CS 524: Introduction to Optimization
Lecture 37 : Nonlinear programs

Michael Ferris

Computer Sciences Department
University of Wisconsin-Madison

December 1, 2023

Nonlinear Programming: Why?

- An anecdote: July, 1948. A young and frightened George Dantzig, presents his newfangled "linear programming" to a meeting of the Econometric Society of Wisconsin, attended by distinguished scientists like Hotelling, Koopmans, and Von Neumann. Following the lecture, Hotelling[a] pronounced to the audience:
- But we all know the world is nonlinear!

[a] in Dantzig's words "a huge whale of a man"

- The world is indeed nonlinear
- Physical Processes and Properties
  - Equilibrium
  - Enthalpy (heat and work)
- Abstract Measures
  - Economies of Scale
  - Covariance
  - Utility of decisions

General (Nonlinear) Optimization Model

$$\max_{x \in \mathbb{R}^n} f(x)$$

subject to

$$g_i(x) \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b_i \quad \forall i \in M$$

$$x \in X$$

- x is an n-dimensional *vector*: $x = (x_1, x_2, \ldots, x_n)$.
- $f(x)$: Objective Function
- M : index set of constraints
- $g_i(x)\{\leq, =, \geq\} b_i$: Constraint
- X : Explicit Constraint Set Maybe $X \subseteq \mathbb{Z}^n$

Objective Functions

- A *linear function* $f : \mathbb{R}^n \to \mathbb{R}$ is a weighted sum:
  - $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} c_i x_i$ for given coefficients $\{c_1, \ldots, c_n\}$.
  - $f(x) = c^T x$
- Everything that is not linear is nonlinear
  - $f(x) = \sin(x)$
  - $f(x, y, z) = 2x - 3y + 14z^2$
  - $f(x) = x^T Q x = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j$
  - $f(x_1, x_2) = (3x_1 - 2x_2 + 7)/(x_1 + x_2)$.

Functions

- multiplication (*), division (/), exponentiation (**)
- `abs`, `arctan`, `ceil`, `cos`, `errorf`, `exp`, `floor`, `log`, `log10`, `max`, `min`, `mod`, `normal`, `power`, `round`, `sign`, `sin`, `sqr`, `sqrt`, `trunc`, `uniform`
- Note that `abs`, `ceil`, `floor`, `max`, `min`, `mod`, `normal`, `round`, `sign`, `trunc`, `uniform` cannot be used in NLP models (not differentiable or deterministic)
- log only defined on $x > 0$. `sqrt` defined on $x \geq 0$, but its derivative is not defined when $x = 0$.

A 'Canonical' Nonlinear Program (NLP)

$$\max f(x)$$

s.t.

$$g_i(x) \leq b_i \quad \forall i = 1, \ldots m$$
$$x_j \geq 0 \quad \forall j = 1, \ldots n$$

- Problem is "easy" if $f(\cdot)$ is concave and $g_i(\cdot)$ is convex $\forall i$
- $f(\cdot)$ concave $\Rightarrow$ Local maximum is global maximum
- $g_i(\cdot)$ is convex $\forall i \Rightarrow$ the feasible region is a convex set

First things first

The labels **nonlinear** or **nonconvex** are not particularly informative or helpful in practice.

- Throughout the course we studied properties of linear constraints, convex quadratics, even MIPs. We can't expect there to be a rigorous science for "everything else".
- It doesn't really make sense to define something as not having a particular property.
- "I'm an ECE professor" is a very informative statement. But using the label "non-(ECE professor)" is virtually meaningless. It could be a student, a horse, a tomato,...

Important categories

- **Continuous vs discrete:** As with LPs, the presence of binary or integer constraints is an important feature.
- **Smoothness:** Are the constraints and the objective function differentiable? twice-differentiable?
- **Qualitative shape:** Are there many local minima?
- **Problem scale:** A few variables? hundreds? thousands?

This sort of information is very useful in practice. It helps you decide on an appropriate solution approach.

Overview of NLP algorithms

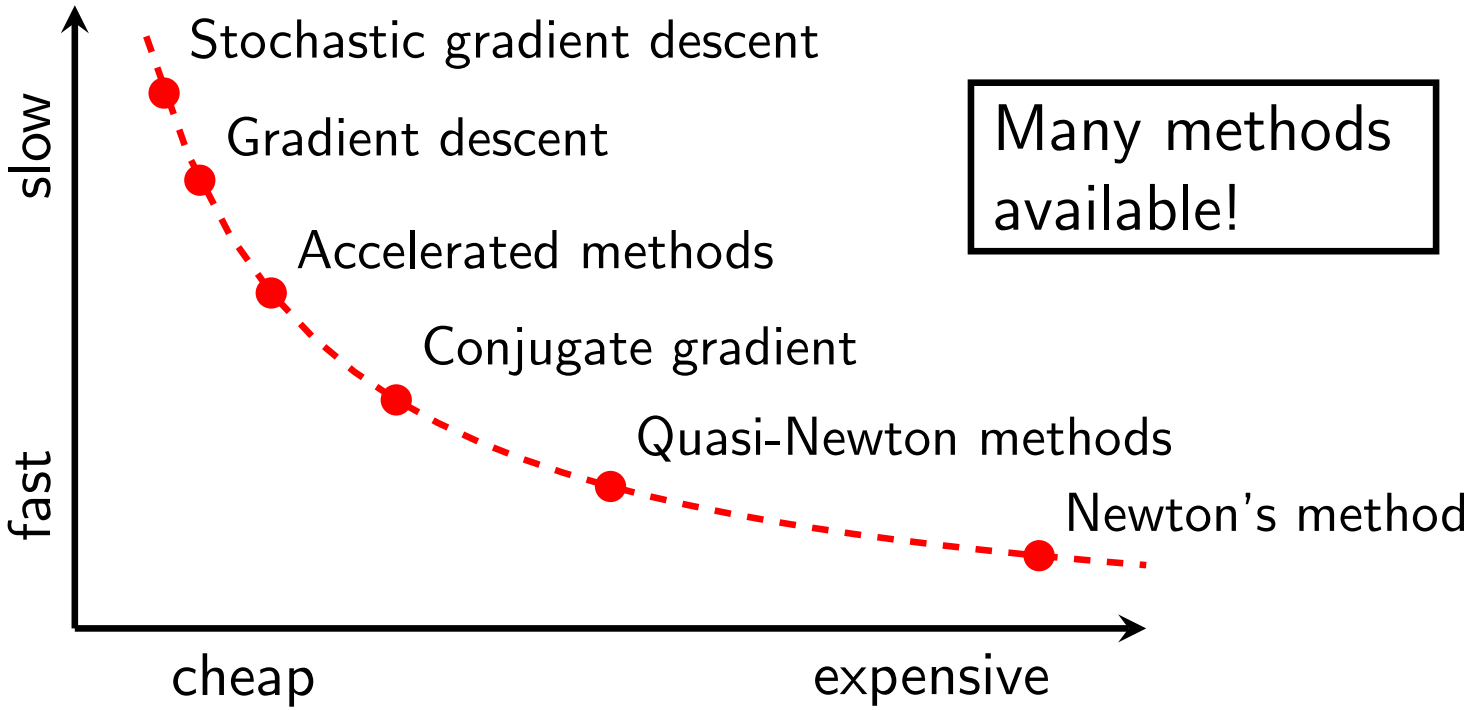
1. Are the functions differentiable? Can we efficiently compute gradients or second derivatives of the f_i?
2. What problem size are we dealing with? a few variables and constraints? hundreds? thousands? millions?
3. Do we want to find local optima, or do we need the global optimum (more difficult!)
4. Does the objective function have a large number of local minima? or a relatively small number?

Note: items **3** and **4** don't matter if the problem is convex. In that case any local minimum is also a global minimum!

Local methods using derivatives

Let's start with the unconstrained case:

$$\underset{x}{\text{minimize}} \quad f(x)$$



Iterative methods

Local methods iteratively step through the space looking for a point where $\nabla f(x) = 0$.

1. pick a starting point x_0.
2. choose a direction to move in Δ_k. This is the part where different algorithms do different things.
3. update your location $x_{k+1} = x_k + \Delta_k$
4. repeat until you're happy with the function value or the algorithm has ceased to make progress.

Vector calculus

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a twice-differentiable function.

- The **gradient** of f is a function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ defined by:

$$\left[\nabla f\right]_i = \frac{\partial f}{\partial x_i}$$

  $\nabla f(x)$ points in the direction of *greatest increase* of f at x.

- The **Hessian** of f is a function $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^{n \times n}$ where:

$$\left[\nabla^2 f\right]_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

  $\nabla^2 f(x)$ is a matrix that encodes the *curvature* of f at x.

Vector calculus

Example: suppose $f(x,y) = x^2 + 3xy + 5y^2 - 7x + 2$

- $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + 3y - 7 \\ 3x + 10y \end{bmatrix}$
- $\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 3 & 10 \end{bmatrix}$

Taylor's theorem in n dimensions

$$f(x) \approx \overbrace{f(x_0) + \nabla f(x_0)^{\mathsf{T}}(x - x_0)}^{\text{best linear approximation}} + \underbrace{\frac{1}{2}(x - x_0)^{\mathsf{T}} \nabla^2 f(x_0)(x - x_0)}_{\text{best quadratic approximation}} + \ldots$$

Gradient descent

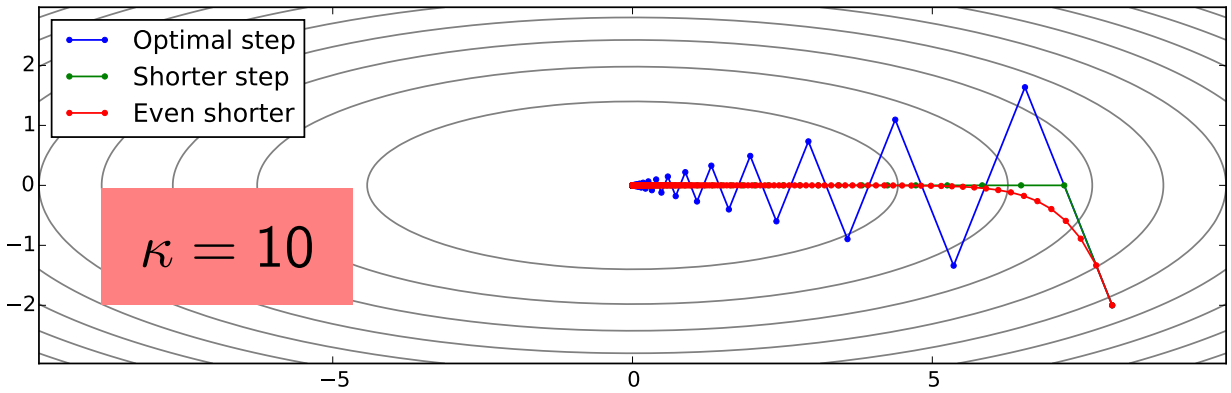
- The simplest of all iterative methods. It's a first-order method, which means it only uses gradient information:

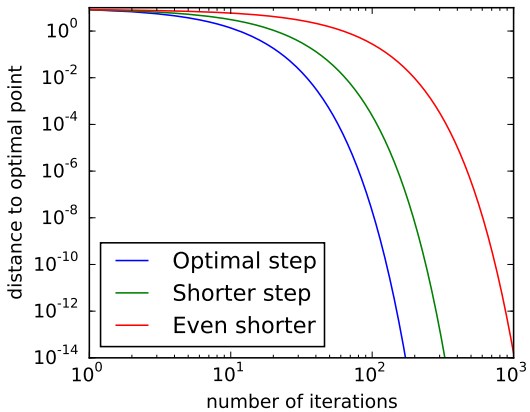
$$x_{k+1} = x_k - t_k \nabla f(x_k)$$

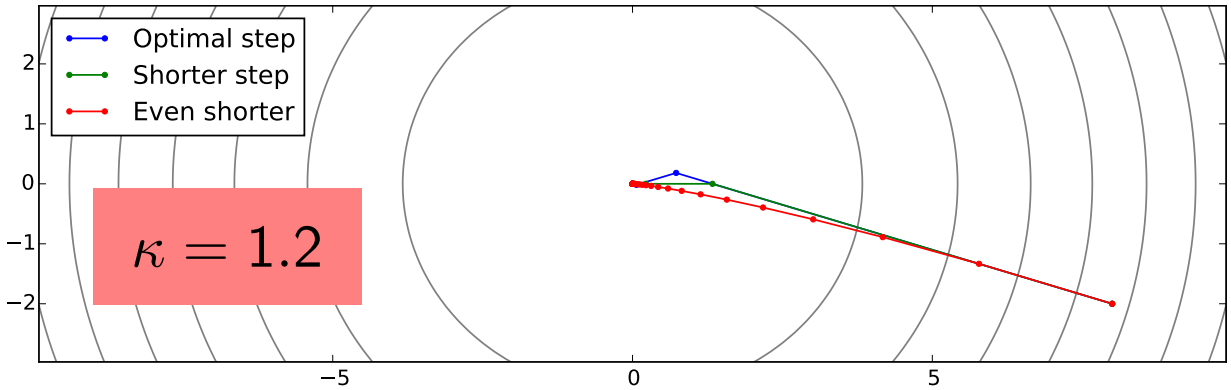
- $-\nabla f(x_k)$ points in the direction of local steepest decrease of the function. We will move in this direction.
- t_k is the stepsize. Many ways to choose it:
  - Pick a constant $t_k = t$
  - Pick a slowly decreasing stepsize, such as $t_k = 1/\sqrt{k}$
  - Exact line search: $t_k = \arg\min_t f(x_k - t\nabla f(x_k))$.
  - A heuristic method (most common in practice). Example: backtracking line search.

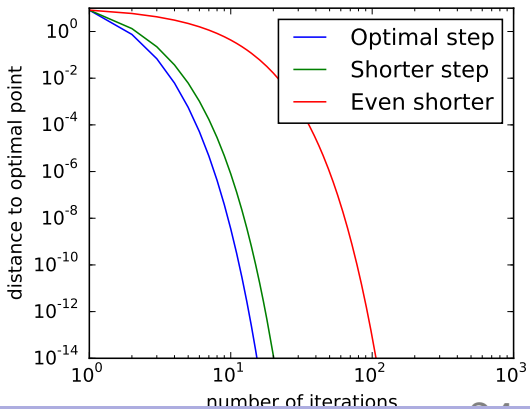
Gradient descent

We can gain insight into the effectiveness of a method by seeing how it perform on a quadratic: $f(x) = \frac{1}{2}x^T Qx$. The **condition number** $\kappa := \frac{\lambda_{\max}(Q)}{\lambda_{\min}(Q)}$ determines convergence.









Gradient descent

Advantages

- Simple to implement and cheap to execute.
- Can be easily adjusted.
- Robust in the presence of noise and uncertainty.

Disadvantages

- Convergence is slow.
- Sensitive to conditioning. Even rescaling a variable can have a substantial effect on performance!
- Not always easy to tune the stepsize.

Note: The idea of preconditioning (rescaling) before solving adds another layer of possible customizations and tradeoffs.

Other first-order methods

Accelerated methods

- Still a first-order method, but makes use of past iterates to accelerate convergence. Example: the Heavy-ball method:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k (x_k - x_{k-1})$$

  Other examples: Nesterov, Beck & Teboulle, others.
- Can achieve substantial improvement over gradient descent with only a moderate increase in computational cost
- Not as robust to noise as gradient descent, and can be more difficult to tune because there are more parameters.

Other first-order methods

Stochastic gradient descent

- Similar to gradient descent, but only evaluate some of the components of $\nabla f(x_k)$, chosen at random.
- Same pros and cons as gradient descent, but allows further tradeoff of speed vs computation.
- Industry standard for big-data problems like deep learning.

Nonlinear conjugate gradient

- Variant of the standard conjugate gradient algorithm for solving $Ax = b$, but adapted for use in general optimization.
- Requires more computation than accelerated methods.
- Converges exactly in a finite number of steps when applied to quadratic functions.

Newton's method

Basic idea: approximate the function as a quadratic, move directly to the minimum of that quadratic, and repeat.

- If we're at x_k, then by Taylor's theorem:

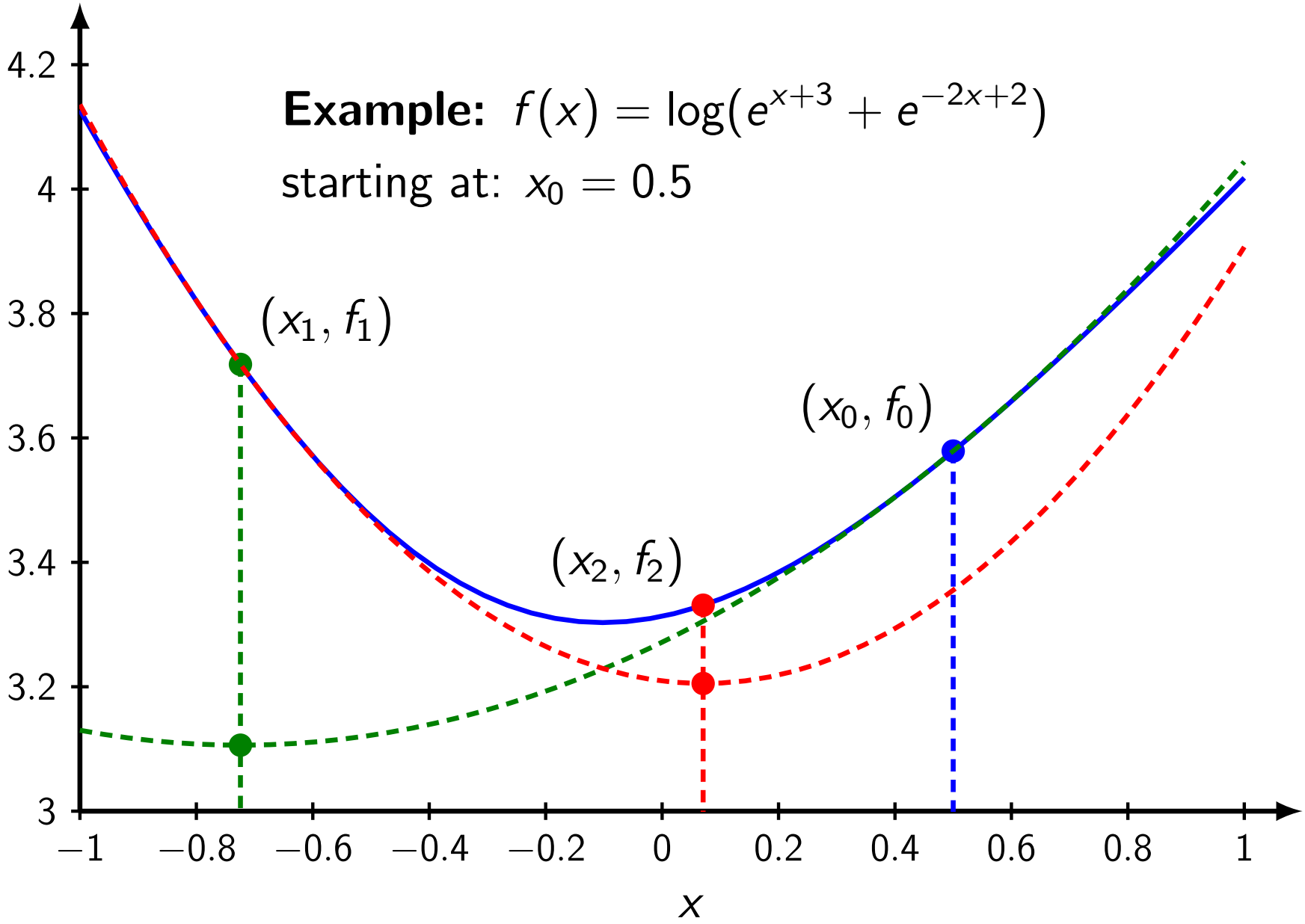
$$f(x) \approx f(x_k) + \nabla f(x_k)^{\mathsf{T}}(x - x_0) + \frac{1}{2}(x - x_k)^{\mathsf{T}} \nabla^2 f(x_k)(x - x_k)$$

- If $\nabla^2 f(x_k) \succ 0$, the minimum of the quadratic occurs at:

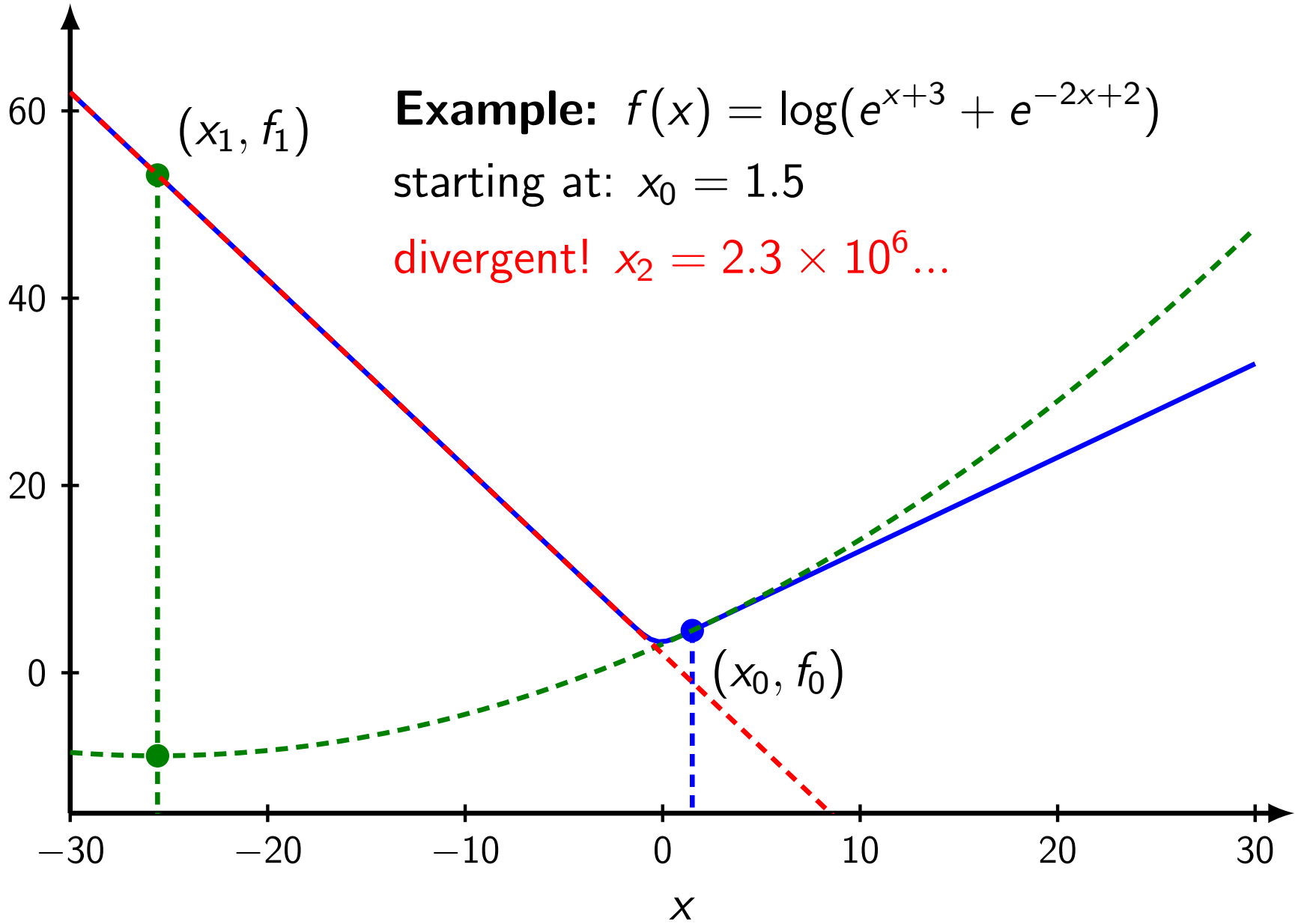
$$x_{k+1} := x_{\text{opt}} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k)$$

- Newton's method is a second-order method; it requires computing the Hessian (second derivatives).

Newton's method in 1D



Newton's method in 1D



Example: $f(x) = \log(e^{x+3} + e^{-2x+2})$

starting at: $x_0 = 1.5$

divergent! $x_2 = 2.3 \times 10^6$...

Newton's method

Advantages

- It's usually *very* fast. Converges to the exact optimum in one iteration if the objective is quadratic.
- It's scale-invariant. Convergence rate is not affected by any linear scaling or transformation of the variables.

Disadvantages

- If n is large, storing the Hessian (an $n \times n$ matrix) and computing $\nabla^2 f(x_k)^{-1} \nabla f(x_k)$ can be prohibitively expensive.
- If $\nabla^2 f(x_k) \not\succ 0$, Newton's method may converge to a local maximum or a saddle point.
- May fail to converge at all if we start too far from the optimal point.

Quasi-Newton methods

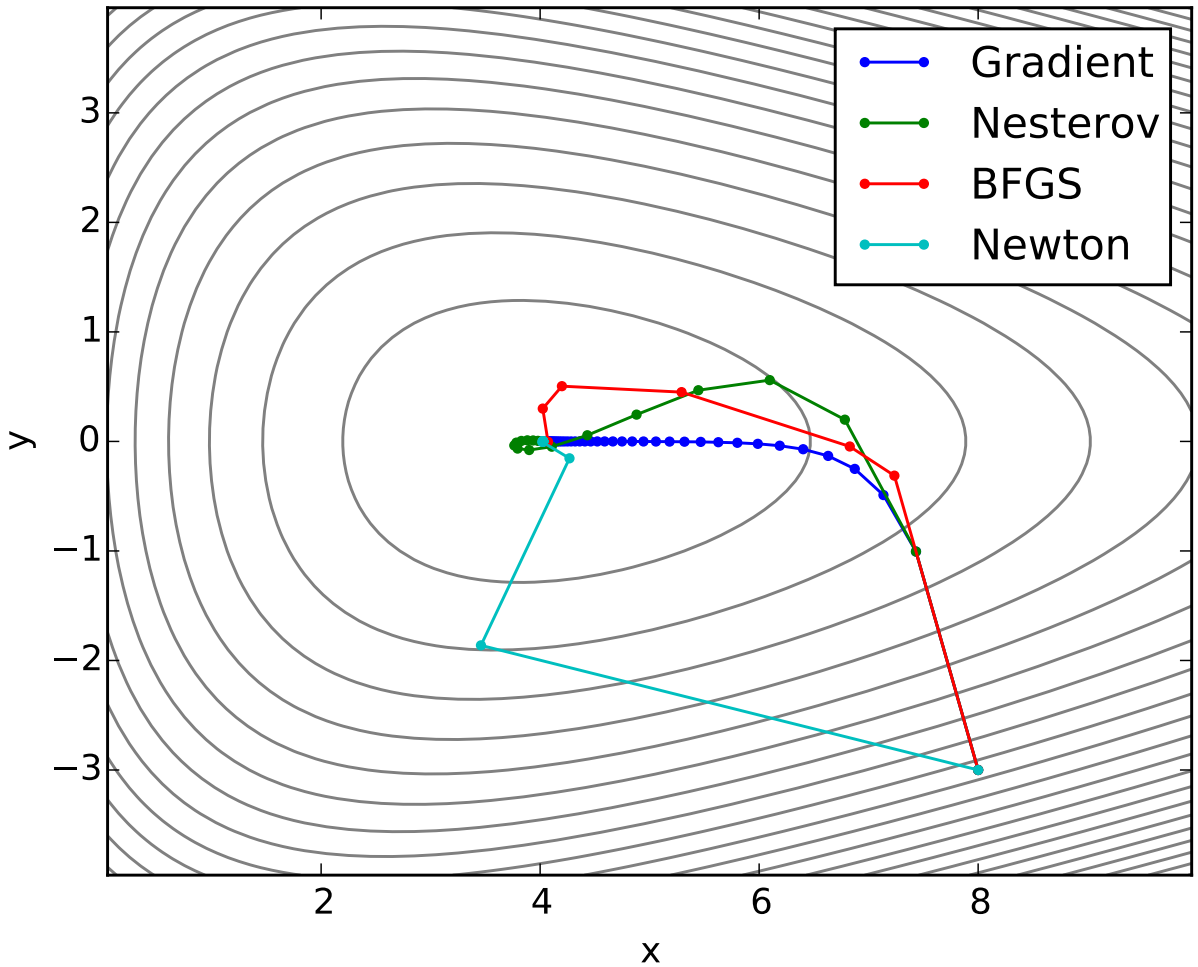
- An approximate Newton's methods that doesn't require computing the Hessian.
- Uses an approximation $H_k \approx \nabla^2 f(x_k)^{-1}$ that can be updated directly and is faster to compute than the full Hessian.

$$x_{k+1} = x_k - H_k \nabla f(x_k)$$
$$H_{k+1} = g(H_k, \nabla f(x_k), x_k)$$

- Several popular update schemes for H_k:
  - DFP (Davidon–Fletcher–Powell)
  - BFGS (Broyden–Fletcher–Goldfarb–Shanno)

Example

- $f(x, y) = e^{-(x-3)/2} + e^{(x+4y)/10} + e^{(x-4y)/10}$
- Function is smooth, with a single minimum near $(4.03, 0)$.



Example code for nonlinear codes

- The above examples are demonstrated in `37iterative.ipynb`.
- It includes a plot comparing the iterations to convergence of each of the above algorithms.
- Illustrates the complexity vs performance tradeoff.
- Nesterov's method doesnt always converge uniformly.

GAMS NLP Solvers

Local Solvers

- CONOPT (4): Generalized Reduced Gradient
- IPOPT (H): Interior Point, open source
- KNITRO: Many, but basically Interior Point
- SNOPT: Sequential Quadratic Programming
- MOSEK: Interior Point
- MINOS: Reduce Gradient/Projected Lagrangian
- PATHNLP: KKT conditions of NLP

Maybe Global Solvers

- BARON: branch and reduce
- Antigone, LindoGlobal, SBB: branch and cut
- SCIP, COUENNE, DICOPT: heuristic and reformulation

Solving Problems with the NEOS Server

- `https://www.neos-server.org`
- Useful when you are on a local system that doesn't have GAMS, provided you don't have to transfer massive data files.
- Try submitting `gandhi2.gms` with `option mip=xpress` (solver is FICO-XPRESS on NEOS).
- Browse the list of NEOS Servers to see what's available and which ones accept GAMS input.