

# PV027 Optimization

Tomáš Brázdil

# Resources & Prerequisites

## Resources:

- ▶ Lectures & tutorials (the **main** resources)
- ▶ Books:

Joaquim R. R. A. Martins and Andrew Ning. Engineering Design Optimization. Cambridge University Press, 2021. ISBN: 9781108833417.

Jorge Nocedal and Stephen J. Wright. Numerical optimization. Springer, 2006. ISBN: 0387303030.

# Resources & Prerequisites

## Resources:

- ▶ Lectures & tutorials (the **main** resources)
- ▶ Books:

Joaquim R. R. A. Martins and Andrew Ning. Engineering Design Optimization. Cambridge University Press, 2021. ISBN: 9781108833417.

Jorge Nocedal and Stephen J. Wright. Numerical optimization. Springer, 2006. ISBN: 0387303030.

We shall need elementary knowledge and understanding of

- ▶ Linear algebra in  $\mathbb{R}^n$   
Operations with vectors and matrices, bases, diagonalization.
- ▶ Multi-variable calculus (i.e., in  $\mathbb{R}^n$ )  
Partial derivatives, gradients, Hessians, Taylor's theorem.

We will refresh our memories during lectures and tutorials.

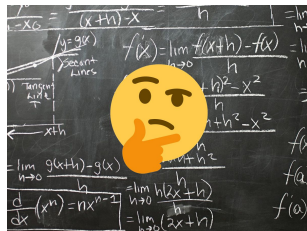
# Evaluation

**Oral exam** - You will get a manual describing the knowledge necessary for **E** and better.

There might be homework assignments that you may discuss at tutorials, but (for this year) there is no mandatory homework.

Please be aware that

This is a **difficult math-based course**.





# What is Optimization

## **Merriam Webster:**

An act, process, or methodology of making something (such as a design, system, or decision) as perfect, functional, or effective as possible.

*specifically*: the mathematical procedures (such as finding the maximum of a function) involved in this.

# What is Optimization

## **Merriam Webster:**

An act, process, or methodology of making something (such as a design, system, or decision) as perfect, functional, or effective as possible.

*specifically*: the mathematical procedures (such as finding the maximum of a function) involved in this.

## **Britannica**

Collection of mathematical principles and methods for solving quantitative problems in many disciplines, including physics, biology, engineering, economics, and business.

Historically, (mathematical/numerical) optimization is called *mathematical programming*.

# Optimization

People optimize in

- ▶ scheduling
  - ▶ transportation,
  - ▶ education,
  - ▶ ...

# Optimization

People optimize in

- ▶ scheduling
  - ▶ transportation,
  - ▶ education,
  - ▶ ...
- ▶ investments
  - ▶ portfolio management,
  - ▶ utility maximization,
  - ▶ ...

# Optimization

People optimize in

- ▶ scheduling
  - ▶ transportation,
  - ▶ education,
  - ▶ ...
- ▶ investments
  - ▶ portfolio management,
  - ▶ utility maximization,
  - ▶ ...
- ▶ industrial design
  - ▶ aerodynamics,
  - ▶ electrical engineering,
  - ▶ ...

# Optimization

People optimize in

- ▶ scheduling
  - ▶ transportation,
  - ▶ education,
  - ▶ ...
- ▶ investments
  - ▶ portfolio management,
  - ▶ utility maximization,
  - ▶ ...
- ▶ industrial design
  - ▶ aerodynamics,
  - ▶ electrical engineering,
  - ▶ ...
- ▶ sciences
  - ▶ molecular modeling,
  - ▶ computational systems biology,
  - ▶ ...

# Optimization

People optimize in

- ▶ scheduling
  - ▶ transportation,
  - ▶ education,
  - ▶ ...
- ▶ investments
  - ▶ portfolio management,
  - ▶ utility maximization,
  - ▶ ...
- ▶ industrial design
  - ▶ aerodynamics,
  - ▶ electrical engineering,
  - ▶ ...
- ▶ sciences
  - ▶ molecular modeling,
  - ▶ computational systems biology,
  - ▶ ...
- ▶ machine learning

# Optimization Algorithms

## scipy.optimize.minimize

```
scipy.optimize.minimize(fun, x0, args=(), method=None, jac=None, hess=None,  
hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None)
```

**method** : *str or callable, optional*

Type of solver. Should be one of

- 'Nelder-Mead' ([see here](#))
- 'Powell' ([see here](#))
- 'CG' ([see here](#))
- 'BFGS' ([see here](#))
- 'Newton-CG' ([see here](#))
- 'L-BFGS-B' ([see here](#))



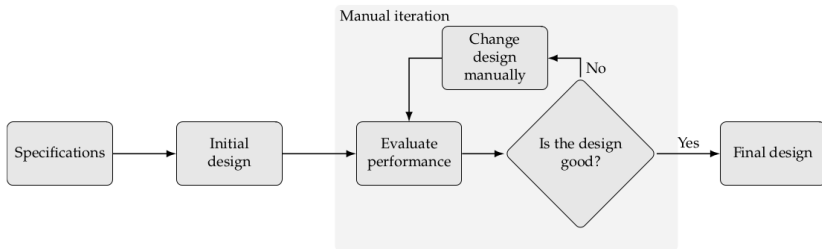
# Optimization Algorithms

## `sklearn.linear_model.LogisticRegression`

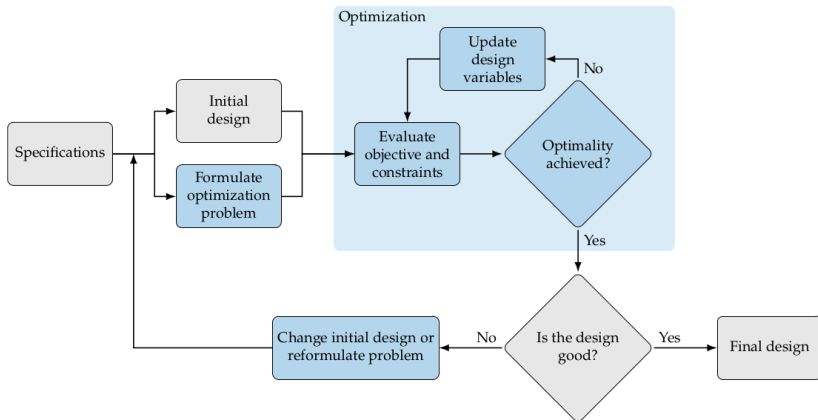
```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```

**solver** : {'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'}, default='lbfgs'  
Algorithm to use in the optimization problem. Default is 'lbfgs'. To choose a solver,

# Design Optimization Process



# Design Optimization Process



A bit naive example:

- ▶ Consider a company with several plants producing a single product but with different efficiency.
- ▶ The goal is to set the production of each plant so that demand for goods is satisfied, but overproduction is minimized.

A bit naive example:

- ▶ Consider a company with several plants producing a single product but with different efficiency.
- ▶ The goal is to set the production of each plant so that demand for goods is satisfied, but overproduction is minimized.
- ▶ **First try:** Model each plant's production and maximize the total production efficiency.

This would lead to a solution where only the most efficient plant will produce.

## A bit naive example:

- ▶ Consider a company with several plants producing a single product but with different efficiency.
- ▶ The goal is to set the production of each plant so that demand for goods is satisfied, but overproduction is minimized.
- ▶ **First try:** Model each plant's production and maximize the total production efficiency.

This would lead to a solution where only the most efficient plant will produce.

- ▶ However, after a certain level of demand, no single plant can satisfy the demand  $\Rightarrow$ , introducing constraints on the maximum production of the plants.

This would maximize production of the most efficient plant and then the second one, etc.

## A bit naive example:

- ▶ Consider a company with several plants producing a single product but with different efficiency.
- ▶ The goal is to set the production of each plant so that demand for goods is satisfied, but overproduction is minimized.
- ▶ **First try:** Model each plant's production and maximize the total production efficiency.

This would lead to a solution where only the most efficient plant will produce.

- ▶ However, after a certain level of demand, no single plant can satisfy the demand  $\Rightarrow$ , introducing constraints on the maximum production of the plants.

This would maximize production of the most efficient plant and then the second one, etc.

- ▶ Then you notice that all plant employees must work.
- ▶ Then you start solving transportation problems depending on the location of the plants.
- ▶ ...

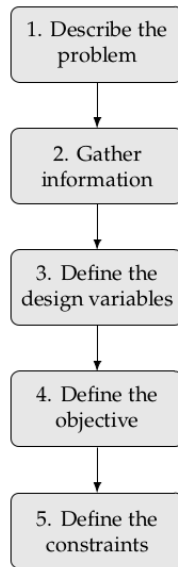
# Optimization Problem Formulation

## 1. Describe the problem

- ▶ Problem formulation is vital since the optimizer exploits any weaknesses in the model formulation.
- ▶ You might get the “right answer to the wrong question.”
- ▶ The problem description is typically informal at the beginning.

## 2. Gather information

- ▶ Identify possible inputs/outputs.
- ▶ Gather data and identify the analysis procedure.





# Optimization Problem Formulation

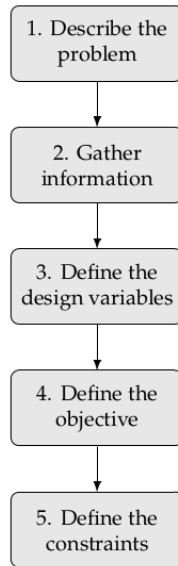
## 3. Define the design **variables**

- Identify the quantities that describe the system:

$$x \in \mathbb{R}^n$$

(i.e., certain characteristics of the system, such as position, investments, etc.)

- The variables are supposed to be independent; the optimizer must be free to choose the components of  $x$  independently.
- The choice of variables is typically not unique (e.g., a square can be described by its side or area).
- The variables may affect the functional form of the objective and constraints (e.g., linear vs non-linear).



# Optimization Problem Formulation

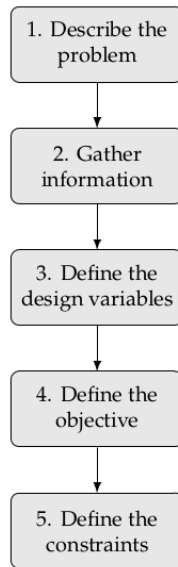
## 4. Define the **objective**

- ▶ The function determines if one design is better than another.
- ▶ Must be a scalar computable from the variables:

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

(e.g., profit, time, potential energy, etc.)

- ▶ The objective function is either maximized or minimized depending on the application.
- ▶ The choice is not always obvious: E.g., minimizing just the weight of a vehicle might result in a vehicle being too expensive to be manufactured.



# Optimization Problem Formulation

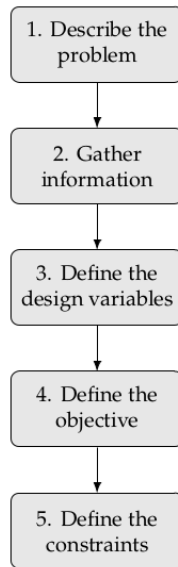
## 5. Define the **constraints**

- ▶ Prescribe allowed values of the variables.
- ▶ May have a general form

$$c(x) \leq 0 \text{ or } c(x) \geq 0 \text{ or } c(x) = 0$$

(e.g., time cannot be negative, bounded amount of money to invest)

Where  $c : \mathbb{R}^n \rightarrow \mathbb{R}$  is a function depending on the variables.



# Modelling and Optimization

The **Optimization Problem** consists of

- ▶ **variables**
- ▶ **objective**
- ▶ **constraints**

The above components constitute a **model**.

# Modelling and Optimization

The **Optimization Problem** consists of

- ▶ **variables**
- ▶ **objective**
- ▶ **constraints**

The above components constitute a **model**.

**Modelling** is concerned with model building, **optimization** with maximization/minimization of the objective for a given model.

We concentrate on the optimization part but keep in mind that it is intertwined with modeling.

# Modelling and Optimization

The **Optimization Problem** consists of

- ▶ **variables**
- ▶ **objective**
- ▶ **constraints**

The above components constitute a **model**.

**Modelling** is concerned with model building, **optimization** with maximization/minimization of the objective for a given model.

We concentrate on the optimization part but keep in mind that it is intertwined with modeling.

The **Optimization Problem (OP)**: Find settings of variables so that the objective is maximized/minimized while satisfying the constraints.

# Modelling and Optimization

The **Optimization Problem** consists of

- ▶ **variables**
- ▶ **objective**
- ▶ **constraints**

The above components constitute a **model**.

**Modelling** is concerned with model building, **optimization** with maximization/minimization of the objective for a given model.

We concentrate on the optimization part but keep in mind that it is intertwined with modeling.

The **Optimization Problem (OP)**: Find settings of variables so that the objective is maximized/minimized while satisfying the constraints.

An **Optimization Algorithm (OA)** solves the above problem and provides a **solution**, some setting of variables satisfying the constraints and minimizing/maximizing the objective.

# Optimization Problems



# Optimization Problem Formally

Denote by

$f : \mathbb{R}^n \rightarrow \mathbb{R}$  an *objective function*,

$x$  a vector of real *variables*,

$g_1, \dots, g_{n_g}$  *inequality constraint functions*  $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$ .

$h_1, \dots, h_{n_h}$  *equality constraint functions*  $h_j : \mathbb{R}^n \rightarrow \mathbb{R}$ .

# Optimization Problem Formally

Denote by

$f : \mathbb{R}^n \rightarrow \mathbb{R}$  an *objective function*,

$x$  a vector of real *variables*,

$g_1, \dots, g_{n_g}$  *inequality constraint functions*  $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$ .

$h_1, \dots, h_{n_h}$  *equality constraint functions*  $h_j : \mathbb{R}^n \rightarrow \mathbb{R}$ .

The optimization problem is to

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{by varying} & x \\ \text{subject to} & g_i(x) \leq 0 \quad i = 1, \dots, n_g \\ & h_j(x) = 0 \quad j = 1, \dots, n_h \end{array}$$

## Optimization Problem - Example

$$f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2$$

$$g_1(x_1, x_2) = x_1^2 - x_2$$

$$g_2(x_1, x_2) = x_1 + x_2 - 2$$

The optimization problem is

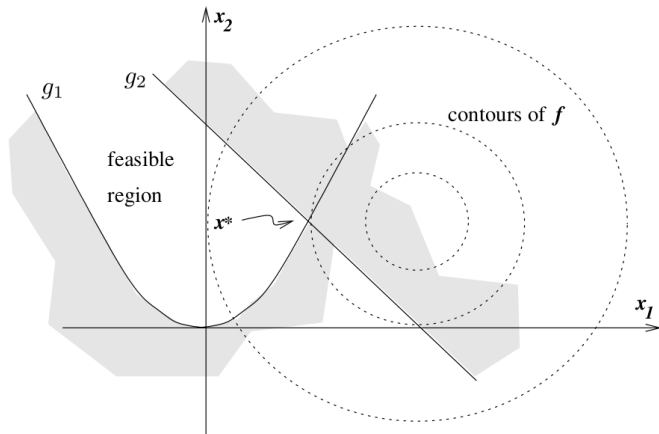
$$\text{minimize } (x_1 - 2)^2 + (x_2 - 1)^2 \quad \text{subject to } \begin{cases} x_1^2 - x_2 \leq 0, \\ x_1 + x_2 - 2 \leq 0. \end{cases}$$

## Optimization Problem - Example

$$f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2$$

$$g_1(x_1, x_2) = x_1^2 - x_2$$

$$g_2(x_1, x_2) = x_1 + x_2 - 2$$



A *contour* of  $f$  is defined, for some  $c \in \mathbb{R}$ , by  $\{x \in \mathbb{R}^n \mid f(x) = c\}$

# Constraints

Consider the constraints

$$g_i(x) \leq 0 \quad i = 1, \dots, n_g$$

$$h_j(x) = 0 \quad j = 1, \dots, n_h$$

# Constraints

Consider the constraints

$$g_i(x) \leq 0 \quad i = 1, \dots, n_g$$

$$h_j(x) = 0 \quad j = 1, \dots, n_h$$

Define the *feasibility region* by

$$\mathcal{F} = \{x \mid g_i(x) \leq 0, h_j(x) = 0, i = 1, \dots, n_g, j = 1, \dots, n_h\}$$

$x \in \mathcal{F}$  is *feasible*,  $x \notin \mathcal{F}$  is *infeasible*.

# Constraints

Consider the constraints

$$\begin{aligned} g_i(x) &\leq 0 & i = 1, \dots, n_g \\ h_j(x) &= 0 & j = 1, \dots, n_h \end{aligned}$$

Define the *feasibility region* by

$$\mathcal{F} = \{x \mid g_i(x) \leq 0, h_j(x) = 0, i = 1, \dots, n_g, j = 1, \dots, n_h\}$$

$x \in \mathcal{F}$  is *feasible*,  $x \notin \mathcal{F}$  is *infeasible*.

Note that constraints of the form  $g_i(x) \geq 0$  can be easily transformed to the inequality constraints  $-g_i(x) \leq 0$

# Constraints

Consider the constraints

$$\begin{aligned} g_i(x) &\leq 0 & i = 1, \dots, n_g \\ h_j(x) &= 0 & j = 1, \dots, n_h \end{aligned}$$

Define the *feasibility region* by

$$\mathcal{F} = \{x \mid g_i(x) \leq 0, h_j(x) = 0, i = 1, \dots, n_g, j = 1, \dots, n_h\}$$

$x \in \mathcal{F}$  is *feasible*,  $x \notin \mathcal{F}$  is *infeasible*.

Note that constraints of the form  $g_i(x) \geq 0$  can be easily transformed to the inequality constraints  $-g_i(x) \leq 0$

$x^* \in \mathcal{F}$  is now a *constrained minimizer* if

$$f(x^*) \leq f(x) \quad \text{for all } x \in \mathcal{F}$$



# Constraints

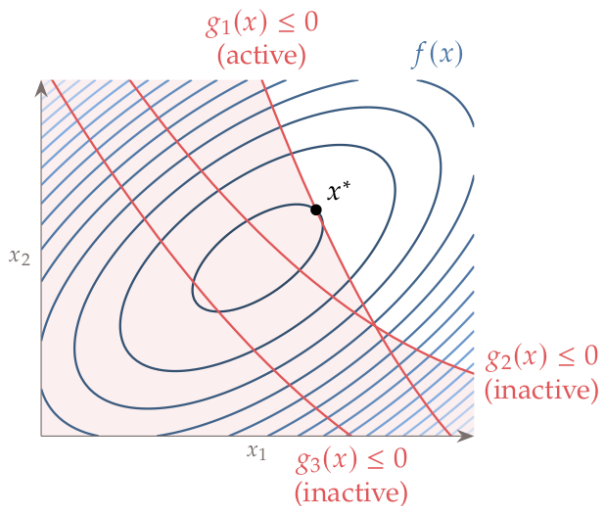
Inequality constraints  $g_i(x) \leq 0$  can be *active* or *inactive*.

*active* at  $x$

$$g_i(x) = 0$$

*inactive* at  $x$

$$g_i(x) < 0$$



## More Practical Example

The problem formulation:

- ▶ A company has two chemical factories  $F_1$  and  $F_2$ , and a dozen retail outlets  $R_1, \dots, R_{12}$ .
- ▶ Each  $F_i$  can produce (maximum of)  $a_i$  tons of a chemical each week.
- ▶ Each retail outlet  $R_j$  demands at least  $b_j$  tons.
- ▶ The cost of shipping one ton from  $F_i$  to  $R_j$  is  $c_{ij}$ .

## More Practical Example

The problem formulation:

- ▶ A company has two chemical factories  $F_1$  and  $F_2$ , and a dozen retail outlets  $R_1, \dots, R_{12}$ .
- ▶ Each  $F_i$  can produce (maximum of)  $a_i$  tons of a chemical each week.
- ▶ Each retail outlet  $R_j$  demands at least  $b_j$  tons.
- ▶ The cost of shipping one ton from  $F_i$  to  $R_j$  is  $c_{ij}$ .

**The problem:** Determine how much each factory should ship to each outlet to satisfy the requirements and minimize cost.

## More Practical Example

Variables:  $x_{ij}$  for  $i = 1, 2$  and  $j = 1, \dots, 12$ . Each  $x_{ij}$  (intuitively) corresponds to tons shipped from  $F_i$  to  $R_j$ .

The objective:

$$\min \sum_{ij} c_{ij} x_{ij}$$

## More Practical Example

Variables:  $x_{ij}$  for  $i = 1, 2$  and  $j = 1, \dots, 12$ . Each  $x_{ij}$  (intuitively) corresponds to tons shipped from  $F_i$  to  $R_j$ .

The objective:

$$\min \sum_{ij} c_{ij} x_{ij}$$

subject to

$$\sum_{j=1}^{12} x_{ij} \leq a_i, \quad i = 1, 2$$

$$\sum_{i=1}^2 x_{ij} \geq b_j, \quad j = 1, \dots, 12,$$

$$x_{ij} \geq 0, \quad i = 1, 2, \quad j = 1, \dots, 12.$$

## More Practical Example

Variables:  $x_{ij}$  for  $i = 1, 2$  and  $j = 1, \dots, 12$ . Each  $x_{ij}$  (intuitively) corresponds to tons shipped from  $F_i$  to  $R_j$ .

The objective:

$$\min \sum_{ij} c_{ij} x_{ij}$$

subject to

$$\sum_{j=1}^{12} x_{ij} \leq a_i, \quad i = 1, 2$$

$$\sum_{i=1}^2 x_{ij} \geq b_j, \quad j = 1, \dots, 12,$$

$$x_{ij} \geq 0, \quad i = 1, 2, \quad j = 1, \dots, 12.$$

The above is *linear programming* problem since both the objective and constraint functions are linear.

# Discrete Optimization

In our original optimization problem definition, we consider real (continuous) variables.

Sometimes, we need to assume discrete values. For example, in the previous example, the factories may produce tractors. In such a case, it does not make sense to produce 4.6 tractors.

# Discrete Optimization

In our original optimization problem definition, we consider real (continuous) variables.

Sometimes, we need to assume discrete values. For example, in the previous example, the factories may produce tractors. In such a case, it does not make sense to produce 4.6 tractors.

Usually, an *integer* constraint is added, such as

$$x_i \in \mathbb{Z}$$

It constrains  $x_i$  only to integer values. This leads to so-called *integer programming*.

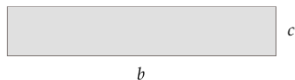
*Discrete optimization* problems have discrete and finite variables.



## Wing Design Example

Our goal is to design the wing shape of an aircraft.

Assume a rectangular wing.

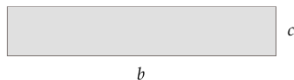


The parameters are called *span*  $b$  and *chord*  $c$ .

# Wing Design Example

Our goal is to design the wing shape of an aircraft.

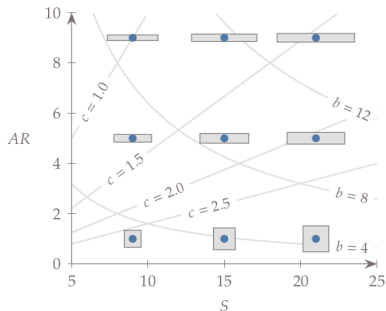
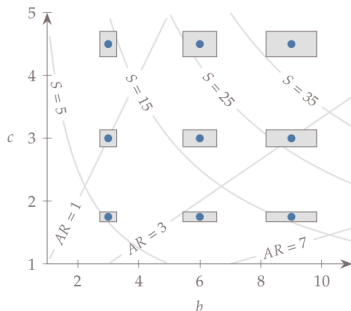
Assume a rectangular wing.



The parameters are called *span*  $b$  and *chord*  $c$ .

However, two other variables are often used in aircraft design: Wing area  $S$  and wing aspect ratio  $AR$ . It holds that

$$S = bc \quad AR = b^2/S$$



# Wing Design Example

What exactly are the objectives and constraints?

# Wing Design Example

What exactly are the objectives and constraints?

Our objective function is the power required to keep level flight:

$$f(b, c) = \frac{Dv}{\eta}$$

Here,

- ▶  $D$  is the drag

That is the aerodynamic force that opposes an aircraft's motion through the air.

- ▶  $\eta$  is the propulsive efficiency

That is the efficiency with which the energy contained in a vehicle's fuel is converted into kinetic energy of the vehicle.

- ▶  $v$  is the lift velocity

That is the velocity needed to lift the aircraft, which depends on its weight.

## Wing Design Example

For illustration, let us look at the lift velocity  $v$ .

## Wing Design Example

For illustration, let us look at the lift velocity  $v$ .

In level flight, the aircraft must generate enough lift  $L$  to equal its weight  $W$ , that is  $L = W$ .

## Wing Design Example

For illustration, let us look at the lift velocity  $v$ .

In level flight, the aircraft must generate enough lift  $L$  to equal its weight  $W$ , that is  $L = W$ .

The weight partially depends on the wing area:

$$W = W_0 + W_S S$$

Here  $S = bc$  is the wing area, and  $W_0$  is the payload weight.

## Wing Design Example

For illustration, let us look at the lift velocity  $v$ .

In level flight, the aircraft must generate enough lift  $L$  to equal its weight  $W$ , that is  $L = W$ .

The weight partially depends on the wing area:

$$W = W_0 + W_S S$$

Here  $S = bc$  is the wing area, and  $W_0$  is the payload weight.

The lift can be approximated using the following formula.

$$L = q \cdot C_L \cdot S$$

Where  $q = \frac{1}{2} \varrho v^2$  is the fluid dynamic pressure, here  $\varrho$  is the air density,  $C_L$  is a lift coefficient (depending on the wing shape).



## Wing Design Example

For illustration, let us look at the lift velocity  $v$ .

In level flight, the aircraft must generate enough lift  $L$  to equal its weight  $W$ , that is  $L = W$ .

The weight partially depends on the wing area:

$$W = W_0 + W_S S$$

Here  $S = bc$  is the wing area, and  $W_0$  is the payload weight.

The lift can be approximated using the following formula.

$$L = q \cdot C_L \cdot S$$

Where  $q = \frac{1}{2}\rho v^2$  is the fluid dynamic pressure, here  $\rho$  is the air density,  $C_L$  is a lift coefficient (depending on the wing shape).

Thus, we may obtain the lift velocity as

$$v = \sqrt{2W/\rho C_L S} = \sqrt{2(W_0 + W_S bc)/\rho C_L bc}$$

Similarly, various physics-based arguments provide approximations of the drag  $D$  and the propulsion efficiency  $\eta$ .

## Wing Design Example

The drag  $D = D_i + D_f$  is the sum of the induced and viscous drag.

## Wing Design Example

The drag  $D = D_i + D_f$  is the sum of the induced and viscous drag.

The induced drag can be approximated by

$$D_i = W^2 / q \pi b^2 e$$

Here,  $e$  is the Oswald efficiency factor, a correction factor that represents the change in drag with the lift of a wing, as compared with an ideal wing having the same aspect ratio.

## Wing Design Example

The drag  $D = D_i + D_f$  is the sum of the induced and viscous drag.

The induced drag can be approximated by

$$D_i = W^2 / q \pi b^2 e$$

Here,  $e$  is the Oswald efficiency factor, a correction factor that represents the change in drag with the lift of a wing, as compared with an ideal wing having the same aspect ratio.

The viscous drag can be approximated by

$$D_f = k C_f q S$$

Here,  $k$  is the form factor (accounts for the pressure drag), and  $C_f$  is the skin friction coefficient that can be approximated by

$$C_f = 0.074 / Re^{0.2}$$

Where  $Re$  is the Reynolds number that somewhat characterizes air flow patterns around the wing and is defined as follows:

$$Re = \rho v c / \mu$$

Here  $\mu$  is the air dynamic viscosity.

# Wing Design Example

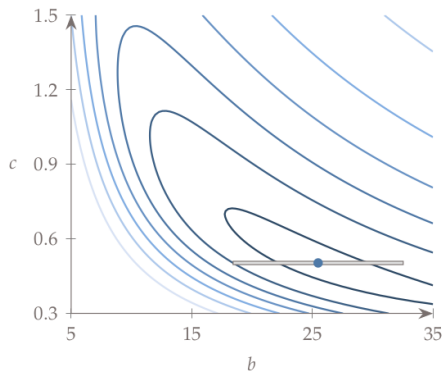
The propulsion efficiency  $\eta$  can be roughly approximated by the Gaussian efficiency curve.

$$\eta = \eta_{\max} \exp\left(\frac{-(v - \bar{v})^2}{2\sigma^2}\right)$$

Here,  $\bar{v}$  is the peak propulsive efficiency velocity, and  $\sigma$  is the std of the efficiency function.

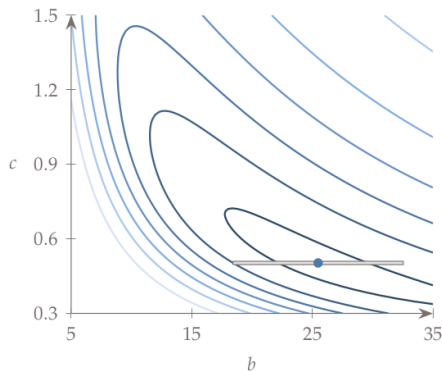
# Wing Design Example

The objective function contours:



# Wing Design Example

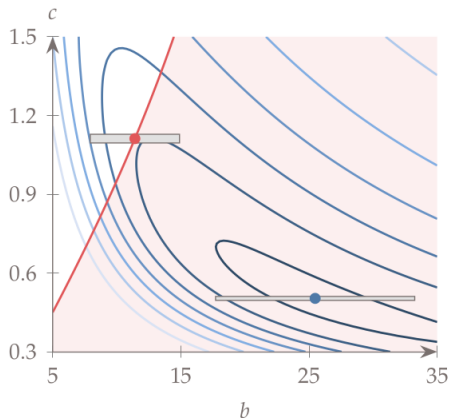
The objective function contours:



The engineers would refuse the solution: The aspect ratio is much higher than typically seen in airplanes. It adversely affects the structural strength. Add constraints!

# Wing Design Example

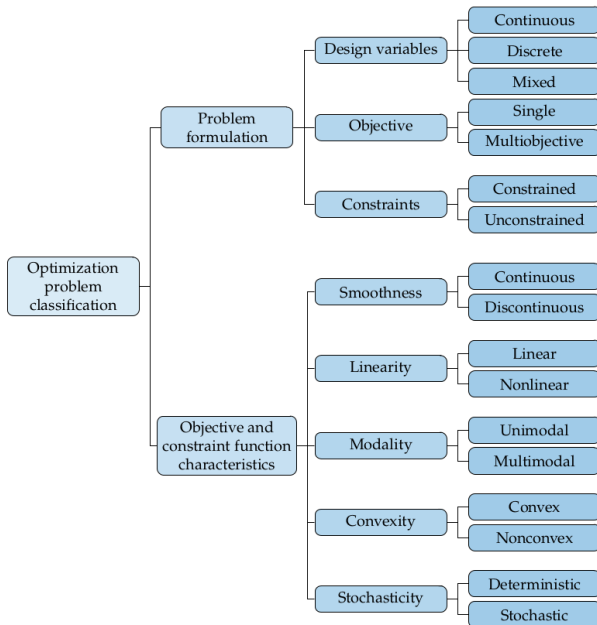
Added a constraint on bending stress at the root of the wing:



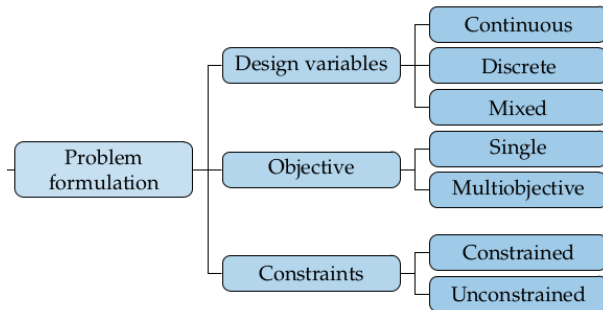
It looks like a reasonable wing ...



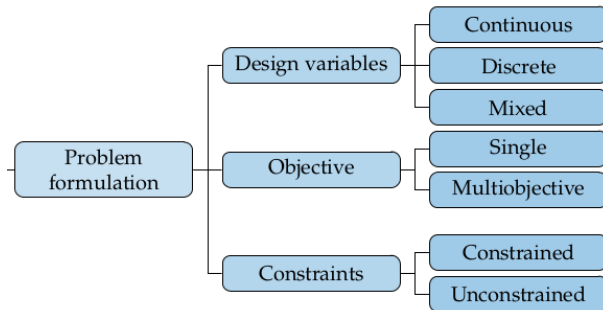
# Optimization Problem Classification



# Optimization Problem Classification

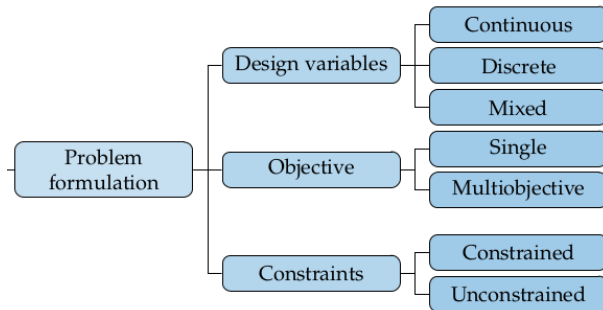


# Optimization Problem Classification



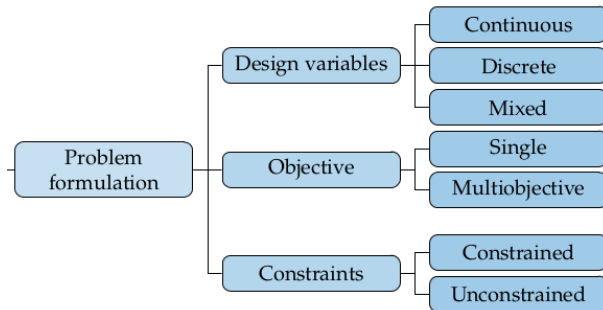
- *Continuous* allows only  $x_i \in \mathbb{R}$ , *discrete* allows only  $x_i \in \mathbb{Z}$ , mixed allows variables of both kinds.

# Optimization Problem Classification



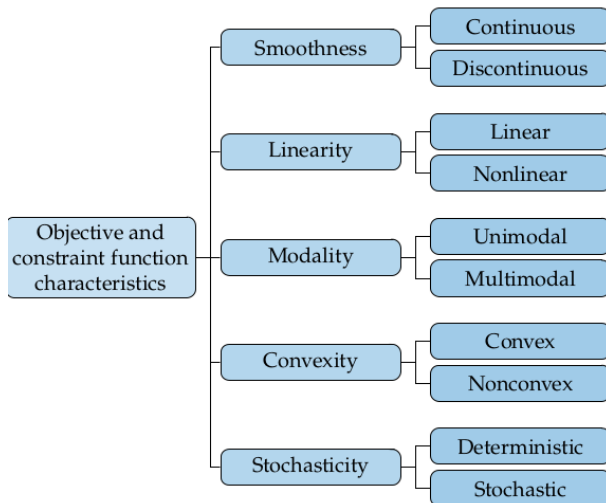
- ▶ *Continuous* allows only  $x_i \in \mathbb{R}$ , *discrete* allows only  $x_i \in \mathbb{Z}$ , mixed allows variables of both kinds.
- ▶ *Single-objective*:  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , *Multi-objective*:  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$

# Optimization Problem Classification



- ▶ *Continuous* allows only  $x_i \in \mathbb{R}$ , *discrete* allows only  $x_i \in \mathbb{Z}$ , mixed allows variables of both kinds.
- ▶ *Single-objective*:  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , *Multi-objective*:  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$
- ▶ *Unconstrained*: No constraints, just the objective function.

# Optimization Problem Classification

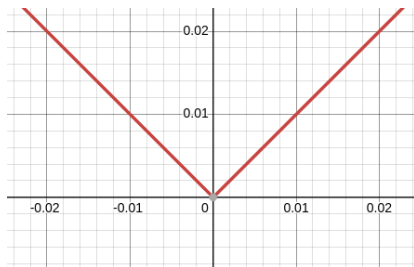


# Smoothness

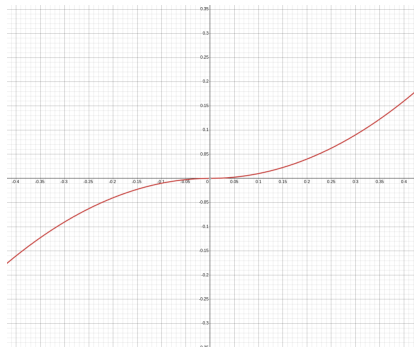
We consider various classes of problems depending on the smoothness properties of the objective/constraint functions:

- ▶  $C^0$ : Continuous function Continuity allows us to estimate value in small neighborhoods.  
Discontinuous functions exist.
- ▶  $C^1$ : Continuous first derivatives  
The derivatives give information on the slope. If continuous, it changes smoothly, allowing us to estimate the slope locally.  
Nondifferentiable continuous functions and differentiable functions with discontinuous derivatives exist.
- ▶  $C^2$ : Continuous second derivatives The second derivatives inform about curvature.  
Continuously differentiable functions without second derivatives and twice differentiable functions with discontinuous second derivatives exist.

$f(x) = |x|$  is continuous,  $f$  is not differentiable at 0

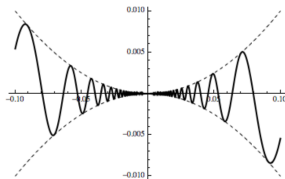


$f(x) = x|x|$  is differentiable on  $\mathbb{R}$ ,  $f'$  has no second derivative at 0

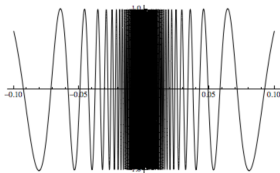




$$f(x) = \begin{cases} x^2 \sin(1/x) & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$$



$$f'(x) = \begin{cases} 2x \sin(1/x) - \cos(1/x), & x \neq 0 \\ 0, & x = 0 \end{cases}$$



$f$  is differentiable on  $\mathbb{R}$ ,  $f'$  is not continuous at 0

$$f(x) = \begin{cases} x^4 \sin(1/x) & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$$

$f$  is differentiable on  $\mathbb{R}$ ,

$$f'(x) = \begin{cases} 4x^3 \sin(1/x) - x^2 \cos(1/x), & x \neq 0 \\ 0, & x = 0 \end{cases}$$

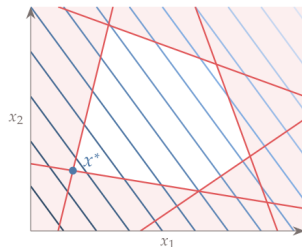
$f'$  is differentiable on  $\mathbb{R}$ ,

$$f''(x) = \begin{cases} 12x^2 \sin(1/x) - 6x \cos(1/x) - \sin(1/x), & x \neq 0 \\ 0, & x = 0 \end{cases}$$

Clearly,  $f''$  does not have a limit at 0 as  $\sin(1/x)$  oscillates between  $-1$  and  $1$  and thus **is not continuous**.

# Linearity

Linear programming: Both the objective and the constraints are linear.



It is possible to solve precisely, efficiently, and in rational numbers (see the linear programming later).

# Multimodality

Denote by  $\mathcal{F}$  the feasibility set.

$x^*$  is a (weak) *local minimiser* if there is  $\varepsilon > 0$  such that

$$f(x^*) \leq f(x) \text{ for all } x \in \mathcal{F} \text{ satisfying } \|x^* - x\| \leq \varepsilon$$

# Multimodality

Denote by  $\mathcal{F}$  the feasibility set.

$x^*$  is a (weak) *local minimiser* if there is  $\varepsilon > 0$  such that

$$f(x^*) \leq f(x) \text{ for all } x \in \mathcal{F} \text{ satisfying } \|x^* - x\| \leq \varepsilon$$

$x^*$  is a (weak) *global minimiser* if

$$f(x^*) \leq f(x) \text{ for all } x \in \mathcal{F}$$

# Multimodality

Denote by  $\mathcal{F}$  the feasibility set.

$x^*$  is a (weak) *local minimiser* if there is  $\varepsilon > 0$  such that

$$f(x^*) \leq f(x) \text{ for all } x \in \mathcal{F} \text{ satisfying } \|x^* - x\| \leq \varepsilon$$

$x^*$  is a (weak) *global minimiser* if

$$f(x^*) \leq f(x) \text{ for all } x \in \mathcal{F}$$

Global/local minimiser is *strict* if the inequality is strict.

# Multimodality

Denote by  $\mathcal{F}$  the feasibility set.

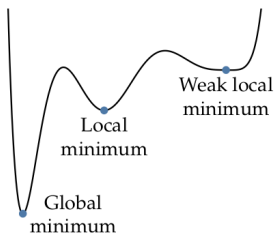
$x^*$  is a (weak) *local minimiser* if there is  $\varepsilon > 0$  such that

$$f(x^*) \leq f(x) \text{ for all } x \in \mathcal{F} \text{ satisfying } \|x^* - x\| \leq \varepsilon$$

$x^*$  is a (weak) *global minimiser* if

$$f(x^*) \leq f(x) \text{ for all } x \in \mathcal{F}$$

Global/local minimiser is *strict* if the inequality is strict.



*Unimodal* functions have a single global minimiser in  $\mathcal{F}$ ,  
*multimodal* have multiple local minimisers in  $\mathcal{F}$ .

# Convexity

$S \subseteq \mathbb{R}^n$  is a *convex set* if the straight line segment connecting any two points in  $S$  lies entirely inside  $S$ . Formally, for any two points  $x \in S$  and  $y \in S$ , we have  $\alpha x + (1 - \alpha)y \in S$  for all  $\alpha \in [0, 1]$



## Convexity

$S \subseteq \mathbb{R}^n$  is a *convex set* if the straight line segment connecting any two points in  $S$  lies entirely inside  $S$ . Formally, for any two points  $x \in S$  and  $y \in S$ , we have  $\alpha x + (1 - \alpha)y \in S$  for all  $\alpha \in [0, 1]$

$f$  is a *convex function* if its domain is a convex set and if for any two points  $x$  and  $y$  in this domain, the graph of  $f$  lies below the straight line connecting  $(x, f(x))$  to  $(y, f(y))$  in the space  $\mathbb{R}^{n+1}$ . That is, we have

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y), \quad \text{for all } \alpha \in (0, 1).$$

# Convexity

$S \subseteq \mathbb{R}^n$  is a **convex set** if the straight line segment connecting any two points in  $S$  lies entirely inside  $S$ . Formally, for any two points  $x \in S$  and  $y \in S$ , we have  $\alpha x + (1 - \alpha)y \in S$  for all  $\alpha \in [0, 1]$

$f$  is a **convex function** if its domain is a convex set and if for any two points  $x$  and  $y$  in this domain, the graph of  $f$  lies below the straight line connecting  $(x, f(x))$  to  $(y, f(y))$  in the space  $\mathbb{R}^{n+1}$ . That is, we have

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y), \quad \text{for all } \alpha \in (0, 1).$$

A **standard form convex optimization** assumes

- ▶ convex objective  $f$  and convex inequality constraint functions  $g_i$
- ▶ affine equality constraint functions  $h_j$

## Implications:

- ▶ Every local minimum is a global minimum.
- ▶ If the above inequality is strict for all  $x \neq y$ , then there is a unique minimum.

# Stochasticity

Sometimes, the parameters of a model cannot be specified with certainty.

For example, in the transportation model, customer demand cannot be predicted precisely in practice.

However, such parameters may often be statistically estimated and modeled using an appropriate probability distribution.

# Stochasticity

Sometimes, the parameters of a model cannot be specified with certainty.

For example, in the transportation model, customer demand cannot be predicted precisely in practice.

However, such parameters may often be statistically estimated and modeled using an appropriate probability distribution.

*Stochastic optimization* problem is to minimize/maximize the expectation of a statistic parametrized with the variables  $x$ :

Find  $x$  maximizing  $\mathbb{E}f(x; W)$

Here,  $W$  is a vector of random variables, and the expectation is taken using the probability distribution of these variables.

In this course, we stick with *deterministic optimization*.

# Optimization Algorithms

# Optimization Algorithm

An *optimization algorithm* solves the optimization problem, i.e., searches for  $x^*$ , which (in some sense) minimizes the objective  $f$  and satisfies the constraints.

Typically, the algorithm computes a set of candidate solutions  $x_0, x_1, \dots$  and then identifies one resembling a solution.

# Optimization Algorithm

An *optimization algorithm* solves the optimization problem, i.e., searches for  $x^*$ , which (in some sense) minimizes the objective  $f$  and satisfies the constraints.

Typically, the algorithm computes a set of candidate solutions  $x_0, x_1, \dots$  and then identifies one resembling a solution.

The problem is to

- ▶ compute the candidate solutions,  
Complexity of the objective function, difficulties in selection of the candidates, etc.
- ▶ Select the one closest to a minimum.  
It is Hard to decide whether a given point is a minimum (even a local one). Example: Neural networks training.

# Optimization Algorithm Properties

Typically, we are concerned with the following issues:



# Optimization Algorithm Properties

Typically, we are concerned with the following issues:

- ▶ *Robustness*: OA should perform well on various problems in their class for all reasonable choices of the initial variables.

# Optimization Algorithm Properties

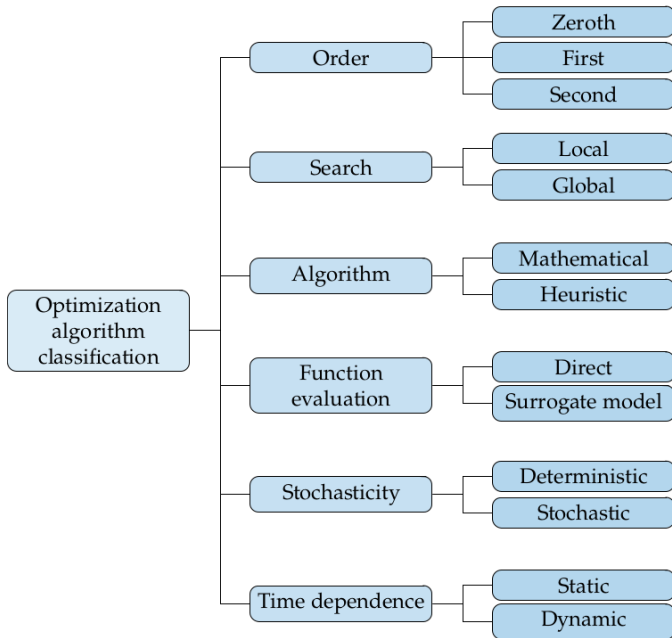
Typically, we are concerned with the following issues:

- ▶ *Robustness*: OA should perform well on various problems in their class for all reasonable choices of the initial variables.
- ▶ *Efficiency*: OA should not require too much computer time or storage.

# Optimization Algorithm Properties

Typically, we are concerned with the following issues:

- ▶ *Robustness*: OA should perform well on various problems in their class for all reasonable choices of the initial variables.
- ▶ *Efficiency*: OA should not require too much computer time or storage.
- ▶ *Accuracy*: OA should be able to identify a solution with precision without being overly sensitive to
  - ▶ errors in the data/model
  - ▶ the arithmetic rounding errors



# Order and Search

## Order

- ▶ Zeroth = *gradient-free*: no info about derivatives is used
- ▶ First = *gradient-based*: use info about first derivatives (e.g., gradient descent)
- ▶ Second = use info about first and second derivatives (e.g., Newton's method)

# Order and Search

## Order

- ▶ Zeroth = *gradient-free*: no info about derivatives is used
- ▶ First = *gradient-based*: use info about first derivatives (e.g., gradient descent)
- ▶ Second = use info about first and second derivatives (e.g., Newton's method)

## Search

- ▶ *Local search* = start at a point and search for a solution by successively updating the current solution (e.g., gradient descent)
- ▶ *Global search* tries to span the whole space (e.g., grid search)

# Mathematical vs Heuristic

For some algorithms and under specific assumptions imposed on the optimization problem, we can do the following:

- ▶ Prove that the algorithm converges to an optimum/minimum.

# Mathematical vs Heuristic

For some algorithms and under specific assumptions imposed on the optimization problem, we can do the following:

- ▶ Prove that the algorithm converges to an optimum/minimum.
- ▶ Determine the rate of convergence.



# Mathematical vs Heuristic

For some algorithms and under specific assumptions imposed on the optimization problem, we can do the following:

- ▶ Prove that the algorithm converges to an optimum/minimum.
- ▶ Determine the rate of convergence.
- ▶ Decide whether we are at (or close to) an optimum/minimum.

# Mathematical vs Heuristic

For some algorithms and under specific assumptions imposed on the optimization problem, we can do the following:

- ▶ Prove that the algorithm converges to an optimum/minimum.
- ▶ Determine the rate of convergence.
- ▶ Decide whether we are at (or close to) an optimum/minimum.

For example, for linear optimization problems, the simplex algorithm converges to a minimum (or says that there is no minimum) in, at most, exponentially many steps, and we may efficiently decide whether we have reached a minimum.

# Mathematical vs Heuristic

For some algorithms and under specific assumptions imposed on the optimization problem, we can do the following:

- ▶ Prove that the algorithm converges to an optimum/minimum.
- ▶ Determine the rate of convergence.
- ▶ Decide whether we are at (or close to) an optimum/minimum.

For example, for linear optimization problems, the simplex algorithm converges to a minimum (or says that there is no minimum) in, at most, exponentially many steps, and we may efficiently decide whether we have reached a minimum.

We may prove only some or none of the properties for some algorithms.

There are (almost) infinitely many heuristic algorithms without provable convergence, often motivated by the behaviors of various animals.

# Deterministic vs Stochastic and Static vs Dynamic

*Stochastic optimization* is based on a random selection of candidate solutions.

Evolutionary algorithms contain some randomness (e.g., in the form of random mutations).

Also, various variants of the gradient-based methods are often randomized (e.g., variants of the stochastic gradient descent).

# Deterministic vs Stochastic and Static vs Dynamic

*Stochastic optimization* is based on a random selection of candidate solutions.

Evolutionary algorithms contain some randomness (e.g., in the form of random mutations).

Also, various variants of the gradient-based methods are often randomized (e.g., variants of the stochastic gradient descent).

---

In this course, we stick to *static* optimization problems where we solve the optimization problem only once.

In contrast, the *dynamic* optimization, a sequence of (usually) dependent optimization problems are solved sequentially.

For example, consider driving a car where the driver must react optimally to changing situations several times per second.

Dynamic optimization problems are usually defined using a kind of (Markov) decision process.

# Summary

The course consists of the following main parts:

- ▶ Unconstrained optimization
  - ▶ Non-linear objectives, (twice) differentiable
  - ▶ Second-order methods (quasi-Newton)
- ▶ Constrained optimization
  - ▶ Non-linear objectives and constraints, (twice) differentiable
  - ▶ Lagrange multipliers, Newton-Lagrange method
  - ▶ Quadratic programming (a little bit)
- ▶ Linear programming
  - ▶ Linear objectives and constraints
  - ▶ Simplex algorithm deep dive (including the degenerate case)
- ▶ Integer linear programming
  - ▶ Linear objectives and mixed integer linear constraints
  - ▶ Branch-and-bound, Gomory cuts algorithms
- ▶ A little bit on non-differentiable algorithms.

You will need to understand: Calculus in  $\mathbb{R}^n$  (gradient, Hessian) and linear algebra in  $\mathbb{R}^n$  (vectors, matrices, geometry)

# Single-variable Objectives

# Unconstrained Single Variable Optimization Problem

An objective function  $f : \mathbb{R} \rightarrow \mathbb{R}$

A variable  $x$

Find  $x^*$  such that

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



# Unconstrained Single Variable Optimization Problem

An objective function  $f : \mathbb{R} \rightarrow \mathbb{R}$

A variable  $x$

Find  $x^*$  such that

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

We consider

- ▶  $f$  continuously differentiable
- ▶  $f$  twice continuously differentiable

Present the following methods:

- ▶ Gradient descent
- ▶ Newton's method
- ▶ Secant method

# Gradient Based Methods

An objective function  $f : \mathbb{R} \rightarrow \mathbb{R}$

A variable  $x \in \mathbb{R}$

Find  $x^*$  such that

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

# Gradient Based Methods

An objective function  $f : \mathbb{R} \rightarrow \mathbb{R}$

A variable  $x \in \mathbb{R}$

Find  $x^*$  such that

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

Assume that

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad \text{for } x \in \mathbb{R}$$

is continuous on  $\mathbb{R}$ .

Denote by  $\mathcal{C}^1$  the set of all continuously differentiable functions.

# Gradient Descent in Single Variable

Gradient descent algorithm for finding a local minimum of a function  $f$ , using a variable step length.

**Input:** Function  $f$  with first derivative  $f'$ , initial point  $x_0$ , initial step length  $\alpha_0 > 0$ , tolerance  $\epsilon > 0$

**Output:** A point  $x$  that approximately minimizes  $f(x)$

- 1: Set  $k \leftarrow 0$
- 2: **while**  $|f'(x_k)| > \epsilon$  **do**
- 3:     Calculate the derivative:  $y' \leftarrow f'(x_k)$
- 4:     Update  $x_{k+1} \leftarrow x_k - \alpha_k \cdot y'$
- 5:     Update step length  $\alpha_k$  to  $\alpha_{k+1}$  based on a certain strategy
- 6:     Increment  $k$
- 7: **end while**
- 8: **return**  $x_k$

# Convergence of Single Variable Gradient Descent

## Theorem 1

Assume that  $f$  is

- ▶ differentiable, i.e., that  $f'$  exists,
- ▶ bounded below, i.e., there is  $B \in \mathbb{R}$  such that  $f(x) \geq B$  for all  $x \in \mathbb{R}$ ,
- ▶  $L$ -smooth, i.e., there is  $L > 0$  such that  $|f'(x) - f'(x')| \leq L|x - x'|$  for all  $x, x' \in \mathbb{R}$ .

Consider a sequence  $x_0, x_1, \dots$  computed by the gradient descent algorithm for  $f$ . Assume a constant step length  $\alpha \leq \frac{1}{L}$ .

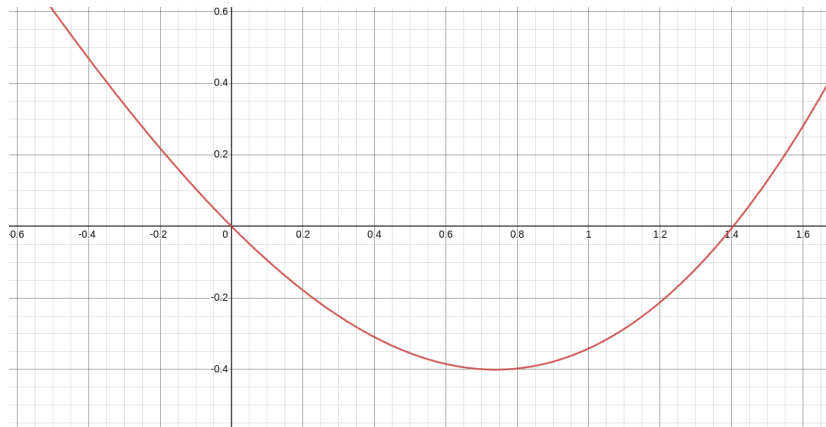
Then  $\lim_{k \rightarrow \infty} |f'(x_k)| = 0$  and, moreover,

$$\min_{0 \leq t < T} |f'(x_t)| \leq \sqrt{\frac{2L(f(x_0) - B)}{T}}$$

## Example

Consider the following objective function  $f$

$$f(x) = \frac{1}{2}x^2 - \sin x$$



## Example

Consider the objective function  $f$

$$f(x) = \frac{1}{2}x^2 - \sin x$$

Assume  $x_0 = 0.5$ , and that the required accuracy is  $\epsilon = 10^{-4}$ , i.e., we stop when  $|x_{k+1} - x_k| < \epsilon$ .

Consider the step length  $\alpha = 1$ .

## Example

Consider the objective function  $f$

$$f(x) = \frac{1}{2}x^2 - \sin x$$

Assume  $x_0 = 0.5$ , and that the required accuracy is  $\epsilon = 10^{-4}$ , i.e., we stop when  $|x_{k+1} - x_k| < \epsilon$ .

Consider the step length  $\alpha = 1$ .

We compute

$$f'(x) = x - \cos x.$$

Then,

$$\begin{aligned}x_1 &= 0.5 - (0.5 - \cos 0.5) \\&= 0.5 - (-0.37758) \\&= 0.87758\end{aligned}$$



## Example

Continuing in the same way:

$$x_1 = 0.87758$$

$$x_2 = 0.63901$$

$$x_3 = 0.80269$$

$$x_4 = 0.69478$$

$$x_5 = 0.76820$$

$$x_6 = 0.71917$$

$$x_7 = 0.75236$$

$$x_8 = 0.73008$$

$$x_9 = 0.74512$$

$$x_{10} = 0.73501$$

$$x_{11} = 0.74183$$

$$x_{12} = 0.73724$$

$$x_{13} = 0.74033$$

$$x_{14} = 0.73825$$

$$x_{15} = 0.73965$$

$$x_{16} = 0.73870$$

$$x_{17} = 0.73934$$

$$x_{18} = 0.73891$$

$$x_{19} = 0.73920$$

$$x_{20} = 0.73901$$

$$x_{21} = 0.73914$$

$$x_{22} = 0.73905$$

Note that  $|x_{22} - x_{21}| < 10^{-4}$ .

## Example

What if we consider the step length  $1/k$ ? Then

$$x_1 = 0.50000$$

$$x_2 = 0.87758$$

$$x_3 = 0.75830$$

$$x_4 = 0.74753$$

$$x_5 = 0.74399$$

$$x_6 = 0.74235$$

$$x_7 = 0.74144$$

$$x_8 = 0.74087$$

$$x_9 = 0.74050$$

$$x_{10} = 0.74024$$

$$x_{11} = 0.74004$$

$$x_{12} = 0.73990$$

$$x_{13} = 0.73978$$

$$x_{14} = 0.73969$$

Note that  $|x_{14} - x_{13}| < 10^{-4}$  but  $x_{14}$  is far from the solution which is 0.7390....

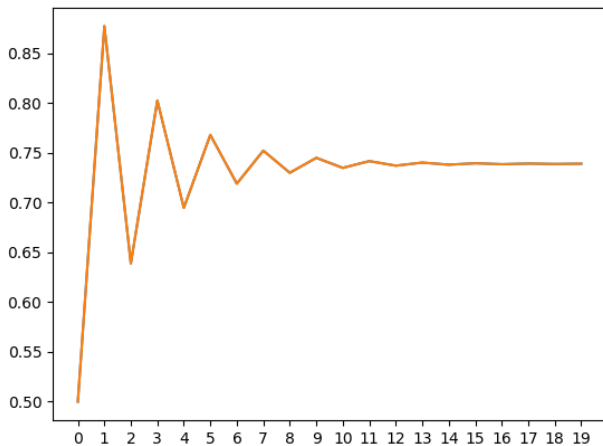
## Example

What if we consider the step length  $1/k$ ? Then

$x_1 = 0.50000$	$x_{115} = 0.739100605$
$x_2 = 0.87758$	$x_{116} = 0.739100379$
$x_3 = 0.75830$	$x_{117} = 0.739100159$
$x_4 = 0.74753$	$x_{118} = 0.739099944$
$x_5 = 0.74399$	$x_{119} = 0.739099734$
$x_6 = 0.74235$	$x_{120} = 0.739099529$
$x_7 = 0.74144$	$x_{121} = 0.739099328$
$x_8 = 0.74087$	$x_{122} = 0.739099132$
$x_9 = 0.74050$	$x_{123} = 0.739098940$
$x_{10} = 0.74024$	$x_{124} = 0.739098752$
$x_{11} = 0.74004$	$x_{125} = 0.739098568$
$x_{12} = 0.73990$	$x_{126} = 0.739098388$
$x_{13} = 0.73978$	$x_{127} = 0.739098212$
$x_{14} = 0.73969$	$x_{128} = 0.739098040$
$\dots$	

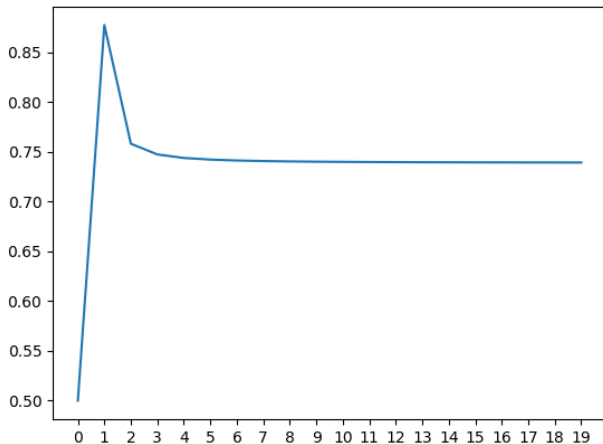
## Example

Gradient descent with the step length = 1.0:



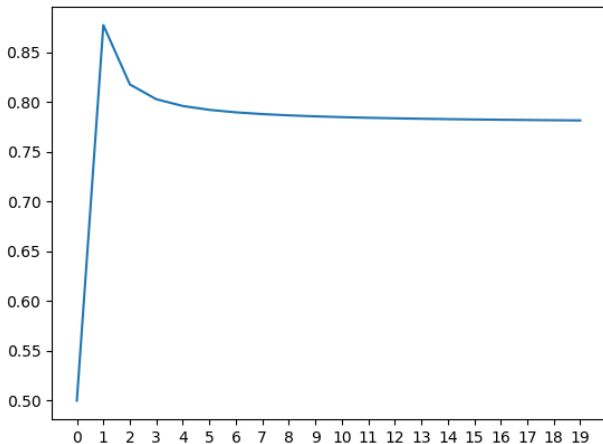
## Example

Gradient descent with the step length  $= 1/k$ :



## Example

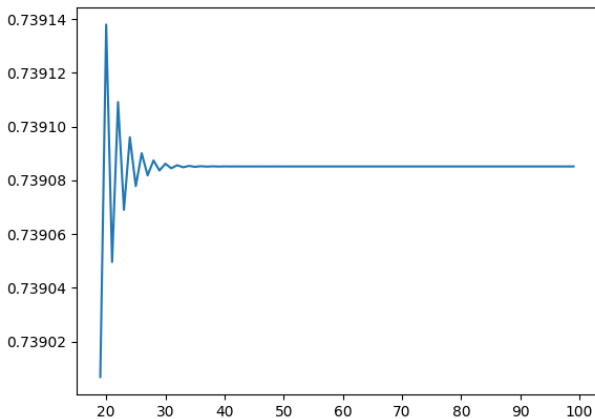
Gradient descent with the step length  $= 1/k^2$ :



It does not seem to converge to the same number as the previous step lengths.

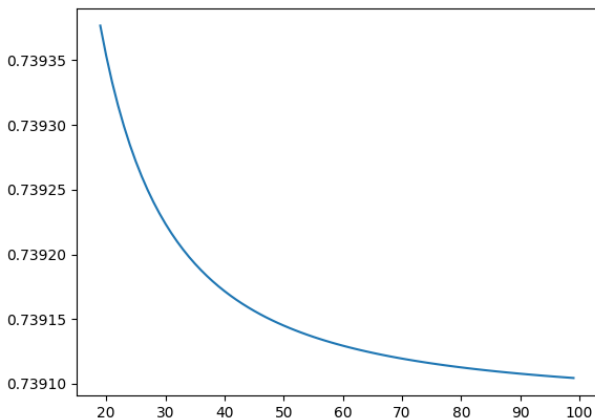
## Example

Gradient descent with the step length = 1.0:



## Example

Gradient descent with the step length  $= 1/k$ :





# Properties of Gradient Descent

- ▶ The objective must be differentiable, however:
  - ▶ Can be extended to functions with few non-linearities by considering differentiable parts or sub-gradients.
  - ▶ There are methods for differentiable approximation of non-differentiable functions.

# Properties of Gradient Descent

- ▶ The objective must be differentiable, however:
  - ▶ Can be extended to functions with few non-linearities by considering differentiable parts or sub-gradients.
  - ▶ There are methods for differentiable approximation of non-differentiable functions.
- ▶ GD is sensitive to the initial point: Converges to a local minimum for a small step length (typically) to the closest one.

# Properties of Gradient Descent

- ▶ The objective must be differentiable, however:
  - ▶ Can be extended to functions with few non-linearities by considering differentiable parts or sub-gradients.
  - ▶ There are methods for differentiable approximation of non-differentiable functions.
- ▶ GD is sensitive to the initial point: Converges to a local minimum for a small step length (typically) to the closest one.
- ▶ GD is quite sensitive to the step length.  
Might be very slow or too fast (even overshoot and diverge).

# Properties of Gradient Descent

- ▶ The objective must be differentiable, however:
  - ▶ Can be extended to functions with few non-linearities by considering differentiable parts or sub-gradients.
  - ▶ There are methods for differentiable approximation of non-differentiable functions.
- ▶ GD is sensitive to the initial point: Converges to a local minimum for a small step length (typically) to the closest one.
- ▶ GD is quite sensitive to the step length.  
Might be very slow or too fast (even overshoot and diverge).
- ▶ For convex functions, the algorithm converges to a minimum (if it converges).

# Properties of Gradient Descent

- ▶ The objective must be differentiable, however:
  - ▶ Can be extended to functions with few non-linearities by considering differentiable parts or sub-gradients.
  - ▶ There are methods for differentiable approximation of non-differentiable functions.
- ▶ GD is sensitive to the initial point: Converges to a local minimum for a small step length (typically) to the closest one.
- ▶ GD is quite sensitive to the step length.  
Might be very slow or too fast (even overshoot and diverge).
- ▶ For convex functions, the algorithm converges to a minimum (if it converges).
- ▶ Straightforward to implement if the derivatives are available.

GD is much more interesting in multiple variables, forming the basis for neural network learning (see later).

Better algorithm for unimodal functions using just derivatives?

# Newton's Method

An objective function  $f : \mathbb{R} \rightarrow \mathbb{R}$

A variable  $x \in \mathbb{R}$

Find  $x^*$  such that

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

# Newton's Method

An objective function  $f : \mathbb{R} \rightarrow \mathbb{R}$

A variable  $x \in \mathbb{R}$

Find  $x^*$  such that

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

Assume that

$$f''(x) = \lim_{h \rightarrow 0} \frac{f'(x+h) - f'(x)}{h} \quad \text{for } x \in \mathbb{R}$$

is continuous on  $\mathbb{R}$ .

Denote by  $\mathcal{C}^2$  the set of all twice continuously differentiable functions.

## Taylor Series Approximation

We would need the  $o$ -notation: Given functions  $f, g : \mathbb{R} \rightarrow \mathbb{R}$  we write  $f = o(g)$  if

$$\lim_{x \rightarrow 0} \frac{f(x)}{g(x)} = 0$$



# Taylor Series Approximation

We would need the  $o$ -notation: Given functions  $f, g : \mathbb{R} \rightarrow \mathbb{R}$  we write  $f = o(g)$  if

$$\lim_{x \rightarrow 0} \frac{f(x)}{g(x)} = 0$$

Consider a function  $f : \mathbb{R} \rightarrow \mathbb{R}$  and  $x_0 \in \mathbb{R}$ . Assume that  $f$  is twice differentiable at  $x_0$ . Then for all  $x \in \mathbb{R}$  we have that

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + o(|x - x_0|^2)$$

# Taylor Series Approximation

We would need the  $o$ -notation: Given functions  $f, g : \mathbb{R} \rightarrow \mathbb{R}$  we write  $f = o(g)$  if

$$\lim_{x \rightarrow 0} \frac{f(x)}{g(x)} = 0$$

Consider a function  $f : \mathbb{R} \rightarrow \mathbb{R}$  and  $x_0 \in \mathbb{R}$ . Assume that  $f$  is twice differentiable at  $x_0$ . Then for all  $x \in \mathbb{R}$  we have that

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + o(|x - x_0|^2)$$

Thus, such  $f$  can be reasonably approximated around  $x_0$  with a quadratic function

$$f(x) \approx q(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2$$

## Newton's Method Idea

The method computes successive approximations  $x_0, x_1, \dots, x_k, \dots$  as the GD.

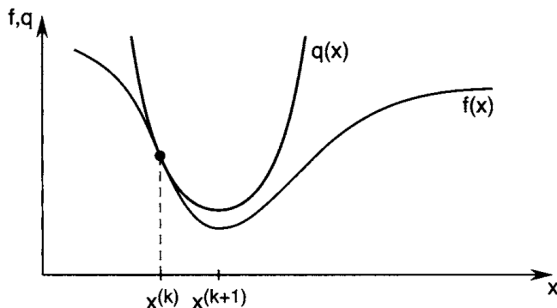
## Newton's Method Idea

The method computes successive approximations  $x_0, x_1, \dots, x_k, \dots$  as the GD.

To compute  $x_{k+1}$ , a quadratic approximation

$$q(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2$$

is considered around  $x_k$ .



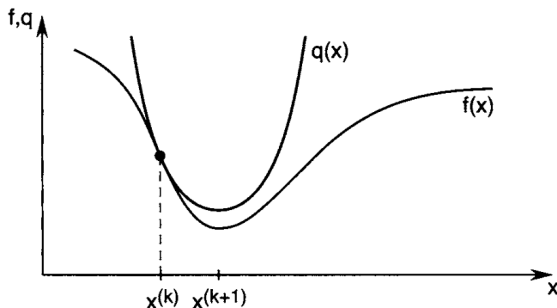
## Newton's Method Idea

The method computes successive approximations  $x_0, x_1, \dots, x_k, \dots$  as the GD.

To compute  $x_{k+1}$ , a quadratic approximation

$$q(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2$$

is considered around  $x_k$ .



Then  $x_{k+1}$  is set to the extreme point of  $q(x)$  (i.e.,  $q'(x_{k+1}) = 0$ ).

# Newton's Method Algorithm

Now note that for

$$q(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2$$

# Newton's Method Algorithm

Now note that for

$$q(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2$$

we have

$$q'(x) = f'(x_k) + f''(x_k)(x - x_k)$$

# Newton's Method Algorithm

Now note that for

$$q(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2$$

we have

$$q'(x) = f'(x_k) + f''(x_k)(x - x_k)$$

and thus

$$q'(x) = 0 \text{ iff } x = x_k - \frac{f'(x_k)}{f''(x_k)}$$



# Newton's Method Algorithm

Now note that for

$$q(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2$$

we have

$$q'(x) = f'(x_k) + f''(x_k)(x - x_k)$$

and thus

$$q'(x) = 0 \text{ iff } x = x_k - \frac{f'(x_k)}{f''(x_k)}$$

Newton's method then sets

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

# Newton's Method Algorithm

**Input:** A function  $f$  with derivative  $f'$  and second derivative  $f''$ ,  
initial point  $x_0$ , tolerance  $\epsilon > 0$

**Output:** A point  $x$  that approximately minimizes  $f(x)$

- 1: Set  $k \leftarrow 0$
- 2: **while**  $|x_{k+1} - x_k| > \epsilon$  **do**
- 3:     Calculate the derivative:  $y' \leftarrow f'(x_k)$
- 4:     Calculate the second derivative :  $y'' \leftarrow f''(x_k)$
- 5:     Update the estimate:  $x_{k+1} \leftarrow x_k - \frac{y'}{y''}$
- 6:     Increment  $k$
- 7: **end while**
- 8: **return**  $x_k$

Note that the method implicitly assumes that  $f''(x_k) \neq 0$  in every iteration.

## Example

Consider the following objective function  $f$

$$f(x) = \frac{1}{2}x^2 - \sin x$$

Assume  $x_0 = 0.5$ , and that the required accuracy is  $\epsilon = 10^{-5}$ , i.e., we stop when  $|x_{k+1} - x_k| \leq \epsilon$ .

## Example

Consider the following objective function  $f$

$$f(x) = \frac{1}{2}x^2 - \sin x$$

Assume  $x_0 = 0.5$ , and that the required accuracy is  $\epsilon = 10^{-5}$ , i.e., we stop when  $|x_{k+1} - x_k| \leq \epsilon$ .

We compute

$$f'(x) = x - \cos x, \quad f''(x) = 1 + \sin x.$$

## Example

Consider the following objective function  $f$

$$f(x) = \frac{1}{2}x^2 - \sin x$$

Assume  $x_0 = 0.5$ , and that the required accuracy is  $\epsilon = 10^{-5}$ , i.e., we stop when  $|x_{k+1} - x_k| \leq \epsilon$ .

We compute

$$f'(x) = x - \cos x, \quad f''(x) = 1 + \sin x.$$

Hence,

$$\begin{aligned} x_1 &= 0.5 - \frac{0.5 - \cos 0.5}{1 + \sin 0.5} \\ &= 0.5 - \frac{-0.3775}{1.479} \\ &= 0.7552 \end{aligned}$$

## Example

Proceeding similarly, we obtain

$$x_2 = x_1 - \frac{f'(x_1)}{f''(x_1)} = x_1 - \frac{0.02710}{1.685} = 0.7391$$

$$x_3 = x_2 - \frac{f'(x_2)}{f''(x_2)} = x_2 - \frac{9.461 \times 10^{-5}}{1.673} = 0.7390851339$$

$$x_4 = x_3 - \frac{f'(x_3)}{f''(x_3)} = x_3 - \frac{1.17 \times 10^{-9}}{1.673} = 0.7390851332$$

...

## Example

Proceeding similarly, we obtain

$$x_2 = x_1 - \frac{f'(x_1)}{f''(x_1)} = x_1 - \frac{0.02710}{1.685} = 0.7391$$

$$x_3 = x_2 - \frac{f'(x_2)}{f''(x_2)} = x_2 - \frac{9.461 \times 10^{-5}}{1.673} = 0.7390851339$$

$$x_4 = x_3 - \frac{f'(x_3)}{f''(x_3)} = x_3 - \frac{1.17 \times 10^{-9}}{1.673} = 0.7390851332$$

...

Note that

$$|x_4 - x_3| < \epsilon = 10^{-5}$$

$$f'(x_4) = -8.6 \times 10^{-6} \approx 0$$

$$f''(x_4) = 1.673 > 0$$

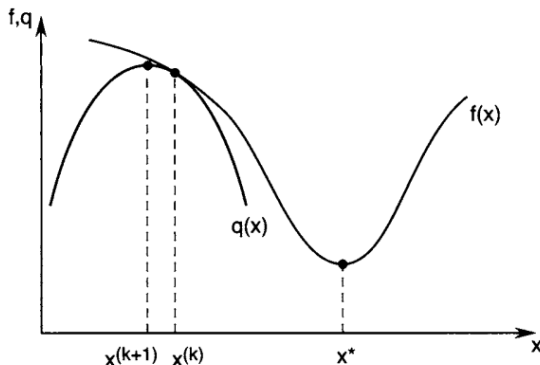
So, we conclude that  $x^* \approx x_4$  is a strict minimizer.

However, remember that the above does not have to be true!

# Convergence

Newton's method works well if  $f''(x) > 0$  everywhere.

However, if  $f''(x) < 0$  for some  $x$ , Newton's method may fail to converge to a minimizer (converges to a point  $x$  where  $f'(x) = 0$ ):



If the method converges to a minimizer, it does so *quadratically*.  
What does this mean?



# Types of Convergence Rates

## Linear Convergence

An algorithm is said to have linear convergence if the error at each step is proportionally reduced by a constant factor:

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|} = r, \quad 0 < r < 1$$

# Types of Convergence Rates

## Linear Convergence

An algorithm is said to have linear convergence if the error at each step is proportionally reduced by a constant factor:

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|} = r, \quad 0 < r < 1$$

## Superlinear Convergence

Convergence is superlinear if:

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|} = 0$$

This often requires an algorithm to utilize second-order information.

# Quadratic Convergence of Newton's Method

## Quadratic Convergence

Quadratic convergence is achieved when the number of accurate digits roughly doubles with each iteration:

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^2} = C, \quad C > 0$$

# Quadratic Convergence of Newton's Method

## Quadratic Convergence

Quadratic convergence is achieved when the number of accurate digits roughly doubles with each iteration:

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^2} = C, \quad C > 0$$

Newton's method is a classic example of an algorithm with quadratic convergence.

## Theorem 2 (Quadratic Convergence of Newton's Method)

*Let  $f : \mathbb{R} \rightarrow \mathbb{R}$  satisfy  $f \in \mathcal{C}^2$  and suppose  $x^*$  is a minimizer of  $f$  such that  $f''(x^*) > 0$ . Assume Lipschitz continuity of  $f''$ . If the initial guess  $x_0$  is sufficiently close to  $x^*$ , then the sequence  $\{x_k\}$  computed by the Newton's method converges quadratically to  $x^*$ .*

## Newton's Method of Tangents

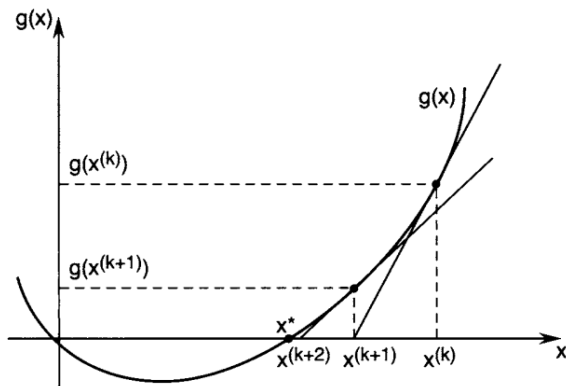
Newton's method is also a technique for finding roots of functions. In our case, this means finding a root of  $f'$ .

## Newton's Method of Tangents

Newton's method is also a technique for finding roots of functions. In our case, this means finding a root of  $f'$ .

Denote  $g = f'$ . Then Newton's approximation goes like this:

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}$$



## Secant Method

What if  $f''$  is unavailable, but we want to use something like Newton's method (with its superlinear convergence)?

## Secant Method

What if  $f''$  is unavailable, but we want to use something like Newton's method (with its superlinear convergence)?

Assume  $f \in \mathcal{C}^1$  and try to approximate  $f''$  around  $x_{k-1}$  with

$$f''(x) \approx \frac{f'(x) - f'(x_{k-1})}{x - x_{k-1}}$$

Substituting  $x$  with  $x_k$ , we obtain

$$\frac{1}{f''(x_k)} \approx \frac{x_k - x_{k-1}}{f'(x_k) - f'(x_{k-1})}$$



## Secant Method

What if  $f''$  is unavailable, but we want to use something like Newton's method (with its superlinear convergence)?

Assume  $f \in \mathcal{C}^1$  and try to approximate  $f''$  around  $x_{k-1}$  with

$$f''(x) \approx \frac{f'(x) - f'(x_{k-1})}{x - x_{k-1}}$$

Substituting  $x$  with  $x_k$ , we obtain

$$\frac{1}{f''(x_k)} \approx \frac{x_k - x_{k-1}}{f'(x_k) - f'(x_{k-1})}$$

Then, we may try to use Newton's step with this approximation:

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

Is the rate of convergence superlinear?

## Example

Consider the following objective function  $f$

$$f(x) = \frac{1}{2}x^2 - \sin x$$

Assume  $x_0 = 0.5$  and  $x_1 = 1.0$ .

Now, we need to initialize the first two values.

## Example

Consider the following objective function  $f$

$$f(x) = \frac{1}{2}x^2 - \sin x$$

Assume  $x_0 = 0.5$  and  $x_1 = 1.0$ .

Now, we need to initialize the first two values.

We have  $f'(x) = x - \cos x$

Hence,

$$\begin{aligned}x_2 &= 1.0 - \frac{1.0 - 0.5}{(1.0 - \cos 1.0) - (0.5 - \cos 0.5)}(0.5 - \cos 0.5) \\&= 0.7254\end{aligned}$$

## Example

Continuing, we obtain:

$$x_0 = 0.5$$

$$x_1 = 1.0$$

$$x_2 = 0.72548$$

$$x_3 = 0.73839$$

$$x_4 = 0.739087$$

$$x_5 = 0.739085132$$

$$x_6 = 0.739085133$$

## Example

Start the secant method with the approximation given by Newton's method:

$$x_0 = 0.5$$

$$x_1 = 0.7552$$

$$x_2 = 0.7381$$

$$x_3 = 0.739081$$

$$x_5 = 0.7390851339$$

$$x_6 = 0.7390851332$$

...

Compare with Newton's method:

$$x_0 = 0.5$$

$$x_1 = 0.7552$$

$$x_2 = 0.7391$$

$$x_3 = 0.7390851339$$

$$x_4 = 0.73908513321516067229$$

$$x_5 = 0.73908513321516067229$$

...

# Superlinear Convergence of Secant Method

## Theorem 3 (Superlinear Convergence of Secant Method)

*Assume  $f : \mathbb{R} \rightarrow \mathbb{R}$  twice continuously differentiable and  $x^*$  a minimizer of  $f$ . Assume  $f''$  Lipschitz continuous and  $f''(x^*) > 0$ . The sequence  $\{x_k\}$  generated by the Secant method converges to  $x^*$  superlinearly if  $x_0$  and  $x_1$  are sufficiently close to  $x^*$ .*

The rate of convergence  $p$  of the Secant method is given by the positive root of the equation  $p^2 - p - 1 = 0$ , which is  $p = \frac{1+\sqrt{5}}{2} \approx 1.618$  (the golden ratio). Formally,

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^{\frac{1+\sqrt{5}}{2}}} = C, \quad C > 0$$

## Secant Method for Root Finding

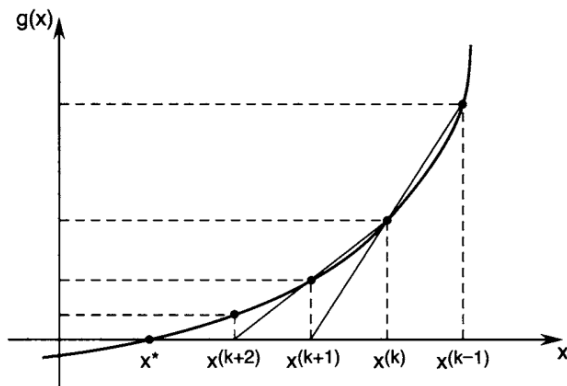
As for Newton's method of tangents, the secant method can be seen as a method for finding a root of  $f'$ .

## Secant Method for Root Finding

As for Newton's method of tangents, the secant method can be seen as a method for finding a root of  $f'$ .

Denote  $g = f'$ . Then the secant method approximation is

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{g(x_k) - g(x_{k-1})} \cdot g(x_k)$$





# General Form

Note that all methods have similar update formula:

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

Different choice of  $a_k$  produce different algorithm:

- ▶  $a_k = 1$  gives the **gradient descent**,
- ▶  $a_k = f''(x_k)$  gives **Newton's method**,
- ▶  $a_k = \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}}$  gives the **secant method**,
- ▶  $a_k = f''(x_m)$  where  $m = \lfloor k/p \rfloor p$  gives **Shamanskii method**.

# Summary

- ▶ Newton's method
  - ▶ Converges quickly to an extremum under rather strict conditions (see Theorem 2)
  - ▶ The choice of the initial point is critical; the method may diverge to a stationary point, which is not a minimizer. The method may also cycle.
  - ▶ If the second derivative is very small, close to the minimizer, the method can be very slow (the quadratic convergence is guaranteed only if the second derivative is non-zero at the minimizer and the constants depend on the second derivative).

# Summary

- ▶ Newton's method
  - ▶ Converges quickly to an extremum under rather strict conditions (see Theorem 2)
  - ▶ The choice of the initial point is critical; the method may diverge to a stationary point, which is not a minimizer. The method may also cycle.
  - ▶ If the second derivative is very small, close to the minimizer, the method can be very slow (the quadratic convergence is guaranteed only if the second derivative is non-zero at the minimizer and the constants depend on the second derivative).
- ▶ Secant method
  - ▶ The second derivative is not needed.
  - ▶ Superlinear (but not quadratic) convergence for an initial point close to a minimum (under rather strict conditions Theorem 3)

# Constrained Single Variable Optimization Problem

An objective function  $f : \mathbb{R} \rightarrow \mathbb{R}$

A variable  $x$

A constraint

$$a_0 \leq x \leq b_0$$

Consider the following cases:

- ▶  $f$  continuously differentiable on  $[a_0, b_0]$
- ▶  $f$  twice continuously differentiable on  $[a_0, b_0]$

**Homework:** Modify the gradient descent and Newton's method to work on the bounded interval (the above definitions guarantee continuous differentiability at  $a_0$  and  $b_0$ ).

# Unconstrained Optimization Overview

## Notation

In what follows, we will work with vectors in  $\mathbb{R}^n$ .

The vectors will be (usually) denoted by  $x \in \mathbb{R}^n$ .

We often consider sequences of vectors,  $x_0, x_1, \dots, x_k, \dots$

The index  $k$  will usually indicate that  $x_k$  is the  $k$ -th vector in a sequence.

When we talk (relatively rarely) about components of vectors, we use  $i$  as an index, i.e.,  $x_i$  will be the  $i$ -th component of  $x \in \mathbb{R}^n$ .

We denote by  $\|x\|$  the Euclidean norm of  $x$ .

We denote by  $\|x\|_\infty$  the  $\mathcal{L}^\infty$  norm giving the maximum of absolute values of components of  $x$ .

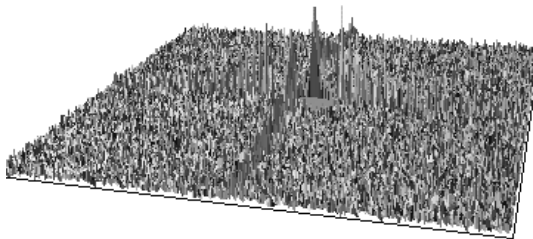
We occasionally use the matrix norm  $\|A\|$ , consistent with the Euclidean norm, defined by

$$\|A\| = \sup_{\|x\|=1} \|Ax\| = \sqrt{\lambda_1}$$

Here  $\lambda_1$  is the largest eigenvalue of  $A^\top A$ .

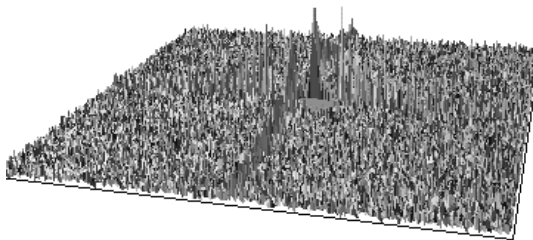
# How to Recognize (Local) Minimum

How do we verify that  $x^* \in \mathbb{R}^n$  is a minimizer of  $f$ ?



# How to Recognize (Local) Minimum

How do we verify that  $x^* \in \mathbb{R}^n$  is a minimizer of  $f$ ?



Technically, we should examine *all* points in the immediate vicinity if one has a smaller value (impractical).

Assuming the smoothness of  $f$ , we may benefit from the “stable” behavior of  $f$  around  $x^*$ .



# Derivatives and Gradients

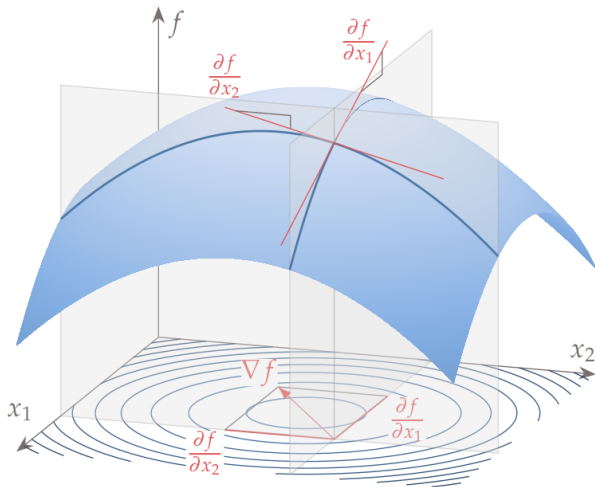
The gradient of  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , denoted by  $\nabla f(x)$ , is a column vector of first-order partial derivatives of the function concerning each variable:

$$\nabla f(x) = \left[ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right]^\top,$$

Where each partial derivative is defined as the following limit:

$$\frac{\partial f}{\partial x_i} = \lim_{\varepsilon \rightarrow 0} \frac{f(x_1, \dots, x_i + \varepsilon, \dots, x_n) - f(x_1, \dots, x_i, \dots, x_n)}{\varepsilon}$$

# Gradient



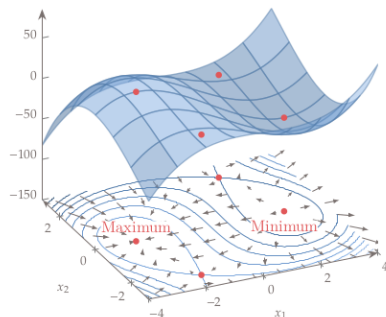
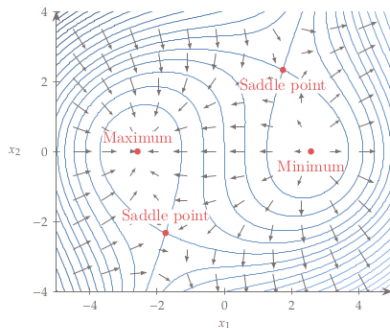
The gradient is a vector pointing in the direction of the most significant function increase from the current point.

# Gradient

Consider the following function of two variables:

$$f(x_1, x_2) = x_1^3 + 2x_1x_2^2 - x_2^3 - 20x_1.$$

$$\nabla f(x_1, x_2) = \begin{bmatrix} 3x_1^2 + 2x_2^2 - 20 \\ 4x_1x_2 - 3x_2^2 \end{bmatrix}$$

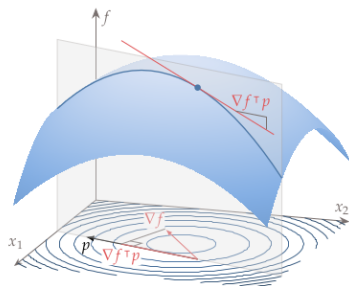
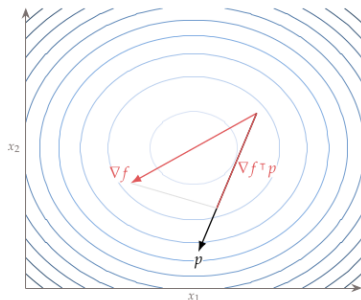


## Directional Derivatives vs Gradient

The rate of change in a direction  $p$  is quantified by a directional derivative, defined as

$$\nabla_p f(x) = \lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon p) - f(x)}{\varepsilon}.$$

We can find this derivative by projecting the gradient onto the desired direction  $p$  using the dot product  $\nabla_p f(x) = (\nabla f(x))^\top p$



(Here, we assume continuous partial derivatives.)

## Geometry of Gradient

Consider the geometric interpretation of the dot product:

$$\nabla_p f(x) = (\nabla f(x))^{\top} p = \|\nabla f\| \|p\| \cos \theta$$

Here  $\theta$  is the angle between  $\nabla f$  and  $p$ .

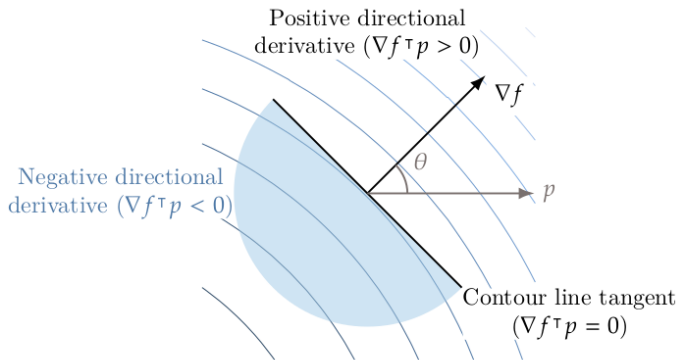
# Geometry of Gradient

Consider the geometric interpretation of the dot product:

$$\nabla_p f(x) = (\nabla f(x))^T p = \|\nabla f\| \|\nabla p\| \cos \theta$$

Here  $\theta$  is the angle between  $\nabla f$  and  $p$ .

The directional derivative is maximized by  $\theta = 0$ , i.e., when  $\nabla f$  and  $p$  point in the same direction.



## Hessian

Taking derivative twice, possibly w.r.t. different variables, gives the *Hessian* of  $f$

$$\nabla^2 f(x) = H(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

Note that the Hessian is a function which takes  $x \in \mathbb{R}^n$  and gives a  $n \times n$ -matrix of second derivatives of  $f$ .

## Hessian

Taking derivative twice, possibly w.r.t. different variables, gives the *Hessian* of  $f$

$$\nabla^2 f(x) = H(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

Note that the Hessian is a function which takes  $x \in \mathbb{R}^n$  and gives a  $n \times n$ -matrix of second derivatives of  $f$ .

We have

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

If  $f$  has continuous second partial derivatives, then  $H$  is symmetric, i.e.,  $H_{ij} = H_{ji}$ .



## Geometry of Hessian

Let  $x$  be fixed and let  $g(t) = f(x + tp)$ .

What exactly are  $g'(0)$  and  $g''(0)$ ?

## Geometry of Hessian

Let  $x$  be fixed and let  $g(t) = f(x + tp)$ .

What exactly are  $g'(0)$  and  $g''(0)$ ?

$$g'(t) = f(x + tp)' = [\nabla f(x + tp)]^\top p = \nabla_p f(x + tp)$$

and thus  $g'(0) = \nabla_p f(x)$ .

## Geometry of Hessian

Let  $x$  be fixed and let  $g(t) = f(x + tp)$ .

What exactly are  $g'(0)$  and  $g''(0)$ ?

$$g'(t) = f(x + tp)' = [\nabla f(x + tp)]^\top p = \nabla_p f(x + tp)$$

and thus  $g'(0) = \nabla_p f(x)$ .

Now note that for all  $z$  we have

$$\nabla(\nabla_p f(z)) = \nabla([\nabla f(z)]^\top p) = (\nabla^2 f(z))p = H(z)p$$

## Geometry of Hessian

Let  $x$  be fixed and let  $g(t) = f(x + tp)$ .

What exactly are  $g'(0)$  and  $g''(0)$ ?

$$g'(t) = f(x + tp)' = [\nabla f(x + tp)]^\top p = \nabla_p f(x + tp)$$

and thus  $g'(0) = \nabla_p f(x)$ .

Now note that for all  $z$  we have

$$\nabla(\nabla_p f(z)) = \nabla([\nabla f(z)]^\top p) = (\nabla^2 f(z))p = H(z)p$$

Thus, for our fixed  $x$  we have

$$\begin{aligned} g''(t) &= (g'(t))' \\ &= (\nabla_p f(x + tp))' = [\nabla(\nabla_p f(x + tp))]^\top p \\ &= (H(x + tp)p)^\top p = p^\top H(x + tp)p \end{aligned}$$

## Geometry of Hessian

Let  $x$  be fixed and let  $g(t) = f(x + tp)$ .

What exactly are  $g'(0)$  and  $g''(0)$ ?

$$g'(t) = f(x + tp)' = [\nabla f(x + tp)]^\top p = \nabla_p f(x + tp)$$

and thus  $g'(0) = \nabla_p f(x)$ .

Now note that for all  $z$  we have

$$\nabla(\nabla_p f(z)) = \nabla([\nabla f(z)]^\top p) = (\nabla^2 f(z))p = H(z)p$$

Thus, for our fixed  $x$  we have

$$\begin{aligned} g''(t) &= (g'(t))' \\ &= (\nabla_p f(x + tp))' = [\nabla(\nabla_p f(x + tp))]^\top p \\ &= (H(x + tp)p)^\top p = p^\top H(x + tp)p \end{aligned}$$

Thus  $g''(0) = p^\top H(x)p$ .

## Principal Curvature Directions

Fix  $x$  and consider  $H = H(x)$ . Consider unit eigenvectors  $\hat{v}_k$  of  $H$ :

$$H\hat{v}_k = \kappa_k \hat{v}_k$$

For symmetric  $H$ , the unit eigenvectors form an orthonormal basis,

# Principal Curvature Directions

Fix  $x$  and consider  $H = H(x)$ . Consider unit eigenvectors  $\hat{v}_k$  of  $H$ :

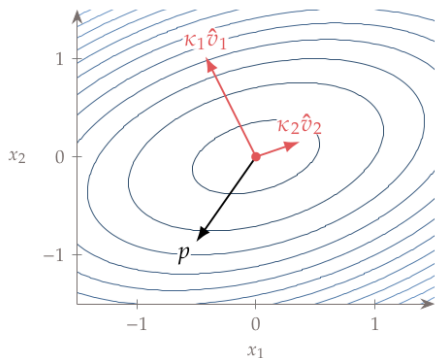
$$H\hat{v}_k = \kappa_k \hat{v}_k$$

For symmetric  $H$ , the unit eigenvectors form an orthonormal basis, and there is a rotation matrix  $R$  such that

$$H = RDR^{-1} = RDR^T$$

Here  $D$  is diagonal with  $\kappa_1, \dots, \kappa_n$  on the diagonal.

If  $\kappa_1 \geq \dots \geq \kappa_n$ , the direction of  $\hat{v}_1$  is the maximum curvature direction of  $f$  at  $x$ .



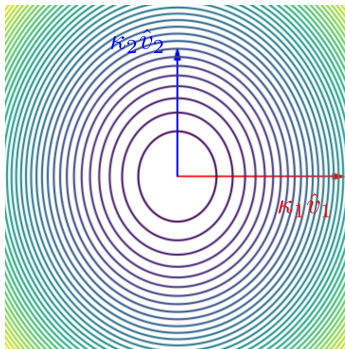
Consider  $f(x) = x^\top Hx$  where

$$H = \begin{pmatrix} 4/3 & 0 \\ 0 & 1 \end{pmatrix}$$

The eigenvalues are

$$\kappa_1 = 4/3 \quad \kappa_2 = 1$$

Their corresponding eigenvectors are  $(1, 0)^\top$  and  $(0, 1)^\top$ .





Consider  $f(x) = x^\top Hx$  where

$$H = \begin{pmatrix} 4/3 & 0 \\ 0 & 1 \end{pmatrix}$$

The eigenvalues are

$$\kappa_1 = 4/3 \quad \kappa_2 = 1$$

Their corresponding eigenvectors are  $(1, 0)^\top$  and  $(0, 1)^\top$ .

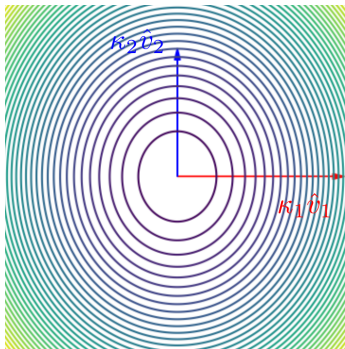
Note that

$$f(x) = \kappa_1 x_1^2 + \kappa_2 x_2^2$$

Considering a direction vector  $p$  and  $x = (0, 0)^\top$  we get

$$g(t) = f(x + tp) = f(tp) = t^2 (\kappa_1 p_1^2 + \kappa_2 p_2^2)$$

which is a parabola with  $g'' = 2 (\kappa_1 p_1^2 + \kappa_2 p_2^2)$ .



Consider  $f(x) = x^\top Hx$  where

$$H = \begin{pmatrix} 4/3 & 1/3 \\ 1/3 & 3/3 \end{pmatrix}$$

Consider  $f(x) = x^\top Hx$  where

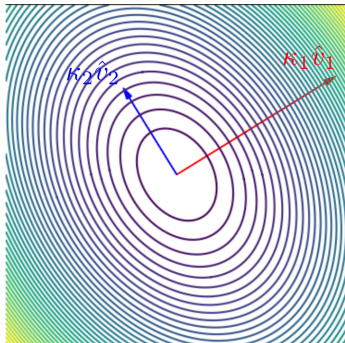
$$H = \begin{pmatrix} 4/3 & 1/3 \\ 1/3 & 3/3 \end{pmatrix}$$

The eigenvalues are

$$\kappa_1 = \frac{1}{6}(7+\sqrt{5}) \quad \kappa_2 = \frac{1}{6}(7-\sqrt{5})$$

Their corresponding eigenvectors are

$$\hat{v}_1 = \left( \frac{1}{2}(1 + \sqrt{5}), 1 \right) \quad \hat{v}_2 = \left( \frac{1}{2}(1 - \sqrt{5}), 1 \right)$$



Consider  $f(x) = x^T H x$  where

$$H = \begin{pmatrix} 4/3 & 1/3 \\ 1/3 & 3/3 \end{pmatrix}$$

The eigenvalues are

$$\kappa_1 = \frac{1}{6}(7+\sqrt{5}) \quad \kappa_2 = \frac{1}{6}(7-\sqrt{5})$$

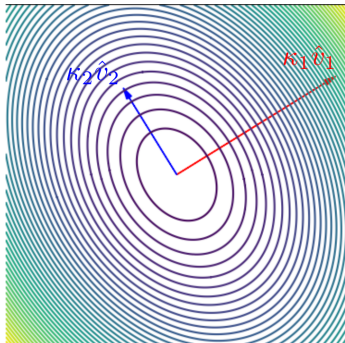
Their corresponding eigenvectors are

$$\hat{v}_1 = \left( \frac{1}{2}(1 + \sqrt{5}), 1 \right) \quad \hat{v}_2 = \left( \frac{1}{2}(1 - \sqrt{5}), 1 \right)$$

Note that

$$H = (\hat{v}_1 \ \hat{v}_2) \begin{pmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{pmatrix} (\hat{v}_1 \ \hat{v}_2)^T$$

Here  $(\hat{v}_1 \ \hat{v}_2)$  is a  $2 \times 2$  matrix whose columns are  $\hat{v}_1, \hat{v}_2$ .



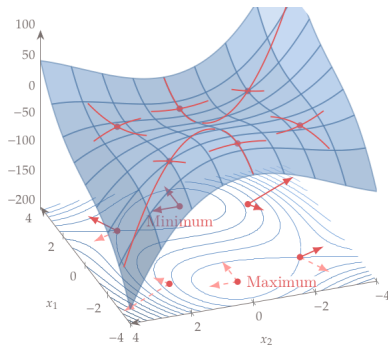
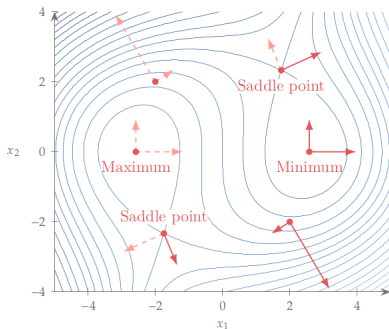
# Hessian Visualization Example

Consider

$$f(x_1, x_2) = x_1^3 + 2x_1x_2^2 - x_2^3 - 20x_1.$$

And it's Hessian.

$$H(x_1, x_2) = \begin{bmatrix} 6x_1 & 4x_2 \\ 4x_2 & 4x_1 - 6x_2 \end{bmatrix}.$$



# Taylor's Theorem

## Theorem 4 (Taylor)

*Suppose that  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is twice continuously differentiable and that  $p \in \mathbb{R}^n$ . Then, we have*

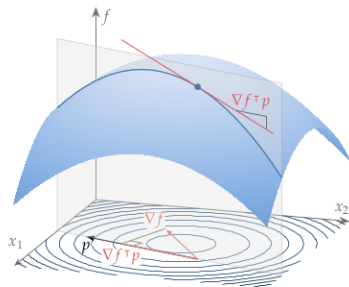
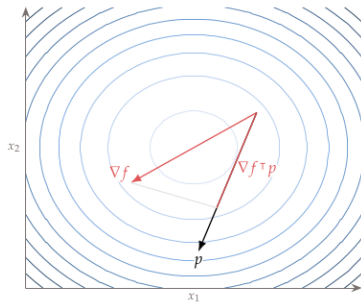
$$f(x + p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T H(x) p + o(\|p\|^2).$$

*Here  $H = \nabla^2 f$  is the Hessian of  $f$ .*

# First-Order Necessary Conditions

## Theorem 5

*If  $x^*$  is a local minimizer and  $f$  is continuously differentiable in an open neighborhood of  $x^*$ , then  $\nabla f(x^*) = 0$ .*



## Second-Order Conditions

Note that  $\nabla f(x^*) = 0$  does not tell us whether  $x^*$  is a minimizer, maximizer, or a saddle point.



## Second-Order Conditions

Note that  $\nabla f(x^*) = 0$  does not tell us whether  $x^*$  is a minimizer, maximizer, or a saddle point.

However, knowing the curvature in all directions from  $x^*$  might tell us what  $x^*$  is, right?

## Second-Order Conditions

Note that  $\nabla f(x^*) = 0$  does not tell us whether  $x^*$  is a minimizer, maximizer, or a saddle point.

However, knowing the curvature in all directions from  $x^*$  might tell us what  $x^*$  is, right?

## Second-Order Conditions

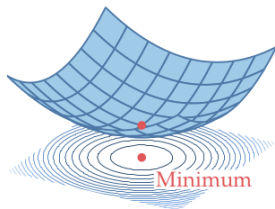
Note that  $\nabla f(x^*) = 0$  does not tell us whether  $x^*$  is a minimizer, maximizer, or a saddle point.

However, knowing the curvature in all directions from  $x^*$  might tell us what  $x^*$  is, right?

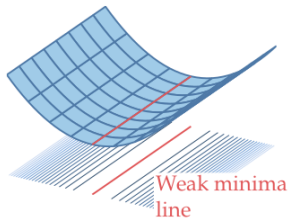
All comes down to the *definiteness* of  $H := H(x^*)$ .

- ▶  $H$  is **positive definite** if  $p^\top H p > 0$  for all  $p$   
iff all eigenvalues of  $H$  are positive
- ▶  $H$  is **positive semi-definite** if  $p^\top H p \geq 0$  for all  $p$   
iff all eigenvalues of  $H$  are nonnegative
- ▶  $H$  is **negative semi-definite** if  $p^\top H p \leq 0$  for all  $p$   
iff all eigenvalues of  $H$  are nonpositive
- ▶  $H$  is **negative definite** if  $p^\top H p < 0$  for all  $p$   
iff all eigenvalues of  $H$  are negative
- ▶  $H$  is **indefinite** if it is not definite in the above sense  
iff  $H$  has at least one positive and one negative eigenvalue.

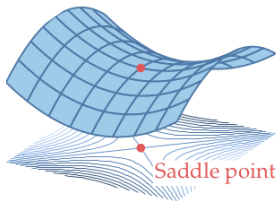
# Definiteness



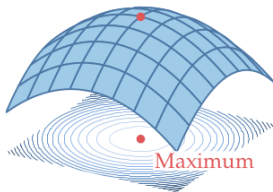
Positive definite



Positive semidefinite



Indefinite



Negative definite

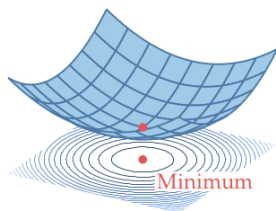
# Second-Order Necessary Condition

## Theorem 6 (Second-Order Necessary Conditions)

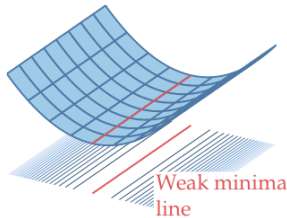
*If  $x^*$  is a local minimizer of  $f$  and  $\nabla^2 f$  is continuous in a neighborhood of  $x^*$ , then  $\nabla f(x^*) = 0$  and  $\nabla^2 f(x^*)$  is positive semidefinite.*

## Theorem 7 (Second-Order Sufficient Conditions)

*Suppose that  $\nabla^2 f$  is continuous in a neighborhood of  $x^*$  and that  $\nabla f(x^*) = 0$  and  $\nabla^2 f(x^*)$  is positive definite. Then  $x^*$  is a strict local minimizer of  $f$ .*



Positive definite



Positive semidefinite

## Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

## Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

Consider the gradient equal to zero:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1^3 + 6x_1^2 + 3x_1 - 2x_2 \\ 2x_2 - 2x_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

## Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

Consider the gradient equal to zero:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1^3 + 6x_1^2 + 3x_1 - 2x_2 \\ 2x_2 - 2x_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

From the second equation, we have that  $x_2 = x_1$ . Substituting this into the first equation yields

$$x_1 (2x_1^2 + 6x_1 + 1) = 0.$$



## Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

Consider the gradient equal to zero:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1^3 + 6x_1^2 + 3x_1 - 2x_2 \\ 2x_2 - 2x_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

From the second equation, we have that  $x_2 = x_1$ . Substituting this into the first equation yields

$$x_1 (2x_1^2 + 6x_1 + 1) = 0.$$

The solution of this equation yields three points:

$$x_A = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad x_B = \begin{bmatrix} -\frac{3}{2} - \frac{\sqrt{7}}{2} \\ -\frac{3}{2} - \frac{\sqrt{7}}{2} \end{bmatrix}, \quad x_C = \begin{bmatrix} \frac{\sqrt{7}}{2} - \frac{3}{2} \\ \frac{\sqrt{7}}{2} - \frac{3}{2} \end{bmatrix}.$$

## Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

## Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

To classify  $x_A, x_B, x_C$ , we need to compute the Hessian matrix:

$$H(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 6x_1^2 + 12x_1 + 3 & -2 \\ -2 & 2 \end{bmatrix}.$$

## Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

To classify  $x_A, x_B, x_C$ , we need to compute the Hessian matrix:

$$H(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 6x_1^2 + 12x_1 + 3 & -2 \\ -2 & 2 \end{bmatrix}.$$

The Hessian, at the first point, is

$$H(x_A) = \begin{bmatrix} 3 & -2 \\ -2 & 2 \end{bmatrix},$$

whose eigenvalues are  $\kappa_1 \approx 0.438$  and  $\kappa_2 \approx 4.561$ . Because both eigenvalues are positive, this point is a local minimum.

## Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

To classify  $x_A, x_B, x_C$ , we need to compute the Hessian matrix:

$$H(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 6x_1^2 + 12x_1 + 3 & -2 \\ -2 & 2 \end{bmatrix}.$$

For the second point,

$$H(x_B) = \begin{bmatrix} 3(3 + \sqrt{7}) & -2 \\ -2 & 2 \end{bmatrix}.$$

The eigenvalues are  $\kappa_1 \approx 1.737$  and  $\kappa_2 \approx 17.200$ , so this point is another local minimum.

## Example

Consider the following function of two variables:

$$f(x_1, x_2) = 0.5x_1^4 + 2x_1^3 + 1.5x_1^2 + x_2^2 - 2x_1x_2.$$

To classify  $x_A, x_B, x_C$ , we need to compute the Hessian matrix:

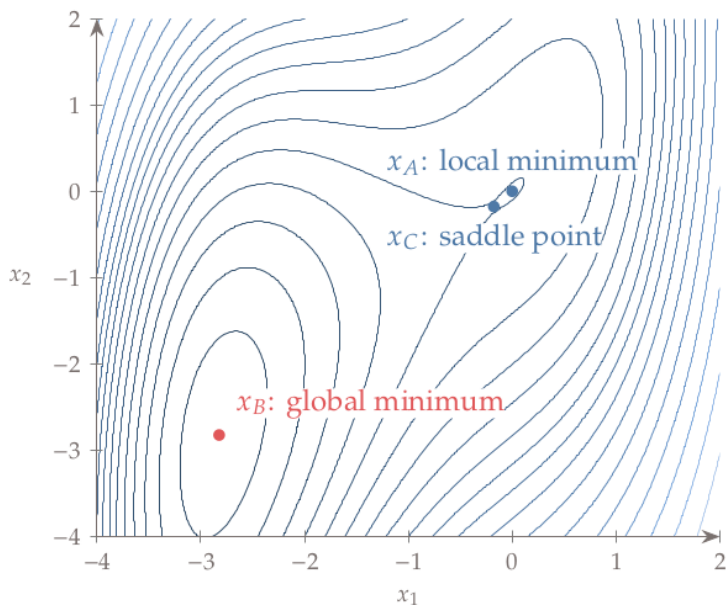
$$H(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 6x_1^2 + 12x_1 + 3 & -2 \\ -2 & 2 \end{bmatrix}.$$

For the third point,

$$H(x_C) = \begin{bmatrix} 9 - 3\sqrt{7} & -2 \\ -2 & 2 \end{bmatrix}.$$

The eigenvalues for this Hessian are  $\kappa_1 \approx -0.523$  and  $\kappa_2 \approx 3.586$ , so this point is a saddle point.

## Example



# Proofs of Some Theorems

## Optional



# Taylor's Theorem

To prove the theorems characterizing minima/maxima, we need the following form of Taylor's theorem:

## Theorem 8 (Taylor)

*Suppose that  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is continuously differentiable and that  $p \in \mathbb{R}^n$ . Then we have that.*

$$f(x + p) = f(x) + \nabla f(x + tp)^T p,$$

*for some  $t \in (0, 1)$ . Moreover, if  $f$  is twice continuously differentiable, we have that*

$$f(x + p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x + tp) p,$$

*for some  $t \in (0, 1)$ .*

## Proof of Theorem 5 (Optional)

We prove that if  $x^*$  is a local minimizer and  $f$  is continuously differentiable in an open neighborhood of  $x^*$ , then  $\nabla f(x^*) = 0$ .

Suppose for contradiction that  $\nabla f(x^*) \neq 0$ . Define the vector  $p = -\nabla f(x^*)$  and note that  $p^T \nabla f(x^*) = -\|\nabla f(x^*)\|^2 < 0$ . Because  $\nabla f$  is continuous near  $x^*$ , there is a scalar  $T > 0$  such that

$$p^T \nabla f(x^* + tp) < 0, \quad \text{for all } t \in [0, T]$$

For any  $\bar{t} \in (0, T]$ , we have by Taylor's theorem that

$$f(x^* + \bar{t}p) = f(x^*) + \bar{t}p^T \nabla f(x^* + tp), \quad \text{for some } t \in (0, \bar{t}).$$

Therefore,  $f(x^* + \bar{t}p) < f(x^*)$  for all  $\bar{t} \in (0, T]$ . We have found a direction leading away from  $x^*$  along which  $f$  decreases, so  $x^*$  is not a local minimizer, and we have a contradiction.  $\square$

## Proof of Theorem 6 (Optional)

We prove that if  $x^*$  is a local minimizer of  $f$  and  $\nabla^2 f$  is continuous in an open neighborhood of  $x^*$ , then  $\nabla f(x^*) = 0$  and  $\nabla^2 f(x^*)$  is positive semidefinite.

We know that  $\nabla f(x^*) = 0$ . For contradiction, assume that  $\nabla^2 f(x^*)$  is not positive semidefinite.

Then we can choose a vector  $p$  such that  $p^T \nabla^2 f(x^*) p < 0$ .

As  $\nabla^2 f$  is continuous near  $x^*$ ,  $p^T \nabla^2 f(x^* + tp) p < 0$  for all  $t \in [0, T]$  where  $T > 0$ .

By Taylor we have for all  $\bar{t} \in (0, T]$  and some  $t \in (0, \bar{t})$

$$f(x^* + \bar{t}p) = f(x^*) + \bar{t}p^T \nabla f(x^*) + \frac{1}{2}\bar{t}^2 p^T \nabla^2 f(x^* + tp) p < f(x^*).$$

Thus,  $x^*$  is not a local minimizer. □

## Proof of Theorem 7 (Optional)

We prove the following: Suppose that  $\nabla^2 f$  is continuous in an open neighborhood of  $x^*$  and that  $\nabla f(x^*) = 0$  and  $\nabla^2 f(x^*)$  is positive definite. Then  $x^*$  is a strict local minimizer of  $f$ .

Because the Hessian is continuous and positive definite at  $x^*$ , we can choose a radius  $r > 0$  so that  $\nabla^2 f(x)$  remains positive definite for all  $x$  in the open ball  $\mathcal{D} = \{z \mid \|z - x^*\| < r\}$ . Taking any nonzero vector  $p$  with  $\|p\| < r$ , we have  $x^* + p \in \mathcal{D}$  and so

$$\begin{aligned} f(x^* + p) &= f(x^*) + p^T \nabla f(x^*) + \frac{1}{2} p^T \nabla^2 f(z) p \\ &= f(x^*) + \frac{1}{2} p^T \nabla^2 f(z) p, \end{aligned}$$

where  $z = x^* + tp$  for some  $t \in (0, 1)$ . Since  $z \in \mathcal{D}$ , we have  $p^T \nabla^2 f(z) p > 0$ , and therefore  $f(x^* + p) > f(x^*)$ , giving the result. □

# Unconstrained Optimization Algorithms

# Search Algorithms

We consider algorithms that

- ▶ Start with an initial guess  $x_0$
- ▶ Generate a sequence of points  $x_0, x_1, \dots$
- ▶ Stop when no progress can be made or when a minimizer seems approximated with sufficient accuracy.

To compute  $x_{k+1}$  the algorithms use the information about  $f$  at the previous iterates  $x_0, x_1, \dots, x_k$ .

# Search Algorithms

We consider algorithms that

- ▶ Start with an initial guess  $x_0$
- ▶ Generate a sequence of points  $x_0, x_1, \dots$
- ▶ Stop when no progress can be made or when a minimizer seems approximated with sufficient accuracy.

To compute  $x_{k+1}$  the algorithms use the information about  $f$  at the previous iterates  $x_0, x_1, \dots, x_k$ .

The *monotone* algorithms satisfy  $f(x_{k+1}) < f(x_k)$ .

# Search Algorithms

We consider algorithms that

- ▶ Start with an initial guess  $x_0$
- ▶ Generate a sequence of points  $x_0, x_1, \dots$
- ▶ Stop when no progress can be made or when a minimizer seems approximated with sufficient accuracy.

To compute  $x_{k+1}$  the algorithms use the information about  $f$  at the previous iterates  $x_0, x_1, \dots, x_k$ .

The *monotone* algorithms satisfy  $f(x_{k+1}) < f(x_k)$ .

There are two overall strategies:

- ▶ Line search
- ▶ Trust region



## Line Search Overview

To compute  $x_{k+1}$ , a line search algorithm chooses

- ▶ *direction*  $p_k$
- ▶ *step size*  $\alpha_k$

and computes

$$x_{k+1} = x_k + \alpha_k p_k$$

# Line Search Overview

To compute  $x_{k+1}$ , a line search algorithm chooses

- ▶ *direction*  $p_k$
- ▶ *step size*  $\alpha_k$

and computes

$$x_{k+1} = x_k + \alpha_k p_k$$

The vector  $p_k$  should be a *descent* direction, i.e., a direction in which  $f$  decreases locally.

## Line Search Overview

To compute  $x_{k+1}$ , a line search algorithm chooses

- ▶ *direction*  $p_k$
- ▶ *step size*  $\alpha_k$

and computes

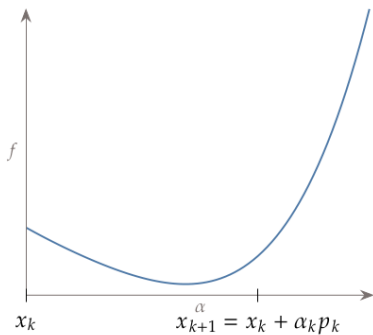
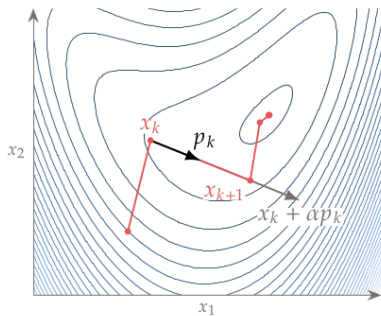
$$x_{k+1} = x_k + \alpha_k p_k$$

The vector  $p_k$  should be a *descent* direction, i.e., a direction in which  $f$  decreases locally.

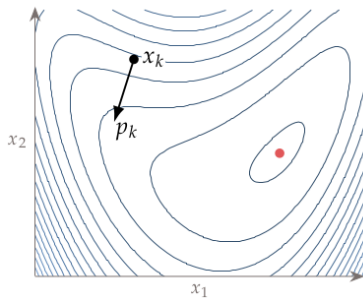
$\alpha_k$  is selected to approximately solve

$$\min_{\alpha > 0} f(x_k + \alpha p_k)$$

However, typically, an exact solution is expensive and unnecessary. Instead, line search algorithms inspect a limited number of trial step lengths and find one that decreases  $f$  appropriately (see later).



A descent direction does not have to be followed to the minimum.



## Trust Region

To compute  $x_{k+1}$ , a trust region algorithm chooses

- ▶ *model function*  $m_k$  whose behavior near  $x_k$  is similar to  $f$
- ▶ a *trust region*  $R \subseteq \mathbb{R}^n$  around  $x_k$ . Usually  $R$  is the ball defined by  $\|x - x_k\| \leq \Delta$  where  $\Delta > 0$  is *trust region radius*.

and finds  $x_{k+1}$  solving

$$\min_{x \in R} m_k(x)$$

## Trust Region

To compute  $x_{k+1}$ , a trust region algorithm chooses

- ▶ *model function*  $m_k$  whose behavior near  $x_k$  is similar to  $f$
- ▶ a *trust region*  $R \subseteq \mathbb{R}^n$  around  $x_k$ . Usually  $R$  is the ball defined by  $\|x - x_k\| \leq \Delta$  where  $\Delta > 0$  is *trust region radius*.

and finds  $x_{k+1}$  solving

$$\min_{x \in R} m_k(x)$$

If the solution does not sufficiently decrease  $f$ , we shrink the trust region and re-solve.

# Trust Region

To compute  $x_{k+1}$ , a trust region algorithm chooses

- ▶ *model function*  $m_k$  whose behavior near  $x_k$  is similar to  $f$
- ▶ a *trust region*  $R \subseteq \mathbb{R}^n$  around  $x_k$ . Usually  $R$  is the ball defined by  $\|x - x_k\| \leq \Delta$  where  $\Delta > 0$  is *trust region radius*.

and finds  $x_{k+1}$  solving

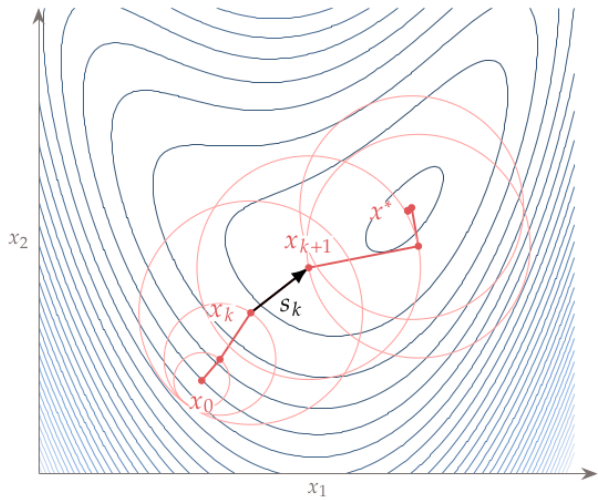
$$\min_{x \in R} m_k(x)$$

If the solution does not sufficiently decrease  $f$ , we shrink the trust region and re-solve.

The model  $m_k$  is usually derived from the Taylor's theorem.

$$m_k(x_k + p) = f_k + p^T \nabla f_k + \frac{1}{2} p^T B_k p$$

Where  $B_k$  approximates the Hessian of  $f$  at  $x_k$ .





# Line Search Methods

# Line Search

For setting the step size, we consider

- ▶ Armijo condition and backtracking algorithm
- ▶ strong Wolfe conditions and bracketing & zooming

# Line Search

For setting the step size, we consider

- ▶ Armijo condition and backtracking algorithm
- ▶ strong Wolfe conditions and bracketing & zooming

For setting the direction, we consider

- ▶ Gradient descent
- ▶ Newton's method
- ▶ quasi-Newton methods (BFGS)
- ▶ (Conjugate gradients)

We start with the step size.

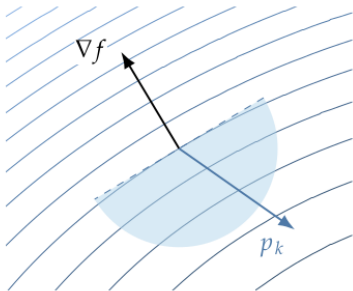
## Step Size

Assume

$$x_{k+1} = x_k + \alpha_k p_k$$

Where  $p_k$  is a descent direction

$$p_k^\top \nabla f_k < 0$$



# Step Size

Assume

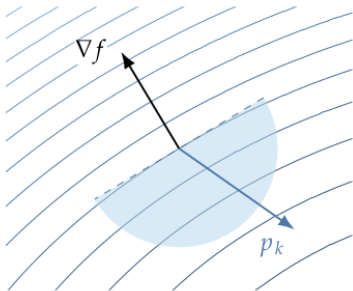
$$x_{k+1} = x_k + \alpha_k p_k$$

Where  $p_k$  is a descent direction

$$p_k^\top \nabla f_k < 0$$

Define

$$\phi(\alpha) = f(x_k + \alpha p_k)$$



## Step Size

Assume

$$x_{k+1} = x_k + \alpha_k p_k$$

Where  $p_k$  is a descent direction

$$p_k^\top \nabla f_k < 0$$

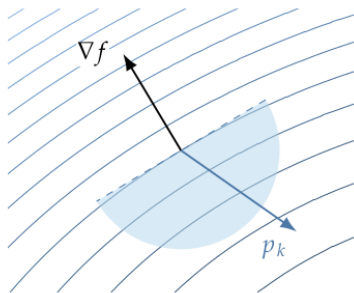
Define

$$\phi(\alpha) = f(x_k + \alpha p_k)$$

We know that

$$\phi'(\alpha) = \nabla f(x_k + \alpha p_k)^\top p_k \quad \text{which means} \quad \phi'(0) = \nabla f_k^\top p_k$$

Note that  $\phi'(0)$  must be negative as  $p_k$  is a descent direction.

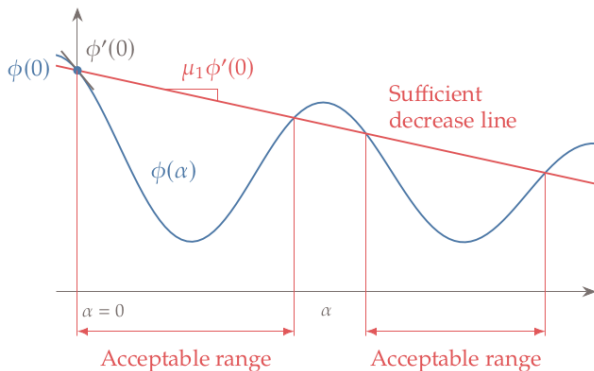


# Armijo Condition

The *sufficient decrease condition* (aka *Armijo condition*)

$$\phi(\alpha) \leq \phi(0) + \alpha (\mu_1 \phi'(0))$$

where  $\mu_1$  is a constant such that  $0 < \mu_1 \leq 1$



In practice,  $\mu_1$  is several orders smaller than 1, typically  $\mu_1 = 10^{-4}$ .

# Backtracking Line Search Algorithm

---

**Algorithm 1** Backtracking Line Search

---

**Input:**  $\alpha_{\text{init}} > 0$ ,  $0 < \mu_1 < 1$ ,  $0 < \rho < 1$

**Output:**  $\alpha^*$  satisfying sufficient decrease condition

- 1:  $\alpha \leftarrow \alpha_{\text{init}}$
  - 2: **while**  $\phi(\alpha) > \phi(0) + \alpha\mu_1\phi'(0)$  **do**
  - 3:      $\alpha \leftarrow \rho\alpha$
  - 4: **end while**
-



# Backtracking Line Search Algorithm

---

**Algorithm 2** Backtracking Line Search

---

**Input:**  $\alpha_{\text{init}} > 0$ ,  $0 < \mu_1 < 1$ ,  $0 < \rho < 1$

**Output:**  $\alpha^*$  satisfying sufficient decrease condition

- 1:  $\alpha \leftarrow \alpha_{\text{init}}$
  - 2: **while**  $\phi(\alpha) > \phi(0) + \alpha\mu_1\phi'(0)$  **do**
  - 3:      $\alpha \leftarrow \rho\alpha$
  - 4: **end while**
- 

The parameter  $\rho$  is typically set to 0.5. It can also be a variable set by a more sophisticated method (interpolation).

The  $\alpha_{\text{init}}$  depends on the method for setting the descent direction  $p_k$ . For Newton and quasi-Newton, it is 1.0, but for other methods, it might be different.

## Issues with Backtracking

There are two scenarios where the method does not perform well:

# Issues with Backtracking

There are two scenarios where the method does not perform well:

- ▶ The guess for the initial step is far too large, and the step sizes that satisfy sufficient decrease are smaller than the starting step by several orders of magnitude. Depending on the value of  $\rho$ , this scenario requires many backtracking evaluations.

## Issues with Backtracking

There are two scenarios where the method does not perform well:

- ▶ The guess for the initial step is far too large, and the step sizes that satisfy sufficient decrease are smaller than the starting step by several orders of magnitude. Depending on the value of  $\rho$ , this scenario requires many backtracking evaluations.
- ▶ The guess for the initial step immediately satisfies sufficient decrease. However, the function's slope is still highly negative, and we could have decreased the function value by much more if we had taken a more significant step. In this case, our guess for the initial step is far too small.

## Issues with Backtracking

There are two scenarios where the method does not perform well:

- ▶ The guess for the initial step is far too large, and the step sizes that satisfy sufficient decrease are smaller than the starting step by several orders of magnitude. Depending on the value of  $\rho$ , this scenario requires many backtracking evaluations.
- ▶ The guess for the initial step immediately satisfies sufficient decrease. However, the function's slope is still highly negative, and we could have decreased the function value by much more if we had taken a more significant step. In this case, our guess for the initial step is far too small.

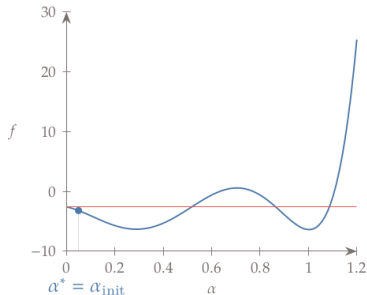
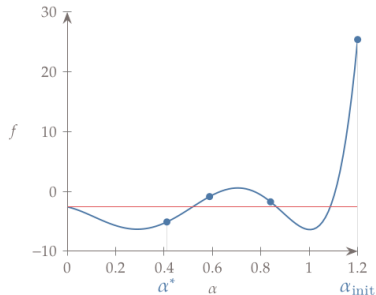
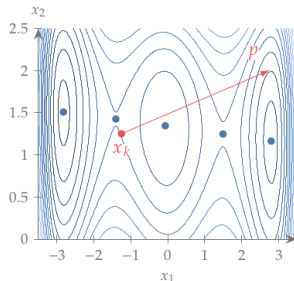
Even if our original step size is not too far from an acceptable one, the basic backtracking algorithm ignores any information we have about the function values and gradients. It blindly takes a reduced step based on a preselected ratio  $\rho$ .

# Backtracking Example

$$f(x_1, x_2) =$$

$$0.1x_1^6 - 1.5x_1^4 + 5x_1^2 \\ + 0.1x_2^4 + 3x_2^2 - 9x_2 + 0.5x_1x_2$$

$$\mu_1 = 10^{-4} \text{ and } \rho = 0.7.$$



## Sufficient Curvature Condition

We want to prevent too short of steps and to “motivate” the search to move closer to the minimum.

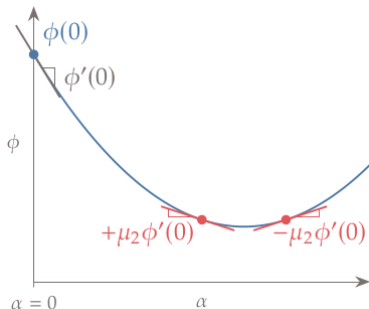
## Sufficient Curvature Condition

We want to prevent too short of steps and to “motivate” the search to move closer to the minimum.

We introduce the *sufficient curvature condition*

$$|\phi'(\alpha)| \leq \mu_2 |\phi'(0)|$$

where  $\mu_1 < \mu_2 < 1$  is a constant.





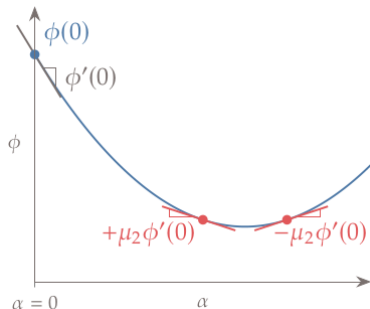
## Sufficient Curvature Condition

We want to prevent too short of steps and to “motivate” the search to move closer to the minimum.

We introduce the *sufficient curvature condition*

$$|\phi'(\alpha)| \leq \mu_2 |\phi'(0)|$$

where  $\mu_1 < \mu_2 < 1$  is a constant.



Typical values of  $\mu_2$  range from 0.1 to 0.9, depending on the direction setting method.

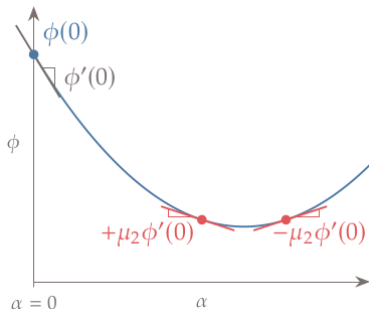
## Sufficient Curvature Condition

We want to prevent too short of steps and to “motivate” the search to move closer to the minimum.

We introduce the *sufficient curvature condition*

$$|\phi'(\alpha)| \leq \mu_2 |\phi'(0)|$$

where  $\mu_1 < \mu_2 < 1$  is a constant.



Typical values of  $\mu_2$  range from 0.1 to 0.9, depending on the direction setting method.

Note that moving  $\mu_2$  close to 0, the condition enforces  $\phi'(\alpha) \approx 0$ , which would yield an (almost) exact line search.

## Strong Wolfe Conditions

Putting together Armijo and sufficient curvature conditions, we obtain *strong Wolfe conditions*

## Strong Wolfe Conditions

Putting together Armijo and sufficient curvature conditions, we obtain *strong Wolfe conditions*

- *Sufficient decrease condition*

$$\phi(\alpha) \leq \phi(0) + \mu_1 \alpha \phi'(0)$$

# Strong Wolfe Conditions

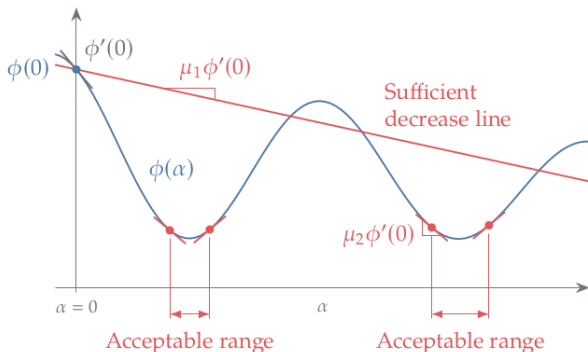
Putting together Armijo and sufficient curvature conditions, we obtain *strong Wolfe conditions*

- *Sufficient decrease condition*

$$\phi(\alpha) \leq \phi(0) + \mu_1 \alpha \phi'(0)$$

- *Sufficient curvature condition*

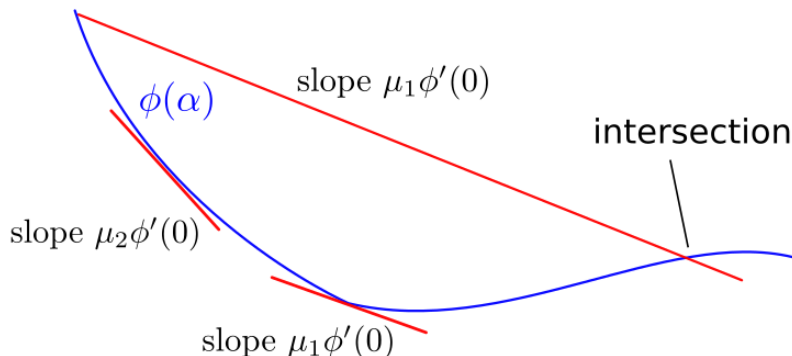
$$|\phi'(\alpha)| \leq \mu_2 |\phi'(0)|$$



# Satisfiability of Strong Wolfe Conditions

## Theorem 9

Suppose  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is continuously differentiable. Let  $p_k$  be a descent direction at  $x_k$ , and assume that  $f$  is bounded below along the ray  $\{x_k + \alpha p_k \mid \alpha > 0\}$ . Then, if  $0 < \mu_1 < \mu_2 < 1$ , step length intervals exist that satisfy the strong Wolfe conditions.



## Convergence of Line Search

Denote by  $\theta_k$  the angle between  $p_k$  and  $-\nabla f_k$ , i.e., satisfying

$$\cos \theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \|p_k\|}$$

## Convergence of Line Search

Denote by  $\theta_k$  the angle between  $p_k$  and  $-\nabla f_k$ , i.e., satisfying

$$\cos \theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \|p_k\|}$$

Recall that  $f$  is  $L$ -smooth for some  $L > 0$  if

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \quad \text{for all } x, \tilde{x} \in \mathbb{R}^n$$



## Convergence of Line Search

Denote by  $\theta_k$  the angle between  $p_k$  and  $-\nabla f_k$ , i.e., satisfying

$$\cos \theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \|p_k\|}$$

Recall that  $f$  is  $L$ -smooth for some  $L > 0$  if

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \quad \text{for all } x, \tilde{x} \in \mathbb{R}^n$$

### Theorem 10 (Zoutendijk)

Consider  $x_{k+1} = x_k + \alpha_k p_k$ , where  $p_k$  is a descent direction and  $\alpha_k$  satisfies the strong Wolfe conditions. Suppose that  $f$  is bounded below, continuously differentiable, and  $L$ -smooth. Then

$$\sum_{k \geq 0} \cos^2 \theta_k \|\nabla f_k\|^2 < \infty.$$

## Line Search Algorithm

How can we find a step size that satisfies *strong* Wolfe conditions?

# Line Search Algorithm

How can we find a step size that satisfies *strong* Wolfe conditions?

Use a bracketing and zoom algorithm, which proceeds in the following two phases:

# Line Search Algorithm

How can we find a step size that satisfies *strong* Wolfe conditions?

Use a bracketing and zoom algorithm, which proceeds in the following two phases:

1. The bracketing phase finds an interval within which we are certain to find a point that satisfies the strong Wolfe conditions.

# Line Search Algorithm

How can we find a step size that satisfies *strong* Wolfe conditions?

Use a bracketing and zoom algorithm, which proceeds in the following two phases:

1. The bracketing phase finds an interval within which we are certain to find a point that satisfies the strong Wolfe conditions.
2. The zooming phase finds a point that satisfies the strong Wolfe conditions within the interval provided by the bracketing phase.

---

### Algorithm 3 Bracketing

---

**Input:**  $\alpha_1 > 0$  and  $\alpha_{\max}$

- 1: Set  $\alpha_0 \leftarrow 0$
- 2:  $i \leftarrow 1$
- 3: **repeat**
- 4:     Evaluate  $\phi(\alpha_i)$
- 5:     **if**  $\phi(\alpha_i) > \phi(0) + \alpha_i \mu_1 \phi'(0)$  or  $[\phi(\alpha_i) \geq \phi(\alpha_{i-1})$  and  $i > 1]$   
      **then**
- 6:          $\alpha^* \leftarrow \mathbf{zoom}(\alpha_{i-1}, \alpha_i)$  and stop
- 7:     **end if**
- 8:     Evaluate  $\phi'(\alpha_i)$
- 9:     **if**  $|\phi'(\alpha_i)| \leq \mu_2 |\phi'(0)|$  **then**
- 10:        set  $\alpha^* \leftarrow \alpha_i$  and stop
- 11:     **else if**  $\phi'(\alpha_i) \geq 0$  **then**
- 12:        set  $\alpha^* \leftarrow \mathbf{zoom}(\alpha_i, \alpha_{i-1})$  and stop
- 13:     **end if**
- 14:     Choose  $\alpha_{i+1} \in (\alpha_i, \alpha_{\max})$
- 15:      $i \leftarrow i + 1$
- 16: **until** a condition is met

## Explanation of Bracketing

Note that the sequence of trial steps  $\alpha_j$  is monotonically increasing.

# Explanation of Bracketing

Note that the sequence of trial steps  $\alpha_i$  is monotonically increasing.

Note that **zoom** is called when one of the following conditions is satisfied:

- ▶  $\alpha_i$  violates the sufficient decrease condition (lines 5 and 6)
- ▶  $\phi(\alpha_i) \geq \phi(\alpha_{i-1})$  (also lines 5 and 6)
- ▶  $\phi'(\alpha_i) \geq 0$  (lines 11 and 12)

The last step increases the  $\alpha_i$ . May use, e.g., a constant multiple.



## Zoom

The following algorithm keeps two step lengths:  $\alpha_{lo}$  and  $\alpha_{hi}$

# Zoom

The following algorithm keeps two step lengths:  $\alpha_{lo}$  and  $\alpha_{hi}$

The following invariants are being preserved:

- ▶ The interval bounded by  $\alpha_{lo}$  and  $\alpha_{hi}$  always contains one or more intervals satisfying the strong Wolfe conditions.

Note that we *do not* assume  $\alpha_{lo} \leq \alpha_{hi}$

The following algorithm keeps two step lengths:  $\alpha_{lo}$  and  $\alpha_{hi}$

The following invariants are being preserved:

- ▶ The interval bounded by  $\alpha_{lo}$  and  $\alpha_{hi}$  always contains one or more intervals satisfying the strong Wolfe conditions.

Note that we *do not* assume  $\alpha_{lo} \leq \alpha_{hi}$

- ▶  $\alpha_{lo}$  is, among all step lengths generated so far and satisfying the sufficient decrease condition, the one giving the smallest value of  $\phi$ ,

The following algorithm keeps two step lengths:  $\alpha_{lo}$  and  $\alpha_{hi}$

The following invariants are being preserved:

- ▶ The interval bounded by  $\alpha_{lo}$  and  $\alpha_{hi}$  always contains one or more intervals satisfying the strong Wolfe conditions.

Note that we *do not* assume  $\alpha_{lo} \leq \alpha_{hi}$

- ▶  $\alpha_{lo}$  is, among all step lengths generated so far and satisfying the sufficient decrease condition, the one giving the smallest value of  $\phi$ ,
- ▶  $\alpha_{hi}$  is chosen so that  $\phi'(\alpha_{lo})(\alpha_{hi} - \alpha_{lo}) < 0$ .

That is,  $\phi$  always slopes down from  $\alpha_{lo}$  to  $\alpha_{hi}$ .

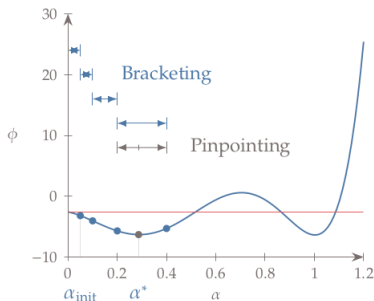
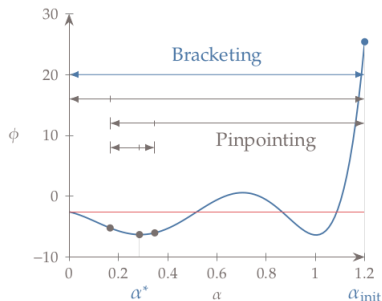
```

1: function ZOOM( $\alpha_{lo}, \alpha_{hi}$ )
2:   repeat
3:     Set  $\alpha$  between  $\alpha_{lo}$  and  $\alpha_{hi}$  using interpolation
      (bisection, quadratic, etc.)
4:     Evaluate  $\phi(\alpha)$ 
5:     if  $\phi(\alpha) > \phi(0) + \alpha\mu_1\phi'(0)$  or  $\phi(\alpha) \geq \phi(\alpha_{lo})$  then
6:        $\alpha_{hi} \leftarrow \alpha$ 
7:     else
8:       Evaluate  $\phi'(\alpha)$ 
9:       if  $|\phi'(\alpha)| \leq \mu_2|\phi'(0)|$  then
10:        Set  $\alpha^* \leftarrow \alpha$  and stop
11:      end if
12:      if  $\phi'(\alpha)(\alpha_{hi} - \alpha_{lo}) \geq 0$  then
13:         $\alpha_{hi} \leftarrow \alpha_{lo}$ 
14:      end if
15:       $\alpha_{lo} \leftarrow \alpha$ 
16:    end if
17:  until a condition is met
18: end function

```

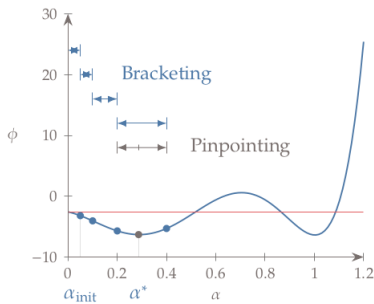
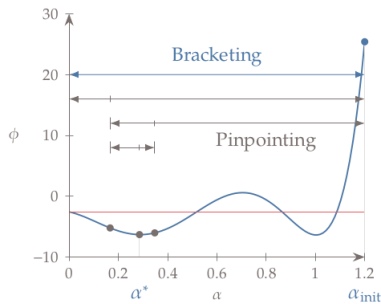
# Bracketing & Zooming Example

We use quadratic interpolation; the bracketing chooses  $\alpha_{i+1} = 2\alpha_i$ , and the sufficient curvature factor is  $\mu_2 = 0.9$ .



## Bracketing & Zooming Example

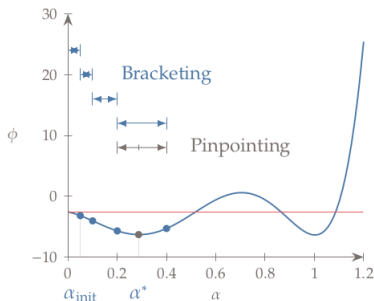
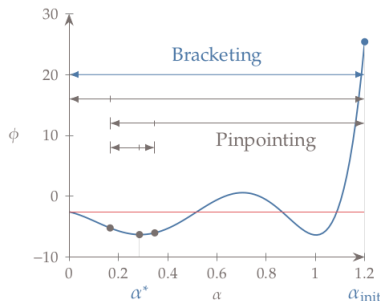
We use quadratic interpolation; the bracketing chooses  $\alpha_{i+1} = 2\alpha_i$ , and the sufficient curvature factor is  $\mu_2 = 0.9$ .



Bracketing is achieved in the first iteration by using a significant initial step of  $\alpha_{\text{init}} = 1.2$  (left). Then, zooming finds an improved point through interpolation.

## Bracketing & Zooming Example

We use quadratic interpolation; the bracketing chooses  $\alpha_{i+1} = 2\alpha_i$ , and the sufficient curvature factor is  $\mu_2 = 0.9$ .



Bracketing is achieved in the first iteration by using a significant initial step of  $\alpha_{\text{init}} = 1.2$  (left). Then, zooming finds an improved point through interpolation.

The small initial step of  $\alpha_{\text{init}} = 0.05$  (right) does not satisfy the strong Wolfe conditions, and the bracketing phase moves forward toward a flatter part of the function.



## Comments on Line Search

- ▶ The interpolation of the zoom phase that determines  $\alpha$  should be safeguarded to ensure that the new step length is not too close to the endpoints of the interval.

## Comments on Line Search

- ▶ The interpolation of the zoom phase that determines  $\alpha$  should be safeguarded to ensure that the new step length is not too close to the endpoints of the interval.
- ▶ Practical line search algorithms also use the interpolating polynomials' properties to make educated guesses of where the next step length should lie.

## Comments on Line Search

- ▶ The interpolation of the zoom phase that determines  $\alpha$  should be safeguarded to ensure that the new step length is not too close to the endpoints of the interval.
- ▶ Practical line search algorithms also use the interpolating polynomials' properties to make educated guesses of where the next step length should lie.
- ▶ A problem that can arise in the implementation is that as the optimization algorithm approaches the solution, two consecutive function values  $f(x_k)$  and  $f(x_{k-1})$  may be indistinguishable in finite-precision arithmetic.

## Comments on Line Search

- ▶ The interpolation of the zoom phase that determines  $\alpha$  should be safeguarded to ensure that the new step length is not too close to the endpoints of the interval.
- ▶ Practical line search algorithms also use the interpolating polynomials' properties to make educated guesses of where the next step length should lie.
- ▶ A problem that can arise in the implementation is that as the optimization algorithm approaches the solution, two consecutive function values  $f(x_k)$  and  $f(x_{k-1})$  may be indistinguishable in finite-precision arithmetic.
- ▶ Some procedures also stop if the relative change in  $x$  is close to machine accuracy or some user-specified threshold.

## Comments on Line Search

- ▶ The interpolation of the zoom phase that determines  $\alpha$  should be safeguarded to ensure that the new step length is not too close to the endpoints of the interval.
- ▶ Practical line search algorithms also use the interpolating polynomials' properties to make educated guesses of where the next step length should lie.
- ▶ A problem that can arise in the implementation is that as the optimization algorithm approaches the solution, two consecutive function values  $f(x_k)$  and  $f(x_{k-1})$  may be indistinguishable in finite-precision arithmetic.
- ▶ Some procedures also stop if the relative change in  $x$  is close to machine accuracy or some user-specified threshold.
- ▶ The presented algorithm is implemented in [https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.line\\_search.html](https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.line_search.html)

# Unconstrained Optimization Algorithms

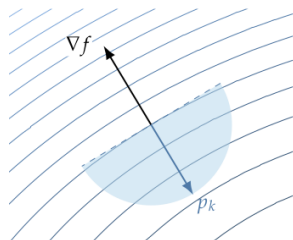
Descent Direction

First-Order Methods

# Gradient Descent

Consider the *gradient descent* method where

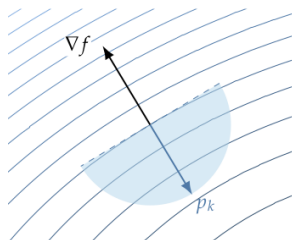
$$x_{k+1} = x_k + \alpha_k p_k \quad p_k = -\nabla f(x_k)$$



# Gradient Descent

Consider the *gradient descent* method where

$$x_{k+1} = x_k + \alpha_k p_k \quad p_k = -\nabla f(x_k)$$



Unfortunately, the gradient does not possess much information about the step size.

So usually, a normalized gradient is used to obtain the direction, and then a line search is performed:

$$x_{k+1} = x_k + \alpha_k p_k \quad p_k = -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}$$

The line search is *exact* if  $\alpha_k$  minimizes  $f(x_k + \alpha_k p_k)$ . Not practical, we usually find  $\alpha_k$  satisfying the strong Wolfe conditions.



# Gradient Descent Algorithm with Line Search

---

**Algorithm 4** Gradient Descent with Line Search

---

**Input:**  $x_0$  starting point,  $\varepsilon > 0$

**Output:**  $x^*$  approximation to a stationary point

```
1:  $k \leftarrow 0$ 
2: while  $\|\nabla f\|_\infty > \varepsilon$  do
3:    $p_k \leftarrow -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}$ 
4:   Set  $\alpha_{\text{init}}$  for line search
5:    $\alpha_k \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$ 
6:    $x_{k+1} \leftarrow x_k + \alpha_k p_k$ 
7:    $k \leftarrow k + 1$ 
8: end while
```

---

# Gradient Descent Algorithm with Line Search

---

**Algorithm 5** Gradient Descent with Line Search

---

**Input:**  $x_0$  starting point,  $\varepsilon > 0$

**Output:**  $x^*$  approximation to a stationary point

```
1:  $k \leftarrow 0$ 
2: while  $\|\nabla f\|_\infty > \varepsilon$  do
3:    $p_k \leftarrow -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}$ 
4:   Set  $\alpha_{\text{init}}$  for line search
5:    $\alpha_k \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$ 
6:    $x_{k+1} \leftarrow x_k + \alpha_k p_k$ 
7:    $k \leftarrow k + 1$ 
8: end while
```

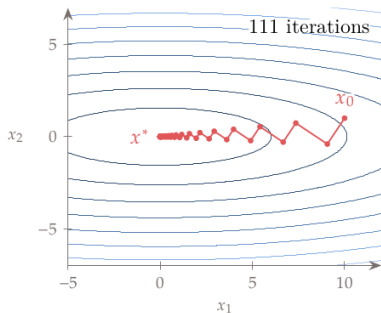
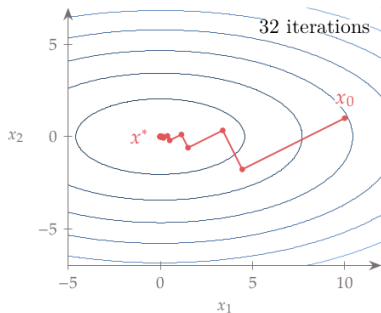
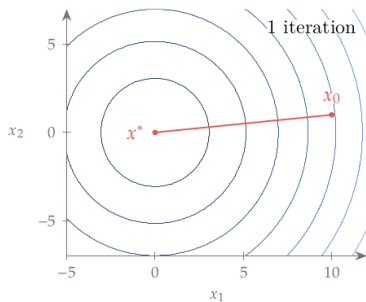
---

Here  $\alpha_{\text{init}}$  can be estimated from the previous step size  $\alpha_{k-1}$  by demanding similar decrease in the objective:

$$\alpha_{\text{init}} p_k^\top \nabla f_k \approx \alpha_{k-1} p_{k-1}^\top \nabla f_{k-1} \quad \Rightarrow \quad \alpha_{\text{init}} = \alpha_{k-1} \frac{p_{k-1}^\top \nabla f_{k-1}}{p_k^\top \nabla f_k}$$

$$f(x_1, x_2) = x_1^2 + \beta x_2^2$$

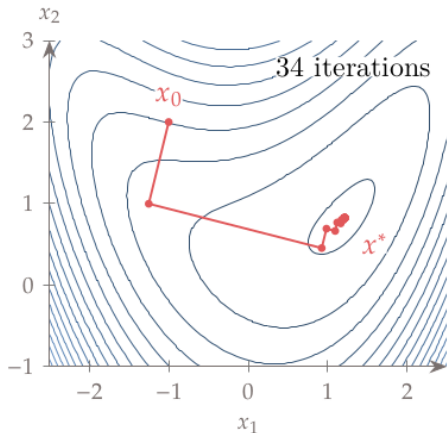
Consider  $\beta = 1, 5, 15$  and  
*exact* line search



Note that  $p_{k+1}$  and  $p_k$  are always orthogonal.

$$f(x_1, x_2) = (1 - x_1)^2 + (1 - x_2)^2 + \frac{1}{2} (2x_2 - x_1^2)^2$$

Stopping:  $\|\nabla f\|_\infty \leq 10^{-6}$ .



## Global Convergence with Line Search

Recall the Zoutendijk's theorem.

Denote by  $\theta_k$  the angle between  $p_k$  and  $-\nabla f_k$ , i.e., satisfying

$$\cos \theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \|p_k\|}$$

Recall that  $f$  is  $L$ -smooth for some  $L > 0$  if

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \quad \text{for all } x, \tilde{x} \in \mathbb{R}^n$$

### Theorem 11 (Zoutendijk)

Consider  $x_{k+1} = x_k + \alpha_k p_k$ , where  $p_k$  is a descent direction and  $\alpha_k$  satisfies the strong Wolfe conditions. Suppose that  $f$  is bounded below, continuously differentiable, and  $L$ -smooth. Then

$$\sum_{k \geq 0} \cos^2 \theta_k \|\nabla f_k\|^2 < \infty.$$

# Global Convergence of Gradient Descent

Assume that each  $\alpha_k$  satisfies strong Wolfe conditions.

# Global Convergence of Gradient Descent

Assume that each  $\alpha_k$  satisfies strong Wolfe conditions.

Note that the angle  $\theta_k$  between  $p_k = -\nabla f_k$  and the negative gradient  $-\nabla f_k$  equals 0. Hence,  $\cos \theta_k = 1$ .

# Global Convergence of Gradient Descent

Assume that each  $\alpha_k$  satisfies strong Wolfe conditions.

Note that the angle  $\theta_k$  between  $p_k = -\nabla f_k$  and the negative gradient  $-\nabla f_k$  equals 0. Hence,  $\cos \theta_k = 1$ .

Thus, under the assumptions of Zoutendijk's theorem, we obtain

$$\sum_{k \geq 0} \cos^2 \theta_k \|\nabla f_k\|^2 = \sum_{k \geq 0} \|\nabla f_k\|^2 < \infty$$

which implies that  $\lim_{k \rightarrow \infty} \|\nabla f_k\| = 0$ .



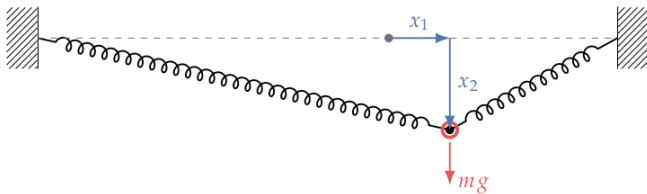
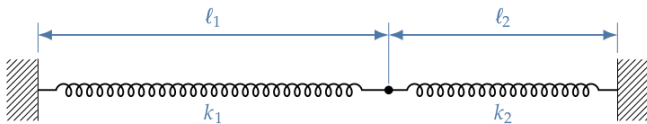
# Local Linear Convergence of Gradient Descent (Optional)

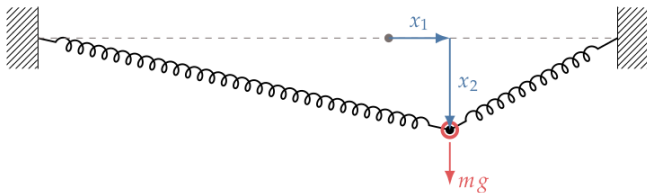
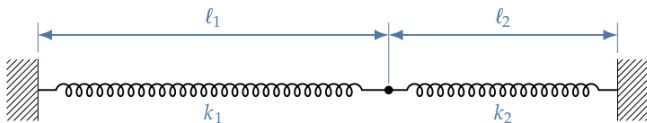
## Theorem 12

*Suppose that  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is twice continuously differentiable, that the line search is exact, and that the descent converges to  $x^*$  where  $\nabla f(x^*) = 0$  and the Hessian matrix  $\nabla^2 f(x^*)$  is positive definite. Then*

$$f(x_{k+1}) - f(x^*) \leq \left( \frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1} \right)^2 [f(x_k) - f(x^*)],$$

*where  $\lambda_1 \leq \dots \leq \lambda_n$  are the eigenvalues of  $\nabla^2 f(x^*)$ .*

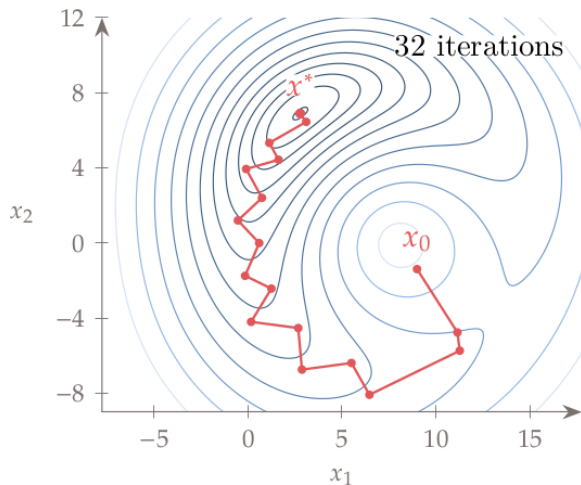




$$f(x_1, x_2) = \frac{1}{2}k_1 \left( \sqrt{(\ell_1 + x_1)^2 + x_2^2} - \ell_1 \right)^2 + \frac{1}{2}k_2 \left( \sqrt{(\ell_2 - x_1)^2 + x_2^2} - \ell_2 \right)^2 - mgx_2$$

Here  $\ell_1 = 12, \ell_2 = 8, k_1 = 1, k_2 = 10, mg = 7$

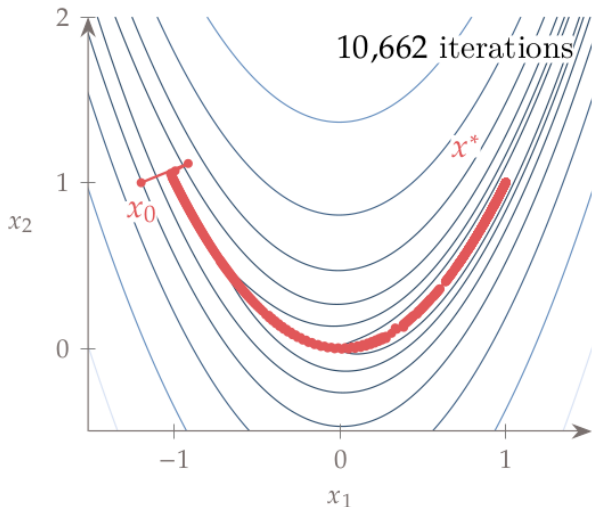
## Two Spring Problem - Gradient Descent



Gradient descent, line search, stop. cond.  $\|\nabla f\|_{\infty} \leq 10^{-6}$ .

## Rosenbrock Function - Gradient Descent

Rosenbrock:  $f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$



Gradient descent, line search, stop. cond.  $\|\nabla f\|_{\infty} \leq 10^{-6}$ .

## Comments on Gradient Descent

- ▶ The method needs evaluation of  $\nabla f$  at each  $x_k$ . If  $f$  is not differentiable at  $x_k$ , subgradients can be considered (out of the scope of this course).

## Comments on Gradient Descent

- ▶ The method needs evaluation of  $\nabla f$  at each  $x_k$ . If  $f$  is not differentiable at  $x_k$ , subgradients can be considered (out of the scope of this course).
- ▶ Slow, zig-zagging, provides insufficient information for line search initialization.

# Comments on Gradient Descent

- ▶ The method needs evaluation of  $\nabla f$  at each  $x_k$ . If  $f$  is not differentiable at  $x_k$ , subgradients can be considered (out of the scope of this course).
- ▶ Slow, zig-zagging, provides insufficient information for line search initialization.
- ▶ Susceptible to scaling of variables (see the paraboloid example).



# Comments on Gradient Descent

- ▶ The method needs evaluation of  $\nabla f$  at each  $x_k$ . If  $f$  is not differentiable at  $x_k$ , subgradients can be considered (out of the scope of this course).
- ▶ Slow, zig-zagging, provides insufficient information for line search initialization.
- ▶ Susceptible to scaling of variables (see the paraboloid example).
- ▶ THE basis for algorithms training neural networks - a huge amount of specific adjustments are developed for working with huge numbers of variables in neural networks (trillions of weights).

# Unconstrained Optimization Algorithms

Descent Direction

Second-Order Methods

## Newton's Method

Consider an objective  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ .

Assume that  $f$  is twice differentiable.

## Newton's Method

Consider an objective  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ .

Assume that  $f$  is twice differentiable.

Then, by the Taylor's theorem,

$$f(x_k + p) \approx f_k + \nabla f_k^\top p + \frac{1}{2} p^\top H_k p$$

Here we denote the gradient  $\nabla f(x_k)$  of  $f$  at  $x_k$  by  $\nabla f_k$  and the Hessian  $\nabla^2 f(x_k)$  by  $H_k$ .

## Newton's Method

Consider an objective  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ .

Assume that  $f$  is twice differentiable.

Then, by the Taylor's theorem,

$$f(x_k + p) \approx f_k + \nabla f_k^\top p + \frac{1}{2} p^\top H_k p$$

Here we denote the gradient  $\nabla f(x_k)$  of  $f$  at  $x_k$  by  $\nabla f_k$  and the Hessian  $\nabla^2 f(x_k)$  by  $H_k$ .

Define

$$q(p) = f_k + \nabla f_k^\top p + \frac{1}{2} p^\top H_k p$$

and minimize  $q$  w.r.t.  $p$  by setting  $\nabla q(p) = 0$ .

## Newton's Method

Consider an objective  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ .

Assume that  $f$  is twice differentiable.

Then, by the Taylor's theorem,

$$f(x_k + p) \approx f_k + \nabla f_k^\top p + \frac{1}{2} p^\top H_k p$$

Here we denote the gradient  $\nabla f(x_k)$  of  $f$  at  $x_k$  by  $\nabla f_k$  and the Hessian  $\nabla^2 f(x_k)$  by  $H_k$ .

Define

$$q(p) = f_k + \nabla f_k^\top p + \frac{1}{2} p^\top H_k p$$

and minimize  $q$  w.r.t.  $p$  by setting  $\nabla q(p) = 0$ . We obtain:

$$H_k p = -\nabla f_k$$

Denote by  $p_k$  the solution, and set  $x_{k+1} = x_k + p_k$ .

# Newton's Method

---

**Algorithm 6** Newton's Method

---

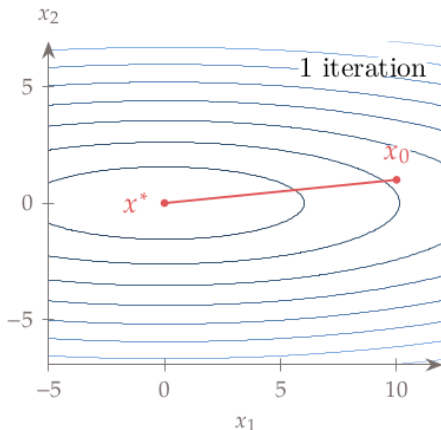
**Input:**  $x_0$  starting point,  $\tau > 0$

**Output:**  $x^*$  approximation to a stationary point

- 1:  $k \leftarrow 0$
  - 2: **while**  $\|\nabla f_k\|_\infty > \tau$  **do**
  - 3:     Compute  $\nabla f_k = \nabla f(x_k)$
  - 4:     Solve  $H_k p_k = -\nabla f_k$  for  $p_k$
  - 5:      $x_{k+1} \leftarrow x_k + p_k$
  - 6:      $k \leftarrow k + 1$
  - 7: **end while**
-

## Newton's Method - Example

Newton's method finds the minimum of a quadratic function in a single step.

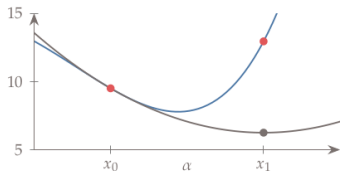
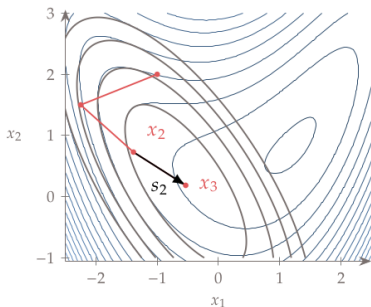
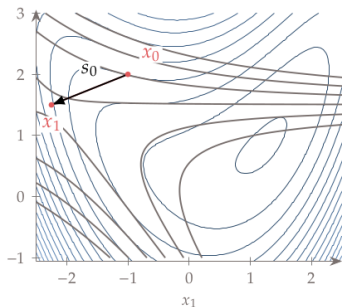


Note that the Newton's method is scale-invariant!

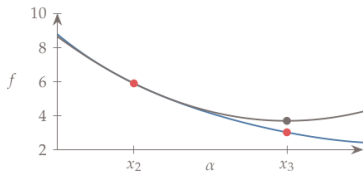


$$f(x_1, x_2) = (1 - x_1)^2 + (1 - x_2)^2 + \frac{1}{2} (2x_2 - x_1^2)^2$$

Stopping:  $\|\nabla f\|_\infty \leq 10^{-6}$ .



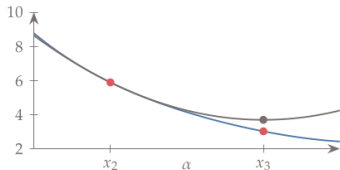
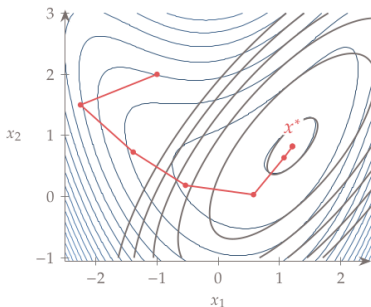
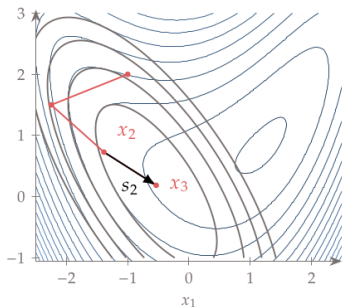
$k = 0$



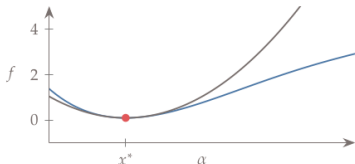
$k = 2$

$$f(x_1, x_2) = (1 - x_1)^2 + (1 - x_2)^2 + \frac{1}{2} (2x_2 - x_1^2)^2$$

Stopping:  $\|\nabla f\|_\infty \leq 10^{-6}$ .

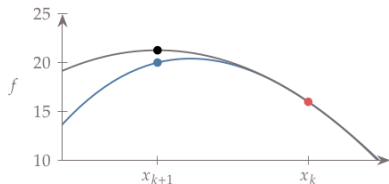
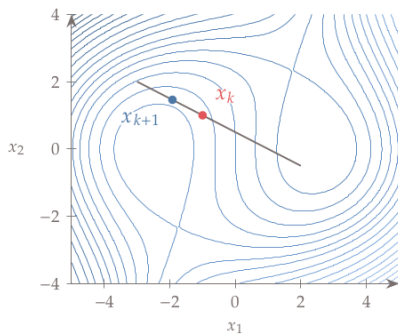


$k = 2$

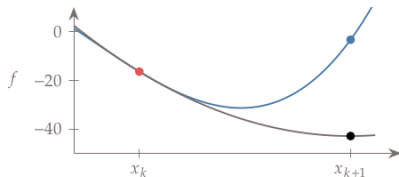
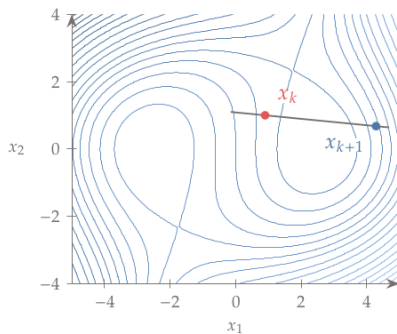


$k = 8$

# Convergence Issues



Negative curvature



Overshoot

Also, the computation of the Hessian is costly.

# Quadratic Convergence of Newton's Method (Optional)

## Theorem 13

*Assume  $f$  is twice differentiable and assume that  $\nabla f$  is  $L$ -smooth. Let  $x_*$  be a minimizer of  $f(x)$  and assume that  $\nabla^2 f(x_*)$  is positive definite.*

*If  $\|x_0 - x_*\|$  is sufficiently small, then  $\{x_k\}$  converges quadratically to  $x_*$ .*

# Quadratic Convergence of Newton's Method (Optional)

## Theorem 13

*Assume  $f$  is twice differentiable and assume that  $\nabla f$  is  $L$ -smooth. Let  $x_*$  be a minimizer of  $f(x)$  and assume that  $\nabla^2 f(x_*)$  is positive definite.*

*If  $\|x_0 - x_*\|$  is sufficiently small, then  $\{x_k\}$  converges quadratically to  $x_*$ .*

Note that the theorem implicitly assumes that  $\nabla^2 f(x_k)$  is nonsingular for every  $k$ .

# Quadratic Convergence of Newton's Method (Optional)

## Theorem 13

*Assume  $f$  is twice differentiable and assume that  $\nabla f$  is  $L$ -smooth. Let  $x_*$  be a minimizer of  $f(x)$  and assume that  $\nabla^2 f(x_*)$  is positive definite.*

*If  $\|x_0 - x_*\|$  is sufficiently small, then  $\{x_k\}$  converges quadratically to  $x_*$ .*

Note that the theorem implicitly assumes that  $\nabla^2 f(x_k)$  is nonsingular for every  $k$ .

As the theorem is concerned only with  $x_k$  approaching  $x^*$ , the continuity of  $\nabla^2 f(x_k)$  and positive definiteness of  $\nabla^2 f(x^*)$  imply that  $\nabla^2 f(x_k)$  is positive definite for all sufficiently large  $k$ .

# Quadratic Convergence of Newton's Method (Optional)

## Theorem 13

*Assume  $f$  is twice differentiable and assume that  $\nabla f$  is  $L$ -smooth. Let  $x_*$  be a minimizer of  $f(x)$  and assume that  $\nabla^2 f(x_*)$  is positive definite.*

*If  $\|x_0 - x_*\|$  is sufficiently small, then  $\{x_k\}$  converges quadratically to  $x_*$ .*

Note that the theorem implicitly assumes that  $\nabla^2 f(x_k)$  is nonsingular for every  $k$ .

As the theorem is concerned only with  $x_k$  approaching  $x^*$ , the continuity of  $\nabla^2 f(x_k)$  and positive definiteness of  $\nabla^2 f(x^*)$  imply that  $\nabla^2 f(x_k)$  is positive definite for all sufficiently large  $k$ .

However, what happens if we start far away from a minimizer?

# Newton's Method with Line Search

---

**Algorithm 7** Newton's Method with Line Search

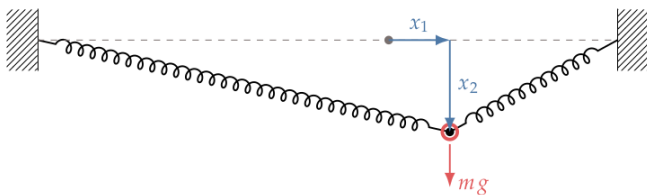
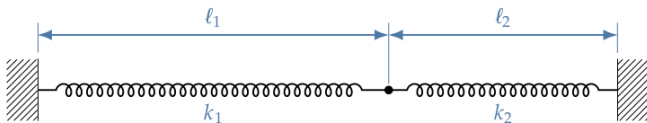
---

**Input:**  $x_0$  starting point,  $\varepsilon > 0$

**Output:**  $x^*$  approximation to a stationary point

- 1:  $k \leftarrow 0$
  - 2:  $\alpha_{\text{init}} \leftarrow 1$
  - 3: **while**  $\|\nabla f_k\|_\infty > \varepsilon$  **do**
  - 4:     Compute  $\nabla f_k = \nabla f(x_k)$
  - 5:     Solve  $H_k p_k = -\nabla f_k$  for  $p_k$
  - 6:      $\alpha \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$
  - 7:      $x_{k+1} \leftarrow x_k + \alpha p_k$
  - 8:      $k \leftarrow k + 1$
  - 9: **end while**
-

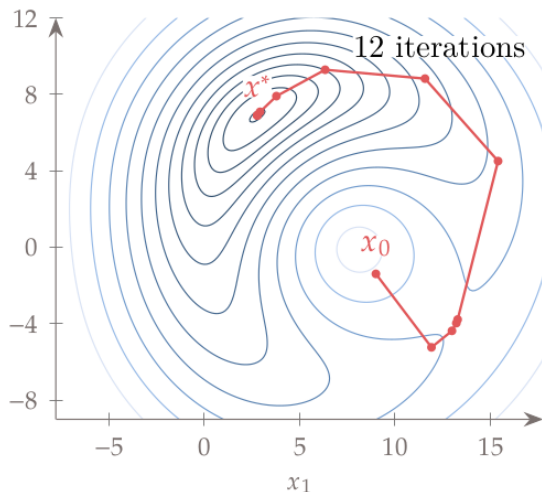




$$f(x_1, x_2) = \frac{1}{2}k_1 \left( \sqrt{(\ell_1 + x_1)^2 + x_2^2} - \ell_1 \right)^2 + \frac{1}{2}k_2 \left( \sqrt{(\ell_2 - x_1)^2 + x_2^2} - \ell_2 \right)^2 - mgx_2$$

Here  $\ell_1 = 12, \ell_2 = 8, k_1 = 1, k_2 = 10, mg = 7$

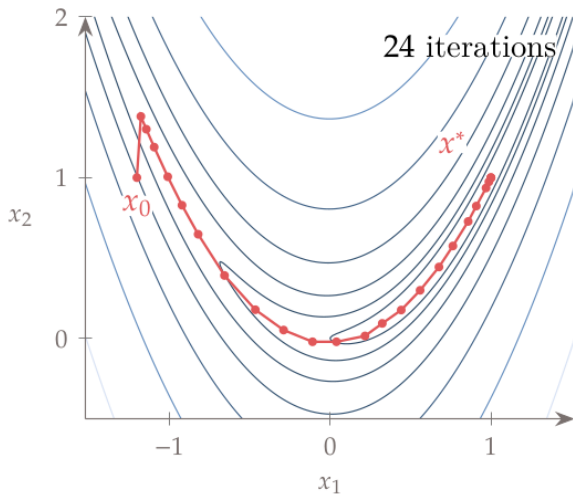
## Two Spring Problem - Newton's Method



Newton's method, line search and stop. cond.  $\|\nabla f\|_\infty \leq 10^{-6}$ .  
Compare this with 32 iterations of gradient descent.

## Rosenbrock Function - Newton's Method

Rosenbrock:  $f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$



Newton's method, line search, stop. cond.  $\|\nabla f\|_\infty \leq 10^{-6}$ .  
Compare this with 10,662 iterations of gradient descent.

## Global Convergence of Line Search

Denote by  $\theta_k$  the angle between  $p_k$  and  $-\nabla f_k$ , i.e., satisfying

$$\cos \theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \|p_k\|}$$

Recall that  $f$  is  $L$ -smooth for some  $L > 0$  if

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \quad \text{for all } x, \tilde{x} \in \mathbb{R}^n$$

### Theorem 14 (Zoutendijk)

Consider  $x_{k+1} = x_k + \alpha_k p_k$ , where  $p_k$  is a descent direction and  $\alpha_k$  satisfies the strong Wolfe conditions. Suppose that  $f$  is bounded below, continuously differentiable, and  $L$ -smooth. Then

$$\sum_{k \geq 0} \cos^2 \theta_k \|\nabla f_k\|^2 < \infty.$$

# Global Convergence of Newton's Method

Assume that all  $\alpha_k$  satisfy strong Wolfe conditions.

## Global Convergence of Newton's Method

Assume that all  $\alpha_k$  satisfy strong Wolfe conditions.

Assume that the Hessians  $H_k$  are positive definite with a uniformly bounded condition number:

$$\|H_k\| \|H_k^{-1}\| \leq M \quad \text{for all } k$$

## Global Convergence of Newton's Method

Assume that all  $\alpha_k$  satisfy strong Wolfe conditions.

Assume that the Hessians  $H_k$  are positive definite with a uniformly bounded condition number:

$$\|H_k\| \|H_k^{-1}\| \leq M \quad \text{for all } k$$

Then  $\theta_k$  between  $p_k = -H_k^{-1} \nabla f_k$  and  $-\nabla f_k$  satisfies

$$\cos \theta_k \geq 1/M$$

## Global Convergence of Newton's Method

Assume that all  $\alpha_k$  satisfy strong Wolfe conditions.

Assume that the Hessians  $H_k$  are positive definite with a uniformly bounded condition number:

$$\|H_k\| \|H_k^{-1}\| \leq M \quad \text{for all } k$$

Then  $\theta_k$  between  $p_k = -H_k^{-1}\nabla f_k$  and  $-\nabla f_k$  satisfies

$$\cos \theta_k \geq 1/M$$

Thus, under the assumptions of Zoutendijk's theorem, we obtain

$$\frac{1}{M^2} \sum_{k \geq 0} \|\nabla f_k\|^2 \leq \sum_{k \geq 0} \cos^2 \theta_k \|\nabla f_k\|^2 < \infty$$

which implies that  $\lim_{k \rightarrow \infty} \|\nabla f_k\| = 0$ .



## Global Convergence of Newton's Method

Assume that all  $\alpha_k$  satisfy strong Wolfe conditions.

Assume that the Hessians  $H_k$  are positive definite with a uniformly bounded condition number:

$$\|H_k\| \|H_k^{-1}\| \leq M \quad \text{for all } k$$

Then  $\theta_k$  between  $p_k = -H_k^{-1} \nabla f_k$  and  $-\nabla f_k$  satisfies

$$\cos \theta_k \geq 1/M$$

Thus, under the assumptions of Zoutendijk's theorem, we obtain

$$\frac{1}{M^2} \sum_{k \geq 0} \|\nabla f_k\|^2 \leq \sum_{k \geq 0} \cos^2 \theta_k \|\nabla f_k\|^2 < \infty$$

which implies that  $\lim_{k \rightarrow \infty} \|\nabla f_k\| = 0$ .

What if  $H_k$  is not positive definite or is (nearly) singular?

## Eigenvalue Modification

Consider  $H_k = \nabla^2 f(x_k)$  and consider its diagonal form:

$$H_k = QDQ^T$$

Where  $D$  contains the eigenvalues of  $H_k$  on the diagonal, i.e.,  $D = \text{diag}(\lambda_1, \dots, \lambda_n)$  and  $Q$  is an orthogonal matrix.

## Eigenvalue Modification

Consider  $H_k = \nabla^2 f(x_k)$  and consider its diagonal form:

$$H_k = QDQ^T$$

Where  $D$  contains the eigenvalues of  $H_k$  on the diagonal, i.e.,  $D = \text{diag}(\lambda_1, \dots, \lambda_n)$  and  $Q$  is an orthogonal matrix.

Observe that

- ▶  $H_k$  is not positive definite iff  $\lambda_i \leq 0$  for some  $i$
- ▶  $\|H_k\|$  grows with  $\max\{\lambda_1, \dots, \lambda_n\}$  going to infinity.
- ▶  $\|H_k^{-1}\|$  grows with  $\min\{\lambda_1, \dots, \lambda_n\}$  going to 0  
(i.e., the matrix becomes close to a singular matrix)

We want to prevent all three cases, i.e., make sure that for some reasonably large  $\delta > 0$  we have  $\lambda_i \geq \delta$  but not too large.

## Eigenvalue Modification

Consider  $H_k = \nabla^2 f(x_k)$  and consider its diagonal form:

$$H_k = QDQ^T$$

Where  $D$  contains the eigenvalues of  $H_k$  on the diagonal, i.e.,  $D = \text{diag}(\lambda_1, \dots, \lambda_n)$  and  $Q$  is an orthogonal matrix.

Observe that

- ▶  $H_k$  is not positive definite iff  $\lambda_i \leq 0$  for some  $i$
- ▶  $\|H_k\|$  grows with  $\max\{\lambda_1, \dots, \lambda_n\}$  going to infinity.
- ▶  $\|H_k^{-1}\|$  grows with  $\min\{\lambda_1, \dots, \lambda_n\}$  going to 0  
(i.e., the matrix becomes close to a singular matrix)

We want to prevent all three cases, i.e., make sure that for some reasonably large  $\delta > 0$  we have  $\lambda_i \geq \delta$  but not too large.

Two questions are in order:

- ▶ What is a reasonably large  $\delta$ ?
- ▶ How to modify  $H_k$  so the minimum is large enough?

## Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \quad \text{and} \quad \nabla^2 f(x_k) = \text{diag}(10, 3, -1)$$

## Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \quad \text{and} \quad \nabla^2 f(x_k) = \text{diag}(10, 3, -1)$$

Now, the diagonalization is trivial:

$$\nabla^2 f(x_k) = Q \text{diag}(10, 3, -1) Q^\top \quad Q = I \text{ is the identity matrix}$$

## Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \quad \text{and} \quad \nabla^2 f(x_k) = \text{diag}(10, 3, -1)$$

Now, the diagonalization is trivial:

$$\nabla^2 f(x_k) = Q \text{diag}(10, 3, -1) Q^\top \quad Q = I \text{ is the identity matrix}$$

What if we consider a minimum modification replacing the negative eigenvalue with a small number, say  $\delta = 10^{-8}$ ?

## Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \quad \text{and} \quad \nabla^2 f(x_k) = \text{diag}(10, 3, -1)$$

Now, the diagonalization is trivial:

$$\nabla^2 f(x_k) = Q \text{diag}(10, 3, -1) Q^\top \quad Q = I \text{ is the identity matrix}$$

What if we consider a minimum modification replacing the negative eigenvalue with a small number, say  $\delta = 10^{-8}$ ? Obtain

$$B_k = Q \text{diag}(10, 3, 10^{-8}) Q^\top = \text{diag}(10, 3, 10^{-8})$$



## Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \quad \text{and} \quad \nabla^2 f(x_k) = \text{diag}(10, 3, -1)$$

Now, the diagonalization is trivial:

$$\nabla^2 f(x_k) = Q \text{diag}(10, 3, -1) Q^\top \quad Q = I \text{ is the identity matrix}$$

What if we consider a minimum modification replacing the negative eigenvalue with a small number, say  $\delta = 10^{-8}$ ? Obtain

$$B_k = Q \text{diag}(10, 3, 10^{-8}) Q^\top = \text{diag}(10, 3, 10^{-8})$$

If used in Newton's method, we obtain the following direction:

$$p_k = -B_k^{-1} \nabla f(x_k) = (-1/10, 1, -(2 \cdot 10^8))$$

Thus, a very long vector almost parallel to the third dimension.

## Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \quad \text{and} \quad \nabla^2 f(x_k) = \text{diag}(10, 3, -1)$$

Now, the diagonalization is trivial:

$$\nabla^2 f(x_k) = Q \text{diag}(10, 3, -1) Q^\top \quad Q = I \text{ is the identity matrix}$$

What if we consider a minimum modification replacing the negative eigenvalue with a small number, say  $\delta = 10^{-8}$ ? Obtain

$$B_k = Q \text{diag}(10, 3, 10^{-8}) Q^\top = \text{diag}(10, 3, 10^{-8})$$

If used in Newton's method, we obtain the following direction:

$$p_k = -B_k^{-1} \nabla f(x_k) = (-1/10, 1, -(2 \cdot 10^8))$$

Thus, a very long vector almost parallel to the third dimension.

Note that the original Newton's direction is  $-\text{diag}(1/10, 1/3, -1)(1, -3, 2)^\top = (-1/10, 1, 2)$  which is completely different.

## Modifying the Eigenvalues

Other strategies for eigenvalue modification can be devised.

## Modifying the Eigenvalues

Other strategies for eigenvalue modification can be devised.

The criteria are rather loose. The resulting matrix  $B_k$  should be

- ▶ positive definite,
- ▶ of bounded norm (for all  $k$ ),
- ▶ not too close to being singular.  
(i.e., the eigenvalues should be sufficiently large)

## Modifying the Eigenvalues

Other strategies for eigenvalue modification can be devised.

The criteria are rather loose. The resulting matrix  $B_k$  should be

- ▶ positive definite,
  - ▶ of bounded norm (for all  $k$ ),
  - ▶ not too close to being singular.
- (i.e., the eigenvalues should be sufficiently large)

Strategies for eigenvalue modification include flipping negative eigenvalues to positive values, substituting negative eigenvalues with small positive ones, etc.

## Modifying the Eigenvalues

Other strategies for eigenvalue modification can be devised.

The criteria are rather loose. The resulting matrix  $B_k$  should be

- ▶ positive definite,
- ▶ of bounded norm (for all  $k$ ),
- ▶ not too close to being singular.

(i.e., the eigenvalues should be sufficiently large)

Strategies for eigenvalue modification include flipping negative eigenvalues to positive values, substituting negative eigenvalues with small positive ones, etc.

There is no consensus on the best method for the modification.

## Modifying the Eigenvalues

Other strategies for eigenvalue modification can be devised.

The criteria are rather loose. The resulting matrix  $B_k$  should be

- ▶ positive definite,
  - ▶ of bounded norm (for all  $k$ ),
  - ▶ not too close to being singular.
- (i.e., the eigenvalues should be sufficiently large)

Strategies for eigenvalue modification include flipping negative eigenvalues to positive values, substituting negative eigenvalues with small positive ones, etc.

There is no consensus on the best method for the modification.

The implementation is based on computing  $B_k = H_k + \Delta H_k$  for an appropriate modification matrix  $\Delta H_k$ .

What is  $\Delta H_k$  in our example?

## Modifying the Eigenvalues

Other strategies for eigenvalue modification can be devised.

The criteria are rather loose. The resulting matrix  $B_k$  should be

- ▶ positive definite,
- ▶ of bounded norm (for all  $k$ ),
- ▶ not too close to being singular.

(i.e., the eigenvalues should be sufficiently large)

Strategies for eigenvalue modification include flipping negative eigenvalues to positive values, substituting negative eigenvalues with small positive ones, etc.

There is no consensus on the best method for the modification.

The implementation is based on computing  $B_k = H_k + \Delta H_k$  for an appropriate modification matrix  $\Delta H_k$ .

What is  $\Delta H_k$  in our example?

Various methods for computing  $\Delta H_k$  have been devised in literature. Typically, it is based on some computationally cheaper decomposition than spectral decomposition (e.g., Cholesky).



# Modified Newton's Method

---

**Algorithm 8** Newton's Method with Line Search

---

**Input:**  $x_0$  starting point,  $\varepsilon > 0$

**Output:**  $x^*$  approximation to a stationary point

```
1:  $k \leftarrow 0$ 
2: while  $\|\nabla f_k\|_\infty > \varepsilon$  do
3:    $H_k \leftarrow \nabla^2 f(x_k)$ 
4:   if  $H_k$  is not sufficiently positive definite then
5:      $H_k \leftarrow H_k + \Delta H_k$  so that  $H_k$  is sufficiently pos. definite
6:   end if
7:   Compute  $\nabla f_k = \nabla f(x_k)$ 
8:   Solve  $H_k p_k = -\nabla f_k$  for  $p_k$ 
9:   Set  $x_{k+1} = x_k + \alpha_k p_k$ , here  $\alpha_k$  sat. the Wolfe cond.
10:   $k \leftarrow k + 1$ 
11: end while
```

---

Convergence theorems are complicated in this case and out of the scope of this course. See Chapter 6 of Numerical Optimization by Nocedal & Wright.

## Comments on Newton's Method

- ▶ Newton's method is scale invariant.

## Comments on Newton's Method

- ▶ Newton's method is scale invariant.
- ▶ Quadratic convergence in a close vicinity of a strict minimizer.

## Comments on Newton's Method

- ▶ Newton's method is scale invariant.
- ▶ Quadratic convergence in a close vicinity of a strict minimizer.
- ▶ Without modification, it may converge to an arbitrary stationary point (maximum, saddle point).

## Comments on Newton's Method

- ▶ Newton's method is scale invariant.
- ▶ Quadratic convergence in a close vicinity of a strict minimizer.
- ▶ Without modification, it may converge to an arbitrary stationary point (maximum, saddle point).
- ▶ Computationally expensive:
  - ▶  $\mathcal{O}(n^2)$  second derivatives (the Hessian), each may be hard to compute.  
Automated derivation methods help but still need store  $\mathcal{O}(n^2)$  results.

## Comments on Newton's Method

- ▶ Newton's method is scale invariant.
- ▶ Quadratic convergence in a close vicinity of a strict minimizer.
- ▶ Without modification, it may converge to an arbitrary stationary point (maximum, saddle point).
- ▶ Computationally expensive:
  - ▶  $\mathcal{O}(n^2)$  second derivatives (the Hessian), each may be hard to compute.  
Automated derivation methods help but still need store  $\mathcal{O}(n^2)$  results.
  - ▶  $\mathcal{O}(n^3)$  arithmetic operations to solve the linear system for the direction  $p_k$ .  
May be mitigated by more efficient methods in case of sparse Hessians.

## Comments on Newton's Method

- ▶ Newton's method is scale invariant.
- ▶ Quadratic convergence in a close vicinity of a strict minimizer.
- ▶ Without modification, it may converge to an arbitrary stationary point (maximum, saddle point).
- ▶ Computationally expensive:
  - ▶  $\mathcal{O}(n^2)$  second derivatives (the Hessian), each may be hard to compute.  
Automated derivation methods help but still need store  $\mathcal{O}(n^2)$  results.
  - ▶  $\mathcal{O}(n^3)$  arithmetic operations to solve the linear system for the direction  $p_k$ .  
May be mitigated by more efficient methods in case of sparse Hessians.

In a sense, Newton's method is an impractical “ideal” with which other methods are compared.

The efficiency issues (and the necessity of second-order derivatives) will be mitigated by using quasi-Newton methods.

# Quasi-Newton Methods



## Quasi-Newton Methods

Recall that Newton's method step  $p_k$  in  $x_{k+1} = x_k + p_k$  comes from minimization of

$$q(p) = f_k + \nabla f_k^\top p + \frac{1}{2} p^\top H_k p$$

w.r.t.  $p$  by setting  $\nabla q(p) = 0$  and solving

$$H_k p = -\nabla f_k$$

So Newton's method needs the second derivative (Hessian), which is computationally hard to obtain.

## Quasi-Newton Methods

Recall that Newton's method step  $p_k$  in  $x_{k+1} = x_k + p_k$  comes from minimization of

$$q(p) = f_k + \nabla f_k^\top p + \frac{1}{2} p^\top H_k p$$

w.r.t.  $p$  by setting  $\nabla q(p) = 0$  and solving

$$H_k p = -\nabla f_k$$

So Newton's method needs the second derivative (Hessian), which is computationally hard to obtain.

Gradient descent needs only the first derivatives but converges slowly.

## Quasi-Newton Methods

Recall that Newton's method step  $p_k$  in  $x_{k+1} = x_k + p_k$  comes from minimization of

$$q(p) = f_k + \nabla f_k^\top p + \frac{1}{2} p^\top H_k p$$

w.r.t.  $p$  by setting  $\nabla q(p) = 0$  and solving

$$H_k p = -\nabla f_k$$

So Newton's method needs the second derivative (Hessian), which is computationally hard to obtain.

Gradient descent needs only the first derivatives but converges slowly.

Can we find a compromise?

## Quasi-Newton Methods

Recall that Newton's method step  $p_k$  in  $x_{k+1} = x_k + p_k$  comes from minimization of

$$q(p) = f_k + \nabla f_k^\top p + \frac{1}{2} p^\top H_k p$$

w.r.t.  $p$  by setting  $\nabla q(p) = 0$  and solving

$$H_k p = -\nabla f_k$$

So Newton's method needs the second derivative (Hessian), which is computationally hard to obtain.

Gradient descent needs only the first derivatives but converges slowly.

Can we find a compromise?

Quasi-Newton methods use first derivatives to approximate the Hessian  $H_k$  in Newton's method with a matrix  $\tilde{H}_k$ .

## Quasi-Newton Methods

Suppose we have just obtained the new point  $x_{k+1}$  after a line search starting from  $x_k$  in the direction  $p_k$ .

## Quasi-Newton Methods

Suppose we have just obtained the new point  $x_{k+1}$  after a line search starting from  $x_k$  in the direction  $p_k$ .

Consider the Hessian  $H_{k+1} = \nabla^2 f(x_{k+1})$  and its approximation denoted by  $\tilde{H}_{k+1}$ .

## Quasi-Newton Methods

Suppose we have just obtained the new point  $x_{k+1}$  after a line search starting from  $x_k$  in the direction  $p_k$ .

Consider the Hessian  $H_{k+1} = \nabla^2 f(x_{k+1})$  and its approximation denoted by  $\tilde{H}_{k+1}$ .

We aim to use  $\tilde{H}_{k+1}$  in the next step, that is, in the equation  $\tilde{H}_{k+1}p = -\nabla f_{k+1}$  yielding  $p_{k+1}$ .

## Quasi-Newton Methods

Suppose we have just obtained the new point  $x_{k+1}$  after a line search starting from  $x_k$  in the direction  $p_k$ .

Consider the Hessian  $H_{k+1} = \nabla^2 f(x_{k+1})$  and its approximation denoted by  $\tilde{H}_{k+1}$ .

We aim to use  $\tilde{H}_{k+1}$  in the next step, that is, in the equation  $\tilde{H}_{k+1}p = -\nabla f_{k+1}$  yielding  $p_{k+1}$ .

What conditions should  $\tilde{H}_{k+1}$  satisfy so that it functions as the “true” Hessian  $H_{k+1}$ ?



## Quasi-Newton Methods

Suppose we have just obtained the new point  $x_{k+1}$  after a line search starting from  $x_k$  in the direction  $p_k$ .

Consider the Hessian  $H_{k+1} = \nabla^2 f(x_{k+1})$  and its approximation denoted by  $\tilde{H}_{k+1}$ .

We aim to use  $\tilde{H}_{k+1}$  in the next step, that is, in the equation  $\tilde{H}_{k+1}p = -\nabla f_{k+1}$  yielding  $p_{k+1}$ .

What conditions should  $\tilde{H}_{k+1}$  satisfy so that it functions as the “true” Hessian  $H_{k+1}$ ?

First, it should be *symmetric positive definite*.

To always yield decrease direction.

## Quasi-Newton Methods

Suppose we have just obtained the new point  $x_{k+1}$  after a line search starting from  $x_k$  in the direction  $p_k$ .

Consider the Hessian  $H_{k+1} = \nabla^2 f(x_{k+1})$  and its approximation denoted by  $\tilde{H}_{k+1}$ .

We aim to use  $\tilde{H}_{k+1}$  in the next step, that is, in the equation  $\tilde{H}_{k+1}p = -\nabla f_{k+1}$  yielding  $p_{k+1}$ .

What conditions should  $\tilde{H}_{k+1}$  satisfy so that it functions as the “true” Hessian  $H_{k+1}$ ?

First, it should be *symmetric positive definite*.

To always yield decrease direction.

Second, extrapolating from the single variable secant method, we demand the *secant condition*:

$$\tilde{H}_{k+1}(x_{k+1} - x_k) = \nabla f_{k+1} - \nabla f_k$$

## Secant Condition

Consider the secant condition:

$$\tilde{H}_{k+1}(x_{k+1} - x_k) = \nabla f_{k+1} - \nabla f_k$$

## Secant Condition

Consider the secant condition:

$$\tilde{H}_{k+1}(x_{k+1} - x_k) = \nabla f_{k+1} - \nabla f_k$$

The notation is usually simplified by

$$s_k = x_{k+1} - x_k \quad y_k = \nabla f_{k+1} - \nabla f_k$$

So that the secant condition becomes

$$\tilde{H}_{k+1}s_k = y_k$$

## Secant Condition

Consider the secant condition:

$$\tilde{H}_{k+1}(x_{k+1} - x_k) = \nabla f_{k+1} - \nabla f_k$$

The notation is usually simplified by

$$s_k = x_{k+1} - x_k \quad y_k = \nabla f_{k+1} - \nabla f_k$$

So that the secant condition becomes

$$\tilde{H}_{k+1}s_k = y_k$$

Note that even if we demand symmetric positive definite solutions to the secant condition, there are potentially infinitely many.

Indeed, there are  $n(n+1)/2$  degrees of freedom in a symmetric matrix, and the secant conditions represent only  $n$  conditions (Sylvester's criterion).

Moreover, we want to obtain  $\tilde{H}_{k+1}$  from  $\tilde{H}_k$  by

$$\tilde{H}_{k+1} = \tilde{H}_k + \text{something}$$

To have a nice iterative algorithm.

## Symmetric Rank One Update (SR1)

Note that the information about the solution is present in  $s_k$  and  $y_k$ , so it is natural to compose the solution using these vectors.

## Symmetric Rank One Update (SR1)

Note that the information about the solution is present in  $s_k$  and  $y_k$ , so it is natural to compose the solution using these vectors.

Consider  $u = (y_k - \tilde{H}_k s_k)$

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{uu^\top}{u^\top s_k}$$

## Symmetric Rank One Update (SR1)

Note that the information about the solution is present in  $s_k$  and  $y_k$ , so it is natural to compose the solution using these vectors.

Consider  $u = (y_k - \tilde{H}_k s_k)$

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{uu^\top}{u^\top s_k}$$

Now, the secant condition is satisfied:

$$\tilde{H}_{k+1} s_k = \tilde{H}_k s_k + \frac{uu^\top s_k}{u^\top s_k} = \tilde{H}_k s_k + u = \tilde{H}_k s_k + (y_k - \tilde{H}_k s_k) = y_k$$

By the way, the matrix  $\frac{uu^\top}{u^\top s_k}$  is of rank one and is a unique symmetric rank one matrix which makes a symmetric  $\tilde{H}_{k+1}$  satisfy the secant condition.



## Symmetric Rank One Update (SR1)

Note that the information about the solution is present in  $s_k$  and  $y_k$ , so it is natural to compose the solution using these vectors.

Consider  $u = (y_k - \tilde{H}_k s_k)$

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{uu^\top}{u^\top s_k}$$

Now, the secant condition is satisfied:

$$\tilde{H}_{k+1} s_k = \tilde{H}_k s_k + \frac{uu^\top s_k}{u^\top s_k} = \tilde{H}_k s_k + u = \tilde{H}_k s_k + (y_k - \tilde{H}_k s_k) = y_k$$

By the way, the matrix  $\frac{uu^\top}{u^\top s_k}$  is of rank one and is a unique symmetric rank one matrix which makes a symmetric  $\tilde{H}_{k+1}$  satisfy the secant condition.

To obtain a quasi-Newton method, it suffices to initialize  $\tilde{H}_0$ , typically to the identity  $I$ , and use  $\tilde{H}_k$  instead of the Hessian  $H_k = \nabla^2 f_k$  in Newton's method.

# Symmetric Rank One Update

---

## Algorithm 9 SR1

---

**Input:**  $x_0$  starting point,  $\varepsilon > 0$

**Output:**  $x^*$  approximation to a stationary point

$k \leftarrow 0$ ,  $\alpha_{\text{init}} \leftarrow 1$ ,  $\tilde{H}_0 \leftarrow I$

**while**  $\|\nabla f_k\|_\infty > \varepsilon$  **do**

    Compute  $\nabla f_k = \nabla f(x_k)$

    Solve for  $p_k$  in  $\tilde{H}_k p_k = -\nabla f_k$

$\alpha \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$

$x_{k+1} \leftarrow x_k + \alpha p_k$

$s \leftarrow x_{k+1} - x_k$

$y \leftarrow \nabla f_{k+1} - \nabla f_k$

$u \leftarrow y - \tilde{H}_k s$

$\tilde{H}_{k+1} \leftarrow \tilde{H}_k + \frac{uu^\top}{u^\top s}$

$k \leftarrow k + 1$

**end while**

---

Note that the denominator  $u^\top s_k$  can be 0, in which case the update is impossible. The usual strategy is to skip the update and set  $\tilde{H}_{k+1} = \tilde{H}_k$ .

## Example

We will look at a three-dimensional quadratic problem

$f(x) = \frac{1}{2}x^\top Qx - c^\top x$  with

$$Q = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix},$$

whose solution is  $x_* = (-4, -3, -2)^\top$ . Use the exact line search.

The initial guesses are  $\tilde{H}_0 = I$  and  $x_0 = (0, 0, 0)^\top$ .

## Example

We will look at a three-dimensional quadratic problem

$f(x) = \frac{1}{2}x^\top Qx - c^\top x$  with

$$Q = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix},$$

whose solution is  $x_* = (-4, -3, -2)^\top$ . Use the exact line search.

The initial guesses are  $\tilde{H}_0 = I$  and  $x_0 = (0, 0, 0)^\top$ .

At the initial point,  $\|\nabla f(x_0)\|_\infty = \|-c\|_\infty = 9$ , so this point is not optimal.

## Example

We will look at a three-dimensional quadratic problem

$f(x) = \frac{1}{2}x^\top Qx - c^\top x$  with

$$Q = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix},$$

whose solution is  $x_* = (-4, -3, -2)^\top$ . Use the exact line search.

The initial guesses are  $\tilde{H}_0 = I$  and  $x_0 = (0, 0, 0)^\top$ .

At the initial point,  $\|\nabla f(x_0)\|_\infty = \|-c\|_\infty = 9$ , so this point is not optimal. The first search direction is

$$p_0 = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix}.$$

The exact line search gives  $\alpha_0 = 0.3333$ .

## Example

The new estimate of the solution, the update vectors, and the new Hessian approximation are:

$$x_1 = \begin{pmatrix} -2.66 \\ -3.00 \\ -2.66 \end{pmatrix}, \nabla f_1 = \begin{pmatrix} 2.66 \\ 0 \\ -2.66 \end{pmatrix}, s_0 = \begin{pmatrix} -2.66 \\ -3.00 \\ -2.66 \end{pmatrix}, y_0 = \begin{pmatrix} -5.33 \\ -9.00 \\ -10.66 \end{pmatrix},$$

## Example

The new estimate of the solution, the update vectors, and the new Hessian approximation are:

$$x_1 = \begin{pmatrix} -2.66 \\ -3.00 \\ -2.66 \end{pmatrix}, \nabla f_1 = \begin{pmatrix} 2.66 \\ 0 \\ -2.66 \end{pmatrix}, s_0 = \begin{pmatrix} -2.66 \\ -3.00 \\ -2.66 \end{pmatrix}, y_0 = \begin{pmatrix} -5.33 \\ -9.00 \\ -10.66 \end{pmatrix},$$

and

$$\tilde{H}_1 = I + \frac{(y_0 - \nabla f_0)(y_0 - \nabla f_0)^\top}{(y_0 - \nabla f_0)^\top s_0} = \begin{pmatrix} 1.1531 & 0.3445 & 0.4593 \\ 0.3445 & 1.7751 & 1.0335 \\ 0.4593 & 1.0335 & 2.3780 \end{pmatrix}.$$

## Example

The new estimate of the solution, the update vectors, and the new Hessian approximation are:

$$x_1 = \begin{pmatrix} -2.66 \\ -3.00 \\ -2.66 \end{pmatrix}, \nabla f_1 = \begin{pmatrix} 2.66 \\ 0 \\ -2.66 \end{pmatrix}, s_0 = \begin{pmatrix} -2.66 \\ -3.00 \\ -2.66 \end{pmatrix}, y_0 = \begin{pmatrix} -5.33 \\ -9.00 \\ -10.66 \end{pmatrix},$$

and

$$\tilde{H}_1 = I + \frac{(y_0 - Is_0)(y_0 - Is_0)^\top}{(y_0 - Is_0)^\top s_0} = \begin{pmatrix} 1.1531 & 0.3445 & 0.4593 \\ 0.3445 & 1.7751 & 1.0335 \\ 0.4593 & 1.0335 & 2.3780 \end{pmatrix}.$$

At this new point  $\|\nabla f(x_1)\|_\infty = 2.66$  so we keep going, obtaining the search direction

$$p_1 = \begin{pmatrix} -2.9137 \\ -0.5557 \\ 1.9257 \end{pmatrix},$$

and the step length  $\alpha_1 = 0.3942$ .



## Example

This gives the new estimates:

$$x_2 = \begin{pmatrix} -3.81 \\ -3.21 \\ -1.90 \end{pmatrix}, \quad \nabla f_2 = \begin{pmatrix} 0.36 \\ -0.65 \\ 0.36 \end{pmatrix}, \quad s_1 = \begin{pmatrix} -1.14 \\ -0.21 \\ 0.75 \end{pmatrix}, \quad y_1 = \begin{pmatrix} -2.29 \\ -0.65 \\ 3.03 \end{pmatrix}$$

and

$$\tilde{H}_2 = \begin{pmatrix} 1.6568 & 0.6102 & -0.3432 \\ 0.6102 & 1.9153 & 0.6102 \\ -0.3432 & 0.6102 & 3.6568 \end{pmatrix}.$$

At the point  $x_2$ ,  $\|\nabla f(x_2)\|_\infty = 0.65$  so we keep going, with

$$p_2 = \begin{pmatrix} -0.4851 \\ 0.5749 \\ -0.2426 \end{pmatrix},$$

and  $\alpha = 0.3810$ .

## Example

This gives

$$x_3 = \begin{pmatrix} -4 \\ -3 \\ -2 \end{pmatrix}, \quad \nabla f_3 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad s_2 = \begin{pmatrix} -0.18 \\ 0.21 \\ -0.09 \end{pmatrix}, \quad y_2 = \begin{pmatrix} -0.36 \\ 0.65 \\ -0.36 \end{pmatrix},$$

and  $\tilde{H}_3 = Q$ . Now  $\|\nabla f(x_3)\|_\infty = 0$ , so we stop.

# Properties of SR1

Does symmetric rank one update satisfy our demands?

We want every  $\tilde{H}_k$  to be a symmetric positive definite solution to the secant condition.

# Properties of SR1

Does symmetric rank one update satisfy our demands?

We want every  $\tilde{H}_k$  to be a symmetric positive definite solution to the secant condition.

Unfortunately, though  $\tilde{H}_k$  is a symmetric positive definite, the updated matrix  $\tilde{H}_{k+1}$  does not have to be positive definite.

# Properties of SR1

Does symmetric rank one update satisfy our demands?

We want every  $\tilde{H}_k$  to be a symmetric positive definite solution to the secant condition.

Unfortunately, though  $\tilde{H}_k$  is a symmetric positive definite, the updated matrix  $\tilde{H}_{k+1}$  does not have to be positive definite.

Still, the symmetric rank one approximation is used in practice, especially in trust region methods.

# Properties of SR1

Does symmetric rank one update satisfy our demands?

We want every  $\tilde{H}_k$  to be a symmetric positive definite solution to the secant condition.

Unfortunately, though  $\tilde{H}_k$  is a symmetric positive definite, the updated matrix  $\tilde{H}_{k+1}$  does not have to be positive definite.

Still, the symmetric rank one approximation is used in practice, especially in trust region methods.

However, for line search, let us try a bit “richer” solution to the secant condition.

## Symmetric Rank Two Update

Consider

$$\tilde{H}_{k+1} = \tilde{H}_k - \frac{\left(\tilde{H}_k s_k\right) \left(\tilde{H}_k s_k\right)^{\top}}{s_k^{\top} \tilde{H}_k s_k} + \frac{y_k y_k^{\top}}{y_k^{\top} s_k}$$

Once again, verifying  $\tilde{H}_{k+1} s_k = y_k$  is not difficult.

# Symmetric Rank Two Update

Consider

$$\tilde{H}_{k+1} = \tilde{H}_k - \frac{\left(\tilde{H}_k s_k\right) \left(\tilde{H}_k s_k\right)^{\top}}{s_k^{\top} \tilde{H}_k s_k} + \frac{y_k y_k^{\top}}{y_k^{\top} s_k}$$

Once again, verifying  $\tilde{H}_{k+1} s_k = y_k$  is not difficult.

## Crucial observation:

Assume that  $\tilde{H}_k$  is symmetric positive definite.

If the next approximation  $x_{k+1}$  is computed using a line search satisfying the strong Wolfe conditions, then  $\tilde{H}_{k+1}$  is also symmetric positive definite.

For proof see Lemma 12.10 and Exercise 3.9 of "Linear and Nonlinear Optimization" by Griva et al.

Thus, starting with a symmetric positive definite  $\tilde{H}_0$  and doing line search satisfying the strong Wolfe conditions, every  $\tilde{H}_k$  is *symmetric positive definite and satisfies the secant condition*.



---

**Algorithm 10** BFGS v1

---

**Input:**  $x_0$  starting point,  $\varepsilon > 0$ **Output:**  $x^*$  approximation to a stationary point $k \leftarrow 0, \alpha_{\text{init}} \leftarrow 1, \tilde{H}_0 \leftarrow I$ **while**  $\|\nabla f_k\|_\infty > \tau$  **do**    Compute  $\nabla f_k = \nabla f(x_k)$     Solve for  $p_k$  in  $\tilde{H}_k p_k = -\nabla f_k$      $\alpha \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$      $x_{k+1} \leftarrow x_k + \alpha p_k$      $s \leftarrow x_{k+1} - x_k$      $y \leftarrow \nabla f_{k+1} - \nabla f_k$     
$$\tilde{H}_{k+1} \leftarrow \tilde{H}_k - \frac{(\tilde{H}_k s)(\tilde{H}_k s)^\top}{s^\top \tilde{H}_k s} + \frac{y y^\top}{y^\top s}$$
     $k \leftarrow k + 1$ **end while**

---

Note that we still have to solve a linear system for  $p_k$ .

## Example

Consider the quadratic problem  $f(x) = \frac{1}{2}x^\top Qx - c^\top x$  with

$$Q = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix},$$

whose solution is  $x_* = (-4, -3, -2)^\top$ . Use the exact line search.

## Example

Consider the quadratic problem  $f(x) = \frac{1}{2}x^\top Qx - c^\top x$  with

$$Q = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix},$$

whose solution is  $x_* = (-4, -3, -2)^\top$ . Use the exact line search.

Choose  $\tilde{H}_0 = I$  and  $x_0 = (0, 0, 0)^\top$ .

## Example

Consider the quadratic problem  $f(x) = \frac{1}{2}x^\top Qx - c^\top x$  with

$$Q = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix},$$

whose solution is  $x_* = (-4, -3, -2)^\top$ . Use the exact line search.

Choose  $\tilde{H}_0 = I$  and  $x_0 = (0, 0, 0)^\top$ .

At iteration 0,  $\|\nabla f(x_0)\|_\infty = 9$ , so this point is not optimal.

## Example

Consider the quadratic problem  $f(x) = \frac{1}{2}x^\top Qx - c^\top x$  with

$$Q = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix},$$

whose solution is  $x_* = (-4, -3, -2)^\top$ . Use the exact line search.

Choose  $\tilde{H}_0 = I$  and  $x_0 = (0, 0, 0)^\top$ .

At iteration 0,  $\|\nabla f(x_0)\|_\infty = 9$ , so this point is not optimal.

The search direction is

$$p_0 = \begin{pmatrix} -8 \\ -9 \\ -8 \end{pmatrix}$$

and  $\alpha_0 = 0.3333$ .

## Example

The new estimate of the solution and the new Hessian approximation are

$$x_1 = \begin{pmatrix} -2.6667 \\ -3.0000 \\ -2.6667 \end{pmatrix} \quad \text{and} \quad \tilde{H}_1 = \begin{pmatrix} 1.1021 & 0.3445 & 0.5104 \\ 0.3445 & 1.7751 & 1.0335 \\ 0.5104 & 1.0335 & 2.3270 \end{pmatrix}.$$

## Example

The new estimate of the solution and the new Hessian approximation are

$$x_1 = \begin{pmatrix} -2.6667 \\ -3.0000 \\ -2.6667 \end{pmatrix} \quad \text{and} \quad \tilde{H}_1 = \begin{pmatrix} 1.1021 & 0.3445 & 0.5104 \\ 0.3445 & 1.7751 & 1.0335 \\ 0.5104 & 1.0335 & 2.3270 \end{pmatrix}.$$

At iteration 1,  $\|\nabla f(x_1)\|_\infty = 2.6667$ , so we continue. The next search direction is

$$p_1 = \begin{pmatrix} -3.2111 \\ -0.6124 \\ 2.1223 \end{pmatrix}$$

and  $\alpha_1 = 0.3577$ .

## Example

This gives the estimates.

$$x_2 = \begin{pmatrix} -3.8152 \\ -3.2191 \\ -1.9076 \end{pmatrix} \quad \text{and} \quad \tilde{H}_2 = \begin{pmatrix} 1.6393 & 0.6412 & -0.3607 \\ 0.6412 & 1.8600 & 0.6412 \\ -0.3607 & 0.6412 & 3.6393 \end{pmatrix}.$$

At iteration 2,  $\|\nabla f(x_2)\|_\infty = 0.6572$ , so we continue, computing

$$p_2 = \begin{pmatrix} -0.5289 \\ 0.6268 \\ -0.2644 \end{pmatrix}$$

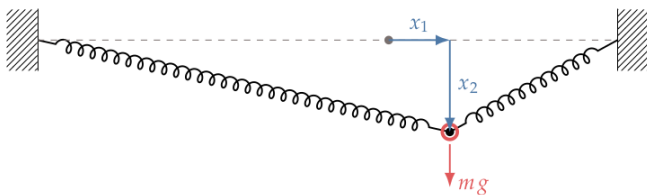
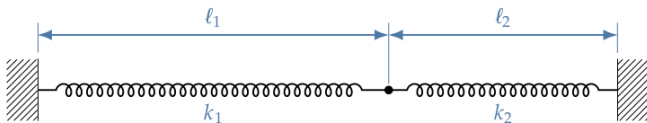
and  $\alpha_2 = 0.3495$ . This gives

$$x_3 = \begin{pmatrix} -4 \\ -3 \\ -2 \end{pmatrix} \quad \text{and} \quad \tilde{H}_3 = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix}.$$

Now  $\|\nabla f(x_3)\|_\infty = 0$ , so we stop.

Notice that we got the same  $x_1, x_2, x_3$  as for SR1. This follows from using the exact line search and the quadratic problem. It does not hold in general.

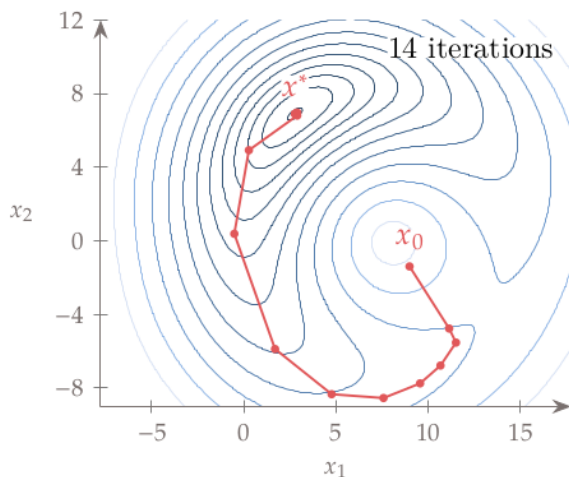




$$f(x_1, x_2) = \frac{1}{2}k_1 \left( \sqrt{(\ell_1 + x_1)^2 + x_2^2} - \ell_1 \right)^2 + \frac{1}{2}k_2 \left( \sqrt{(\ell_2 - x_1)^2 + x_2^2} - \ell_2 \right)^2 - mgx_2$$

Here  $\ell_1 = 12, \ell_2 = 8, k_1 = 1, k_2 = 10, mg = 7$

## Two Spring Problem - BFGS

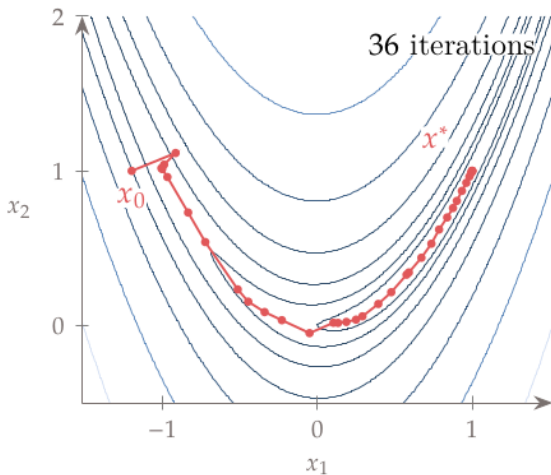


BFGS, line search, stop. cond.  $\|\nabla f\|_\infty \leq 10^{-6}$ .

Compare this with 32 iterations of gradient descent and 12 iterations of Newton's method.

## Rosenbrock Function - BFGS

Rosenbrock:  $f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$



BFGS, line search, stop. cond.  $\|\nabla f\|_{\infty} \leq 10^{-6}$ .

Compare with 10,662 iterations of gradient descent and 24 iterations of Newton's method.

## Sherman–Morrison–Woodbury Formula

**Problem:** SR1 and BFGS solve  $\tilde{H}_k p = -\nabla f_k$  repeatedly. What if we could iteratively update  $H_k^{-1}$ ?

## Sherman–Morrison–Woodbury Formula

**Problem:** SR1 and BFGS solve  $\tilde{H}_k p = -\nabla f_k$  repeatedly. What if we could iteratively update  $H_k^{-1}$ ?

The equation would be solved by  $p_k = -H_k^{-1} \nabla f_k$ .

## Sherman–Morrison–Woodbury Formula

**Problem:** SR1 and BFGS solve  $\tilde{H}_k p = -\nabla f_k$  repeatedly. What if we could iteratively update  $H_k^{-1}$ ?

The equation would be solved by  $p_k = -H_k^{-1} \nabla f_k$ .

Ideally, we would like to compute  $\tilde{H}_k^{-1}$  iteratively along the optimization, i.e.,

$$\tilde{H}_{k+1}^{-1} = \tilde{H}_k^{-1} + \text{something}$$

## Sherman–Morrison–Woodbury Formula

**Problem:** SR1 and BFGS solve  $\tilde{H}_k p = -\nabla f_k$  repeatedly. What if we could iteratively update  $H_k^{-1}$ ?

The equation would be solved by  $p_k = -H_k^{-1} \nabla f_k$ .

Ideally, we would like to compute  $\tilde{H}_k^{-1}$  iteratively along the optimization, i.e.,

$$\tilde{H}_{k+1}^{-1} = \tilde{H}_k^{-1} + \text{something}$$

To get such a “something” we use the following Sherman–Morrison–Woodbury (SMW) formula:

$$\left(A + UV^T\right)^{-1} = A^{-1} - A^{-1}U \left(I + V^T A^{-1}U\right)^{-1} V^T A^{-1}$$

Here  $A$  is a  $(n \times n)$ -matrix,  $U, V$  are  $(n \times m)$ -matrices with  $m \leq n$ .

# Rank 1 – Iterative Inverse Hessian Approximation

Applying SMW to the rank one update

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{\left(y_k - \tilde{H}_k s_k\right) \left(y_k - \tilde{H}_k s_k\right)^{\top}}{\left(y_k - \tilde{H}_k s_k\right)^{\top} s_k}$$



## Rank 1 – Iterative Inverse Hessian Approximation

Applying SMW to the rank one update

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{\left(y_k - \tilde{H}_k s_k\right) \left(y_k - \tilde{H}_k s_k\right)^{\top}}{\left(y_k - \tilde{H}_k s_k\right)^{\top} s_k}$$

yields

$$\tilde{H}_{k+1}^{-1} = \tilde{H}_k^{-1} + \frac{\left(s_k - \tilde{H}_k^{-1} y_k\right) \left(s_k - \tilde{H}_k^{-1} y_k\right)^{\top}}{\left(s_k - \tilde{H}_k^{-1} y_k\right)^{\top} y_k}$$

Yes, only  $y$  and  $s$  swapped places.

## Rank 1 – Iterative Inverse Hessian Approximation

Applying SMW to the rank one update

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{\left(y_k - \tilde{H}_k s_k\right) \left(y_k - \tilde{H}_k s_k\right)^{\top}}{\left(y_k - \tilde{H}_k s_k\right)^{\top} s_k}$$

yields

$$\tilde{H}_{k+1}^{-1} = \tilde{H}_k^{-1} + \frac{\left(s_k - \tilde{H}_k^{-1} y_k\right) \left(s_k - \tilde{H}_k^{-1} y_k\right)^{\top}}{\left(s_k - \tilde{H}_k^{-1} y_k\right)^{\top} y_k}$$

Yes, only  $y$  and  $s$  swapped places.

This allows us to avoid solving  $\tilde{H}_k p_k = -\nabla f_k$  for  $p_k$  in every iteration.

Masochists may study details of the proof, e.g., in "On the derivation of quasi-Newton formulas for optimization in function spaces" by Vuchkov et al. *Journal of Numerical Functional Analysis and Optimization*, 2021

## Rank One Update V2

---

**Algorithm 11** Rank 1 update v1

---

**Input:**  $x_0$  starting point,  $\varepsilon > 0$

**Output:**  $x^*$  approximation to a stationary point

- 1:  $k \leftarrow 0, \alpha_{\text{init}} \leftarrow 1, \tilde{H}_0 \leftarrow I$
  - 2: **while**  $\|\nabla f_k\|_\infty > \varepsilon$  **do**
  - 3:     Compute  $\nabla f_k = \nabla f(x_k)$
  - 4:      $p_k \leftarrow -\tilde{H}_k^{-1} \nabla f_k$
  - 5:      $\alpha \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$
  - 6:      $x_{k+1} \leftarrow x_k + \alpha p_k$
  - 7:      $s \leftarrow x_{k+1} - x_k$
  - 8:      $y \leftarrow \nabla f_{k+1} - \nabla f_k$
  - 9:      $\tilde{H}_{k+1}^{-1} \leftarrow \tilde{H}_k^{-1} + \frac{(s - \tilde{H}_k^{-1} y)(s - \tilde{H}_k^{-1} y)^\top}{(s - \tilde{H}_k^{-1} y)^\top y}$
  - 10:     $k \leftarrow k + 1$
  - 11: **end while**
-

Applying SMW to the BFGS Hessian update

$$\tilde{H}_{k+1} = \tilde{H}_k - \frac{\left(\tilde{H}_k s_k\right) \left(\tilde{H}_k s_k\right)^{\top}}{s_k^{\top} \tilde{H}_k s_k} + \frac{y_k y_k^{\top}}{y_k^{\top} s_k}$$

Applying SMW to the BFGS Hessian update

$$\tilde{H}_{k+1} = \tilde{H}_k - \frac{\left(\tilde{H}_k s_k\right) \left(\tilde{H}_k s_k\right)^{\top}}{s_k^{\top} \tilde{H}_k s_k} + \frac{y_k y_k^{\top}}{y_k^{\top} s_k}$$

yields

$$\tilde{H}_{k+1}^{-1} = \left(I - \frac{s_k y_k^{\top}}{s_k^{\top} y_k}\right) \tilde{H}_k^{-1} \left(I - \frac{y_k s_k^{\top}}{s_k^{\top} y_k}\right) + \frac{s_k s_k^{\top}}{s_k^{\top} y_k}$$

We avoid solving the linear system for  $p_k$ .

For a proof see, e.g., in "On the derivation of quasi-Newton formulas for optimization in function spaces" by Vuchkov et al. Journal of Numerical Functional Analysis and Optimization, 2021

---

**Algorithm 12** BFGS v2

---

**Input:**  $x_0$  starting point,  $\varepsilon > 0$ **Output:**  $x^*$  approximation to a stationary point

```

1:  $k \leftarrow 0$ ,  $\alpha_{\text{init}} \leftarrow 1$ ,  $\tilde{H}_0 \leftarrow I$ 
2: while  $\|\nabla f_k\|_\infty > \varepsilon$  do
3:   Compute  $\nabla f_k = \nabla f(x_k)$ 
4:    $p_k \leftarrow -\tilde{H}_k^{-1} \nabla f_k$ 
5:    $\alpha \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$ 
6:    $x_{k+1} \leftarrow x_k + \alpha p_k$ 
7:    $s \leftarrow x_{k+1} - x_k$ 
8:    $y \leftarrow \nabla f_{k+1} - \nabla f_k$ 
9:    $\tilde{H}_{k+1}^{-1} \leftarrow \left( I - \frac{sy^\top}{s^\top y} \right) \tilde{H}_k^{-1} \left( I - \frac{ys^\top}{s^\top y} \right) + \frac{ss^\top}{s^\top y}$ 
10:   $k \leftarrow k + 1$ 
11: end while

```

---

Still, we must drag the quadratic size matrix  $\tilde{H}_{k+1}^{-1}$  along.

## Limited Memory BFGS Idea

Let us denote by  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$  the values of the variables  $s$  and  $y$ , resp., during the iterations  $1, \dots, k$  of BFGS.

## Limited Memory BFGS Idea

Let us denote by  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$  the values of the variables  $s$  and  $y$ , resp., during the iterations  $1, \dots, k$  of BFGS.

Observe that  $\tilde{H}_k^{-1}$  is determined completely by  $H_0^{-1}$  and the two sequences  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$ .



## Limited Memory BFGS Idea

Let us denote by  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$  the values of the variables  $s$  and  $y$ , resp., during the iterations  $1, \dots, k$  of BFGS.

Observe that  $\tilde{H}_k^{-1}$  is determined completely by  $H_0^{-1}$  and the two sequences  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$ .

So, the matrix  $\tilde{H}_k^{-1}$  does not have to be stored if the algorithm remembers the values  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$ .

Note that this would be more space efficient for  $k < n/2$ .

## Limited Memory BFGS Idea

Let us denote by  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$  the values of the variables  $s$  and  $y$ , resp., during the iterations  $1, \dots, k$  of BFGS.

Observe that  $\tilde{H}_k^{-1}$  is determined completely by  $H_0^{-1}$  and the two sequences  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$ .

So, the matrix  $\tilde{H}_k^{-1}$  does not have to be stored if the algorithm remembers the values  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$ .

Note that this would be more space efficient for  $k < n/2$ .

However, we may go further and observe that typically only a few, say  $m$ , past values of  $s$  and  $y$  are sufficient for a good approximation of  $\tilde{H}_k^{-1}$  when we set  $\tilde{H}_{k-m-1}^{-1} = I$ .

## Limited Memory BFGS Idea

Let us denote by  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$  the values of the variables  $s$  and  $y$ , resp., during the iterations  $1, \dots, k$  of BFGS.

Observe that  $\tilde{H}_k^{-1}$  is determined completely by  $H_0^{-1}$  and the two sequences  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$ .

So, the matrix  $\tilde{H}_k^{-1}$  does not have to be stored if the algorithm remembers the values  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$ .

Note that this would be more space efficient for  $k < n/2$ .

However, we may go further and observe that typically only a few, say  $m$ , past values of  $s$  and  $y$  are sufficient for a good approximation of  $\tilde{H}_k^{-1}$  when we set  $\tilde{H}_{k-m-1}^{-1} = I$ .

This is the basic idea behind limited-memory BFGS, which stores only the running window  $s_{k-m}, \dots, s_k$  and  $y_{k-m}, \dots, y_k$  and computes  $\tilde{H}_k^{-1} \nabla f_k$  using these values.

## Limited Memory BFGS Idea

Let us denote by  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$  the values of the variables  $s$  and  $y$ , resp., during the iterations  $1, \dots, k$  of BFGS.

Observe that  $\tilde{H}_k^{-1}$  is determined completely by  $H_0^{-1}$  and the two sequences  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$ .

So, the matrix  $\tilde{H}_k^{-1}$  does not have to be stored if the algorithm remembers the values  $s_0, \dots, s_k$  and  $y_0, \dots, y_k$ .

Note that this would be more space efficient for  $k < n/2$ .

However, we may go further and observe that typically only a few, say  $m$ , past values of  $s$  and  $y$  are sufficient for a good approximation of  $\tilde{H}_k^{-1}$  when we set  $\tilde{H}_{k-m-1}^{-1} = I$ .

This is the basic idea behind limited-memory BFGS, which stores only the running window  $s_{k-m}, \dots, s_k$  and  $y_{k-m}, \dots, y_k$  and computes  $\tilde{H}_k^{-1} \nabla f_k$  using these values.

The space complexity becomes  $nm$ , which is beneficial when  $n$  is large.

## Another View on BFGS (Optional)

We search for  $\tilde{H}_{k+1}^{-1}$  where  $\tilde{H}_{k+1}$  satisfies  $\tilde{H}_{k+1}s_k = y_k$ . Search for a solution  $\tilde{V}$  for  $\tilde{V}y_k = s_k$ .

The idea is to use  $\tilde{V}$  close to  $\tilde{H}_k^{-1}$  (in some sense):

$$\begin{aligned} \min_{\tilde{H}} \quad & \left\| \tilde{V} - \tilde{H}_k^{-1} \right\| \\ \text{subject to} \quad & \tilde{V} = \tilde{V}^\top, \quad \tilde{V}y_k = s_k \end{aligned}$$

Here the norm is *weighted Frobenius norm*:

$$\|A\| \equiv \left\| W^{1/2} A W^{1/2} \right\|_F,$$

where  $\|\cdot\|_F$  is defined by  $\|C\|_F^2 = \sum_{i=1}^n \sum_{j=1}^n c_{ij}^2$ . The weight  $W$  can be chosen as any matrix satisfying the relation  $Wy_k = s_k$ .

BFGS is obtained with  $W = \bar{G}_k^{-1}$  where  $\bar{G}_k$  is the average Hessian defined by  $\bar{G}_k = \left[ \int_0^1 \nabla^2 f(x_k + \tau \alpha_k p_k) d\tau \right]$

Solving this gives precisely the BFGS formula for  $\tilde{H}_{k+1}^{-1}$ .

## Global Convergence of Line Search

Denote by  $\theta_k$  the angle between  $p_k$  and  $-\nabla f_k$ , i.e., satisfying

$$\cos \theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \|p_k\|}$$

Recall that  $f$  is  $L$ -smooth for some  $L > 0$  if

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \quad \text{for all } x, \tilde{x} \in \mathbb{R}^n$$

### Theorem 15 (Zoutendijk)

Consider  $x_{k+1} = x_k + \alpha_k p_k$ , where  $p_k$  is a descent direction and  $\alpha_k$  satisfies the strong Wolfe conditions. Suppose that  $f$  is bounded below, continuously differentiable, and  $L$ -smooth. Then

$$\sum_{k \geq 0} \cos^2 \theta_k \|\nabla f_k\|^2 < \infty.$$

# Global Convergence of Quasi-Newton's Method

Assume that all  $\alpha_k$  satisfy strong Wolfe conditions.

Assume that the approximations to the Hessians  $\tilde{H}_k$  are positive definite with a uniformly bounded condition number:

$$\left\| \tilde{H}_k \right\| \left\| \tilde{H}_k^{-1} \right\| \leq M \quad \text{for all } k$$

# Global Convergence of Quasi-Newton's Method

Assume that all  $\alpha_k$  satisfy strong Wolfe conditions.

Assume that the approximations to the Hessians  $\tilde{H}_k$  are positive definite with a uniformly bounded condition number:

$$\left\| \tilde{H}_k \right\| \left\| \tilde{H}_k^{-1} \right\| \leq M \quad \text{for all } k$$

Then  $\theta_k$  between  $p_k = -\tilde{H}_k^{-1} \nabla f_k$  and  $-\nabla f_k$  and satisfies

$$\cos \theta_k \geq 1/M$$



# Global Convergence of Quasi-Newton's Method

Assume that all  $\alpha_k$  satisfy strong Wolfe conditions.

Assume that the approximations to the Hessians  $\tilde{H}_k$  are positive definite with a uniformly bounded condition number:

$$\left\| \tilde{H}_k \right\| \left\| \tilde{H}_k^{-1} \right\| \leq M \quad \text{for all } k$$

Then  $\theta_k$  between  $p_k = -\tilde{H}_k^{-1} \nabla f_k$  and  $-\nabla f_k$  and satisfies

$$\cos \theta_k \geq 1/M$$

Thus, under the assumptions of Zoutendijk's theorem, we obtain

$$\frac{1}{M^2} \sum_{k \geq 0} \|\nabla f_k\|^2 \leq \sum_{k \geq 0} \cos^2 \theta_k \|\nabla f_k\|^2 < \infty$$

which implies that  $\lim_{k \rightarrow \infty} \|\nabla f_k\| = 0$ .

## Behavior of BFGS

- It may happen that  $\tilde{H}_k$  becomes a poor approximation of the Hessian  $H_k$ . If, e.g.,  $y_k^\top$  is tiny, then  $\tilde{H}_{k+1}$  will be huge.

However, it has been proven experimentally that if  $\tilde{H}_k$  wrongly estimates the curvature of  $f$  and this estimate slows down the iteration, then the approximation will tend to correct the bad Hessian approximations.

The above self-correction works only if an appropriate line search is performed (strong Wolfe conditions).

## Behavior of BFGS

- ▶ It may happen that  $\tilde{H}_k$  becomes a poor approximation of the Hessian  $H_k$ . If, e.g.,  $y_k^\top$  is tiny, then  $\tilde{H}_{k+1}$  will be huge.

However, it has been proven experimentally that if  $\tilde{H}_k$  wrongly estimates the curvature of  $f$  and this estimate slows down the iteration, then the approximation will tend to correct the bad Hessian approximations.

The above self-correction works only if an appropriate line search is performed (strong Wolfe conditions).

- ▶ There are more sophisticated ways of setting the initial Hessian approximation  $H_0$ .

See Numerical Optimization, Nocedal & Wright, page 201.

## Quasi-Newton Methods - Comments

- ▶ Each iteration is performed for  $\mathcal{O}(n^2)$  operations as opposed to  $\mathcal{O}(n^3)$  for methods involving solutions of linear systems.

## Quasi-Newton Methods - Comments

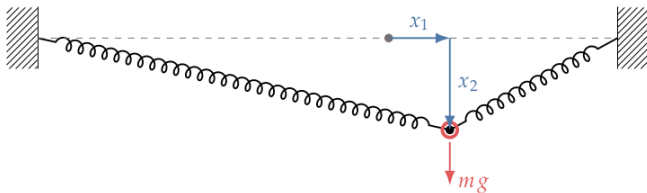
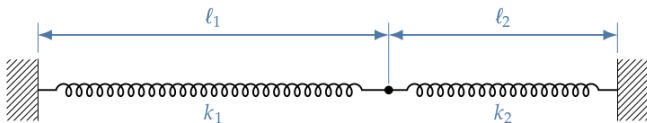
- ▶ Each iteration is performed for  $\mathcal{O}(n^2)$  operations as opposed to  $\mathcal{O}(n^3)$  for methods involving solutions of linear systems.
- ▶ There is even a memory-limited variant (L-BFGS) that uses only information from past  $m$  steps, and its single iteration complexity is  $\mathcal{O}(mn)$ .

## Quasi-Newton Methods - Comments

- ▶ Each iteration is performed for  $\mathcal{O}(n^2)$  operations as opposed to  $\mathcal{O}(n^3)$  for methods involving solutions of linear systems.
- ▶ There is even a memory-limited variant (L-BFGS) that uses only information from past  $m$  steps, and its single iteration complexity is  $\mathcal{O}(mn)$ .
- ▶ Compared with Newton's method, no second derivatives are computed.

## Quasi-Newton Methods - Comments

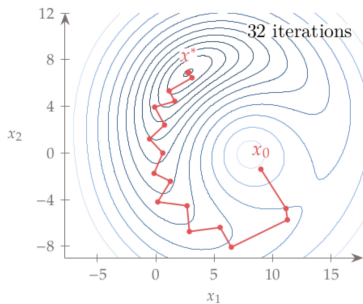
- ▶ Each iteration is performed for  $\mathcal{O}(n^2)$  operations as opposed to  $\mathcal{O}(n^3)$  for methods involving solutions of linear systems.
- ▶ There is even a memory-limited variant (L-BFGS) that uses only information from past  $m$  steps, and its single iteration complexity is  $\mathcal{O}(mn)$ .
- ▶ Compared with Newton's method, no second derivatives are computed.
- ▶ Local superlinear convergence can be proved under specific conditions.  
Compare with local quadratic convergence of Newton's method and linear convergence of gradient descent.



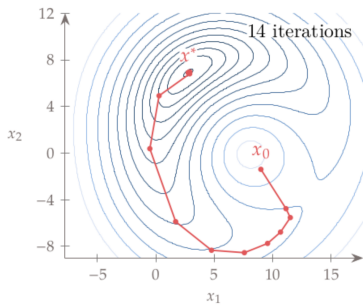
$$f(x_1, x_2) = \frac{1}{2}k_1 \left( \sqrt{(\ell_1 + x_1)^2 + x_2^2} - \ell_1 \right)^2 + \frac{1}{2}k_2 \left( \sqrt{(\ell_2 - x_1)^2 + x_2^2} - \ell_2 \right)^2 - mgx_2$$

Here  $\ell_1 = 12, \ell_2 = 8, k_1 = 1, k_2 = 10, mg = 7$

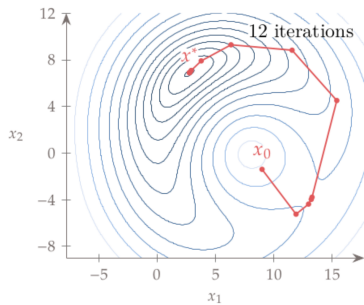




Steepest descent

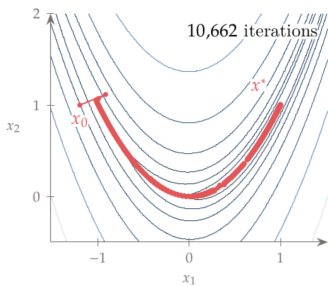


Quasi-Newton

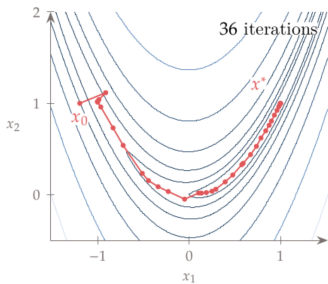


Newton

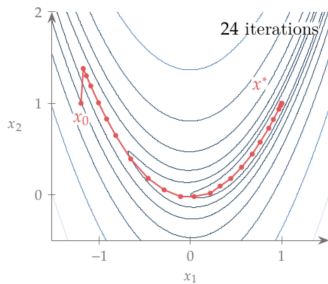
*Rosenbrock:*  $f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$



Steepest descent



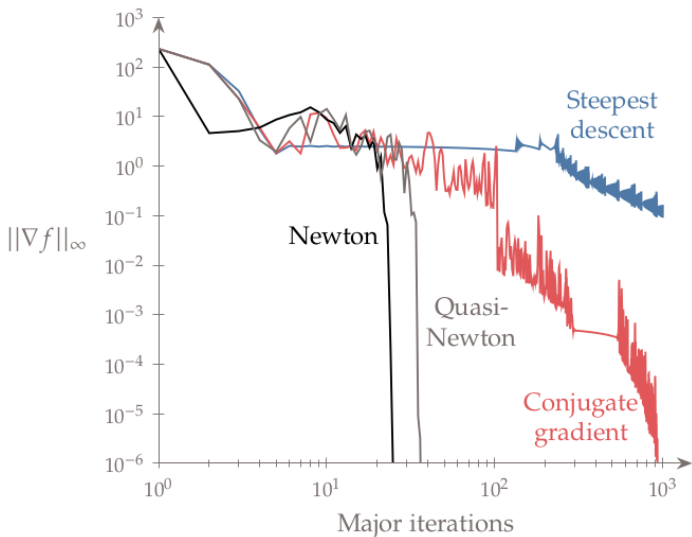
Quasi-Newton



Newton

## Rosenbrock:

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$



# Computational Complexity

Algorithm	Computational Complexity
Steepest Descent	$O(n)$ per iteration
Newton's Method	$O(n^3)$ to compute Hessian and solve system
BFGS	$O(n^2)$ to update Hessian approximation

**Table:** Summary of the computational complexity for each optimization algorithm.

- ▶ Steepest Descent: Simple but often slow, requiring many iterations.
- ▶ Newton's Method: Fast convergence but expensive per iteration.
- ▶ BFGS: Quasi-Newton, no Hessian needed, good speed and iteration count balance.

This slide was a test of ChatGPT 4.0. The prompt was something like "Give me a beamer slide with complexity analysis of gradient descent, Newton's method, BFGS." The algorithm preferred steepest descent though.

# Constrained Optimization

# Constrained Optimization Problem

Recall that the constrained optimization problem is

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{by varying} & x \\ \text{subject to} & g_i(x) \leq 0 \quad i = 1, \dots, n_g \\ & h_j(x) = 0 \quad j = 1, \dots, n_h\end{array}$$

$x^*$  is now a *constrained minimizer* if

$$f(x^*) \leq f(x) \quad \text{for all} \quad x \in \mathcal{F}$$

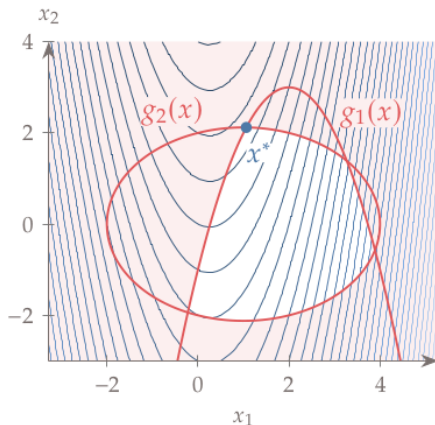
where  $\mathcal{F}$  is the feasibility region

$$\mathcal{F} = \{x \mid g_i(x) \leq 0, h_j(x) = 0, i = 1, \dots, n_g, j = 1, \dots, n_h\}$$

Thus, to find a constrained minimizer, we have to inspect unconstrained minima of  $f$  inside of  $\mathcal{F}$  and points along the boundary of  $\mathcal{F}$ .

# COP - Example

$$\begin{array}{ll}\text{minimize}_{x_1, x_2} & f(x_1, x_2) = x_1^2 - \frac{1}{2}x_1 - x_2 - 2 \\ \text{subject to} & g_1(x_1, x_2) = x_1^2 - 4x_1 + x_2 + 1 \leq 0 \\ & g_2(x_1, x_2) = \frac{1}{2}x_1^2 + x_2^2 - x_1 - 4 \leq 0\end{array}$$



# Equality Constraints

Let us restrict our problem only to the equality constraints:

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{by varying} & x \\ \text{subject to} & h_j(x) = 0 \quad j = 1, \dots, n_h\end{array}$$

Assume that  $f$  and  $h_j$  have continuous second derivatives.

Now, we try to imitate the theory from the unconstrained case and characterize minima using gradients.

This time, we must consider the gradients of  $f$  and  $h_j$ .



## Unconstrained Minimizer

Consider the first-order Taylor approximation of  $f$  at  $x$

$$f(x + p) \approx f(x) + \nabla f(x)^\top p$$

## Unconstrained Minimizer

Consider the first-order Taylor approximation of  $f$  at  $x$

$$f(x + p) \approx f(x) + \nabla f(x)^\top p$$

Note that if  $x^*$  is an unconstrained minimizer of  $f$ , then

$$f(x^* + p) \geq f(x^*)$$

for all  $p$  small enough.

## Unconstrained Minimizer

Consider the first-order Taylor approximation of  $f$  at  $x$

$$f(x + p) \approx f(x) + \nabla f(x)^\top p$$

Note that if  $x^*$  is an unconstrained minimizer of  $f$ , then

$$f(x^* + p) \geq f(x^*)$$

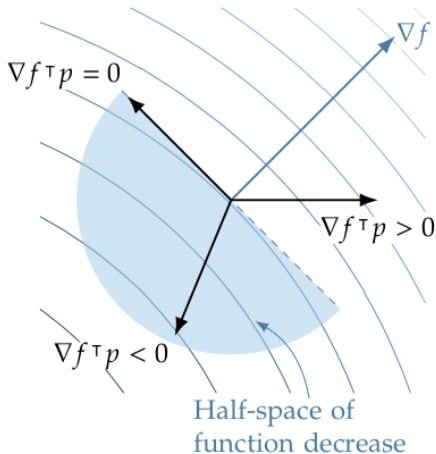
for all  $p$  small enough.

Together with the Taylor approximation, we obtain

$$f(x^*) + \nabla f(x^*)^\top p \geq f(x^*)$$

and hence

$$\nabla f(x^*)^\top p \geq 0$$



The hyperplane defined by  $\nabla f^\top p = 0$  contains directions  $p$  of zero variation in  $f$ .

In the unconstrained case,  $x^*$  is minimizer only if  $\nabla f(x^*) = 0$  because otherwise there would be a direction  $p$  satisfying  $\nabla f(x^*)p < 0$ , a *decrease direction*.

## Decrease Direction in COP

In COP,  $p$  is a decrease direction in  $x \in \mathcal{F}$  if  $\nabla f(x)^\top p < 0$  and if  $p$  is a *feasible direction*!

That is, points into the feasible region.

## Decrease Direction in COP

In COP,  $p$  is a decrease direction in  $x \in \mathcal{F}$  if  $\nabla f(x)^\top p < 0$  and if  $p$  is a *feasible direction*!

That is, points into the feasible region.

How do we characterize feasible directions?

## Decrease Direction in COP

In COP,  $p$  is a decrease direction in  $x \in \mathcal{F}$  if  $\nabla f(x)^\top p < 0$  and if  $p$  is a *feasible direction*!

That is, points into the feasible region.

How do we characterize feasible directions?

Consider Taylor approximation of  $h_j$  for all  $j$ :

$$h_j(x + p) \approx h_j(x) + \nabla h_j(x)^\top p$$

## Decrease Direction in COP

In COP,  $p$  is a decrease direction in  $x \in \mathcal{F}$  if  $\nabla f(x)^\top p < 0$  and if  $p$  is a *feasible direction*!

That is, points into the feasible region.

How do we characterize feasible directions?

Consider Taylor approximation of  $h_j$  for all  $j$ :

$$h_j(x + p) \approx h_j(x) + \nabla h_j(x)^\top p$$

Assuming  $x \in \mathcal{F}$ , we have  $h_j(x) = 0$  for all  $j$  and thus

$$h_j(x + p) \approx \nabla h_j(x)^\top p$$



## Decrease Direction in COP

In COP,  $p$  is a decrease direction in  $x \in \mathcal{F}$  if  $\nabla f(x)^\top p < 0$  and if  $p$  is a *feasible direction*!

That is, points into the feasible region.

How do we characterize feasible directions?

Consider Taylor approximation of  $h_j$  for all  $j$ :

$$h_j(x + p) \approx h_j(x) + \nabla h_j(x)^\top p$$

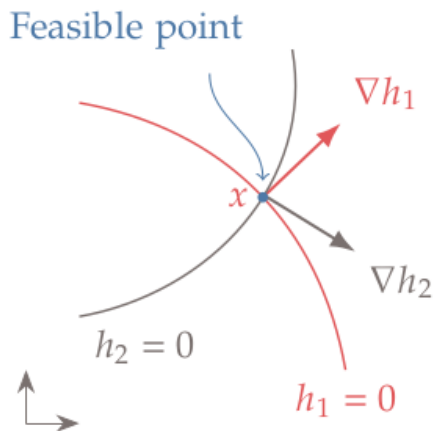
Assuming  $x \in \mathcal{F}$ , we have  $h_j(x) = 0$  for all  $j$  and thus

$$h_j(x + p) \approx \nabla h_j(x)^\top p$$

As  $p$  is a feasible direction iff  $h_j(x + p) = 0$ , we obtain that

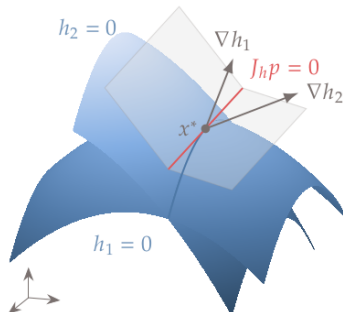
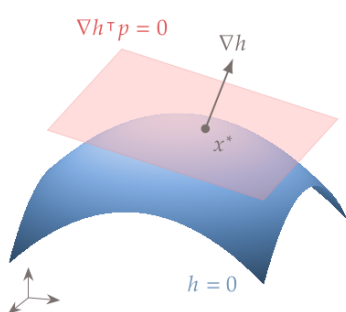
$$p \text{ is a } \textit{feasible direction} \quad \text{iff} \quad \nabla h_j(x)^\top p = 0 \text{ for all } j$$

# Feasible Points and Directions



Here, the only feasible direction at  $x$  is  $p = 0$ .

# Feasible Points and Directions



Here the feasible directions at  $x^*$  point along the red line, i.e.,

$$\nabla h_1(x^*)^\top p = 0 \quad \nabla h_2(x^*)^\top p = 0$$

## Necessary Condition for Constrained Minima

Consider a direction  $p$ . Observe that

- ▶ If  $\nabla h_j(x)^\top p \neq 0$ , then moving a short step in the direction  $p$  violates the constraint  $h_j(x) = 0$ .

# Necessary Condition for Constrained Minima

Consider a direction  $p$ . Observe that

- ▶ If  $\nabla h_j(x)^\top p \neq 0$ , then moving a short step in the direction  $p$  violates the constraint  $h_j(x) = 0$ .
- ▶ If  $\nabla h_j(x)^\top p = 0$  for all  $j$  and
  - ▶  $\nabla f(x)^\top p > 0$ , then moving a short step in the direction  $p$  increases  $f$  and stays in  $\mathcal{F}$ .

# Necessary Condition for Constrained Minima

Consider a direction  $p$ . Observe that

- ▶ If  $\nabla h_j(x)^\top p \neq 0$ , then moving a short step in the direction  $p$  violates the constraint  $h_j(x) = 0$ .
- ▶ If  $\nabla h_j(x)^\top p = 0$  for all  $j$  and
  - ▶  $\nabla f(x)^\top p > 0$ , then moving a short step in the direction  $p$  increases  $f$  and stays in  $\mathcal{F}$ .
  - ▶  $\nabla f(x)^\top p < 0$ , then moving a short step in the direction  $p$  decreases  $f$  and stays in  $\mathcal{F}$ .

# Necessary Condition for Constrained Minima

Consider a direction  $p$ . Observe that

- ▶ If  $\nabla h_j(x)^\top p \neq 0$ , then moving a short step in the direction  $p$  violates the constraint  $h_j(x) = 0$ .
- ▶ If  $\nabla h_j(x)^\top p = 0$  for all  $j$  and
  - ▶  $\nabla f(x)^\top p > 0$ , then moving a short step in the direction  $p$  increases  $f$  and stays in  $\mathcal{F}$ .
  - ▶  $\nabla f(x)^\top p < 0$ , then moving a short step in the direction  $p$  decreases  $f$  and stays in  $\mathcal{F}$ .
  - ▶  $\nabla f(x)^\top p = 0$ , then moving a short step in the direction  $p$  does not change  $f$  and stays  $\mathcal{F}$ .

## Necessary Condition for Constrained Minima

Consider a direction  $p$ . Observe that

- ▶ If  $\nabla h_j(x)^\top p \neq 0$ , then moving a short step in the direction  $p$  violates the constraint  $h_j(x) = 0$ .
- ▶ If  $\nabla h_j(x)^\top p = 0$  for all  $j$  and
  - ▶  $\nabla f(x)^\top p > 0$ , then moving a short step in the direction  $p$  increases  $f$  and stays in  $\mathcal{F}$ .
  - ▶  $\nabla f(x)^\top p < 0$ , then moving a short step in the direction  $p$  decreases  $f$  and stays in  $\mathcal{F}$ .
  - ▶  $\nabla f(x)^\top p = 0$ , then moving a short step in the direction  $p$  does not change  $f$  and stays  $\mathcal{F}$ .

To be a minimizer,  $x^*$  must be feasible and every direction satisfying  $\nabla h_j(x^*)^\top p = 0$  for all  $j$  must also satisfy  $\nabla f(x^*)^\top p \geq 0$ .



## Necessary Condition for Constrained Minima

Consider a direction  $p$ . Observe that

- ▶ If  $\nabla h_j(x)^\top p \neq 0$ , then moving a short step in the direction  $p$  violates the constraint  $h_j(x) = 0$ .
- ▶ If  $\nabla h_j(x)^\top p = 0$  for all  $j$  and
  - ▶  $\nabla f(x)^\top p > 0$ , then moving a short step in the direction  $p$  increases  $f$  and stays in  $\mathcal{F}$ .
  - ▶  $\nabla f(x)^\top p < 0$ , then moving a short step in the direction  $p$  decreases  $f$  and stays in  $\mathcal{F}$ .
  - ▶  $\nabla f(x)^\top p = 0$ , then moving a short step in the direction  $p$  does not change  $f$  and stays  $\mathcal{F}$ .

To be a minimizer,  $x^*$  must be feasible and every direction satisfying  $\nabla h_j(x^*)^\top p = 0$  for all  $j$  must also satisfy  $\nabla f(x^*)^\top p \geq 0$ .

Note that if  $p$  is a feasible direction, then  $-p$  is also, and thus  $\nabla f(x^*)^\top (-p) \geq 0$ . So finally,

## Necessary Condition for Constrained Minima

Consider a direction  $p$ . Observe that

- ▶ If  $\nabla h_j(x)^\top p \neq 0$ , then moving a short step in the direction  $p$  violates the constraint  $h_j(x) = 0$ .
- ▶ If  $\nabla h_j(x)^\top p = 0$  for all  $j$  and
  - ▶  $\nabla f(x)^\top p > 0$ , then moving a short step in the direction  $p$  increases  $f$  and stays in  $\mathcal{F}$ .
  - ▶  $\nabla f(x)^\top p < 0$ , then moving a short step in the direction  $p$  decreases  $f$  and stays in  $\mathcal{F}$ .
  - ▶  $\nabla f(x)^\top p = 0$ , then moving a short step in the direction  $p$  does not change  $f$  and stays  $\mathcal{F}$ .

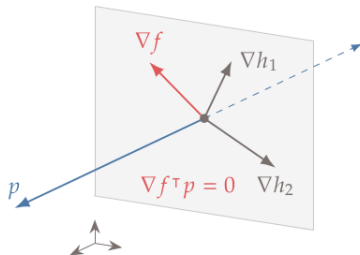
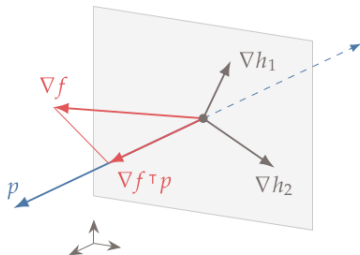
To be a minimizer,  $x^*$  must be feasible and every direction satisfying  $\nabla h_j(x^*)^\top p = 0$  for all  $j$  must also satisfy  $\nabla f(x^*)^\top p \geq 0$ .

Note that if  $p$  is a feasible direction, then  $-p$  is also, and thus  $\nabla f(x^*)^\top (-p) \geq 0$ . So finally,

If  $x^*$  is a *constrained minimizer*, then

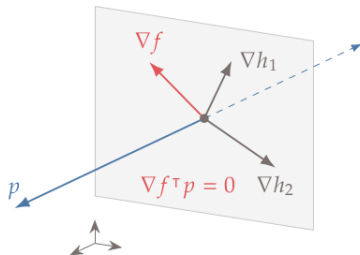
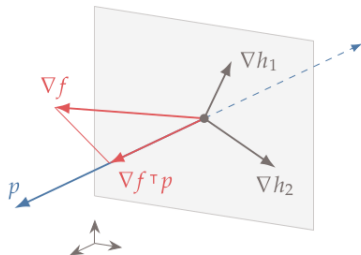
$$\nabla f(x^*)^\top p = 0 \text{ for all } p \text{ satisfying } (\forall j : \nabla h_j(x^*)^\top p = 0)$$

# Lagrange Multipliers



**Left:**  $f$  increases along  $p$ . **Right:**  $f$  does not change along  $p$ .

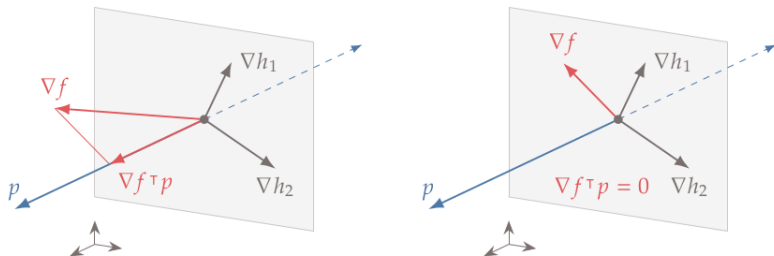
# Lagrange Multipliers



**Left:**  $f$  increases along  $p$ . **Right:**  $f$  does not change along  $p$ .

Observe that at an optimum,  $\nabla f$  lies in the space spanned by the gradients of constraint functions.

# Lagrange Multipliers



**Left:**  $f$  increases along  $p$ . **Right:**  $f$  does not change along  $p$ .

Observe that at an optimum,  $\nabla f$  lies in the space spanned by the gradients of constraint functions.

There are *Lagrange multipliers*  $\lambda_1, \lambda_2$  satisfying

$$\nabla f(x^*) = -(\lambda_1 \nabla h_1 + \lambda_2 \nabla h_2)$$

The minus sign is arbitrary for equality constraints but will be significant when dealing with inequality constraints.

## Lagrange Multipliers

We know that if  $x^*$  is a constrained minimizer, then.

$$\nabla f(x^*)^\top p = 0 \text{ for all } p \text{ satisfying } (\forall j : \nabla h_j(x^*)^\top p = 0)$$

## Lagrange Multipliers

We know that if  $x^*$  is a constrained minimizer, then.

$$\nabla f(x^*)^\top p = 0 \text{ for all } p \text{ satisfying } (\forall j : \nabla h_j(x^*)^\top p = 0)$$

But then, from the geometry of the problem, we obtain

### Theorem 16

*Consider the COP with only equality constraints and  $f$  and all  $h_j$  twice continuously differentiable.*

*Assume that  $x^*$  is a constrained minimizer and that  $x^*$  is regular, which means that  $\nabla h_j(x^*)$  are linearly independent.*

*Then there are  $\lambda_1, \dots, \lambda_{n_h} \in \mathbb{R}$  satisfying*

$$\nabla f(x^*) = - \sum_{j=1}^{n_h} \lambda_j \nabla h_j(x^*)$$

The coefficients  $\lambda_1, \dots, \lambda_{n_h}$  are called *Lagrange multipliers*.

## Lagrangian Function

Try to transform the constrained problem into an unconstrained one by moving the constraints  $h_j(x) = 0$  into the objective.



## Lagrangian Function

Try to transform the constrained problem into an unconstrained one by moving the constraints  $h_j(x) = 0$  into the objective.

Consider *Lagrangian function*  $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \rightarrow \mathbb{R}$  defined by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \quad \text{here} \quad h(x) = (h_1(x), \dots, h_{n_h}(x))^\top$$

## Lagrangian Function

Try to transform the constrained problem into an unconstrained one by moving the constraints  $h_j(x) = 0$  into the objective.

Consider *Lagrangian function*  $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \rightarrow \mathbb{R}$  defined by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \quad \text{here} \quad h(x) = (h_1(x), \dots, h_{n_h}(x))^\top$$

Note that the stationary point of  $\mathcal{L}$  gives us the Lagrange multipliers:

$$\nabla_x \mathcal{L} = \nabla f(x) + \sum_{j=1}^{n_h} \lambda_j \nabla h_j(x)$$

$$\nabla_\lambda \mathcal{L} = h(x)$$

## Lagrangian Function

Try to transform the constrained problem into an unconstrained one by moving the constraints  $h_j(x) = 0$  into the objective.

Consider *Lagrangian function*  $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \rightarrow \mathbb{R}$  defined by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \quad \text{here} \quad h(x) = (h_1(x), \dots, h_{n_h}(x))^\top$$

Note that the stationary point of  $\mathcal{L}$  gives us the Lagrange multipliers:

$$\nabla_x \mathcal{L} = \nabla f(x) + \sum_{j=1}^{n_h} \lambda_j \nabla h_j(x)$$

$$\nabla_\lambda \mathcal{L} = h(x)$$

Now putting  $\nabla \mathcal{L}(x) = 0$ , we obtain precisely the above properties of the constrained minimizer:

$$h(x) = 0 \quad \text{and} \quad \nabla f(x) = - \sum_{j=1}^{n_h} \lambda_j \nabla h_j(x)$$

So we can now use methods for searching stationary points. This will lead to the Lagrange-Newton method.

$$\begin{array}{ll} \underset{x_1, x_2}{\text{minimize}} & f(x_1, x_2) = x_1 + 2x_2 \\ \text{subject to} & h(x_1, x_2) = \frac{1}{4}x_1^2 + x_2^2 - 1 = 0 \end{array}$$

The Lagrangian function

$$\mathcal{L}(x_1, x_2, \lambda) = x_1 + 2x_2 + \lambda \left( \frac{1}{4}x_1^2 + x_2^2 - 1 \right)$$

$$\begin{array}{ll} \underset{x_1, x_2}{\text{minimize}} & f(x_1, x_2) = x_1 + 2x_2 \\ \text{subject to} & h(x_1, x_2) = \frac{1}{4}x_1^2 + x_2^2 - 1 = 0 \end{array}$$

The Lagrangian function

$$\mathcal{L}(x_1, x_2, \lambda) = x_1 + 2x_2 + \lambda \left( \frac{1}{4}x_1^2 + x_2^2 - 1 \right)$$

Differentiating this to get the first-order optimality conditions,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial x_1} &= 1 + \frac{1}{2}\lambda x_1 = 0 & \frac{\partial \mathcal{L}}{\partial x_2} &= 2 + 2\lambda x_2 = 0 \\ \frac{\partial \mathcal{L}}{\partial \lambda} &= \frac{1}{4}x_1^2 + x_2^2 - 1 = 0. \end{aligned}$$

$$\begin{array}{ll} \underset{x_1, x_2}{\text{minimize}} & f(x_1, x_2) = x_1 + 2x_2 \\ \text{subject to} & h(x_1, x_2) = \frac{1}{4}x_1^2 + x_2^2 - 1 = 0 \end{array}$$

The Lagrangian function

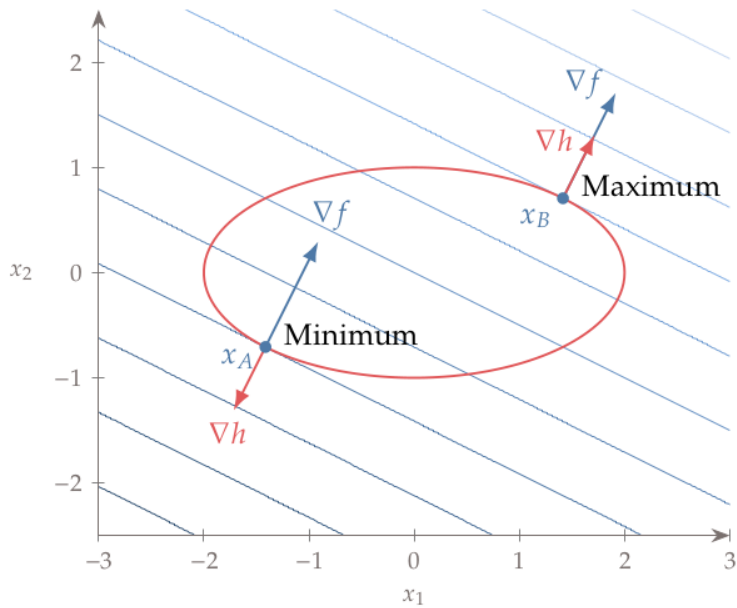
$$\mathcal{L}(x_1, x_2, \lambda) = x_1 + 2x_2 + \lambda \left( \frac{1}{4}x_1^2 + x_2^2 - 1 \right)$$

Differentiating this to get the first-order optimality conditions,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial x_1} &= 1 + \frac{1}{2}\lambda x_1 = 0 & \frac{\partial \mathcal{L}}{\partial x_2} &= 2 + 2\lambda x_2 = 0 \\ \frac{\partial \mathcal{L}}{\partial \lambda} &= \frac{1}{4}x_1^2 + x_2^2 - 1 = 0. \end{aligned}$$

Solving these three equations for the three unknowns  $(x_1, x_2, \lambda)$ , we obtain two possible solutions:

$$\begin{aligned} x_A &= (x_1, x_2) = (-\sqrt{2}, -\sqrt{2}/2), & \lambda_A &= \sqrt{2} \\ x_B &= (x_1, x_2) = (\sqrt{2}, \sqrt{2}/2), & \lambda_B &= -\sqrt{2} \end{aligned}$$



## Second-Order Sufficient Conditions

As in the unconstrained case, the first-order conditions characterize any “stable” point (minimum, maximum, saddle).



## Second-Order Sufficient Conditions

As in the unconstrained case, the first-order conditions characterize any “stable” point (minimum, maximum, saddle).

Consider *Lagrangian Hessian*:

$$H(x, \lambda) = H_f(x) + \sum_{j=1}^{n_h} \lambda_j H_{h_j}(x)$$

Here  $H_f$  is the Hessian of  $f$ , and each  $H_{h_j}$  is the Hessian of  $h_j$ .

Note that Lagrangian Hessian is NOT the Hessian of the Lagrangian!

## Second-Order Sufficient Conditions

As in the unconstrained case, the first-order conditions characterize any “stable” point (minimum, maximum, saddle).

Consider *Lagrangian Hessian*:

$$H(x, \lambda) = H_f(x) + \sum_{j=1}^{n_h} \lambda_j H_{h_j}(x)$$

Here  $H_f$  is the Hessian of  $f$ , and each  $H_{h_j}$  is the Hessian of  $h_j$ .

Note that Lagrangian Hessian is NOT the Hessian of the Lagrangian!

The second-order sufficient conditions are as follows: Assume  $x^*$  is regular and feasible. Also, assume that there is  $\lambda^*$  s.t.

$$\nabla f(x^*) = \sum_{j=1}^{n_h} -\lambda_j^* \nabla h_j(x^*)$$

## Second-Order Sufficient Conditions

As in the unconstrained case, the first-order conditions characterize any “stable” point (minimum, maximum, saddle).

Consider *Lagrangian Hessian*:

$$H(x, \lambda) = H_f(x) + \sum_{j=1}^{n_h} \lambda_j H_{h_j}(x)$$

Here  $H_f$  is the Hessian of  $f$ , and each  $H_{h_j}$  is the Hessian of  $h_j$ .

Note that Lagrangian Hessian is NOT the Hessian of the Lagrangian!

The second-order sufficient conditions are as follows: Assume  $x^*$  is regular and feasible. Also, assume that there is  $\lambda^*$  s.t.

$$\nabla f(x^*) = \sum_{j=1}^{n_h} -\lambda_j^* \nabla h_j(x^*)$$

and that

$$p^\top H(x^*, \lambda^*) p > 0 \text{ for all } p \text{ satisfying } (\forall j : \nabla h_j(x^*)^\top p = 0)$$

Then,  $x^*$  is a constrained minimizer of  $f$ .

# Inequality Constraints

Recall that the constrained optimization problem is

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{by varying} & x \\ \text{subject to} & g_i(x) \leq 0 \quad i = 1, \dots, n_g \\ & h_j(x) = 0 \quad j = 1, \dots, n_h\end{array}$$

# Inequality Constraints

Recall that the constrained optimization problem is

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{by varying} & x \\ \text{subject to} & g_i(x) \leq 0 \quad i = 1, \dots, n_g \\ & h_j(x) = 0 \quad j = 1, \dots, n_h\end{array}$$

Lagrange multipliers and the Lagrangian function can be extended to deal with inequality constraints.

The resulting necessary conditions for constrained minima are called Karush-Tucker-Kuhn (KKT) conditions.

In this course, Lagrange methods are considered only for equality-constrained problems. So, we omit further discussion of KKT.

# Constrained Optimization

## Penalty Methods

## Penalty methods

**The idea:** Transform a constrained problem into an unconstrained one by adding a penalty to the objective function when constraints are violated or close to being violated.

## Penalty methods

**The idea:** Transform a constrained problem into an unconstrained one by adding a penalty to the objective function when constraints are violated or close to being violated.

Assuming an objective function  $f$ , the penalized objective is of the form

$$\hat{f}(x) = f(x) + \mu\pi(x)$$

Here,  $\mu$  is a fixed constant determining how strong the penalty should be, and  $\pi$  is the penalty function.



## Penalty methods

**The idea:** Transform a constrained problem into an unconstrained one by adding a penalty to the objective function when constraints are violated or close to being violated.

Assuming an objective function  $f$ , the penalized objective is of the form

$$\hat{f}(x) = f(x) + \mu\pi(x)$$

Here,  $\mu$  is a fixed constant determining how strong the penalty should be, and  $\pi$  is the penalty function.

Now we may apply the unconstrained optimization methods (e.g., L-BFGS) to  $\hat{f}$  and obtain an approximation of a minimizer of  $f$ .

## Penalty methods

**The idea:** Transform a constrained problem into an unconstrained one by adding a penalty to the objective function when constraints are violated or close to being violated.

Assuming an objective function  $f$ , the penalized objective is of the form

$$\hat{f}(x) = f(x) + \mu\pi(x)$$

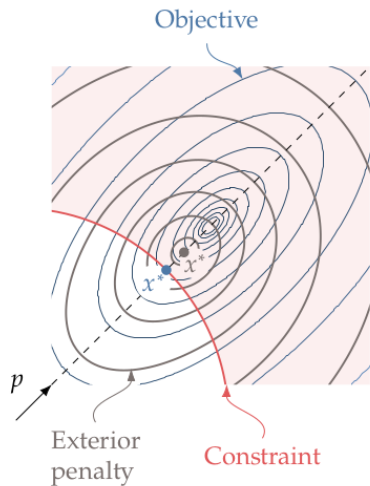
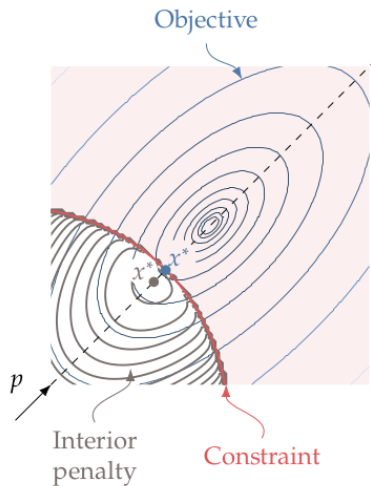
Here,  $\mu$  is a fixed constant determining how strong the penalty should be, and  $\pi$  is the penalty function.

Now we may apply the unconstrained optimization methods (e.g., L-BFGS) to  $\hat{f}$  and obtain an approximation of a minimizer of  $f$ .

There are two kinds of penalty methods:

- ▶ *exterior* - penalizing infeasible  $x$
- ▶ *interior* - penalizing  $x$  close to being infeasible

# Interior vs Exterior Penalty



# Exterior Penalty Methods - Quadratic Penalty

Consider equality-constrained problems:

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{by varying} & x \\ \text{subject to} & h_j(x) = 0 \quad j = 1, \dots, n_h\end{array}$$

# Exterior Penalty Methods - Quadratic Penalty

Consider equality-constrained problems:

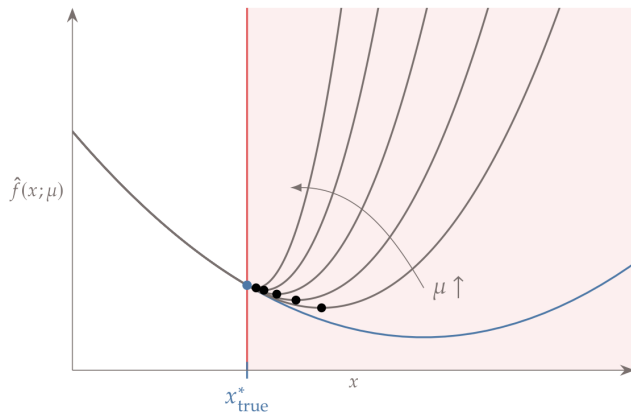
$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{by varying} & x \\ \text{subject to} & h_j(x) = 0 \quad j = 1, \dots, n_h\end{array}$$

Consider *quadratic penalty*:

$$\hat{f}(x; \mu) = f(x) + \frac{\mu}{2} \sum_{j=1}^{n_h} h_j(x)^2$$

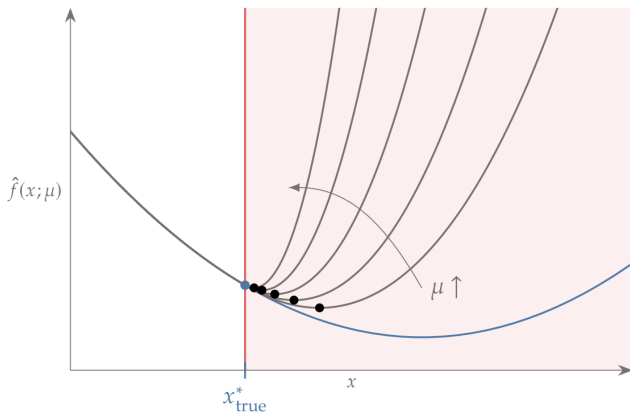
If  $f$  is continuously differentiable,  $\hat{f}$  is as well (w.r.t.  $x$ ).

# Quadratic Penalty



The true solution would be recovered for  $\mu = \infty$ .

# Quadratic Penalty



The true solution would be recovered for  $\mu = \infty$ .

However, large  $\mu$  means large condition number of the Hessian of  $\hat{f}$   
Intuitively, large curvature of  $\hat{f}$ , not good for optimization.

Need to choose  $\mu$  carefully, possibly iteratively.

---

**Algorithm 13** Exterior Penalty Method

---

- 1: Choose starting point  $x_0$
  - 2: Choose an initial penalty parameter  $\mu_0$
  - 3: Choose a penalty increase factor  $\rho > 1$
  - 4:  $k \leftarrow 0$
  - 5: **repeat**
  - 6:      $x_{k+1} \leftarrow x$  minimizing  $\hat{f}(x; \mu_k)$
  - 7:      $\mu_{k+1} \leftarrow \rho \mu_k$
  - 8:      $k \leftarrow k + 1$
  - 9: **until** convergence
-



# Convergence of Quadratic Penalty Method

## Theorem 17

*Assume that  $f$  and all  $h_j$  have continuous second derivatives.*

*Suppose that each  $x_k$  is the exact global minimizer of  $\hat{f}(x; \mu_k)$  and that  $\lim_{k \rightarrow \infty} \mu_k = \infty$ . Then, every limit point  $x^*$  of the sequence  $\{x_k\}$  solves the constrained optimization problem.*

In practice, inexact methods are used to minimize  $\hat{f}(x; \mu_k)$

# Convergence of Quadratic Penalty Method

## Theorem 17

*Assume that  $f$  and all  $h_j$  have continuous second derivatives.*

*Suppose that each  $x_k$  is the exact global minimizer of  $\hat{f}(x; \mu_k)$  and that  $\lim_{k \rightarrow \infty} \mu_k = \infty$ . Then, every limit point  $x^*$  of the sequence  $\{x_k\}$  solves the constrained optimization problem.*

In practice, inexact methods are used to minimize  $\hat{f}(x; \mu_k)$

Let  $x^*$  be a limit point of  $x_k$  and let  $\lambda^*$  be such that  $(x^*, \lambda^*)$  satisfy the Lagrange conditions for the constrained problem.

# Convergence of Quadratic Penalty Method

## Theorem 17

*Assume that  $f$  and all  $h_j$  have continuous second derivatives.*

*Suppose that each  $x_k$  is the exact global minimizer of  $\hat{f}(x; \mu_k)$  and that  $\lim_{k \rightarrow \infty} \mu_k = \infty$ . Then, every limit point  $x^*$  of the sequence  $\{x_k\}$  solves the constrained optimization problem.*

In practice, inexact methods are used to minimize  $\hat{f}(x; \mu_k)$

Let  $x^*$  be a limit point of  $x_k$  and let  $\lambda^*$  be such that  $(x^*, \lambda^*)$  satisfy the Lagrange conditions for the constrained problem.

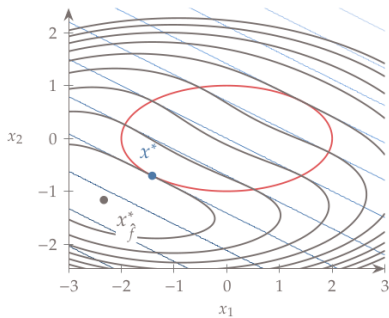
Then, for a subsequence of points  $x_k$ , which converges to  $x^*$ , we have that

$$\lim_{k \rightarrow \infty} -\mu_k h_j(x_k) = \lambda_j^*$$

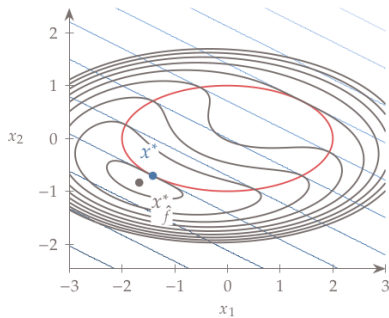
# Practical Problems

- ▶ Small  $\mu$  may result in so weak penalty that  $f$  unbounded below results in  $\hat{f}$  unbounded as well
- ▶ As  $\mu = \infty$  is impossible, the solution is always slightly infeasible
- ▶ Growing curvature of  $\hat{f}$  as  $\mu$  grows makes the Hessian of  $\hat{f}$  almost singular

$$\hat{f}(x; \mu) = x_1 + 2x_2 + \frac{\mu}{2} \left( \frac{1}{4}x_1^2 + x_2^2 - 1 \right)^2$$

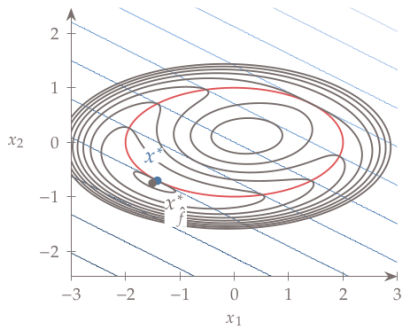


$\mu = 0.5$



$\mu = 3.0$

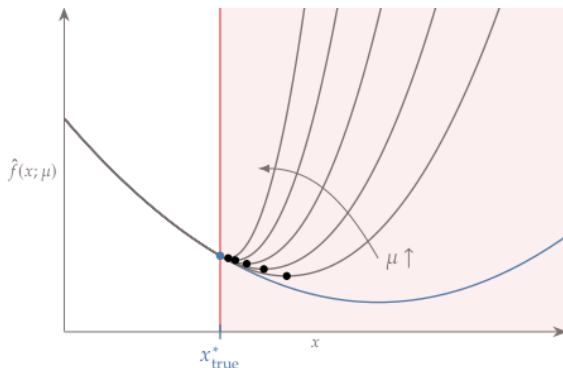
$$\hat{f}(x; \mu) = x_1 + 2x_2 + \frac{\mu}{2} \left( \frac{1}{4}x_1^2 + x_2^2 - 1 \right)^2$$



$$\mu = 10.0$$

# Quadratic Penalty for Inequality Constraints

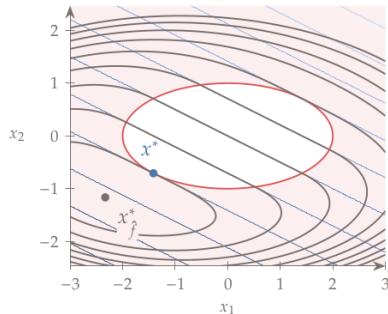
$$\hat{f}(x; \mu) = f(x) + \frac{\mu h}{2} \sum_{j=1}^{n_h} h_j(x)^2 + \frac{\mu g}{2} \sum_{i=1}^{n_g} \max(0, g_i(x))^2$$



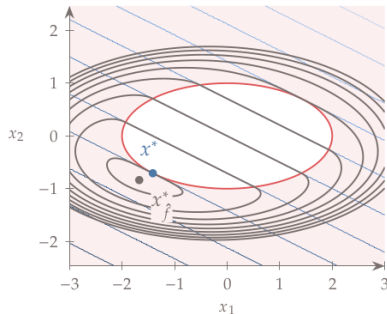
Minimizer approached from the infeasible side.

## Example

$$\hat{f}(x; \mu) = x_1 + 2x_2 + \frac{\mu}{2} \max \left( 0, \frac{1}{4}x_1^2 + x_2^2 - 1 \right)^2$$



$\mu = 0.5$

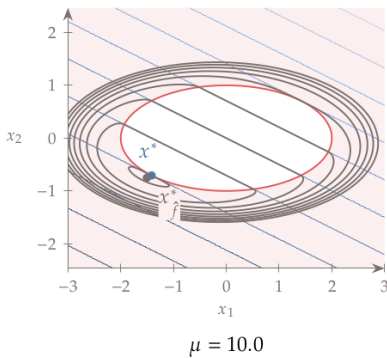


$\mu = 3.0$



## Example

$$\hat{f}(x; \mu) = x_1 + 2x_2 + \frac{\mu}{2} \max \left( 0, \frac{1}{4}x_1^2 + x_2^2 - 1 \right)^2$$



## Augmented Lagrangian (Optional)

We may augment the Lagrangian  $\mathcal{L} = f(x) + \sum_{j=1}^{n_h} \lambda_j h_j(x)$  with penalty and optimize the augmented Lagrangian

$$\hat{f}(x; \lambda, \mu) = f(x) + \sum_{j=1}^{n_h} \lambda_j h_j(x) + \frac{\mu}{2} \sum_{j=1}^{n_h} h_j(x)^2$$

## Augmented Lagrangian (Optional)

We may augment the Lagrangian  $\mathcal{L} = f(x) + \sum_{j=1}^{n_h} \lambda_j h_j(x)$  with penalty and optimize the augmented Lagrangian

$$\hat{f}(x; \lambda, \mu) = f(x) + \sum_{j=1}^{n_h} \lambda_j h_j(x) + \frac{\mu}{2} \sum_{j=1}^{n_h} h_j(x)^2$$

Note the relationship between optimality conditions for  $\mathcal{L}$  and  $\hat{f}$

$$\nabla_x \hat{f}(x; \lambda, \mu) = \nabla f(x) + \sum_{j=1}^{n_h} (\lambda_j + \mu h_j(x)) \nabla h_j(x) = 0$$

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla f(x^*) + \sum_{j=1}^{n_h} \lambda_j^* \nabla h_j(x^*) = 0.$$

## Augmented Lagrangian (Optional)

We may augment the Lagrangian  $\mathcal{L} = f(x) + \sum_{j=1}^{n_h} \lambda_j h_j(x)$  with penalty and optimize the augmented Lagrangian

$$\hat{f}(x; \lambda, \mu) = f(x) + \sum_{j=1}^{n_h} \lambda_j h_j(x) + \frac{\mu}{2} \sum_{j=1}^{n_h} h_j(x)^2$$

Note the relationship between optimality conditions for  $\mathcal{L}$  and  $\hat{f}$

$$\nabla_x \hat{f}(x; \lambda, \mu) = \nabla f(x) + \sum_{j=1}^{n_h} (\lambda_j + \mu h_j(x)) \nabla h_j(x) = 0$$

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla f(x^*) + \sum_{j=1}^{n_h} \lambda_j^* \nabla h_j(x^*) = 0.$$

Comparing these two conditions suggests an approximation:

$$\lambda_j^* \approx \lambda_j + \mu h_j.$$

# Augmented Lagrangian Penalty Method (Optional)

## Inputs:

- ▶  $x_0$ : Starting point
- ▶  $\lambda_0 = 0$ : Initial Lagrange multiplier
- ▶  $\mu_0 > 0$ : Initial penalty parameter
- ▶  $\rho > 1$ : Penalty increase factor

## Outputs:

- ▶  $x^*$ : Optimal point
- ▶  $f(x^*)$ : Corresponding function value

## Algorithm:

$k = 0$

**repeat**

$x_{k+1} \leftarrow x \text{ minimizing } \hat{f}(x; \lambda_k, \mu_k)$

$\lambda_{k+1} = \lambda_k + \mu_k h(x_k)$

$\mu_{k+1} \leftarrow \rho \mu_k$

$k \leftarrow k + 1$

**until** convergence

## Comparison of Quadratic and Lagrangian Penalty (Optional)

Compare

$$h_j \approx \frac{1}{\mu} (\lambda_j^* - \lambda_j) .$$

with the corresponding approximation of  $h_j$  in the quadratic penalty method is

$$h_j \approx \frac{\lambda_j^*}{\mu}$$

Thus, the quadratic penalty relies solely on increasing  $\mu$ .

## Comparison of Quadratic and Lagrangian Penalty (Optional)

Compare

$$h_j \approx \frac{1}{\mu} (\lambda_j^* - \lambda_j) .$$

with the corresponding approximation of  $h_j$  in the quadratic penalty method is

$$h_j \approx \frac{\lambda_j^*}{\mu}$$

Thus, the quadratic penalty relies solely on increasing  $\mu$ .

However, the augmented Lagrangian also controls the numerator via estimating  $\lambda_j$ .

If  $\lambda_j$  is close to  $\lambda_j^*$ , we may obtain a close solution for modest values of  $\mu$ .

Several variants of the Lagrangian penalty exist for inequality constraints; see Nocedal & Wright.

# Interior Penalty Methods

Always seek to maintain feasibility as opposed to the exterior methods.

Instead of adding a penalty only when constraints are violated; add a penalty as the constraint is approached from the feasible region.



# Interior Penalty Methods

Always seek to maintain feasibility as opposed to the exterior methods.

Instead of adding a penalty only when constraints are violated; add a penalty as the constraint is approached from the feasible region.

Desirable if the objective function is ill-defined outside the feasible region.

The interior methods are also referred to as *barrier methods* because the penalty function acts as a barrier preventing iterates from leaving the feasible region.

## Barrier Methods

Consider inequality-constrained problems:

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{by varying} & x \\ \text{subject to} & g_i(x) \leq 0 \quad i = 1, \dots, n_g\end{array}$$

## Barrier Methods

Consider inequality-constrained problems:

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{by varying} & x \\ \text{subject to} & g_i(x) \leq 0 \quad i = 1, \dots, n_g\end{array}$$

Minimize the augmented objective function.

$$\hat{f}(x; \mu) = f(x) + \mu \pi(x)$$

Here  $\pi$  is a penalty function.

## Barrier Methods

Consider inequality-constrained problems:

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{by varying} & x \\ \text{subject to} & g_i(x) \leq 0 \quad i = 1, \dots, n_g\end{array}$$

Minimize the augmented objective function.

$$\hat{f}(x; \mu) = f(x) + \mu\pi(x)$$

Here  $\pi$  is a penalty function.

*Inverse barrier*

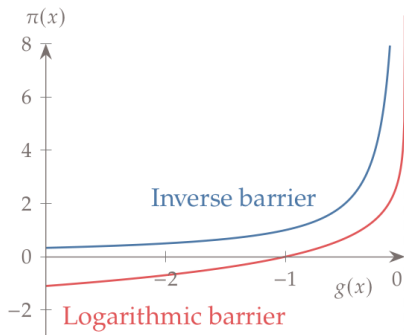
$$\pi(x) = \sum_{i=1}^{n_g} -\frac{1}{g_i(x)}$$

*Logarithmic barrier*

$$\pi(x) = \sum_{i=1}^{n_g} -\ln(-g_i(x))$$

Algorithms based on these penalties must be prevented from evaluating infeasible points.

# Barrier Methods



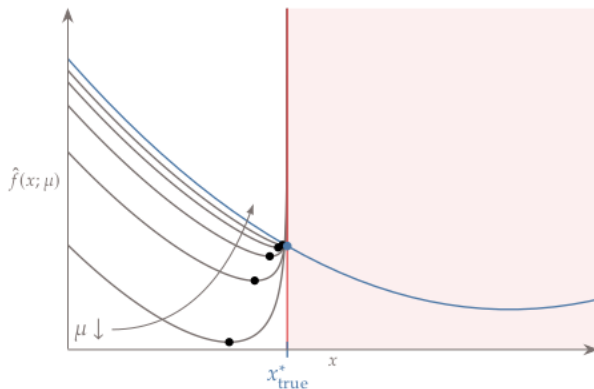
*Inverse barrier*

$$\pi(x) = \sum_{i=1}^{n_g} -\frac{1}{g_i(x)}$$

*Logarithmic barrier*

$$\pi(x) = \sum_{i=1}^{n_g} -\ln(-g_i(x))$$

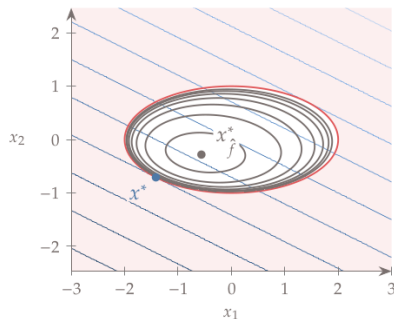
## Barrier methods



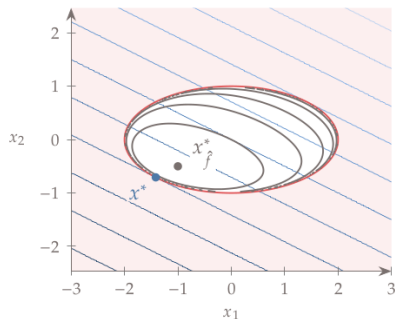
Solve a sequence of unconstrained problems for  $\hat{f}$  with  $\mu \rightarrow 0$ .

## Example

$$\hat{f}(x; \mu) = x_1 + 2x_2 - \mu \ln \left( -\frac{1}{4}x_1^2 - x_2^2 + 1 \right)$$



$\mu = 3.0$

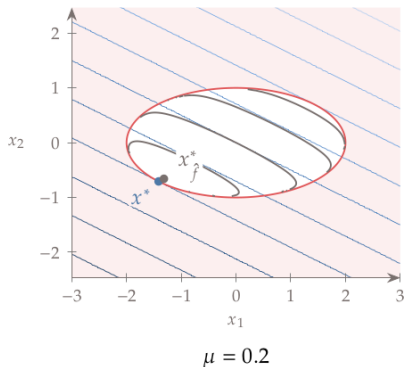


$\mu = 1.0$

As for exterior methods, the Hessian becomes increasingly ill-conditioned as  $\mu \rightarrow 0$ .

## Example

$$\hat{f}(x; \mu) = x_1 + 2x_2 - \mu \ln \left( -\frac{1}{4}x_1^2 - x_2^2 + 1 \right)$$



As for exterior methods, the Hessian becomes increasingly ill-conditioned as  $\mu \rightarrow 0$ .



## Comments on Interior Penalty Methods

Interior penalty methods must stay in the feasible region:

- ▶ Every unconstrained optimization must start at an initial point feasible for the constrained problem.
- ▶ The line search must check for feasibility and backtrack from steps to infeasible points.

# Comments on Interior Penalty Methods

Interior penalty methods must stay in the feasible region:

- ▶ Every unconstrained optimization must start at an initial point feasible for the constrained problem.
- ▶ The line search must check for feasibility and backtrack from steps to infeasible points.

Convergence issues:

- ▶ As  $\mu \rightarrow 0$  solutions of  $\hat{f}$  converge to solutions of the constrained problem.
- ▶ On the other hand, with  $\mu \rightarrow 0$  the Hessian of  $\hat{f}$  becomes increasingly ill-conditioned.

Various modifications exist to alleviate the problem with ill-conditioned Hessians.

These methods lead to a class of modern *interior point methods*.

# Summary of Penalty Methods

Penalty methods penalize approximations that either leave the feasible region (exterior methods), or are close to the border of the feasible region (interior methods).

Penalty methods are simple and easy to implement.

Both exterior and interior methods lead to ill-conditioned Hessians when approaching the correct solutions to the constrained problem.

# Constrained Optimization

## Sequential Quadratic Programming

## Quadratic Programming

The *quadratic optimization problem with equality constraints* is to

$$\begin{array}{ll}\text{minimize} & \frac{1}{2}x^{\top}Qx + q^{\top}x \\ \text{by varying} & x \\ \text{subject to} & Ax + b = 0\end{array}$$

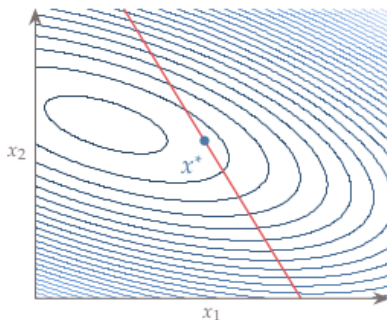
# Quadratic Programming

The *quadratic optimization problem with equality constraints* is to

$$\begin{array}{ll}\text{minimize} & \frac{1}{2}x^\top Qx + q^\top x \\ \text{by varying} & x \\ \text{subject to} & Ax + b = 0\end{array}$$

Here

- ▶  $Q$  is a  $n \times n$  symmetric matrix. For simplicity assume positive definite.
- ▶  $A$  is a  $m \times n$  matrix. Assume full rank.



# Quadratic Programming

How to solve the quadratic program?

# Quadratic Programming

How to solve the quadratic program?

Consider the Lagrangian function

$$L(x, \lambda) = \frac{1}{2}x^{\top}Qx + q^{\top}x + \lambda^{\top}(Ax + b)$$



# Quadratic Programming

How to solve the quadratic program?

Consider the Lagrangian function

$$L(x, \lambda) = \frac{1}{2}x^{\top}Qx + q^{\top}x + \lambda^{\top}(Ax + b)$$

and its partial derivatives:

$$\nabla_x L(x) = Qx + q + A^{\top}\lambda = 0$$

$$\nabla_{\lambda} L(x) = Ax + b = 0$$

# Quadratic Programming

How to solve the quadratic program?

Consider the Lagrangian function

$$L(x, \lambda) = \frac{1}{2}x^{\top}Qx + q^{\top}x + \lambda^{\top}(Ax + b)$$

and its partial derivatives:

$$\nabla_x L(x) = Qx + q + A^{\top}\lambda = 0$$

$$\nabla_{\lambda} L(x) = Ax + b = 0$$

For  $Q$  positive definite, we know that a solution to the above system is a minimizer.

So in order to solve the quadratic program, it suffices to solve the system of linear equations.

## Lagrange-Newton

Now consider an arbitrary  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  and arbitrary constraint functions  $h_j : \mathbb{R}^n \rightarrow \mathbb{R}$ .

Consider the Lagrangian function  $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \rightarrow \mathbb{R}$  defined by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \quad \text{here} \quad h(x) = (h_1(x), \dots, h_{n_h}(x))^\top$$

## Lagrange-Newton

Now consider an arbitrary  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  and arbitrary constraint functions  $h_j : \mathbb{R}^n \rightarrow \mathbb{R}$ .

Consider the Lagrangian function  $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \rightarrow \mathbb{R}$  defined by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \quad \text{here} \quad h(x) = (h_1(x), \dots, h_{n_h}(x))^\top$$

We search for the stationary point of  $\mathcal{L}$ , that is  $(x^*, \lambda^*)$  satisfying

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla f(x^*) + \sum_{j=1}^{n_h} \lambda_j^* \nabla h_j(x^*) = 0$$

$$\nabla_\lambda \mathcal{L}(x^*, \lambda^*) = h(x^*) = 0$$

These are  $n + n_h$  equations in unknowns  $(x^*, \lambda^*)$ .

## Lagrange-Newton

Now consider an arbitrary  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  and arbitrary constraint functions  $h_j : \mathbb{R}^n \rightarrow \mathbb{R}$ .

Consider the Lagrangian function  $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \rightarrow \mathbb{R}$  defined by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \quad \text{here} \quad h(x) = (h_1(x), \dots, h_{n_h}(x))^\top$$

We search for the stationary point of  $\mathcal{L}$ , that is  $(x^*, \lambda^*)$  satisfying

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla f(x^*) + \sum_{j=1}^{n_h} \lambda_j^* \nabla h_j(x^*) = 0$$

$$\nabla_\lambda \mathcal{L}(x^*, \lambda^*) = h(x^*) = 0$$

These are  $n + n_h$  equations in unknowns  $(x^*, \lambda^*)$ .

From Lagrange theorem: If  $x^*$  is regular and solves the COP, then there exists  $\lambda^*$  such that  $(x^*, \lambda^*)$  solves the system of equations.

## Lagrange-Newton

Now consider an arbitrary  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  and arbitrary constraint functions  $h_j : \mathbb{R}^n \rightarrow \mathbb{R}$ .

Consider the Lagrangian function  $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \rightarrow \mathbb{R}$  defined by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \quad \text{here} \quad h(x) = (h_1(x), \dots, h_{n_h}(x))^\top$$

We search for the stationary point of  $\mathcal{L}$ , that is  $(x^*, \lambda^*)$  satisfying

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla f(x^*) + \sum_{j=1}^{n_h} \lambda_j^* \nabla h_j(x^*) = 0$$

$$\nabla_\lambda \mathcal{L}(x^*, \lambda^*) = h(x^*) = 0$$

These are  $n + n_h$  equations in unknowns  $(x^*, \lambda^*)$ .

From Lagrange theorem: If  $x^*$  is regular and solves the COP, then there exists  $\lambda^*$  such that  $(x^*, \lambda^*)$  solves the system of equations.

We use Newton's method to solve the system of equations.

## Lagrange-Newton

Start with some  $(x_0, \lambda_0)$  and compute  $(x_1, \lambda_1), \dots, (x_k, \lambda_k), \dots$

## Lagrange-Newton

Start with some  $(x_0, \lambda_0)$  and compute  $(x_1, \lambda_1), \dots, (x_k, \lambda_k), \dots$

In every step we compute  $(x_{k+1}, \lambda_{k+1})$  from  $(x_k, \lambda_k)$  using Newton's step.



## Lagrange-Newton

Start with some  $(x_0, \lambda_0)$  and compute  $(x_1, \lambda_1), \dots, (x_k, \lambda_k), \dots$

In every step we compute  $(x_{k+1}, \lambda_{k+1})$  from  $(x_k, \lambda_k)$  using Newton's step.

Consider the gradient of the Lagrangian:

$$\begin{aligned}\nabla \mathcal{L}(x_k, \lambda_k) &= (\nabla_x \mathcal{L}(x_k, \lambda_k), \nabla_\lambda \mathcal{L}(x_k, \lambda_k))^T \\ &= (\nabla f(x_k) + \sum_{j=1}^{n_h} \lambda_{kj} \nabla h_j(x_k), \quad h(x_k))^T \in \mathbb{R}^{n+n_h}\end{aligned}$$

## Lagrange-Newton

Start with some  $(x_0, \lambda_0)$  and compute  $(x_1, \lambda_1), \dots, (x_k, \lambda_k), \dots$

In every step we compute  $(x_{k+1}, \lambda_{k+1})$  from  $(x_k, \lambda_k)$  using Newton's step.

Consider the gradient of the Lagrangian:

$$\begin{aligned}\nabla \mathcal{L}(x_k, \lambda_k) &= (\nabla_x \mathcal{L}(x_k, \lambda_k), \nabla_\lambda \mathcal{L}(x_k, \lambda_k))^T \\ &= (\nabla f(x_k) + \sum_{j=1}^{n_h} \lambda_{kj} \nabla h_j(x_k), \quad h(x_k))^T \in \mathbb{R}^{n+n_h}\end{aligned}$$

and the Hessian matrix of the (complete) Lagrangian

$$\nabla^2 \mathcal{L}(x_k, \lambda_k) \in \mathbb{R}^{n+n_h} \times \mathbb{R}^{n+n_h}$$

We compute this Hessian in the next slide.

## Lagrange-Newton

Start with some  $(x_0, \lambda_0)$  and compute  $(x_1, \lambda_1), \dots, (x_k, \lambda_k), \dots$

In every step we compute  $(x_{k+1}, \lambda_{k+1})$  from  $(x_k, \lambda_k)$  using Newton's step.

Consider the gradient of the Lagrangian:

$$\begin{aligned}\nabla \mathcal{L}(x_k, \lambda_k) &= (\nabla_x \mathcal{L}(x_k, \lambda_k), \nabla_\lambda \mathcal{L}(x_k, \lambda_k))^\top \\ &= (\nabla f(x_k) + \sum_{j=1}^{n_h} \lambda_{kj} \nabla h_j(x_k), h(x_k))^\top \in \mathbb{R}^{n+n_h}\end{aligned}$$

and the Hessian matrix of the (complete) Lagrangian

$$\nabla^2 \mathcal{L}(x_k, \lambda_k) \in \mathbb{R}^{n+n_h} \times \mathbb{R}^{n+n_h}$$

We compute this Hessian in the next slide.

The Newton's step is then computed by

$$\begin{aligned}x_{k+1} &= x_k + p_k & \lambda_{k+1} &= \lambda_k + \mu_k \\ (p_k, \mu_k) &= -(\nabla^2 \mathcal{L}(x_k, \lambda_k))^{-1} \nabla \mathcal{L}(x_k, \lambda_k)\end{aligned}$$

# Hessian of Lagrangian

Note that

$$\begin{aligned}\nabla^2 \mathcal{L}(x_k, \lambda_k) &= \begin{pmatrix} \nabla_{xx} \mathcal{L}(x_k, \lambda_k) & \nabla_{x\lambda} \mathcal{L}(x_k, \lambda_k) \\ \nabla_{\lambda x} \mathcal{L}(x_k, \lambda_k) & \nabla_{\lambda\lambda} \mathcal{L}(x_k, \lambda_k) \end{pmatrix} \\ &= \begin{pmatrix} H(x_k, \lambda_k) & \nabla h(x_k) \\ \nabla h(x_k)^\top & 0 \end{pmatrix}\end{aligned}$$

Here  $H$  is the Lagrangian-Hessian:

$$H(x_k, \lambda_k) = H_f(x_k) + \sum_{j=1}^{n_h} \lambda_{kj} H_{h_j}(x_k)$$

Here  $H_f$  is the Hessian of  $f$ , and each  $H_{h_j}$  is the Hessian of  $h_j$ .

$$\nabla h(x_k) = (\nabla h_1(x_k) \cdots \nabla h_{n_h}(x_k))$$

is the matrix of columns  $\nabla h_j(x_k)$  for  $j = 1, \dots, n_h$ .

# Lagrange-Newton for Equality Constraints

---

**Algorithm 14** Lagrange-Newton

---

- 1: Choose starting point  $x_0$
  - 2:  $k \leftarrow 0$
  - 3: **repeat**
  - 4:   Compute  $\nabla f(x_k), \nabla h(x_k), h(x_k)$
  - 5:   Compute  $\nabla \mathcal{L}(x_k, \lambda_k)$
  - 6:   Compute Hessians  $H_f(x_k), H_{h_j}(x_k)$  for  $j = 1, \dots, n_h$
  - 7:   Compute Lagrangian-Hessian  $H(x_k, \lambda_k)$
  - 8:   Compute  $\nabla^2 \mathcal{L}(x_k, \lambda_k)$
  - 9:   Compute  $(p_k, \mu_k)^\top = -(\nabla^2 \mathcal{L}(x_k, \lambda_k))^{-1} \nabla \mathcal{L}(x_k, \lambda_k)$
  - 10:    $x_{k+1} \leftarrow x_k + p_k$
  - 11:    $\lambda_{k+1} \leftarrow \lambda_k + \mu_k$
  - 12:    $k \leftarrow k + 1$
  - 13: **until** convergence
-

# Sequential Quadratic Programming for Inequality Constraints

Introducing inequality constraints brings serious problems.

The main problem is caused by the fact that active constraints behave differently from inactive ones.

# Sequential Quadratic Programming for Inequality Constraints

Introducing inequality constraints brings serious problems.

The main problem is caused by the fact that active constraints behave differently from inactive ones.

Roughly speaking, algorithms proceed by searching through possible combinations of active/inactive constraints and solve for each combination as if only equality constraints were present.

This is very closely related to the support enumeration algorithm from game theory.

# Sequential Quadratic Programming for Inequality Constraints

Introducing inequality constraints brings serious problems.

The main problem is caused by the fact that active constraints behave differently from inactive ones.

Roughly speaking, algorithms proceed by searching through possible combinations of active/inactive constraints and solve for each combination as if only equality constraints were present.

This is very closely related to the support enumeration algorithm from game theory.

We will consider this type of algorithm only for linear programming (the simplex algorithm).



# Summary of Differentiable Optimization

We have considered optimization for differentiable  $f$  and  $h_j$ 's.

We have considered both constrained and unconstrained optimization problems.

Primarily line-search methods: Local search, in every step set a direction and a step length.

The step length should satisfy the strong Wolfe conditions.

# Summary of Unconstrained Methods

Consider only  $f$  without constraints.

For setting direction we used several methods

- ▶ Gradient descent

Go downhill. Only first-order derivatives needed. Zig-zags.

- ▶ Newton's method

Always minimize the local quadratic approximation of  $f$ . Second-order derivatives needed. Better behavior than GD, computationally heavy.

- ▶ quasi-Newton (SR1, BFGS, L-BFGS)

Approximate the quadratic approximation of  $f$ . Only first-order derivatives needed. Behaves similarly to Newton's method. Much more computationally efficient.

# Summary of Constrained Optimization

Penalty methods, both exterior and interior.

Penalize minimizer approximations out of the feasible region (exterior), or close to the border (interior).

- ▶ Exterior

Penalize minimizer approximations out of the feasible region.

Quadratic penalty, both for equality and inequality constraints.

- ▶ Interior

Penalize minimizer approximations close to the border (interior).

Inverse barrier, logarithmic barrier, only for inequality constraints.

Finally, we have considered the Lagrange-Newton method for equality constraints.

# Linear Programming

# Linear Optimization Problem

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{by varying} & x \in \mathbb{R}^n \\ \text{subject to} & g_i(x) \leq 0 \quad i = 1, \dots, n_g \\ & h_j(x) = 0 \quad j = 1, \dots, n_h\end{array}$$

We assume that

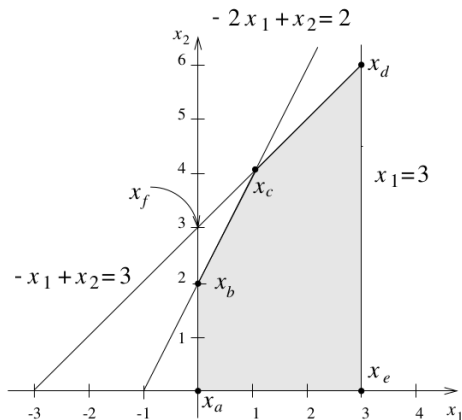
- ▶  $f$  is linear, i.e.,

$$f(x) = c^\top x \quad \text{here } c \in \mathbb{R}^n$$

- ▶ each  $g_i$  is linear,
- ▶ each  $h_j$  is linear.

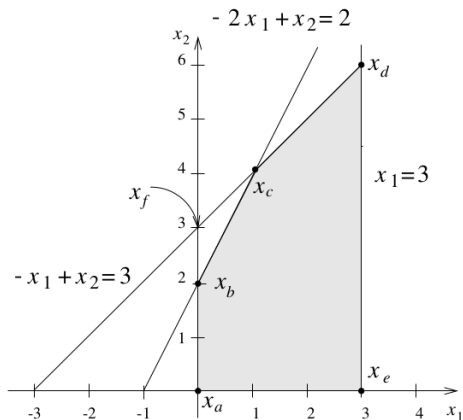
For convenience, in what follows, we also allow constraints of the form  $g_i(x) \geq 0$ .

## Example



$$\begin{array}{ll}\text{minimize} & z = -x_1 - 2x_2 \\ \text{subject to} & -2x_1 + x_2 - 2 \leq 0 \\ & -x_1 + x_2 - 3 \leq 0 \\ & x_1 - 3 \leq 0 \\ & x_1, x_2 \geq 0.\end{array}$$

## Example



The lines define the boundaries of the feasible region

$$-2x_1 + x_2 = 2$$

$$-x_1 + x_2 = 3$$

$$x_1 = 3$$

$$x_1 = 0$$

$$x_2 = 0$$

# Standard Form

The *standard form linear program*

$$\begin{array}{ll}\text{minimize} & c^\top x \\ \text{subject to} & Ax = b \\ & x \geq 0\end{array}$$

Here

- ▶  $x = (x_1, \dots, x_n)^\top \in \mathbb{R}^n$
- ▶  $c = (c_1, \dots, c_n)^\top \in \mathbb{R}^n$
- ▶  $A$  is an  $m \times n$  matrix of elements  $a_{ij}$  where  $m < n$  and  $\text{rank}(A) = m$

That is, all rows of  $A$  are linearly independent.

- ▶  $b = (b_1, \dots, b_m)^\top \geq 0$   
 $b \geq 0$  means  $b_i \geq 0$  for all  $i$ .



# Standard Form

The *standard form linear program*

$$\begin{array}{ll}\text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0\end{array}$$

Here

- ▶  $x = (x_1, \dots, x_n)^T \in \mathbb{R}^n$
- ▶  $c = (c_1, \dots, c_n)^T \in \mathbb{R}^n$
- ▶  $A$  is an  $m \times n$  matrix of elements  $a_{ij}$  where  $m < n$  and  $\text{rank}(A) = m$   
That is, all rows of  $A$  are linearly independent.
- ▶  $b = (b_1, \dots, b_m)^T \geq 0$   
 $b \geq 0$  means  $b_i \geq 0$  for all  $i$ .

Every linear optimization problem can be transformed into a standard linear program such that there is a one-to-one correspondence between solutions of the constraints preserving values of the objective.

# Transformation to Standard Form

1. For every variable  $x_i$  introduce new variables  $x'_i, x''_i$ , replace every occurrence of  $x_i$  with  $x'_i - x''_i$ , and introduce constraints  $x'_i, x''_i \geq 0$ .

Note that if a constraint is in the form  $x_i + \zeta \geq 0$  we may simply replace  $x_i$  with  $x'_i - \zeta$  and introduce  $x'_i \geq 0$ .

# Transformation to Standard Form

1. For every variable  $x_i$  introduce new variables  $x'_i, x''_i$ , replace every occurrence of  $x_i$  with  $x'_i - x''_i$ , and introduce constraints  $x'_i, x''_i \geq 0$ .

Note that if a constraint is in the form  $x_i + \zeta \geq 0$  we may simply replace  $x_i$  with  $x'_i - \zeta$  and introduce  $x'_i \geq 0$ .

2. Transform every  $g_i(x) \leq 0$  to  $g_i(x) + s_i = 0, s_i \geq 0$ . Here  $s_i$  are new variables (*slack variables*).

# Transformation to Standard Form

1. For every variable  $x_i$  introduce new variables  $x'_i, x''_i$ , replace every occurrence of  $x_i$  with  $x'_i - x''_i$ , and introduce constraints  $x'_i, x''_i \geq 0$ .

Note that if a constraint is in the form  $x_i + \zeta \geq 0$  we may simply replace  $x_i$  with  $x'_i - \zeta$  and introduce  $x'_i \geq 0$ .

2. Transform every  $g_i(x) \leq 0$  to  $g_i(x) + s_i = 0, s_i \geq 0$ . Here  $s_i$  are new variables (*slack variables*).
3. Move all constant terms to the right side of the constraints.

Now we have constraints of the form  $Ax = b, x \geq 0$ .

# Transformation to Standard Form

1. For every variable  $x_i$  introduce new variables  $x'_i, x''_i$ , replace every occurrence of  $x_i$  with  $x'_i - x''_i$ , and introduce constraints  $x'_i, x''_i \geq 0$ .

Note that if a constraint is in the form  $x_i + \zeta \geq 0$  we may simply replace  $x_i$  with  $x'_i - \zeta$  and introduce  $x'_i \geq 0$ .

2. Transform every  $g_i(x) \leq 0$  to  $g_i(x) + s_i = 0, s_i \geq 0$ . Here  $s_i$  are new variables (*slack variables*).
3. Move all constant terms to the right side of the constraints.

Now we have constraints of the form  $Ax = b, x \geq 0$ .

4. Remove linearly dependent equations from  $Ax = b$ .

This step does not alter the set of solutions.

# Transformation to Standard Form

1. For every variable  $x_i$  introduce new variables  $x'_i, x''_i$ , replace every occurrence of  $x_i$  with  $x'_i - x''_i$ , and introduce constraints  $x'_i, x''_i \geq 0$ .  
Note that if a constraint is in the form  $x_i + \zeta \geq 0$  we may simply replace  $x_i$  with  $x'_i - \zeta$  and introduce  $x'_i \geq 0$ .
2. Transform every  $g_i(x) \leq 0$  to  $g_i(x) + s_i = 0, s_i \geq 0$ . Here  $s_i$  are new variables (*slack variables*).
3. Move all constant terms to the right side of the constraints.

Now we have constraints of the form  $Ax = b, x \geq 0$ .

4. Remove linearly dependent equations from  $Ax = b$ .  
This step does not alter the set of solutions.
5. If  $m \geq n$ , the constraints have a unique or no solution. Neither of the cases is interesting for optimization. Hence,  $m < n$ .

# Transformation to Standard Form

1. For every variable  $x_i$  introduce new variables  $x'_i, x''_i$ , replace every occurrence of  $x_i$  with  $x'_i - x''_i$ , and introduce constraints  $x'_i, x''_i \geq 0$ .

Note that if a constraint is in the form  $x_i + \zeta \geq 0$  we may simply replace  $x_i$  with  $x'_i - \zeta$  and introduce  $x'_i \geq 0$ .

2. Transform every  $g_i(x) \leq 0$  to  $g_i(x) + s_i = 0, s_i \geq 0$ . Here  $s_i$  are new variables (*slack variables*).
3. Move all constant terms to the right side of the constraints.

Now we have constraints of the form  $Ax = b, x \geq 0$ .

4. Remove linearly dependent equations from  $Ax = b$ .

This step does not alter the set of solutions.

5. If  $m \geq n$ , the constraints have a unique or no solution. Neither of the cases is interesting for optimization. Hence,  $m < n$ .
6. Multiplying equations with  $b_i < 0$  by  $-1$  gives  $b \geq 0$

## Transformation Example

$$\begin{array}{ll}\text{maximize} & z = -5x_1 - 3x_2 \\ \text{subject to} & 3x_1 - 5x_2 - 5 \leq 0 \\ & -4x_1 - 9x_2 + 4 \leq 0\end{array}$$



## Transformation Example

$$\begin{array}{ll}\text{maximize} & z = -5x_1 - 3x_2 \\ \text{subject to} & 3x_1 - 5x_2 - 5 \leq 0 \\ & -4x_1 - 9x_2 + 4 \leq 0\end{array}$$

Introduce the bounded variables:

$$\begin{array}{ll}\text{maximize} & z = -5x'_1 + 5x''_1 - 3x'_2 + 3x''_2 \\ \text{subject to} & 3x'_1 - 3x''_1 - 5x'_2 + 5x''_2 - 5 \leq 0 \\ & -4x'_1 + 4x''_1 - 9x'_2 + 9x''_2 + 4 \leq 0 \\ & x'_1, x''_1, x'_2, x''_2 \geq 0\end{array}$$

## Transformation Example

$$\begin{array}{ll}\text{maximize} & z = -5x_1 - 3x_2 \\ \text{subject to} & 3x_1 - 5x_2 - 5 \leq 0 \\ & -4x_1 - 9x_2 + 4 \leq 0\end{array}$$

Introduce the bounded variables:

$$\begin{array}{ll}\text{maximize} & z = -5x'_1 + 5x''_1 - 3x'_2 + 3x''_2 \\ \text{subject to} & 3x'_1 - 3x''_1 - 5x'_2 + 5x''_2 - 5 \leq 0 \\ & -4x'_1 + 4x''_1 - 9x'_2 + 9x''_2 + 4 \leq 0 \\ & x'_1, x''_1, x'_2, x''_2 \geq 0\end{array}$$

Introduce the slack variables:

$$\begin{array}{ll}\text{maximize} & z = -5x'_1 + 5x''_1 - 3x'_2 + 3x''_2 \\ \text{subject to} & 3x'_1 - 3x''_1 - 5x'_2 + 5x''_2 + s_1 - 5 = 0 \\ & -4x'_1 + 4x''_1 - 9x'_2 + 9x''_2 + s_2 + 4 = 0 \\ & x'_1, x''_1, x'_2, x''_2, s_1, s_2 \geq 0\end{array}$$

## Transformation Example

$$\begin{array}{ll}\text{maximize} & z = -5x_1' + 5x_1'' - 3x_2' + 3x_2'' \\ \text{subject to} & 3x_1' - 3x_1'' - 5x_2' + 5x_2'' + s_1 - 5 = 0 \\ & -4x_1' + 4x_1'' - 9x_2' + 9x_2'' + s_2 + 4 = 0 \\ & x_1', x_1'', x_2', x_2'', s_1, s_2 \geq 0\end{array}$$

## Transformation Example

$$\begin{array}{ll}\text{maximize} & z = -5x'_1 + 5x''_1 - 3x'_2 + 3x''_2 \\ \text{subject to} & 3x'_1 - 3x''_1 - 5x'_2 + 5x''_2 + s_1 - 5 = 0 \\ & -4x'_1 + 4x''_1 - 9x'_2 + 9x''_2 + s_2 + 4 = 0 \\ & x'_1, x''_1, x'_2, x''_2, s_1, s_2 \geq 0\end{array}$$

Move constants to the right:

$$\begin{array}{ll}\text{maximize} & z = -5x'_1 + 5x''_1 - 3x'_2 + 3x''_2 \\ \text{subject to} & 3x'_1 - 3x''_1 - 5x'_2 + 5x''_2 + s_1 = 5 \\ & -4x'_1 + 4x''_1 - 9x'_2 + 9x''_2 + s_2 = -4 \\ & x'_1, x''_1, x'_2, x''_2, s_1, s_2 \geq 0\end{array}$$

## Transformation Example

$$\begin{array}{ll}\text{maximize} & z = -5x'_1 + 5x''_1 - 3x'_2 + 3x''_2 \\ \text{subject to} & 3x'_1 - 3x''_1 - 5x'_2 + 5x''_2 + s_1 - 5 = 0 \\ & -4x'_1 + 4x''_1 - 9x'_2 + 9x''_2 + s_2 + 4 = 0 \\ & x'_1, x''_1, x'_2, x''_2, s_1, s_2 \geq 0\end{array}$$

Move constants to the right:

$$\begin{array}{ll}\text{maximize} & z = -5x'_1 + 5x''_1 - 3x'_2 + 3x''_2 \\ \text{subject to} & 3x'_1 - 3x''_1 - 5x'_2 + 5x''_2 + s_1 = 5 \\ & -4x'_1 + 4x''_1 - 9x'_2 + 9x''_2 + s_2 = -4 \\ & x'_1, x''_1, x'_2, x''_2, s_1, s_2 \geq 0\end{array}$$

Check if all equations are linearly independent.

Multiply the last one with  $-1$ :

$$\begin{array}{ll}\text{maximize} & z = -5x'_1 + 5x''_1 - 3x'_2 + 3x''_2 \\ \text{subject to} & 3x'_1 - 3x''_1 - 5x'_2 + 5x''_2 + s_1 = 5 \\ & 4x'_1 - 4x''_1 + 9x'_2 - 9x''_2 - s_2 = 4 \\ & x'_1, x''_1, x'_2, x''_2, s_1, s_2 \geq 0\end{array}$$

## Transformation Example

$$\begin{array}{ll}\text{maximize} & z = -5x_1' + 5x_1'' - 3x_2' + 3x_2'' \\ \text{subject to} & 3x_1' - 3x_1'' - 5x_2' + 5x_2'' + s_1 = 5 \\ & 4x_1' - 4x_1'' + 9x_2' - 9x_2'' - s_2 = 4 \\ & x_1', x_1'', x_2', x_2'', s_1, s_2 \geq 0\end{array}$$

In the standard form:

$$A = \begin{pmatrix} 3 & -3 & -5 & 5 & 1 & 0 \\ 4 & -4 & 9 & -9 & 0 & -1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5, x_6)^\top$$

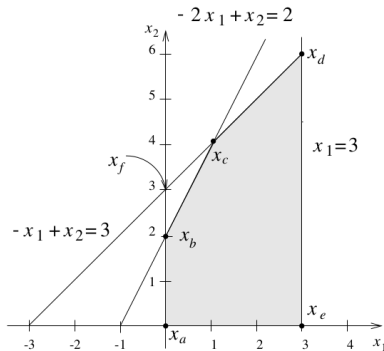
Note that we have renamed the variables.

$$b = (5, 4)^\top$$

$$Ax = b \text{ where } x \geq 0$$

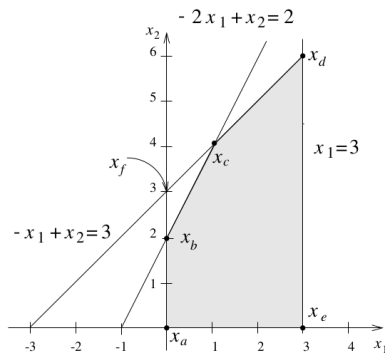
$$c = (-5, 5, -3, 3)^\top$$

## Example



$$\begin{array}{ll}\text{minimize} & z = -x_1 - 2x_2 \\ \text{subject to} & -2x_1 + x_2 - 2 \leq 0 \\ & -x_1 + x_2 - 3 \leq 0 \\ & x_1 - 3 \leq 0 \\ & x_1, x_2 \geq 0.\end{array}$$

## Example

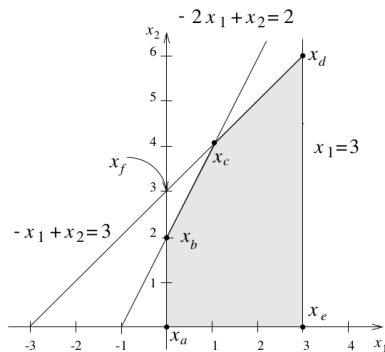


Transform to

$$\begin{array}{ll}\text{minimize} & z = -x_1 - 2x_2 \\ \text{subject to} & -2x_1 + x_2 + s_1 = 2 \\ & -x_1 + x_2 + s_2 = 3 \\ & x_1 + s_3 = 3 \\ & x_1, x_2, s_1, s_2, s_3 \geq 0\end{array}$$



## Example



The standard form:

$$A = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$b = (2, 3, 3)^\top$$

$$Ax = b$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$c = (-1, -2, 0, 0, 0)^\top$$

# Assumptions

Consider a linear programming problem in the standard form:

$$\begin{array}{ll}\text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0\end{array}$$

# Assumptions

Consider a linear programming problem in the standard form:

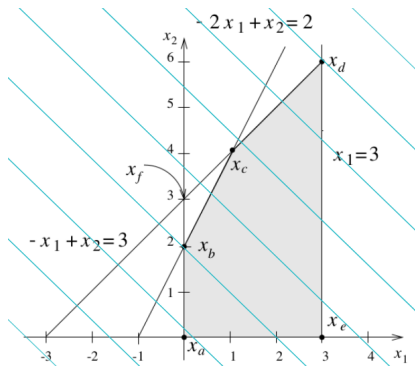
$$\begin{array}{ll}\text{minimize} & c^\top x \\ \text{subject to} & Ax = b \\ & x \geq 0\end{array}$$

In what follows, we will use the following shorthand: Given two column vectors  $x, x'$ , we write  $[x, x']$  to denote the vector resulting from stacking  $x$  on top of  $x'$ .

# Solutions

There are (typically) infinitely many solutions to the constraints.

Are there some distinguished ones? How do you find minimizers?



Here, the blue lines are contours of  $-x_1 - x_2$ .

## Basic Solutions

Assume that the matrix  $A$  has full row rank (w.l.o.g.).

## Basic Solutions

Assume that the matrix  $A$  has full row rank (w.l.o.g.).

Let  $B$  be a set of  $m$  indices of columns of  $A$  for a linearly independent set. Such a  $B$  is called a *basis*.

## Basic Solutions

Assume that the matrix  $A$  has full row rank (w.l.o.g.).

Let  $B$  be a set of  $m$  indices of columns of  $A$  for a linearly independent set. Such a  $B$  is called a *basis*.

Denote by  $N$  the set of indices of columns not in  $B$ .

## Basic Solutions

Assume that the matrix  $A$  has full row rank (w.l.o.g.).

Let  $B$  be a set of  $m$  indices of columns of  $A$  for a linearly independent set. Such a  $B$  is called a *basis*.

Denote by  $N$  the set of indices of columns not in  $B$ .

Given  $x \in \mathbb{R}^n$ , we let

- ▶  $x_B \in \mathbb{R}^m$  consist of components of  $x$  with indices in  $B$
- ▶  $x_N \in \mathbb{R}^{n-m}$  consist of components of  $x$  with indices in  $N$



## Basic Solutions

Assume that the matrix  $A$  has full row rank (w.l.o.g.).

Let  $B$  be a set of  $m$  indices of columns of  $A$  for a linearly independent set. Such a  $B$  is called a *basis*.

Denote by  $N$  the set of indices of columns not in  $B$ .

Given  $x \in \mathbb{R}^n$ , we let

- ▶  $x_B \in \mathbb{R}^m$  consist of components of  $x$  with indices in  $B$
- ▶  $x_N \in \mathbb{R}^{n-m}$  consist of components of  $x$  with indices in  $N$

Abusing notation, we denote by  $B$  and  $N$  the submatrices of  $A$  consisting of columns with indices in  $B$  and  $N$ , resp.

## Basic Solutions

Assume that the matrix  $A$  has full row rank (w.l.o.g.).

Let  $B$  be a set of  $m$  indices of columns of  $A$  for a linearly independent set. Such a  $B$  is called a *basis*.

Denote by  $N$  the set of indices of columns not in  $B$ .

Given  $x \in \mathbb{R}^n$ , we let

- ▶  $x_B \in \mathbb{R}^m$  consist of components of  $x$  with indices in  $B$
- ▶  $x_N \in \mathbb{R}^{n-m}$  consist of components of  $x$  with indices in  $N$

Abusing notation, we denote by  $B$  and  $N$  the submatrices of  $A$  consisting of columns with indices in  $B$  and  $N$ , resp.

## Basic Solutions

Assume that the matrix  $A$  has full row rank (w.l.o.g.).

Let  $B$  be a set of  $m$  indices of columns of  $A$  for a linearly independent set. Such a  $B$  is called a *basis*.

Denote by  $N$  the set of indices of columns not in  $B$ .

Given  $x \in \mathbb{R}^n$ , we let

- ▶  $x_B \in \mathbb{R}^m$  consist of components of  $x$  with indices in  $B$
- ▶  $x_N \in \mathbb{R}^{n-m}$  consist of components of  $x$  with indices in  $N$

Abusing notation, we denote by  $B$  and  $N$  the submatrices of  $A$  consisting of columns with indices in  $B$  and  $N$ , resp.

### Definition

Consider  $x \in \mathbb{R}^n$  and a basis  $B$ , and consider the decomposition of  $x$  into  $x_B \in \mathbb{R}^m$  and  $x_N \in \mathbb{R}^{n-m}$ .

Then  $x$  is a *basic solution w.r.t. the basis  $B$*  if  $Ax = b$  and  $x_N = 0$ .

Components of  $x_B$  are *basic variables*.

A basic solution  $x$  is *feasible* if  $x \geq 0$ .

## Example (Whiteboard)

Add slack variables  $x_3, x_4$ :

$$x_1 + x_2 \leq 2$$

$$x_1 \leq 1$$

$$x_1, x_2 \geq 0$$

$$x_1 + x_2 + x_3 = 2$$

$$x_1 + x_4 = 1$$

$$x_1, x_2, x_3, x_4 \geq 0$$

$$A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

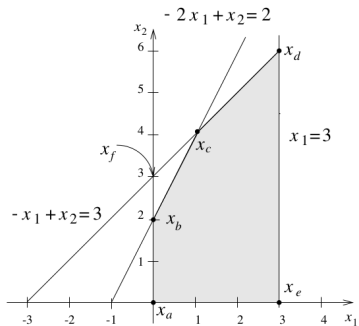
$$x = (x_1, x_2, x_3, x_4)^\top$$

$$b = (2, 1)^\top$$

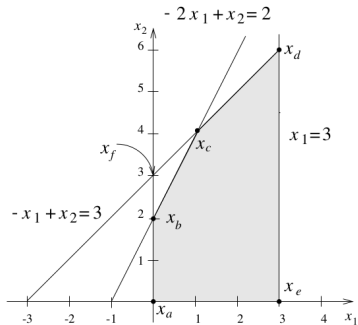
$$Ax = b \text{ where } x \geq 0$$

For now, let us ignore the objective function and play with the polyhedron defined by the inequalities above.

$$\begin{aligned}
 -2x_1 + x_2 + x_3 &= 2 \\
 -x_1 + x_2 + x_4 &= 3 \\
 x_1 + x_5 &= 3 \\
 x_1, x_2, x_3, x_4, x_5 &\geq 0
 \end{aligned}$$

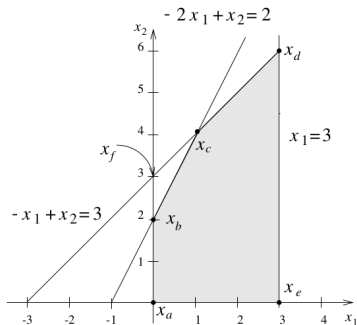


$$\begin{aligned}
 -2x_1 + x_2 + x_3 &= 2 \\
 -x_1 + x_2 + x_4 &= 3 \\
 x_1 + x_5 &= 3 \\
 x_1, x_2, x_3, x_4, x_5 &\geq 0
 \end{aligned}$$



$$A = (u_1 \ u_2 \ u_3 \ u_4 \ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{aligned}
 -2x_1 + x_2 + x_3 &= 2 \\
 -x_1 + x_2 + x_4 &= 3 \\
 x_1 + x_5 &= 3 \\
 x_1, x_2, x_3, x_4, x_5 &\geq 0
 \end{aligned}$$



$$A = (u_1 \ u_2 \ u_3 \ u_4 \ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

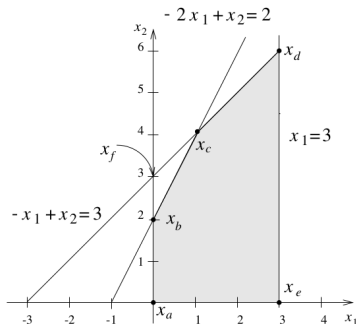
$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$-2x_1 + x_2 + x_3 = 2$$

$$-x_1 + x_2 + x_4 = 3$$

$$x_1 + x_5 = 3$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0$$



$$A = (u_1 \ u_2 \ u_3 \ u_4 \ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$b = (2, 3, 3)^\top$$

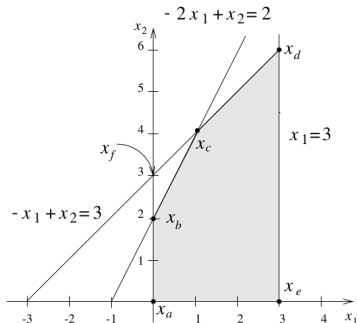


$$-2x_1 + x_2 + x_3 = 2$$

$$-x_1 + x_2 + x_4 = 3$$

$$x_1 + x_5 = 3$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0$$



$$A = (u_1 \ u_2 \ u_3 \ u_4 \ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$b = (2, 3, 3)^\top$$

$$Ax = b \text{ where } x \geq 0$$

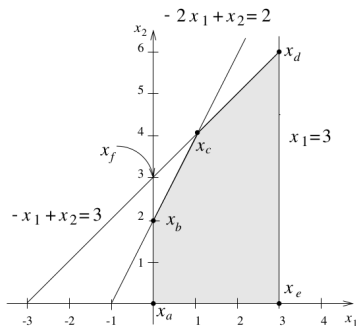
$$A = (u_1 \ u_2 \ u_3 \ u_4 \ u_5)$$

$$= \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



Consider a basis  $\{x_3, x_4, x_5\}$  with

$$B = (u_3 \ u_4 \ u_5) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

What is  $x_B$  satisfying  $Bx_B = b$ ?

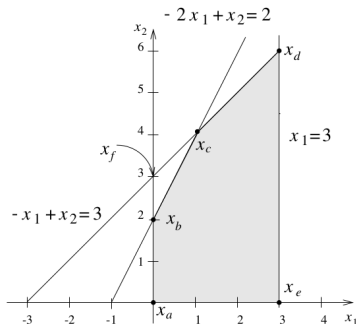
$$A = (u_1 \ u_2 \ u_3 \ u_4 \ u_5)$$

$$= \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



Consider a basis  $\{x_3, x_4, x_5\}$  with

$$B = (u_3 \ u_4 \ u_5) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

What is  $x_B$  satisfying  $Bx_B = b$ ?  $x_B = (x_3, x_4, x_5)^\top = (2, 3, 3)^\top$ .

The corresponding basic solution is

$$x = (x_1, x_2, x_3, x_4, x_5)^\top = (0, 0, 2, 3, 3)^\top = x_a \quad \text{Feasible!}$$

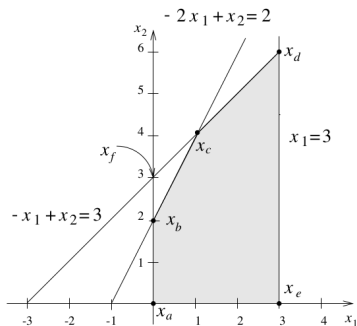
$$A = (u_1 \ u_2 \ u_3 \ u_4 \ u_5)$$

$$= \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



Consider a basis  $\{x_2, x_3, x_5\}$  with

$$B = (a_2 \ a_3 \ a_5) = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

What is  $x_B$  satisfying  $Bx_B = b$ ?

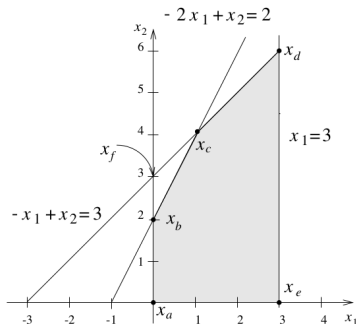
$$A = (u_1 \ u_2 \ u_3 \ u_4 \ u_5)$$

$$= \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



Consider a basis  $\{x_2, x_3, x_5\}$  with

$$B = (a_2 \ a_3 \ a_5) = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

What is  $x_B$  satisfying  $Bx_B = b$ ?  $x_B = (x_2, x_3, x_5)^\top = (3, -1, 3)^\top$ .

The corresponding basic solution is

$$x = (x_1, x_2, x_3, x_4, x_5)^\top = (0, 3, -1, 0, 3)^\top = x_f \quad \text{Not feasible!}$$

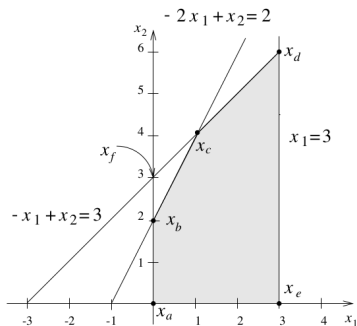
$$A = (u_1 \ u_2 \ u_3 \ u_4 \ u_5)$$

$$= \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



Consider a basis  $\{x_1, x_2, x_3\}$  with

$$B = (u_1 \ u_2 \ u_3) = \begin{pmatrix} -2 & 1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

What is  $x_B$  satisfying  $Bx_B = b$ ?

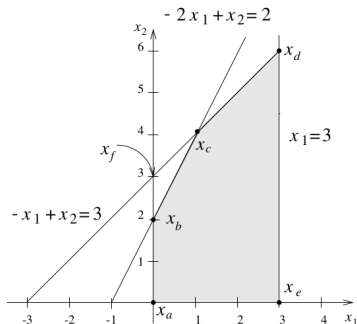
$$A = (u_1 \ u_2 \ u_3 \ u_4 \ u_5)$$

$$= \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^T$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^T$$



Consider a basis  $\{x_1, x_2, x_3\}$  with

$$B = (u_1 \ u_2 \ u_3) = \begin{pmatrix} -2 & 1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

What is  $x_B$  satisfying  $Bx_B = b$ ?  $x_B = (x_1, x_2, x_3)^T = (3, 6, 2)^T$ .

The corresponding basic solution is

$$x = (x_1, x_2, x_3, x_4, x_5)^T = (3, 6, 2, 0, 0)^T = x_d \quad \text{Feasible!}$$

# Existence of Basic Feasible Solutions

## Theorem 18 (Fundamental Theorem of LP)

*Consider a linear program in standard form.*

1. *If a feasible solution exists, then a basic feasible solution exists.*
2. *If an optimal feasible solution exists, then an optimal basic feasible solution exists.*



# Existence of Basic Feasible Solutions

## Theorem 18 (Fundamental Theorem of LP)

*Consider a linear program in standard form.*

- 1. If a feasible solution exists, then a basic feasible solution exists.*
- 2. If an optimal feasible solution exists, then an optimal basic feasible solution exists.*

Note that the theorem reduces solving a linear programming problem to searching for basic feasible solutions.

There are finitely many of them, which implies decidability.

# Existence of Basic Feasible Solutions

## Theorem 18 (Fundamental Theorem of LP)

*Consider a linear program in standard form.*

- 1. If a feasible solution exists, then a basic feasible solution exists.*
- 2. If an optimal feasible solution exists, then an optimal basic feasible solution exists.*

Note that the theorem reduces solving a linear programming problem to searching for basic feasible solutions.

There are finitely many of them, which implies decidability.

However, the enumeration of all basic feasible solutions would be impractical; the number of basic feasible solutions is potentially

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

For  $n = 100$  and  $m = 10$ , we get 535,983,370,403,809,682,970.

## Extreme Points

Note that the set  $\Theta$  of points  $x$  satisfying  $Ax = b, x \geq 0$  is *convex polyhedron*.

By definition, a convex hull of a finite set of points.

## Extreme Points

Note that the set  $\Theta$  of points  $x$  satisfying  $Ax = b, x \geq 0$  is *convex polyhedron*.

By definition, a convex hull of a finite set of points.

A point  $x \in \Theta$  is an *extreme point* of  $\Theta$  if there are no two points  $x', x'' \in \Theta \setminus \{x\}$  such that  $x = \alpha x' + (1 - \alpha)x''$  for some  $\alpha \in (0, 1)$ .

## Extreme Points

Note that the set  $\Theta$  of points  $x$  satisfying  $Ax = b, x \geq 0$  is *convex polyhedron*.

By definition, a convex hull of a finite set of points.

A point  $x \in \Theta$  is an *extreme point* of  $\Theta$  if there are no two points  $x', x'' \in \Theta \setminus \{x\}$  such that  $x = \alpha x' + (1 - \alpha)x''$  for some  $\alpha \in (0, 1)$ .

### Theorem 19

*Let  $\Theta$  be the convex set consisting of all feasible solutions, that is, all  $x \in \mathbb{R}^n$  satisfying:*

$$Ax = b, \quad x \geq 0,$$

*where  $A \in \mathbb{R}^{m \times n}$ ,  $m < n$ ,  $\text{rank}(A) = m$ .*

*Then,  $x$  is an extreme point of  $\Theta$  if and only if  $x$  is a basic feasible solution to  $Ax = b, x \geq 0$ .*

## Extreme Points

Note that the set  $\Theta$  of points  $x$  satisfying  $Ax = b, x \geq 0$  is *convex polyhedron*.

By definition, a convex hull of a finite set of points.

A point  $x \in \Theta$  is an *extreme point* of  $\Theta$  if there are no two points  $x', x'' \in \Theta \setminus \{x\}$  such that  $x = \alpha x' + (1 - \alpha)x''$  for some  $\alpha \in (0, 1)$ .

### Theorem 19

*Let  $\Theta$  be the convex set consisting of all feasible solutions, that is, all  $x \in \mathbb{R}^n$  satisfying:*

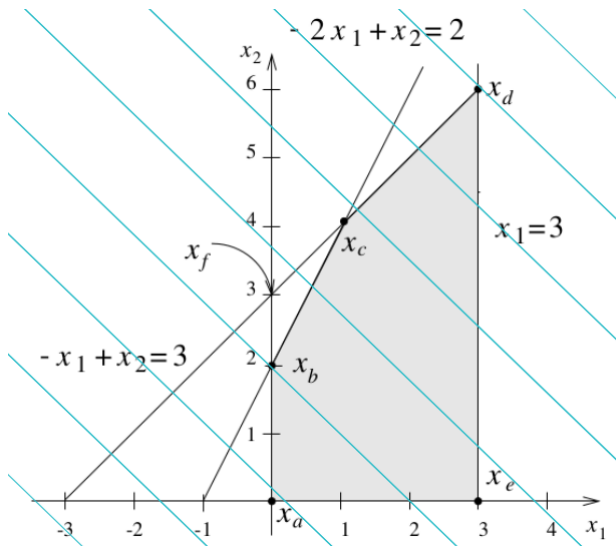
$$Ax = b, \quad x \geq 0,$$

*where  $A \in \mathbb{R}^{m \times n}$ ,  $m < n$ ,  $\text{rank}(A) = m$ .*

*Then,  $x$  is an extreme point of  $\Theta$  if and only if  $x$  is a basic feasible solution to  $Ax = b, x \geq 0$ .*

Thus, as a corollary, we obtain that to find an optimal solution to the linear optimization problem, we need to consider only extreme points of the feasibility region.

# Optimal Solutions



Here, the blue lines are contours of  $-x_1 - x_2$ . The minimizer is  $x_d$ .

## Degenerate Basic Solutions

A basic solution  $x = [x_B, x_N] \in \mathbb{R}^n$  is *degenerate* if at least one component of  $x_B$  is 0.



## Degenerate Basic Solutions

A basic solution  $x = [x_B, x_N] \in \mathbb{R}^n$  is *degenerate* if at least one component of  $x_B$  is 0.

Two different bases can correspond to the same point. To see this, consider the constraints defined by

$$Ax = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 6 \\ 13 \\ 12 \end{pmatrix} = b.$$

## Degenerate Basic Solutions

A basic solution  $x = [x_B, x_N] \in \mathbb{R}^n$  is *degenerate* if at least one component of  $x_B$  is 0.

Two different bases can correspond to the same point. To see this, consider the constraints defined by

$$Ax = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 6 \\ 13 \\ 12 \end{pmatrix} = b.$$

There are two bases

$\{x_1, x_2, x_3\}$  giving

$$B = \begin{pmatrix} 2 & 1 & 0 \\ 3 & 0 & 1 \\ 4 & 0 & 0 \end{pmatrix}$$

$\{x_1, x_3, x_4\}$  giving

$$B' = \begin{pmatrix} 2 & 0 & 0 \\ 3 & 1 & 0 \\ 4 & 0 & 1 \end{pmatrix}$$

Each gives the same *degenerate* basic solution  $x = (3, 0, 4, 0)^\top$ .

# Simplex Algorithm

## Intuition

The algorithm proceeds as follows:

- ▶ Start in a vertex of the polyhedron defined by the constraints.

## Intuition

The algorithm proceeds as follows:

- ▶ Start in a vertex of the polyhedron defined by the constraints.
- ▶ Move to each of the neighboring vertices and check whether it is better from the point of view of the objective.

# Intuition

The algorithm proceeds as follows:

- ▶ Start in a vertex of the polyhedron defined by the constraints.
- ▶ Move to each of the neighboring vertices and check whether it is better from the point of view of the objective.
- ▶ If yes, move to such a neighbor (there may be more than one better than the current one; choose one of them).

# Intuition

The algorithm proceeds as follows:

- ▶ Start in a vertex of the polyhedron defined by the constraints.
- ▶ Move to each of the neighboring vertices and check whether it is better from the point of view of the objective.
- ▶ If yes, move to such a neighbor (there may be more than one better than the current one; choose one of them).
- ▶ If there is no better neighbor, the algorithm stops.

# Intuition

The algorithm proceeds as follows:

- ▶ Start in a vertex of the polyhedron defined by the constraints.
- ▶ Move to each of the neighboring vertices and check whether it is better from the point of view of the objective.
- ▶ If yes, move to such a neighbor (there may be more than one better than the current one; choose one of them).
- ▶ If there is no better neighbor, the algorithm stops.
- ▶ (It may happen that the polyhedron is unbounded if the algorithm finds out that the objective may be infinitely improved.)



# Intuition

The algorithm proceeds as follows:

- ▶ Start in a vertex of the polyhedron defined by the constraints.
- ▶ Move to each of the neighboring vertices and check whether it is better from the point of view of the objective.
- ▶ If yes, move to such a neighbor (there may be more than one better than the current one; choose one of them).
- ▶ If there is no better neighbor, the algorithm stops.
- ▶ (It may happen that the polyhedron is unbounded if the algorithm finds out that the objective may be infinitely improved.)

Now, how do you move from one vertex to another one algebraically?

First, we consider LP problems where each basic solution is non-degenerate.

Later we drop this assumption.

## Changing Basis (Non-Degenerate Case)

Consider a basis  $B$  and write  $A = (B \ N) = (u_1 \dots u_m \ u_{m+1} \dots u_n)$  where  $B = (u_1 \dots u_m)$  and  $N = (u_{m+1} \dots u_n)$ .

Note that each  $u_i$  is a column vector of dimension  $m$ .

## Changing Basis (Non-Degenerate Case)

Consider a basis  $B$  and write  $A = (B \ N) = (u_1 \dots u_m \ u_{m+1} \dots u_n)$  where  $B = (u_1 \dots u_m)$  and  $N = (u_{m+1} \dots u_n)$ .

Note that each  $u_i$  is a column vector of dimension  $m$ .

Consider a basic feasible solution  $x = [x_B \ x_N]$  where  $x_N = 0$ . Then

$$x_1 u_1 + \dots + x_m u_m = b$$

For a non-degenerate case, we have  $x_j > 0$  for all  $j = 1, \dots, m$ .

## Changing Basis (Non-Degenerate Case)

Consider a basis  $B$  and write  $A = (B \ N) = (u_1 \dots u_m \ u_{m+1} \dots u_n)$  where  $B = (u_1 \dots u_m)$  and  $N = (u_{m+1} \dots u_n)$ .

Note that each  $u_i$  is a column vector of dimension  $m$ .

Consider a basic feasible solution  $x = [x_B \ x_N]$  where  $x_N = 0$ . Then

$$x_1 u_1 + \dots + x_m u_m = b$$

For a non-degenerate case, we have  $x_j > 0$  for all  $j = 1, \dots, m$ .

Now as  $B$  is a basis, we have that for each  $i \in \{m+1, \dots, n\}$  there are coefficients  $y_1, \dots, y_m$  such that  $y_1 u_1 + \dots + y_m u_m = u_i$ .

## Changing Basis (Non-Degenerate Case)

Consider a basis  $B$  and write  $A = (B \ N) = (u_1 \dots u_m \ u_{m+1} \dots u_n)$  where  $B = (u_1 \dots u_m)$  and  $N = (u_{m+1} \dots u_n)$ .

Note that each  $u_i$  is a column vector of dimension  $m$ .

Consider a basic feasible solution  $x = [x_B \ x_N]$  where  $x_N = 0$ . Then

$$x_1 u_1 + \dots + x_m u_m = b$$

For a non-degenerate case, we have  $x_j > 0$  for all  $j = 1, \dots, m$ .

Now as  $B$  is a basis, we have that for each  $i \in \{m+1, \dots, n\}$  there are coefficients  $y_1, \dots, y_m$  such that  $y_1 u_1 + \dots + y_m u_m = u_i$ . Then

$$\begin{aligned} b &= x_1 u_1 + \dots + x_m u_m \\ &= x_1 u_1 + \dots + x_m u_m - \alpha u_i + \alpha u_i \\ &= x_1 u_1 + \dots + x_m u_m - \alpha (y_1 u_1 + \dots + y_m u_m) + \alpha u_i \\ &= (x_1 - \alpha y_1) u_1 + \dots + (x_m - \alpha y_m) u_m + \alpha u_i \end{aligned}$$

## Changing Basis (Non-Degenerate Case)

Consider a basis  $B$  and write  $A = (B \ N) = (u_1 \dots u_m \ u_{m+1} \dots u_n)$  where  $B = (u_1 \dots u_m)$  and  $N = (u_{m+1} \dots u_n)$ .

Note that each  $u_i$  is a column vector of dimension  $m$ .

Consider a basic feasible solution  $x = [x_B \ x_N]$  where  $x_N = 0$ . Then

$$x_1 u_1 + \dots + x_m u_m = b$$

For a non-degenerate case, we have  $x_j > 0$  for all  $j = 1, \dots, m$ .

Now as  $B$  is a basis, we have that for each  $i \in \{m+1, \dots, n\}$  there are coefficients  $y_1, \dots, y_m$  such that  $y_1 u_1 + \dots + y_m u_m = u_i$ . Then

$$\begin{aligned} b &= x_1 u_1 + \dots + x_m u_m \\ &= x_1 u_1 + \dots + x_m u_m - \alpha u_i + \alpha u_i \\ &= x_1 u_1 + \dots + x_m u_m - \alpha (y_1 u_1 + \dots + y_m u_m) + \alpha u_i \\ &= (x_1 - \alpha y_1) u_1 + \dots + (x_m - \alpha y_m) u_m + \alpha u_i \end{aligned}$$

Now consider maximum  $\alpha > 0$  such that  $x_j - \alpha y_j \geq 0$  for all  $j$ .

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all  $y_j \leq 0$ , the problem is unbounded because one component grows indefinitely and others do not decrease with  $\alpha \rightarrow \infty$ .



$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all  $y_j \leq 0$ , the problem is unbounded because one component grows indefinitely and others do not decrease with  $\alpha \rightarrow \infty$ .

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\} > 0$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all  $y_j \leq 0$ , the problem is unbounded because one component grows indefinitely and others do not decrease with  $\alpha \rightarrow \infty$ .

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\} > 0$$

There would be a *unique*  $j \in \{1, \dots, m\}$  such that  $x_j - \alpha y_j = 0$ .

The uniqueness follows from non-degeneracy because otherwise, we would move to a basis giving a degenerate solution.

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all  $y_j \leq 0$ , the problem is unbounded because one component grows indefinitely and others do not decrease with  $\alpha \rightarrow \infty$ .

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\} > 0$$

There would be a *unique*  $j \in \{1, \dots, m\}$  such that  $x_j - \alpha y_j = 0$ .

The uniqueness follows from non-degeneracy because otherwise, we would move to a basis giving a degenerate solution.

Note that such  $j$  can be computed using:

$$j = \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all  $y_j \leq 0$ , the problem is unbounded because one component grows indefinitely and others do not decrease with  $\alpha \rightarrow \infty$ .

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\} > 0$$

There would be a *unique*  $j \in \{1, \dots, m\}$  such that  $x_j - \alpha y_j = 0$ .

The uniqueness follows from non-degeneracy because otherwise, we would move to a basis giving a degenerate solution.

Note that such  $j$  can be computed using:

$$j = \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$$

Obtain a basis  $B_{j \rightarrow i} = B \setminus \{j\} \cup \{i\}$  and a basic feasible solution

$$x_{j \rightarrow i} = (x'_1, \dots, x'_{j-1}, 0, x'_{j+1}, \dots, x'_m, 0, \dots, 0, \alpha, 0, \dots, 0)^\top$$

Here  $x'_k = x_k - \alpha y_k$  for each  $k \in \{1, \dots, j-1, j+1, \dots, m\}$ .

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all  $y_j \leq 0$ , the problem is unbounded because one component grows indefinitely and others do not decrease with  $\alpha \rightarrow \infty$ .

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\} > 0$$

There would be a *unique*  $j \in \{1, \dots, m\}$  such that  $x_j - \alpha y_j = 0$ .

The uniqueness follows from non-degeneracy because otherwise, we would move to a basis giving a degenerate solution.

Note that such  $j$  can be computed using:

$$j = \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$$

Obtain a basis  $B_{j \rightarrow i} = B \setminus \{j\} \cup \{i\}$  and a basic feasible solution

$$x_{j \rightarrow i} = (x'_1, \dots, x'_{j-1}, 0, x'_{j+1}, \dots, x'_m, 0, \dots, 0, \alpha, 0, \dots, 0)^\top$$

Here  $x'_k = x_k - \alpha y_k$  for each  $k \in \{1, \dots, j-1, j+1, \dots, m\}$ .

We say that we *pivot about*  $(j, i)$ .

---

**Algorithm 15** Simplex - **Non-degenerate**

---

```
1: Choose a starting basis  $B = (u_1 \dots u_m)$  (here  $A = (B \ N)$ )
2: repeat
3:   Compute the basic solution  $x$  for the basis  $B$ 
4:   for  $i \in \{m + 1, \dots, n\}$  do
5:     Solve  $B(y_1, \dots, y_m)^\top = u_i$ 
6:     if  $y_k \leq 0$  for all  $k \in \{1, \dots, m\}$  then
7:       Stop, unbounded problem.
8:     end if
9:     Select  $j = \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$ 
10:    Compute  $x_{j \rightarrow i}$ 
11:  end for
12:  if  $c^\top(x_{j \rightarrow i} - x) \geq 0$  for all  $i \in \{m + 1, \dots, n\}$  then
13:    Stop, we have an optimal solution.
14:  end if
15:  Select  $i \in \{m + 1, \dots, n\}$  such that  $c^\top(x_{j \rightarrow i} - x) < 0$ 
16:   $B \leftarrow B_{j \rightarrow i}$ 
17: until convergence
```

---

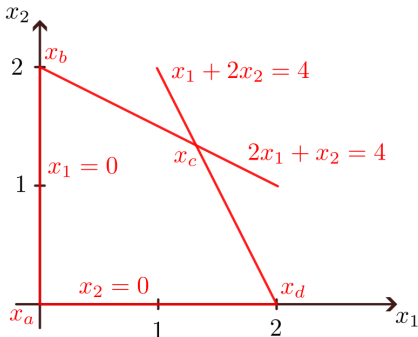
$$A = (u_1 \ u_2 \ u_3 \ u_4)$$

$$= \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4)^\top$$

$$b = (4, 4)^\top$$

$$c = (-1, -1, 0, 0)^\top$$



minimize  $c^\top x$  subject to  $Ax = b$  where  $x \geq 0$

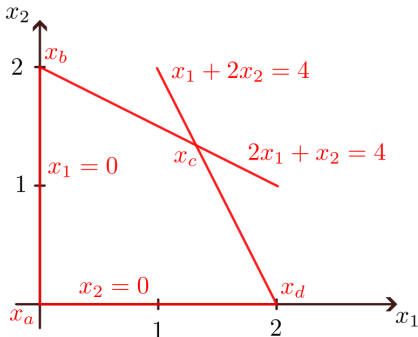
$$A = (u_1 \ u_2 \ u_3 \ u_4)$$

$$= \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4)^\top$$

$$b = (4, 4)^\top$$

$$c = (-1, -1, 0, 0)^\top$$



minimize  $c^\top x$  subject to  $Ax = b$  where  $x \geq 0$

Consider a basis

$$B = (a_3 \ a_4) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The basic solution is  $x = (x_1, x_2, x_3, x_4)^\top = (0, 0, 4, 4)^\top$



## Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis  $\{x_3, x_4\}$  giving  $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$ .

## Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis  $\{x_3, x_4\}$  giving  $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$ .

Consider  $x_1$  as a candidate to the basis, i.e., consider the first column  $u_1$  of  $A$  expressed in the basis  $B$ :

$$u_1 = (1, 2)^\top = B (1, 2)^\top \text{ thus } y = (y_3, y_4) = (1, 2)$$

## Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis  $\{x_3, x_4\}$  giving  $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$ .

Consider  $x_1$  as a candidate to the basis, i.e., consider the first column  $u_1$  of  $A$  expressed in the basis  $B$ :

$$u_1 = (1, 2)^\top = B (1, 2)^\top \text{ thus } y = (y_3, y_4) = (1, 2)$$

Now  $x_4/y_4 = 4/2 < 4/1 = x_3/y_3$ , pivot about  $(4, 1)$  and  $\alpha = x_4/y_4 = 2$ .

## Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis  $\{x_3, x_4\}$  giving  $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$ .

Consider  $x_1$  as a candidate to the basis, i.e., consider the first column  $u_1$  of  $A$  expressed in the basis  $B$ :

$$u_1 = (1, 2)^\top = B (1, 2)^\top \text{ thus } y = (y_3, y_4) = (1, 2)$$

Now  $x_4/y_4 = 4/2 < 4/1 = x_3/y_3$ , pivot about  $(4, 1)$  and  $\alpha = x_4/y_4 = 2$ .

$$x_{4 \rightarrow 1} = (\alpha, 0, (x_3 - \alpha y_3), (x_4 - \alpha y_4)) = (2, 0, 2, 0)$$

## Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis  $\{x_3, x_4\}$  giving  $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$ .

Consider  $x_1$  as a candidate to the basis, i.e., consider the first column  $u_1$  of  $A$  expressed in the basis  $B$ :

$$u_1 = (1, 2)^\top = B (1, 2)^\top \text{ thus } y = (y_3, y_4) = (1, 2)$$

Now  $x_4/y_4 = 4/2 < 4/1 = x_3/y_3$ , pivot about  $(4, 1)$  and  $\alpha = x_4/y_4 = 2$ .

$$x_{4 \rightarrow 1} = (\alpha, 0, (x_3 - \alpha y_3), (x_4 - \alpha y_4)) = (2, 0, 2, 0)$$

As a result we get the basis  $\{x_1, x_3\}$  and the basic solution  $(2, 0, 2, 0)$ .

## Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis  $\{x_3, x_4\}$  giving  $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$ .

Consider  $x_1$  as a candidate to the basis, i.e., consider the first column  $u_1$  of  $A$  expressed in the basis  $B$ :

$$u_1 = (1, 2)^\top = B (1, 2)^\top \text{ thus } y = (y_3, y_4) = (1, 2)$$

Now  $x_4/y_4 = 4/2 < 4/1 = x_3/y_3$ , pivot about  $(4, 1)$  and  $\alpha = x_4/y_4 = 2$ .

$$x_{4 \rightarrow 1} = (\alpha, 0, (x_3 - \alpha y_3), (x_4 - \alpha y_4)) = (2, 0, 2, 0)$$

As a result we get the basis  $\{x_1, x_3\}$  and the basic solution  $(2, 0, 2, 0)$ .

Similarly, we may also put  $x_2$  into the basis instead of  $x_3$  and obtain the basis  $\{x_2, x_4\}$  and the basic solution  $(0, 2, 0, 2)$ .

## Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis  $\{x_3, x_4\}$  giving  $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$ .

Consider  $x_1$  as a candidate to the basis, i.e., consider the first column  $u_1$  of  $A$  expressed in the basis  $B$ :

$$u_1 = (1, 2)^\top = B (1, 2)^\top \text{ thus } y = (y_3, y_4) = (1, 2)$$

Now  $x_4/y_4 = 4/2 < 4/1 = x_3/y_3$ , pivot about  $(4, 1)$  and  $\alpha = x_4/y_4 = 2$ .

$$x_{4 \rightarrow 1} = (\alpha, 0, (x_3 - \alpha y_3), (x_4 - \alpha y_4)) = (2, 0, 2, 0)$$

As a result we get the basis  $\{x_1, x_3\}$  and the basic solution  $(2, 0, 2, 0)$ .

Similarly, we may also put  $x_2$  into the basis instead of  $x_3$  and obtain the basis  $\{x_2, x_4\}$  and the basic solution  $(0, 2, 0, 2)$ .

We have  $c^\top (x_{4 \rightarrow 1} - x) = -2 < 0$

So let us move to the basis  $\{x_1, x_3\}$ .

## Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Consider the basis  $\{x_1, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (2, 0, 2, 0)$ .



## Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Consider the basis  $\{x_1, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (2, 0, 2, 0)$ .

Consider  $x_2$  as a candidate for the basis, i.e., consider the second column  $u_2$  of  $A$  expressed in the basis  $B$ :

$$u_2 = (2, 1)^\top = B \ (1/2, 3/2)^\top \text{ thus } y = (y_1, y_3) = (1/2, 3/2)$$

## Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Consider the basis  $\{x_1, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (2, 0, 2, 0)$ .

Consider  $x_2$  as a candidate for the basis, i.e., consider the second column  $u_2$  of  $A$  expressed in the basis  $B$ :

$$u_2 = (2, 1)^\top = B \begin{pmatrix} 1/2 \\ 3/2 \end{pmatrix}^\top \text{ thus } y = (y_1, y_3) = (1/2, 3/2)$$

Now  $\alpha = x_3/y_3 = 4/3 < 2/(1/2) = 4 = x_1/y_1$ , pivot about  $(3, 2)$

## Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Consider the basis  $\{x_1, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (2, 0, 2, 0)$ .

Consider  $x_2$  as a candidate for the basis, i.e., consider the second column  $u_2$  of  $A$  expressed in the basis  $B$ :

$$u_2 = (2, 1)^\top = B \begin{pmatrix} 1/2 \\ 3/2 \end{pmatrix}^\top \text{ thus } y = (y_1, y_3) = (1/2, 3/2)$$

Now  $\alpha = x_3/y_3 = 4/3 < 2/(1/2) = 4 = x_1/y_1$ , pivot about  $(3, 2)$

$$x_{3 \rightarrow 2} = ((x_1 - \alpha y_1), \alpha, (x_3 - \alpha y_3), 0) = (4/3, 4/3, 0, 0)$$

## Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Consider the basis  $\{x_1, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (2, 0, 2, 0)$ .

Consider  $x_2$  as a candidate for the basis, i.e., consider the second column  $u_2$  of  $A$  expressed in the basis  $B$ :

$$u_2 = (2, 1)^\top = B \begin{pmatrix} 1/2 \\ 3/2 \end{pmatrix}^\top \text{ thus } y = (y_1, y_3) = (1/2, 3/2)$$

Now  $\alpha = x_3/y_3 = 4/3 < 2/(1/2) = 4 = x_1/y_1$ , pivot about  $(3, 2)$

$$x_{3 \rightarrow 2} = ((x_1 - \alpha y_1), \alpha, (x_3 - \alpha y_3), 0) = (4/3, 4/3, 0, 0)$$

$$c^\top (x_{3 \rightarrow 2} - x) = c(-2/3, 4/3)^\top = -2/3 < 0$$

We have reached a minimizer. All changes would lead to a higher objective value.

We may exchange  $x_1$  with  $x_4$ , but this would give us the initial basis with a higher objective value.

# Non-Degenerate Case Convergence

## Theorem 20

*Suppose the simplex method is applied to a linear program, and every basic variable is strictly positive at every iteration. Then, in a finite number of iterations, the method either terminates at an optimal basic feasible solution or determines that the problem is unbounded.*

# Non-Degenerate Case Convergence

## Theorem 20

*Suppose the simplex method is applied to a linear program, and every basic variable is strictly positive at every iteration. Then, in a finite number of iterations, the method either terminates at an optimal basic feasible solution or determines that the problem is unbounded.*

However, what happens if we meet a degenerate solution?

# Non-Degenerate Case Convergence

## Theorem 20

*Suppose the simplex method is applied to a linear program, and every basic variable is strictly positive at every iteration. Then, in a finite number of iterations, the method either terminates at an optimal basic feasible solution or determines that the problem is unbounded.*

However, what happens if we meet a degenerate solution?

So, let us drop the non-degeneracy assumption.

## Changing Basis (Degenerate Case)

Consider a basis  $B$  and write  $A = (B \ N) = (u_1 \dots u_m \ u_{m+1} \dots u_n)$  where  $B = (u_1 \dots u_m)$  and  $N = (u_{m+1} \dots u_n)$ .

Note that each  $u_i$  is a column vector of dimension  $m$ .



## Changing Basis (Degenerate Case)

Consider a basis  $B$  and write  $A = (B \ N) = (u_1 \dots u_m \ u_{m+1} \dots u_n)$  where  $B = (u_1 \dots u_m)$  and  $N = (u_{m+1} \dots u_n)$ .

Note that each  $u_i$  is a column vector of dimension  $m$ .

Consider a basic feasible solution  $x = [x_B \ x_N]$  where  $x_N = 0$ . Then

$$x_1 u_1 + \dots + x_m u_m = b$$

For a degenerate case, we have  $x_j \geq 0$  for all  $j \in \{1, \dots, m\}$ , and *may have  $x_j = 0$  for some  $j \in \{1, \dots, m\}$ .*

## Changing Basis (Degenerate Case)

Consider a basis  $B$  and write  $A = (B \ N) = (u_1 \dots u_m \ u_{m+1} \dots u_n)$  where  $B = (u_1 \dots u_m)$  and  $N = (u_{m+1} \dots u_n)$ .

Note that each  $u_i$  is a column vector of dimension  $m$ .

Consider a basic feasible solution  $x = [x_B \ x_N]$  where  $x_N = 0$ . Then

$$x_1 u_1 + \dots + x_m u_m = b$$

For a degenerate case, we have  $x_j \geq 0$  for all  $j \in \{1, \dots, m\}$ , and *may have  $x_j = 0$  for some  $j \in \{1, \dots, m\}$ .*

Now as  $B$  is a basis, we have that for each  $i \in \{m+1, \dots, n\}$  there are coefficients  $y_1, \dots, y_m$  such that  $y_1 u_1 + \dots + y_m u_m = u_i$ .

## Changing Basis (Degenerate Case)

Consider a basis  $B$  and write  $A = (B \ N) = (u_1 \dots u_m \ u_{m+1} \dots u_n)$  where  $B = (u_1 \dots u_m)$  and  $N = (u_{m+1} \dots u_n)$ .

Note that each  $u_i$  is a column vector of dimension  $m$ .

Consider a basic feasible solution  $x = [x_B \ x_N]$  where  $x_N = 0$ . Then

$$x_1 u_1 + \dots + x_m u_m = b$$

For a degenerate case, we have  $x_j \geq 0$  for all  $j \in \{1, \dots, m\}$ , and *may have  $x_j = 0$  for some  $j \in \{1, \dots, m\}$* .

Now as  $B$  is a basis, we have that for each  $i \in \{m+1, \dots, n\}$  there are coefficients  $y_1, \dots, y_m$  such that  $y_1 u_1 + \dots + y_m u_m = u_i$ . Then

$$\begin{aligned} b &= x_1 u_1 + \dots + x_m u_m \\ &= x_1 u_1 + \dots + x_m u_m - \alpha u_i + \alpha u_i \\ &= x_1 u_1 + \dots + x_m u_m - \alpha (y_1 u_1 + \dots + y_m u_m) + \alpha u_i \\ &= (x_1 - \alpha y_1) u_1 + \dots + (x_m - \alpha y_m) u_m + \alpha u_i \end{aligned}$$

## Changing Basis (Degenerate Case)

Consider a basis  $B$  and write  $A = (B \ N) = (u_1 \dots u_m \ u_{m+1} \dots u_n)$  where  $B = (u_1 \dots u_m)$  and  $N = (u_{m+1} \dots u_n)$ .

Note that each  $u_i$  is a column vector of dimension  $m$ .

Consider a basic feasible solution  $x = [x_B \ x_N]$  where  $x_N = 0$ . Then

$$x_1 u_1 + \dots + x_m u_m = b$$

For a degenerate case, we have  $x_j \geq 0$  for all  $j \in \{1, \dots, m\}$ , and *may have  $x_j = 0$  for some  $j \in \{1, \dots, m\}$ .*

Now as  $B$  is a basis, we have that for each  $i \in \{m+1, \dots, n\}$  there are coefficients  $y_1, \dots, y_m$  such that  $y_1 u_1 + \dots + y_m u_m = u_i$ . Then

$$\begin{aligned} b &= x_1 u_1 + \dots + x_m u_m \\ &= x_1 u_1 + \dots + x_m u_m - \alpha u_i + \alpha u_i \\ &= x_1 u_1 + \dots + x_m u_m - \alpha (y_1 u_1 + \dots + y_m u_m) + \alpha u_i \\ &= (x_1 - \alpha y_1) u_1 + \dots + (x_m - \alpha y_m) u_m + \alpha u_i \end{aligned}$$

Now consider maximum  $\alpha \geq 0$  such that  $x_j - \alpha y_j \geq 0$  for all  $j$ .

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all  $y_j \leq 0$ , the problem is unbounded because one component grows indefinitely and others do not decrease with  $\alpha \rightarrow \infty$ .

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all  $y_j \leq 0$ , the problem is unbounded because one component grows indefinitely and others do not decrease with  $\alpha \rightarrow \infty$ .

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all  $y_j \leq 0$ , the problem is unbounded because one component grows indefinitely and others do not decrease with  $\alpha \rightarrow \infty$ .

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$$

Otherwise, there *exists*  $j \in \{1, \dots, m\}$  such that  $x_j - \alpha y_j = 0$ .

$j$  DOES NOT have to be unique in a degenerate case.



$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all  $y_j \leq 0$ , the problem is unbounded because one component grows indefinitely and others do not decrease with  $\alpha \rightarrow \infty$ .

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$$

Otherwise, there *exists*  $j \in \{1, \dots, m\}$  such that  $x_j - \alpha y_j = 0$ .  
 $j$  DOES NOT have to be unique in a degenerate case.

Note that such  $j$  can be computed using:

$$j \in \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all  $y_j \leq 0$ , the problem is unbounded because one component grows indefinitely and others do not decrease with  $\alpha \rightarrow \infty$ .

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$$

Otherwise, there *exists*  $j \in \{1, \dots, m\}$  such that  $x_j - \alpha y_j = 0$ .

$j$  DOES NOT have to be unique in a degenerate case.

Note that such  $j$  can be computed using:

$$j \in \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$$

Obtain a basis  $B_{j \rightarrow i} = B \setminus \{j\} \cup \{i\}$  and a basic feasible solution

$$x_{j \rightarrow i} = (x'_1, \dots, x'_{j-1}, 0, x'_{j+1}, \dots, x'_m, 0, \dots, 0, \alpha, 0, \dots, 0)^\top$$

Here  $x'_k = x_k - \alpha y_k$  for each  $k \in \{1, \dots, j-1, j+1, \dots, m\}$ .

Note that if  $\alpha = 0$ , the solution does not change. The basis, however, changes.

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all  $y_j \leq 0$ , the problem is unbounded because one component grows indefinitely and others do not decrease with  $\alpha \rightarrow \infty$ .

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$$

Otherwise, there *exists*  $j \in \{1, \dots, m\}$  such that  $x_j - \alpha y_j = 0$ .  
 $j$  DOES NOT have to be unique in a degenerate case.

Note that such  $j$  can be computed using:

$$j \in \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$$

Obtain a basis  $B_{j \rightarrow i} = B \setminus \{j\} \cup \{i\}$  and a basic feasible solution

$$x_{j \rightarrow i} = (x'_1, \dots, x'_{j-1}, 0, x'_{j+1}, \dots, x'_m, 0, \dots, 0, \alpha, 0, \dots, 0)^\top$$

Here  $x'_k = x_k - \alpha y_k$  for each  $k \in \{1, \dots, j-1, j+1, \dots, m\}$ .

Note that if  $\alpha = 0$ , the solution does not change. The basis, however, changes.

We say that we *pivot about*  $(j, i)$ .

## Degenerate Example

$$c = (-1, 0, 0, 0)^{\top} \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

## Degenerate Example

$$c = (-1, 0, 0, 0)^T \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4)^T = (0, 1, 0, 0)^T$  with  $c^T x = 0$ .

## Degenerate Example

$$c = (-1, 0, 0, 0)^{\top} \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4)^{\top} = (0, 1, 0, 0)^{\top}$  with  $c^{\top}x = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^{\top} = B(1, -1)^{\top} \text{ thus } y = (y_2, y_3) = (1, -1)$$

## Degenerate Example

$$c = (-1, 0, 0, 0)^T \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4)^T = (0, 1, 0, 0)^T$  with  $c^T x = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^T = B(1, -1)^T \text{ thus } y = (y_2, y_3) = (1, -1)$$

Pivot about  $(2, 4)$ , that is  $x_2$  exchanges with  $x_4$  and  $\alpha = x_2/y_2 = 1$

## Degenerate Example

$$c = (-1, 0, 0, 0)^{\top} \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4)^{\top} = (0, 1, 0, 0)^{\top}$  with  $c^{\top}x = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^{\top} = B(1, -1)^{\top} \text{ thus } y = (y_2, y_3) = (1, -1)$$

Pivot about  $(2, 4)$ , that is  $x_2$  exchanges with  $x_4$  and  $\alpha = x_2/y_2 = 1$

$$x_{2 \rightarrow 4} = (0, (x_2 - \alpha y_2), (x_3 - \alpha y_3), \alpha)^{\top} = (0, 0, 1, 1)^{\top}$$



## Degenerate Example

$$c = (-1, 0, 0, 0)^T \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4)^T = (0, 1, 0, 0)^T$  with  $c^T x = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^T = B(1, -1)^T \text{ thus } y = (y_2, y_3) = (1, -1)$$

Pivot about  $(2, 4)$ , that is  $x_2$  exchanges with  $x_4$  and  $\alpha = x_2/y_2 = 1$

$$x_{2 \rightarrow 4} = (0, (x_2 - \alpha y_2), (x_3 - \alpha y_3), \alpha)^T = (0, 0, 1, 1)^T$$

Note that  $c^T x_{2 \rightarrow 4} = 0$ .

Thus **no effect on the objective value!**

## Degenerate Example

$$c = (-1, 0, 0, 0)^{\top} \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4)^{\top} = (0, 1, 0, 0)^{\top}$  with  $c^{\top}x = 0$ .

## Degenerate Example

$$c = (-1, 0, 0, 0)^{\top} \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4)^{\top} = (0, 1, 0, 0)^{\top}$  with  $c^{\top}x = 0$ .

Consider  $x_1$  as a candidate for the basis:

$$u_1 = (1, -1)^{\top} = B(-1, 2)^{\top} \text{ thus } y = (y_2, y_3) = (-1, 2)$$

## Degenerate Example

$$c = (-1, 0, 0, 0)^T \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4)^T = (0, 1, 0, 0)^T$  with  $c^T x = 0$ .

Consider  $x_1$  as a candidate for the basis:

$$u_1 = (1, -1)^T = B(-1, 2)^T \text{ thus } y = (y_2, y_3) = (-1, 2)$$

Pivot about  $(3, 1)$ , that is  $x_3$  exchanges with  $x_1$  and  $\alpha = x_3/y_3 = 0$ .

## Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4)^\top = (0, 1, 0, 0)^\top$  with  $c^\top x = 0$ .

Consider  $x_1$  as a candidate for the basis:

$$u_1 = (1, -1)^\top = B(-1, 2)^\top \text{ thus } y = (y_2, y_3) = (-1, 2)$$

Pivot about  $(3, 1)$ , that is  $x_3$  exchanges with  $x_1$  and  $\alpha = x_3/y_3 = 0$ .

$$x_{3 \rightarrow 1} = (\alpha, (x_2 - \alpha y_2), (x_3 - \alpha y_3), 0)^\top = (0, 1, 0, 0)^\top$$

## Degenerate Example

$$c = (-1, 0, 0, 0)^T \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4)^T = (0, 1, 0, 0)^T$  with  $c^T x = 0$ .

Consider  $x_1$  as a candidate for the basis:

$$u_1 = (1, -1)^T = B(-1, 2)^T \text{ thus } y = (y_2, y_3) = (-1, 2)$$

Pivot about  $(3, 1)$ , that is  $x_3$  exchanges with  $x_1$  and  $\alpha = x_3/y_3 = 0$ .

$$x_{3 \rightarrow 1} = (\alpha, (x_2 - \alpha y_2), (x_3 - \alpha y_3), 0)^T = (0, 1, 0, 0)^T$$

No change in the basic solution, and thus  $c^T x_{3 \rightarrow 1} = c^T x = 0$ .

Thus **no effect on the objective value either!**

## Degenerate Example

$$c = (-1, 0, 0, 0)^T \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4)^T = (0, 1, 0, 0)^T$  with  $c^T x = 0$ .

Consider  $x_1$  as a candidate for the basis:

$$u_1 = (1, -1)^T = B(-1, 2)^T \text{ thus } y = (y_2, y_3) = (-1, 2)$$

Pivot about  $(3, 1)$ , that is  $x_3$  exchanges with  $x_1$  and  $\alpha = x_3/y_3 = 0$ .

$$x_{3 \rightarrow 1} = (\alpha, (x_2 - \alpha y_2), (x_3 - \alpha y_3), 0)^T = (0, 1, 0, 0)^T$$

No change in the basic solution, and thus  $c^T x_{3 \rightarrow 1} = c^T x = 0$ .

Thus **no effect on the objective value either!**

*Which variable should go to the basis?!*

## Reduced Cost

Given a basis  $B$ , we denote by  $c_B$  the vector of components of  $c$  that correspond to the variables of  $B$ .



## Reduced Cost

Given a basis  $B$ , we denote by  $c_B$  the vector of components of  $c$  that correspond to the variables of  $B$ .

One can prove that for every  $i \in \{m+1, \dots, n\}$  we have

$$c^\top x_{j \rightarrow i} - c^\top x = (c_i - c_B^\top y)\alpha$$

Here  $y = (y_1, \dots, y_m)^\top$  where  $By = u_i$ .

## Reduced Cost

Given a basis  $B$ , we denote by  $c_B$  the vector of components of  $c$  that correspond to the variables of  $B$ .

One can prove that for every  $i \in \{m+1, \dots, n\}$  we have

$$c^\top x_{j \rightarrow i} - c^\top x = (c_i - c_B^\top y)\alpha$$

Here  $y = (y_1, \dots, y_m)^\top$  where  $By = u_i$ .

For non-degenerate case, we have  $\alpha > 0$  and thus

$$c^\top x_{j \rightarrow i} < c^\top x \quad \text{iff} \quad c_i - c_B^\top y < 0$$

For the degenerate case, we may have  $\alpha = 0$  and  $c_i - c_B^\top y < 0$ .

## Reduced Cost

Given a basis  $B$ , we denote by  $c_B$  the vector of components of  $c$  that correspond to the variables of  $B$ .

One can prove that for every  $i \in \{m+1, \dots, n\}$  we have

$$c^\top x_{j \rightarrow i} - c^\top x = (c_i - c_B^\top y)\alpha$$

Here  $y = (y_1, \dots, y_m)^\top$  where  $By = u_i$ .

For non-degenerate case, we have  $\alpha > 0$  and thus

$$c^\top x_{j \rightarrow i} < c^\top x \quad \text{iff} \quad c_i - c_B^\top y < 0$$

For the degenerate case, we may have  $\alpha = 0$  and  $c_i - c_B^\top y < 0$ .

Define the *reduced cost* by

$$r_i = c_i - c_B^\top y$$

Intuitively,  $c_i$  is the cost of  $x_i$  in the new basis and  $c_B^\top y$  in the old one.

## Derivation of Reduced Cost

$$\begin{aligned}c^\top x_{j \rightarrow i} &= c^\top (x'_1, \dots, x'_{j-1}, 0, x'_{j+1}, \dots, x'_m, 0, \dots, 0, \alpha, 0, \dots, 0)^\top \\&= c^\top (x'_1, \dots, x'_{j-1}, x'_j, x'_{j+1}, \dots, x'_m, 0, \dots, 0, \alpha, 0, \dots, 0)^\top \\&= c_1 x'_1 + \dots + c_m x'_m + c_i \alpha \\&= c_1 (x_1 - \alpha y_1) + \dots + c_m (x_m - \alpha y_m) + c_i \alpha \\&= (c_1 x_1 + \dots + c_m x_m) - (c_1 y_1 + \dots + c_m y_m - c_i) \alpha \\&= c^\top x - (-c_i + c_B y) \alpha\end{aligned}$$

Here we use the fact that  $x'_k = x_k - \alpha y_k$  for each  $k \in \{1, \dots, j-1, j+1, \dots, m\}$  and that  $x_j - \alpha y_j = 0$ .

Then clearly

$$c^\top x_{j \rightarrow i} - c^\top x = (c_i - c_B y) \alpha$$

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$$

## Degenerate Example

$$c = (-1, 0, 0, 0)^T \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$  with  $cx = 0$ .

## Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$  with  $cx = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^\top = B(1, -1)^\top \text{ thus } y = (y_2, y_3) = (1, -1)$$

## Degenerate Example

$$c = (-1, 0, 0, 0)^{\top} \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$  with  $cx = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^{\top} = B(1, -1)^{\top} \text{ thus } y = (y_2, y_3) = (1, -1)$$

The reduced cost is:

$$r_4 = c_4 - (c_2 y_2 + c_3 y_3) = 0 - (0 \cdot 1 + 0 \cdot (-1)) = 0$$

## Degenerate Example

$$c = (-1, 0, 0, 0)^{\top} \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$  with  $cx = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^{\top} = B(1, -1)^{\top} \text{ thus } y = (y_2, y_3) = (1, -1)$$

The reduced cost is:

$$r_4 = c_4 - (c_2 y_2 + c_3 y_3) = 0 - (0 \cdot 1 + 0 \cdot (-1)) = 0$$

Consider  $x_1$  as a candidate for the basis:

$$u_1 = (1, -1)^{\top} = B(-1, 2)^{\top} \text{ thus } y = (y_2, y_3) = (-1, 2)$$



## Degenerate Example

$$c = (-1, 0, 0, 0)^{\top} \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$  with  $cx = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^{\top} = B(1, -1)^{\top} \text{ thus } y = (y_2, y_3) = (1, -1)$$

The reduced cost is:

$$r_4 = c_4 - (c_2 y_2 + c_3 y_3) = 0 - (0 \cdot 1 + 0 \cdot (-1)) = 0$$

Consider  $x_1$  as a candidate for the basis:

$$u_1 = (1, -1)^{\top} = B(-1, 2)^{\top} \text{ thus } y = (y_2, y_3) = (-1, 2)$$

The reduced cost is

$$r_1 = c_1 - (c_2 y_2 + c_3 y_3) = -1 - (0 \cdot (-1) + 0 \cdot 2) = -1 < 0$$

## Degenerate Example

$$c = (-1, 0, 0, 0)^T \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis  $\{x_2, x_3\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$  with  $cx = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^T = B(1, -1)^T \text{ thus } y = (y_2, y_3) = (1, -1)$$

The reduced cost is:

$$r_4 = c_4 - (c_2 y_2 + c_3 y_3) = 0 - (0 \cdot 1 + 0 \cdot (-1)) = 0$$

Consider  $x_1$  as a candidate for the basis:

$$u_1 = (1, -1)^T = B(-1, 2)^T \text{ thus } y = (y_2, y_3) = (-1, 2)$$

The reduced cost is

$$r_1 = c_1 - (c_2 y_2 + c_3 y_3) = -1 - (0 \cdot (-1) + 0 \cdot 2) = -1 < 0$$

So, we should put  $x_1$  into the basis (the reduced cost gets smaller).

---

**Algorithm 16** Simplex

---

```
1: Choose a starting basis  $B = (u_1 \dots u_m)$  (here  $A = (B \ N)$ )
2: repeat
3:   Compute the basic solution  $x$  for the basis  $B$ 
4:   for  $i \in \{m+1, \dots, n\}$  do
5:     Solve  $B(y_1, \dots, y_m)^\top = u_i$ 
6:     if  $y_k \leq 0$  for all  $k \in \{1, \dots, m\}$  then
7:       Stop, unbounded problem.
8:     end if
9:     Select  $j \in \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$ 
10:    Compute  $r_i = c_i - c_B^\top y$  where  $y = (y_1, \dots, y_m)^\top$ 
11:  end for
12:  if  $r_i \geq 0$  for all  $i \in \{m+1, \dots, n\}$  then
13:    Stop, we have an optimal solution.
14:  end if
15:  Select  $i \in \{m+1, \dots, n\}$  such that  $r_i < 0$ 
16:   $B \leftarrow B_{j \rightarrow i}$ 
17: until convergence
```

---

## Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^{\top} \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

## Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^{\top} \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

After following the reduced cost from the basis  $\{x_2, x_3\}$ , we end up in the basis  $\{x_1, x_2\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$  with  $c^{\top}x = 0$ .

## Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^T \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

After following the reduced cost from the basis  $\{x_2, x_3\}$ , we end up in the basis  $\{x_1, x_2\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$  with  $c^T x = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^T = B(-1/2, 1/2)^T \text{ thus } y = (y_1, y_2) = (-1/2, 1/2)$$

## Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^T \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

After following the reduced cost from the basis  $\{x_2, x_3\}$ , we end up in the basis  $\{x_1, x_2\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$  with  $c^T x = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^T = B(-1/2, 1/2)^T \text{ thus } y = (y_1, y_2) = (-1/2, 1/2)$$

Pivot about  $(2, 4)$ , that is  $x_2$  exchanges with  $x_4$  and  $\alpha = x_2/y_2 = 2$

## Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^T \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

After following the reduced cost from the basis  $\{x_2, x_3\}$ , we end up in the basis  $\{x_1, x_2\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$  with  $c^T x = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^T = B(-1/2, 1/2)^T \text{ thus } y = (y_1, y_2) = (-1/2, 1/2)$$

Pivot about  $(2, 4)$ , that is  $x_2$  exchanges with  $x_4$  and  $\alpha = x_2/y_2 = 2$

$$x_{2 \rightarrow 4} = ((x_1 - \alpha y_1), (x_2 - \alpha y_2), 0, \alpha) = (1, 0, 0, 2)$$

This is the minimizer!



## Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

After following the reduced cost from the basis  $\{x_2, x_3\}$ , we end up in the basis  $\{x_1, x_2\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$  with  $c^\top x = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^\top = B(-1/2, 1/2)^\top \text{ thus } y = (y_1, y_2) = (-1/2, 1/2)$$

Pivot about  $(2, 4)$ , that is  $x_2$  exchanges with  $x_4$  and  $\alpha = x_2/y_2 = 2$

$$x_{2 \rightarrow 4} = ((x_1 - \alpha y_1), (x_2 - \alpha y_2), 0, \alpha) = (1, 0, 0, 2)$$

This is the minimizer!

Does this always work?

## Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^T \quad A = (u_1 \ u_2 \ u_3 \ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

After following the reduced cost from the basis  $\{x_2, x_3\}$ , we end up in the basis  $\{x_1, x_2\}$  giving  $B = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$  and the basic solution  $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$  with  $c^T x = 0$ .

Consider  $x_4$  as a candidate for the basis:

$$u_4 = (0, 1)^T = B(-1/2, 1/2)^T \text{ thus } y = (y_1, y_2) = (-1/2, 1/2)$$

Pivot about  $(2, 4)$ , that is  $x_2$  exchanges with  $x_4$  and  $\alpha = x_2/y_2 = 2$

$$x_{2 \rightarrow 4} = ((x_1 - \alpha y_1), (x_2 - \alpha y_2), 0, \alpha) = (1, 0, 0, 2)$$

This is the minimizer!

Does this always work? Unfortunately, NO!

## Degenerate Case - Looping

Consider the following linear program:

$$\begin{array}{ll}\text{minimize} & z = -\frac{3}{4}x_1 + 150x_2 - \frac{1}{50}x_3 + 6x_4 \\ \text{subject to} & \frac{1}{4}x_1 - 60x_2 - \frac{1}{25}x_3 + 9x_4 + x_5 = 0 \\ & \frac{1}{2}x_1 - 90x_2 - \frac{1}{50}x_3 + 3x_4 + x_6 = 0 \\ & x_3 + x_7 = 1 \\ & x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0\end{array}$$

Executing the simplex method on this program starting with the basis  $\{x_5, x_6, x_7\}$  and always choosing  $i$  minimizing the reduced cost at line 15, eventually ends up back in the basis  $\{x_5, x_6, x_7\}$ . In other words, even though the reduced cost is always negative, the overall effect on the objective is 0.

# Convergence of Simplex Method

A solution is to use Bland's rule:

- ▶ Select the smallest index  $j$  at line 9.
- ▶ Select the smallest index  $i$  at line 15.

## Theorem 21

*If the simplex method is implemented using Bland's rule to select the entering and leaving variables, then the simplex method is guaranteed to terminate.*

# Simplex Convergence Summary

In a non-degenerate case:

- ▶ There is always a unique  $j$  to be selected at line 9.
- ▶ The objective of the basic solution decreases with each step.

Thus, a deterministic algorithm always terminates in a non-degenerate case.

# Simplex Convergence Summary

In a **non-degenerate case**:

- ▶ There is always a unique  $j$  to be selected at line 9.
- ▶ The objective of the basic solution decreases with each step.

Thus, a deterministic algorithm always terminates in a non-degenerate case.

In a **degenerate case**:

- ▶ We may have several  $j$  from which to select at line 9.
- ▶ Even though the reduced cost is negative, the basic solution may remain the same.

The simplex algorithm may cycle!

Using Bland's rule, the simplex method always converges to a minimizer or detects an unbounded LP.

## Two-Phase Simplex Algorithm

A Simplex algorithm is initialized with a basic feasible solution.

## Two-Phase Simplex Algorithm

A Simplex algorithm is initialized with a basic feasible solution.

How do we obtain such a solution? Given a standard form LP

$$\begin{array}{ll}\text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0\end{array}$$



## Two-Phase Simplex Algorithm

A Simplex algorithm is initialized with a basic feasible solution.

How do we obtain such a solution? Given a standard form LP

$$\begin{array}{ll}\text{minimize} & c^\top x \\ \text{subject to} & Ax = b \\ & x \geq 0\end{array}$$

We construct an artificial LP problem.

$$\begin{array}{ll}\text{minimize} & y_1 + y_2 + \cdots + y_m \\ \text{subject to} & (A \ I_m) \begin{pmatrix} x \\ y \end{pmatrix} = b \\ & \begin{pmatrix} x \\ y \end{pmatrix} \geq 0\end{array}$$

Here  $y = (y_1, \dots, y_m)^\top$  is a vector of artificial variables,  $I_m$  is the identity matrix of dimensions  $m \times m$ .

# Two-Phase Simplex Algorithm

Solve the *artificial LP problem*:

$$\begin{array}{ll}\text{minimize} & y_1 + y_2 + \cdots + y_m \\ \text{subject to} & [A \ I_m] \begin{pmatrix} x \\ y \end{pmatrix} = b \\ & \begin{pmatrix} x \\ y \end{pmatrix} \geq 0\end{array}$$

## Proposition 1

The original LP problem has a basic feasible solution iff the associated artificial LP problem has an optimal feasible solution with the objective function 0.

If we solve the artificial problem with  $y = 0$ , we obtain  $x$  such that  $Ax = b, x \geq 0$  is a basic feasible solution for the original problem.

If there is no such a solution to the artificial problem, there is no basic feasible solution, and hence no feasible solution, to the original problem.

# Two-Phase Simplex Algorithm

The procedure for solving a given LP problem using the Two-Phase Simplex algorithm is following:

- ▶ Solve the artificial LP problem using the Simplex algorithm:  
Initialize with the basic solution of the form:

$$(0, \dots, 0, b_1, \dots, b_m)^\top$$

- ▶ If the algorithm reaches a basic feasible solution  $(x_1, \dots, x_n, y_1, \dots, y_m)^\top$  of the artificial LP problem where

$$y_1 = \dots = y_m = 0$$

Use  $(x_1, \dots, x_n)^\top$  as the initial basic feasible solution to the original LP problem.

- ▶ Else stop, there is no feasible solution for the original LP problem.

# Linear Programming

## Properties

## LP Complexity

Iterations of the simplex algorithm can be implemented to compute the first step using  $\mathcal{O}(m^2n)$  arithmetic operations and each next step  $\mathcal{O}(mn)$ .

## LP Complexity

Iterations of the simplex algorithm can be implemented to compute the first step using  $\mathcal{O}(m^2n)$  arithmetic operations and each next step  $\mathcal{O}(mn)$ .

There are as many as  $\binom{n}{m}$  basic solutions (many of them likely infeasible). How large are these numbers?

$m$	$\binom{2m}{m}$
1	2
5	252
10	184756
20	$1 \times 10^{11}$
50	$1 \times 10^{29}$
100	$9 \times 10^{58}$
200	$1 \times 10^{119}$
300	$1 \times 10^{179}$
400	$2 \times 10^{239}$
500	$3 \times 10^{299}$

The number of iterations may be proportional to  $\binom{n}{m}$  that is EXPTIME.

# Linear Programming Complexity

Complexity of the simplex algorithm:

- ▶ In the worst case, the time complexity of the simplex algorithm is exponential. This holds for any deterministic pivoting rule.  
For details, see "How good is the simplex algorithm?" by Klee, Victor, and Minty, George J. Inequalities 1972.

# Linear Programming Complexity

Complexity of the simplex algorithm:

- ▶ In the worst case, the time complexity of the simplex algorithm is exponential. This holds for any deterministic pivoting rule. For details, see "How good is the simplex algorithm?" by Klee, Victor, and Minty, George J. *Inequalities* 1972.
- ▶ There is a theory that shows that examples with exponential complexity are rare. More precisely (but still very imprecisely)



# Linear Programming Complexity

Complexity of the simplex algorithm:

- ▶ In the worst case, the time complexity of the simplex algorithm is exponential. This holds for any deterministic pivoting rule.  
For details, see "How good is the simplex algorithm?" by Klee, Victor, and Minty, George J. *Inequalities* 1972.
- ▶ There is a theory that shows that examples with exponential complexity are rare. More precisely (but still very imprecisely)
  - ▶ Consider small random perturbations of the coefficients in the LP (use Gaussian noise with a small variance)

# Linear Programming Complexity

Complexity of the simplex algorithm:

- ▶ In the worst case, the time complexity of the simplex algorithm is exponential. This holds for any deterministic pivoting rule.  
For details, see "How good is the simplex algorithm?" by Klee, Victor, and Minty, George J. *Inequalities* 1972.
- ▶ There is a theory that shows that examples with exponential complexity are rare. More precisely (but still very imprecisely)
  - ▶ Consider small random perturbations of the coefficients in the LP (use Gaussian noise with a small variance)
  - ▶ Then, the expected computation time for the resulting instances of LP is polynomial.

For details, see "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time" by Daniel A. Spielman and Shang-Hua Teng in JACM 2004.

# Linear Programming Complexity

Complexity of the simplex algorithm:

- ▶ In the worst case, the time complexity of the simplex algorithm is exponential. This holds for any deterministic pivoting rule.  
For details, see "How good is the simplex algorithm?" by Klee, Victor, and Minty, George J. *Inequalities* 1972.
- ▶ There is a theory that shows that examples with exponential complexity are rare. More precisely (but still very imprecisely)
  - ▶ Consider small random perturbations of the coefficients in the LP (use Gaussian noise with a small variance)
  - ▶ Then, the expected computation time for the resulting instances of LP is polynomial.

For details, see "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time" by Daniel A. Spielman and Shang-Hua Teng in *JACM* 2004.

Is there a deterministic polynomial time algorithm for solving LP?

# Linear Programming Complexity

We assume that all coefficients are encoded in binary (more precisely, as fractions of two integers encoded in binary).

# Linear Programming Complexity

We assume that all coefficients are encoded in binary (more precisely, as fractions of two integers encoded in binary).

**Theorem 22 (Khachiyan, Doklady Akademii Nauk SSSR, 1979)**

*There is an algorithm that, for any linear program, computes an optimal solution in polynomial time.*

The algorithm uses the so-called ellipsoid method.

# Linear Programming Complexity

We assume that all coefficients are encoded in binary (more precisely, as fractions of two integers encoded in binary).

**Theorem 22 (Khachiyan, Doklady Akademii Nauk SSSR, 1979)**

*There is an algorithm that, for any linear program, computes an optimal solution in polynomial time.*

The algorithm uses the so-called ellipsoid method.

In practice, the Khachiyan's is not used.

# Linear Programming Complexity

We assume that all coefficients are encoded in binary (more precisely, as fractions of two integers encoded in binary).

**Theorem 22 (Khachiyan, Doklady Akademii Nauk SSSR, 1979)**

*There is an algorithm that, for any linear program, computes an optimal solution in polynomial time.*

The algorithm uses the so-called ellipsoid method.

In practice, the Khachiyan's is not used.

There is also a polynomial time algorithm (by Karmarkar) that has lower complexity upper bounds than the Khachiyan's and sometimes works even better than the simplex.

# Linear Programming in Practice

Heavily used tools for solving practical problems.

Several advanced linear programming solvers (usually parts of larger optimization packages) implement various heuristics for solving large-scale problems, such as sensitivity analysis.

See an overview of tools here:

[http://en.wikipedia.org/wiki/Linear\\_programming#Solvers\\_and\\_scripting\\_languages](http://en.wikipedia.org/wiki/Linear_programming#Solvers_and_scripting_languages)

For example, the well-known Gurobi solver uses the simplex algorithm to solve LP problems.