# Programming, numerics and optimization

## Lecture C-2: Unconstrained optimization I

Łukasz Jankowski
ljank@ippt.pan.pl

Institute of Fundamental Technological Research
Room 4.32, Phone +22.8261281 ext. 428

May 4, 2021[1]

[1] Current version is available at http://info.ippt.pan.pl/~ljank.

# Outline

# Outline

# Unconstrained minimization

## Unconstrained minimization problem

Given an objective function $F : \mathbf{D} = \mathbb{R}^n \to \mathbb{R}$, find such $\hat{\mathbf{x}} \in \mathbf{D}$ that

$$\forall_{\mathbf{x} \in \mathbf{D}} F(\hat{\mathbf{x}}) \leq F(\mathbf{x}).$$

# Unconstrained minimization

## Iterative solution process

The basic idea of all classical optimization algorithms

### 1. Initialization

Select any feasible initial point $\mathbf{x}_0$, for example

- a first guess,
- the best point from a set of random points (Monte Carlo),
- zero, etc.

### 2. Iteration

**while** (stop conditions are not satisfied)
**do** having calculated points $\mathbf{x}_0$, $\mathbf{x}_1$, ..., $\mathbf{x}_k$,
compute the successive point $\mathbf{x}_{k+1}$.

# Unconstrained minimization

## Stop conditions

At some step the iteration has to stop. Commonly used stop conditions:

1. No significant improvement of the objective function
$$|F(\mathbf{x}_{k+1}) - F(\mathbf{x}_k)| \leq \varepsilon.$$
2. Sufficiently small gradient norm
$$\|\nabla F(\mathbf{x}_k)\| \leq \varepsilon.$$
3. Steps getting too small
$$\|\mathbf{x}_{k+1} - \mathbf{x}_k\| \leq \varepsilon.$$
4. Practical (time, no of steps, etc.).
5. Solution attained (discrete problems, some constrained optimization problems).

# Unconstrained minimization

## Rate and order of convergence

An optimization algorithm can be “fast” or “slow”.

General performance measures of an optimization algorithm are given by its

- rate of convergence,
- order of convergence.

# Unconstrained minimization

## Rate of convergence

### Rate of convergence

Let $\mathbf{x}_k \to \mathbf{x}^\star$. The convergence rate is $r$, if

$$\frac{\|\mathbf{x}_{k+1} - \mathbf{x}^\star\|}{\|\mathbf{x}_k - \mathbf{x}^\star\|} \longrightarrow r.$$

If the optimization is convergent, the limit $r$ must satisfy $r \in [0, 1]$.

This is sometimes called *Q-convergence* (or Q-linear convergence), where Q stands for "quotient".

| | | |
|---|---|---|
| superlinear | if $r = 0$ | (for example $x_k = \mathrm{e}^{-k^2}$) |
| linear | if $r \in (0, 1)$ | (for example $x_k = \mathrm{e}^{-k}$) |
| sublinear | if $r = 1$ | (for example $x_k = 1/k$) |

# Unconstrained minimization

## Order of convergence

Practical optimization algorithms converge linearly or superlinearly. They can be further classified by their *order of convergence*.

### Order of convergence

Let $\mathbf{x}_k \to \mathbf{x}^\star$. The order of convergence is $p$ ($p \geq 1$), if

$$\frac{\|\mathbf{x}_{k+1} - \mathbf{x}^\star\|}{\|\mathbf{x}_k - \mathbf{x}^\star\|^p} \to \text{const.}$$

Q-convergence requires the error to decrease in each step. A more general notion is R-convergence, which

1. Dominates the sequence of errors by a sequence of scalars, $\|\mathbf{x}_k - \mathbf{x}^\star\| \leq e_k$;
2. Investigates the Q-convergence of $e_k$.

# Unconstrained minimization

## Rate and order of convergence

Typical optimization algorithms have rates/orders of convergence ranging from sublinear (very slow) to quadratic (very quick). If the search point is sufficiently close to the minimum, then typically

| | |
|---|---|
| Below linear | Coordinate descent (or non-convergent) |
| Linear | Steepest descent method |
| Superlinear | Quasi-Newton method |
| Quadratic | Newton and conjugate gradient methods |

Usually, there is a tradeoff between the

- order of convergence (total number of steps) and
- numerical cost of each step.

If sensitivity analysis is very costly, the highest possible order might not always be the best choice.

# Outline

# Two main approaches

In general, all classical optimization algorithms fall into two main categories:

1. line search methods
2. trust region methods

# Line search methods

The basic outline of all *line search* algorithms is:

- Select a feasible point $\mathbf{x}_0$.
- Having calculated points $\mathbf{x}_0$, $\mathbf{x}_1$, ..., $\mathbf{x}_k$, iteratively calculate the successive point $\mathbf{x}_{k+1}$:
  1. Choose the direction $\mathbf{d}_k$ of optimization.
  2. Starting from $\mathbf{x}_k$, perform a (usually approximate) 1D optimization in the direction of $\mathbf{d}_k$, that is find $s_k \in \mathbb{R}$ that sufficiently[2] decreases $f_k$ and $|f_k'|$, where

$$f_k(s) = f(\mathbf{x}_k + s\,\mathbf{d}_k).$$

  Then, take the step

$$\mathbf{x}_{k+1} := \mathbf{x}_k + s_k\,\mathbf{d}_k.$$

  3. Check the stop conditions.

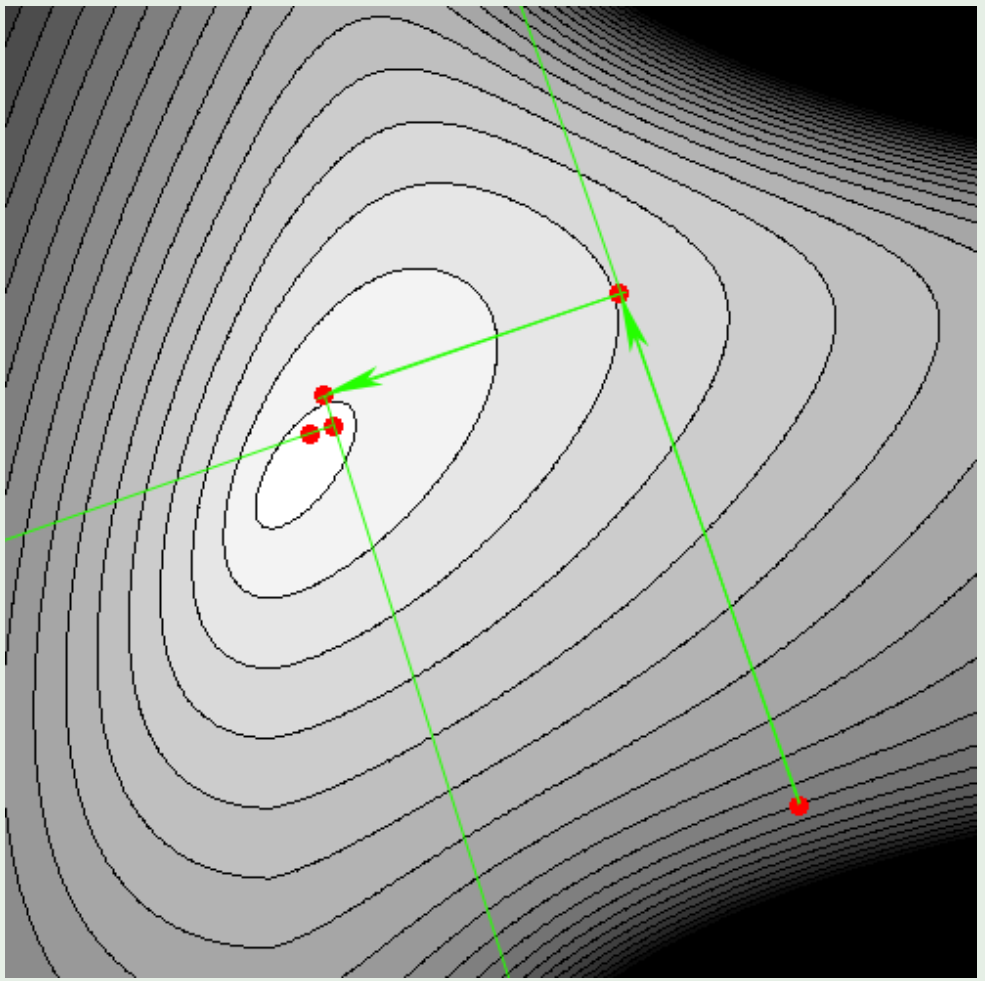
[2] Sufficiently, that is significantly enough to guarantee convergence to the minimum. Exact minimization is usually not necessary; it can be costly and sometimes it can even make the convergence slower.

# Line search methods

1. Choose the direction of optimization.
2. Perform a (usually approximate) 1D optimization in that direction.
3. Check the stop conditions

Two problems:

1. Direction choice
2. Step size



# Trust region methods

- Trust region methods use a simple approximation to $f$, which is trusted only in a given vicinity of $\mathbf{x}_k$.
- This trust region is usually defined as $\{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{D}\mathbf{x}\| \leq \Delta\}$.

Thus, the basic outline of all *trust region* methods is:

- Select a feasible point $\mathbf{x}_0$, as well as
  - trust region shape $\mathbf{D} \in \mathbb{R}^{n \times n}$,
  - trust region maximum and minimum sizes $\Delta_{\max}, \Delta_0 \in \mathbb{R}$,
- Having computed points $\mathbf{x}_0$, $\mathbf{x}_1$, ..., $\mathbf{x}_k$,
  1. approximate $f$ with $f_k$
  2. minimize $f_k$ within the trust region to obtain $\mathbf{x}_{k+1}$
  3. compute $f(\mathbf{x}_{k+1})$ to verify accuracy of approximation and
     - update the size of the trust region
     - if necessary, repeat the current step
  4. check the stop conditions

# Trust region methods

1. Approximate the objective function $f$ in the vicinity of $\mathbf{x}_k$ with a simple-to-optimize[3] function $f_k$,

$$f(\mathbf{x}_k + \mathbf{x}) \approx f_k(\mathbf{x}_k + \mathbf{x}), \qquad \text{for } \|\mathbf{D}\mathbf{x}\| \leq \Delta_k.$$

   The vicinity of $\mathbf{x}_k$ in which we trust the approximation is called the trust region. Its size in the $k$th step is defined by $\Delta_k$.

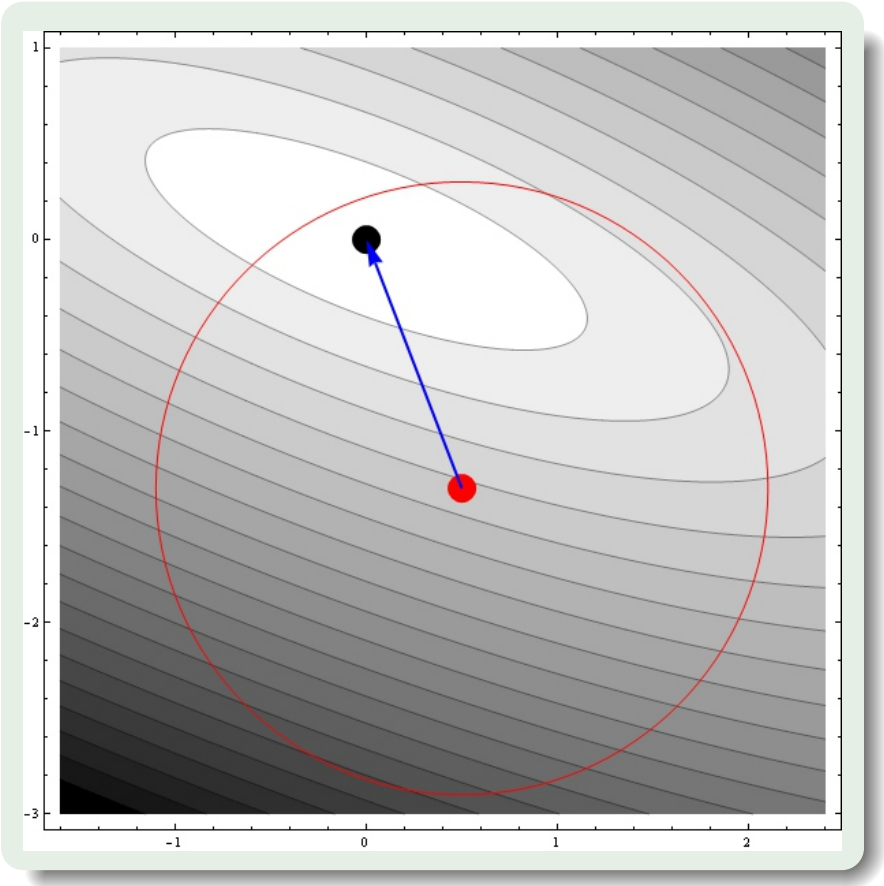
2. Set $\mathbf{x}_{k+1}$ to the minimum of $f_k$ within the current trust region

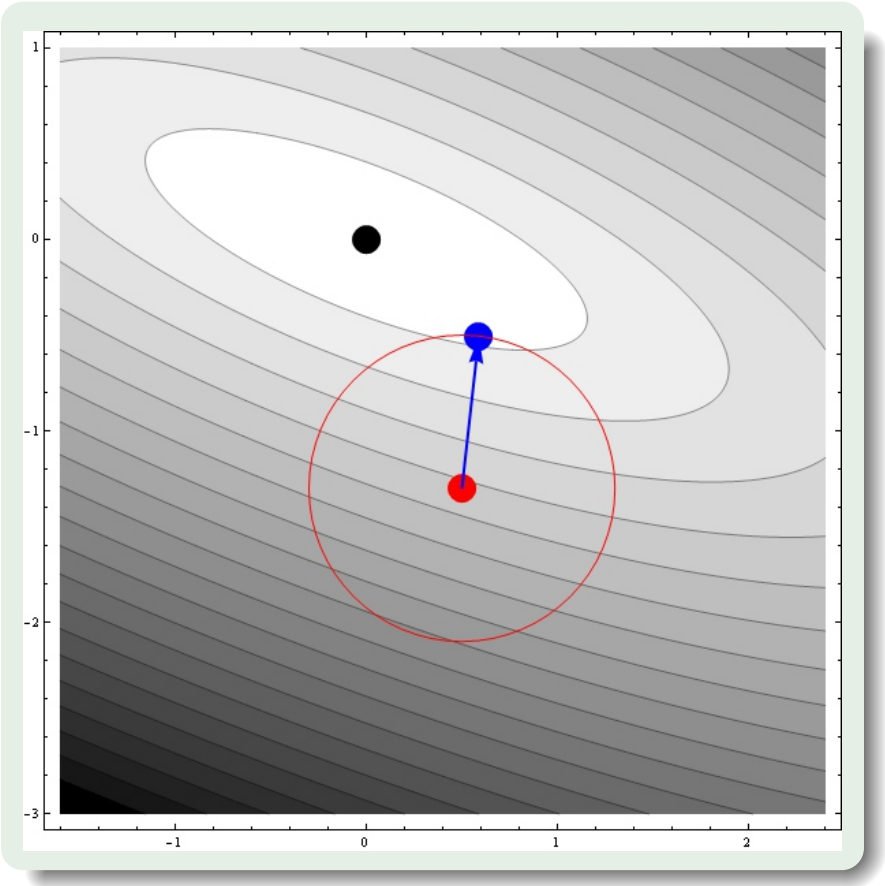
$$\mathbf{x}_{k+1} := \mathbf{x}_k + \operatorname*{arg\,min}_{\|\mathbf{D}\mathbf{x}\| \leq \Delta_k} f_k(\mathbf{x}_k + \mathbf{x}).$$

[3] Usually quadratic: $f_k(\mathbf{x}_k + \mathbf{x}) := f(\mathbf{x}_k) + \mathbf{x}^\mathsf{T} \nabla f(\mathbf{x}_k) + \frac{1}{2}\mathbf{x}^\mathsf{T} \nabla^2 f(\mathbf{x}_k)\mathbf{x}$, but other choices are also possible (e.g. rational).

# Trust region methods

## Size of the trust region does matter: ease of minimization



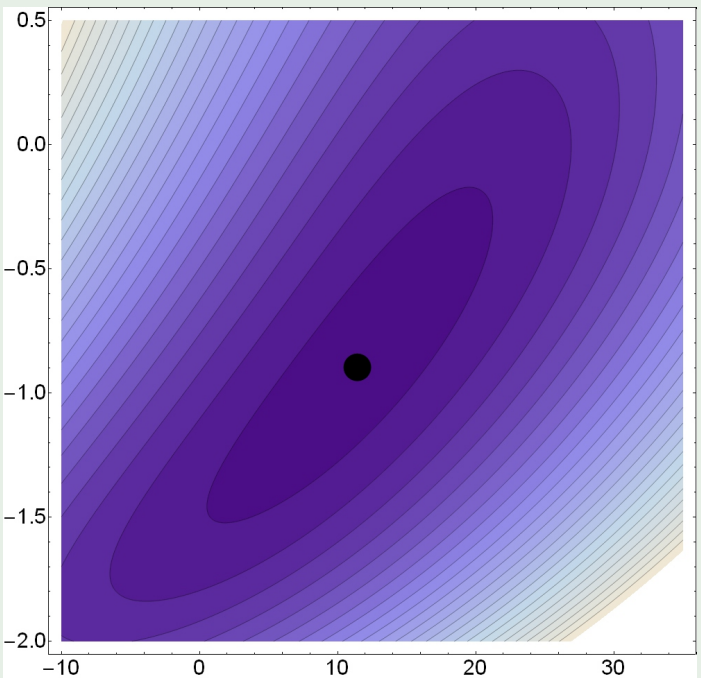

- If $\Delta_k$ is large, the minimum of $f_k$ lies inside the trust region and can be usually easily found.
- If it is small, the minimum lies on the border (constraints!).

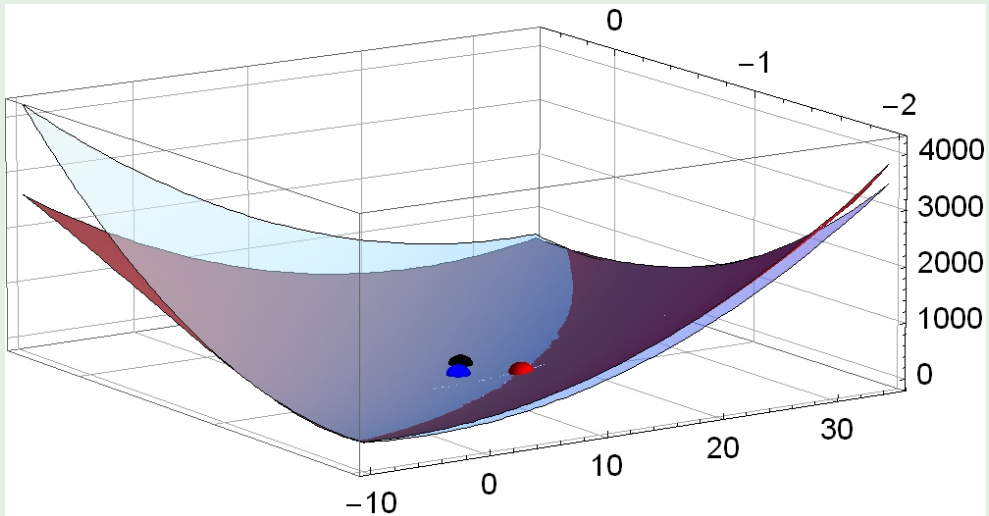
# Trust region methods

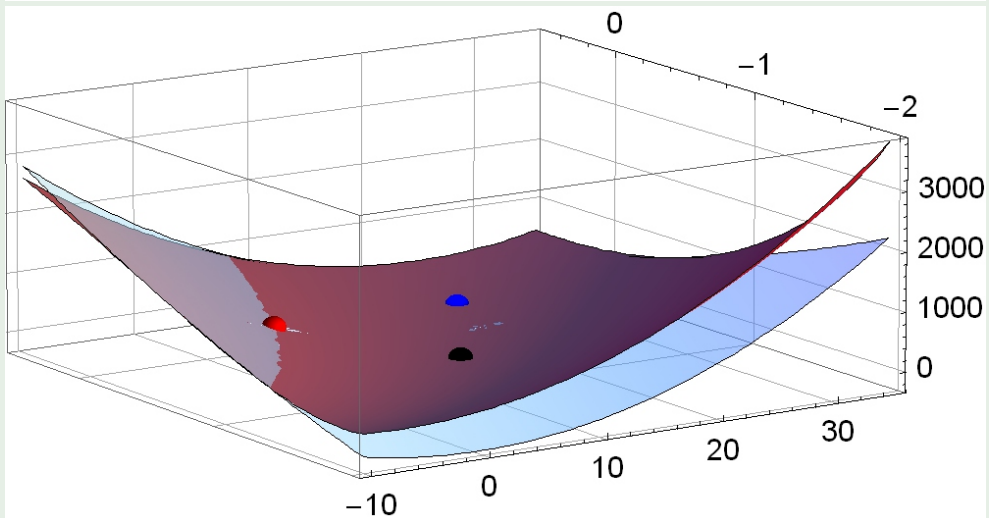
## Size of the trust region does matter: accuracy

Size of the trust region expresses our trust in the accuracy of approximation.

### Freudenstein & Roth function



# Trust region methods

## Trust region size

3. Chose the new trust region size $\Delta_{k+1}$ based on

$$\rho = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_{k+1})}{f_k(\mathbf{x}_k) - f_k(\mathbf{x}_{k+1})} = \frac{\text{actual improvement}}{\text{predicted improvement}}.$$

   If $\rho < 0$, repeat the approximation with a smaller $\Delta_k$.

4. Check the stop conditions.

# Trust region methods

## Constrained vs. unconstrained approach

Trust region methods rely on constrained minimization of $f_k(\mathbf{x}_k + \mathbf{x})$, subject to trust region constraint $\|\mathbf{D}\mathbf{x}\| \leq \Delta_k$.

In general, this is equivalent to unconstrained minimization of

$$f_k(\mathbf{x}_k + \mathbf{x}) + \alpha_k \mathbf{x}^\mathsf{T} \mathbf{D} \mathbf{x}, \qquad (\star)$$

where $\alpha > 0$ controls the trust region size[4].

- If $f_k$ is a quadratic approximation, minimization of $(\star)$ reduces to solving a linear system.
- This approach is used by the modified Newton method and the Levenberg-Marquardt method (Lecture C-3).

[4] $\Delta_k$ is a non-increasing function of $\alpha_k$

# Outline

3. Line search — search directions
   - Steepest descent
   - Conjugate direction
   - Newton direction

# Search directions

In general, if gradient/Hessian is available, there are three common choices for the next search direction:

1. steepest descent direction (slow),
2. direction conjugate to the previous search direction,
3. Newton or quasi-Newton direction (quick).

If gradient/Hessian is not available, an approximation can be used.

# Search directions

## Steepest descent direction

The steepest descent direction is the most obvious search direction.

**The steepest descent direction**

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

It is based on the linear approximation to the objective function:

$$f(\mathbf{x}_k + \mathbf{x}) \approx f(\mathbf{x}_k) + \mathbf{x}^{\mathsf{T}} \nabla f(\mathbf{x}_k).$$

It gives rise to the steepest descent optimization method (in each step simply choose the steepest descent direction), which is intuitive, however can be very slow for difficult objective functions.

# Search directions

## Steepest descent direction

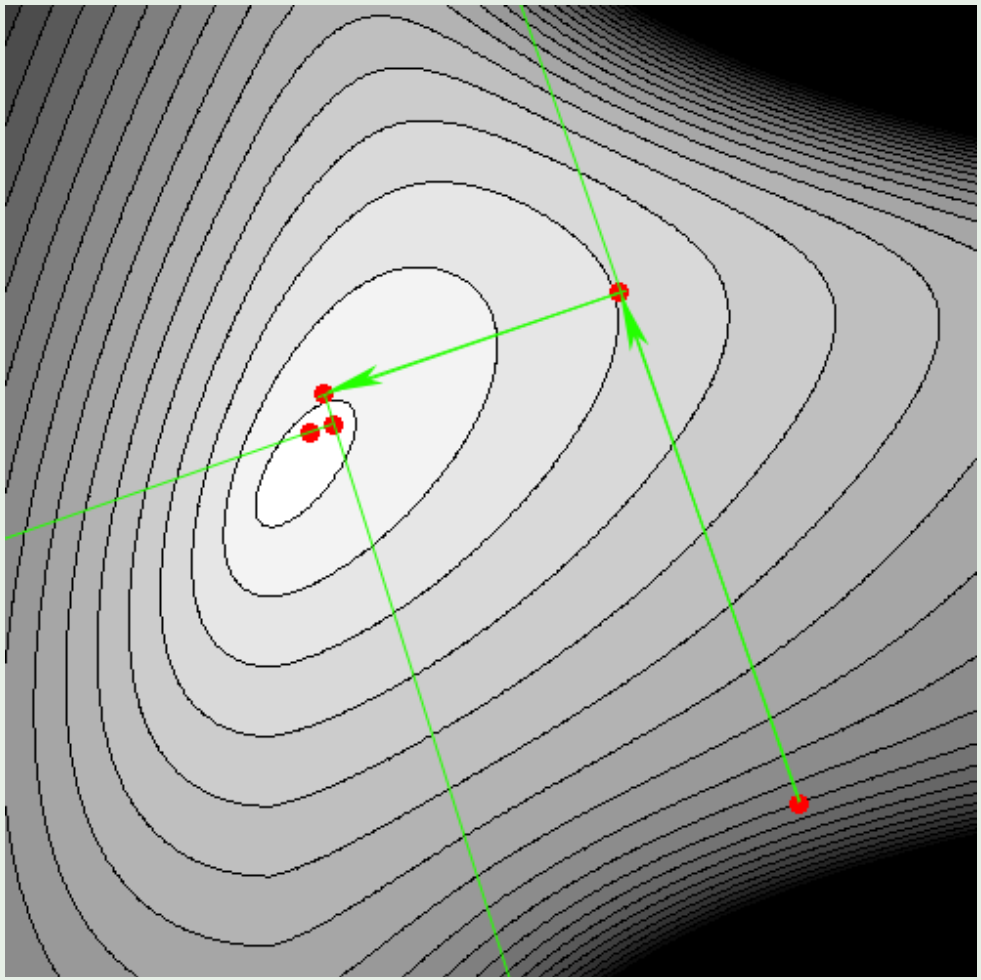
If the exact minimum $\mathbf{x}_{k+1}$ of $f$ along the previous direction $\mathbf{d}_k$ is found,

$$\mathbf{x}_{k+1} := \mathbf{x}_k + s\,\mathbf{d}_k,$$

$$0 = \frac{d}{ds} f(\mathbf{x}_k + s\,\mathbf{d}_k),$$

the steepest descent direction at the line minimum, $-\nabla f(\mathbf{x}_{k+1})$, is perpendicular to $\mathbf{d}_k$:

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



# Search directions

## Conjugate direction

Any (smooth enough) function can be approximated by a quadratic form, that is by its second-order Taylor series,

$$f(\mathbf{x}_k + s\,\mathbf{d}_k) \;\approx\; f(\mathbf{x}_k) + s\,\mathbf{d}_k^{\mathsf{T}}\nabla f(\mathbf{x}_k) + \frac{1}{2}s^2\mathbf{d}_k^{\mathsf{T}}\nabla^2 f(\mathbf{x}_k)\,\mathbf{d}_k.$$

The gradient of the approximation is

$$\nabla f(\mathbf{x}_k + s\,\mathbf{d}_k) \;\approx\; \nabla f(\mathbf{x}_k) + s\,\nabla^2 f(\mathbf{x}_k)\,\mathbf{d}_k.$$

# Search directions

## Conjugate direction

If $f(\mathbf{x}_k)$ is the exact minimum along the previous search direction $\mathbf{d}_{k-1}$, the gradient at the minimum $\mathbf{x}_k$ is perpendicular to the search direction $\mathbf{d}_{k-1}$,

$$\mathbf{d}_{k-1}^{\mathsf{T}} \nabla f(\mathbf{x}_k) = 0.$$

It is reasonable to expect that the minimization along the next direction $\mathbf{d}_k$ does not jeopardize the minimization along the previous direction $\mathbf{d}_{k-1}$. Therefore, the gradient in the points along the new search direction $\mathbf{d}_k$ should still stay perpendicular to $\mathbf{d}_{k-1}$. Thus we require that

$$\begin{aligned} 0 &= \mathbf{d}_{k-1}^{\mathsf{T}} \nabla f(\mathbf{x}_k + s\,\mathbf{d}_k) \\ &\approx \mathbf{d}_{k-1}^{\mathsf{T}} \left[\nabla f(\mathbf{x}_k) + s\,\nabla^2 f(\mathbf{x}_k)\,\mathbf{d}_k\right] \\ &= s\,\mathbf{d}_{k-1}^{\mathsf{T}} \nabla^2 f(\mathbf{x}_k)\,\mathbf{d}_k. \end{aligned}$$

# Search directions

## Conjugate direction

### Conjugate direction

Every direction $\mathbf{d}_k$, which satisfies

$$0 = \mathbf{d}_{k-1}^{\mathsf{T}} \nabla^2 f(\mathbf{x}_k)\, \mathbf{d}_k,$$

is said to be conjugate to $\mathbf{d}_{k-1}$ (at $\mathbf{x}_k$ with respect to $f$).

### Conjugate set

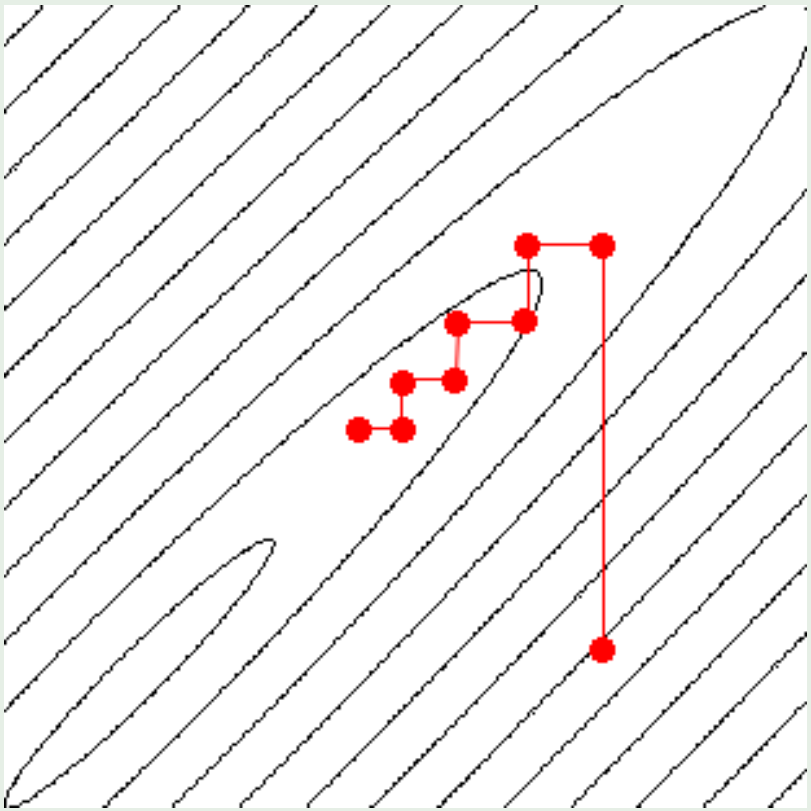
A set of vectors $\mathbf{d}_i$ that pairwise satisfy $0 = \mathbf{d}_i^{\mathsf{T}} \nabla^2 f(\mathbf{x})\, \mathbf{d}_j$, is called a *conjugate set* (at $\mathbf{x}$ with respect to $f$).

If $f$ is an $n$-dimensional quadratic form, then $n$ *global* conjugate directions can be always found. If $f$ is also positive definite, then single exact optimization along each of them leads directly to the global minimum.

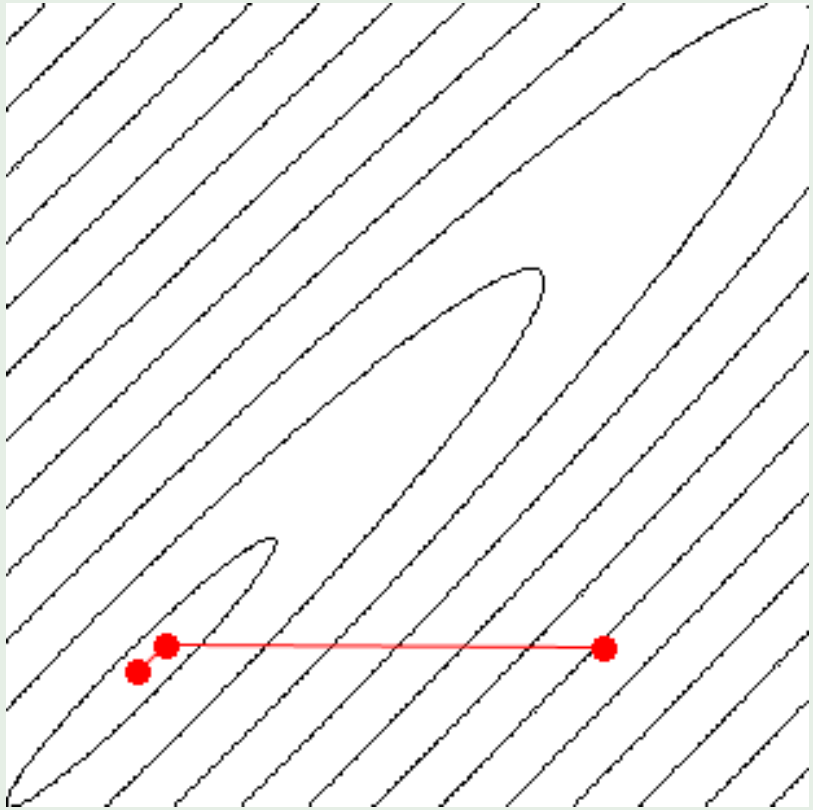
# Search directions

## Conjugate direction

### Coordinate/steepest descent



### Conjugate directions



# Search directions

## Newton direction

### Newton direction

$$\mathbf{d}_k = -\left[\nabla^2 f(\mathbf{x}_k)\right]^{-1} \nabla f(\mathbf{x}_k)$$

The Newton direction is based on the quadratic approximation to the objective function,

$$f(\mathbf{x}_k + \mathbf{d}_k) \approx f(\mathbf{x}_k) + \mathbf{d}_k^{\mathrm{T}} \nabla f(\mathbf{x}_k) + \frac{1}{2} \mathbf{d}_k^{\mathrm{T}} \nabla^2 f(\mathbf{x}_k)\, \mathbf{d}_k.$$

If the Hessian $\nabla^2 f(\mathbf{x}_k)$ is positive definite (the function is convex), the minimum is found by solving

$$\mathbf{0} = \nabla f(\mathbf{x}_k + \mathbf{d}_k) \approx \nabla f(\mathbf{x}_k) + \nabla^2 f(\mathbf{x}_k)\, \mathbf{d}_k.$$

There is a natural step length in the Newton direction (unlike in the case of the steepest descent or the conjugate directions).

# Search directions

## Quasi-Newton direction

### Quasi-Newton direction

$$\mathbf{d}_k = -\mathbf{H}_k^{-1}\nabla f(\mathbf{x}_k), \qquad \text{where } \mathbf{H}_k \approx \nabla^2 f(\mathbf{x}_k)$$

The quasi-Newton direction is computed using an approximation to the exact Hessian or to its inverse.

The approximation is based on the gradients and can be either

- iteratively updated in each step using the previous step length and the related change of the gradient (e.g. BFGS method) or
- in least-squares problems, built from scratch in every step using some gradient-related information from the current step only (Gauss-Newton or Levenberg-Marquardt methods).

# Outline

4. Line search — step size
   - Wolfe and strong Wolfe conditions
   - Goldstein and Price conditions
   - Implementation
   - Convergence

# Step size

At every step of the optimization, when the search direction $\mathbf{d}_k$ is already decided, the next problem is to find a step size $s_k$ (along this direction) that decreases

$$f_k(s) = f(\mathbf{x}_k + s\,\mathbf{d}_k)$$

and guarantees that $\{\mathbf{x}_i\}$ converges to a minimum.

The first natural idea is to look for the exact minimizer along $\mathbf{d}_k$.

- It is often too expensive to find.
- In certain cases, it may even make the convergence slower.

On the other hand, requiring only *any* reduction of the objective function, $f_k(s_k) < f_k(0)$, is not enough, since the steps can be then too short and the optimization may stall far from the minimum.

# Step size

Practical conditions require the step size to

1. sufficiently decrease the objective function and
2. be not too short

The most commonly used conditions for inexact line search are

- Wolfe (or strong Wolfe) conditions that are used when the gradient is numerically inexpensive
- Goldstein and Price conditions that are used when the gradient is numerically costly.

# Wolfe conditions

Let $f_k(s) = f(\mathbf{x}_k + s\,\mathbf{d}_k)$.

## Wolfe conditions

The step size $s_k$ satisfies the Wolfe conditions, if it

- sufficiently decreases the objective function,

$$f_k(s_k) \leq f_k(0) + c_1\, s_k\, f_k'(0). \qquad \text{(W1)}$$

  This condition prevents also too large steps and is called the *Armijo condition*.

- is not too short,

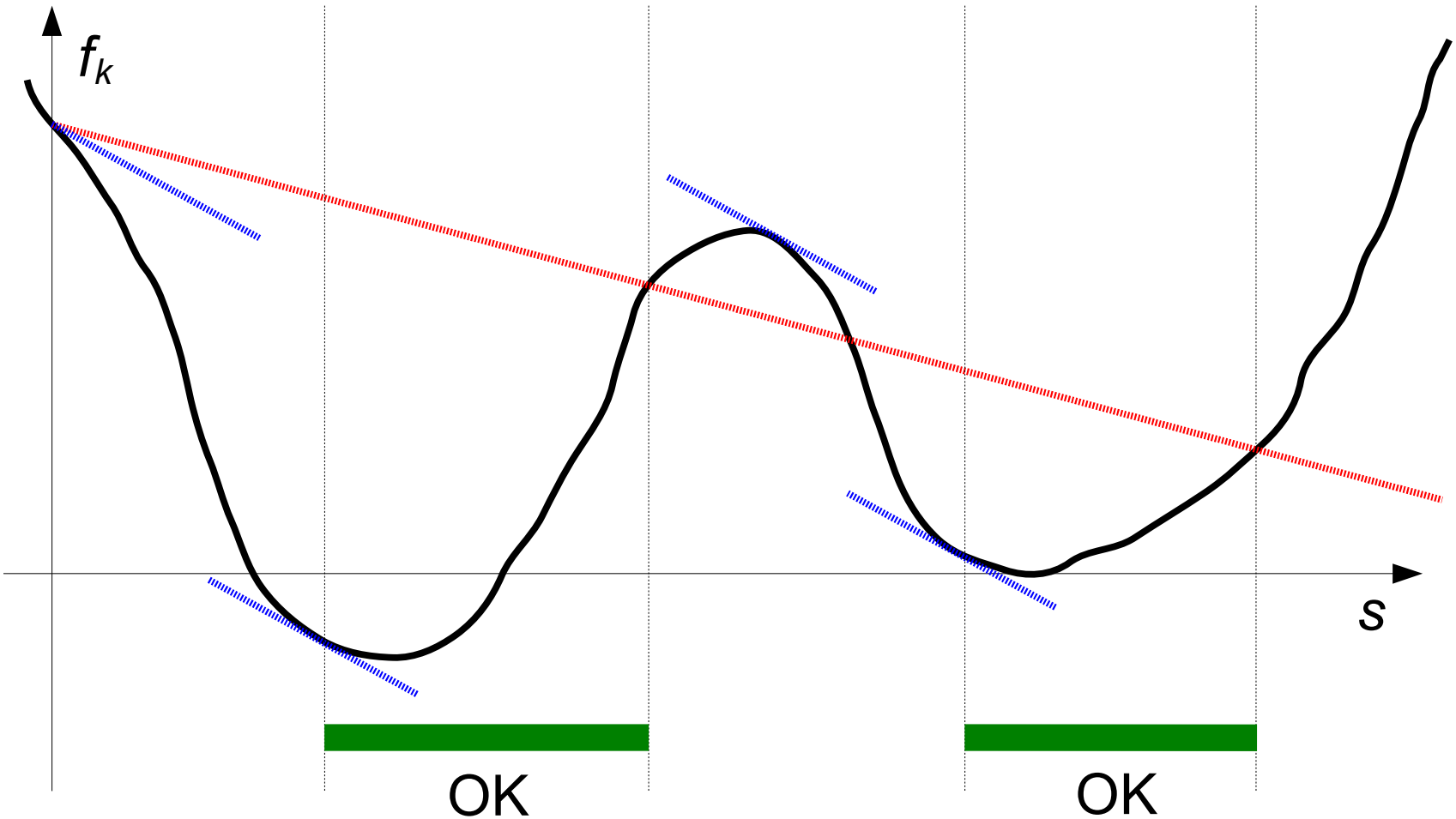
$$f_k'(s_k) \geq c_2 f_k'(0). \qquad \text{(W2)}$$

The constants $c_1$ and $c_2$ should satisfy $0 < c_1 < c_2 < 1$. It is usually recommended that $c_1 < 0.5$.

$f_k'(s)$ denotes the directional derivative, $f_k'(s) = \mathbf{d}_k^{\mathsf{T}} \nabla f(\mathbf{x}_k + s\,\mathbf{d}_k)$.

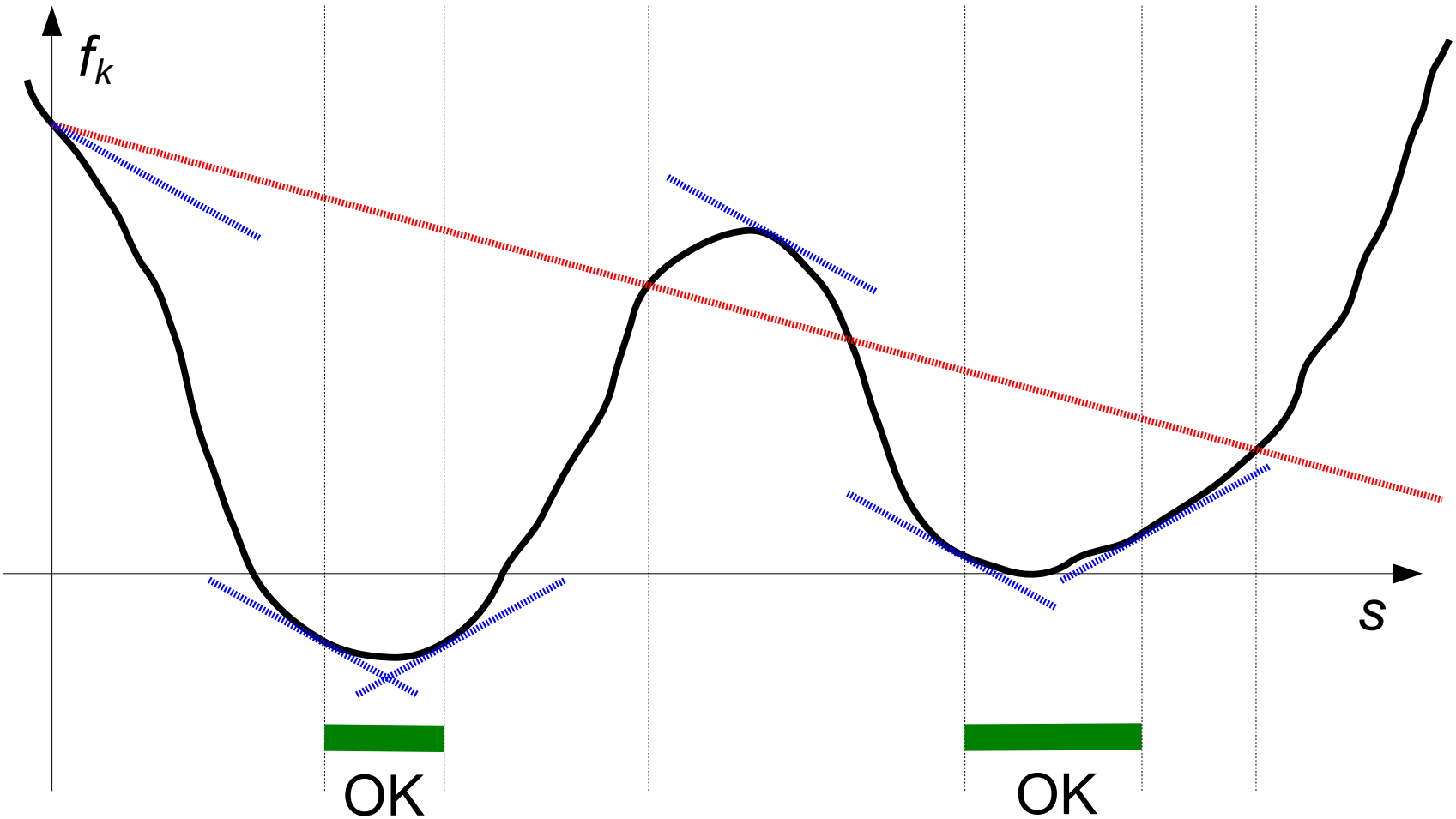
# Wolfe conditions

(W1): $f_k(s_k) \leq f_k(0) + c_1\, s_k\, f_k'(0)$ (W2): $f_k'(s_k) \geq c_2 f_k'(0)$



# Strong Wolfe conditions

(W1): $f_k(s_k) \leq f_k(0) + c_1\, s_k\, f_k'(0)$ (W2): $|f_k'(s_k)| \leq c_2 |f_k'(0)|$



# Goldstein and Price conditions

The second of the Wolfe (or strong Wolfe) conditions

$$f_k(s_k) \leq f_k(0) + c_1 s_k f_k'(0),$$
$$f_k'(s_k) \geq c_2 f_k'(0),$$

requires gradient computations ($f_k'(s_k)$) at each trial step $s_k$. If the gradient is numerically costly, an approximation can be used,

$$f_k'(s_k) \approx \frac{f_k(s_k) - f_k(0)}{s_k},$$

which yields *Goldstein and Price conditions*.

# Goldstein and Price conditions

Let $f_k(s) = f(\mathbf{x}_k + s\,\mathbf{d}_k)$.

## Goldstein and Price conditions

The step size $s_k$ satisfies the Goldstein and Price conditions, if

$$f_k(s_k) \leq f_k(0) + c_1 s_k f_k'(0), \tag{G1}$$

$$\frac{f_k(s_k) - f_k(0)}{s_k} \geq c_2 f_k'(0). \tag{G2}$$

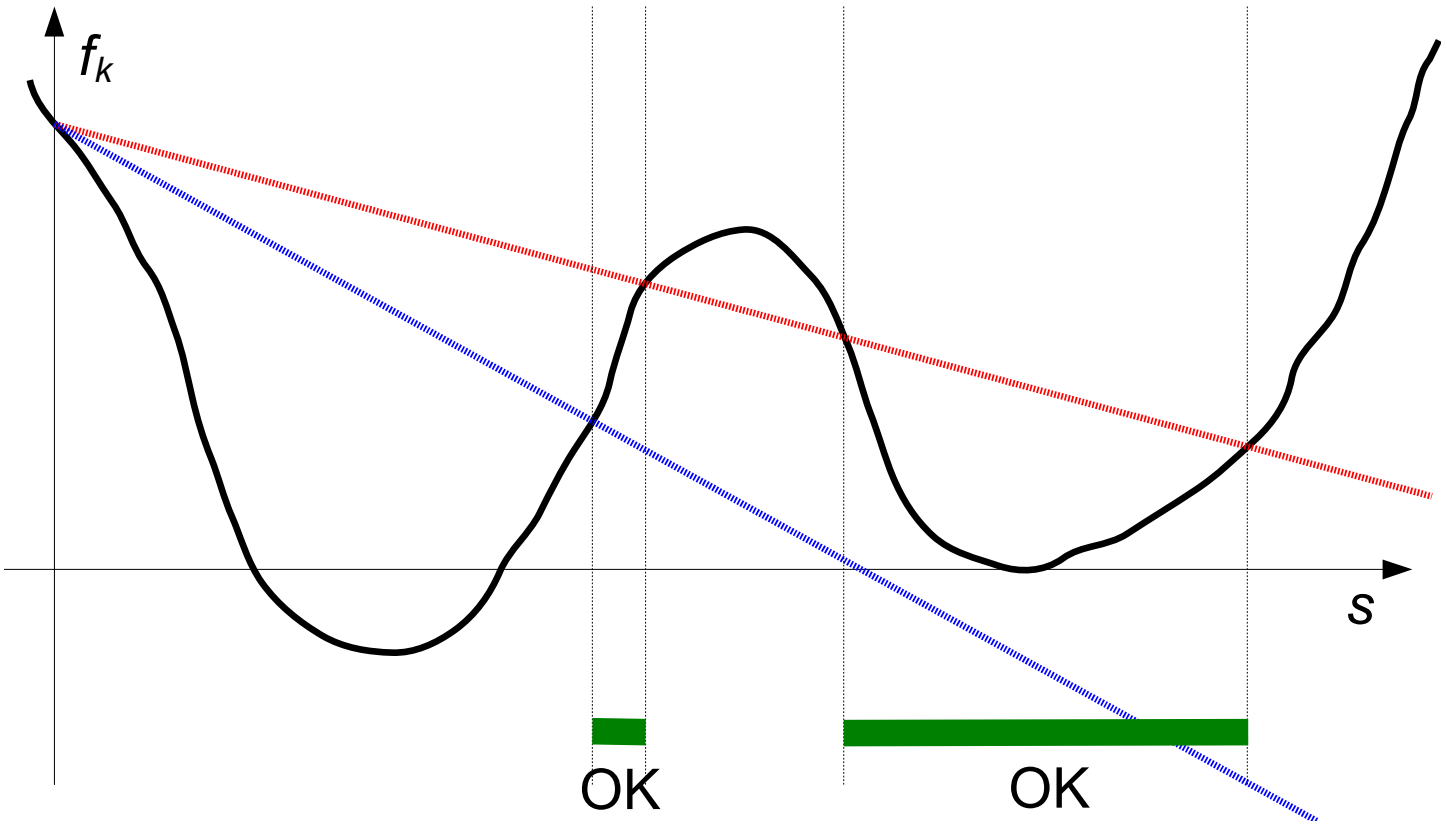
Goldstein and Price conditions are equivalent to two bounds on the average slope,

$$c_2 f_k'(0) \leq \frac{f_k(s_k) - f_k(0)}{s_k} \leq c_1 f_k'(0).$$

The constants should satisfy $0 < c_1 < c_2 < 1$ and are often chosen so that $c_1 + c_2 = 1$.

# Goldstein and Price conditions

$$c_2 f_k'(0) \leq \frac{f_k(s_k) - f_k(0)}{s_k} \leq c_1 f_k'(0).$$




Note that Goldstein and Price conditions may exclude all local minimizers of $f$.

# Implementation

In Wolfe, strong Wolfe as well as in Goldstein and Price conditions:

**First** condition ensures a sufficient decrease of the objective function and prevents too long steps

**Second** condition ensures the step is not too short.

that is, if the

- first condition fails, the step should be decreased,
- second condition fails, the step should be increased.

If $f$ is smooth and bounded from below, there exist step lengths that satisfy the Wolfe (strong Wolfe, Goldstein and Price) conditions.

# Implementation — backtracking

The *backtracking* approach might be easier implementable and still prevents too short steps. Initially, select

- the initial step length $s_{\text{ini}}$
- the Armijo constant $c \in (0, 1)$
- the contraction factor $\rho \in (0, 1)$

In each iteration:

- $s = s_{\text{ini}}$
- **while** $f_k(s) > f_k(0) + c\, s\, f_k'(0)$ **do** $s = \rho s$
- $s_k = s$

In practice

- The initial step size $s_{\text{ini}}$ can be varied in successive iterations (but should not converge to zero).
- The contraction factor $\rho$ might be allowed to vary in each iteration, provided that $0 < \rho_{\min} \leq \rho \leq \rho_{\max} < 1$.

# Convergence

after Nocedal, Wright

Let the objective function $f$ be bounded from below and have Lipschitz-continuous gradients (bounded slopes of gradient norms). If the line search satisfies Wolfe (or strong Wolfe, or Goldstein and Price) conditions, then

$$\sum_k \|\nabla f(\mathbf{x}_k)\|^2 \cos^2 \alpha_k < \infty, \qquad (*)$$

where $\alpha_i$ is the angle between the steepest descent and the search directions in the $i$th step,

$$\cos \alpha_i = \frac{-\mathbf{d}_i^{\mathsf{T}} \nabla f(\mathbf{x}_i)}{\|\mathbf{d}_i\| \|\nabla f(\mathbf{x}_i)\|}.$$

Equation (*) is called the *Zoutendijk condition* and implies that

$$\|\nabla f(\mathbf{x}_k)\| \cos \alpha_k \to 0.$$

# Convergence

## after Nocedal, Wright

The *Zoutendijk condition*, provided the search directions do not become too perpendicular to gradients[5], implies $\|\nabla f(\mathbf{x}_k)\| \to 0$.

Therefore, the search is attracted to a stationary point for:

- *Steepest descent* line search, since $\cos \alpha_k = 1$.
- *Newton* and *quasi-Newton* line searches, if all Hessians $\nabla^2 f(\mathbf{x}_k)$ (or approximated Hessians $\mathbf{H}_k$)
  - are positive definite and
  - have uniformly bounded condition numbers, $\|\mathbf{H}_k\|\|\mathbf{H}_k^{-1}\| < c$, since then $\cos \alpha_k \geq \frac{1}{c}$.

For *conjugate gradient* line searches it is possible to prove that only a subsequence of gradient norms converges to zero,

$$\liminf_{k\to\infty} \|\nabla f(\mathbf{x}_k)\| = 0.$$

[5] That is $\cos \alpha_k \geq \epsilon > 0$, at least for $k$ large enough.

# Outline

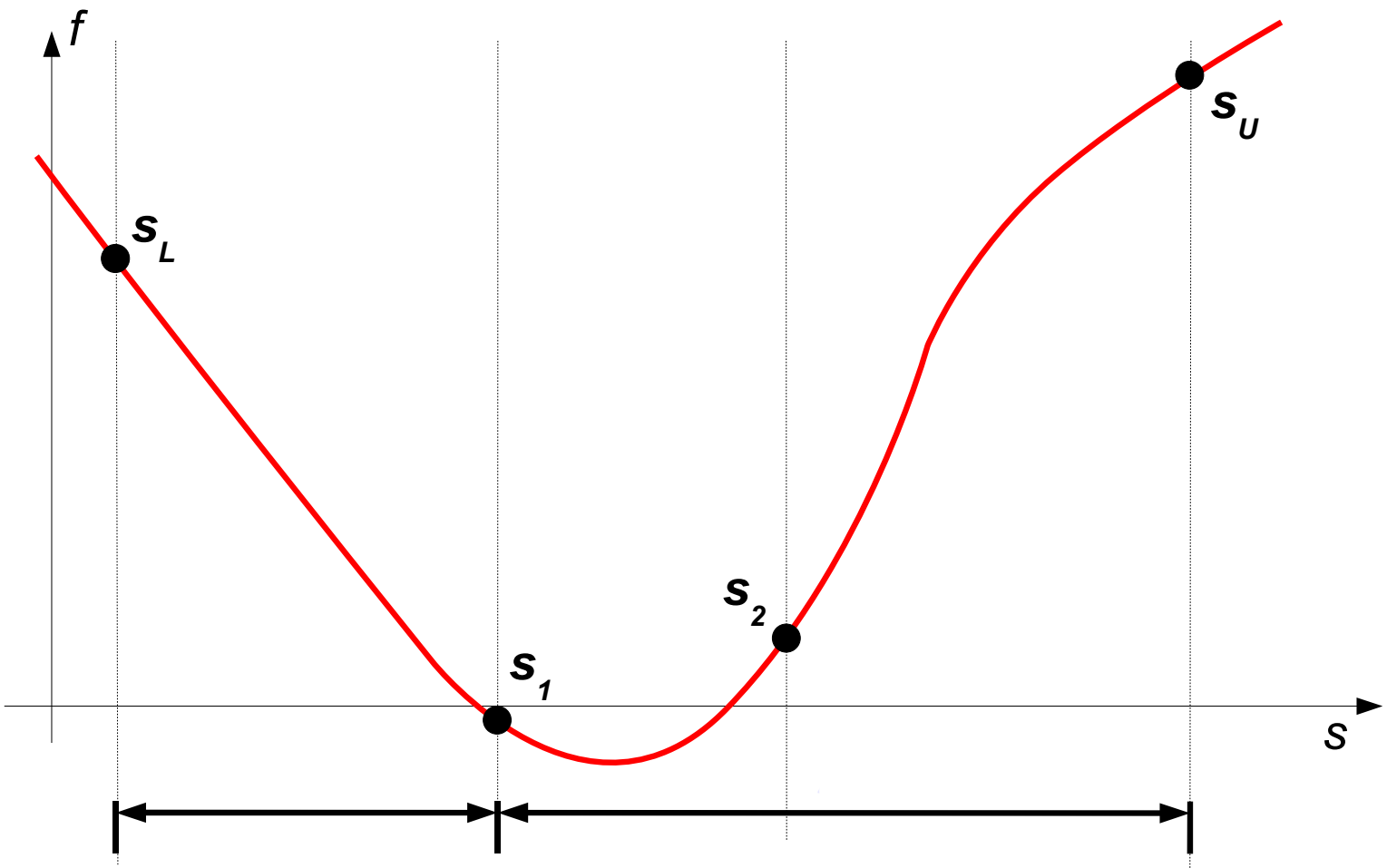
# 1D optimization methods

- $0^{\text{th}}$ order (non gradient-based, only $f$ known)
  - golden section and bracketing
  - quadratic interpolation or approximation
- $1^{\text{st}}$ order ($f$ and $f'$ known)
  - steepest descent method
  - quasi-Newton and modified quasi-Newton method
- $2^{\text{nd}}$ order ($f$, $f'$ and $f''$ known)
  - Newton method
  - modified Newton method

Hybrid methods (like the Brent's method) are also possible.

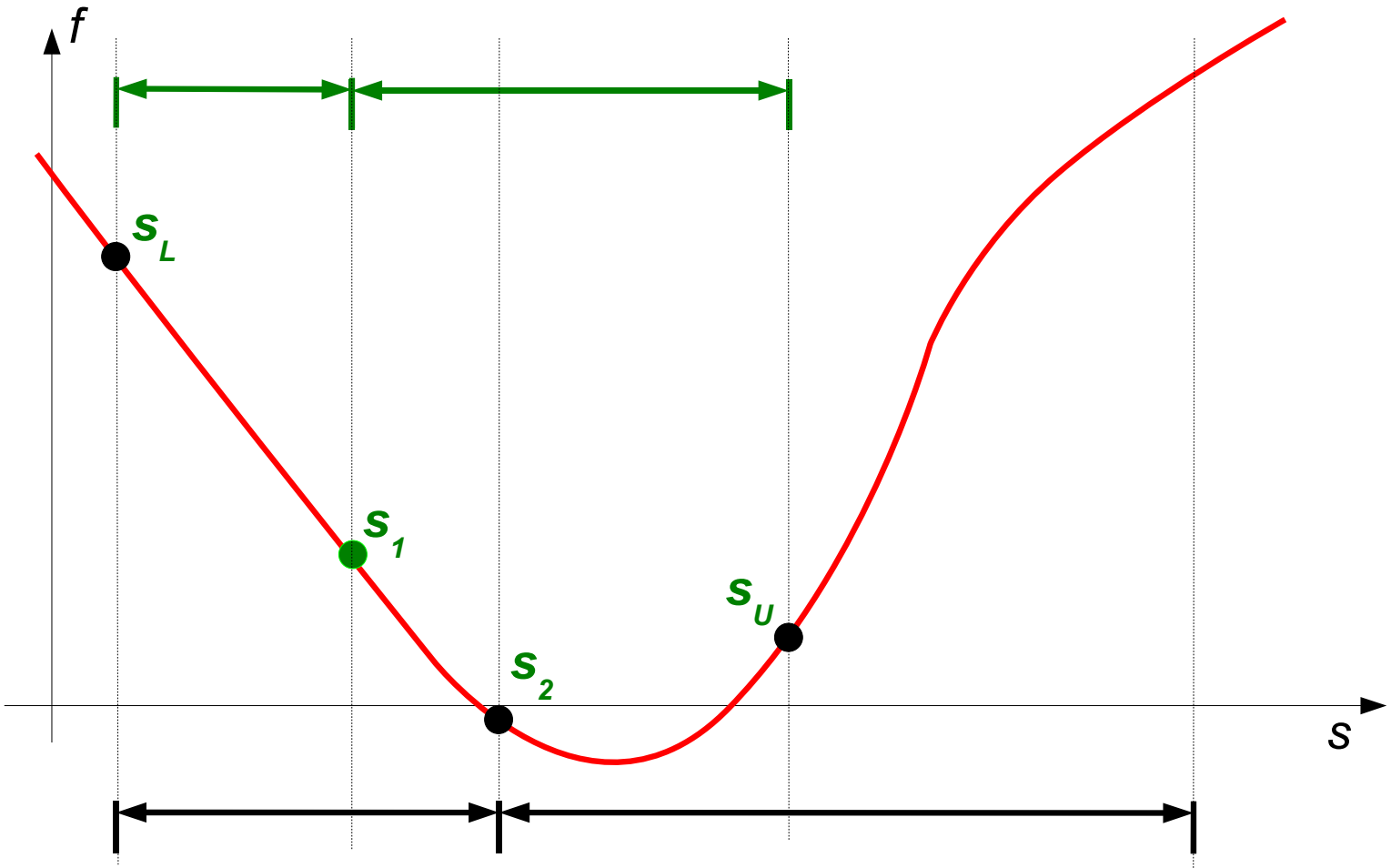
In principle, most of these methods should be combined with step length conditions checking. Then they can be also used for inexact search.

# Zero-order methods — golden section




- The minimum must be bracketed and the function unimodal
- Exact error estimation (minimum bounds)

# Zero-order methods — golden section




- The minimum must be bracketed and the function unimodal
- Exact error estimation (minimum bounds)

# Zero-order methods — golden section

Assumptions:

1. The minimum is bracketed between given $s^L$ and $s^U$
2. The function $f(s)$ is unimodal between $s^L$ and $s^U$

Initialization:

$s^U > s^L \geq 0$ bracketing the minimum of $f(s)$

$\phi := \frac{1+\sqrt{5}}{2}$ (the golden fraction)

$s_1 := s^U - \frac{s^U - s^L}{\phi}$ $\quad$ $s_2 := s^L + \frac{s^U - s^L}{\phi}$

While $s^U - s^L > \epsilon$ do

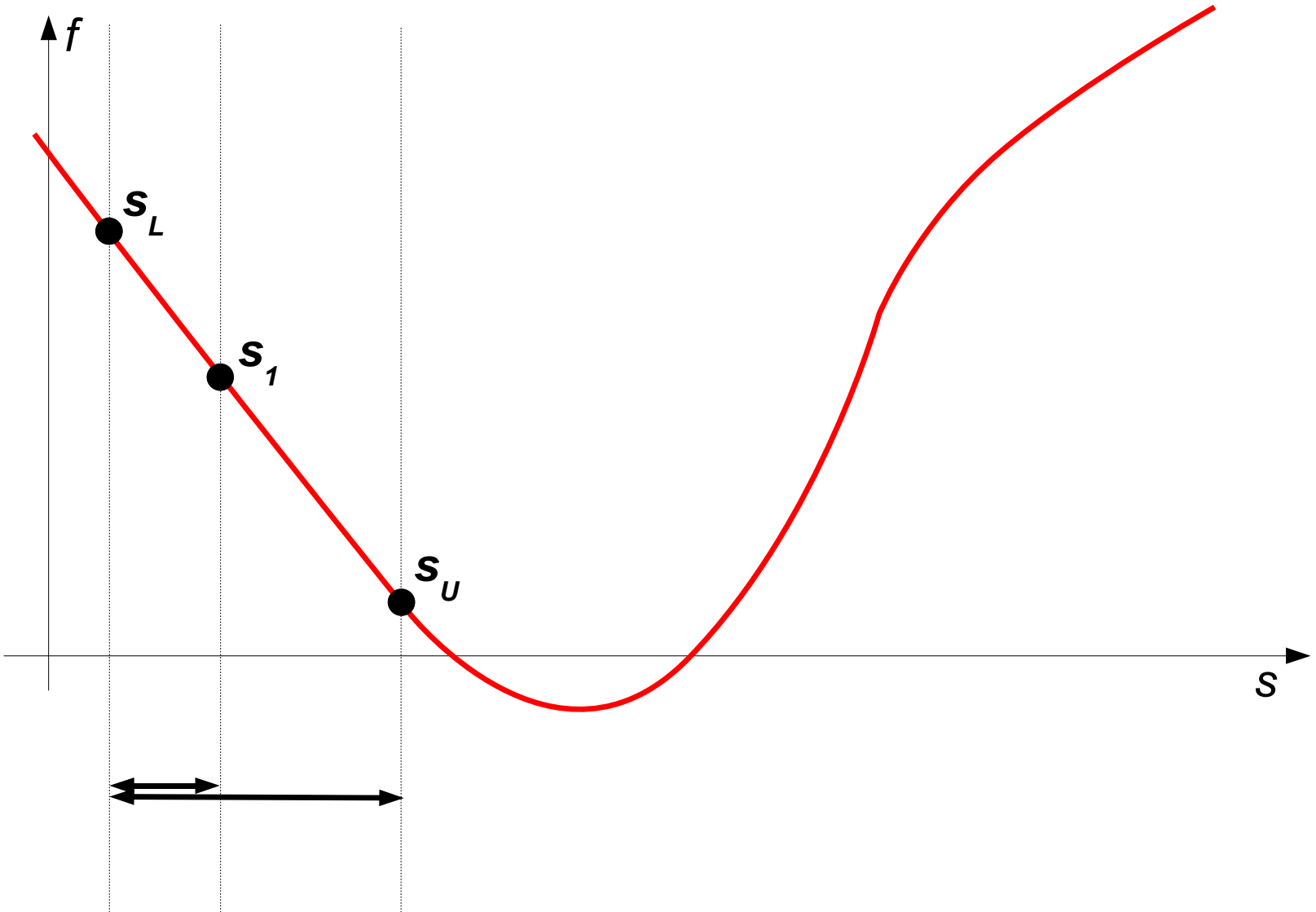
If $f(s_1) < f(s_2)$

then $s^U := s_2$, $s_2 := s_1$, $s_1 := s^L + s^U - s_2$

else $s^L := s_1$, $s_1 := s_2$, $s_2 := s^L + s^U - s_1$

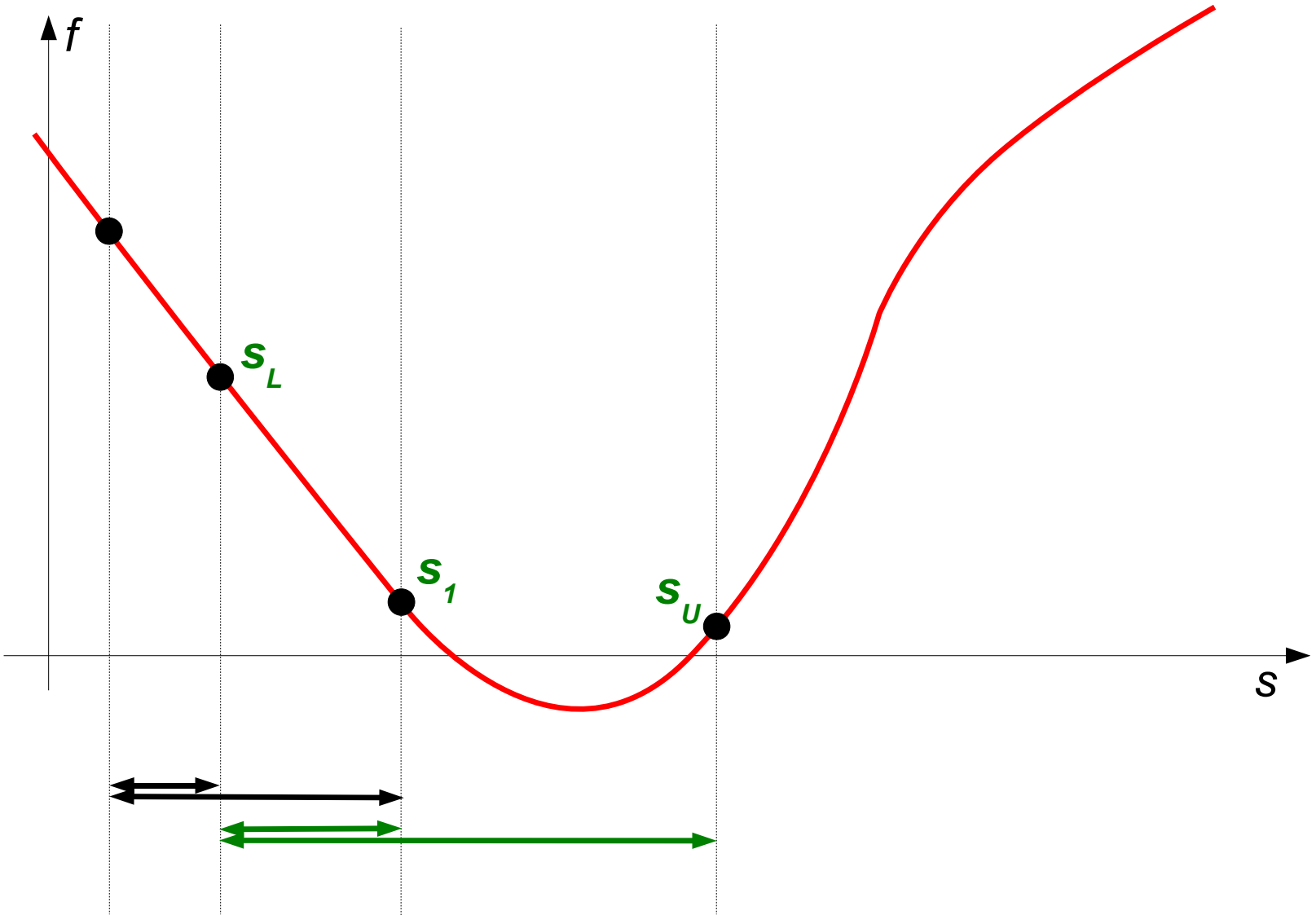
# Zero-order methods — bracketing

Bracketing can used to bracket the minimum.



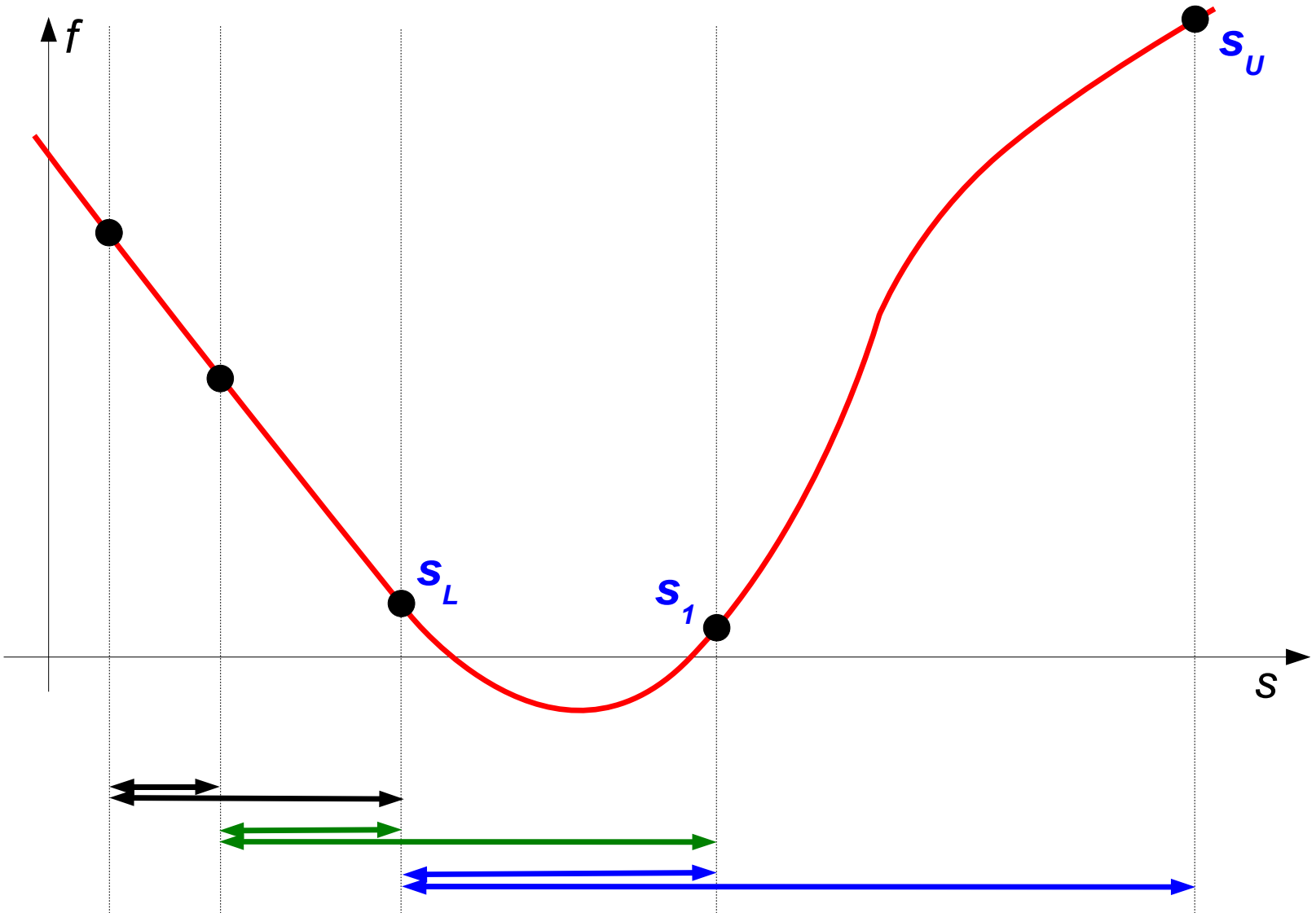
# Zero-order methods — bracketing

Bracketing can used to bracket the minimum.



# Zero-order methods — bracketing

Bracketing can used to bracket the minimum.



# Zero-order methods — bracketing

Assumptions:

- The function $f(s)$ is unimodal for $s \geq s^{\mathrm{L}}$

Initialization:

$s^{\mathrm{L}}$ and $\Delta s$

$\phi := \frac{1+\sqrt{5}}{2}$ (the golden fraction)

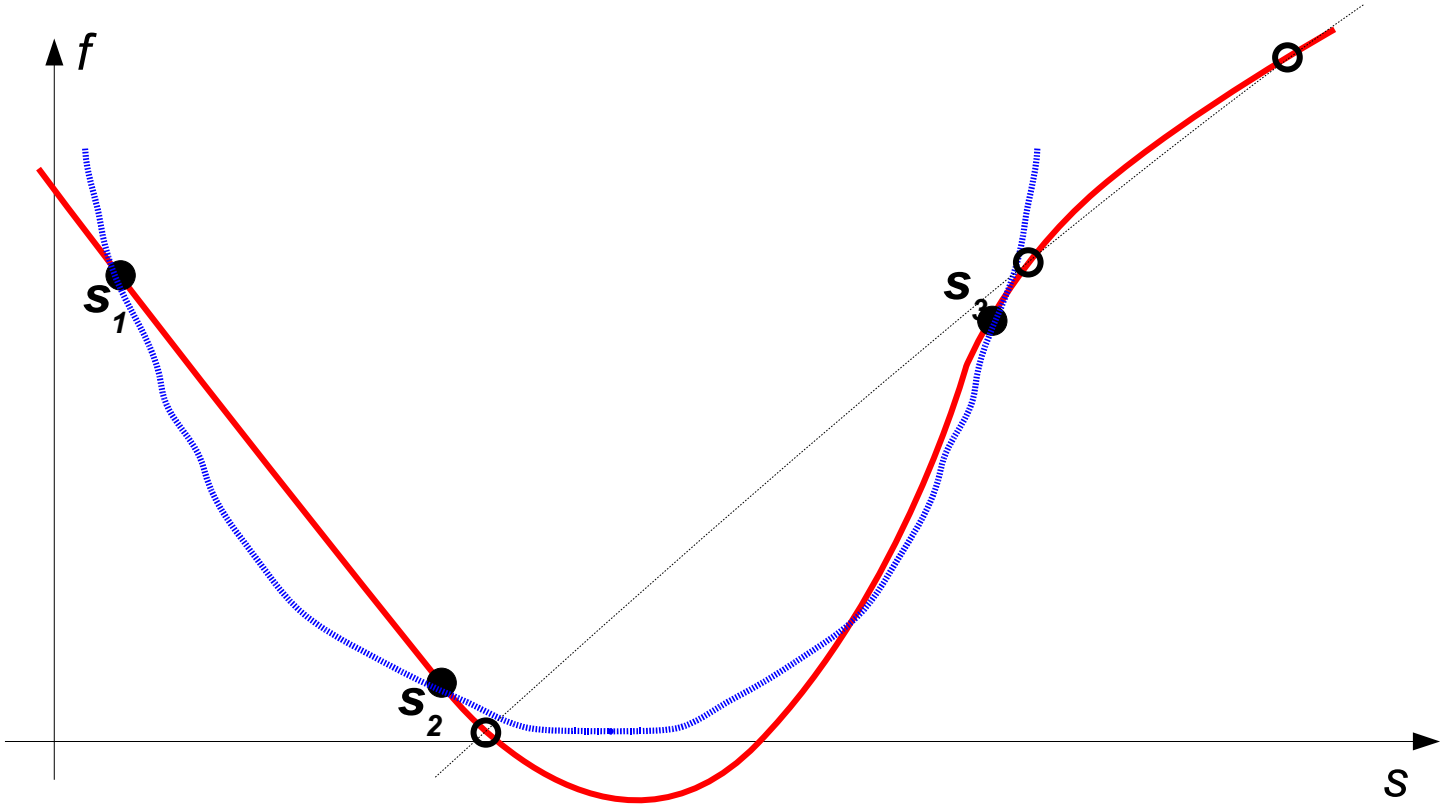
$s^{\mathrm{U}} := s^{\mathrm{L}} + \Delta s \qquad s_1 := s^{\mathrm{U}} - \frac{s^{\mathrm{U}} - s^{\mathrm{L}}}{\phi}$

While $f(s_1) > f(s^{\mathrm{U}})$ do

$s^{\mathrm{L}} := s_1$, $s_1 := s^{\mathrm{U}}$, $s^{\mathrm{U}} := s^{\mathrm{U}} + \phi(s_1 - s^{\mathrm{L}})$

As the result, the minimum is bracketed between $s^{\mathrm{L}}$ and $s^{\mathrm{U}}$

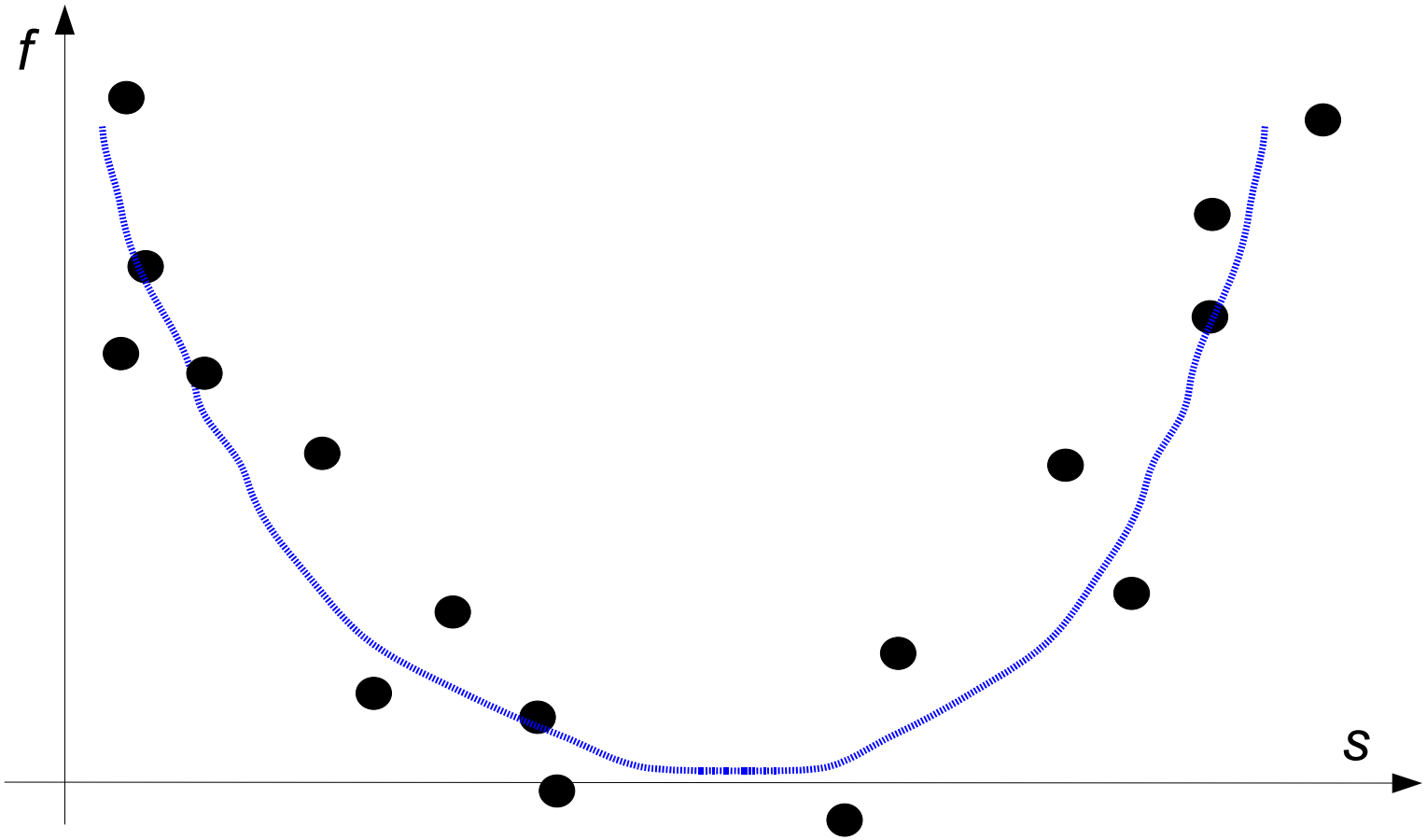
# Zero-order methods — quadratic interpolation




- No error estimation
- Concavity problem

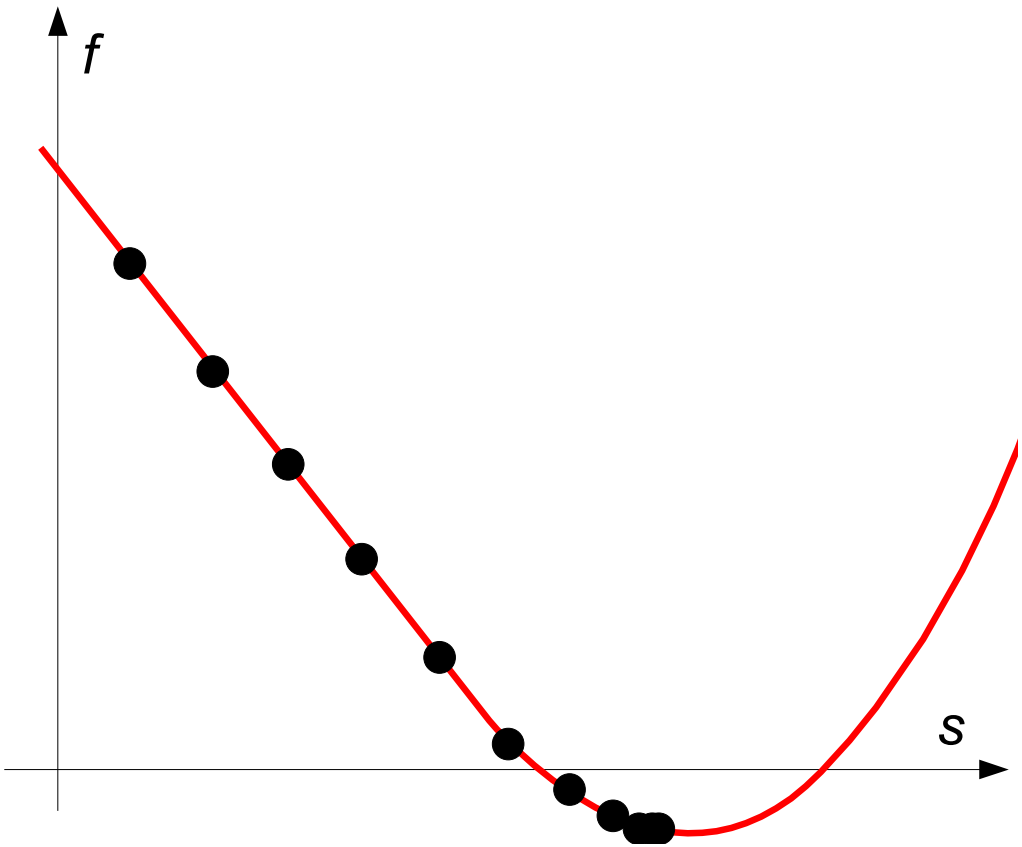
- Quick near the minimum
- Can be combined with the golden section method (Brent's method)
- Backtracking is natural

# Zero-order methods — quadratic approximation




The only method, if the error level (from measurements or numerical simulation) is significant.

# First-order methods — steepest descent




$$s := s - \alpha f'(s)$$

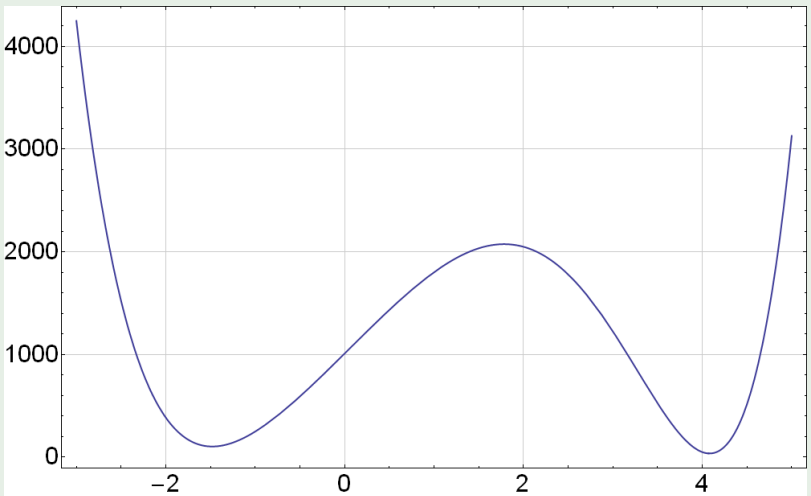
- Always goes in the right direction
- The further from the minimum, the larger steps

- No step length control (e.g. by Wolfe etc.)
- Overshooting possible
- Very slow in plateaux and shallow minima

# First-order methods — steepest descent

## Freudenstein & Roth function (cross-section)

### Freudenstein & Roth function




$$s := s - \alpha f'(s)$$

The optimization sequence is fully defined by two parameters:

1. starting point $x_0$
2. coefficient $\alpha$

Optimization can be assessed in terms of two measures

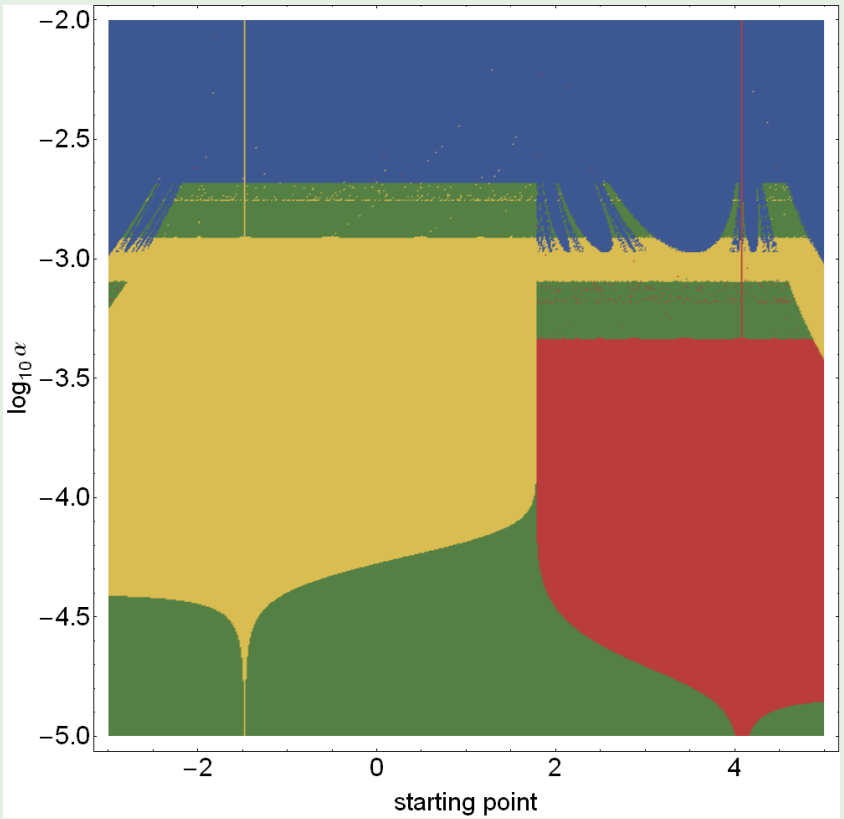
1. number of iterations to achieve the minimum
2. attraction basins of the minima

Up to 100 iterations have been computed for each $(x_0, \alpha)$.

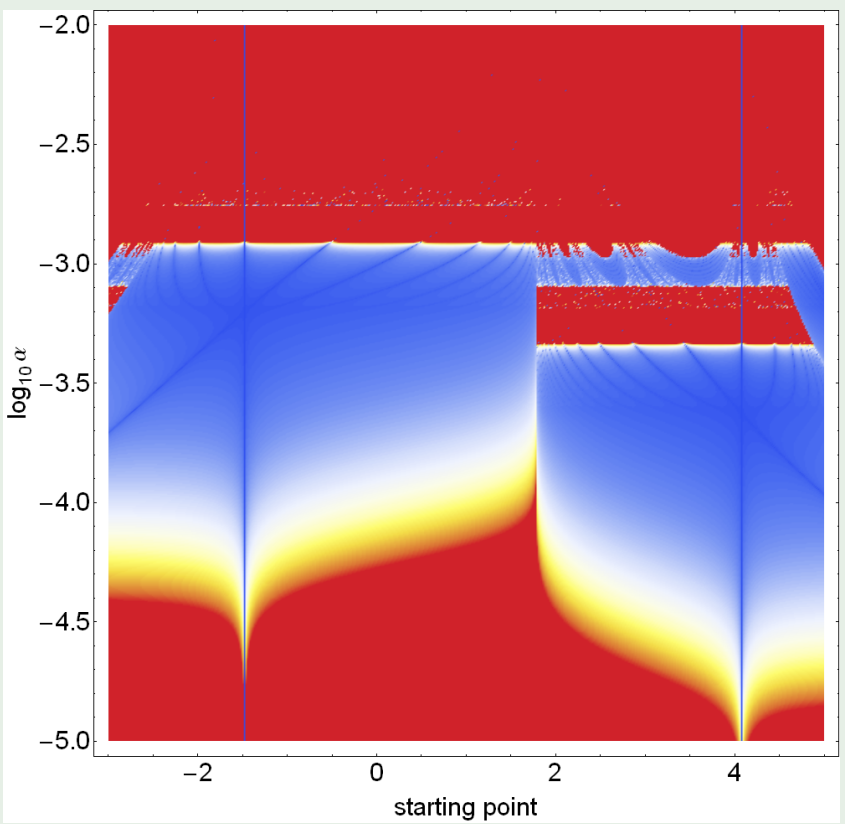
# First-order methods — steepest descent

## Freudenstein & Roth function (cross-section)

### attraction basins of minima



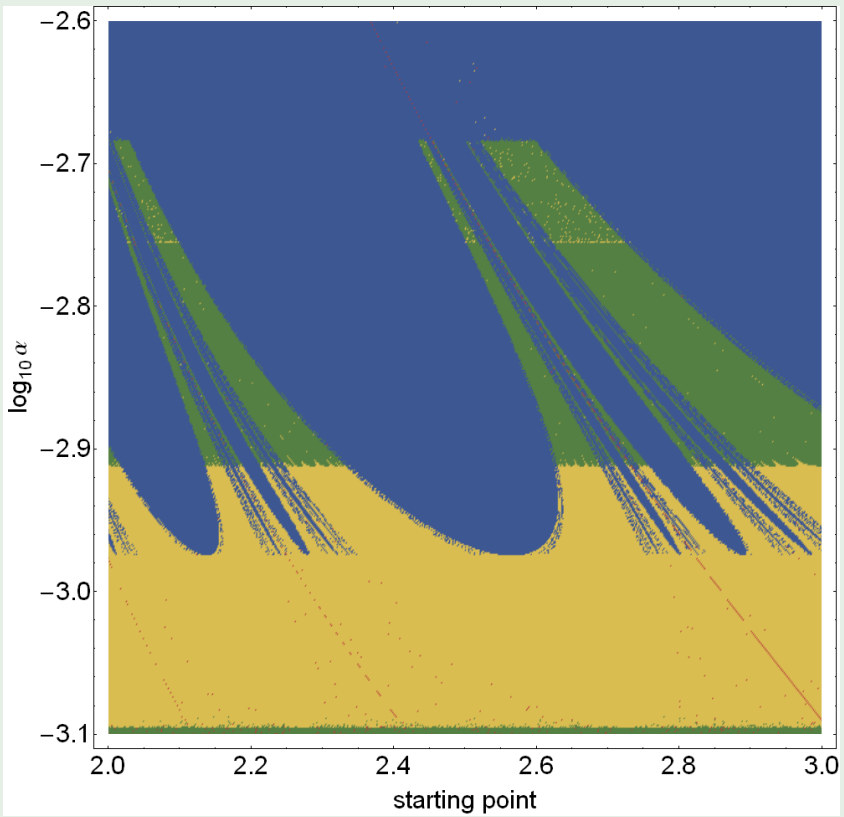

### no. of iterations (up to 100)



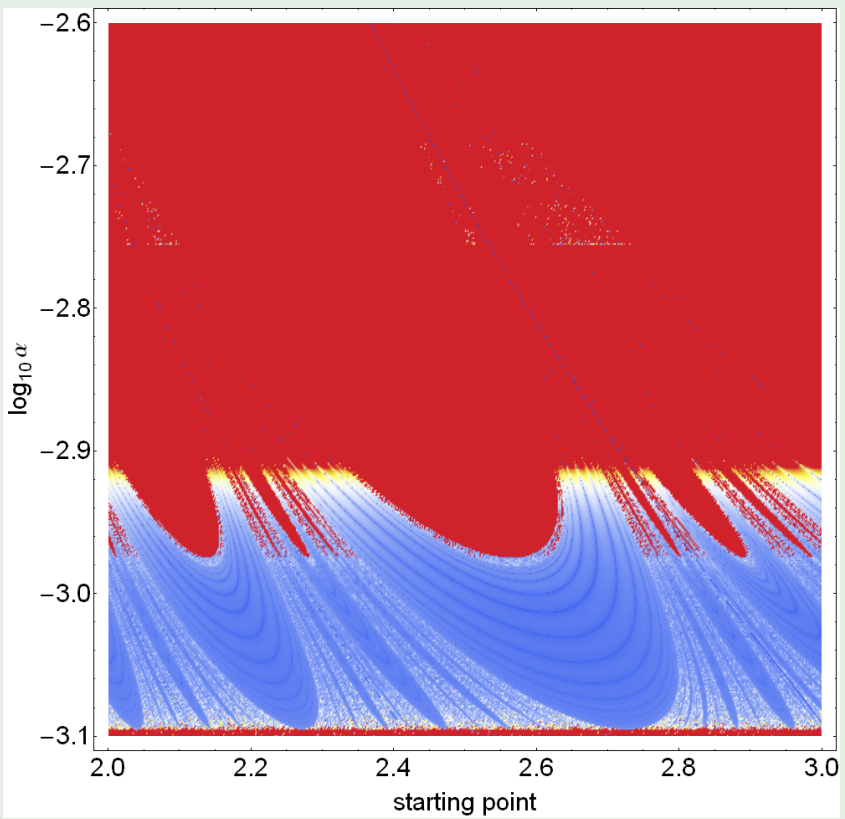
# First-order methods — steepest descent

## Freudenstein & Roth function (cross-section)

### attraction basins of minima



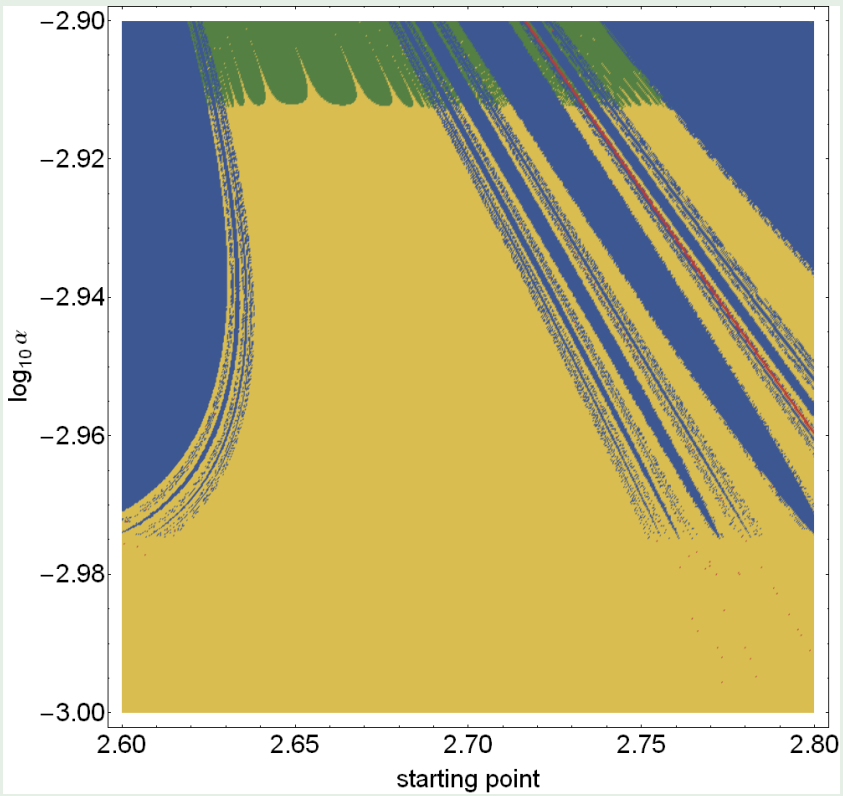

### no. of iterations (up to 100)



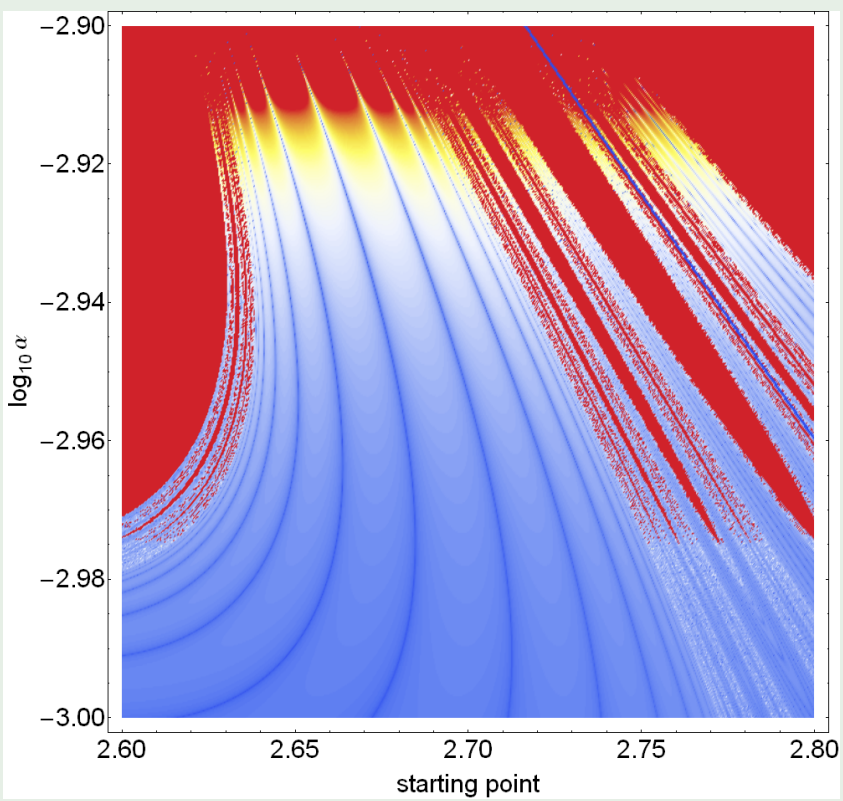
# First-order methods — steepest descent

## Freudenstein & Roth function (cross-section)

### attraction basins of minima



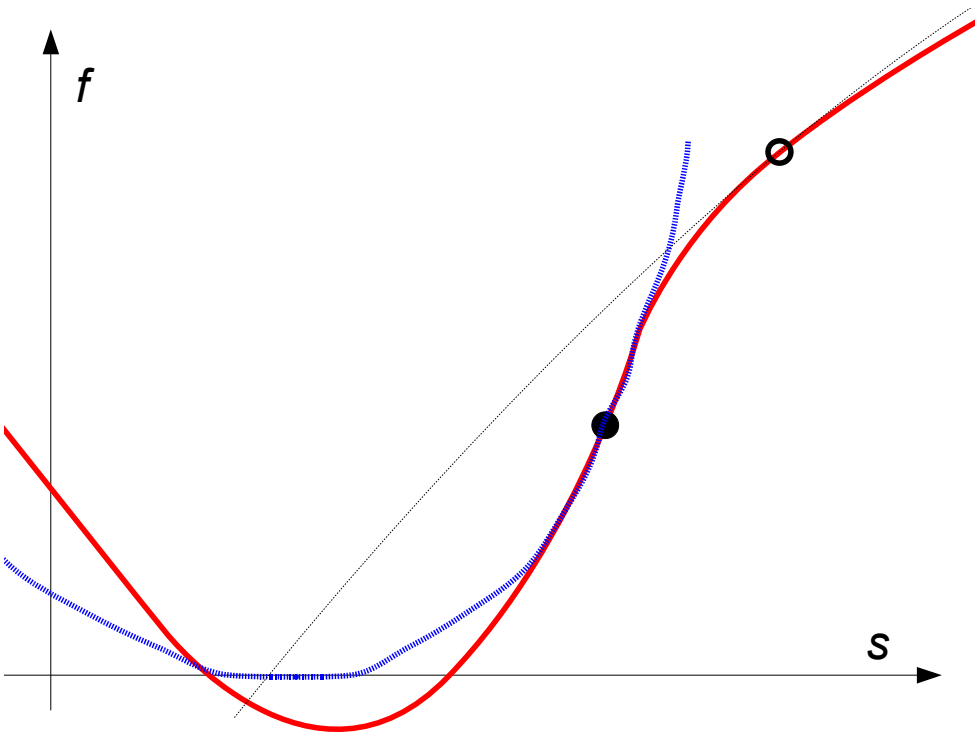

### no. of iterations (up to 100)



# Second-order methods — Newton

The Newton method uses the

- quadratic interpolation (based on $f(s)$, $f'(s)$ and $f''(s)$) or
- the Newton (tangent) method to find the root of $f'(s) = 0$




$$s := s - \alpha \frac{f'(s)}{f''(s)}$$

- Concavity problem
- Unreliable far from the minimum
- Step length problem (damped version: $\alpha < 1$)

- Very quick near the minimum
- Backtracking possible

# Second-order methods — modified Newton

The modified Newton method tries to take advantage of both

- steepest descent (effective further from the minimum) and
- Newton (effective near the minimum).

$$s := s - \alpha \frac{f'(s)}{\beta + f''(s)}$$

1. At first $\beta$ is positive (to emulate the steepest descent method),
2. then gradually decreased (to emulate the Newton method).

# First-order approximations to second-order methods

In the Newton and the modified Newton methods, the second derivative $f''$ can be approximated using the data from the last two iteration points $s_i$ and $s_{i-1}$ as

$$f''(s_i) \approx \frac{f'(s_i) - f'(s_{i-1})}{s_i - s_{i-1}}$$

or, if $f(s_{i-1})$ is preferred over $f'(s_{i-1})$,

$$f''(s_i) \approx 2\frac{f'(s_i)}{s_i - s_{i-1}} - 2\frac{f(s_i) - f(s_{i-1})}{(s_i - s_{i-1})^2}.$$

In this way, first-order approximations are obtained, which yield the quasi-Newton and the modified quasi-Newton method.

# Second-order methods — Brent's method

The Brent's method combines the

- reliable golden section method and
- quick Newton method.

> *... a scheme that relies on a sure-but-slow golden section search, when the function is not cooperative, but that switches over to parabolic interpolation when the function allows (Numerical Recipes)*