Lecture 10

Nonlinear Optimization

CS328 - Numerical Methods for Visual Computing and Machine Learning

Prof. Wenzel Jakob

Today's topics

- **First-order methods**
  - Gradient & stochastic gradient descent (SGD)
  - Momentum
  - Adaptive gradients (RMSProp, Adam)

Crazy activity here in the last few years

- **Second-order methods**
  - Newton's method for optimization
  - Quasi-newton methods:
    - Gauss-Newton, BFGS

Classic optimization topic

Minimization

$$\mathbf{x}^* = \arg\min f(\mathbf{x})$$

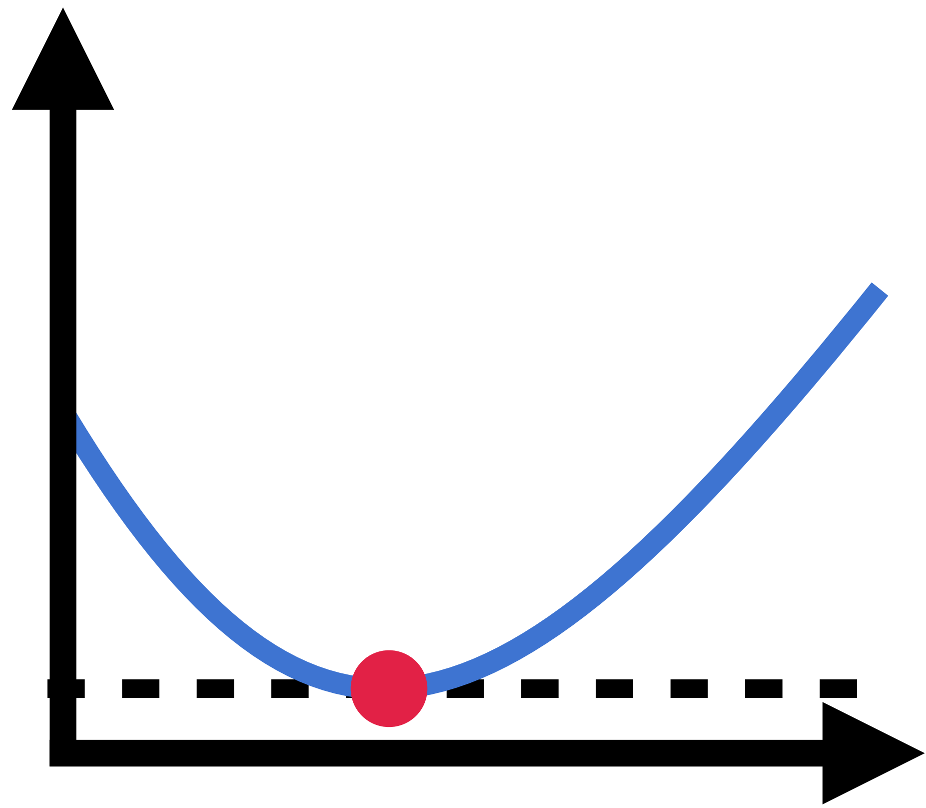
Covered in CS328

- This is the *unconstrained* case because we do not require the solution to lie on the boundary or interior of a set. Extra steps need to be taken to solve such *constrained* optimization tasks:

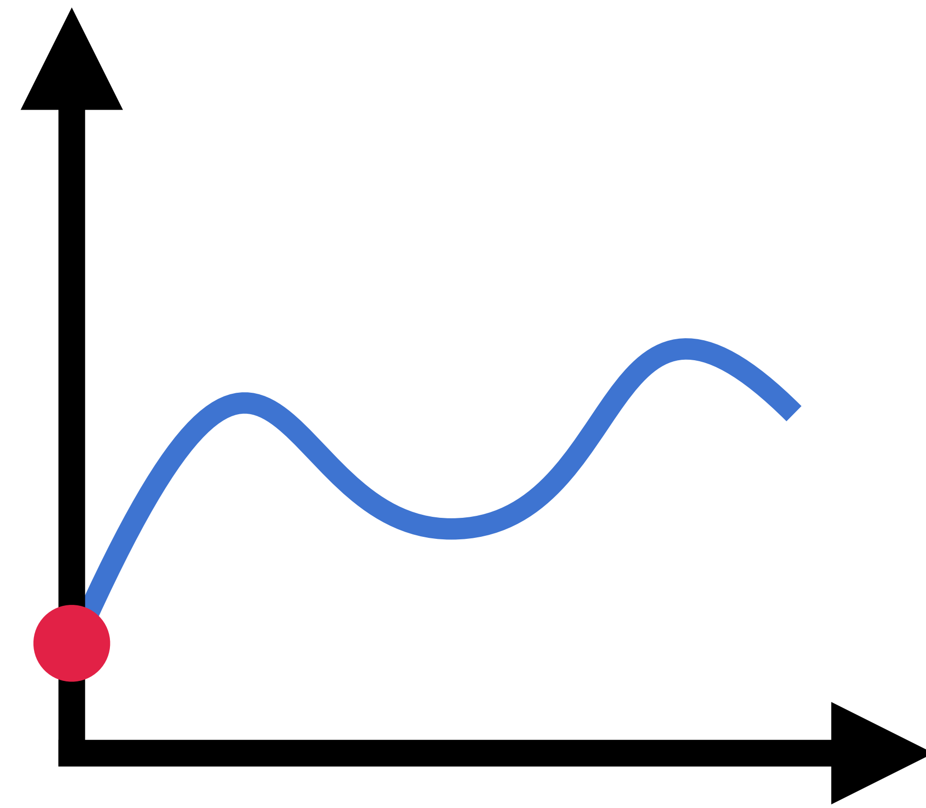
$$\mathbf{x}^* = \arg\min f(\mathbf{x})$$

$$\text{where } \; h(\mathbf{x}) = 0, \quad g(\mathbf{x}) > 0$$

Minimization via root finding



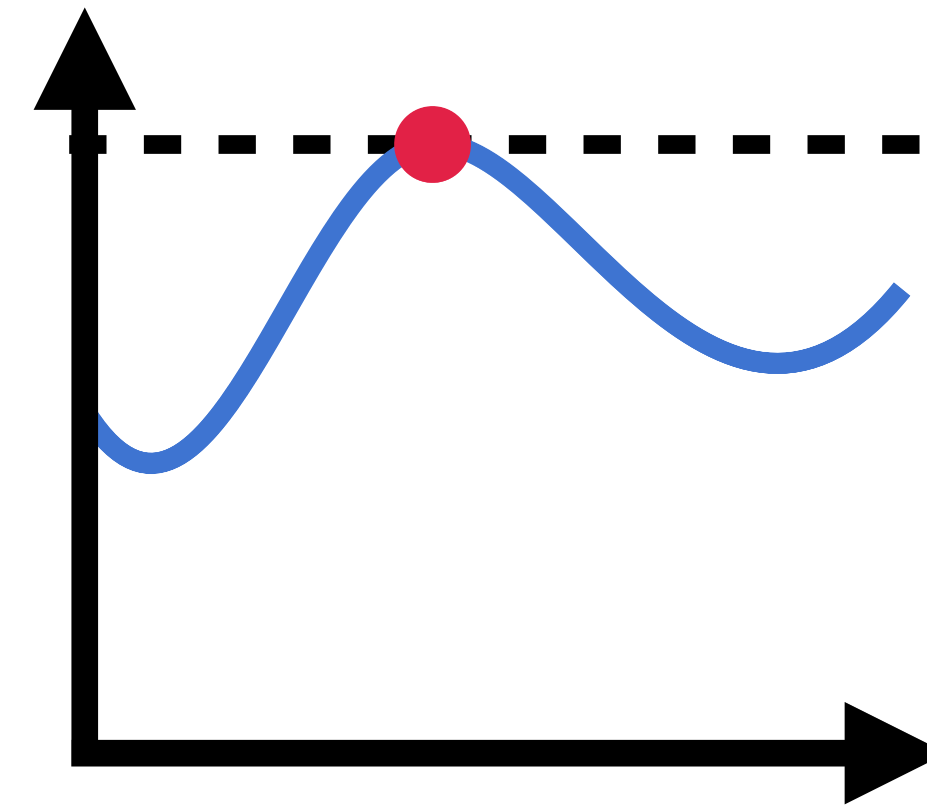
Derivative at interior minima is zero.



Minimum could be at domain boundaries.

This case arises in constrained optimization.

(methods covered today do not handle this case)



Points with zero derivative could also indicate maxima...

(not interested in such solutions, hopefully we won't find any of this type)

General approach

- Problem cannot be solved all at once.
  - Instead, algorithm repeatedly performs *steps* of the form $\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta_i$
  - Δ_i computed using *local* information: value, derivatives, 2nd derivatives, etc.
- Main questions: in *what direction* and *how far* should we step?
- Should each step be completely independent?
  .. or can it be influenced by information from prior iterations?

- No guarantees in general: optimizer can get stuck, explode, etc.

Optimization via root finding

Bump all quantities by 1 derivative order..

- Newton's method for *root finding* (1D):

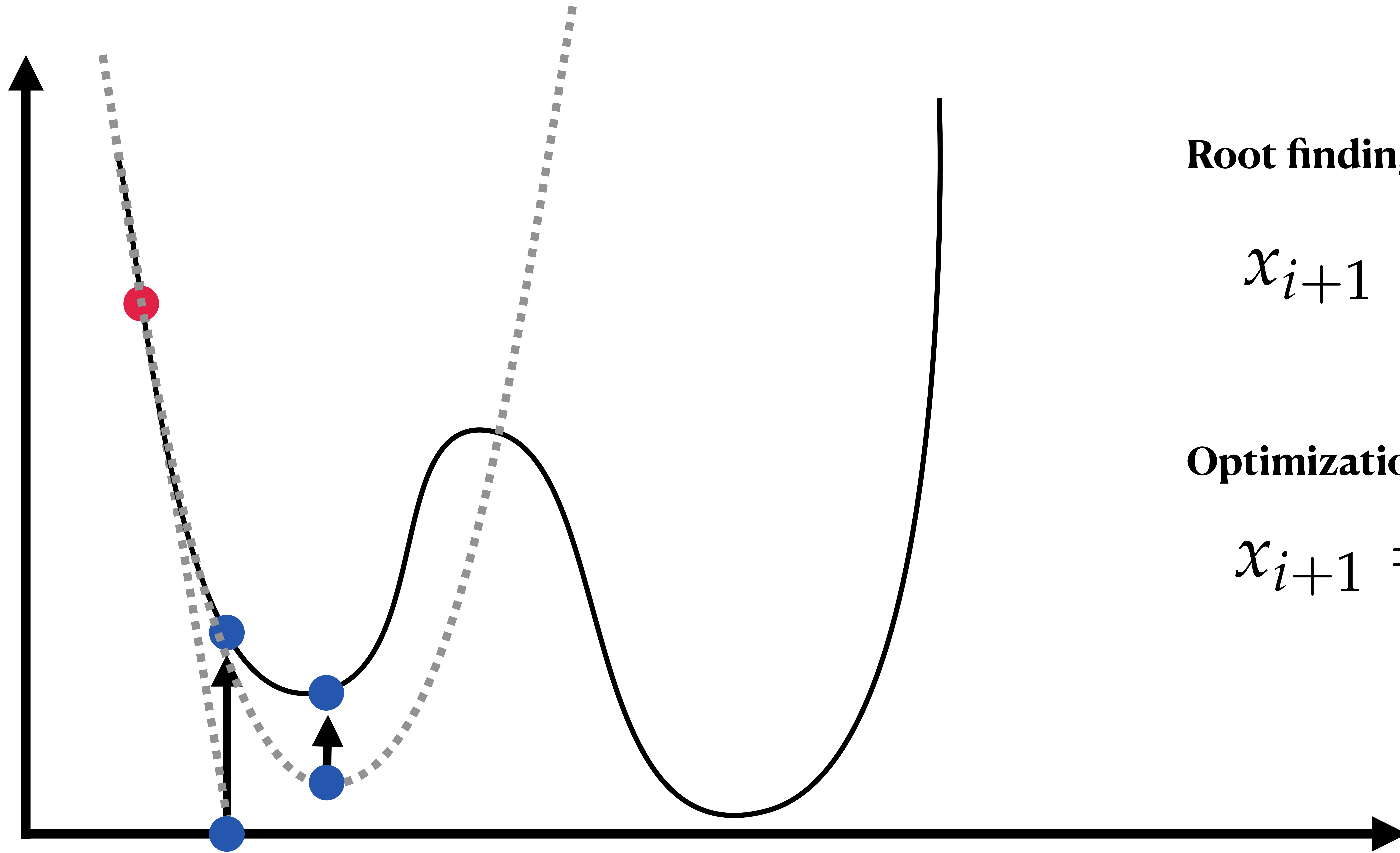
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

- Newton's method for *optimization* (1D):

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

Finds a root of the derivative
(Hopefully, a local minimum)

Newton's method in 1D



Root finding:

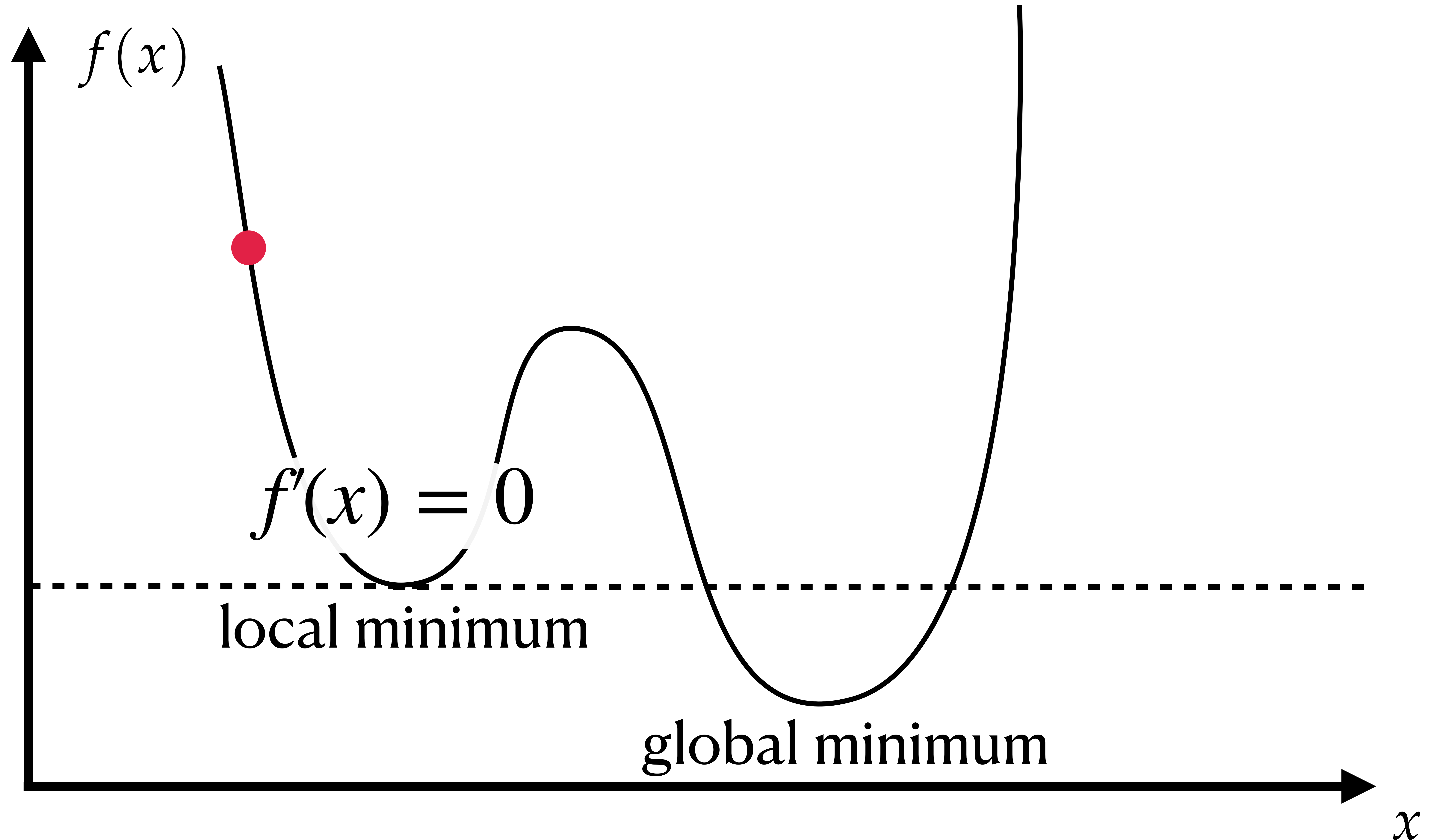
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Optimization:

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

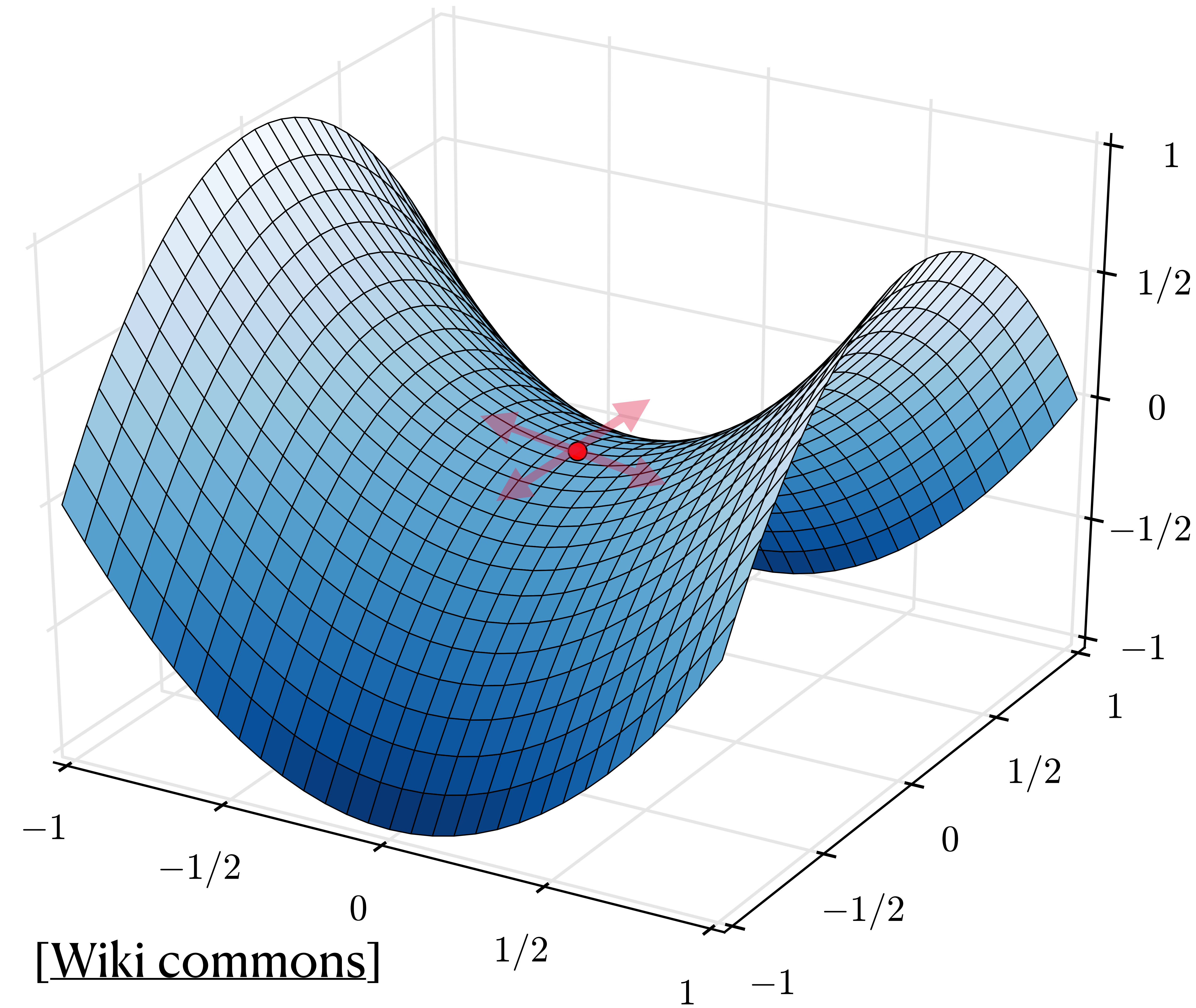
Optimization challenges

Global vs local minima



Optimization challenges

Saddle points



[Wiki commons]

- **Saddle point**: a local minimum in one direction, a local maximum in the other.
  - Cannot happen in 1D
  - Becomes possible in 2D
  - Far more likely in higher dimensions 😱 ($2^n - 1$ saddle configurations in n dim.)

- Can often escape saddle points using *momentum* and/or *noise*.

Gradient descent

a.k.a. steepest descent

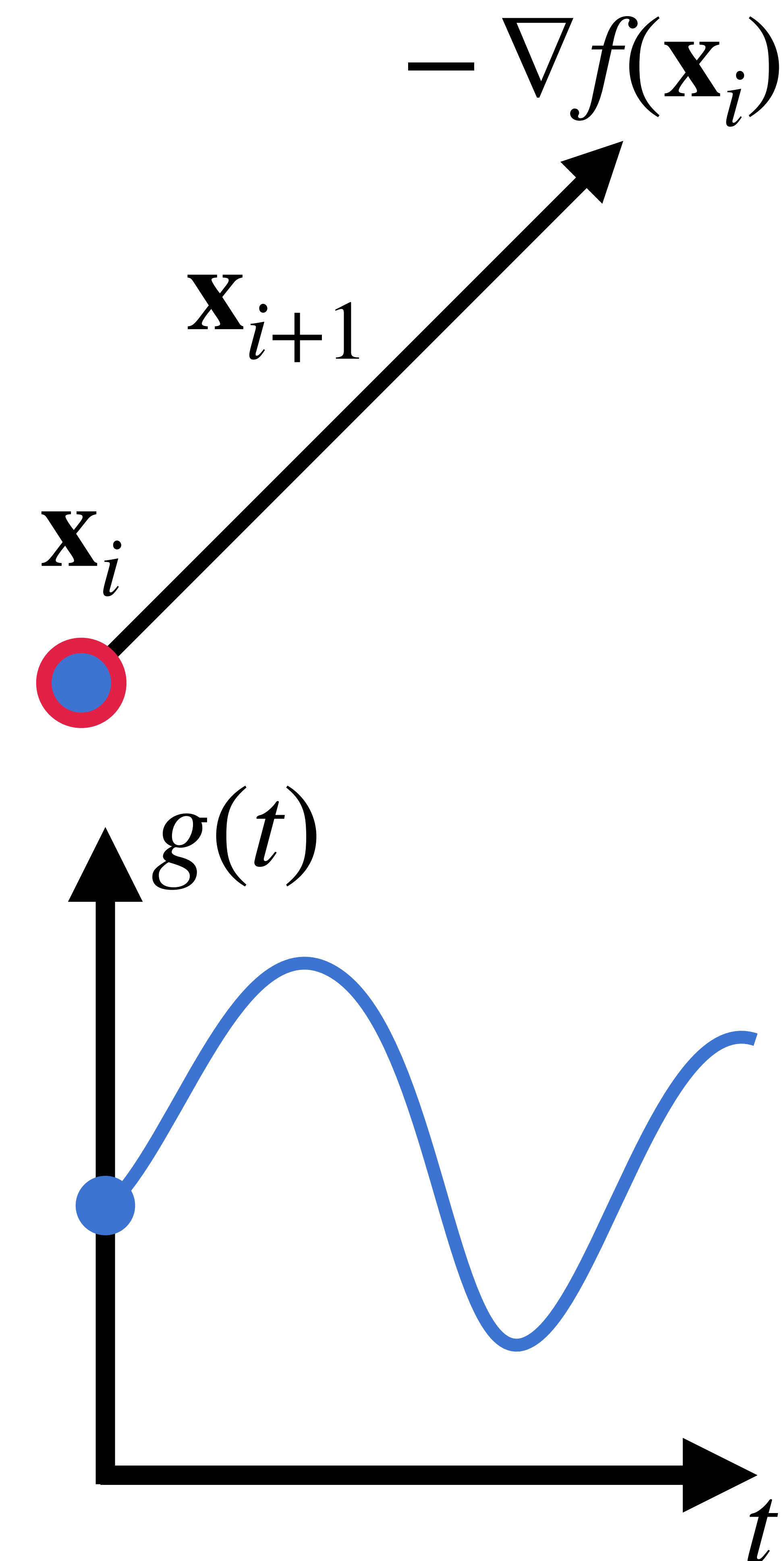
$$\mathbf{x}_{i+1} = \mathbf{x}_i - \eta \nabla f(\mathbf{x}_i)$$

- η is the *step size* (a fixed constant in this simple case).
- This is a *hyper*-parameter: a parameter of a method that itself optimizes parameters.
- Other (*implicit*) hyper-parameters:
  - number of descent steps
  - algorithm used to select the initial state $\mathbf{x}_1$.

Problems with Vanilla Gradient Descent (i)

- Gradient provides the *direction*. But how *far* should we step? Fixed choice of η can be both fragile and problem-dependent.
  - For example, suppose we set $\hat{f}(\mathbf{x}) := 1000 \cdot f(\mathbf{x})$. Then $\hat{f}(\mathbf{x})$ has the *same minima* as $f(\mathbf{x})$. But gradient descent on this function will take steps that are 1000 times larger!
  - **Line search**: define $g(t) := f(\mathbf{x} - t\nabla f(\mathbf{x}))$. Find the value of t that minimizes $g(t)$, which yields the *optimal* step size. This is a simple 1D optimization that can be handled using many different algorithms.

  (Not used much in ML. Does not play well with stochastic variants discussed next.)



Problems with Vanilla Gradient Descent (ii)

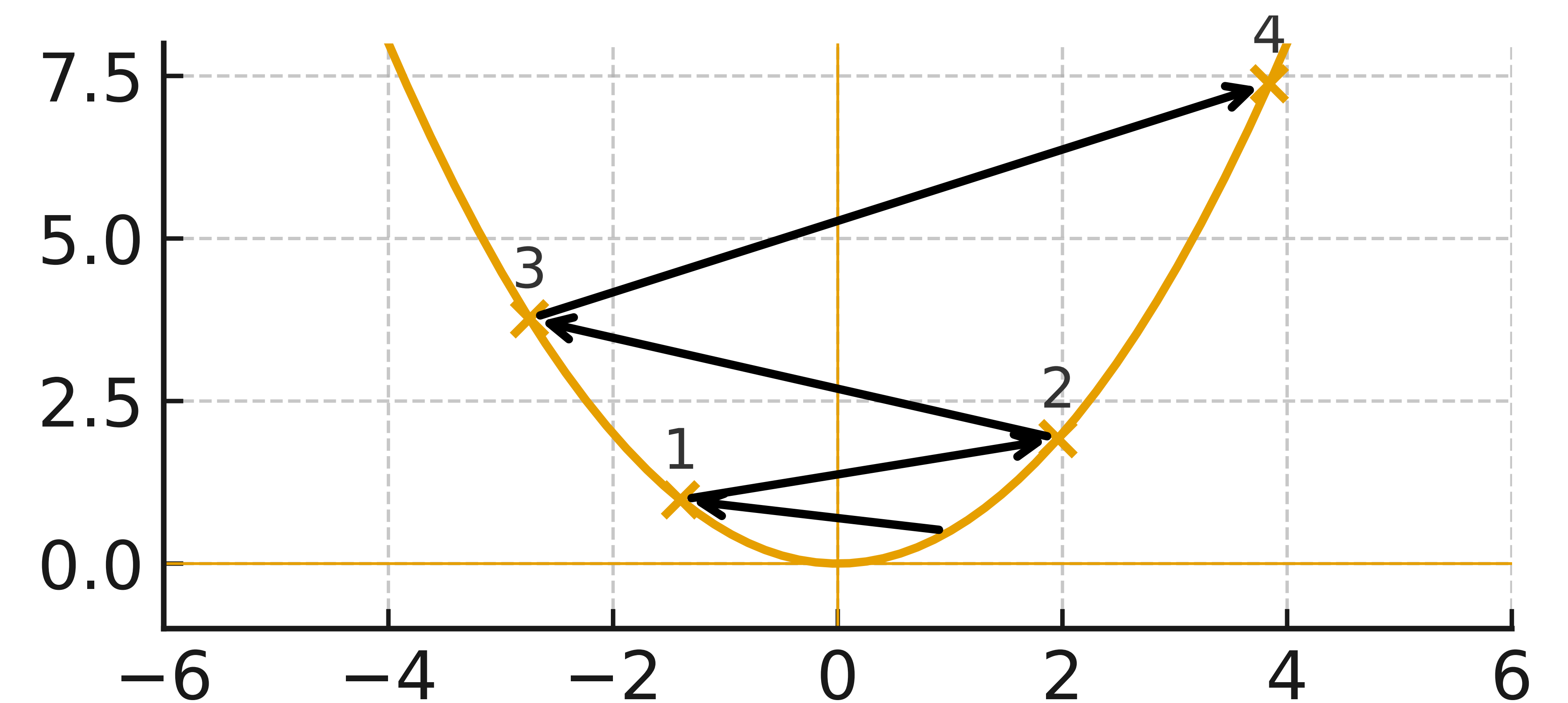
- Non-uniform convergence:
  - Y component immediately converges and then over/under-shoots.
  - X component converges very slowly.
  - (This is an axis-aligned example. This kind of issue can also happen *diagonally*.)
- Gradient isn't doing a good job at pointing towards the minimum.
- Hessian (matrix of *second derivatives*) describes the shape of this ellipse. Things get tricky when it has a large condition number.

Problems with Vanilla Gradient Descent (iii)

Divergence..

- Let's optimize a simple quadratic $f(x) := \frac{1}{2}cx^2$. Then $f'(x) = cx$
- At each iteration, gradient descent computes

$$x_{k+1} = x_k - \eta f'(x_k) = (1 - c\eta)\, x_k$$



- If $(1 - c\eta) > 1$, then this will explode 💥
- Solving for the step size, this means that convergence requires $\lambda < \frac{2}{c}$.

Problems with Vanilla Gradient Descent (iv)

- What if f is a sum of n terms? If n is very large, it may be **expensive** to evaluate f & ∇f.

$$f(\mathbf{x}) = \frac{1}{n}\sum_{k=1}^{n} f_k(\mathbf{x})$$

- Solution: pick one k' at random (with prob. $1/n$). Then $\nabla f_{k'}$ has the expected value

prob. of event

$$E[\nabla f_{k'}(\mathbf{x})] = \sum_{k=1}^{n} \frac{1}{n} \nabla f_k(\mathbf{x}) = \nabla f(\mathbf{x})$$

sum over all possible outcomes

which equals the (costly) full gradient. In other words, it is an *unbiased estimator of* ∇f.

Stochastic gradient descent (SGD)

- SGD is a *Monte Carlo* method. We will see many more examples in the next lecture.
- **Motivation**: SGD is approximate but it takes the right step *in expectation ("on average")*.
- Noise actually helps training.
- In typical ML applications, n is the number of training examples (*large!*). Picking just a single k', while unbiased, gives an *extremely bad* gradient approximation.

- "**Mini-batch**" SGD:
  - pick a subset of m (where $m << n$) samples from the training dataset.
  - take gradient descent steps using the average gradient from this batch.
  - Batching greatly improves utilization on GPUs

Gradient descent with momentum

Previous methods did not use any information from the past. Let's fix that.

Velocity → ... ← Typ. values: 0.9 or 0.99. Smaller values add more "friction".

$$\mathbf{v}_{i+1} = \rho \mathbf{v}_i - \nabla f(\mathbf{x}_i)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \eta \mathbf{v}_{i+1}$$

- **Motivation** (also works with stochastic variant):
  - If one parameter has a very small derivative, but it is consistently positive or negative, GD with momentum will become progressively faster.
  - Parameters, whose derivative sign oscillates back and forth do not build up a large velocity.

MidJourney: a ball rolling down a mountain

Exponentially moving averages

Taking a closer look at the velocity update in GD with momentum

$$
\begin{aligned}
\mathbf{v}_1 &= 0 \\
\mathbf{v}_2 &= -\Delta f(\mathbf{x}_1) \\
\mathbf{v}_3 &= -\rho\Delta f(\mathbf{x}_1) - \Delta f(\mathbf{x}_2) \\
\mathbf{v}_4 &= -\rho^2\Delta f(\mathbf{x}_1) - \rho\Delta f(\mathbf{x}_2) - \Delta f(\mathbf{x}_3) \\
\mathbf{v}_5 &= -\rho^3\Delta f(\mathbf{x}_1) - \rho^2\Delta f(\mathbf{x}_2) - \rho\Delta f(\mathbf{x}_3) - \Delta f(\mathbf{x}_4)
\end{aligned}
$$

Average of past information that progressively "fades out" older data.

Exponentially moving averages (**EMAs**) play a major role in many gradient-based optimizers.

RMSProp

A side note in a Cousera course by Geoffrey Hinton. Chooses step size adaptively per parameter

$$\mathbf{g}_i = \nabla f(\mathbf{x}_{i-1})$$

$$\bar{\mathbf{G}}_i = \rho\bar{\mathbf{G}}_{i-1} + (1-\rho)\mathbf{g}_i^2$$

$$\Delta_i = -\frac{\eta}{\sqrt{\bar{\mathbf{G}}_i} + \varepsilon}\mathbf{g}_i$$

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \Delta_i$$

(Slight change in notation: we're now computing the state of iteration i from i-1).

A small positive value so that this fraction cannot explode

- **Motivation 1**: subsequent steps of opposite sign cancel in normal gradient descent. The variable $\bar{\mathbf{G}}_i$ provides an EMA of the *squared* gradient, where this cancellation doesn't happen. Division by the square root of $\bar{\mathbf{G}}_i$ reduces the step size when close to the solution.
- **Motivation 2**: Let's think about the the example $\hat{f}(\mathbf{x}) := 1000 \cdot f(\mathbf{x})$ from earlier.

(You do not need to memorize the update rules of this algorithm for the exam)

Adam ≈ Momentum + RMSProp

These ideas can be combined. Why not do both at once?

$$
\begin{aligned}
\mathbf{g}_i &= \nabla f(\mathbf{x}_{i-1}) \\
\bar{\mathbf{g}}_i &= \rho_1 \bar{\mathbf{g}}_{i-1} + (1-\rho_1)\,\mathbf{g}_i \\
\bar{\mathbf{G}}_i &= \rho_2 \bar{\mathbf{G}}_{i-1} + (1-\rho_2)\,\mathbf{g}_i^2 \\
\Delta_i &= -\frac{\eta}{\sqrt{\bar{\mathbf{G}}_i + \varepsilon}}\bar{\mathbf{g}}_i \\
\mathbf{x}_i &= \mathbf{x}_{i-1} + \Delta_i
\end{aligned}
$$

EMA of gradients

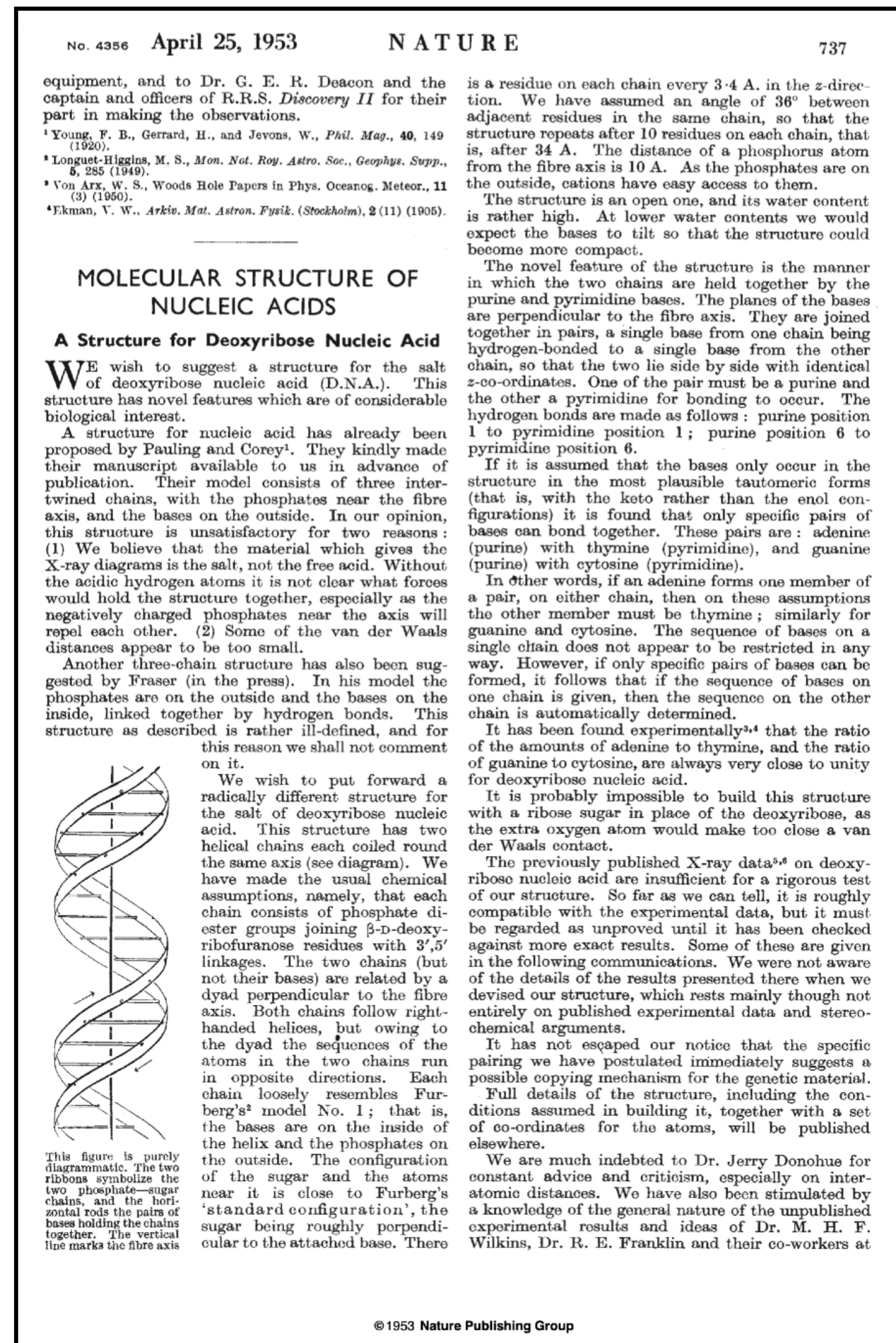
EMA of gradient squares

Step into *averaged* gradient direction (→ momentum), scale adaptively based on history of past (squared) gradients.

- **Default parameters:** $\rho_1 = 0.9, \rho_2 = 0.999$.
- **Some details not shown:** proper implementations scale $\bar{\mathbf{g}}_i$ and $\bar{\mathbf{G}}_i$ with a correction factor to overcome *startup bias* in the first few iterations.

(You do not need to memorize the update rules of this algorithm for the exam)

Adam is nowadays often the default



No. 4356 April 25, 1953 NATURE 737

equipment, and to Dr. G. E. R. Deacon and the captain and officers of R.R.S. *Discovery II* for their part in making the observations.

[1] Young, F. B., Gerrard, H., and Jevons, W., *Phil. Mag.*, **40**, 149 (1920).
[2] Longuet-Higgins, M. S., *Mon. Not. Roy. Astro. Soc., Geophys. Supp.*, **5**, 285 (1949).
[3] Von Arx, W. S., Woods Hole Papers in Phys. Oceanog. Meteor., **11** (3) (1950).
[4] Ekman, V. W., *Arkiv. Mat. Astron. Fysik. (Stockholm)*, **2** (11) (1905).

MOLECULAR STRUCTURE OF NUCLEIC ACIDS

A Structure for Deoxyribose Nucleic Acid

WE wish to suggest a structure for the salt of deoxyribose nucleic acid (D.N.A.). This structure has novel features which are of considerable biological interest.

A structure for nucleic acid has already been proposed by Pauling and Corey[1]. They kindly made their manuscript available to us in advance of publication. Their model consists of three intertwined chains, with the phosphates near the fibre axis, and the bases on the outside. In our opinion, this structure is unsatisfactory for two reasons: (1) We believe that the material which gives the X-ray diagrams is the salt, not the free acid. Without the acidic hydrogen atoms it is not clear what forces would hold the structure together, especially as the negatively charged phosphates near the axis will repel each other. (2) Some of the van der Waals distances appear to be too small.

Another three-chain structure has also been suggested by Fraser (in the press). In his model the phosphates are on the outside and the bases on the inside, linked together by hydrogen bonds. This structure as described is rather ill-defined, and for this reason we shall not comment on it.

We wish to put forward a radically different structure for the salt of deoxyribose nucleic acid. This structure has two helical chains each coiled round the same axis (see diagram). We have made the usual chemical assumptions, namely, that each chain consists of phosphate diester groups joining β-D-deoxyribofuranose residues with 3′,5′ linkages. The two chains (but not their bases) are related by a dyad perpendicular to the fibre axis. Both chains follow right-handed helices, but owing to the dyad the sequences of the atoms in the two chains run in opposite directions. Each chain loosely resembles Furberg's[2] model No. 1; that is, the bases are on the inside of the helix and the phosphates on the outside. The configuration of the sugar and the atoms near it is close to Furberg's 'standard configuration', the sugar being roughly perpendicular to the attached base. There is a residue on each chain every 3·4 A. in the z-direction. We have assumed an angle of 36° between adjacent residues in the same chain, so that the structure repeats after 10 residues on each chain, that is, after 34 A. The distance of a phosphorus atom from the fibre axis is 10 A. As the phosphates are on the outside, cations have easy access to them.

This figure is purely diagrammatic. The two ribbons symbolize the two phosphate—sugar chains, and the horizontal rods the pairs of bases holding the chains together. The vertical line marks the fibre axis

The structure is an open one, and its water content is rather high. At lower water contents we would expect the bases to tilt so that the structure could become more compact.

The novel feature of the structure is the manner in which the two chains are held together by the purine and pyrimidine bases. The planes of the bases are perpendicular to the fibre axis. They are joined together in pairs, a single base from one chain being hydrogen-bonded to a single base from the other chain, so that the two lie side by side with identical z-co-ordinates. One of the pair must be a purine and the other a pyrimidine for bonding to occur. The hydrogen bonds are made as follows: purine position 1 to pyrimidine position 1; purine position 6 to pyrimidine position 6.

If it is assumed that the bases only occur in the structure in the most plausible tautomeric forms (that is, with the keto rather than the enol configurations) it is found that only specific pairs of bases can bond together. These pairs are: adenine (purine) with thymine (pyrimidine), and guanine (purine) with cytosine (pyrimidine).

In other words, if an adenine forms one member of a pair, on either chain, then on these assumptions the other member must be thymine; similarly for guanine and cytosine. The sequence of bases on a single chain does not appear to be restricted in any way. However, if only specific pairs of bases can be formed, it follows that if the sequence of bases on one chain is given, then the sequence on the other chain is automatically determined.

It has been found experimentally[3,4] that the ratio of the amounts of adenine to thymine, and the ratio of guanine to cytosine, are always very close to unity for deoxyribose nucleic acid.

It is probably impossible to build this structure with a ribose sugar in place of the deoxyribose, as the extra oxygen atom would make too close a van der Waals contact.

The previously published X-ray data[5,6] on deoxyribose nucleic acid are insufficient for a rigorous test of our structure. So far as we can tell, it is roughly compatible with the experimental data, but it must be regarded as unproved until it has been checked against more exact results. Some of these are given in the following communications. We were not aware of the details of the results presented there when we devised our structure, which rests mainly though not entirely on published experimental data and stereochemical arguments.

It has not escaped our notice that the specific pairing we have postulated immediately suggests a possible copying mechanism for the genetic material.

Full details of the structure, including the conditions assumed in building it, together with a set of co-ordinates for the atoms, will be published elsewhere.

We are much indebted to Dr. Jerry Donohue for constant advice and criticism, especially on interatomic distances. We have also been stimulated by a knowledge of the general nature of the unpublished experimental results and ideas of Dr. M. H. F. Wilkins, Dr. R. E. Franklin and their co-workers at

©1953 Nature Publishing Group

Published as a conference paper at ICLR 2015

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma* University of Amsterdam, OpenAI dpkingma@openai.com
Jimmy Lei Ba* University of Toronto jimmy@psi.utoronto.ca

ABSTRACT

We introduce *Adam*, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Some connections to related algorithms, on which *Adam* was inspired, are discussed. We also analyze the theoretical convergence properties of the algorithm and provide a regret bound on the convergence rate that is comparable to the best known results under the online convex optimization framework. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods. Finally, we discuss *AdaMax*, a variant of *Adam* based on the infinity norm.

1 INTRODUCTION

Stochastic gradient-based optimization is of core practical importance in many fields of science and engineering. Many problems in these fields can be cast as the optimization of some scalar parameterized objective function requiring maximization or minimization with respect to its parameters. If the function is differentiable w.r.t. its parameters, gradient descent is a relatively efficient optimization method, since the computation of first-order partial derivatives w.r.t. all the parameters is of the same computational complexity as just evaluating the function. Often, objective functions are stochastic. For example, many objective functions are composed of a sum of subfunctions evaluated at different subsamples of data; in this case optimization can be made more efficient by taking gradient steps w.r.t. individual subfunctions, i.e. stochastic gradient descent (SGD) or ascent. SGD proved itself as an efficient and effective optimization method that was central in many machine learning success stories, such as recent advances in deep learning (Deng et al., 2013; Krizhevsky et al., 2012; Hinton & Salakhutdinov, 2006; Hinton et al., 2012a; Graves et al., 2013). Objectives may also have other sources of noise than data subsampling, such as dropout (Hinton et al., 2012b) regularization. For all such noisy objectives, efficient stochastic optimization techniques are required. The focus of this paper is on the optimization of stochastic objectives with high-dimensional parameters spaces. In these cases, higher-order optimization methods are ill-suited, and discussion in this paper will be restricted to first-order methods.

We propose *Adam*, a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients; the name *Adam* is derived from adaptive moment estimation. Our method is designed to combine the advantages of two recently popular methods: AdaGrad (Duchi et al., 2011), which works well with sparse gradients, and RMSProp (Tieleman & Hinton, 2012), which works well in on-line and non-stationary settings; important connections to these and other stochastic optimization methods are clarified in section 5. Some of Adam's advantages are that the magnitudes of parameter updates are invariant to rescaling of the gradient, its stepsizes are approximately bounded by the stepsize hyperparameter, it does not require a stationary objective, it works with sparse gradients, and it naturally performs a form of step size annealing.

*Equal contribution. Author ordering determined by coin flip over a Google Hangout.

1

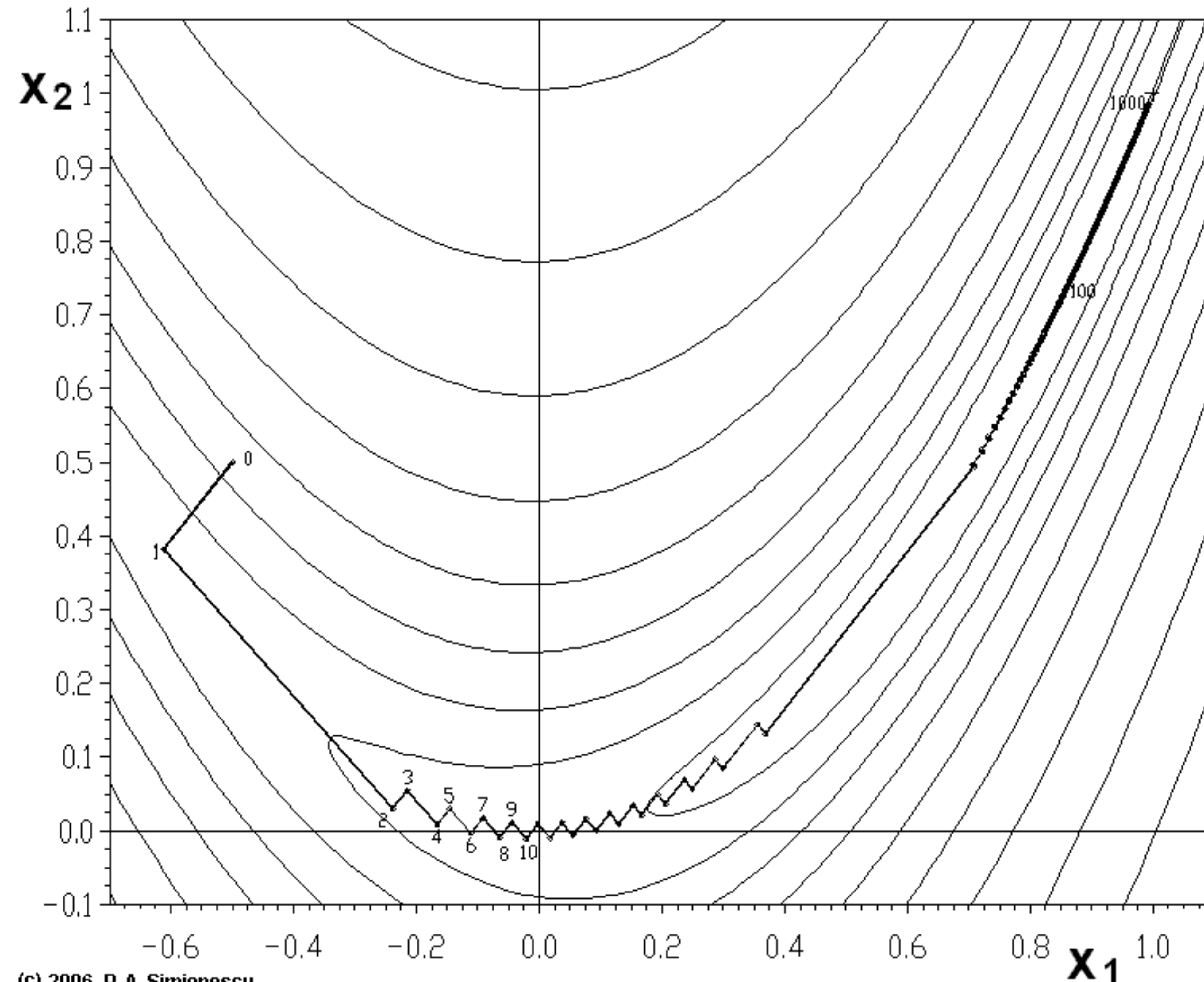
Molecular Structure of Nucleic Acids [..]
Watson & Crick
20 817 citations

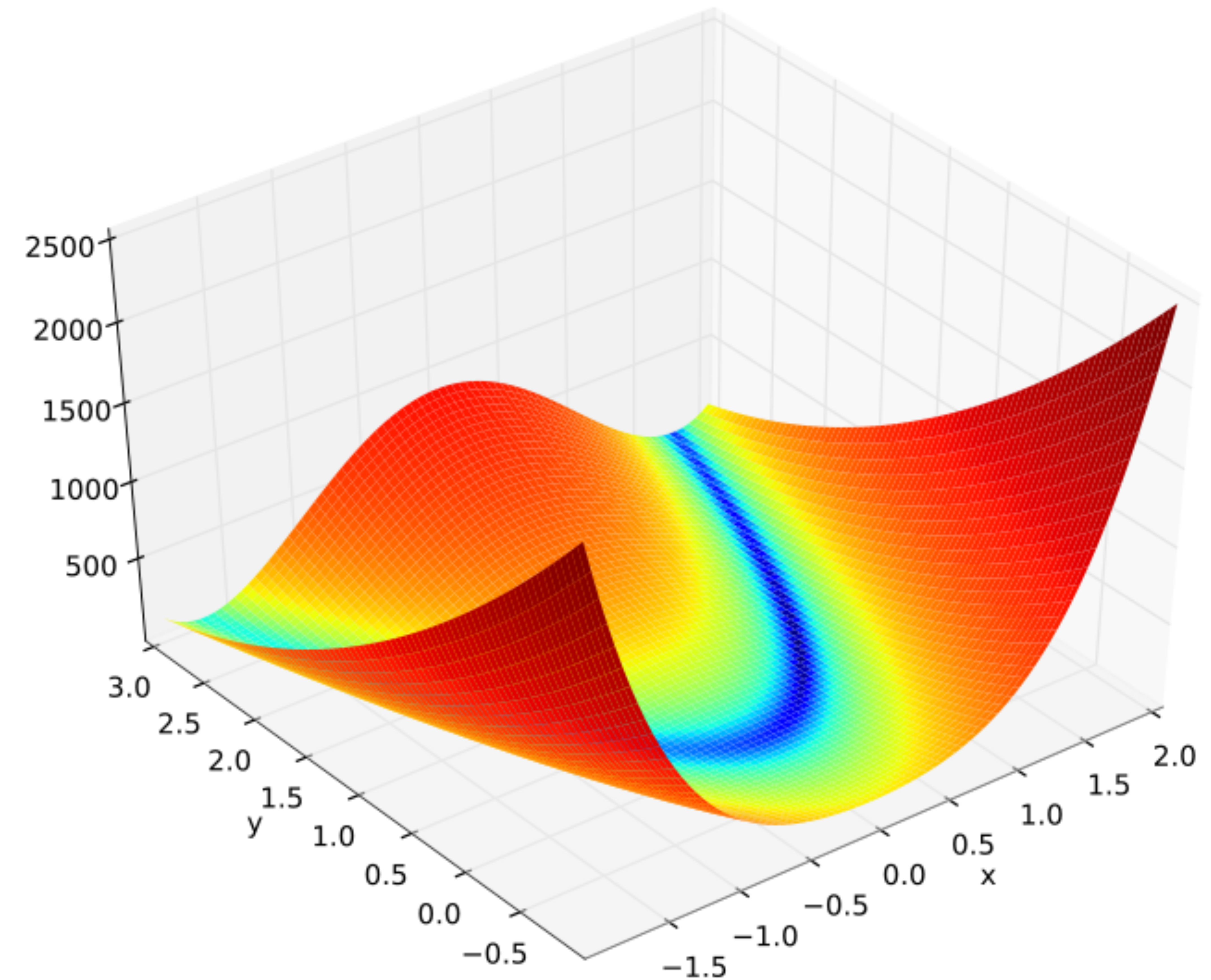
Adam: A Method for Stochastic Optimization
Kingma & Ba
230 641 citations 🤯

Benchmark: ellipses & the Rosenbrock function

aka. the *Banana* function

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$





Demo time

A zoo of different optimizers

The minimize function supports the following methods:

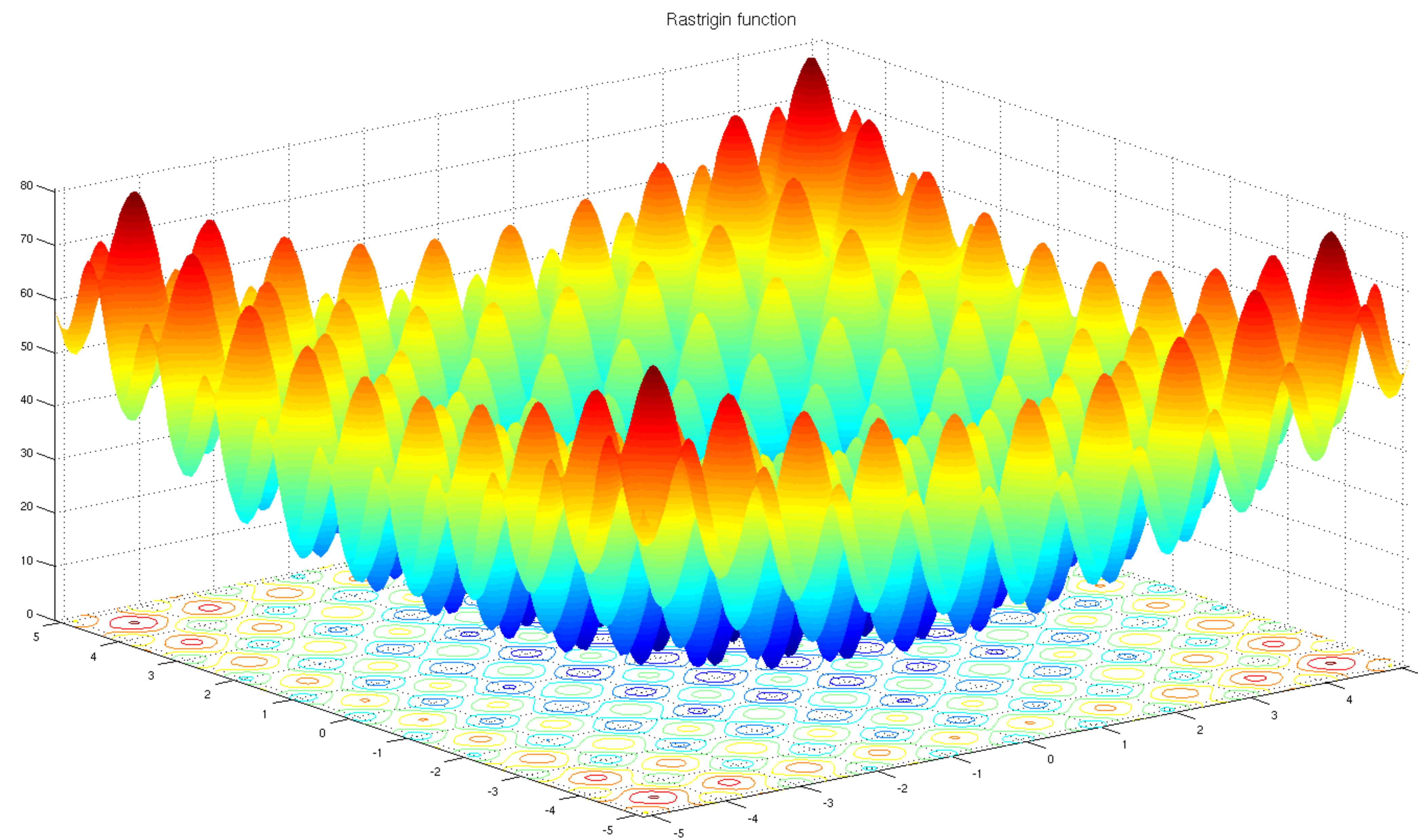
- minimize(method='Nelder-Mead')
- minimize(method='Powell')
- minimize(method='**CG**')
- minimize(method='**BFGS**')
- minimize(method='**Newton-CG**')
- minimize(method='**L-BFGS-B**')
- minimize(method='TNC')
- minimize(method='COBYLA')
- minimize(method='SLSQP')
- minimize(method='dogleg')
- minimize(method='trust-ncg')
- minimize(method='trust-krylov')
- minimize(method='trust-exact')

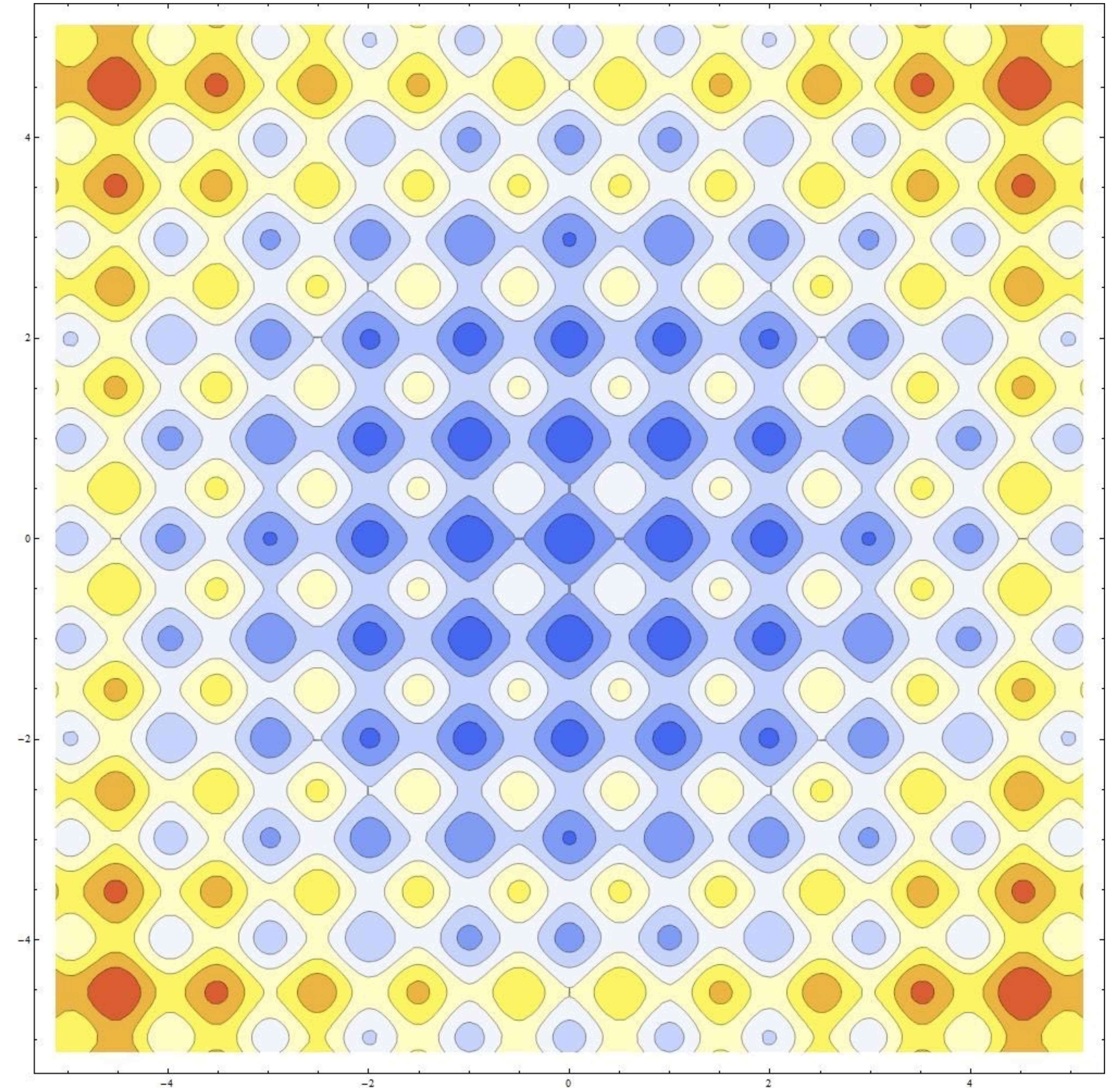
The minimize_scalar function supports the following methods:

- minimize_scalar(method='**brent**')
- minimize_scalar(method='bounded')
- minimize_scalar(method='**golden**')

A harder benchmark?

The Rastrigin function





Second-order methods

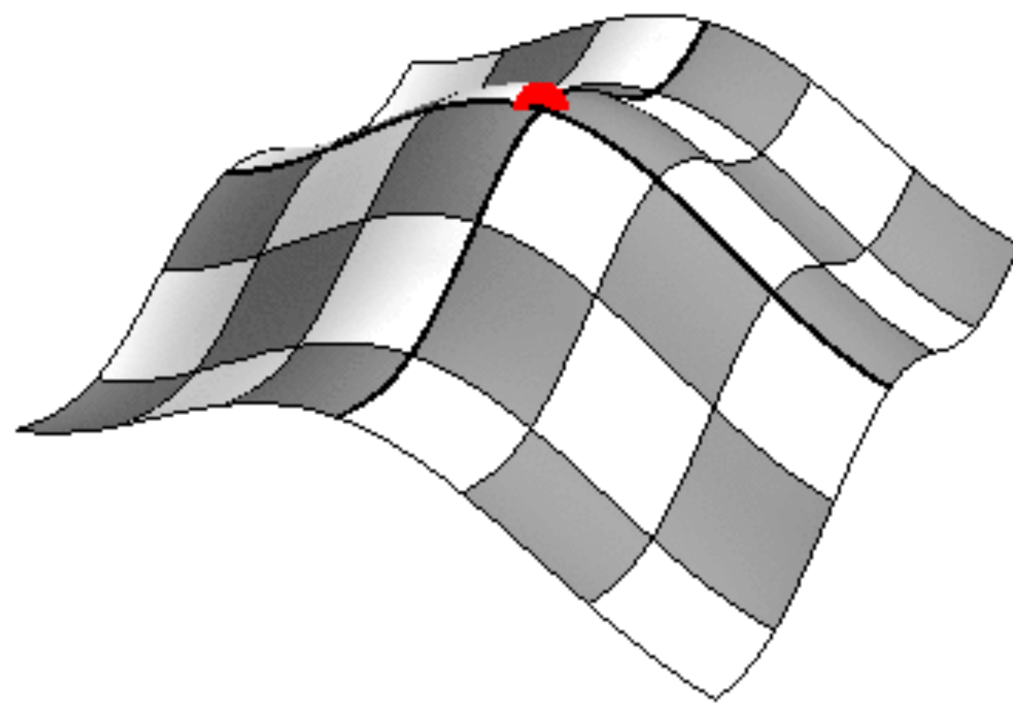
Based on a second-order Taylor approximation of the objective function.

$$f(\mathbf{x}+\mathbf{h}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot \mathbf{h} + \frac{1}{2}\mathbf{h}^T \mathbf{H}_f(\mathbf{x})\mathbf{h}$$

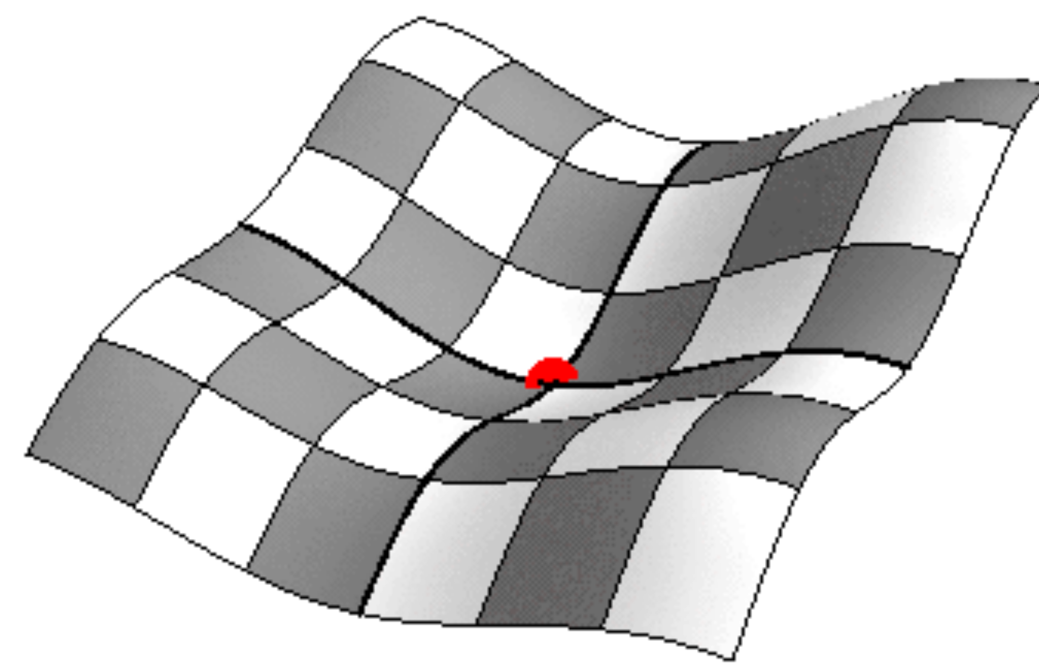
Gradient

Hessian

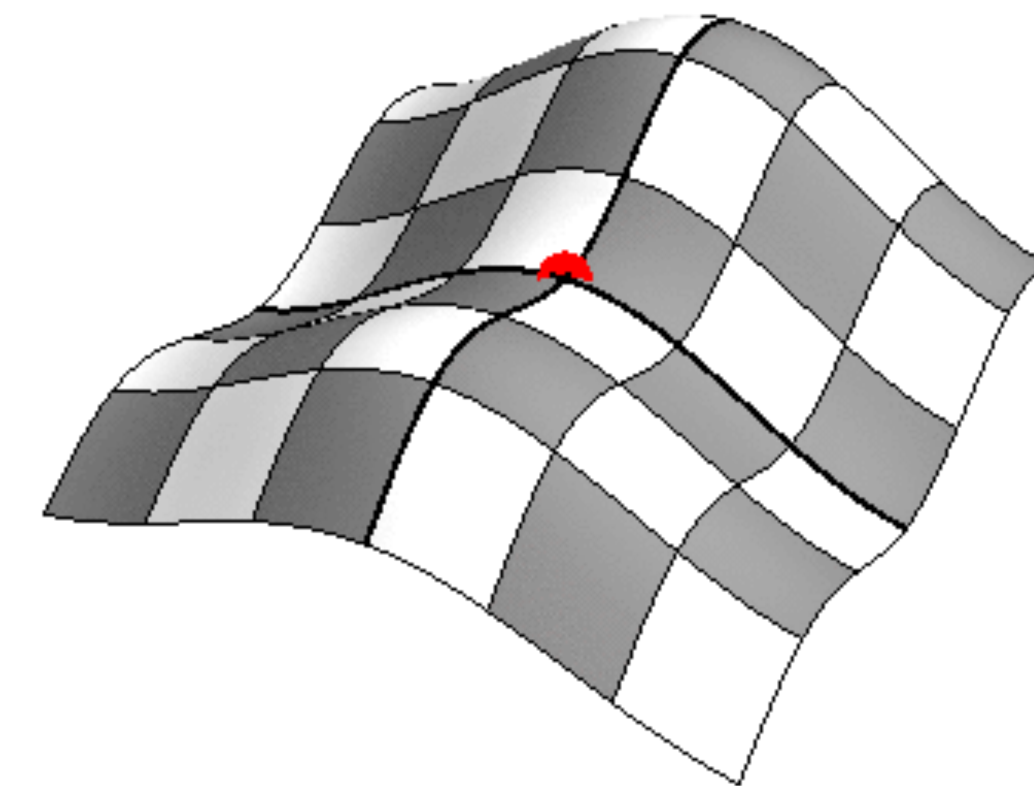
[Wikipedia]



Local maximum



Local minimum



Saddle point

Newton's method for optimization

Like in the 1D example from earlier, we just increase the order of differentiation by 1.

Newton's method for root finding

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{J}_f^{-1}(\mathbf{x}_i)\mathbf{f}(\mathbf{x}_i)$$

Finds the root ($\mathbf{f}(\mathbf{x}) = \mathbf{0}$) of a vector-valued function $\mathbf{f}$.

Newton's method for optimization

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{H}_f^{-1}(\mathbf{x}_i)\nabla f(\mathbf{x}_i)$$

Finds the root ($\nabla f(\mathbf{x}) = \mathbf{0}$) of the gradient of a scalar function f).

Intuition about second-order methods

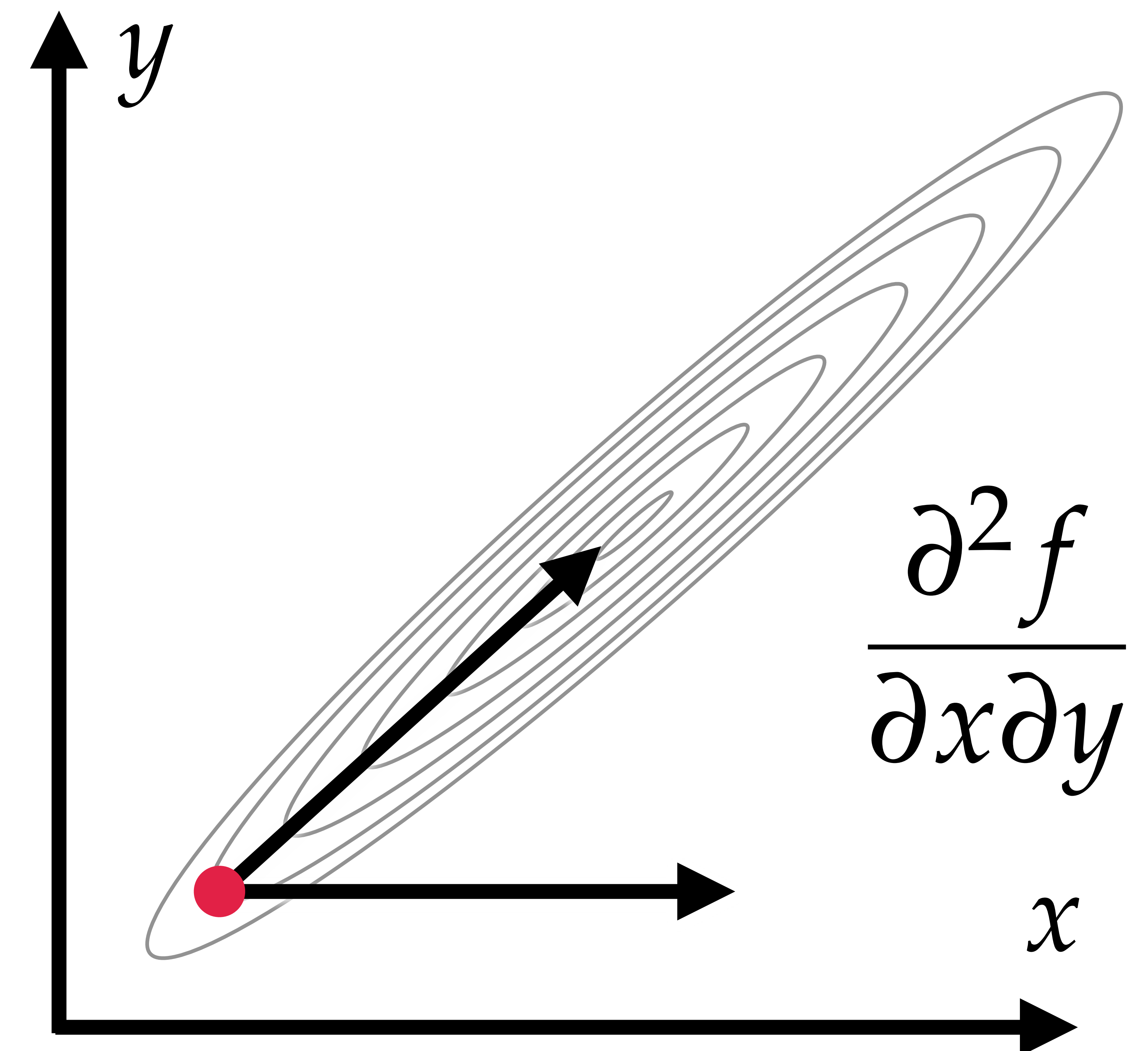
Everything you know about Newton's method applies here as well.

- **Step size**: Newton's method simply does not have this parameter. It knows where to go, and how far. *Whether those choices are ultimately good is another question.*

- **Second derivatives**, curvature, and ellipses..
  How do they relate?

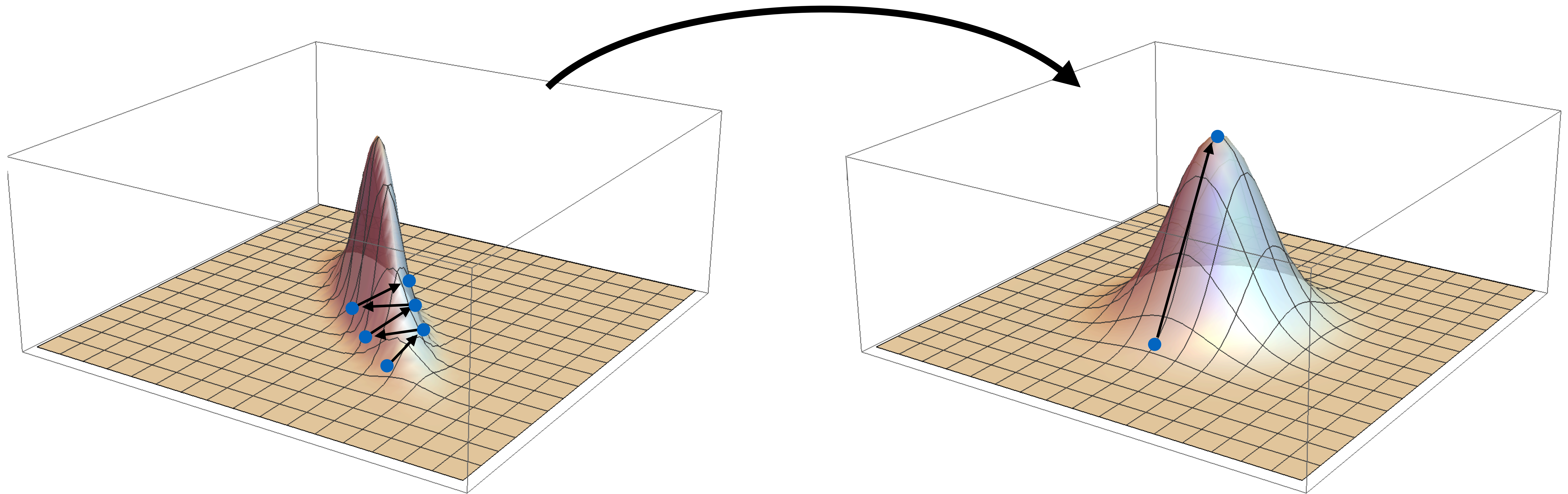
- **Computational cost**. AD can compute entries of the Hessian, but getting all of them is extremely costly when the number of dimensions is large.

  (Also, need to solve a different linear systems per step)



What solving a linear system accomplishes

The Hessian encodes local ellipsoid approximation we saw earlier.
Multiplying by its inverse "unwarps" the space



(To be consistent with previous figures, please imagine these plots were flipped vertically, so that this is a minimization problem instead of a maximization problem)

Quasi-Newton methods

- A true Newton step would be ideal. But it is too expensive.
- Let's compromise..
- **Idea (v1.0)**: Using gradients, compute an approximation of the Hessian!
- **Idea (v2.0)**: Using gradients, compute an approximation of the inverse Hessian!

ChatGPT: Isaac Newton and his impostor.

Core ingredient: Sherman-Morrison formula

Powerful: tells us how the inverse $\mathbf{A}^{-1}$ changes, when we make a standard modification to $\mathbf{A}$.

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}$$

Inverse of
($\mathbf{A}$ plus rank-1 matrix)

(Inverse of $\mathbf{A}$) -
(different rank-1 matrix).

Where $\mathbf{A}, \mathbf{A}^{-1} \in \mathbb{R}^{n \times n}$ a matrix and its inverse

$\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ vectors of rank-1 update

(You do not need to memorize this equation for the exam)

BFGS

Broyden–Fletcher–Goldfarb–Shanno algorithm

- Maintain an approximate inverse Hessian $\mathbf{H}^{-1}$ (initially set to the identity).
- After taking a step $\Delta\mathbf{x}$, the gradient ∇f changes. Let's call this change $\Delta\nabla f$
- We would expect that the Hessian "explains" the observed change: $\mathbf{H}\,\Delta\mathbf{x} = \Delta\nabla f$.
  - In practice, it won't (because the Hessian is wrong). But this gives a nice recipe for updating the Hessian so that it is more accurate.
  - Formulated as a rank-1 update to both $\mathbf{H}$ and $\mathbf{H}^{-1}$, which can be cheaply performed via the Sherman-Morrison formula. (Details are beyond the scope of CS328)
- That's it. BFGS starts out like gradient descent but then becomes more "Newton-like" the more accurate its Hessian becomes.