Ch. 9
Numerical
Optimization

What to know

- Definition of the problem
- Golden-section search
- Newton's method
- Steepest descent
- Nelder-Mead simplex method
- Example

TU Delft

Problem statement

$$\min_{\vec{x}\in\Omega} f(\vec{x}) \,, \quad \text{subject to}$$

$$g(\vec{x}) = \vec{0} \quad , \quad \text{and}$$

$$h(\vec{x}) \geq \vec{0}$$

$$\vec{x} = (x_0 \,, \ldots \,, x_{N-1})$$

Problem statement

objective/cost function

$$\min_{\vec{x}\in\Omega} f(\vec{x}) \quad , \quad \text{subject to}$$

design variables

design space

equality constraint $\longrightarrow$ $g(\vec{x}) = \vec{0}$, and

inequality constraint $\longrightarrow$ $h(\vec{x}) \geq \vec{0}$

$$\vec{x} = (x_0, \ldots, x_{N-1})$$

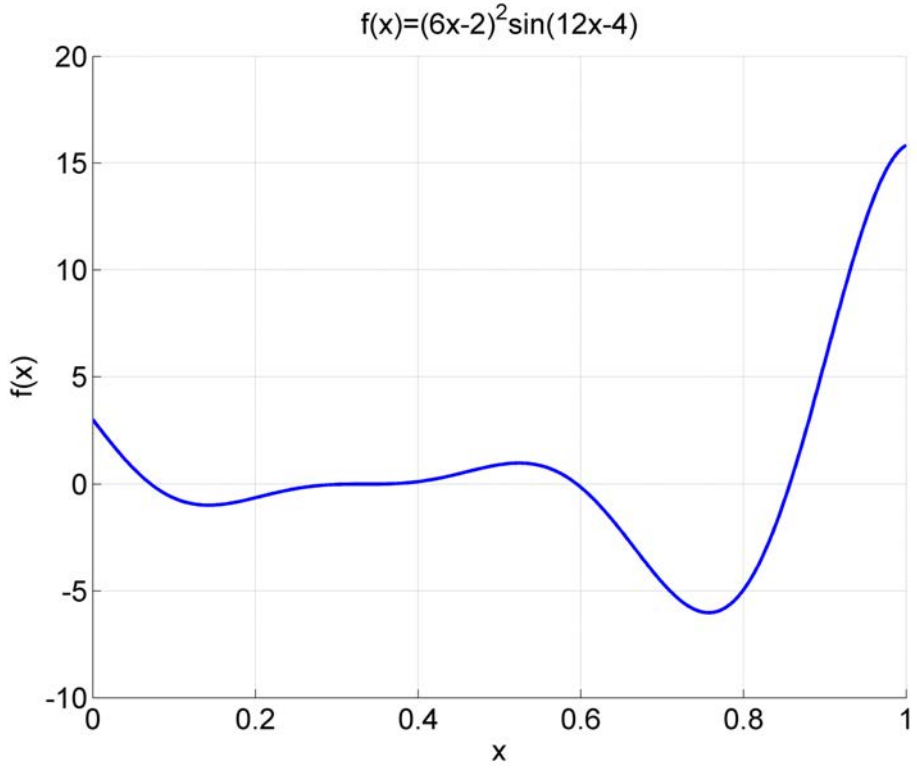
N is the dimension of the problem

- maximization = -minimization
- *N* can be large
- *f*, *g*, *h* can be expensive to evaluate
- *f', f"* may (not) be available

Optimization algorithms

$$\min_{0 \le x \le 1} f(x)$$

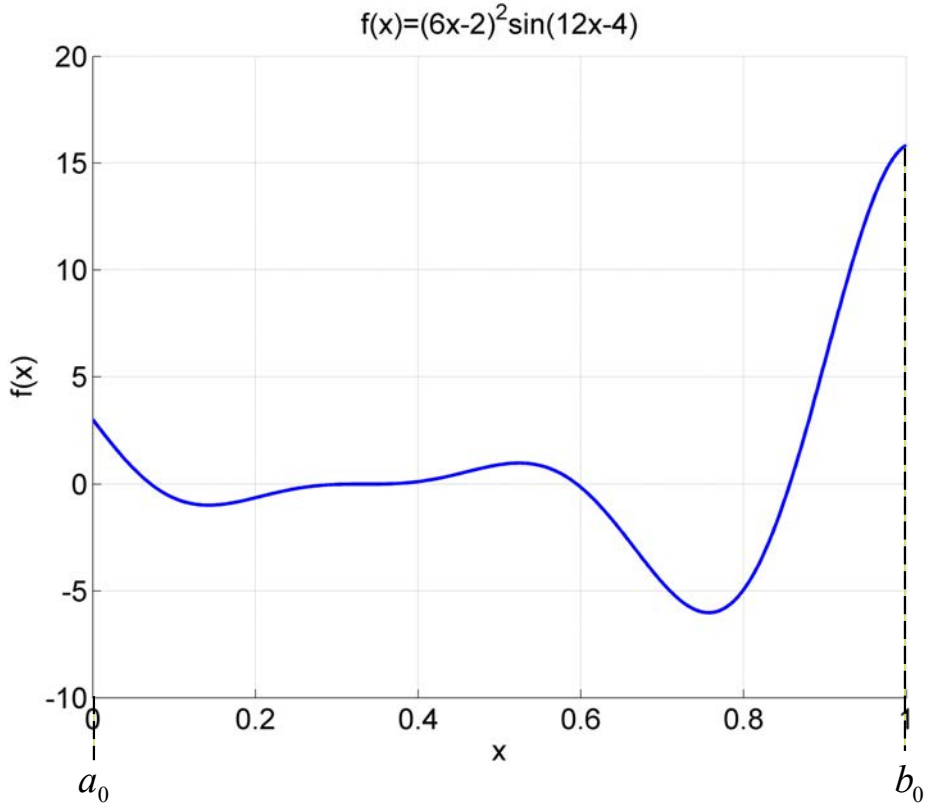
Recursive interval minimization



Optimization algorithms

$$\min_{0 \le x \le 1} f(x)$$

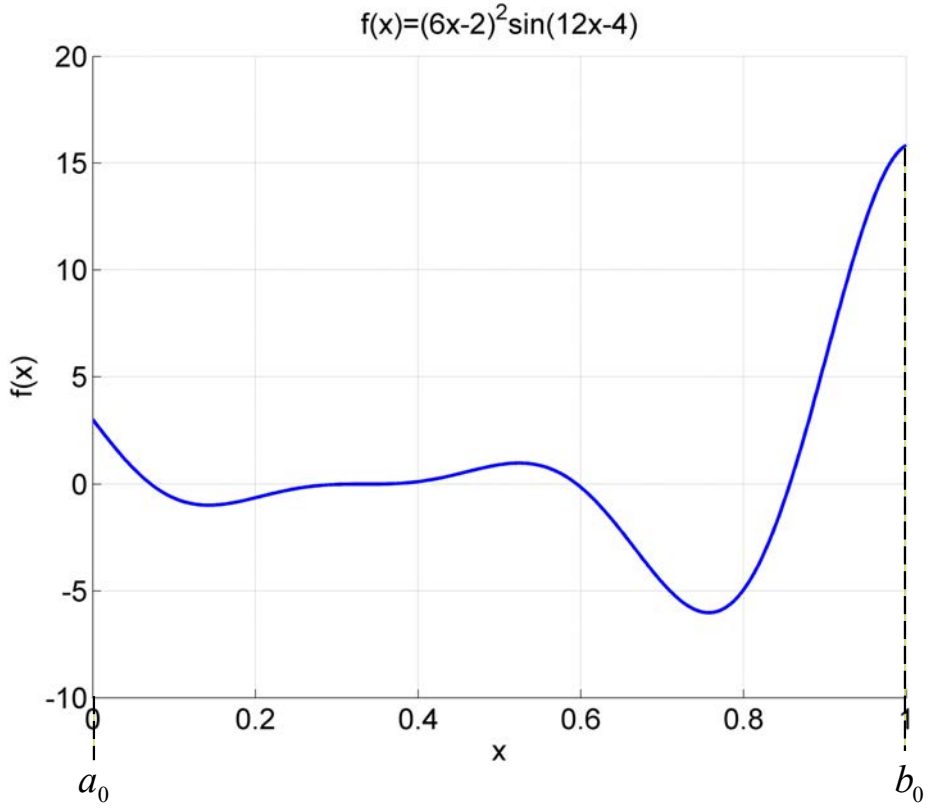
Recursive interval minimization



Optimization algorithms

Recursive interval minimization

$$\min_{0 \le x \le 1} f(x)$$



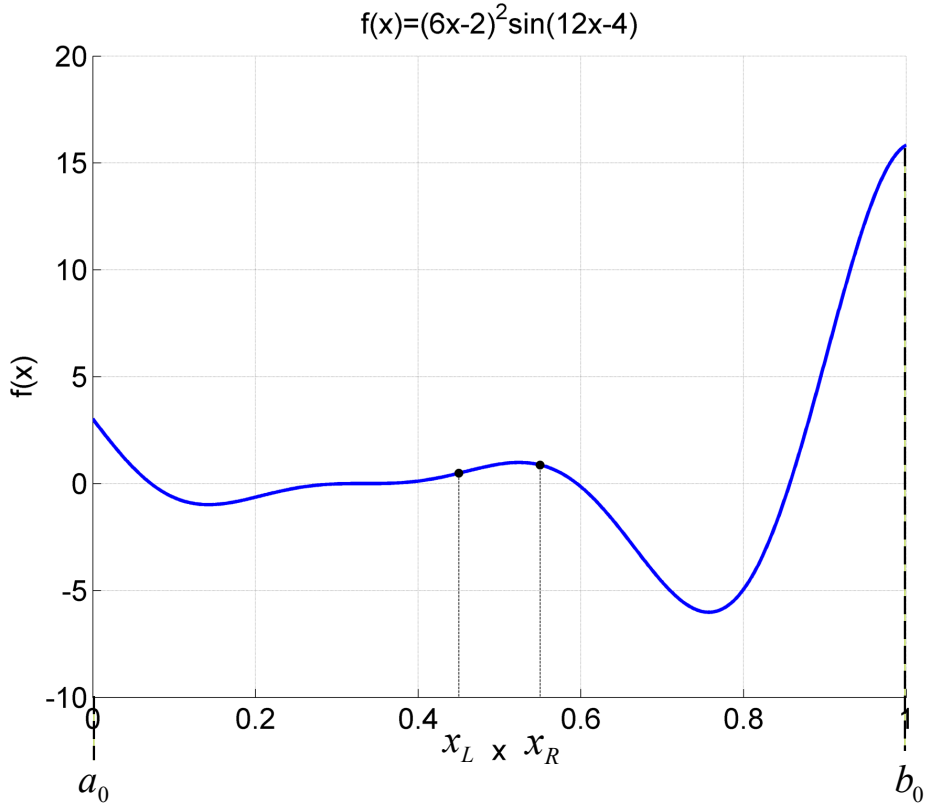
$$x_L = a_0 + \alpha(b_0 - a_0)$$
$$x_R = a_0 + \beta(b_0 - a_0)$$

$$\alpha, \beta \in (0,1)$$
$$\alpha < \beta$$

Optimization algorithms

Recursive interval minimization

$$\min_{0 \le x \le 1} f(x)$$



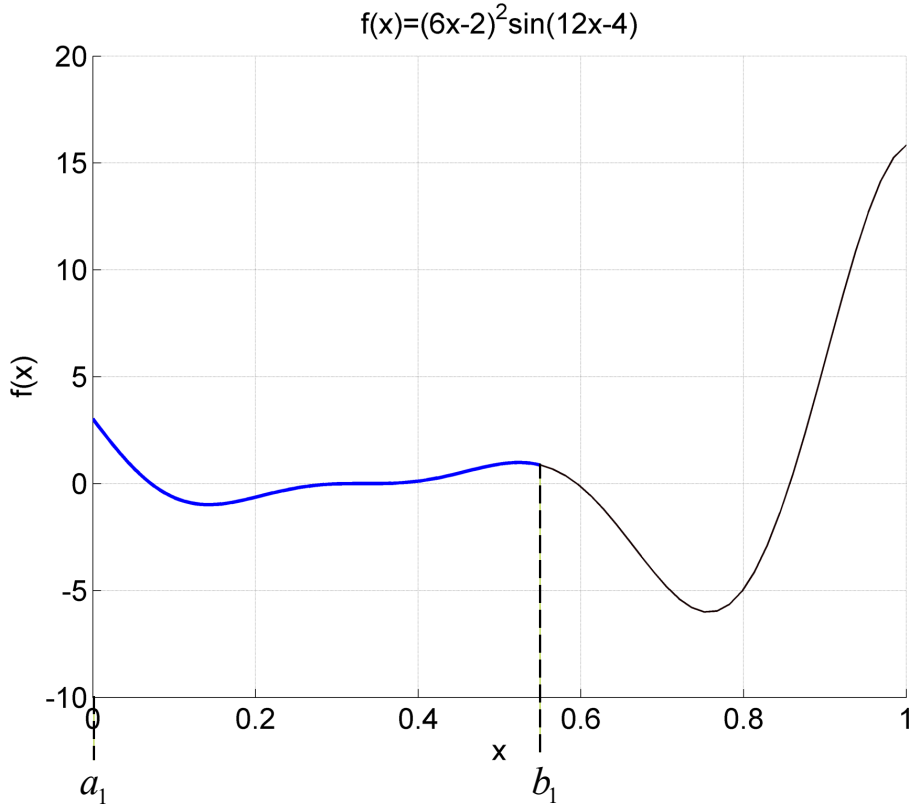
$$x_L = a_0 + 0.45(b_0 - a_0)$$
$$x_R = a_0 + 0.55(b_0 - a_0)$$

$$\alpha = 0.45$$
$$\beta = 0.55$$

Optimization algorithms

$$\min_{0 \le x \le 1} f(x)$$

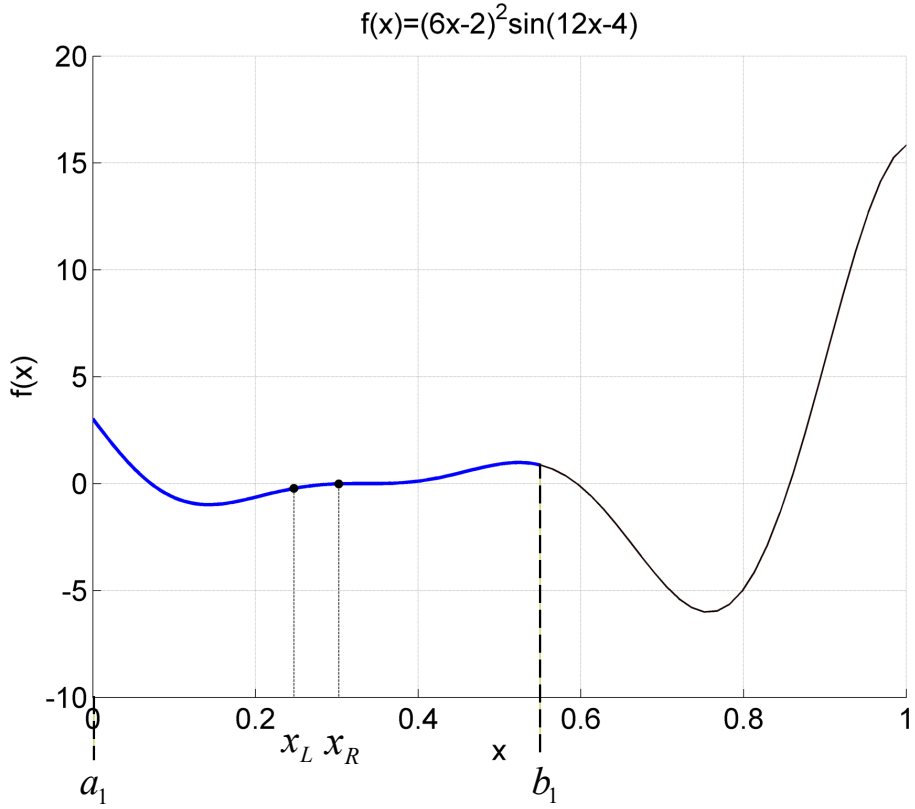
Recursive interval minimization



Optimization algorithms

Recursive interval minimization

$$\min_{0 \le x \le 1} f(x)$$



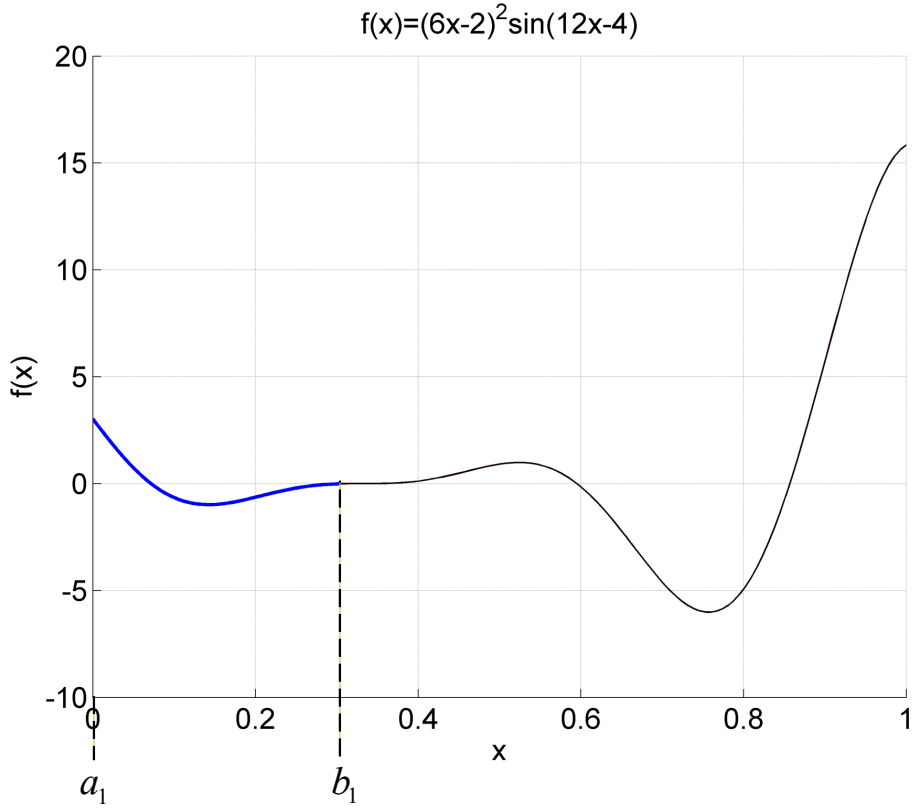
$$x_L = a_1 + 0.45(b_1 - a_1)$$
$$x_R = a_1 + 0.55(b_1 - a_1)$$

$$\alpha = 0.45$$
$$\beta = 0.55$$

Optimization algorithms

$$\min_{0 \le x \le 1} f(x)$$

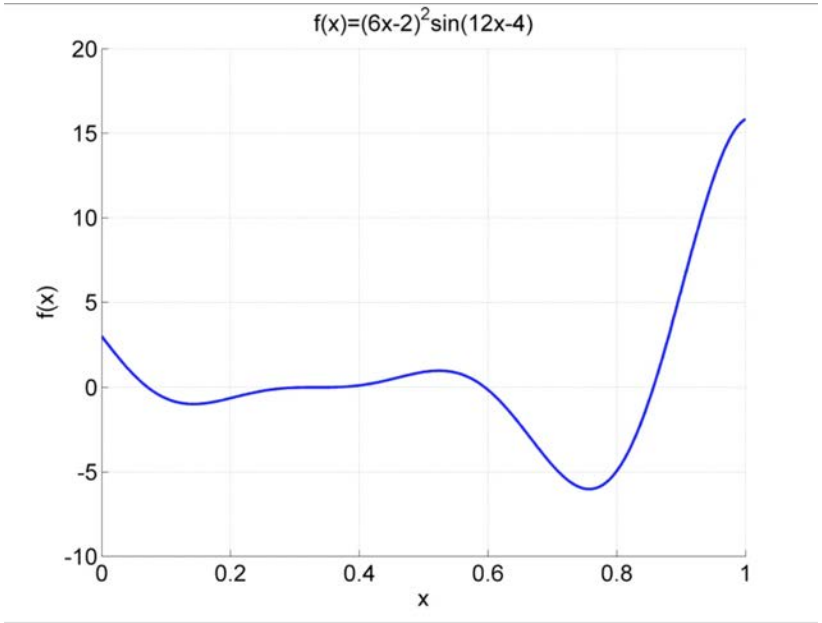
Recursive interval minimization

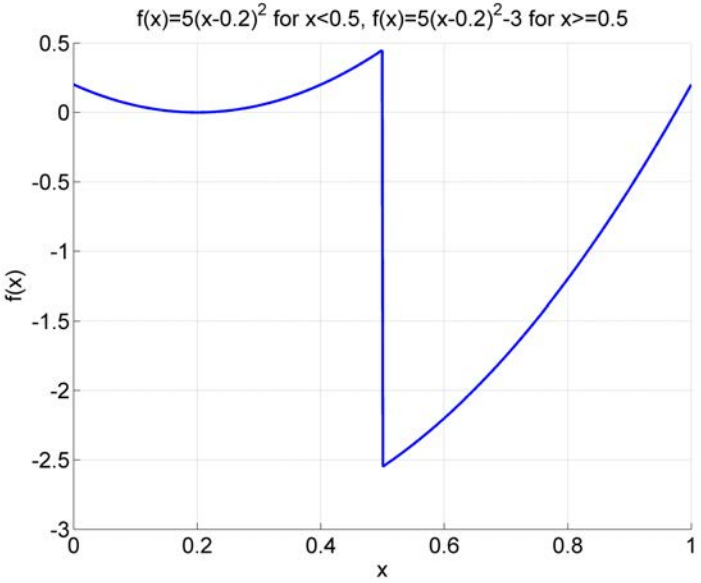


Optimization algorithms

$$\min_{0 \le x \le 1} f(x)$$

Recursive interval minimization

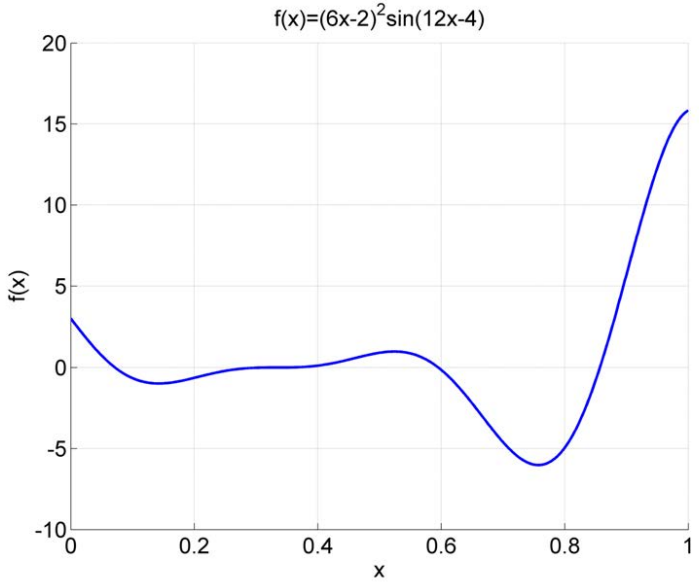


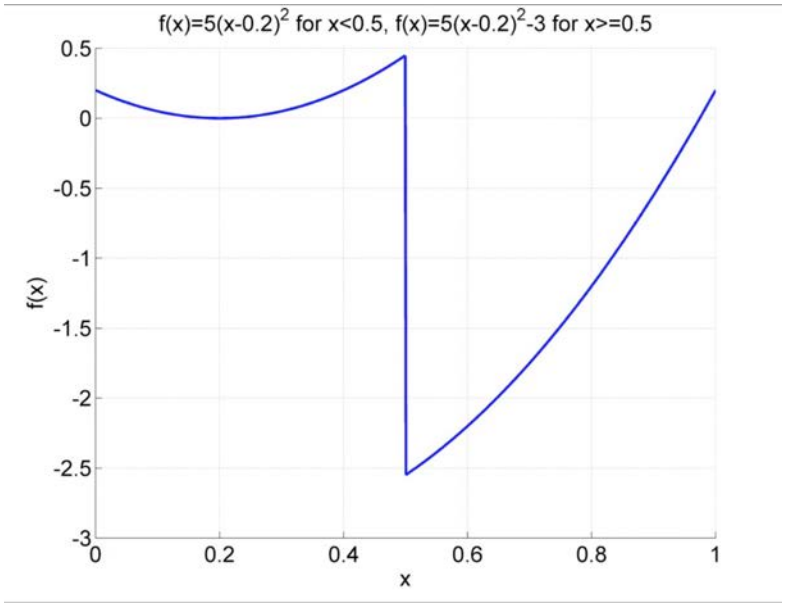


Optimization algorithms

$$\min_{0 \le x \le 1} f(x)$$

Recursive interval minimization





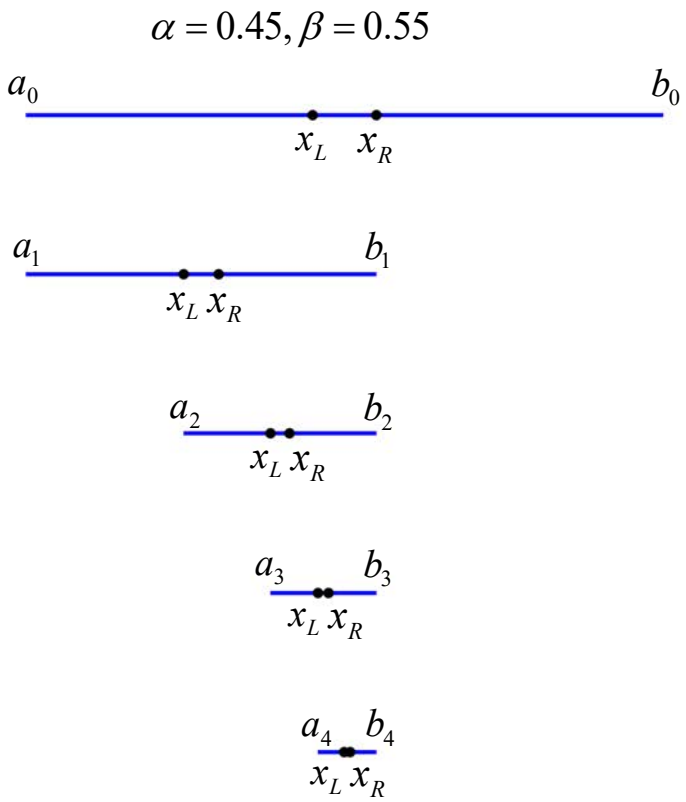
Optimization algorithms

Recursive interval minimization

- Gradient-free method
- Guaranteed convergence to **local** minimum (provided the objective function is continuous!!!)
- Only applicable in 1D (but useful for line-search in multi-D algorithms)
- 2 function evaluations per iteration

Optimization algorithms

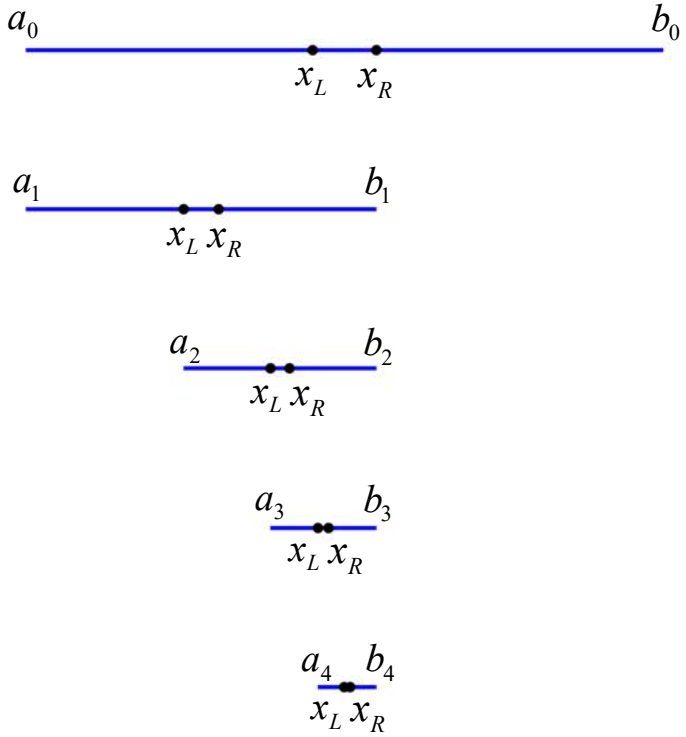
Recursive interval minimization



Optimization algorithms

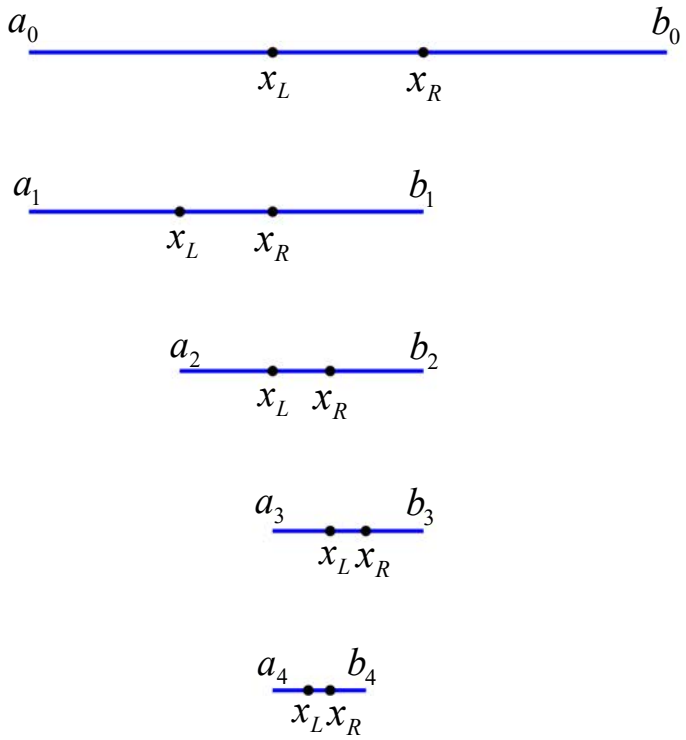
Recursive interval minimization

$\alpha = 0.45, \beta = 0.55$



$\varphi = ?$

$\alpha = 1 - \varphi^{-1}, \beta = \varphi^{-1}$

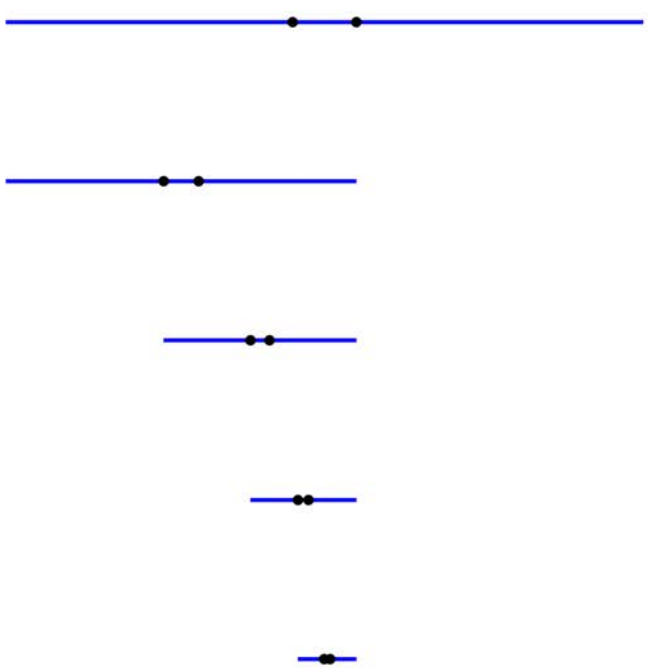


Optimization algorithms

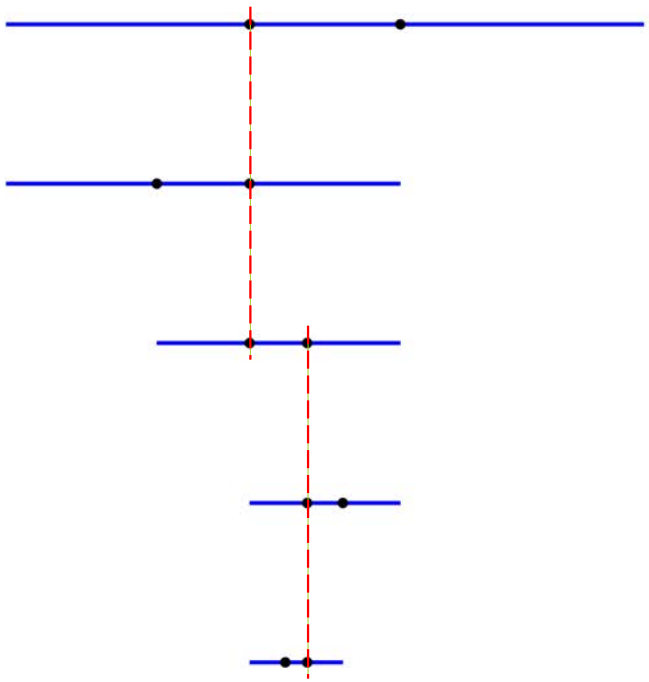
Recursive interval minimization

$\varphi = ?$

$\alpha = 0.45, \beta = 0.55$



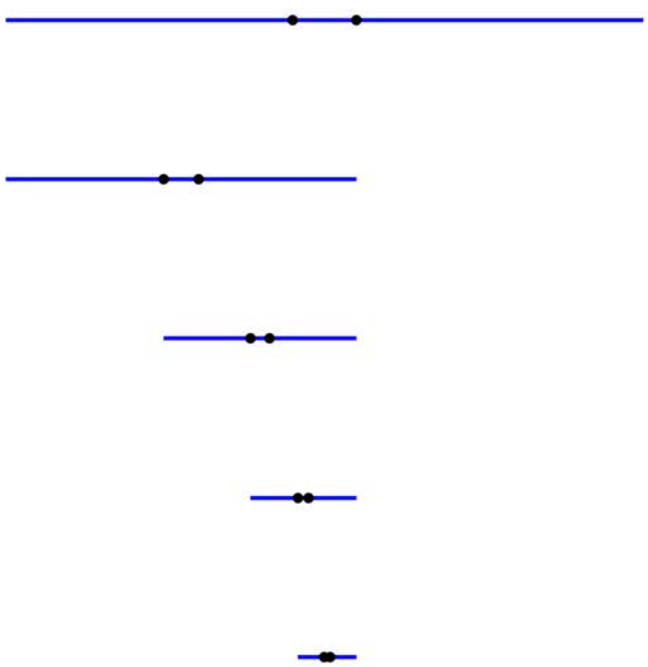
$\alpha = 1 - \varphi^{-1}, \beta = \varphi^{-1}$



Optimization algorithms

Recursive interval minimization

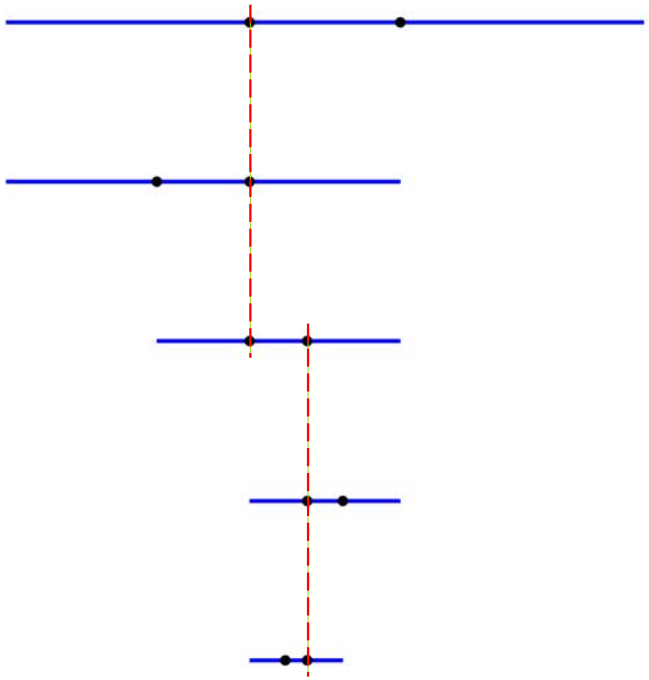
$\alpha = 0.45, \beta = 0.55$



2 function evaluations per iteration

$\varphi = ?$

$\alpha = 1 - \varphi^{-1}, \beta = \varphi^{-1}$



1 function evaluation per iteration !!!

Optimization algorithms

Recursive interval minimization

$$\frac{x_{L,0}-a_0}{x_{R,0}-a_0}=\frac{x_{R,1}-a_1}{b-a_1}=\beta=\frac{1}{\varphi}$$

$$\frac{a_0+\alpha(b_0-a_0)-a_0}{a_0+\beta(b_0-a_0)-a_0}=\frac{x_{R,1}-a_1}{b-a_1}=\beta=\frac{1}{\varphi}$$

$$\frac{\alpha}{\beta}=\frac{x_{R,1}-a_1}{b-a_1}=\beta=\frac{1}{\varphi}$$

$$\underbrace{\frac{1-\varphi^{-1}}{\varphi^{-1}}=\frac{x_{R,1}-a_1}{b-a_1}=\beta=\frac{1}{\varphi}}$$

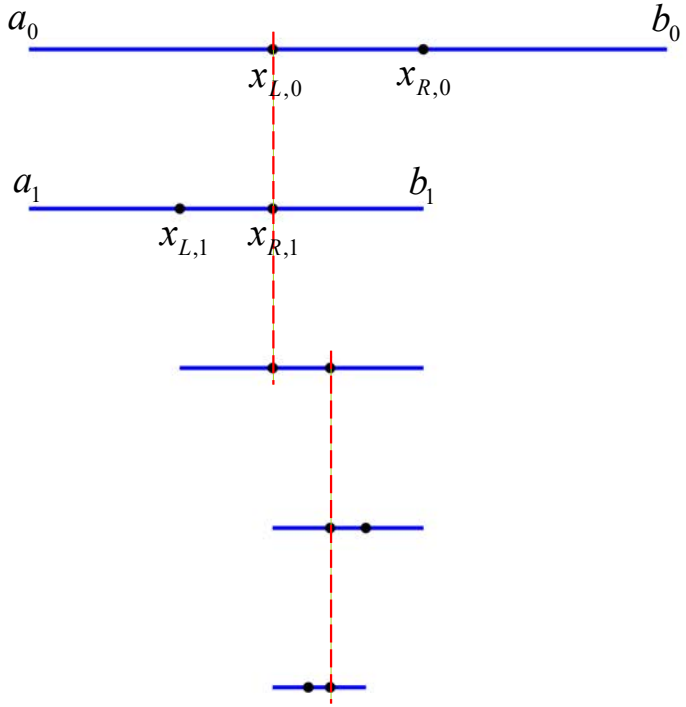
$$\varphi^2-\varphi-1=0$$

$$\varphi=\frac{1+\sqrt{5}}{2}$$

the "Golden Ratio"

$\varphi=?$

$\alpha=1-\varphi^{-1},\beta=\varphi^{-1}$



Optimization algorithms

Golden-section search

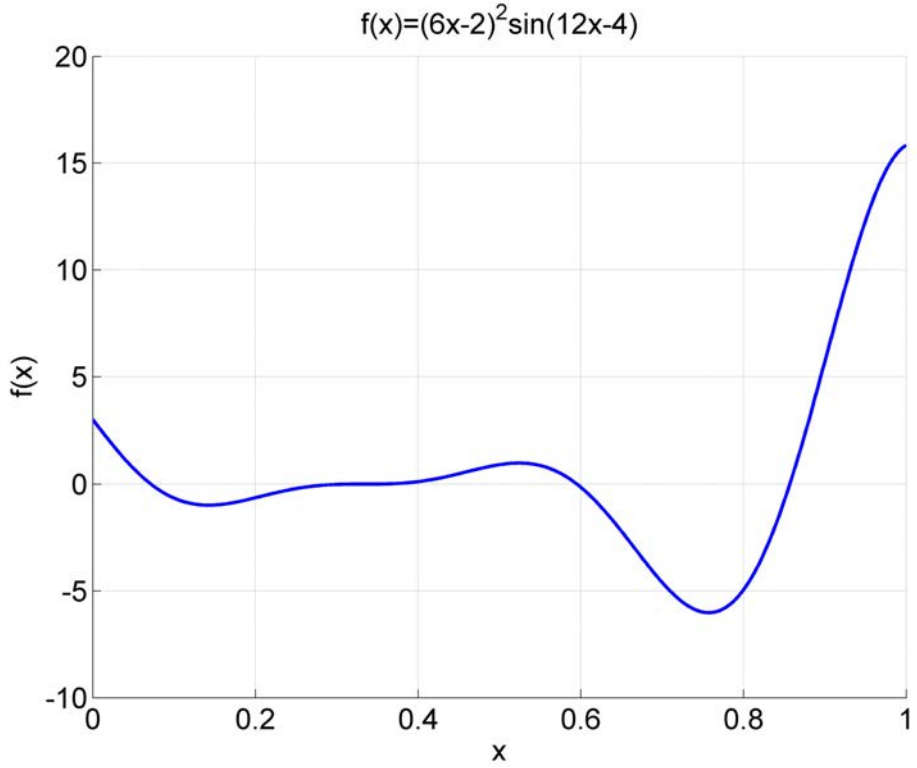
- **Gradient-free** method
- Guaranteed convergence to **local minimum** (provided the objective function is continuous!!!)
- Only applicable in **1D** (but useful in line-search for multi-D problems)
- **1 function evaluation** per iteration
- If $\varepsilon_0 = \frac{b-a}{2}$, then $\varepsilon_n = \left(\varphi^{-1}\right)^n \frac{b-a}{2}$ on the *n*-th step:

linear convergence

Optimization algorithms

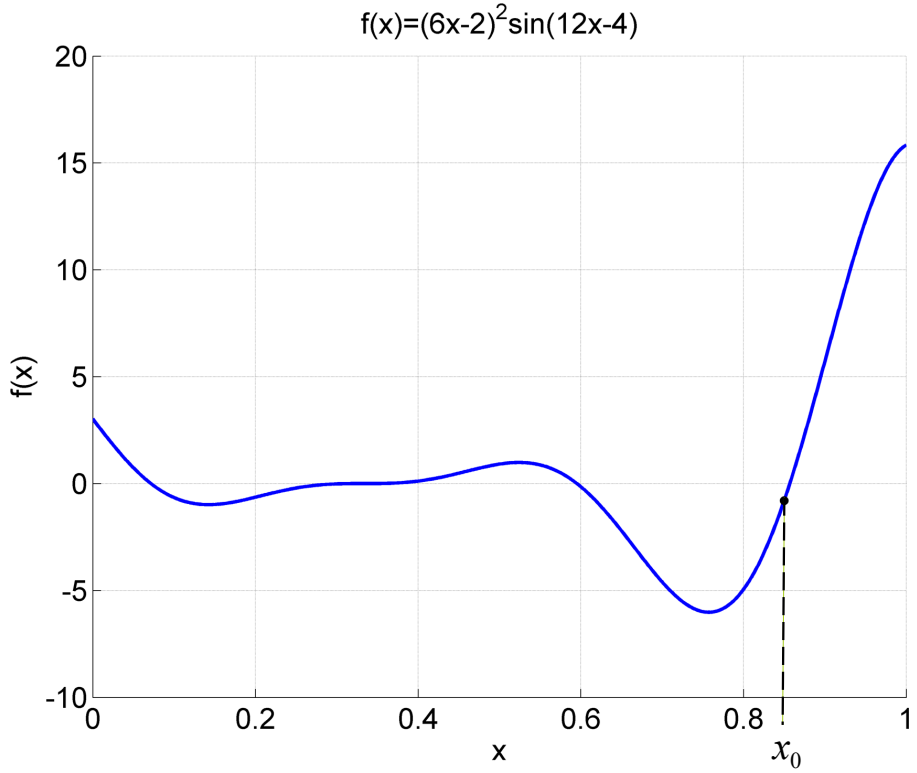
Newton's method

Let's use gradient information!



Optimization algorithms

Newton's method



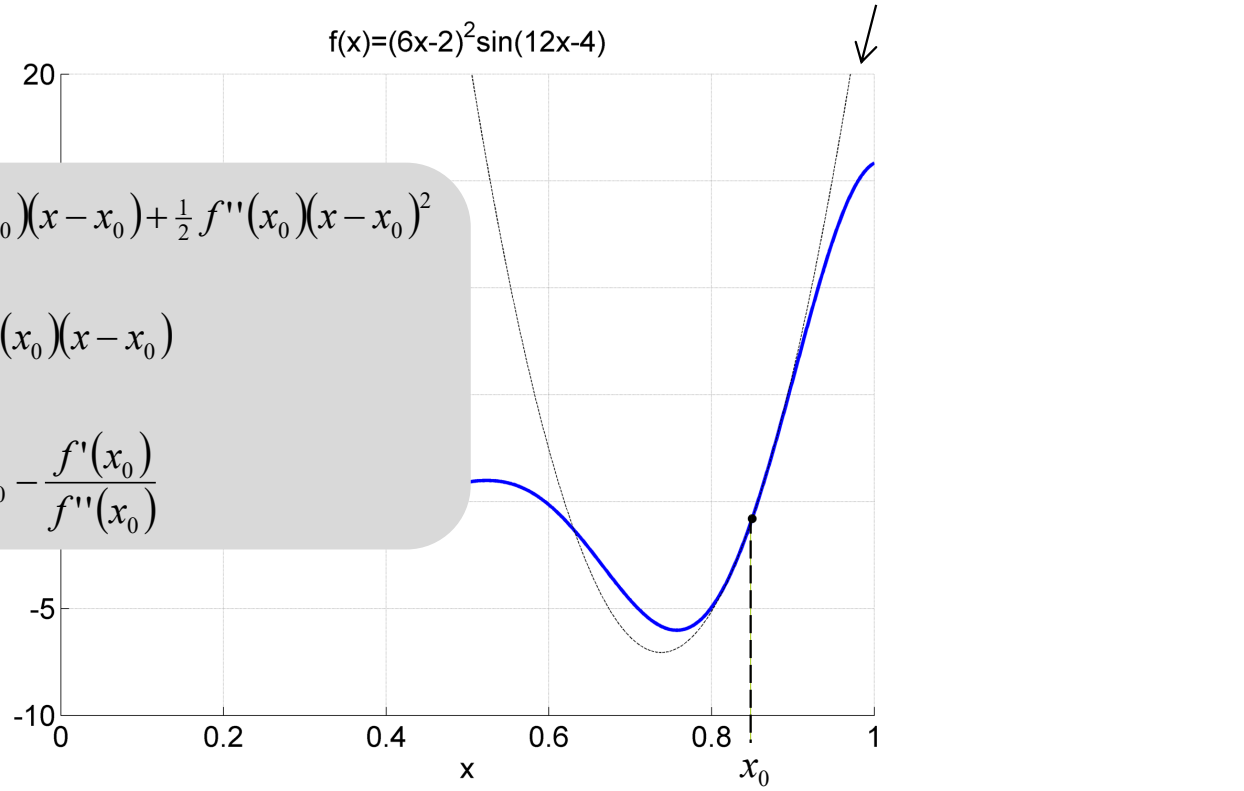
Optimization algorithms

Newton's method

$$y_0(x) = f(x_0) + f'(x_0)(x - x_0) + \tfrac{1}{2} f''(x_0)(x - x_0)^2$$

$$y_0'(x) = f'(x_0) + f''(x_0)(x - x_0)$$

$$y_0'(x_1) = 0 \Rightarrow x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}$$



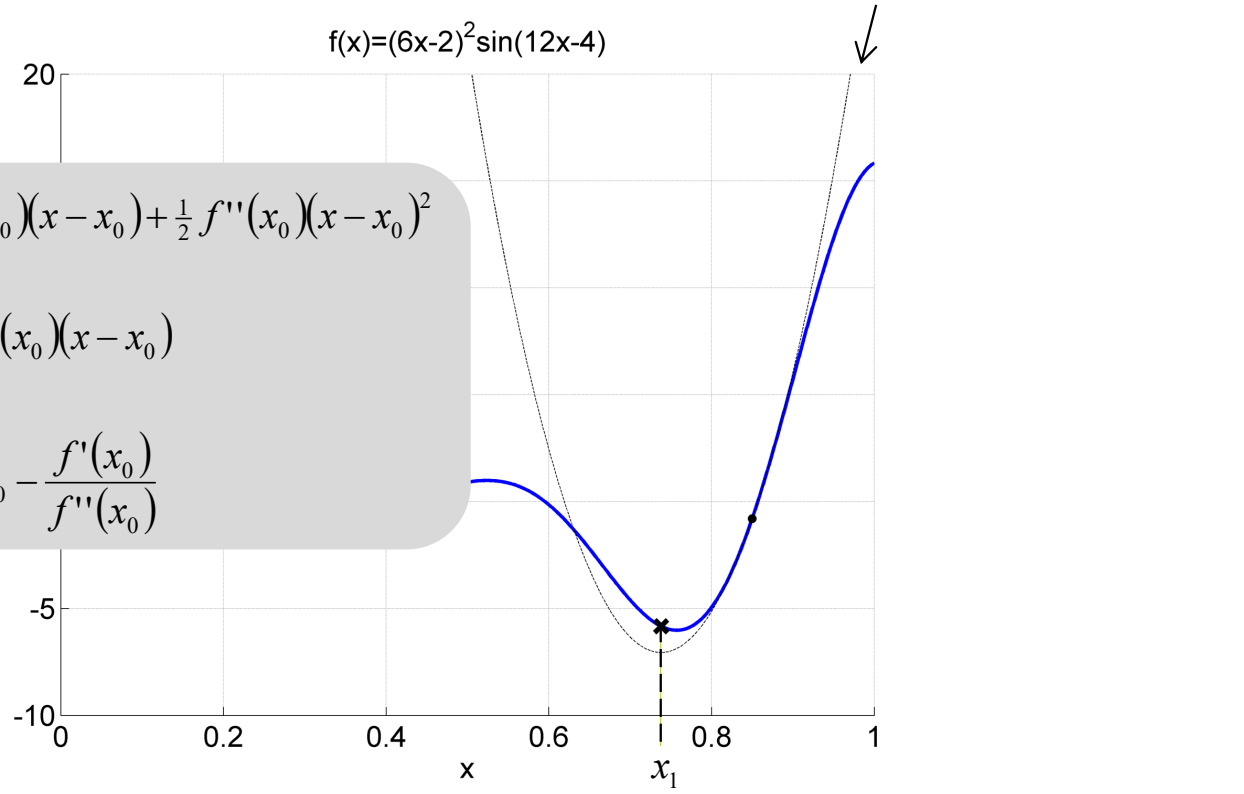
Optimization algorithms

Newton's method

$$y_0(x) = f(x_0) + f'(x_0)(x - x_0) + \tfrac{1}{2} f''(x_0)(x - x_0)^2$$

$$y_0'(x) = f'(x_0) + f''(x_0)(x - x_0)$$

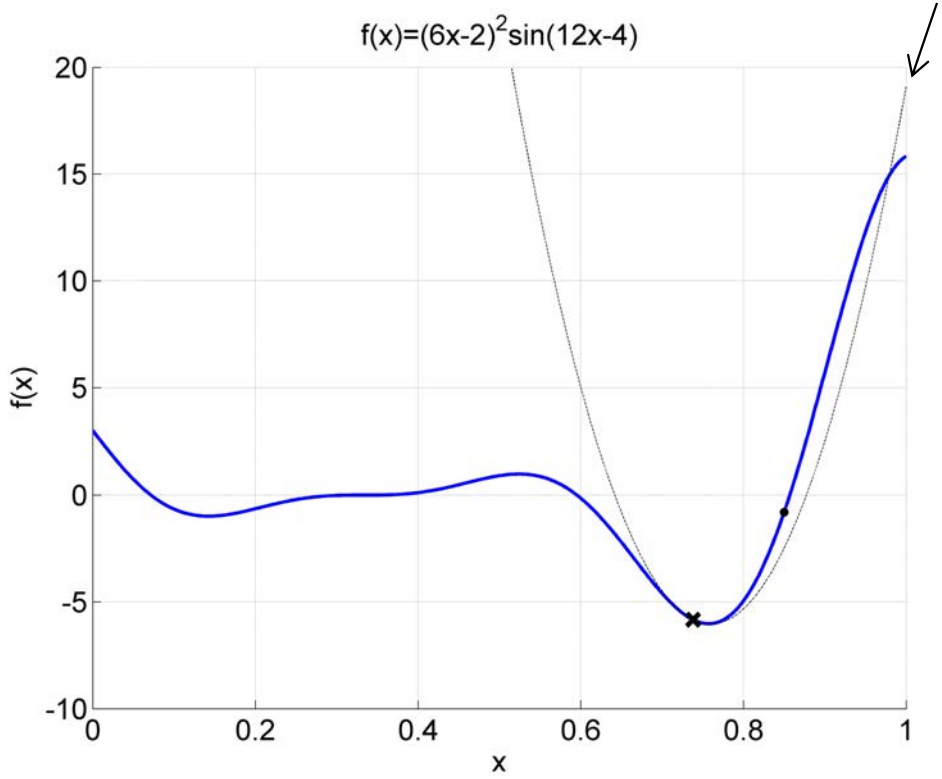
$$y_0'(x_1) = 0 \Rightarrow x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}$$



Optimization algorithms

Newton's method

$$y_1(x) = f(x_1) + f'(x_1)(x - x_1) + \tfrac{1}{2} f''(x_1)(x - x_1)^2$$



Optimization algorithms

Newton's method

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} \qquad \text{1-D}$$

$$\vec{x}_{k+1} = \vec{x}_k - \left[Hf(\vec{x}_k)\right]^{-1} \nabla f(\vec{x}_k) \qquad \text{N-D}$$

$$\nabla f(\vec{x}_k) = \begin{pmatrix} \frac{\partial f}{\partial x_0} \\ \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_{N-1}} \end{pmatrix}_{\vec{x}_k} \in \Re^{N\times 1}$$

Gradient

$$Hf(\vec{x}_k) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_0^2} & \frac{\partial^2 f}{\partial x_0 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_0 \partial x_{N-1}} \\ \frac{\partial^2 f}{\partial x_1 \partial x_0} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_{N-1} \partial x_0} & \cdots & \cdots & \frac{\partial^2 f}{\partial x_{N-1}^2} \end{pmatrix}_{\vec{x}_k} \in \Re^{N\times N}$$

Hessian

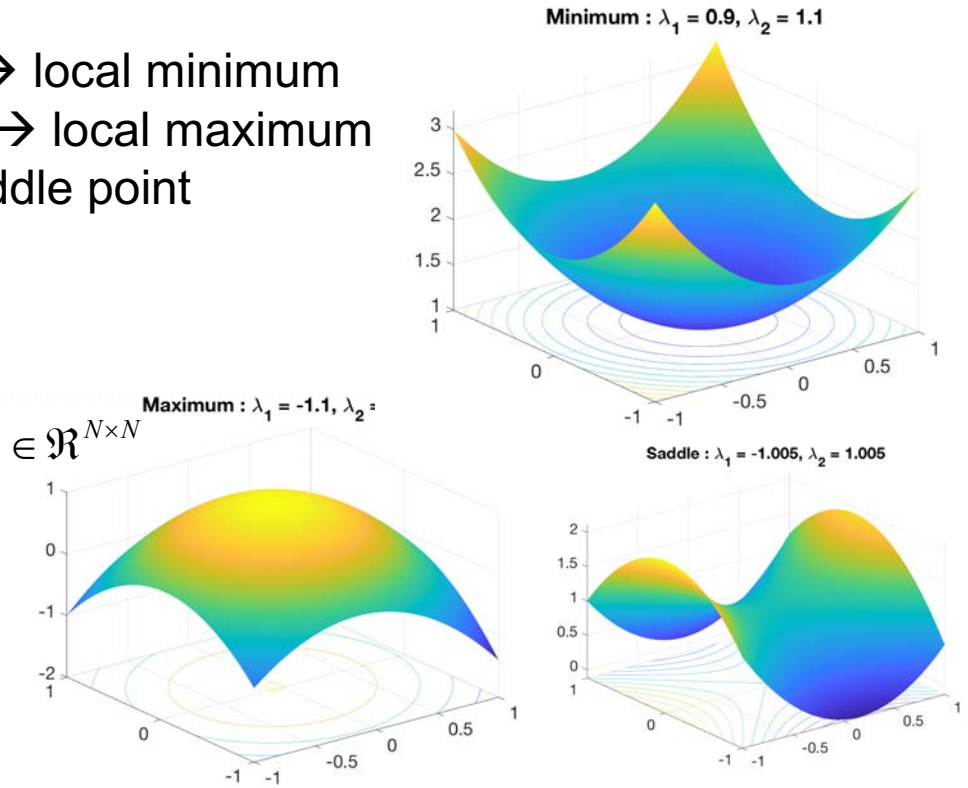
Optimization algorithms

Newton's method

- Hessian per definition symmetric → real eigenvalues λ_i
  - All eigenvalues positive → local minimum
  - All eigenvalues negative → local maximum
  - Mixed eigenvalues → saddle point

$$Hf(\vec{x}_k) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_0^2} & \frac{\partial^2 f}{\partial x_0 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_0 \partial x_{N-1}} \\ \frac{\partial^2 f}{\partial x_1 \partial x_0} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_{N-1} \partial x_0} & \cdots & \cdots & \frac{\partial^2 f}{\partial x_{N-1}^2} \end{pmatrix}_{\vec{x}_k} \in \Re^{N \times N}$$

Hessian



Optimization algorithms

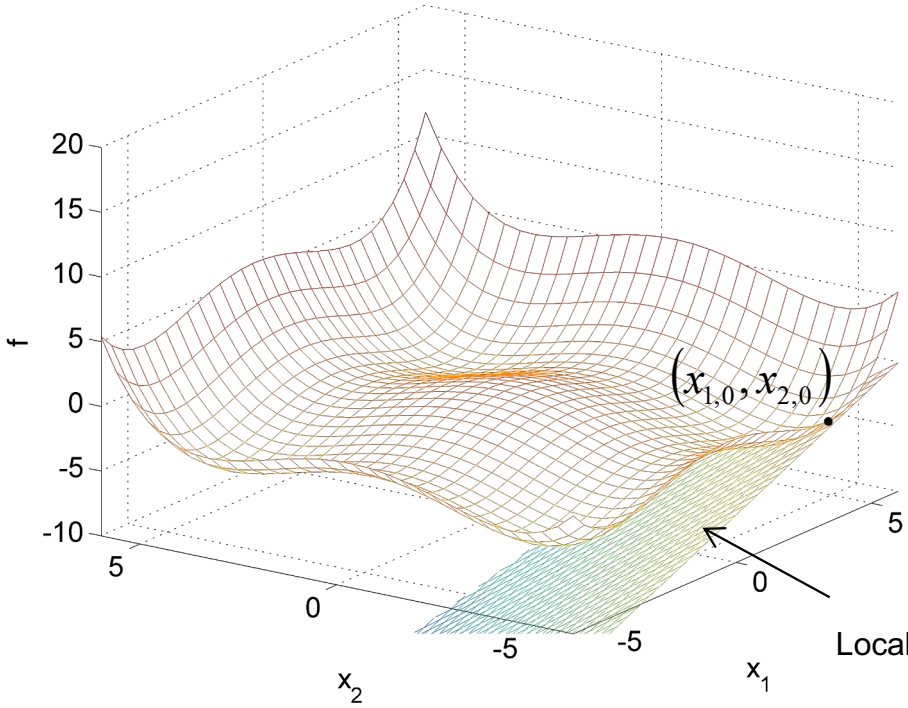
Newton's method

- Gradient method (1st and 2nd derivatives needed)
- N-dimensional problems (N ≥ 1)
- Quadratic convergence
- Gradient & Hessian evaluated per iteration
- Local minimum found
- Starting point needed

Optimization algorithms

Steepest descent method

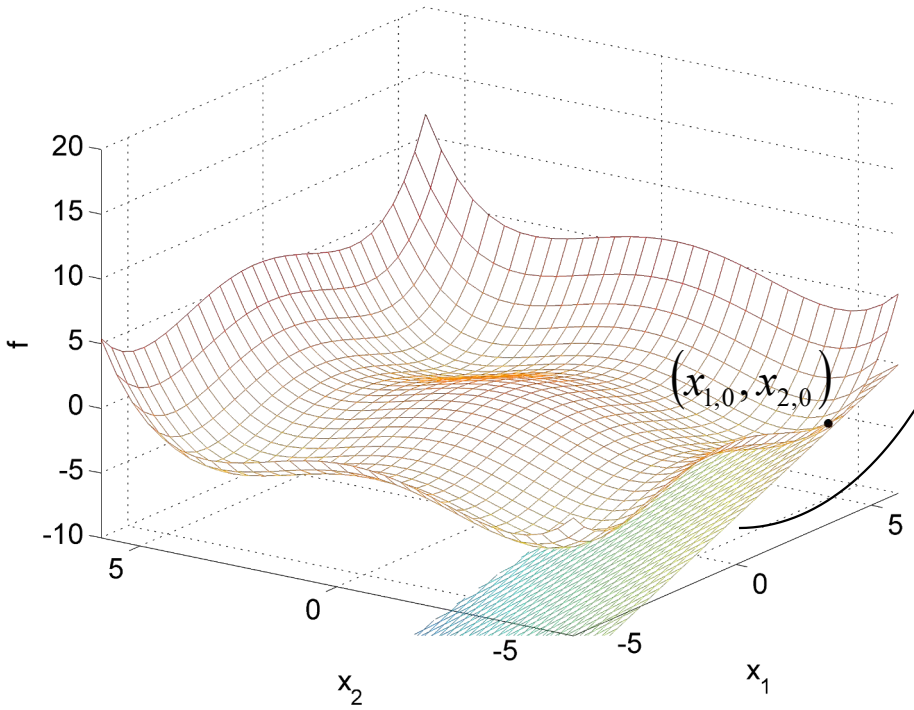
What if Hessian too expensive/unavailable?
→ Use gradient info only!



Local approximation of objective function

Optimization algorithms

Steepest descent method



$$y(\vec{x}) = f(\vec{x}_0) + \nabla f^T(\vec{x}_0)(\vec{x} - \vec{x}_0)$$

Gradient is direction of greatest rate of increase: move in opposite direction

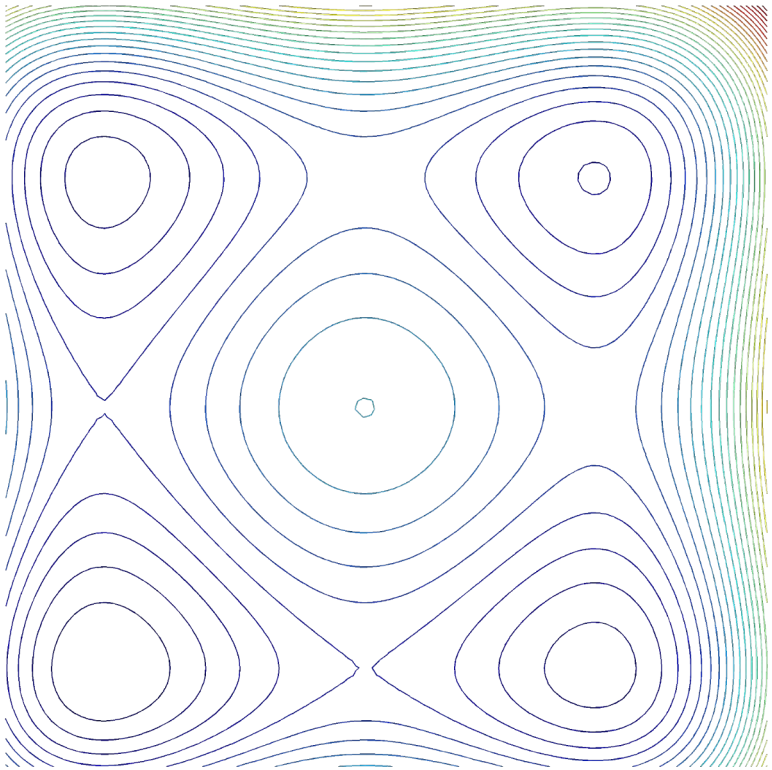
$$\vec{r}_k = -\nabla f(\vec{x}_k)$$

How big a step in that direction?

Golden-section search

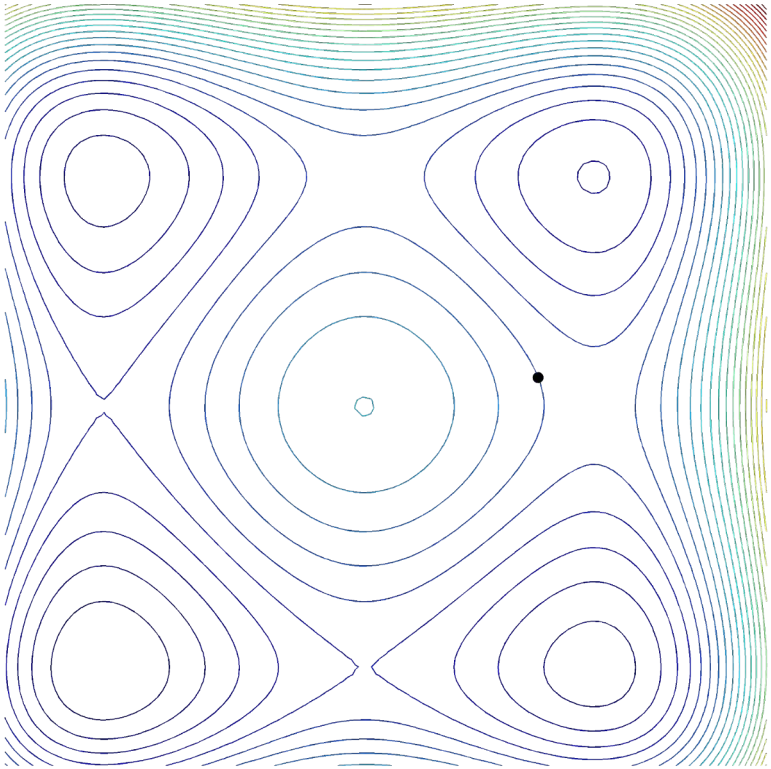
Optimization algorithms

Steepest descent method



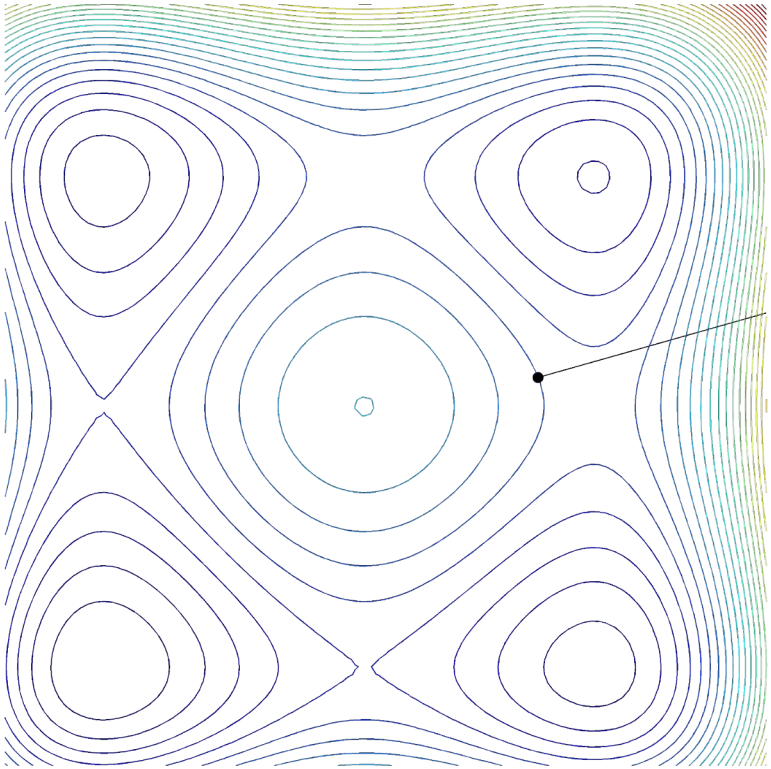
Optimization algorithms

Steepest descent method



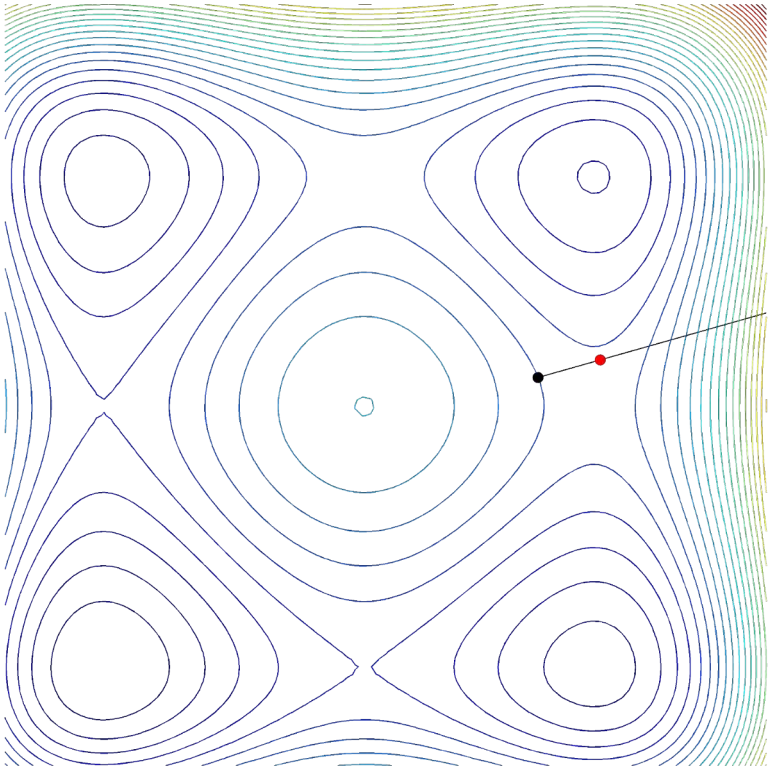
Optimization algorithms

Steepest descent method



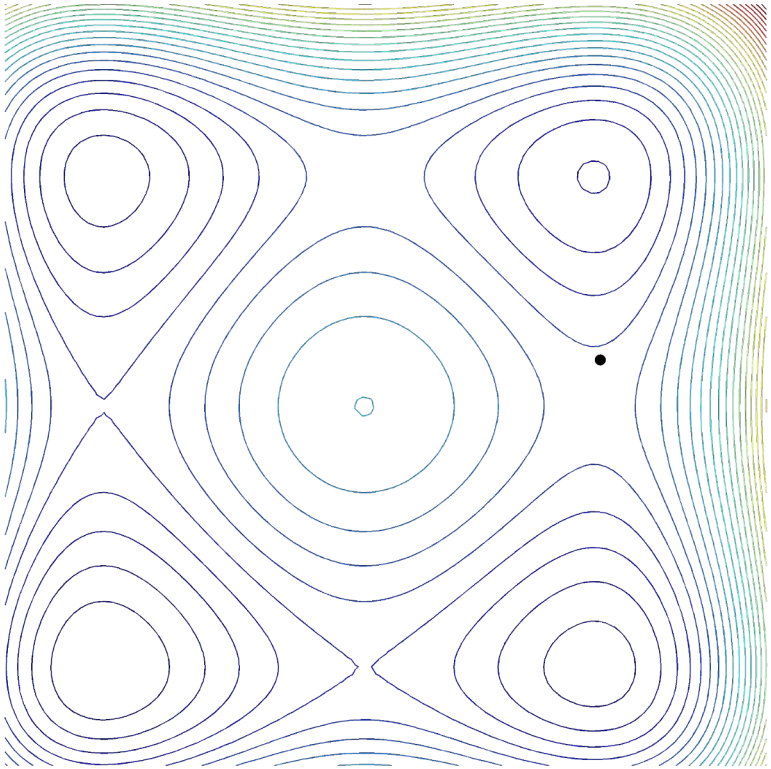
Optimization algorithms

Steepest descent method



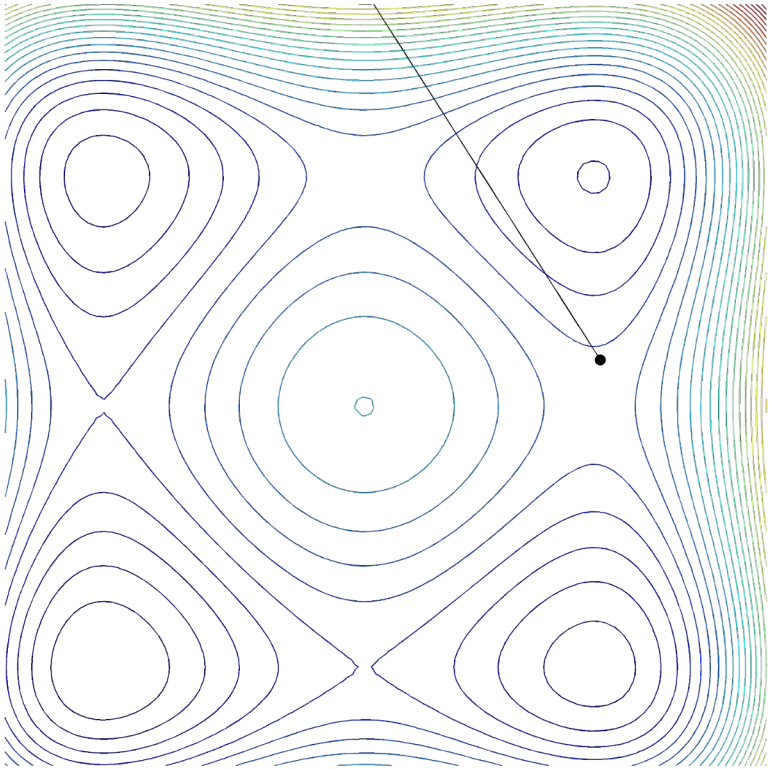
Optimization algorithms

Steepest descent method



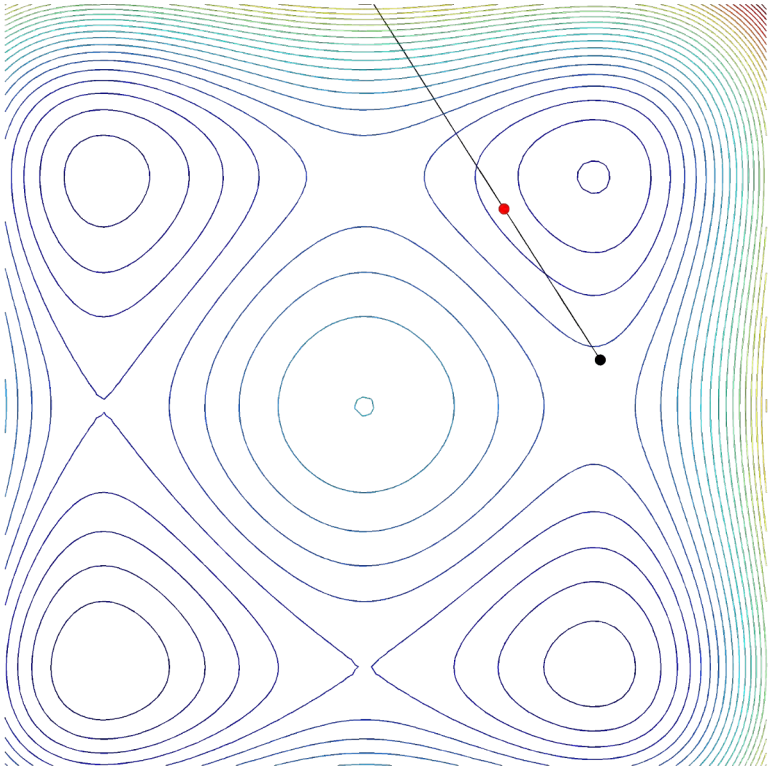
Optimization algorithms

Steepest descent method



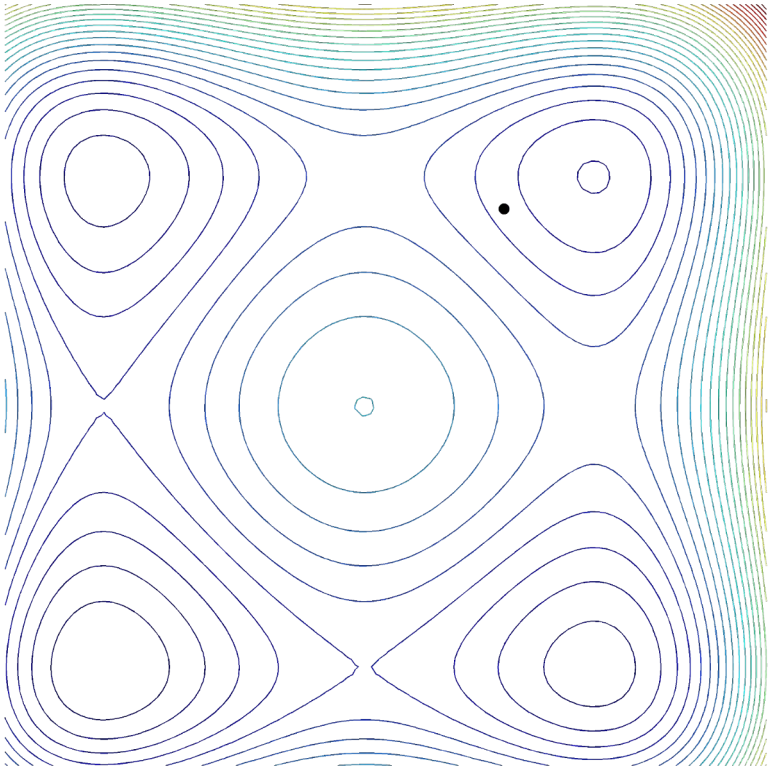
Optimization algorithms

Steepest descent method



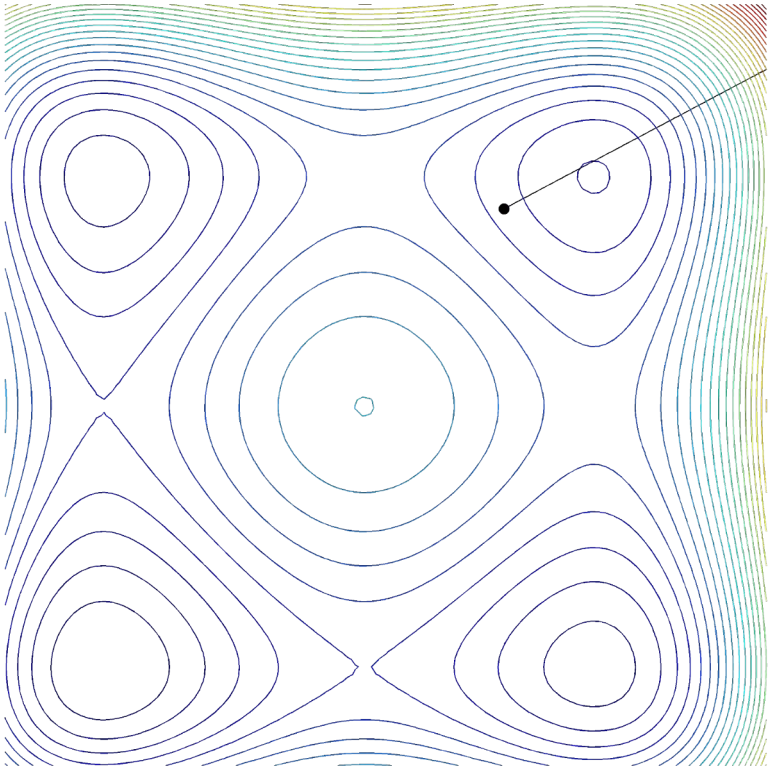
Optimization algorithms

Steepest descent method



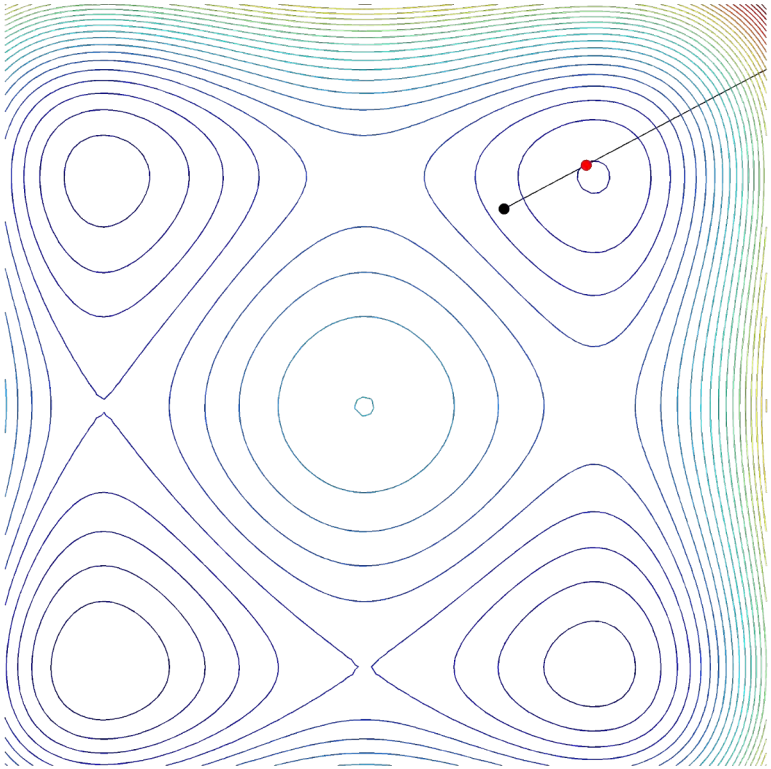
Optimization algorithms

Steepest descent method



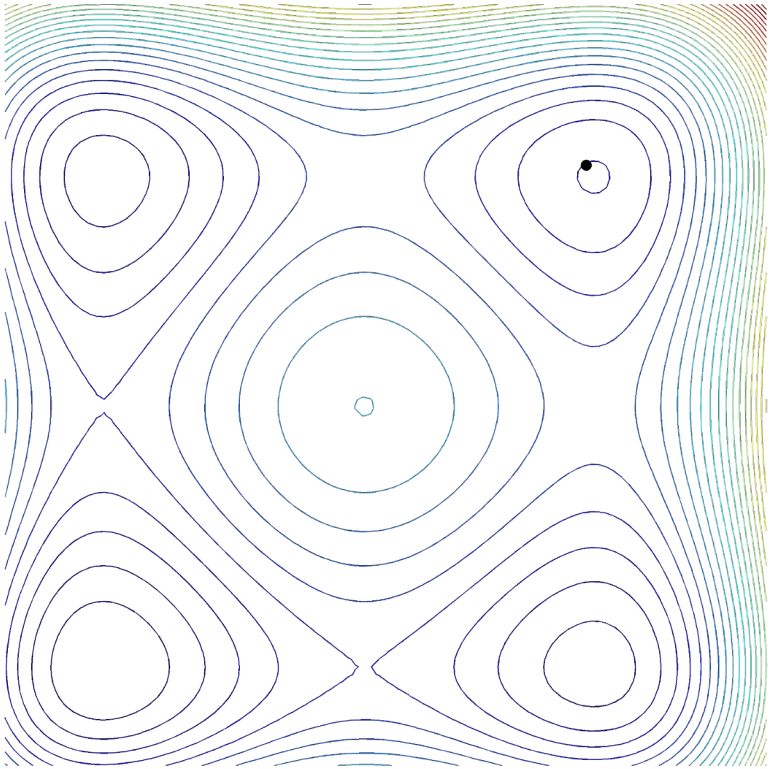
Optimization algorithms

Steepest descent method



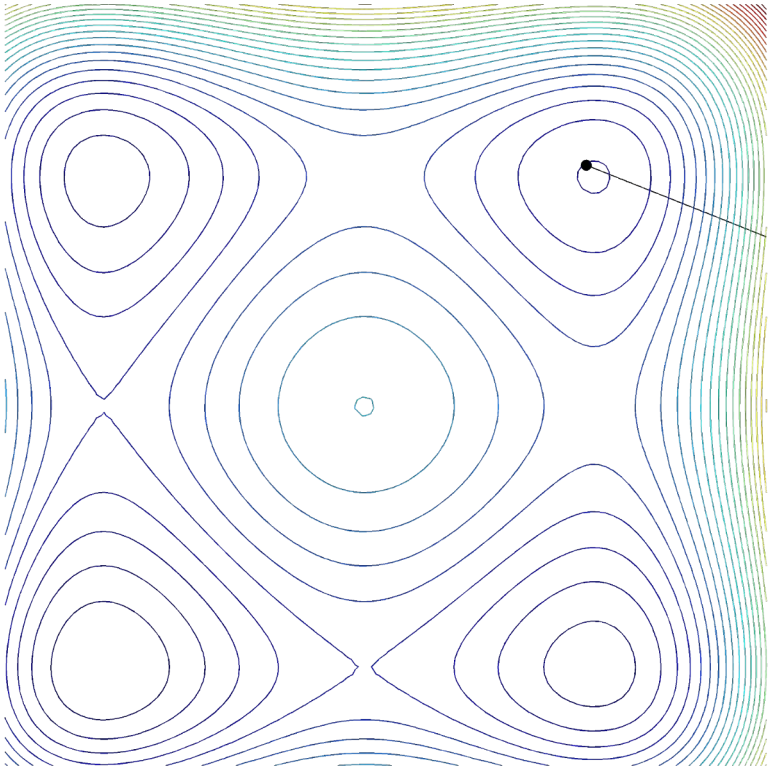
Optimization algorithms

Steepest descent method



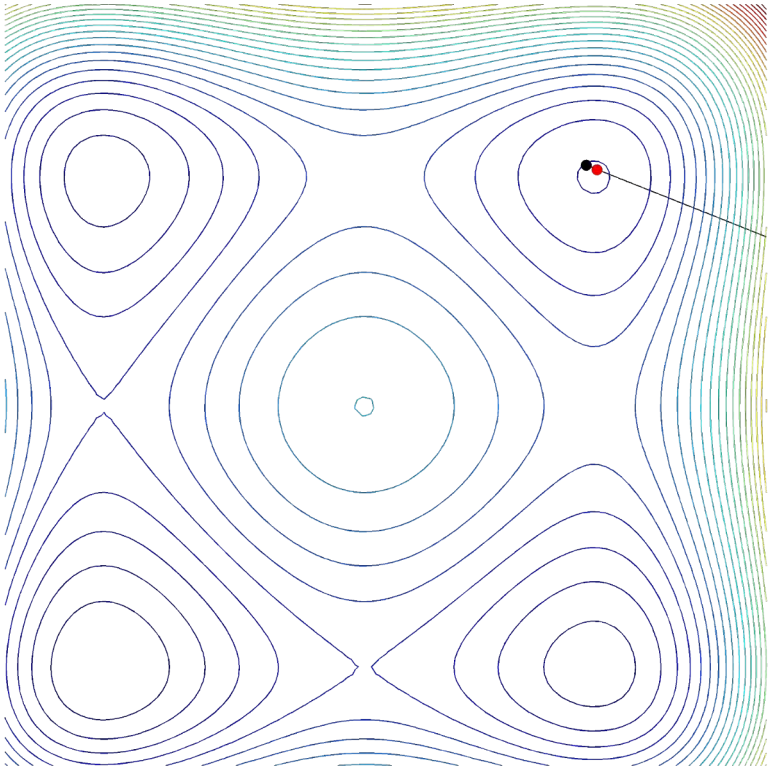
Optimization algorithms

Steepest descent method



Optimization algorithms

Steepest descent method



Optimization algorithms

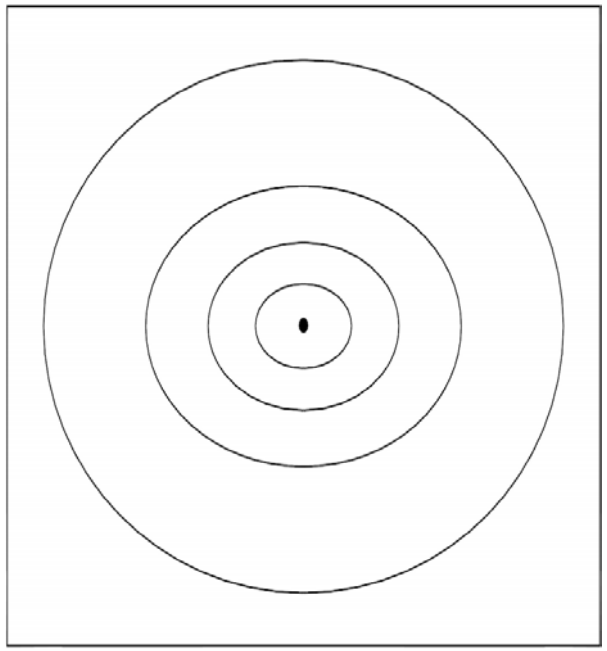
Steepest descent method

- Gradient method (1st derivative needed)
- N-dimensional problems (N ≥ 1)
- Linear convergence (not quadratic!)
- Gradient evaluated per iteration
- Local minimum found
- Starting point needed

Optimization algorithms

Newton's method vs. Steepest descent method

Which method finds minimum first?



Quadratic function

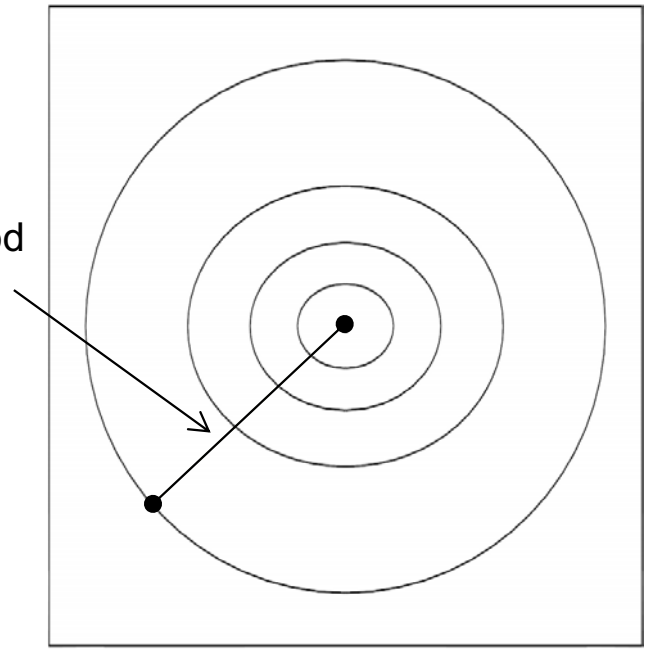
Concentric circles

Optimization algorithms

Newton's method vs. Steepest descent method

Which method finds minimum first?

Steepest descent method
&
Newton's method

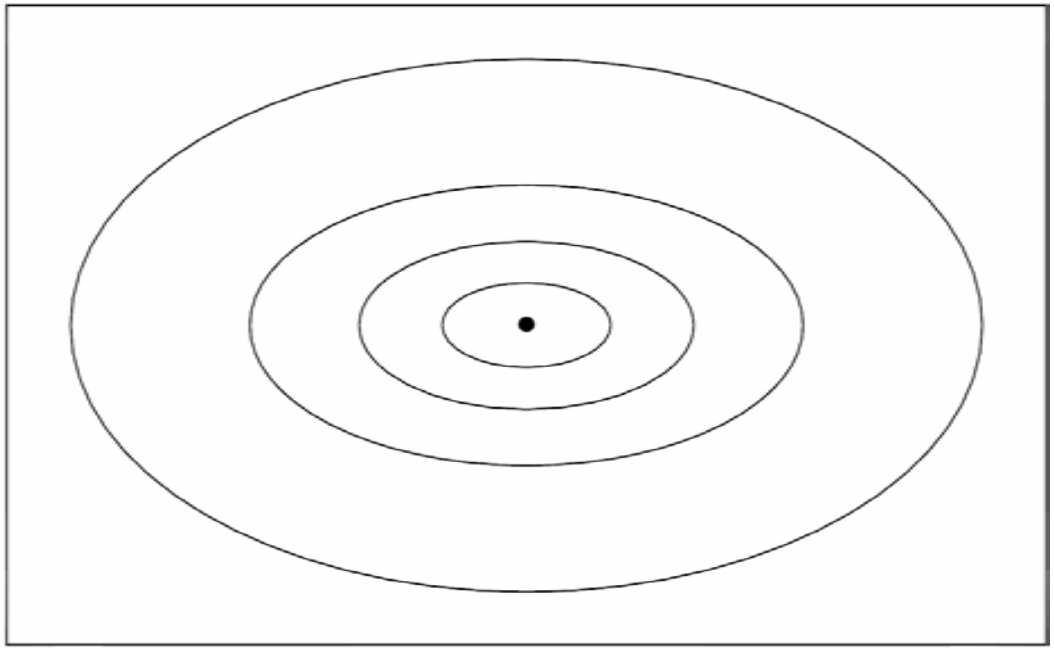


Quadratic function

Optimization algorithms

Newton's method vs. Steepest descent method

Which method finds minimum first?



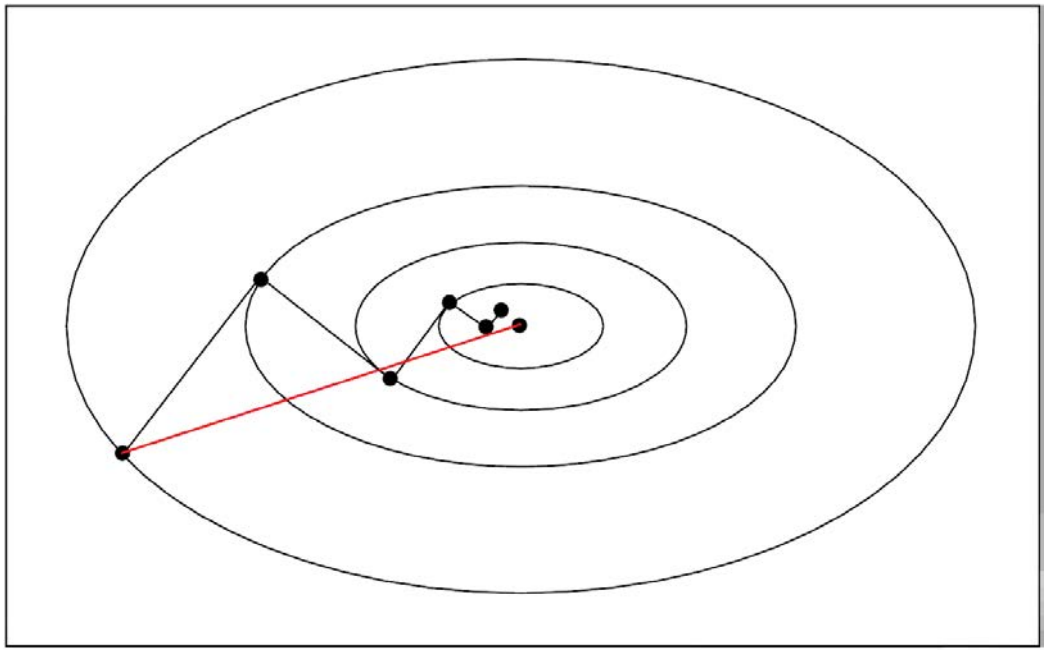
Quadratic function

Ellipse shape

Optimization algorithms

Newton's method vs. Steepest descent method

Which method finds minimum first?

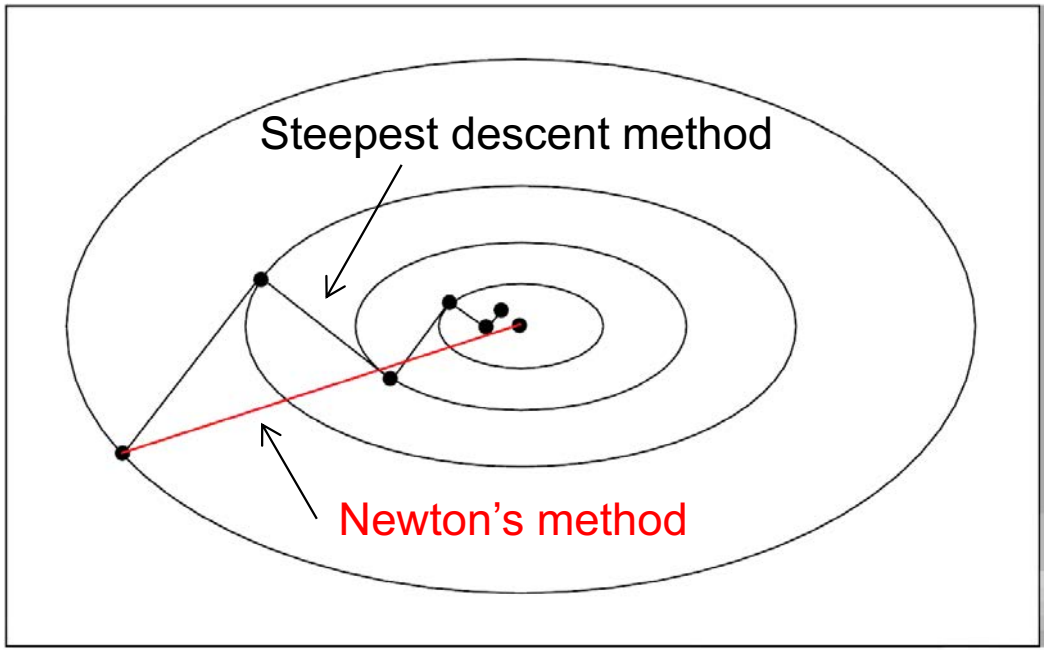


Quadratic function

Optimization algorithms

Newton's method vs. Steepest descent method

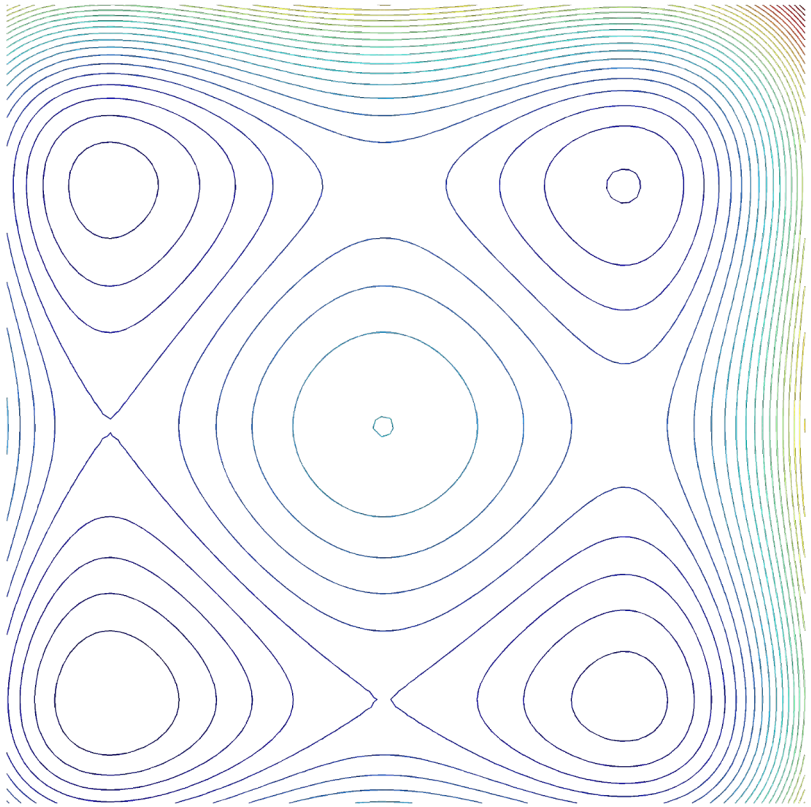
Which method finds minimum first?



Quadratic function

Optimization algorithms

Nelder-Mead simplex method



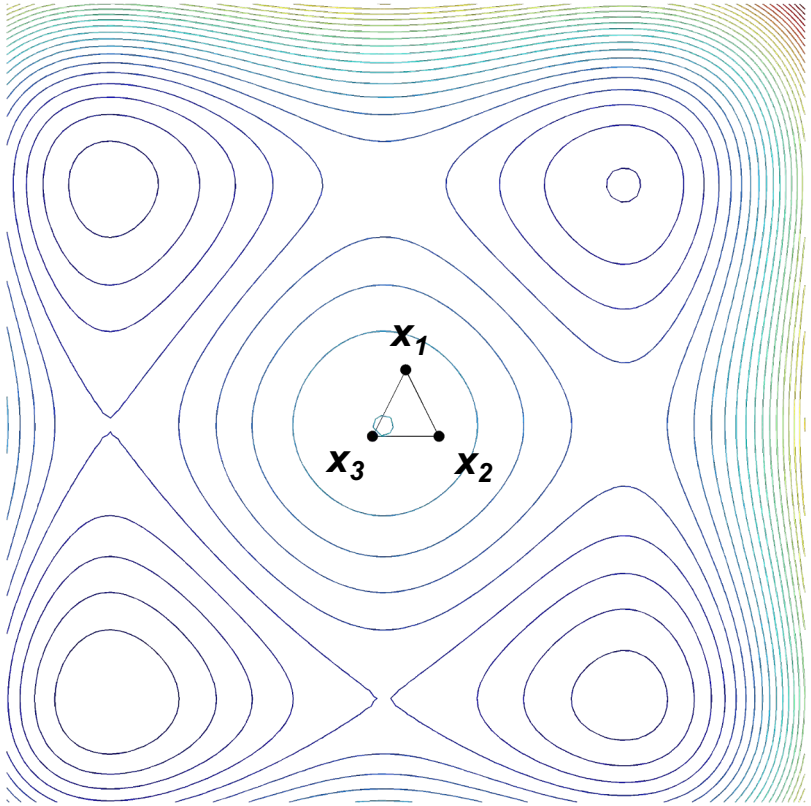
Optimization algorithms

Nelder-Mead simplex method

$+0.0500 = f(\vec{x}_3)$

$-0.1400 = f(\vec{x}_2)$

$-0.1644 = f(\vec{x}_1)$



Optimization algorithms

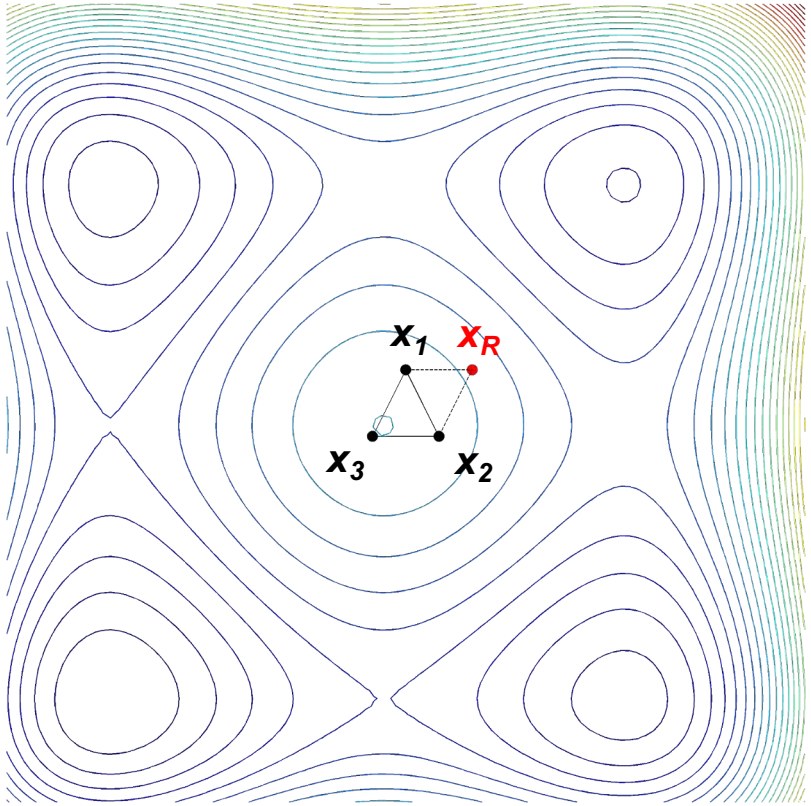
Nelder-Mead simplex method

$+0.0500 = f(\vec{x}_3)$

$-0.1400 = f(\vec{x}_2)$

$-0.1644 = f(\vec{x}_1)$

$-0.6144 = f(\vec{x}_R)$



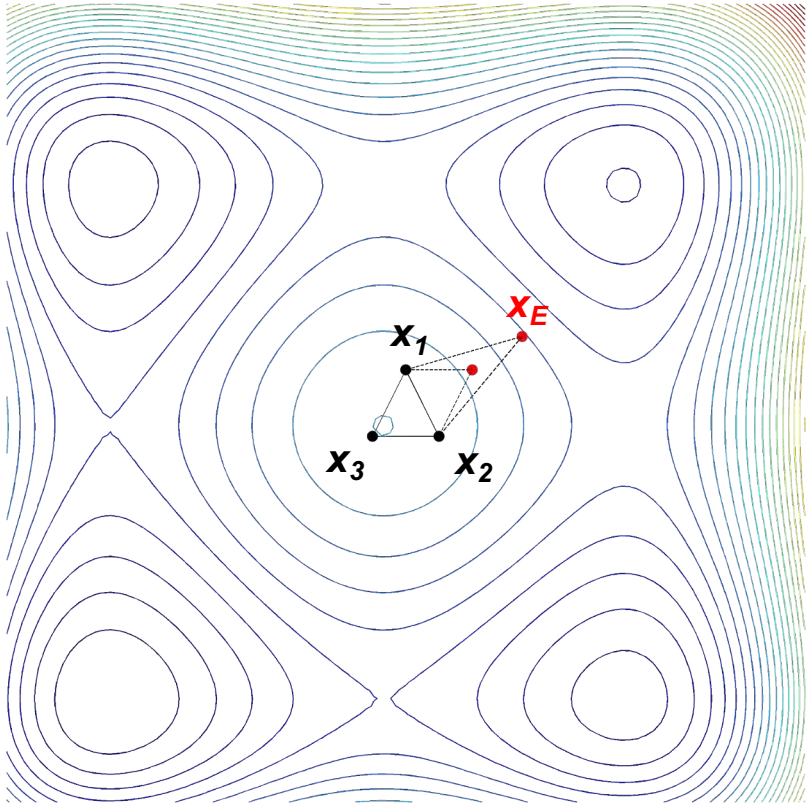
REFLECT

Optimization algorithms

Nelder-Mead simplex method

$$-0.6144 = f\left(\vec{x}_R\right)$$

$$-1.4618 = f\left(\vec{x}_E\right)$$



EXPAND

Optimization algorithms

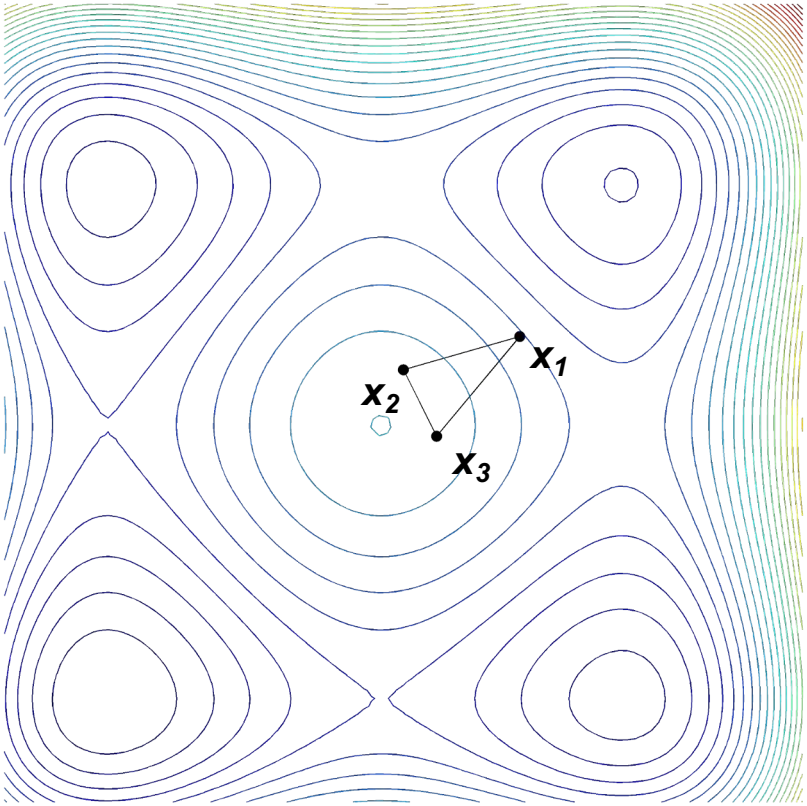
Nelder-Mead simplex method

Simplex has stretched

$-0.1400 = f(\vec{x}_3)$

$-0.1644 = f(\vec{x}_2)$

$-1.4618 = f(\vec{x}_1)$



Optimization algorithms

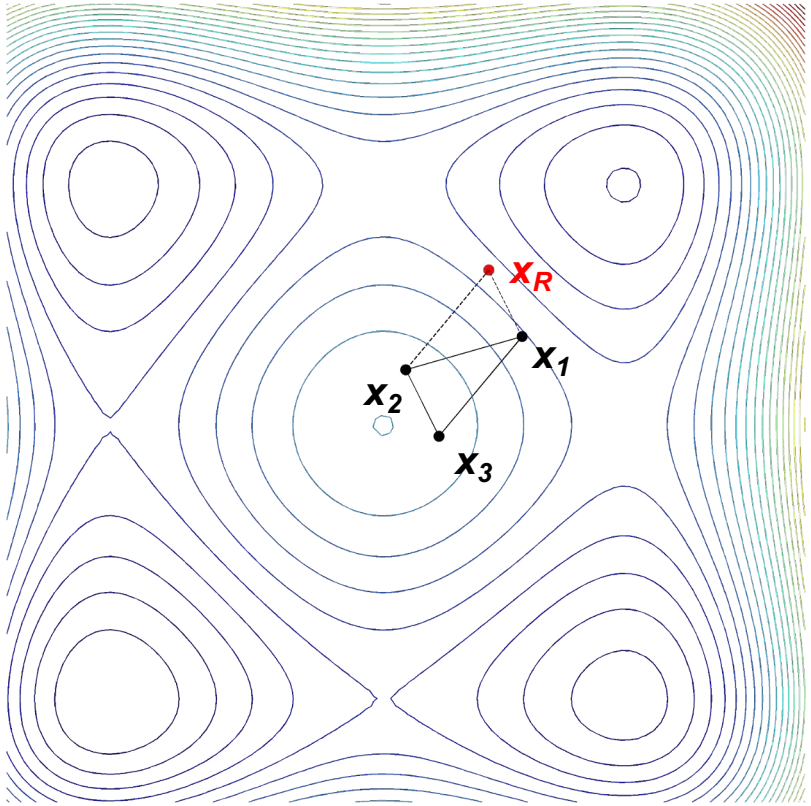
Nelder-Mead simplex method

$$-0.1400 = f(\vec{x}_3)$$

$$-0.1644 = f(\vec{x}_2)$$

$$-1.4618 = f(\vec{x}_1)$$

$$-1.8343 = f(\vec{x}_R)$$



REFLECT

Optimization algorithms

Nelder-Mead simplex method

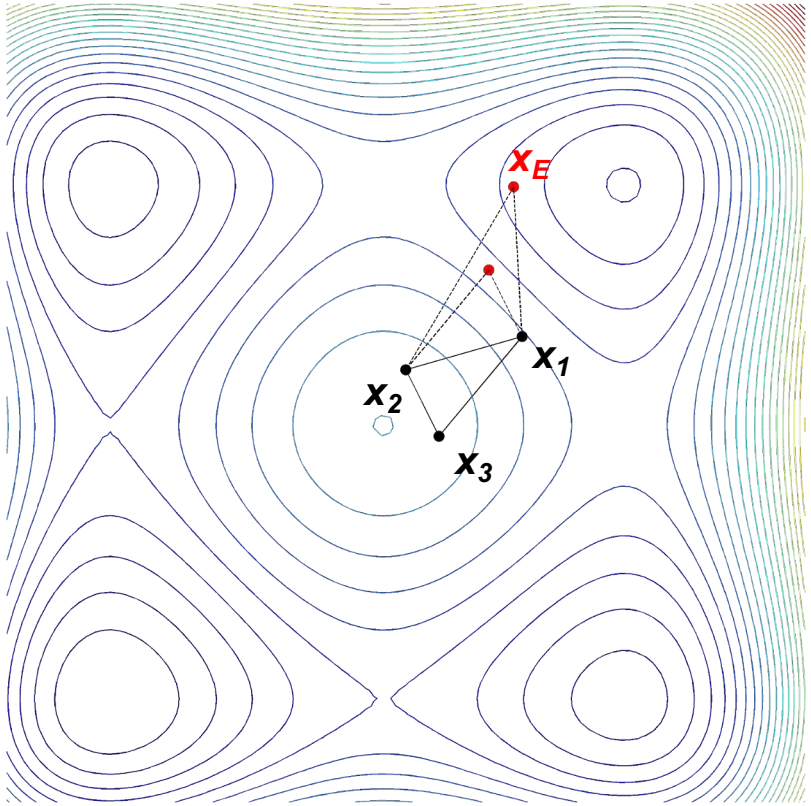
$-0.1400 = f(\vec{x}_3)$

$-0.1644 = f(\vec{x}_2)$

$-1.4618 = f(\vec{x}_1)$

$-1.8343 = f(\vec{x}_R)$

$-2.7545 = f(\vec{x}_E)$



EXPAND

Optimization algorithms

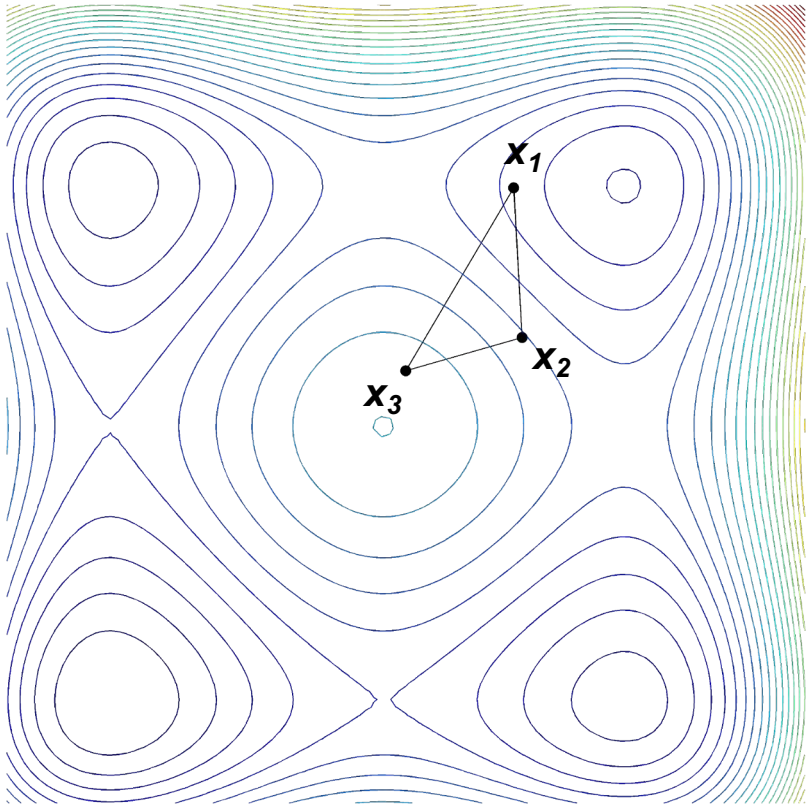
Nelder-Mead simplex method

Simplex has stretched

$-0.1644 = f(\vec{x}_3)$

$-1.4618 = f(\vec{x}_2)$

$-2.7545 = f(\vec{x}_1)$



Optimization algorithms

Nelder-Mead simplex method

- Gradient-free method
- Basic operations: Reflect // Expand // Contract // Reduce
- Triangle in 2D, tetrahedron in 3D, hyper-tetrahedron in higher dimensions
- Converges to local minimum, but lack of solid convergence theory
- Robust (noisy functions)
- Effective for N ≈ O(10) (gradient-based effective for any N)
- Initial triangle needs to be supplied

Example

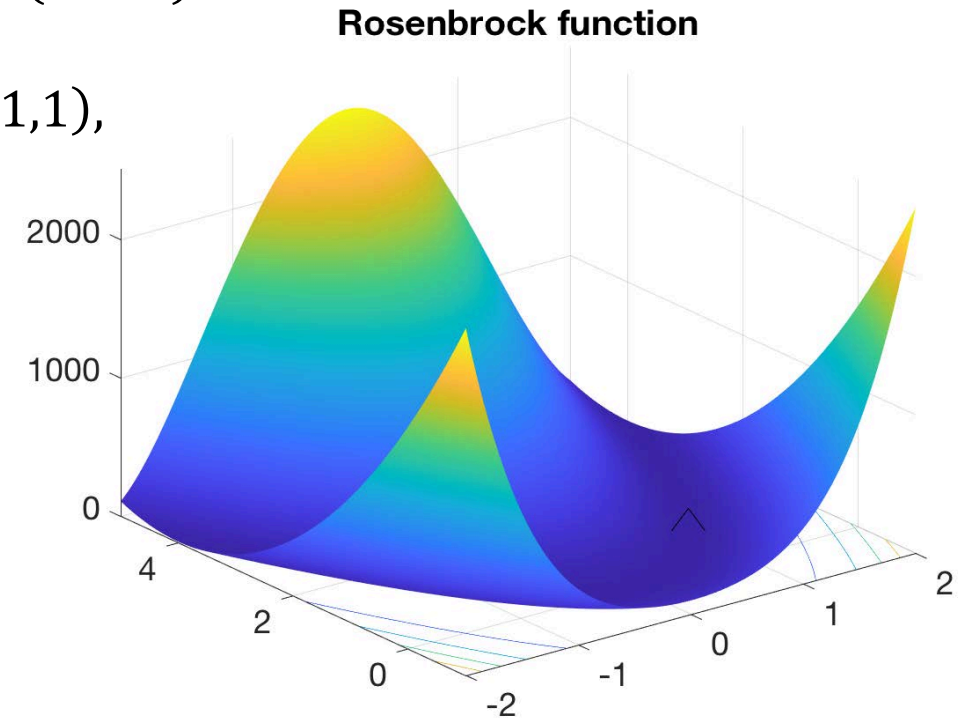
Find minimum for Rosenbrock function:

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

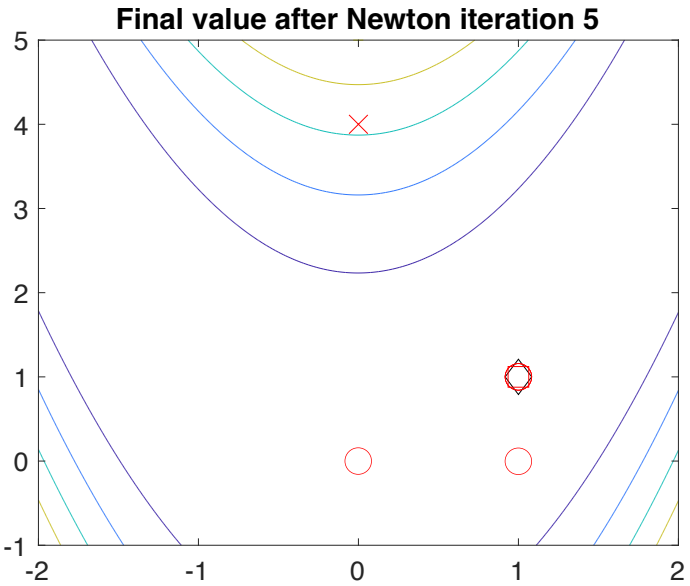
Exact minimum at $(x, y) = (1,1)$,

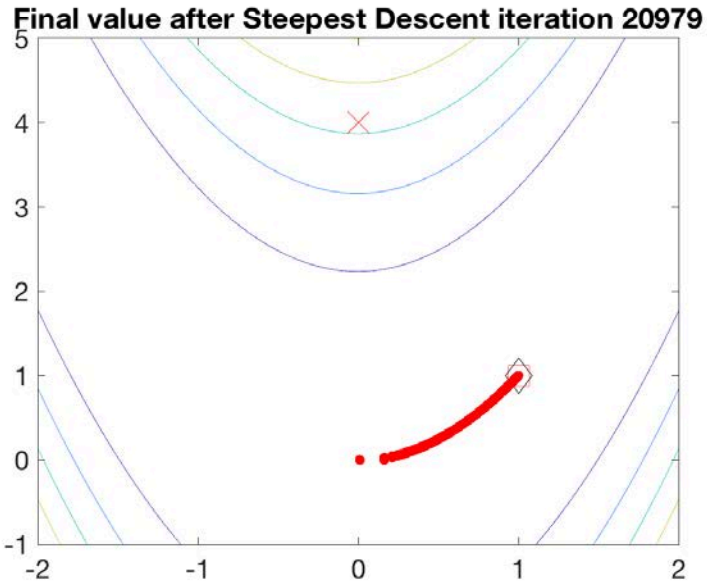
Find minimum, starting at $(x_0, y_0) = (0,4)$, using:

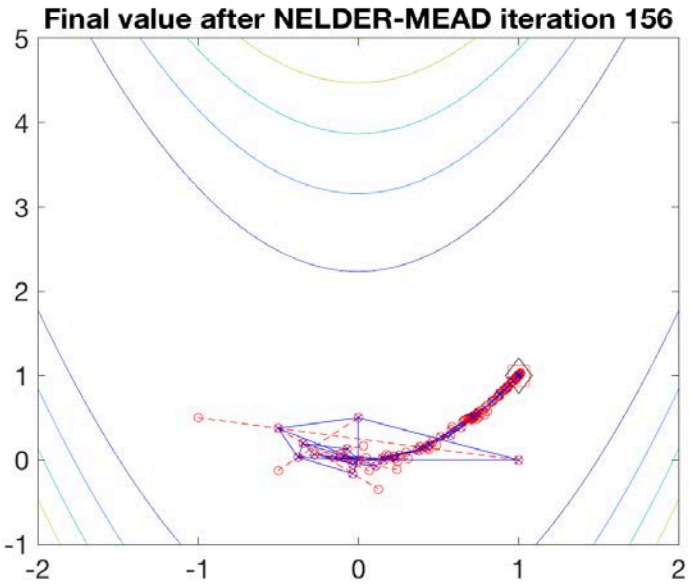
- Newton's method
- Steepest descent
- Nelder-Mead



Example


Final value after Newton iteration 5
5
4
3
2
1
0
-1
-2
-1
0
1
2


Final value after Steepest Descent iteration 20979
5
4
3
2
1
0
-1
-2
-1
0
1
2


Final value after NELDER-MEAD iteration 156
5
4
3
2
1
0
-1
-2
-1
0
1
2