Computational practicum
Lecture 4
Numerical Optimization - Introduction

Zoë J.G. Gromotka, Ferdinand Grozema, Artur M. Schweidtmann, Tanuj Karia

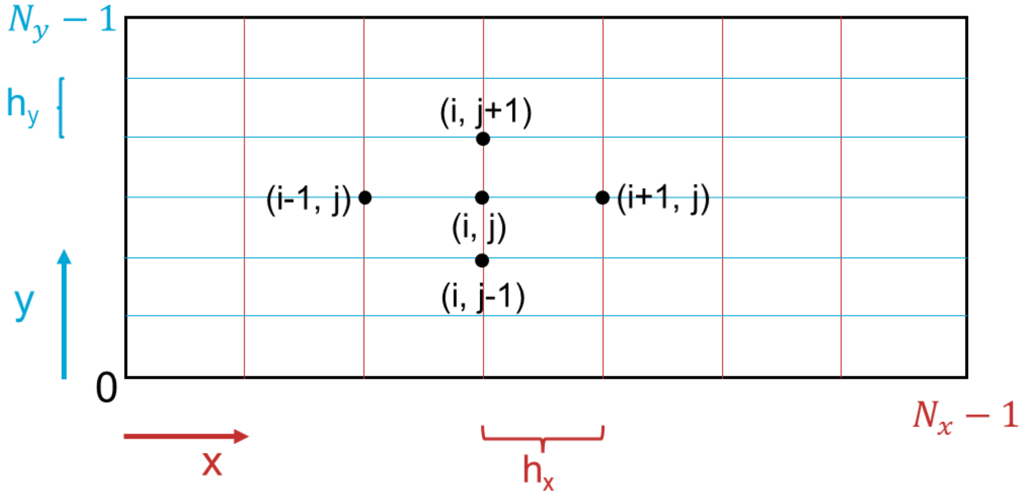
Computational Practicum
Dept. Chemical Engineering
Delft University of Technology

Wednesday, 3 December 2025

TU Delft
Process Intelligence RESEARCH
Delft Institute of Applied Mathematics

Recap

- Elliptic PDEs (a.k.a Laplace equation) – Steady state heat equation in 2D
  - Boundary conditions at the corners
  - 2D spatial finite-differences
  - Poisson matrix
- Implementation of sparse matrices using Scipy



Learning goals of this lecture

After successfully completing this lecture, you are able to. . . .

- explain what constitutes an unconstraint mathematical optimization problem
- apply the optimality conditions to unconstraint optimization problems
- explain numerical methods to solve unconstrained optimization problems
- implement numerical methods to solve unconstrained optimization problems

Agenda

- **Basics of mathematical optimization**
- **Conditions for optimality**
  - Necessary condition for optimality
  - Convexity of a function
  - Sufficient condition for optimality
- **Unconstrained Optimization**
  - Gradient descent algorithm pseudo-code
  - First-order search methods
  - Second-order search methods
  - Unconstrained optimization using SciPy

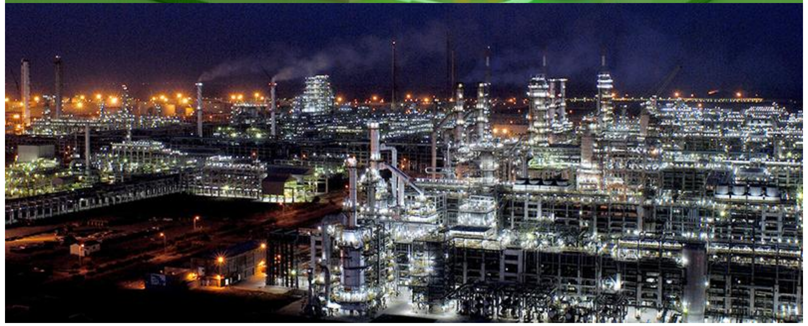
Agenda

- **Basics of mathematical optimization**
- **Conditions for optimality**
  - Necessary condition for optimality
  - Convexity of a function
  - Sufficient condition for optimality
- **Unconstrained Optimization**
  - Gradient descent algorithm pseudo-code
  - First-order search methods
  - Second-order search methods
  - Unconstrained optimization using SciPy

Decision-making in engineering

- Decision-making is inherent in engineering
  - Schedule train/airline routes (*Operations Research)*
  - Schedule charging/discharging of batteries (Energy systems engineering)
  - Design of heat exchanger networks to minimize energy consumption *(Chemical Engineering)*

- How can we make decisions in the "*best*" way possible?
  → **Optimal decision-making using optimization**



Picture credits:
[1] https://www.ns.nl/en/travel-information/improving-services/train-schedules.html
[2] "Windsurfing At St Thomas" by 2Stef34 is marked with CC0 1.0.
[3] "Jamnagar Refinery" by Reliance Industries is licensed under CC BY-SA 4.0.

Definition of mathematical optimization

“Mathematical optimization or mathematical programming is the selection of a **best element**, with regard to some **criteria**, from some set of **available alternatives**.” [1][2]

Example TU Delft food truck:

“Criteria” → best taste, most healthy, cheapest
“Alternatives” → menue
“Constraints” → vegetarian/vegan

[1] Bradley, S. P., Hax, A. C., & Magnanti, T. L. (1977).. *Applied mathematical programming* Addison-Wesley.
[2] Williams, H. P. (2013). *Model building in mathematical programming*. John Wiley & Sons.
https://en.wikipedia.org/wiki/Mathematical_optimization
Picture credit: https://www.facebook.com/FoodAndMoreTUDelft/

Objective function

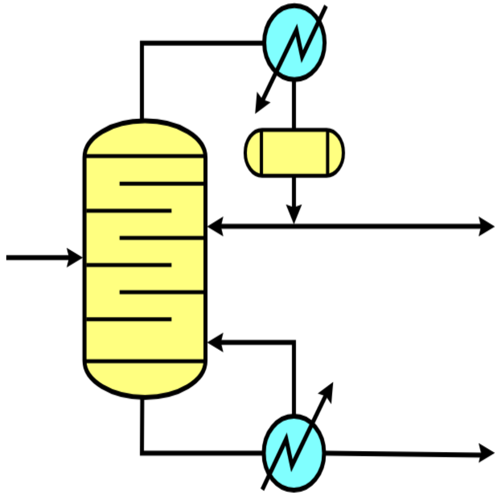
- Objective function quantifies the performance of different decisions (→ Criteria for judging what is best), e.g.,
  - Maximize profit
  - Minimize CO_2 emissions
- Choosing an appropriate objective is hard
- It is possible to specify multiple objective functions → "Multi-objective optimization"
- Equivalence of formulations: $\min_{\mathbf{x}} f(\mathbf{x}) \equiv \max_{\mathbf{x}} -f(\mathbf{x})$
- **Convention**: We formulate everything as minimization problems

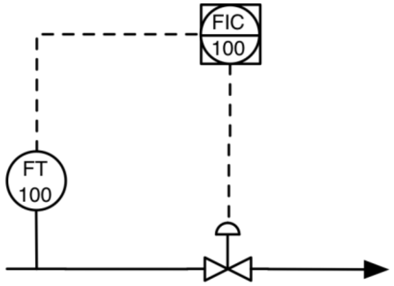
Variables and degrees of freedom

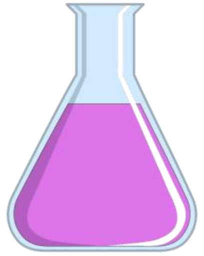
- Variables $\boldsymbol{x}$ can be varied and acts as arguments to the objective function $f(\boldsymbol{x})$
- If no constraints exist, the number of variables corresponds to the **degree of freedom** (DoF)
- $\boldsymbol{x}$ is usually a set of N continuous variables that can have any value within a given compact domain $[\boldsymbol{x}^L, \boldsymbol{x}^U]$ (bounds) , over a subset of $\mathbb{R}^N$
- Variables can also be a set of discrete choices (c.f., Lecture Q2 L5)

See CP course Q2 week 5

Example of variables in chemical engineering







Process design

size of heat exchanger, number of stages, volume of reactor,…

Process operation

operating pressure, valve opening,….

Experiments

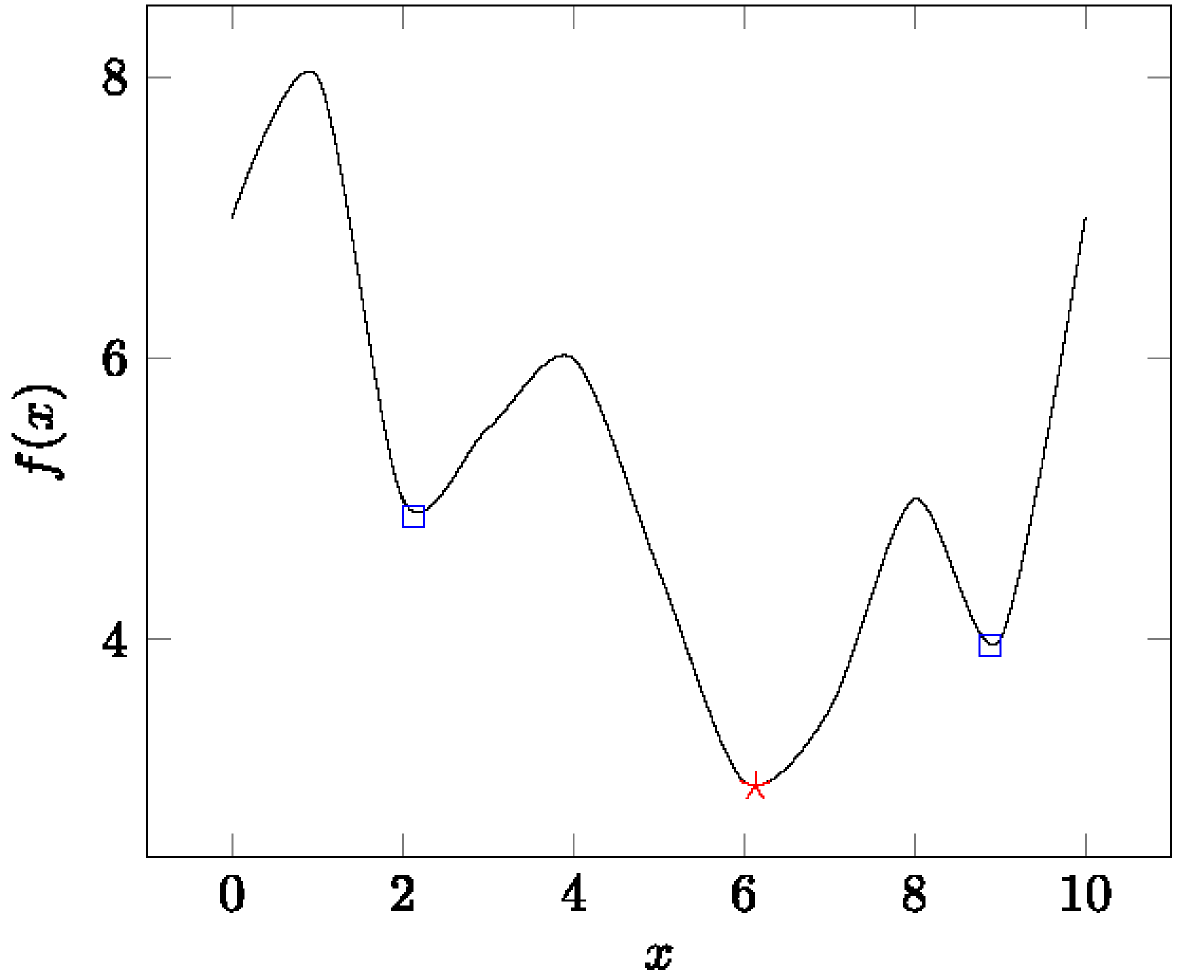
flowrate of pumps, temperature of reactor,…

https://upload.wikimedia.org/wikipedia/commons/4/45/Chembioeng.svg
https://ndla.no/nn/subject:1:5a5cac3f-46ff-4f4d-ba95-b256a706ec48/topic:2:58329/topic:2:116448/resource:1:116924
https://images.rawpixel.com/image_800/czNmcy1wcml2YXRlL3Jhd3BpeGVsX2ltYWdlcy93ZWJzaXRlX2NvbnRlbnQvam9iNjgwLTA3OC1sMWRidHM1Zi5qcGc.jpg

Agenda

- **Basics of mathematical optimization**
- **Conditions for optimality**
  - Necessary condition for optimality
  - Convexity of a function
  - Sufficient condition for optimality
- **Unconstrained Optimization**
  - Gradient descent algorithm pseudo-code
  - First-order search methods
  - Second-order search methods
  - Unconstrained optimization using SciPy

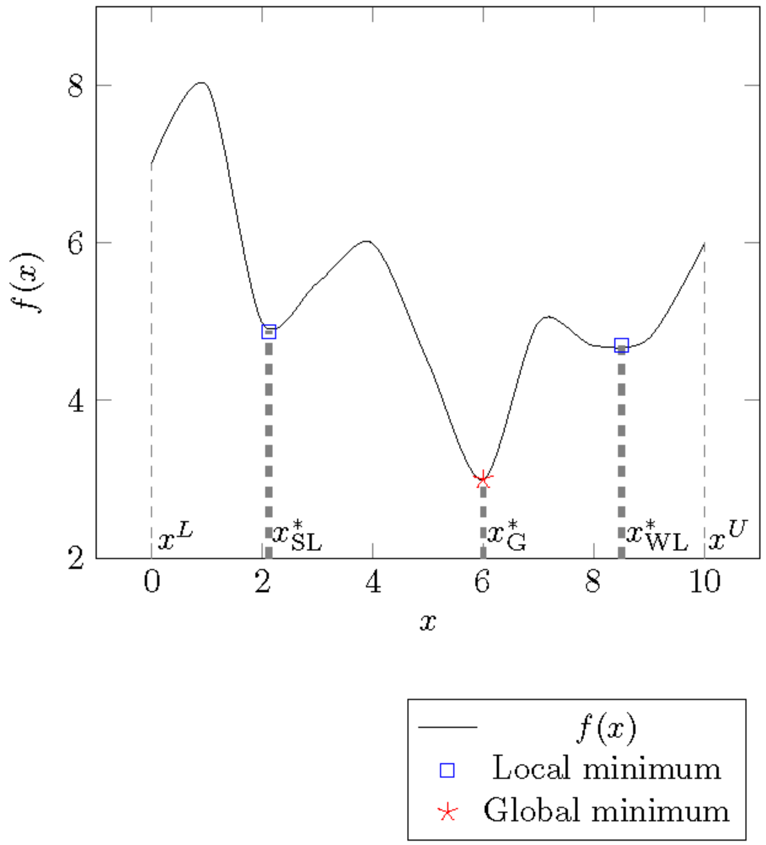
1D example: minimize $f(x)$ with x in $[x^L, x^U]$



Definition: Local minimum

Iff $f(x^*) \leq f(x)$ for all x in neighborhood of x^*,

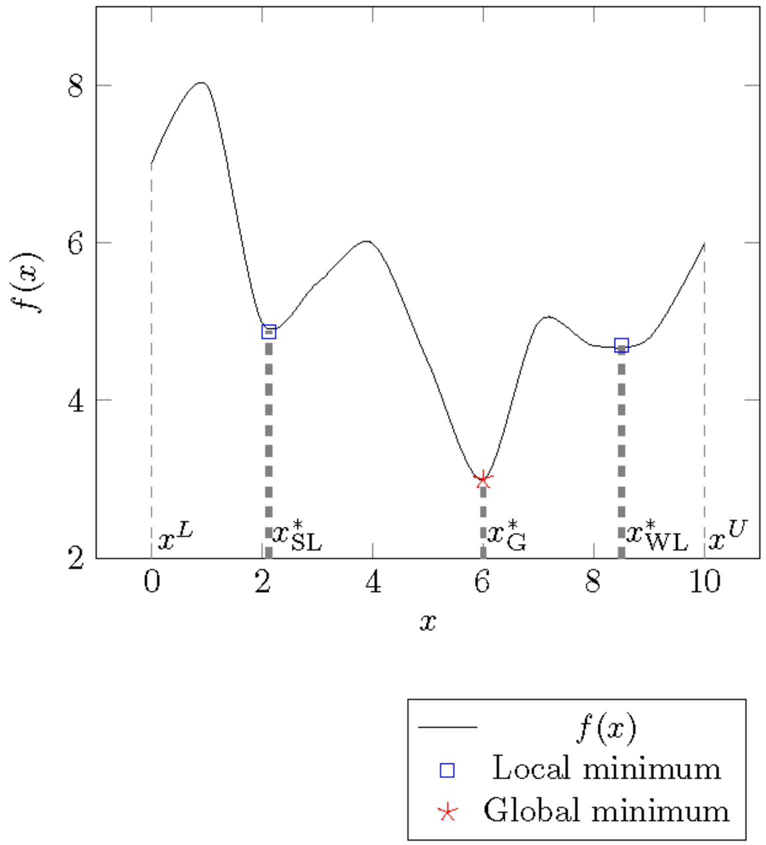
then x^* is a **local minimum.**



Iff: "if and only if"

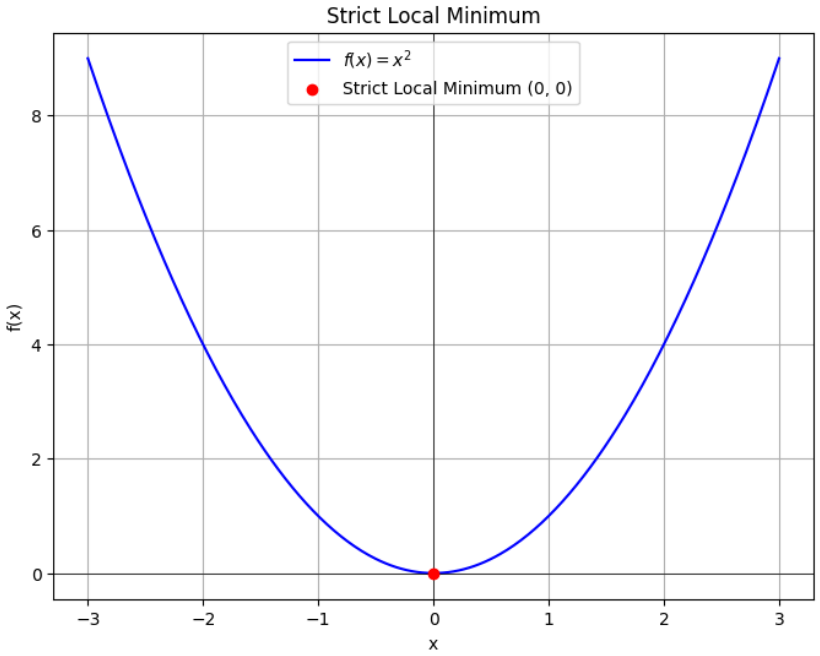
Definition: Global minimum

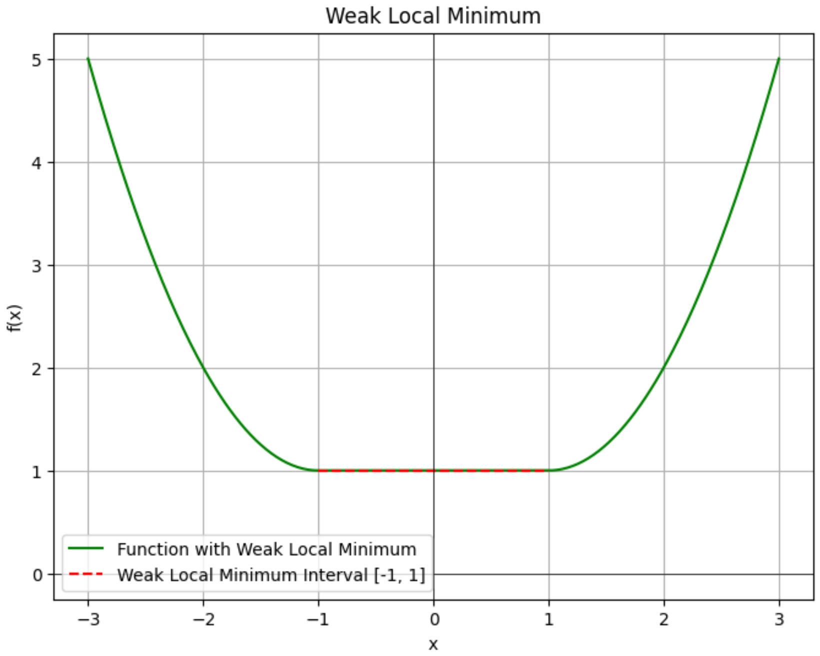
Iff $f(x^*) \leq f(x)$ **for all** x in $[x^L, x^U]$, then x^* is a **global minimum.**



Iff: "if and only if"

Definition: Weak and strict minima

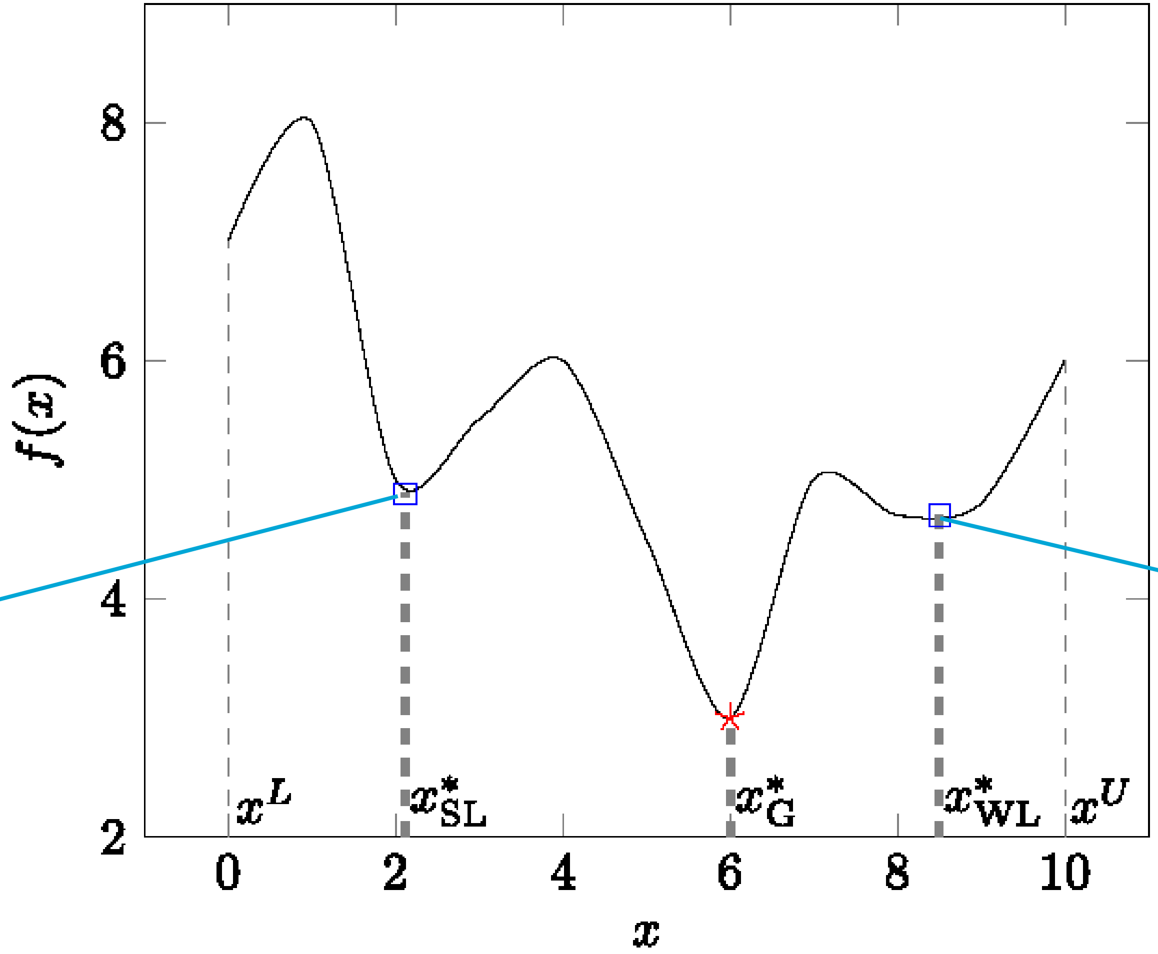




$f(x^*) < f(x)$ for all x in neighbourhood of x^* , then x^* is a **strict** local minimum

$f(x^*) \leq f(x)$ for all x in neighbourhood of x^* , then x^* is a **weak** local minimum

Example for weak and strict minima

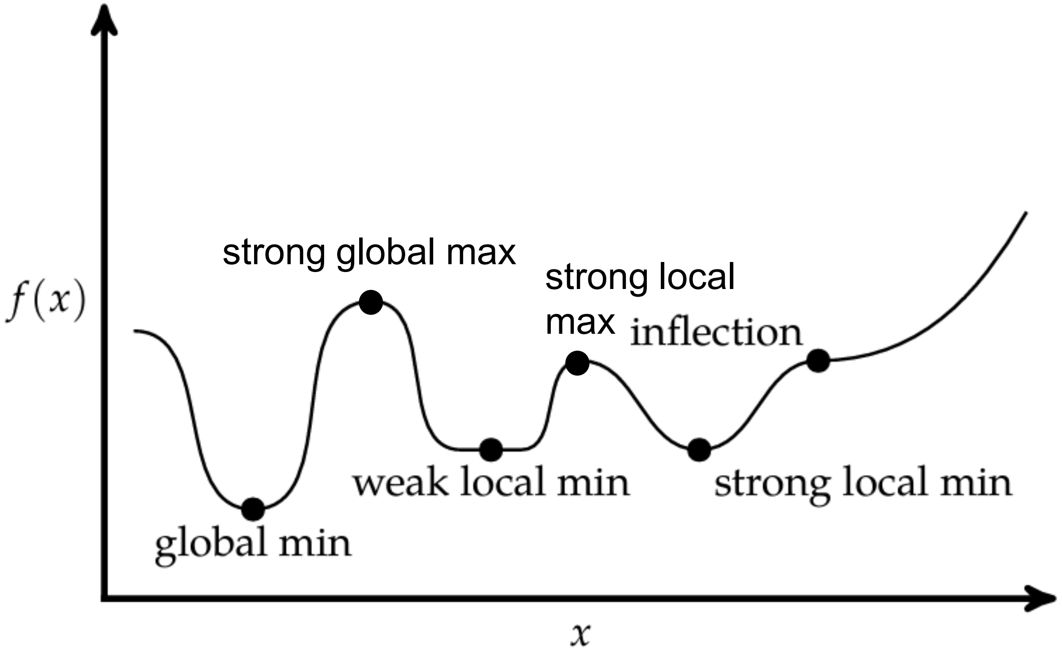


Strict minimum

Weak minimum

Critical point, aka stationary points

- For a Lipschitz continuous and twice-differentiable function, a critical point x^* is defined as:
  - $f'(x^*) = 0$ → First-derivative of $f(x)$ at x^* is zero
- A critical point can be a:
  - Minimum
  - Maximum
  - Inflection point
- **$f'(x^*) = 0$ is a necessary condition for (local) optimality**



Definition: Lipschitz continuous

Deep Dive

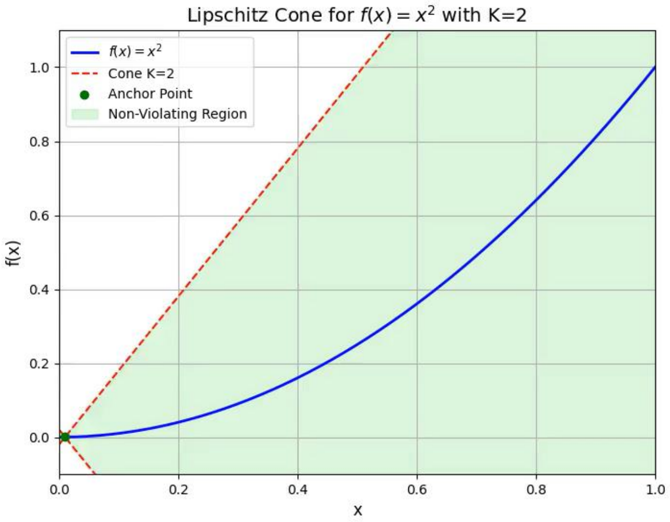
- A function $f\colon \mathbb{R}^n \rightarrow \mathbb{R}$ is Lipschitz continuous if there exists a constant $L \geq 0$, called the Lipschitz constant, such that for all points $x, y \in \mathbb{R}^n$:

$$|f(x) - f(y)| \leq L||x - y||$$

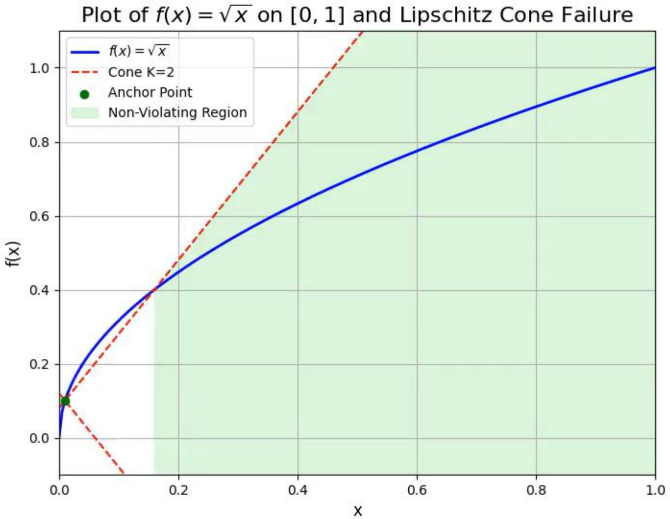
Here, $||x - y||$ is the distance between x and y measured using the Euclidean norm

- All Lipschitz continuous functions are continuous, but not all continuous functions are Lipschitz continuous

Lipschitz continuous



Non-Lipschitz continuous

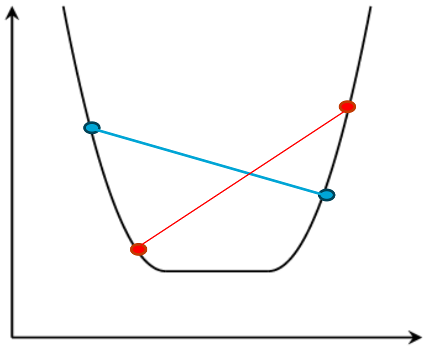


Agenda

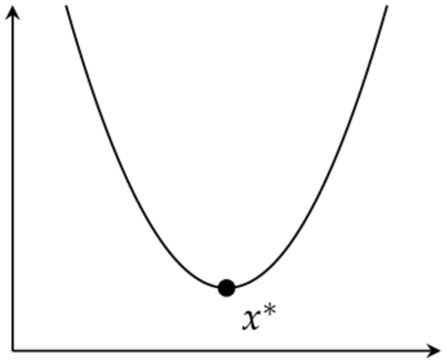
- Basics of mathematical optimization
- **Conditions for optimality**
  - Necessary condition for optimality
  - Convexity of a function
  - Sufficient condition for optimality
- Unconstrained Optimization
  - Gradient descent algorithm pseudo-code
  - First-order search methods
  - Second-order search methods
  - Unconstrained optimization using SciPy

Definition: Convexity of functions

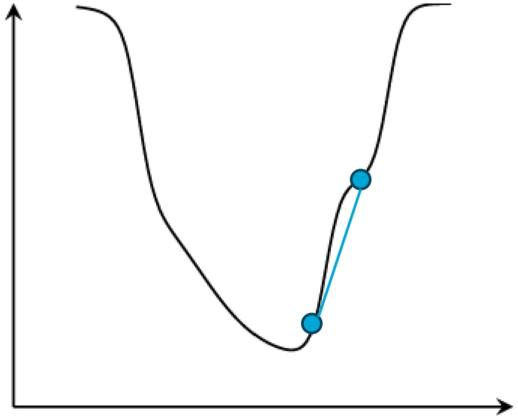
- A function $f(\boldsymbol{x})$ is (strictly) convex over a domain $[\boldsymbol{x}^L, \boldsymbol{x}^U] \in X$, if and only if for any two points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ in X, if
  - Convex:
    $f(\alpha \boldsymbol{x}_1 + (1 - \alpha)\, \boldsymbol{x}_2) \le \alpha f(\boldsymbol{x}_1) + (1 - \alpha) f(\boldsymbol{x}_2), \forall\, \alpha \in [0,1]$
  - Strictly convex:
    $f(\alpha \boldsymbol{x}_1 + (1 - \alpha)\, \boldsymbol{x}_2) < \alpha f(\boldsymbol{x}_1) + (1 - \alpha) f(\boldsymbol{x}_2), \forall\, \alpha \in [0,1]$ → strictly convex



Convex function



Strictly convex function

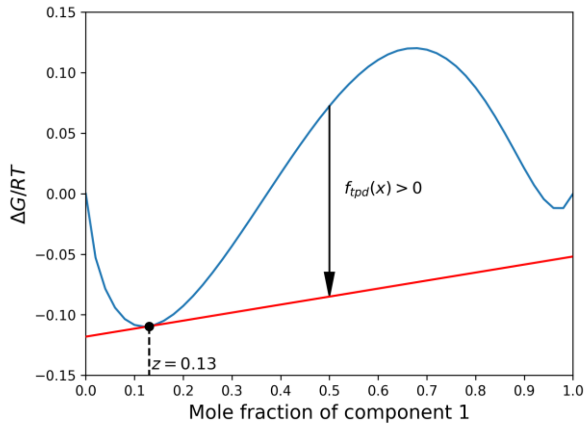


Non-convex function

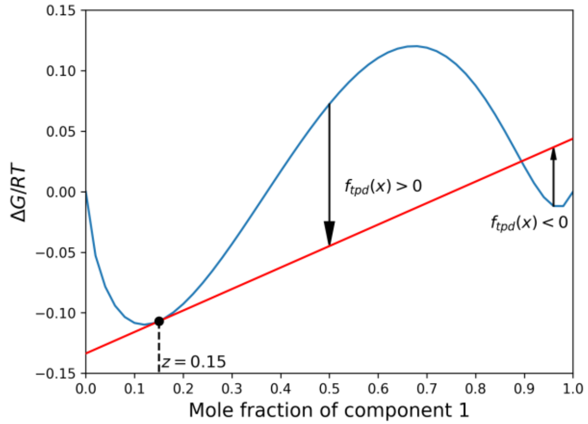
Importance of convexity for optimization

- Convexity helps us to determine whether a function will have a local minima or not (or determine global minimum):
  - If a function is convex → the critical points are weak global minimum
  - If a function is **strictly** convex → the critical point is a unique, **strong** global minimum
  - If a function is non-convex → it may have critical points some of which are not global minimum
- Global optimization is important for several problems in chemical engineering:
  - Phase stability and phase equilibrium calculations[1]
- "… in fact, the great watershed in optimization isn't between *linearity* and *nonlinearity*, but *convexity* and *nonconvexity*."[2]

Stable



Metastable



[1] Michelsen, M. L. (1982). The isothermal flash problem. Part I. Stability. *Fluid phase equilibria*, *9*(1), 1-19.
[2] Rockafellar, R. T. (1993). Lagrange multipliers and optimality. *SIAM review*, *35*(2), 183-238.

Testing for convexity – univariate function

- Previously shown definition of convexity not very practical for testing
- Convexity depends on the curvature of the function → second-order derivative information required
- Conditions for convexity:
  - $f(x)$ is convex if $f''(x) \geq 0 \; \forall \, x \; \in [x^L, x^U]$
  - $f(x)$ is **strictly** convex if $f''(x) > 0 \; \forall \, x \; \in [x^L, x^U]$
- Example for univariate function:
  - $f(x) = x^2$ is strictly convex as $f''(x)$ = 2 > 0 for any value of x
  - $f(x) = 2x^2 - x^3$ can be either convex or concave depending on the values of x since $f''(x) = 4 - 6x$

Testing for convexity – multivariate function

- For multivariate functions:
  - $f(\boldsymbol{x})$ is convex if its Hessian matrix $\boldsymbol{H}(\boldsymbol{x})$ is positive semi-definite $\forall\, \boldsymbol{x} \in [\boldsymbol{x}^L, \boldsymbol{x}^U]$
  - $f(\boldsymbol{x})$ is **strictly** convex if its Hessian matrix $\boldsymbol{H}(\boldsymbol{x})$ is positive definite $\forall\, \boldsymbol{x} \in [\boldsymbol{x}^L, \boldsymbol{x}^U]$
- **Definition**: A matrix is…
  - positive semi-definite if and only all of its eigenvalues are non-negative
  - positive definite if and only all of its eigenvalues are positive
- Refer to Q1 Lecture 3 to understand how to determine eigenvalues of a matrix

See CP course Q1 week 3

Example: Test multivariate function for convexity

$f(x_1, x_2) = 2x_1^2 + 2x_1x_2 + 1.5x_2^2 + 7x_1 + 8x_2 + 24$

$$H(x_1, x_2) = \nabla^2 f(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 2 \\ 2 & 3 \end{bmatrix}$$

Determine eigenvalues via characteristic polynomial $|H - \lambda I| = 0$

- Both eigenvalues are positive → f is **strictly** convex

```python
# Import relevant libraries
import numpy as np
import scipy

# Defining the Hessian matrix as calculated in the slides
hessian_matrix = np.array([[4,2], [2,3]])

# Refer to Q1L3 - Slide 26 to recap how to determine eigenvalues of a matrix
eigenvalue, eigenvector = scipy.linalg.eig(hessian_matrix)

print("Eigenvalues are: ", eigenvalue)
```

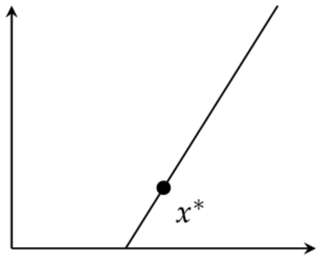
```
Eigenvalues are:  [5.56155281+0.j 1.43844719+0.j]
```

Agenda

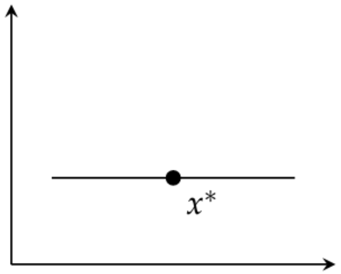
- **Basics of mathematical optimization**
- **Conditions for optimality**
  - Necessary condition for optimality
  - Convexity of a function
  - Sufficient condition for optimality
- **Unconstrained Optimization**
  - Gradient descent algorithm pseudo-code
  - First-order search methods
  - Second-order search methods
  - Unconstrained optimization using SciPy

Conditions for local optimality – univariate function

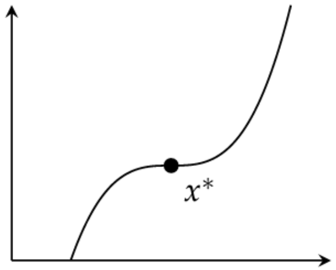
- For x^* to be a **local** minimum, we need:
  - $f'(x^*) = 0$ → First-order *necessary* condition (**FONC**)
  - $f''(x^*) \geq 0$ → Second-order *necessary* condition (**SONC**)
- For x^* to be a **strong local** minimum, we need:
  - $f'(x^*) = 0$ → First-order *necessary* condition (**FONC**)
  - $f''(x^*) > 0$ → Second-order *sufficient* condition (**SOSC**)
- SOSC can distinguish between weak **local** minima, inflection points and strong **local** minima



SONC but not FONC → not a minima



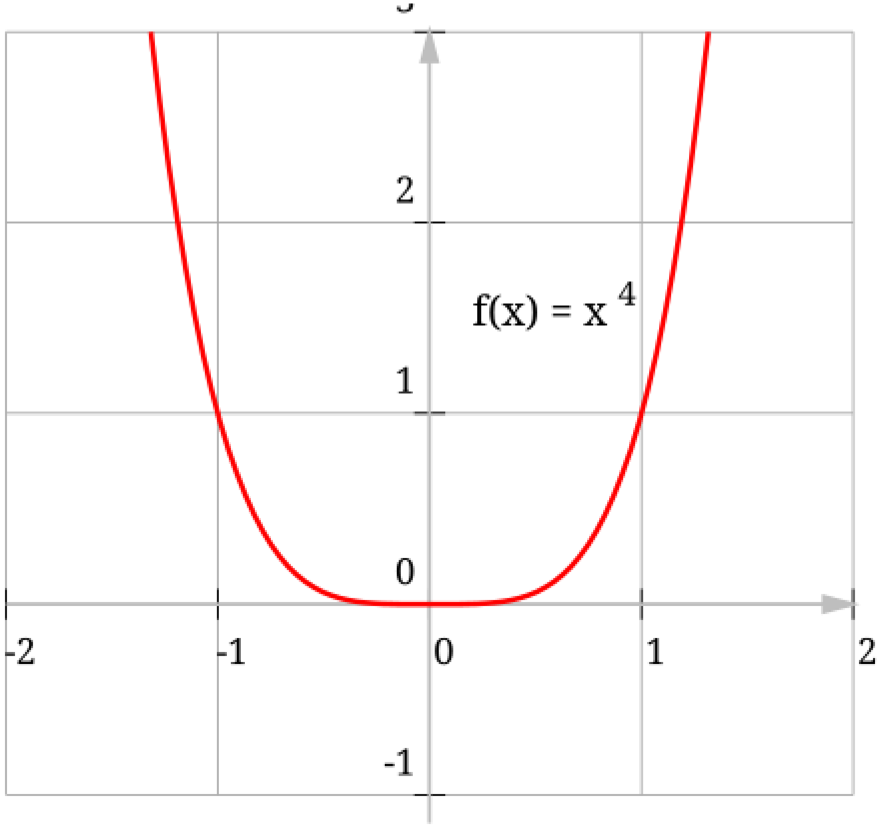
FONC and SONC → not a strong minima



FONC and SONC → not a strong minima

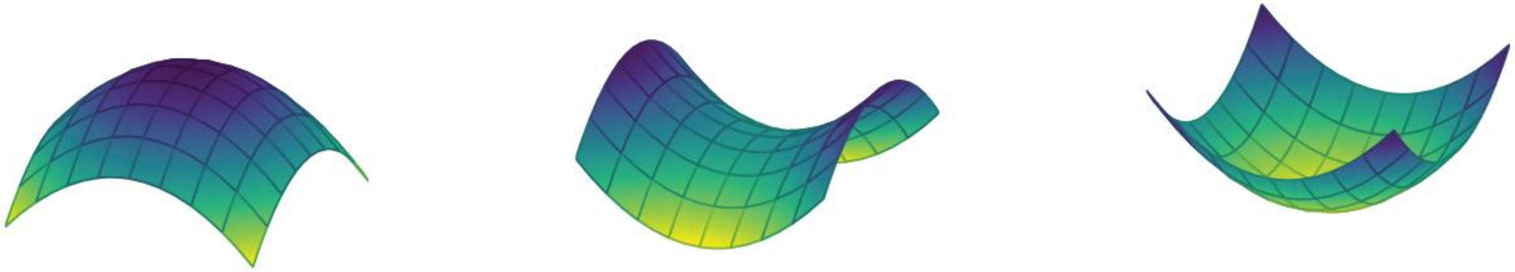
Try this at home!

- Apply the first and second order optimality conditions to the function
  - $f(x) = x^4$
- Answer the following questions
  - Does the function have a minimum?
  - Is it s local or a global minimum?
  - Are the FONC and SONC, and SOSC met?
  - What did you learn from this example?



Conditions for local optimality for **multivariate** function

- $\nabla f(\boldsymbol{x}^*) \ = 0$ → First-order *necessary* condition (**FONC**)
- $\nabla^2 f(\boldsymbol{x}^*)$ is a **positive semi-definite matrix** → Second-order *necessary* condition (**SONC**)
- $\nabla^2 f(\boldsymbol{x}^*)$ is a **positive definite matrix** → Second-order *sufficient* condition (**SOSC**)



A *local maximum*. The gradient at the center is zero, but the Hessian is negative definite.

A *saddle*. The gradient at the center is zero, but it is not a local minimum.

A *bowl*. The gradient at the center is zero and the Hessian is positive definite. It is a local minimum.

Agenda

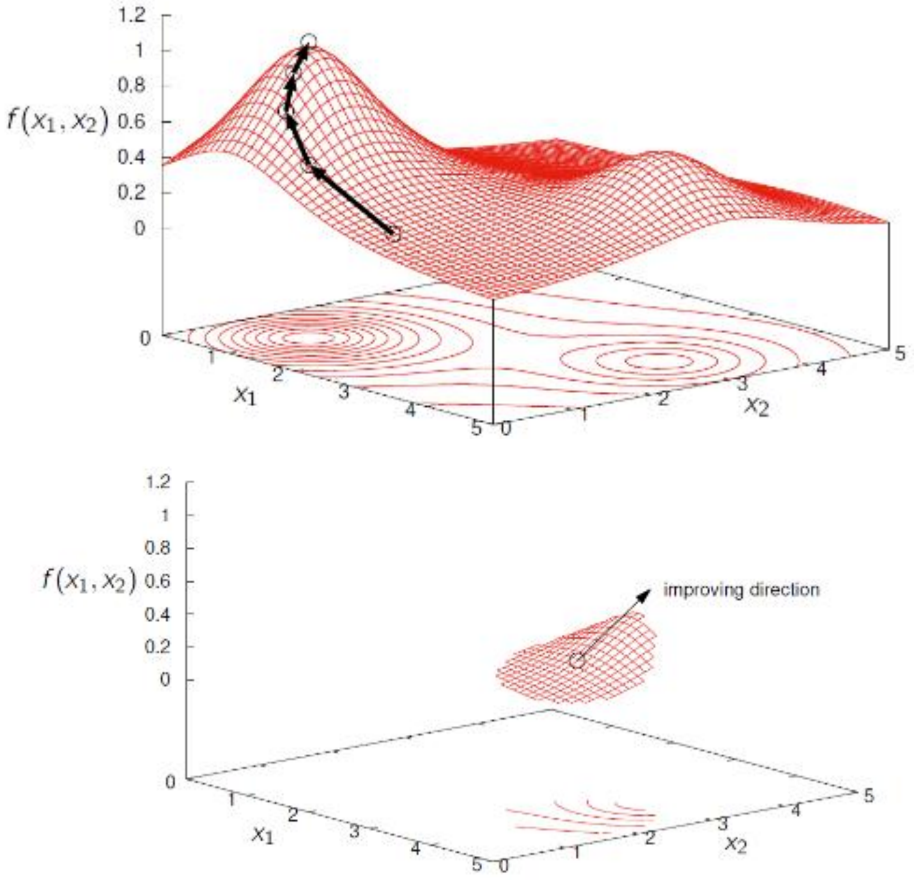
- **Basics of mathematical optimization**
- **Conditions for optimality**
  - Necessary and sufficient conditions
  - Convexity of a function
- **Unconstrained Optimization**
  - Gradient descent algorithm pseudo-code
  - First-order search methods
  - Second-order search methods
  - Unconstrained optimization using SciPy

Agenda

- **Basics of mathematical optimization**
- **Conditions for optimality**
  - Necessary and sufficient conditions
  - Convexity of a function
- **Unconstrained Optimization**
  - Gradient descent algorithm pseudo-code
  - First-order search methods
  - Second-order search methods
  - Unconstrained optimization using SciPy

Methods for unconstrained optimization

- How do we numerically search for optimal points?
- Descent direction algorithms
  - Start at an initial point
  - Determine a descent direction based on some local information
  - Make a step in this direction
  - Repeat this procedure to converge to a local minimum

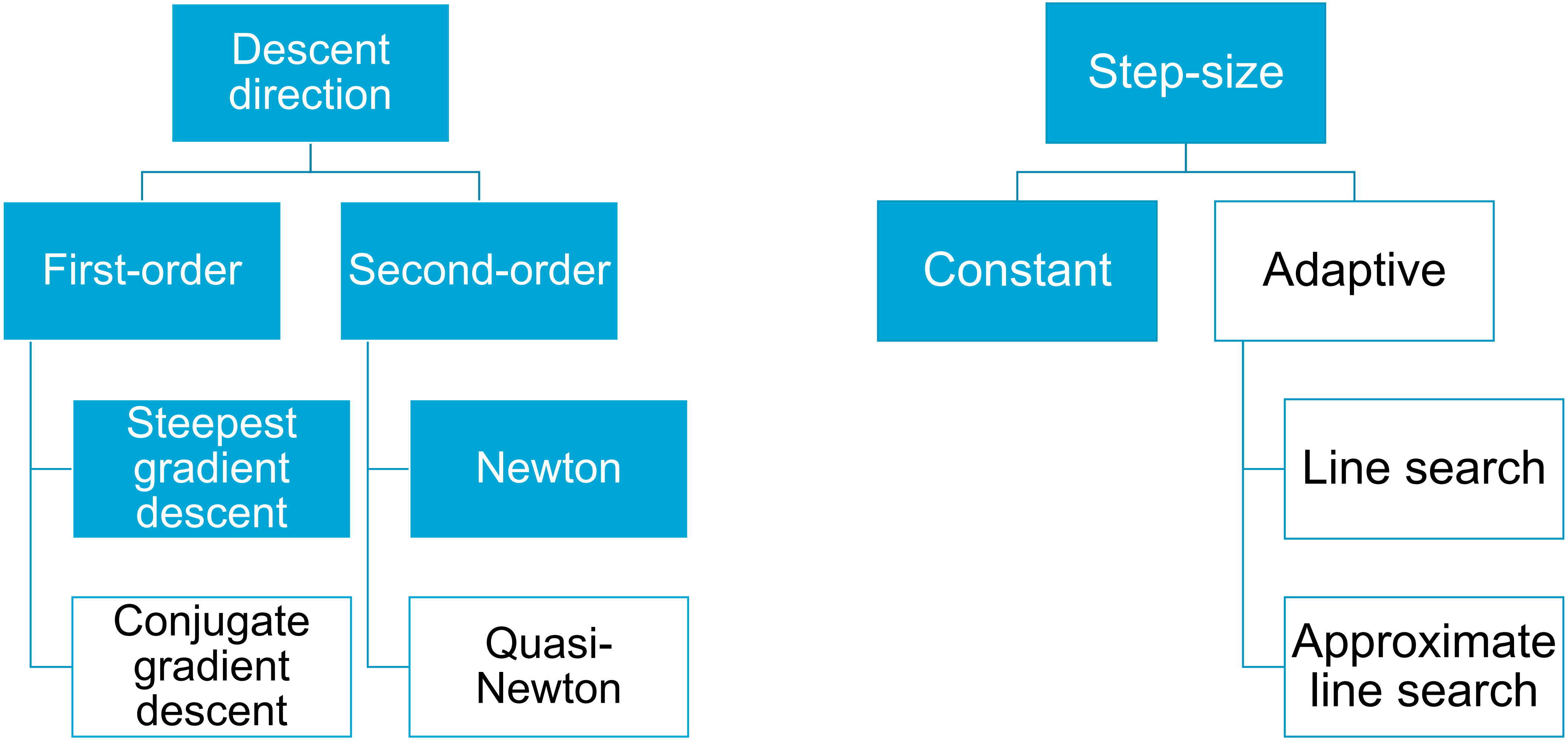


[1] Image courtesy: Lecture slides for Process Optimization by Professor Benoit Chachuat (Imperial College London)

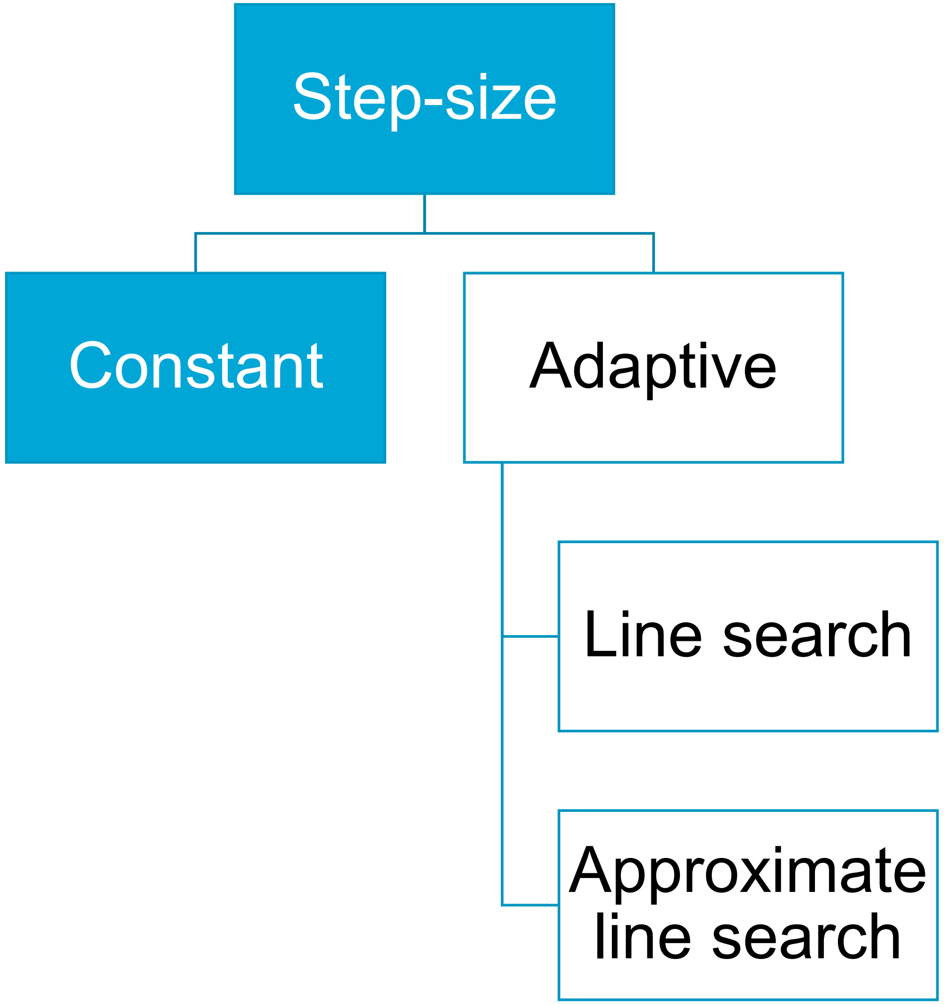
Descent direction algorithm: pseudo-code

- Algorithm
  a. Initialize $\boldsymbol{x}^{(1)}$ and $k = 1$
  b. Determine the *descent* direction $\boldsymbol{d}^{(k)}$ using information such as the gradient or hessian.
  c. Determine the step size $\alpha^{(k)}$ (called *learning rate* in machine learning)
  d. Set $\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{d}^{(k)}$ and $k \leftarrow k+1$
  e. Terminate if $f(\boldsymbol{x}^{(k+1)}) - f\left(\boldsymbol{x}^{(k)}\right) < \epsilon$ or if $k > k^{max}$
- Vector $\boldsymbol{d}^{(k)}$ is a descent direction at $\boldsymbol{x}^{(k)}$ if $\nabla f\left(\boldsymbol{x}^{(k)}\right)^T \boldsymbol{d} < 0$
- **Open questions:**
  - Determining *descent* direction $\boldsymbol{d}^{(k)}$
  - Determining step size $\alpha^{(k)}$

Determining the descent direction and step-size

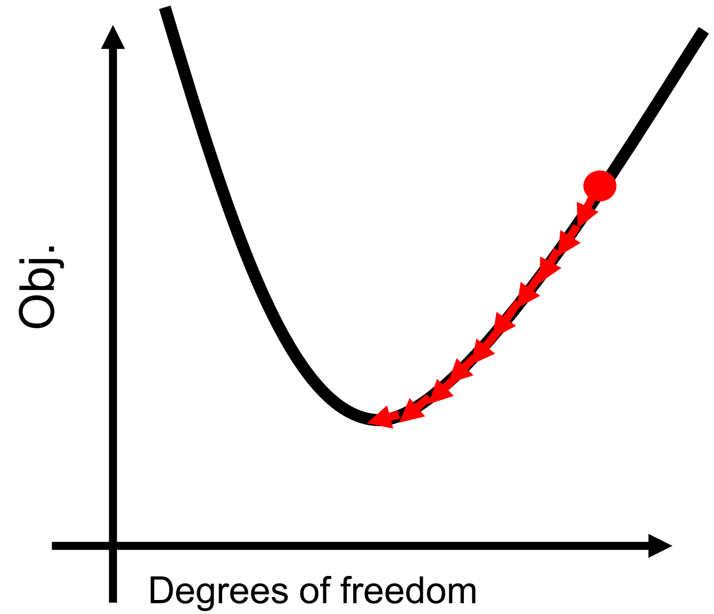


Determining the descent direction and step-size



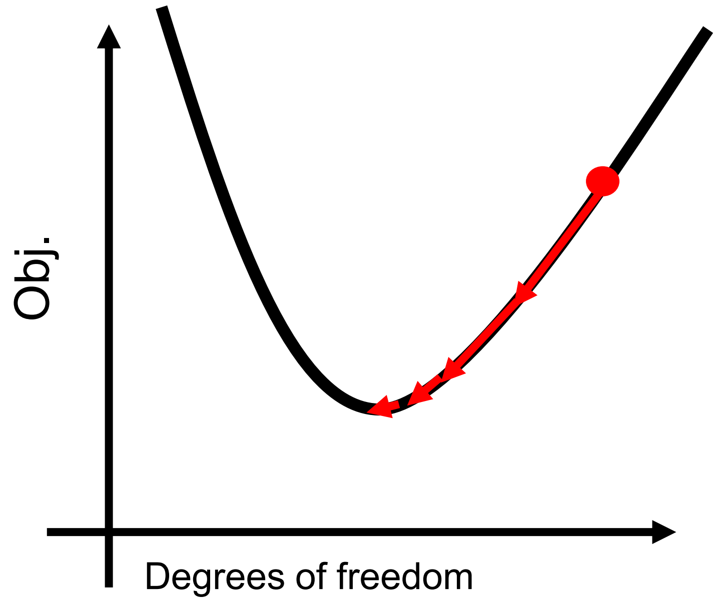
The influence of the step-size (learning rate)

Too low



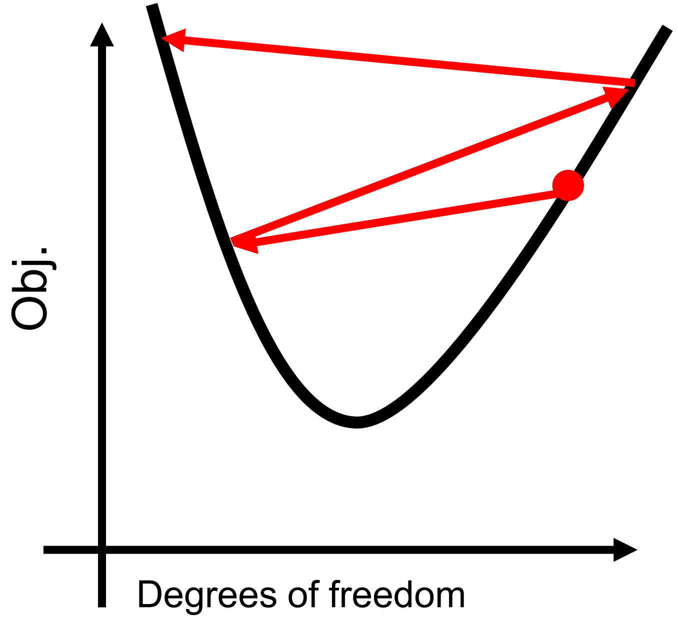
A small step size requires many updates before reaching the minimum point

Just right



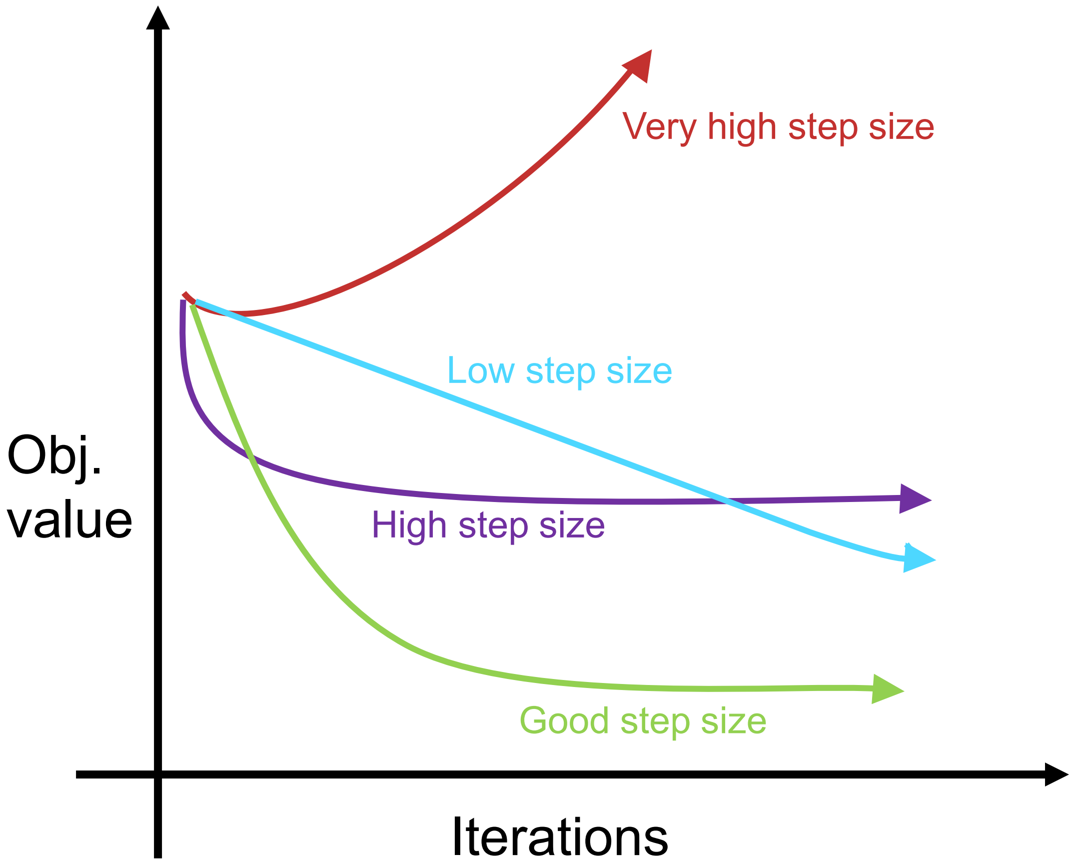
The optimal step size swiftly researches the minimum point

Too high

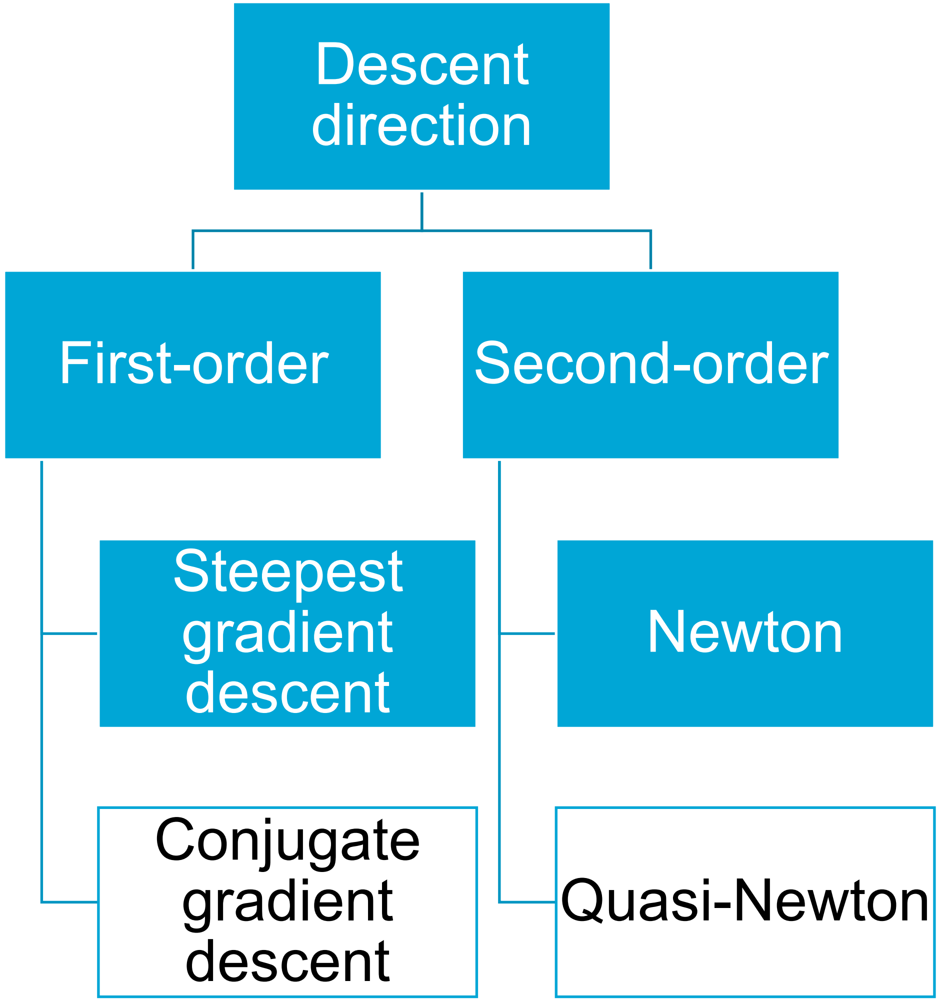


Too large of a step size causes drastic updates which lead to divergent behaviours (e.g. overshooting)

The influence of step-size (learning rate) on convergence



Determining the descent direction and step-size

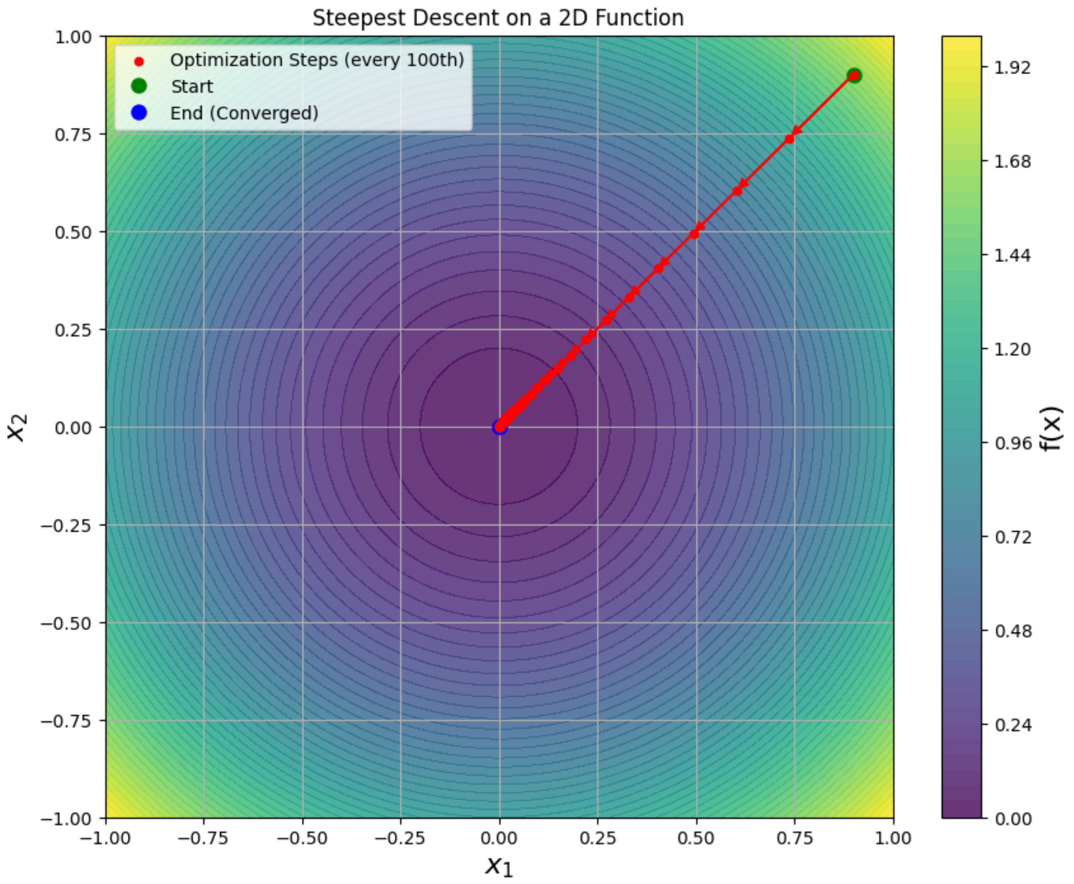


Agenda

- **Basics of mathematical optimization**
- **Conditions for optimality**
  - Necessary and sufficient conditions
  - Convexity of a function
- **Unconstrained Optimization**
  - Gradient descent algorithm pseudo-code
  - First-order search methods
  - Second-order search methods
  - Unconstrained optimization using SciPy

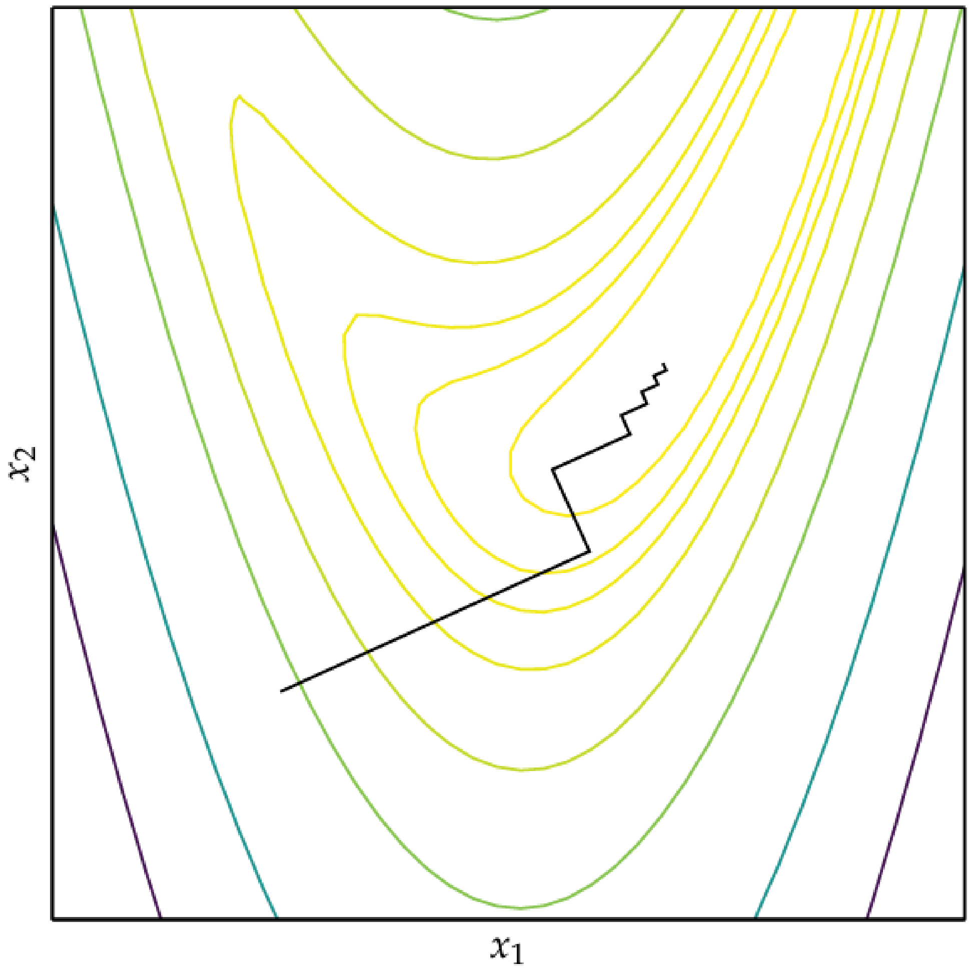
First-order methods

- Use first derivative information to update the descent direction to determine the local minimum
- Defining $\boldsymbol{g}^{(k)} = \nabla f(\boldsymbol{x}^{(k)})$
- For steepest descent method:
  - $\boldsymbol{d}^{(k)} = -\frac{\boldsymbol{g}^{(k)}}{\|\boldsymbol{g}^{(k)}\|}$ (normalize the direction of steepest descent)



Gradient descent be slow for some problems

- If the functional behavior exhibits narrow valleys, gradient descent performs **poorly**
- Example: Slow convergence of steepest descent method for Rosenbrock function
- More advanced methods incorporate faster gradient calculations such as Momentum, Nesterov, Adam, etc. and are also used to train deep neural networks
  → Our AI elective cover some of those



Agenda

- **Basics of mathematical optimization**
- **Conditions for optimality**
  - Necessary and sufficient conditions
  - Convexity of a function
- **Unconstrained Optimization**
  - Gradient descent algorithm pseudo-code
  - First-order search methods
  - Second-order search methods
  - Unconstrained optimization using SciPy

Second-order methods

- First-order methods lack information about the curvature of the function → cannot determine how far a step to take
- Second-order information can help address this issue and accordingly determine the descent direction
- Notion of step size can be eliminated using second-order information:

$$\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}^{(k)} + \boldsymbol{d}^{(k)}$$

- How can we link second-order information to step size?

How to determine the descent direction?

- Consider second-order Taylor series approximation of univariate function $f(x)$ at point $x^{(k)}$

$$f(x) \approx f(x^{(k)}) + (x - x^{(k)})f'(x^{(k)}) + \frac{(x - x^{(k)})^2}{2} f''(x^{(k)})$$

- We wish to minimize $f(x)$, on applying First-order necessary condition (FONC):

$$f'(x) = 0$$

$$f'\left(x^{(k)}\right) + \left(x - x^{(k)}\right)f''\left(x^{(k)}\right) = 0$$

$$x^{(k+1)} = x^{(k)} - {f'\left(x^{(k)}\right)}\Big/{f''(x^{(k)})}$$

Determining descent direction can be interpreted as a root-finding method applied to $\nabla f(\boldsymbol{x})$ → check Q1 Lecture 3

See CP course Q1 week 3

Determining the descent direction – multivariate case

- Consider second-order Taylor series approximation of multivariate function $f(\boldsymbol{x})$ at point $\boldsymbol{x}^{(k)}$

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(k)}) + \left(\mathbf{g}^{(k)}\right)^{\top}(\boldsymbol{x} - \boldsymbol{x}^{(k)}) + \frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{x}^{(k)}\right)^{\top} \mathbf{H}^{(\mathrm{k})}(\boldsymbol{x} - \boldsymbol{x}^{(k)})$$

We wish to minimize $f(x)$, on applying First-order necessary condition (FONC):

$$\nabla f(x) = 0$$

$$\mathbf{g}^{(k)} + \mathbf{H}^{(\mathrm{k})}(\boldsymbol{x} - \boldsymbol{x}^{(k)}) = 0$$

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - {\mathbf{H}^{(\mathrm{k})}}^{-1}\mathbf{g}^{(k)}$$

$\mathbf{g}^{(k)}$ is the gradient of the function and $\mathbf{H}^{(\mathrm{k})}$ is the Hessian of the function at $\boldsymbol{x}^{(k)}$

- If the function is quadratic and its Hessian is positive definite, the update step in Newton method converges to global minimum in one step
- Newton methods exhibit *quadratic* convergence behavior

Live coding: Basic implementation of Newton's method

- Open Colab: [Implementation of Newton's method]

- Find more in the Github repository of the course: https://github.com/process-intelligence-research/computational_practicum_lecture_coding/tree/main

Limitations of Newton's method

- Some limitations of Newton's method:
  - Determining the Hessian matrix is computationally expensive
  - Inverting matrices further adds to the computational burden (c.f. Q1 Lecture 3)
- Newton method **does not** work well when:
  - Second derivative is zero → first derivative is a linear hyperplane
  - Second derivative is very close to zero → next iterate will lie very far from the current point
  - The descent direction is reliable when the difference between the true function and the quadratic approximation is not too large
  - When $\boldsymbol{H}^{(k)}$ is not positive definite, the descent direction may be ill-defined or may not satisfy the condition for descent
- Quasi-Newton methods aim to address some of these limitations (implementations available in popular packages)

Agenda

- **Basics of mathematical optimization**
- **Conditions for optimality**
  - Necessary and sufficient conditions
  - Convexity of a function
- **Unconstrained Optimization**
  - Gradient descent algorithm pseudo-code
  - First-order search methods
  - Second-order search methods
  - Unconstrained optimization using SciPy

Unconstrained optimization with **SciPy**

- SciPy (scipy.optimize) provides powerful tools for numerical optimization.
- Ideal for solving linear, nonlinear, and constrained optimization problems.
- Easy-to-use interface for various optimization tasks.

```python
# Import SciPy
from scipy.optimize import minimize

# Define the objective function
def objective(x):
    return (x - 3)**2

# Perform optimization
result = minimize(objective, x0=0)

# Print the result
print("Optimal value:", result.x)
```

Output from the example:

```
Optimal value: [2.99999998]
```

Live coding: Unconstrained optimization in SciPy

- Open Colab: [Unconstrained optimization in SciPy]

- Find more in the Github repository of the course: https://github.com/process-intelligence-research/computational_practicum_lecture_coding/tree/main

Learning goals of this lecture

After successfully completing this lecture, you are able to. . . .

- explain what constitutes an unconstraint mathematical optimization problem
- apply the optimality conditions to unconstraint optimization problems
- explain numerical methods to solve unconstrained optimization problems
- implement numerical methods to solve unconstrained optimization problems

Thank you very much for your attention!