EECE 5550
Mobile Robotics

Lecture 12: Optimization

Derya Aksaray

Assistant Professor

Department of Electrical and Computer Engineering

Northeastern University

Recap

Last time:

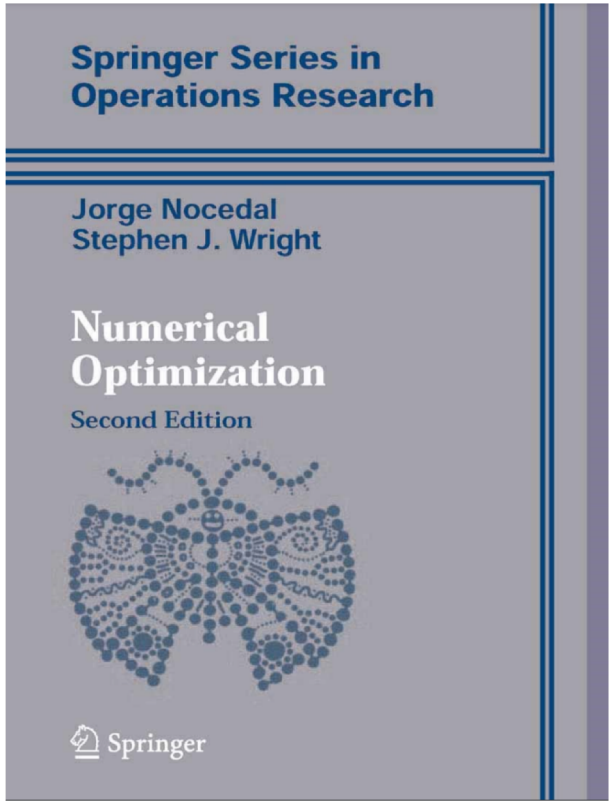
- Factor graphs
- Simultaneous localization and mapping (SLAM)
- Maximum likelihood estimation

This time:

- Basic theory of optimization

  (i.e. how to *actually do* MLE)

References



Convex Optimization
Stephen Boyd & Lieven Vandenberghe

https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

Typical optimization problems

- Minimize the estimation error in a state of a dynamical system
- Maximize the probability of making a correct decision
- Minimize the tracking error in a controlled system
- **Minimize** the **time** or **energy** required to achieve an objective

Estimation

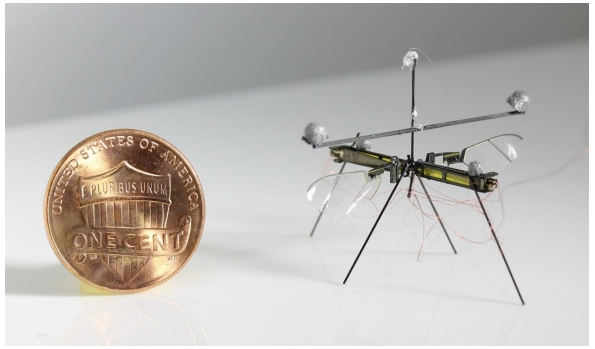
Planning

Control

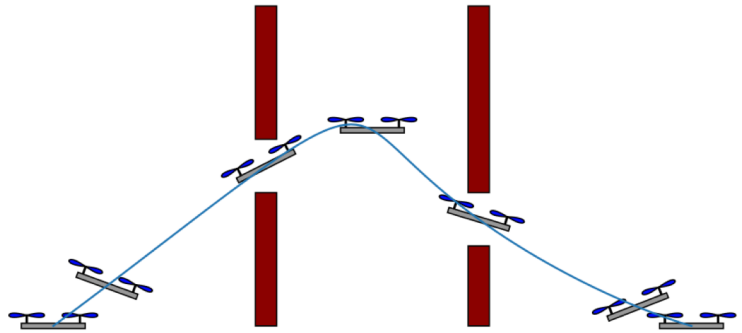
Optimization implies choice

- Choice of best strategy
- Choice of best design parameters
- Choice of best control input
- Choice of best estimate

- Objective function (J)
- Constraints (equality, inequality)



RoboFly
https://arxiv.org/pdf/2001.02320.pdf



Taxonomy of optimization problems

Static vs. Dynamic

- **Static:** $J^* = J(x^*, u^*)$ where x^* is the optimal state and u^* is the optimal control
  - x^* and u^* do not change over time
  - Functional minimization (maximization), parameter optimization
- **Dynamic:** $J^* = J(x^*(t), u^*(t))$
  - x^* and u^* vary over time
  - Optimal trajectory, optimal feedback control

Deterministic vs. Stochastic

- **Deterministic:** $J^* = J(x^*, u^*)$
  - System model, parameters, initial conditions, and disturbances are known without error
- **Stochastic:** optimal cost = $E[J(x^*, u^*)]$
  - Uncertainty in system model, parameters, initial conditions, disturbances, and cost function

Necessary condition for static optimality

Let x ∈ $\mathbb{R}$ and the cost function is $J(x)$.

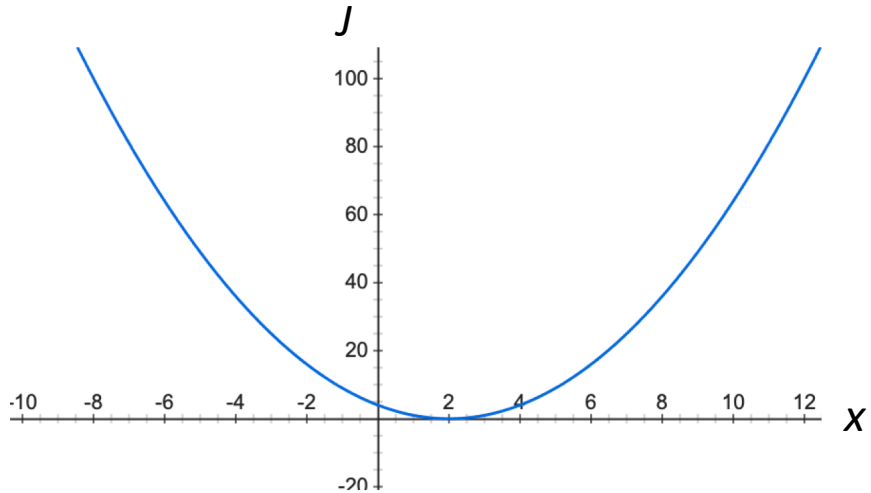
$$J^* = \min_{x \in \mathbb{R}} J(x)$$

$$x^* = \underset{x \in \mathbb{R}}{\operatorname{argmin}} J(x)$$

The slope is zero at the optimum point.

$$\left.\frac{dJ}{dx}\right|_{x=x^*} = 0$$

If x ∈ $\mathbb{R}^n$, then $\left.\frac{dJ}{dx}\right|_{x=x^*} = \left[\frac{\partial J}{\partial x_1} \quad \frac{\partial J}{\partial x_2} \quad \dots \quad \frac{\partial J}{\partial x_n}\right]\Big|_{x=x^*} = 0$



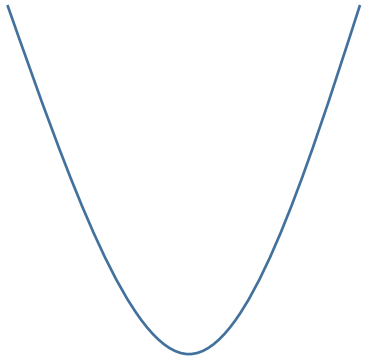
$$J = (x-2)^2$$

$\frac{dJ}{dx} = 2(x-2) = 0$ when $x^* = 2$

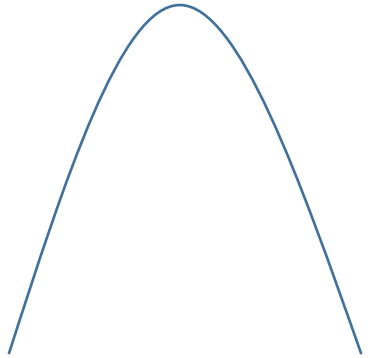
All slopes are concurrently zero at the optimum.

Is the slope zero only at the optimum?

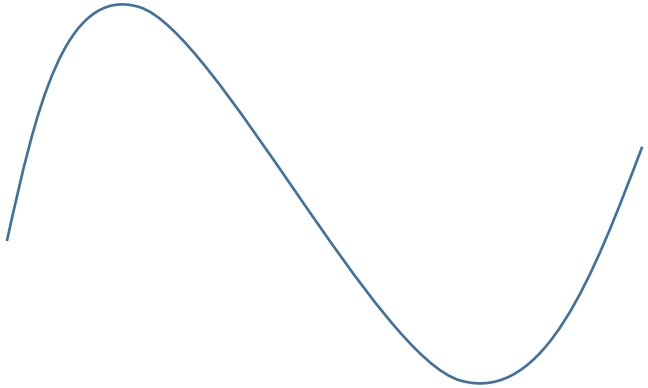
Minimum



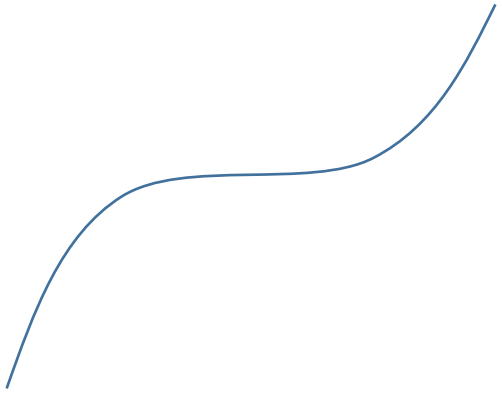
Maximum



Either



Inflection point



Sufficient condition for static optimality

Maximum

Satisfy necessary condition **+**

$$\left.\frac{d^2J}{dx^2}\right|_{x=x^*} < 0$$

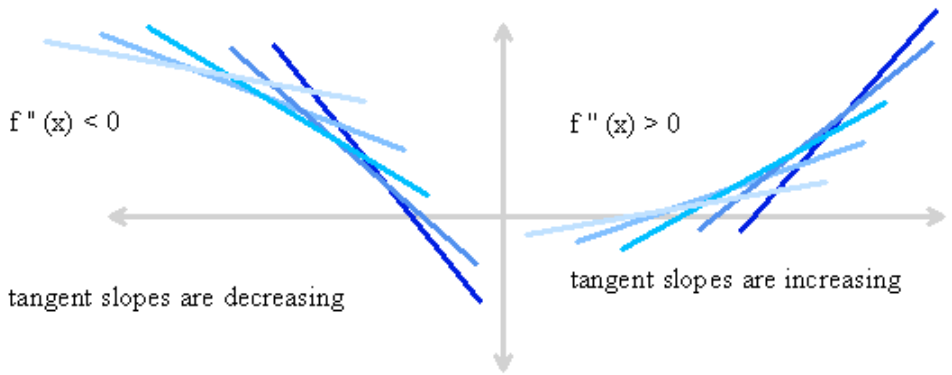
i.e., curvature is negative at optimum.

Minimum

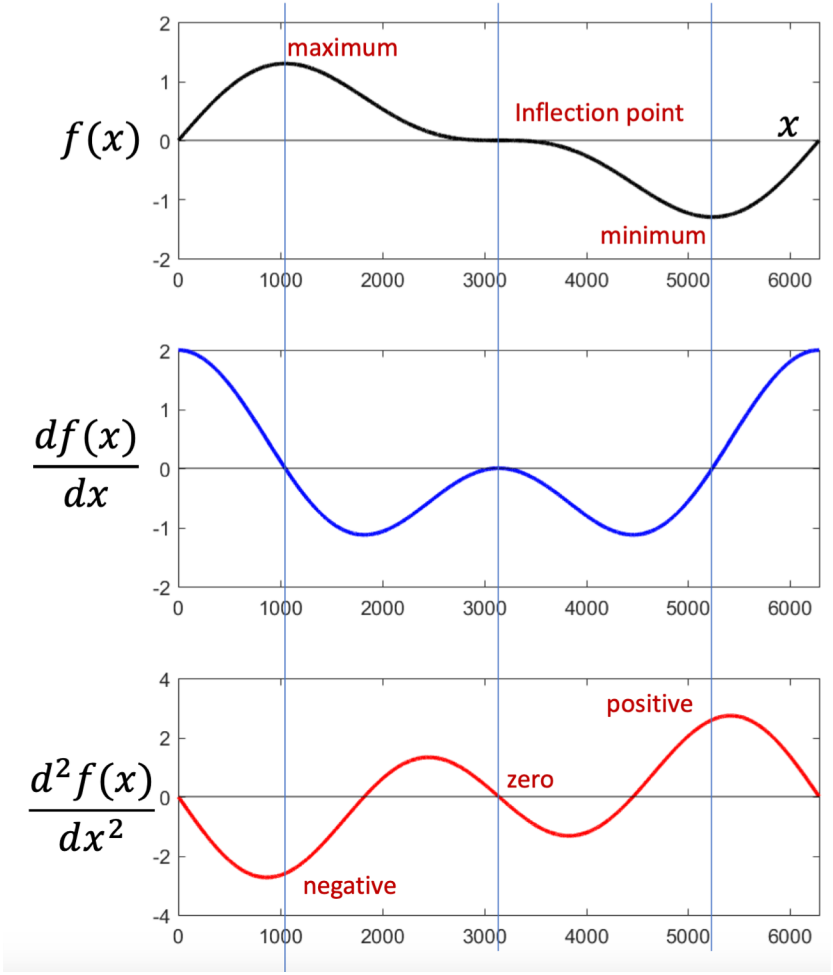
Satisfy necessary condition **+**

$$\left.\frac{d^2J}{dx^2}\right|_{x=x^*} > 0$$

i.e., curvature is positive at optimum.



First and second derivatives of f(x)



All locations with zero derivative are *critical* points

The *second* derivative is

- > 0 at minima
- < 0 at maxima
- Zero at inflection points

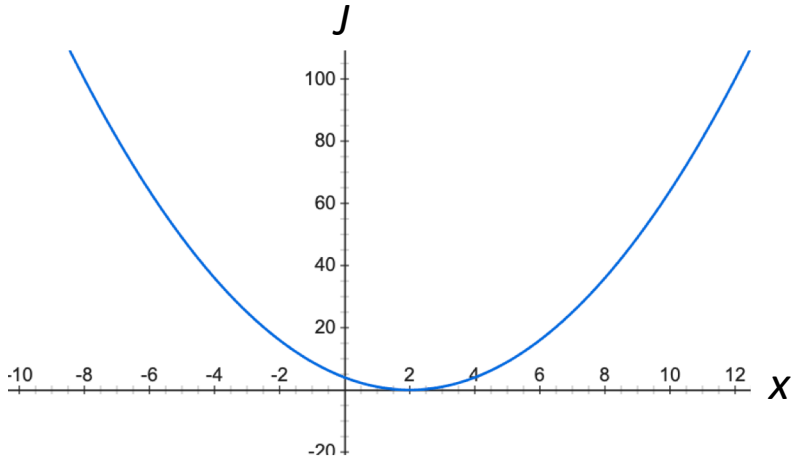
Sufficient condition for static optimality

Maximum

Satisfy necessary condition **+**

$$\left.\frac{d^2J}{dx^2}\right|_{x=x^*} < 0$$

i.e., curvature is negative at optimum.



Minimum

Satisfy necessary condition **+**

$$\left.\frac{d^2J}{dx^2}\right|_{x=x^*} > 0$$

i.e., curvature is positive at optimum.

$$J = (x-2)^2$$

$$\frac{dJ}{dx} = 2(x-2) = 0 \quad \text{when } x^* = 2$$

$$\frac{d^2J}{dx^2} = 2 > 0$$

Sufficient condition for static optimality

Maximum

Satisfy necessary condition **+**

$$\left.\frac{d^2J}{dx^2}\right|_{x=x^*} < 0$$

i.e., curvature is negative at optimum.

Minimum

Satisfy necessary condition **+**

$$\left.\frac{d^2J}{dx^2}\right|_{x=x^*} > 0$$

i.e., curvature is positive at optimum.

n-dimensional case:
Satisfy necessary condition
+

$$\left.\frac{\partial^2 J}{\partial x^2}\right|_{x=x^*} = \begin{bmatrix} \frac{\partial^2 J}{\partial x_1^2} & \frac{\partial^2 J}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 J}{\partial x_1 \partial x_n} \\ \frac{\partial^2 J}{\partial x_2 \partial x_1} & \frac{\partial^2 J}{\partial x_2^2} & \cdots & \frac{\partial^2 J}{\partial x_2 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 J}{\partial x_n \partial x_1} & \frac{\partial^2 J}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 J}{\partial x_n^2} \end{bmatrix} > 0 \text{ (for minimization)}$$

Positive definite

Hessian matrix

Constrained optimization problems

$$\min_x J(x)$$

Subject to $g_i(x) = 0$ for $i = 1, \dots n$ equality constraints

$h_j(x) \leq 0$ for $j = 1, \dots m$ inequality constraints

Optimization with equality constraints

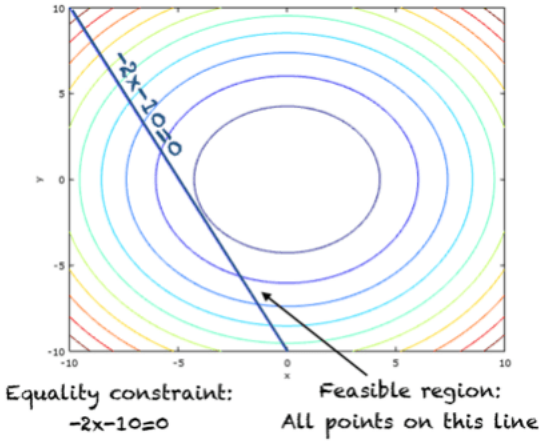
- **Lagrange multipliers:** Augment the cost function

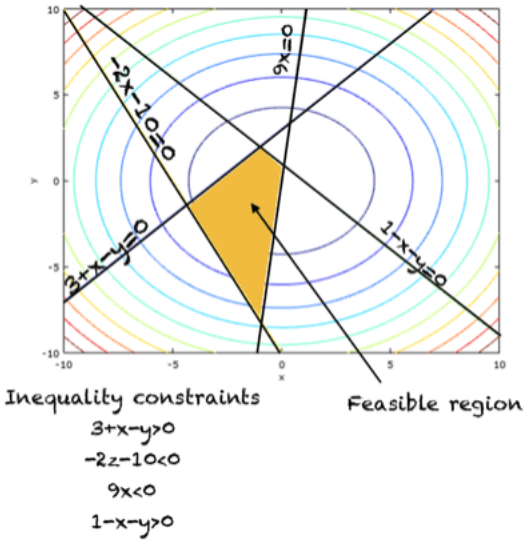
$$J'(x) = J(x) + \lambda^T g(x)$$

$$\frac{\partial J}{\partial x} = 0; \frac{\partial J}{\partial \lambda} = 0$$

Optimization with inequality constraints:

- **Karush–Kuhn–Tucker theorem**



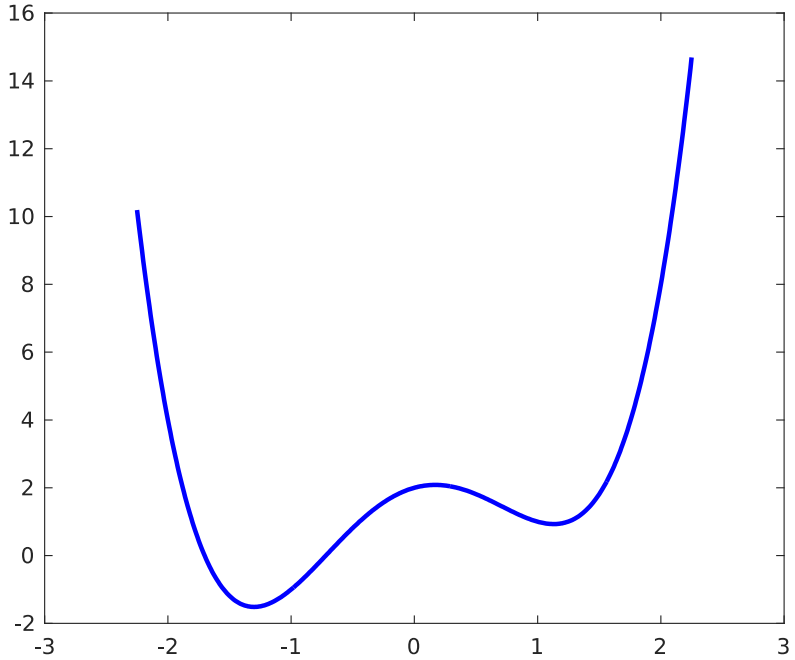


https://machinelearningmastery.com/a-gentle-introduction-to-optimization-mathematical-programming/

Numerical optimization

Given: $f: \mathbb{R}^n \to \mathbb{R}$, we want to

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



Problem: We have *no idea* how to actually do this …

- If f is too complicated, taking the derivatives expensive
- Finding the zeros of a system of equations challenging

Main idea: Let's *approximate* f with a simple *model function* m, and use that to search for a minimizer of f.

Optimization Meta-Algorithm

Given: A function $f: \mathbb{R}^n \to \mathbb{R}$ and an initial guess $x_0 \in \mathbb{R}^n$ for a minimizer

Iterate:

- Construct a *model* $m_i(h) \approx f(x_i + h)$ of f near x_i.
- Use m_i to search for a *descent direction h* $(f(x + h) < f(x))$
- Update $x_{i+1} \leftarrow x_i + h$

until convergence

A first example

Let's consider applying the Basic Algorithm to minimize

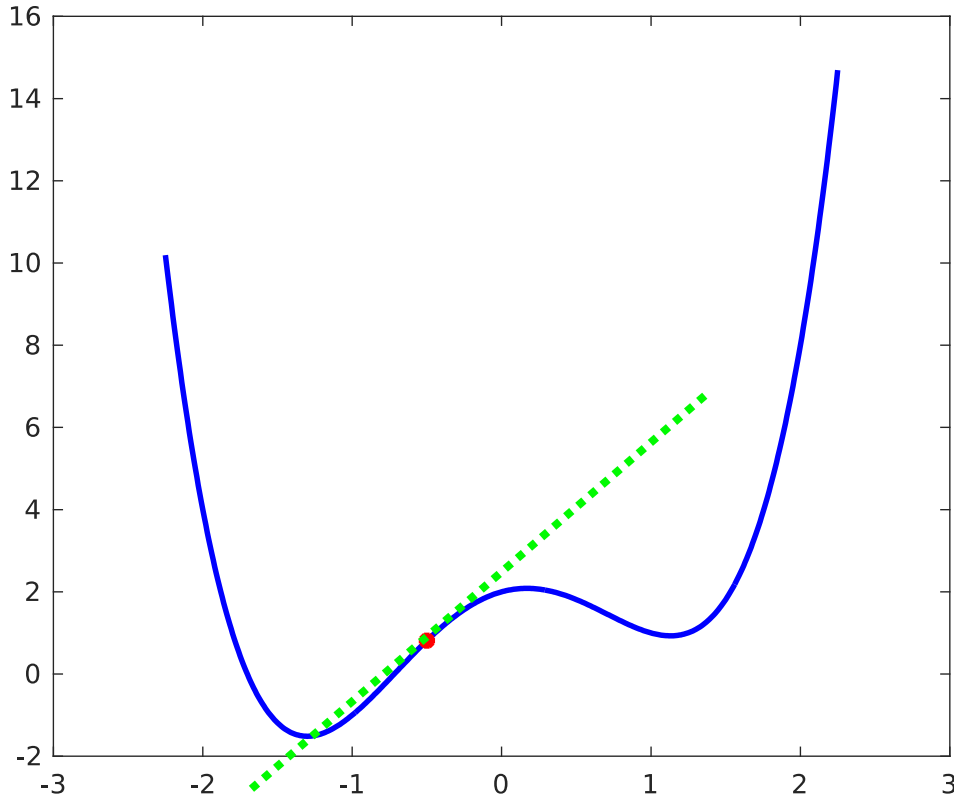
$$f(x) = x^4 - 3x^2 + x + 2$$

starting at $x_0 = -\frac{1}{2}$.

Q: How can we *approximate* (*model*) f near x_0?

A: Let's try linearizing! Take

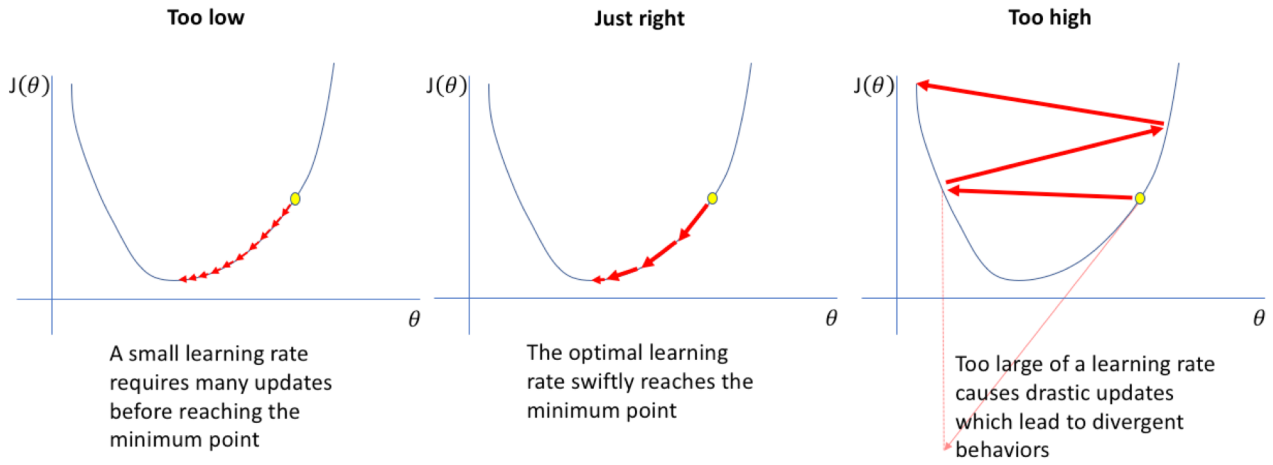
$$m_0(h) \triangleq f(x_0) + f'(x_0)h$$

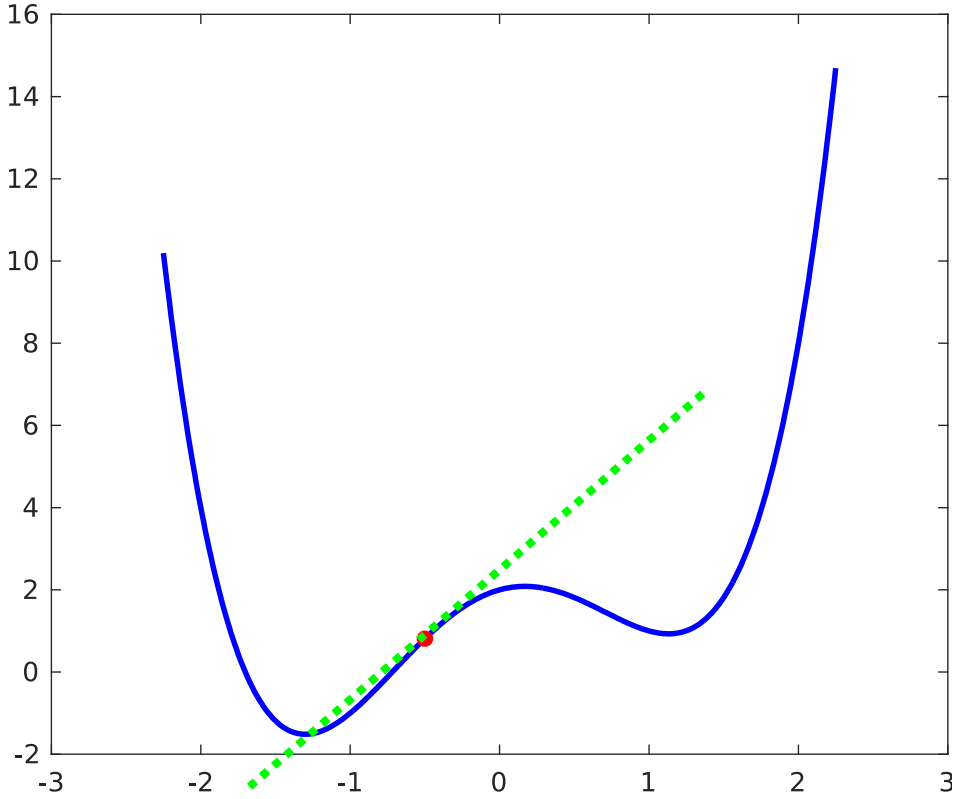


Gradient descent

How to choose how much we will move over the gradient?

- Fixed rate





- Adaptive rate

Gradient descent

Given:

- A function $f: \mathbb{R}^n \to \mathbb{R}$
- An initial guess $x_0 \in \mathbb{R}^n$ for a minimizer
- Sufficient decrease parameter $c \in (0,1)$, stepsize shrinkage parameter $\tau \in (0,1)$
- Gradient tolerance $\epsilon > 0$

Iterate:

- Compute search direction $p = -\nabla f(x_i)$ at x_i
- Set initial step size $\alpha = 1$
- *Backtracking line search*: Update $\alpha \leftarrow \tau\alpha$ until the *Armijo-Goldstein sufficient decrease condition*:

$$f(x_i + \alpha p) < f(x_i) - c\alpha\|p\|^2$$

  is satisfied
- Update $x_{i+1} \leftarrow x_i + \alpha p$

until $\|\nabla f(x_i)\| < \epsilon$

Convergence to a local optimum

Exercise: Minimizing a quadratic

Try minimizing the quadratic:

$$f(x, y) = x^2 - xy + \kappa y^2$$

using gradient descent, starting at $x_0 = (1,1)$ and using $c, \tau = \frac{1}{2}$ and $\epsilon = 10^{-3}$, for a few different values of κ, say

$$\kappa \in \{1, 10, 100, 1000\}$$

Q: If you plot function value $f(x_i)$ vs. iteration number *i*, what do you notice?

Gradient Descent

Given:

- A function $f: \mathbb{R}^n \to \mathbb{R}$
- An initial guess $x_0 \in \mathbb{R}^n$ for a minimizer
- Sufficient decrease parameter $c \in (0,1)$, stepsize shrinkage parameter $\tau \in (0,1)$
- Gradient tolerance $\epsilon > 0$

Iterate:

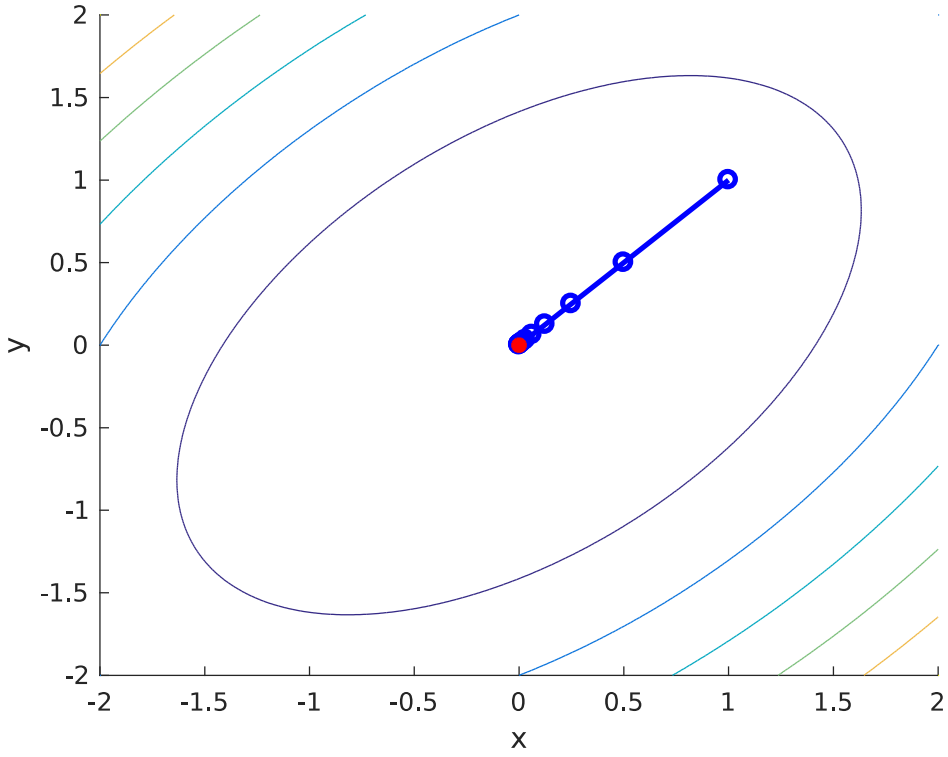
- Compute search direction $p = -\nabla f(x_i)$ at x_i
- Set initial stepsize $\alpha = 1$
- Line search: update $\alpha \leftarrow \tau\alpha$ until

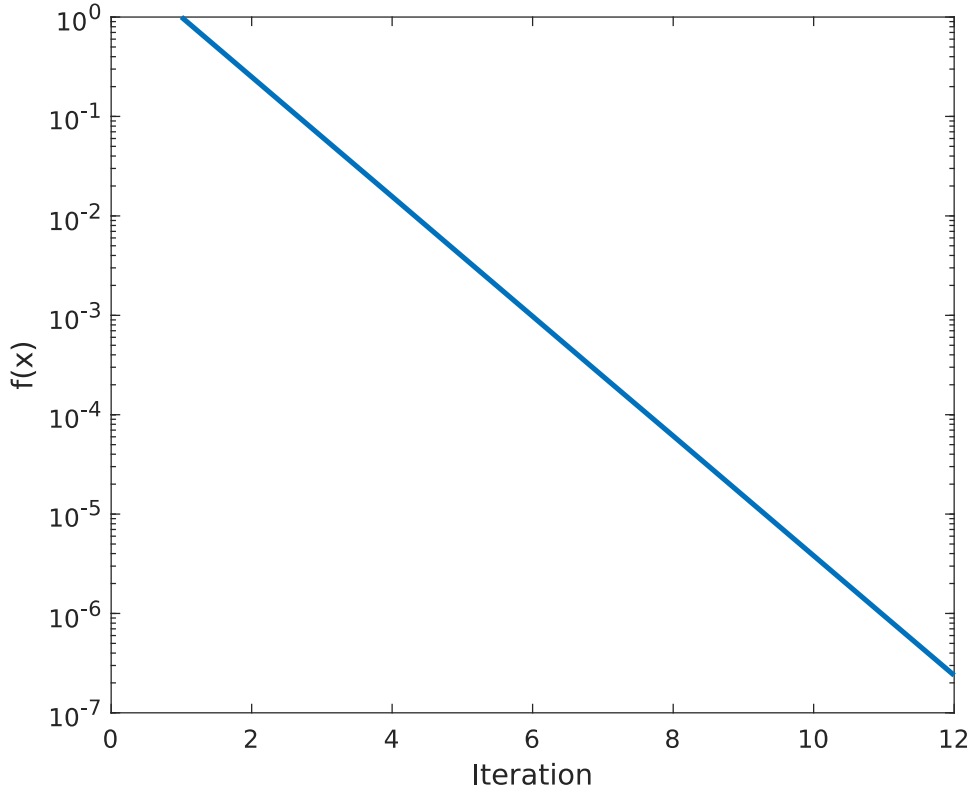
$$f(x_i + \alpha p) < f(x_i) - c\alpha\|p\|^2$$

- Update $x_{i+1} \leftarrow x_i + \alpha p$

until $\|\nabla f(x_i)\| < \epsilon$

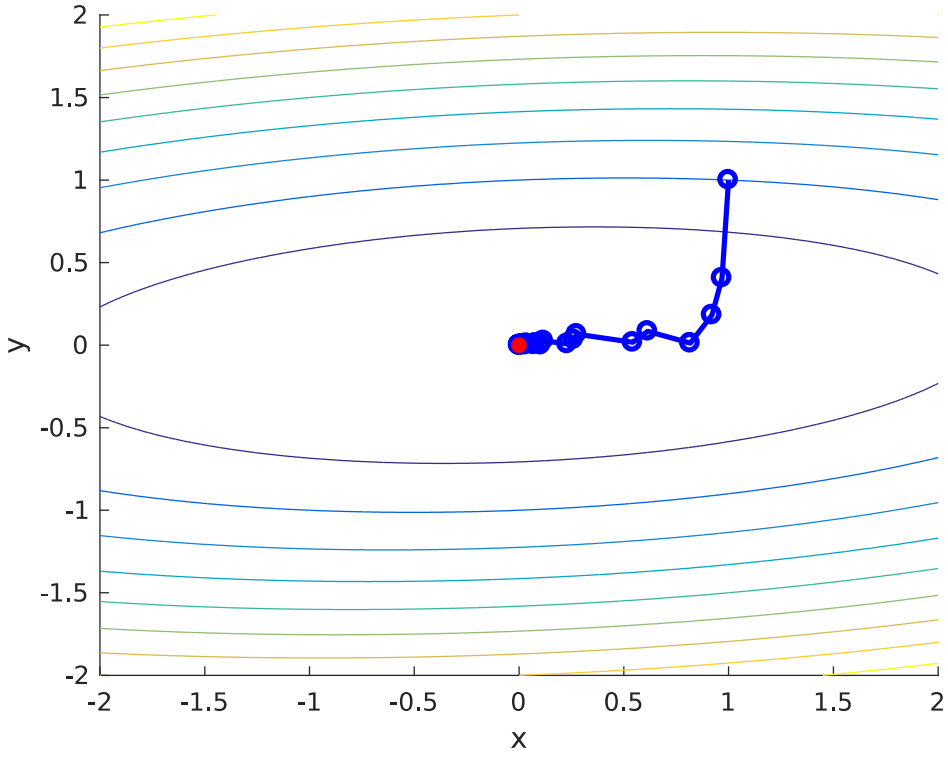
Exercise: Minimizing a quadratic

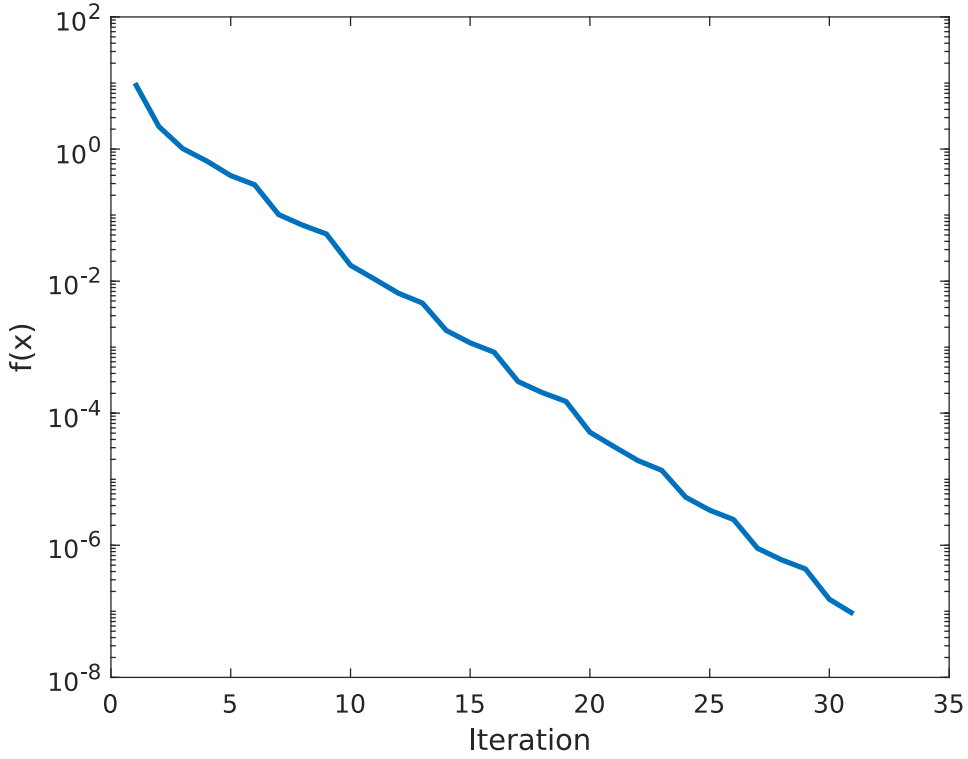




$\kappa = 1$

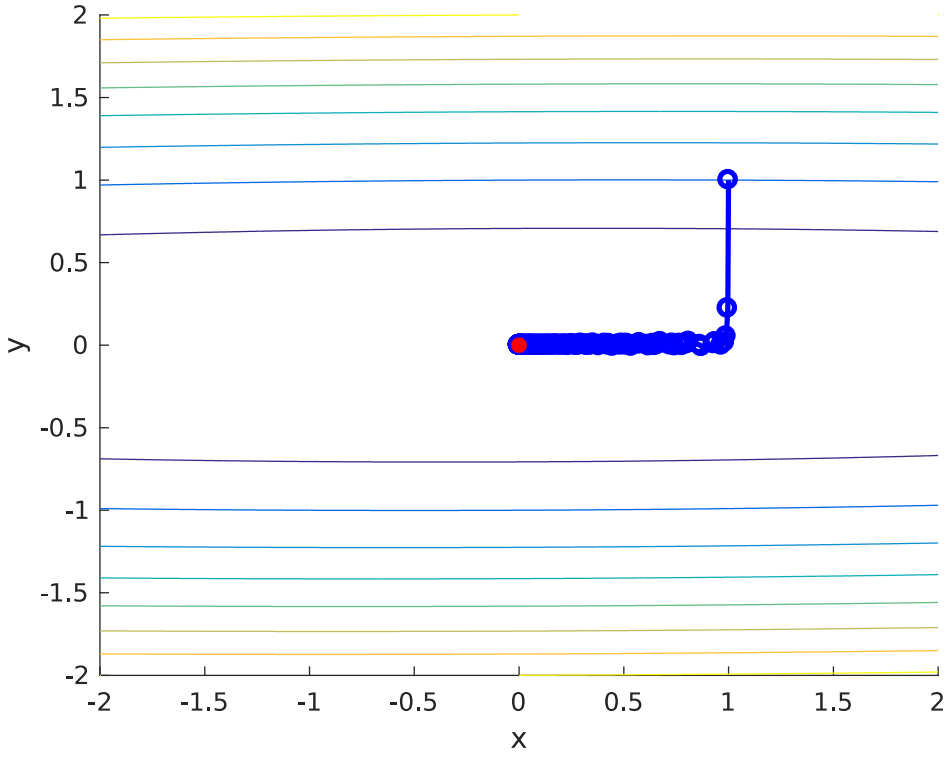
Exercise: Minimizing a quadratic

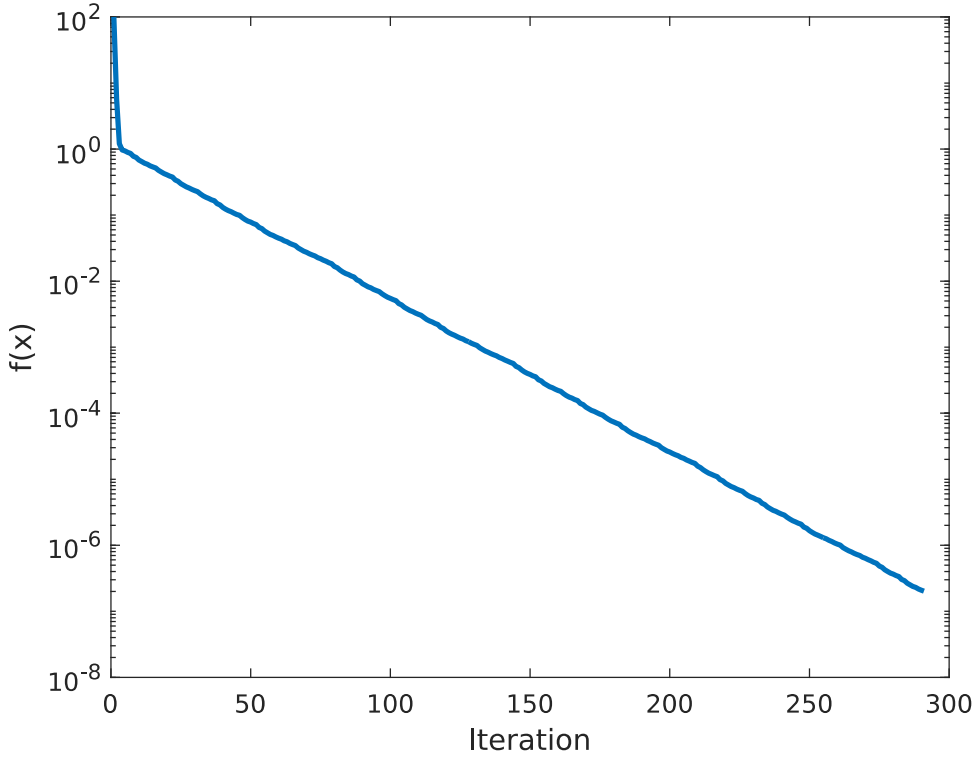




$\kappa = 10$

Exercise: Minimizing a quadratic

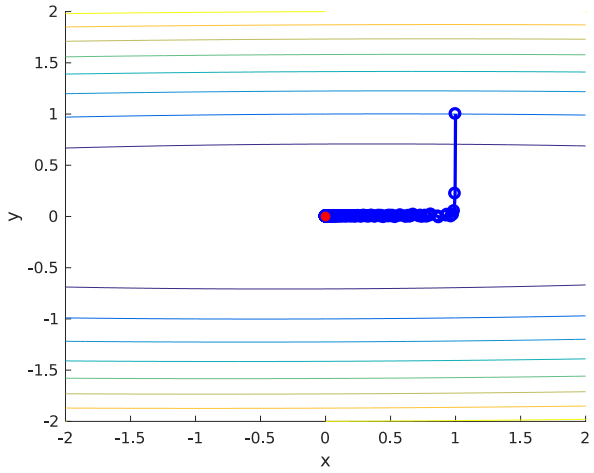




$\kappa = 100$

The problem of conditioning

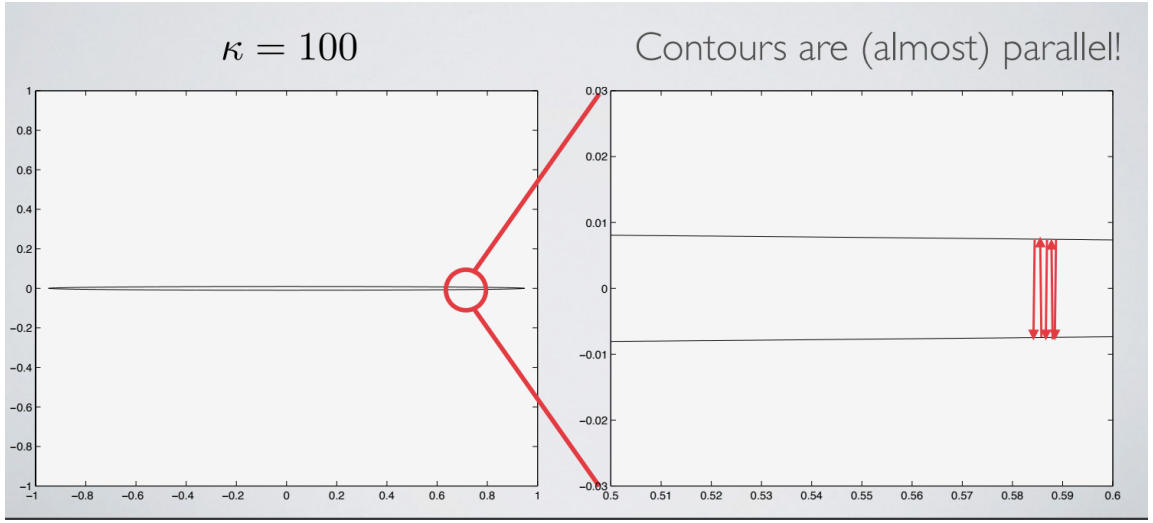
Gradient descent doesn't perform well when f is *poorly conditioned* (has "stretched" contours).



Q: How can we improve our local model:

$$m_i(h) = f(x_i) + \nabla f(x_i)^T h$$

so that it handles curvature better?



Second-order methods

Let's try adding in curvature information using a *second-order* model for f:

$$m_i(h) = f(x_i) + \nabla f(x_i)^T h + \frac{1}{2} h^T \nabla^2 f(x) h$$

NB: If $\nabla^2 f(x) \succ 0$, then $m_i(h)$ has a *unique minimizer*:

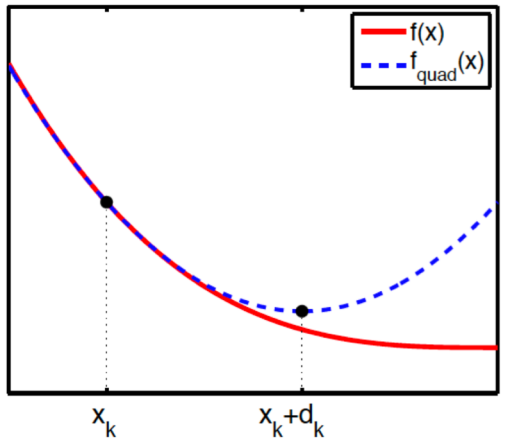
$$h_N = -\left(\nabla^2 f(x_0)\right)^{-1} \nabla f(x_0)$$

In that case, using the update rule:

$$x_{i+1} \leftarrow x_i + h_N$$

gives ***Newton's method.***

Take step to local minima of second-order Taylor approximation of loss function



Newton's method

Given:

- A function $f: \mathbb{R}^n \to \mathbb{R}$
- An initial guess $x_0 \in \mathbb{R}^n$ for a minimizer
- Gradient tolerance $\epsilon > 0$

Iterate:

- Compute gradient $\nabla f(x_i)$ and Hessian $\nabla^2 f(x_i)$
- Compute Newton step:

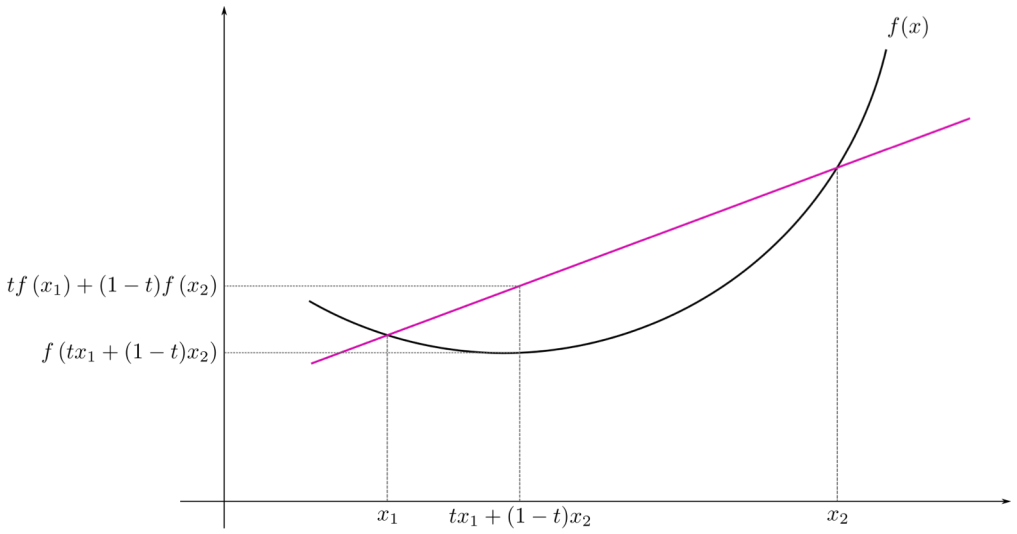
$$h_N = -\left(\nabla^2 f(x_0)\right)^{-1} \nabla f(x_0)$$

- Update $x_{i+1} \leftarrow x_i + h_N$
- **until** $\|\nabla f(x_i)\| < \epsilon$

It requires less iteration, but each iteration is more complex.

Convexity

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$



Newton's method

Given:

- A function $f: \mathbb{R}^n \to \mathbb{R}$
- An initial guess $x_0 \in \mathbb{R}^n$ for a minimizer
- Gradient tolerance $\epsilon > 0$

Iterate:

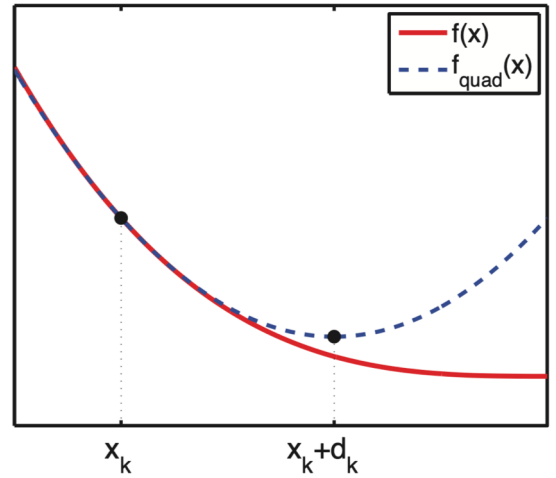
- Compute gradient $\nabla f(x_i)$ and Hessian $\nabla^2 f(x_i)$
- Compute Newton step:

$$h_N = -\left(\nabla^2 f(x_0)\right)^{-1} \nabla f(x_0)$$

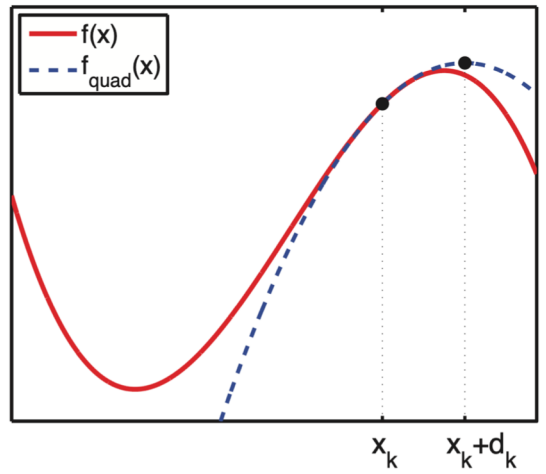
- Update $x_{i+1} \leftarrow x_i + h_N$
- **until** $\|\nabla f(x_i)\| < \epsilon$

It requires less iteration, but each iteration is more complex.

Convex function



Non-convex function



Quasi-Newton methods

Newton's method is ***fast***! (It has a *quadratic* convergence rate)

But:

- h_N is only guaranteed to be a descent direction if $\nabla^2 f(x_i) \succ 0$
- Computing exact Hessians can be expensive!

Quasi-Newton methods: Use a *positive-definite approximate Hessian* B_i in the model function:

$$m_i(h) = f(x_i) + \nabla f(x_i)^T h + \frac{1}{2} h^T B_i h$$

$\Rightarrow m_i(h)$ *always* has a unique minimizer:

$$h_{QN} = -B_i^{-1} \nabla f(x_i)$$

$\Rightarrow h_{QN}$ is *always* a descent direction!

Quasi-Newton method with line search

Given:

- A function $f: \mathbb{R}^n \to \mathbb{R}$
- An initial guess $x_0 \in \mathbb{R}^n$ for a minimizer
- Sufficient decrease parameter $c \in (0,1)$, stepsize shrinkage parameter $\tau \in (0,1)$
- Gradient tolerance $\epsilon > 0$

Iterate:

- Compute gradient $g_i = \nabla f(x_i)$ and positive-definite Hessian approximation B_i at x_i
- Compute quasi-Newton step:

$$h_{QN} = -B_i^{-1} g_i$$

- Set initial stepsize $\alpha = 1$
- *Backtracking line search*: Update $\alpha \leftarrow \tau\alpha$ until the *Armijo-Goldstein sufficient decrease condition*:

$$f\left(x_i + \alpha h_{QN}\right) < f(x_i) + c\alpha g_i^T h_{QN}$$

  is satisfied
- Update $x_{i+1} \leftarrow x_i + \alpha h_{QN}$

until $\|g_i\| < \epsilon$

Quasi-Newton methods (cont'd)

Different choices of B_i give different QN algorithms

$\Rightarrow$ Can trade off *accuracy* of B_i with *computational cost*

LOTS of possibilities here!

- Gauss-Newton
- Levenberg-Marquardt
- (L-)BFGS
- Broyden
- etc …

Special case: The Gauss-Newton method

***Nonlinear least-squares*:**

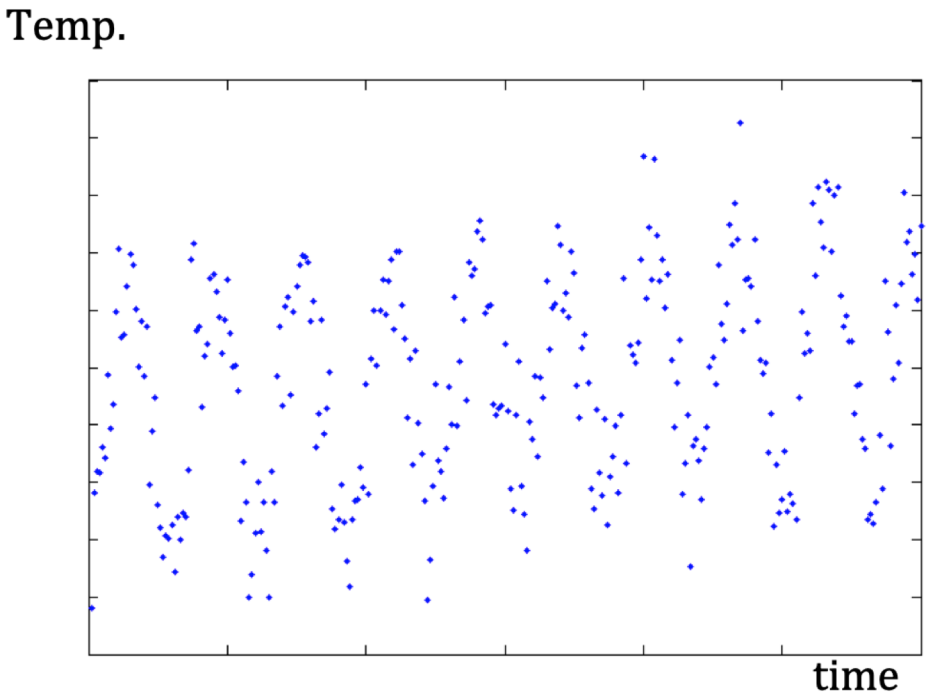
Suppose that you consider a parametric model to represent your data:

$$T(t) = \mathrm{a}\,\sin(\omega t + \phi) + bt$$

The goal is finding the parameters: a, ω, ϕ, b

Let's say you have n data points (t_i, T_i). Then:
The goal is to minimize:

$$f(a, \omega, \phi, b) = \sum_{i=1}^{n} (T_i - \mathrm{a}\,\sin(\omega t + \phi) + bt)^2 = \sum_{i=1}^{n} r_i(x)^2$$



Special case: The Gauss-Newton method

A quasi-Newton algorithm for minimizing a *nonlinear least-squares objective*:

$$f(x) = \|r(x)\|^2 = \sum_{i=1}^{n} r_i(x)^2$$

where r: $\mathbb{R}^m \to \mathbb{R}^n$ is a *vector-valued residual function*.

Let $J(x_i)$ be the $n \times m$ Jacobian of *r, i.e.,* $J(x_i) \triangleq \frac{\partial r}{\partial x}(x_i)$.

Then:

$$g_i = 2J(x_i)^T r(x_i), \qquad B_i = 2J(x_i)^T J(x_i)$$

$$h_{QN} = -B_i^{-1} g_i \quad \Rightarrow \quad x_{i+1} \leftarrow x_i + h_{QN}$$

$$x_{i+1} = x_i - \left(2J(x_i)^T J(x_i)\right)^{-1} 2J(x_i)^T r(x_i)$$

A word on linear algebra

The dominant cost (memory + time) in a QN method is *linear algebra*:

- Constructing the Hessian approximation B_i
- Solving the linear system:

$$B_i h_{QN} = -g_i$$

$\Rightarrow$ Fast/robust linear algebra is ***essential*** for efficient QN methods

- Take advantage of sparsity in B_i!
- **NEVER, NEVER, NEVER INVERT B_i directly!!!**
  - It's incredibly expensive, and unnecessary
  - **Use instead** [cf. Golub & Van Loan's *Matrix Computations*]:
    - *Matrix factorizations:* QR, Cholesky, LDL^T
    - *Iterative linear methods:* conjugate gradient

Optimization methods: Cheat sheet

First-order methods

Use only gradient information

- **Pro:** Local model is inexpensive
- **Con:** Slow (linear) convergence rate

Canonical example: Gradient descent

Best for:

- Moderate accuracy
- Very large problems

Second-order methods

Use (some) 2nd-order information

- **Pro:** Fast (superlinear) convergence
- **Con:** Local model can be expensive

Canonical example: Newton's method

Best for:

- High accuracy
- Small to moderately large problems

Optimization on Manifolds

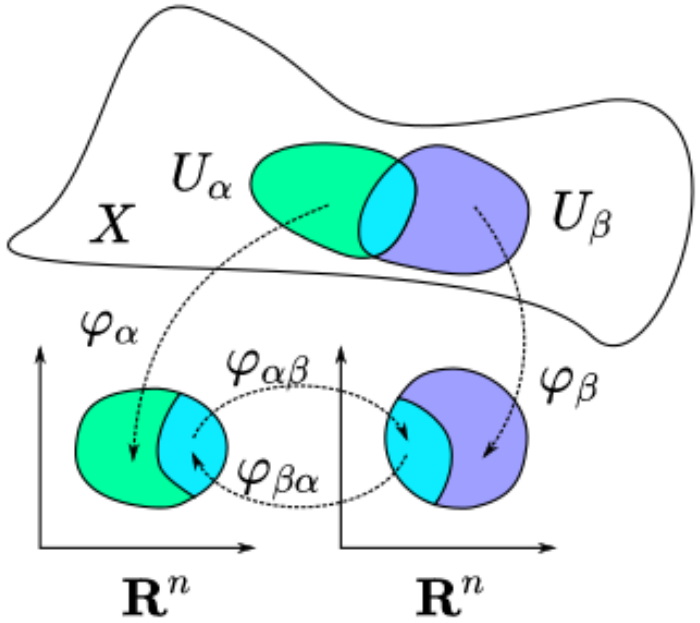
Main idea (recap): Search for a descent direction *h* using a *local model* *m* of the objective *f*:

$$m_i(h) = f(x_i) + \nabla f(x_i)^T h + \frac{1}{2} h^T B_i h$$

NB: model *m* is built using (approximate) derivative information ($B_i \approx \nabla^2 f(x_i)$)

Recall:

- **Key point:** Derivatives are *local*: df_x only depends upon *f*'s behavior in an infinitesimally small open set around *x*.
- Smooth manifolds are spaces in which every point *x* has an open set U that is diffeomorphic to an open set in $\mathbb{R}^n$.



⇒We can apply exactly the same approach to optimize functions on smooth manifolds!

Quasi-Newton optimization on $\mathbb{R}^n$

Iterate:

1. Compute gradient $g_i = \nabla f(x_i)$ and positive-definite Hessian approximation $B_i \approx \nabla^2 f(x_i)$ at x_i
2. Compute quasi-Newton step:

$$h_{QN} = -B_i^{-1} g_i$$

3. Set initial stepsize $\alpha = 1$
4. *Backtracking line search*: Update $\alpha \leftarrow \tau\alpha$ until the *Armijo-Goldstein sufficient decrease condition*:

$$f\left(x_i + \alpha h_{QN}\right) < f(x_i) + c\alpha g_i^T h_{QN}$$

is satisfied

5. Update $x_{i+1} \leftarrow x_i + \alpha h_{QN}$

until $\|g_i\| < \epsilon$

Quasi-Newton optimization on a manifold *M*

Iterate:

1. Compute gradient $g_i = \nabla f(x_i)$ and positive-definite Hessian approximation $B_i \approx \nabla^2 f(x_i)$ at x_i
2. Compute quasi-Newton step:

$$h_{QN} = -B_i^{-1} g_i$$

3. Set initial stepsize $\alpha = 1$
4. *Backtracking line search*: Update $\alpha \leftarrow \tau\alpha$ until the *Armijo-Goldstein sufficient decrease condition*:

$$f\left(Retr_{x_i}(\alpha h_{QN})\right) < f(x_i) + c\alpha g_i^T h_{QN}$$

is satisfied

5. Update $x_{i+1} \leftarrow Retr_{x_i}(\alpha h_{QN})$

until $\|g_i\| < \epsilon$

⇒ The only difference is that on a general manifold, in steps 4 & 5 we need a *retraction operator* $Retr_x: T_x(M) \to M$ that describes how to move along the manifold *M* from *x* in the direction of a tangent vector $\dot{v} \in T_x(M)$