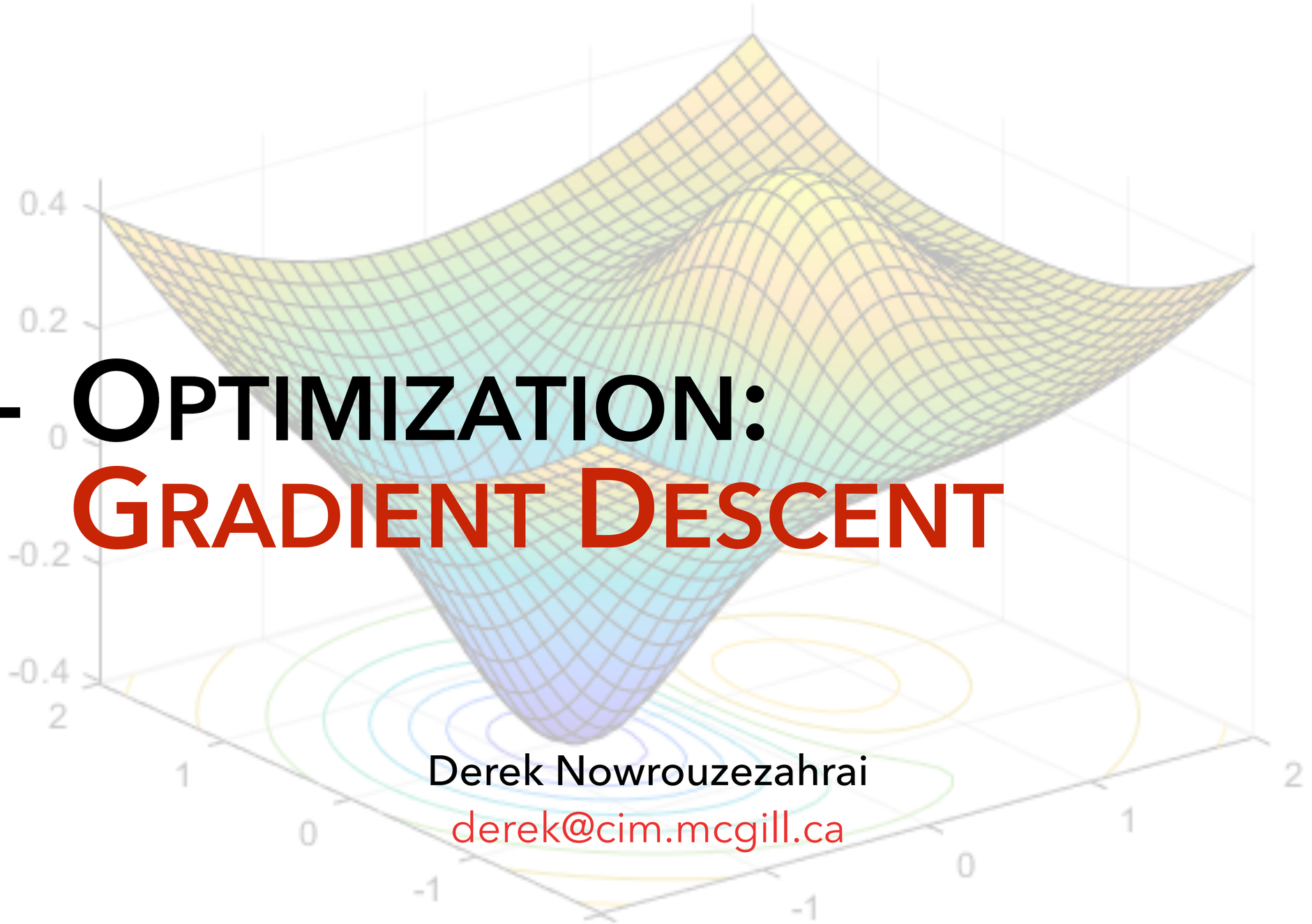


8A – Optimization: Gradient Descent

Derek Nowrouzezahrai
derek@cim.mcgill.ca

Back to Optimizing Scalar Functions

Consider *scalar functions* $f : \mathbb{R}^m \to \mathbb{R}$ again and recall that (quasi-)Newton methods sought estimates of the extrema of f

$$\mathbf{x}_* = \underset{\mathbf{x}}{\operatorname{argmin}}\, f(\mathbf{x})$$

by iterating as $\mathbf{x}_{k+1} = \mathbf{x}_k - \boldsymbol{\Gamma}\nabla f(\mathbf{x}_k)$, where different values of $\Gamma \in \mathbb{R}^{m \times m}$ correspond to different methods:

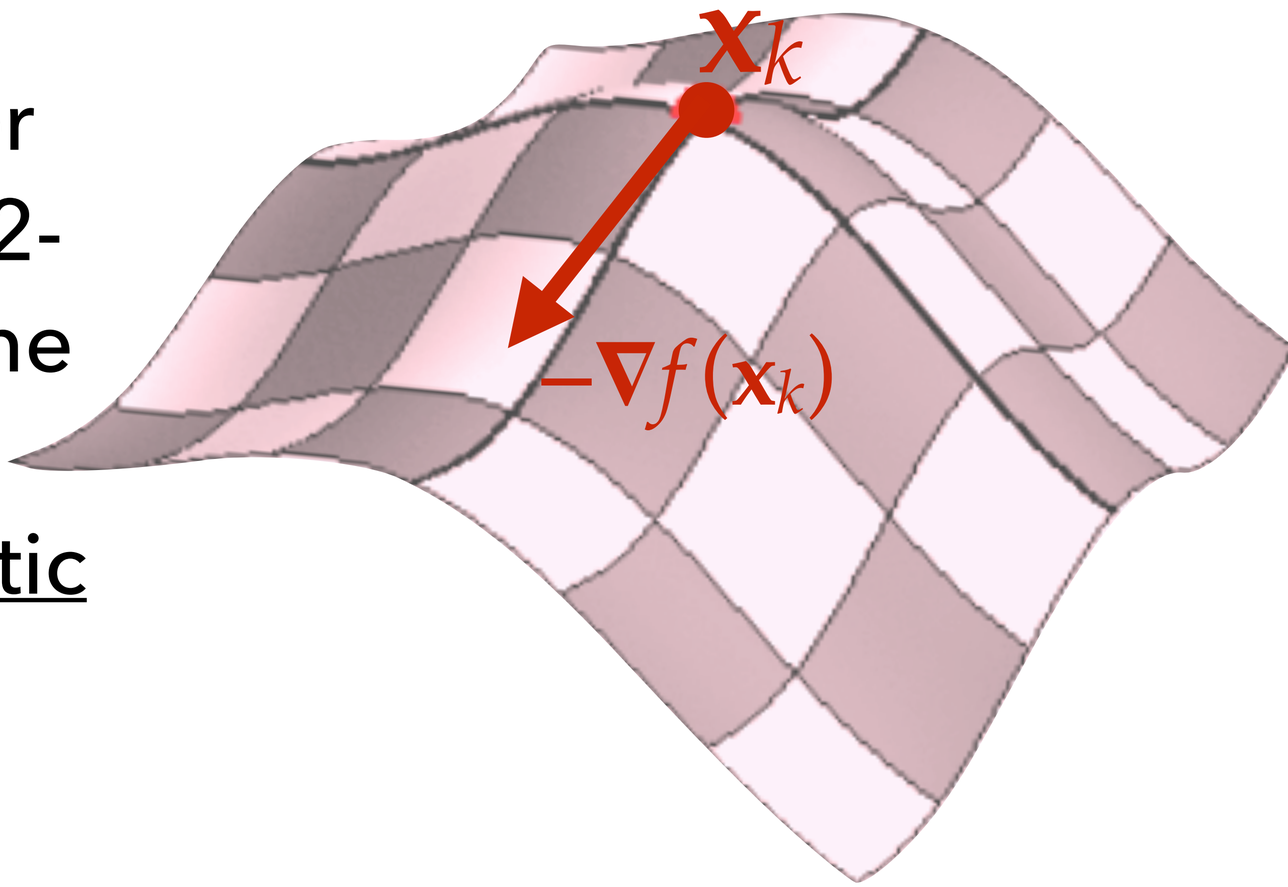
- $\boldsymbol{\Gamma} \coloneqq [\mathbf{H}_{f(\mathbf{x}_k)}]^{-1}$ leads to the standard Newton method,
- $\boldsymbol{\Gamma} \coloneqq \lambda\,[\mathbf{H}_{f(\mathbf{x}_k)}]^{-1}$ leads to the damped Newton method,
  - with or without applying *line search* to choose λ
- $\boldsymbol{\Gamma} \coloneqq [\mathbf{B}_{f(\mathbf{x}_k)}]^{-1}$ leads to Broyden-like quasi-Newton variants (DFP or BFGS)

Gradient Descent – Basic Idea

Remember that the iteration

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \boldsymbol{\Gamma}\nabla f(\mathbf{x}_k)\,,$$

with $\Gamma \in \mathbb{R}^{m\times m}$, uses 2nd-order information encoded in the $\boldsymbol{\Gamma}$ 2-tensor in order to ***transform*** the gradient $\nabla f(\mathbf{x}_k)$ vector to point directly away from the quadratic manifold's root

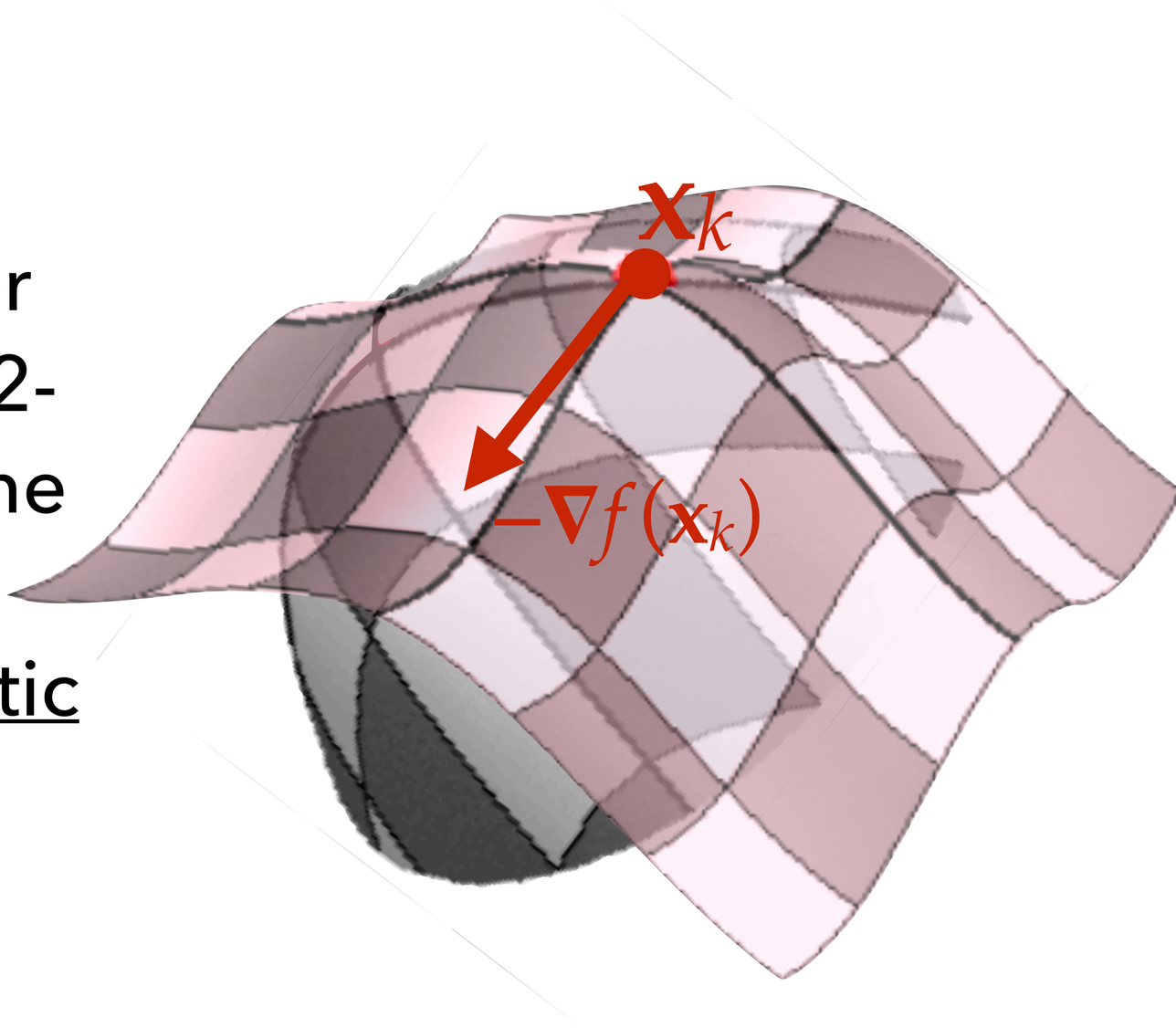


Gradient Descent – Basic Idea

Remember that the iteration

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \boldsymbol{\Gamma}\nabla f(\mathbf{x}_k)\,,$$

with $\Gamma \in \mathbb{R}^{m\times m}$, uses 2nd-order information encoded in the $\boldsymbol{\Gamma}$ 2-tensor in order to ***transform*** the gradient $\nabla f(\mathbf{x}_k)$ vector to point directly away from the quadratic manifold's root

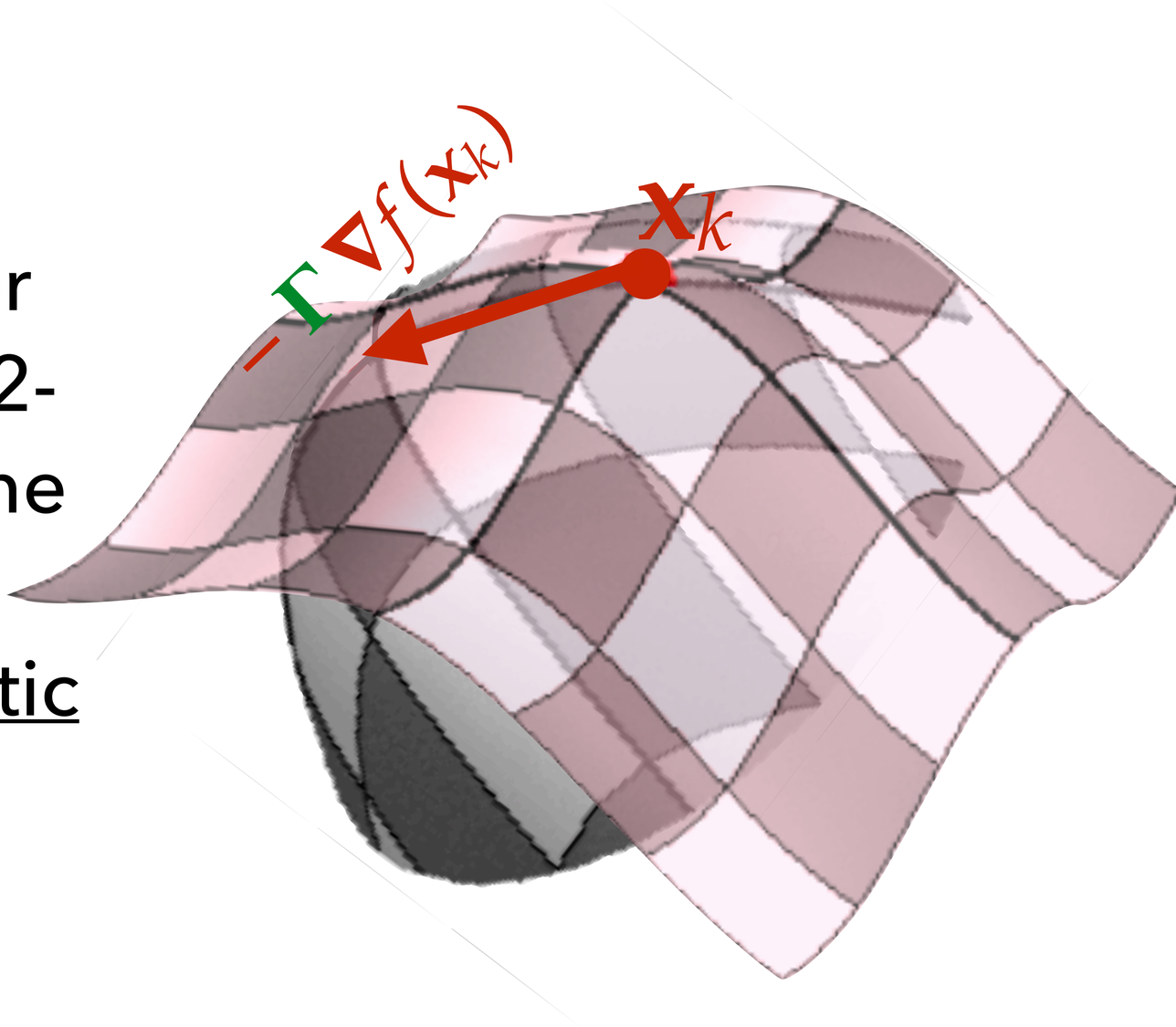


Gradient Descent – Basic Idea

Remember that the iteration

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \boldsymbol{\Gamma}\nabla f(\mathbf{x}_k)\,,$$

with $\Gamma \in \mathbb{R}^{m\times m}$, uses 2nd-order information encoded in the $\boldsymbol{\Gamma}$ 2-tensor in order to ***transform*** the gradient $\nabla f(\mathbf{x}_k)$ vector to point directly away from the quadratic manifold's root



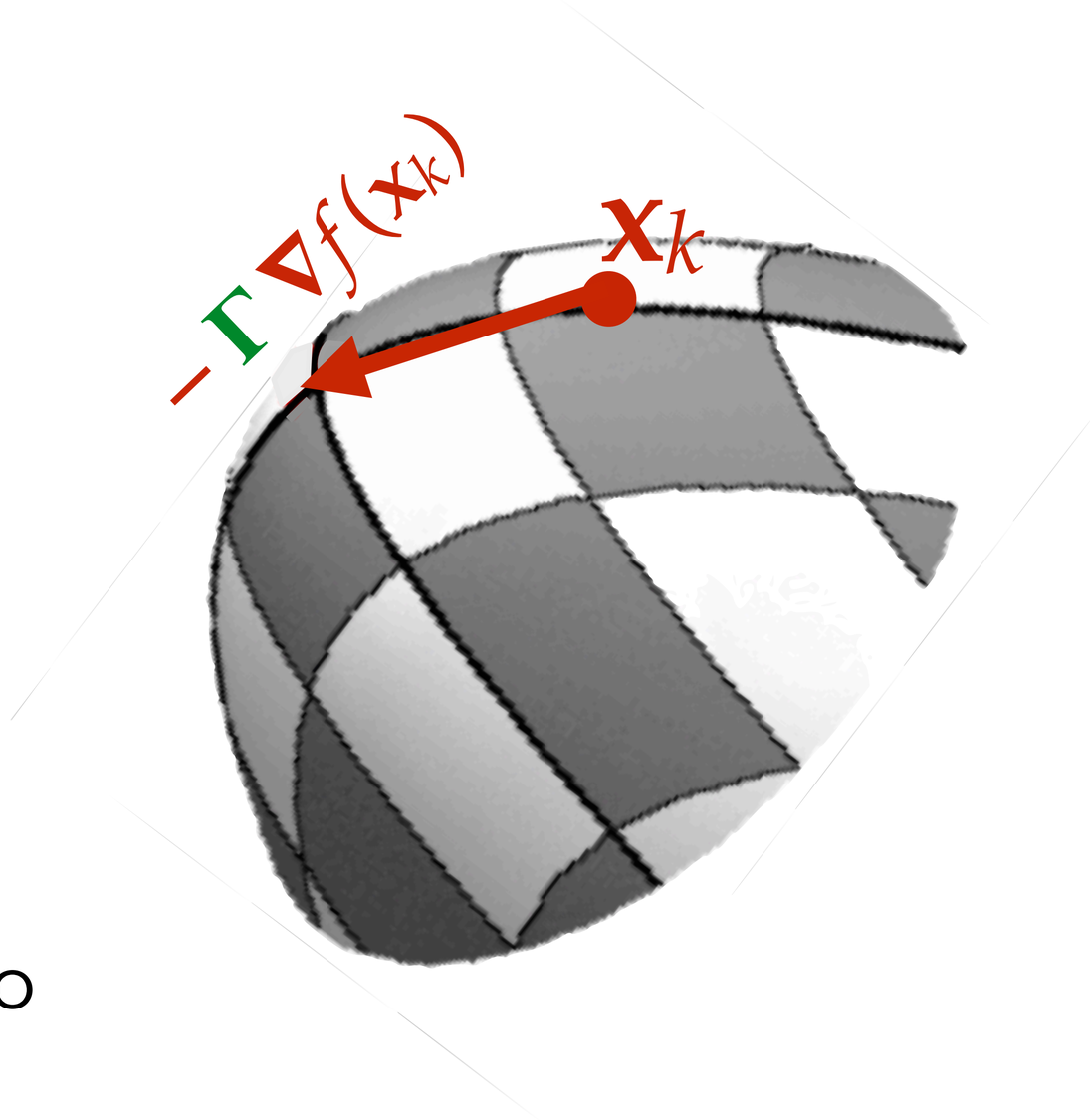
Gradient Descent – Basic Idea

Remember that the iteration

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \boldsymbol{\Gamma}\nabla f(\mathbf{x}_k)\,,$$

with $\Gamma \in \mathbb{R}^{m\times m}$, uses 2nd-order information encoded in the $\boldsymbol{\Gamma}$ 2-tensor in order to ***transform*** the gradient $\nabla f(\mathbf{x}_k)$ vector to point directly away from the quadratic manifold's root

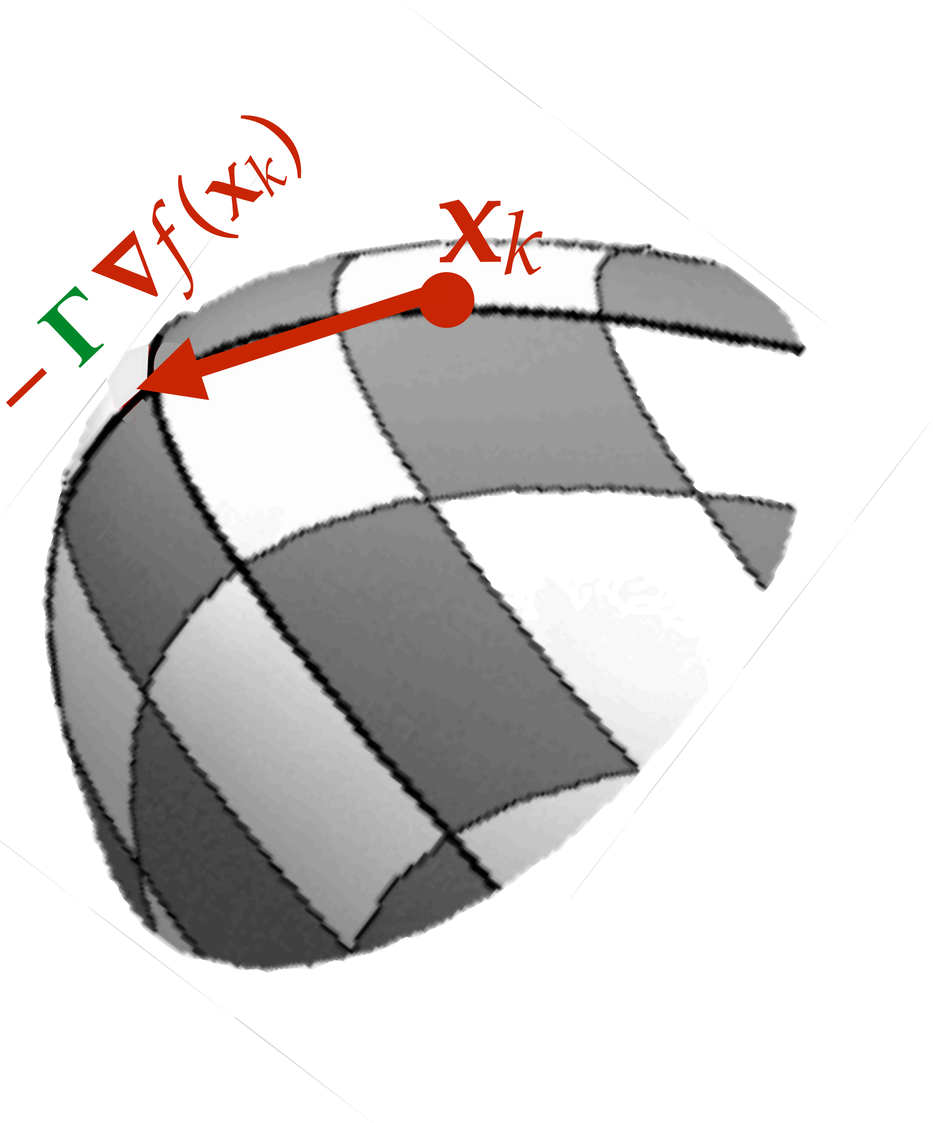
- it also changes $-\nabla f(\mathbf{x}_k)$'s length to "connect" $\mathbf{x}_k$ to the approx. root



Gradient Descent – Basic Idea

Note, however, that while the direction $-\Gamma\nabla f(\mathbf{x}_k)$ uses higher-order local information at $\mathbf{x}_k$ in the hopes that the new iterate brings us closer to a minimum, and so that

$$f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$$



There's no guarantee this will hold

- the condition of Γ plays a role in this

Gradient Descent – Basic Idea

What we do know, however, is *for small lengths* $\alpha_k \in \mathbb{R}$, walking along the direction of steepest descent at $\mathbf{x}_k$, $-\nabla f(\mathbf{x}_k)$, leads to a new point

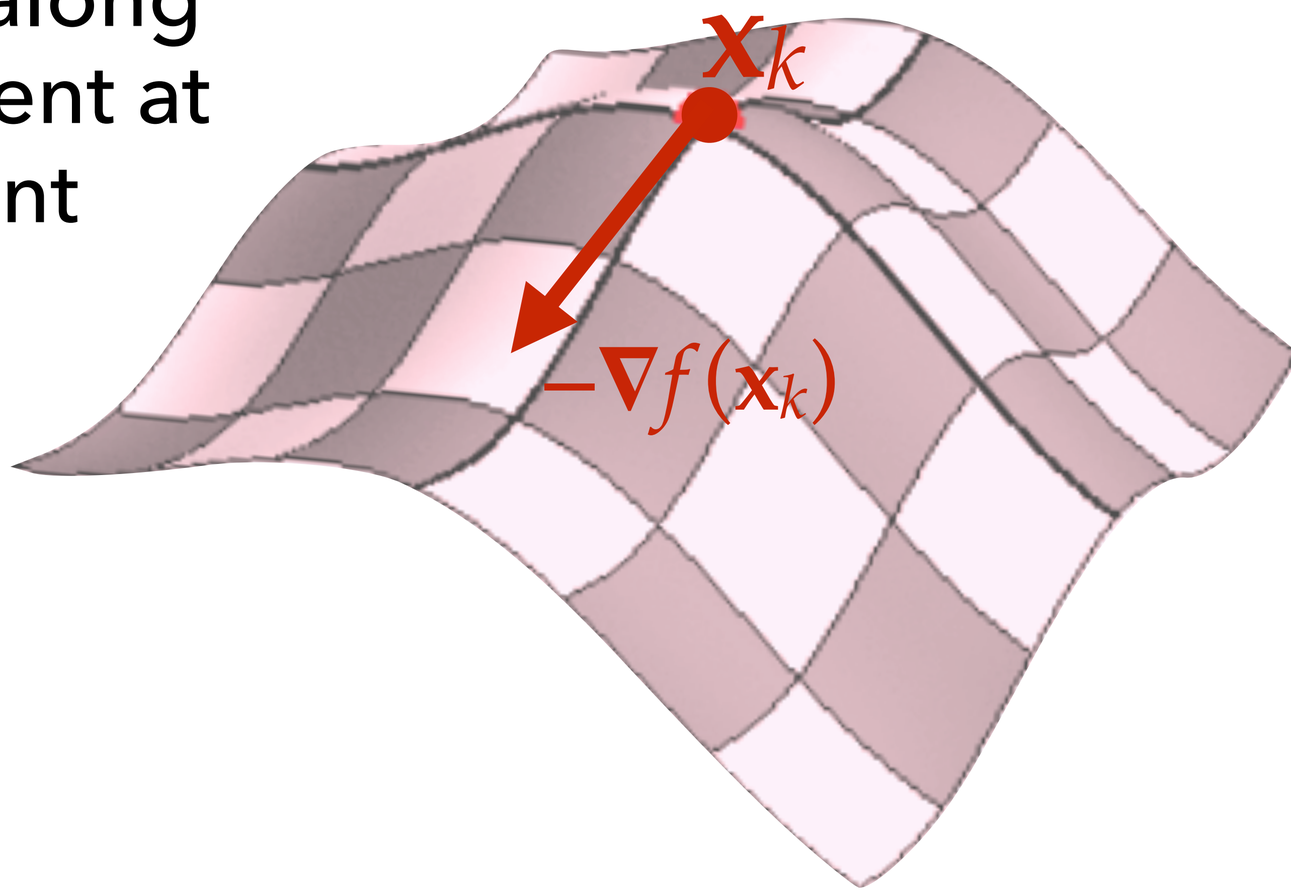
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$$

that, with $\alpha_k > 0$, **will satisfy**

$$f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$$

as long as $\nabla f(\mathbf{x}_k) \neq 0$

- so, how do we choose α_k?



Gradient Descent – Line Search (again)

Similarly to a Newton iteration along direction $-[\mathbf{H}_{f(\mathbf{x}_k)}]^{-1}\,\nabla f(\mathbf{x}_k)$, the **gradient descent** method can take (at least) one step, but now in the direction of steepest descent, $-\nabla f(\mathbf{x}_k)$

- we can set the distance α_k (that we step along $-\nabla f(\mathbf{x}_k)$) manually, *or*…
- we can use a ***line search*** to solve for a "locally optimal" distance

Doing so amounts, yet again, to solving a 1D minimization problem that "walks" along a 1D slice $g_k(\gamma)$ of the multivariate $f(\mathbf{x})$ along $-\nabla f(\mathbf{x}_k)$

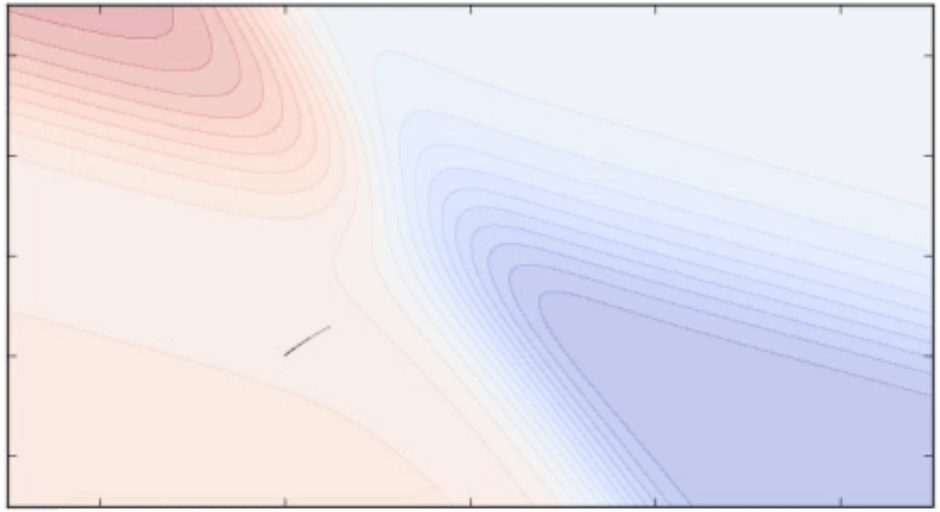
$$\alpha_k = \underset{\gamma}{\operatorname{argmin}} \underbrace{f\left(\mathbf{x}_k - \gamma\,\nabla f(\mathbf{x}_k)\right)}_{g_k(\gamma)} \text{ and } \mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k\,\nabla f(\mathbf{x}_k)$$

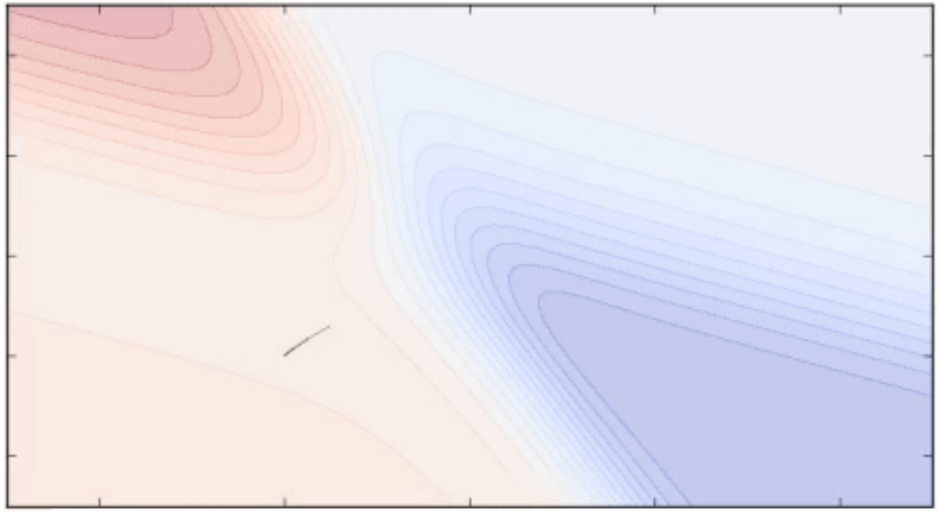
- again, apply any 1D optimization method to the α_k optimization problem

Gradient Descent – Observations

Compared to Newton's multivariate method, each iteration of gradient descent is much less expensive

- we don't have to form, nor invert (!), the Hessian
- set α_k with line search (expensive!) or manually/heuristically or hybrid
  - e.g., take many (possibly non-constant) steps along $-\nabla f(\mathbf{x}_k)$ per iterate



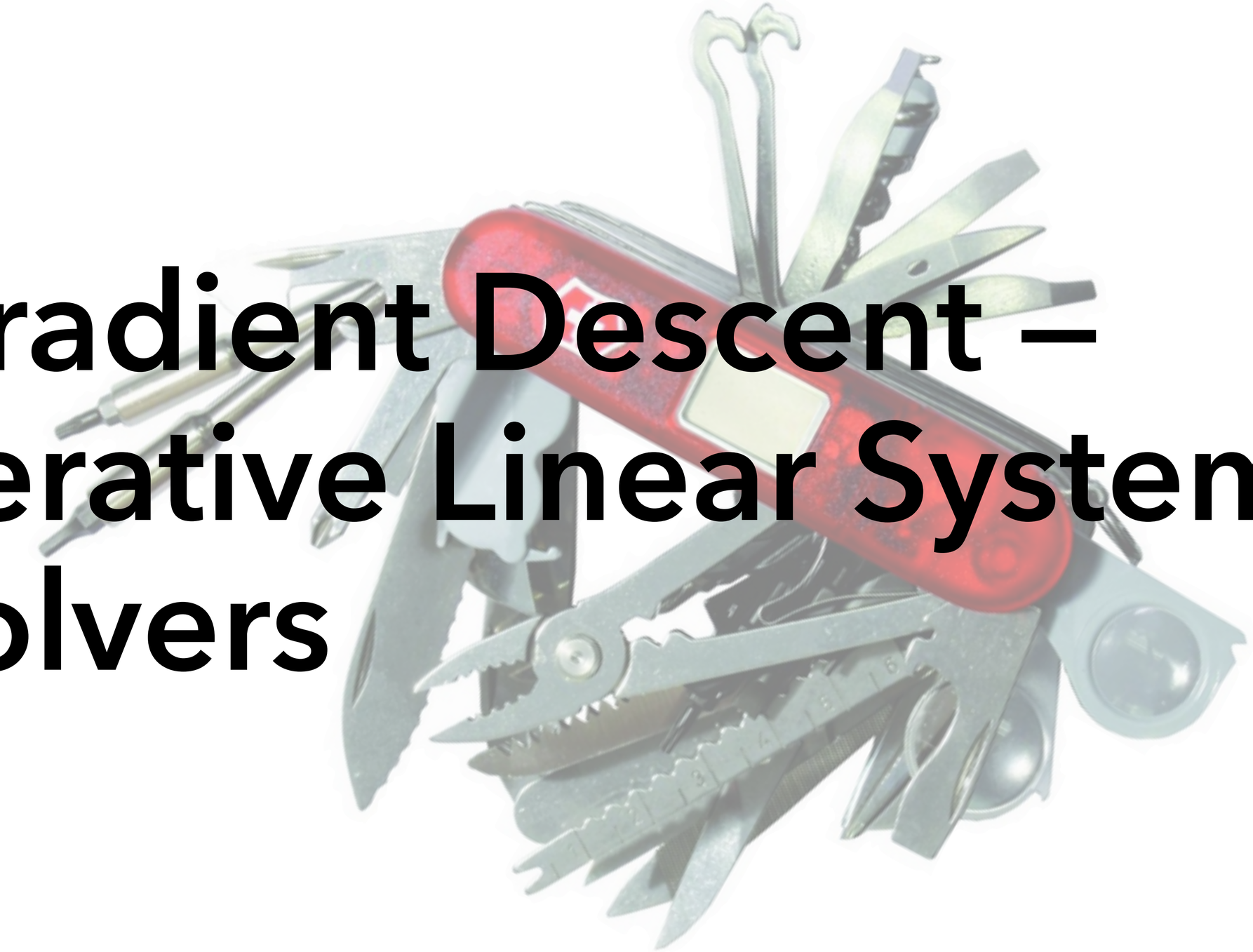


Gradient Descent – Observations

Compared to Newton's multivariate method, each iteration of gradient descent is much less expensive

- we don't have to form, nor invert (!), the Hessian
- set α_k with line search (expensive!) or manually/heuristically or hybrid
  - e.g., take many (possibly non-constant) steps along $-\nabla f(\mathbf{x}_k)$ per iterate
  - this latter strategy is commonly employed in machine learning
- need gradient evaluation, which can be approximated *à la* secant

Gradient Descent – Iterative Linear System Solvers



GD – Extensions & Other Applications

Gradient descent is a very robust tool

- applicable to scalar function minimization, with many variants
  - parallelizable and stochastic function evaluation
  - advanced step size selection processes (without 2^{nd}-order costs)
  - adaptive and multiple-steps per iteration, to avoid line search

What we'll see next, however, is that a very familiar (and important) problem can also be tackled using gradient descent – with ***significant*** benefits

Motivating Gradient Descent for Linear Systems

We will show how to tailor gradient descent to very efficiently solve **systems of linear equations** – why would you do this?

- we already have many **direct methods**: LU/Cholesky, QR and SVD
  + these approaches **directly** yield a solution
    - the unique solution for fully constrained systems,
    - a least-squares result for overconstrained systems without solutions, and
    - one of an infinite number of solutions for underdetermined systems
  + these approaches can amortize computation cost over many instances of the problem, for different RHS values $\mathbf{b}$ in $\mathbf{Ax} = \mathbf{b}$

Motivating Gradient Descent for Linear Systems

We will show how to tailor gradient descent to very efficiently solve **systems of linear equations** – why would you do this?

- we already have many **direct methods**: LU/Cholesky, QR and SVD
  - for an n x n matrix $\mathbf{A}$, these methods all require about $O(n^3)$ time
    - the direct methods become prohibitively costly as n increases
  - these methods require $O(n^2)$ storage, regardless of the *form* of $\mathbf{A}$
    - an important example is sparse matrices with, e.g., $O(n)$ non-zero elements – intermediate stages of direct methods (e.g., Gaussian Elimination) are almost certain to not yield intermediate sparse matrices

Gradient Descent for Linear Systems

Applications of gradient descent to minimization of non-linear functions focussed on *scalar functions* $f : \mathbb{R}^m \to \mathbb{R}$

- when solving linear systems, we are dealing with restricted forms (i.e., linear) of vector-valued functions, $\mathbf{f} : \mathbb{R}^m \to \mathbb{R}^n$
- we will, however, be able to reformulate the problem of solving a linear system to that of *minimizing a scalar function*

GD for SPD Systems – Set the Stage

First, some assumptions:

- $\mathbf{A}$ is an n x n square matrix
- $\mathbf{A}$ has full rank
- $\mathbf{A}$ is symmetric positive definite
  - $\mathbf{A} = \mathbf{A}^{\mathrm{T}}$
  - $\mathbf{x}^{\mathrm{T}}\,\mathbf{A}\,\mathbf{x} > 0 \quad \forall \mathbf{x} \neq 0$
  - $\mathbf{A} > 0$

Iterative Linear Solvers

Unlike **direct methods** that leverage decompositions to solve for a solution, **iterative solvers** progressively build better and better estimates of a solution

- given the k^{th} solution estimate $\mathbf{x}_k$, an iterative solution updates the estimate according to the following general update rule:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$$

  where α_k is the *step size* (or length) and $\mathbf{d}_k$ the *step direction*

The goal when designing iterative methods is to find α_k's and $\mathbf{d}_k$'s that drive $\|\mathbf{A}\mathbf{x}_k - \mathbf{b}\|$ to zero as quickly as possible

Gradient Descent for SPD Systems

We must express the solution $\mathbf{x}$ of $\mathbf{Ax} = \mathbf{b}$ as the minimizer of an $f(\mathbf{x})$ before using gradient descent to find $\mathbf{x}$

- recall: every root finding problem can be framed as a minimization, and vice-versa
- we want to solve for the root of $\mathbf{Ax} - \mathbf{b} = \mathbf{0}$ and so seek the minimum of the $f(\mathbf{x})$ such that $\nabla f(\mathbf{x}) = 0 = \mathbf{Ax} - \mathbf{b}$
  - the minimum $\mathbf{x}$ (if it exists) will satisfy our linear system
  - we terminate iterative solvers *close* to the solution, so when

$$\|\nabla f(\mathbf{x}_k)\| < \epsilon_\nabla \quad \text{or} \quad |f(\mathbf{x}_k) - f(\mathbf{x}_{k-1})| < \epsilon_f$$

Gradient Descent for SPD Systems

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$$

This is actually all we need to apply a (very) basic version of gradient descent that iterates towards a solution as

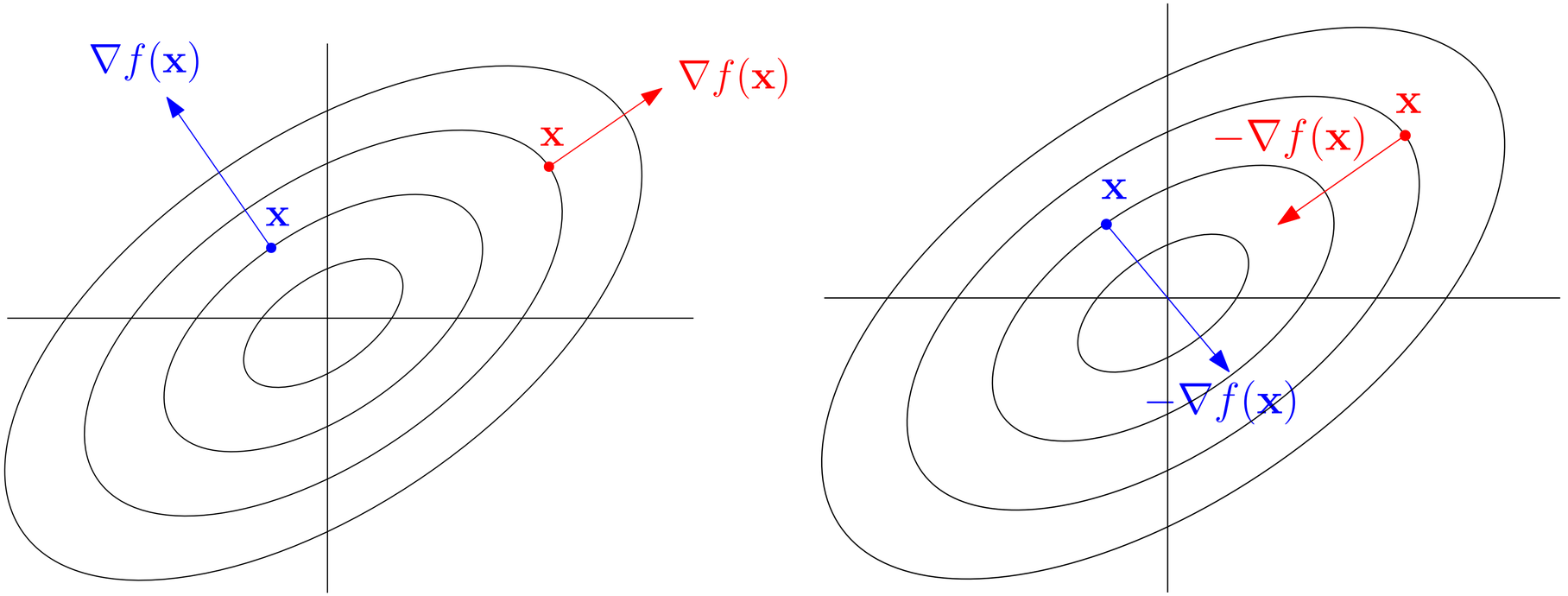
$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) \\ &= \mathbf{x}_k - \alpha_k (\mathbf{A}\mathbf{x}_k - \mathbf{b})\end{aligned}$$

so long as we choose a "sufficiently small" step size α_k this iteration will (hopefully) inch towards a solution as

$$\|\nabla f(\mathbf{x}_0)\| > \|\nabla f(\mathbf{x}_1)\| > \ldots > \|\nabla f(\mathbf{x}_k)\| \quad \text{or} \quad f(\mathbf{x}_0) > f(\mathbf{x}_1) > \ldots > f(\mathbf{x}_k)$$

Gradient Descent for SPD Systems

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) \\ &= \mathbf{x}_k - \alpha_k (\mathbf{A}\mathbf{x}_k - \mathbf{b})\end{aligned}$$



Choosing a Good Step Size

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k (\mathbf{A}\mathbf{x}_k - \mathbf{b})$$

If we want to be more effective in our choice for α_k, then we can choose an optimal step size α_k^* with a line search:

$$\alpha_k^* = \underset{\gamma}{\operatorname{argmin}}\, f\left(\mathbf{x}_k - \gamma \nabla f(\mathbf{x}_k)\right) = \underset{\gamma}{\operatorname{argmin}}\, f\left(\mathbf{x}_k - \gamma \left(\mathbf{A}\mathbf{x}_k - \mathbf{b}\right)\right)$$

but we need to first find $f(\mathbf{x})$ in order to, e.g., perform a line search with a 1D optimization routine

Choosing the Optimal Step Size

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$$

Since the gradient of $f(\mathbf{x})$ is a linear function of $\mathbf{x}$, $f(\mathbf{x})$ must be quadratic in $\mathbf{x}$ and it's easy to show that it has the form

$$f(\mathbf{x}) = \tfrac{1}{2}\,\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} - \mathbf{b}^{\mathrm{T}}\mathbf{x} + c \quad \text{for any } c \in \mathbb{R}$$

and now we could apply a 1D optimization with $f(\mathbf{x})$ to solve for the optimal step size

$$\alpha_k^* = \underset{\gamma}{\operatorname{argmin}}\, f\left(\mathbf{x}_k - \gamma\left(\mathbf{A}\mathbf{x}_k - \mathbf{b}\right)\right)$$

- this, however, won't be necessary: we can obtain α_k^* analytically!

Choosing the Optimal Step Size

$$f(\mathbf{x}) = \tfrac{1}{2}\,\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} - \mathbf{b}^{\mathrm{T}}\mathbf{x} + c$$

$$\alpha_k^* = \underset{\gamma}{\operatorname{argmin}}\, f\left(\mathbf{x}_k - \gamma\left(\mathbf{A}\mathbf{x}_k - \mathbf{b}\right)\right)$$

Derivation strategy:

1. expand $f(\mathbf{x}_k - \gamma(\mathbf{A}\mathbf{x}_k - \mathbf{b})) = g_k(\gamma)$,
2. solve for α_k^*, the value of γ that minimizes $g_k(\gamma)$, as $\frac{dg_k(\gamma)}{d\gamma} = g_k'(\gamma) = 0$

Choosing the Optimal Step Size

$$f(\mathbf{x}) = \tfrac{1}{2}\,\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c$$

Derivation strategy:

1. expand $f(\mathbf{x}_k - \gamma(\mathbf{A}\mathbf{x}_k - \mathbf{b})) = g_k(\gamma)$,
2. solve for α_k^*, the value of γ that minimizes $g_k(\gamma)$, as $\frac{dg_k(\gamma)}{d\gamma} = g_k'(\gamma) = 0$

---

let $\mathbf{y}_k \coloneqq \mathbf{A}\mathbf{x}_k - \mathbf{b} = \nabla f(\mathbf{x}_k)$

$$\begin{aligned} g_k(\gamma) = f(\mathbf{x}_k - \gamma\,\mathbf{y}_k) &= \tfrac{1}{2}\,(\mathbf{x}_k - \gamma\,\mathbf{y}_k)^T\mathbf{A}(\mathbf{x}_k - \gamma\,\mathbf{y}_k) - \mathbf{b}^T(\mathbf{x}_k - \gamma\,\mathbf{y}_k) + c \\ &= \tfrac{1}{2}\,(\mathbf{x}_k^T\mathbf{A}\mathbf{x}_k - 2\gamma\,\mathbf{x}_k^T\mathbf{A}\mathbf{y}_k + \gamma^2\mathbf{y}_k^T\mathbf{A}\mathbf{y}_k) - \mathbf{b}^T\mathbf{x}_k + \gamma\mathbf{b}^T\mathbf{y}_k + c \\ &= \tfrac{1}{2}\,\gamma^2\mathbf{y}_k^T\mathbf{A}\mathbf{y}_k - \gamma(\mathbf{x}_k^T\mathbf{A}\mathbf{y}_k - \mathbf{b}^T\mathbf{y}_k) - \mathbf{b}^T\mathbf{x}_k + \mathbf{x}_k^T\mathbf{A}\mathbf{x}_k + c \end{aligned}$$

Choosing the Optimal Step Size

$$f(\mathbf{x}) = \tfrac{1}{2}\,\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c$$

Derivation strategy:

1. expand $f(\mathbf{x}_k - \gamma(\mathbf{A}\mathbf{x}_k - \mathbf{b})) = g_k(\gamma)$,
2. solve for α_k^*, the value of γ that minimizes $g_k(\gamma)$, as $\frac{dg_k(\gamma)}{d\gamma} = g_k'(\gamma) = 0$

$$g_k(\gamma) = f(\mathbf{x}_k - \gamma\,\mathbf{y}_k) = \tfrac{1}{2}\,\gamma^2\mathbf{y}_k^T\mathbf{A}\mathbf{y}_k - \gamma(\mathbf{x}_k^T\mathbf{A}\mathbf{y}_k - \mathbf{b}^T\mathbf{y}_k) - \mathbf{b}^T\mathbf{x}_k + \mathbf{x}_k^T\mathbf{A}\mathbf{x}_k + c$$

$$g_k'(\gamma) = \gamma\,\mathbf{y}_k^T\mathbf{A}\mathbf{y}_k - \mathbf{x}_k^T\mathbf{A}\mathbf{y}_k + \mathbf{b}^T\mathbf{y}_k$$

$$g_k'(\gamma^*) = \gamma^*\,\mathbf{y}_k^T\mathbf{A}\mathbf{y}_k - \mathbf{x}_k^T\mathbf{A}\mathbf{y}_k + \mathbf{b}^T\mathbf{y}_k = 0$$

$$\gamma^* = (\mathbf{x}_k^T\mathbf{A}\mathbf{y}_k - \mathbf{b}^T\mathbf{y}_k)/(\mathbf{y}_k^T\mathbf{A}\mathbf{y}_k)$$

$$= (\mathbf{x}_k^T\mathbf{A} - \mathbf{b}^T)\,\mathbf{y}_k/(\mathbf{y}_k^T\mathbf{A}\mathbf{y}_k)$$

$$= (\mathbf{A}\mathbf{x}_k - \mathbf{b})^T\,\mathbf{y}_k/(\mathbf{y}_k^T\mathbf{A}\mathbf{y}_k)$$

let $\mathbf{y}_k \coloneqq \mathbf{A}\mathbf{x}_k - \mathbf{b} = \nabla f(\mathbf{x}_k)$

$$\gamma^* = \|\mathbf{y}_k\|^2 / (\mathbf{y}_k^T\mathbf{A}\mathbf{y}_k)$$

Steepest Descent for SPD Systems

Unlike the step size you would obtain by line search, in this linear setting we derived an *optimal* step size for each gradient descent iteration: $\alpha_k^* = \|\mathbf{y}_k\|^2 / (\mathbf{y}_k{}^T \mathbf{A} \mathbf{y}_k)$

So, gradient descent for linear systems iterates as follows:

- choosing the descent direction $\mathbf{y}_k := \nabla f(\mathbf{x}_k) = \mathbf{A}\mathbf{x}_k - \mathbf{b}$
- computing the optimal step size as $\alpha_k^* = \|\mathbf{y}_k\|^2 / (\mathbf{y}_k{}^T \mathbf{A} \mathbf{y}_k)$
- updating the estimate of the minimum as $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k^* (\mathbf{A}\mathbf{x}_k - \mathbf{b})$

GD using the optimal step size is referred to as the *steepest descent algorithm*

SD for SPD Systems – Analysis

We can derive an interesting relationship between subsequent descent directions $\nabla f(\mathbf{x}_k) = \mathbf{A}\mathbf{x}_k - \mathbf{b}$ and $\nabla f(\mathbf{x}_{k+1}) = \mathbf{A}\mathbf{x}_{k+1} - \mathbf{b}$ in the iterative process:

- subsequent gradient descent directions are mutually <u>perpendicular</u>!
- this leads to an iterative "zig-zag" behaviour over iterations

$$\|\mathbf{y}_k\|^2 / (\mathbf{y}_k{}^T \mathbf{A}\mathbf{y}_k) = \alpha_k^*$$

$$\alpha_k^* (\mathbf{y}_k{}^T \mathbf{A}\mathbf{y}_k) = \|\mathbf{y}_k\|^2$$

$$(\mathbf{y}_k)^T \mathbf{y}_k - \alpha_k^* (\mathbf{A}\mathbf{y}_k)^T \mathbf{y}_k = 0$$

$$(\mathbf{A}\mathbf{x}_k - \mathbf{b})^T \mathbf{y}_k - \alpha_k^* (\mathbf{A}\mathbf{y}_k)^T \mathbf{y}_k = 0$$

$$(\mathbf{A}\mathbf{x}_k - \mathbf{b} - \alpha_k^* \mathbf{A}\mathbf{y}_k)^T \mathbf{y}_k = 0$$

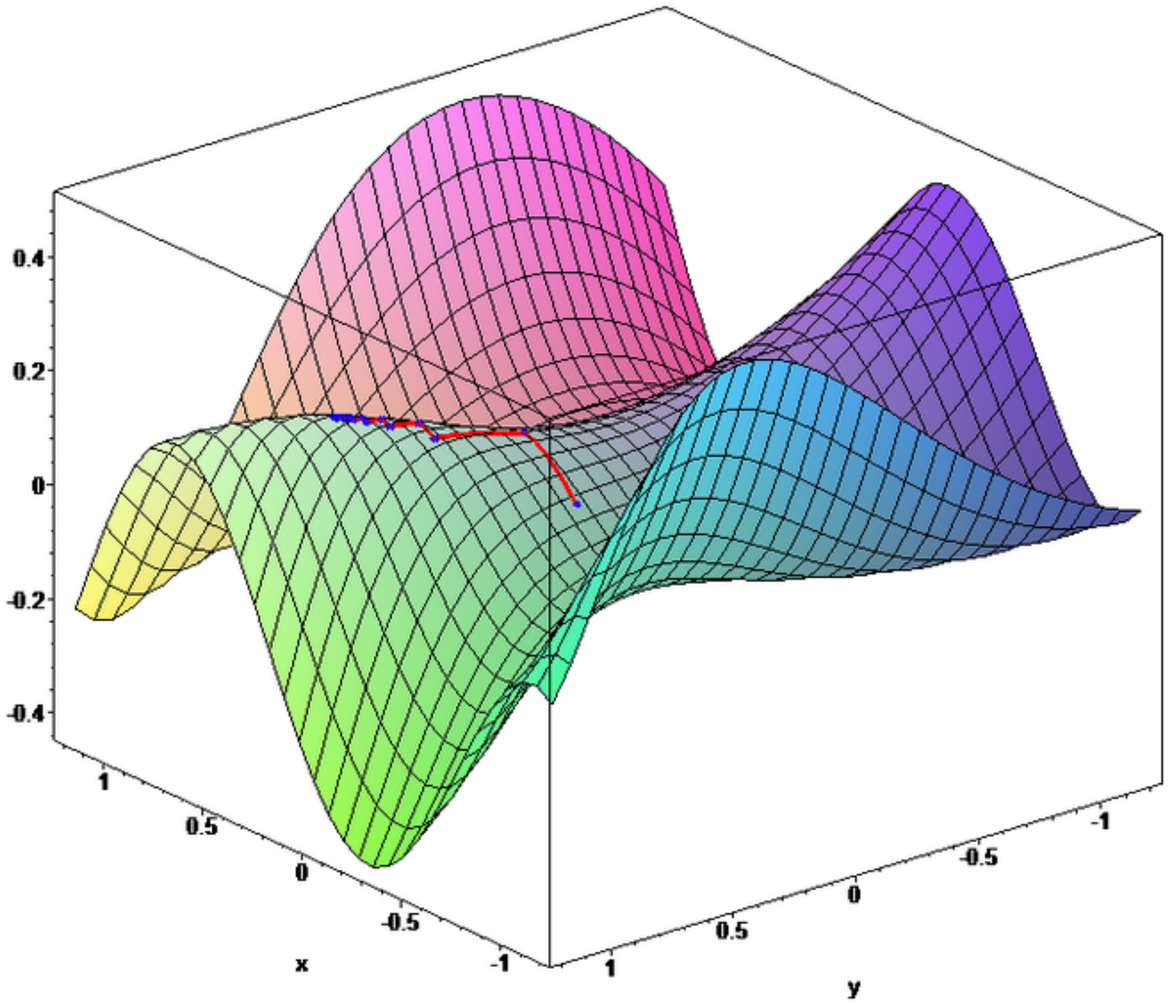
$$(\mathbf{A}(\mathbf{x}_k - \alpha_k^* \mathbf{y}_k) - \mathbf{b})^T \mathbf{y}_k = 0$$

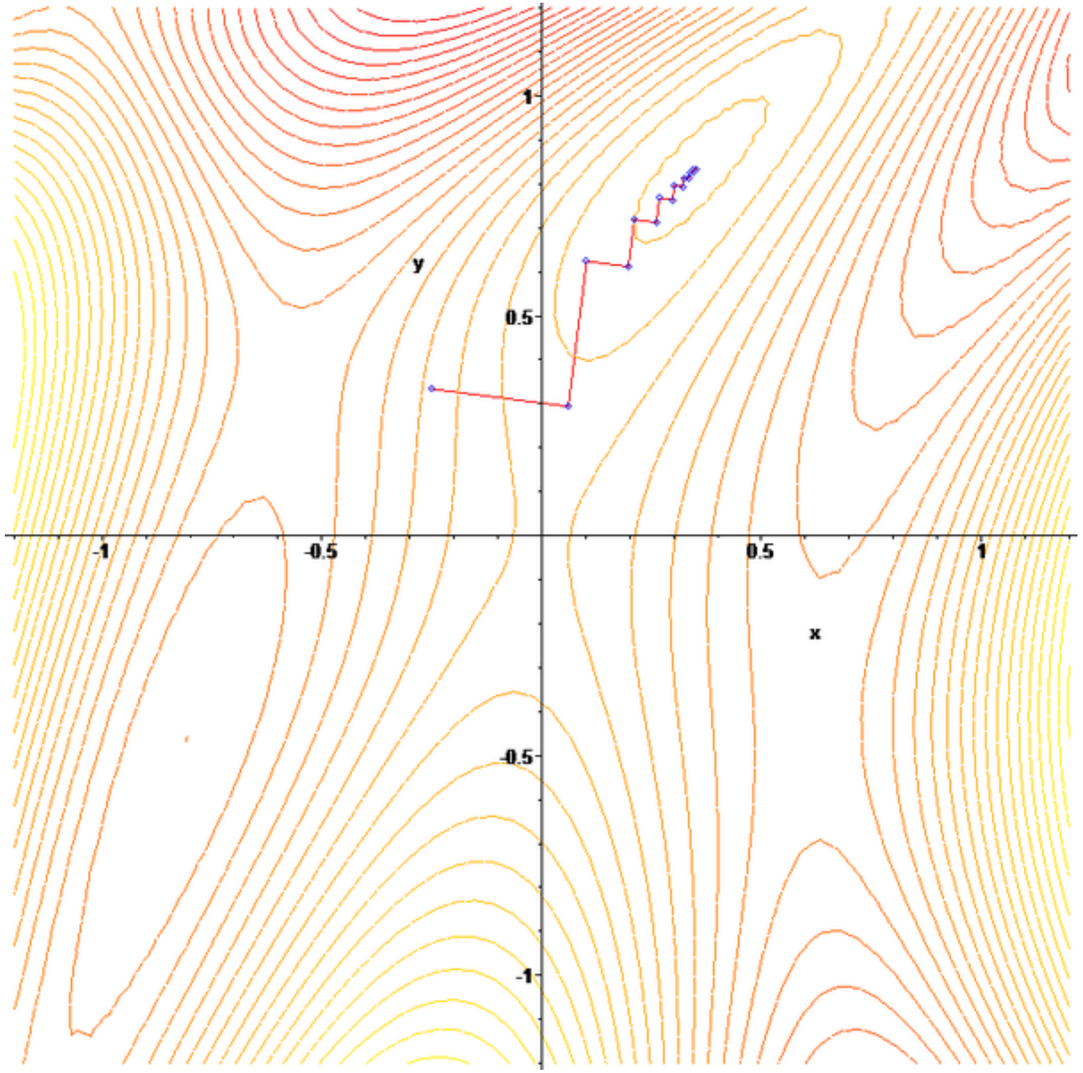
$$(\mathbf{A}\mathbf{x}_{k+1} - \mathbf{b})^T \mathbf{y}_k = 0$$

$$(-\mathbf{y}_{k+1})^T \mathbf{y}_k = 0$$

$$\nabla f(\mathbf{x}_{k+1})^T \nabla f(\mathbf{x}_k) = 0$$

Steepest Descent for SPD Systems



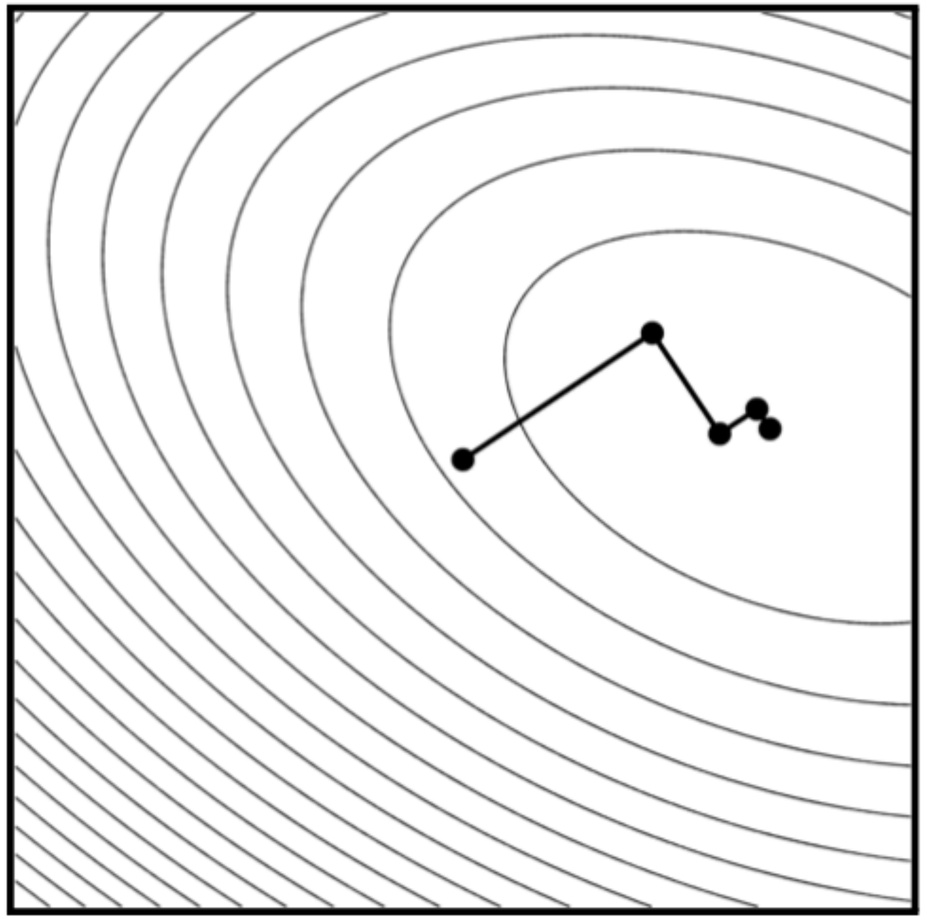


SD for SPD Systems – Convergence

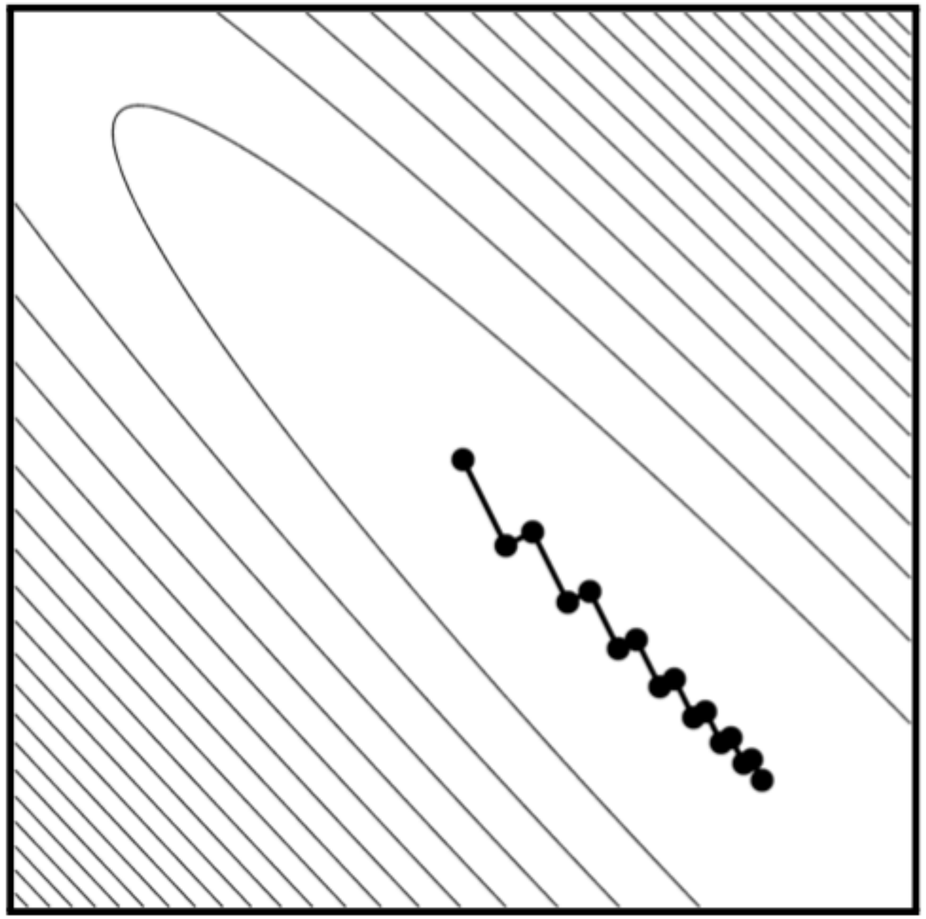
A thorough convergence analysis for gradient descent yields three notable conclusions [Solomon; section 11.1.2]

1. it is **guaranteed** to eventually converge to a unique* minimum,
2. change in *backward error* is bounded above by $1 - 1/\ \text{cond}(\mathbf{A})$, and
3. since $\text{cond}(\mathbf{A}) \geq 1$, the convergence of gradient descent improves as the conditioning of $\mathbf{A}$ improves

SD for SPD Systems – Convergence



Well conditioned A



Poorly conditioned A

SD for SPD Systems – Convergence

When compared to direct methods – costing roughly $O(n^3)$ – the number of iterations needed to get sufficiently close to a solution with steepest descent may be very large (depending on the conditioning of the system)

- if $\mathbf{A}$ is sparse with $O(n)$ non-zero elements, then each iteration of SD costs roughly $O(n)$ and we can afford to take $O(n^2)$ iterations before we start becoming more expensive than direct methods
- however, for dense $\mathbf{A}$, each iteration costs $O(n^2)$ and after $O(n)$ iterations SD can become more expensive than a direct method

SD for SPD Systems – Behaviour

As we showed earlier, SD ping-pongs between mutually perpendicular directions when solving a linear system

- this leads to us possible following many “parallel” directions during the process of convergence