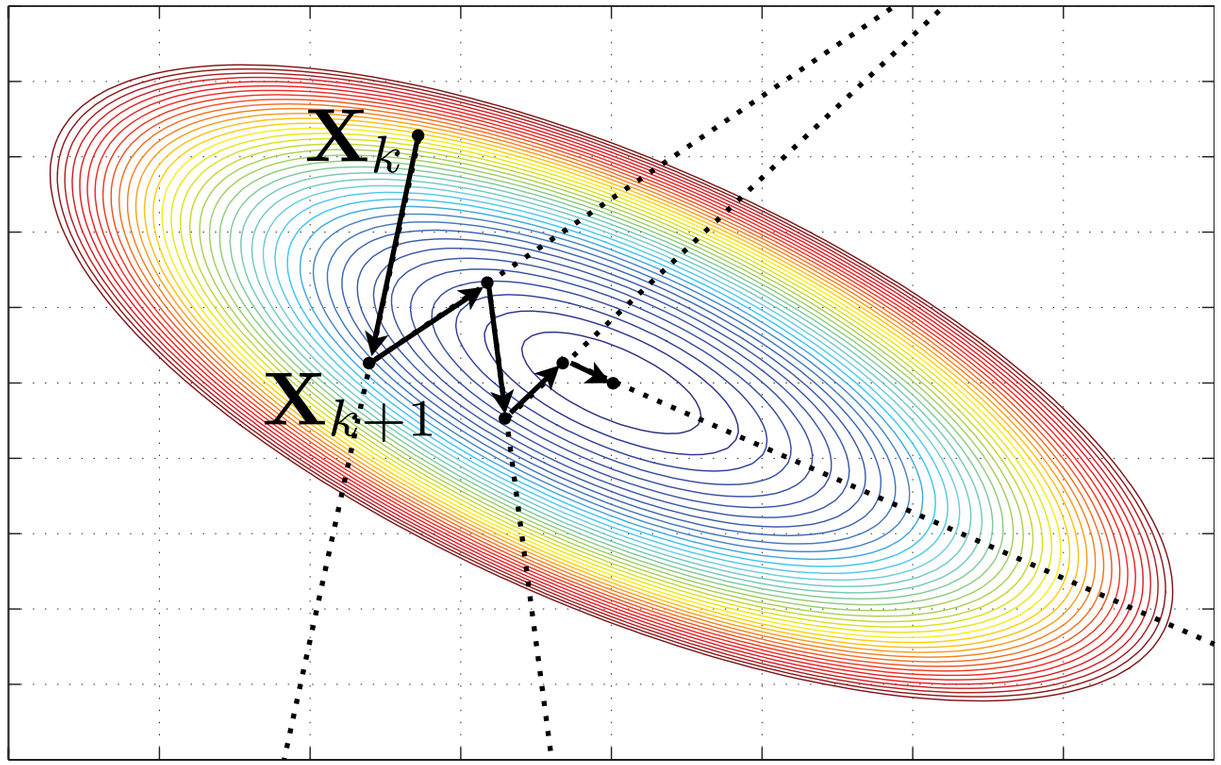
  - compute the direction. For example, gradient descent:

$$\mathbf{P}_k = -\mathbf{G}_k$$

  - compute new iteration $\mathbf{X}_{k+1}$ using a line search:

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \eta\mathbf{P}_k$$

  - repeat till convergence.

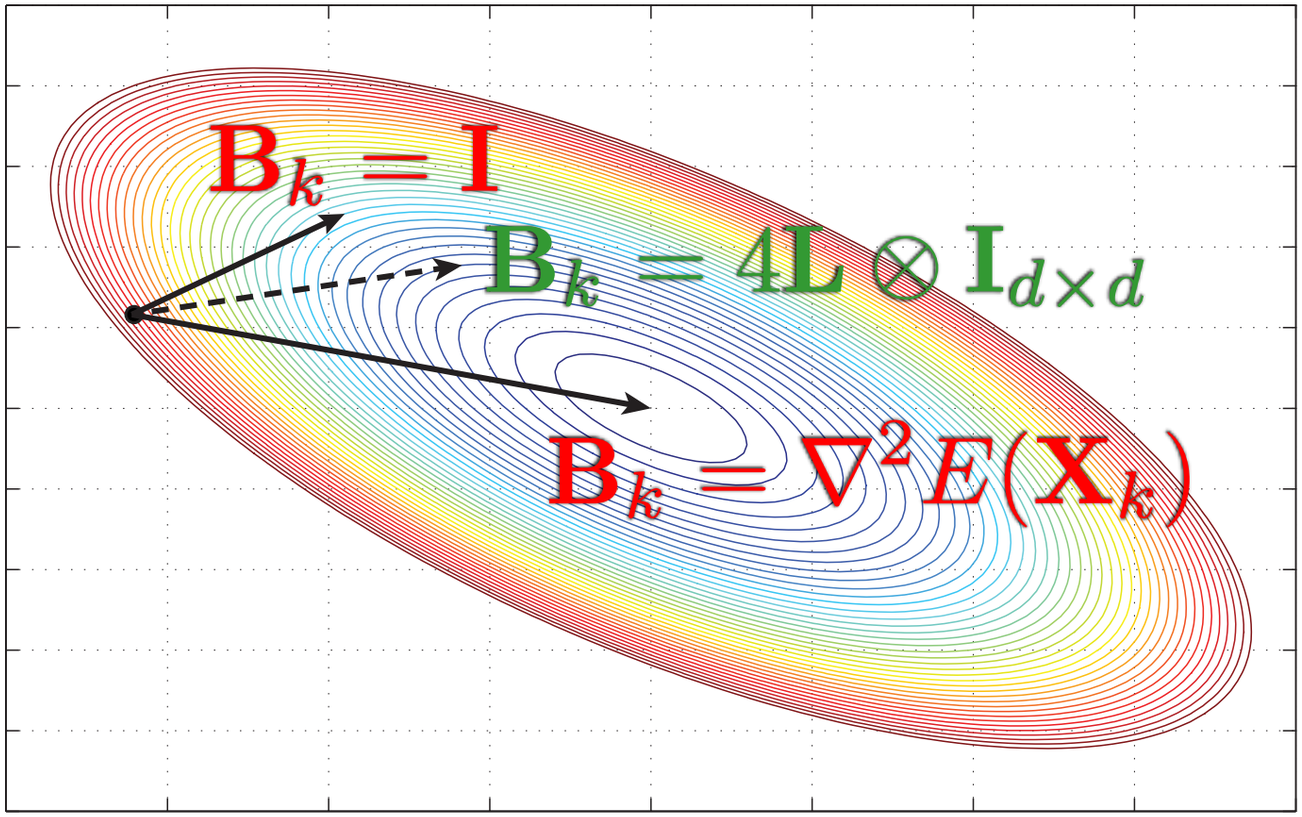


- Other gradient-based optimization methods are applicable: L-BFGS, Conjugate Gradient, etc..

Spectral direction (Vladymyrov and Carreira-Perpiñán, '12)

Currently, the fastest optimization algorithm to train nonlinear embedding is spectral direction.

1. Precompute the Cholesky factor of positive definite, constant Hessian approx.
   $\mathbf{B} = 4\mathbf{L} \otimes \mathbf{I}_{d \times d}$
2. For every iteration k
   - find search direction using the solution to the linear system:
     $\mathbf{B}_k \mathbf{P}_k = -\mathbf{G}_k$
   - use line search to find a step size η for the next iteration $\mathbf{X}_{k+1} = \mathbf{X}_k + \eta \mathbf{P}_k$



- This method is much faster than gradient descent.
- However, spectral direction, as well as other gradient-based methods require gradient evaluation for every iteration.

NLE: Gradient

The gradient is given by $\mathbf{G} = 4\mathbf{X}(\mathbf{L} - \lambda\widetilde{\mathbf{L}})$

where graph Laplacians are defined as:

$$\mathbf{L} = \operatorname{diag}\left(\sum_{n=1}^{N} w_{nm}\right) - \mathbf{W} \;;\; \widetilde{\mathbf{L}} = \operatorname{diag}\left(\sum_{n=1}^{N} \widetilde{w}_{nm}\right) - \widetilde{\mathbf{W}}$$

Weights w_{nm} are constants and can be sparsified.

Weights $\widetilde{w}_{nm}$ depend on parameters $\mathbf{X}$ and should be recomputed for every point.

For example, in elastic embedding algorithm:

$$E_{EE}(\mathbf{X}) = \sum_{n,m=1}^{N} w_{nm} \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2 + \lambda \sum_{n=1}^{N} S(\mathbf{x}_n)$$

$$G_{EE}(\mathbf{X}) = 4\mathbf{X}\mathbf{L} - 4\lambda\mathbf{X}\operatorname{diag}\left(S(\mathbf{X})\right) + 4\lambda S^x(\mathbf{X})$$

with $S(\mathbf{x}_n) = \sum_{m=1}^{N} e^{-\|\mathbf{x}_n - \mathbf{x}_m\|^2}$; $S^x(\mathbf{x}_n) = \sum_{m=1}^{N} \mathbf{x}_m e^{-\|\mathbf{x}_n - \mathbf{x}_m\|^2}$

Computing $S^x(\mathbf{x}_n)$ and $S(\mathbf{x}_n)$ for every $n = 1, \ldots, N$ is $\mathcal{O}(N^2)$.

Computational bottleneck of NLE

- The bottleneck of the algorithm consists in computing pairwise interaction between data points (*N*-body problem).

$$S(\mathbf{x}_n) = \sum_{m=1}^{N} e^{-\|\mathbf{x}_n - \mathbf{x}_m\|^2} \qquad S^x(\mathbf{x}_n) = \sum_{m=1}^{N} \mathbf{x}_m e^{-\|\mathbf{x}_n - \mathbf{x}_m\|^2}$$

- Solution: use approximate methods to compute these interactions!
  - tree-based methods;
  - fast multipole methods.

Computational bottleneck of NLE

- The bottleneck of the algorithm consists in computing pairwise interaction between data points (*N*-body problem).

$$S(\mathbf{x}_n) = \sum_{m=1}^{N} e^{-\|\mathbf{x}_n - \mathbf{x}_m\|^2} \qquad S^x(\mathbf{x}_n) = \sum_{m=1}^{N} \mathbf{x}_m e^{-\|\mathbf{x}_n - \mathbf{x}_m\|^2}$$

- Solution: use approximate methods to compute these interactions!
  - tree-based methods;
  - fast multipole methods.

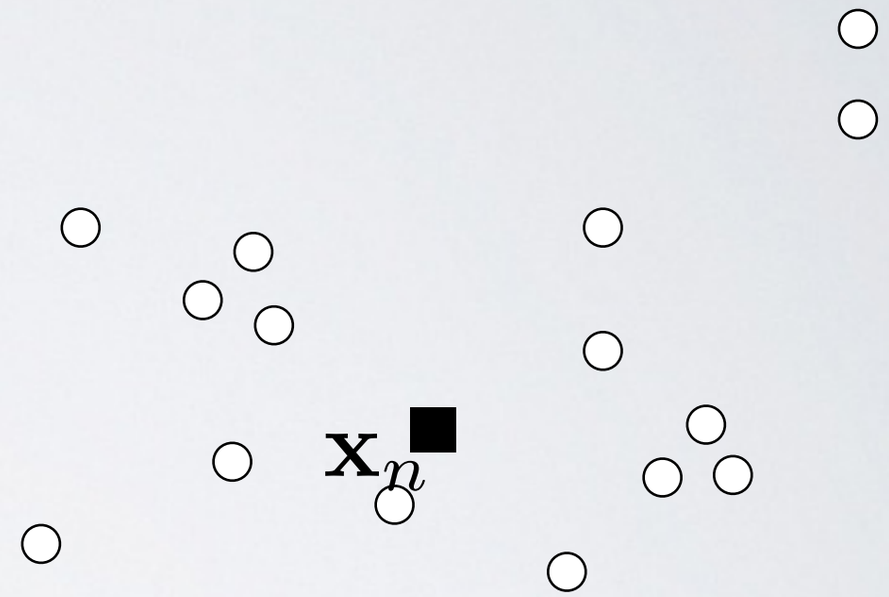
Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$
- Query the nodes of the tree rather than individual points.

  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.



- Complexity is usually $\mathcal{O}(N \log N)$
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually

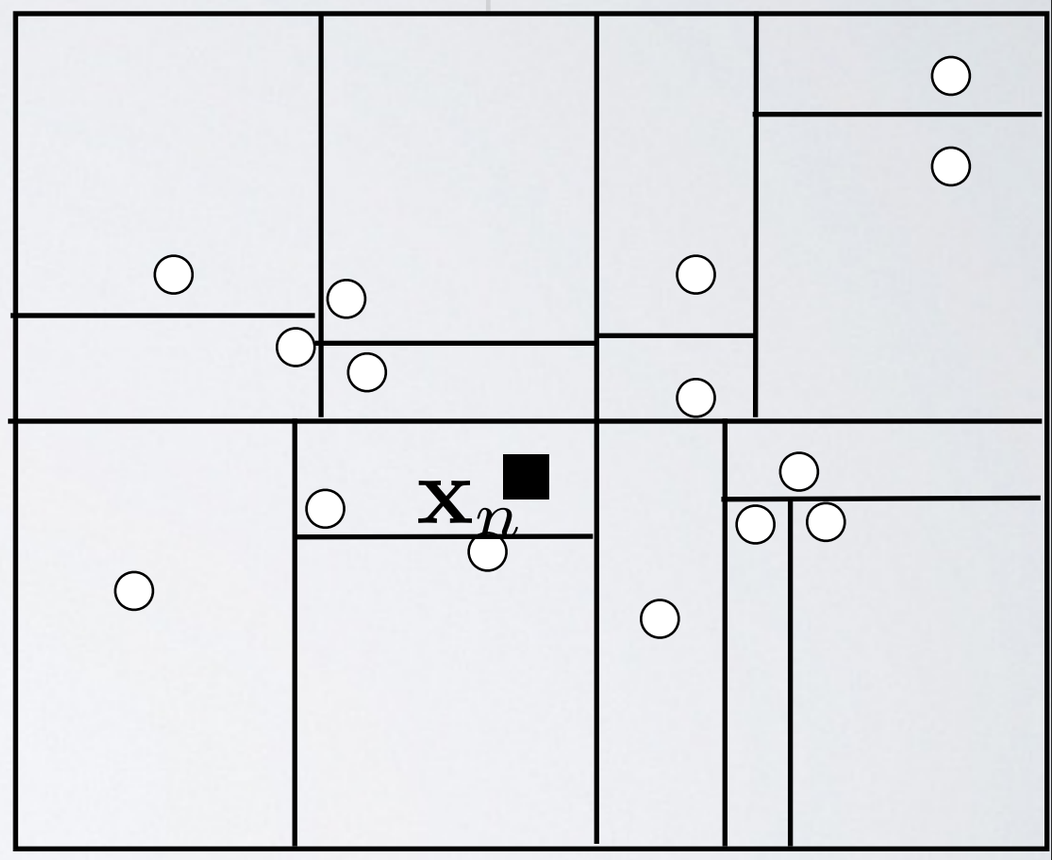
Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$.
- Query the nodes of the tree rather than individual points.

  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.
- Complexity is usually $\mathcal{O}(N \log N)$
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually



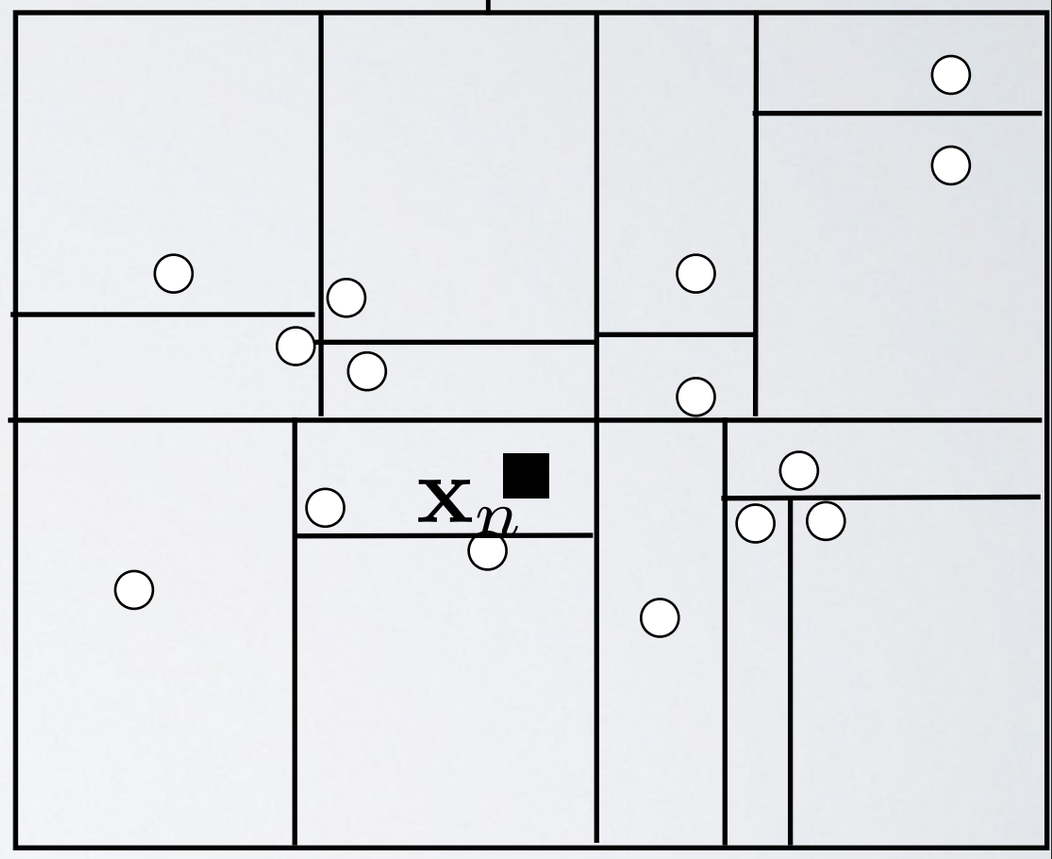
Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$.
- Query the nodes of the tree rather than individual points.

  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.
- Complexity is usually $\mathcal{O}(N \log N)$
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually



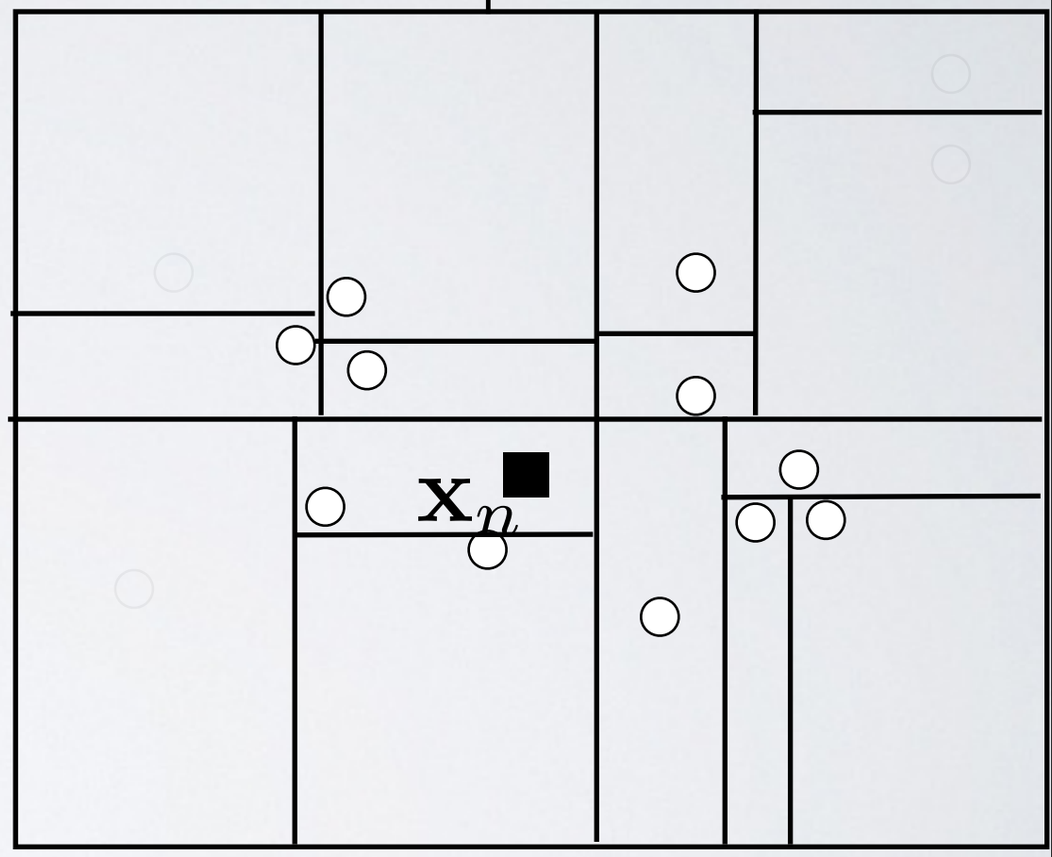
Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$.
- Query the nodes of the tree rather than individual points.

  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.
- Complexity is usually $\mathcal{O}(N \log N)$
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually



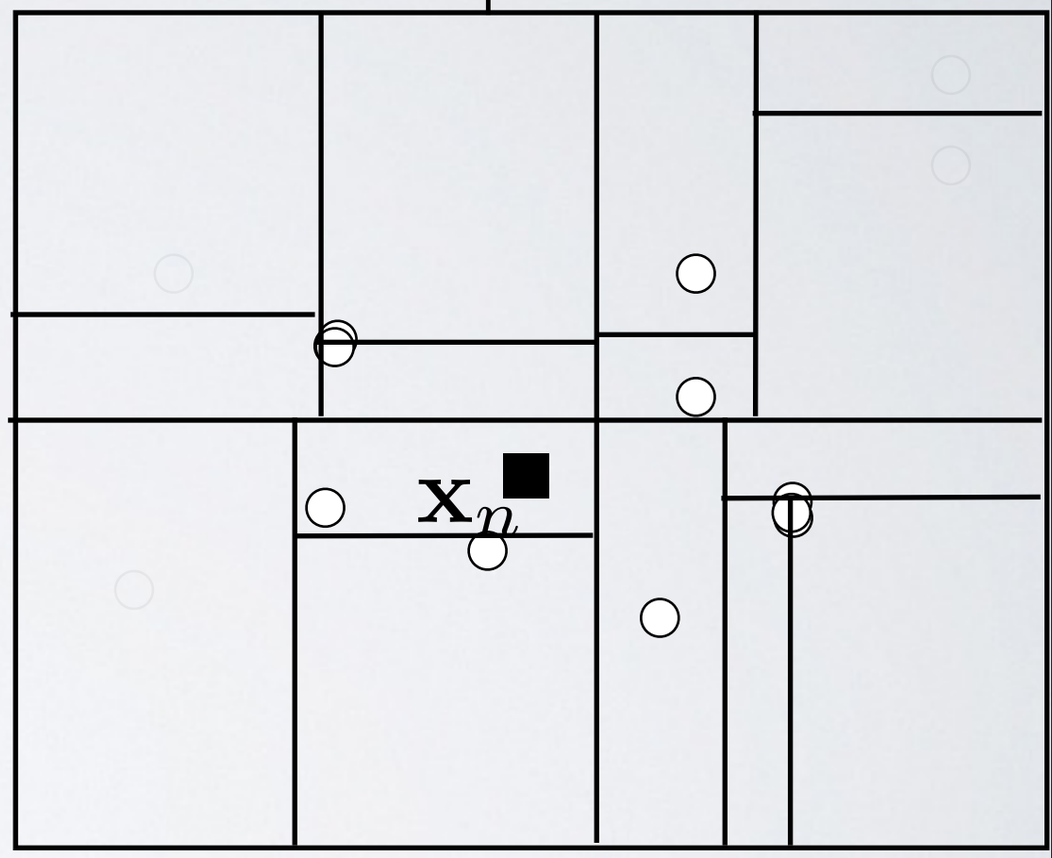
Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$.
- Query the nodes of the tree rather than individual points.

  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.



- Complexity is usually $\mathcal{O}(N \log N)$
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually

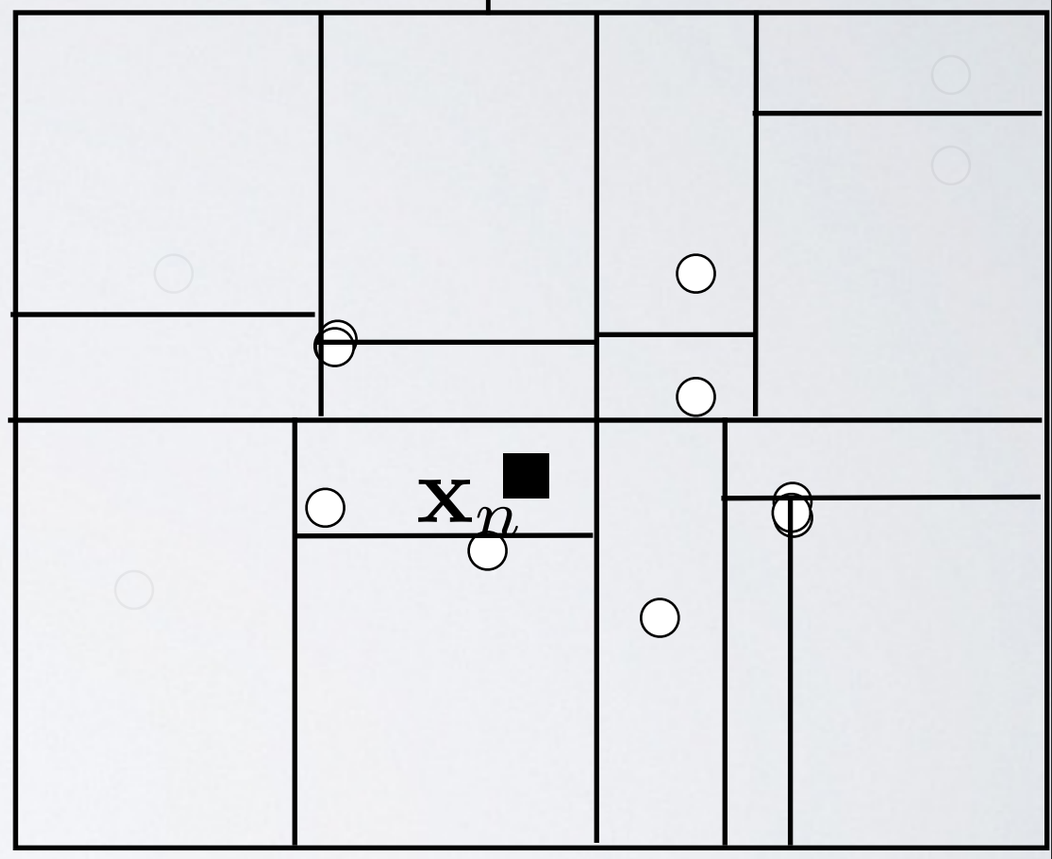
Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$.
- Query the nodes of the tree rather than individual points.

  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.
- Complexity is usually $\mathcal{O}(N \log N)$.
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually



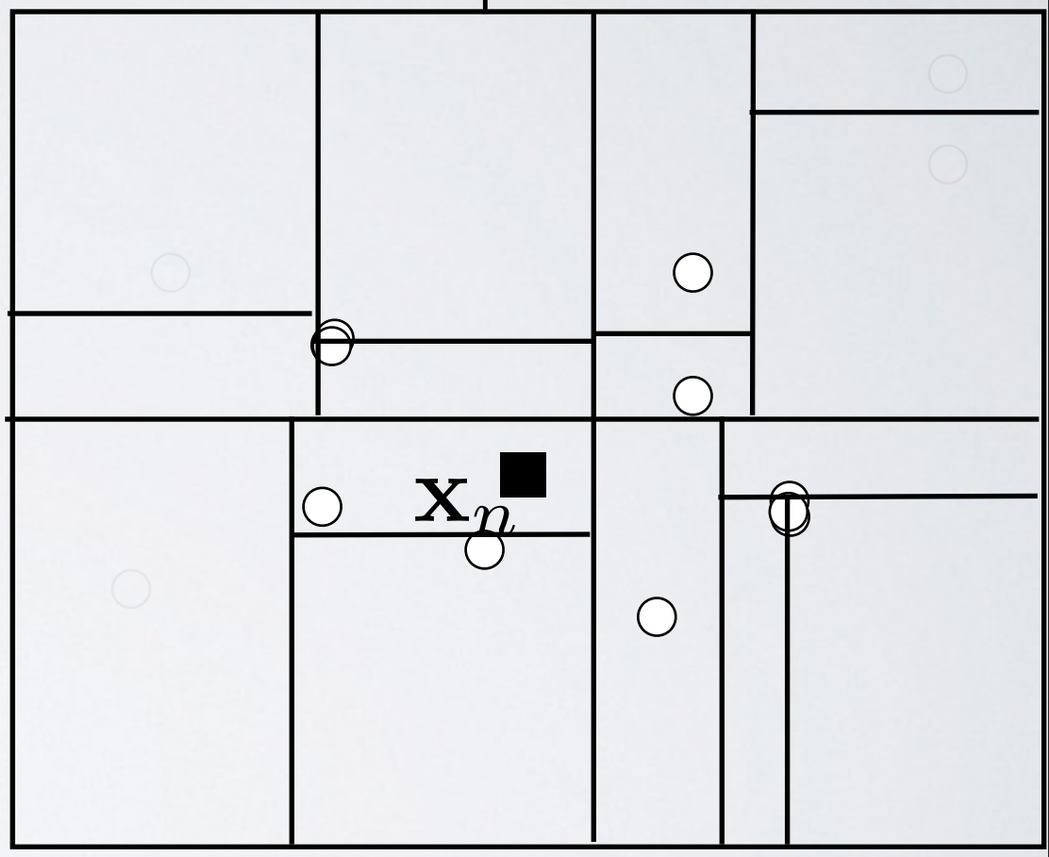
Tree-based methods

Example: *kd*-tree, dual-trees, Barnes-Hut algorithm, etc.

To compute the interaction between $\mathbf{x}_n$ and others points:

- Build a tree around $\mathbf{X}$.
- Query the nodes of the tree rather than individual points.

  Gains come from:
  - pruning interaction between points that are too far away.
  - approximating the interactions between points that are located at a similar distance.



- Complexity is usually $\mathcal{O}(N \log N)$.
- Problems:
  - do not scale well with dimensions of latent space,
  - error bounds are usually hard to derive.

Barnes-Hut algorithm (Barnes and Hut '86)

☺ Can be applicable to any kind of interaction (Euclidean distances, Gaussian distances, etc).

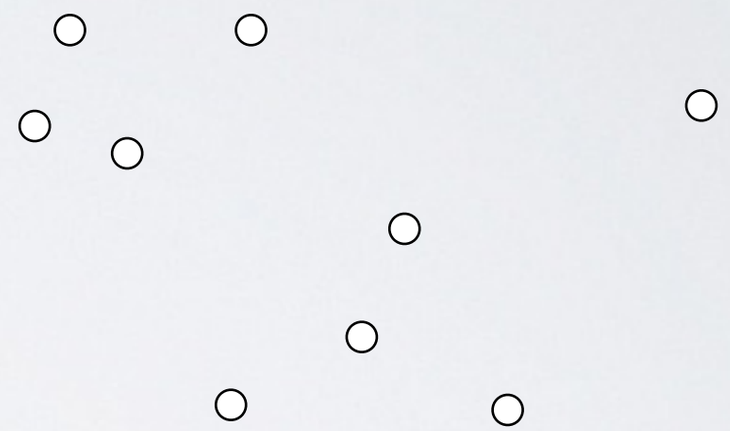
☺ Single parameter to control the trade-off between speed and approximation error.

☹ No clearly defined error bounds.

Were used in the context of nonlinear embedding algorithm in Maaten, '13 and Yang et al., '13.

Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0, 1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

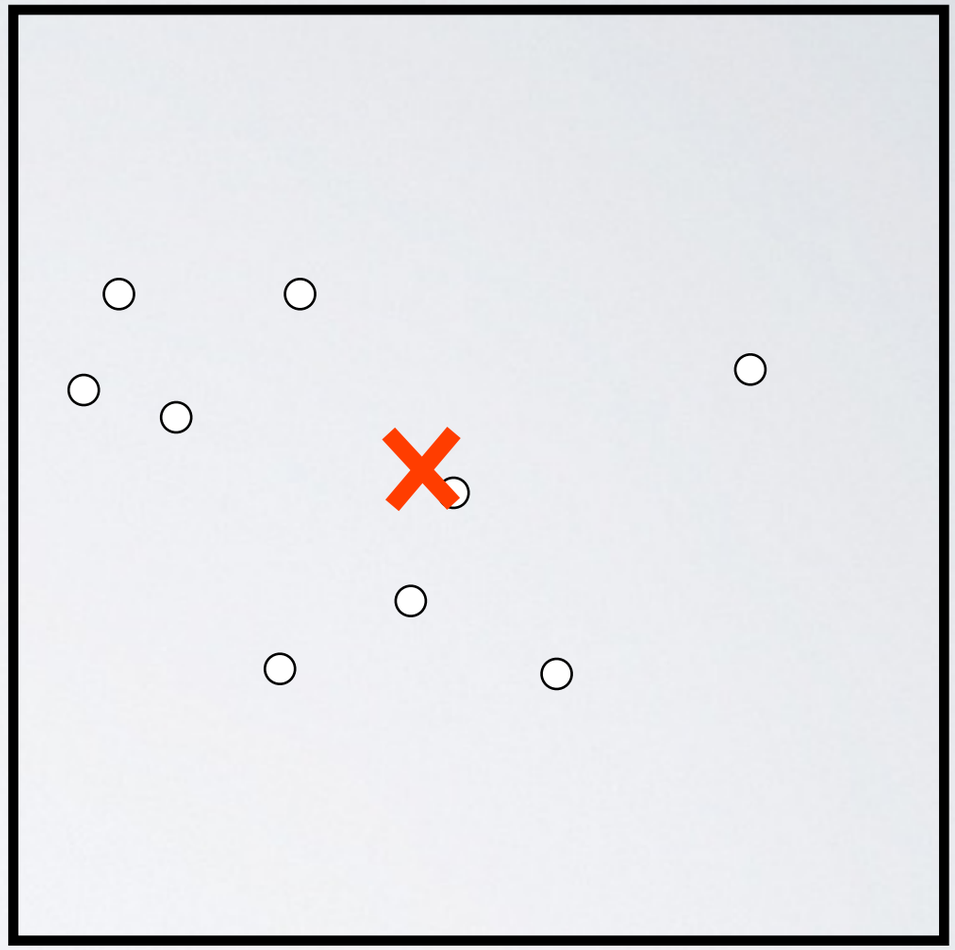


Complexity: $\mathcal{O}(N \log N)$

Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0,1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

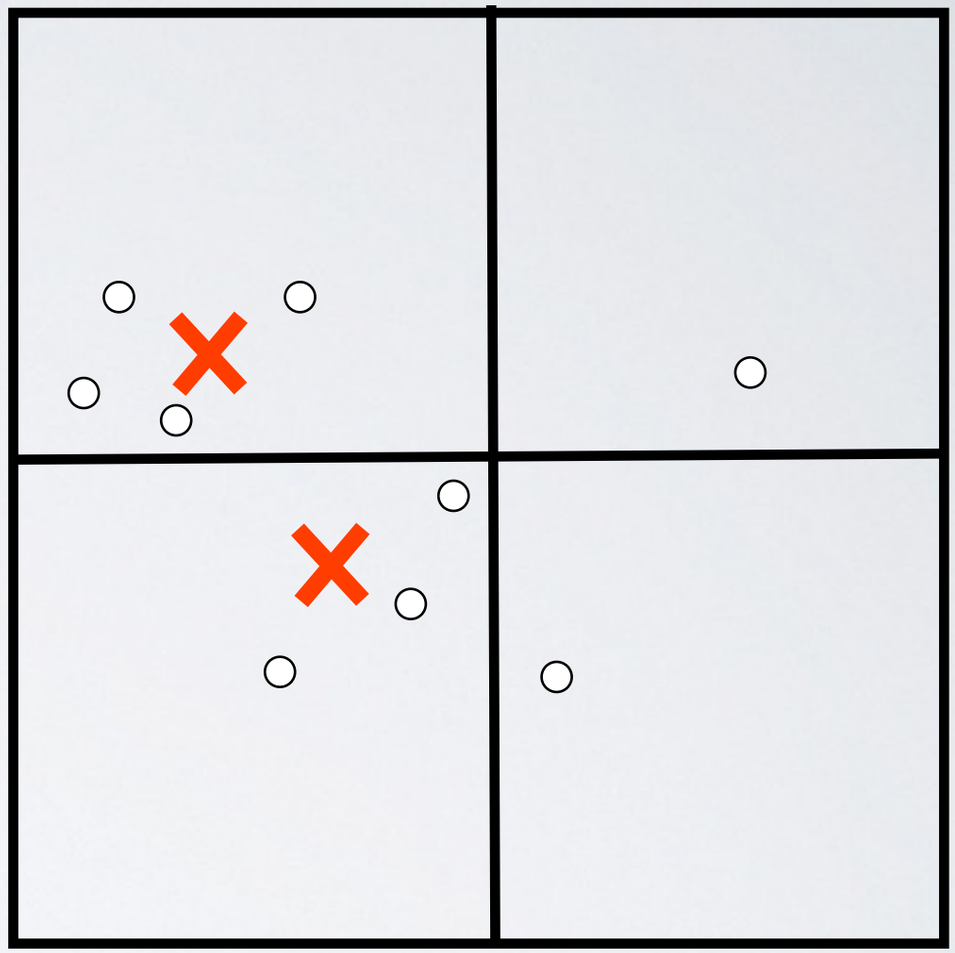
Complexity: $\mathcal{O}(N \log N)$



Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0,1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

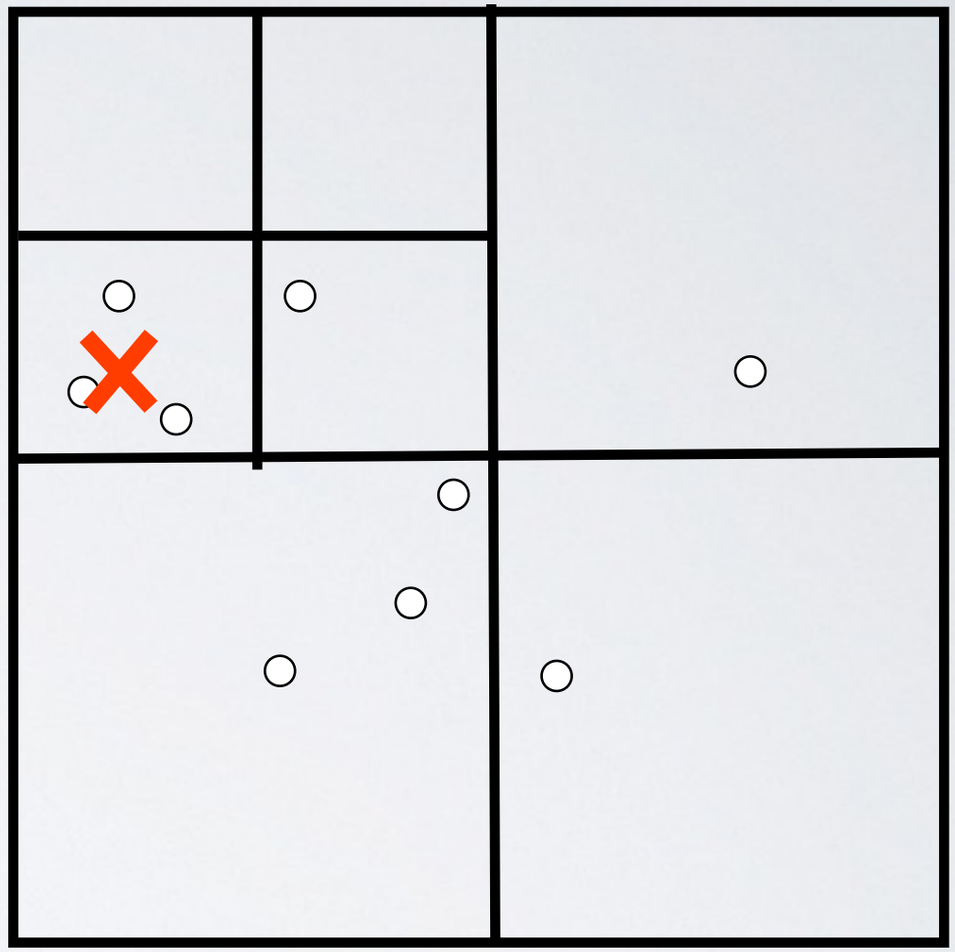
Complexity: $\mathcal{O}(N \log N)$



Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0, 1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

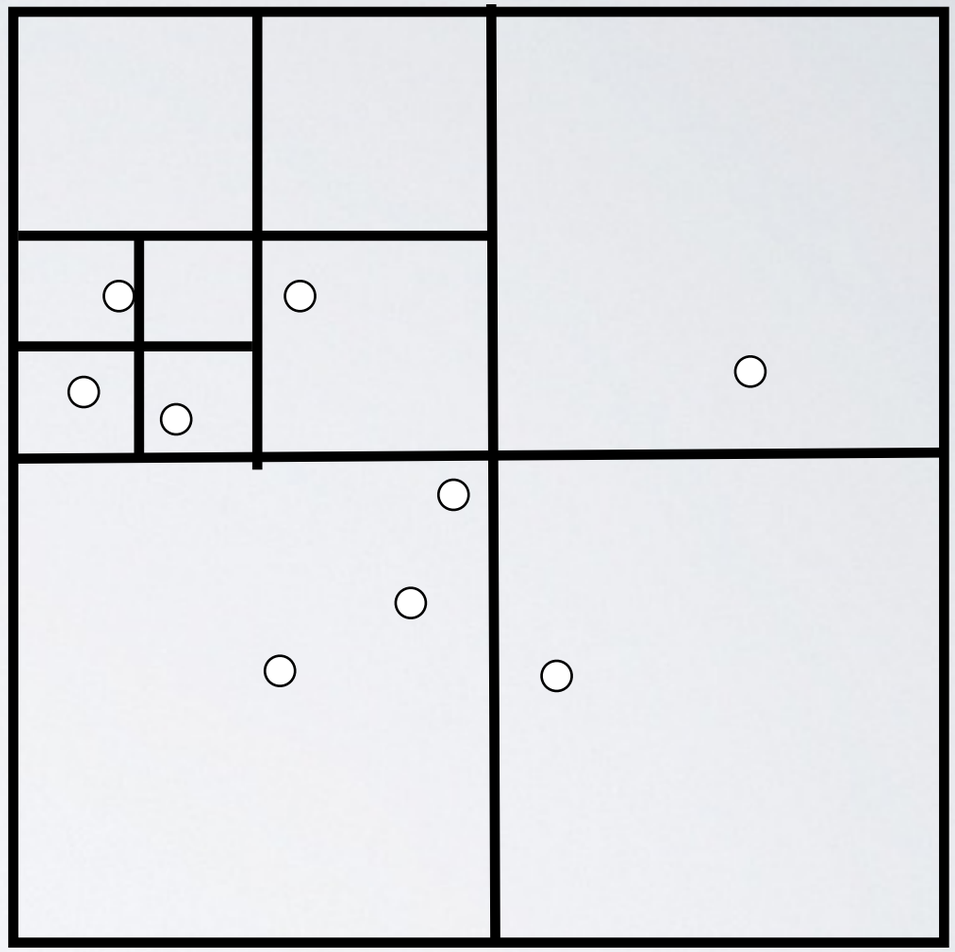
Complexity: $\mathcal{O}(N \log N)$



Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0, 1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

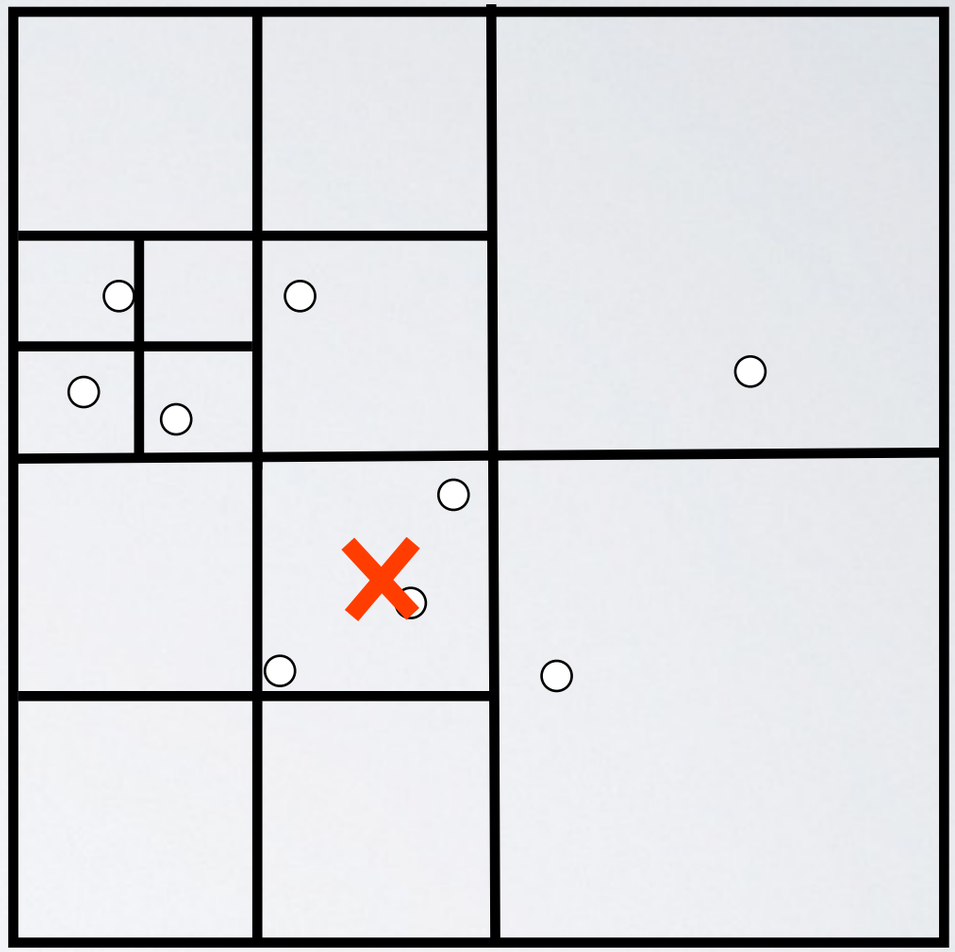
Complexity: $\mathcal{O}(N \log N)$



Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0,1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

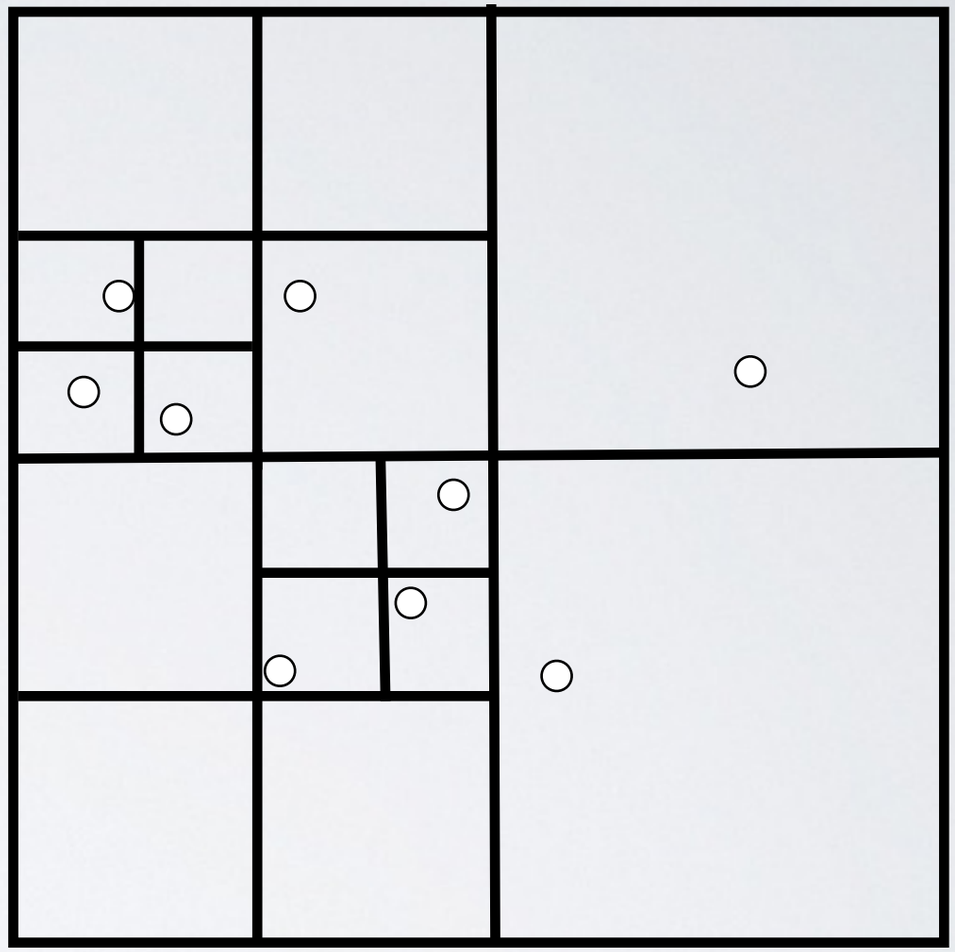
Complexity: $\mathcal{O}(N \log N)$



Barnes-Hut: building a quad-tree

1. Make sure that the points are located in the box $[0,1]^d$.
2. If there are more than two points in the cell, compute its centroid and split it.

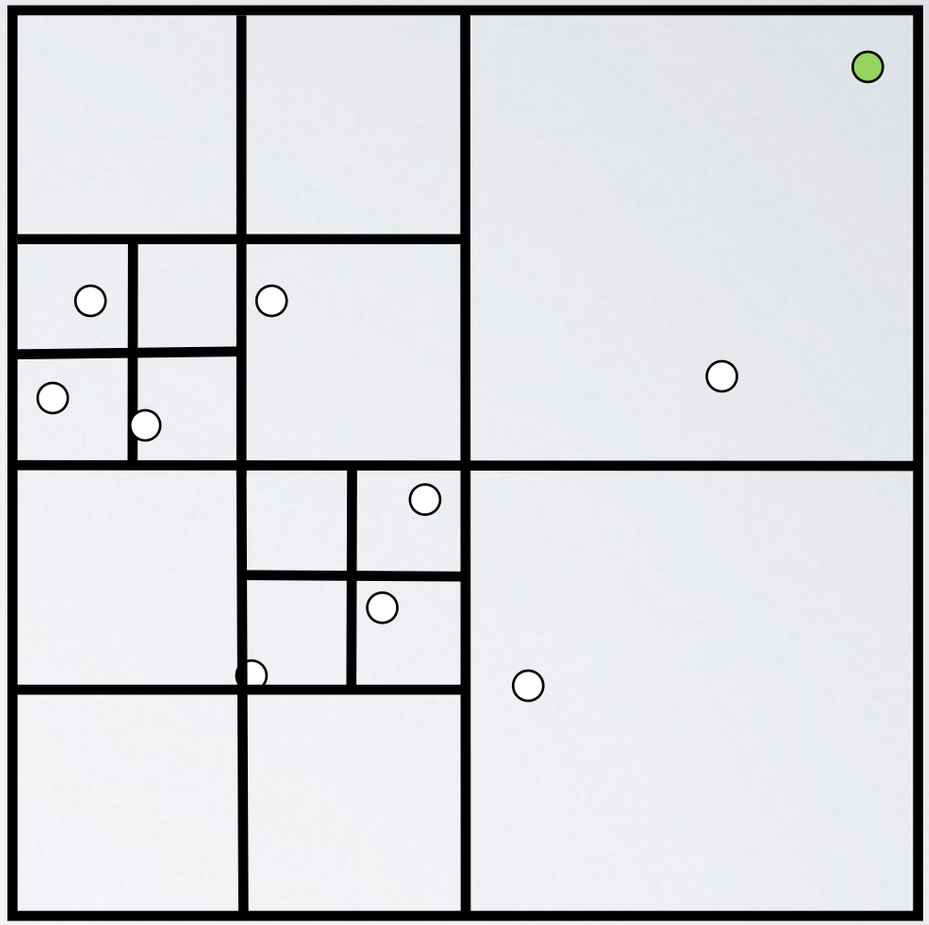
Complexity: $\mathcal{O}(N \log N)$



Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

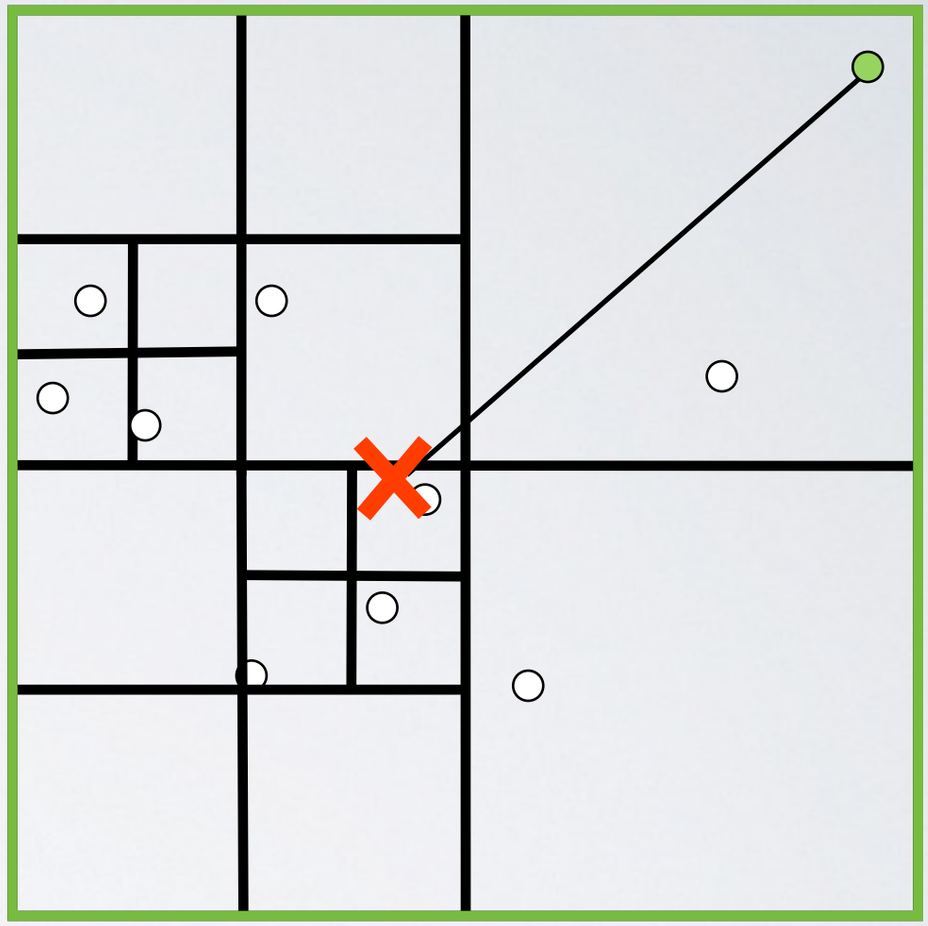
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

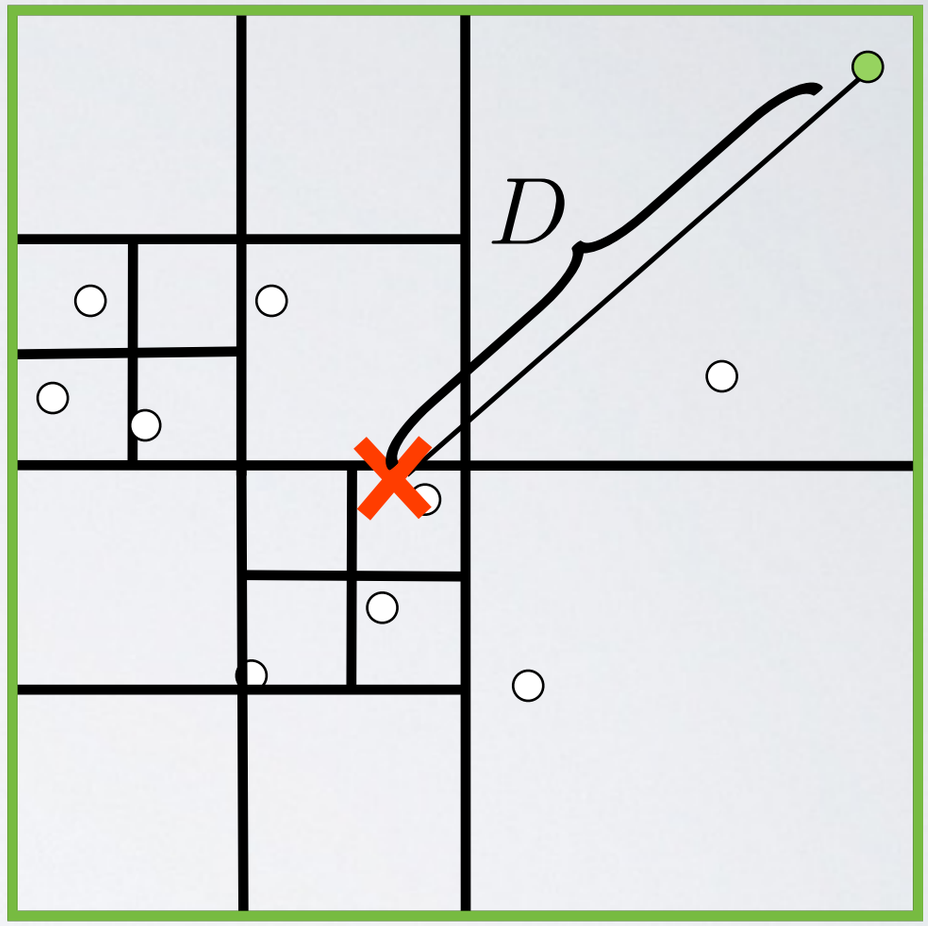
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

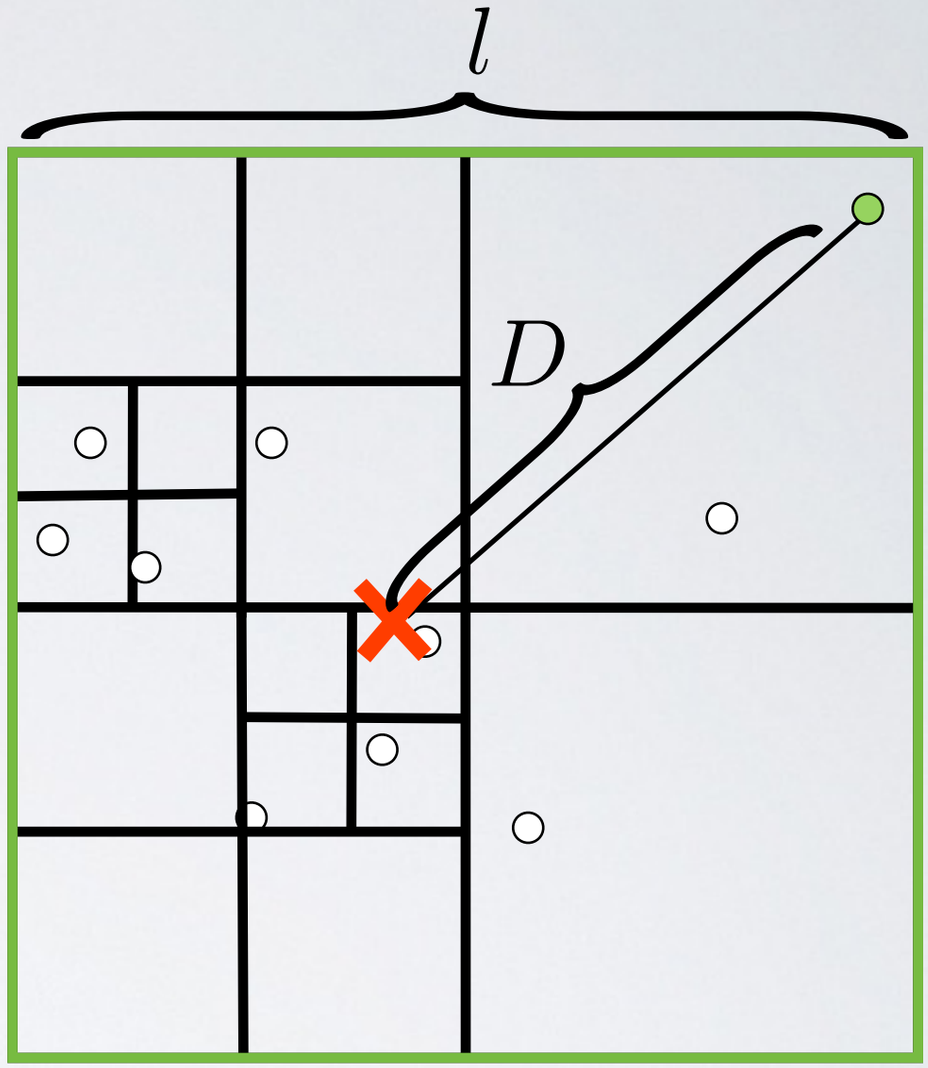
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

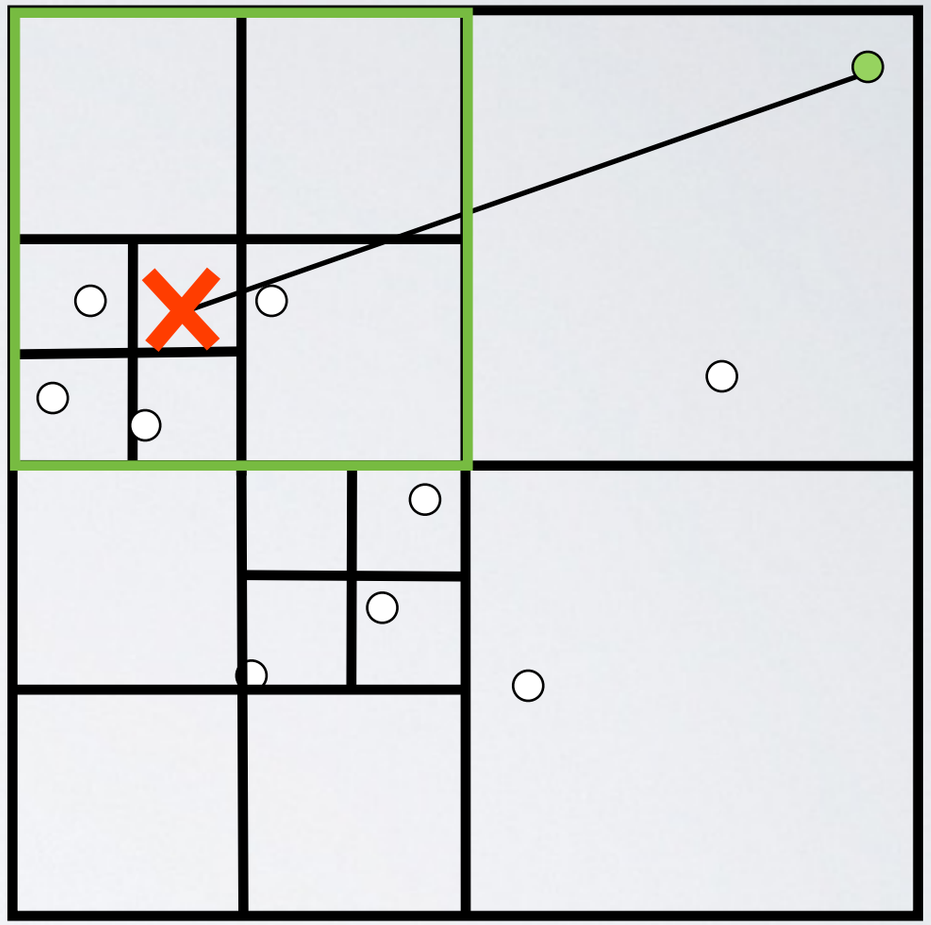
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

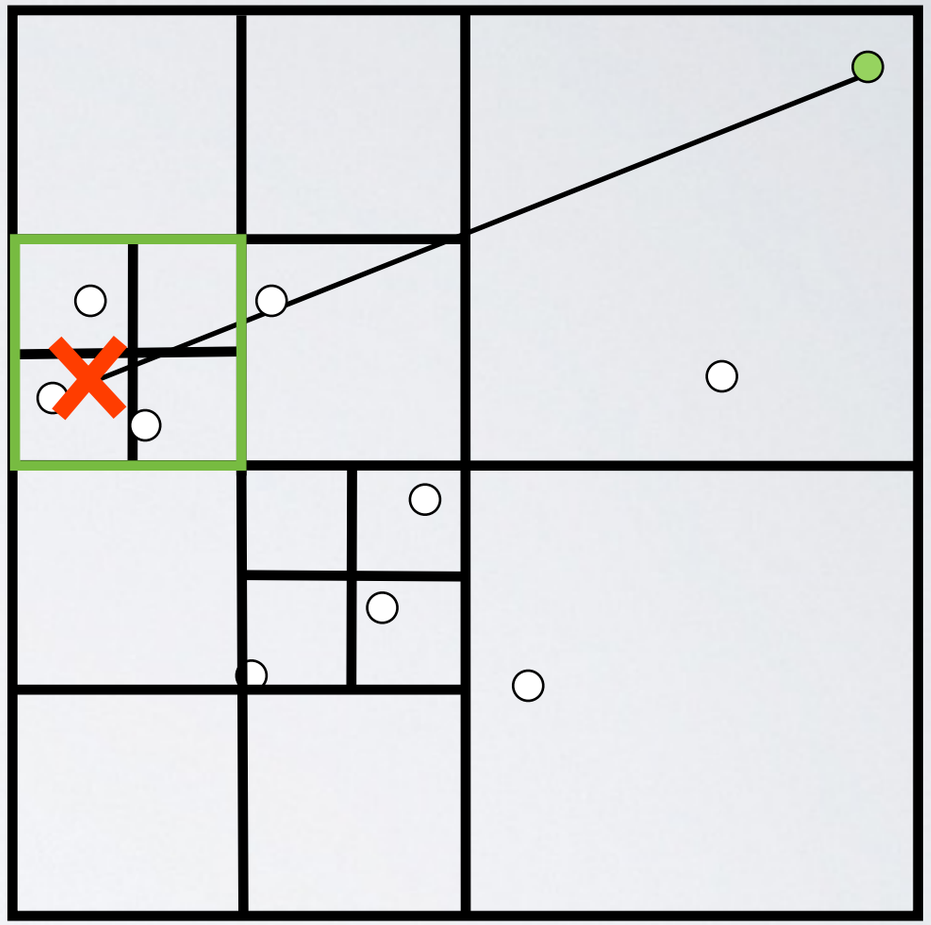
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

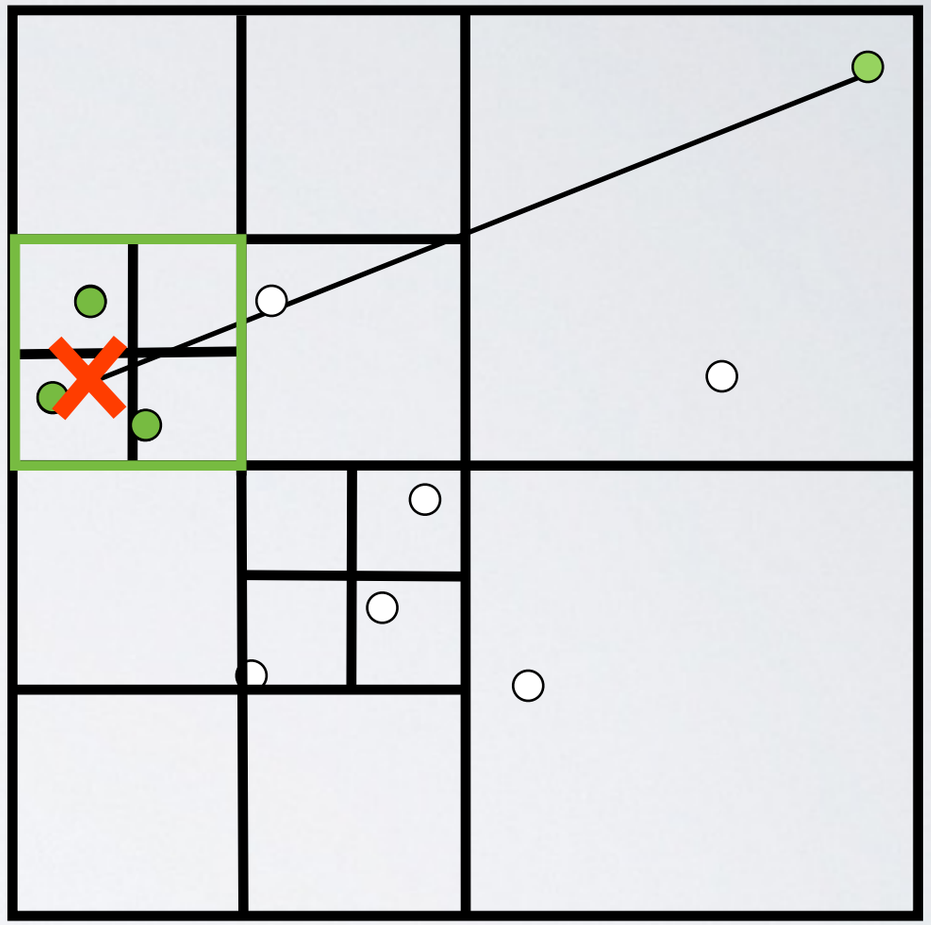
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

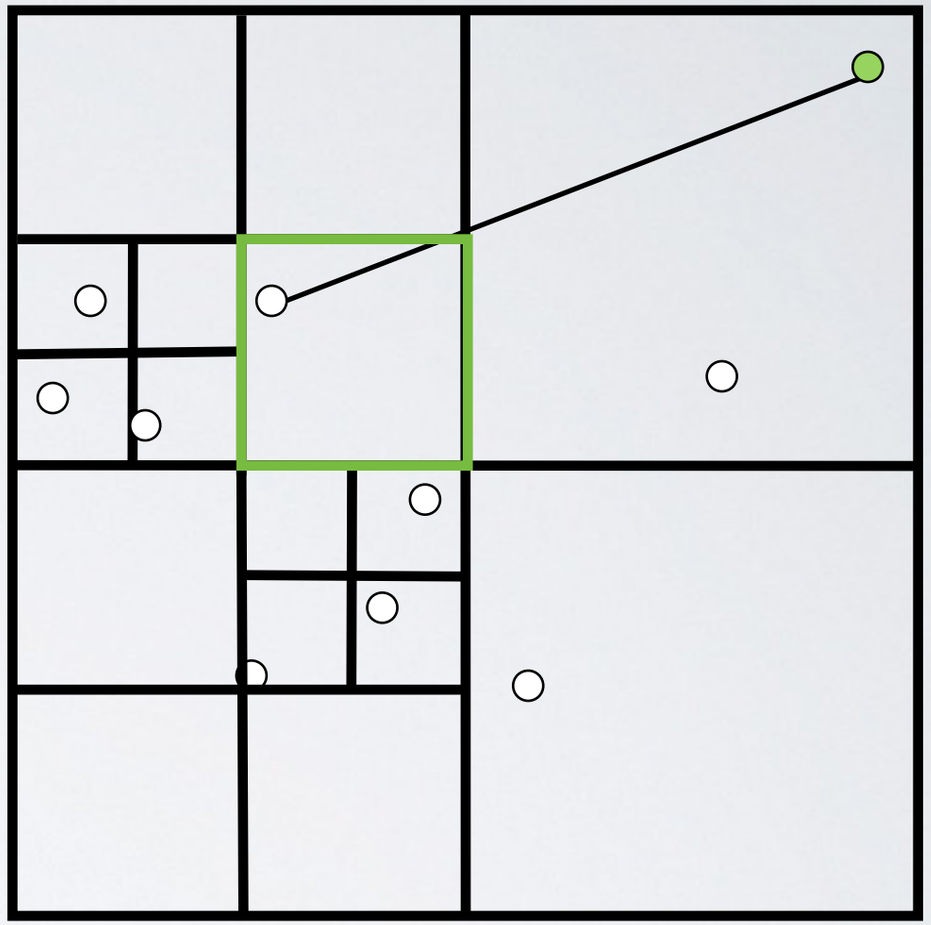
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

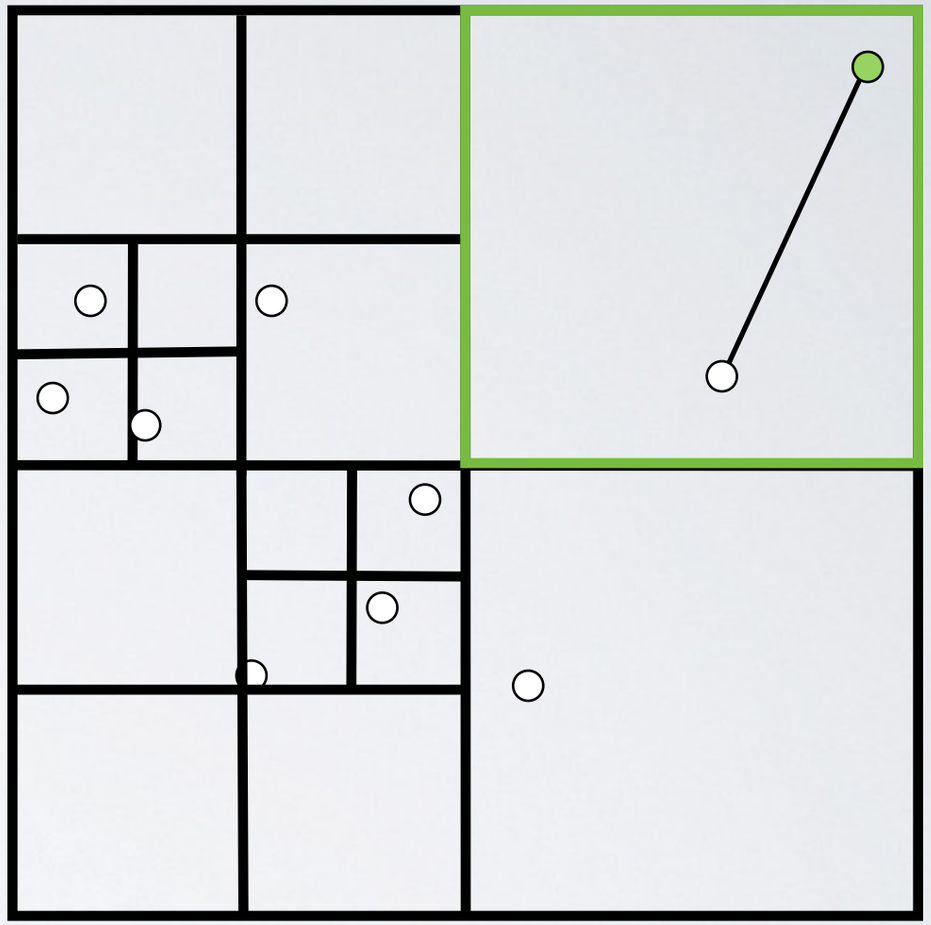
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

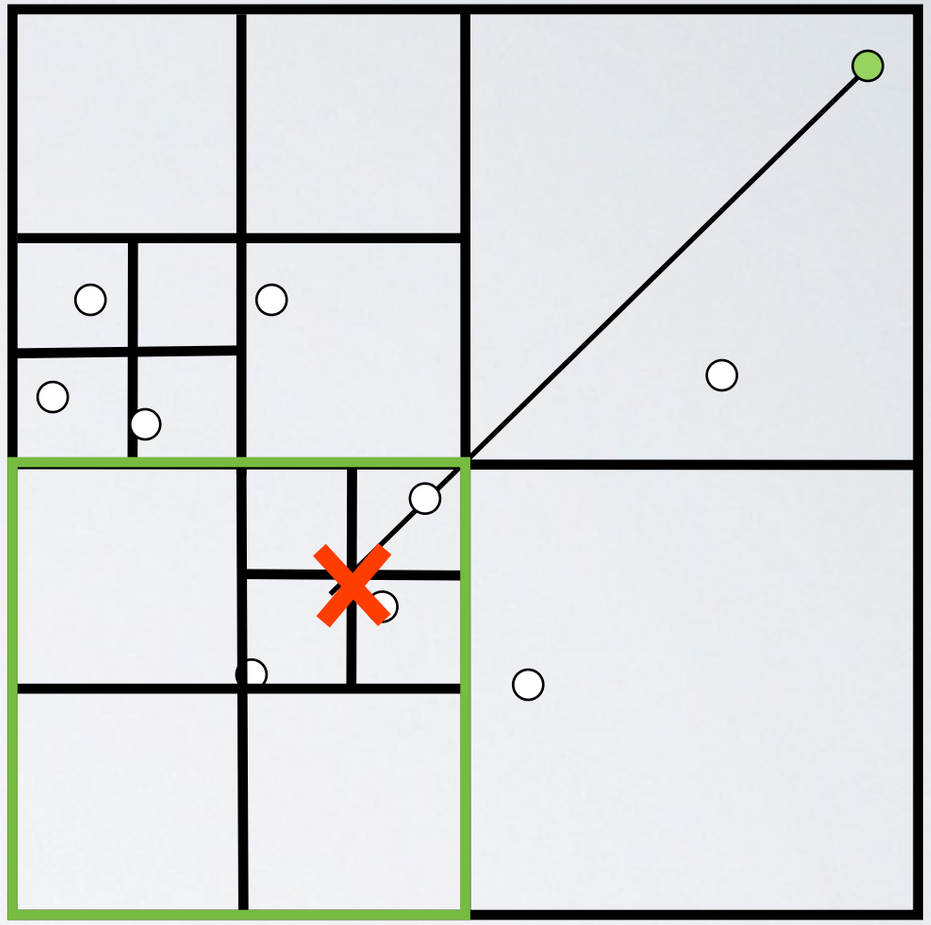
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

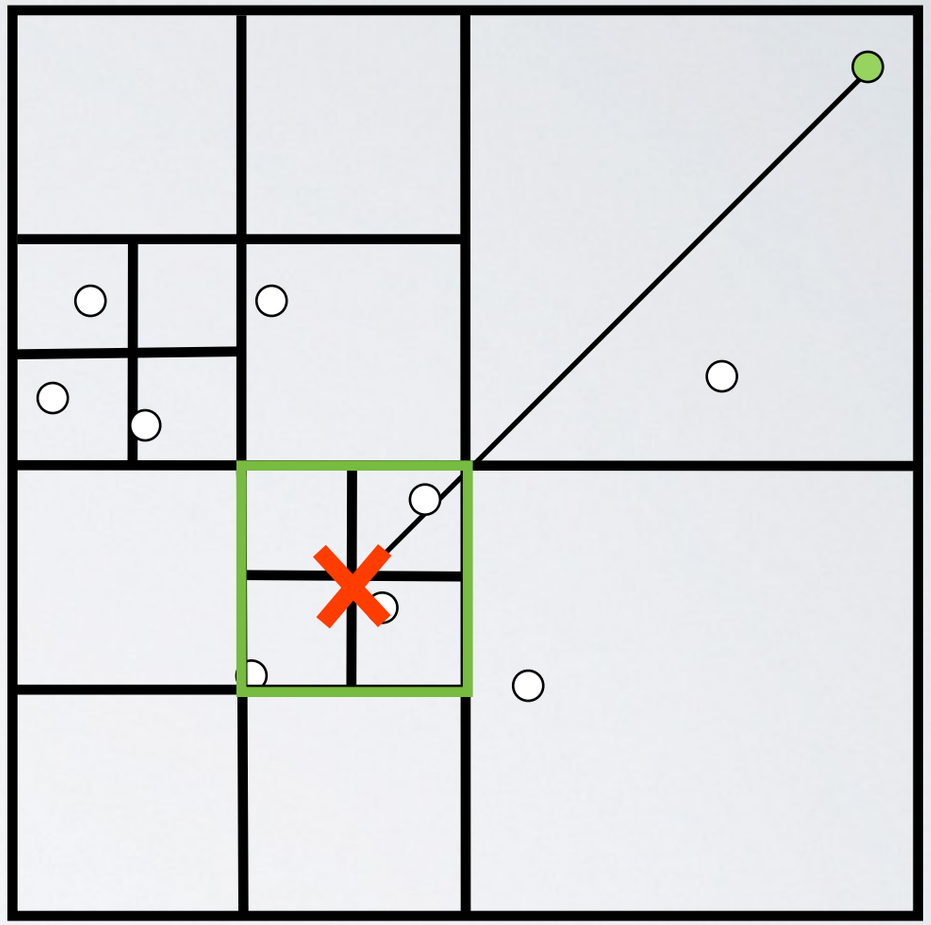
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

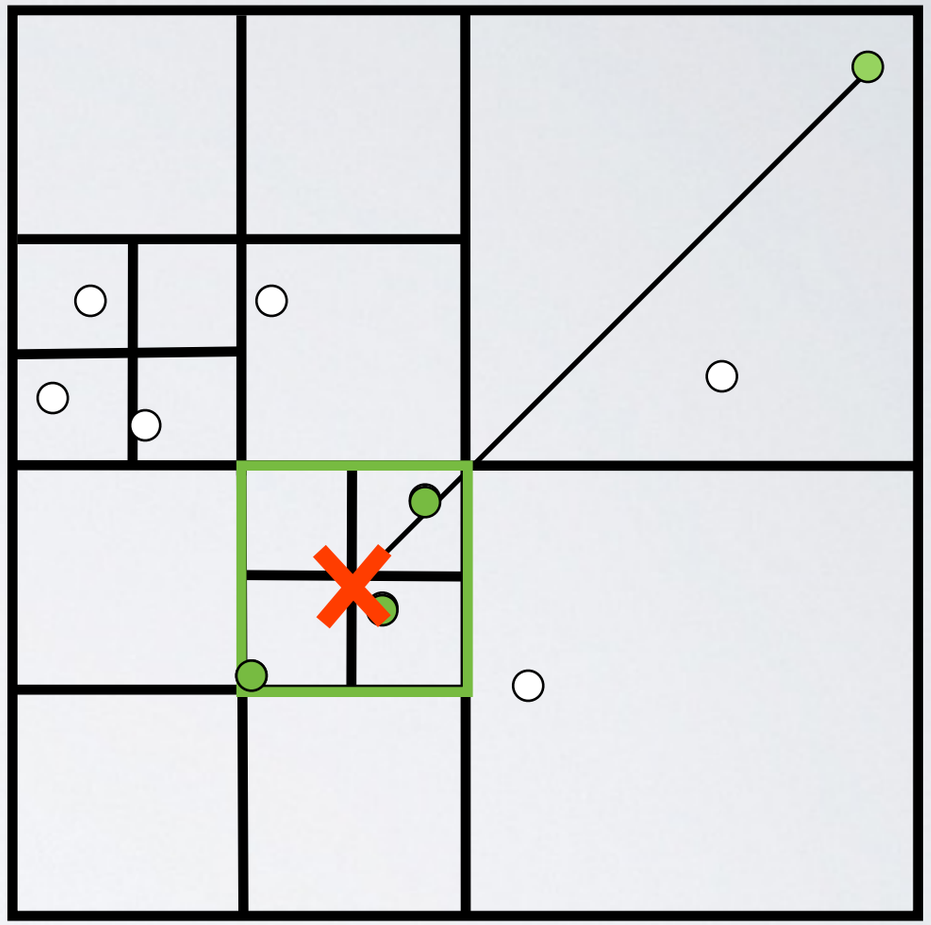
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

$$\frac{l}{D} < \theta$$

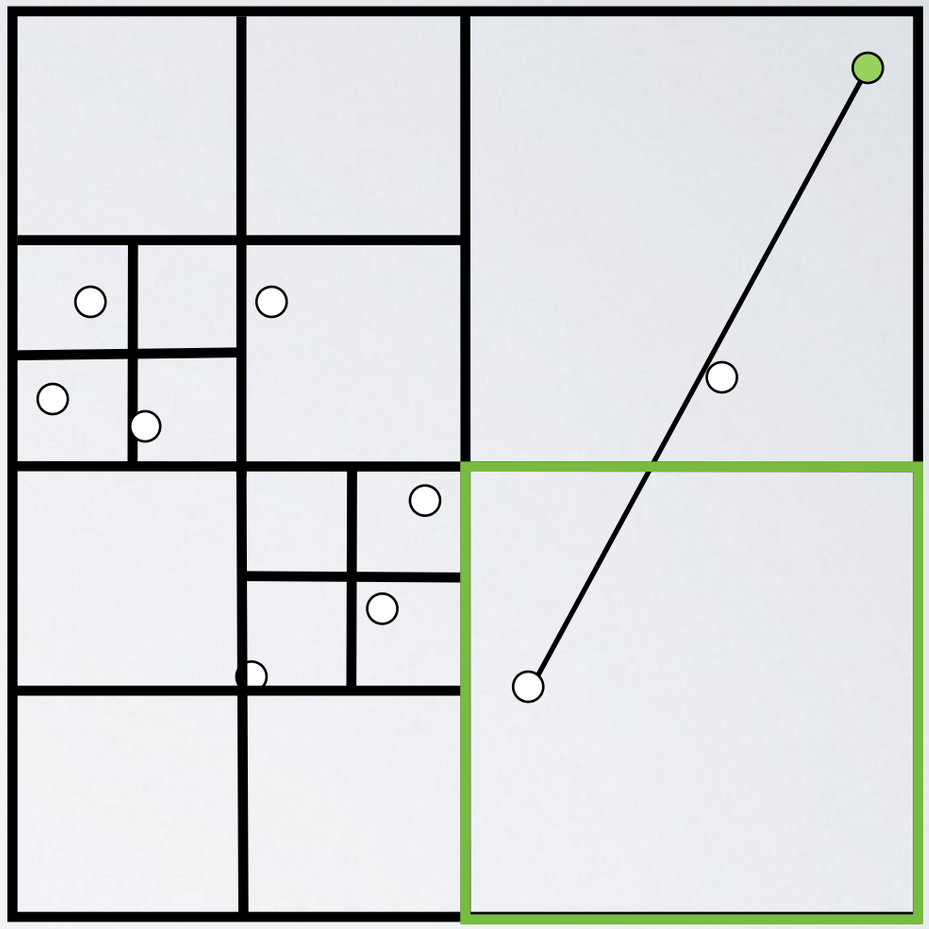
where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Barnes-Hut: querying a quad-tree

D - distance from the query point to the centroid

l - side length of the current cell,



Approximate the interaction with all points in the cell if

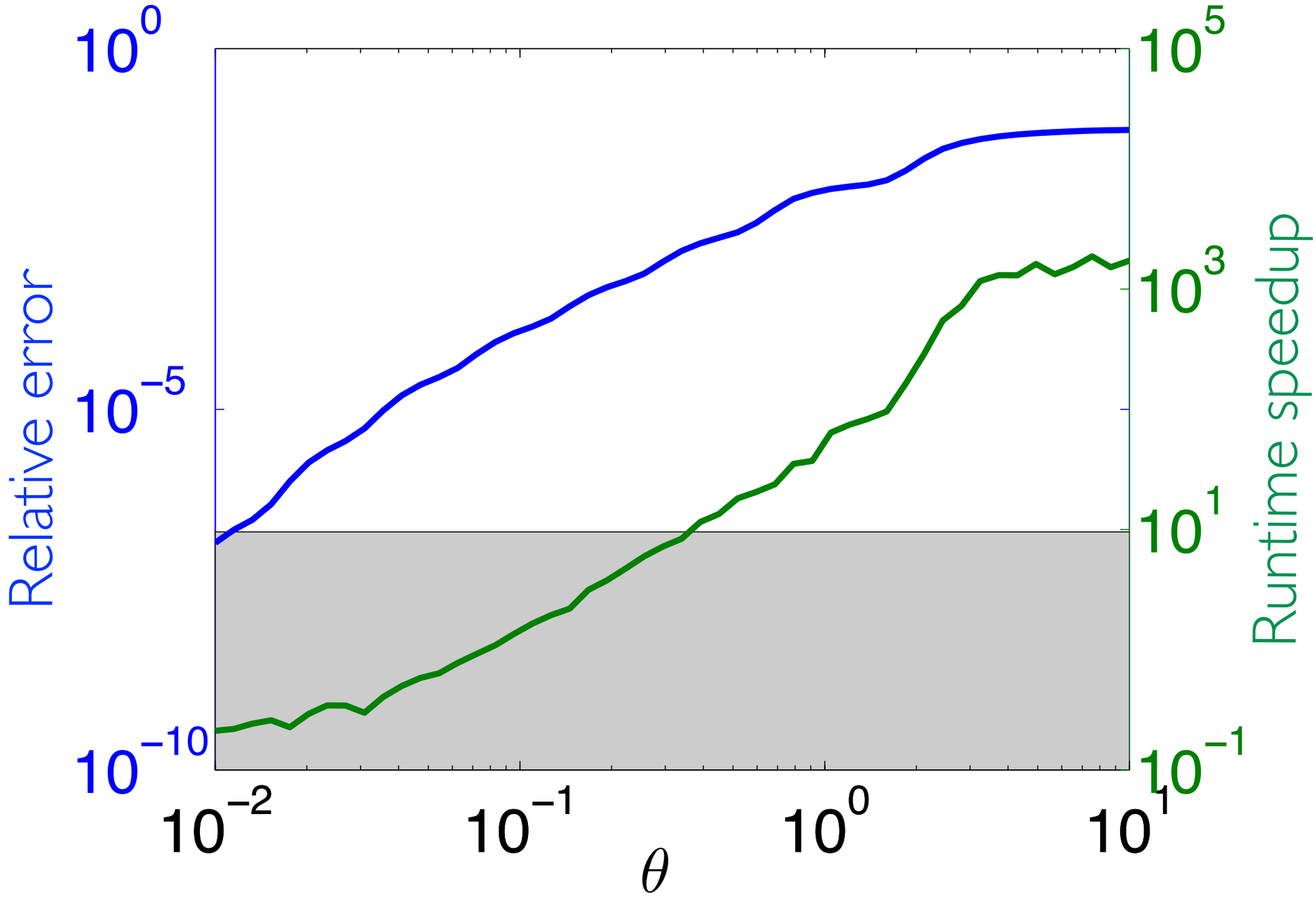
$$\frac{l}{D} < \theta$$

where θ is a user parameter, that controls the approximation:

- smaller θ gives more accurate prediction,
- larger θ gives better speedup.

Behavior with respect to θ

Change in error and speedup with respect to exact.
Bigger θ: faster computation, but larger error.



Fast multipole methods

Properties:

☺ Time complexity $\mathcal{O}(N)$.

☺ Well defined error bounds.

☹ Expansion for each new kernel needs to be derived separately. The performance may vary.

☹ Computational cost grows exponentially with number of dimensions.

Extensions:

- Fast Gauss Transform: deals exclusively with Gaussian kernel.

  (Greengard and Strain, '91; Yang et al, '03;)

- Different expansions (Taylor, Hermite, interpolation, SVD, etc.)
- Were first used in the context of NLE by de Freitas et al., '06, but their description was limited to one experiment.

Fast multipole methods (Greengard and Rokhlin '87)

Approximate the interactions of the form:

$$Q(\mathbf{x}_n) = \sum_{m=1}^{N} q_m K(\|(\mathbf{x}_n - \mathbf{x}_m)/\sigma\|^2)$$

The idea is to do a series expansion of the kernel K, such that the sum decouples over $\mathbf{x}_n$ and $\mathbf{x}_m$:

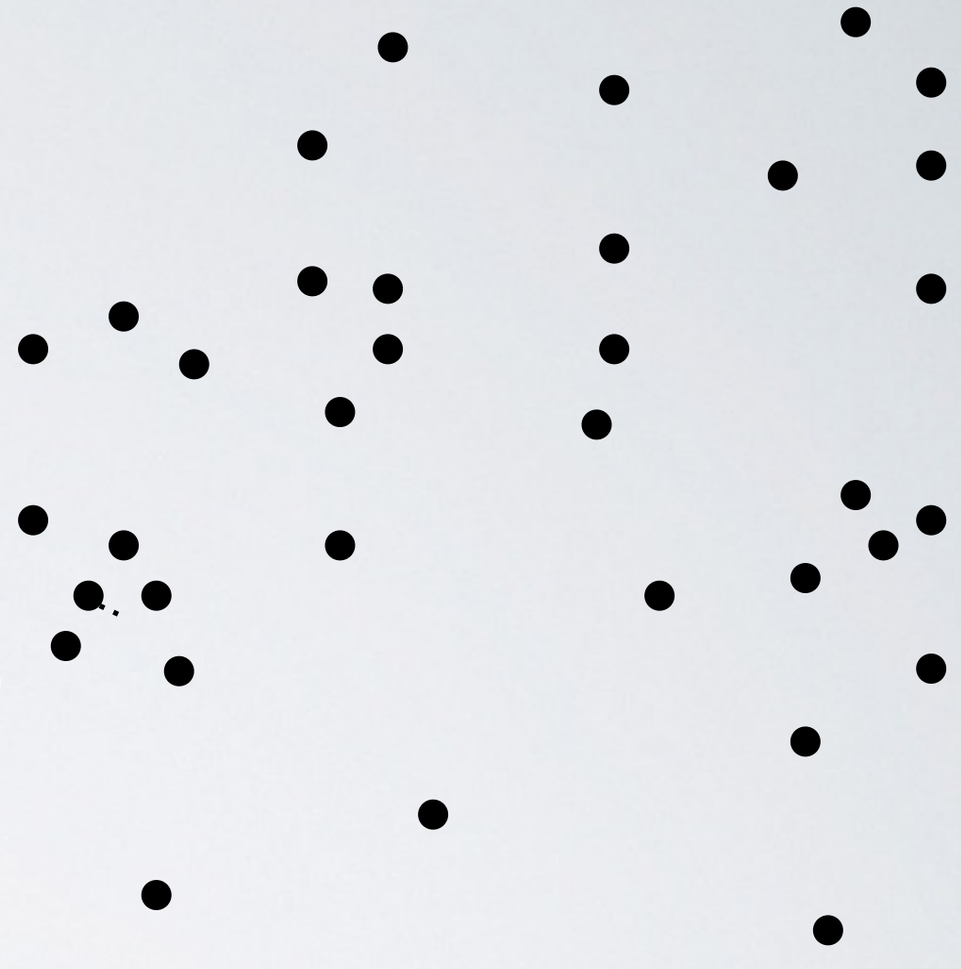
$$K(\|(\mathbf{x}_n - \mathbf{x}_m)/\sigma\|^2) = \sum_{\boldsymbol{\alpha} \geq 0} f_{\boldsymbol{\alpha}}(\mathbf{x}_n)\, g_{\boldsymbol{\alpha}}(\mathbf{x}_m)$$

we used multi-index notation $\boldsymbol{\alpha} \geq 0 \Rightarrow \alpha_1, \ldots, \alpha_d \geq 0$

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



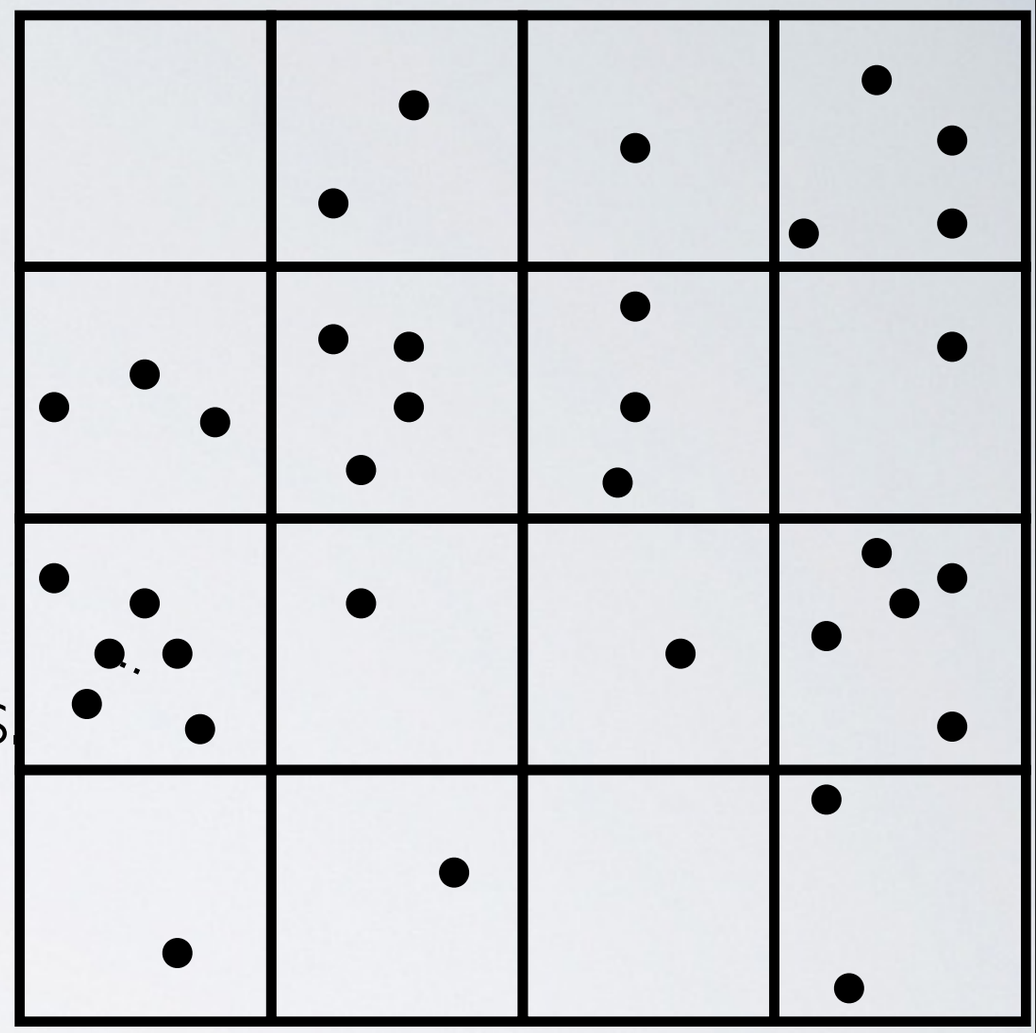
Parameters of the approximation:

- p number of terms in the expansion,
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



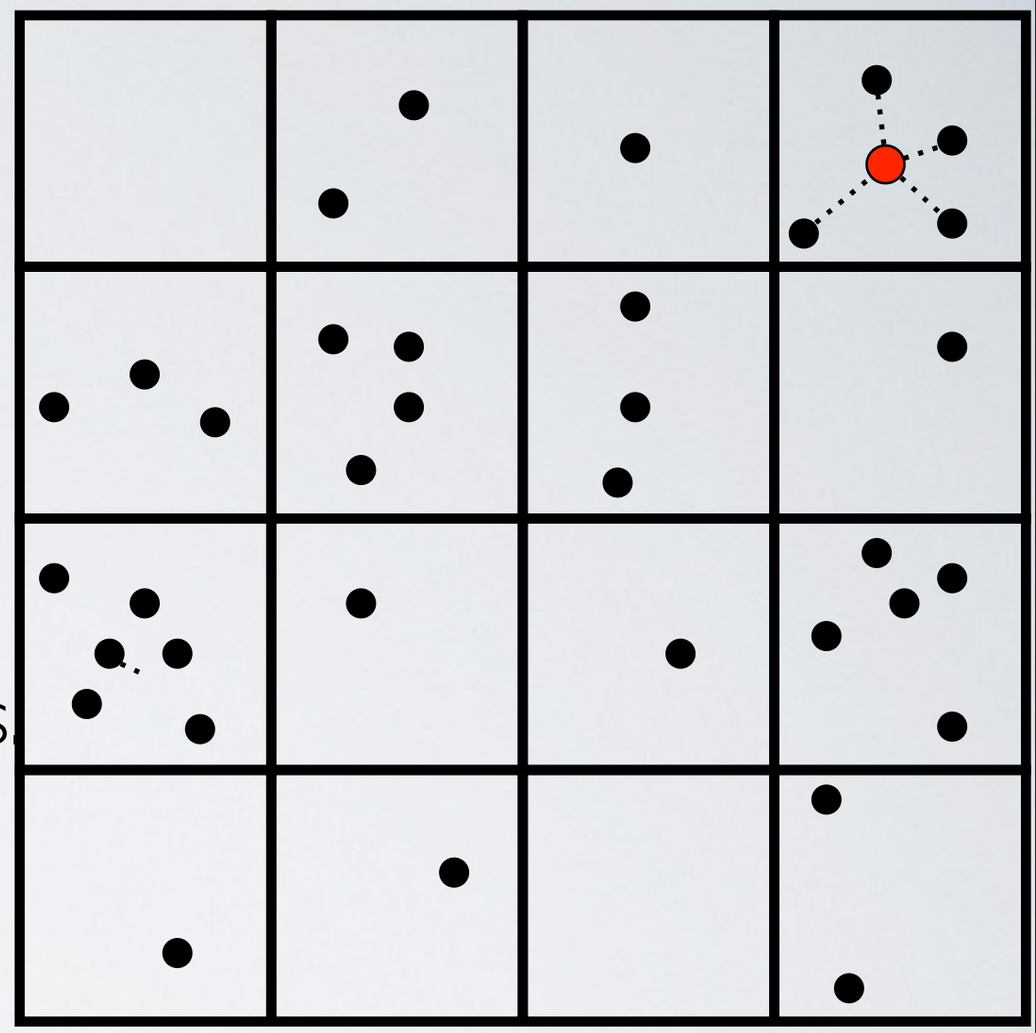
Parameters of the approximation:

- p number of terms in the expansion,
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



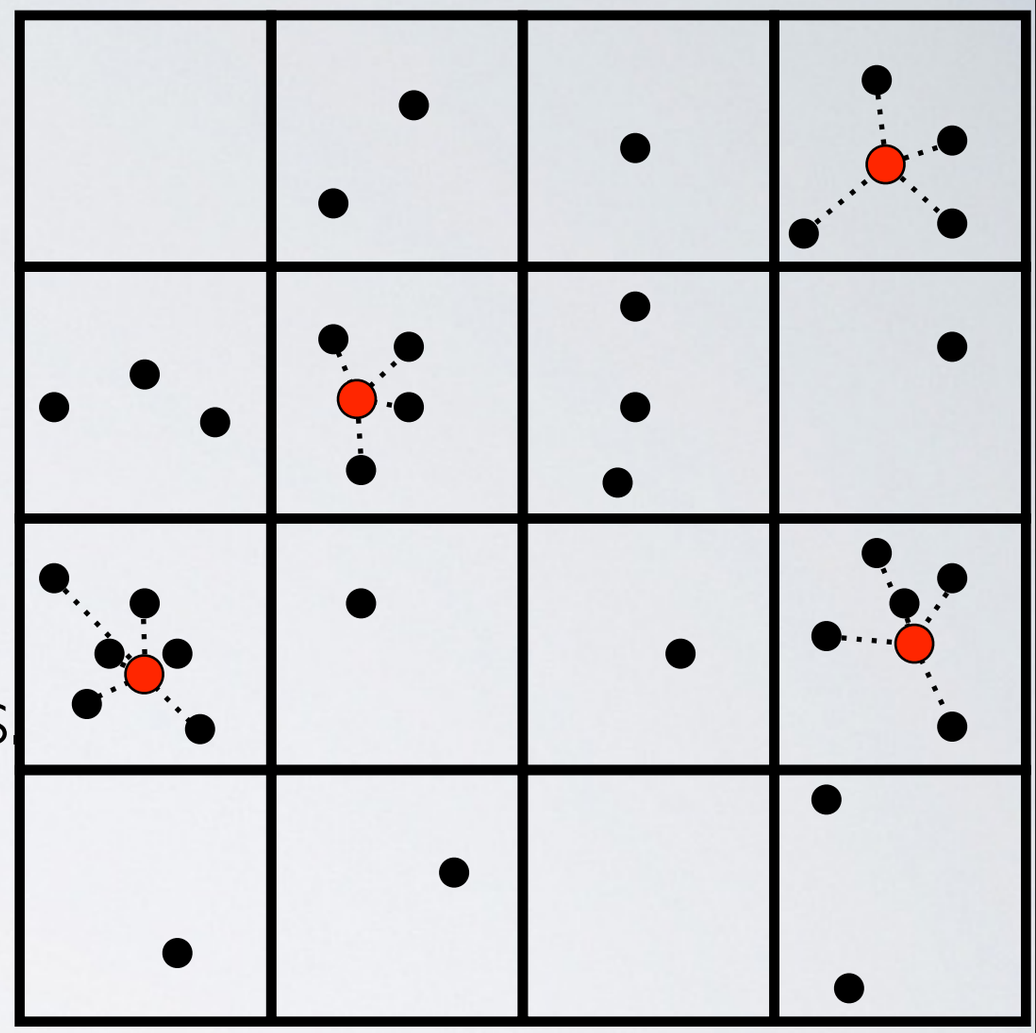
Parameters of the approximation:

- p number of terms in the expansion,
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



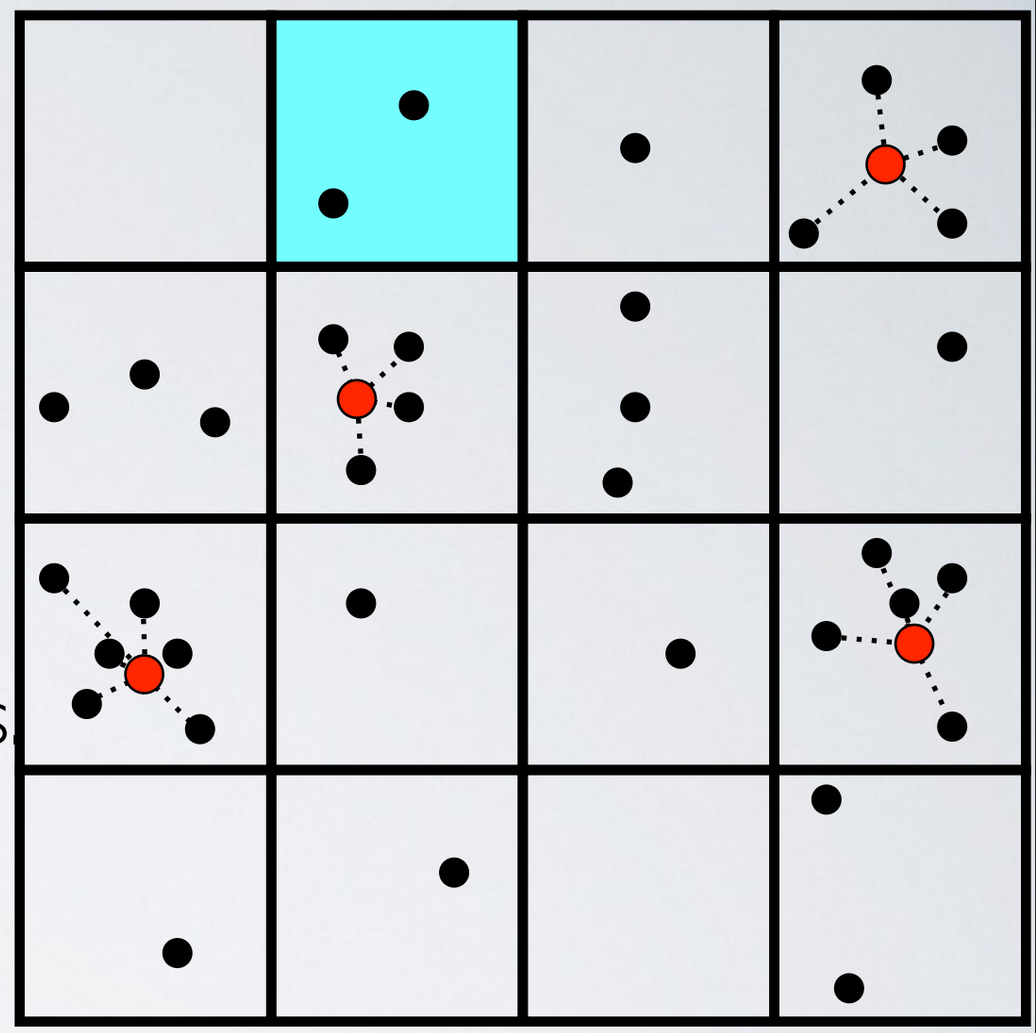
Parameters of the approximation:

- p number of terms in the expansion,
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



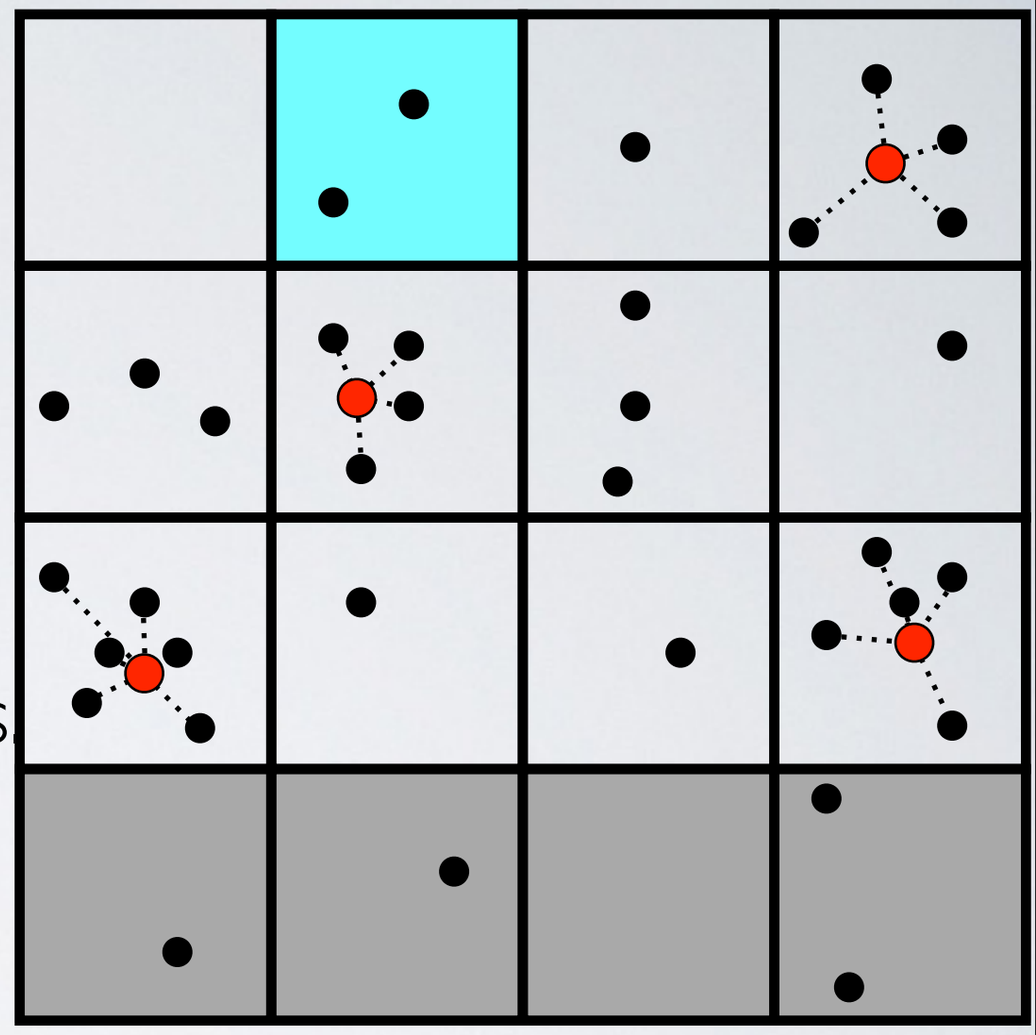
Parameters of the approximation:

- p number of terms in the expansion,
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



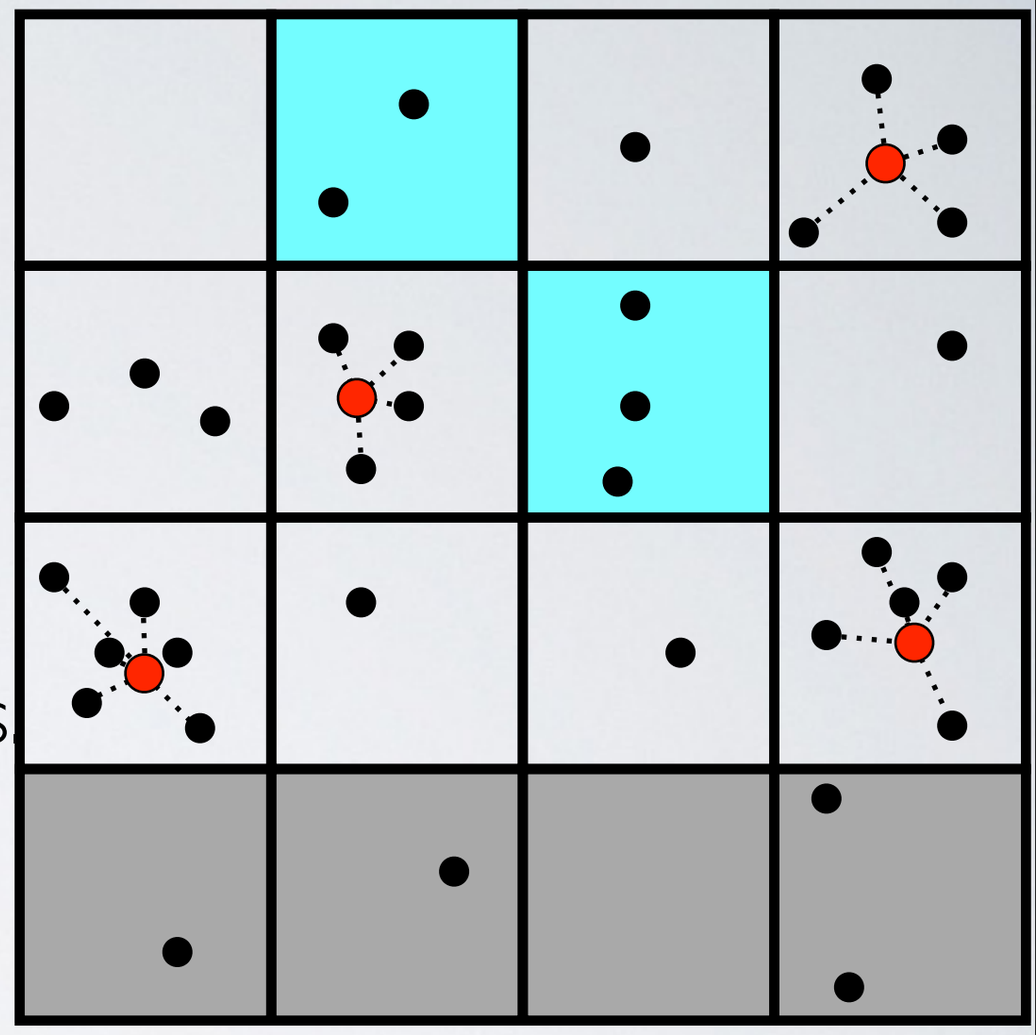
Parameters of the approximation:

- p number of terms in the expansion,
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



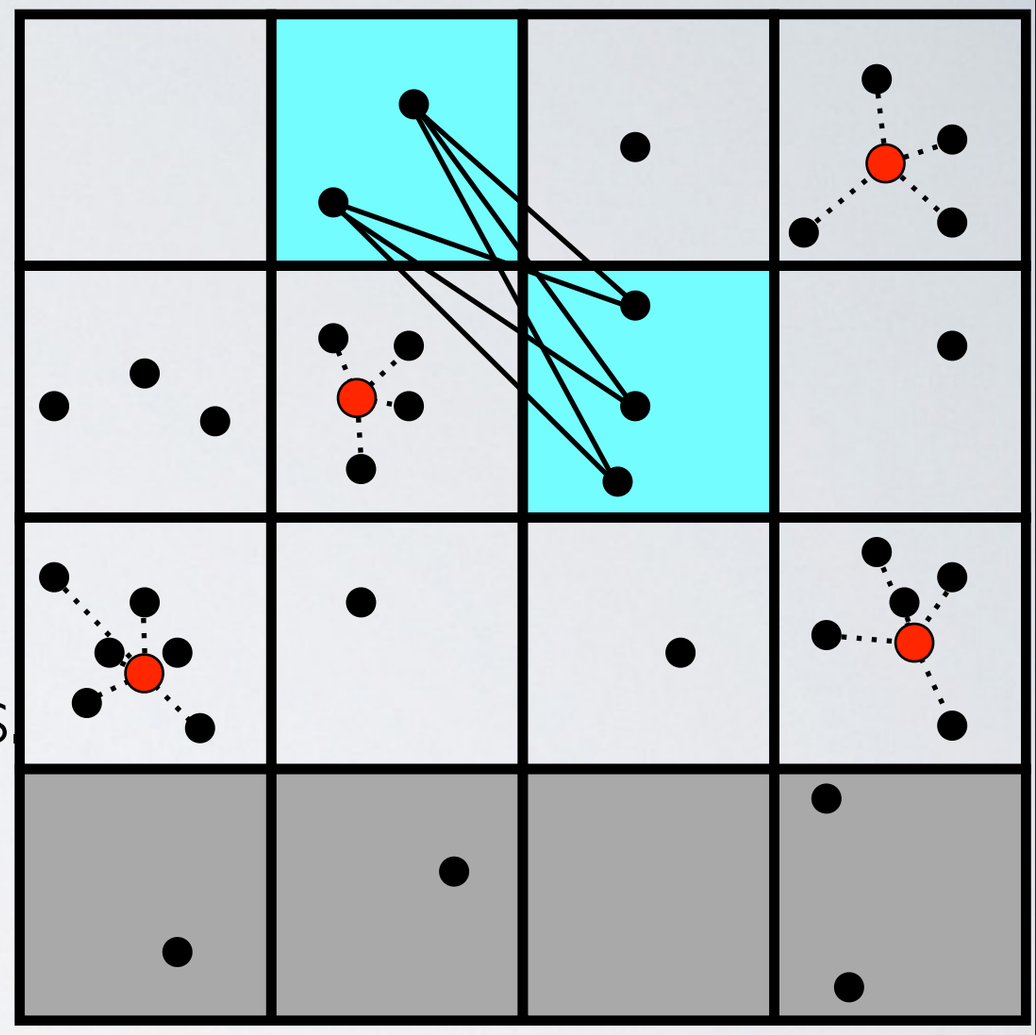
Parameters of the approximation:

- p number of terms in the expansion,
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



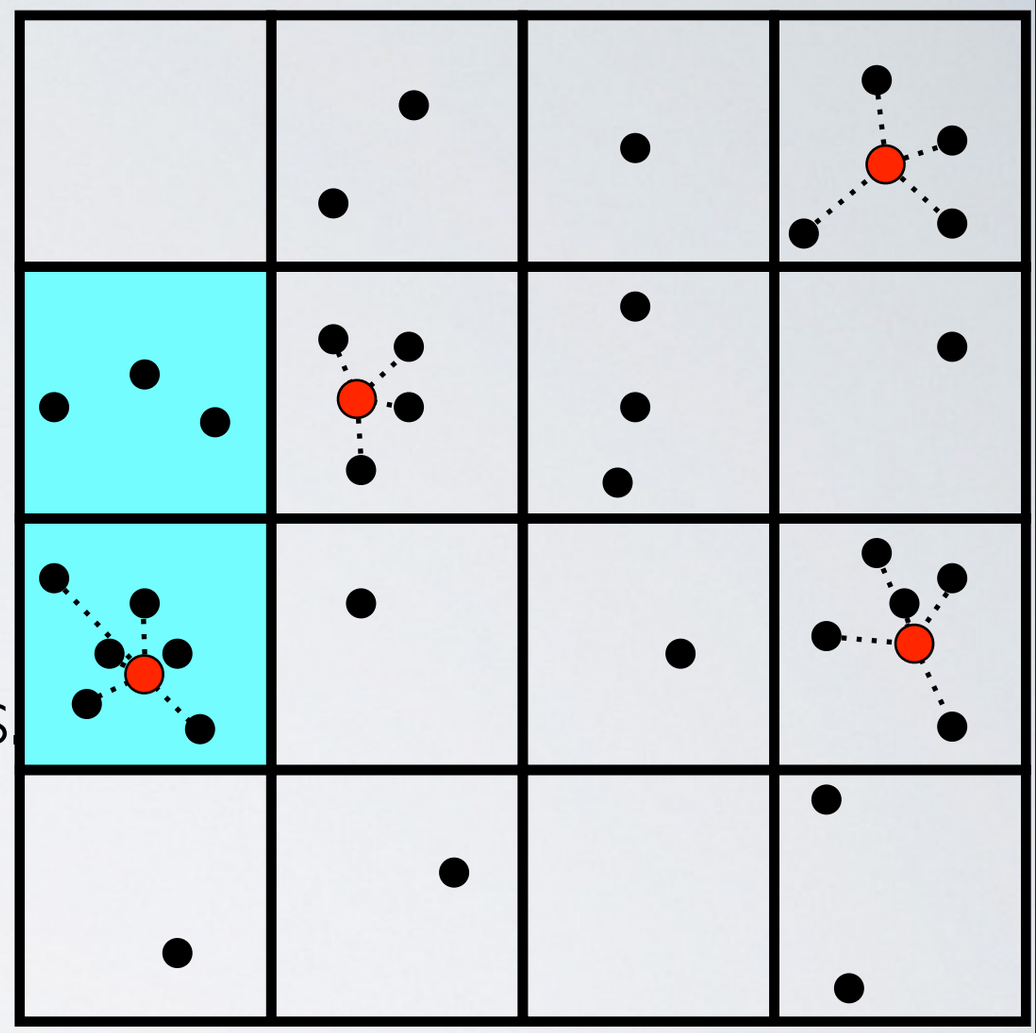
Parameters of the approximation:

- p number of terms in the expansion,
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



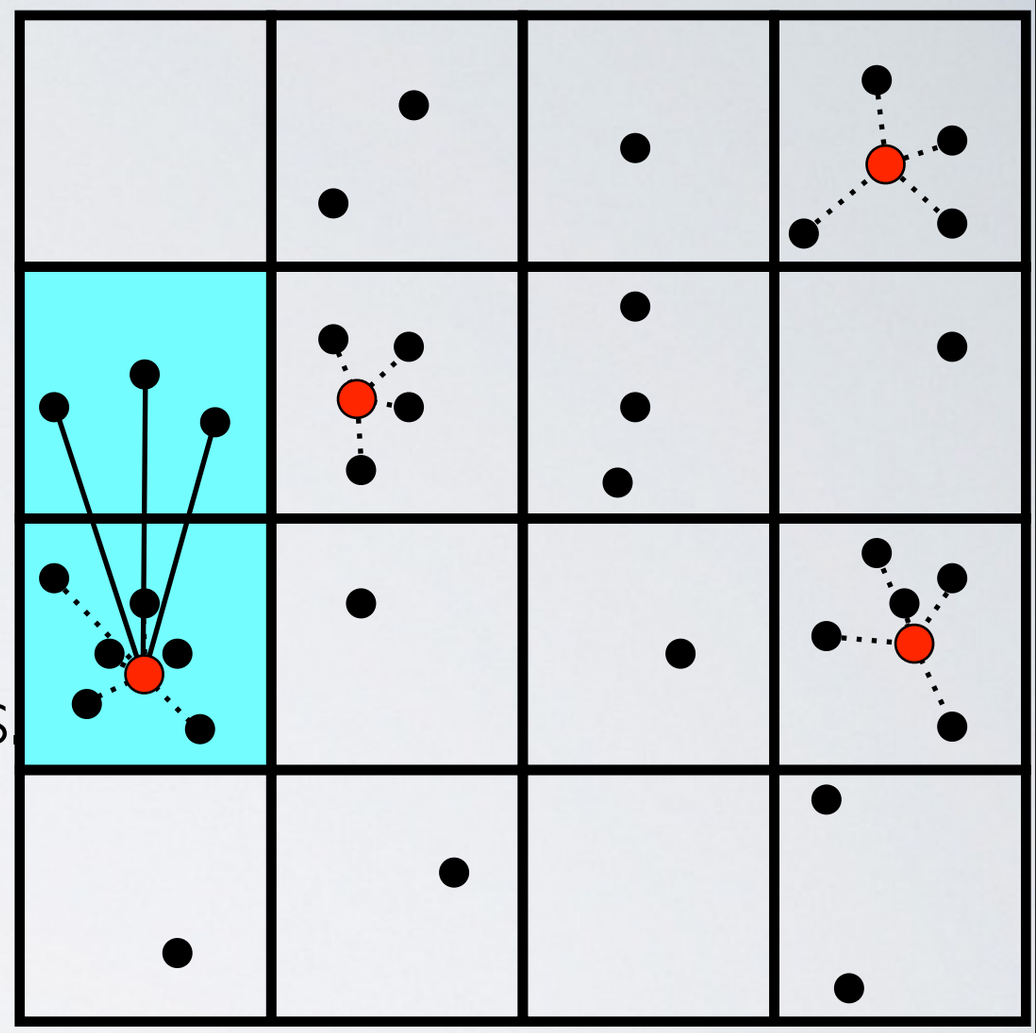
Parameters of the approximation:

- p number of terms in the expansion,
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



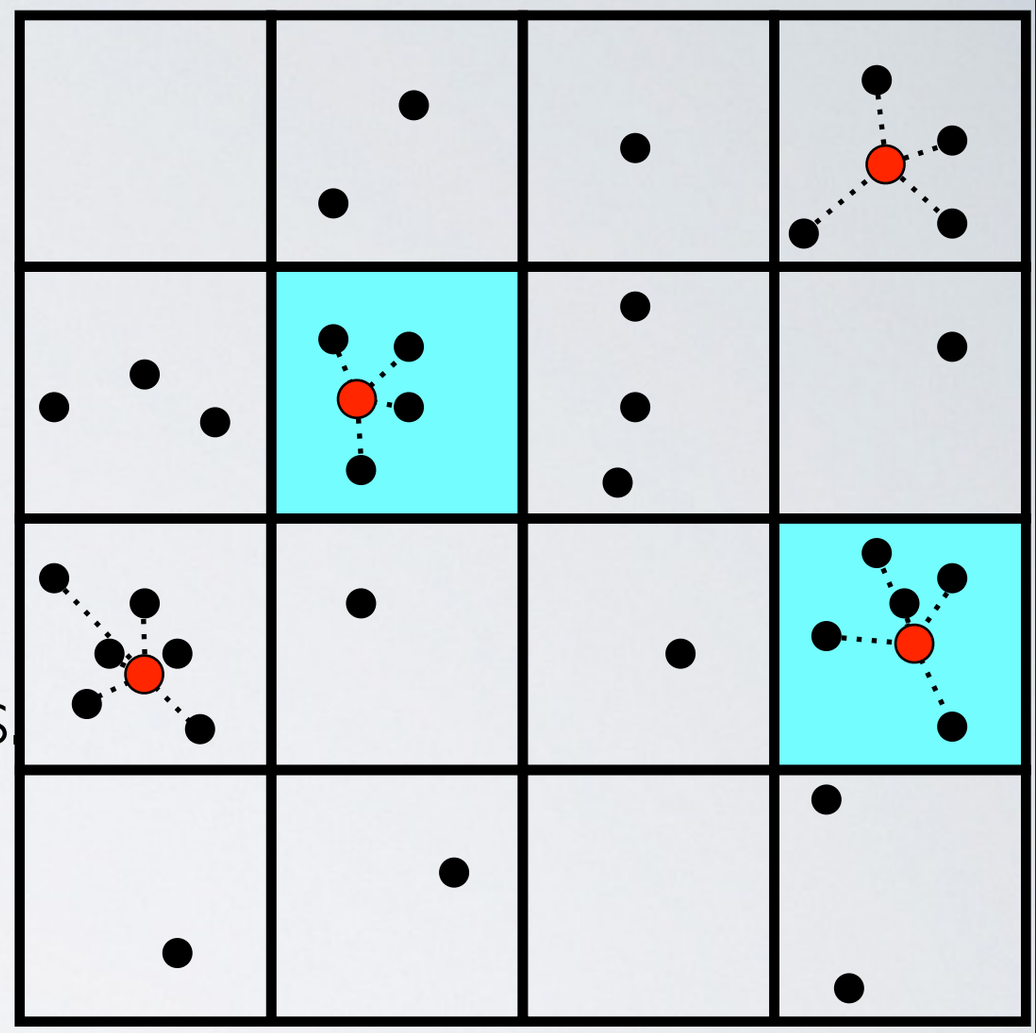
Parameters of the approximation:

- p number of terms in the expansion,
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



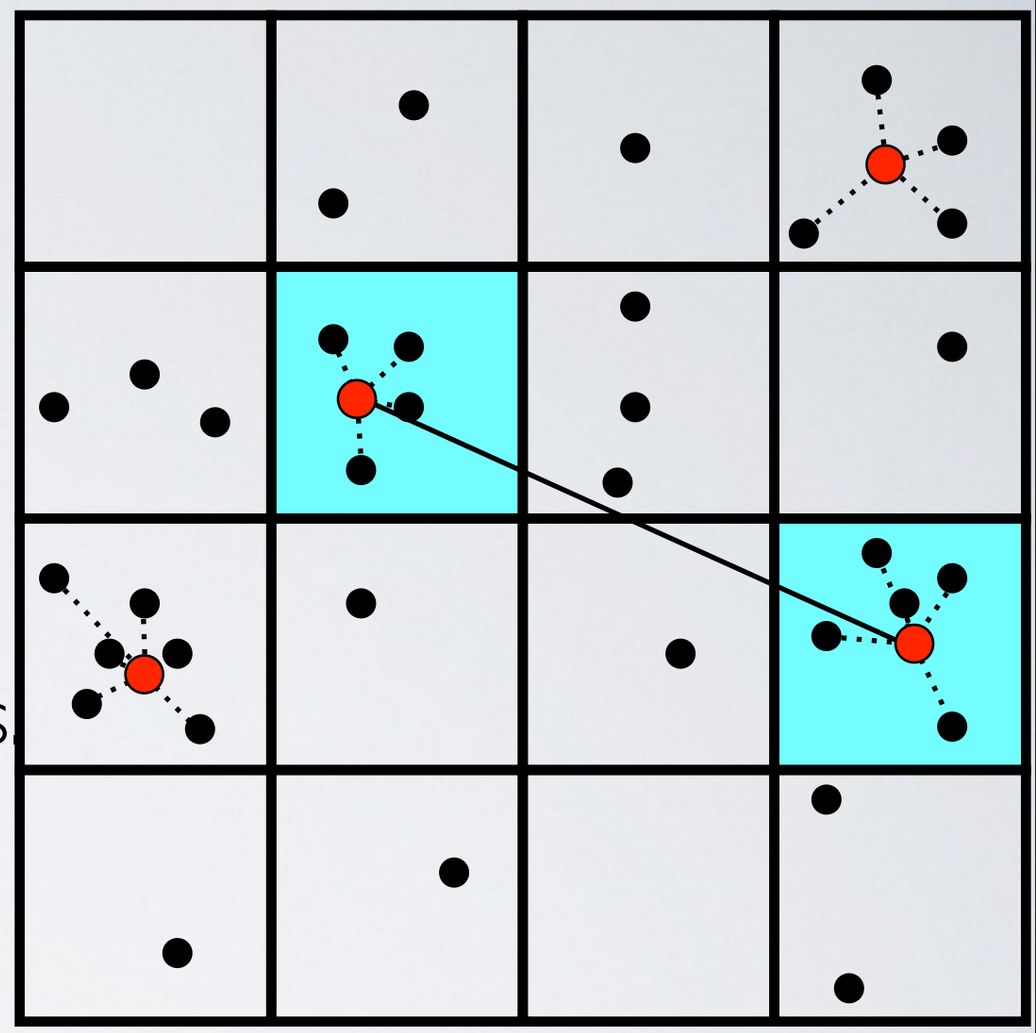
Parameters of the approximation:

- p number of terms in the expansion,
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



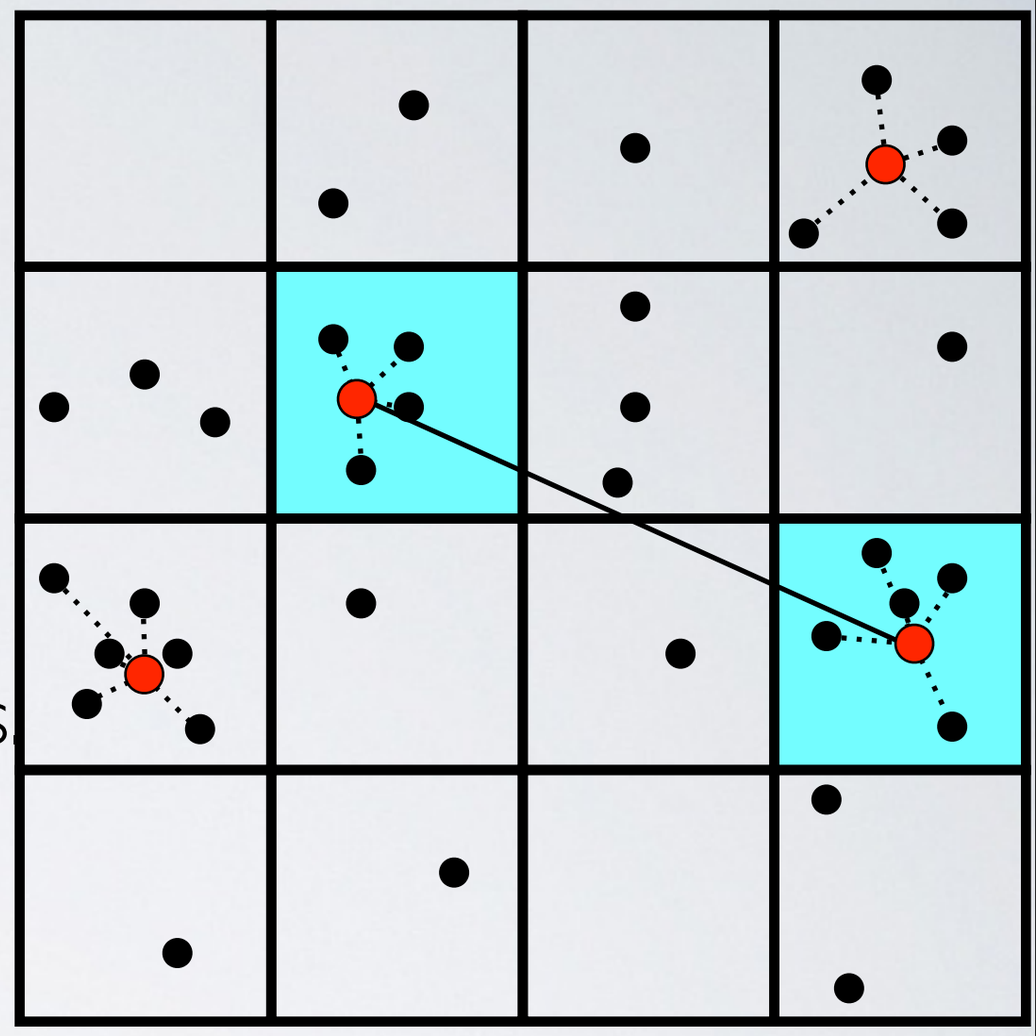
Parameters of the approximation:

- p number of terms in the expansion,
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



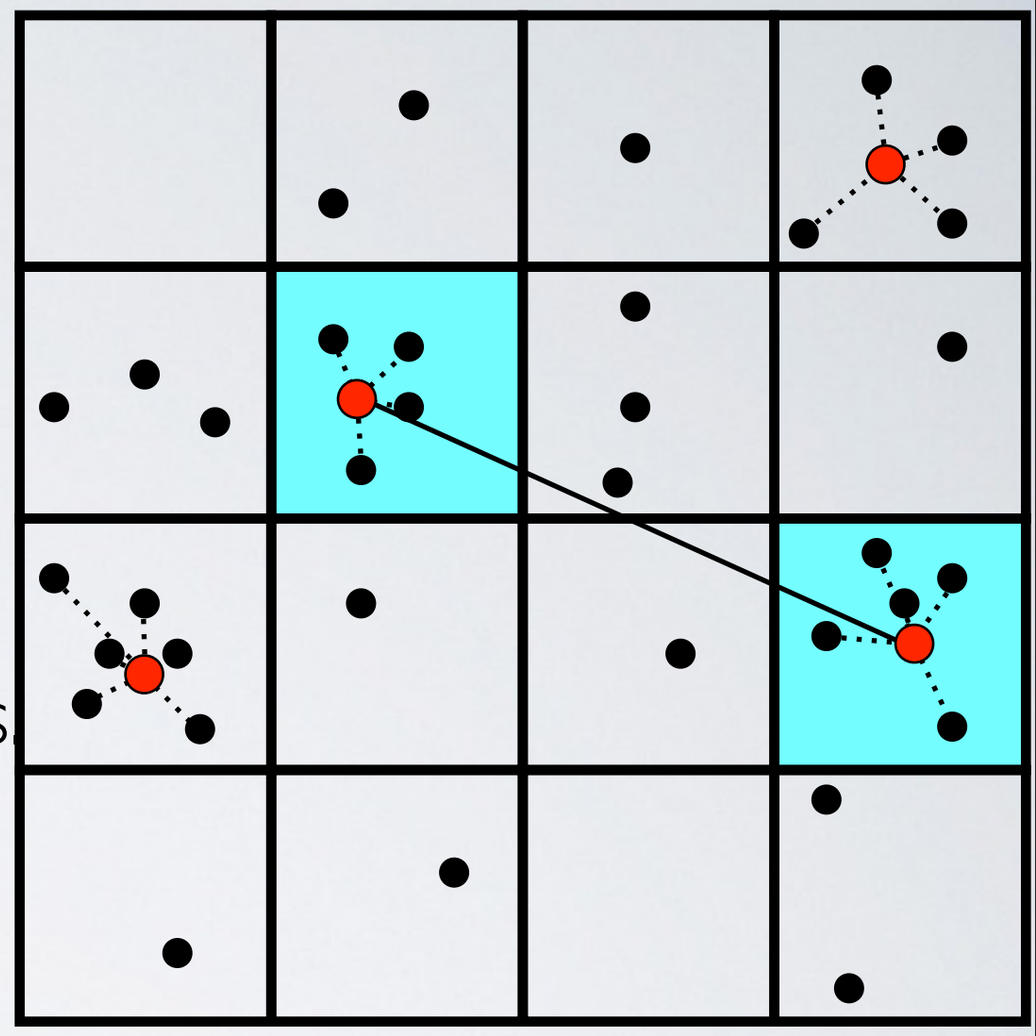
Parameters of the approximation:

- p number of terms in the expansion, ← use to control the accuracy
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

Fast Gauss Transform (Greengard and Strain, '91)

Algorithm:

1. Normalize the dataset to lie in a unit box.
2. Grid the box into smaller boxes (either uniformly or based on density),
3. A lot of points in a cell $\Rightarrow$ do a series expansion around the center of the box.
4. Ignore interactions between distant boxes.
5. Compute the interaction:
   - few points in the box $\Rightarrow$ exactly,
   - a lot of points $\Rightarrow$ use center of mass.



Parameters of the approximation:

- p number of terms in the expansion, ← use to control the accuracy
- M_0 number of points in the box for the expansion to occur,
- r size of the grid,
- K number of boxes to which we compute the interaction.

} fixed

Application of *N*-Body to NLE

- We can approximate the following interaction with *N*-Body methods

$$S(\mathbf{x}_n) = \sum_{m=1}^{N} K(||\mathbf{x}_n - \mathbf{x}_m||^2) \qquad S^x(\mathbf{x}_n) = \sum_{m=1}^{N} \mathbf{x}_m K(||\mathbf{x}_n - \mathbf{x}_m||^2)$$

- The objective function and the gradient of EE:

$$E_{EE}(\mathbf{X}) = \sum_{n,m=1}^{N} w_{nm} \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2 + \lambda \sum_{n=1}^{N} S(\mathbf{x}_n)$$

$$G_{EE}(\mathbf{X}) = 4\mathbf{X}\mathbf{L} - 4\lambda\mathbf{X}\,\mathrm{diag}\,(S(\mathbf{X})) + 4\lambda S^x(\mathbf{X})$$

- Given $S(\mathbf{x}_n)$and $S^x(\mathbf{x}_n)$, each term is can be computed in $\mathcal{O}(N)$.

- Objective function and the gradient of other NLE methods can be defined analogously.

NLE: Optimization (revisited)

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^{+}(\mathbf{X}) + \lambda E^{-}(\mathbf{X}) \quad \lambda \geq 0$$

- Optimization:
  - compute the gradient

$$\mathbf{G}_k = 4\mathbf{X}_k(\mathbf{L}^{+} - \mathbf{L}^{-})$$

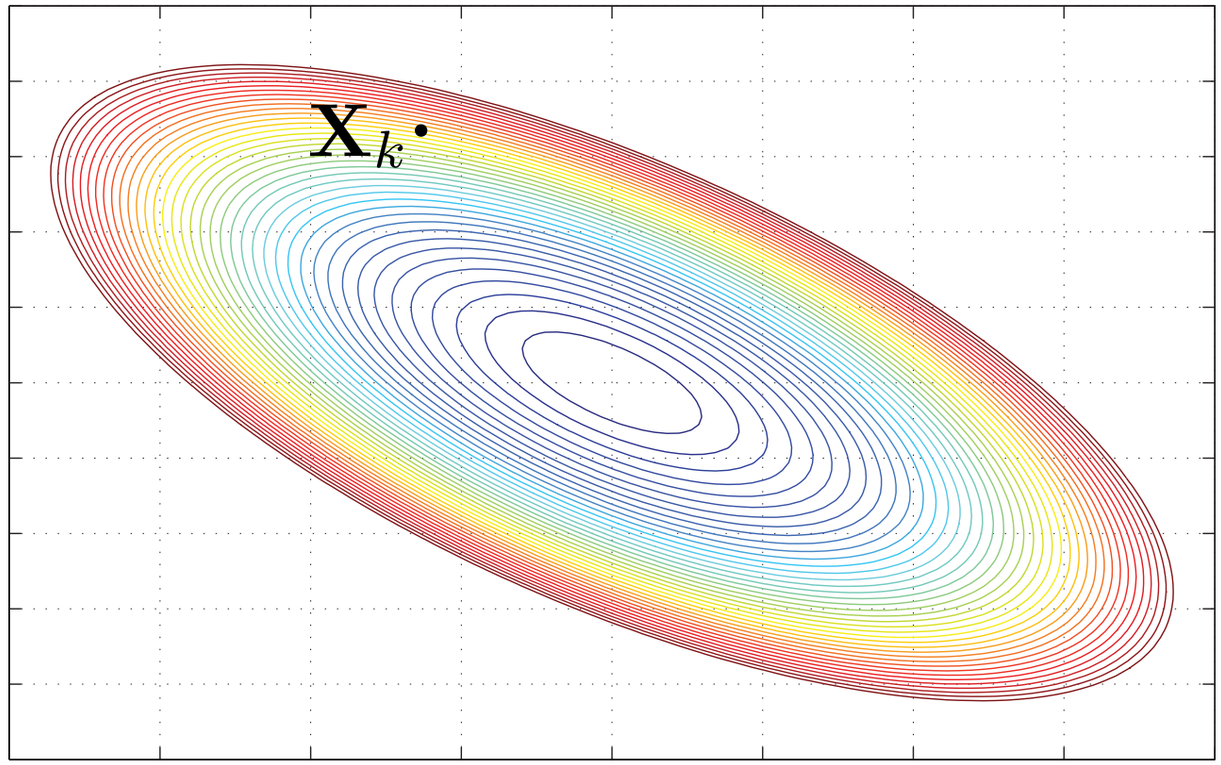
  - compute the direction

$$\mathbf{P}_k = -\mathbf{G}_k$$

  - compute new iteration $\mathbf{X}_{k+1}$ using a line search:

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \eta \mathbf{P}_k$$

  - repeat till convergence.



NLE: Optimization (revisited)

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^{+}(\mathbf{X}) + \lambda E^{-}(\mathbf{X}) \quad \lambda \geq 0$$

- Optimization:
  - compute the gradient

$$\mathbf{G}_k = 4\mathbf{X}_k(\mathbf{L}^{+} - \mathbf{L}^{-})$$

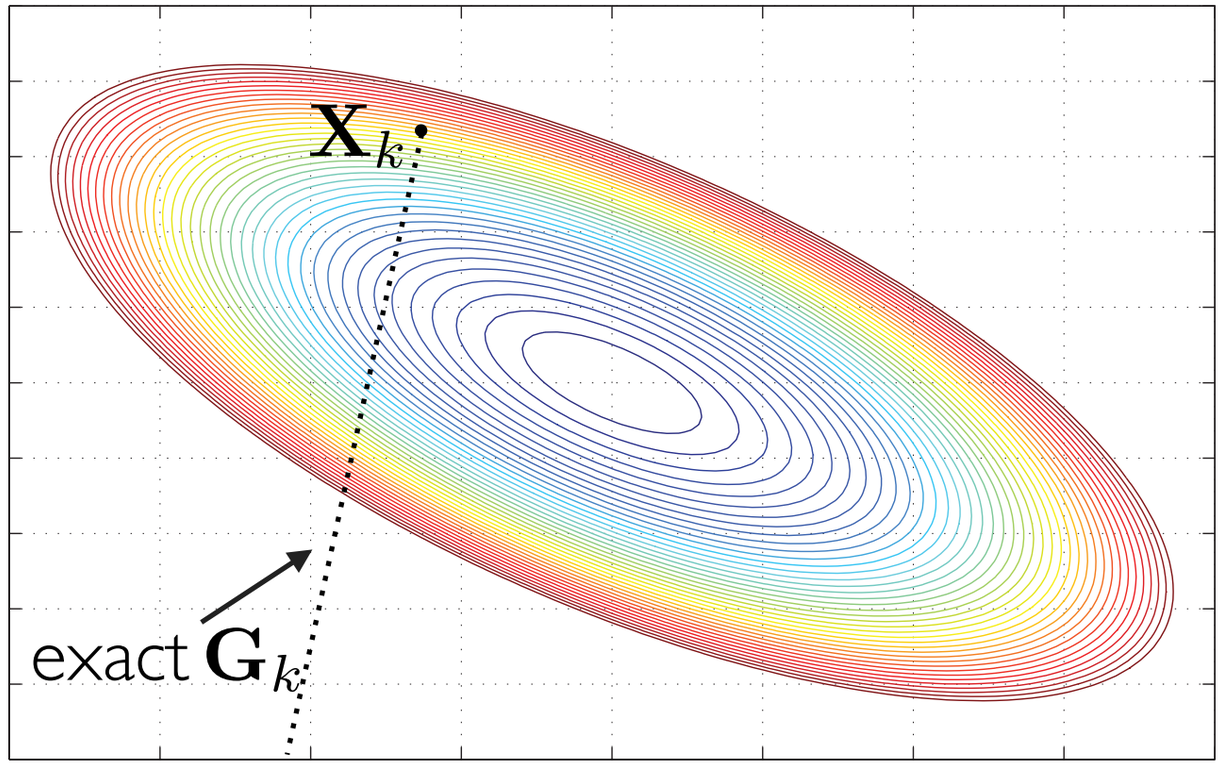
  - compute the direction

$$\mathbf{P}_k = -\mathbf{G}_k$$

  - compute new iteration $\mathbf{X}_{k+1}$ using a line search:

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \eta \mathbf{P}_k$$

  - repeat till convergence.



NLE: Optimization (revisited)

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$

- Optimization:
  - compute the gradient with accuracy p (approximate)

$$\mathbf{G}_k = 4\mathbf{X}_k(\mathbf{L}^+ - \mathbf{L}^-)$$

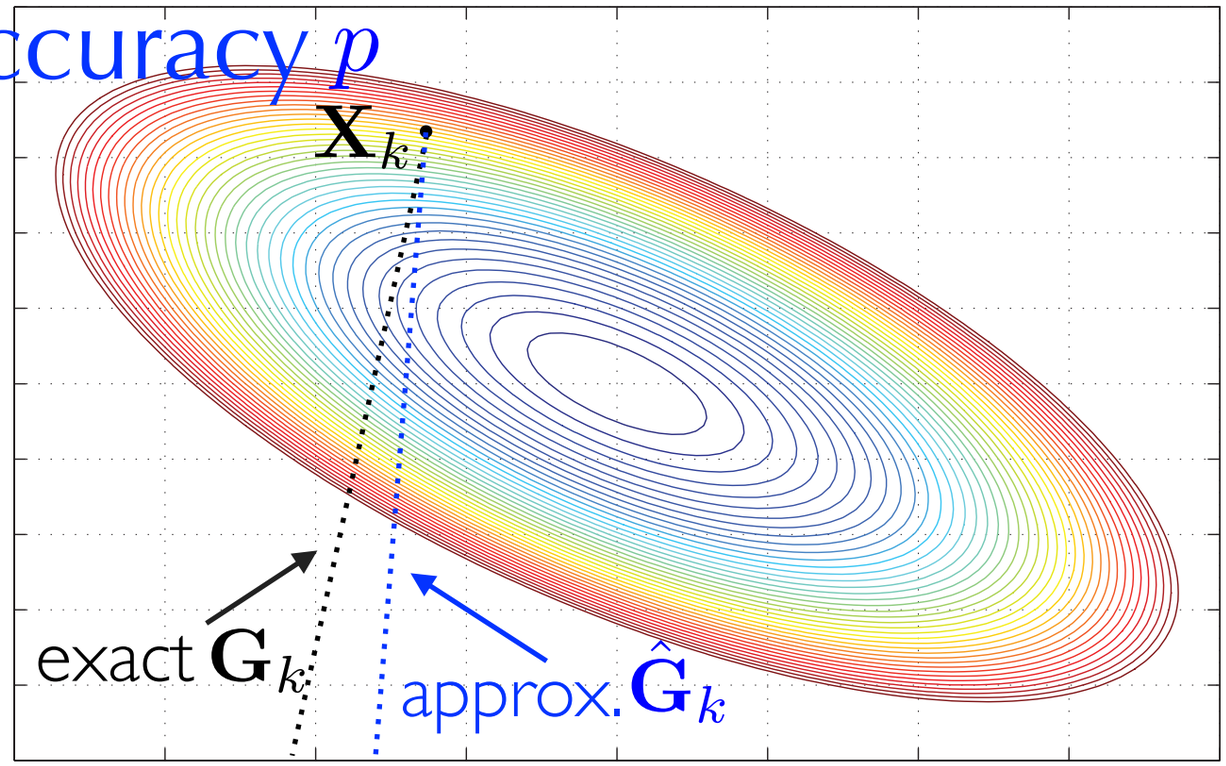
  - compute the direction

$$\mathbf{P}_k = -\mathbf{G}_k$$

  - compute new iteration $\mathbf{X}_{k+1}$ using a line search:

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \eta \mathbf{P}_k$$

  - repeat till convergence.



NLE: Optimization (revisited)

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$

- Optimization:
  - compute the gradient with accuracy p (approximate)

$$\mathbf{G}_k = 4\mathbf{X}_k(\mathbf{L}^+ - \mathbf{L}^-)$$

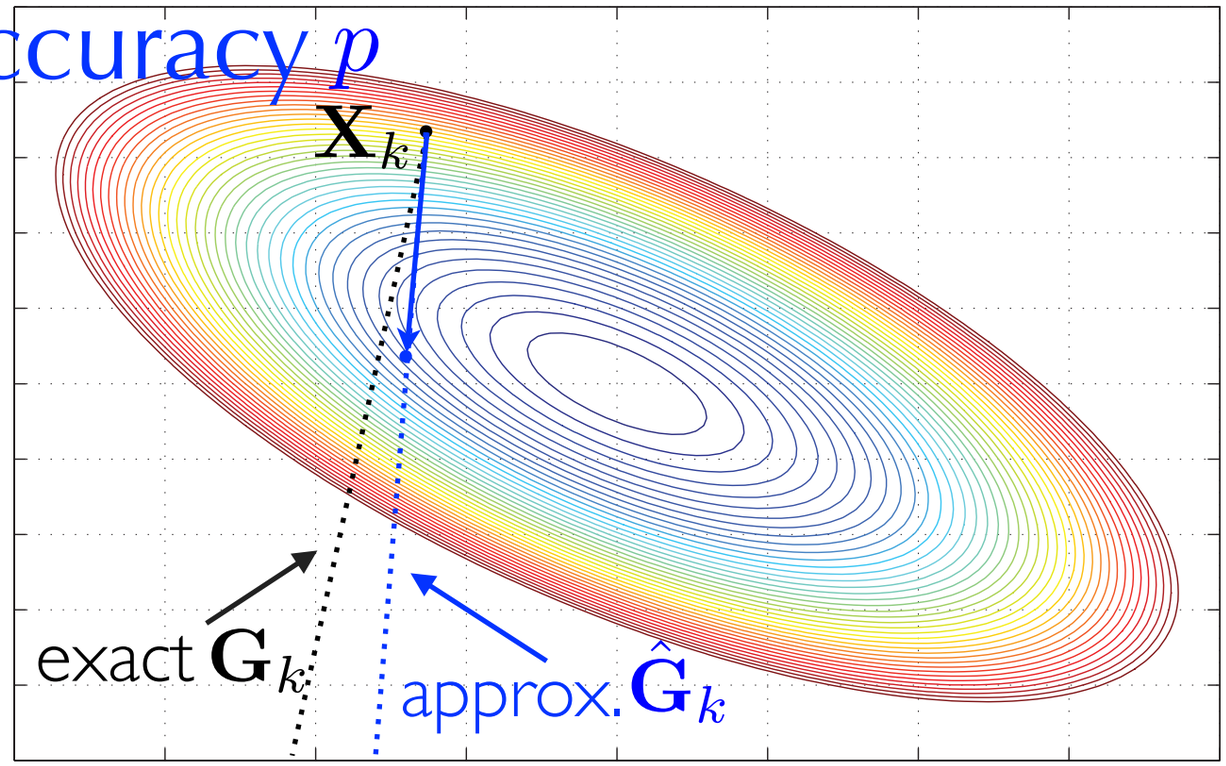
  - compute the direction

$$\mathbf{P}_k = -\mathbf{G}_k$$

  - compute new iteration $\mathbf{X}_{k+1}$ using a line search:

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \eta\mathbf{P}_k$$

  - repeat till convergence.



NLE: Optimization (revisited)

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$

- Optimization:
  - compute the gradient with accuracy p (approximate)

$$\mathbf{G}_k = 4\mathbf{X}_k(\mathbf{L}^+ - \mathbf{L}^-)$$

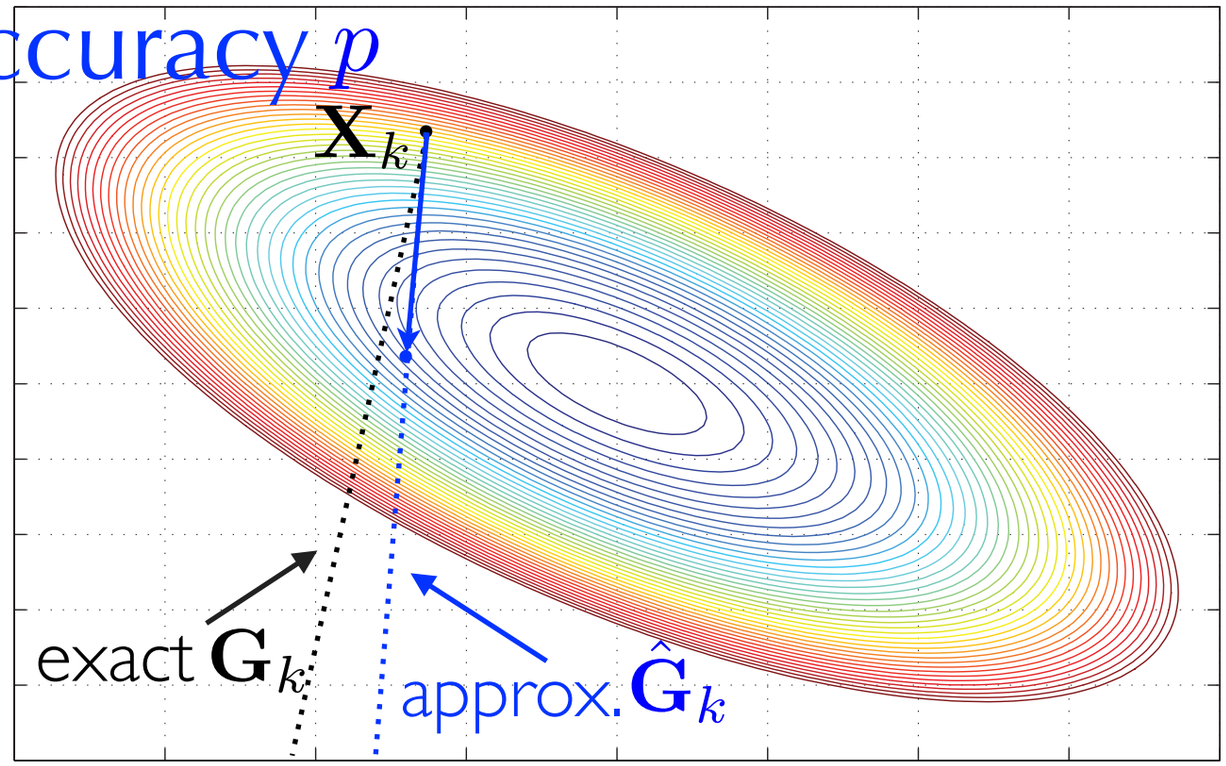
  - compute the direction

$$\mathbf{P}_k = -\mathbf{G}_k$$

  - compute new iteration $\mathbf{X}_{k+1}$ using a line search:

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \eta \mathbf{P}_k$$

  - repeat till convergence.



NLE: Optimization (revisited)

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$

- Optimization:
  - compute the gradient with accuracy p (approximate)

$$\mathbf{G}_k = 4\mathbf{X}_k(\mathbf{L}^+ - \mathbf{L}^-)$$

  - compute the direction

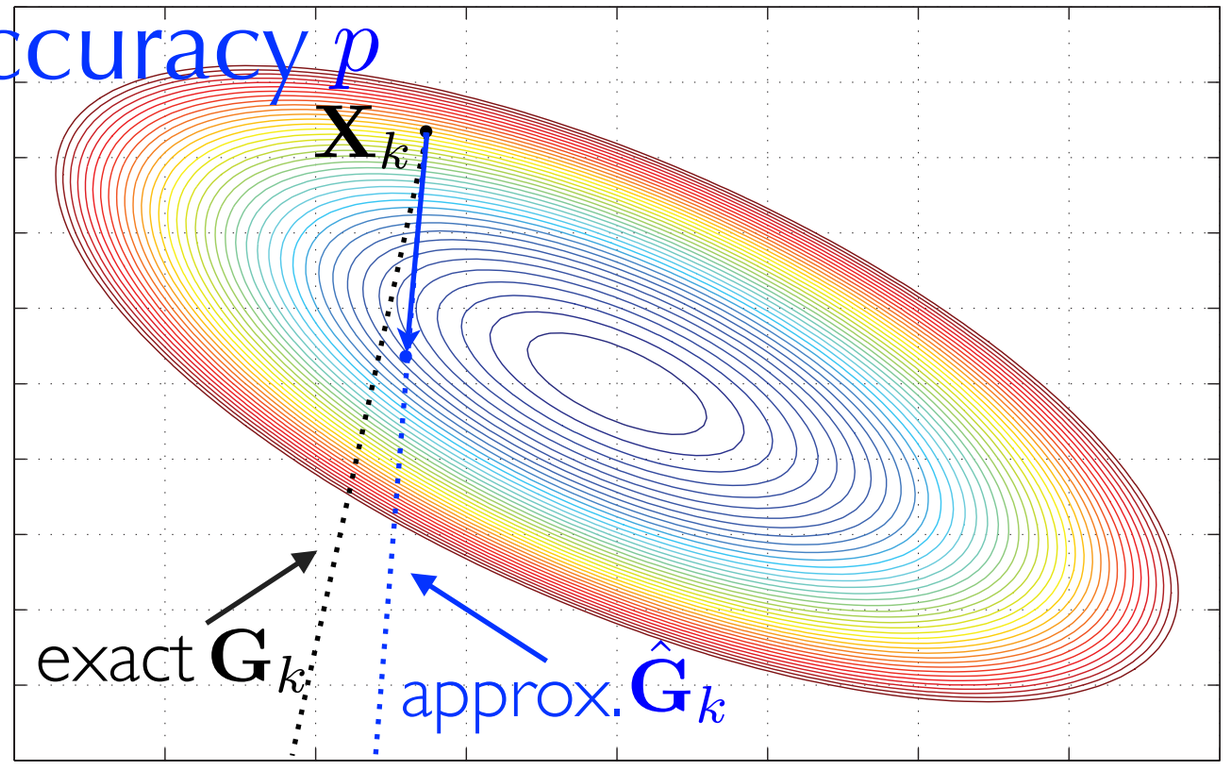
$$\mathbf{P}_k = -\mathbf{G}_k$$

  - compute new iteration $\mathbf{X}_{k+1}$ using a line search:

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \eta_0 \mathbf{P}_k$$

(use fixed step)

  - repeat till convergence.



NLE: Optimization (revisited)

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$

- Optimization:
  - compute the gradient with accuracy p (approximate)

$$\mathbf{G}_k = 4\mathbf{X}_k(\mathbf{L}^+ - \mathbf{L}^-)$$

  - compute the direction

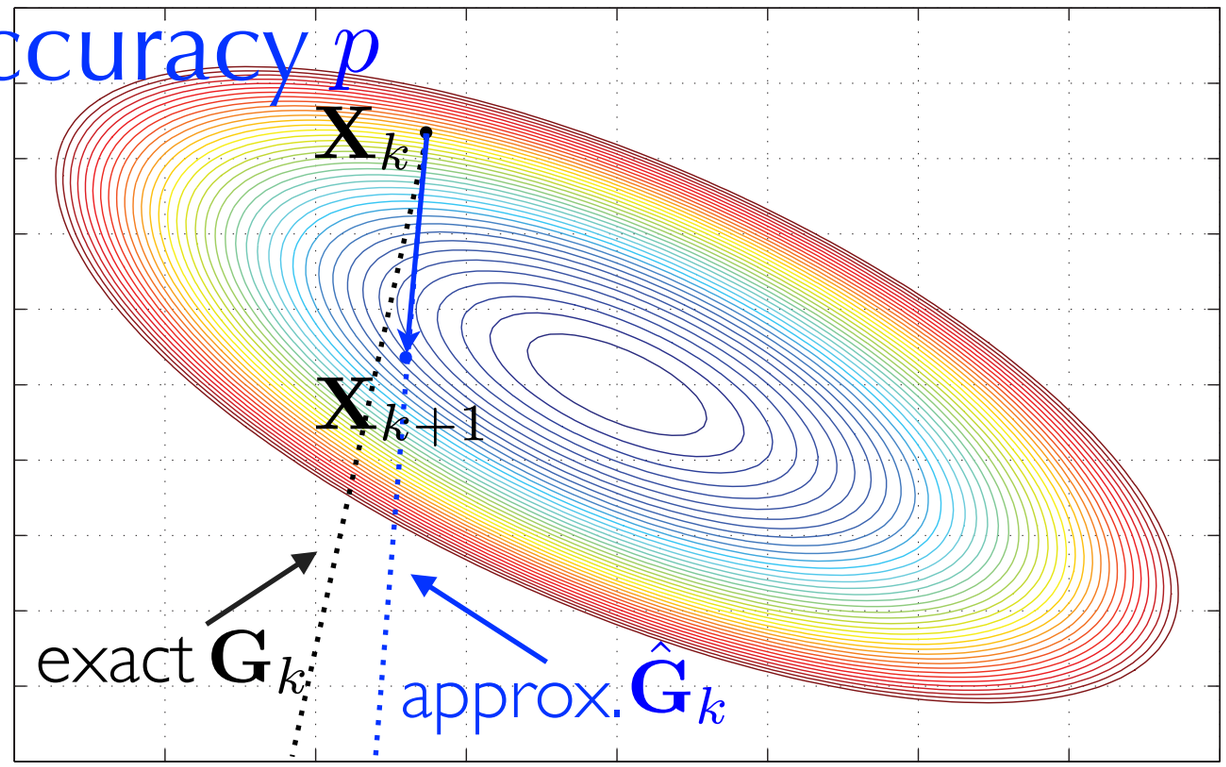
$$\mathbf{P}_k = -\mathbf{G}_k$$

  - compute new iteration $\mathbf{X}_{k+1}$ using a line search:

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \eta_0 \mathbf{P}_k$$

(use fixed step)

  - repeat till convergence.



NLE: Optimization (revisited)

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$

- Optimization:
  - compute the gradient with accuracy p (approximate)

$$\mathbf{G}_k = 4\mathbf{X}_k(\mathbf{L}^+ - \mathbf{L}^-)$$

  - compute the direction

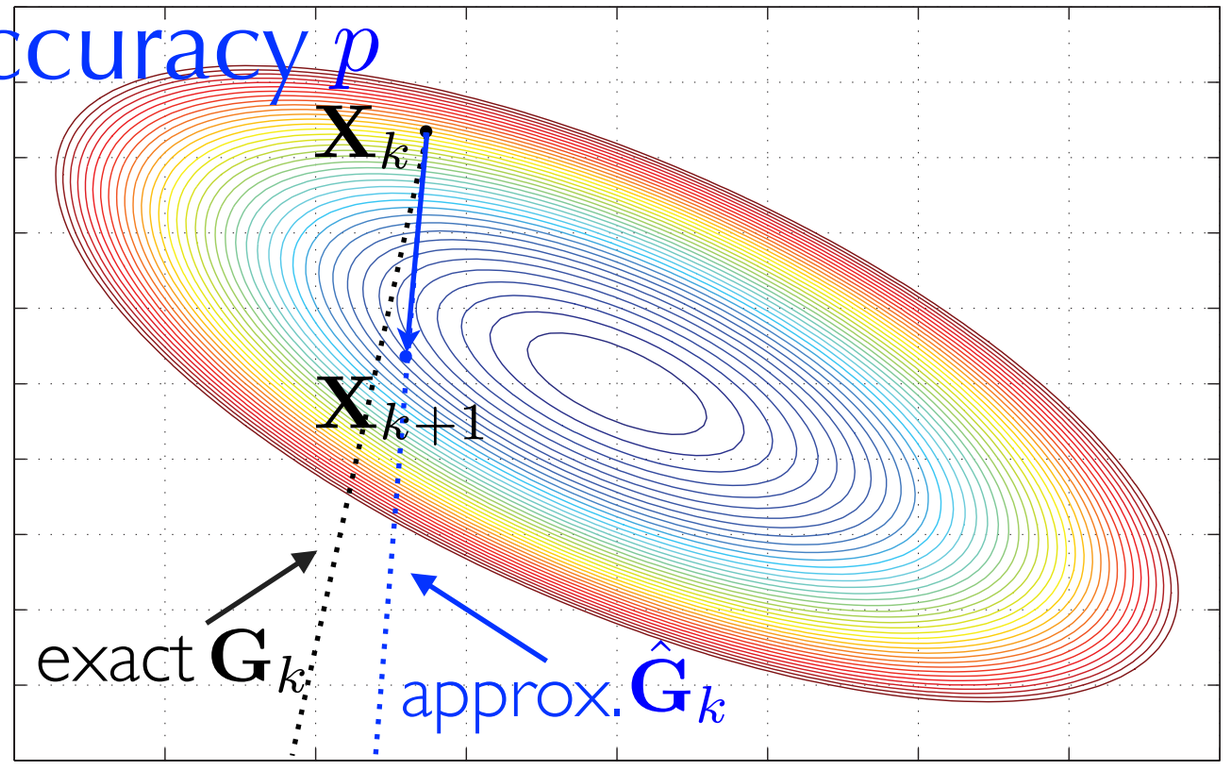
$$\mathbf{P}_k = -\mathbf{G}_k$$

  - compute new iteration $\mathbf{X}_{k+1}$ using a line search:

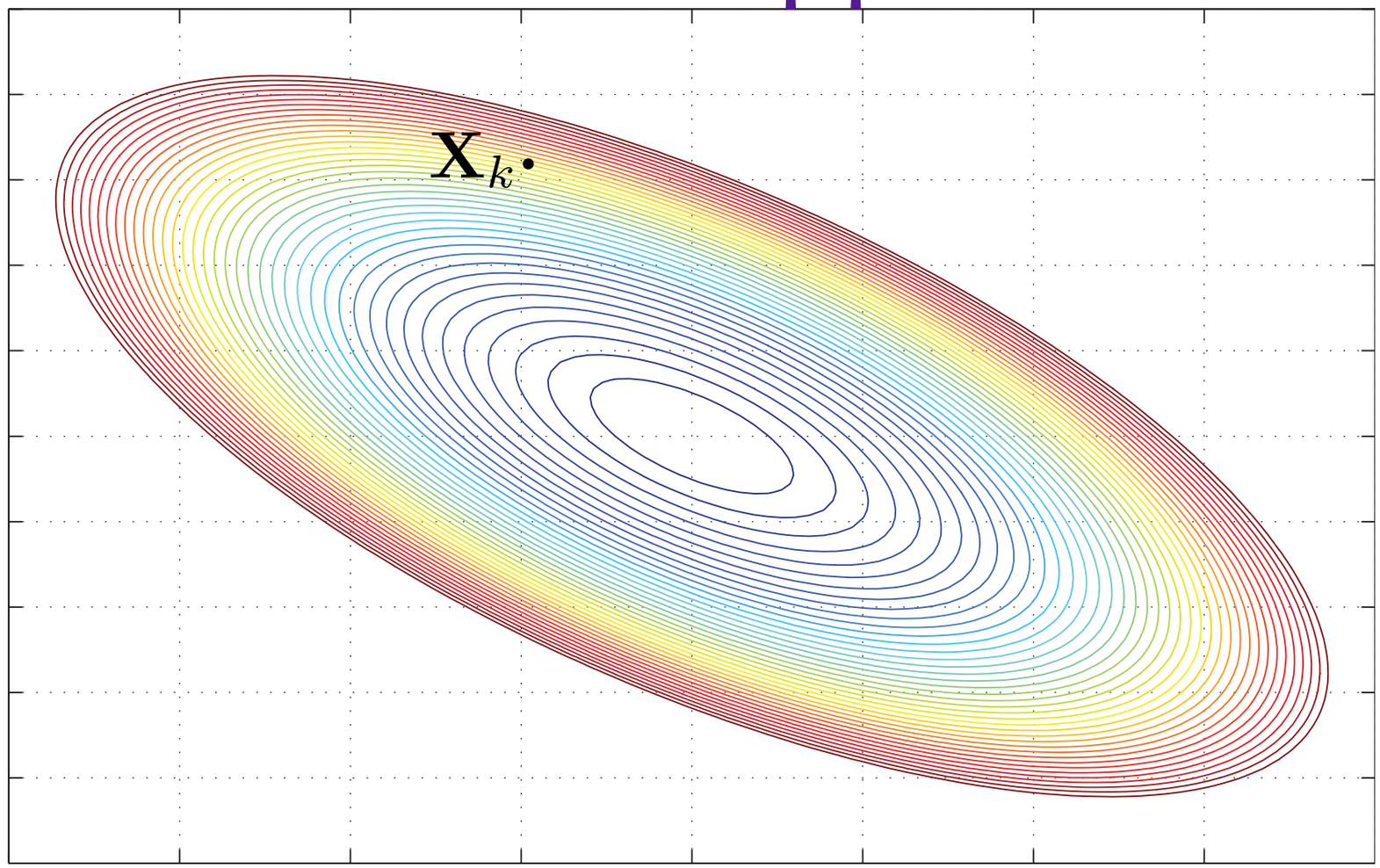
$$\mathbf{X}_{k+1} = \mathbf{X}_k + \eta_0 \mathbf{P}_k$$

(use fixed step)

  - repeat till convergence.

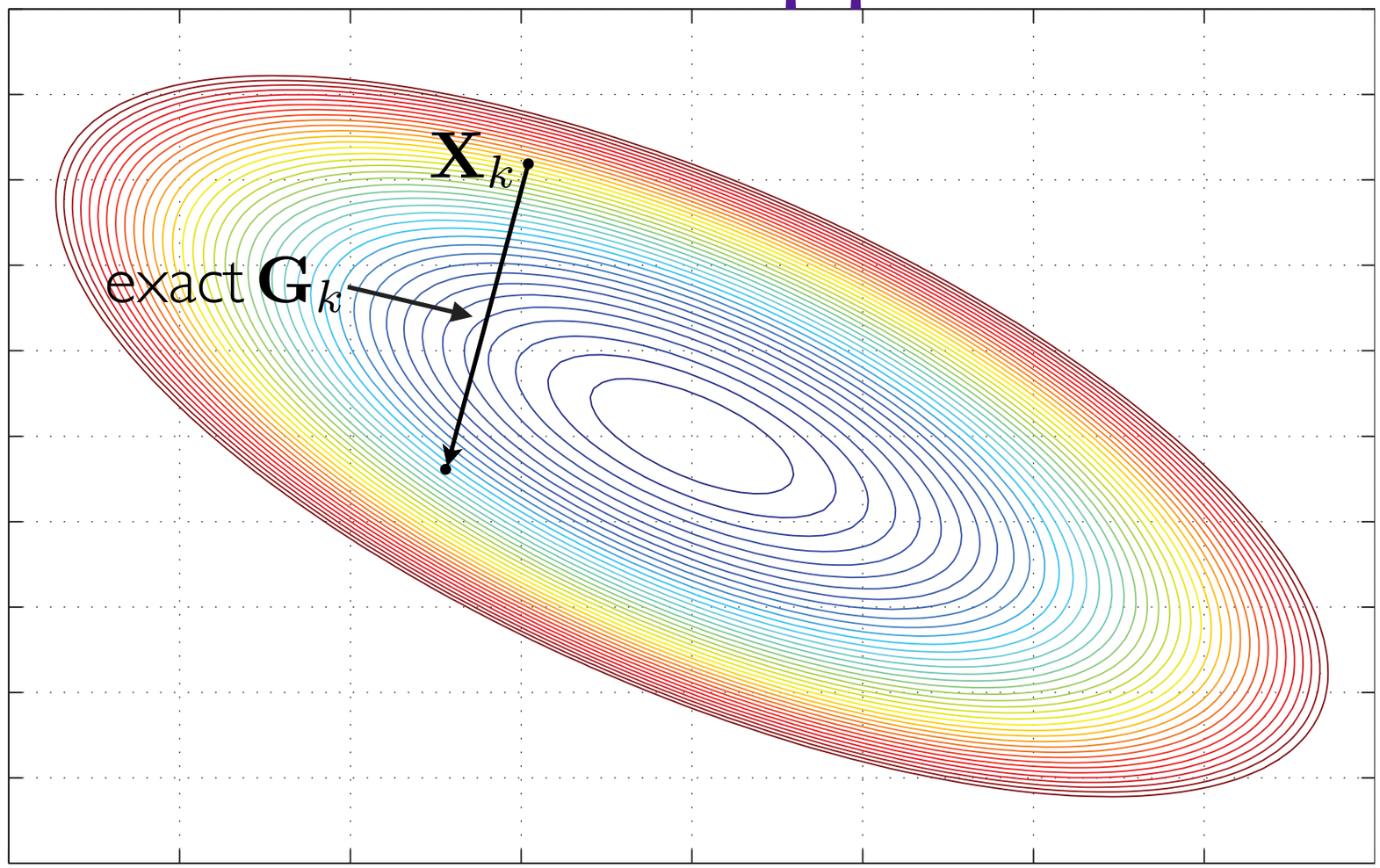


Model the effect of the approximate gradient



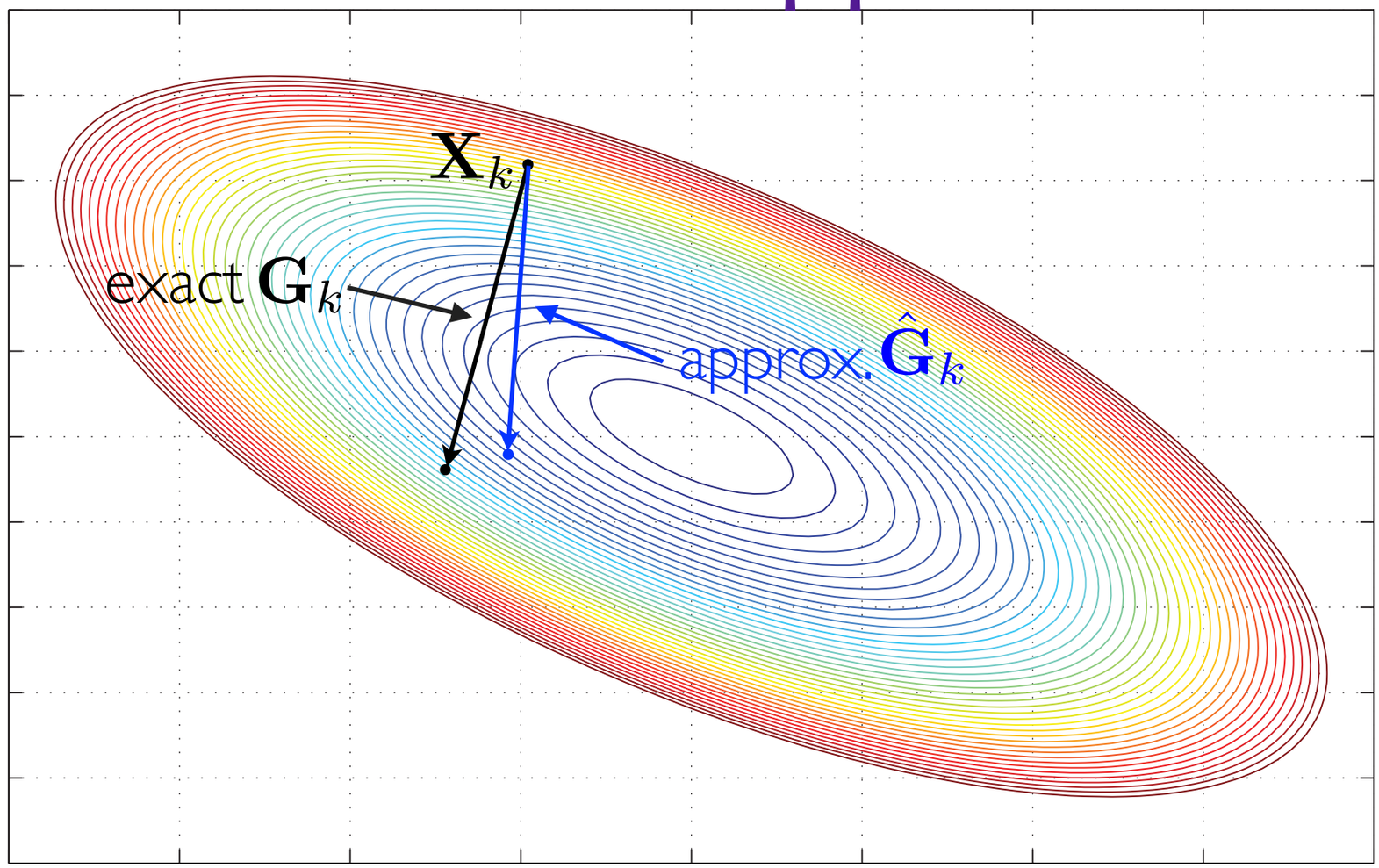
- For each iteration we incur the error $\mathbf{X}_{k+1} = \mathbf{X}_k + \boldsymbol{\epsilon}_k$.
- Approximation the error with the model $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$.
- σ is a model parameter and represents the accuracy of the approximation.

Model the effect of the approximate gradient



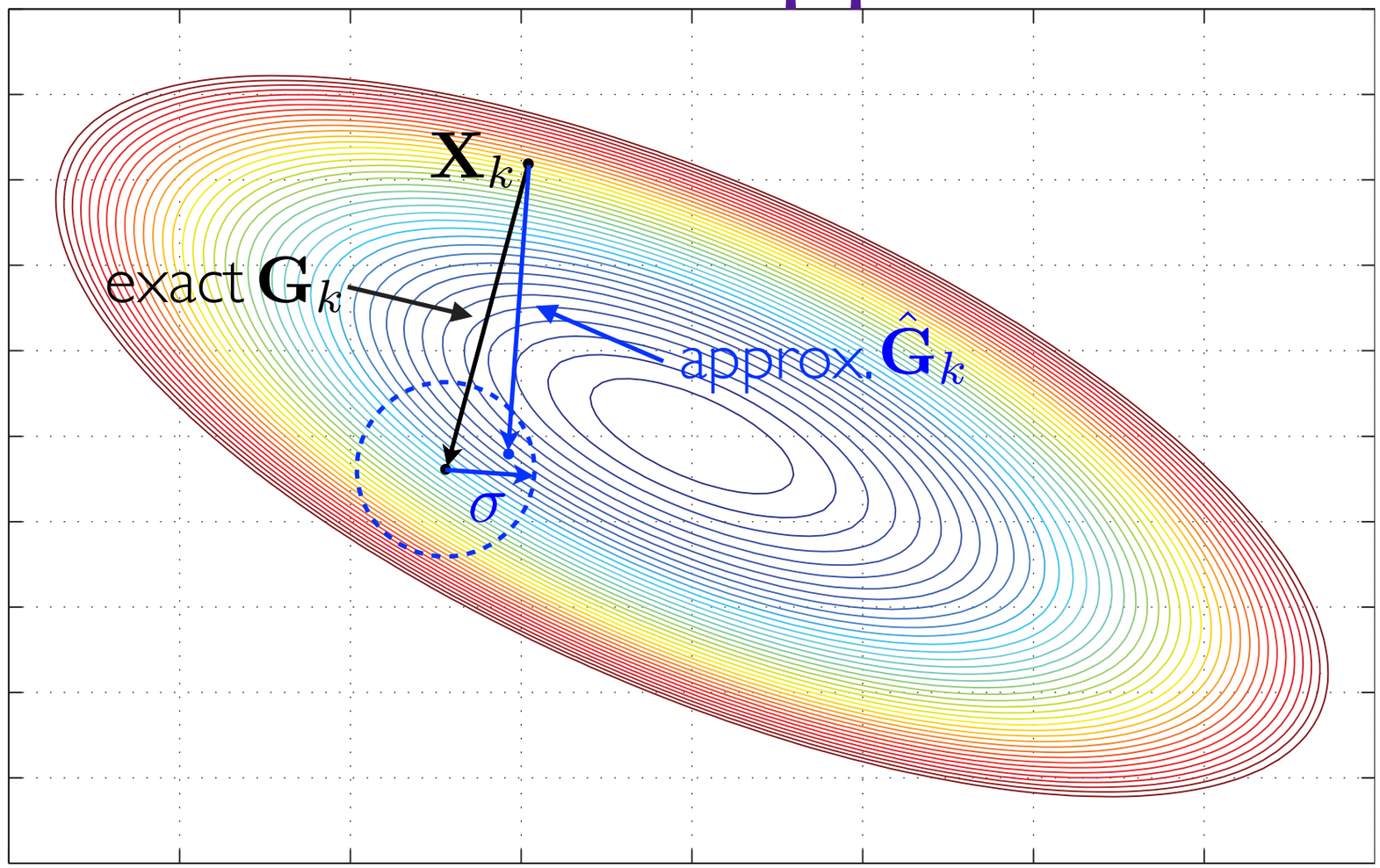
- For each iteration we incur the error $\mathbf{X}_{k+1} = \mathbf{X}_k + \boldsymbol{\epsilon}_k$.
- Approximation the error with the model $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$.
- σ is a model parameter and represents the accuracy of the approximation.

Model the effect of the approximate gradient



- For each iteration we incur the error $\mathbf{X}_{k+1} = \mathbf{X}_k + \boldsymbol{\epsilon}_k$.
- Approximation the error with the model $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$.
- σ is a model parameter and represents the accuracy of the approximation.

Model the effect of the approximate gradient



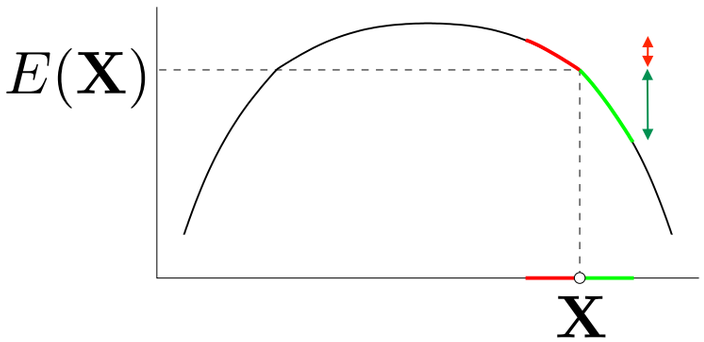
- For each iteration we incur the error $\mathbf{X}_{k+1} = \mathbf{X}_k + \boldsymbol{\epsilon}_k$.
- Approximation the error with the model $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$.
- σ is a model parameter and represents the accuracy of the approximation.

Model the effect of the approximate gradient

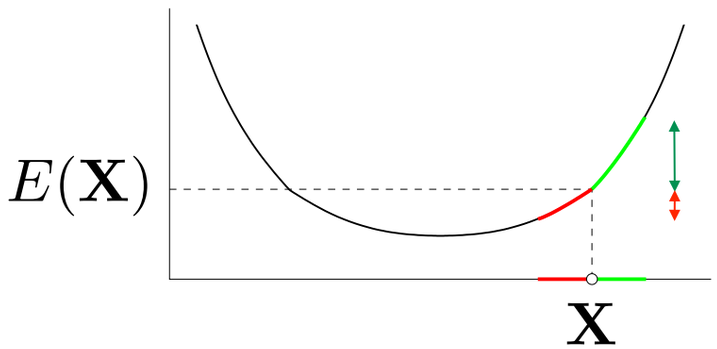
Mean of the absolute error:

$$\langle E(\mathbf{X}+\boldsymbol{\epsilon}) - E(\mathbf{X})\rangle = \frac{1}{2}\sigma^2 \operatorname{tr}\left(\nabla^2 E(\mathbf{X})\right) + \mathcal{O}(\sigma^4)$$

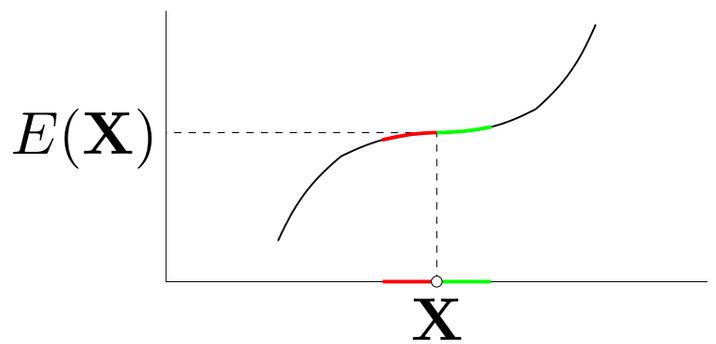
Negative curvature



Positive curvature



No curvature

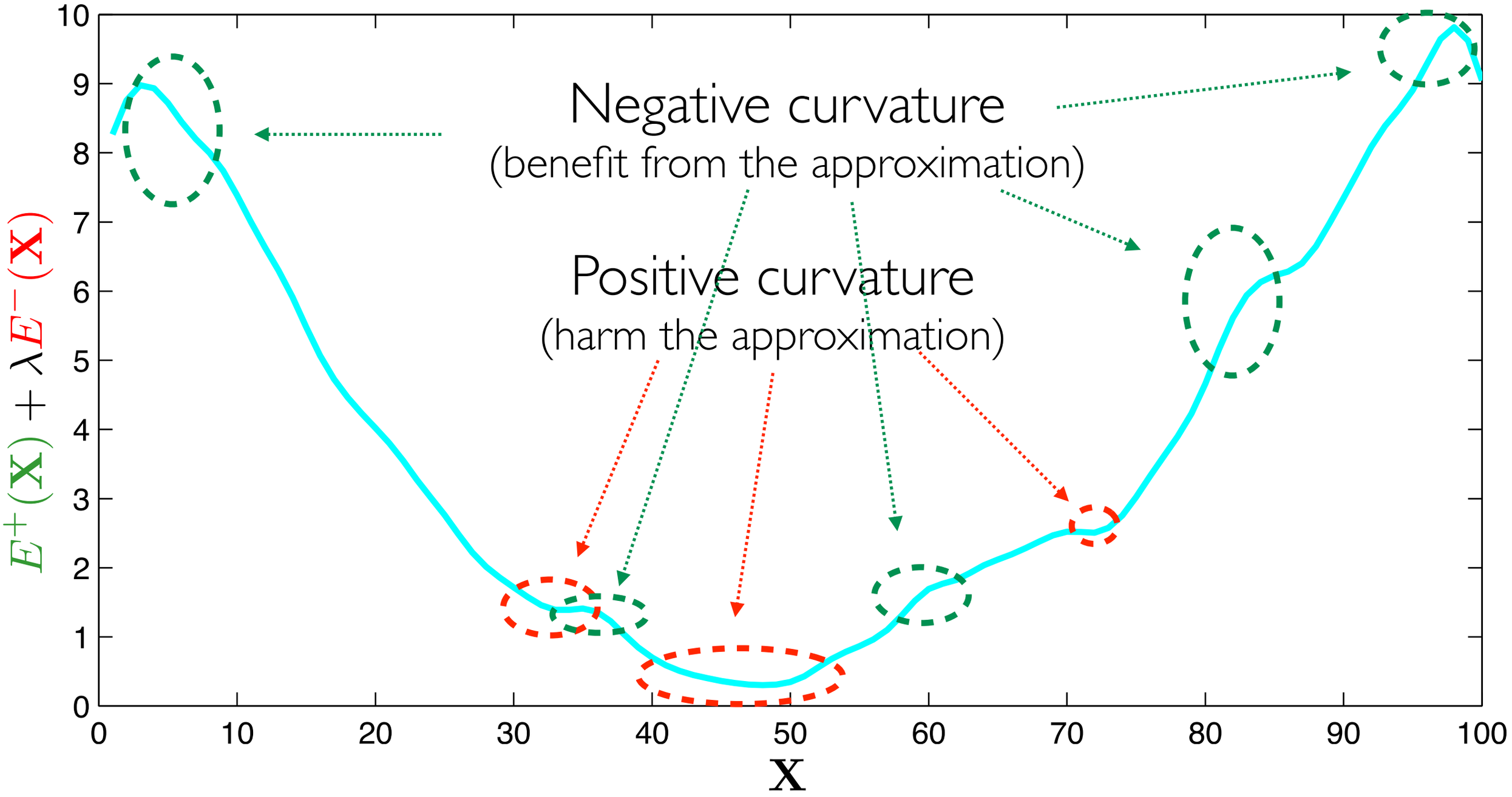


We have qualitative predictions:

1. Adding noise will be beneficial only where the mean curvature $\frac{1}{n}\operatorname{tr}\left(\nabla^2 E(\mathbf{X})\right)$ is negative
2. When the mean curvature is positive, the lower the accuracy the worse the optimization;
3. $\Delta E(\mathbf{X})$ will vary widely at the beginning of the optimization and become approximately constant and equal to $\frac{1}{2}\sigma^2 \operatorname{tr}\left(\nabla^2 E(\mathbf{X})\right)$.

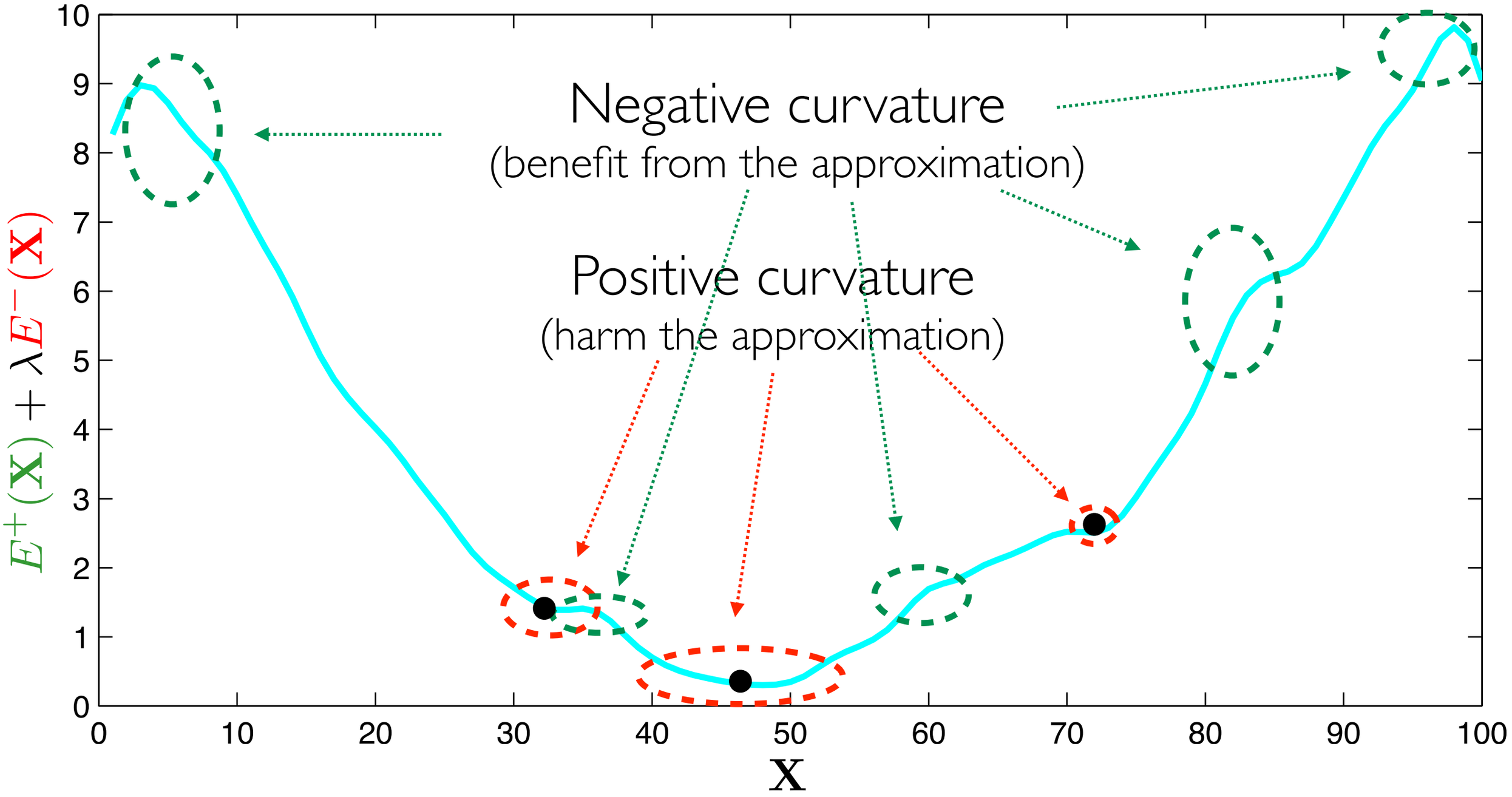
Model the effect of the approximate gradient

Under this model, we can suggest to increase the accuracy parameter as we proceed with iterations.



Model the effect of the approximate gradient

Under this model, we can suggest to increase the accuracy parameter as we proceed with iterations.



NLE: Optimization (revisited)

- Minimize objective function:

$$E(\mathbf{X}, \lambda) = E^+(\mathbf{X}) + \lambda E^-(\mathbf{X}) \quad \lambda \geq 0$$

- for a sequence of non-decreasing parameters $\mathbf{p} = p_0, \ldots, p_\infty$
  - compute the approximate gradient $\hat{\mathbf{G}}_k$ using accuracy p_l.
  - compute the direction (e.g. with grad. descent $\mathbf{P}_k = -\hat{\mathbf{G}}_k$).
  - compute new iteration $\mathbf{X}_{k+1}$ using fixed step η_0

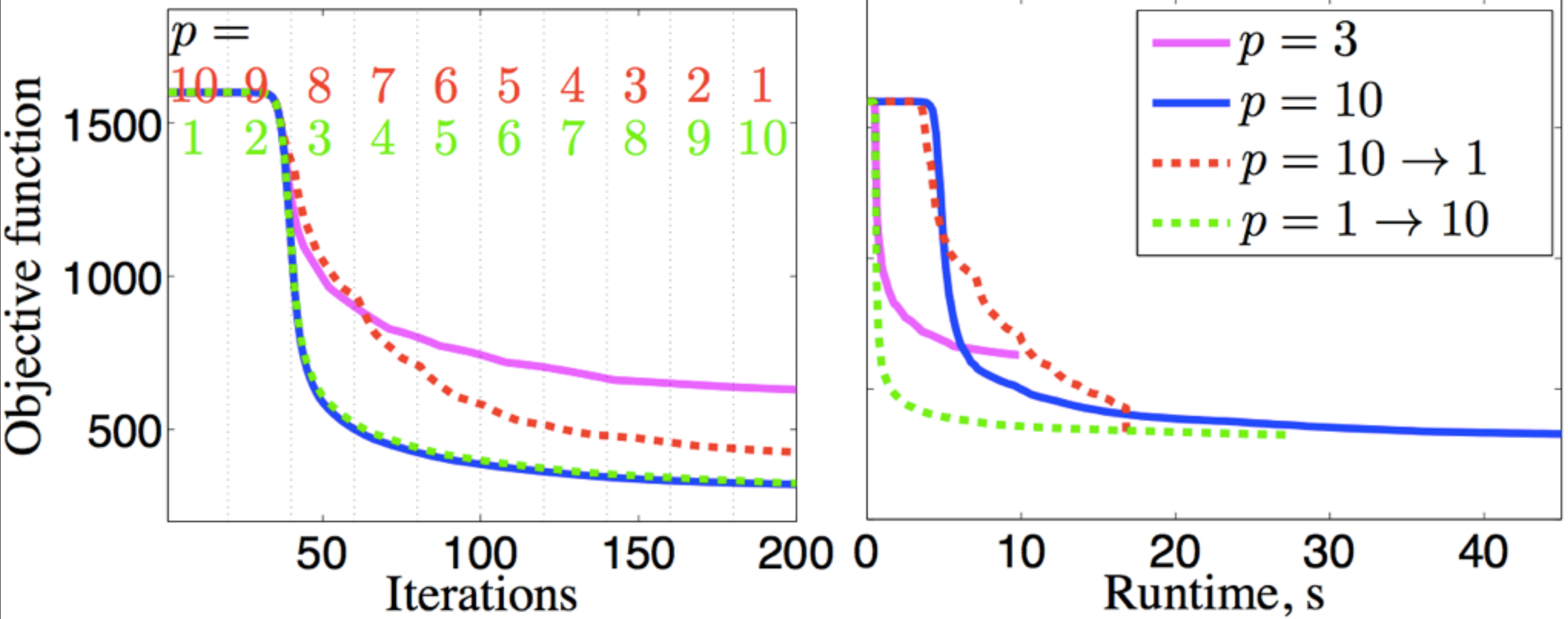
$$\mathbf{X}_{k+1} = \mathbf{X}_k + \eta_0 \mathbf{P}_k$$

- convergence is guaranteed.

Accuracy of the approximation

Compare different ways to change the accuracy of the approximation:

- fixed large,
- fixed small,
- changing from small to large,
- changing from large to small.



Fast out-of-sample mapping

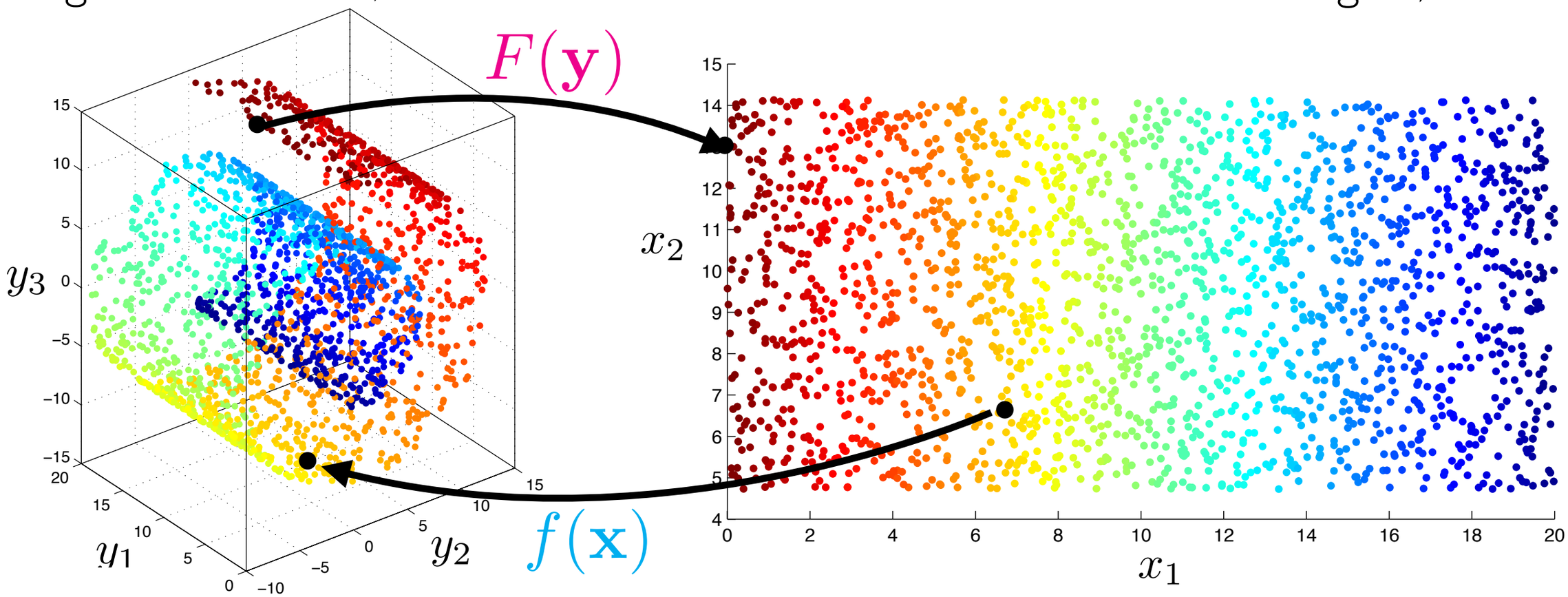
- Given a new point $\mathbf{y} \in \mathbb{R}^D$, we solve the original problem over $(\mathbf{X}\,\mathbf{x})$ and $(\mathbf{Y}\,\mathbf{y})$, subject to keeping the embedding $\mathbf{X}$ fixed:

$$E'(\mathbf{x}, \mathbf{y}, \lambda) = E^+(\mathbf{x}, \mathbf{y}) + \lambda E^-(\mathbf{x}, \mathbf{y}) \qquad \lambda \geq 0$$

- Project new high-d point $\mathbf{y}$: $F(\mathbf{y}) = \arg\min_{\mathbf{x}} E'(\mathbf{x}, \mathbf{y})$
- Reconstruct new low-d point $\mathbf{x}$: $f(\mathbf{x}) = \arg\min_{\mathbf{y}} E'(\mathbf{x}, \mathbf{y})$

Original dataset $\mathbf{Y}, D = 3$

Low-dimensional embedding $\mathbf{X}$, $d = 2$



Fast out-of-sample mapping

- If we are given M new test points we can use fast out-of-sample extension to project them all in parallel:

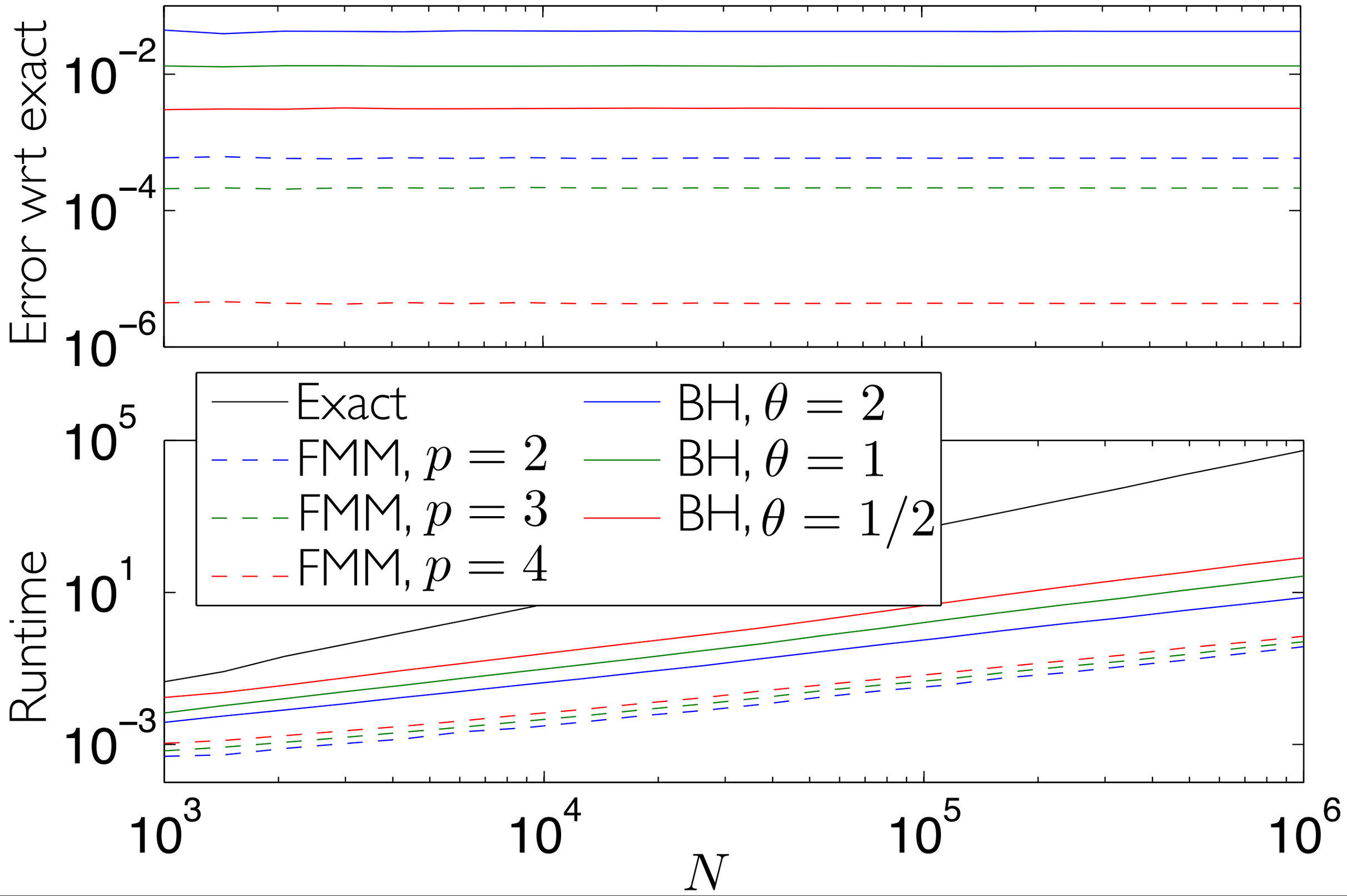
$$E'(\hat{\mathbf{X}}, \hat{\mathbf{Y}}) = 2 \sum_{m=1}^{M} \sum_{n=1}^{N} \left(w(\hat{\mathbf{y}}_m, \mathbf{y}_n) \, \|\hat{\mathbf{x}}_m - \mathbf{x}_n\|^2 + \lambda \exp\left(- \|\hat{\mathbf{x}}_m - \mathbf{x}_n\|^2 \right) \right)$$

- If computed exactly, this would take $\mathcal{O}(MN)$.
- Using FMM, we can reduce this cost to $\mathcal{O}(M + N)$.

Experiments

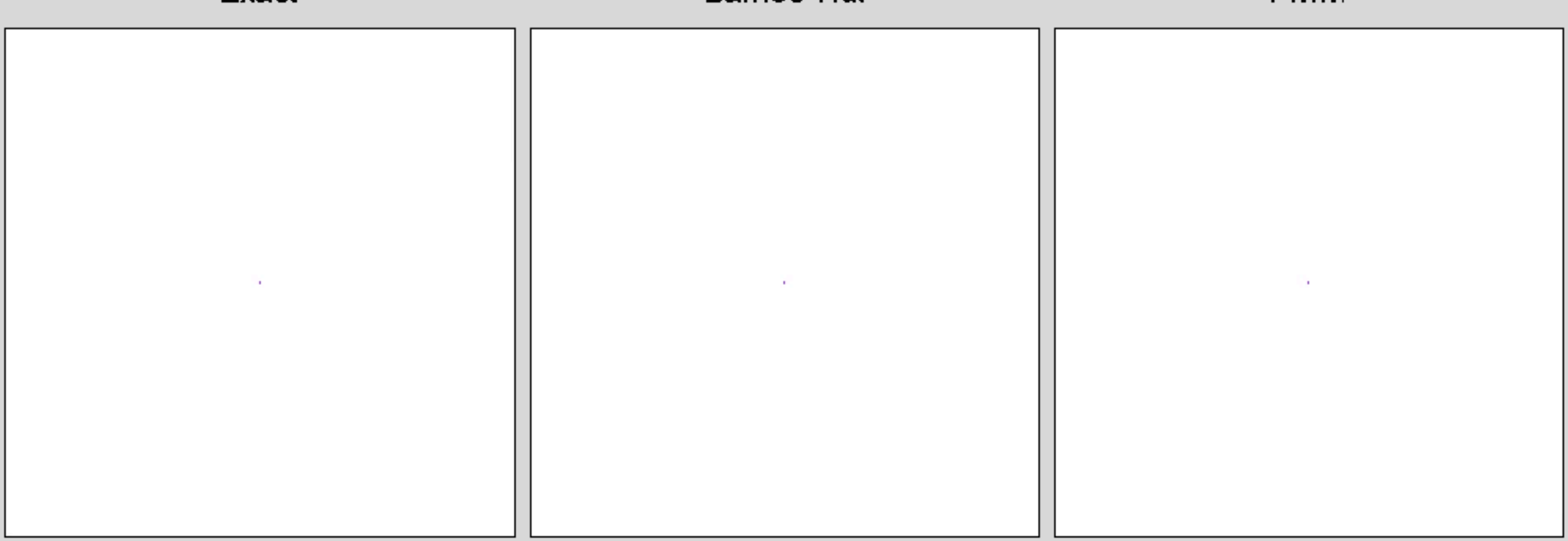
- All experiments were performed using Elastic Embedding, however the method generalizes over all NLE algorithms.
- We mostly used L-BFGS for the optimization method, but the results are general over all optimization methods.
- For the accuracy schedule, we change the accuracy logarithmically for the first 100 iterations:
  - from $\theta = 2$ to $\theta = 0.1$ for Barnes-Hut (BH) algorithm.
  - from $p = 1$ to $p = 10$ for Fast Multipole Methods (FMM).
- For FMM, we fixed additional parameters of the approximation:
  $r = 1/2$, $M_0 = 5$, $K = 4$.
- We experimented with standard 60000 *MNIST* digit dataset and *inifiniteMNIST*, where digits were generated using an elastic transformation of the original ones.

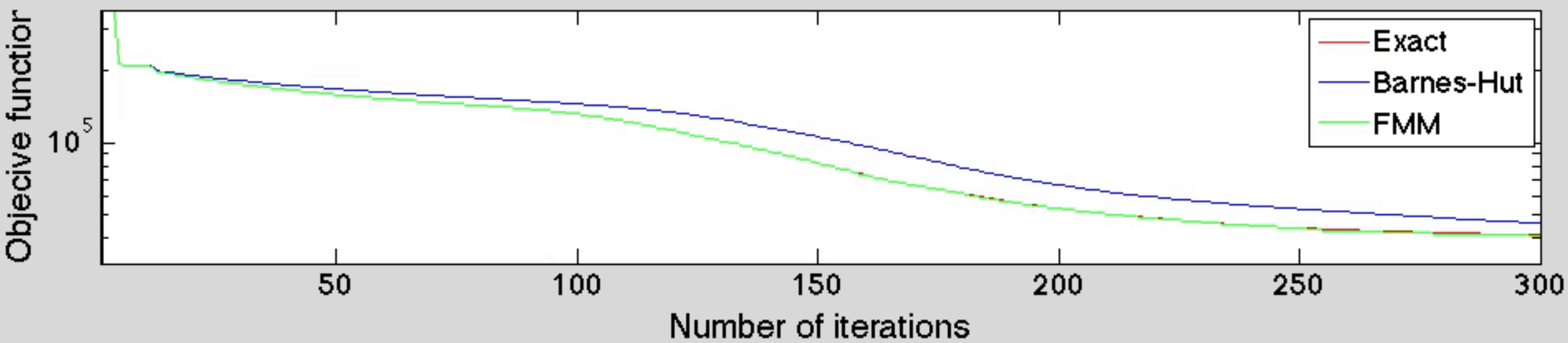
Experiments: computational cost



Experiments: 60 000 handwritten digits

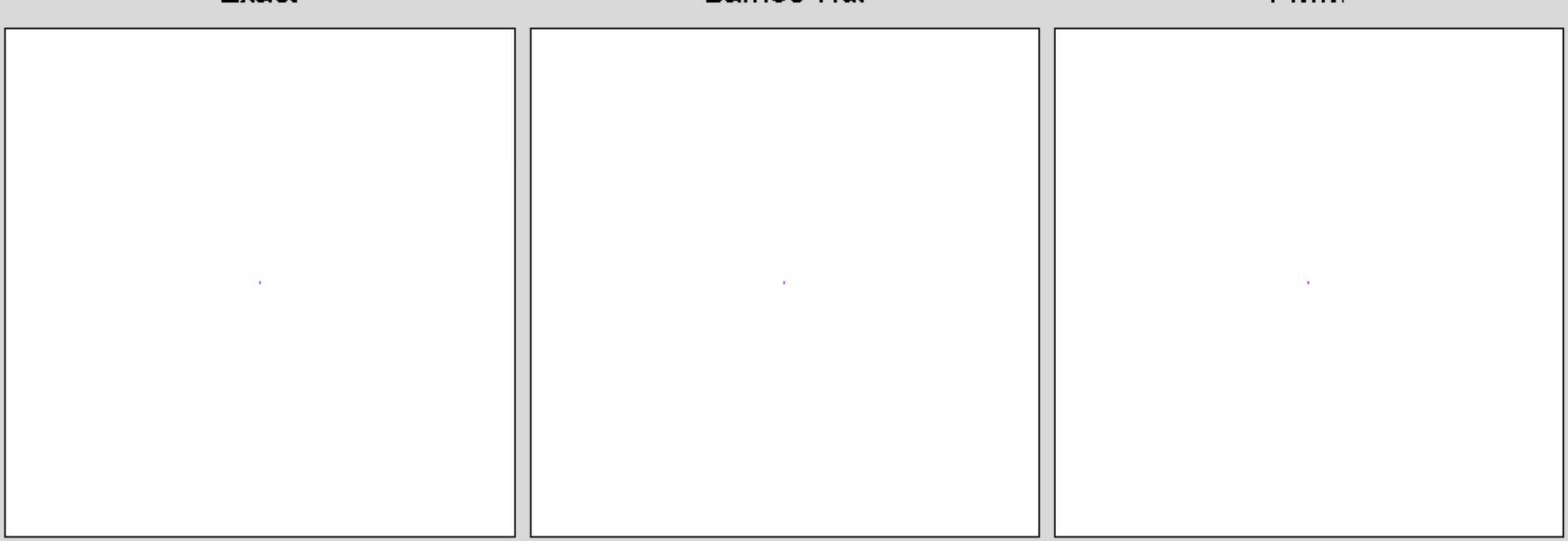
All methods show similar decrease in the objective function *per iteration.*

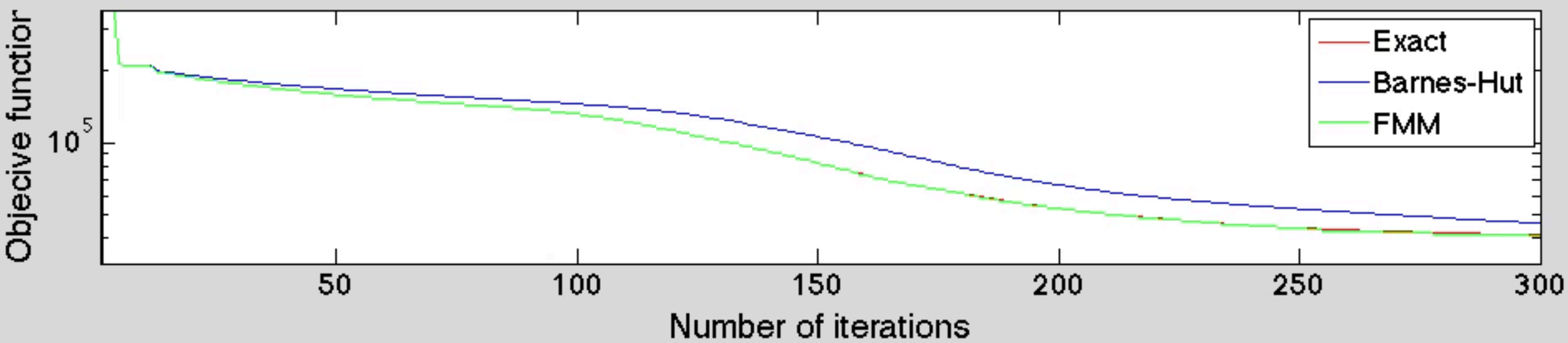




Experiments: 60 000 handwritten digits

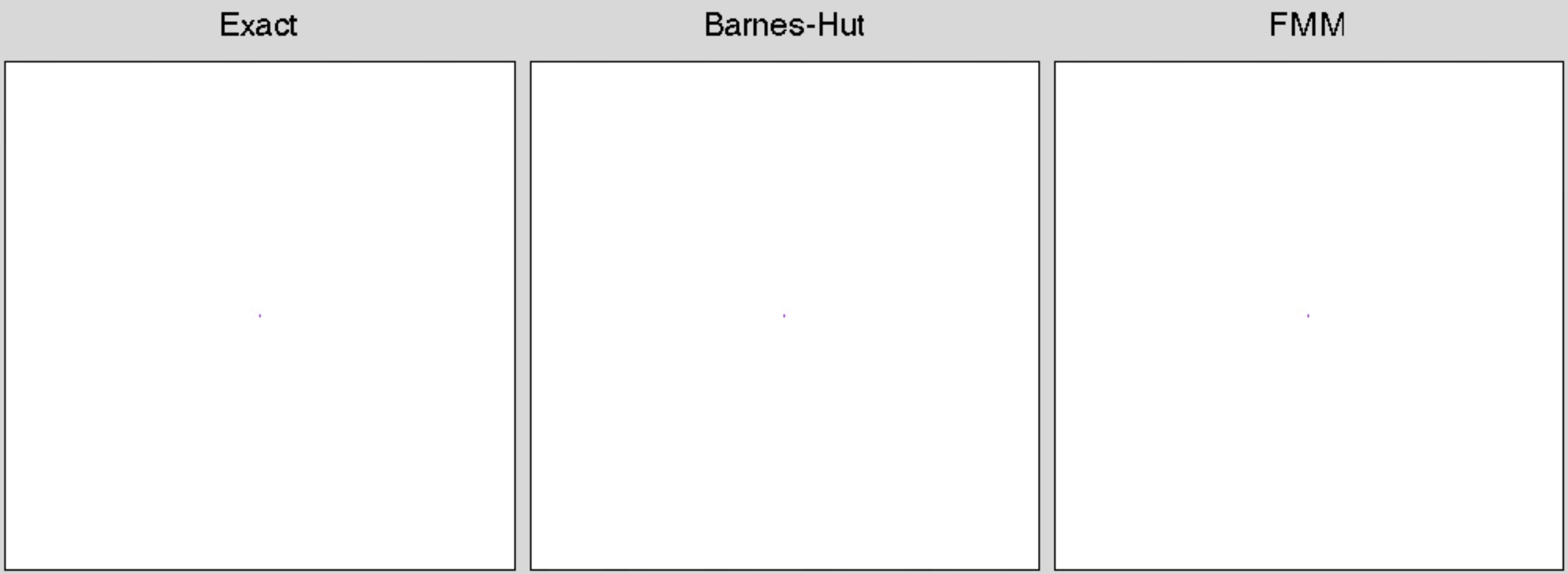
All methods show similar decrease in the objective function *per iteration.*

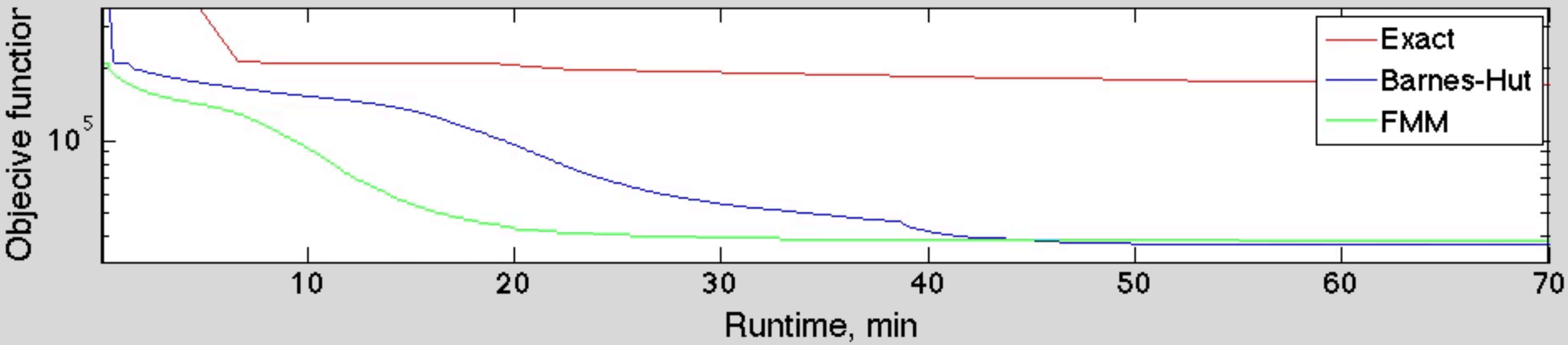




Experiments: 60 000 handwritten digits

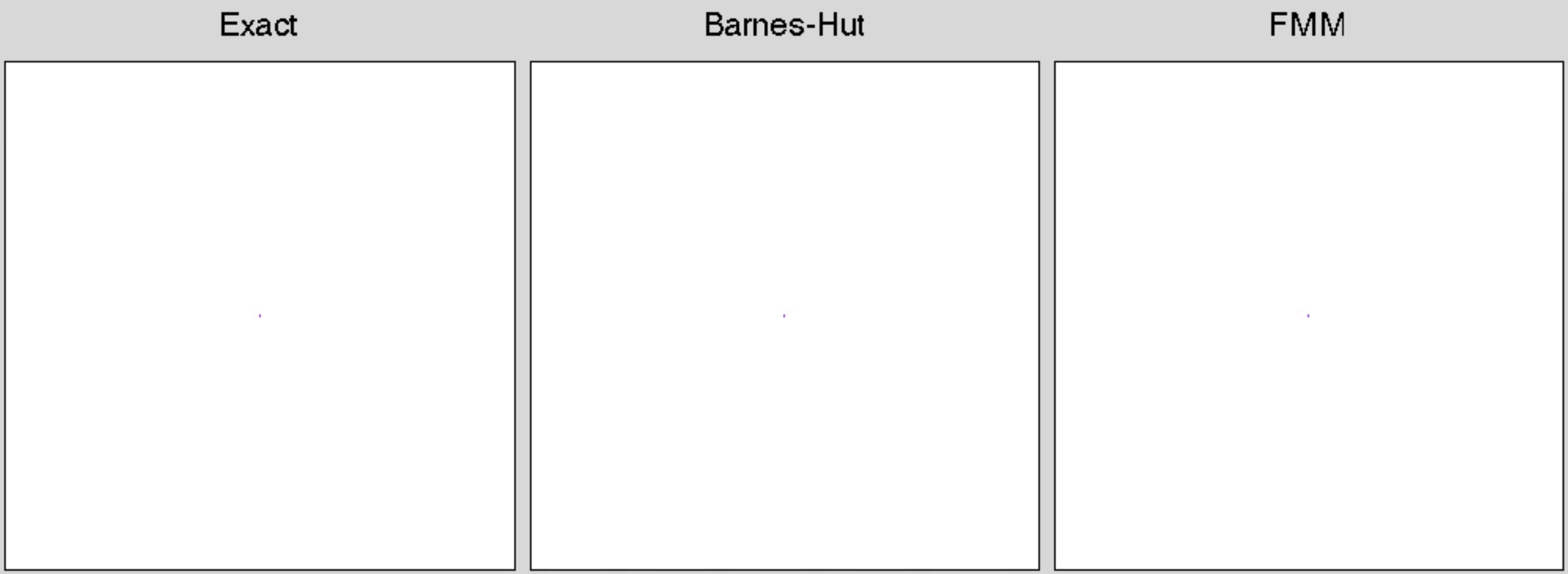
The decrease is very different if considered *per minute of runtime.*

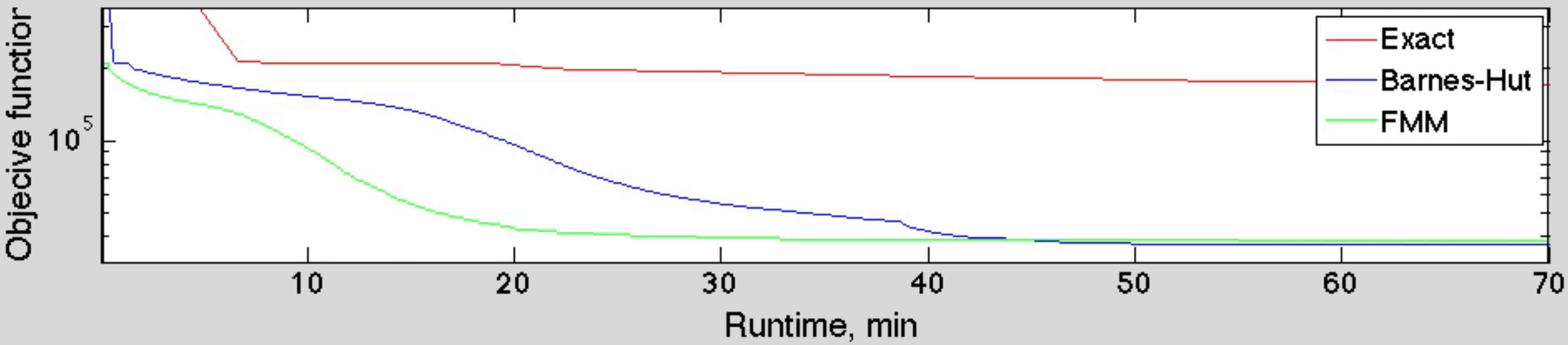




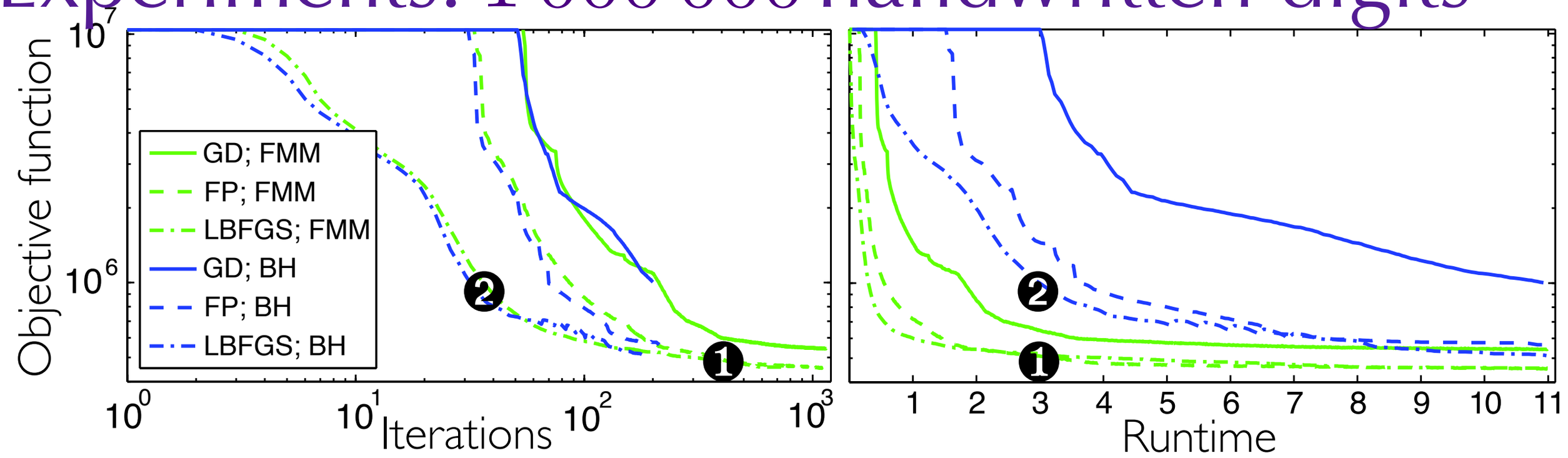
Experiments: 60 000 handwritten digits

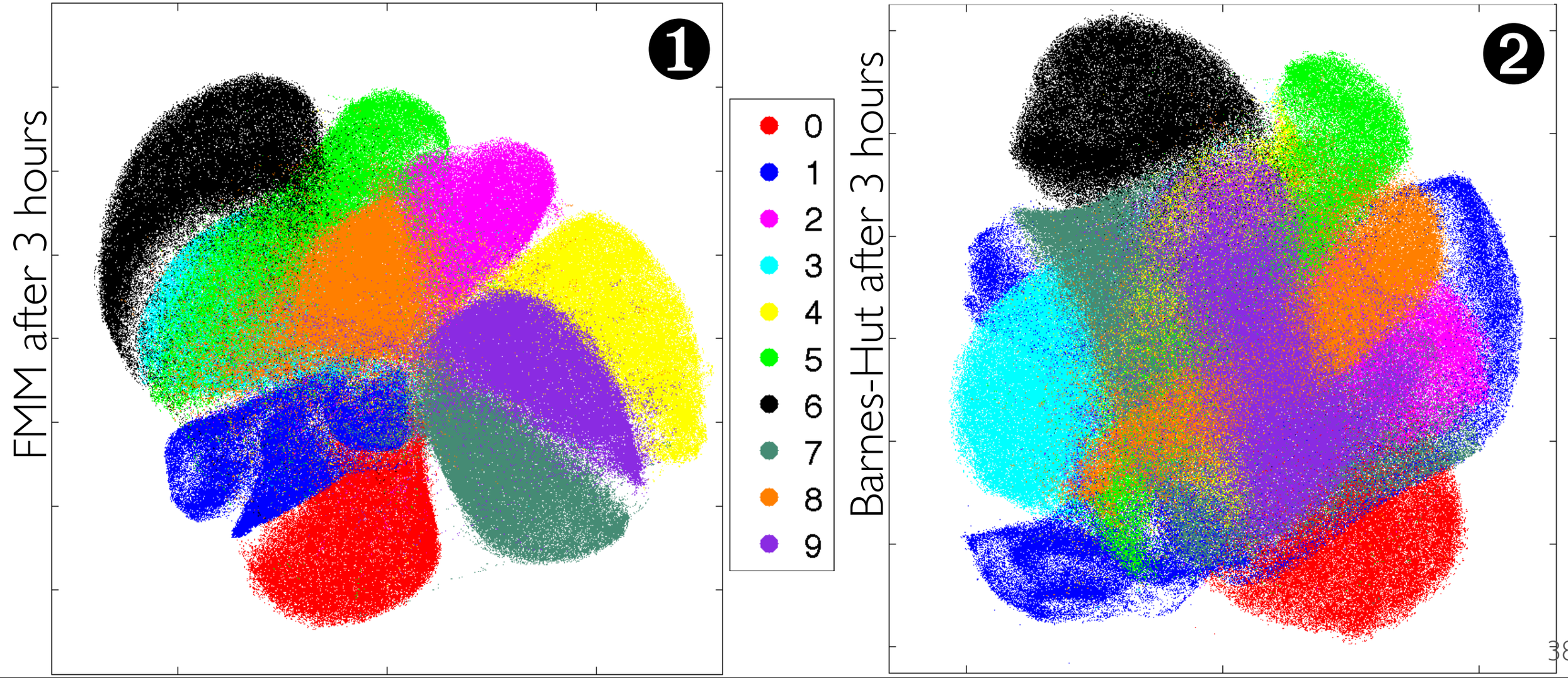
The decrease is very different if considered *per minute of runtime.*



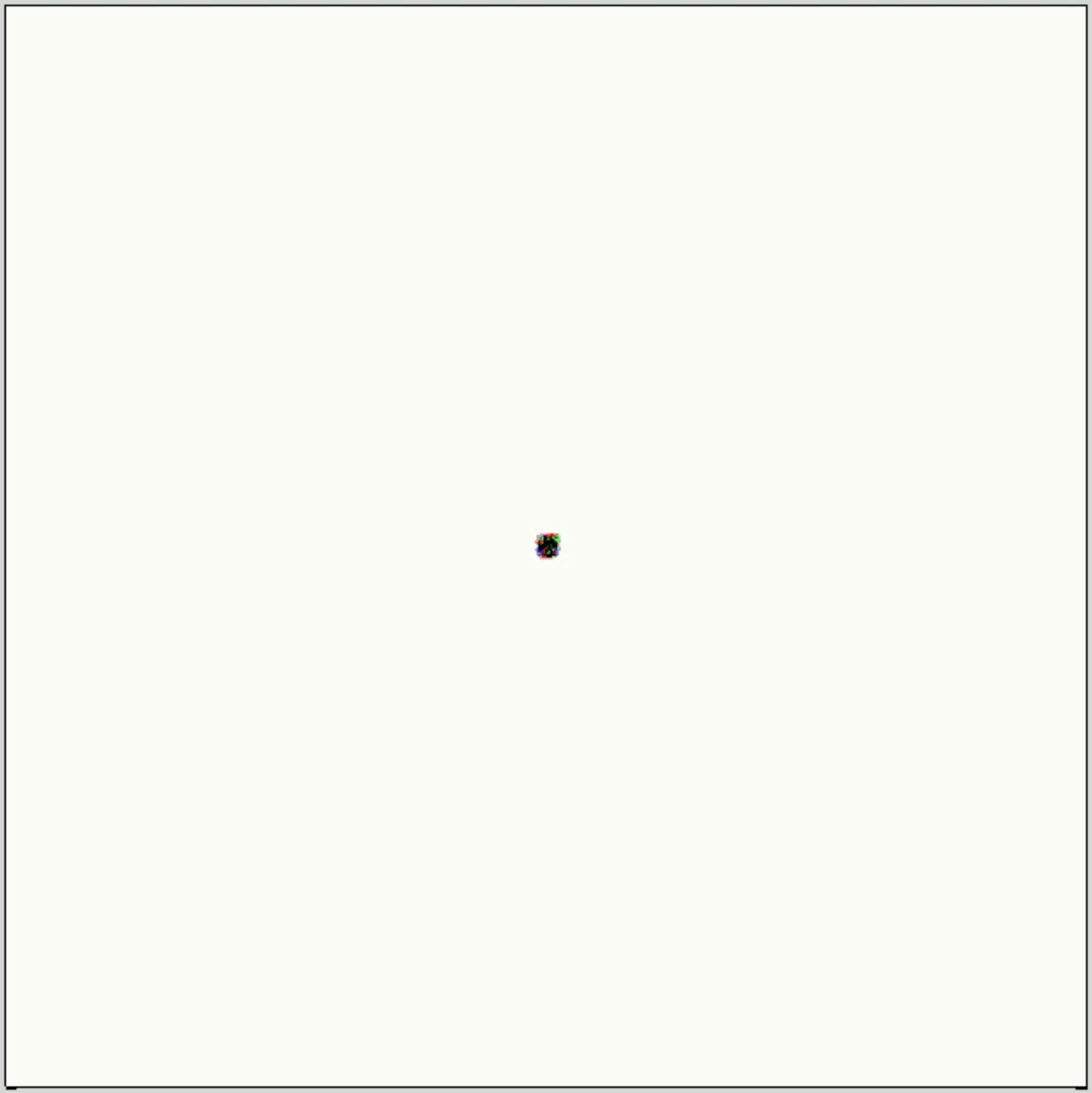


Experiments: 1 000 000 handwritten digits

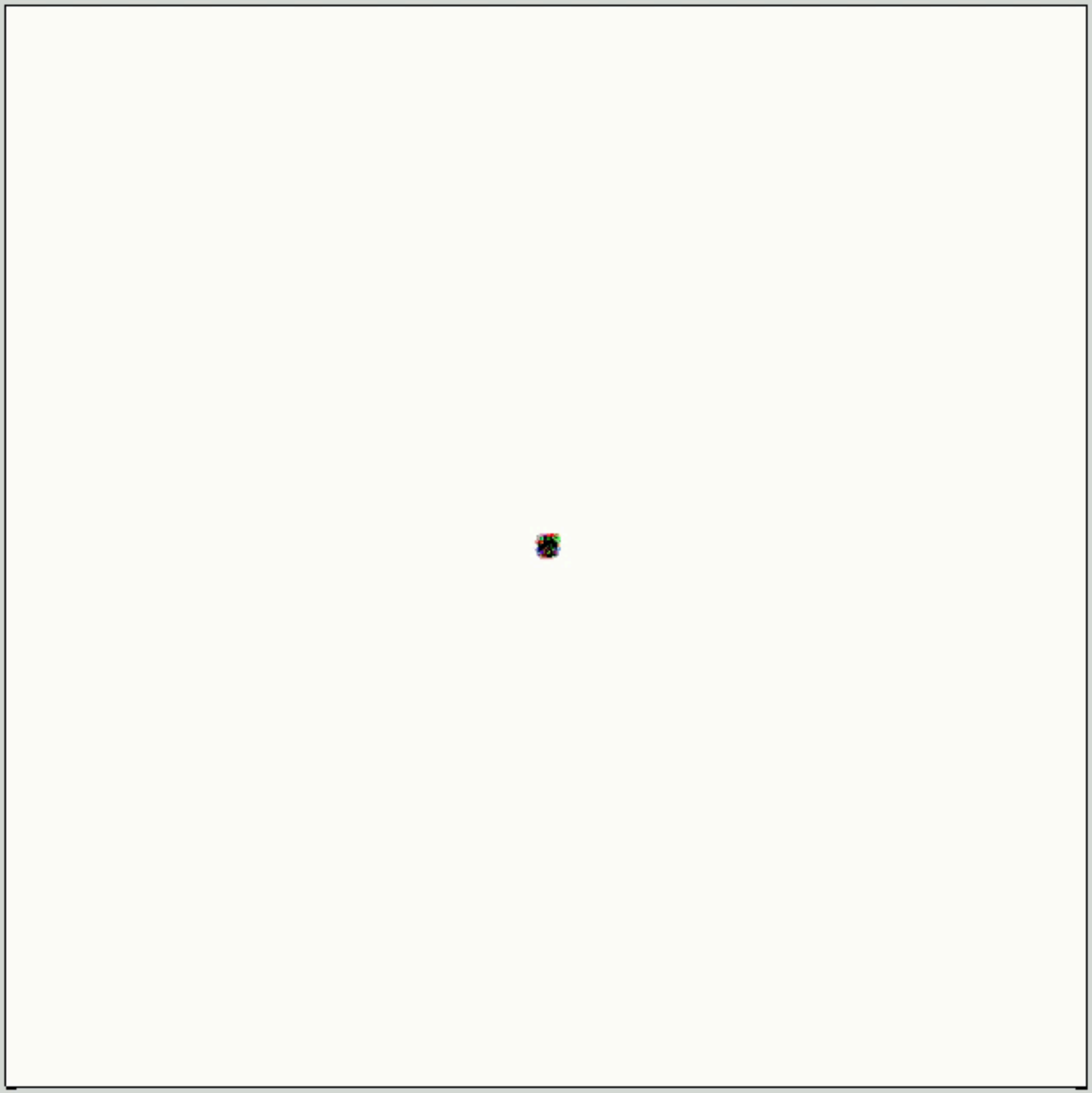




Experiments: 1 000 000 handwritten digits



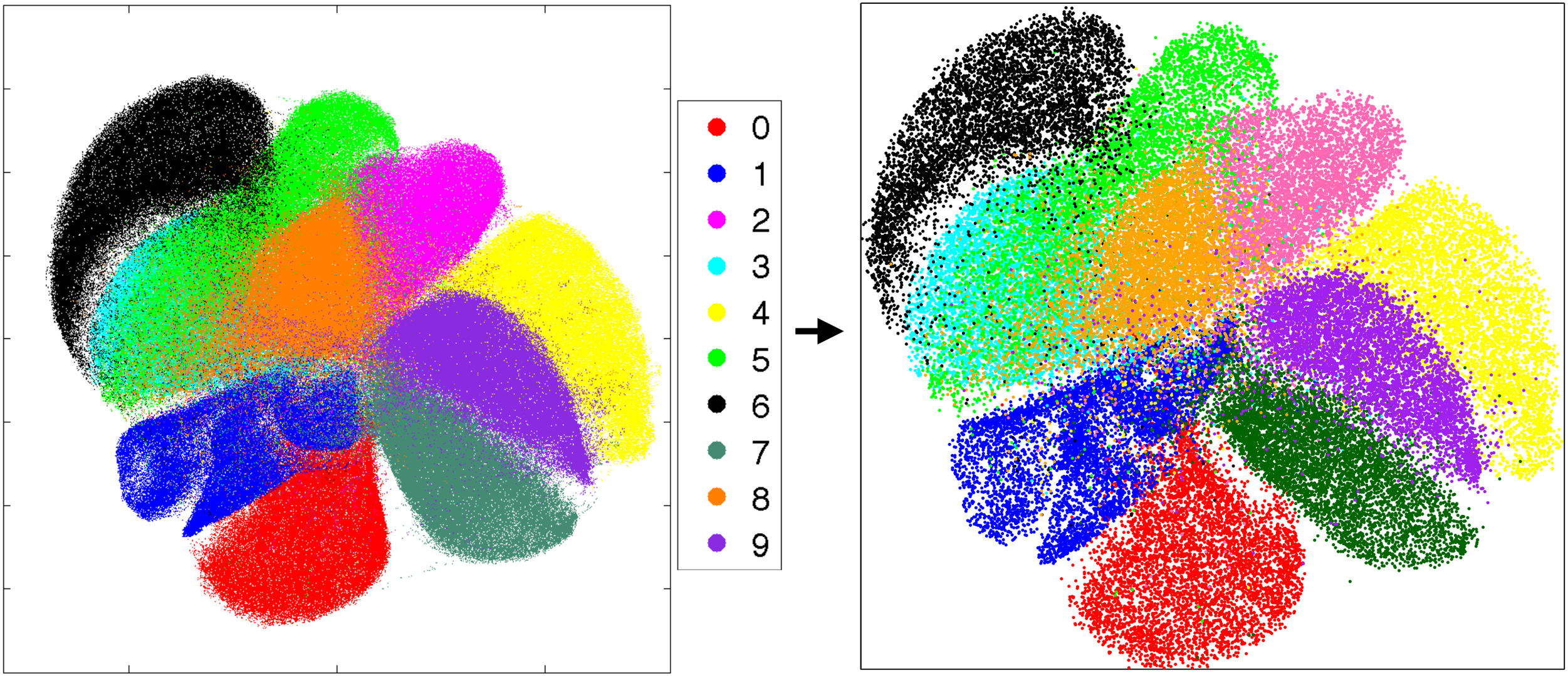
Experiments: 1 000 000 handwritten digits



Experiments: out-of-sample

Project $60\,000$ new digits into low-dimensional space using $1\,000\,000$ for training.

Runtime: 11 minutes.



Conclusions

- Nonlinear dimensionality reduction give good results, but usually expensive to train.
- We propose a new way to scale-up NLE algorithms to datasets with $> 10^6$ using fast multipole methods, that beats exact methods by $100 - 1000\times$ and Barnes-Hut method by $5 - 7\times$.
- We analyze the effect of the approximate gradient by proposing simple noise model that suggests use of the schedule for the accuracy of the approximations. Experiments confirm this benefit o this schedule.
- Future work:
  - Analyze the convergence guarantees using finite set of accuracy parameters.
  - Extend FMM to other kernels (e.g. *t*-SNE).

Conclusions

- Nonlinear dimensionality reduction give good results, but usually expensive to train.
- We propose a new way to scale-up NLE algorithms to datasets with $> 10^6$ using fast multipole methods, that beats exact methods by $100 - 1000\times$ and Barnes-Hut method by $5 - 7\times$.
- We analyze the effect of the approximate gradient by proposing simple noise model that suggests use of the schedule for the accuracy of the approximations. Experiments confirm this benefit o this schedule.
- Future work:
  - Analyze the convergence guarantees using finite set of accuracy parameters.
  - Extend FMM to other kernels (e.g. *t*-SNE).

Thank you! Questions?