

# Numerical optimization

Mines Nancy – Fall 2024

session 4 – unconstrained optimization  
first and second-order descent methods

**Lecturer:** Julien Flamant (CNRS, CRAN)

✉ [julien.flamant@univ-lorraine.fr](mailto:julien.flamant@univ-lorraine.fr)

📍 Office 425, FST 1er cycle

**Course material:**

🌐 [arche.univ-lorraine.fr/course/view.php?id=74098](http://arche.univ-lorraine.fr/course/view.php?id=74098)

🐙 [github.com/jflamant/mines-nancy-fall24-optimization](https://github.com/jflamant/mines-nancy-fall24-optimization)

# Context

We aim to solve the unconstrained optimization problem

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

In many cases, it is not possible to find an explicit solution to the problem (as in the case of least squares).

**Solution:** iterative method(s) (algorithms)

We seek to find a minimizer of the problem iteratively:

- We fix one (or more) initial point  $\mathbf{x}^{(0)} = \mathbf{x}_0$ ;
- At each iteration  $k$ , we obtain a new point  $\mathbf{x}^{(k+1)}$  following an update, constructed from the function  $f$  to be minimized
- We repeat the process until a stopping criterion is satisfied

# Outline

- ➊ Gradient-free optimization: the Nelder-Mead algorithm
- ➋ Descent algorithms: general principles
- ➌ Gradient descent methods
- ➍ Newton's Method
- ➎ Quasi-Newton Methods
- ➏ Nonlinear Least Squares
- ➐ Comparison: convergence rate vs. complexity

# Nelder-Mead algorithm

Gradient-free optimization algorithm

→ It is enough to be able to evaluate the function  $f(\mathbf{x})$

✓ A practical heuristic method when  $f$  is not differentiable

✗ No guarantee of convergence to a local minimizer

General idea for  $f : \mathbb{R}^N \rightarrow \mathbb{R}$

- 1 Randomly select  $N + 1$  points  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{N+1}$  in  $\mathbb{R}^N$

# Nelder-Mead algorithm

Gradient-free optimization algorithm

→ It is enough to be able to evaluate the function  $f(\mathbf{x})$

✓ A practical heuristic method when  $f$  is not differentiable

✗ No guarantee of convergence to a local minimizer

General idea for  $f : \mathbb{R}^N \rightarrow \mathbb{R}$

- ➊ Randomly select  $N + 1$  points  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{N+1}$  in  $\mathbb{R}^N$
- ➋ Construct the associated simplex (a convex polytope with  $N + 1$  vertices: a triangle in 2D, a tetrahedron in 3D...)

# Nelder-Mead algorithm

Gradient-free optimization algorithm

→ It is enough to be able to evaluate the function  $f(\mathbf{x})$

✓ A practical heuristic method when  $f$  is not differentiable

✗ No guarantee of convergence to a local minimizer

General idea for  $f : \mathbb{R}^N \rightarrow \mathbb{R}$

- ➊ Randomly select  $N + 1$  points  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{N+1}$  in  $\mathbb{R}^N$
- ➋ Construct the associated simplex (a convex polytope with  $N + 1$  vertices: a triangle in 2D, a tetrahedron in 3D...)
- ➌ Evaluate  $f$  at each vertex

# Nelder-Mead algorithm

Gradient-free optimization algorithm

→ It is enough to be able to evaluate the function  $f(\mathbf{x})$

✓ A practical heuristic method when  $f$  is not differentiable

✗ No guarantee of convergence to a local minimizer

General idea for  $f : \mathbb{R}^N \rightarrow \mathbb{R}$

- ➊ Randomly select  $N + 1$  points  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{N+1}$  in  $\mathbb{R}^N$
- ➋ Construct the associated simplex (a convex polytope with  $N + 1$  vertices: a triangle in 2D, a tetrahedron in 3D...)
- ➌ Evaluate  $f$  at each vertex
- ➍ Perform a transformation of this simplex to make it "crawl" towards a (local) minimizer using operations of: reflection, expansion, contraction, shrinkage

# Nelder-Mead algorithm

Gradient-free optimization algorithm

→ It is enough to be able to evaluate the function  $f(\mathbf{x})$

✓ A practical heuristic method when  $f$  is not differentiable

✗ No guarantee of convergence to a local minimizer

General idea for  $f : \mathbb{R}^N \rightarrow \mathbb{R}$

- ➊ Randomly select  $N + 1$  points  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{N+1}$  in  $\mathbb{R}^N$
- ➋ Construct the associated simplex (a convex polytope with  $N + 1$  vertices: a triangle in 2D, a tetrahedron in 3D...)
- ➌ Evaluate  $f$  at each vertex
- ➍ Perform a transformation of this simplex to make it "crawl" towards a (local) minimizer using operations of: reflection, expansion, contraction, shrinkage
- ➎ Repeat the process until the simplex reaches a sufficiently small size



## Example

The Nelder-Mead algorithm can converge to a local minimizer ...

## Example

The Nelder-Mead algorithm can converge to a local minimizer ...

# Example

... but not always!

# Outline

- ① Gradient-free optimization: the Nelder-Mead algorithm
- ② Descent algorithms: general principles
- ③ Gradient descent methods
- ④ Newton's Method
- ⑤ Quasi-Newton Methods
- ⑥ Nonlinear Least Squares
- ⑦ Comparison: convergence rate vs. complexity

# Principle of descent algorithms

At iteration  $k$ , we construct

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}_k$$

where  $\alpha_k \geq 0$  is the step size and  $\mathbf{d}_k$  is a direction.

Dream goal:

We want  $\mathbf{x}^{(k)} \rightarrow \mathbf{x}^*$  with  $\mathbf{x}^*$  being a global minimizer of  $f$ .

→ quite unlikely, since the function  $f$  is not necessarily convex.

# Principle of descent algorithms

At iteration  $k$ , we construct

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}_k$$

where  $\alpha_k \geq 0$  is the step size and  $\mathbf{d}_k$  is a direction.

**Dream goal:**

We want  $\mathbf{x}^{(k)} \rightarrow \mathbf{x}^*$  with  $\mathbf{x}^*$  being a global minimizer of  $f$ .

→ quite unlikely, since the function  $f$  is not necessarily convex.

**Realistic goal:**

Construct  $\mathbf{x}^{(k)} \rightarrow \mathbf{x}^*$  with  $\mathbf{x}^*$  being a local minimizer of  $f$  and require

- The sequence  $\{\mathbf{x}^{(k)}\}$  describes a descent algorithm, i.e.,

$$f(\mathbf{x}^{(0)}) > f(\mathbf{x}^{(1)}) > \dots > f(\mathbf{x}^{(k)}) > f(\mathbf{x}^{(k+1)}) > \dots$$

- The algorithm stops when it no longer makes progress: at that point, a stopping criterion is defined.

## How to choose $\mathbf{d}_k$

We want the sequence  $\{\mathbf{x}^{(k)}\}$  to "decrease" the objective at each iteration,

$$f(\mathbf{x}^{(0)}) > f(\mathbf{x}^{(1)}) > \dots > f(\mathbf{x}^{(k)}) > f(\mathbf{x}^{(k+1)}) > \dots$$

We must choose  $\mathbf{d}_k$  as a descent direction.

$$\text{Descent Condition} \quad \nabla f(\mathbf{x}^{(k)})^\top \mathbf{d}_k < 0$$

# A zoo of descent algorithms

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}_k$$

Choosing the direction  $\mathbf{d}_k \rightarrow$  leads to different optimization algorithms.

General form  $\mathbf{d}_k = -\mathbf{B}_k^{-1} \nabla f(\mathbf{x}^{(k)})$  where  $\mathbf{B}_k$  is symmetric and invertible

## Examples

- Gradient Descent Method:  $\mathbf{B}_k = \mathbf{I}_N$  (identity matrix)
- Newton's Method:  $\mathbf{B}_k = \nabla^2 f(\mathbf{x}^{(k)})$  (Hessian of  $f$ )
- Quasi-Newton Methods:  $\mathbf{B}_k \approx \nabla^2 f(\mathbf{x}^{(k)})$  (approx. Hessian)



# How to define a stopping criterion ...

... or how to check that an algorithm has “converged”.

Idea: stop the algorithm when

                    criterion value at iteration  $k \leq \varepsilon$

where  $\varepsilon$  is a (small) tolerance/precision set in advance (e.g.,  $\varepsilon = 10^{-6}$ ).

# How to define a stopping criterion ...

... or how to check that an algorithm has “converged”.

Idea: stop the algorithm when

    criterion value at iteration  $k \leq \varepsilon$

where  $\varepsilon$  is a (small) tolerance/precision set in advance (e.g.,  $\varepsilon = 10^{-6}$ ).

## Some Possible Criteria

- Change in objective function:  $|f(\mathbf{x}^{(k+1)}) - f(\mathbf{x}^{(k)})| \leq \varepsilon$
- Change in solution:  $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| \leq \varepsilon$
- Gradient norm:  $\|\nabla f(\mathbf{x}^{(k)})\| \leq \varepsilon$

Preferably, we use a criterion calculated in *relative* terms, for example:

$$\frac{|f(\mathbf{x}^{(k+1)}) - f(\mathbf{x}^{(k)})|}{|f(\mathbf{x}^{(k)})|} \leq \varepsilon \quad \text{or} \quad \frac{\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|}{\|\mathbf{x}^{(k)}\|} \leq \varepsilon$$

in addition, a condition on the maximal number of iterations is often used to control the computational burden

# Outline

- ① Gradient-free optimization: the Nelder-Mead algorithm
- ② Descent algorithms: general principles
- ③ Gradient descent methods**
- ④ Newton's Method
- ⑤ Quasi-Newton Methods
- ⑥ Nonlinear Least Squares
- ⑦ Comparison: convergence rate vs. complexity

# Gradient Descent Algorithm

Starting from an initial point  $\mathbf{x}^{(0)} = \mathbf{x}_0$ , iterate:

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

For proper choice of the step size  $\alpha_k$ ,  $\mathbf{x}^{(k)}$  converges to a stationary point of  $f$  (a point  $\mathbf{x}^*$  such that  $\nabla f(\mathbf{x}^*) = 0$ ).

## Step size selection strategies

- Constant step size: set  $\alpha_k = \alpha$  for all  $k$ ;
- Optimal step size: at each iteration  $k$ , solve

$$\alpha_k = \arg \min_{\alpha \geq 0} f(\mathbf{x}^{(k)} - \alpha \nabla f(\mathbf{x}^{(k)}))$$

- Line search or *backtracking* methods.

# Constant step size strategy

## Constant step size gradient algorithm

Starting from an initial point  $\mathbf{x}^{(0)} = \mathbf{x}_0$ , iterate:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \nabla f(\mathbf{x}^{(k)})$$

Intuitively,

- $\alpha$  too small  $\rightarrow$  very slow convergence
- $\alpha$  too large  $\rightarrow$  **RISK OF DIVERGENCE**

The step size  $\alpha$  must be chosen to balance speed of convergence and convergence guarantees.

# Constant step size strategy

## Constant step size gradient algorithm

Starting from an initial point  $\mathbf{x}^{(0)} = \mathbf{x}_0$ , iterate:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \nabla f(\mathbf{x}^{(k)})$$

**Important Result:** if  $f$  has an  $L$ -Lipschitz gradient, i.e.,

$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^N, \quad \|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|$$

Then, if

$$0 < \alpha < \frac{2}{L}$$

the algorithm converges to a stationary point of  $f$ .

more details on convergence results in future class

# Optimal step size strategy

## Optimal step size gradient algorithm

Starting from an initial point  $\mathbf{x}^{(0)} = \mathbf{x}_0$ , iterate:

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

where  $\alpha_k$  is obtained by minimizing:

$$\alpha_k = \arg \min_{\alpha \geq 0} f(\mathbf{x}^{(k)} - \alpha \nabla f(\mathbf{x}^{(k)}))$$

at each iteration.

On paper, the strategy is ideal.

**Main issue:** calculating the optimal step size can be as difficult as the original problem and/or very computationally expensive.

## A 2D example

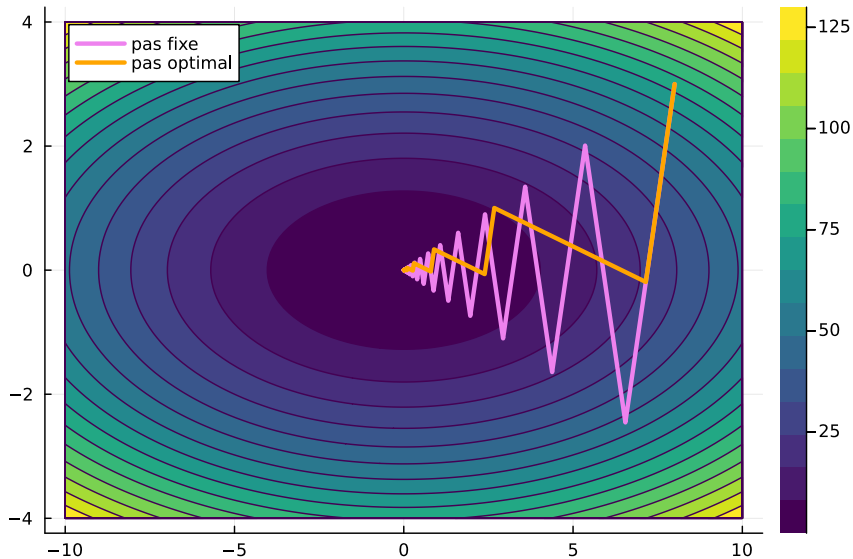
$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x}, \text{ with } \mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & \eta \end{bmatrix}, \eta > 0$$

- 1 Calculate  $\nabla f(\mathbf{x})$
- 2 Give the explicit expression for  $\mathbf{x}^{(k+1)}$  in terms of  $\mathbf{x}^{(k)}$
- 3 Give the expression for  $\mathbf{x}^{(1)}$  in terms of  $\alpha_0$  for  $\mathbf{x}^{(0)} = [\eta, 1]^T$
- 4 Find the optimal step size for this iteration
- 5 Show that the iterations with optimal step size take the form:

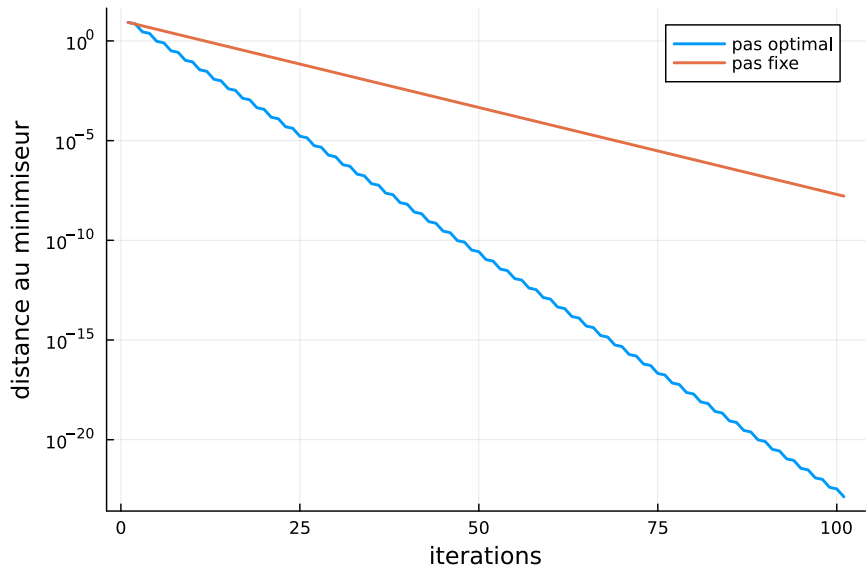
$$x_1^{(k)} = \eta \left( \frac{\eta - 1}{\eta + 1} \right)^k \quad x_2^{(k)} = \left( -\frac{\eta - 1}{\eta + 1} \right)^k$$



## Example for $\eta = 10$



## Example for $\eta = 10$



## Line search or backtracking strategies

We seek a step size  $\alpha_k$  that sufficiently decreases the objective  $f$ :

$$\alpha_k \approx \arg \min_{\alpha \geq 0} f(\mathbf{x}^{(k)} - \alpha \nabla f(\mathbf{x}^{(k)}))$$

→ This is the most commonly used approach in practice, for which many “recipes” exist.

### Example: backtracking for gradient algorithm

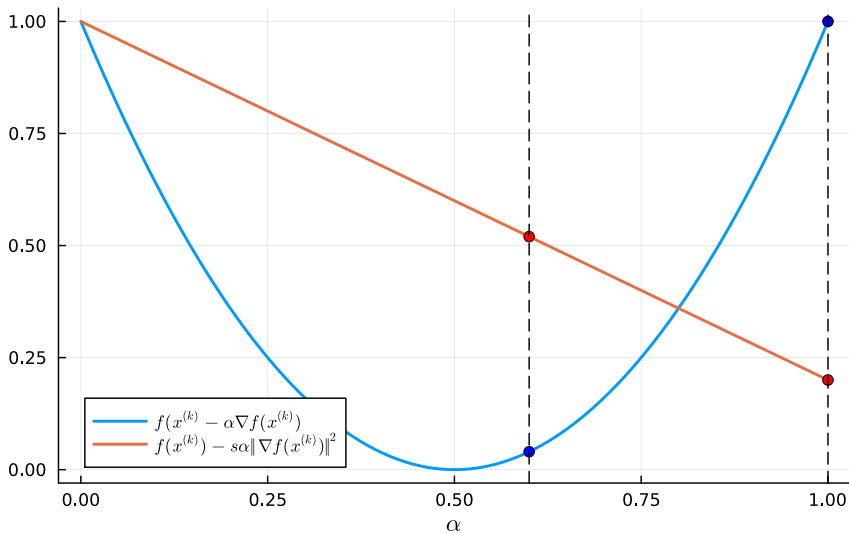
Fix  $s \in (0, 0.5)$  and  $\eta \in (0, 1)$  with an initial step size  $\alpha = 1$ .

```
1:  $\alpha := 1$   
2: while  $f(\mathbf{x}^{(k)} - \alpha \nabla f(\mathbf{x}^{(k)})) > f(\mathbf{x}^{(k)}) - s\alpha \|\nabla f(\mathbf{x}^{(k)})\|^2$  do  
3:    $\alpha := \eta\alpha$   
4: end while
```

Approximate step size strategies are constructed to ensure algorithm convergence.

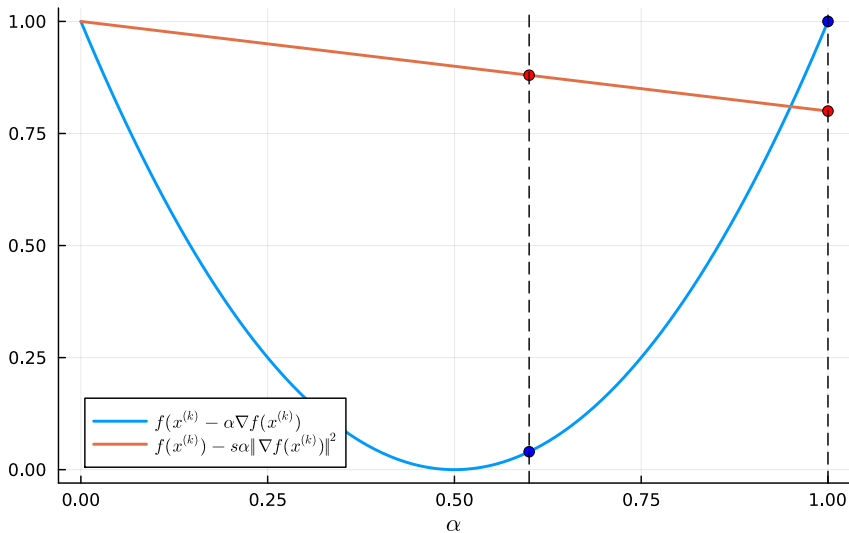
# Example: backtracking

backtracking line search,  $s = 0.2$ ,  $\eta = 0.6$



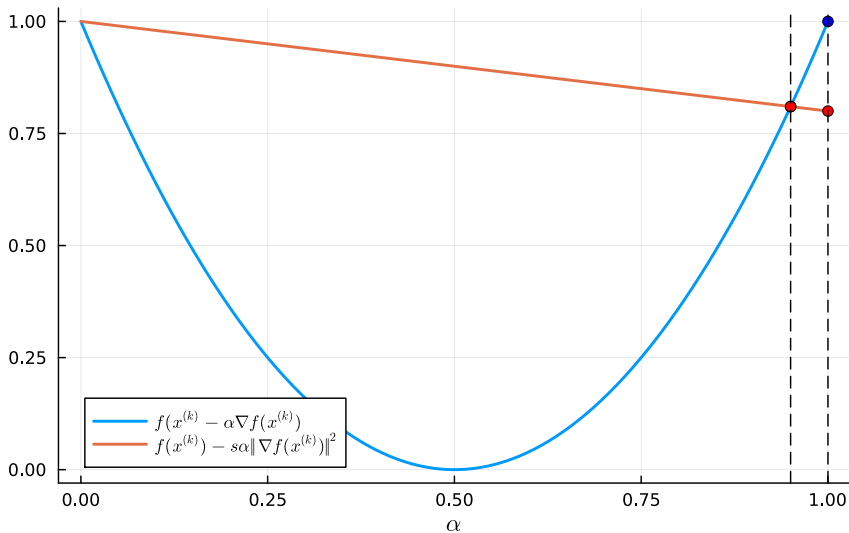
# Example: backtracking

backtracking line search,  $s = 0.05$ ,  $\eta = 0.6$



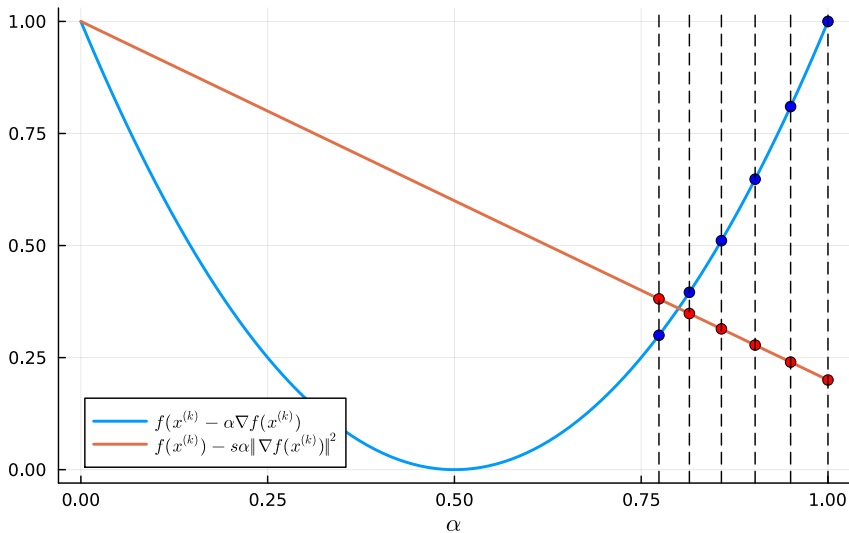
# Example: backtracking

backtracking line search,  $s = 0.05$ ,  $\eta = 0.95$

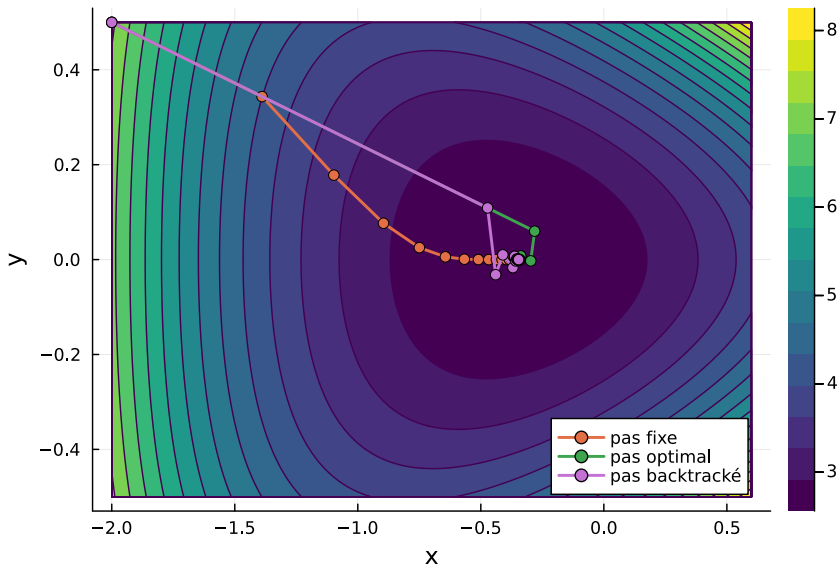


# Example: backtracking

backtracking line search,  $s = 0.2$ ,  $\eta = 0.95$

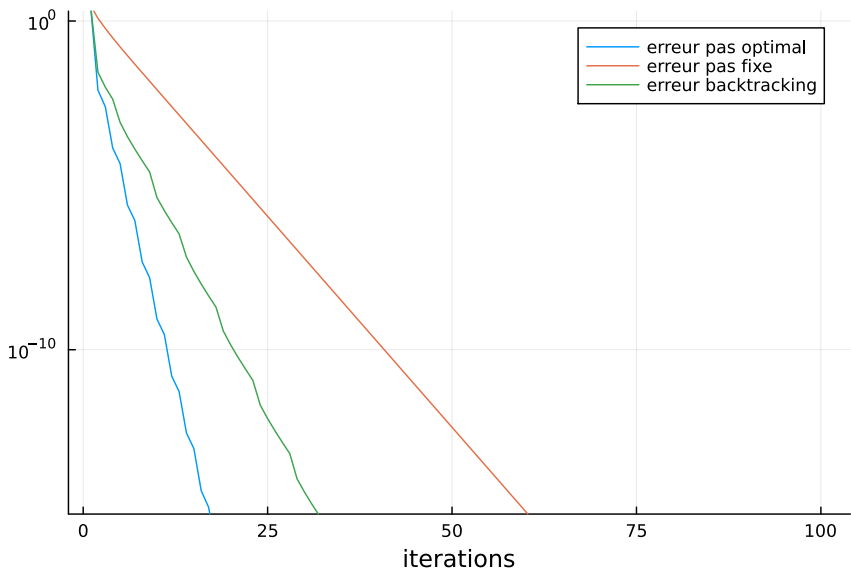


# Comparison of step size strategies





# Comparison of step size strategies



# Outline

- ① Gradient-free optimization: the Nelder-Mead algorithm
- ② Descent algorithms: general principles
- ③ Gradient descent methods
- ④ Newton's Method**
- ⑤ Quasi-Newton Methods
- ⑥ Nonlinear Least Squares
- ⑦ Comparison: convergence rate vs. complexity

# Newton's Algorithm

Starting from an initial point  $\mathbf{x}^{(0)} = \mathbf{x}_0$ , iterate

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \left[ \nabla^2 f(\mathbf{x}^{(k)}) \right]^{-1} \nabla f(\mathbf{x}^{(k)})$$

The quantity  $\mathbf{d}_k = - \left[ \nabla^2 f(\mathbf{x}^{(k)}) \right]^{-1} \nabla f(\mathbf{x}^{(k)})$  is indeed a descent direction as long as the Hessian of  $f$  at  $\mathbf{x}^{(k)}$  is positive definite:

$$\nabla f(\mathbf{x}^{(k)})^\top \mathbf{d}_k = -\nabla f(\mathbf{x}^{(k)})^\top \left[ \nabla^2 f(\mathbf{x}^{(k)}) \right]^{-1} \nabla f(\mathbf{x}^{(k)}) < 0$$

## Step-size selection strategies

- constant step-size: we fix  $\alpha_k = \alpha$  for all  $k$  (often  $\alpha = 1$ );
- possible use of *backtracking*.

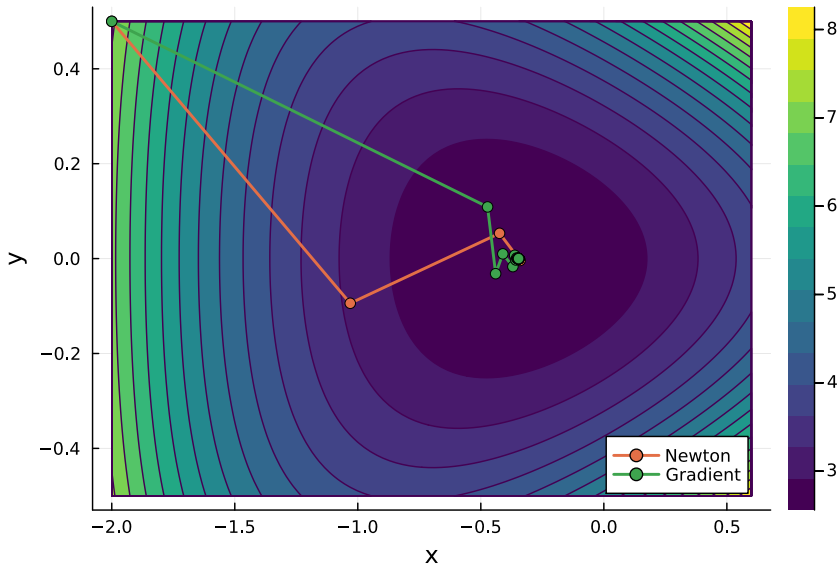
# The return of the 2D unconstrained QP example

We consider the function

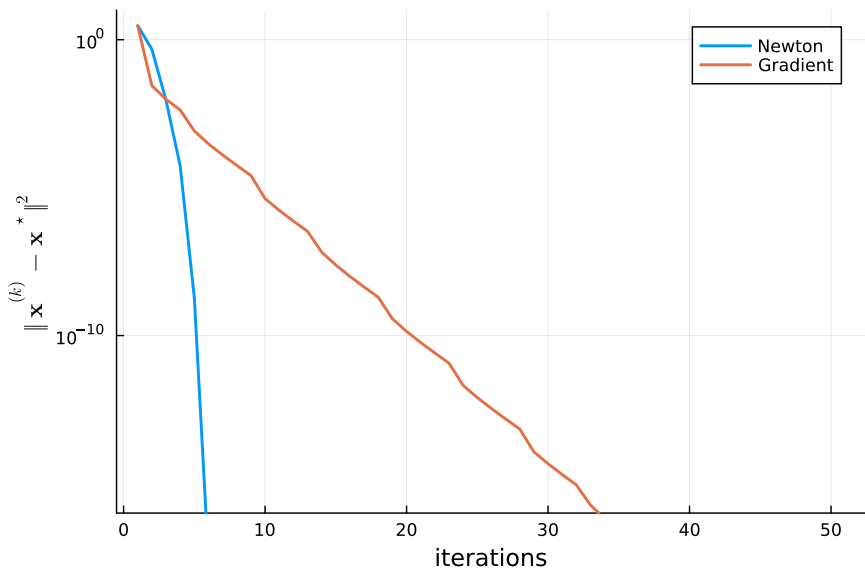
$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^\top \mathbf{Q} \mathbf{x} - \mathbf{b}^\top \mathbf{x}, \text{ with } \mathbf{Q} > 0$$

- ① Compute  $\nabla f(\mathbf{x})$  and deduce the minimizer of  $f$
- ② Compute  $\nabla^2 f(\mathbf{x})$
- ③ Give the explicit expression of  $\mathbf{x}^{(k+1)}$  as a function of  $\mathbf{x}^{(k)}$  for Newton's method with constant step-size
- ④ What can we deduce from this?

## Example: Non-Quadratic Case



## Example: Non-Quadratic Case



# Outline

- ① Gradient-free optimization: the Nelder-Mead algorithm
- ② Descent algorithms: general principles
- ③ Gradient descent methods
- ④ Newton's Method
- ⑤ Quasi-Newton Methods**
- ⑥ Nonlinear Least Squares
- ⑦ Comparison: convergence rate vs. complexity

# Motivation

Despite its numerous advantages, Newton's method suffers from two disadvantages:

- Computing Newton's step involves calculating and inverting the Hessian matrix  $\nabla^2 f(\mathbf{x}^{(k)})$ , which becomes very costly in high dimensions;
- The Hessian matrix  $\nabla^2 f(\mathbf{x}^{(k)})$  is not always positive definite, so it's not always possible to calculate its inverse.

## Solution: Quasi-Newton Methods

We replace the Hessian matrix with an approximation  $\mathbf{B}_k \approx \nabla^2 f(\mathbf{x}^{(k)})$ , so that the iterations become:

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



# Principle of Quasi-Newton Methods

We aim to recursively estimate the matrices  $\mathbf{B}_k$  (or  $\mathbf{H}_k$ ) such that

$$\mathbf{B}_k \approx \nabla^2 f(\mathbf{x}^{(k)}) \quad \text{or} \quad \mathbf{H}_k \approx \left[ \nabla^2 f(\mathbf{x}^{(k)}) \right]^{-1}$$

An update rule  $\rightarrow$  a quasi-Newton method

## Some examples

- Broyden-Fletcher-Goldfarb-Shanno (BFGS): the most popular
- L-BFGS: memory-efficient BFGS
- Symmetric rank-one (SR1)
- Davidon-Fletcher-Powell (DFP)
- ...

# Broyden-Fletcher-Goldfarb-Shanno (BFGS)

At iteration  $k$ ,

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

We define

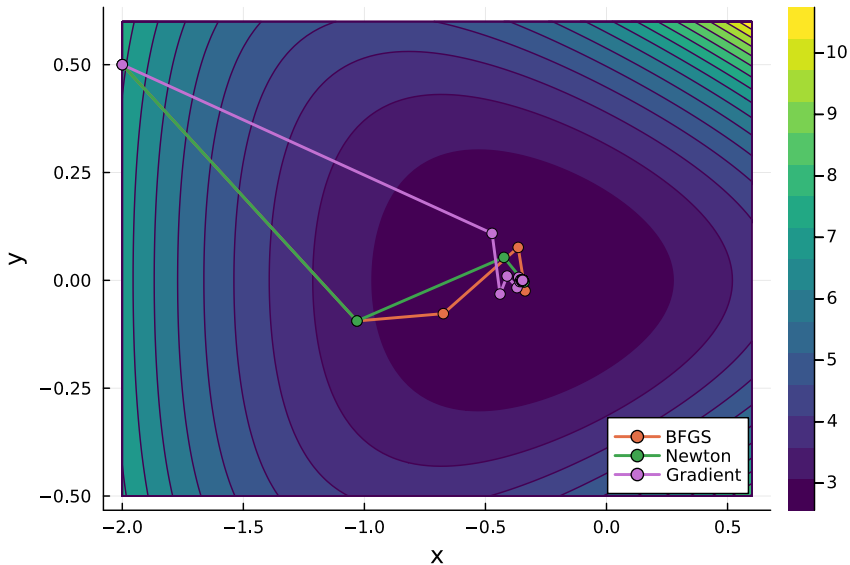
$$\mathbf{s}_k = -\alpha_k \mathbf{H}_k \nabla f(\mathbf{x}^{(k)}), \quad \mathbf{y}_k = \nabla f(\mathbf{x}^{(k+1)}) - \nabla f(\mathbf{x}^{(k)}), \quad \rho_k = \frac{1}{\mathbf{s}_k^\top \mathbf{y}_k}$$

Update of  $\mathbf{H}_k$  in BFGS

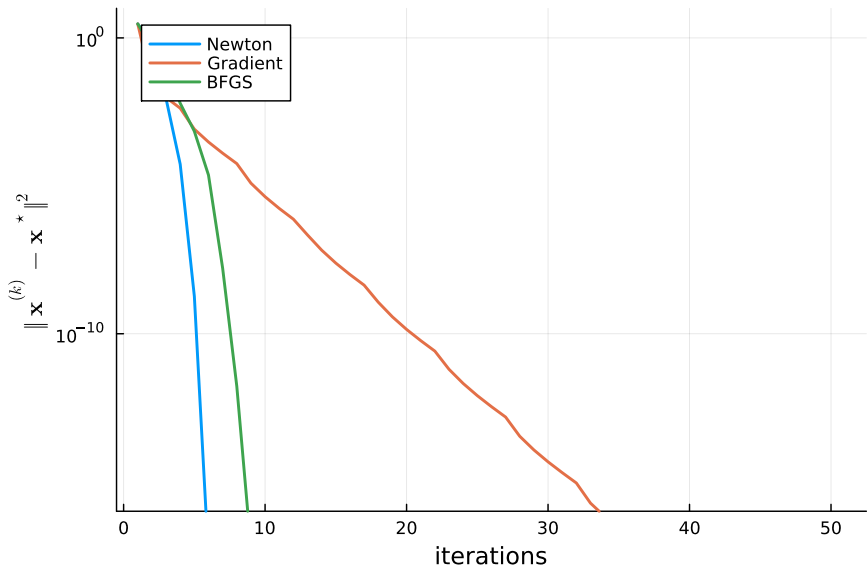
$$\mathbf{H}_{k+1} = (\mathbf{I}_N - \rho_k \mathbf{s}_k \mathbf{y}_k^\top) \mathbf{H}_k (\mathbf{I}_N - \rho_k \mathbf{y}_k \mathbf{s}_k^\top) + \rho_k \mathbf{s}_k \mathbf{s}_k^\top$$

Step-size  $\alpha_k$  is chosen using a line search technique (e.g., backtracking).

Example: Non-Quadratic Case ( $\mathbf{H}_0 = [\nabla^2 f(\mathbf{x}^{(0)})]^{-1}$ )



Example: Non-Quadratic Case ( $\mathbf{H}_0 = [\nabla^2 f(\mathbf{x}^{(0)})]^{-1}$ )



# Outline

- ① Gradient-free optimization: the Nelder-Mead algorithm
- ② Descent algorithms: general principles
- ③ Gradient descent methods
- ④ Newton's Method
- ⑤ Quasi-Newton Methods
- ⑥ Nonlinear Least Squares**
- ⑦ Comparison: convergence rate vs. complexity

# Nonlinear Least Squares

## General Form

$$\min_{\mathbf{x} \in \mathbb{R}^N} f(\mathbf{x}) := \frac{1}{2} \sum_{j=1}^M r_j(\mathbf{x})^2, \quad r_j : \mathbb{R}^N \rightarrow \mathbb{R}$$

Let us define

$$R(\mathbf{x}) = (r_1(\mathbf{x}), r_2(\mathbf{x}), \dots, r_M(\mathbf{x}))^\top$$

We can rewrite the problem as

$$\min_{\mathbf{x} \in \mathbb{R}^N} f(\mathbf{x}) := \frac{1}{2} \|R(\mathbf{x})\|_2^2, \quad R(\mathbf{x}) : \mathbb{R}^N \rightarrow \mathbb{R}^M$$

### Remark

if  $R(\mathbf{x}) = \mathbf{Ax} - \mathbf{b}$ , this reduces to a linear least squares problem.

# Nonlinear Least Squares: Jacobian Matrix

The particular form of  $f(\mathbf{x})$  can be exploited to construct efficient algorithms.

Key Tool: Jacobian Matrix of  $\mathbf{R} : \mathbb{R}^N \rightarrow \mathbb{R}^M$

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_1}{\partial x_N} \\ \frac{\partial r_2}{\partial x_1} & \cdots & \frac{\partial r_2}{\partial x_N} \\ \vdots & & \vdots \\ \frac{\partial r_M}{\partial x_1} & \cdots & \frac{\partial r_M}{\partial x_N} \end{bmatrix} = \begin{bmatrix} \nabla r_1(\mathbf{x})^\top \\ \nabla r_2(\mathbf{x})^\top \\ \vdots \\ \nabla r_M(\mathbf{x})^\top \end{bmatrix} \in \mathbb{R}^{M \times N}$$

# Gradient and Hessian Expressions

Using the Jacobian matrix allows for a simple expression of the gradient and Hessian of  $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{R}(\mathbf{x})\|_2^2$ .

## Gradient of $f$

$$\nabla f(\mathbf{x}) = \sum_{j=1}^M r_j(\mathbf{x}) \nabla r_j(\mathbf{x}) = \mathbf{J}(\mathbf{x})^\top \mathbf{R}(\mathbf{x})$$

## Hessian of $f$

$$\begin{aligned} \nabla^2 f(\mathbf{x}) &= \sum_{j=1}^M \nabla r_j(\mathbf{x}) \nabla r_j(\mathbf{x})^\top + \sum_{j=1}^M r_j(\mathbf{x}) \nabla^2 r_j(\mathbf{x}) \\ &= \mathbf{J}(\mathbf{x})^\top \mathbf{J}(\mathbf{x}) + \sum_{j=1}^M r_j(\mathbf{x}) \nabla^2 r_j(\mathbf{x}) \end{aligned}$$



# Why is this interesting?

In many applications:

- Computing the Jacobian  $\mathbf{J}(\mathbf{x})$  is inexpensive (or easy to obtain)
- The gradient is directly obtained from  $\nabla f(\mathbf{x}) = \mathbf{J}(\mathbf{x})^\top \mathbf{R}(\mathbf{x})$
- Often, the second term in the Hessian can be neglected (especially near the solution), so that

$$\nabla^2 f(\mathbf{x}) \approx \mathbf{J}(\mathbf{x})^\top \mathbf{J}(\mathbf{x})$$

meaning the Hessian can be computed without second derivatives.

→ We exploit these properties to construct algorithms:

- Gauss-Newton method
- Levenberg-Marquardt method

# Algorithms for nonlinear least squares

## Gauss-Newton Method

This is a quasi-Newton method with  $\mathbf{B}_k = \mathbf{J}(\mathbf{x}^{(k)})^\top \mathbf{J}(\mathbf{x}^{(k)})$

At iteration  $k$ , ( $\alpha_k = 1$ ):

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left[ \mathbf{J}(\mathbf{x}^{(k)})^\top \mathbf{J}(\mathbf{x}^{(k)}) \right]^{-1} \nabla f(\mathbf{x}^{(k)})$$

## Levenberg-Marquardt Method

Can also be interpreted as a quasi-Newton method

At iteration  $k$ , ( $\alpha_k = 1$ ), we set  $\mathbf{C}_k = \mathbf{J}(\mathbf{x}^{(k)})^\top \mathbf{J}(\mathbf{x}^{(k)})$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - [\mathbf{C}_k + \lambda_k \text{diag}(\mathbf{C}_k)]^{-1} \nabla f(\mathbf{x}^{(k)})$$

where  $\lambda_k$  is a regularization parameter.

→ Algorithm commonly used in practice for nonlinear least squares problems.

# Outline

- ① Gradient-free optimization: the Nelder-Mead algorithm
- ② Descent algorithms: general principles
- ③ Gradient descent methods
- ④ Newton's Method
- ⑤ Quasi-Newton Methods
- ⑥ Nonlinear Least Squares
- ⑦ Comparison: convergence rate vs. complexity

# How to choose an algorithm for a given problem?

For a given problem, choosing an algorithm can depend on:

- **Convergence rate:** It indicates how fast the iterates approach their limit, measured by the distance

$$\varepsilon_k = \|\mathbf{x}^{(k)} - \mathbf{x}^*\|_2 \text{ where } \mathbf{x}^* \text{ is a local minimizer}$$

The rate of convergence is *asymptotic*, meaning it appears as  $k \rightarrow \infty$ .

- **Numerical complexity:** It measures the computational cost associated with an algorithm (usually, the cost per iteration).

# Different types of convergence

Distance to a local minimizer

$$\varepsilon_k = \|\mathbf{x}^{(k)} - \mathbf{x}^*\|_2 \text{ where } \mathbf{x}^* \text{ is a local minimizer}$$

- Linear convergence:

$$\frac{\varepsilon_{k+1}}{\varepsilon_k} \xrightarrow{k \rightarrow \infty} \mu, \quad 0 < \mu < 1$$

- Super-linear convergence:

$$\frac{\varepsilon_{k+1}}{\varepsilon_k} \xrightarrow{k \rightarrow \infty} 0$$

- Quadratic convergence:

$$\frac{\varepsilon_{k+1}}{\varepsilon_k^2} \xrightarrow{k \rightarrow \infty} \mu, \quad 0 < \mu < 1$$

# Computational complexity

The complexity is measured in flops = number of elementary operations (addition, multiplication)

## Complexity of basic operations

- Dot product  $\mathbf{a}^\top \mathbf{b}$  with  $\mathbf{a}, \mathbf{b} \in \mathbb{R}^N$   $\mathcal{O}(N)$  flops
- Matrix-vector product  $\mathbf{A}\mathbf{b}$  with  $\mathbf{A} \in \mathbb{R}^{M \times N}, \mathbf{b} \in \mathbb{R}^N$   $\mathcal{O}(MN)$  flops
- Matrix-matrix product  $\mathbf{A}\mathbf{B}$  with  $\mathbf{A} \in \mathbb{R}^{M \times N}, \mathbf{B} \in \mathbb{R}^{N \times P}$   
 $\mathcal{O}(MNP)$  flops
- Matrix inversion  $\mathbf{A}^{-1}$  with  $\mathbf{A} \in \mathbb{R}^{N \times N}$   $\mathcal{O}(N^3)$  flops

The overall complexity of a sequence of operations can sometimes be difficult to calculate or may not be sufficient to characterize the computational cost (issues related to memory footprint / swapping, in particular)

# Properties of some algorithms

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

- Gradient method with constant step size

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \nabla f(\mathbf{x}^{(k)})$$

- Complexity:  $\mathcal{O}(N)$  per iteration
- Linear convergence rate [under some conditions]

- Newton's method

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \left[ \nabla^2 f(\mathbf{x}^{(k)}) \right]^{-1} \nabla f(\mathbf{x}^{(k)})$$

- Complexity:  $\mathcal{O}(N^3)$  per iteration
- Quadratic convergence rate [under some conditions]

more on convergence in the dedicated lecture

# After the holidays

## Class of 05/11/24

- ~ 30 min course exam
- ~ 2h30 TP on first and second-order descent methods

## Course exam details

- simple questions on the content of sessions 1-4
- short answers and justifications required
- no documents
- doing examples from the lectures slides can be useful!