PV027 Optimization

Tomáš Brázdil

Resources & Prerequisities

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 & Prerequisities

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.

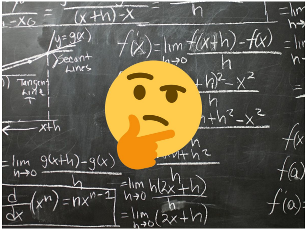
Evaluation

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 fully 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 fully 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 used 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

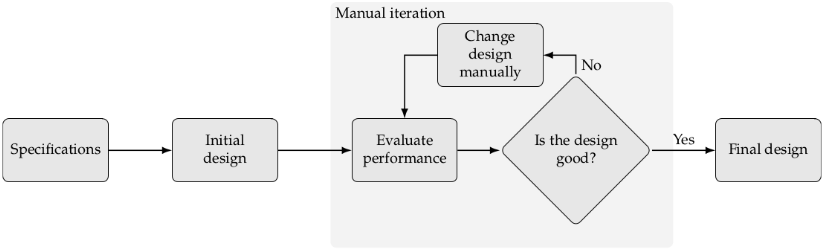
sklearn.linear_model.LogisticRegression

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*)

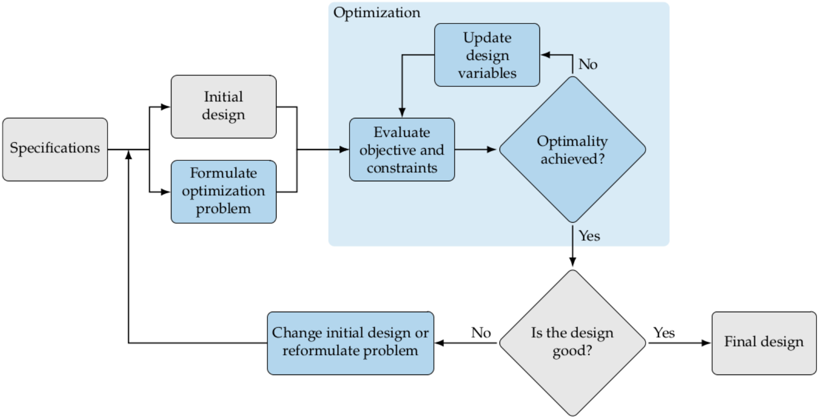
solver : *{'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'}, default='lbfgs'*

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:

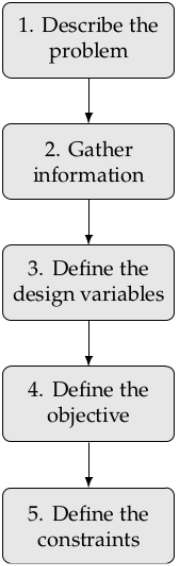
- Consider a company with several plants producing a single product but with different efficiency.
- The goal is to set the production of each plant so that demand for goods is satisfied, but overproduction is minimized.
- **First try**: Model each plant's production and maximize the total production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

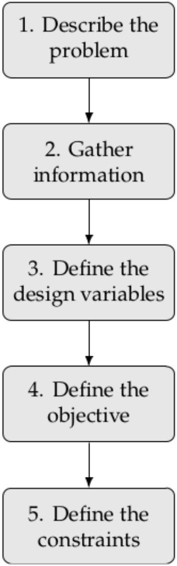


Optimization Problem Formulation

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).

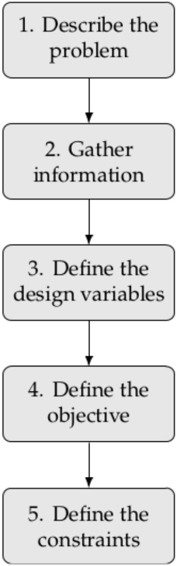


Optimization Problem Formulation

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

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

     (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.

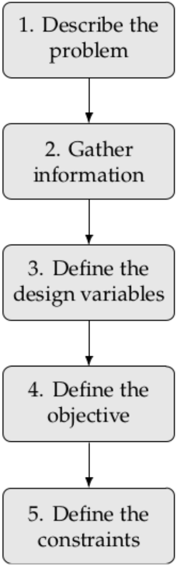


Optimization Problem Formulation

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 $c : \mathbb{R}^n \to \mathbb{R}$ 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

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

x a vector of real *variables*,

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

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

Optimization Problem Formally

Denote by

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

x a vector of real *variables*,

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

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

The optimization problem is to

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

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

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

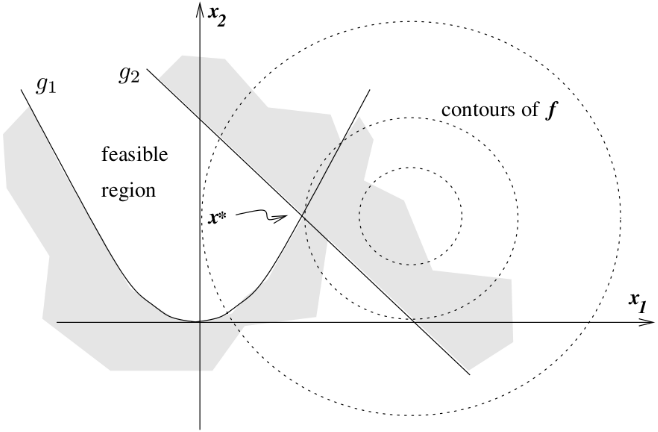
I.e.,

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

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

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

Constraints

Consider the constraints

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

Define the *feasibility region* by

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

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

Constraints

Consider the constraints

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

Define the *feasibility region* by

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

$x \in \mathcal{F}$ 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 contraints $-g_i(x) \leq 0$

Constraints

Consider the constraints

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

Define the *feasibility region* by

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

$x \in \mathcal{F}$ 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 contraints $-g_i(x) \leq 0$

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

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

Constraints

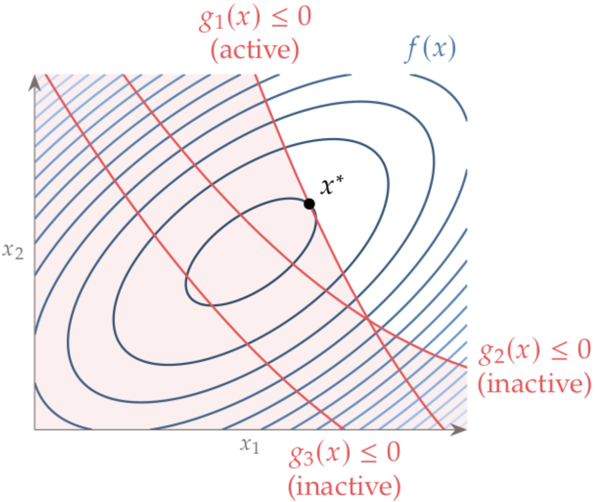
Inequality constraints $g_i(x) \leq 0$ can be *active* or *inactive*.

active

$$g_i(x^*) = 0$$

inactive

$$g_i(x^*) < 0$$



More Practical Example

The problem formulation:

- A company has two chemical factories F_1 and F_2, and a dozen retail outlets $R_1, \ldots, R_{12}$.
- 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 $R_1, \ldots, R_{12}$.
- 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 $j = 1, \ldots, 12$. 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 $j = 1, \ldots, 12$. 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

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

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

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

More Practical Example

Variables: x_{ij} for $i = 1, 2$ and $j = 1, \ldots, 12$. 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

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

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

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

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

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

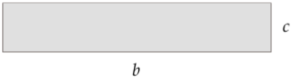
It constrains x_i only to integer values. This leads to so-called *integer programming*.

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.

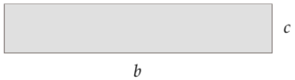


The parameters are call *span* b and *chord* c.

Wing Design Example

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

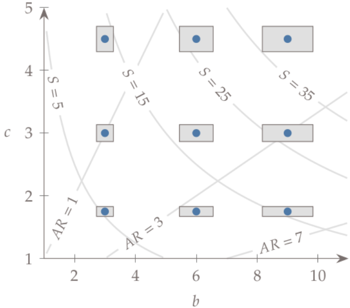
Assume a rectangular wing.

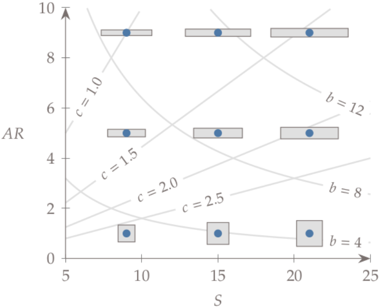


The parameters are call *span* b and *chord* c.

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

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





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 draft

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

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

Thus, we may obtain the lift velocity as

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

Similarly, various physics-based arguments provide approximations of the draft D and the propulsion efficiency η.

Wing Design Example

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

Wing Design Example

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

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

The induced draft 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 draft can be approximated by

$$D_f = k\,C_f\,q\,2.05\,S$$

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 μ is the air dynamic viscosity.

Wing Design Example

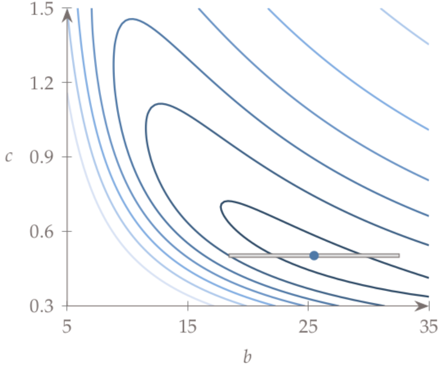
The propulsion efficiency η can be roughly approximated by the Gaussian efficiency curve.

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

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

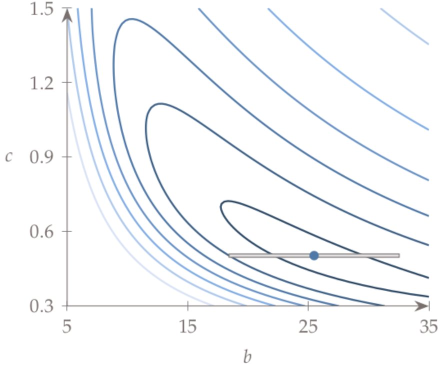
Wing Design Example

The objective function contours:



Wing Design Example

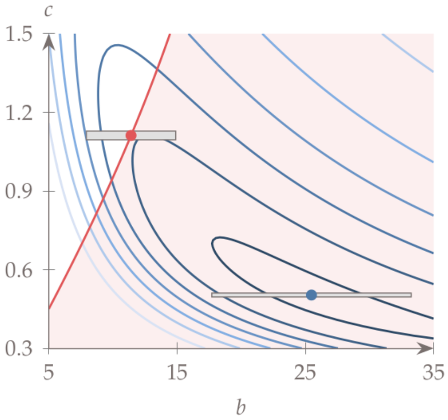
The objective function contours:



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!

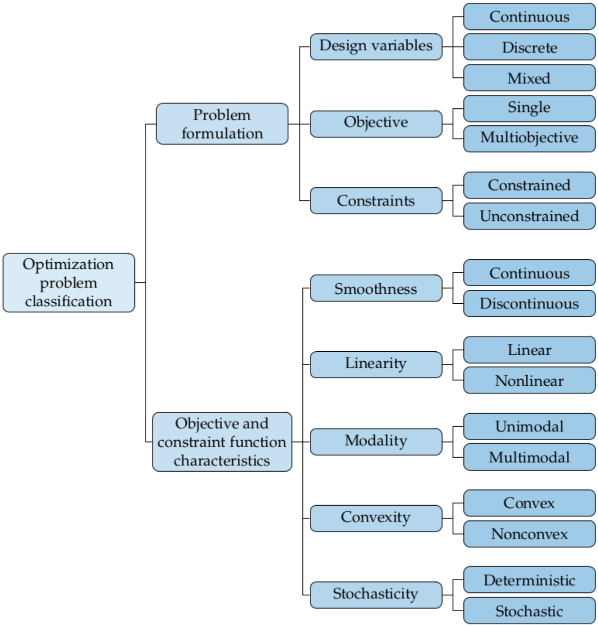
Wing Design Example

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

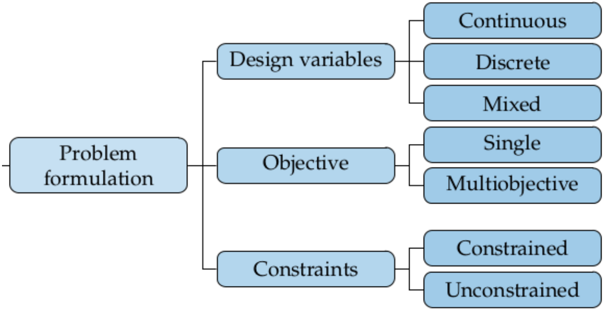


It looks like a reasonable wing ...

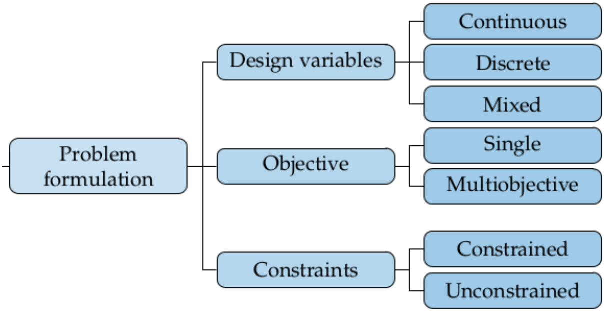
Optimization Problem Classification



Optimization Problem Classification

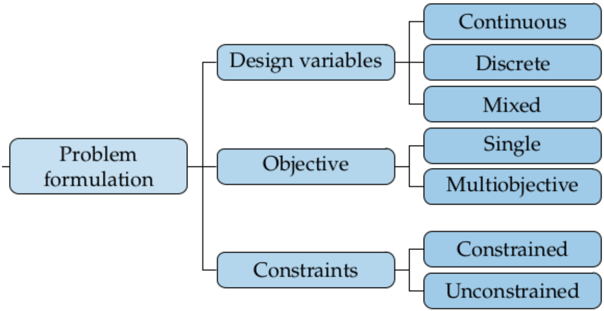


Optimization Problem Classification



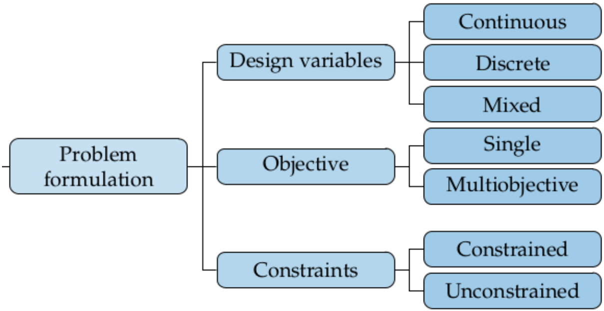
- *Continuous* allows only $x_i \in \mathbb{R}$, *discrete* allows only $x_i \in \mathbb{Z}$, mixed allows variables of both kinds.

Optimization Problem Classification



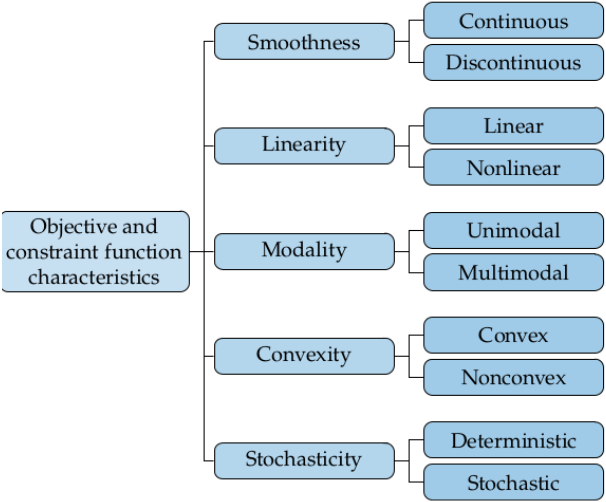
- *Continuous* allows only $x_i \in \mathbb{R}$, *discrete* allows only $x_i \in \mathbb{Z}$, mixed allows variables of both kinds.
- *Single-objective*: $f : \mathbb{R}^n \to \mathbb{R}$, *Multi-objective*: $f : \mathbb{R}^n \to \mathbb{R}^m$

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*: $f : \mathbb{R}^n \to \mathbb{R}$, *Multi-objective*: $f : \mathbb{R}^n \to \mathbb{R}^m$
- *Unconstrained*: No constraints, just the objective function.

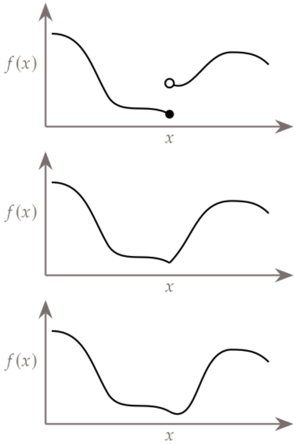
Optimization Problem Classification



Smoothness

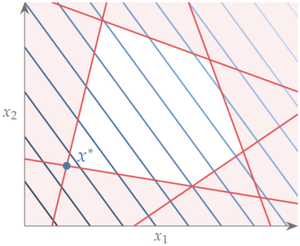
We consider various classes of problems depending on the smoothness properties of the objective/constraint functions:

- C^0: Continuous function
  Continuity allows us to estimate value in small neighborhoods.
- C^1: Continuous first derivatives
  Derivatives give information about the slope. If continuous, it changes smoothly, allowing us to estimate the slope locally.
- C^2: Continuous second derivatives
  Second derivatives inform about curvature.



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.

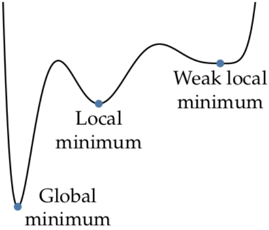
x^* is a *(weak) local minimiser* if there is $\varepsilon > 0$ such that

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

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:

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

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 $x_0, x_1, \ldots$ 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 $x_0, x_1, \ldots$ 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.
  (hard to decide whether a given point is a minimum (even a local one))

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

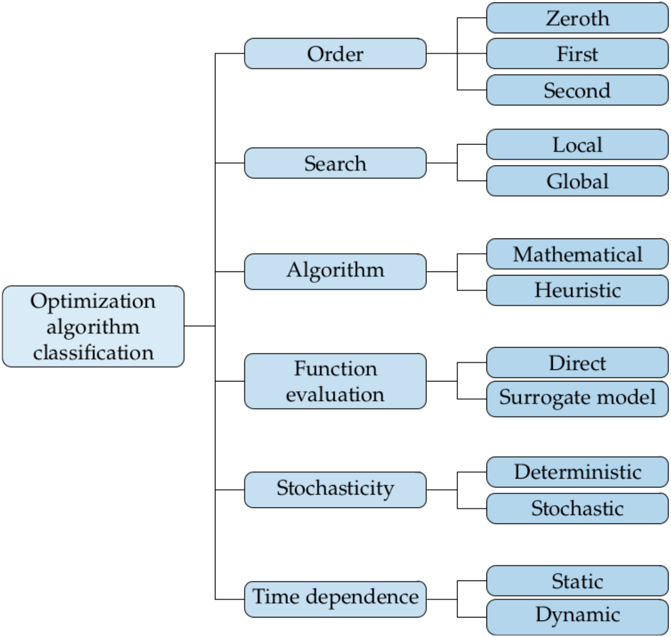
Typically, we are concerned with the following issues:

- *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


Optimization algorithm classification
Order
Zeroth
First
Second
Search
Local
Global
Algorithm
Mathematical
Heuristic
Function evaluation
Direct
Surrogate model
Stochasticity
Deterministic
Stochastic
Time dependence
Static
Dynamic

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 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 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.

Single-variable Objectives

Unconstrained Single Variable Optimization Problem

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

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 $f : \mathbb{R} \to \mathbb{R}$

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 $f : \mathbb{R} \to \mathbb{R}$

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 $f : \mathbb{R} \to \mathbb{R}$

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

Find x^* such that

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

Assume that

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

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 $f(x)$

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

Convergence of Single Variable Gradient Descent

Theorem 1

Assume that f is

- *continuously 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 $x, x' \in \mathbb{R}$.*

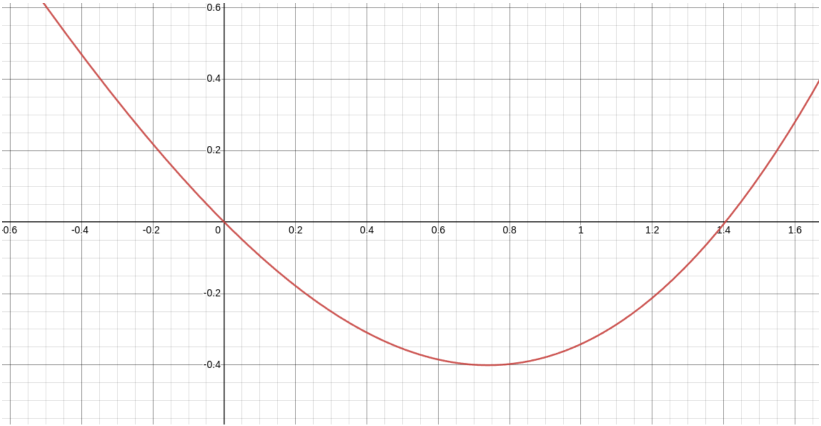
Consider a sequence $x_0, x_1, \ldots$ computed by the gradient descent algorithm for f. Assume a constant step length $\alpha \leq \frac{1}{L}$.
Then $\lim_{k\to\infty} |f'(x_k)| = 0$ and, moreover,

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

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:

$$x_1 = 0.87758 \qquad x_{12} = 0.73724$$
$$x_2 = 0.63901 \qquad x_{13} = 0.74033$$
$$x_3 = 0.80269 \qquad x_{14} = 0.73825$$
$$x_4 = 0.69478 \qquad x_{15} = 0.73965$$
$$x_5 = 0.76820 \qquad x_{16} = 0.73870$$
$$x_6 = 0.71917 \qquad x_{17} = 0.73934$$
$$x_7 = 0.75236 \qquad x_{18} = 0.73891$$
$$x_8 = 0.73008 \qquad x_{19} = 0.73920$$
$$x_9 = 0.74512 \qquad x_{20} = 0.73901$$
$$x_{10} = 0.73501 \qquad x_{21} = 0.73914$$
$$x_{11} = 0.74183 \qquad x_{22} = 0.73905$$

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

Example

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

$$
\begin{aligned}
x_1 &= 0.50000 \\
x_2 &= 0.87758 \\
x_3 &= 0.75830 \\
x_4 &= 0.74753 \\
x_5 &= 0.74399 \\
x_6 &= 0.74235 \\
x_7 &= 0.74144 \\
x_8 &= 0.74087 \\
x_9 &= 0.74050 \\
x_{10} &= 0.74024 \\
x_{11} &= 0.74004 \\
x_{12} &= 0.73990 \\
x_{13} &= 0.73978 \\
x_{14} &= 0.73969
\end{aligned}
$$

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

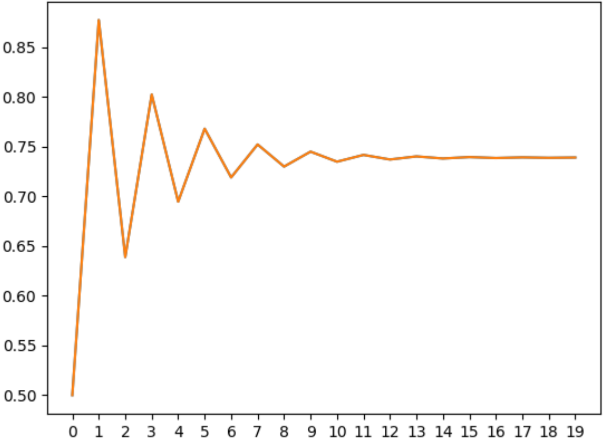
Frame Title

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

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

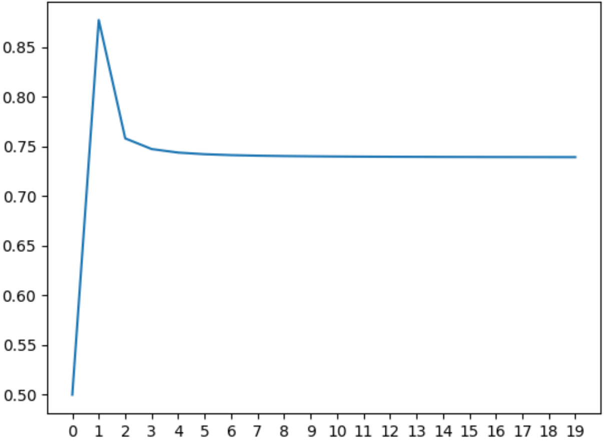
Example

Gradient descent with the step length $= 1.0$:



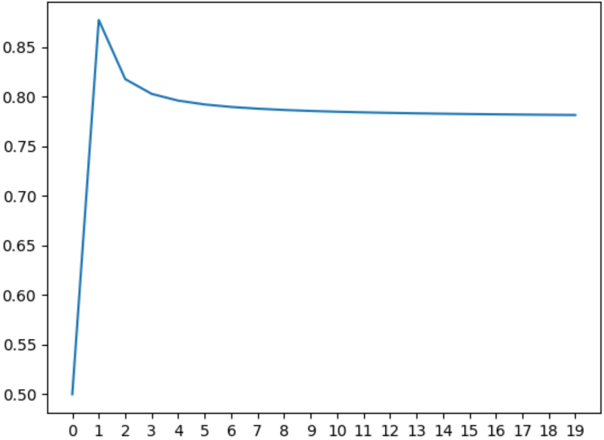
Example

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



Example

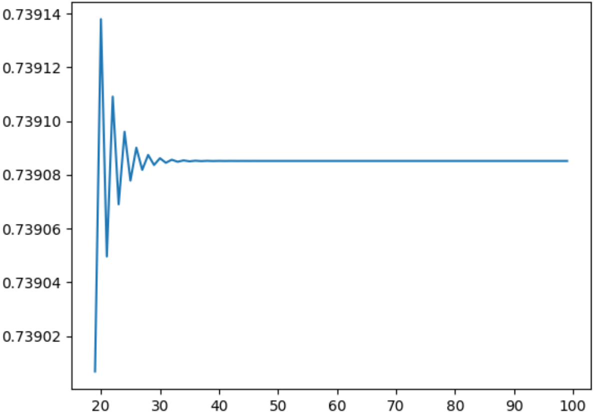
Gradient descent with the step length $= 1/k^2$:



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

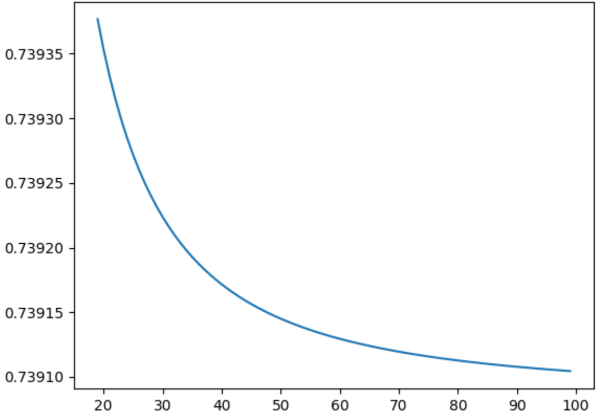
Example

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 $f : \mathbb{R} \to \mathbb{R}$

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 $f : \mathbb{R} \to \mathbb{R}$

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

Find x^* such that

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

Assume that

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

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 $f, g : \mathbb{R} \to \mathbb{R}$ we write $f = o(g)$ if

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

Taylor Series Approximation

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

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

Consider a function $f : \mathbb{R} \to \mathbb{R}$ 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 $f, g : \mathbb{R} \to \mathbb{R}$ we write $f = o(g)$ if

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

Consider a function $f : \mathbb{R} \to \mathbb{R}$ 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 $x_0, x_1, \ldots, x_k, \ldots$ as the GD.

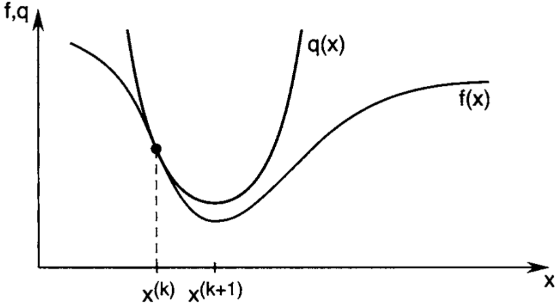
Newton's Method Idea

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.



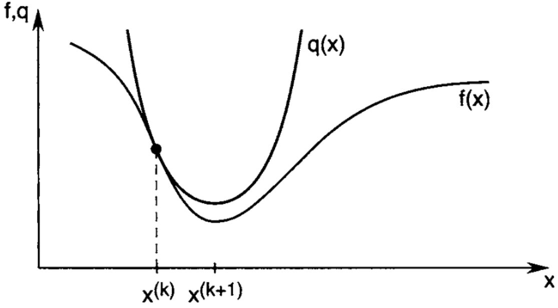
Newton's Method Idea

The method computes successive approximations $x_0, x_1, \ldots, x_k, \ldots$ 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 $f(x)$

```
Set k ← 0
while |x_{k+1} − x_k| > ε do
    Calculate the derivative: y′ ← f′(x_k)
    Calculate the second derivative : y″ ← f″(x_k)
    Update the estimate: x_{k+1} ← x_k − y′/y″
    Increment k
end while
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

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

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

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

$\ldots$

Example

Proceeding similarly, we obtain

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

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

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

$$\ldots$$

Note that

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

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

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

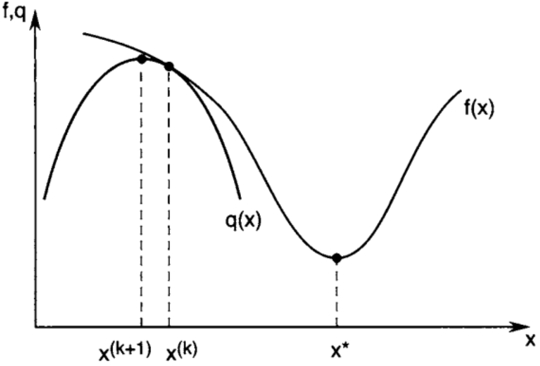
So, we conclude that $x^* \approx x_4$ is a strict minimizer.

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:

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

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:

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

Superlinear Convergence

Convergence is superlinear if:

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

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:

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

Quadratic Convergence of Newton's Method

Quadratic Convergence

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

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

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

Theorem 2 (Quadratic Convergence of Newton's Method)

Let $f : \mathbb{R} \to \mathbb{R}$ 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

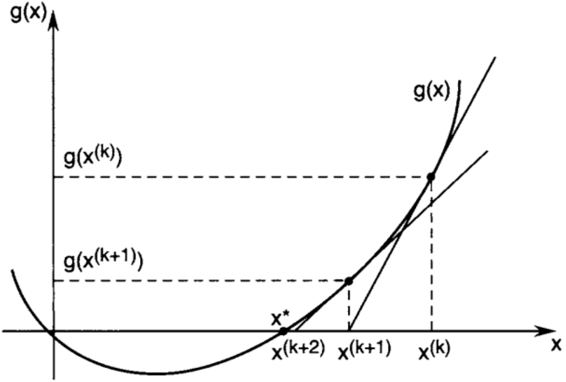
Newton's method is also a technique for finding roots of functions.
In our case, this means finding a root of f'.

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,

$$x_2 = 1.0 - \frac{1.0 - 0.5}{(1.0 - \cos 1.0) - (0.5 - \cos 0.5)}(0.5 - \cos 0.5)$$
$$= 0.7254$$

Example

Continuing, we obtain:

$$
\begin{aligned}
x_0 &= 0.5 \\
x_1 &= 1.0 \\
x_2 &= 0.72548 \\
x_3 &= 0.73839 \\
x_4 &= 0.739087 \\
x_5 &= 0.739085132 \\
x_6 &= 0.739085133
\end{aligned}
$$

Example

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

$$x_0 = 0.5$$
$$x_1 = 0.7552$$
$$x_2 = 0.7381$$
$$x_3 = 0.739081$$
$$x_5 = 0.7390851339$$
$$x_6 = 0.7390851332$$
$$\ldots$$

Compare with Newton's method:

$$x_0 = 0.5$$
$$x_1 = 0.7552$$
$$x_2 = 0.7391$$
$$x_3 = 0.7390851339$$
$$x_4 = 0.7390851321516067229$$
$$x_5 = 0.7390851321516067229$$
$$\ldots$$

Superlinear Convergence of Secant Method

Theorem 3 (Superlinear Convergence of Secant Method)

Assume $f : \mathbb{R} \to \mathbb{R}$ 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,

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

Secant Method for Root Finding

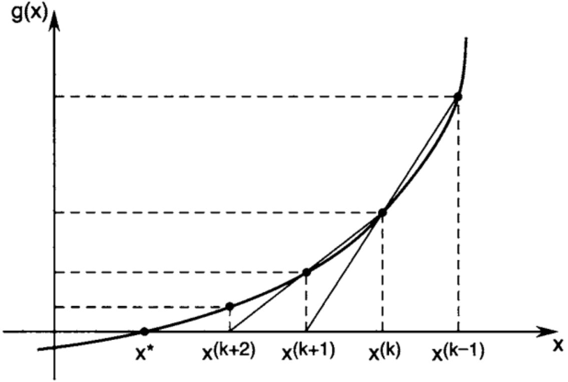
As for Newton's method of tangents, the secant method can be seen as a method for finding a root of f'.

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 to an extremum under $\mathcal{C}^2$ assumption (quadratic convergence)
  - 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 to an extremum under $\mathcal{C}^2$ assumption (quadratic convergence)
  - 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.

Constrained Single Variable Optimization Problem

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

A variable x

A constraint

$$a_0 \leq x \leq b_0$$

Consider the following cases:

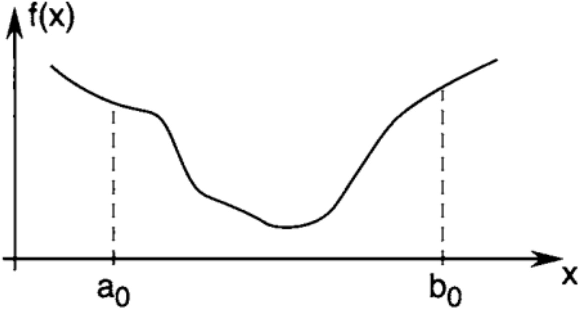
- f unimodal on $[a_0, b_0]$
- f continuously differentiable on $[a_0, b_0]$
- f twice continuously differentiable on $[a_0, b_0]$

Unimodal Function Minimization

We assume only unimodality on $[a_0, b_0]$ where the single extremum is a minimum.

More precisely, we assume that there is x^* such that

- $f(x') > f(x'')$ for all $x', x'' \in [a_0, x^*]$ satisfying $x' < x''$
- $f(x') < f(x'')$ for all $x', x'' \in [x^*, b_0]$ satisfying $x' < x''$

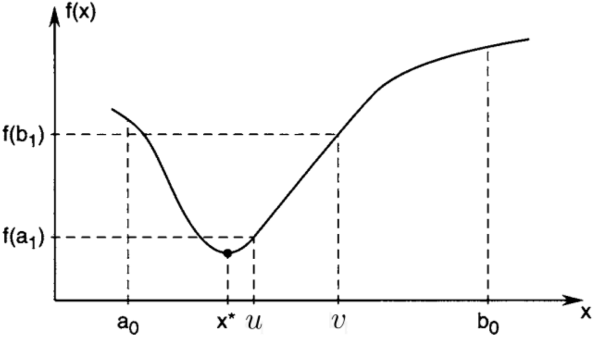


Assume that even a single evaluation of f is costly.

Minimize the number of evaluations searching for the minimum.

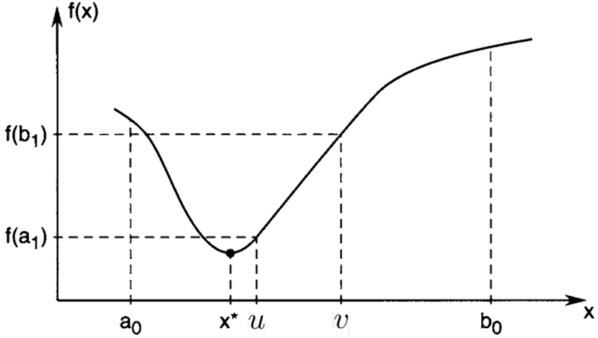
Simple Algorithm

Select u, v such that $a_0 < u < v < b_0$.



Simple Algorithm

Select u, v such that $a_0 < u < v < b_0$.



Observe that

- If $f(u) < f(v)$, then the minimizer must lie in $[a_0, v]$.
- If $f(u) \geq f(v)$, then the minimizer must lie in $[u, b_0]$.

Continue the search in the resulting interval.

The Algorithm

An abstract search algorithm:

```
Initialize a_0 < b_0
for k = 0 to K − 1 do
    Choose u_k, v_k such that a_k < u_k < v_k < b_k
    if f(u_k) < f(v_k) then
        a_{k+1} ← a_k and b_{k+1} ← v_k
    else
        a_{k+1} ← u_k and b_{k+1} ← b_k
    end if
end for
```

The Algorithm

An abstract search algorithm:

```
Initialize $a_0 < b_0$
for $k = 0$ to $K - 1$ do
    Choose $u_k, v_k$ such that $a_k < u_k < v_k < b_k$
    if $f(u_k) < f(v_k)$ then
        $a_{k+1} \leftarrow a_k$ and $b_{k+1} \leftarrow v_k$
    else
        $a_{k+1} \leftarrow u_k$ and $b_{k+1} \leftarrow b_k$
    end if
end for
```

The algorithm produces a sequence of intervals:

$$[a_0, b_0] \supset [a_1, b_1] \supset [a_2, b_2] \supset \cdots \supset [a_K, b_K]$$

where $[a_K, b_K]$ contains the minimizer of f.

The algorithm evaluates f twice in every iteration.

Is it necessary?

Intermediate Points

Choose u_k, v_k symmetrically in the following sense:

$$u_k - a_k = b_k - v_k = \varrho(b_k - a_k)$$

for some $\varrho \in (0, 1)$.

Intermediate Points

Choose u_k, v_k symmetrically in the following sense:

$$u_k - a_k = b_k - v_k = \varrho(b_k - a_k)$$

for some $\varrho \in (0, 1)$. The algorithm will then look as follows:

```
Initialize a_0 < b_0
for k = 0 to K − 1 do
    u_k ← a_k + ρ(b_k − a_k)
    v_k ← b_k − ρ(b_k − a_k)
    if f(u_k) < f(v_k) then
        a_{k+1} ← a_k and b_{k+1} ← v_k
    else
        a_{k+1} ← u_k and b_{k+1} ← b_k
    end if
end for
```

Intermediate Points

Assume $a_0 = 0$ and $b_0 = 1$.

Intermediate Points

Assume $a_0 = 0$ and $b_0 = 1$.

Suppose that we have just computed a_1 and b_1 and that, e.g., the minimizer lies in $[a_0, v_0]$, i.e., $a_1 = a_0$, $b_1 = v_0$, and $u_0 \in [a_0, b_1]$.

Intermediate Points

Assume $a_0 = 0$ and $b_0 = 1$.

Suppose that we have just computed a_1 and b_1 and that, e.g., the minimizer lies in $[a_0, v_0]$, i.e., $a_1 = a_0$, $b_1 = v_0$, and $u_0 \in [a_0, b_1]$.

We are computing u_1, v_1 and need to get $f(u_1)$ and $f(v_1)$.

Note that we have already computed $f(u_0)$. So let us set ϱ so that v_1 coincides with u_0.

Intermediate Points

Assume $a_0 = 0$ and $b_0 = 1$.

Suppose that we have just computed a_1 and b_1 and that, e.g., the minimizer lies in $[a_0, v_0]$, i.e., $a_1 = a_0$, $b_1 = v_0$, and $u_0 \in [a_0, b_1]$.

We are computing u_1, v_1 and need to get $f(u_1)$ and $f(v_1)$.

Note that we have already computed $f(u_0)$. So let us set ϱ so that v_1 coincides with u_0.

As $v_1 = b_1 - \rho(b_1 - a_1) = b_1 - \rho(b_1 - a_0)$,

Intermediate Points

Assume $a_0 = 0$ and $b_0 = 1$.

Suppose that we have just computed a_1 and b_1 and that, e.g., the minimizer lies in $[a_0, v_0]$, i.e., $a_1 = a_0$, $b_1 = v_0$, and $u_0 \in [a_0, b_1]$.

We are computing u_1, v_1 and need to get $f(u_1)$ and $f(v_1)$.

Note that we have already computed $f(u_0)$. So let us set ϱ so that v_1 coincides with u_0.

As $v_1 = b_1 - \rho(b_1 - a_1) = b_1 - \rho(b_1 - a_0)$, demanding $v_1 = u_0$ implies

$$u_0 = b_1 - \rho(b_1 - a_0) \qquad \Rightarrow \qquad \varrho(b_1 - a_0) = b_1 - u_0$$

Intermediate Points

Assume $a_0 = 0$ and $b_0 = 1$.

Suppose that we have just computed a_1 and b_1 and that, e.g., the minimizer lies in $[a_0, v_0]$, i.e., $a_1 = a_0$, $b_1 = v_0$, and $u_0 \in [a_0, b_1]$.

We are computing u_1, v_1 and need to get $f(u_1)$ and $f(v_1)$.

Note that we have already computed $f(u_0)$. So let us set ϱ so that v_1 coincides with u_0.

As $v_1 = b_1 - \rho(b_1 - a_1) = b_1 - \rho(b_1 - a_0)$, demanding $v_1 = u_0$ implies

$$u_0 = b_1 - \rho(b_1 - a_0) \qquad \Rightarrow \qquad \varrho(b_1 - a_0) = b_1 - u_0$$

Since $b_1 - a_0 = 1 - \varrho$ and $b_1 - u_0 = 1 - 2\varrho$ we have

$$\varrho(1 - \varrho) = 1 - 2\varrho \quad \Leftrightarrow \quad \varrho^2 - 3\varrho + 1 = 0$$

Intermediate Points

Assume $a_0 = 0$ and $b_0 = 1$.

Suppose that we have just computed a_1 and b_1 and that, e.g., the minimizer lies in $[a_0, v_0]$, i.e., $a_1 = a_0$, $b_1 = v_0$, and $u_0 \in [a_0, b_1]$.

We are computing u_1, v_1 and need to get $f(u_1)$ and $f(v_1)$.

Note that we have already computed $f(u_0)$. So let us set ϱ so that v_1 coincides with u_0.

As $v_1 = b_1 - \rho(b_1 - a_1) = b_1 - \rho(b_1 - a_0)$, demanding $v_1 = u_0$ implies

$$u_0 = b_1 - \rho(b_1 - a_0) \qquad \Rightarrow \qquad \varrho(b_1 - a_0) = b_1 - u_0$$

Since $b_1 - a_0 = 1 - \varrho$ and $b_1 - u_0 = 1 - 2\varrho$ we have

$$\varrho(1 - \varrho) = 1 - 2\varrho \quad \Leftrightarrow \quad \varrho^2 - 3\varrho + 1 = 0$$

Solving to $\rho_1 = \frac{3+\sqrt{5}}{2}, \quad \rho_2 = \frac{3-\sqrt{5}}{2}$, we consider $\varrho = \frac{3-\sqrt{5}}{2}$

Golden Section Search

Choosing $u_k = a_k + \rho(b_k - a_k)$ and $v_k = b_k - \rho(b_k - a_k)$ allows us to reuse one of the values of $f(u_{k-1})$ and $f(v_{k-1})$.

Initialize $a_0 < b_0$
for $k = 0$ **to** $K - 1$ **do**
    $u_k \leftarrow a_k + \rho(b_k - a_k)$
    $v_k \leftarrow b_k - \rho(b_k - a_k)$
    if $u_k = v_{k-1}$ **then**
        $fu_k \leftarrow fv_{k-1}$ and $fu_k \leftarrow f(v_k)$
    else
        $fu_k \leftarrow f(u_k)$ and set $fv_k = fu_{k-1}$
    end if
    if $fu_k < fv_k$ **then**
        $a_{k+1} \leftarrow a_k$ and $b_{k+1} \leftarrow v_k$
    else
        $a_{k+1} \leftarrow u_k$ and $b_{k+1} \leftarrow b_k$
    end if
end for

Golden Section Search

Note that

$$\rho = \frac{3-\sqrt{5}}{2} \approx 0.61803$$

and thus

$$b_k - a_k \approx 0.61803 \cdot (b_{k-1} - a_{k-1})$$

which for $a_0 = 0$ and $b_0 = 1$ means

$$b_k - a_k = (1 - \varrho)^k \approx (0.61803)^k$$

Example

Consider f defined by

$$f(x) = x^4 - 14x^3 + 60x^2 - 70x$$

on the interval $[0, 2]$.

Example

Consider f defined by

$$f(x) = x^4 - 14x^3 + 60x^2 - 70x$$

on the interval $[0, 2]$.

By definition, $a_0 = 0$ and $b_0 = 2$.

$$u_0 = a_0 + \rho\,(b_0 - a_0) = 0.7639$$
$$v_0 = a_0 + (1 - \rho)\,(b_0 - a_0) = 1.236$$

Here $\rho = (3 - \sqrt{5})/2$.

Example

Consider f defined by

$$f(x) = x^4 - 14x^3 + 60x^2 - 70x$$

on the interval $[0, 2]$.

By definition, $a_0 = 0$ and $b_0 = 2$.

$$u_0 = a_0 + \rho(b_0 - a_0) = 0.7639$$
$$v_0 = a_0 + (1 - \rho)(b_0 - a_0) = 1.236$$

Here $\rho = (3 - \sqrt{5})/2$.

In the first step, we have to compute both fu_0 and fv_0:

$$fu_0 = f(u_0) = -24.36$$
$$fv_0 = f(v_0) = -18.96$$

$fu_0 < fv_0$ and thus $a_1 = a_0 = 0$ and $b_1 = v_0 = 1.236$.

Example

We have $a_1 = a_0 = 0$ and $b_1 = v_0 = 1.236$.

Example

We have $a_1 = a_0 = 0$ and $b_1 = v_0 = 1.236$.

Now compute u_1 and v_1 as follows

$$u_1 = a_1 + \rho\,(b_1 - a_1) = 0.4721$$
$$v_1 = a_1 + (1 - \rho)\,(b_1 - a_1) = 0.7639$$

Note that v_1 coincides with u_0 as expected.

Example

We have $a_1 = a_0 = 0$ and $b_1 = v_0 = 1.236$.

Now compute u_1 and v_1 as follows

$$u_1 = a_1 + \rho\,(b_1 - a_1) = 0.4721$$
$$v_1 = a_1 + (1 - \rho)\,(b_1 - a_1) = 0.7639$$

Note that v_1 coincides with u_0 as expected.

So we only have to compute

$$fu_1 = f\,(u_1) = -21.1$$

and put $fv_1 = fu_0$.

As $fv_1 < fu_1$ we obtain $a_2 = 0.4721$ and $b_2 = 1.236$.

... and so on.

Summary of Golden Search

A method for solving constrained problems where the objective is unimodal.

Straightforward method with guaranteed convergence, which in every step evaluates the objective only once.

The implementation in Scipy:
`https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.golden.html`

Constrained Gradient Descent and Newton's Method

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

A variable x

A constraints

$$a_0 \leq x \leq b_0$$

(find your $\boldsymbol{c}$ functions and the constraints)

Constrained Gradient Descent and Newton's Method

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

A variable x

A constraints

$$a_0 \leq x \leq b_0$$

(find your $\boldsymbol{c}$ functions and the constraints)

Consider the following cases:

- f unimodal on $[a_0, b_0]$
- 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, $x_0, x_1, \ldots, x_k, \ldots$.
The index k will usually indicate that x_k is the k-the 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.

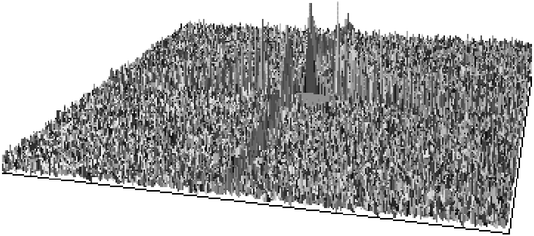
We ocasionally use the matrix morn $||A||$, consistent with the Euclidean norm, defined by

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

Here λ_1 is the largest eigenvalue of $A^\top A$.

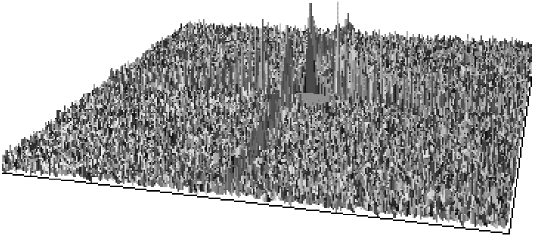
How to Recognize (Local) Minimum

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

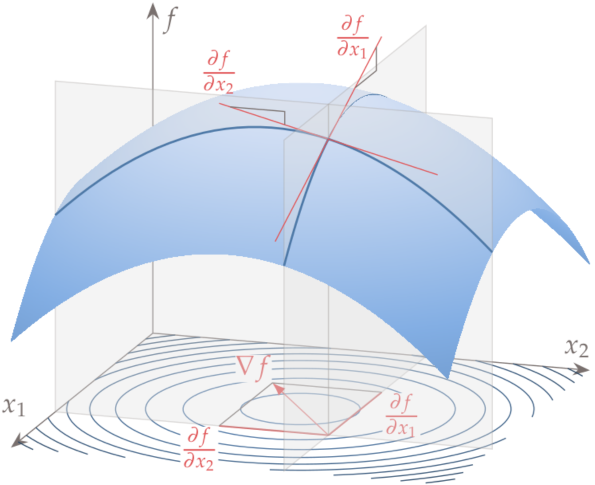
The gradient of $f : \mathbb{R}^n \to \mathbb{R}$, denoted by $\nabla f(x)$, is a column vector of first-order partial derivatives of the function concerning each variable:

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

Where each partial derivative is defined as the following limit:

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

Gradient



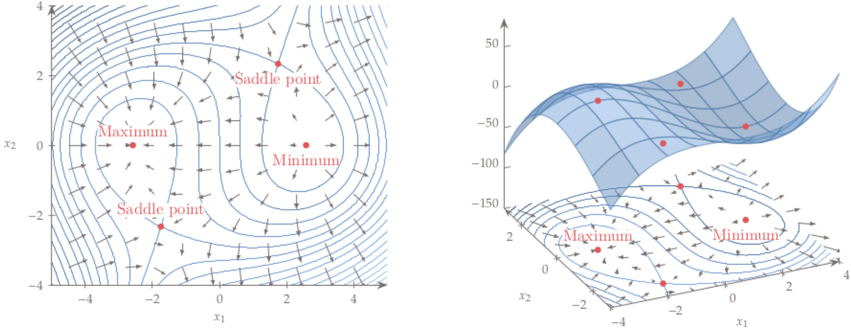
The gradient is a vector pointing in the direction of the most significant function increase from the current point.

Gradient

Consider the following function of two variables:

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

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

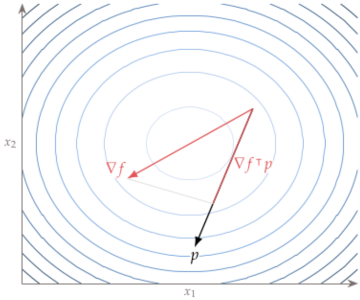


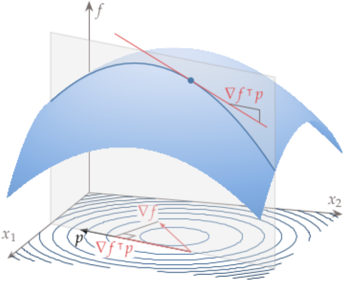
Directional Derivatives vs Gradient

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

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

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 θ is the angle between ∇f and p.

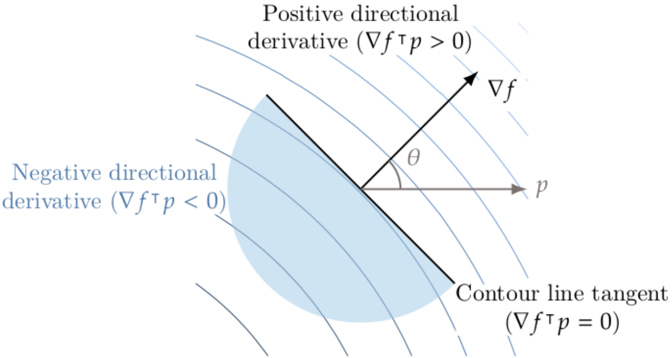
Geometry of Gradient

Consider the geometric interpretation of the dot product:

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

Here θ is the angle between ∇f and p.

The directional derivative is maximized by $\theta = 0$, i.e. when ∇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 $g(t) = f(x + tp)$ and let $h_i(t) = \frac{\partial f}{\partial x_i}(x + tp)$ for $t \in \mathbb{R}$.

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

Geometry of Hessian

Let x be fixed and let $g(t) = f(x + tp)$ and let $h_i(t) = \frac{\partial f}{\partial x_i}(x + tp)$ for $t \in \mathbb{R}$.

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

$$g'(t) = f(x + tp)' = [\nabla f(x + tp)]^\top p = \sum_{i=1}^{n} h_i(t) p_i$$

Geometry of Hessian

Let x be fixed and let $g(t) = f(x + tp)$ and let $h_i(t) = \frac{\partial f}{\partial x_i}(x + tp)$ for $t \in \mathbb{R}$.

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

$$g'(t) = f(x + tp)' = [\nabla f(x + tp)]^\top p = \sum_{i=1}^{n} h_i(t) p_i$$

$$h_i'(t) = \left[\nabla \frac{\partial f}{\partial x_i}(x + tp)\right]^\top p = \sum_{j=1}^{n} \left(\frac{\partial f}{\partial x_i \partial x_j}(x + tp) \right) p_j$$

$$= [H(x + tp)p]_i$$

Geometry of Hessian

Let x be fixed and let $g(t) = f(x + tp)$ and let $h_i(t) = \frac{\partial f}{\partial x_i}(x + tp)$ for $t \in \mathbb{R}$.

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

$$g'(t) = f(x + tp)' = [\nabla f(x + tp)]^\top p = \sum_{i=1}^{n} h_i(t) p_i$$

$$\begin{aligned} h_i'(t) &= \left[\nabla \frac{\partial f}{\partial x_i}(x + tp)\right]^\top p = \sum_{j=1}^{n} \left(\frac{\partial f}{\partial x_i \partial x_j}(x + tp)\right) p_j \\ &= [H(x + tp)p]_i \end{aligned}$$

$$g''(t) = \sum_{i=1}^{n} h_i'(t) p_i = \sum_{i=1}^{n} [H(x + tp)p]_i p_i = p^\top H(x + tp) p$$

Geometry of Hessian

Let x be fixed and let $g(t) = f(x + tp)$ and let $h_i(t) = \frac{\partial f}{\partial x_i}(x + tp)$ for $t \in \mathbb{R}$.

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

$$g'(t) = f(x + tp)' = [\nabla f(x + tp)]^\top p = \sum_{i=1}^{n} h_i(t) p_i$$

$$h_i'(t) = \left[\nabla \frac{\partial f}{\partial x_i}(x + tp)\right]^\top p = \sum_{j=1}^{n} \left(\frac{\partial f}{\partial x_i \partial x_j}(x + tp) \right) p_j$$

$$= [H(x + tp)p]_i$$

$$g''(t) = \sum_{i=1}^{n} h_i'(t) p_i = \sum_{i=1}^{n} [H(x + tp)p]_i p_i = p^\top H(x + tp)p$$

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:

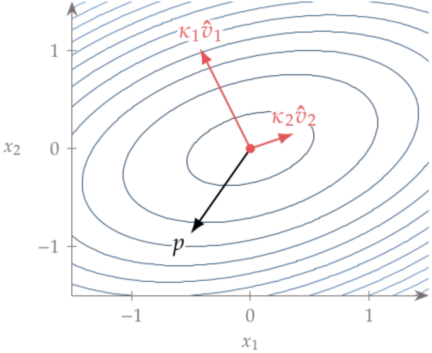
$$H\hat{v}_k = \kappa_k \hat{v}_k$$

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

$$H = RDR^{-1} = RDR^\top$$

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

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



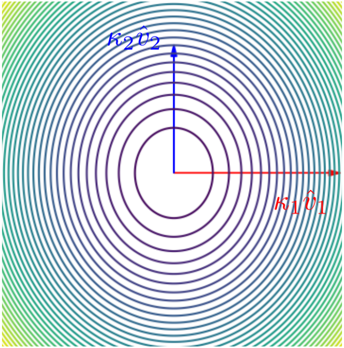
Consider $f(x) = x^\top Hx$ where

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

The eigenvalues are

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

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



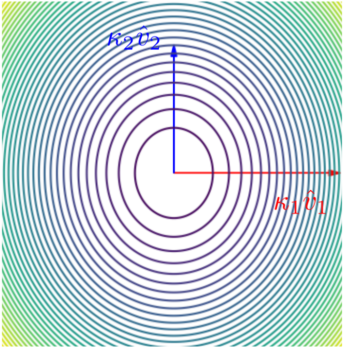
Consider $f(x) = x^\top Hx$ where

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

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 we get

$$g(t) = f(0 + tp) = t^2 \left(\kappa_1 p_1^2 + \kappa_2 p_2^2\right)$$

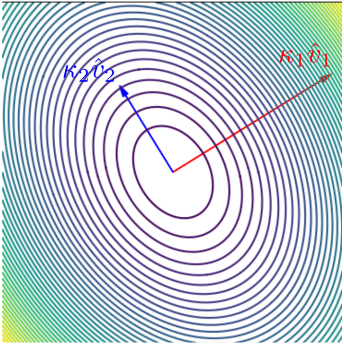
which is a parabola with $g'' = 2\left(\kappa_1 p_1^2 + \kappa_2 p_2^2\right)$.

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

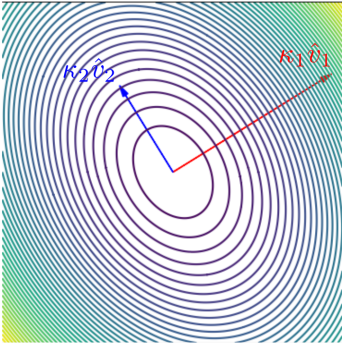
$$\kappa_1 = \frac{1}{6}(7+\sqrt{5}) \quad \kappa_2 = \frac{1}{6}(7-\sqrt{5})$$

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 $f(x) = x^\top H x$ 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

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

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

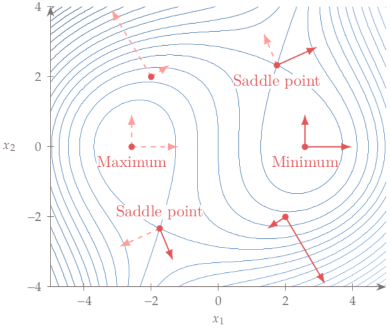
Hessian Visualization Example

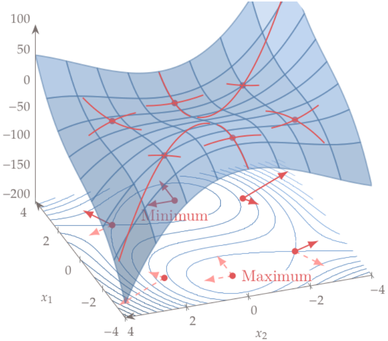
Consider

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

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

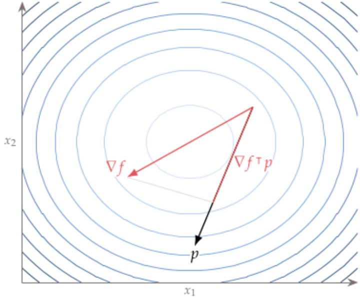
$$f(x+p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T H(x) p + o(||p||^2).$$

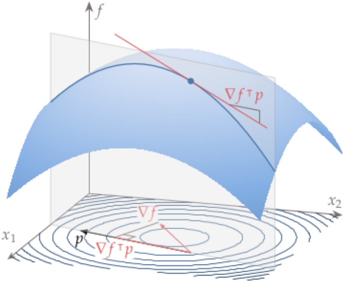
Here $H = \nabla^2 f$ *is the Hessian of* f.

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

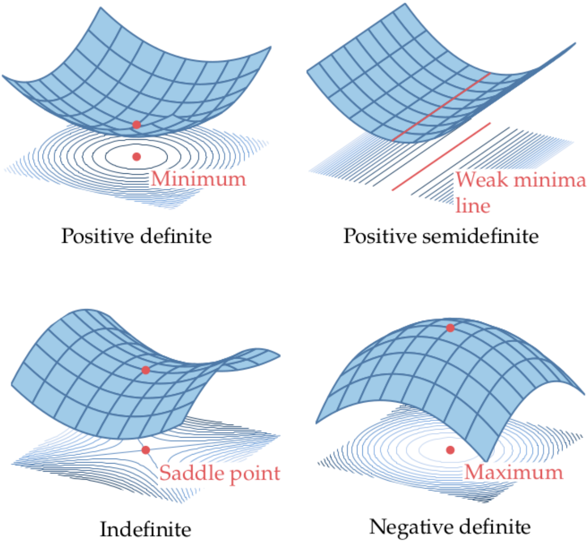
Note that $\nabla f(x^*) = 0$ does not tell us whether x^* is a minimizer, maximizer, or a saddle point.

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 $p^\top Hp > 0$ for all p
  iff all eigenvalues of H are positive
- H is positive semi-definite if $p^\top Hp \geq 0$ for all p
  iff all eigenvalues of H are nonnegative
- H is negative semi-definite if $p^\top Hp \leq 0$ for all p
  iff all eigenvalues of H are nonpositive
- H is negative definite if $p^\top Hp < 0$ 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



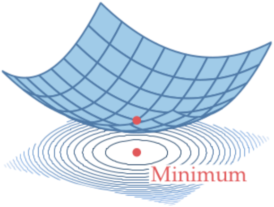
Second-Order Necessary Condition

Theorem 6 (Second-Order Necessary Conditions)

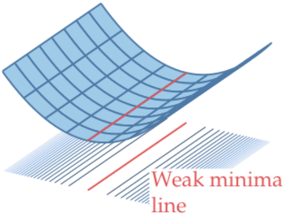
If x^* is a local minimizer of f and $\nabla^2 f$ is continuous in a neighborhood of x^*, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite.

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

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

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

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

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:

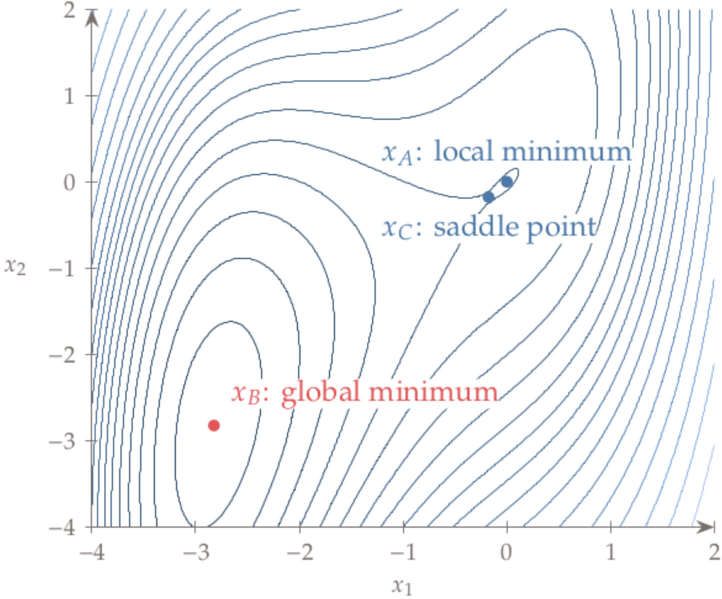
$$H(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 6x_1^2 + 12x_1 + 3 & -2 \\ -2 & 2 \end{bmatrix}.$$

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



x_A: local minimum
x_C: saddle point
x_B: global minimum
x_2
x_1
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2

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 $f : \mathbb{R}^n \to \mathbb{R}$ *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 ∇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. □

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 $x_0, x_1, \ldots$
- 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 $x_0, x_1, \ldots, x_k$.

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 $x_0, x_1, \ldots$
- 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 $x_0, x_1, \ldots, x_k$.

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* α_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* α_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* α_k

and computes

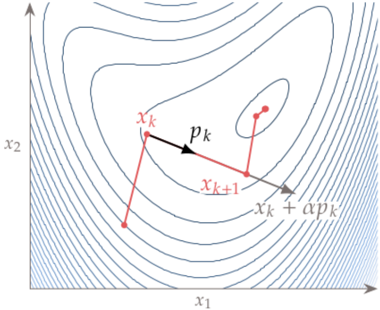
$$x_{k+1} = x_k + \alpha_k p_k$$

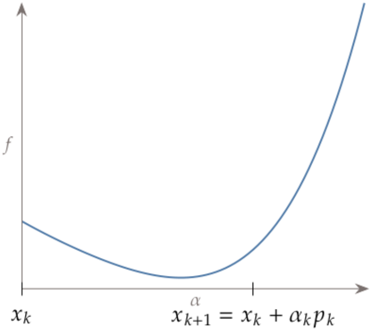
The vector p_k should be a *descent* direction, i.e., a direction in which f decreases locally.

α_k is selected to approximately solve

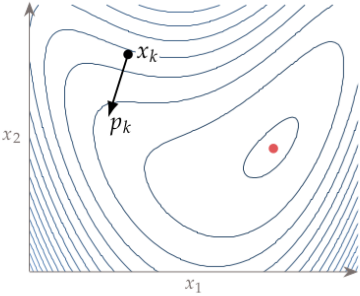
$$\min_{\alpha > 0} f(x_k + \alpha p_k)$$

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

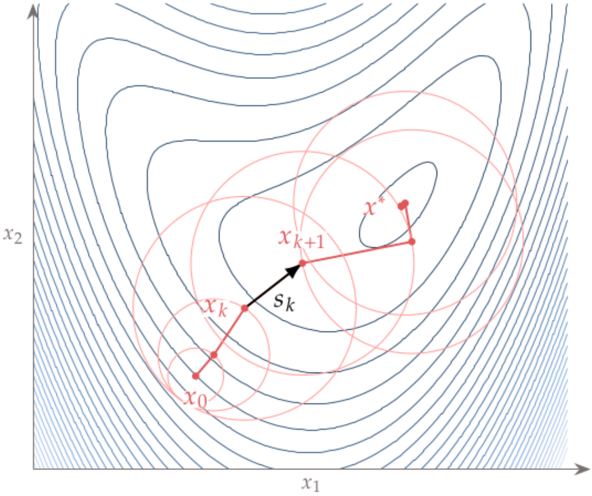
$$\min_{x \in R} m_k(x)$$

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.

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

Where B_k approximates the Hessian of f at x_k.


x_2
x_1
x_0
x_k
s_k
x_{k+1}
x^*

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.

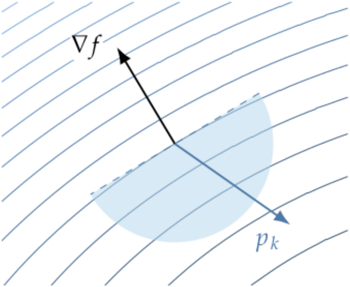
Step Size

Assume

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

Where p_k is a descent direction

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



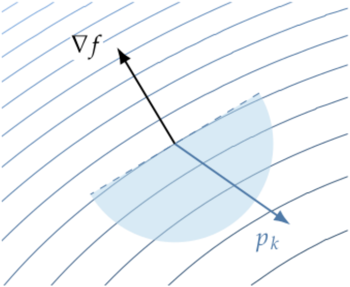
Step Size

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

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

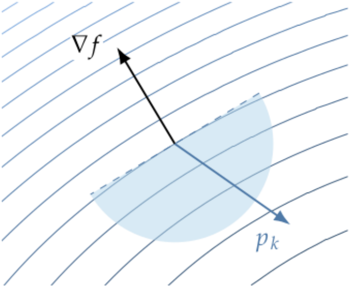
Step Size

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

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

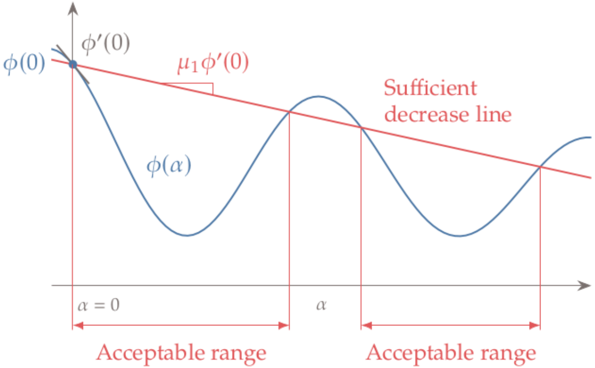
Note that $\phi'(0)$ must be negative as p_k is a descent direction.

Armijo Condition

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

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

where μ_1 is a constant such that $0 < \mu_1 \leq 1$



In practice, μ_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: α^* satisfying sufficient decrease condition

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

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

The α_{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 ρ, 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 ρ, 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:

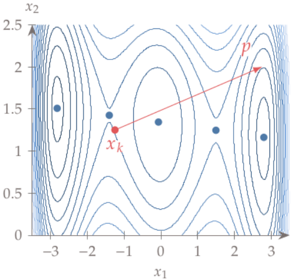
- The guess for the initial step is far too large, and the step sizes that satisfy sufficient decrease are smaller than the starting step by several orders of magnitude. Depending on the value of ρ, this scenario requires many backtracking evaluations.
- The guess for the initial step immediately satisfies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

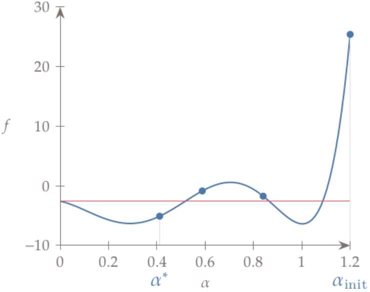
Even if our original step size is not too far from an acceptable one, the basic backtracking algorithm ignores any information we have about the function values and gradients. It blindly takes a reduced step based on a preselected ratio ρ.

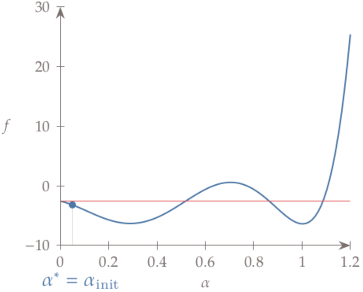
Backtracking Example

$$f(x_1, x_2) =$$
$$0.1x_1^6 - 1.5x_1^4 + 5x_1^2$$
$$+ 0.1x_2^4 + 3x_2^2 - 9x_2 + 0.5x_1x_2$$

$\mu_1 = 10^{-4}$ 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.

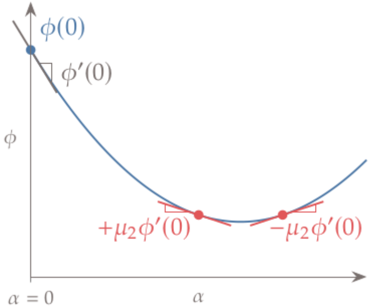
Sufficient Curvature Condition

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*

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

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



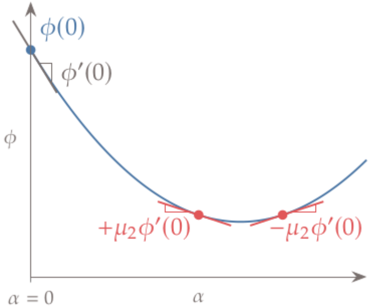
Sufficient Curvature Condition

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*

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

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



Typical values of μ_2 range from 0.1 to 0.9, depending on the direction setting method.

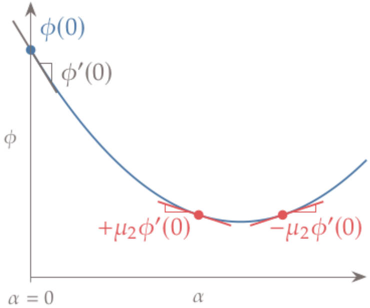
Sufficient Curvature Condition

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*

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

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



Typical values of μ_2 range from 0.1 to 0.9, depending on the direction setting method.

As μ_2 tends to 0, the condition enforces $\phi'(\alpha) = 0$, which would yield an 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

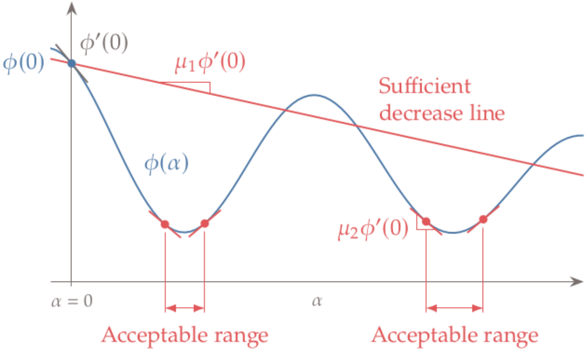
Putting together Armijo and sufficient curvature conditions, we obtain *strong Wolfe conditions*

- *Sufficient decrease condition*

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

- *Sufficient curvature condition*

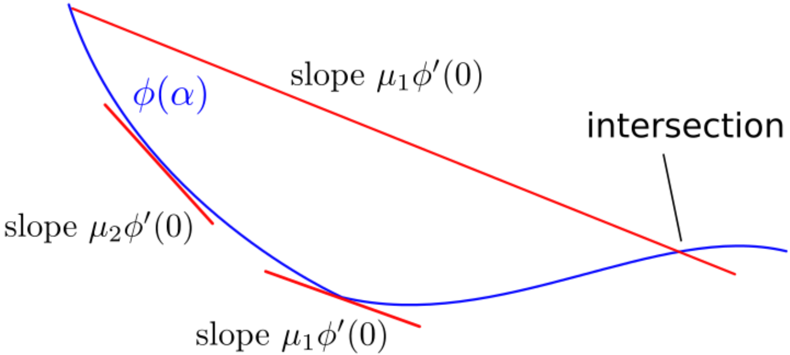
$$\left|\phi'(\alpha)\right| \leq \mu_2 \left|\phi'(0)\right|$$



Satisfiability of Strong Wolfe Conditions

Theorem 9

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ 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 θ_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 θ_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

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

Convergence of Line Search

Denote by θ_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

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

Theorem 10 (Zoutendijk)

Consider $x_{k+1} = x_k + \alpha_k p_k$, *where* p_k *is a* descent direction *and* α_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 2 Bracketing

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

Set $\alpha_0 \leftarrow 0$

$i \leftarrow 1$

repeat

  Evaluate $\phi(\alpha_i)$

  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**

    $\alpha^* \leftarrow$ **zoom**(α_{i-1}, α_i) and stop

  end if

  Evaluate $\phi'(\alpha_i)$

  if $|\phi'(\alpha_i)| \leq \mu_2 |\phi'(0)|$ **then**

    set $\alpha^* \leftarrow \alpha_i$ and stop

  else if $\phi'(\alpha_i) \geq 0$ **then**

    set $\alpha^* \leftarrow$ **zoom**(α_i, α_{i-1}) and stop

  end if

  Choose $\alpha_{i+1} \in (\alpha_i, \alpha_{\max})$

  $i \leftarrow i + 1$

until a condition is met

Explanation of Bracketing

Note that the sequence of trial steps α_i is monotonically increasing.

Explanation of Bracketing

Note that the sequence of trial steps α_i is monotonically increasing.

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

- α_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 α_i. May use, e.g., a constant multiple.

Zoom

The following algorithm keeps two step lengths: α_{lo} and α_{hi}

Zoom

The following algorithm keeps two step lengths: α_{lo} and α_{hi}

The following invariants are being preserved:

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

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

Zoom

The following algorithm keeps two step lengths: α_{lo} and α_{hi}

The following invariants are being preserved:

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

  Note that we *do not* assume $\alpha_{\mathsf{lo}} \leq \alpha_{\mathsf{hi}}$
- α_{lo} is, among all step lengths generated so far and satisfying the sufficient decrease condition, the one giving the smallest value of ϕ,

Zoom

The following algorithm keeps two step lengths: α_{lo} and α_{hi}

The following invariants are being preserved:

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

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

- α_{lo} is, among all step lengths generated so far and satisfying the sufficient decrease condition, the one giving the smallest value of ϕ,

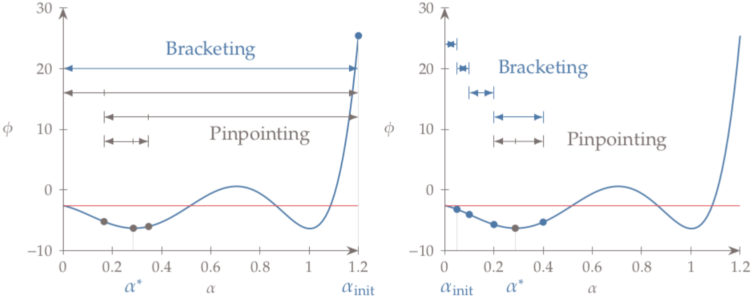
- α_{hi} is chosen so that $\phi'(\alpha_{\mathsf{lo}})(\alpha_{\mathsf{hi}} - \alpha_{\mathsf{lo}}) < 0$.

  That is, ϕ always slopes down from α_{lo} to α_{hi}.

function ZOOM($\alpha_{\text{lo}}, \alpha_{\text{hi}}$)
  repeat
    Set α between α_{lo} and α_{hi} using interpolation
    (bisection, quadratic, etc.)
    Evaluate $\phi(\alpha)$
    if $\phi(\alpha) > \phi(0) + \alpha\mu_1\phi'(0)$ or $\phi(\alpha) \geq \phi(\alpha_{\text{lo}})$ **then**
      $\alpha_{\text{hi}} \leftarrow \alpha$
    else
      Evaluate $\phi'(\alpha)$
      if $|\phi'(\alpha)| \leq \mu_2|\phi'(0)|$ **then**
        Set $\alpha^* \leftarrow \alpha$ and stop
      end if
      if $\phi'(\alpha)(\alpha_{\text{hi}} - \alpha_{\text{lo}}) \geq 0$ **then**
        $\alpha_{\text{hi}} \leftarrow \alpha_{\text{lo}}$
      end if
      $\alpha_{\text{lo}} \leftarrow \alpha$
    end if
  until a condition is met
end function

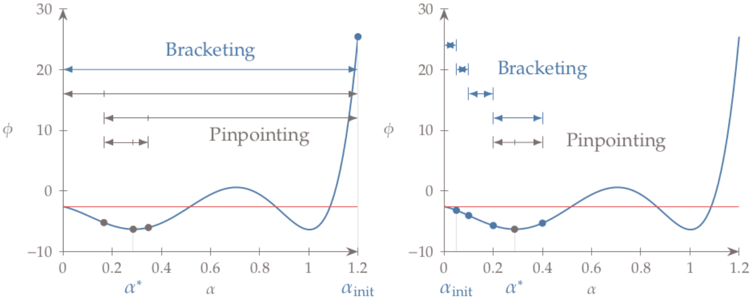
Bracketing & Zooming Example

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

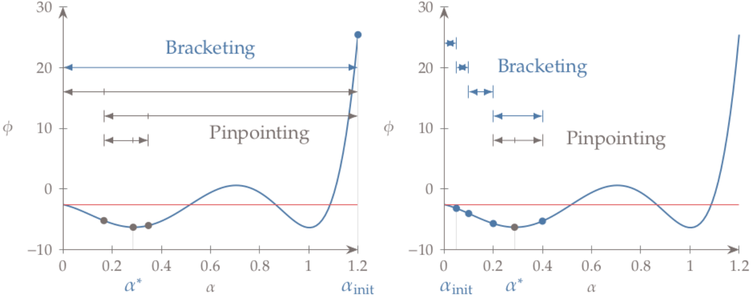
We use quadratic interpolation; the bracketing chooses $\alpha_{i+1} = 2\alpha_i$, and the sufficient curvature factor is $\mu_2 = 0.9$.



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 α 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 α 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 α 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 α 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 α 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`

Unconstrained Optimization Algorithms

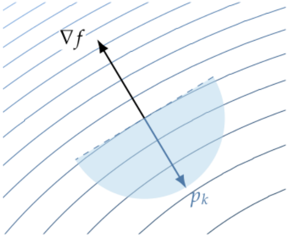
Descent Direction

First-Order Methods

Gradient Descent

Consider the *gradient descent* (aka *gradient descent*) method where

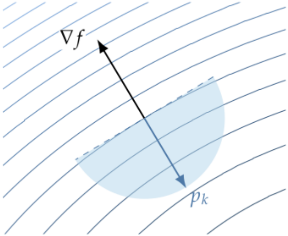
$$x_{k+1} = x_k + \alpha_k p_k \qquad p_k = -\nabla f(x_k)$$



Gradient Descent

Consider the *gradient descent* (aka *gradient descent*) method where

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



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:

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

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

Gradient Descent Algorithm with Line Search

Algorithm 3 Gradient Descent with Line Search

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

Output: x^* approximation to a stationary point

$k \leftarrow 0$
while $\|\nabla f\|_\infty > \varepsilon$ **do**
    $p_k \leftarrow -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}$
    Set α_{init} for line search
    $\alpha_k \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$
    $x_{k+1} \leftarrow x_k + \alpha_k p_k$
    $k \leftarrow k + 1$
end while

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

```
k ← 0
while ‖∇f‖_∞ > ε do
    p_k ← −∇f(x_k)/‖∇f(x_k)‖
    Set α_init for line search
    α_k ← linesearch(p_k, α_init)
    x_{k+1} ← x_k + α_k p_k
    k ← k + 1
end while
```

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

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

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

```
k ← 0
while ‖∇f‖_∞ > ε do
    p_k ← −∇f(x_k) / ‖∇f(x_k)‖
    Set α_init for line search
    α_k ← linesearch(p_k, α_init)
    x_{k+1} ← x_k + α_k p_k
    k ← k + 1
end while
```

Gradient Descent Algorithm with Line Search

Algorithm 6 Gradient Descent with Line Search

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

Output: x^* approximation to a stationary point

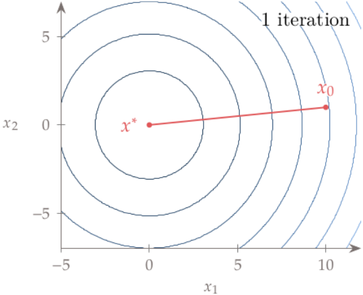
1: $k \leftarrow 0$
2: **while** $\|\nabla f\|_\infty > \varepsilon$ **do**
3: $\quad p_k \leftarrow -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}$
4: $\quad$ Set α_{init} for line search
5: $\quad \alpha_k \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$
6: $\quad x_{k+1} \leftarrow x_k + \alpha_k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

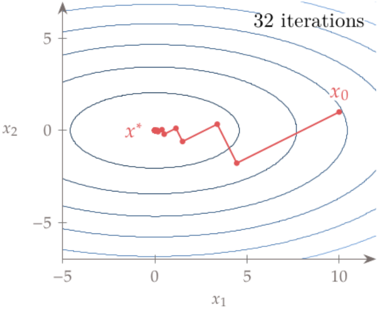
Here α_{init} can be estimated from the previous step size α_{k-1} by demanding similar decrease in the objective:

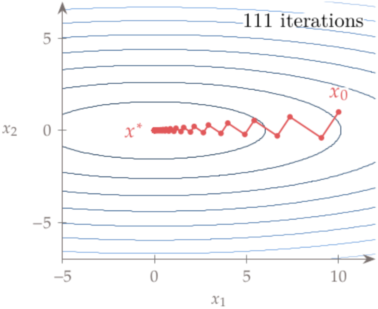
$$\alpha_{init} p_k^\top \nabla f_k^\top \approx \alpha_{k-1} p_{k-1}^\top \nabla f_{k-1}^\top \quad \Rightarrow \quad \alpha_{init} = \alpha_{k-1} \frac{\alpha_{k-1} p_{k-1}^\top \nabla f_{k-1}^\top}{\nabla p_k^\top f_k^\top}$$

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

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



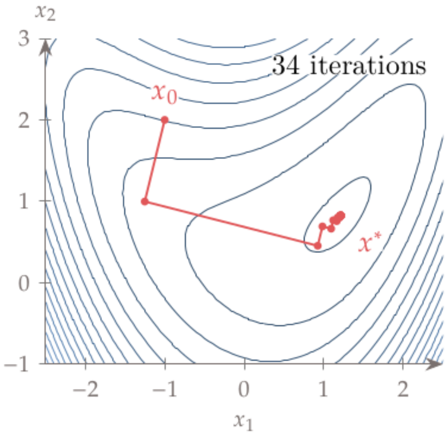




Note that p_{k+1} and p_k are always orthogonal.

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

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



The gradient descent can be prolonged.

Global Convergence with Line Search

Recall the Zoutendijk's theorem.

Denote by θ_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 on a set $\mathcal{N}$ for some $L > 0$ if

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \qquad \text{for all } x, \tilde{x} \in \mathcal{N}$$

Theorem 11 (Zoutendijk)

Consider $x_{k+1} = x_k + \alpha_k p_k$, where p_k is a descent direction and α_k satisfies the strong Wolfe conditions. Suppose that f is bounded below in $\mathbb{R}^n$ and that f is continuously differentiable in an open set $\mathcal{N}$ containing the level set $\{x : f(x) \leq f(x_0)\}$. Assume also that f is L-smooth on $\mathcal{N}$. Then

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

Global Convergence of Gradient Descent

Assume that each α_k satisfies strong Wolfe conditions.

Global Convergence of Gradient Descent

Assume that each α_k satisfies strong Wolfe conditions.

Note that the angle θ_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 α_k satisfies strong Wolfe conditions.

Note that the angle θ_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 $\lim_{k\to\infty} ||\nabla f_k|| = 0$.

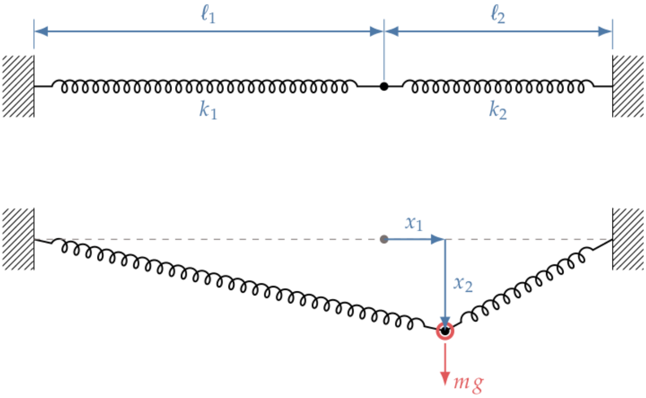
Local Linear Convergence of Gradient Descent

Theorem 12

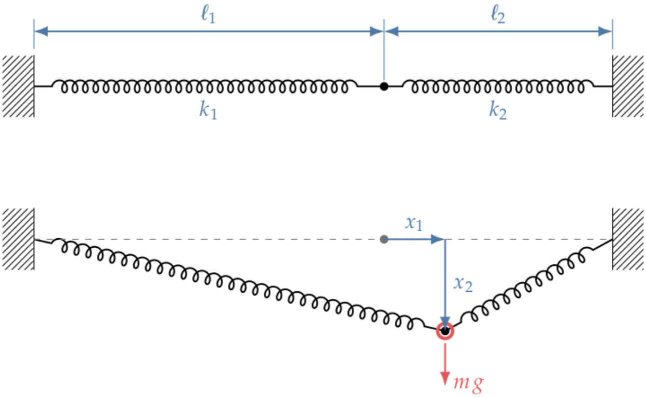
Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable, that the line search is exact, and that the descent converges to x^* where $\nabla f(x^*) = 0$ and the Hessian matrix $\nabla^2 f(x^*)$ is positive definite. Then

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

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



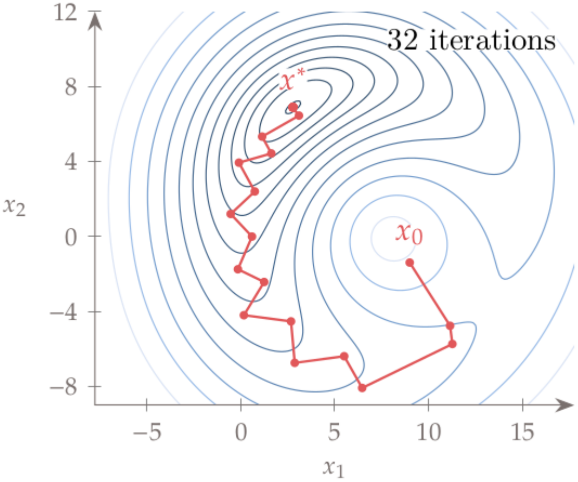
ℓ_1
ℓ_2
k_1
k_2
x_1
x_2
mg



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

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

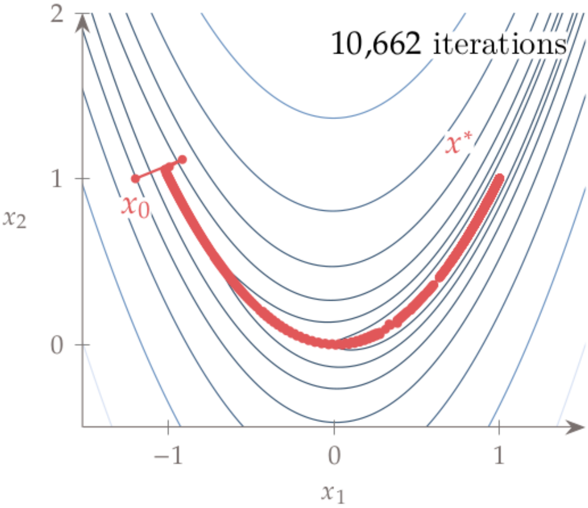
Two Spring Problem - Gradient Descent



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

Rosenbrock Function - Gradient Descent

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



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

Comments on Gradient Descent

- The method needs evaluation of ∇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 ∇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 ∇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 ∇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 $f : \mathbb{R}^n \to \mathbb{R}$.

Assume that f is twice differentiable.

Newton's Method

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

Assume that f is twice differentiable.

Then, by the Taylor's theorem,

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

where we denote the Hessian $\nabla^2 f(x_k)$ by H_k.

Newton's Method

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

Assume that f is twice differentiable.

Then, by the Taylor's theorem,

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

where we denote the Hessian $\nabla^2 f(x_k)$ by H_k.

Define

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

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

Newton's Method

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

Assume that f is twice differentiable.

Then, by the Taylor's theorem,

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

where we denote the Hessian $\nabla^2 f(x_k)$ by H_k.

Define

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

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

$$H_k s = -\nabla f_k$$

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

Newton's Method

Algorithm 7 Newton's Method

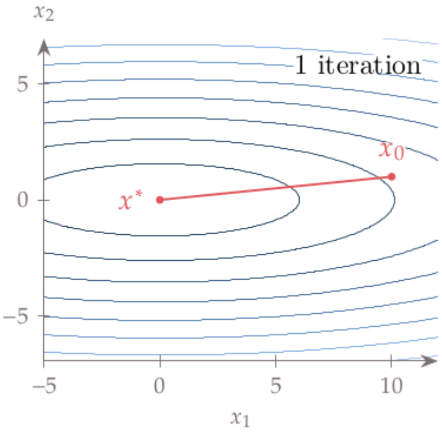
Input: x_0 starting point, $\varepsilon > 0$

Output: x^* approximation to a stationary point

1: $k \leftarrow 0$
2: **while** $\|\nabla f_k\|_\infty > \varepsilon$ **do**
3: 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

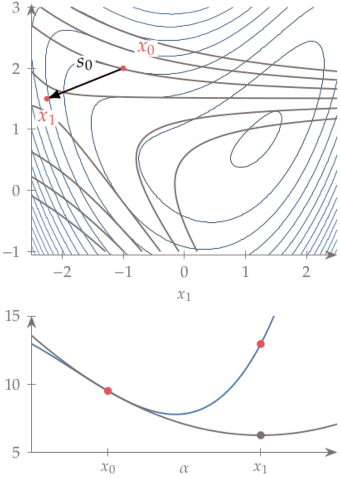
Newton's method finds the minimum of a quadratic function in a single step.



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

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

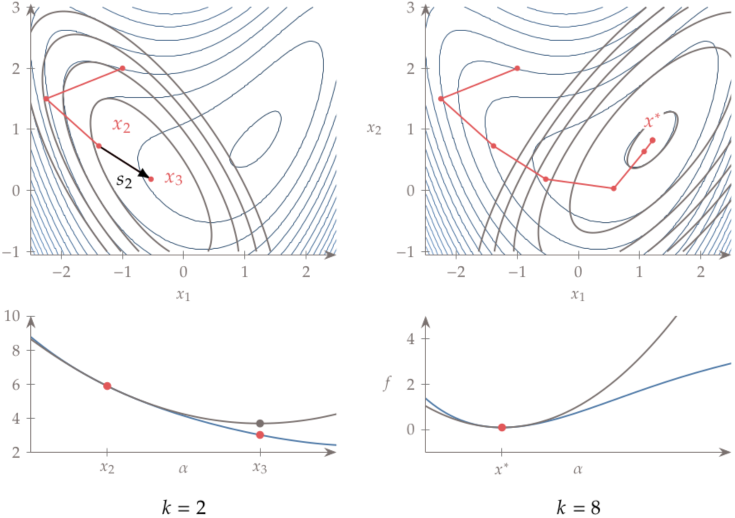
Stopping: $||\nabla f||_\infty \leq 10^{-6}$.



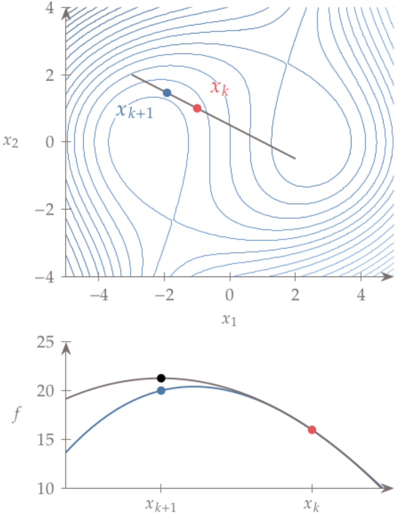
$k = 0$ $k = 2$

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

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



Convergence Issues



Negative curvature

Overshoot

Also, the computation of the Hessian is costly.

Local Quadratic Convergence of Newton's Method

Theorem 13

Assume f is defined and twice differentiable and assume that ∇f is L-smooth on $\mathcal{N}$.
Let x_* be a minimizer of $f(x)$ in $\mathcal{N}$ and assume that $\nabla^2 f(x_*)$ is positive definite.
If $\|x_0 - x_*\|$ is sufficiently small, then $\{x_k\}$ converges quadratically to x_*.

Local Quadratic Convergence of Newton's Method

Theorem 13

Assume f is defined and twice differentiable and assume that ∇f is L-smooth on $\mathcal{N}$.
Let x_* be a minimizer of $f(x)$ in $\mathcal{N}$ 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.

Local Quadratic Convergence of Newton's Method

Theorem 13

Assume f is defined and twice differentiable and assume that ∇f is L-smooth on $\mathcal{N}$.
Let x_* be a minimizer of $f(x)$ in $\mathcal{N}$ 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.

Local Quadratic Convergence of Newton's Method

Theorem 13

Assume f is defined and twice differentiable and assume that ∇f is L-smooth on $\mathcal{N}$.
Let x_* be a minimizer of $f(x)$ in $\mathcal{N}$ 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?

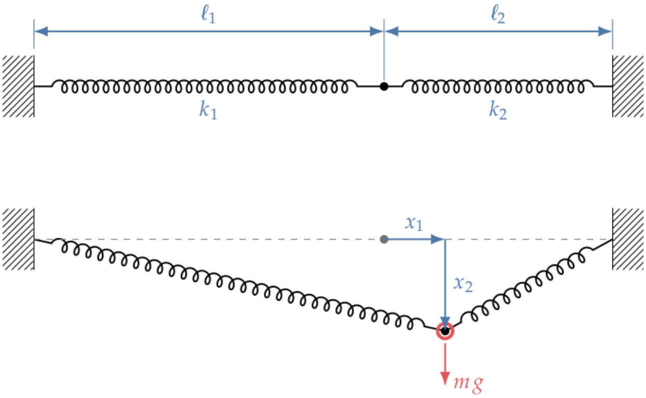
Newton's Method with Line Search

Algorithm 8 Newton's Method with Line Search

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

Output: x^* approximation to a stationary point

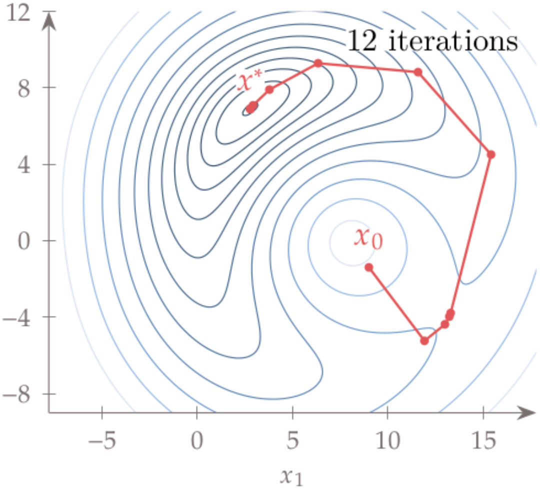
```
k ← 0
α_init ← 1
while ‖∇f_k‖_∞ > ε do
    Compute ∇f_k = ∇f(x_k)
    Solve H_k p_k = −∇f_k for p_k
    α_k ← linesearch(p_k, α_init)
    x_{k+1} ← x_k + α_k p_k
    k ← k + 1
end while
```



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

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

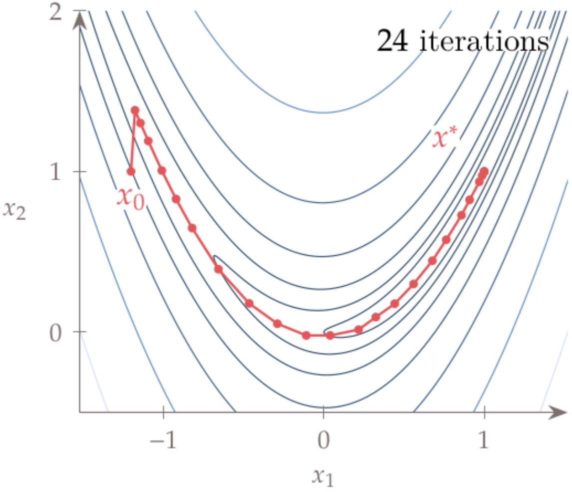
Two Spring Problem - Newton's Method



Gradient descent, line search, stop. cond. $||\nabla f||_\infty \leq 10^{-6}$.
Compare this with 32 iterations of gradient descent.

Rosenbrock Function - Newton's Method

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



Gradient descent, 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 θ_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 α_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 α_k satisfy strong Wolfe conditions.

Global Convergence of Newton's Method

Assume that all α_k satisfy strong Wolfe conditions.

Assume that the Hessians H_k are positive definite with a uniformly bounded condition number:

$$||H_k|| \, \left|\left|H_k^{-1}\right|\right| \leq M \quad \text{for all } k$$

Global Convergence of Newton's Method

Assume that all α_k satisfy strong Wolfe conditions.

Assume that the Hessians H_k are positive definite with a uniformly bounded condition number:

$$||H_k|| \, \left|\left|H_k^{-1}\right|\right| \leq M \quad \text{for all } k$$

Then θ_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 α_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 θ_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 $\lim_{k\to\infty} ||\nabla f_k|| = 0$.

Global Convergence of Newton's Method

Assume that all α_k satisfy strong Wolfe conditions.

Assume that the Hessians H_k are positive definite with a uniformly bounded condition number:

$$||H_k|| \, \left|\left|H_k^{-1}\right|\right| \leq M \quad \text{for all } k$$

Then θ_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 $\lim_{k \to \infty} ||\nabla f_k|| = 0$.

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., $D = \text{diag}(\lambda_1, \ldots, \lambda_n)$ 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., $D = \text{diag}(\lambda_1, \ldots, \lambda_n)$ 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 $\max\{\lambda_1, \ldots, \lambda_n\}$ going to infinity.
- $\left|\left|H_k^{-1}\right|\right|$ grows with $\min\{\lambda_1, \ldots, \lambda_n\}$ 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., $D = \text{diag}(\lambda_1, \ldots, \lambda_n)$ 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 $\max\{\lambda_1, \ldots, \lambda_n\}$ going to infinity.
- $\left|\left|H_k^{-1}\right|\right|$ grows with $\min\{\lambda_1, \ldots, \lambda_n\}$ 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 δ?
- How to modify H_k so the minimum is large enough?

Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \qquad \text{and} \qquad \nabla^2 f(x_k) = \text{diag}(10, 3, -1)$$

Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \qquad \text{and} \qquad \nabla^2 f(x_k) = \text{diag}(10, 3, -1)$$

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:

$$\nabla f(x_k) = (1, -3, 2) \qquad \text{and} \qquad \nabla^2 f(x_k) = \mathrm{diag}(10, 3, -1)$$

Now, the diagonalization is trivial:

$$\nabla^2 f(x_k) = Q\,\mathrm{diag}(10, 3, -1)\, Q^\top \quad Q = I \text{ is the identity matrix}$$

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:

$$\nabla f(x_k) = (1, -3, 2) \qquad \text{and} \qquad \nabla^2 f(x_k) = \mathrm{diag}(10, 3, -1)$$

Now, the diagonalization is trivial:

$$\nabla^2 f(x_k) = Q\,\mathrm{diag}(10, 3, -1)\ Q^\top \quad Q = I \text{ is the identity matrix}$$

What if we consider a minimum modification replacing the negative eigenvalue with a small number, say $\delta = 10^{-8}$? Obtain

$$B_k = Q\ \mathrm{diag}(10, 3, 10^{-8})\ Q^\top = \mathrm{diag}(10, 3, 10^{-8})$$

Sufficiently Large Eigenvalues

Consider an example:

$$\nabla f(x_k) = (1, -3, 2) \qquad \text{and} \qquad \nabla^2 f(x_k) = \text{diag}(10, 3, -1)$$

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:

$$\nabla f(x_k) = (1, -3, 2) \qquad \text{and} \qquad \nabla^2 f(x_k) = \text{diag}(10, 3, -1)$$

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.

Even though f decreases along p_k, it is far from the minimum of the quadratic approximation of f.

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 ΔH_k.

What is ΔH_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 ΔH_k.

What is ΔH_k in our example?

Various methods for computing ΔH_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 9 Newton's Method with Line Search

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

Output: x^* approximation to a stationary point

$k \leftarrow 0$
while $\|\nabla f_k\|_\infty > \varepsilon$ **do**
  $H_k \leftarrow \nabla^2 f(x_k)$
  if H_k is **not** sufficiently positive definite **then**
    $H_k \leftarrow H_k + \Delta H_k$ so that H_k is sufficiently pos. definite
  end if
  Compute $\nabla f_k = \nabla f(x_k)$
  Solve $H_k p_k = -\nabla f_k$ for p_k
  Set $x_{k+1} = x_k + \alpha_k p_k$, here α_k sat. the Wolfe cond.
  $k \leftarrow k + 1$
end while

Convergence of Modified Newton's Method

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 in 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 in 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 in 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

$$\tilde{H}_{k+1}(x_{k+1} - x_k) = \nabla f_{k+1} - \nabla f_k$$

This is the *secant condition*.

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

$$s_k = x_{k+1} - x_k \qquad y_k = \nabla f_{k+1} - \nabla f_k$$

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

$$s_k = x_{k+1} - x_k \qquad y_k = \nabla f_{k+1} - \nabla f_k$$

So that the secant condition becomes

$$\tilde{H}_{k+1} s_k = y_k$$

Does it have a symmetric positive definite solution?

Curvature Condition

Consider the secant condition:

$$\tilde{H}_{k+1} s_k = y_k$$

Curvature Condition

Consider the secant condition:

$$\tilde{H}_{k+1} s_k = y_k$$

The following is true:

- The secant condition has a symmetric positive definite solution iff the following condition is satisfied:

$$s_k^\top y_k > 0$$

Curvature Condition

Consider the secant condition:

$$\tilde{H}_{k+1} s_k = y_k$$

The following is true:

- The secant condition has a symmetric positive definite solution iff the following condition is satisfied:

  $$s_k^\top y_k > 0$$

- The condition $s_k^\top y_k > 0$ is satisfied if the line search satisfies the strong Wolfe conditions.

Curvature Condition

Consider the secant condition:

$$\tilde{H}_{k+1} s_k = y_k$$

The following is true:

- The secant condition has a symmetric positive definite solution iff the following condition is satisfied:

  $$s_k^\top y_k > 0$$

- The condition $s_k^\top y_k > 0$ is satisfied if the line search satisfies the strong Wolfe conditions.

As a corollary, we obtain the following:

Theorem 15

Assume that we use line search satisfying strong Wolfe conditions. Then in every step, the secant condition

$$\tilde{H}_{k+1} s_k = y_k$$

has a symmetric positive definite solution $\tilde{H}_{k+1}$.

Now, we can obtain an approximate Hessian $\tilde{H}_{k+1}$ by solving the secant condition $\tilde{H}_{k+1} s_k = y_k$.

Now, we can obtain an approximate Hessian $\tilde{H}_{k+1}$ by solving the secant condition $\tilde{H}_{k+1} s_k = y_k$.

Note that even if we demand symmetric positive definite solutions to the secant condition, there are infinitely many.

Indeed, there are $n(n+1)/2$ degrees of freedom in a symmetric matrix, and the secant conditions represent only n conditions.

Now, we can obtain an approximate Hessian $\tilde{H}_{k+1}$ by solving the secant condition $\tilde{H}_{k+1} s_k = y_k$.

Note that even if we demand symmetric positive definite solutions to the secant condition, there are infinitely many.

Indeed, there are $n(n+1)/2$ degrees of freedom in a symmetric matrix, and the secant conditions represent only n conditions.

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.

Now, we can obtain an approximate Hessian $\tilde{H}_{k+1}$ by solving the secant condition $\tilde{H}_{k+1}s_k = y_k$.

Note that even if we demand symmetric positive definite solutions to the secant condition, there are infinitely many.

Indeed, there are $n(n+1)/2$ degrees of freedom in a symmetric matrix, and the secant conditions represent only n conditions.

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.

We also want $\tilde{H}_{k+1}$ to be symmetric positive definite.

Now, we can obtain an approximate Hessian $\tilde{H}_{k+1}$ by solving the secant condition $\tilde{H}_{k+1}s_k = y_k$.

Note that even if we demand symmetric positive definite solutions to the secant condition, there are infinitely many.

Indeed, there are $n(n+1)/2$ degrees of freedom in a symmetric matrix, and the secant conditions represent only n conditions.

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.

We also want $\tilde{H}_{k+1}$ to be symmetric positive definite.

We strive to choose $\tilde{H}_{k+1}$ "close" to $\tilde{H}_k$.

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.

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 $u = \left(y_k - \tilde{H}_k s_k\right)$

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{uu^\top}{u^\top s_k}$$

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 $u = \left(y_k - \tilde{H}_k s_k\right)$

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{uu^\top}{u^\top s_k}$$

Now, the secant condition is satisfied:

$$\tilde{H}_{k+1} s_k = \tilde{H}_k s_k + \frac{uu^\top s_k}{u^\top s_k} = \tilde{H}_k s_k + u = \tilde{H}_k s_k + \left(y_k - \tilde{H}_k s_k\right) = y_k$$

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 $\tilde{H}_{k+1}$ 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 $u = \left(y_k - \tilde{H}_k s_k\right)$

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{uu^\top}{u^\top s_k}$$

Now, the secant condition is satisfied:

$$\tilde{H}_{k+1} s_k = \tilde{H}_k s_k + \frac{uu^\top s_k}{u^\top s_k} = \tilde{H}_k s_k + u = \tilde{H}_k s_k + \left(y_k - \tilde{H}_k s_k\right) = y_k$$

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 $\tilde{H}_{k+1}$ 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 10 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

$$\tilde{H}_1 = I + \frac{(y_0 - Is_0)(y_0 - Is_0)^\top}{(y_0 - Is_0)^\top s_0} = \begin{pmatrix} 1.1531 & 0.3445 & 0.4593 \\ 0.3445 & 1.7751 & 1.0335 \\ 0.4593 & 1.0335 & 2.3780 \end{pmatrix}.$$

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 a 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 a 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 a 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.

Lemma 1

Assume that $\tilde{H}_k$ is symmetric positive definite.
Then $\tilde{H}_{k+1}$ is symmetric positive definite iff $y_k^\top s_k > 0$.

We know that line search satisfying the strong Wolfe conditions preserves $y_k^\top s_k > 0$.

Thus, starting with a symmetric positive definite $\tilde{H}_0$ (e.g., a scalar multiple of I), every $\tilde{H}_k$ is symmetric positive definite and satisfies the secant condition.

BFGS

Algorithm 11 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 $\tilde{H}_k p_k = -\nabla f_k$

  $\alpha \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$

  $x_{k+1} \leftarrow x_k + \alpha p_k$

  $s \leftarrow x_{k+1} - x_k$

  $y \leftarrow \nabla f_{k+1} - \nabla f_k$

  $\tilde{H}_{k+1} \leftarrow \tilde{H}_k - \frac{(\tilde{H}_k s)(\tilde{H}_k s)^\top}{s^\top \tilde{H}_k s} + \frac{yy^\top}{y^\top s}$

  $k \leftarrow k + 1$

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 $x_0 = (0, 0, 0)^T$.

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 $x_0 = (0, 0, 0)^T$.

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 $x_0 = (0, 0, 0)^T$.

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, $\left\| \nabla f\left(x_1\right) \right\|_\infty = 2.6667$, 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

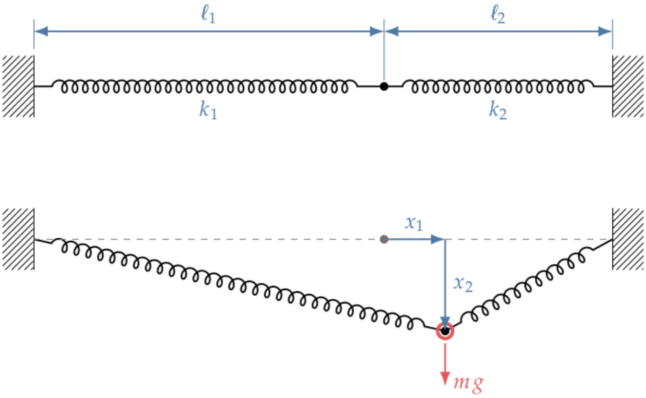
$$p_2 = \begin{pmatrix} -0.5289 \\ 0.6268 \\ -0.2644 \end{pmatrix}$$

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.

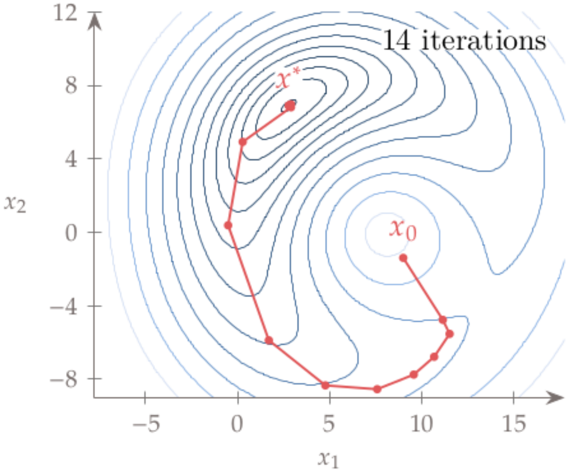
Notice that we got the same x_1, x_2, x_3 as for SR1. This follows from using the exact line search and the quadratic problem. It does not hold in general.



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

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

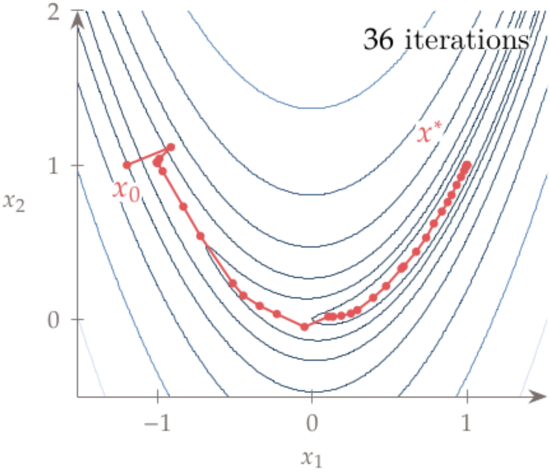
Two Spring Problem - BFGS



Gradient descent, 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: $f(x_1, x_2) = (1 - x_1)^2 + 100\left(x_2 - x_1^2\right)^2$



Gradient descent, 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.

Rank One Update V2

Algorithm 12 Rank 1 update 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 $\|\nabla f_k\|_\infty > \varepsilon$ **do**
  Compute $\nabla f_k = \nabla f(x_k)$
  $p_k \leftarrow -\tilde{H}_k^{-1} \nabla f_k$
  $\alpha \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$
  $x_{k+1} \leftarrow x_k + \alpha p_k$
  $s \leftarrow x_{k+1} - x_k$
  $y \leftarrow \nabla f_{k+1} - \nabla f_k$
  $\tilde{H}_{k+1}^{-1} \leftarrow \tilde{H}_k^{-1} + \frac{\left(s - \tilde{H}_k^{-1} y\right)\left(s - \tilde{H}_k^{-1} y\right)^\top}{\left(s - \tilde{H}_k^{-1} y\right)^\top y}$
  $k \leftarrow k + 1$
end while

BFGS

Applying SMW to the BFGS Hessian update

$$\tilde{H}_{k+1} = \tilde{H}_k - \frac{\left(\tilde{H}_k s_k\right)\left(\tilde{H}_k s_k\right)^\top}{s_k^\top \tilde{H}_k s_k} + \frac{y_k y_k^\top}{y_k^\top s_k}$$

BFGS

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.

BFGS V2

Algorithm 13 BFGS v2

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 $\nabla f_k = \nabla f(x_k)$
    $p_k \leftarrow -\tilde{H}_k^{-1} \nabla f_k$
    $\alpha \leftarrow \text{linesearch}(p_k, \alpha_{\text{init}})$
    $x_{k+1} \leftarrow x_k + \alpha p_k$
    $s \leftarrow x_{k+1} - x_k$
    $y \leftarrow \nabla f_{k+1} - \nabla f_k$
    $\tilde{H}_{k+1}^{-1} \leftarrow \left(I - \frac{sy^\top}{s^\top y}\right) \tilde{H}_k^{-1} \left(I - \frac{ys^\top}{s^\top y}\right) + \frac{ss^\top}{s^\top y}$
    $k \leftarrow k + 1$
end while

Limited Memory BFGS Idea

Let us denote by $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$ the values of the variables s and y, resp., during the iterations $1, \ldots, k$ of BFGS.

Limited Memory BFGS Idea

Let us denote by $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$ the values of the variables s and y, resp., during the iterations $1, \ldots, k$ of BFGS.

Observe that $\tilde{H}_k$ is determined completely by H_0 and the two sequences $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

Limited Memory BFGS Idea

Let us denote by $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$ the values of the variables s and y, resp., during the iterations $1, \ldots, k$ of BFGS.

Observe that $\tilde{H}_k$ is determined completely by H_0 and the two sequences $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

So, the matrix $\tilde{H}_k$ does not have to be stored if the algorithm remembers the values $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

Note that this would be more space efficient for $k < n$.

Limited Memory BFGS Idea

Let us denote by $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$ the values of the variables s and y, resp., during the iterations $1, \ldots, k$ of BFGS.

Observe that $\tilde{H}_k$ is determined completely by H_0 and the two sequences $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

So, the matrix $\tilde{H}_k$ does not have to be stored if the algorithm remembers the values $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

Note that this would be more space efficient for $k < n$.

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 $\tilde{H}_k$ when we set $\tilde{H}_{k-m-1} = I$.

Limited Memory BFGS Idea

Let us denote by $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$ the values of the variables s and y, resp., during the iterations $1, \ldots, k$ of BFGS.

Observe that $\tilde{H}_k$ is determined completely by H_0 and the two sequences $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

So, the matrix $\tilde{H}_k$ does not have to be stored if the algorithm remembers the values $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

Note that this would be more space efficient for $k < n$.

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 $\tilde{H}_k$ when we set $\tilde{H}_{k-m-1} = I$.

This is the basic idea behind limited-memory BFGS which stores only the running window $s_{k-m}, \ldots, s_k$ and $y_{k-m}, \ldots, y_k$ and computes $\tilde{H}_k^{-1} \nabla f_k$ using these values.

Limited Memory BFGS Idea

Let us denote by $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$ the values of the variables s and y, resp., during the iterations $1, \ldots, k$ of BFGS.

Observe that $\tilde{H}_k$ is determined completely by H_0 and the two sequences $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

So, the matrix $\tilde{H}_k$ does not have to be stored if the algorithm remembers the values $s_0, \ldots, s_k$ and $y_0, \ldots, y_k$.

Note that this would be more space efficient for $k < n$.

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 $\tilde{H}_k$ when we set $\tilde{H}_{k-m-1} = I$.

This is the basic idea behind limited-memory BFGS which stores only the running window $s_{k-m}, \ldots, s_k$ and $y_{k-m}, \ldots, y_k$ 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):

$$\min_{\tilde{H}} \left\| \tilde{V} - \tilde{H}_k^{-1} \right\|$$
$$\text{subject to} \quad \tilde{V} = \tilde{V}^\top, \quad \tilde{V} y_k = s_k$$

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 $W y_k = s_k$.

BFGS is obtained with $W = \bar{G}_k^{-1}$ where $\bar{G}_k$ is the average Hessian defined by $\bar{G}_k = \left[\int_0^1 \nabla^2 f\left(x_k + \tau \alpha_k p_k\right) d\tau \right]$

Solving this gives precisely the BFGS formula for $\tilde{H}_{k+1}^{-1}$.

Global Convergence of Line Search

Denote by θ_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

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

Theorem 16 (Zoutendijk)

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

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

Global Convergence of Quasi-Newton's Method

Assume that all α_k satisfy strong Wolfe conditions.

Assume that the approximations to the Hessians $\tilde{H}_k$ are positive definite with a uniformly bounded condition number:

$$\left|\left|\tilde{H}_k\right|\right| \left|\left|\tilde{H}_k^{-1}\right|\right| \leq M \quad \text{for all } k$$

Global Convergence of Quasi-Newton's Method

Assume that all α_k satisfy strong Wolfe conditions.

Assume that the approximations to the Hessians $\tilde{H}_k$ are positive definite with a uniformly bounded condition number:

$$\left|\left|\tilde{H}_k\right|\right| \left|\left|\tilde{H}_k^{-1}\right|\right| \leq M \quad \text{for all } k$$

Then θ_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 α_k satisfy strong Wolfe conditions.

Assume that the approximations to the Hessians $\tilde{H}_k$ are positive definite with a uniformly bounded condition number:

$$\left|\left|\tilde{H}_k\right|\right| \left|\left|\tilde{H}_k^{-1}\right|\right| \leq M \quad \text{for all } k$$

Then θ_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 $\lim_{k\to\infty} ||\nabla f_k|| = 0$.

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 $\mathcal{O}(mn)$.

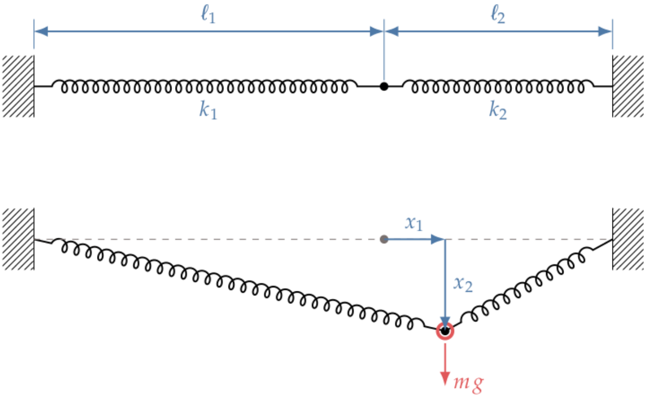
Quasi-Newton Methods - Comments

- 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

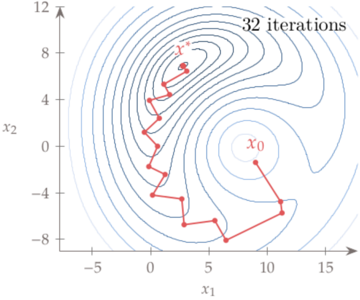
- Each iteration is performed for $\mathcal{O}(n^2)$ operations as opposed to $\mathcal{O}(n^3)$ for methods involving solutions of linear systems.
- There is even a memory-limited variant (L-BFGS) that uses only information from past m steps, and its single iteration complexity is $\mathcal{O}(mn)$.
- 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.

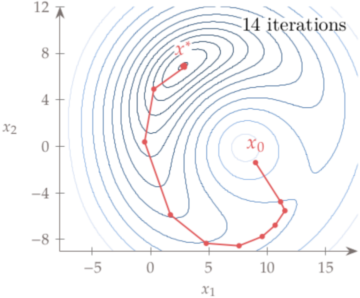


$$f(x_1, x_2) = \frac{1}{2} k_1 \left(\sqrt{(\ell_1 + x_1)^2 + x_2^2} - \ell_1 \right)^2 + \frac{1}{2} k_2 \left(\sqrt{(\ell_2 - x_1)^2 + x_2^2} - \ell_2 \right)^2 - mg x_2$$

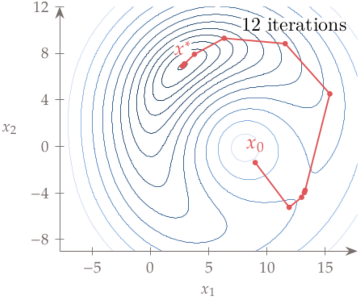
Here $\ell_1 = 12, \ell_2 = 8, k_1 = 1, k_2 = 10, mg = 7$



Steepest descent

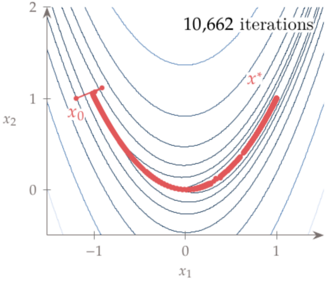


Quasi-Newton

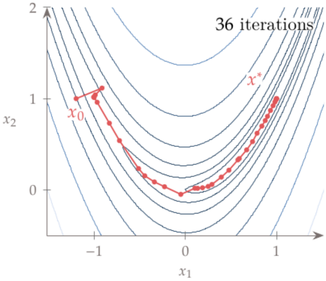


Newton

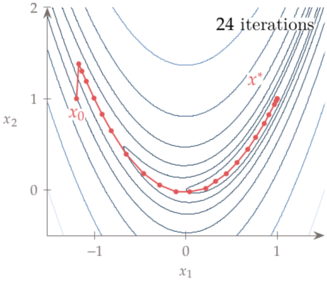
Rosenbrock: $f(x_1, x_2) = (1 - x_1)^2 + 100\left(x_2 - x_1^2\right)^2$



Steepest descent



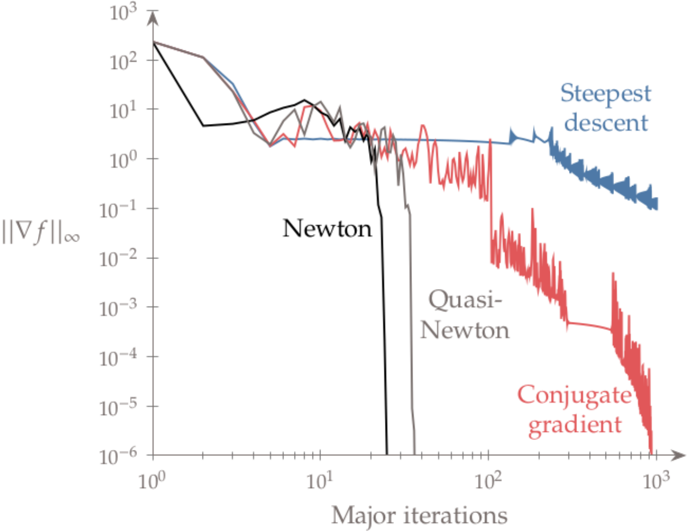
Quasi-Newton



Newton

Rosenbrock:

$$f(x_1, x_2) = (1 - x_1)^2 + 100\left(x_2 - x_1^2\right)^2$$



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.

Constrained Optimization

Constrained Optimization Problem

Recall that the constrained optimization problem is

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

x^* is now a *constrained minimizer* if

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

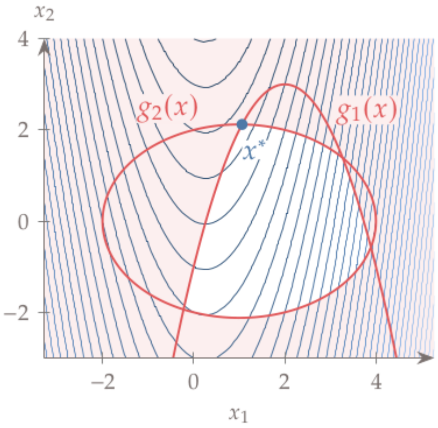
where $\mathcal{F}$ is the feasibility region

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

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

$$\begin{aligned} \underset{x_1, x_2}{\text{minimize}} \quad & f(x_1, x_2) = x_1^2 - \tfrac{1}{2}x_1 - x_2 - 2 \\ \text{subject to} \quad & g_1(x_1, x_2) = x_1^2 - 4x_1 + x_2 + 1 \leq 0 \\ & g_2(x_1, x_2) = \tfrac{1}{2}x_1^2 + x_2^2 - x_1 - 4 \leq 0 \end{aligned}$$



Equality Constraints

Let us restrict our problem only to the equality constraints:

$$\begin{array}{rl} \text{minimize} & f(x) \\ \text{by varying} & x \\ \text{subject to} & h_j(x) = 0 \quad j = 1, \ldots, n_h \end{array}$$

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 gradient 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

$$f(x^* + p) \geq f(x^*)$$

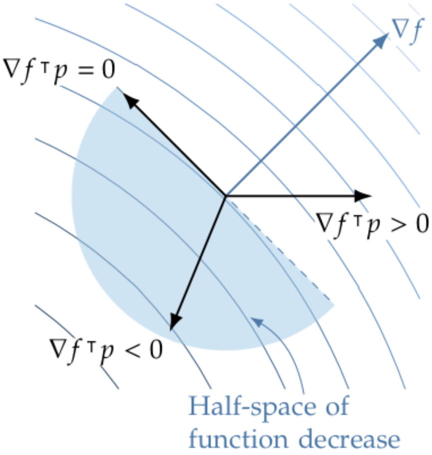
for all p small enough.

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*!

I.e., point 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*!

I.e., point 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*!

I.e., point 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*!

I.e., point 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*!

I.e., point into the feasible region. How do we characterize feasible directions?

Consider Taylor approximation of h_j for all j:

$$h_j(x + p) \approx h_j(x) + \nabla h_j(x)^\top p$$

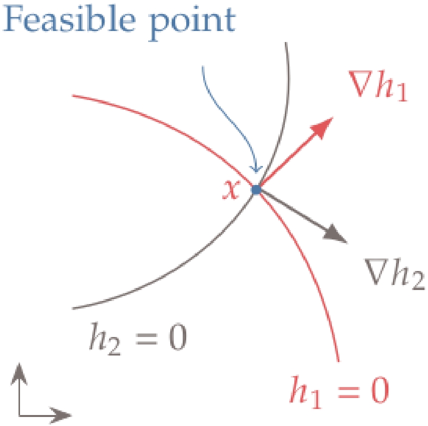
Assuming $x \in \mathcal{F}$, we have $h_j(x) = 0$ for all j and thus

$$h_j(x + p) \approx \nabla h_j(x)^\top p$$

As p is a feasible direction iff $h_j(x + p) = 0$, we obtain that

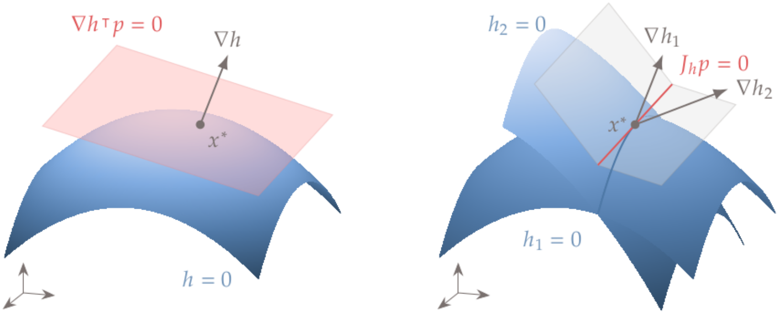
$$p \text{ is a } \textit{feasible direction} \quad \text{iff} \quad \nabla h_j(x)^\top p = 0 \text{ for all } j$$

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.,

$$\nabla h_1(x^*)p = 0 \qquad \nabla h_2(x^*)p = 0$$

Necessary Condition for Constrained Minima

Consider a direction p. Observe that

- If $h_j(x)^\top p \neq 0$, 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 $h_j(x)^\top p \neq 0$, then moving a short step in the direction p violates the constraint $h_j(x) = 0$.
- If $h_j(x)^\top p = 0$ for all j and
  - $\nabla f(x) p > 0$, 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 $h_j(x)^\top p \neq 0$, then moving a short step in the direction p violates the constraint $h_j(x) = 0$.
- If $h_j(x)^\top p = 0$ for all j and
  - $\nabla f(x) p > 0$, then moving a short step in the direction p increases f and stays in $\mathcal{F}$.
  - $\nabla f(x) p < 0$, 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 $h_j(x)^\top p \neq 0$, then moving a short step in the direction p violates the constraint $h_j(x) = 0$.
- If $h_j(x)^\top p = 0$ for all j and
  - $\nabla f(x) p > 0$, then moving a short step in the direction p increases f and stays in $\mathcal{F}$.
  - $\nabla f(x) p < 0$, then moving a short step in the direction p decreases f and stays in $\mathcal{F}$.
  - $\nabla f(x) p = 0$, 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 $h_j(x)^\top p \neq 0$, then moving a short step in the direction p violates the constraint $h_j(x) = 0$.
- If $h_j(x)^\top p = 0$ for all j and
  - $\nabla f(x) p > 0$, then moving a short step in the direction p increases f and stays in $\mathcal{F}$.
  - $\nabla f(x) p < 0$, then moving a short step in the direction p decreases f and stays in $\mathcal{F}$.
  - $\nabla f(x) p = 0$, 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 $h_j(x^*)^\top p = 0$ 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 $h_j(x)^\top p \neq 0$, then moving a short step in the direction p violates the constraint $h_j(x) = 0$.
- If $h_j(x)^\top p = 0$ for all j and
  - $\nabla f(x)p > 0$, then moving a short step in the direction p increases f and stays in $\mathcal{F}$.
  - $\nabla f(x)p < 0$, then moving a short step in the direction p decreases f and stays in $\mathcal{F}$.
  - $\nabla f(x)p = 0$, 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 $h_j(x^*)^\top p = 0$ 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 $h_j(x)^\top p \neq 0$, then moving a short step in the direction p violates the constraint $h_j(x) = 0$.
- If $h_j(x)^\top p = 0$ for all j and
  - $\nabla f(x)p > 0$, then moving a short step in the direction p increases f and stays in $\mathcal{F}$.
  - $\nabla f(x)p < 0$, then moving a short step in the direction p decreases f and stays in $\mathcal{F}$.
  - $\nabla f(x)p = 0$, then moving a short step in the direction p does not change f and stays $\mathcal{F}$.

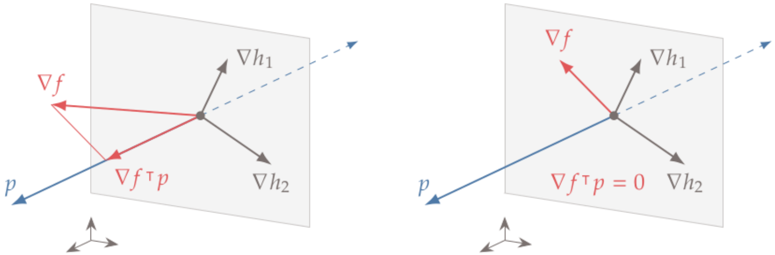
To be a minimizer, x^* must be feasible and every direction satisfying $h_j(x^*)^\top p = 0$ for all j must also satisfy $\nabla f(x^*)^\top p \geq 0$.

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

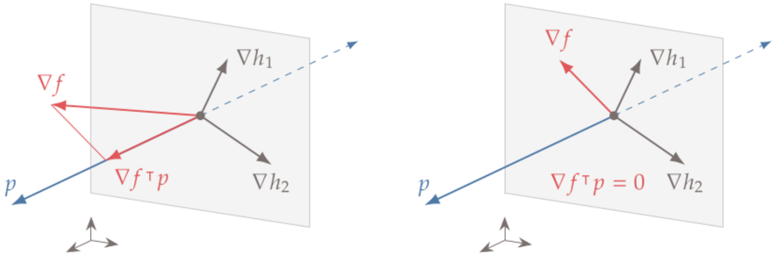
$$\nabla f(x^*)^\top p = 0 \text{ for all } p \text{ satisfying } (\forall j : \nabla h_j(x^*)^\top p = 0)$$

Lagrange Multipliers



Left: f increases along p. **Right:** f does not change along p.

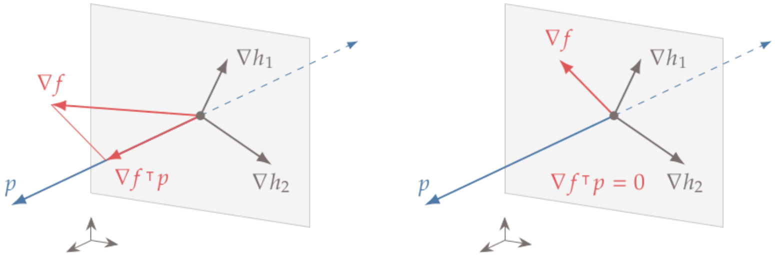
Lagrange Multipliers



Left: f increases along p. **Right:** f does not change along p.

Observe that at an optimum, ∇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, ∇f lies in the space spanned by the gradients of constraint functions.

There are *Lagrange multipliers* λ_1, λ_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 17

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 $\lambda_1, \ldots, \lambda_{n_h} \in \mathbb{R}$ satisfying

$$\nabla f(x^*) = -\sum_{j=1}^{n_h} \lambda_j \nabla h_j(x^*)$$

The coefficients $\lambda_1, \ldots, \lambda_{n_h}$ 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* $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \to \mathbb{R}$ defined by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \qquad \text{here} \qquad h(x) = (h_1(x), \ldots, h_{n_h}(x))^\top$$

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* $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \to \mathbb{R}$ defined by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \qquad \text{here} \qquad h(x) = (h_1(x), \ldots, h_{n_h}(x))^\top$$

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* $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \to \mathbb{R}$ defined by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \qquad \text{here} \qquad h(x) = (h_1(x), \ldots, h_{n_h}(x))^\top$$

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:

$$h(x) = 0 \qquad \text{and} \qquad \nabla f(x) = -\sum_{j=1}^{n_h} \lambda_j \nabla h_j(x)$$

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

$$\mathcal{L}(x_1, x_2, \lambda) = x_1 + 2x_2 + \lambda \left(\frac{1}{4}x_1^2 + x_2^2 - 1 \right)$$

$$\begin{aligned} \underset{x_1, x_2}{\text{minimize}} \quad & f(x_1, x_2) = x_1 + 2x_2 \\ \text{subject to} \quad & h(x_1, x_2) = \tfrac{1}{4}x_1^2 + x_2^2 - 1 = 0 \end{aligned}$$

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,

$$\frac{\partial \mathcal{L}}{\partial x_1} = 1 + \frac{1}{2}\lambda x_1 = 0 \qquad \frac{\partial \mathcal{L}}{\partial x_2} = 2 + 2\lambda x_2 = 0$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \frac{1}{4}x_1^2 + x_2^2 - 1 = 0.$$

$$\begin{array}{ll} \underset{x_1, x_2}{\text{minimize}} & f(x_1, x_2) = x_1 + 2x_2 \\ \text{subject to} & h(x_1, x_2) = \frac{1}{4}x_1^2 + x_2^2 - 1 = 0 \end{array}$$

The Lagrangian function

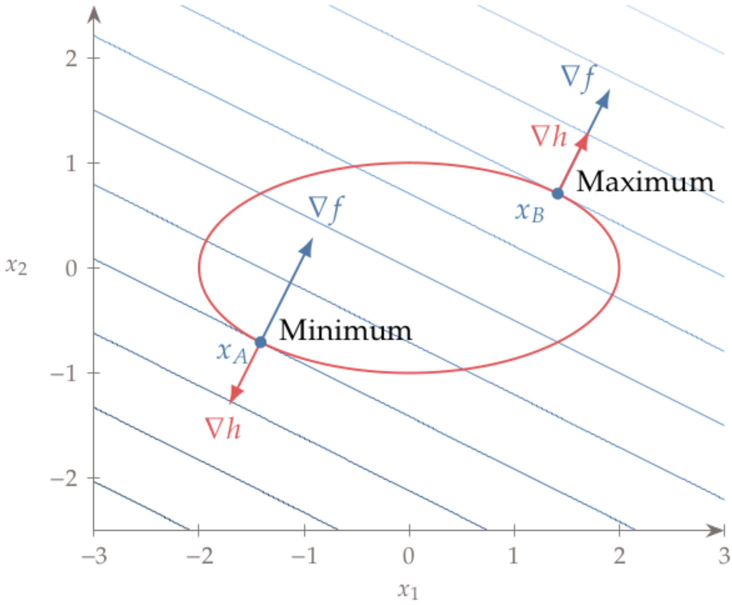
$$\mathcal{L}(x_1, x_2, \lambda) = x_1 + 2x_2 + \lambda\left(\frac{1}{4}x_1^2 + x_2^2 - 1\right)$$

Differentiating this to get the first-order optimality conditions,

$$\frac{\partial \mathcal{L}}{\partial x_1} = 1 + \frac{1}{2}\lambda x_1 = 0 \qquad \frac{\partial \mathcal{L}}{\partial x_2} = 2 + 2\lambda x_2 = 0$$
$$\frac{\partial \mathcal{L}}{\partial \lambda} = \frac{1}{4}x_1^2 + x_2^2 - 1 = 0.$$

Solving these three equations for the three unknowns (x_1, x_2, λ), we obtain two possible solutions:

$$x_A = (x_1, x_2) = (-\sqrt{2}, -\sqrt{2}/2), \quad \lambda_A = \sqrt{2}$$
$$x_B = (x_1, x_2) = (\sqrt{2}, \sqrt{2}/2), \quad \lambda_A = -\sqrt{2}$$



∇f
∇h
Maximum
x_B
∇f
Minimum
x_A
∇h
x_2
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
x_1

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 λ^* 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 λ^* 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

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

Inequality Constraints

Recall that the constrained optimization problem is

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

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

Sequential Quadratic Programming

Quadratic Programming

The *quadratic optimization problem with equality constraints* is to

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2} x^\top Q x + q^\top x \\ \text{by varying} \quad & x \\ \text{subject to} \quad & Ax + b = 0 \end{aligned}$$

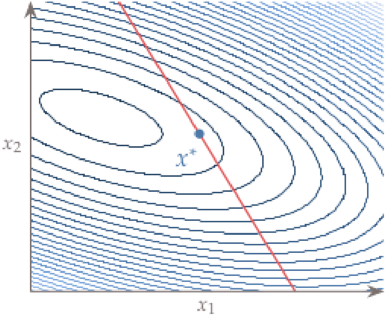
Quadratic Programming

The *quadratic optimization problem with equality constraints* is to

$$\begin{array}{rl} \text{minimize} & \frac{1}{2}x^\top Q x + q^\top x \\ \text{by varying} & x \\ \text{subject to} & Ax + b = 0 \end{array}$$

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

$$L(x, \lambda) = \frac{1}{2} x^\top Q x + q^\top x + \lambda^\top (Ax + b)$$

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

$$L(x, \lambda) = \frac{1}{2} x^\top Q x + q^\top x + \lambda^\top (Ax + b)$$

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 $f : \mathbb{R}^n \to \mathbb{R}$ and arbitrary constraint functions $h_j : \mathbb{R}^n \to \mathbb{R}$.

Consider the Lagrangian function $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \to \mathbb{R}$ 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 $f : \mathbb{R}^n \to \mathbb{R}$ and arbitrary constraint functions $h_j : \mathbb{R}^n \to \mathbb{R}$.

Consider the Lagrangian function $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \to \mathbb{R}$ defined by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \quad \text{here} \quad h(x) = (h_1(x), \ldots, h_{n_h}(x))^\top$$

We search for the stationary point of $\mathcal{L}$, that is (x^*, λ^*) 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^*, λ^*).

Lagrange-Newton

Now consider an arbitrary $f : \mathbb{R}^n \to \mathbb{R}$ and arbitrary constraint functions $h_j : \mathbb{R}^n \to \mathbb{R}$.

Consider the Lagrangian function $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \to \mathbb{R}$ defined by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \quad \text{here} \quad h(x) = (h_1(x), \ldots, h_{n_h}(x))^\top$$

We search for the stationary point of $\mathcal{L}$, that is (x^*, λ^*) 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^*, λ^*).

From Lagrange theorem: If x^* is regular and solves the COP, then there exists λ^* such that (x^*, λ^*) solves the system of equations.

Lagrange-Newton

Now consider an arbitrary $f : \mathbb{R}^n \to \mathbb{R}$ and arbitrary constraint functions $h_j : \mathbb{R}^n \to \mathbb{R}$.

Consider the Lagrangian function $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^{n_h} \to \mathbb{R}$ defined by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \quad \text{here} \quad h(x) = (h_1(x), \ldots, h_{n_h}(x))^\top$$

We search for the stationary point of $\mathcal{L}$, that is (x^*, λ^*) 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^*, λ^*).

From Lagrange theorem: If x^* is regular and solves the COP, then there exists λ^* such that (x^*, λ^*) solves the system of equations.

We use Newton's method to solve the system of equations.

Lagrange-Newton

Start with some (x_0, λ_0) and compute $(x_1, \lambda_1), \ldots, (x_k, \lambda_k), \ldots$

Lagrange-Newton

Start with some (x_0, λ_0) and compute $(x_1, \lambda_1), \dots, (x_k, \lambda_k), \dots$

In every step we compute (x_{k+1}, λ_{k+1}) from (x_k, λ_k) using Newton's step.

Lagrange-Newton

Start with some (x_0, λ_0) and compute $(x_1, \lambda_1), \ldots, (x_k, \lambda_k), \ldots$

In every step we compute (x_{k+1}, λ_{k+1}) from (x_k, λ_k) using Newton's step.

Consider the gradient of the Lagrangian:

$$\begin{aligned}\nabla\mathcal{L}(x_k, \lambda_k) &= (\nabla_x\mathcal{L}(x_k, \lambda_k), \nabla_\lambda\mathcal{L}(x_k, \lambda_k))^\top \\ &= (\nabla f(x_k) + \sum_{j=1}^{n_h} \lambda_{kj}\nabla h_j(x_k), \quad h(x_k))^\top \in \mathbb{R}^{n+n_h}\end{aligned}$$

Lagrange-Newton

Start with some (x_0, λ_0) and compute $(x_1, \lambda_1), \ldots, (x_k, \lambda_k), \ldots$

In every step we compute (x_{k+1}, λ_{k+1}) from (x_k, λ_k) using Newton's step.

Consider the gradient of the Lagrangian:

$$\nabla \mathcal{L}(x_k, \lambda_k) = (\nabla_x \mathcal{L}(x_k, \lambda_k), \nabla_\lambda \mathcal{L}(x_k, \lambda_k))^\top$$

$$= (\nabla f(x_k) + \sum_{j=1}^{n_h} \lambda_{kj} \nabla h_j(x_k), \quad h(x_k))^\top \in \mathbb{R}^{n+n_h}$$

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, λ_0) and compute $(x_1, \lambda_1), \ldots, (x_k, \lambda_k), \ldots$

In every step we compute (x_{k+1}, λ_{k+1}) from (x_k, λ_k) using Newton's step.

Consider the gradient of the Lagrangian:

$$\nabla \mathcal{L}(x_k, \lambda_k) = (\nabla_x \mathcal{L}(x_k, \lambda_k), \nabla_\lambda \mathcal{L}(x_k, \lambda_k))^\top$$
$$= (\nabla f(x_k) + \sum_{j=1}^{n_h} \lambda_{kj} \nabla h_j(x_k), \quad h(x_k))^\top \in \mathbb{R}^{n+n_h}$$

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

$$x_{k+1} = x_k + p_k \qquad \lambda_{k+1} = \lambda_k + \mu_k$$
$$(p_k, \mu_k) = -\left(\nabla^2 \mathcal{L}(x_k, \lambda_k)\right)^{-1} \nabla \mathcal{L}(x_k, \lambda_k)$$

Hessian of Lagrangian

Note that

$$\nabla^2 \mathcal{L}(x_k, \lambda_k) = \begin{pmatrix} \nabla_{xx}\mathcal{L}(x_k, \lambda_k) & \nabla_{x\lambda}\mathcal{L}(x_k, \lambda_k) \\ \nabla_{\lambda x}\mathcal{L}(x_k, \lambda_k) & \nabla_{\lambda\lambda}\mathcal{L}(x_k, \lambda_k) \end{pmatrix}$$
$$= \begin{pmatrix} H(x_k, \lambda_k) & \nabla h(x_k) \\ \nabla h(x_k)^\top & 0 \end{pmatrix}$$

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 $j = 1, \ldots, n_h$.

Lagrange-Newton for Equality Constraints

Algorithm 14 Lagrange-Newton

Choose starting point x_0
$k \leftarrow 0$
repeat
    Compute $\nabla f(x_k)$, $\nabla h(x_k)$, $h(x_k)$
    Compute $\nabla \mathcal{L}(x_k, \lambda_k)$
    Compute Hessians $H_f(x_k), H_{h_j}(x_k)$ for $j = 1, \ldots, n_h$
    Compute Lagrangian-Hessian $H(x_k, \lambda_k)$
    Compute $\nabla^2 \mathcal{L}(x_k, \lambda_k)$
    Compute $(p_k, \mu_k)^\top = -\left(\nabla^2 \mathcal{L}(x_k, \lambda_k)\right)^{-1} \nabla \mathcal{L}(x_k, \lambda_k)$
    $x_{k+1} \leftarrow x_k + p_k$
    $\lambda_{k+1} \leftarrow \lambda_k + \mu_k$
    $k \leftarrow k + 1$
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

$$\begin{array}{rll} \text{minimize} & f(x) & \\ \text{by varying} & x \in \mathbb{R}^n & \\ \text{subject to} & g_i(x) \leq 0 & i = 1, \ldots, n_g \\ & h_j(x) = 0 & j = 1, \ldots, n_h \end{array}$$

We assume that

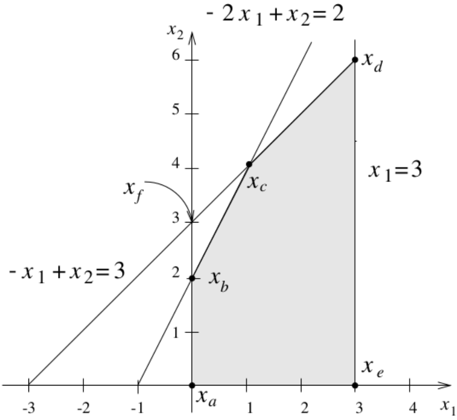
- f is linear, i.e.,

$$f(x) = c^\top x \qquad \text{here } c \in \mathbb{R}^n$$

- each g_i is linear,
- each h_j is linear.

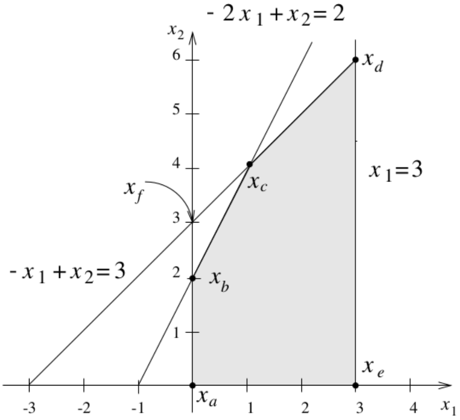
For convenience, in what follows, we also allow constraints of the form $g_i(x) \geq 0$.

Example



$$
\begin{aligned}
\text{minimize} \quad & z = -x_1 - 2x_2 \\
\text{subject to} \quad & -2x_1 + x_2 - 2 \le 0 \\
& -x_1 + x_2 - 3 \le 0 \\
& x_1 - 3 \le 0 \\
& x_1, x_2 \ge 0.
\end{aligned}
$$

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*

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Here

- $x = (x_1, \ldots, x_n)^\top \in \mathbb{R}^n$
- $c = (c_1, \ldots, c_n)^\top \in \mathbb{R}^n$
- 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.
- $b = (b_1, \ldots, b_m)^\top \geq 0$

  $b \geq 0$ means $b_i \geq 0$ for all i.

Standard Form

The *standard form linear program*

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Here

- $x = (x_1, \ldots, x_n)^\top \in \mathbb{R}^n$
- $c = (c_1, \ldots, c_n)^\top \in \mathbb{R}^n$
- 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.
- $b = (b_1, \ldots, b_m)^\top \geq 0$

  $b \geq 0$ 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 either 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 either 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

$$\begin{aligned} \text{maximize} \quad & z = -5x_1 - 3x_2 \\ \text{subject to} \quad & 3x_1 - 5x_2 - 5 \leq 0 \\ & -4x_1 - 9x_2 + 4 \leq 0 \end{aligned}$$

Introduce the bounded variables:

$$\begin{aligned} \text{maximize} \quad & z = -5x_1' + 5x_1'' - 3x_2' + 3x_2'' \\ \text{subject to} \quad & 3x_1' - 3x_1'' - 5x_2' + 5x_2'' - 5 \leq 0 \\ & -4x_1' + 4x_1'' - 9x_2' + 9x_2'' + 4 \leq 0 \\ & x_1', x_1'', x_2', x_2'' \geq 0 \end{aligned}$$

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:

$$A = \begin{pmatrix} 3 & -3 & -5 & 5 & 1 & 0 \\ 4 & -4 & 9 & -9 & 0 & -1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5, x_6)^\top$$

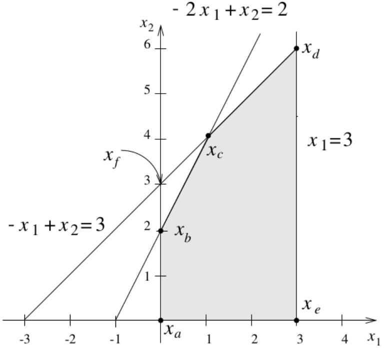
Note that we have renamed the variables.

$$b = (5, 4)^\top$$

$$Ax = b \text{ where } x \geq 0$$

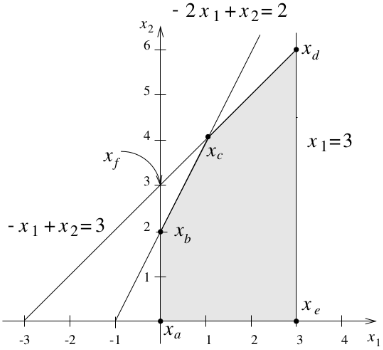
$$c = (-5, 5, -3, 3)^\top$$

Example



$$
\begin{aligned}
\text{minimize} \quad & z = -x_1 - 2x_2 \\
\text{subject to} \quad & -2x_1 + x_2 - 2 \le 0 \\
& -x_1 + x_2 - 3 \le 0 \\
& x_1 - 3 \le 0 \\
& x_1, x_2 \ge 0.
\end{aligned}
$$

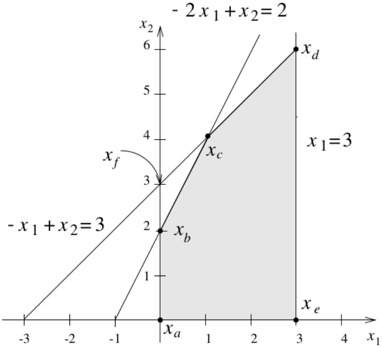
Example



Transform to

$$
\begin{aligned}
\text{minimize} \quad & z = -x_1 - 2x_2 \\
\text{subject to} \quad & -2x_1 + x_2 + s_1 = 2 \\
& -x_1 + x_2 + s_2 = 3 \\
& x_1 + s_3 = 3 \\
& x_1, x_2, s_1, s_2, s_3 \geq 0
\end{aligned}
$$

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:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Assumptions

Consider a linear programming problem in the standard form:

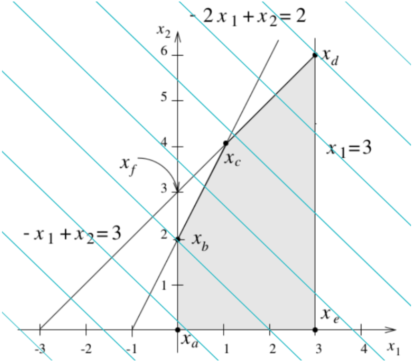
$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

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)

$$\begin{aligned} x_1 + x_2 &\leq 2 \\ x_1 &\leq 1 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Add slack variables x_3, x_4:

$$\begin{aligned} x_1 + x_2 + x_3 &= 2 \\ x_1 + x_4 &= 1 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

$$A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

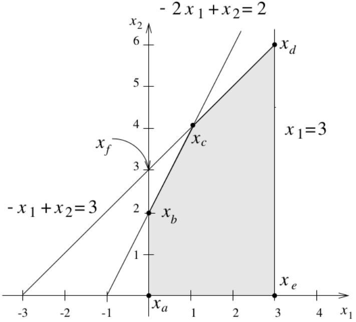
$$x = (x_1, x_2, x_3, x_4)^\top$$

$$b = (2, 1)^\top$$

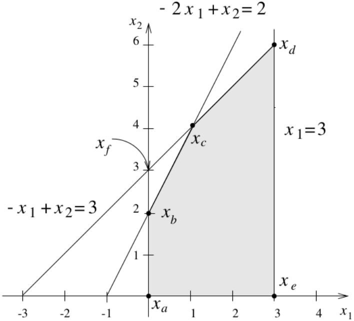
$$Ax = b \text{ where } x \geq 0$$

For now let us ignore the objective function and play with the polyhedron defined by the above inequalities.

$$
\begin{aligned}
-2x_1 + x_2 + x_3 &= 2 \\
-x_1 + x_2 + x_4 &= 3 \\
x_1 + x_5 &= 3 \\
x_1, x_2, x_3, x_4, x_5 &\geq 0
\end{aligned}
$$

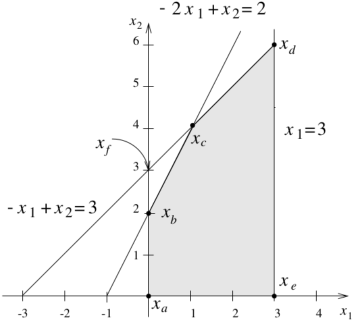


$$
\begin{aligned}
-2x_1 + x_2 + x_3 &= 2 \\
-x_1 + x_2 + x_4 &= 3 \\
x_1 + x_5 &= 3 \\
x_1, x_2, x_3, x_4, x_5 &\geq 0
\end{aligned}
$$



$$
A = (u_1 \; u_2 \; u_3 \; u_4 \; u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}
$$

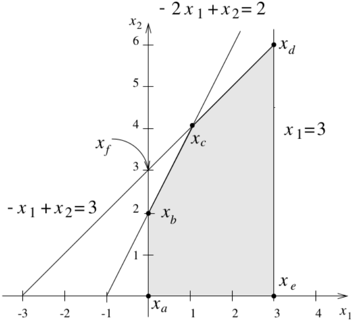
$$\begin{aligned} -2x_1 + x_2 + x_3 &= 2 \\ -x_1 + x_2 + x_4 &= 3 \\ x_1 + x_5 &= 3 \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$



$$A = (u_1 \; u_2 \; u_3 \; u_4 \; u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$\begin{aligned} -2x_1 + x_2 + x_3 &= 2 \\ -x_1 + x_2 + x_4 &= 3 \\ x_1 + x_5 &= 3 \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$

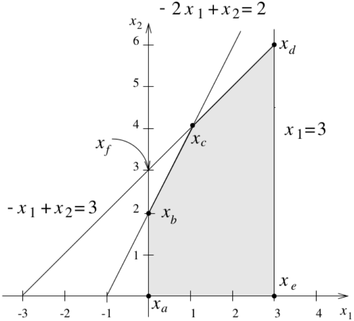


$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$b = (2, 3, 3)^\top$$

$$
\begin{aligned}
-2x_1 + x_2 + x_3 &= 2 \\
-x_1 + x_2 + x_4 &= 3 \\
x_1 + x_5 &= 3 \\
x_1, x_2, x_3, x_4, x_5 &\geq 0
\end{aligned}
$$



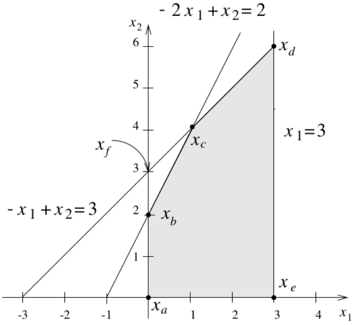
$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$b = (2, 3, 3)^\top$$

$Ax = b$ where $x \geq 0$

$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$



$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$

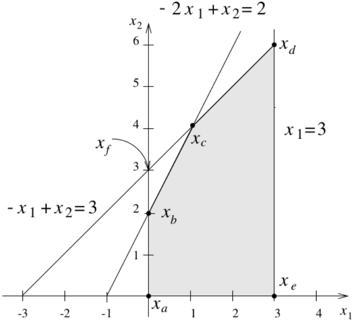
Consider a basis $\{x_3, x_4, x_5\}$ with

$$B = (u_3\ u_4\ u_5) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

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 $Bx_B = b$? $\quad x_B = (x_3, x_4, x_5)^\top = (2, 3, 3)^\top$.

The corresponding basic solution is

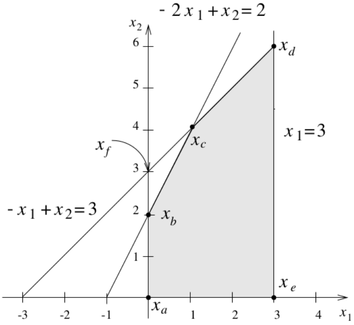
$$x = (x_1, x_2, x_3, x_4, x_5)^\top = (0, 0, 2, 3, 3)^\top = x_a \quad \text{Feasible!}$$

$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



Consider a basis $\{x_2, x_3, x_5\}$ with

$$B = (a_2\ a_3\ a_5) = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

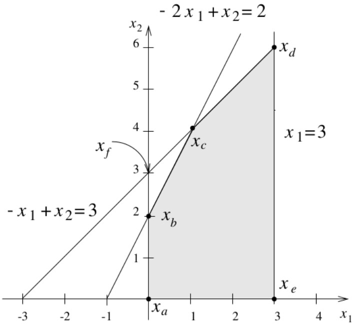
What is x_B satisfying $Bx_B = b$?

$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



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

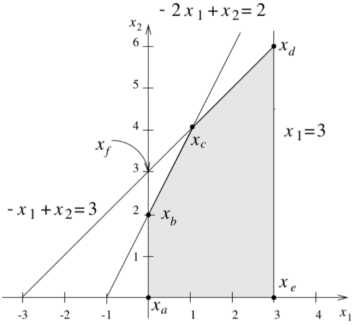
$$x = (x_1, x_2, x_3, x_4, x_5)^\top = (0, 3, -1, 0, 3)^\top = x_f \quad \text{Not feasible!}$$

$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



Consider a basis $\{x_1, x_2, x_3\}$ with

$$B = (u_1\ u_2\ u_3) = \begin{pmatrix} -2 & 1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

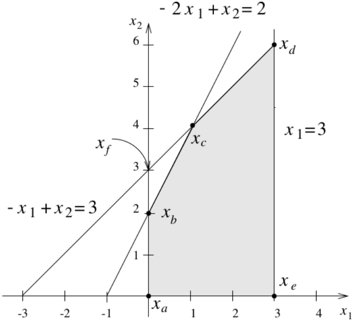
What is x_B satisfying $Bx_B = b$?

$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



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 $Bx_B = b$? $x_B = (x_1, x_2, x_3)^\top = (3, 6, 2)^\top$.

The corresponding basic solution is

$$x = (x_1, x_2, x_3, x_4, x_5)^\top = (3, 6, 2, 0, 0)^\top = x_d \quad \text{Feasible!}$$

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 Θ 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 Θ 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 Θ if there are no two points x' and x'' in Θ such that $x = \alpha x' + (1 - \alpha)x''$ for some $\alpha \in (0, 1)$.

Extreme Points

Note that the set Θ 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 Θ if there are no two points x' and x'' in Θ such that $x = \alpha x' + (1 - \alpha)x''$ for some $\alpha \in (0, 1)$.

Theorem 19

Let Θ 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 $A \in \mathbb{R}^{m \times n}, m < n$, $rank(A) = m$.
Then, x is an extreme point of Θ if and only if x is a basic feasible solution to $Ax = b, x \geq 0$.

Extreme Points

Note that the set Θ 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 Θ if there are no two points x' and x'' in Θ such that $x = \alpha x' + (1 - \alpha)x''$ for some $\alpha \in (0, 1)$.

Theorem 19

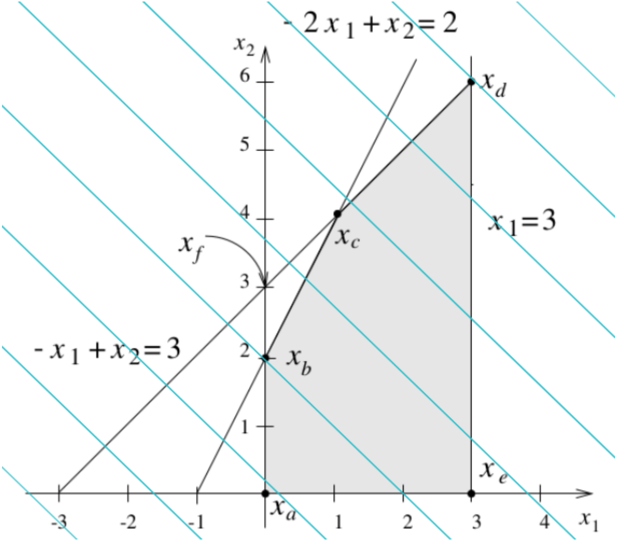
Let Θ be the convex set consisting of all feasible solutions that is, all $x \in \mathbb{R}^n$ satisfying:

$$Ax = b, \quad x \geq 0,$$

where $A \in \mathbb{R}^{m \times n}, m < n$, $rank(A) = m$.
Then, x is an extreme point of Θ 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 $A = (B\ N) = (u_1 \ldots u_m\, u_{m+1} \ldots u_n)$ where $B = (u_1 \ldots u_m)$ and $N = (u_{m+1} \ldots u_n)$.

Note that each u_i is a column vector of dimension m.

Changing Basis (Non-Degenerate Case)

Consider a basis B and write $A = (B\ N) = (u_1 \ldots u_m\, u_{m+1} \ldots u_n)$ where $B = (u_1 \ldots u_m)$ and $N = (u_{m+1} \ldots u_n)$.

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

$$x_1 u_1 + \cdots x_m u_m = b$$

For a non-degenerate case, we have $x_j > 0$ for all $j = 1, \ldots, m$.

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

$$x_1 u_1 + \cdots x_m u_m = b$$

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 $y_1 u_1 + \cdots + y_m u_m = u_i$.

Changing Basis (Non-Degenerate Case)

Consider a basis B and write $A = (B\ N) = (u_1 \ldots u_m\, u_{m+1} \ldots u_n)$ where $B = (u_1 \ldots u_m)$ and $N = (u_{m+1} \ldots u_n)$.

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

$$x_1 u_1 + \cdots x_m u_m = b$$

For a non-degenerate case, we have $x_j > 0$ for all $j = 1, \ldots, m$.

Now as B is a basis, we have that for each $i \in \{m+1, \ldots, n\}$ there are coefficients $y_1, \ldots, y_m$ such that $y_1 u_1 + \cdots + y_m u_m = u_i$. Then

$$\begin{aligned} b &= x_1 u_1 + \cdots x_m u_m \\ &= x_1 u_1 + \cdots x_m u_m - \alpha u_i + \alpha u_i \\ &= x_1 u_1 + \cdots x_m u_m - \alpha(y_1 u_1 + \cdots + y_m u_m) + \alpha u_i \\ &= (x_1 - \alpha y_1) u_1 + \cdots + (x_m - \alpha y_m) u_m + \alpha u_i \end{aligned}$$

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

$$x_1 u_1 + \cdots x_m u_m = b$$

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 $y_1 u_1 + \cdots + y_m u_m = u_i$. Then

$$\begin{aligned} b &= x_1 u_1 + \cdots x_m u_m \\ &= x_1 u_1 + \cdots x_m u_m - \alpha u_i + \alpha u_i \\ &= x_1 u_1 + \cdots x_m u_m - \alpha(y_1 u_1 + \cdots + y_m u_m) + \alpha u_i \\ &= (x_1 - \alpha y_1) u_1 + \cdots + (x_m - \alpha y_m) u_m + \alpha u_i \end{aligned}$$

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 $\alpha \to \infty$.

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \land k = 1, \ldots, m\} > 0$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\} > 0$$

There would be a *unique* $j \in \{1, \ldots, m\}$ 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 $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\} > 0$$

There would be a *unique* $j \in \{1, \ldots, m\}$ 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:

$$j = \text{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \land k = 1, \ldots, m\} > 0$$

There would be a *unique* $j \in \{1, \ldots, m\}$ 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:

$$j = \text{argmin}\{x_k/y_k \mid y_k > 0 \land k = 1, \ldots, m\}$$

Obtain a basis $B_{j \to i} = B \setminus \{j\} \cup \{i\}$ and a basic feasible solution

$$x_{j \to i} = (x'_1, \ldots, x'_{j-1}, 0, x'_{j+1}, \ldots, x'_m, 0, \ldots, 0, \alpha, 0, \ldots, 0)^\top$$

Here $x'_k = x_k - \alpha y_k$ for each $k \in \{1, \ldots, j-1, j+1, \ldots, m\}$.

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k / y_k \mid y_k > 0 \wedge k = 1, \ldots, m\} > 0$$

There would be a *unique* $j \in \{1, \ldots, m\}$ 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:

$$j = \text{argmin}\{x_k / y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Obtain a basis $B_{j \to i} = B \setminus \{j\} \cup \{i\}$ and a basic feasible solution

$$x_{j \to i} = (x'_1, \ldots, x'_{j-1}, 0, x'_{j+1}, \ldots, x'_m, 0, \ldots, 0, \alpha, 0, \ldots, 0)^\top$$

Here $x'_k = x_k - \alpha y_k$ for each $k \in \{1, \ldots, j-1, j+1, \ldots, m\}$.
We say that we *pivot about (j, i)*.

Algorithm 15 Simplex - **Non-degenerate**

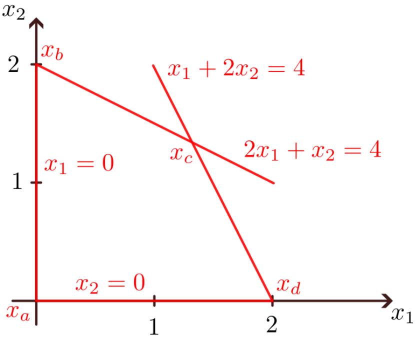
Choose a starting basis $B = (u_1 \dots u_m)$ (here $A = (B\ N)$)
repeat
    Compute the basic solution x for the basis B
    for $i \in \{m+1, \dots, n\}$ **do**
        Solve $B(y_1, \dots, y_m)^\top = u_i$
        if $y_k \leq 0$ for all $k \in \{1, \dots, m\}$ **then**
            Stop, unbounded problem.
        end if
        Select $j = \text{argmin}\{x_k / y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$
        Compute $x_{j \to i}$
    end for
    if $c^\top(x_{j \to i} - x) \geq 0$ for all $i \in \{m+1, \dots, n\}$ **then**
        Stop, we have an optimal solution.
    end if
    Select $i \in \{m+1, \dots, n\}$ such that $c^\top(x_{j \to i} - x) < 0$
    $B \leftarrow B_{j \to i}$
until convergence

$$A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4)^\top$$

$$b = (4, 4)^\top$$

$$c = (-1, -1, 0, 0)^\top$$



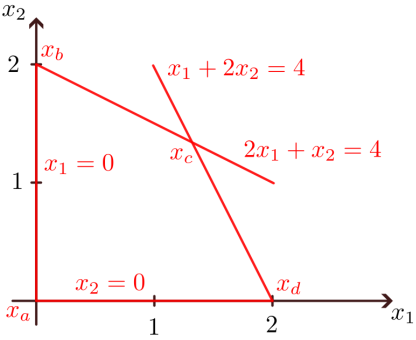
minimize $c^\top x$ subject to $Ax = b$ where $x \geq 0$

$$A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4)^\top$$

$$b = (4, 4)^\top$$

$$c = (-1, -1, 0, 0)^\top$$



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 $\alpha = x_4/y_4 = 2$.

$$x_{4\to1} = (\alpha, 0, (x_3 - \alpha y_3), (x_4 - \alpha y_4)) = (2, 0, 2, 0)$$

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 $\alpha = x_4/y_4 = 2$.

$$x_{4\to 1} = (\alpha, 0, (x_3 - \alpha y_3), (x_4 - \alpha y_4)) = (2, 0, 2, 0)$$

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 $\alpha = x_4/y_4 = 2$.

$$x_{4\to 1} = (\alpha, 0, (x_3 - \alpha y_3), (x_4 - \alpha y_4)) = (2, 0, 2, 0)$$

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 $\alpha = x_4/y_4 = 2$.

$$x_{4\to1} = (\alpha, 0, (x_3 - \alpha y_3), (x_4 - \alpha y_4)) = (2, 0, 2, 0)$$

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 $c^\top (x_{4\to1} - x) = -2 < 0$
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:

$$u_2 = (2, 1)^\top = B\ (1/2, 3/2)^\top \text{ thus } y = (y_1, y_3) = (1/2, 3/2)$$

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:

$$u_2 = (2, 1)^\top = B\ (1/2, 3/2)^\top \text{ thus } y = (y_1, y_3) = (1/2, 3/2)$$

Now $\alpha = x_3/y_3 = 4/3 < 2/(1/2) = 4 = x_1/y_1$, pivot about $(3, 2)$

$$x_{3\to 2} = ((x_1 - \alpha y_1), \alpha, (x_3 - \alpha y_3), 0) = (4/3, 4/3, 0, 0)$$

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:

$$u_2 = (2, 1)^\top = B\ (1/2, 3/2)^\top \text{ thus } y = (y_1, y_3) = (1/2, 3/2)$$

Now $\alpha = x_3/y_3 = 4/3 < 2/(1/2) = 4 = x_1/y_1$, pivot about $(3, 2)$

$$x_{3\to 2} = ((x_1 - \alpha y_1), \alpha, (x_3 - \alpha y_3), 0) = (4/3, 4/3, 0, 0)$$

$$c^\top (x_{3\to 2} - x) = c(-2/3, 4/3)^\top = -2/3 < 0$$

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 that the simplex method is applied to a linear program and that 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 that the simplex method is applied to a linear program and that 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 that the simplex method is applied to a linear program and that 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

$$x_1 u_1 + \cdots + x_m u_m = b$$

For a degenerate case, we have $x_j \geq 0$ for all $j \in \{1, \dots, m\}$, and *may have* $x_i = 0$ *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 $x_i = 0$ 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

$$x_1 u_1 + \cdots + x_m u_m = b$$

For a degenerate case, we have $x_j \geq 0$ for all $j \in \{1, \dots, m\}$, and *may have* $x_i = 0$ *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 $y_1 u_1 + \cdots + y_m u_m = u_i$. Then

$$\begin{aligned} b &= x_1 u_1 + \cdots + x_m u_m \\ &= x_1 u_1 + \cdots + x_m u_m - \alpha u_i + \alpha u_i \\ &= x_1 u_1 + \cdots + x_m u_m - \alpha(y_1 u_1 + \cdots + y_m u_m) + \alpha u_i \\ &= (x_1 - \alpha y_1) u_1 + \cdots + (x_m - \alpha y_m) u_m + \alpha u_i \end{aligned}$$

Changing Basis (Degenerate Case)

Consider a basis B and write $A = (B \; N) = (u_1 \ldots u_m \, u_{m+1} \ldots u_n)$ where $B = (u_1 \ldots u_m)$ and $N = (u_{m+1} \ldots u_n)$.

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

$$x_1 u_1 + \cdots + x_m u_m = b$$

For a degenerate case, we have $x_j \geq 0$ for all $j \in \{1, \ldots, m\}$, and *may have* $x_i = 0$ *for some* $j \in \{1, \ldots, m\}$.

Now as B is a basis, we have that for each $i \in \{m+1, \ldots, n\}$ there are coefficients $y_1, \ldots, y_m$ such that $y_1 u_1 + \cdots + y_m u_m = u_i$. Then

$$\begin{aligned} b &= x_1 u_1 + \cdots + x_m u_m \\ &= x_1 u_1 + \cdots + x_m u_m - \alpha u_i + \alpha u_i \\ &= x_1 u_1 + \cdots + x_m u_m - \alpha (y_1 u_1 + \cdots + y_m u_m) + \alpha u_i \\ &= (x_1 - \alpha y_1) u_1 + \cdots + (x_m - \alpha y_m) u_m + \alpha u_i \end{aligned}$$

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 $\alpha \to \infty$.

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Otherwise, there *exists* $j \in \{1, \ldots, m\}$ 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 $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Otherwise, there *exists* $j \in \{1, \ldots, m\}$ 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:

$$j \in \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Otherwise, there *exists* $j \in \{1, \ldots, m\}$ 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:

$$j \in \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Obtain a basis $B_{j \to i} = B \smallsetminus \{j\} \cup \{i\}$ and a basic feasible solution

$$x_{j \to i} = (x'_1, \ldots, x'_{j-1}, 0, x'_{j+1}, \ldots, x'_m, 0, \ldots, 0, \alpha, 0, \ldots, 0)^\top$$

Here $x'_k = x_k - \alpha y_k$ for each $k \in \{1, \ldots, j-1, j+1, \ldots, m\}$.
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 $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Otherwise, there *exists* $j \in \{1, \ldots, m\}$ 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:

$$j \in \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Obtain a basis $B_{j \to i} = B \smallsetminus \{j\} \cup \{i\}$ and a basic feasible solution

$$x_{j \to i} = (x'_1, \ldots, x'_{j-1}, 0, x'_{j+1}, \ldots, x'_m, 0, \ldots, 0, \alpha, 0, \ldots, 0)^\top$$

Here $x'_k = x_k - \alpha y_k$ for each $k \in \{1, \ldots, j-1, j+1, \ldots, m\}$.

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

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

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_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 $x = (x_1, x_2, x_3, x_4)^\top = (0, 1, 0, 0)^\top$ 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 $\alpha = x_2/y_2 = 1$

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 $\alpha = x_2/y_2 = 1$

$$x_{2\to 4} = (0, (x_2 - \alpha y_2), (x_3 - \alpha y_3), \alpha)^\top = (0, 0, 1, 1)^\top$$

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

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 $\alpha = x_2/y_2 = 1$

$$x_{2\to4} = (0, (x_2 - \alpha y_2), (x_3 - \alpha y_3), \alpha)^\top = (0, 0, 1, 1)^\top$$

Note that $c^\top x_{2\to4} = 0$.

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

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

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 $\alpha = x_3/y_3 = 0$.

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 $\alpha = x_3/y_3 = 0$.

$$x_{3\to1} = (\alpha, (x_2 - \alpha y_2), (x_3 - \alpha y_3), 0)^\top = (0, 1, 0, 0)^\top$$

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\; u_2\; u_3\; u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

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 $\alpha = x_3/y_3 = 0$.

$$x_{3\to1} = (\alpha, (x_2 - \alpha y_2), (x_3 - \alpha y_3), 0)^\top = (0, 1, 0, 0)^\top$$

No change in the basic solution, and thus $c^\top x_{3\to1} = c^\top x = 0$.

Thus **no effect on the objective value either!**

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 $\alpha = x_3/y_3 = 0$.

$$x_{3\to1} = (\alpha, (x_2 - \alpha y_2), (x_3 - \alpha y_3), 0)^\top = (0, 1, 0, 0)^\top$$

No change in the basic solution, and thus $c^\top x_{3\to1} = c^\top x = 0$.

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 $i \in \{m+1, \ldots, n\}$ we have

$$c^\top x_{j \to i} - c^\top x = (c_i - c_B^\top y)\alpha$$

Here $y = (y_1, \ldots, y_m)^\top$ 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 $i \in \{m+1, \ldots, n\}$ we have

$$c^\top x_{j\to i} - c^\top x = (c_i - c_B^\top y)\alpha$$

Here $y = (y_1, \ldots, y_m)^\top$ where $By = u_i$.

For non-degenerate case, we have $\alpha > 0$ and thus

$$c^\top x_{j\to i} < c^\top x \quad \text{iff} \quad c_i - c_B^\top y < 0$$

For the degenerate case, we may have $\alpha = 0$ and $c_i - c_B y < 0$.

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 $i \in \{m+1, \ldots, n\}$ we have

$$c^\top x_{j \to i} - c^\top x = (c_i - c_B^\top y)\alpha$$

Here $y = (y_1, \ldots, y_m)^\top$ where $By = u_i$.

For non-degenerate case, we have $\alpha > 0$ and thus

$$c^\top x_{j \to i} < c^\top x \quad \text{iff} \quad c_i - c_B^\top y < 0$$

For the degenerate case, we may have $\alpha = 0$ and $c_i - c_B y < 0$.

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

$$
\begin{aligned}
c^\top x_{j\to i} &= c^\top(x'_1,\ldots,x'_{j-1},0,x'_{j+1},\ldots,x'_m,0,\ldots,0,\alpha,0,\ldots,0)^\top \\
&= c^\top(x'_1,\ldots,x'_{j-1},x'_j,x'_{j+1},\ldots,x'_m,0,\ldots,0,\alpha,0,\ldots,0)^\top \\
&= c_1x'_1+\cdots+c_mx'_m+c_i\alpha \\
&= c_1(x_1-\alpha y_1)+\cdots c_m(x_m-\alpha y_m)+c_i\alpha \\
&= (c_1x_1+\cdots+c_mx_m)-(c_1y_1+\cdots+c_my_m-c_i)\alpha \\
&= c^\top x-(-c_i+c_By)\alpha
\end{aligned}
$$

Here we use the fact that $x'_k = x_k - \alpha y_k$ for each $k \in \{1,\ldots,j-1,j+1,\ldots,m\}$ and that $x_j - \alpha y_j = 0$.

Then clearly

$$c^\top x_{j\to i} - c^\top x = (c_i - c_By)\alpha$$

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1,\ldots,m\}$$

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 $cx = 0$.

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

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

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

$$r_1 = c_1 - (c_2 y_2 + c_3 y_3) = -1 - (0 \cdot (-1) + 0 \cdot 2) = -1 < 0$$

So we should put x_1 into the basis (the reduced cost gets smaller).

Algorithm 16 Simplex

Choose a starting basis $B = (u_1 \dots u_m)$ (here $A = (B \ N)$)
repeat
  Compute the basic solution x for the basis B
  for $i \in \{m+1, \dots, n\}$ **do**
    Solve $B\,(y_1, \dots, y_m)^\top = u_i$
    if $y_k \leq 0$ for all $k \in \{1, \dots, m\}$ **then**
      Stop, unbounded problem.
    end if
    Select $j \in \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \land k = 1, \dots, m\}$
    Compute $r_i = c_i - c_B^\top y$ where $y = (y_1, \dots, y_m)^\top$
  end for
  if $r_i \geq 0$ for all $i \in \{m+1, \dots, n\}$ **then**
    Stop, we have an optimal solution.
  end if
  Select $i \in \{m+1, \dots, n\}$ such that $r_i < 0$
  $B \leftarrow B_{j \to i}$
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.)

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

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:

$$u_4 = (0, 1)^\top = B\,(-1/2, 1/2)^\top \text{ thus } y = (y_1, y_2) = (-1/2, 1/2)$$

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 $\alpha = x_2/y_2 = 2$

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 $\alpha = x_2/y_2 = 2$

$$x_{2\to 4} = ((x_1 - \alpha y_1), (x_2 - \alpha y_2), 0, \alpha) = (1, 0, 0, 2)$$

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 $\alpha = x_2/y_2 = 2$

$$x_{2\to 4} = ((x_1 - \alpha y_1), (x_2 - \alpha y_2), 0, \alpha) = (1, 0, 0, 2)$$

This is the minimizer!

Does this always work?

Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

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 $\alpha = x_2/y_2 = 2$

$$x_{2\to 4} = ((x_1 - \alpha y_1), (x_2 - \alpha y_2), 0, \alpha) = (1, 0, 0, 2)$$

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, we have a deterministic algorithm that 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, we have a deterministic algorithm that 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

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

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

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

We construct an artificial LP problem.

$$\begin{aligned} \text{minimize} \quad & y_1 + y_2 + \cdots + y_m \\ \text{subject to} \quad & (A \ I_m) \begin{pmatrix} x \\ y \end{pmatrix} = b \\ & \begin{pmatrix} x \\ y \end{pmatrix} \geq 0 \end{aligned}$$

Here $y = (y_1, \ldots, y_m)^\top$ 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:

$$\begin{aligned} \text{minimize} \quad & y_1 + y_2 + \cdots + y_m \\ \text{subject to} \quad & [A \ I_m] \begin{pmatrix} x \\ y \end{pmatrix} = b \\ & \begin{pmatrix} x \\ y \end{pmatrix} \geq 0 \end{aligned}$$

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.

Linear Programming
Properties

LP Complexity

Iterations of the simplex algorithm can be implemented to compute the first step using $\mathcal{O}(m^2 n)$ 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 $\mathcal{O}(m^2 n)$ 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	1×10^{11}
50	1×10^{29}
100	9×10^{58}
200	1×10^{119}
300	1×10^{179}
400	2×10^{239}
500	3×10^{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 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 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 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_.28programming.29_languages

For example, the well-known Gurobi solver uses the simplex algorithm to solve LP problems.

Linear Programming - Tableaus

Tableau

Consider a linear program in the standard form:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Tableau

Consider a linear program in the standard form:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

We have considered the simplex algorithm, which searches for the minimum by moving around the vertices of the feasible region.

Tableau

Consider a linear program in the standard form:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

We have considered the simplex algorithm, which searches for the minimum by moving around the vertices of the feasible region.

The algorithm is relatively straightforward but, in its original form, not so suitable for computations by hand.

Tableau

Consider a linear program in the standard form:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

We have considered the simplex algorithm, which searches for the minimum by moving around the vertices of the feasible region.

The algorithm is relatively straightforward but, in its original form, not so suitable for computations by hand.

Tableaus provide all information about the current state of the simplex algorithm and can be used to streamline the process.

Keep in mind that we are not developing a new algorithm. Tableau just provides another view of the same simplex algorithm as presented before.

Tableau (Matrix Form)

Consider LP with a matrix A and vectors b, c. Assume $A = (B\, N)$ where B consists of basic columns and N of the non-basic ones.

Tableau (Matrix Form)

Consider LP with a matrix A and vectors b, c. Assume $A = (B\, N)$ where B consists of basic columns and N of the non-basic ones.

Consider the following matrix (the *initial tableau*):

$$\begin{pmatrix} A & b \\ c^\top & 0 \end{pmatrix} = \begin{pmatrix} B & N & b \\ c_B^\top & c_N^\top & 0 \end{pmatrix}$$

Tableau (Matrix Form)

Consider LP with a matrix A and vectors b, c. Assume $A = (B\,N)$ where B consists of basic columns and N of the non-basic ones.

Consider the following matrix (the *initial tableau*):

$$\begin{pmatrix} A & b \\ c^\top & 0 \end{pmatrix} = \begin{pmatrix} B & N & b \\ c_B^\top & c_N^\top & 0 \end{pmatrix}$$

Apply elementary row operations so that the matrix B is turned into I_m (preserving the last row for now). That is, multiply with

$$\begin{pmatrix} B^{-1} & 0 \\ 0 & 1 \end{pmatrix}$$

The result is

$$\begin{pmatrix} B^{-1} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} B & N & b \\ c_B^\top & c_N^\top & 0 \end{pmatrix} = \begin{pmatrix} I_m & B^{-1}N & B^{-1}b \\ c_B^\top & c_N^\top & 0 \end{pmatrix}$$

Tableau (Matrix Form)

We have

$$\begin{pmatrix} I_m & B^{-1}N & B^{-1}b \\ c_B^\top & c_N^\top & 0 \end{pmatrix}$$

Tableau (Matrix Form)

We have

$$\begin{pmatrix} I_m & B^{-1}N & B^{-1}b \\ c_B^\top & c_N^\top & 0 \end{pmatrix}$$

We apply row operations to the last row to eliminate the $c_B^\top$. This corresponds to multiplying the matrix with

$$\begin{pmatrix} I_m & 0 \\ -c_B^\top & 1 \end{pmatrix}$$

Tableau (Matrix Form)

We have

$$\begin{pmatrix} I_m & B^{-1}N & B^{-1}b \\ c_B^\top & c_N^\top & 0 \end{pmatrix}$$

We apply row operations to the last row to eliminate the $c_B^\top$. This corresponds to multiplying the matrix with

$$\begin{pmatrix} I_m & 0 \\ -c_B^\top & 1 \end{pmatrix}$$

We obtain

$$\begin{pmatrix} I_m & 0 \\ -c_B^\top & 1 \end{pmatrix} \begin{pmatrix} I_m & B^{-1}N & B^{-1}b \\ c_B^\top & c_N^\top & 0 \end{pmatrix} = \begin{pmatrix} I_m & B^{-1}N & B^{-1}b \\ 0 & c_N^\top - c_B^\top B^{-1}N & -c_B^\top B^{-1}b \end{pmatrix}$$

This is the *canonical form tableau for the basis B*.

Tableau (Components)

Let $A = (u_1 \; \ldots, u_n)$, the basis $\{x_1, \ldots, x_m\}$, $B = (u_1 \; \ldots, u_m)$.

Assume $u_k = (u_{1k}, \ldots, u_{nk})$. Then the initial tableau is

$$\begin{pmatrix} B & N & b \\ c_B^\top & c_N^\top & 0 \end{pmatrix} = \begin{pmatrix} u_{11} & \cdots & u_{1m} & u_{1(m+1)} & \cdots & u_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ u_{m1} & \cdots & u_{mm} & u_{m(m+1)} & \cdots & u_{mn} & b_m \\ c_1 & \cdots & c_m & c_{m+1} & \cdots & c_n & 0 \end{pmatrix}$$

Tableau (Components)

Let $A = (u_1 \ \ldots, u_n)$, the basis $\{x_1, \ldots, x_m\}$, $B = (u_1 \ \ldots, u_m)$.

Assume $u_k = (u_{1k}, \ldots, u_{nk})$. Then the initial tableau is

$$\begin{pmatrix} B & N & b \\ c_B^\top & c_N^\top & 0 \end{pmatrix} = \begin{pmatrix} u_{11} & \cdots & u_{1m} & u_{1(m+1)} & \cdots & u_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ u_{m1} & \cdots & u_{mm} & u_{m(m+1)} & \cdots & u_{mn} & b_m \\ c_1 & \cdots & c_m & c_{m+1} & \cdots & c_n & 0 \end{pmatrix}$$

Now transform all columns of the upper part of the matrix (except the last row) to the basis B:

$$u_k = B(y_{1k}, \ldots, y_{mk})^\top \text{ for } k = 1, \ldots, n \text{ and } b' = B^{-1}b$$

Tableau (Components)

Let $A = (u_1 \ldots, u_n)$, the basis $\{x_1, \ldots, x_m\}$, $B = (u_1 \ldots, u_m)$.

Assume $u_k = (u_{1k}, \ldots, u_{nk})$. Then the initial tableau is

$$\begin{pmatrix} B & N & b \\ c_B^\top & c_N^\top & 0 \end{pmatrix} = \begin{pmatrix} u_{11} & \cdots & u_{1m} & u_{1(m+1)} & \cdots & u_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ u_{m1} & \cdots & u_{mm} & u_{m(m+1)} & \cdots & u_{mn} & b_m \\ c_1 & \cdots & c_m & c_{m+1} & \cdots & c_n & 0 \end{pmatrix}$$

Now transform all columns of the upper part of the matrix (except the last row) to the basis B:

$$u_k = B(y_{1k}, \ldots, y_{mk})^\top \text{ for } k = 1, \ldots, n \text{ and } b' = B^{-1}b$$

and obtain $u_k = y_{1k}u_1 + \cdots + y_{mk}u_m$ for $k = m+1, \ldots, n$ and thus

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ c_1 & \cdots & c_m & c_{m+1} & \cdots & c_n & 0 \end{pmatrix}$$

Tableau (Components)

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ c_1 & \cdots & c_m & c_{m+1} & \cdots & c_n & 0 \end{pmatrix}$$

Tableau (Components)

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ c_1 & \cdots & c_m & c_{m+1} & \cdots & c_n & 0 \end{pmatrix}$$

Use row operations to eliminate $c_1, \ldots, c_m$. This is equivalent to multiplying the above matrix with

$$\begin{pmatrix} I_m & 0 \\ -c_B^\top & 1 \end{pmatrix} = \begin{pmatrix} 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \\ -c_1 & \cdots & -c_m & 1 \end{pmatrix}$$

from the left. We obtain ...

Tableau (Components)

... the canonical form for the basis $\{x_1, \ldots, x_m\}$:

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ 0 & \cdots & 0 & c'_{m+1} & \cdots & c'_n & -z \end{pmatrix}$$

Tableau (Components)

... the canonical form for the basis $\{x_1, \ldots, x_m\}$:

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ 0 & \cdots & 0 & c'_{m+1} & \cdots & c'_n & -z \end{pmatrix}$$

Here, $(b'_1, \ldots, b'_m)^\top = B^{-1}b$ is the vector b transformed to the basis B, and for $k = m+1, \ldots, n$ we have

$$c'_k = c_k - (y_{1k}c_1 + \cdots + y_{mk}c_m)$$

the reduced cost for the k-th column (non-basic).

Tableau (Components)

... the canonical form for the basis $\{x_1, \ldots, x_m\}$:

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ 0 & \cdots & 0 & c'_{m+1} & \cdots & c'_n & -z \end{pmatrix}$$

Here, $(b'_1, \ldots, b'_m)^\top = B^{-1}b$ is the vector b transformed to the basis B, and for $k = m+1, \ldots, n$ we have

$$c'_k = c_k - (y_{1k}c_1 + \cdots + y_{mk}c_m)$$

the reduced cost for the k-th column (non-basic). Also, note that the basic solution is $x = (b'_1, \ldots, b'_m, 0, \ldots, 0)$, and hence

$$-z = (-c_1)b'_1 + \cdots + (-c_m)b'_m$$

is the negative of the value of the objective for the basic solution corresponding to the basis $\{x_1, \ldots, x_m\}$.

Recall that, by definition, the basic solution x satisfies $x_{m+1} = \cdots = x_n = 0$.

Tableau Simplex

Assume that for a basis B we have obtained the canonical tableau:

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ 0 & \cdots & 0 & c'_{m+1} & \cdots & c'_n & -z \end{pmatrix}$$

The simplex algorithm then proceeds as follows:

1. Choose $i \in \{m+1, \ldots, n\}$ such that $c'_i < 0$.
2. Choose $j \in \{1, \ldots, m\}$ minimizing b'_j / y_{ji} over all j satisfying $y_{ji} > 0$.

   Note that $b'_j = x_j$ for the basic solution x w.r.t. B.
3. Move the i-the column into the basis and the j-th column out of the basis.
4. Use elementary row operations to transform the tableau into the canonical form for the new basis.
5. Repeat until $b'_1, \ldots, b'_m \geq 0$,

Example

$$\begin{aligned} x_1 + x_2 &\leq 2 \\ x_1 &\leq 1 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Add slack variables x_3, x_4:

$$\begin{aligned} x_1 + x_2 + x_3 &= 2 \\ x_1 + x_4 &= 1 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

$$A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

$x = (x_1, x_2, x_3, x_4)^\top$

$b = (2, 1)^\top$

$Ax = b$ where $x \geq 0$

$c = (-3, -2, 0, 0)^\top$

Example

$$\begin{aligned} x_1 + x_2 &\leq 2 \\ x_1 &\leq 1 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Add slack variables x_3, x_4:

$$\begin{aligned} x_1 + x_2 + x_3 &= 2 \\ x_1 + x_4 &= 1 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

$$A = (u_1\, u_2\, u_3\, u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4)^\top$$

$$b = (2, 1)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$c = (-3, -2, 0, 0)^\top$$

Tableau for the basis $\{x_3, x_4\}$:

$$\left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

is already in the canonical form.

Note that the last row of the tableau corresponds to writing the objective as $-z + c^\top x = 0$ where z is a new variable and x is the basic solution for $\{x_3, x_4\}$.

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Choose x_1 to enter the basis (x_1 has the reduced cost -3 and x_2 has the reduced costs -2).

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Choose x_1 to enter the basis (x_1 has the reduced cost -3 and x_2 has the reduced costs -2).Now $b_1/y_{31} = 2/1 > 1/1 = b_2/y_{41}$. Thus, remove x_4 from the basis.

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Choose x_1 to enter the basis (x_1 has the reduced cost -3 and x_2 has the reduced costs -2).Now $b_1/y_{31} = 2/1 > 1/1 = b_2/y_{41}$. Thus, remove x_4 from the basis.We move to the basis $\{x_1, x_3\}$ and transform the tableau into the canonical form for this basis:

$$\left[\begin{array}{c|cccc|c} x_1 & 1 & y_{12} & 0 & y_{14} & b_1' \\ x_3 & 0 & y_{32} & 1 & y_{34} & b_2' \\ \hline -z & c_1' & c_2' & c_3' & c_4' & 3 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_1 & 1 & 0 & 0 & 1 & 1 \\ x_3 & 0 & 1 & 1 & -1 & 1 \\ \hline -z & 0 & -2 & 0 & 3 & 3 \end{array}\right]$$

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Choose x_1 to enter the basis (x_1 has the reduced cost -3 and x_2 has the reduced costs -2).Now $b_1/y_{31} = 2/1 > 1/1 = b_2/y_{41}$. Thus, remove x_4 from the basis.We move to the basis $\{x_1, x_3\}$ and transform the tableau into the canonical form for this basis:

$$\left[\begin{array}{c|cccc|c} x_1 & 1 & y_{12} & 0 & y_{14} & b_1' \\ x_3 & 0 & y_{32} & 1 & y_{34} & b_2' \\ \hline -z & c_1' & c_2' & c_3' & c_4' & 3 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_1 & 1 & 0 & 0 & 1 & 1 \\ x_3 & 0 & 1 & 1 & -1 & 1 \\ \hline -z & 0 & -2 & 0 & 3 & 3 \end{array}\right]$$

Here, the reduced cost of x_2 is -2, and of x_4 is 3. Thus, x_2 enters the basis.

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Choose x_1 to enter the basis (x_1 has the reduced cost -3 and x_2 has the reduced costs -2).Now $b_1/y_{31} = 2/1 > 1/1 = b_2/y_{41}$. Thus, remove x_4 from the basis.We move to the basis $\{x_1, x_3\}$ and transform the tableau into the canonical form for this basis:

$$\left[\begin{array}{c|cccc|c} x_1 & 1 & y_{12} & 0 & y_{14} & b'_1 \\ x_3 & 0 & y_{32} & 1 & y_{34} & b'_2 \\ \hline -z & c'_1 & c'_2 & c'_3 & c'_4 & 3 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_1 & 1 & 0 & 0 & 1 & 1 \\ x_3 & 0 & 1 & 1 & -1 & 1 \\ \hline -z & 0 & -2 & 0 & 3 & 3 \end{array}\right]$$

Here, the reduced cost of x_2 is -2, and of x_4 is 3. Thus, x_2 enters the basis.Now x_3 leaves the basis because $y_{12} = 0$ but $y_{32} > 0$.

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Choose x_1 to enter the basis (x_1 has the reduced cost -3 and x_2 has the reduced costs -2).Now $b_1/y_{31} = 2/1 > 1/1 = b_2/y_{41}$. Thus, remove x_4 from the basis.We move to the basis $\{x_1, x_3\}$ and transform the tableau into the canonical form for this basis:

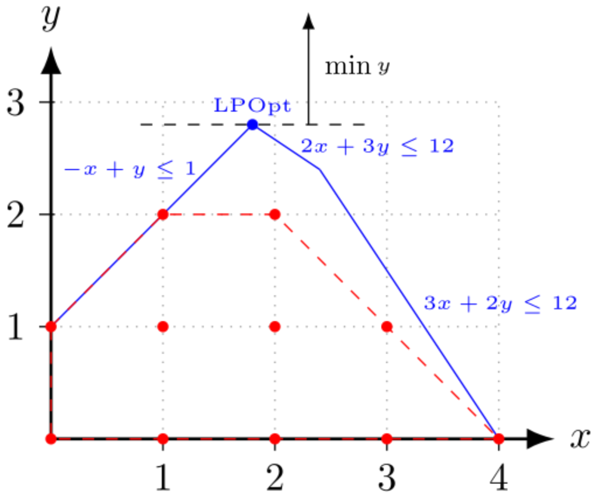
$$\left[\begin{array}{c|cccc|c} x_1 & 1 & y_{12} & 0 & y_{14} & b'_1 \\ x_3 & 0 & y_{32} & 1 & y_{34} & b'_2 \\ \hline -z & c'_1 & c'_2 & c'_3 & c'_4 & 3 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_1 & 1 & 0 & 0 & 1 & 1 \\ x_3 & 0 & 1 & 1 & -1 & 1 \\ \hline -z & 0 & -2 & 0 & 3 & 3 \end{array}\right]$$

Here, the reduced cost of x_2 is -2, and of x_4 is 3. Thus, x_2 enters the basis.Now x_3 leaves the basis because $y_{12} = 0$ but $y_{32} > 0$.We move to the basis $\{x_1, x_2\}$ and transform the tableau into the canonical form:

$$\left[\begin{array}{c|cccc|c} x_1 & 1 & 0 & 0 & 1 & 1 \\ x_2 & 0 & 1 & 1 & -1 & 1 \\ \hline -z & 0 & 0 & 2 & 1 & 5 \end{array}\right]$$

Integer Linear Programming

Integer Linear Programming



ILP = LP + variables constrained to integer values

Integer Linear Programming

We consider several variants of integer programming:

- 0-1 integer linear programming
- Mixed 0-1 integer linear programming
- Integer linear programming
- Mixed integer linear programming

Integer Linear Programming

We consider several variants of integer programming:

- 0-1 integer linear programming
- Mixed 0-1 integer linear programming
- Integer linear programming
- Mixed integer linear programming

We consider the basic branch and bound algorithm.

Integer Linear Programming

We consider several variants of integer programming:

- 0-1 integer linear programming
- Mixed 0-1 integer linear programming
- Integer linear programming
- Mixed integer linear programming

We consider the basic branch and bound algorithm.

We also consider a cutting-plane method for integer programming.

Integer Linear Programming

We consider several variants of integer programming:

- 0-1 integer linear programming
- Mixed 0-1 integer linear programming
- Integer linear programming
- Mixed integer linear programming

We consider the basic branch and bound algorithm.

We also consider a cutting-plane method for integer programming.

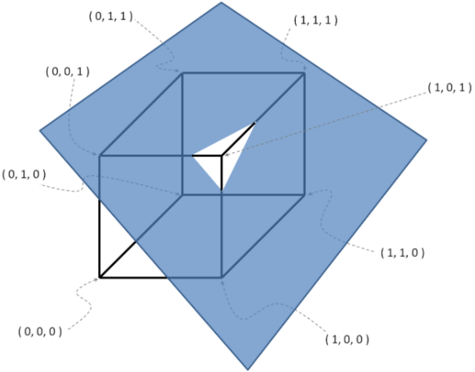
Integer linear programming is a huge subject; we shall only scratch its surface slightly.

0-1 Integer Linear Programming

Let us start with a special case where variables are constrained to values from $\{0, 1\}$.

0-1 integer linear program (0-1 ILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax \leq b \\ & x_i \in \{0, 1\} \end{aligned}$$



0-1 Integer Linear Programming

Consider the following example:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & a^\top x \leq b \\ & x \geq 0 \\ & x_i \in \{0, 1\} \end{aligned}$$

Here $c, a \in \mathbb{R}^n$ and $b \in \mathbb{R}$.

Do you recognize the problem?

0-1 Integer Linear Programming

Consider the following example:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & a^\top x \leq b \\ & x \geq 0 \\ & x_i \in \{0, 1\} \end{aligned}$$

Here $c, a \in \mathbb{R}^n$ and $b \in \mathbb{R}$.

Do you recognize the problem? It is the 0-1 knapsack problem.

0-1 Integer Linear Programming

Consider the following example:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & a^\top x \leq b \\ & x \geq 0 \\ & x_i \in \{0, 1\} \end{aligned}$$

Here $c, a \in \mathbb{R}^n$ and $b \in \mathbb{R}$.

Do you recognize the problem? It is the 0-1 knapsack problem.

Theorem 23

Finding $x \in \{0, 1\}^n$ satisfying the constraints of a given 0-1 integer linear program is NP-complete.

It is one of Karp's 21 NP-complete problems.

0-1 Mixed Integer Linear Programming

0-1 mixed integer linear program (0-1 MILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x_i \in \{0,1\} \text{ for } x_i \in \mathcal{D} \end{aligned}$$

Here $\mathcal{D} \subseteq \{x_1, \ldots, x_n\}$ is a set of *binary variables*.

0-1 Mixed Integer Linear Programming

0-1 mixed integer linear program (0-1 MILP) is

$$
\begin{aligned}
\text{minimize} \quad & c^\top x \\
\text{subject to} \quad & Ax = b \\
& x \geq 0 \\
& x_i \in \{0, 1\} \text{ for } x_i \in \mathcal{D}
\end{aligned}
$$

Here $\mathcal{D} \subseteq \{x_1, \ldots, x_n\}$ is a set of *binary variables*.

The problem is NP-hard; the simplex algorithm cannot be used directly.

0-1 Mixed Integer Linear Programming

0-1 mixed integer linear program (0-1 MILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x_i \in \{0,1\} \text{ for } x_i \in \mathcal{D} \end{aligned}$$

Here $\mathcal{D} \subseteq \{x_1, \ldots, x_n\}$ is a set of *binary variables*.

The problem is NP-hard; the simplex algorithm cannot be used directly.

The problem can be solved by searching for possible values 0 and 1 in the binary variables and solving the linear programs with binary variables fixed to concrete values.

0-1 Mixed Integer Linear Programming

0-1 mixed integer linear program (0-1 MILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x_i \in \{0, 1\} \text{ for } x_i \in \mathcal{D} \end{aligned}$$

Here $\mathcal{D} \subseteq \{x_1, \ldots, x_n\}$ is a set of *binary variables*.

The problem is NP-hard; the simplex algorithm cannot be used directly.

The problem can be solved by searching for possible values 0 and 1 in the binary variables and solving the linear programs with binary variables fixed to concrete values.

An exhaustive search through all possible binary assignments would be infeasible for many variables.

Usually, a sequential search that fixes only some of the binary variables and leaves the rest unrestricted to 0 or 1 is used.

Notation

In what follows, *LP relaxation* is the linear program obtained from 0-1 MILP by removing the constraints $x_i \in \{0, 1\}$ for $x_i \in \mathcal{D}$ and adding constraints $x_i \geq 0$ and $x \leq 1$ for all $x_i \in \mathcal{D}$.

Notation

In what follows, *LP relaxation* is the linear program obtained from 0-1 MILP by removing the constraints $x_i \in \{0, 1\}$ for $x_i \in \mathcal{D}$ and adding constraints $x_i \geq 0$ and $x \leq 1$ for all $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the 0-1 MILP constraints. Initialized with the undefined symbol $\perp$.

Notation

In what follows, *LP relaxation* is the linear program obtained from 0-1 MILP by removing the constraints $x_i \in \{0, 1\}$ for $x_i \in \mathcal{D}$ and adding constraints $x_i \geq 0$ and $x \leq 1$ for all $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the 0-1 MILP constraints. Initialized with the undefined symbol $\perp$.

Assume a global variable f^*, keeping the value of the best solution satisfying the 0-1 MILP constraints. Initialize with $f^* = \infty$.

Notation

In what follows, *LP relaxation* is the linear program obtained from 0-1 MILP by removing the constraints $x_i \in \{0, 1\}$ for $x_i \in \mathcal{D}$ and adding constraints $x_i \geq 0$ and $x \leq 1$ for all $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the 0-1 MILP constraints. Initialized with the undefined symbol $\perp$.

Assume a global variable f^*, keeping the value of the best solution satisfying the 0-1 MILP constraints. Initialize with $f^* = \infty$.

Keep a pool of 0-1 MILP problems $\mathcal{P}$ initialized with $\mathcal{P} = \{P\}$ where P is the original 0-1 MILP to be solved.

Algorithm 17 Branch and Bound (Non-Deterministic)

repeat
  Choose $P \in \mathcal{P}$
  if LP relaxation of P is feasible **then**
    Find a solution x of the LP relaxation of P
    if $c^\top x < f^*$ **then**
      if $x_i \in \{0, 1\}$ for all $x_i \in \mathcal{D}$ **then**
        $x^* \leftarrow x$
        $f^* \leftarrow c^\top x$
      else
        Choose $x_i \in \mathcal{D}$ such that $x_i \notin \{0, 1\}$
        Generate LP P_0 by adding $x_i = 0$ to P
        Generate LP P_1 by adding $x_i = 1$ to P
        Add P_0 and P_1 to $\mathcal{P}$.
      end if
    end if
  end if
  $\mathcal{P} \leftarrow \mathcal{P} \setminus \{P\}$
until $\mathcal{P} = \emptyset$

Strategies

There are many possible strategies for choosing the problem to be solved next:

- DFS, BFS, etc.
- heuristics using solutions to the relaxations

Strategies

There are many possible strategies for choosing the problem to be solved next:

- DFS, BFS, etc.
- heuristics using solutions to the relaxations

There are heuristics for choosing the variable to be bounded:

- Simplest one: Choose x_i which maximizes $\min\{x_i, 1 - x_i\}$
- Look ahead to the relaxations of the possible subdivisions

Strategies

There are many possible strategies for choosing the problem to be solved next:

- DFS, BFS, etc.
- heuristics using solutions to the relaxations

There are heuristics for choosing the variable to be bounded:

- Simplest one: Choose x_i which maximizes $\min\{x_i, 1 - x_i\}$
- Look ahead to the relaxations of the possible subdivisions

The solutions to the LP relaxations can be reused. Some methods (dual simplex) exploit that we are just adding a single constraint $x_i = 0$ or $x_i = 1$.

Strategies

There are many possible strategies for choosing the problem to be solved next:

- DFS, BFS, etc.
- heuristics using solutions to the relaxations

There are heuristics for choosing the variable to be bounded:

- Simplest one: Choose x_i which maximizes $\min\{x_i, 1 - x_i\}$
- Look ahead to the relaxations of the possible subdivisions

The solutions to the LP relaxations can be reused. Some methods (dual simplex) exploit that we are just adding a single constraint $x_i = 0$ or $x_i = 1$.

The procedure may be stopped when we find a solution x, which gives a small enough value of the objective.

(Mixed) Integer Programming

Integer linear program (ILP) is

$$
\begin{aligned}
\text{minimize} \quad & c^\top x \\
\text{subject to} \quad & Ax \leq b \\
& x \geq 0 \\
& x \in \mathbb{Z}^n
\end{aligned}
$$

(Mixed) Integer Programming

Integer linear program (ILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \\ & x \in \mathbb{Z}^n \end{aligned}$$

Mixed integer linear program (MILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x_i \in \mathbb{Z} \text{ for } x_i \in \mathcal{D} \end{aligned}$$

Here $\mathcal{D} \subseteq \{x_1, \ldots, x_n\}$ is a set of integer variables.

(Mixed) Integer Programming

Integer linear program (ILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \\ & x \in \mathbb{Z}^n \end{aligned}$$

Mixed integer linear program (MILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x_i \in \mathbb{Z} \text{ for } x_i \in \mathcal{D} \end{aligned}$$

Here $\mathcal{D} \subseteq \{x_1, \ldots, x_n\}$ is a set of integer variables.

We may use a similar branch and bound approach as for the binary variables. The problem is that now, each integer variable has an infinite domain.

Notation

In what follows, *LP relaxation* is the linear program obtained from MILP by removing the constraints $x_i \in \mathbb{Z}$ for $x_i \in \mathcal{D}$.

Notation

In what follows, *LP relaxation* is the linear program obtained from MILP by removing the constraints $x_i \in \mathbb{Z}$ for $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the MILP constraints. Initialized with the undefined symbol $\perp$.

Notation

In what follows, *LP relaxation* is the linear program obtained from MILP by removing the constraints $x_i \in \mathbb{Z}$ for $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the MILP constraints. Initialized with the undefined symbol $\perp$.

Assume a global variable f^*, keeping the value of the best solution satisfying the MILP constraints. Initialize with $f^* = \infty$.

Notation

In what follows, *LP relaxation* is the linear program obtained from MILP by removing the constraints $x_i \in \mathbb{Z}$ for $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the MILP constraints. Initialized with the undefined symbol $\perp$.

Assume a global variable f^*, keeping the value of the best solution satisfying the MILP constraints. Initialize with $f^* = \infty$.

Keep a pool of MILP problems $\mathcal{P}$ initialized with $\mathcal{P} = \{P\}$ where P is the original MILP to be solved.

Notation

In what follows, *LP relaxation* is the linear program obtained from MILP by removing the constraints $x_i \in \mathbb{Z}$ for $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the MILP constraints. Initialized with the undefined symbol $\perp$.

Assume a global variable f^*, keeping the value of the best solution satisfying the MILP constraints. Initialize with $f^* = \infty$.

Keep a pool of MILP problems $\mathcal{P}$ initialized with $\mathcal{P} = \{P\}$ where P is the original MILP to be solved.

In what follows, we temporarily cease to abuse notation and use $\bar{x}$ to denote the vector of values of the vector of variables x. Then $\bar{x}_i$ will denote the concrete value of the variable x_i.

Algorithm 18 Branch and Bound (Non-Deterministic)

repeat
  Choose $P \in \mathcal{P}$
  if LP relaxation of P is feasible **then**
    Find a solution $\bar{x}$ of the LP relaxation of P
    if $c^\top \bar{x} < f^*$ **then**
      if $\bar{x}_i \in \mathbb{Z}$ for all $x_i \in \mathcal{D}$ **then**
        $x^* \leftarrow \bar{x}$
        $f^* \leftarrow c^\top \bar{x}$
      else
        Choose $x_i \in \mathcal{D}$ such that $\bar{x}_i \notin \mathbb{Z}$
        Generate LP P_- by adding $x_i \leq \lfloor \bar{x}_i \rfloor$ to P
        Generate LP P_+ by adding $x_i \geq \lceil \bar{x}_i \rceil$ to P
        Add P_0 and P_1 to $\mathcal{P}$.
      end if
    end if
  end if
  $\mathcal{P} \leftarrow \mathcal{P} \setminus \{P\}$
until $\mathcal{P} = \emptyset$

Example

Consider the following MILP P:

$$\begin{array}{ll} \text{minimize} & -x_1 - 2x_2 - 3x_3 - 1.5x_4 \\ \text{subject to} & x_1 + x_2 + 2x_3 + 2x_4 \leq 10 \\ & 7x_1 + 8x_2 + 5x_3 + x_4 = 31.5 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{array}$$

and assume $\mathcal{D} = \{x_1, x_2, x_3\}$. That is, $x_1, x_2, x_3 \in \mathbb{Z}$.

Example

Consider the following MILP P:

$$\begin{array}{ll} \text{minimize} & -x_1 - 2x_2 - 3x_3 - 1.5x_4 \\ \text{subject to} & x_1 + x_2 + 2x_3 + 2x_4 \leq 10 \\ & 7x_1 + 8x_2 + 5x_3 + x_4 = 31.5 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{array}$$

and assume $\mathcal{D} = \{x_1, x_2, x_3\}$. That is, $x_1, x_2, x_3 \in \mathbb{Z}$.

The algorithm starts with $\mathcal{P} = \{P\}$ and $x^* = \perp$ and $f^* = \infty$.

Example

Consider the following MILP P:

$$\begin{aligned} \text{minimize} \quad & -x_1 - 2x_2 - 3x_3 - 1.5x_4 \\ \text{subject to} \quad & x_1 + x_2 + 2x_3 + 2x_4 \leq 10 \\ & 7x_1 + 8x_2 + 5x_3 + x_4 = 31.5 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

and assume $\mathcal{D} = \{x_1, x_2, x_3\}$. That is, $x_1, x_2, x_3 \in \mathbb{Z}$.

The algorithm starts with $\mathcal{P} = \{P\}$ and $x^* = \perp$ and $f^* = \infty$.

The solution to the LP relaxation of P is:

$$x = [0, 1.1818, 4.4091, 0], \quad \text{the objective value is } -15.59$$

Example

Consider the following MILP P:

$$\begin{array}{ll} \text{minimize} & -x_1 - 2x_2 - 3x_3 - 1.5x_4 \\ \text{subject to} & x_1 + x_2 + 2x_3 + 2x_4 \leq 10 \\ & 7x_1 + 8x_2 + 5x_3 + x_4 = 31.5 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{array}$$

and assume $\mathcal{D} = \{x_1, x_2, x_3\}$. That is, $x_1, x_2, x_3 \in \mathbb{Z}$.

The algorithm starts with $\mathcal{P} = \{P\}$ and $x^* = \perp$ and $f^* = \infty$.

The solution to the LP relaxation of P is:

$$x = [0, 1.1818, 4.4091, 0], \quad \text{the objective value is } -15.59$$

Let us choose x_3. So, consider two programs:

- P_- where we add $x_3 \leq 4$ to P
- P_+ where we add $x_3 \geq 5$ to P

Now $\mathcal{P} = \{P_-, P_+\}$.

Consider first P_+.

P_+ is P with the added constraint $x_3 \geq 5$. The LP relaxation of P_+ is infeasible. We get $\mathcal{P} = \{P_-\}$.

Consider first P_+.

P_+ is P with the added constraint $x_3 \geq 5$. The LP relaxation of P_+ is infeasible. We get $\mathcal{P} = \{P_-\}$.

P_- is P with the additional constraint $x_3 \leq 4$.

Consider first P_+.

P_+ is P with the added constraint $x_3 \geq 5$. The LP relaxation of P_+ is infeasible. We get $\mathcal{P} = \{P_-\}$.

P_- is P with the additional constraint $x_3 \leq 4$.

The LP relaxation of P_- solves to

$$\bar{x} = [0, 1.4, 4, 0.3], \qquad \text{the objective value is } -15.25$$

Consider first P_+.

P_+ is P with the added constraint $x_3 \geq 5$. The LP relaxation of P_+ is infeasible. We get $\mathcal{P} = \{P_-\}$.

P_- is P with the additional constraint $x_3 \leq 4$.

The LP relaxation of P_- solves to

$$\bar{x} = [0, 1.4, 4, 0.3], \qquad \text{the objective value is } -15.25$$

We still have $f^* = \infty$ so we split P_- by constraining x_2:

- P_{--} is obtained from P_- by adding $x_2 \leq 1$
- P_{-+} is obtained from P_- by adding $x_2 \geq 2$

and we continue with $\mathcal{P} = \{P_{--}, P_{-+}\}$.

Consider first P_+.

P_+ is P with the added constraint $x_3 \geq 5$. The LP relaxation of P_+ is infeasible. We get $\mathcal{P} = \{P_-\}$.

P_- is P with the additional constraint $x_3 \leq 4$.

The LP relaxation of P_- solves to

$$\bar{x} = [0, 1.4, 4, 0.3], \quad \text{the objective value is } -15.25$$

We still have $f^* = \infty$ so we split P_- by constraining x_2:

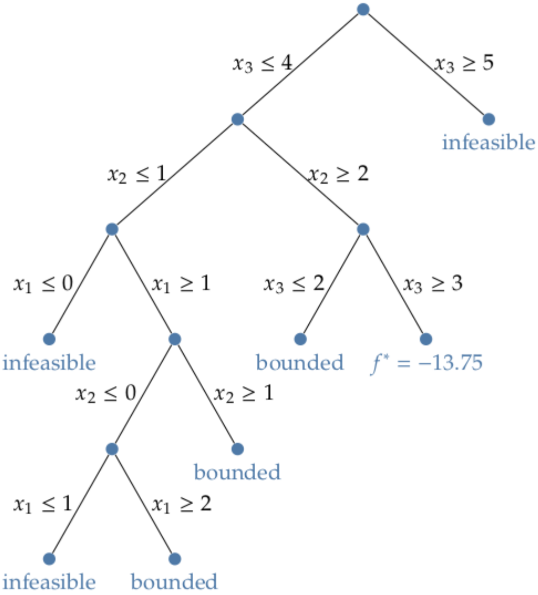
- P_{--} is obtained from P_- by adding $x_2 \leq 1$
- P_{-+} is obtained from P_- by adding $x_2 \geq 2$

and we continue with $\mathcal{P} = \{P_{--}, P_{-+}\}$.

Adding one more constraint $x_3 \geq 3$ to P_{-+} would yield a MILP solution $(0, 2, 3, 0.5)$ to the LP relaxation with the objective value equal to -13.75.

The algorithm assigns $f^* = -13.75$ and $x^* = (0, 2, 3, 0.5)$.

The remaining search always leads either to an infeasible relaxation or to a relaxation with an objective value worse than f^*.



The final solution: $x^* = (0, 2, 3, 0.5)$ and $f^* = -13.75$.

Cutting Planes

Removing Non-Integer Solutions

The basic branch and bound method generates two new problems in every step.

Removing Non-Integer Solutions

The basic branch and bound method generates two new problems in every step.

Another strategy might be to successively cut out non-integer optimal solutions and preserve the integer ones until an integer optimal solution is computed by the LP relaxation

Removing Non-Integer Solutions

The basic branch and bound method generates two new problems in every step.

Another strategy might be to successively cut out non-integer optimal solutions and preserve the integer ones until an integer optimal solution is computed by the LP relaxation

We consider a concrete method for obtaining such cuts from the ILP constraints called *Gomory cuts*.

Gomory Cuts

Consider an ILP and transform it into a MILP by adding slack variables:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x \in \mathbb{Z} \text{ for } x \in \mathcal{D} \end{aligned}$$

Here, $\mathcal{D}$ contains the original (i.e., non-slack) variables of the ILP.

Gomory Cuts

Consider an ILP and transform it into a MILP by adding slack variables:

$$
\begin{aligned}
\text{minimize} \quad & c^\top x \\
\text{subject to} \quad & Ax = b \\
& x \geq 0 \\
& x \in \mathbb{Z} \text{ for } x \in \mathcal{D}
\end{aligned}
$$

Here, $\mathcal{D}$ contains the original (i.e., non-slack) variables of the ILP.

We demand the integer solution only for the original $\mathcal{D}$ variables.

Gomory Cuts

Consider an ILP and transform it into a MILP by adding slack variables:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x \in \mathbb{Z} \text{ for } x \in \mathcal{D} \end{aligned}$$

Here, $\mathcal{D}$ contains the original (i.e., non-slack) variables of the ILP.

We demand the integer solution only for the original $\mathcal{D}$ variables.

However, one can prove that if all constants in the ILP are integer, then there is an optimal solution where all variables (including the slacks) are integer-valued.

Gomory Cuts

Let $A = (u_1 \ \ldots, u_n)$, the basis $\{x_1, \ldots, x_n\}$, $B = (u_1 \ \ldots, u_m)$.

Gomory Cuts

Let $A = (u_1 \ldots, u_n)$, the basis $\{x_1, \ldots, x_n\}$, $B = (u_1 \ldots, u_m)$.

Consider the canonical tableau for B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

The $-z$ row is omitted as it is unnecessary for the discussion.

$$u_k = B(y_{1k}, \ldots, y_{mk})^\top \text{ for } k = 1, \ldots, n \text{ and } b' = B^{-1}b$$

Gomory Cuts

Let $A = (u_1 \ldots, u_n)$, the basis $\{x_1, \ldots, x_n\}$, $B = (u_1 \ldots, u_m)$.

Consider the canonical tableau for B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

The $-z$ row is omitted as it is unnecessary for the discussion.

$$u_k = B(y_{1k}, \ldots, y_{mk})^\top \text{ for } k = 1, \ldots, n \text{ and } b' = B^{-1}b$$

Consider a basic solution $x = (b'_1, \ldots, b'_m, 0, \ldots, 0)$.

If all $b'_1, \ldots, b'_m$ are integers, then also x solves the ILP.

Otherwise, assume that b'_i is not an integer.

Gomory Cuts

From the tableau, we know that every feasible solution x satisfies:

$$x_i + y_{i(m+1)}x_{m+1} + \cdots + y_{in}x_n = b'_i$$

Gomory Cuts

From the tableau, we know that every feasible solution x satisfies:

$$x_i + y_{i(m+1)}x_{m+1} + \cdots + y_{in}x_n = b_i'$$

Then, x also satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq b_i'$$

Gomory Cuts

From the tableau, we know that every feasible solution x satisfies:

$$x_i + y_{i(m+1)}x_{m+1} + \cdots + y_{in}x_n = b'_i$$

Then, x also satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq b'_i$$

Moreover, any *integer feasible solution* x satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq \lfloor b'_i \rfloor$$

Gomory Cuts

From the tableau, we know that every feasible solution x satisfies:

$$x_i + y_{i(m+1)} x_{m+1} + \cdots + y_{in} x_n = b'_i$$

Then, x also satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq b'_i$$

Moreover, any *integer feasible solution* x satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq \lfloor b'_i \rfloor$$

But, subtracting the inequalities, integer feasible solutions x satisfy:

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor) x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor) x_n \geq b'_i - \lfloor b'_i \rfloor$$

Gomory Cuts

From the tableau, we know that every feasible solution x satisfies:

$$x_i + y_{i(m+1)} x_{m+1} + \cdots + y_{in} x_n = b_i'$$

Then, x also satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq b_i'$$

Moreover, any *integer feasible solution* x satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq \lfloor b_i' \rfloor$$

But, subtracting the inequalities, integer feasible solutions x satisfy:

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor) x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor) x_n \geq b_i' - \lfloor b_i' \rfloor$$

But note that the *basic feasible solution* $x = (b_1', \ldots, b_m', 0, \ldots, 0)$ *does not* satisfy the last inequality because $b_i' > \lfloor b_i' \rfloor$ and $x_{m+1} = \cdots = x_n = 0$.

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B.

Assume that the basic solution x w.r.t. B is non-integer.

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B.
Assume that the basic solution x w.r.t. B is non-integer.

Consider the canonical tableau for the basis B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B.
Assume that the basic solution x w.r.t. B is non-integer.

Consider the canonical tableau for the basis B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

Choose a non-integer component $x_i = b'_i$ of the basic feasible solution w.r.t. B

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B.
Assume that the basic solution x w.r.t. B is non-integer.

Consider the canonical tableau for the basis B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

Choose a non-integer component $x_i = b'_i$ of the basic feasible solution w.r.t. B and consider the constraint

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor)x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor)x_n \geq b'_i - \lfloor b'_i \rfloor$$

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B.
Assume that the basic solution x w.r.t. B is non-integer.

Consider the canonical tableau for the basis B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

Choose a non-integer component $x_i = b'_i$ of the basic feasible solution w.r.t. B and consider the constraint

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor)x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor)x_n \geq b'_i - \lfloor b'_i \rfloor$$

Transform the above inequality into equality by introducing a new variable x_{n+1} and obtain the following constraint (*Gomory cut*)

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor)x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor)x_n - x_{n+1} = b'_i - \lfloor b'_i \rfloor$$

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B.
Assume that the basic solution x w.r.t. B is non-integer.

Consider the canonical tableau for the basis B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

Choose a non-integer component $x_i = b'_i$ of the basic feasible solution w.r.t. B and consider the constraint

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor)x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor)x_n \geq b'_i - \lfloor b'_i \rfloor$$

Transform the above inequality into equality by introducing a new variable x_{n+1} and obtain the following constraint (*Gomory cut*)

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor)x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor)x_n - x_{n+1} = b'_i - \lfloor b'_i \rfloor$$

Add the Gomory cut and the constraint $x_{n+1} \geq 0$ to the program.

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B. Assume that the basic solution x w.r.t. B is non-integer.

Consider the canonical tableau for the basis B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

Choose a non-integer component $x_i = b'_i$ of the basic feasible solution w.r.t. B and consider the constraint

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor)x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor)x_n \geq b'_i - \lfloor b'_i \rfloor$$

Transform the above inequality into equality by introducing a new variable x_{n+1} and obtain the following constraint (*Gomory cut*)

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor)x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor)x_n - x_{n+1} = b'_i - \lfloor b'_i \rfloor$$

Add the Gomory cut and the constraint $x_{n+1} \geq 0$ to the program.

Repeat until an integer solution is reached.

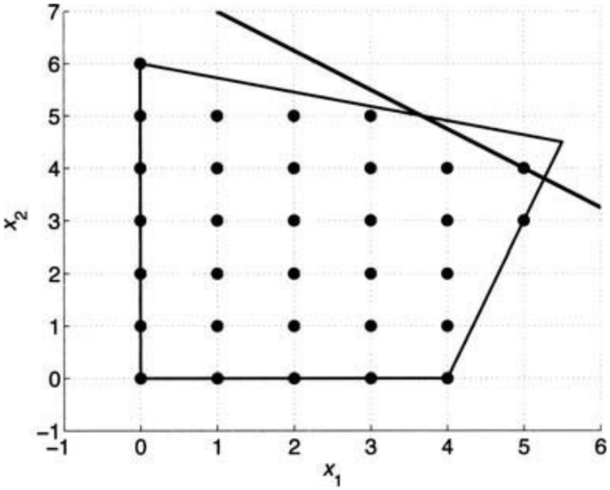
Example

Consider ILP:

$$\begin{array}{rl} \text{minimize} & -3x_1 - 4x_2 \\ \text{subject to} & 3x_1 - x_2 \leq 12 \\ & 3x_1 + 11x_2 \leq 66 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{array}$$

Adding slack variables x_3, x_4 we obtain the following MILP:

$$\begin{array}{rl} \text{minimize} & -3x_1 - 4x_2 \\ \text{subject to} & 3x_1 - x_2 + x_3 = 12 \\ & 3x_1 + 11x_2 + x_4 = 66 \\ & x_1, x_2, x_3, x_4 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{array}$$


7
6
5
4
3
2
1
0
−1
x_2
−1
0
1
2
3
4
5
6
x_1

We have

$$\begin{aligned} \text{minimize} \quad & -3x_1 - 4x_2 \\ \text{subject to} \quad & 3x_1 - x_2 + x_3 = 12 \\ & 3x_1 + 11x_2 + x_4 = 66 \\ & x_1, x_2, x_3, x_4 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{aligned}$$

We have

$$\begin{array}{rl} \text{minimize} & -3x_1 - 4x_2 \\ \text{subject to} & 3x_1 - x_2 + x_3 = 12 \\ & 3x_1 + 11x_2 + x_4 = 66 \\ & x_1, x_2, x_3, x_4 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{array}$$

An optimal basic solution to the LP relaxation is

$$\left(\frac{11}{2}, \frac{9}{2}, 0, 0\right)^\top$$

and the canonical tableau w.r.t. the basis $\{x_1, x_2\}$ is

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & b' \\ 1 & 0 & \frac{11}{36} & \frac{1}{36} & \frac{11}{2} \\ 0 & 1 & -\frac{1}{12} & \frac{1}{12} & \frac{9}{2} \end{pmatrix}$$

Let us introduce the Gomory cut corresponding to the variable x_1.

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & b' \\ 1 & 0 & \frac{11}{36} & \frac{1}{36} & \frac{11}{2} \\ 0 & 1 & -\frac{1}{12} & \frac{1}{12} & \frac{9}{2} \end{pmatrix}$$

Then

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor) x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor) x_n - x_{n+1} = b'_i - \lfloor b'_i \rfloor$$

with $i = 1$ and $m = 2$ turns into

$$\left(\frac{11}{36} - 0 \right) x_3 + \left(\frac{1}{36} - 0 \right) x_4 - x_5 = \frac{1}{2} \quad (= \frac{11}{2} - 5)$$

We add this constraint to our MILP.

$$\begin{array}{rl} \text{minimize} & -3x_1 - 4x_2 \\ \text{subject to} & 3x_1 - x_2 + x_3 = 12 \\ & 3x_1 + 11x_2 + x_4 = 66 \\ & \frac{11}{36}x_3 + \frac{1}{36}x_4 - x_5 = \frac{1}{2} \\ & x_1, x_2, x_3, x_4 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{array}$$

Solving the LP relaxation yields

$$\left(5, \frac{51}{11}, \frac{18}{11}, 0, 0\right)^\top$$

The canonical tableau for the solution is

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & b' \\ 1 & 0 & 0 & 0 & 1 & 5 \\ 0 & 1 & 0 & \frac{1}{11} & -\frac{3}{11} & \frac{51}{11} \\ 0 & 0 & 1 & \frac{1}{11} & -\frac{36}{11} & \frac{18}{11} \end{pmatrix}$$

Introduce the Gomory cut for x_2.

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & b' \\ 1 & 0 & 0 & 0 & 1 & 5 \\ 0 & 1 & 0 & \frac{1}{11} & -\frac{3}{11} & \frac{51}{11} \\ 0 & 0 & 1 & \frac{1}{11} & -\frac{36}{11} & \frac{18}{11} \end{pmatrix}$$

Then

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor) x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor) x_n - x_{n+1} = b'_i - \lfloor b'_i \rfloor$$

with $i = 2$ and $m = 3$ turns into

$$\left(\frac{1}{11} - 0\right) x_4 + \left(-\frac{3}{11} + \frac{11}{11}\right) x_5 - x_6 = \frac{7}{11} \quad (= \frac{51}{11} - \frac{44}{11})$$

We add this to our MILP.

$$
\begin{aligned}
\text{minimize} \quad & -3x_1 - 4x_2 \\
\text{subject to} \quad & 3x_1 - x_2 + x_3 = 12 \\
& 3x_1 + 11x_2 + x_4 = 66 \\
& \tfrac{11}{36}x_3 + \tfrac{1}{36}x_4 - x_5 = \tfrac{1}{2} \\
& \tfrac{1}{11}x_4 + \tfrac{8}{11}x_5 - x_6 = \tfrac{7}{11} \\
& x_1, x_2, x_3, x_4 \geq 0 \\
& x_1, x_2 \in \mathbb{Z}
\end{aligned}
$$

Once more the solution of the above is non-integer. However, introducing another Gomory cut (and a variable x_7) would yield a solution:

$$(5, 4, 1, 7, 0, 0, 0)^\top$$

Which gives the point $(x_1, x_2) = (5, 4)$ corresponding to the graphical solution.

Cutting Planes Technique

The method based on Gomory cuts was one of the first solutions to the integer linear programming problem with proven convergence (in the 1950s).

Cutting Planes Technique

The method based on Gomory cuts was one of the first solutions to the integer linear programming problem with proven convergence (in the 1950s).

The convergence rate is unsatisfactory in practice; many more methods have been devised based on algebraic principles (combinations of inequalities and rounding), geometry, etc.

Cutting Planes Technique

The method based on Gomory cuts was one of the first solutions to the integer linear programming problem with proven convergence (in the 1950s).

The convergence rate is unsatisfactory in practice; many more methods have been devised based on algebraic principles (combinations of inequalities and rounding), geometry, etc.

Cutting planes are also used in other non-linear, non-smooth optimization methods.

Cutting Planes Technique

The method based on Gomory cuts was one of the first solutions to the integer linear programming problem with proven convergence (in the 1950s).

The convergence rate is unsatisfactory in practice; many more methods have been devised based on algebraic principles (combinations of inequalities and rounding), geometry, etc.

Cutting planes are also used in other non-linear, non-smooth optimization methods.

Most importantly, cutting plane techniques are combined with branch and bound methods. The constraints are introduced before branching to eliminate some solutions before the split.

The resulting method is called *branch and cut*.

Summary of Integer Linear Programming

We have considered:

- Linear Programming (LP)

  Linear objective and constraints.

Summary of Integer Linear Programming

We have considered:

- Linear Programming (LP)
  Linear objective and constraints.
- 0-1 Integer Linear Programming (0-1 ILP)
  Linear objective and constraints. All variables restricted to $\{0, 1\}$.

Summary of Integer Linear Programming

We have considered:

- Linear Programming (LP)
  Linear objective and constraints.
- 0-1 Integer Linear Programming (0-1 ILP)
  Linear objective and constraints. All variables restricted to $\{0, 1\}$.
- 0-1 Mixed Integer Programming (0-1 MILP)
  Linear objective and constraints. Some variables restricted to $\{0, 1\}$.

Summary of Integer Linear Programming

We have considered:

- Linear Programming (LP)
  Linear objective and constraints.
- 0-1 Integer Linear Programming (0-1 ILP)
  Linear objective and constraints. All variables restricted to $\{0, 1\}$.
- 0-1 Mixed Integer Programming (0-1 MILP)
  Linear objective and constraints. Some variables restricted to $\{0, 1\}$.
- Integer Linear Programming (ILP)
  Linear objective and constraints. All variables restricted to $\mathbb{Z}$.

Summary of Integer Linear Programming

We have considered:

- Linear Programming (LP)
  Linear objective and constraints.
- 0-1 Integer Linear Programming (0-1 ILP)
  Linear objective and constraints. All variables restricted to $\{0, 1\}$.
- 0-1 Mixed Integer Programming (0-1 MILP)
  Linear objective and constraints. Some variables restricted to $\{0, 1\}$.
- Integer Linear Programming (ILP)
  Linear objective and constraints. All variables restricted to $\mathbb{Z}$.
- Mixed Integer Linear Programming (MILP)
  Linear objective and constraints. Some variables restricted to $\mathbb{Z}$.

Summary of Integer Linear Programming

Complexity:

- Even the 0-1 Integer Linear Programming is NP-hard.

  Linear programming is in $\mathbb{P}$-time.

Summary of Integer Linear Programming

Complexity:

- Even the 0-1 Integer Linear Programming is NP-hard.
  Linear programming is in $\mathbb{P}$-time.

Algorithms:

- Branch and Bound
  - 0-1 MILP: Search through possible assignments of 0 and 1 to some discrete variables while solving the LP relaxations
    Branching with the choice of 0/1 values of variables, bounding with a solution found so far.

Summary of Integer Linear Programming

Complexity:

- Even the 0-1 Integer Linear Programming is NP-hard.
  Linear programming is in $\mathbb{P}$-time.

Algorithms:

- Branch and Bound
  - 0-1 MILP: Search through possible assignments of 0 and 1 to some discrete variables while solving the LP relaxations
    Branching with the choice of 0/1 values of variables, bounding with a solution found so far.
  - MILP: Solve LP relaxation, use non-integer values of the solution to introduce constraints, removing such values from the solution.

Summary of Integer Linear Programming

Complexity:

- Even the 0-1 Integer Linear Programming is NP-hard.
  Linear programming is in $\mathbb{P}$-time.

Algorithms:

- Branch and Bound
  - 0-1 MILP: Search through possible assignments of 0 and 1 to some discrete variables while solving the LP relaxations
    Branching with the choice of 0/1 values of variables, bounding with a solution found so far.
  - MILP: Solve LP relaxation, use non-integer values of the solution to introduce constraints, removing such values from the solution.
- Cutting planes
  - Sequentially cut out portions of the LP relaxation feasible space by introducing cuts based on solutions of LP relaxations.

Summary of Integer Linear Programming

Complexity:

- Even the 0-1 Integer Linear Programming is NP-hard.

  Linear programming is in $\mathbb{P}$-time.

Algorithms:

- Branch and Bound
  - 0-1 MILP: Search through possible assignments of 0 and 1 to some discrete variables while solving the LP relaxations

    Branching with the choice of 0/1 values of variables, bounding with a solution found so far.
  - MILP: Solve LP relaxation, use non-integer values of the solution to introduce constraints, removing such values from the solution.
- Cutting planes
  - Sequentially cut out portions of the LP relaxation feasible space by introducing cuts based on solutions of LP relaxations.
  - Does not branch but is usually combined with branch and bound (branch and cut).

Gradient-Free Optimization

Gradient-Free Methods

So far, we have explored problems where the objective f and the constraint functions h_j, g_i are known and (at least) differentiable.

Gradient-Free Methods

So far, we have explored problems where the objective f and the constraint functions h_j, g_i are known and (at least) differentiable.

What if the functions are just black boxes that can be evaluated but nothing else?

Gradient-Free Methods

So far, we have explored problems where the objective f and the constraint functions h_j, g_i are known and (at least) differentiable.

What if the functions are just black boxes that can be evaluated but nothing else?

What if the evaluation itself is costly?

Example: GPU parameters fine-tunning:

- Tens of parameters.
- The objective is to execute GPU software as efficiently as possible (tested by execution of a benchmark software suite)
- Evaluation of the objective function = Execution of a benchmark software suite
- How do we optimize the parameters?

Nothing is (possibly) differentiable here. Small changes in the parameters may give wildly different results.

There are many methods for such optimization. Most of them, of course, are without any convergence and efficiency guarantees.

Gradient-Free Methods Zoo

	Search		Algorithm		Function evaluation		Stochas-ticity	
	Local	Global	Mathematical	Heuristic	Direct	Surrogate	Deterministic	Stochastic
Nelder–Mead	•			•	•		•	
GPS		•	•		•		•	
MADS		•	•		•			•
Trust region	•		•			•	•	
Implicit filtering	•		•			•	•	
DIRECT		•	•		•		•	
MCS		•	•		•		•	
EGO		•	•			•	•	
Hit and run		•		•	•			•
Evolutionary		•		•	•			•

For more details see "Engineering Design Optimization" by Joaquim R. R. A. Martins and Andrew Ning

Evolutionary

"Evolutionary algorithms are inspired by processes that occur in nature or society. There is a plethora of evolutionary algorithms in the literature, thanks to the fertile imagination of the research community and a never-ending supply of phenomena for inspiration."

Evolutionary

"Evolutionary algorithms are inspired by processes that occur in nature or society. There is a plethora of evolutionary algorithms in the literature, thanks to the fertile imagination of the research community and a never-ending supply of phenomena for inspiration."

ant colony optimization, bee colony algorithm, fish swarm, artificial flora optimization algorithm, bacterial foraging optimization, bat algorithm, big bang–big crunch algorithm, biogeography-based optimization, bird mating optimizer, cat swarm, cockroach swarm, cuckoo search, design by shopping paradigm, dolphin echolocation algorithm, elephant herding optimization, firefly algorithm, flower pollination algorithm, fruit fly optimization algorithm, galactic swarm optimization, gray wolf optimizer, grenade explosion method, harmony search algorithm, hummingbird optimization algorithm, hybrid glowworm swarm optimization algorithm, imperialist competitive algorithm, intelligent water drops, invasive weed optimization, mine bomb algorithm, monarch butterfly optimization, moth-flame optimization algorithm, penguin search optimization algorithm, quantum-behaved particle swarm optimization, salp swarm algorithm, teaching–learning-based optimization, whale optimization algorithm, and water cycle algorithm, ...

Two Methods

To appreciate the gradient-free approaches, we shall (rather arbitrarily) concentrate on two methods:

- Nelder-Mead
- Particle Swarm Optimization

Both methods are somehow biologically motivated.

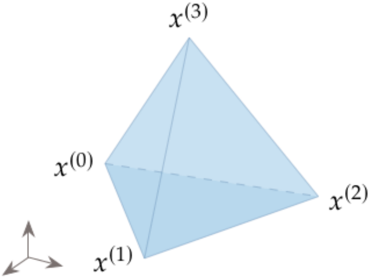
We consider the unconstrained optimization. That is, assume an objective function $f : \mathbb{R}^n \to \mathbb{R}$.

Nelder-Mead

The Nelder-Mead algorithm is based on a *simplex* defined by a set of $n + 1$ points in $\mathbb{R}^n$:

$$X = \left\{x^{(0)}, x^{(1)}, \ldots, x^{(n)}\right\} \subseteq \mathbb{R}^n$$

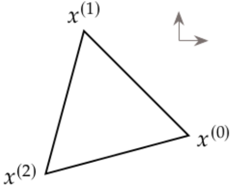
In two dimensions, the simplex is a triangle, and in three dimensions, it becomes a tetrahedron



A minimizer is approximated by a simplex node with a minimum value of f. The simplex changes in every step.

Nelder-Mead

Initially, $n+1$ nodes of the simplex need to be chosen: Typically, equal-length of edges and $x^{(0)}$ will be our starting point x_0.



$$x^{(i)} = x^{(0)} + s^{(i)},$$

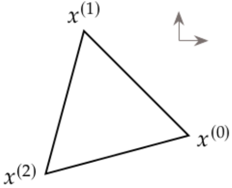
where $s^{(i)}$ is a vector whose components j are defined by

$$s_j^{(i)} = \begin{cases} \frac{L}{n\sqrt{2}}(\sqrt{n+1}-1) + \frac{L}{\sqrt{2}}, & \text{if } j = i \\ \frac{L}{n\sqrt{2}}(\sqrt{n+1}-1), & \text{if } j \neq i. \end{cases}$$

Here, L is the length of each side.

Nelder-Mead

Initially, $n+1$ nodes of the simplex need to be chosen: Typically, equal-length of edges and $x^{(0)}$ will be our starting point x_0.



$$x^{(i)} = x^{(0)} + s^{(i)},$$

where $s^{(i)}$ is a vector whose components j are defined by

$$s_j^{(i)} = \begin{cases} \frac{L}{n\sqrt{2}}(\sqrt{n+1}-1) + \frac{L}{\sqrt{2}}, & \text{if } j = i \\ \frac{L}{n\sqrt{2}}(\sqrt{n+1}-1), & \text{if } j \neq i. \end{cases}$$

Here, L is the length of each side.

Nelder-Mead method proceeds by modifying the simplex so that the values of f in the vertices (hopefully) decrease.

Nelder-Mead

The Nelder-Mead algorithm performs five main operations on the simplex to create a new one: *reflection*, *expansion*, *outside contraction*, *inside contraction*, and *shrinking*.

Nelder-Mead

The Nelder-Mead algorithm performs five main operations on the simplex to create a new one: *reflection, expansion, outside contraction, inside contraction, and shrinking*.

Except for shrinking, each operation generates a new point,

$$x = x_c + \alpha \left(x_c - x^{(n)} \right),$$

Here $\alpha \in \mathbb{R}$ and x_c is the centroid of all the points except for the worst one, that is, assuming $x^{(n)}$ maximizes f among the nodes

$$x_c = \frac{1}{n} \sum_{i=0}^{n-1} x^{(i)}$$

Nelder-Mead

The Nelder-Mead algorithm performs five main operations on the simplex to create a new one: *reflection, expansion, outside contraction, inside contraction, and shrinking*.

Except for shrinking, each operation generates a new point,

$$x = x_c + \alpha \left(x_c - x^{(n)} \right),$$

Here $\alpha \in \mathbb{R}$ and x_c is the centroid of all the points except for the worst one, that is, assuming $x^{(n)}$ maximizes f among the nodes

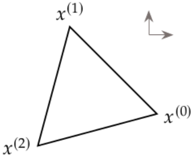
$$x_c = \frac{1}{n} \sum_{i=0}^{n-1} x^{(i)}$$

This generates a new point along the line that connects the worst point, $x^{(n)}$, and the centroid of the remaining points, x_c.

This direction can be seen as a possible descent direction.

Nelder-Mead Algorithm

1. Start with a simplex $x^{(0)}, \ldots, x^{(n)}$



Assume an order of these points:

$$f(x^{(0)}) \leq \ldots \leq f(x^{(n)})$$

2. Calculate the centroid

$$x_c = \frac{1}{n} \sum_{i=0}^{n-1} x^{(i)}$$

Nelder-Mead Algorithm (Reflection)

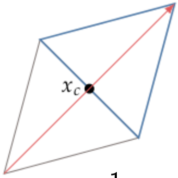
3. **Reflection** of $x^{(n)}$ over the centroid:

$$x_r = x_c + \alpha\left(x_c - x^{(n)}\right) \qquad \text{for } \alpha > 0$$

If $f(x^{(0)}) \leq f(x_r) < f(x^{(n-1)})$, then

Replace $x^{(n)}$ with x_r

Go to 1.



$\alpha = 1$

Now going further we know that either $f(x_r) < f(x^{(0)})$, or $f(x_r) \geq f(x^{(n-1)})$

Nelder-Mead Algorithm (Expansion)

4. **Expansion**

If $f(x_r) < f(x^{(0)})$, then

Compute

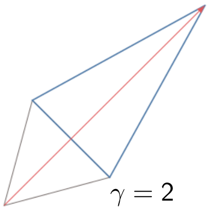
$$x_e = x_c + \gamma \left(x_c - x^{(n)} \right) \qquad \text{for } \gamma > 1$$

If $f(x_e) < f(x_r)$, then

Replace $x^{(n)}$ with x_e.

Else, replace $x^{(n)}$ with x_r.

Go to 1.



$\gamma = 2$

Now going further we know that $f(x_r) \geq f(x^{(n-1)})$

Nelder-Mead (Contraction)

5. **Contraction**

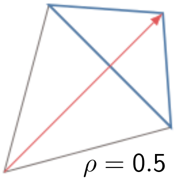
If $f(x_r) < f(x^{(n)})$, then compute outside contraction

$$x_{oc} = x_c + \rho\,(x_r - x_c) \qquad \text{for } 0 < \rho \leq 0.5$$

If $f(x_{oc}) < f(x_r)$, then

Replace $x^{(n)}$ with x_{oc}

Go to 1.



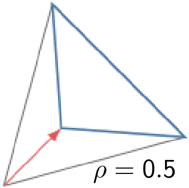
If $f(x_r) \geq f(x^{(n)})$, then compute inside contraction

$$x_{ic} = x_c + \rho\left(x^{(n)} - x_c\right) \qquad \text{for } 0 < \rho \leq 0.5$$

If $f(x_{ic}) < f(x^{(n)})$, then

Replace $x^{(n)}$ with x_{ic}

Go to 1.



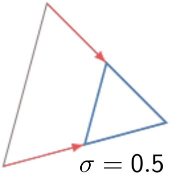
Nelder-Mead (Shrink)

6. **Shrink**

   Replace all points $x^{(k)}$ for $k > 0$ with

   $$x^{(k)} = x^{(k)} + \sigma(x^{(k)} - x^{(0)}) \quad \text{for } 0 < \sigma < 1$$

   Go to 1.


$\sigma = 0.5$

Nelder-Mead

The above procedure is repeated until convergence. This may be decided, e.g., based on the size of the simplex:

$$\Delta_x = \sum_{i=0}^{n-1} \left\| x^{(i)} - x^{(n)} \right\| < \epsilon$$

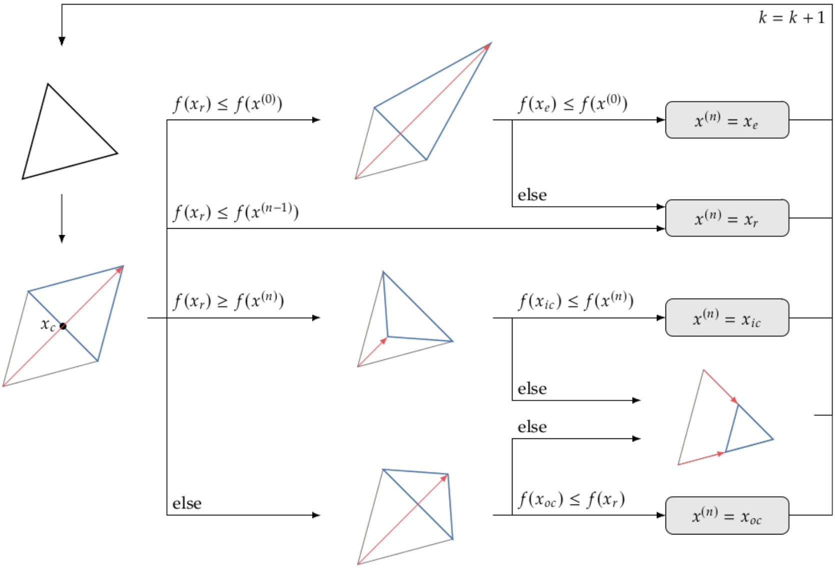
Nelder-Mead

The above procedure is repeated until convergence. This may be decided, e.g., based on the size of the simplex:

$$\Delta_x = \sum_{i=0}^{n-1} \left\| x^{(i)} - x^{(n)} \right\| < \epsilon$$

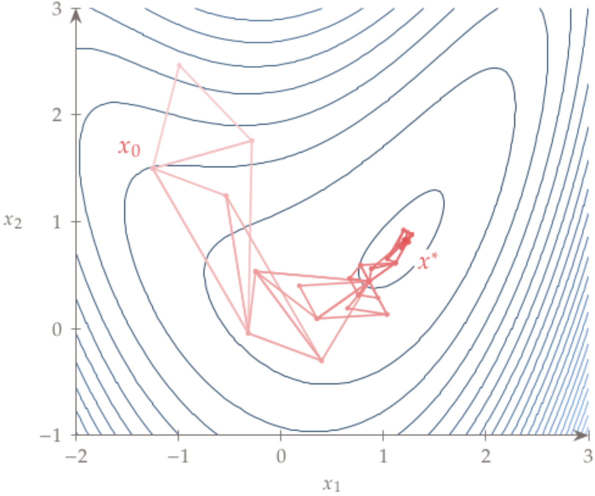
Standard values for constants are:

- Reflection $\alpha = 1$
- Expansion $\gamma = 2$
- Contraction $\rho = 0.5$
- Shrink $\sigma = 0.5$



$k = k + 1$
$f(x_r) \leq f(x^{(0)})$
$f(x_e) \leq f(x^{(0)})$
$x^{(n)} = x_e$
else
$f(x_r) \leq f(x^{(n-1)})$
$x^{(n)} = x_r$
x_c
$f(x_r) \geq f(x^{(n)})$
$f(x_{ic}) \leq f(x^{(n)})$
$x^{(n)} = x_{ic}$
else
else
else
$f(x_{oc}) \leq f(x_r)$
$x^{(n)} = x_{oc}$

Nelder-Mead Example



Particle Swarm Optimization

- The "swarm" in PSO is a set of points (agents or particles) that move in space, looking for the best solution.

Particle Swarm Optimization

- The “swarm” in PSO is a set of points (agents or particles) that move in space, looking for the best solution.
- Each particle moves according to its velocity.

Particle Swarm Optimization

- The “swarm” in PSO is a set of points (agents or particles) that move in space, looking for the best solution.
- Each particle moves according to its velocity.
- This velocity changes according to the past objective function values of that particle and the current objective values of the rest of the particles.

Particle Swarm Optimization

- The “swarm” in PSO is a set of points (agents or particles) that move in space, looking for the best solution.
- Each particle moves according to its velocity.
- This velocity changes according to the past objective function values of that particle and the current objective values of the rest of the particles.
- Each particle remembers the point where it found its best result so far, and it exchanges the information with the swarm.

PSO

The position of particle i for iteration $k+1$ is updated according to

$$x_{k+1}^{(i)} = x_k^{(i)} + v_{k+1}^{(i)} \Delta t,$$

Where Δt is a constant artificial time step. The velocity for each particle is updated as follows:

$$v_{k+1}^{(i)} = \alpha v_k^{(i)} + \beta \frac{x_{\text{best}}^{(i)} - x_k^{(i)}}{\Delta t} + \gamma \frac{x_{\text{best}} - x_k^{(i)}}{\Delta t}$$

PSO

The position of particle i for iteration $k+1$ is updated according to

$$x_{k+1}^{(i)} = x_k^{(i)} + v_{k+1}^{(i)} \Delta t,$$

Where Δt is a constant artificial time step. The velocity for each particle is updated as follows:

$$v_{k+1}^{(i)} = \alpha v_k^{(i)} + \beta \frac{x_{\text{best}}^{(i)} - x_k^{(i)}}{\Delta t} + \gamma \frac{x_{\text{best}} - x_k^{(i)}}{\Delta t}$$

- The first term is momentum.

  α is usually set from the interval $[0.8, 1.2]$, higher α motivates exploration, smaller α convergence towards (a local) minimizer.

PSO

The position of particle i for iteration $k+1$ is updated according to

$$x_{k+1}^{(i)} = x_k^{(i)} + v_{k+1}^{(i)} \Delta t,$$

Where Δt is a constant artificial time step. The velocity for each particle is updated as follows:

$$v_{k+1}^{(i)} = \alpha v_k^{(i)} + \beta \frac{x_{\text{best}}^{(i)} - x_k^{(i)}}{\Delta t} + \gamma \frac{x_{\text{best}} - x_k^{(i)}}{\Delta t}$$

- The first term is momentum.
  α is usually set from the interval $[0.8, 1.2]$, higher α motivates exploration, smaller α convergence towards (a local) minimizer.
- $x_{\text{best}}^{(i)}$ is the first minimum objective point visited by the i-th particle.
  β is usually set randomly from $[0, \beta_{\text{max}}]$. β_{max} is usually selected from the interval $[0, 2]$, closer to 2.
- x_{best} is a minimum objective point visited by any particle.
  γ is also usually set randomly from the interval $[0, \gamma_{\text{max}}]$. γ_{max} is usually selected from the interval $[0, 2]$, closer to 2.

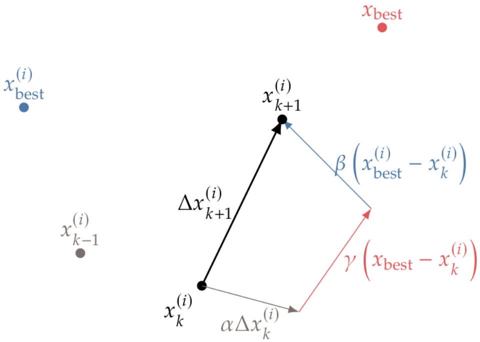
$$v_{k+1}^{(i)} = \alpha v_k^{(i)} + \beta \frac{x_{\text{best}}^{(i)} - x_k^{(i)}}{\Delta t} + \gamma \frac{x_{\text{best}} - x_k^{(i)}}{\Delta t}.$$

Eliminate Δt by multiplying with Δt:

$$\Delta x_{k+1}^{(i)} = \alpha \Delta x_k^{(i)} + \beta \left(x_{\text{best}}^{(i)} - x_k^{(i)} \right) + \gamma \left(x_{\text{best}} - x_k^{(i)} \right)$$

Then, update the particle position for the next iteration:

$$x_{k+1}^{(i)} = x_k^{(i)} + \Delta x_{k+1}^{(i)}.$$



PSO

- Initialization is usually done randomly.

PSO

- Initialization is usually done randomly.
- The particles should stay in a bounded region. When a particle wants to leave the region, reorient the velocity or reset the position of the particle.

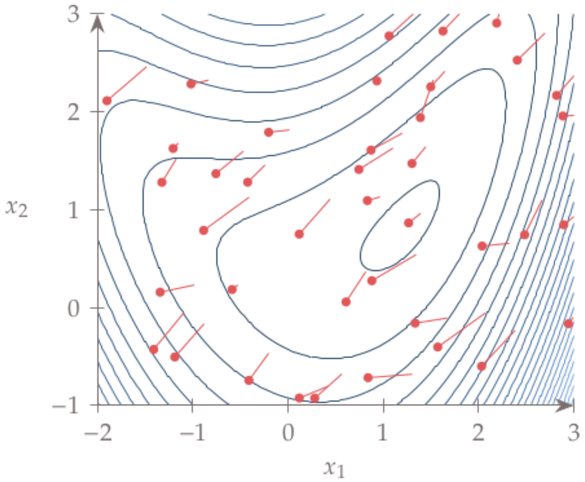
PSO

- Initialization is usually done randomly.
- The particles should stay in a bounded region. When a particle wants to leave the region, reorient the velocity or reset the position of the particle.
- It is also helpful to impose a maximum velocity. Otherwise, updates completely unrelated to the previous positions might be made.

PSO

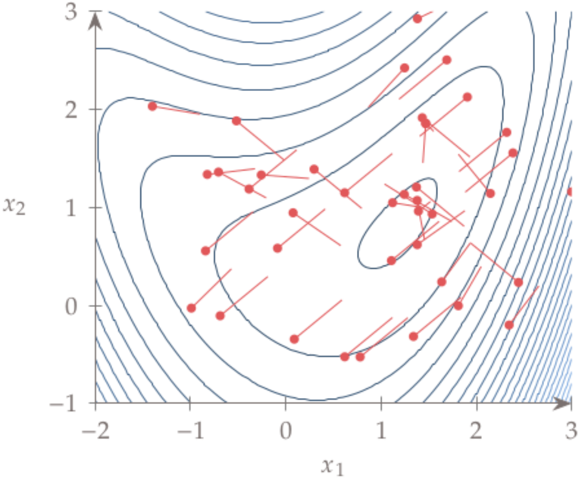
- Initialization is usually done randomly.
- The particles should stay in a bounded region. When a particle wants to leave the region, reorient the velocity or reset the position of the particle.
- It is also helpful to impose a maximum velocity. Otherwise, updates completely unrelated to the previous positions might be made.
- The velocity may be decreased gradually to exchange exploitation with exploration.

Example



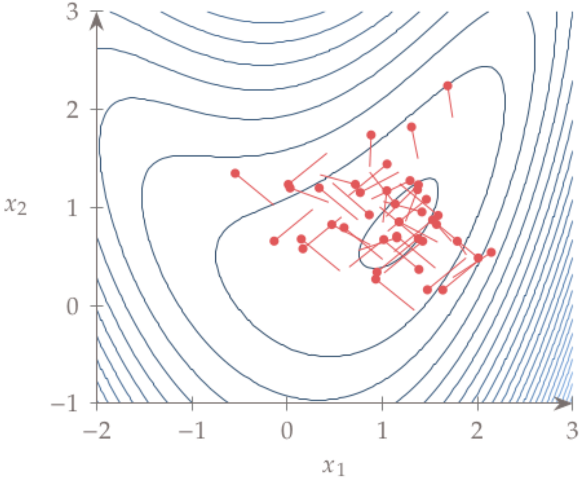
$K = 0$

Example



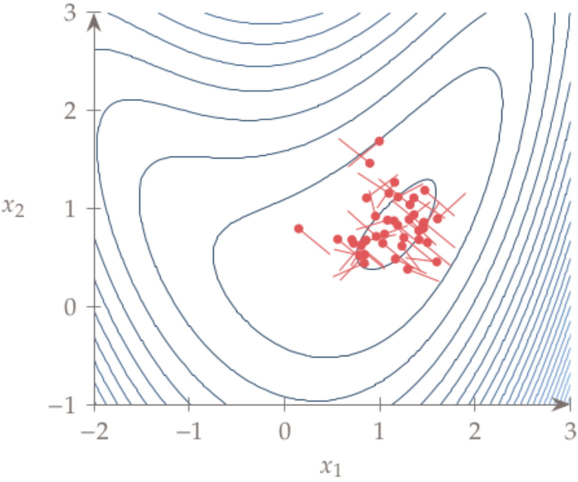
$K = 1$

Example



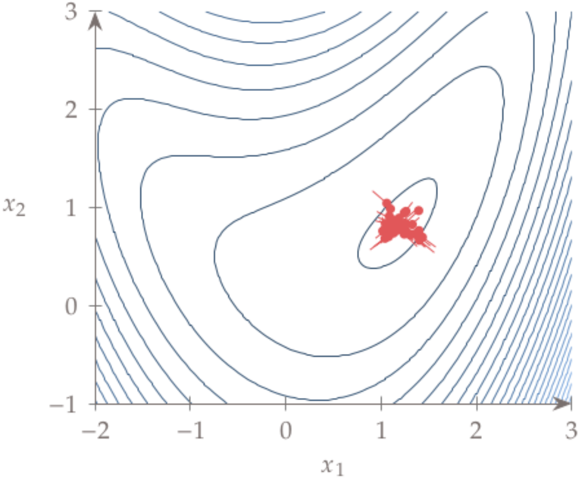
$K = 3$

Example



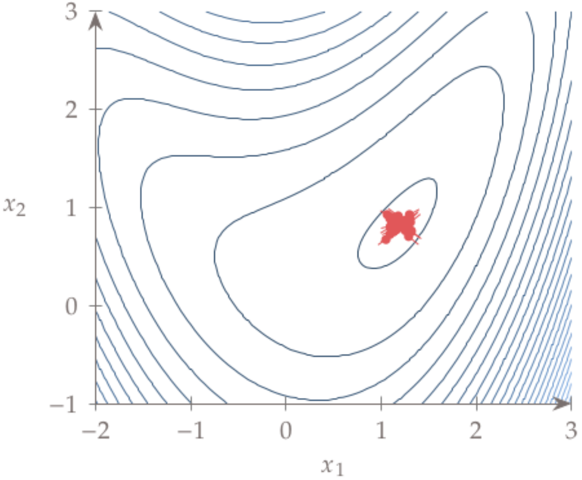
$K = 5$

Example



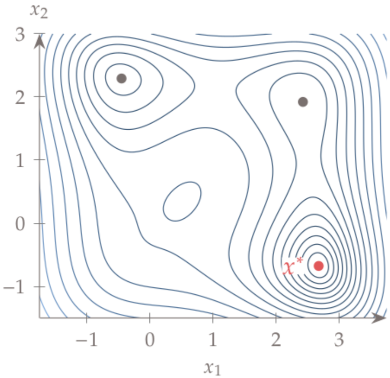
$K = 12$

Example



$K = 17$

Jones Function



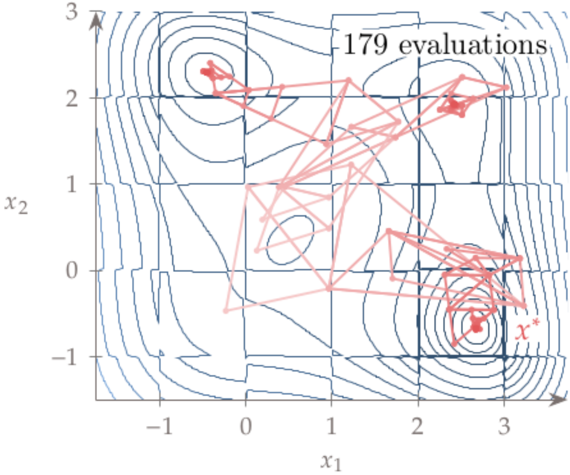
$$f(x_1, x_2) = x_1^4 + x_2^4 - 4x_1^3 - 3x_2^3 + 2x_1^2 + 2x_1x_2$$

Global minimum: $f(x^*) = -13.5320$ at $x^* = (2.6732, -0.6759)$.

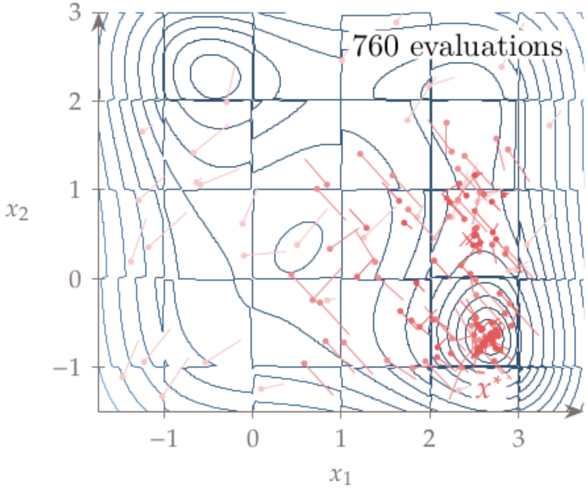
Local minima: $f(x) = -9.7770$ at $x = (-0.4495, 2.2928)$

$f(x) = -9.0312$ at $x = (2.4239, 1.9219)$

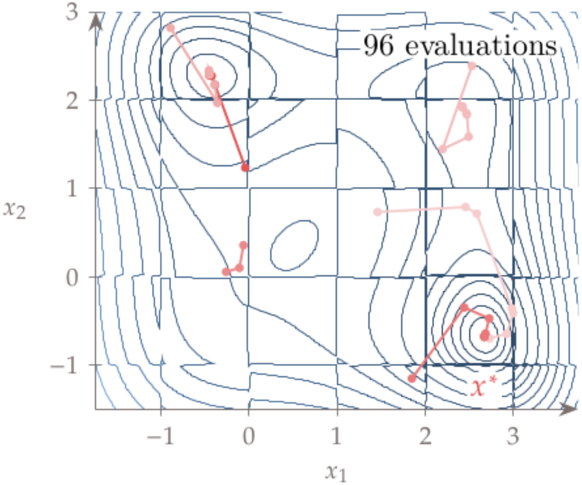
Make it discontinuous by adding $4\lceil \sin(\pi x_1)\sin(\pi x_2)\rceil$



Nelder-Mead: 179 evaluations were needed to reach the minimum (with restarts due to local minima).



Particle Swarm Optimization: 760 evaluations found the global minimum without restarts.



Quasi-Newton with restarts: 96 evaluations needed. Converged in two out of six random restarts.

FINALE!

Summary

We have considered the following methods:

- Unconstrained & Differentiable Objective
  - Line Search with step size determined by Wolfe conditions and direction determined by
    - Gradient Descent
    - Newton's Method (2nd derivatives needed)
    - Quasi-Newton: SR-1, BFGS

Summary

We have considered the following methods:

- Unconstrained & Differentiable Objective
  - Line Search with step size determined by Wolfe conditions and direction determined by
    - Gradient Descent
    - Newton's Method (2nd derivatives needed)
    - Quasi-Newton: SR-1, BFGS
- Constrained & Differentiable Objective
  - Penalty Methods
    - Exterior (quadratic penalty)
    - Interior/Barrier (inverse and logarithmic barriers)
  - Lagrangian used in Sequential Quadratic Programming

Summary

We have considered the following methods:

- Unconstrained & Differentiable Objective
  - Line Search with step size determined by Wolfe conditions and direction determined by
    - Gradient Descent
    - Newton's Method (2nd derivatives needed)
    - Quasi-Newton: SR-1, BFGS
- Constrained & Differentiable Objective
  - Penalty Methods
    - Exterior (quadratic penalty)
    - Interior/Barrier (inverse and logarithmic barriers)
  - Lagrangian used in Sequential Quadratic Programming
- Linear Objective and Constraints
  - Simplex Method (including degenerate case and tableaus)
  - Branch & Bound
  - Gomory Cuts

Summary

We have considered the following methods:

- Unconstrained & Differentiable Objective
  - Line Search with step size determined by Wolfe conditions and direction determined by
    - Gradient Descent
    - Newton's Method (2nd derivatives needed)
    - Quasi-Newton: SR-1, BFGS
- Constrained & Differentiable Objective
  - Penalty Methods
    - Exterior (quadratic penalty)
    - Interior/Barrier (inverse and logarithmic barriers)
  - Lagrangian used in Sequential Quadratic Programming
- Linear Objective and Constraints
  - Simplex Method (including degenerate case and tableaus)
  - Branch & Bound
  - Gomory Cuts
- Unconstrained & Non-Differentiable (just a few examples)
  - Nelder-Mead
  - Particle Swarm Optimization

Most Notable Omissions

- Conjugate Gradient Methods

  Unfortunately, I had to choose between quasi-Newton and CG.
- Trust Region Methods
- Combinatorial, Multiobjective, Stochastic, Bayesian (etc.) Optimization

  Completely different areas with different methods.
- Infinitely many non-differentiable optimization methods motivated by arbitrary phenomena from:
  - biology
  - chemistry
  - physics
  - economics
  - politics
  - mathematics
  - agriculture
  - pop-culture
  - Scientology
  - astrology
  - ... ...