EE270
Large scale matrix computation, optimization and learning

Instructor : Mert Pilanci

Stanford University

Tuesday, Jan 12 2021

Outline

- Introduction
- Administrative
- Overview of topics and applications

Administrative

Teaching staff

- Instructor: Mert Pilanci
  - Email: pilanci@stanford.edu
  - Office hours: Wednesdays 11am-12pm via Zoom (see Canvas homepage for the Zoom link)

  TA: Tolga Ergen, ergen@stanford.edu
  - TA office hours: TBA
- Public web page : `http://web.stanford.edu/class/ee270/`

Please check Canvas for up-to-date info
For all questions please use Piazza

About EE-270

This course will explore the theory and practice of randomized matrix computation and optimization for large-scale problems to address challenges in modern massive data sets.

- Our goal in this course is to help you to learn:
  - **randomized methods** for linear algebra, optimization and machine learning

About EE-270

This course will explore the theory and practice of randomized matrix computation and optimization for large-scale problems to address challenges in modern massive data sets.

- Our goal in this course is to help you to learn:
  - **randomized methods** for linear algebra, optimization and machine learning
  - **probabilistic tools** for analyzing randomized approximations

About EE-270

This course will explore the theory and practice of randomized matrix computation and optimization for large-scale problems to address challenges in modern massive data sets.

- Our goal in this course is to help you to learn:
  - **randomized methods** for linear algebra, optimization and machine learning
  - **probabilistic tools** for analyzing randomized approximations
  - how to implement **optimization algorithms** for large scale problems

About EE-270

This course will explore the theory and practice of randomized matrix computation and optimization for large-scale problems to address challenges in modern massive data sets.

- Our goal in this course is to help you to learn:
  - **randomized methods** for linear algebra, optimization and machine learning
  - **probabilistic tools** for analyzing randomized approximations
  - how to implement **optimization algorithms** for large scale problems
  - applications in **machine learning, statistics, signal processing and data mining**.

Resources

- EE 270 Lecture notes, M. Pilanci (`https://web.stanford.edu/class/ee270/`)
- Lecture notes on randomized linear algebra. M. Mahoney
- Sketching as a tool for numerical linear algebra. D. Woodruff
- Randomized algorithms in numerical linear algebra. R. Kannan and S. Vempala
- A Practical Guide to Randomized Matrix Computations with MATLAB Implementations. S. Wang
- Information-theoretic bounds on sketching. M. Pilanci

Prerequisites

- Familiarity with linear algebra (EE 103 or equivalent).
- Probability theory and statistics (EE 178 or equivalent)
- Basic programming skills

Grading policy

- Homework: 60%, submission via Gradescope.
- Project: 40% (proposal 10%, poster presentation 15%, report 15%)
- Extra credit: 5% for Piazza contributions, 5% for scribing/revising lecture notes.

Homework

- Assigned homeworks will be bi-weekly.
- The problem sets will include programming assignments to implement algorithms covered in the class.
- We will also analyze randomized algorithms using probabilistic tools.
- We support Python and Matlab.

Group Study

- Homework:
  - Working in groups is allowed, but each member must submit their own writeup.
  - Write the members of your group on your solutions (Up to four people are allowed).
- Project:
  - You will be asked to form groups of about 1-2 people and choose a topic
  - We'll suggest a list of research problems on the course website
  - Proposal (1 page)
  - Poster presentation (last week of classes via Gather.town)
  - Project report (6 pages max)

Topics

- randomized linear algebra
  - approximate matrix and tensor multiplication
  - approximate verification
  - sampling and projection methods
  - bootstrap
- randomized linear system solvers and regression
  - Johnson Lindenstrauss embeddings
  - leverage scores
  - iterative sketching and preconditioning
  - high dimensional problems
  - robust regression
- matrix decompositions
  - subspace embeddings
  - randomized QR decomposition
  - randomized Singular Value Decomposition
  - column subset selection

Topics

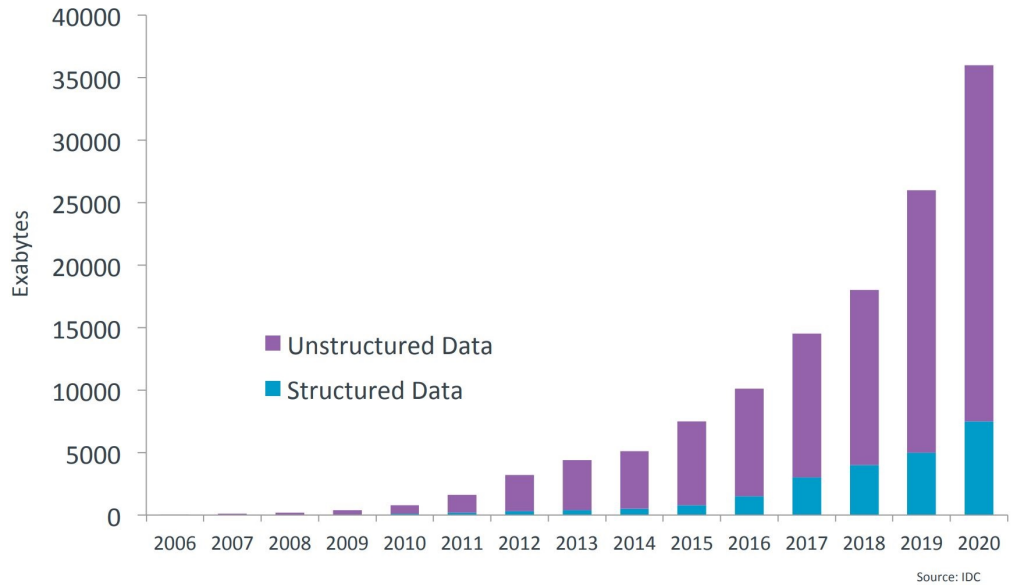
- large-scale optimization
  - empirical risk minimization
  - gradient descent and acceleration
  - stochastic gradient methods and variants
  - second order methods and Hessian approximations
- kernel methods
  - Reproducing kernel Hilbert spaces
  - Nystrom approximations
  - Randomized kernel approximations
- neural networks
  - large scale non-convex optimization
  - kernel approximations
  - convex optimization for neural networks

For details see Canvas!

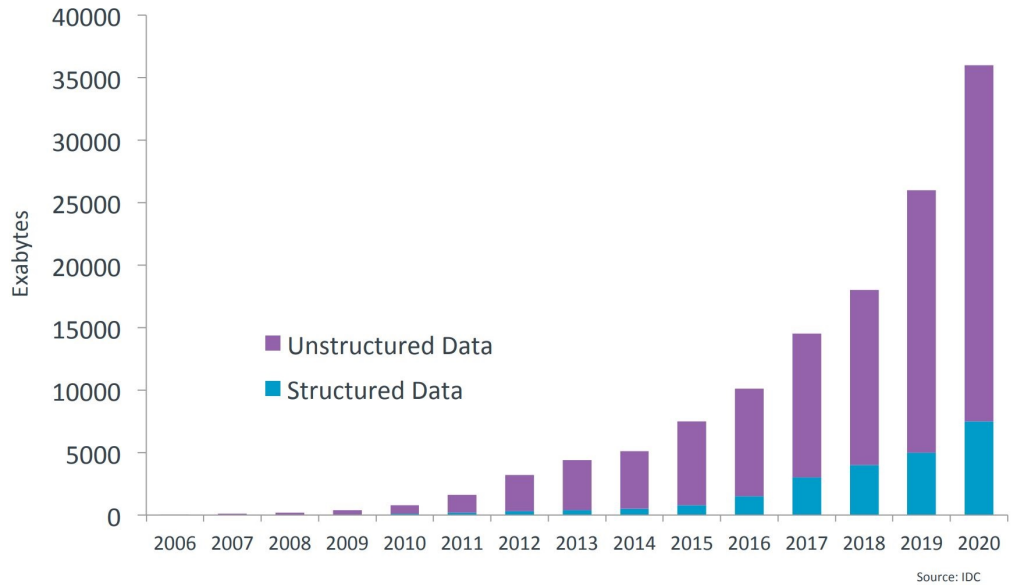
Any questions?

Overview of topics and applications

Scale of data

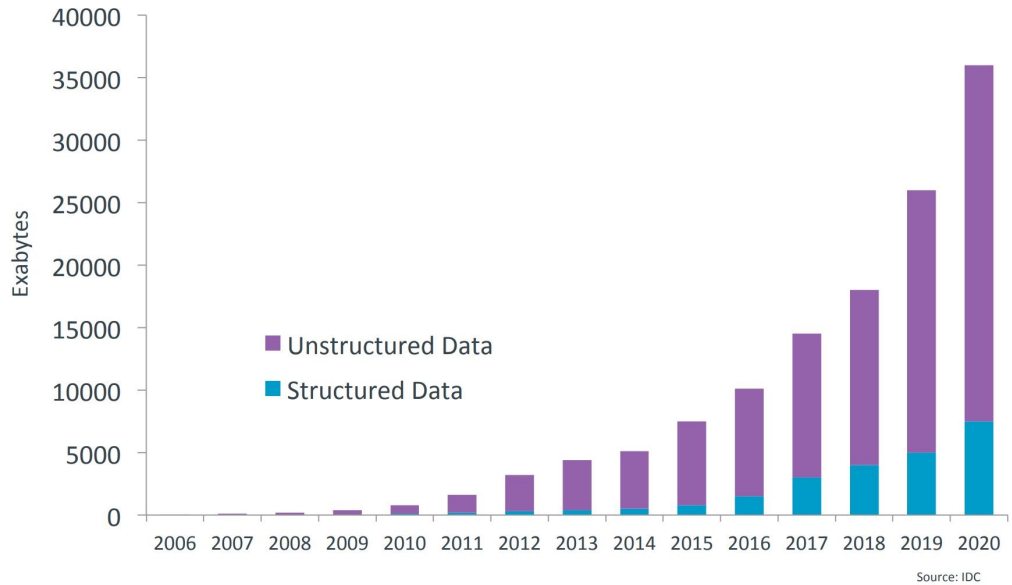


Scale of data



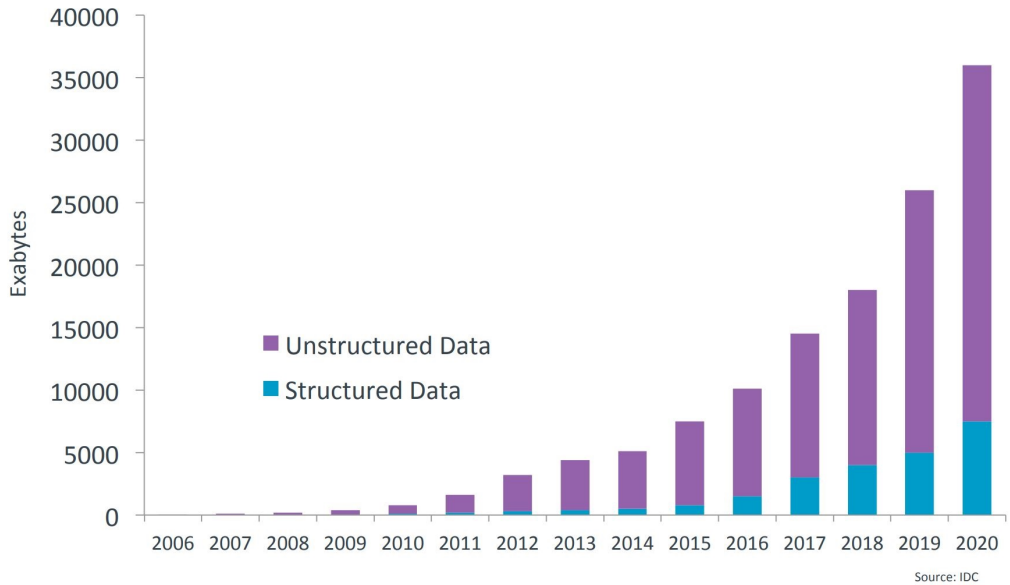
- Every day, we create 2.5 billion gigabytes of data

Scale of data

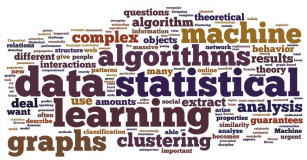


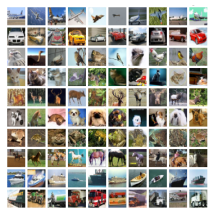
- Every day, we create 2.5 billion gigabytes of data
- Data stored grows 4x faster than world economy (Mayer-Schonberger)

Scale of data

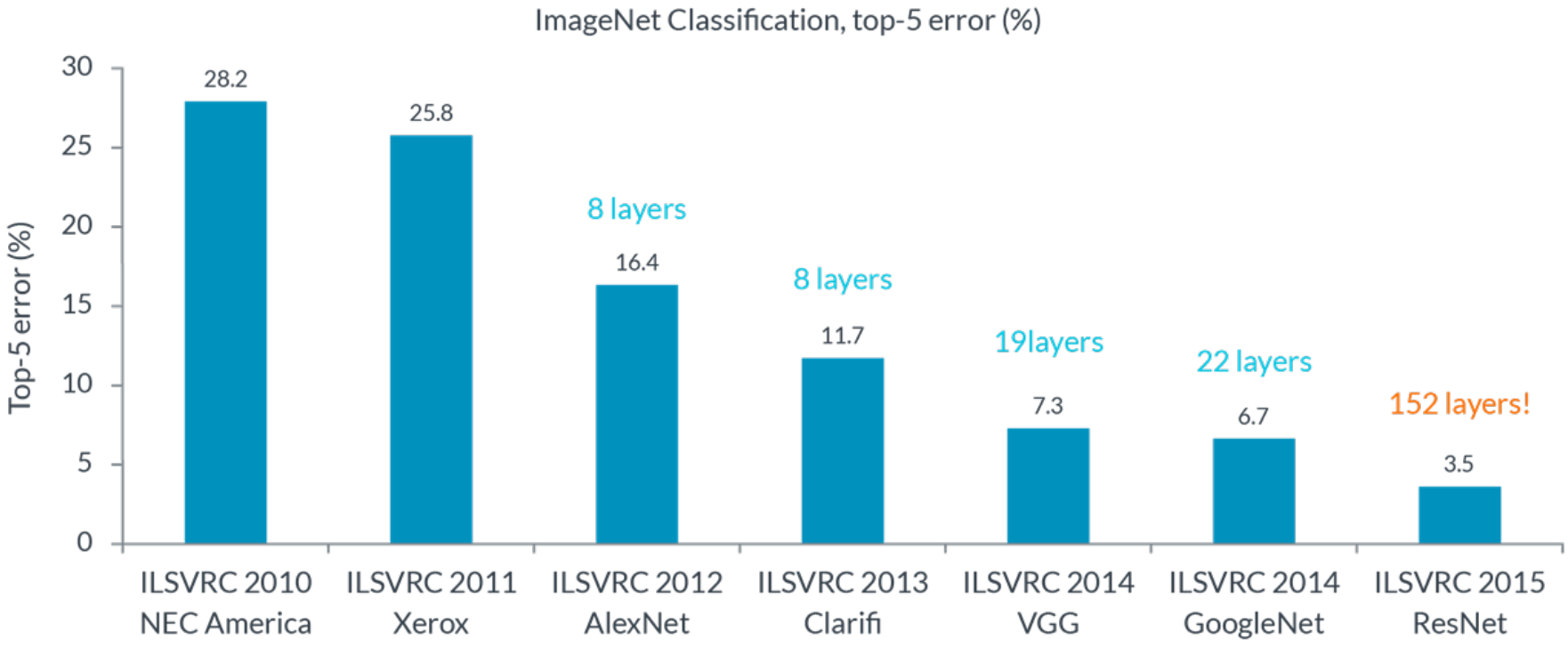


- Every day, we create 2.5 billion gigabytes of data
- Data stored grows 4x faster than world economy (Mayer-Schonberger)



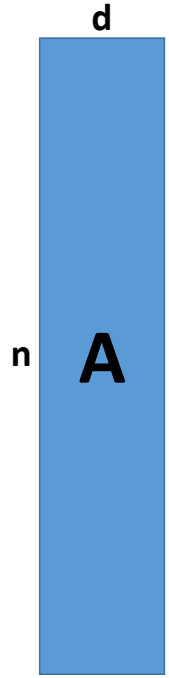


Deep learning revolution



Big data matrices

- $n \times d$ datamatrix



- **Small:** we can look at the data and find solutions easily
- **Medium:** Fits into RAM and one can run computations in reasonable time
- **Large:** Doesn't easily fit into RAM. One can't relatively easily run computations.

Typical data matrices

- Rectangular data (object-feature data): n objects, each of which are described by d features, e.g., document-term data, people-SNPs data.
- Correlation matrices
- Kernels and similarity matrices
- Laplacians or Adjacency matrices of graphs.
- Discretizations of dynamical systems, ODEs and PDEs
- Constraint matrices
- ...

Typical data matrices

- Rectangular data:

  essentially a two-dimensional matrix with rows indicating records (cases) and columns indicating features (variables)

- Example: Airline dataset

depart	**arrive**	**origin**	**dest**	**dist**	**weather delay**	**cancelled**
00:00:01	13:35:01	RNO	LAS	345	0	1
07:20:01	08:40:01	SFO	SAN	447	40	0
07:25:01	10:15:01	OAK	PHX	646	0	0
07:30:01	08:30:01	OAK	BUR	325	0	0
⋮						

in machine learning, statistics and signal processing

- More data points typically increase the accuracy of models
  $\rightarrow$ large scale matrix computation and optimization problems

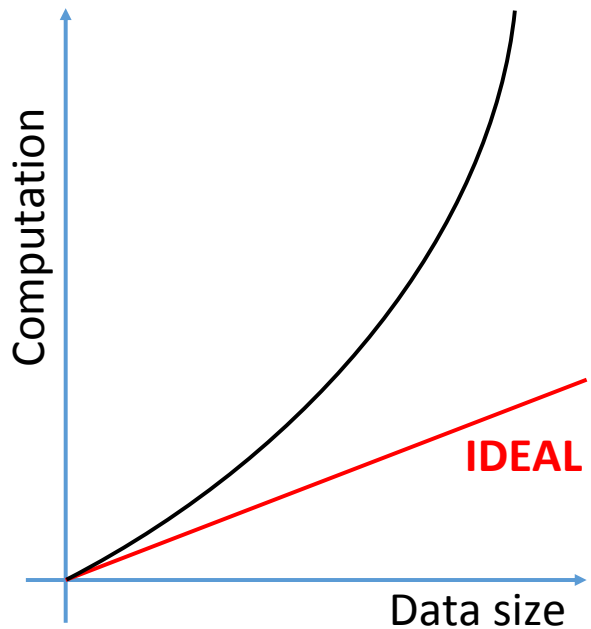
  e.g. matrix multiplication, matrix factorization, singular value decomposition, convex optimization, non-convex optimization...

in machine learning, statistics and signal processing

- More data points typically increase the accuracy of models
  $\rightarrow$ large scale matrix computation and optimization problems

  e.g. matrix multiplication, matrix factorization, singular value decomposition, convex optimization, non-convex optimization...

Can we reduce the data volume with minimal loss of information ?



Matrix Computations

- Data matrix $A \in R^{n \times d}$ where n, d are extremely large

Matrix Computations

- Data matrix $A \in R^{n \times d}$ where n, d are extremely large

Examples:

- Airline dataset (120GB) $n = 120 \times 10^6, \ d = 28$
  Flight arrival and departure details from 1987 to 2008

Matrix Computations

- Data matrix $A \in R^{n \times d}$ where n, d are extremely large

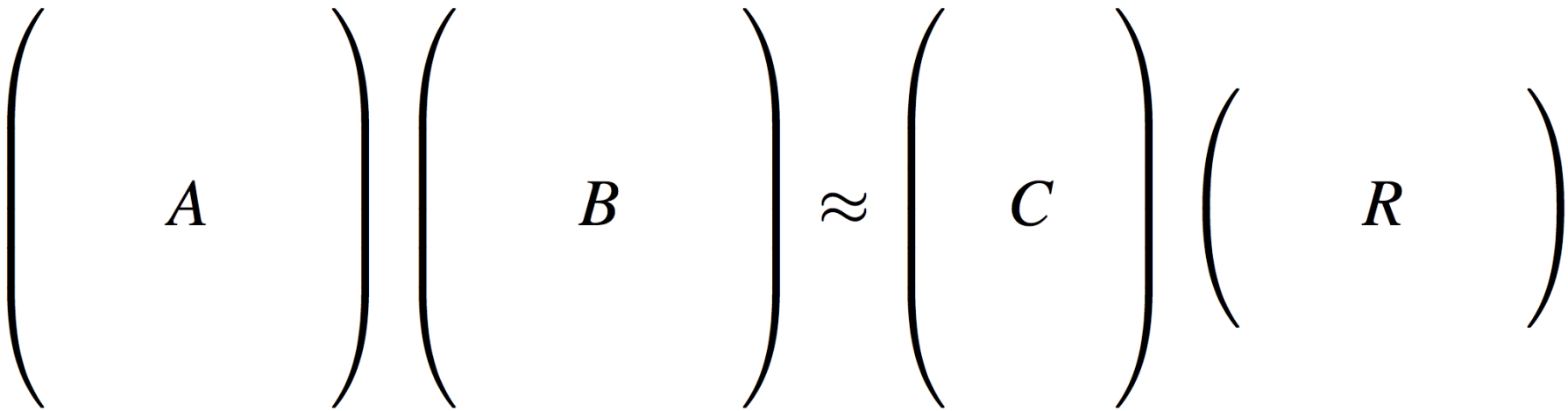
Examples:

- Airline dataset (120GB) $n = 120 \times 10^6,\ d = 28$
  Flight arrival and departure details from 1987 to 2008
- Imagenet dataset (1.31TB) $n = 14 \times 10^6$, $d = 2 \times 10^5$
  14 Million images for visual recognition

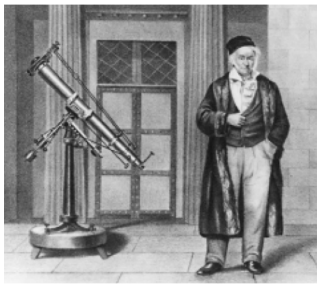
[US Department of Transportation]
[Deng et al. 2009]

Approximate Matrix Multiplication

- How to approximate the matrix product AB fast ?



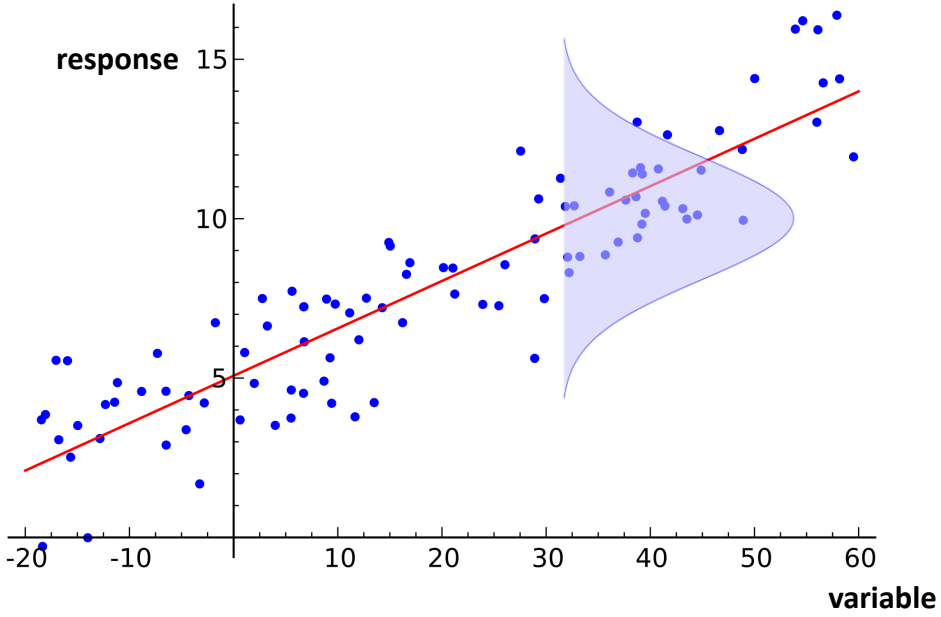
Least Squares Problems



Least squares

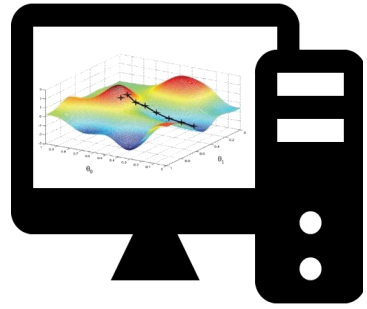
$$\min_x \|Ax - y\|^2$$

[Gauss, 1795]



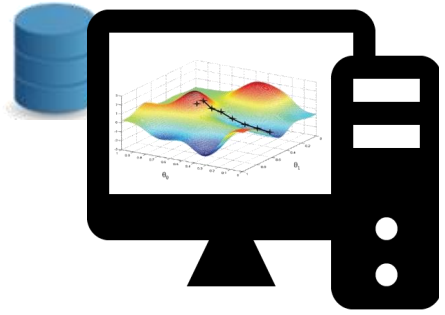
DATA

OPTIMIZER



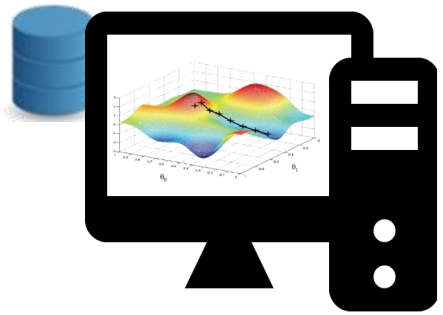
DATA

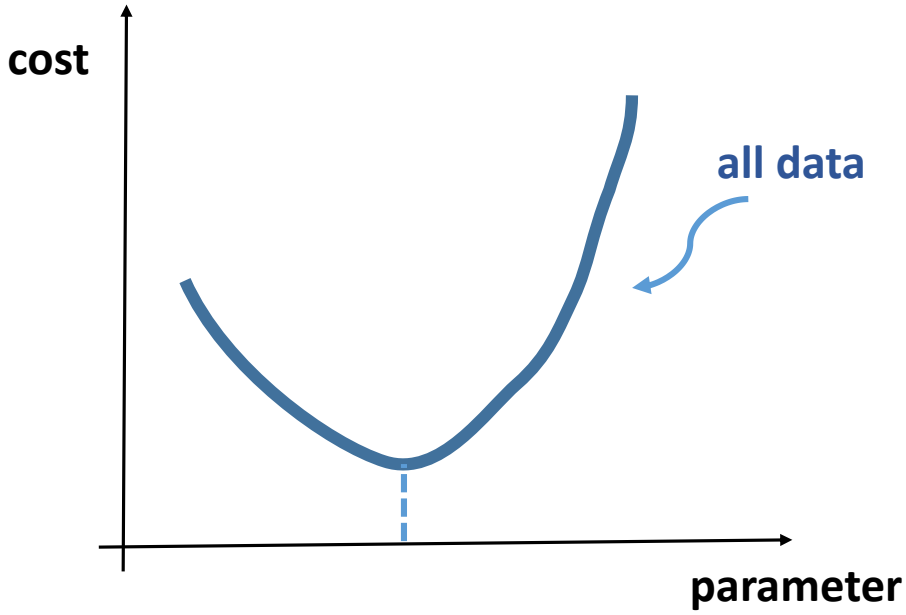
OPTIMIZER

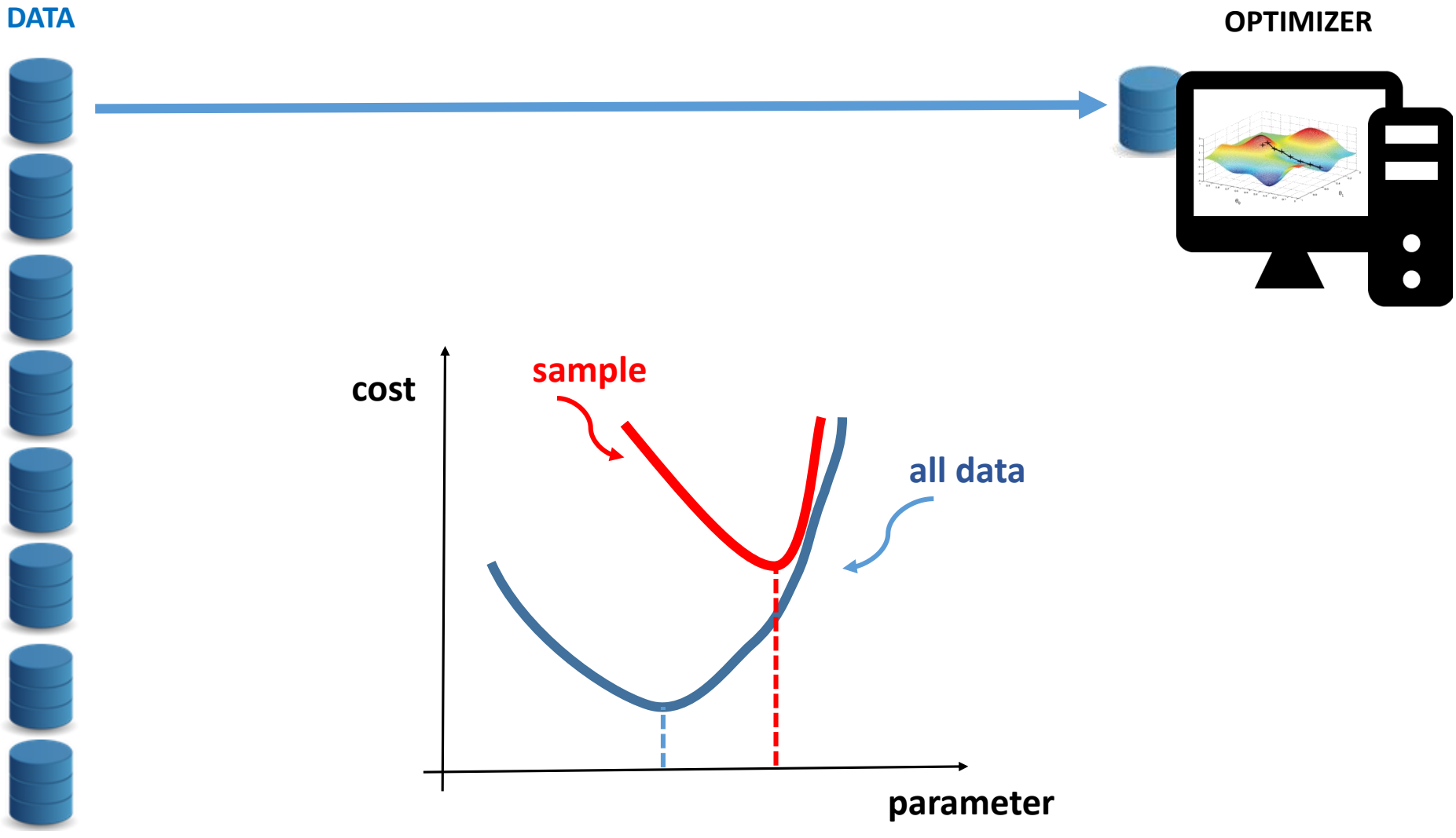


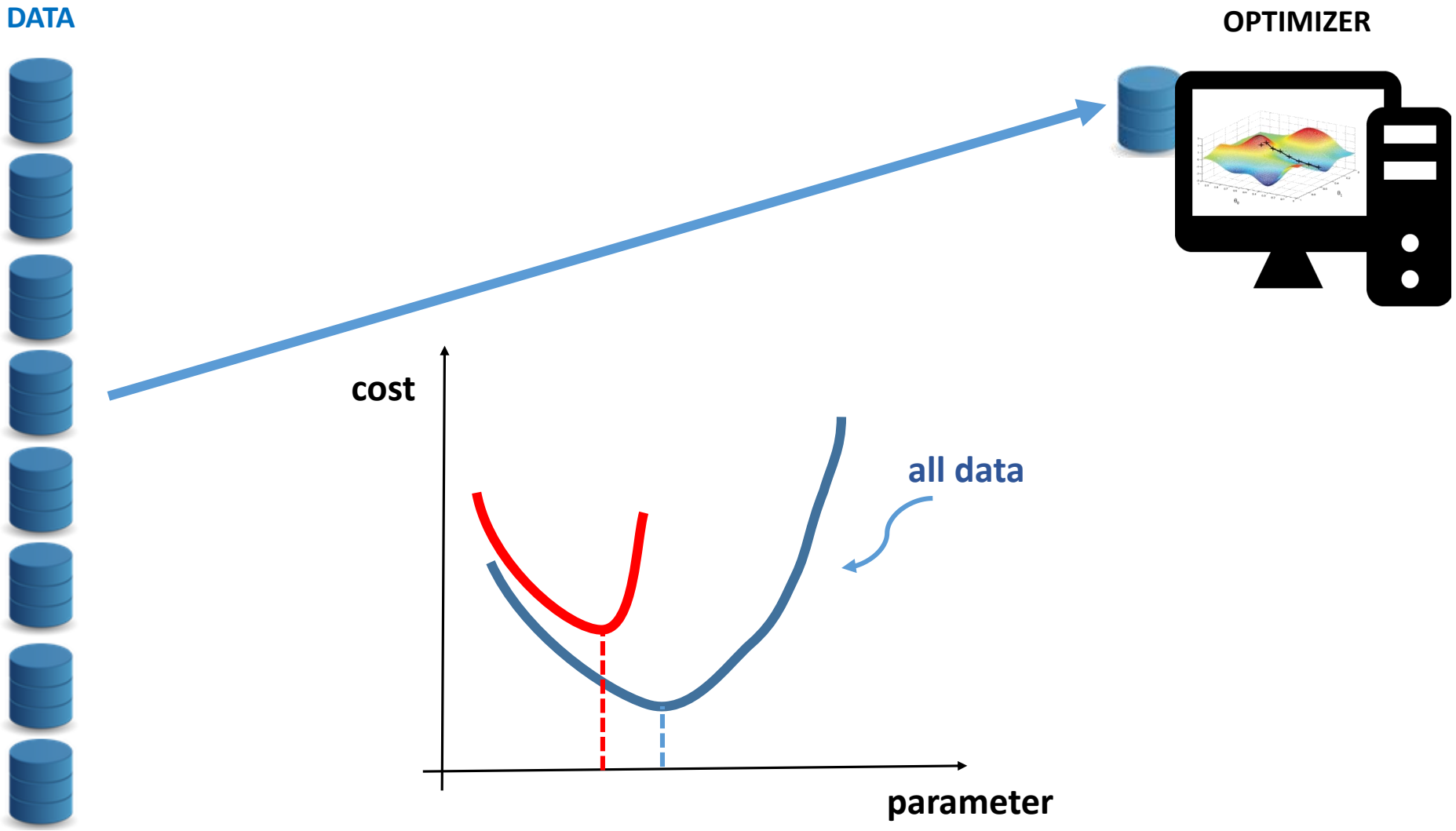
DATA

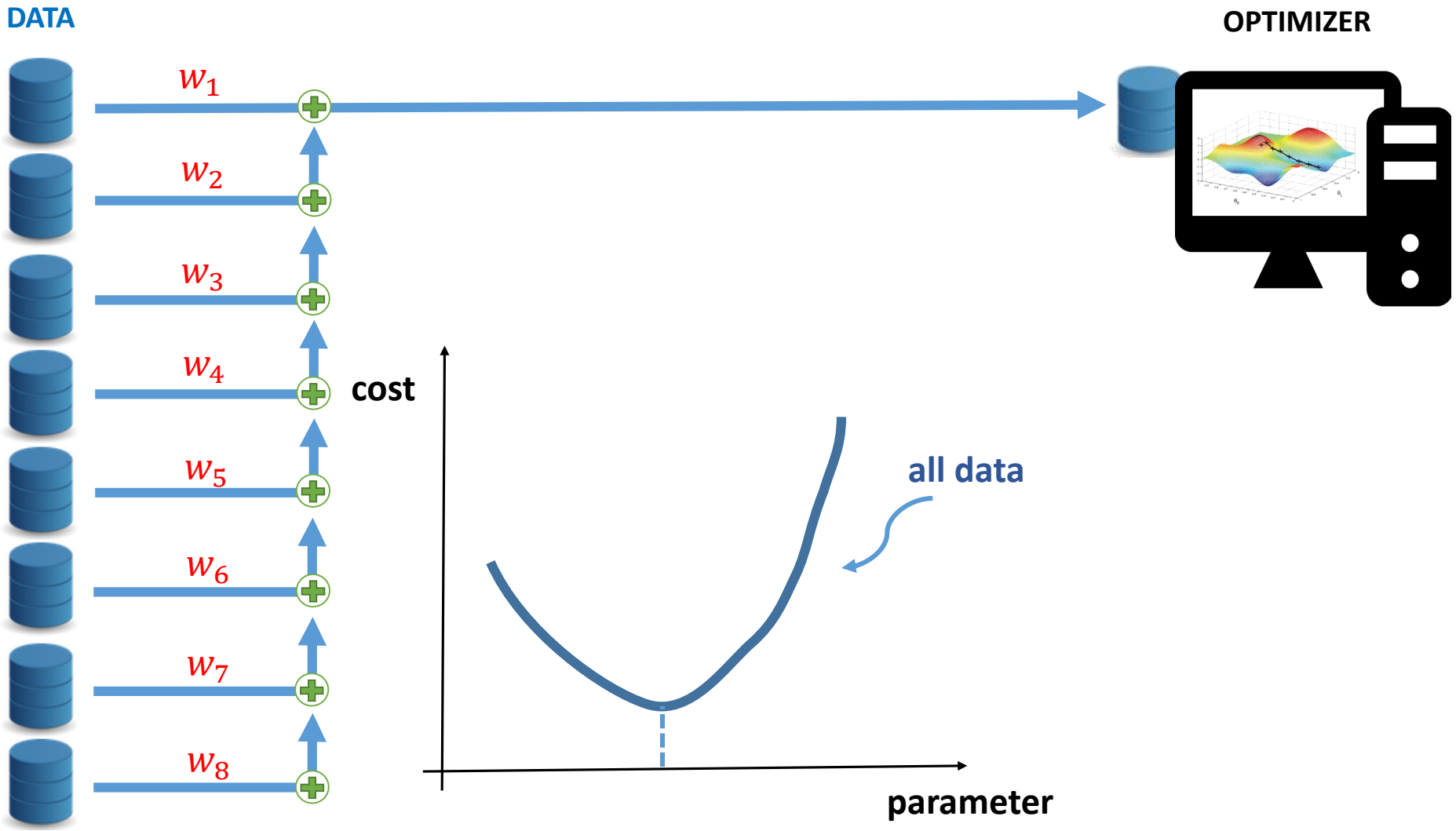
OPTIMIZER

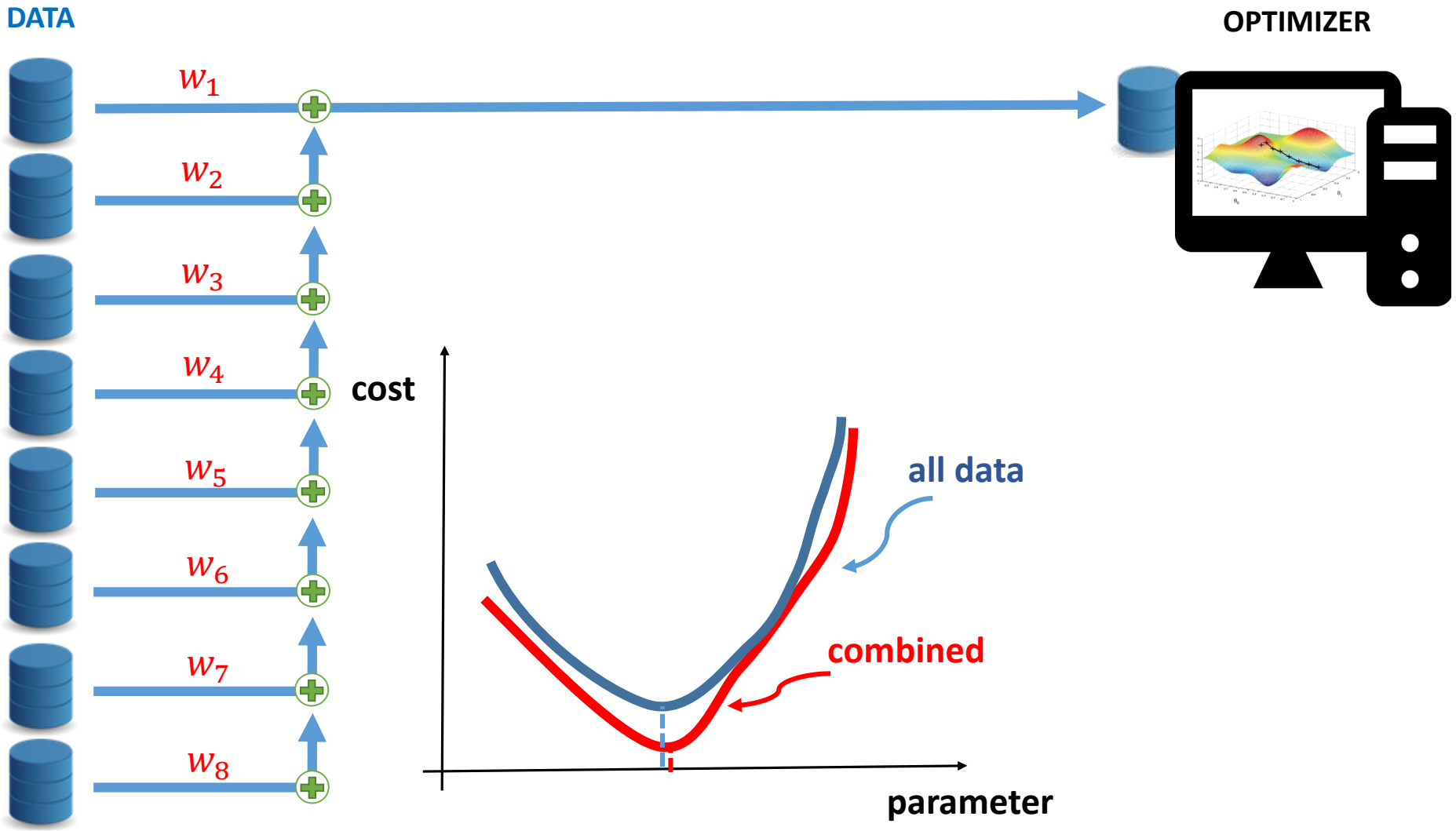




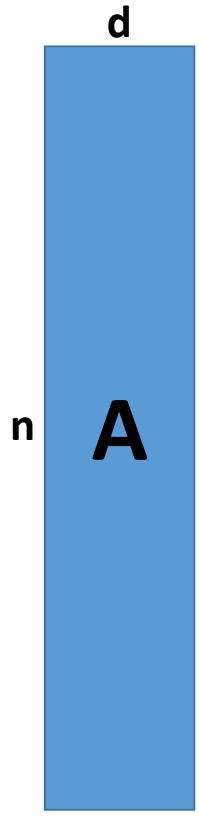

DATA
OPTIMIZER
cost
sample
all data
parameter


DATA
OPTIMIZER
cost
all data
parameter

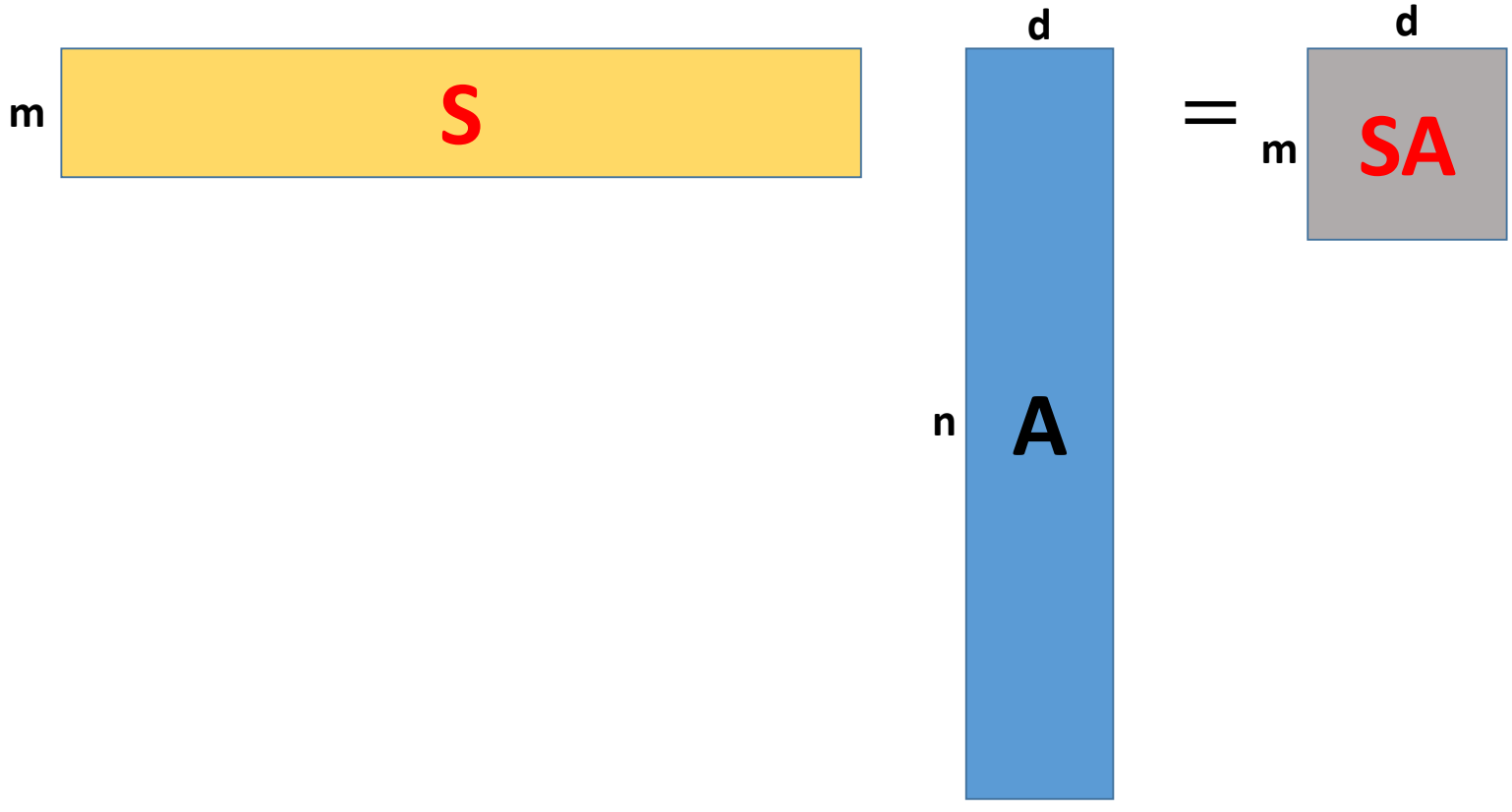

DATA
OPTIMIZER
w_1
w_2
w_3
w_4
w_5
w_6
w_7
w_8
cost
all data
parameter


DATA
OPTIMIZER
w_1
w_2
w_3
w_4
w_5
w_6
w_7
w_8
cost
all data
combined
parameter

Randomized Sketching



Randomized Sketching

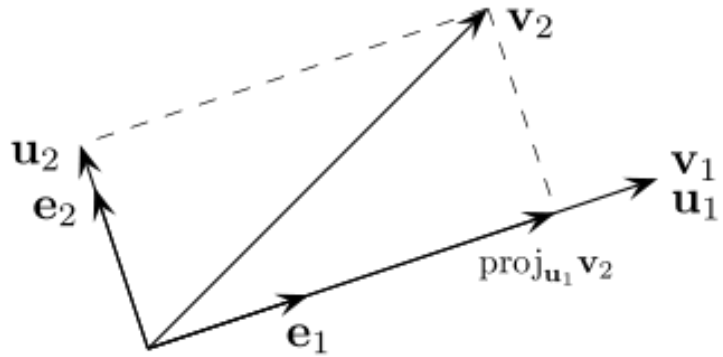


Randomized Least Squares Solvers

- $A : n \times d$ feature matrix, and $y : n \times 1$ response vector
- Original problem $\quad \mathbf{OPT} = \min_{x \in \mathcal{C}} \underbrace{\|Ax - y\|^2}$
- Randomized approximation $\quad \min_{x \in \mathcal{C}} \underbrace{\|\tilde{A}x - \tilde{y}\|^2}$
- $\tilde{A}$ and $\tilde{y}$ are smaller approximations

QR decomposition

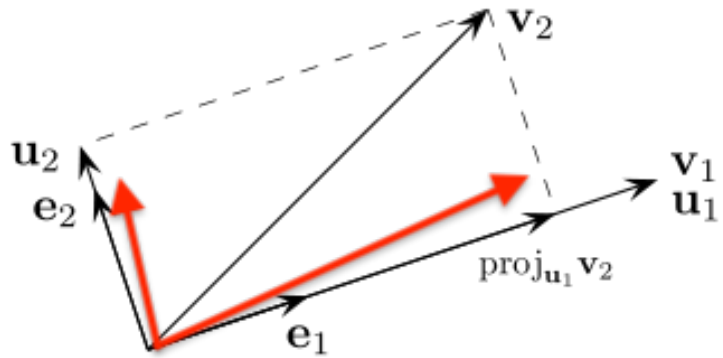
- The Gram–Schmidt process takes a finite, linearly independent set of vectors $v_1, ..., v_n \in \mathbb{R}^d$ generates an orthogonal set $u_1, ..., u_k \in \mathbb{R}^d$ that spans the same n-dimensional subspace.



$$
\begin{aligned}
\mathbf{u}_1 &= \mathbf{v}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|} \\
\mathbf{u}_2 &= \mathbf{v}_2 - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_2), & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|} \\
\mathbf{u}_3 &= \mathbf{v}_3 - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_3) - \text{proj}_{\mathbf{u}_2}(\mathbf{v}_3), & \mathbf{e}_3 &= \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|} \\
\mathbf{u}_4 &= \mathbf{v}_4 - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_4) - \text{proj}_{\mathbf{u}_2}(\mathbf{v}_4) - \text{proj}_{\mathbf{u}_3}(\mathbf{v}_4), & \mathbf{e}_4 &= \frac{\mathbf{u}_4}{\|\mathbf{u}_4\|} \\
&\vdots & &\vdots
\end{aligned}
$$

QR decomposition

- The Gram–Schmidt process takes a finite, linearly independent set of vectors $v_1, ..., v_n \in \mathbb{R}^d$ generates an orthogonal set $u_1, ..., u_k \in \mathbb{R}^d$ that spans the same n-dimensional subspace.



- complexity $O(dn^2)$
- randomized algorithm complexity $\approx O(dn)$

  produces an *approximately orthogonal* basis

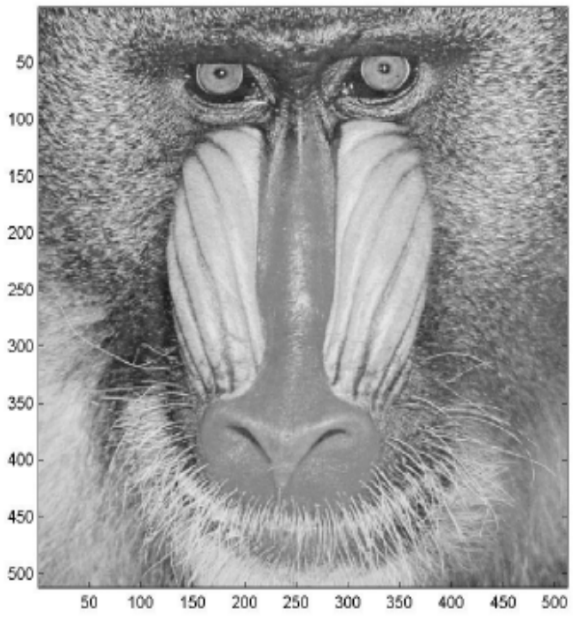
Low-rank matrix approximations

- Singular Value Decomposition (SVD)
- $A = U\Sigma V^T$
- takes $O(nd^2)$ time for $A \in R^{n \times d}$
- best rank-k approximation is $A_k := U_k \Sigma_k V_k^T = \sum_{i=1}^{k} \sigma_i u_i v_i^T$
- $\|A - A_k\|_2 \leq \sigma_{k+1}$

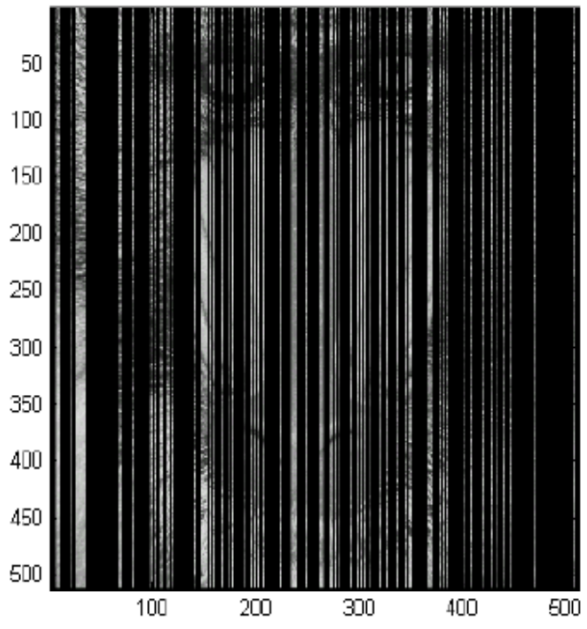
Randomized low-rank matrix approximations

- Randomized (SVD)
- approximation C (e.g. a subset of the columns of A)
- $AA^T \approx CC^T$
- $\tilde{A}_k = CC^\dagger A$ is a randomized rank-k approximation
- $\|A - \tilde{A}_k\|_2^2 \leq \sigma_{k+1}^2 + \epsilon \|A\|_2^2$

Randomized low-rank approximation example

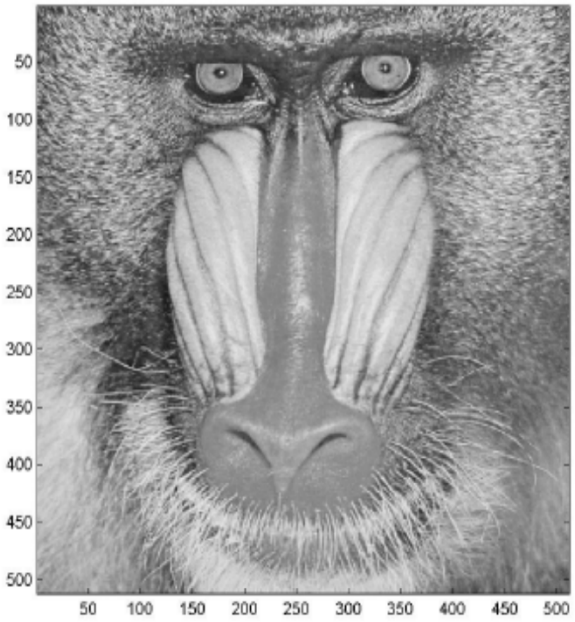


Original matrix

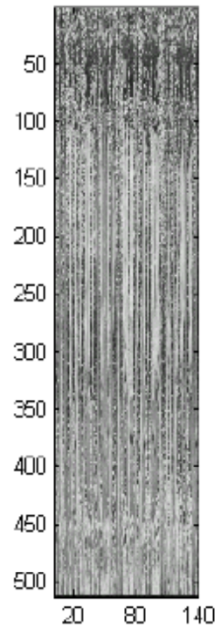


Sampling (c = 140 columns)

Randomized low-rank approximation example

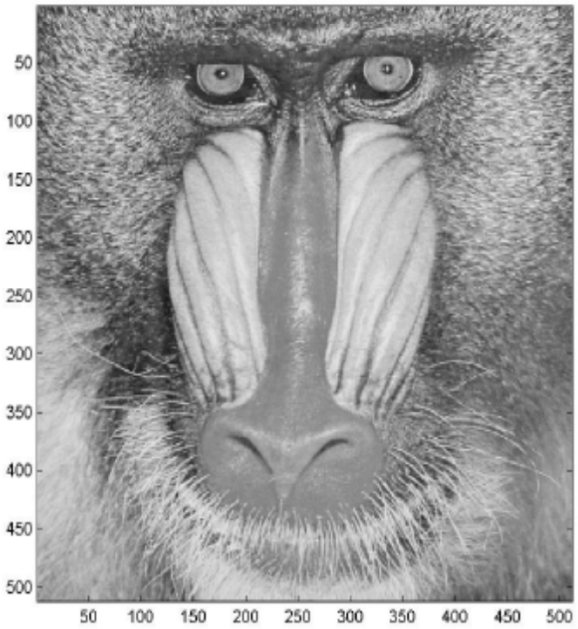


Original matrix

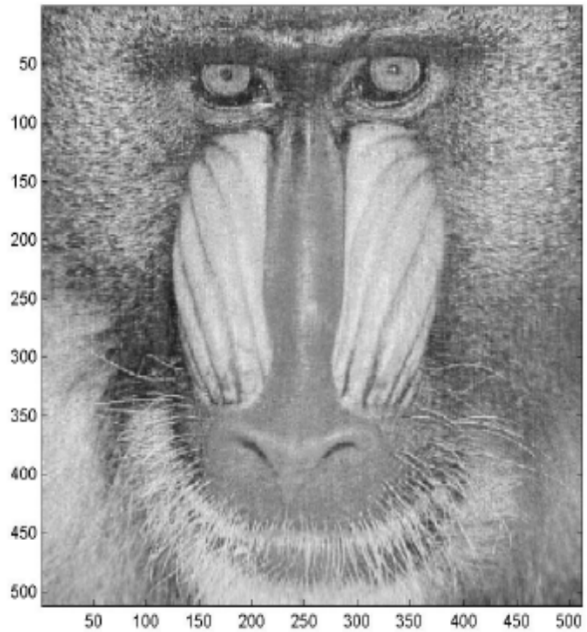


Sampling (c = 140 columns)

Randomized low-rank approximation example



A



CX

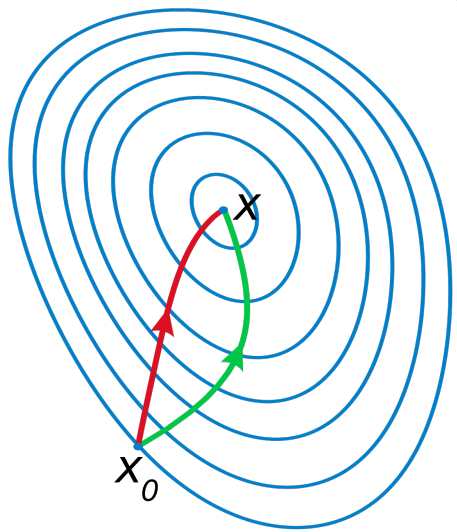
Iterative Methods

- Gradient descent and momentum acceleration
- Iterative sketching methods
- Conjugate gradient
- Preconditioning
- Newton's method
- Quasi-Newton methods
- Stochastic Gradient Descent
- Variance reduction and adaptive gradient methods

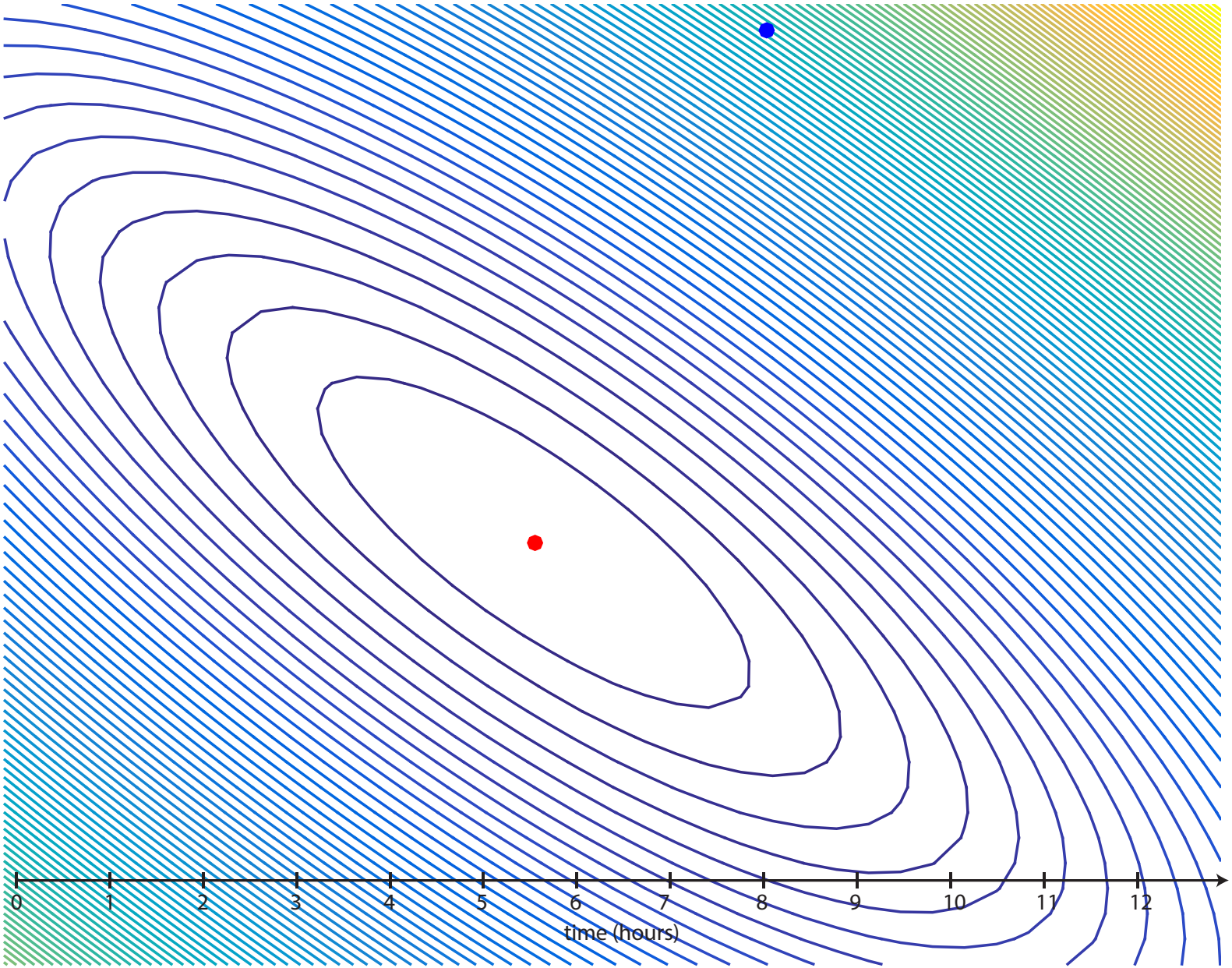
Newton's Method

$$\min_{x \in \mathcal{C}} g(x)$$

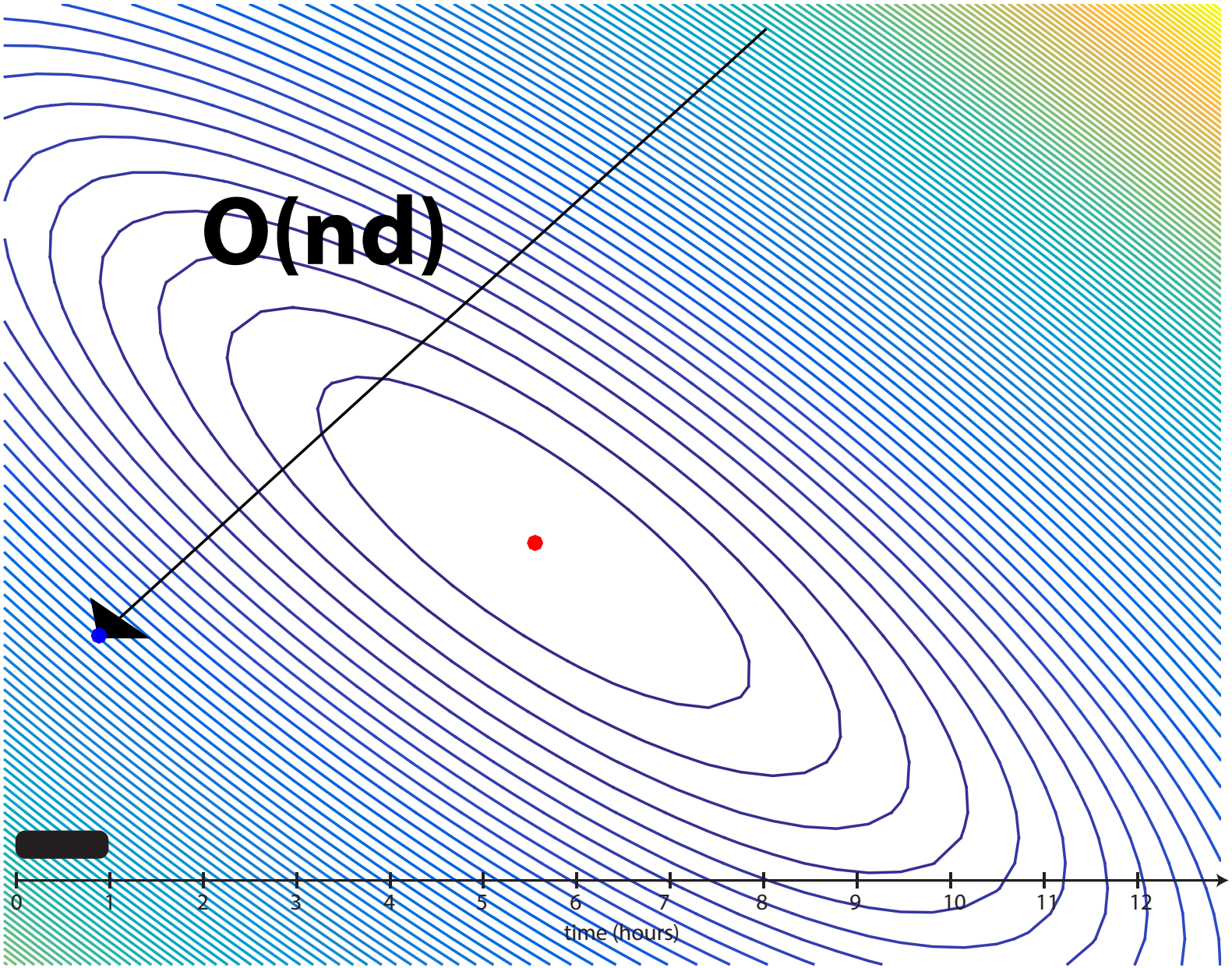
$$x^{t+1} = \arg\min_{x \in \mathcal{C}} \ \langle \nabla g(x^t),\, x - x^t \rangle + \frac{1}{2}(x - x^t)^T \nabla^2 g(x^t)(x - x^t)$$



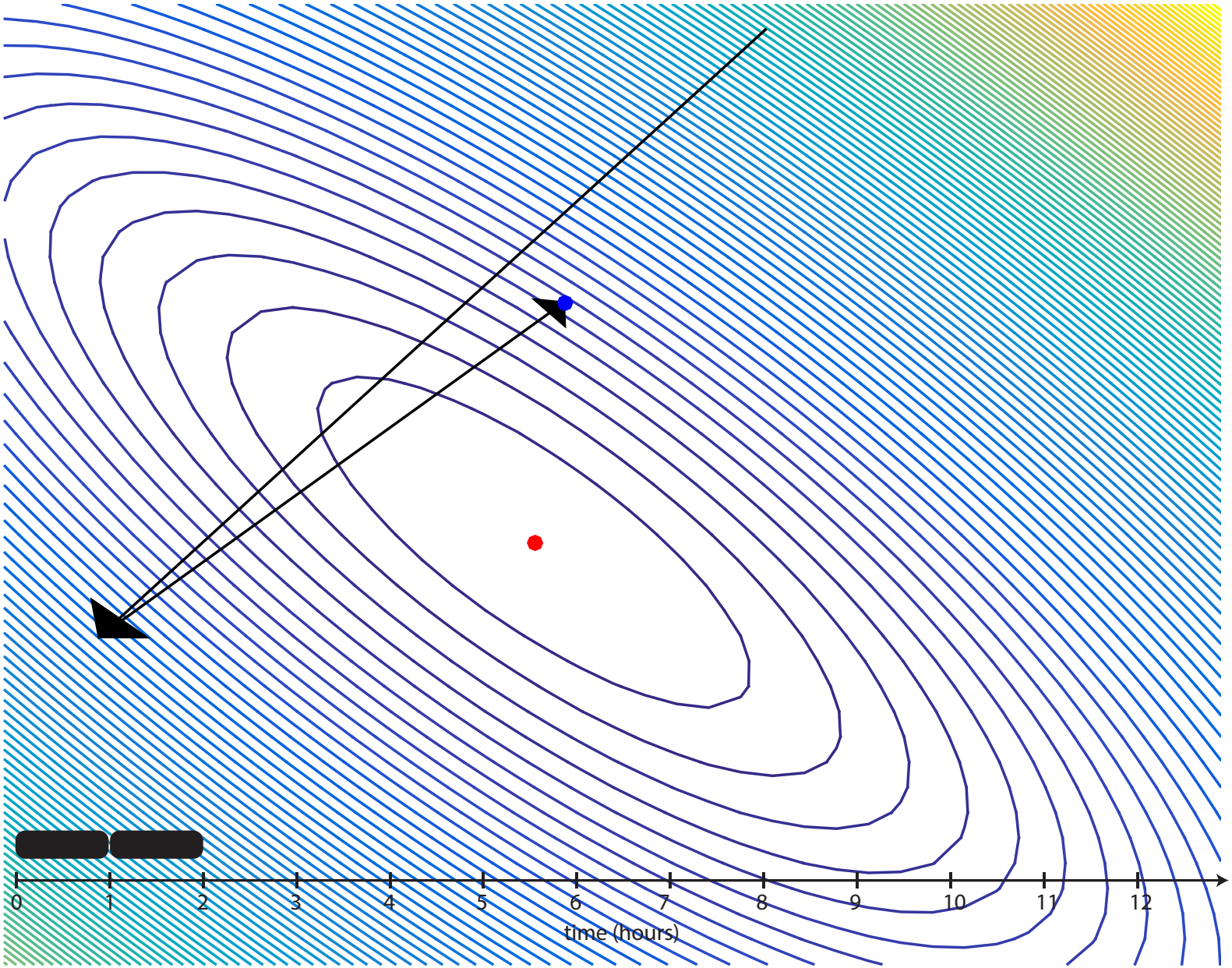
Gradient Descent



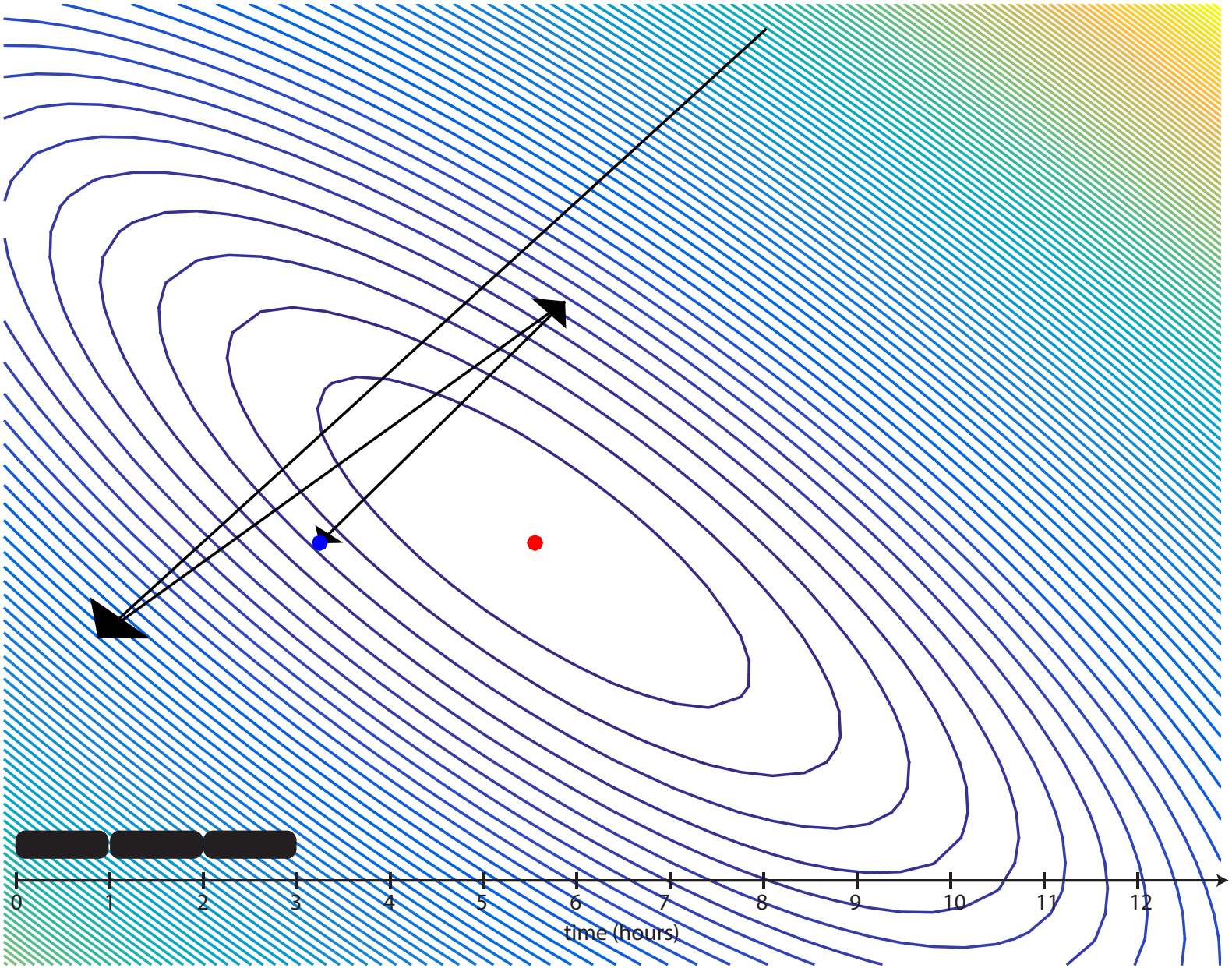
Gradient Descent



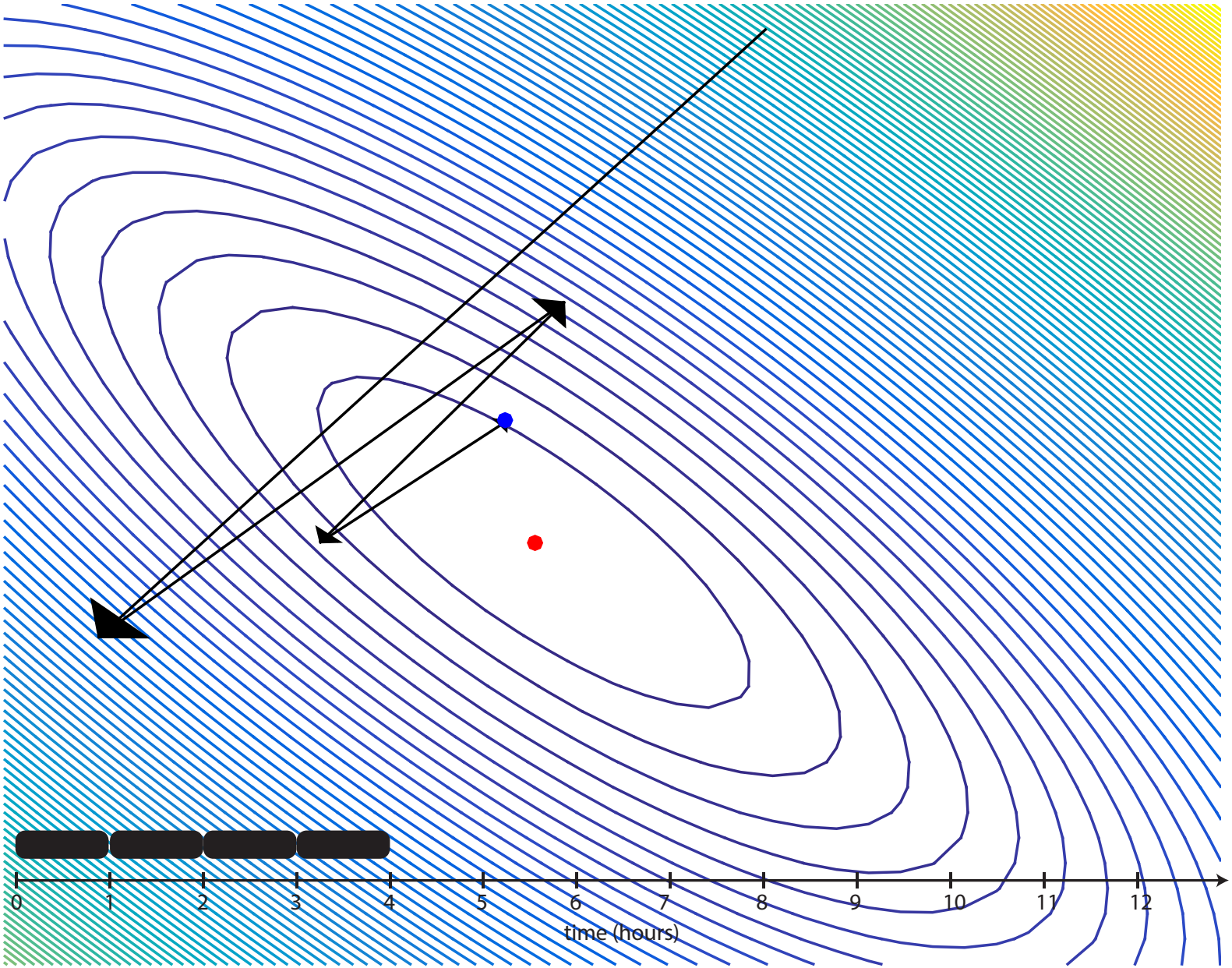
Gradient Descent



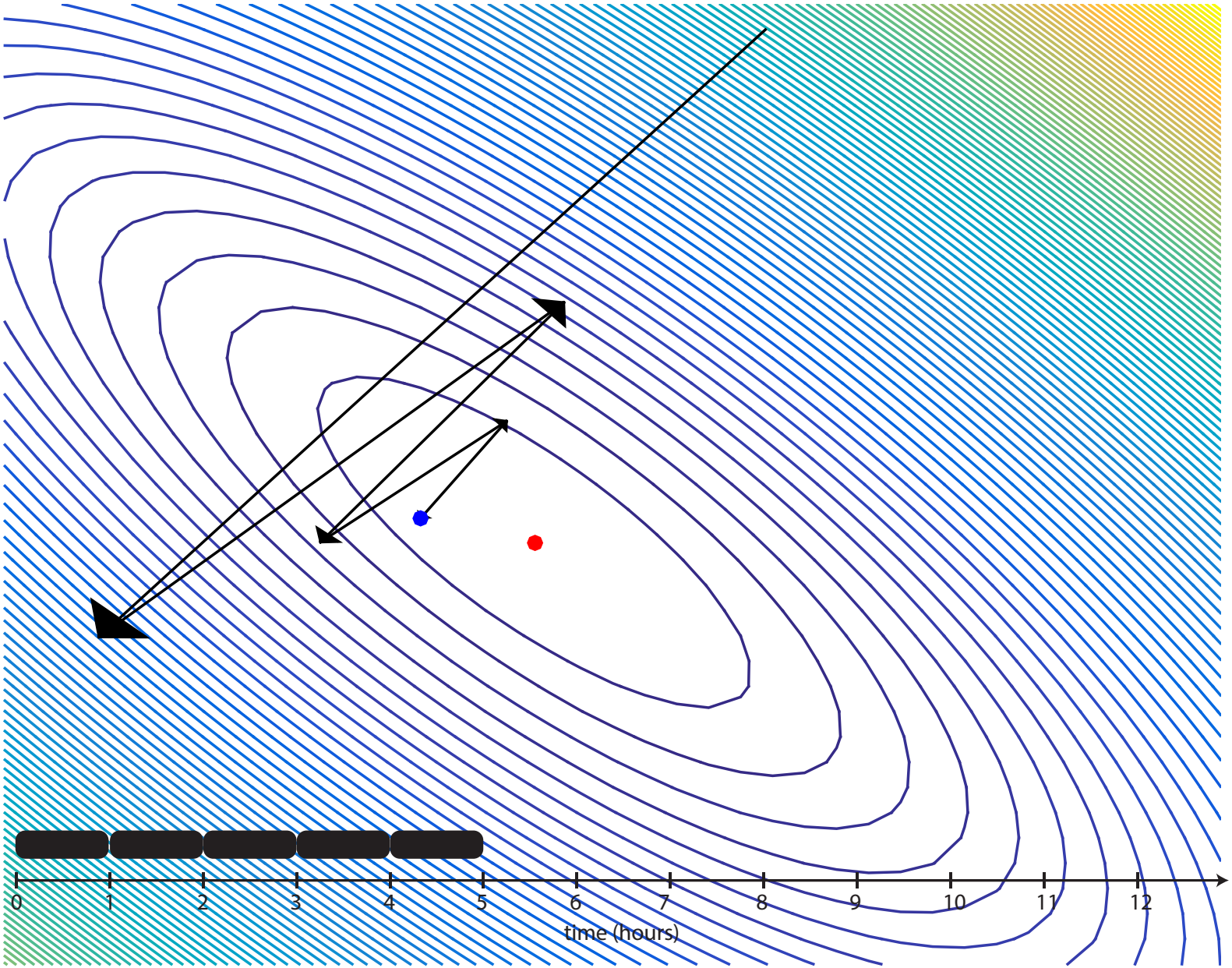
Gradient Descent



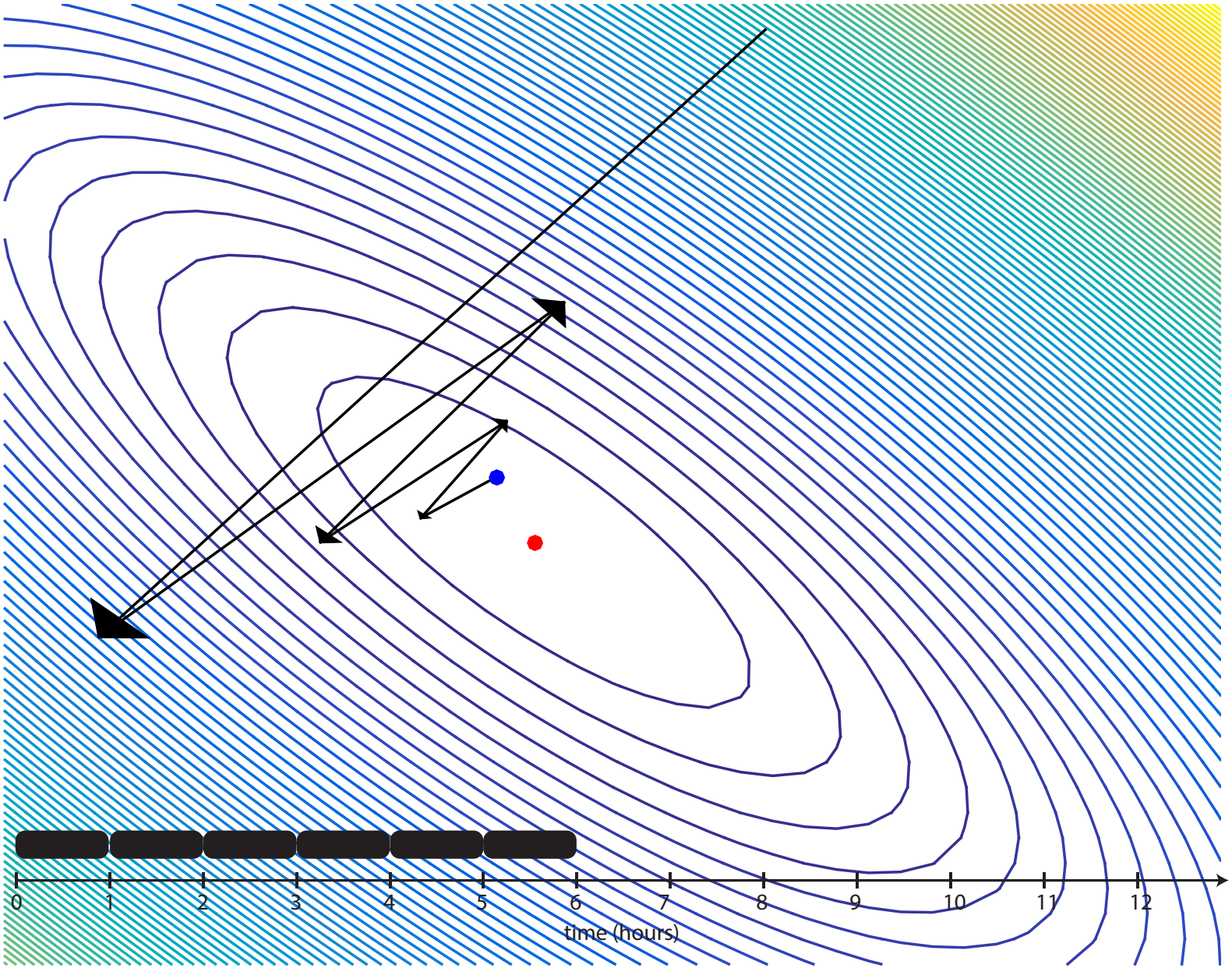
Gradient Descent



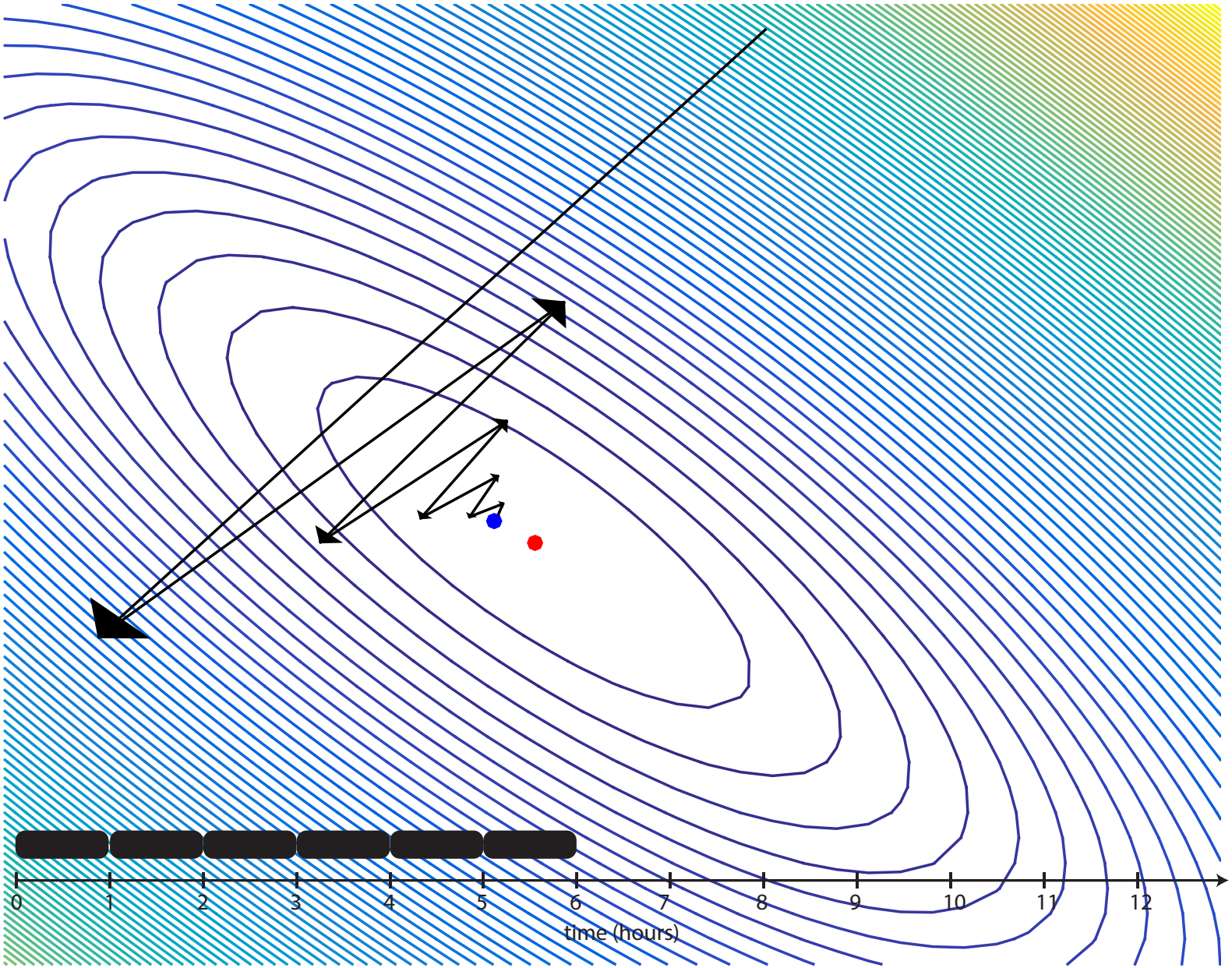
Gradient Descent



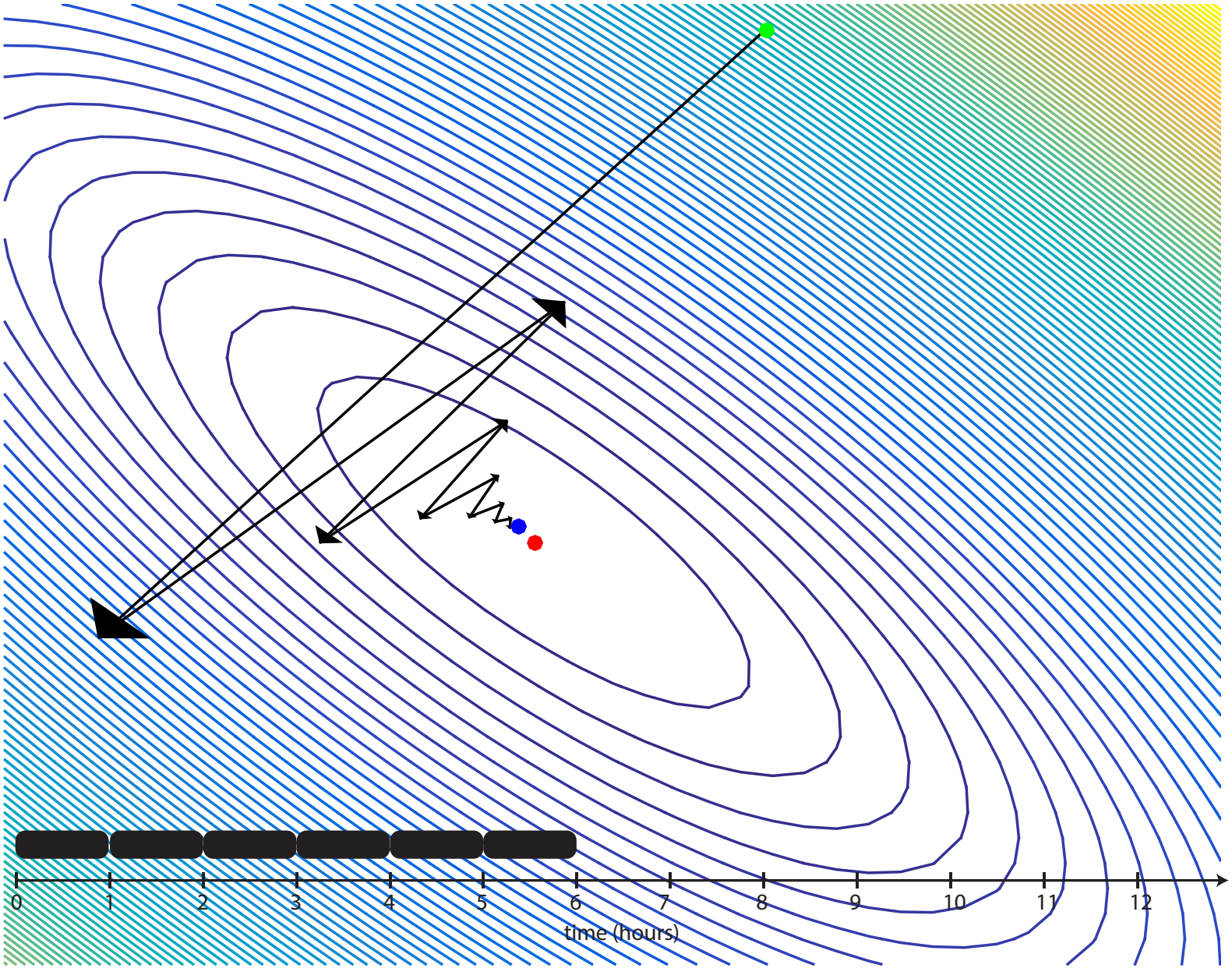
Gradient Descent



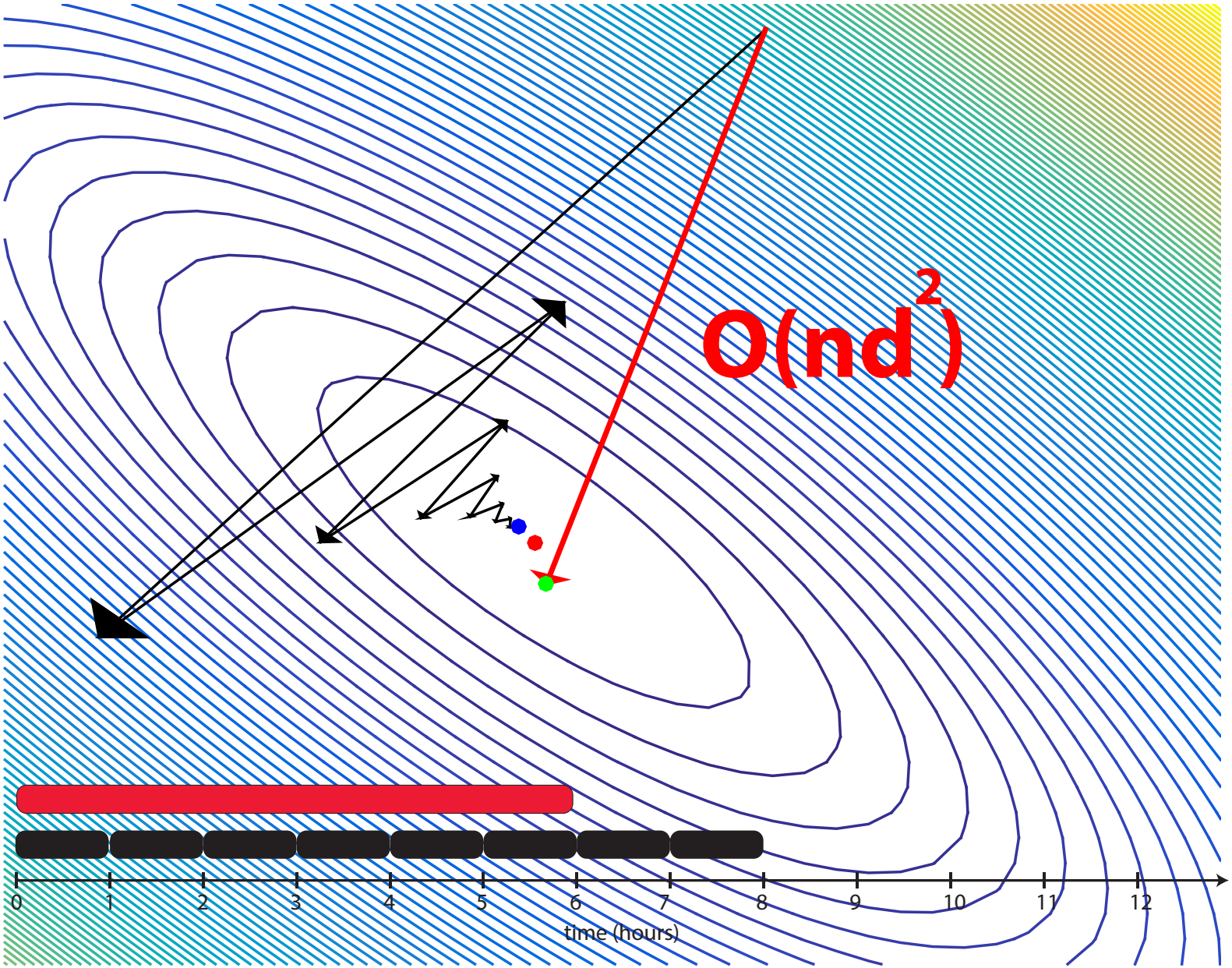
Gradient Descent vs



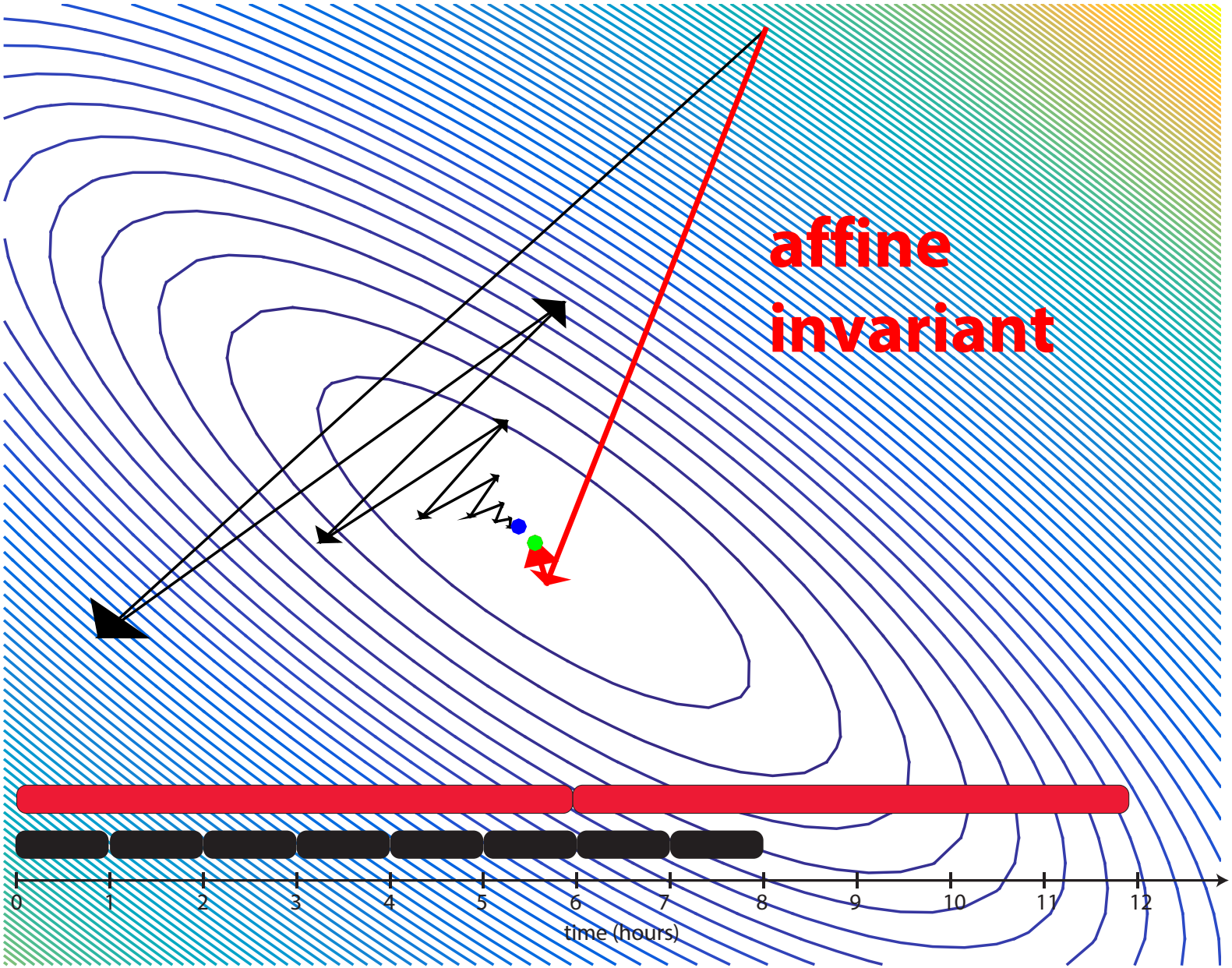
Gradient Descent vs Newton's Method



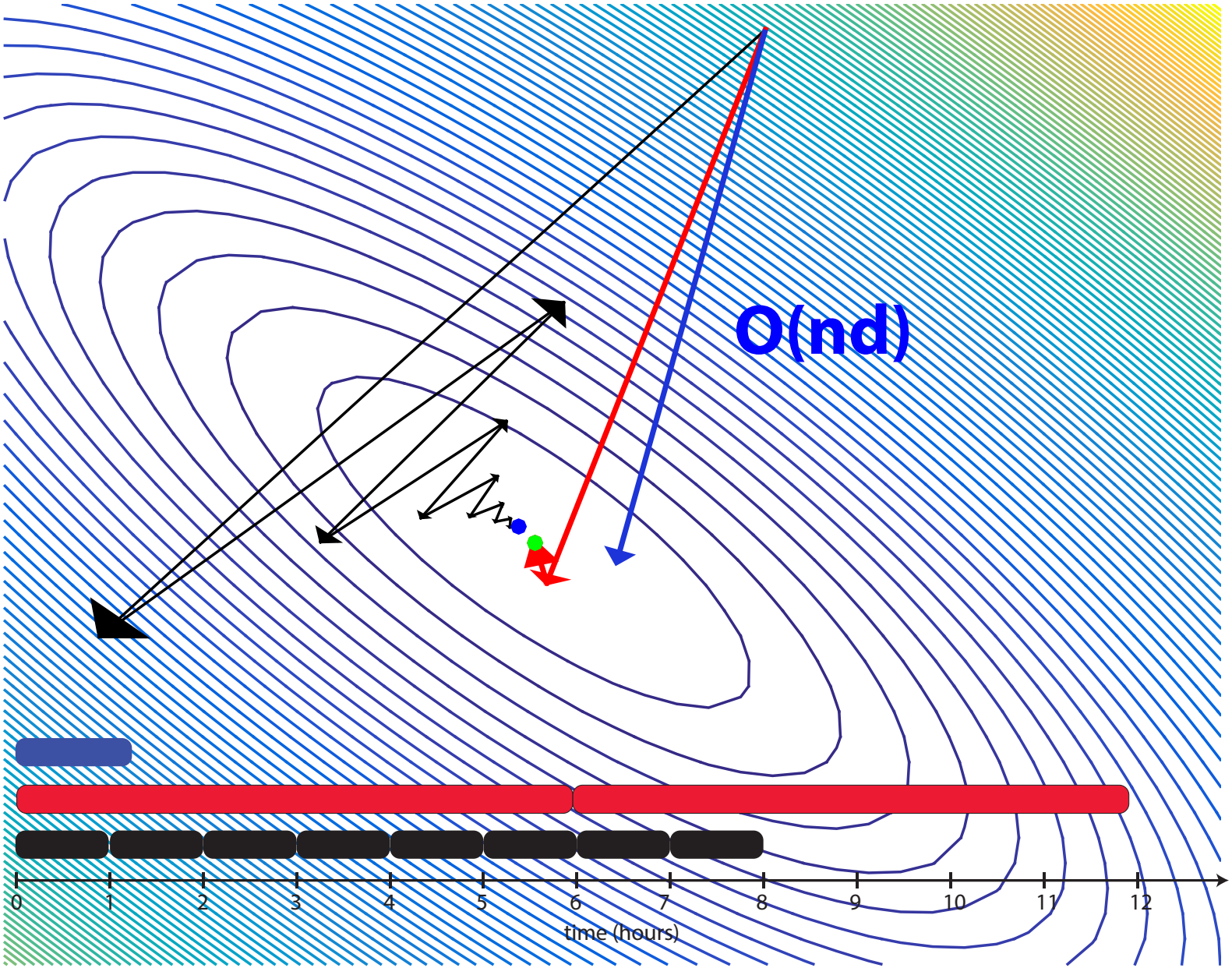
Gradient Descent vs Newton's Method



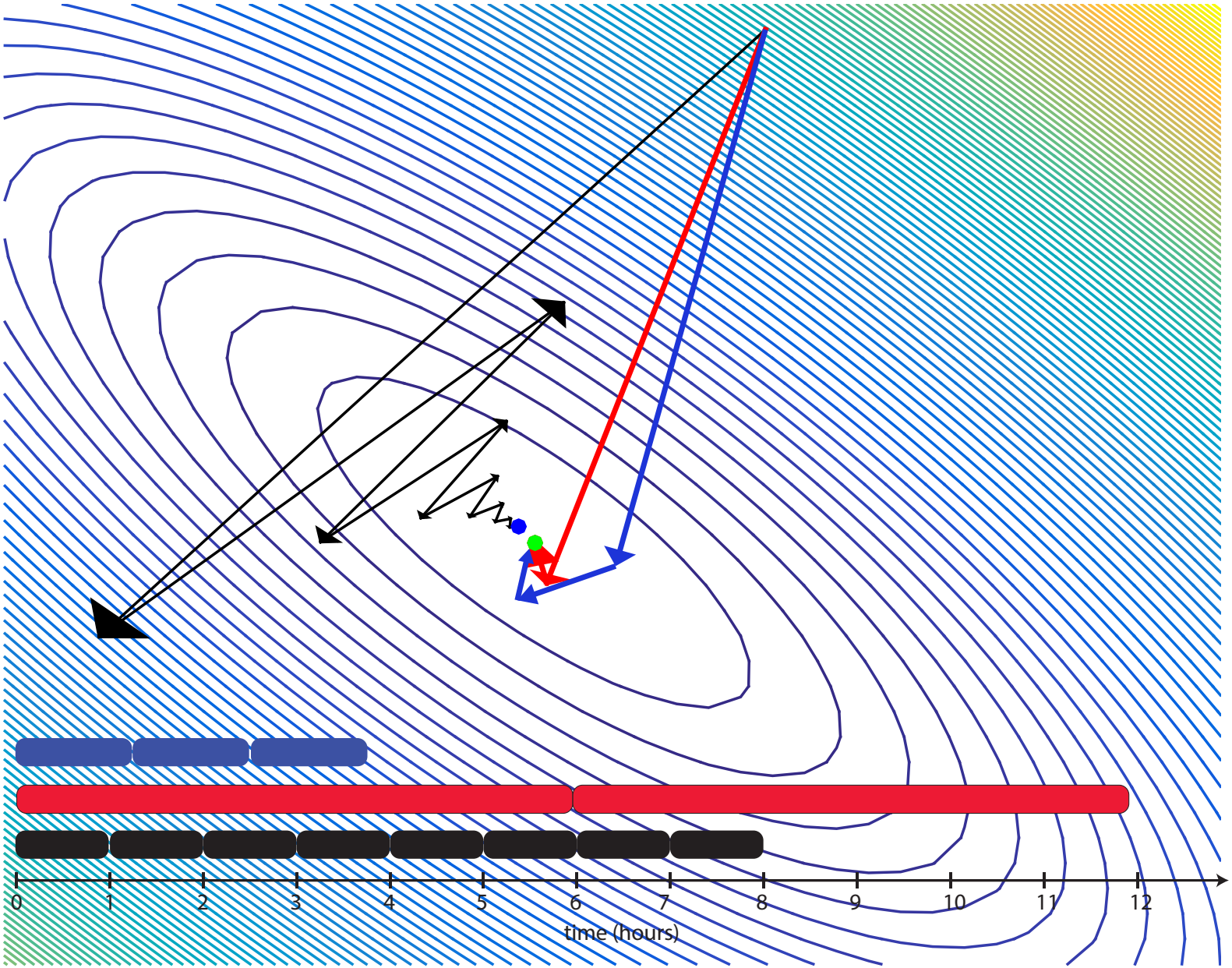
Gradient Descent vs Newton's Method



Gradient Descent vs Newton's Method



Gradient Descent vs Newton's Method

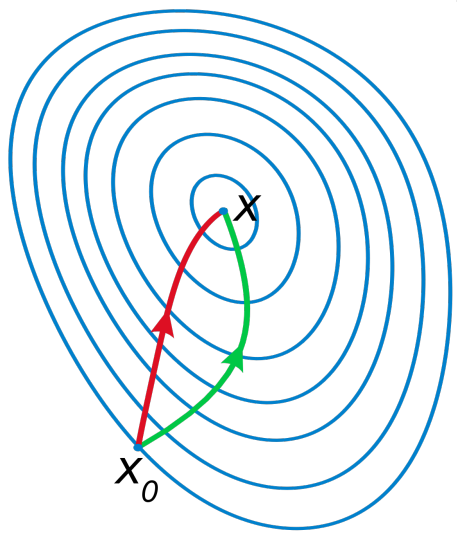


Randomized Newton's Method

$$\min_{x \in \mathcal{C}} g(x)$$

$$x^{t+1} = \arg\min_{x \in \mathcal{C}} \ \langle \nabla g(x^t),\, x - x^t \rangle + \frac{1}{2}(x - x^t)^T \tilde{\nabla}^2 g(x^t)(x - x^t)$$

- $\tilde{\nabla}^2 g(x^t) \approx \nabla^2 g(x^t)$ is an approximate Hessian



Randomized Newton's Method

$$\min_{x \in \mathcal{C}} g(x)$$

$$x^{t+1} = \arg\min_{x \in \mathcal{C}} \ \langle \nabla g(x^t),\, x - x^t \rangle + \frac{1}{2}(x - x^t)^T \tilde{\nabla}^2 g(x^t)(x - x^t)$$

- $\tilde{\nabla}^2 g(x^t) \approx \nabla^2 g(x^t)$ is a randomly projected Hessian matrix

Other diagonal, subsampled, low-rank approximations:

- Adagrad, ADAM
- Stochastic Variance Reduced Gradient (SVRG)
- Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm

Linear Programming

- LP in standard form where $A \in R^{n \times d}$

$$\min_{Ax \leq b} \; c^T x$$

- Log barrier

$$\min_x \mu c^T x - \sum_{i=1}^{n} \log(b_i - a_i^T x)$$

Linear Programming

- LP in standard form where $A \in R^{n \times d}$

$$\min_{Ax \leq b} \; c^T x$$

- Log barrier

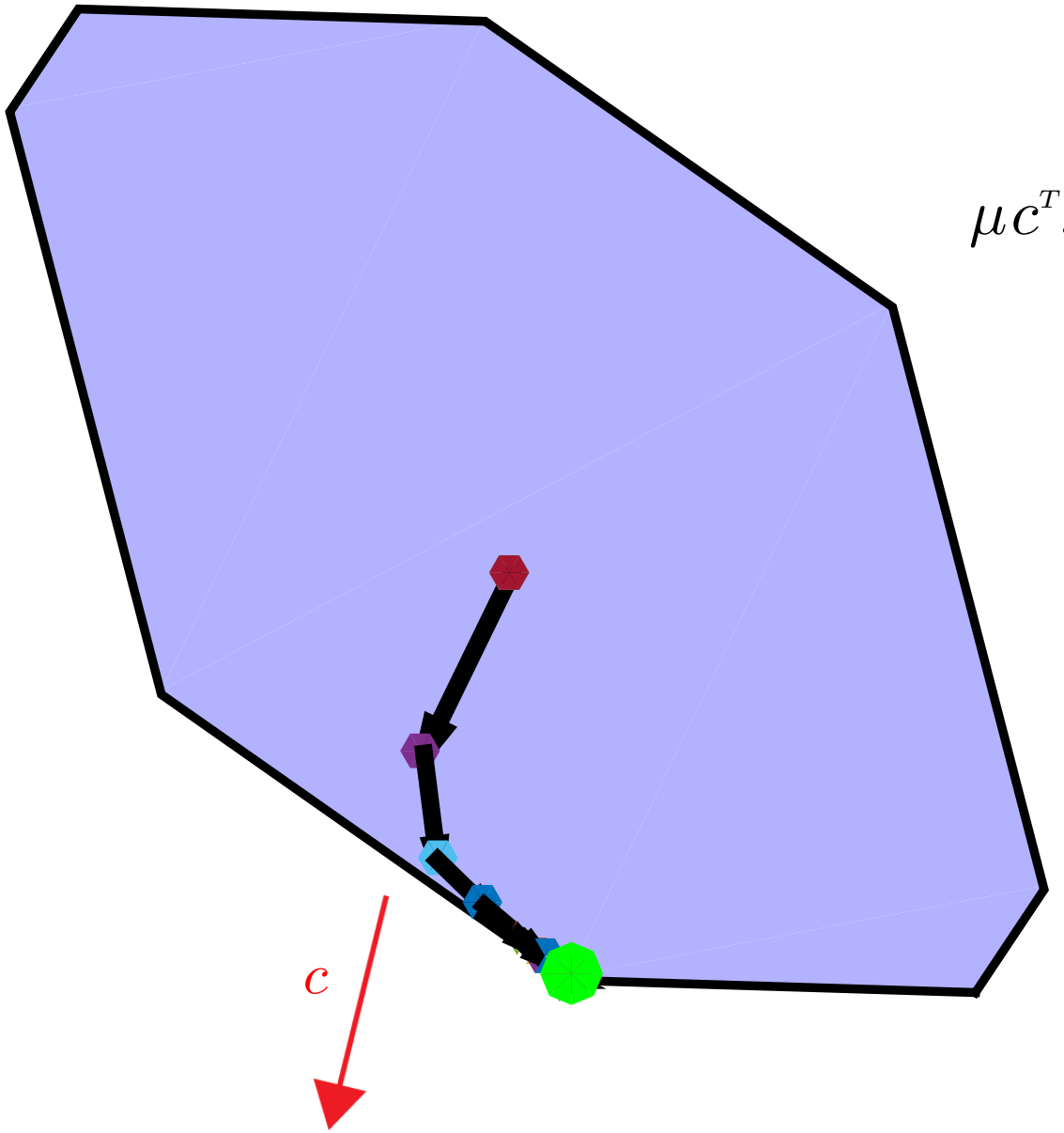
$$\min_x \mu c^T x - \sum_{i=1}^{n} \log(b_i - a_i^T x)$$

- Hessian $A^T diag\left(\frac{1}{(b_i - a_i^T x)^2}\right) A$ takes $O(nd^2)$ operations

$$\min_{Ax \leq b} c^T x$$

Exact Newton

$$\mu c^T x - \sum_{i=1}^{n} \log(b_i - a_i{}^T x)$$



Interior Point Methods for Linear Programming

- Hessian of $f(x) = c^T x - \sum_{i=1}^{n} \log(b_i - a_i^T x)$

$$\nabla^2 f(x) = A^T \, diag\left(\frac{1}{(b_i - a_i^T x)^2}\right) A \ ,$$

Interior Point Methods for Linear Programming

- Hessian of $f(x) = c^T x - \sum_{i=1}^{n} \log(b_i - a_i^T x)$

$$\nabla^2 f(x) = A^T \, diag\left(\frac{1}{(b_i - a_i^T x)^2}\right) A \ ,$$

- Root of the Hessian

$$(\nabla^2 f(x))^{1/2} = diag\left(\frac{1}{|b_i - a_i^T x|}\right) A \ ,$$

Interior Point Methods for Linear Programming

- Hessian of $f(x) = c^T x - \sum_{i=1}^{n} \log(b_i - a_i^T x)$

$$\nabla^2 f(x) = A^T diag\left(\frac{1}{(b_i - a_i^T x)^2}\right) A \ ,$$

- Root of the Hessian

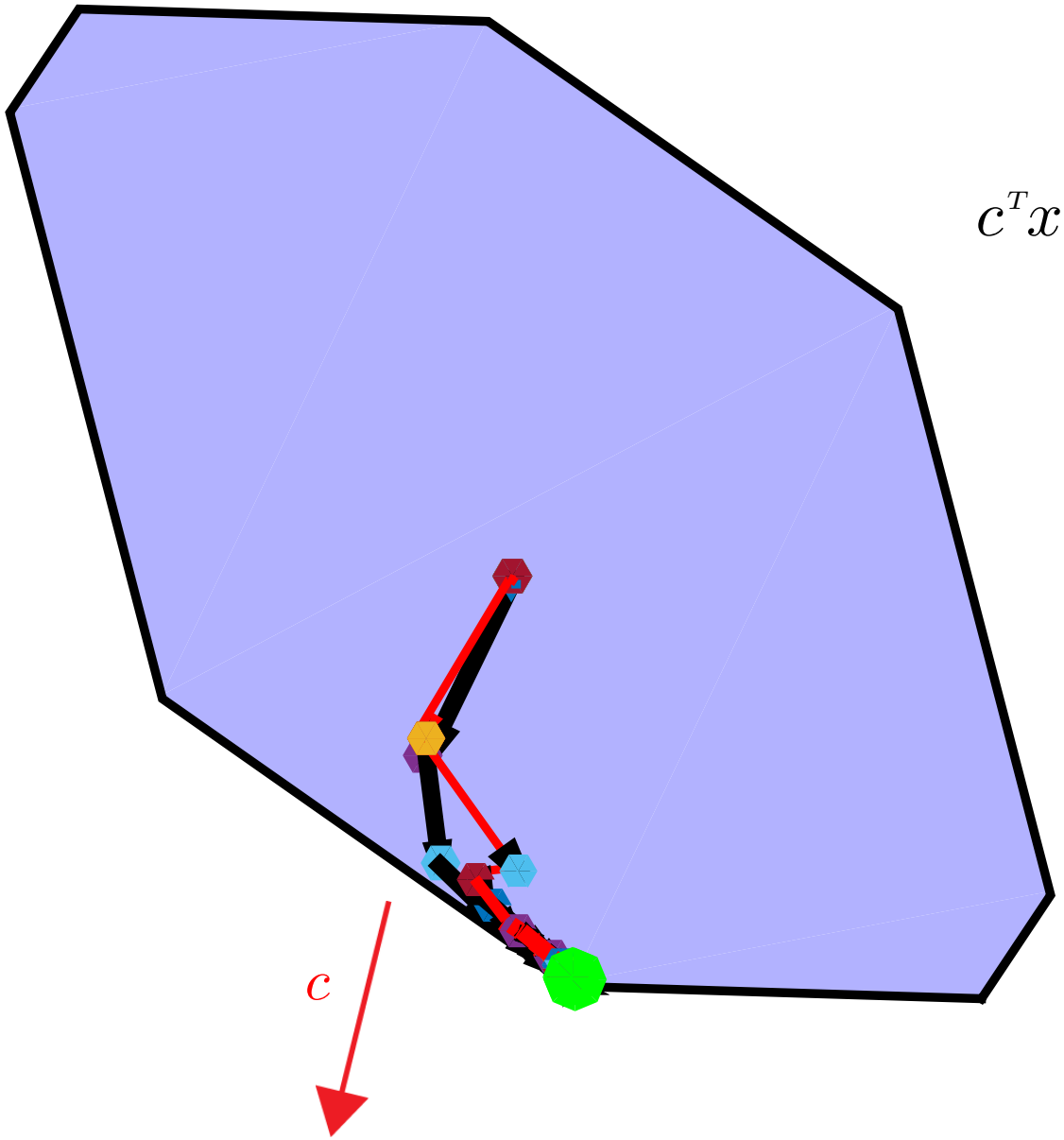
$$(\nabla^2 f(x))^{1/2} = diag\left(\frac{1}{|b_i - a_i^T x|}\right) A \ ,$$

- Sketch of the Hessian

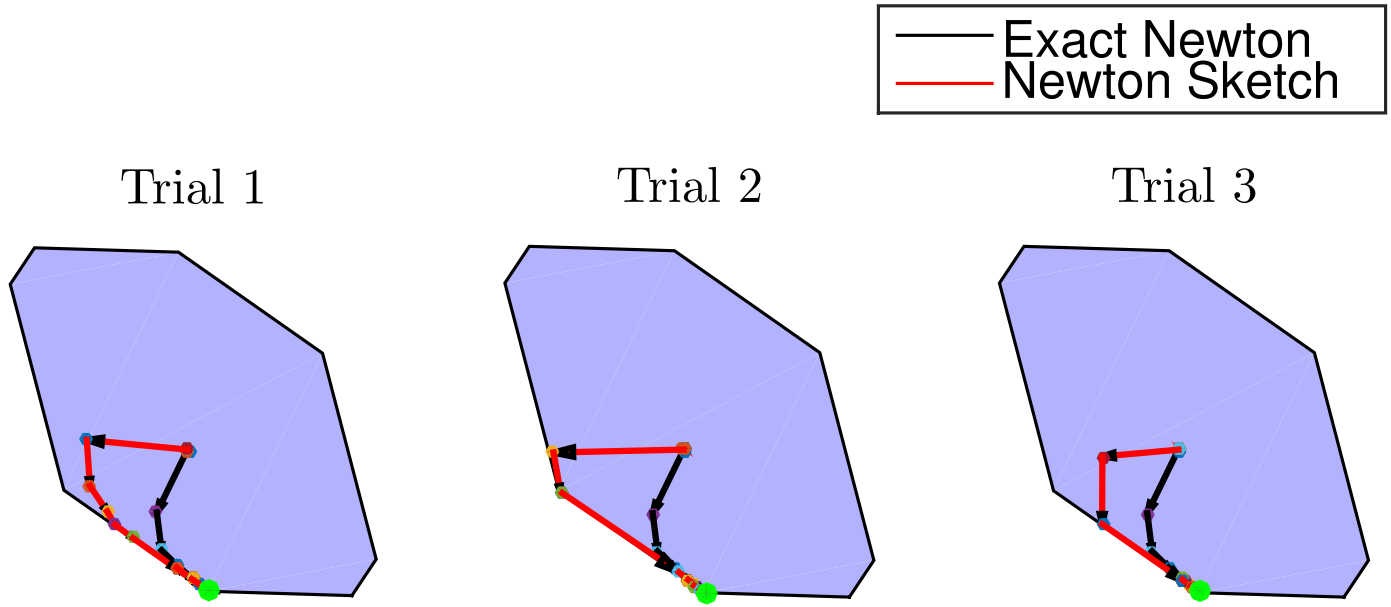
$$S^t(\nabla^2 f(x))^{1/2} = S^t diag\left(\frac{1}{|b_i - a_i^T x|}\right) A$$

takes $O(md^2)$ operations

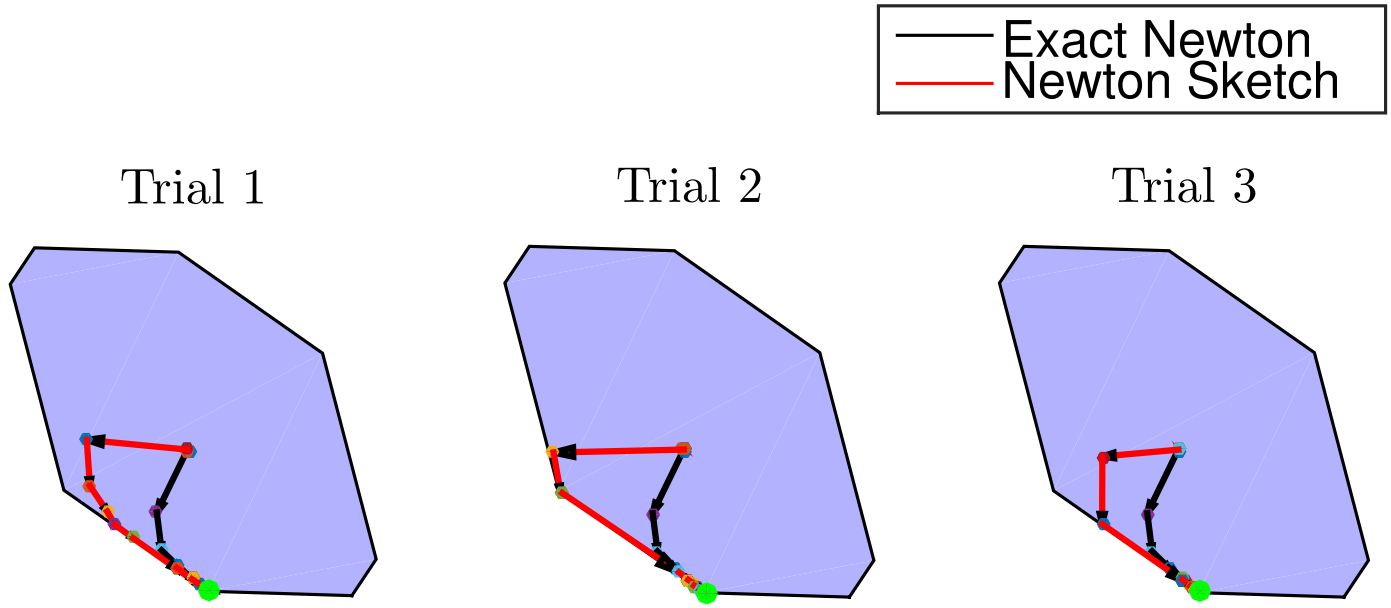
$$\min_{Ax \leq b} c^T x$$



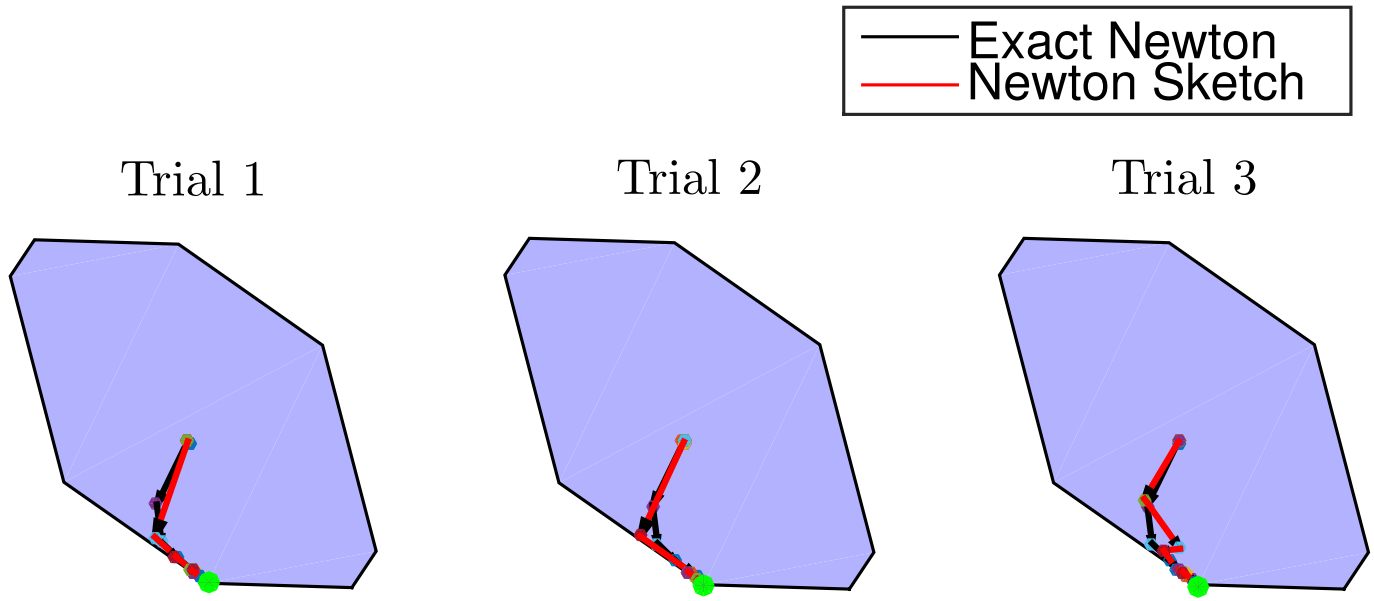
$$c^T x - \mu \sum_{i=1}^{n} \log(b_i - {a_i}^T x)$$



(a) sketch size $m = d$



(a) sketch size $m = d$



(b) sketch size $m = 4d$

High dimensional problems $n \ll d$

- $x^{t+1} = Ax_t + Bu_t, \quad t = 1, ..., T$

High dimensional problems $n \ll d$

- $x^{t+1} = Ax_t + Bu_t, \quad t = 1, ..., T$
- minimum fuel control from $0 \to x_f$

$$\min_u \ \|u\|_1$$
$$\text{s.t. } [\ B \quad AB \quad A^2B \quad \cdots\ ]u = x_f$$

- nT decision variables
- We can apply sampling and sketching for the variables $u \in \mathbb{R}^{nT}$
- Basic idea: dual linear program has nT constraints

Kernel methods

- Kernel matrices

  given data points $x_1, ..., x_n \in \mathbb{R}^d$

  e.g., Gaussian kernel $K_{ij} = e^{-\frac{1}{\sigma^2}\|x_i - x_j\|_2^2}$
- large $n \times n$ square matrices

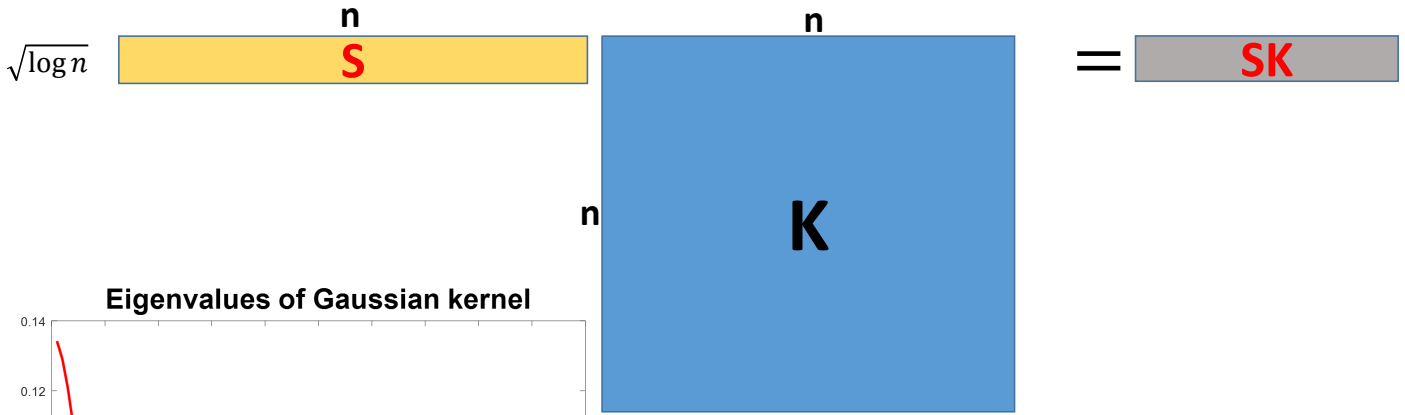
Kernel methods

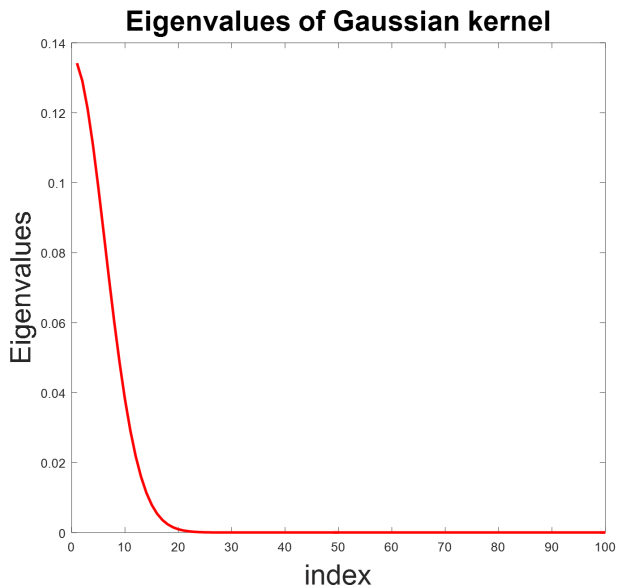
- Kernel matrices

  given data points $x_1, ..., x_n \in \mathbb{R}^d$

  e.g., Gaussian kernel $K_{ij} = e^{-\frac{1}{\sigma^2}\|x_i - x_j\|_2^2}$

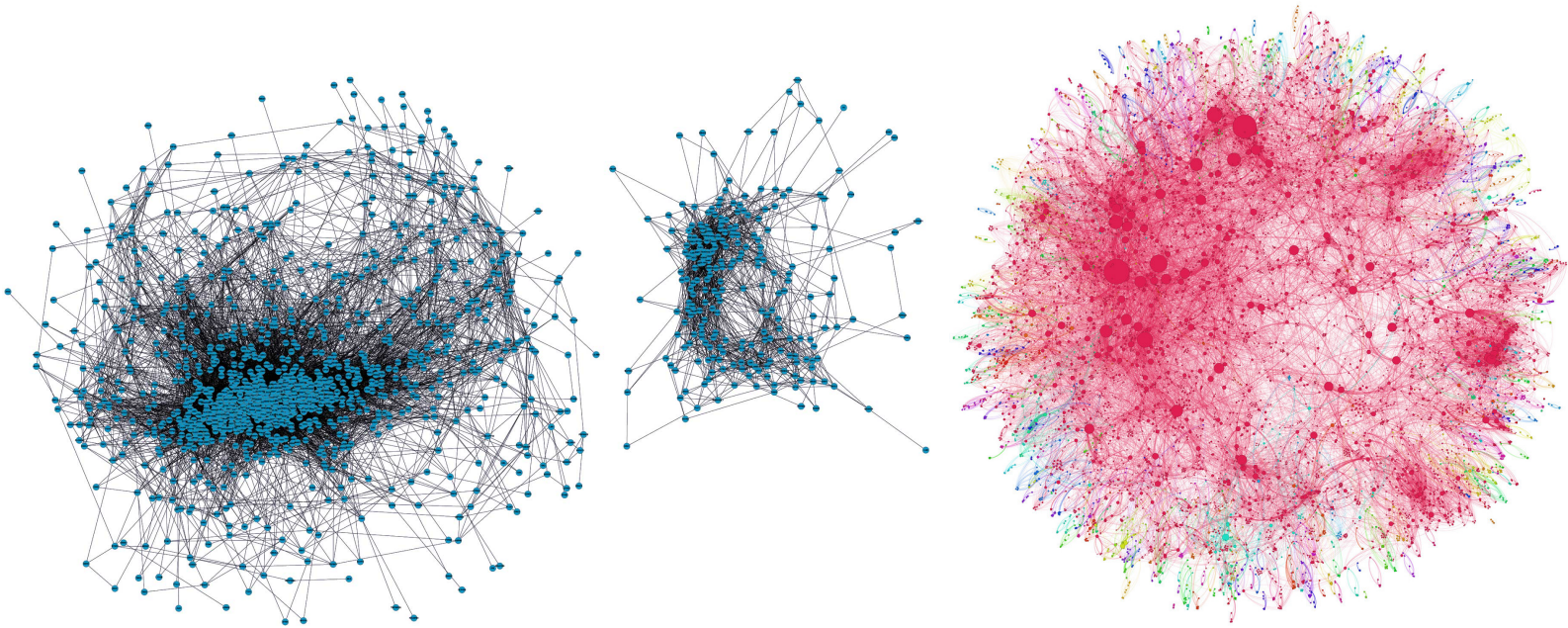
- large $n \times n$ square matrices





Large Graphs

- Adjacency matrix or Laplacian
- Examples: a gene network and a co-authorship network graph



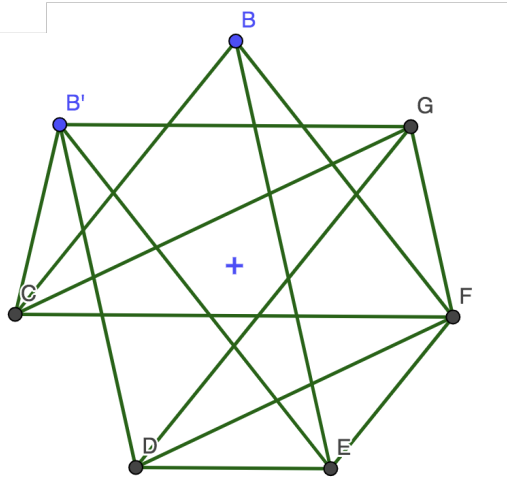
Sampling Graphs

- Random sampling graphs

14 edges

Adjacency Matrix:

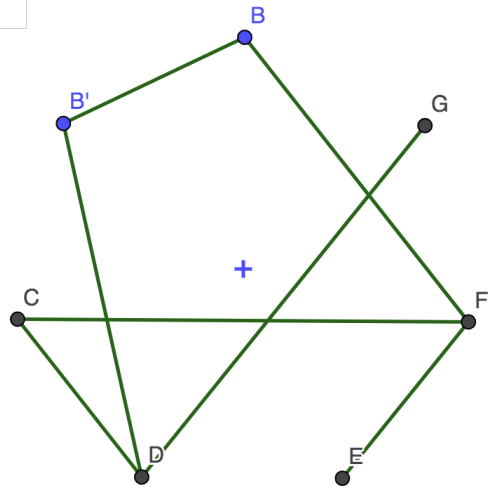
$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$



7 edges

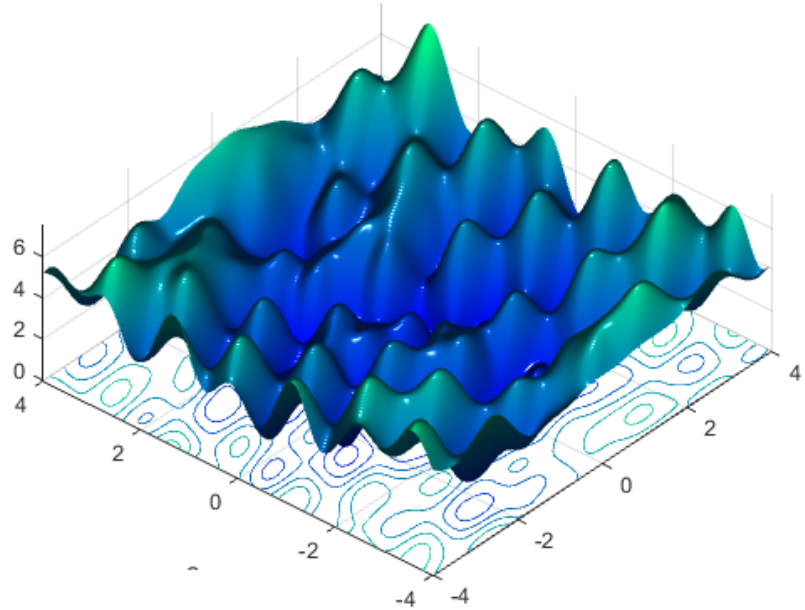
Adjacency Matrix:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$



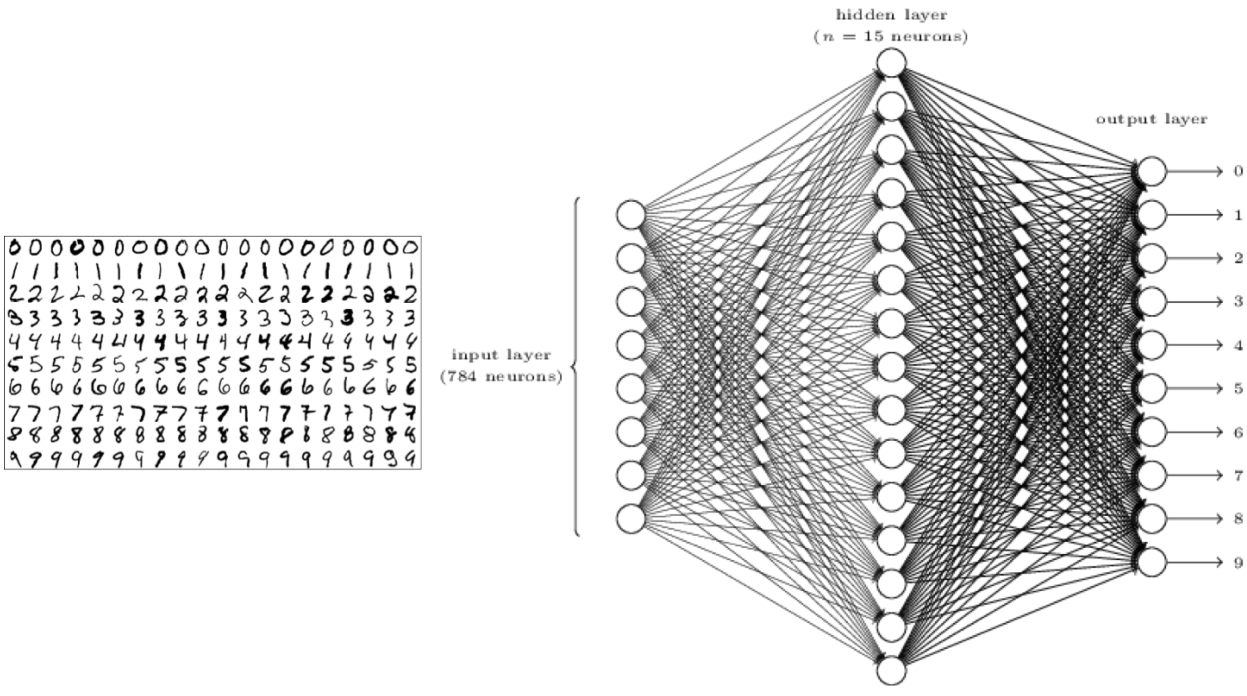
Non-convex Optimization Problems

- In general, very difficult to solve globally
- Need to make further assumptions



Non-convex Optimization Problems

$$\min_x \sum_{i=1}^{n} (f_x(a_i) - y_i)^2$$



Non-convex Optimization Problems

$$\min_x \sum_{i=1}^{n} (f_x(a_i) - y_i)^2$$

$\rightarrow$ Heuristic: Gauss-Newton method

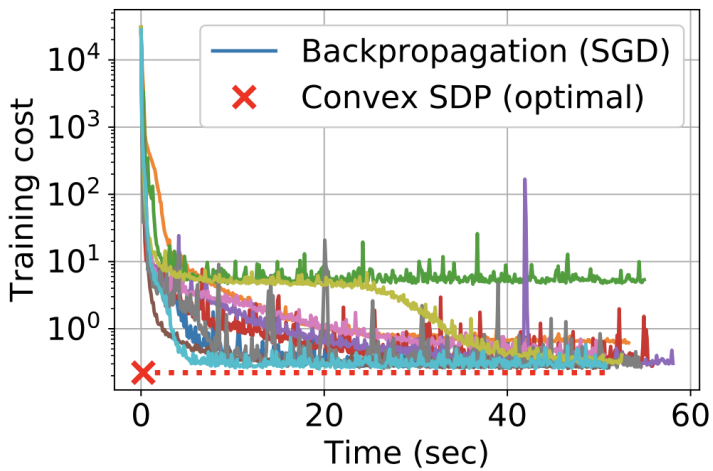
$$x_{t+1} = \arg\min_x \| \underbrace{f_{x_t}(A) + J_t x}_{\text{Taylor's approx for } f_x} - y \|_2^2$$

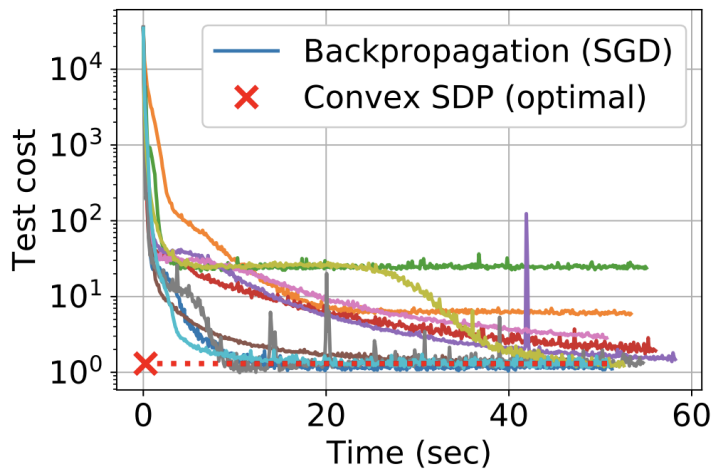
where $(J_t)_{ij} = \frac{\partial}{\partial x_j} f_x(a_i)$ is the Jacobian matrix

- Jacobian can be sampled for faster computations

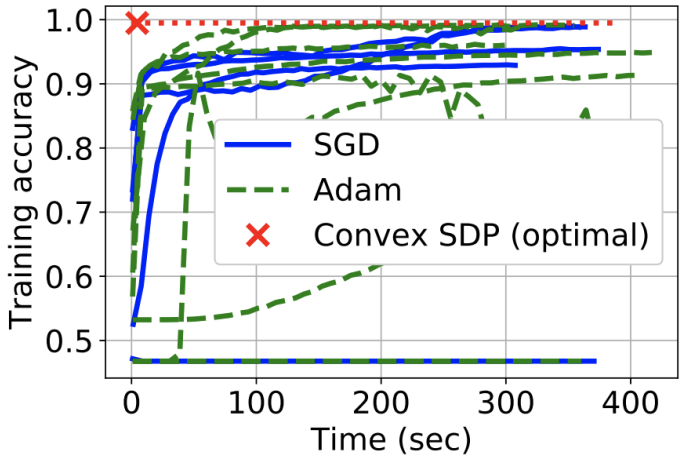
Convex Optimization for Neural Networks

- Neural Network problems can be formulated as convex optimization problems in higher dimensions by *lifting*
- enables global optimization in polynomial time

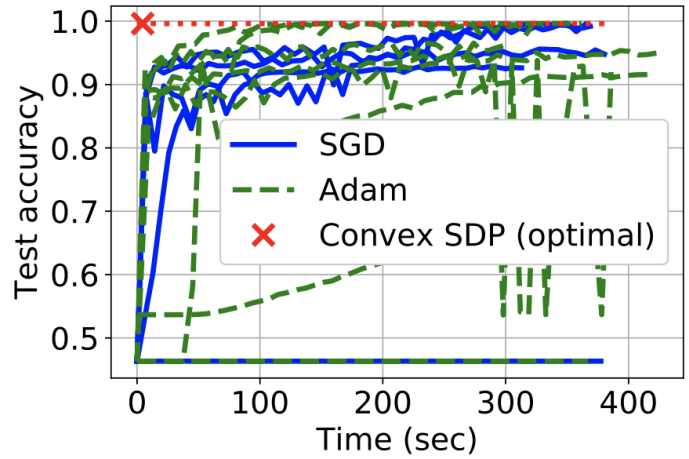




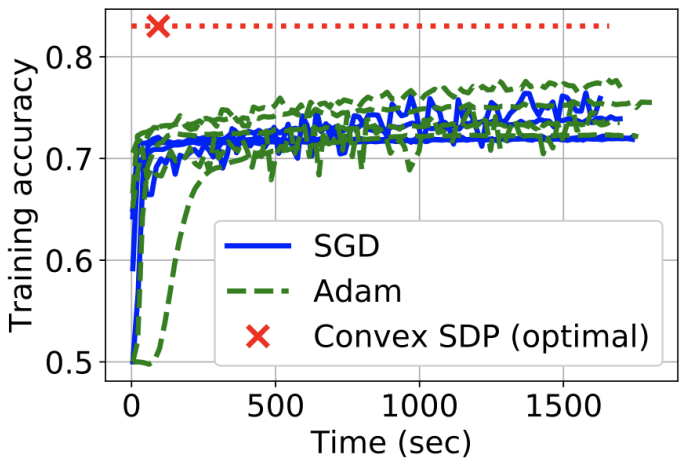
Convex Optimization for Neural Networks

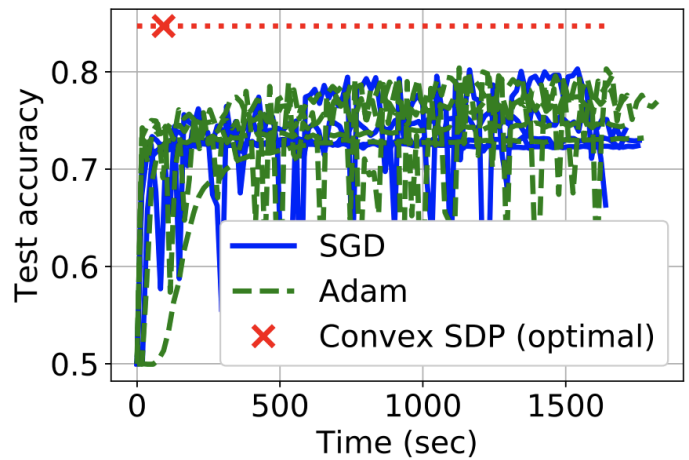


(a) CNN, MNIST, training accuracy



(b) CNN, MNIST, test accuracy





Questions?