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.

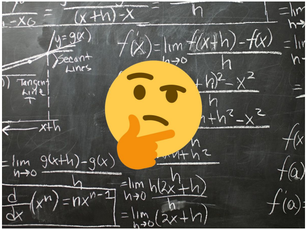
Evaluation

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

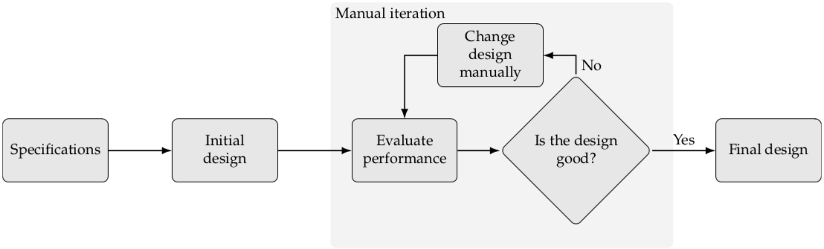
sklearn.linear_model.LogisticRegression

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*)

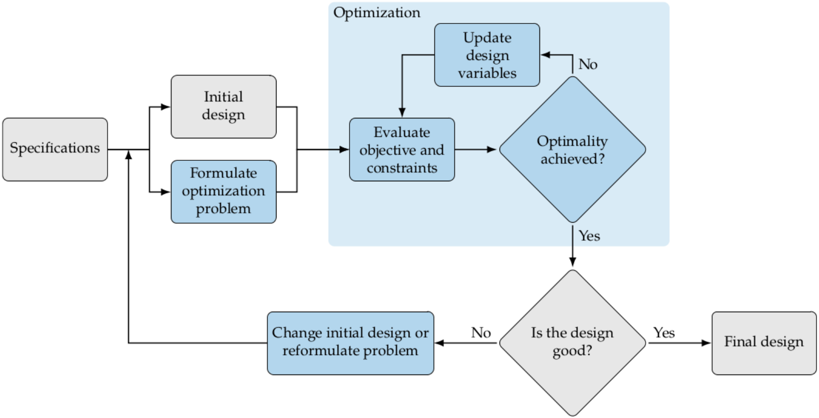
solver : *{'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'}, default='lbfgs'*

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:

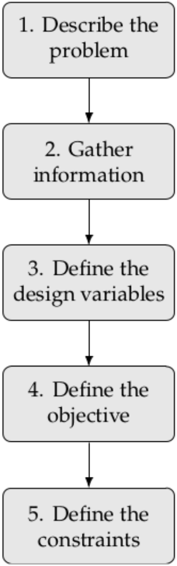
- Consider a company with several plants producing a single product but with different efficiency.
- The goal is to set the production of each plant so that demand for goods is satisfied, but overproduction is minimized.
- **First try**: Model each plant's production and maximize the total production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

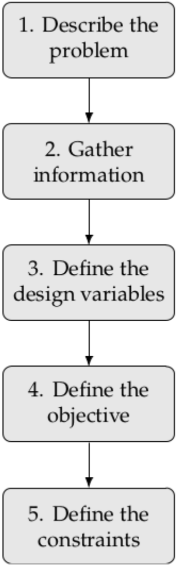


Optimization Problem Formulation

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).

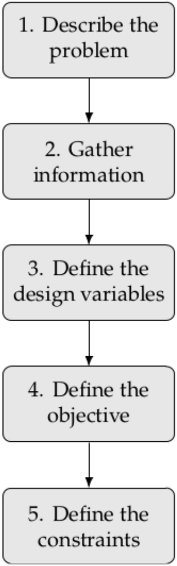


Optimization Problem Formulation

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.

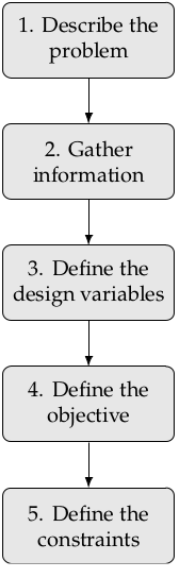


Optimization Problem Formulation

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

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

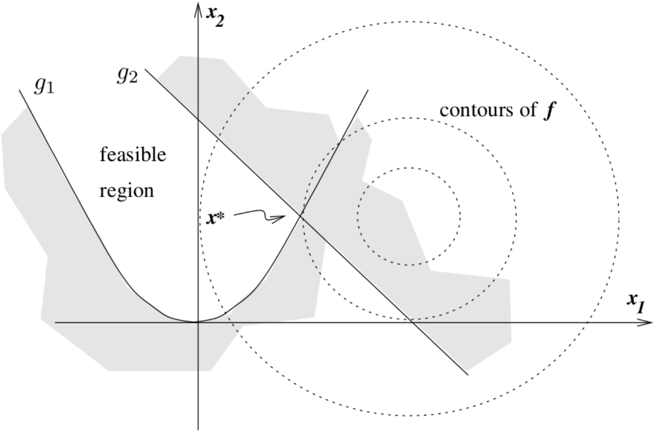
I.e.,

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

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

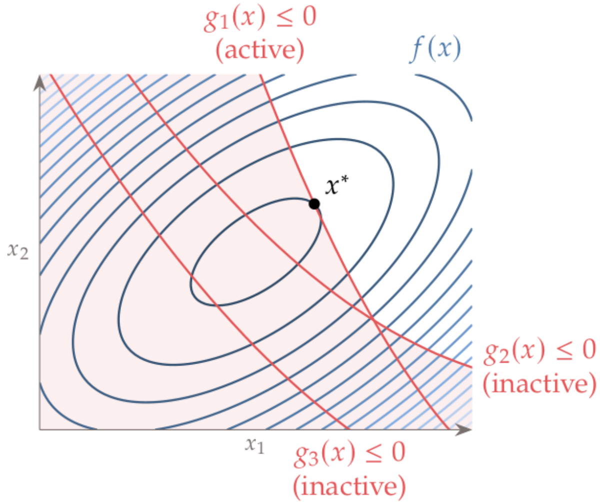
Inequality constraints $g_i(x) \leq 0$ can be *active* or *inactive*.

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

$$\begin{aligned}
\sum_{j=1}^{12} x_{ij} &\leq a_i, \quad i = 1, 2 \\
\sum_{i=1}^{2} x_{ij} &\geq b_j, \quad j = 1, \ldots, 12, \\
x_{ij} &\geq 0, \quad i = 1, 2, \quad j = 1, \ldots, 12.
\end{aligned}$$

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

$$\begin{aligned} \sum_{j=1}^{12} x_{ij} &\leq a_i, \quad i = 1, 2 \\ \sum_{i=1}^{2} x_{ij} &\geq b_j, \quad j = 1, \ldots, 12, \\ x_{ij} &\geq 0, \quad i = 1, 2, \quad j = 1, \ldots, 12. \end{aligned}$$

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

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

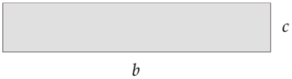
It constrains x_i only to integer values. This leads to so-called *integer programming*.

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.

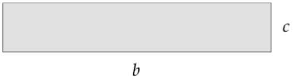


The parameters are call *span* b and *chord* c.

Wing Design Example

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

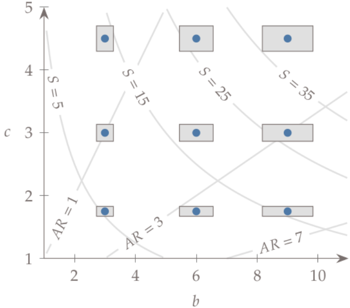
Assume a rectangular wing.

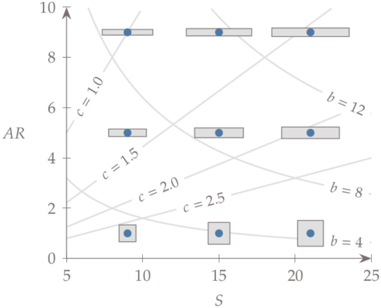


The parameters are call *span* b and *chord* c.

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

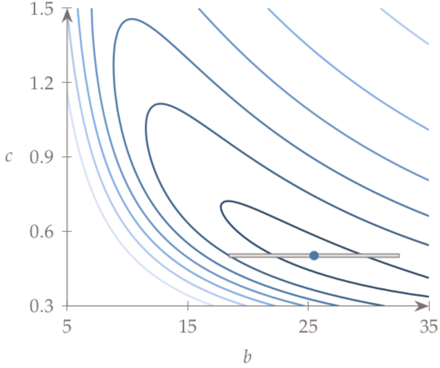
The propulsion efficiency η can be roughly approximated by the Gaussian efficiency curve.

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

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

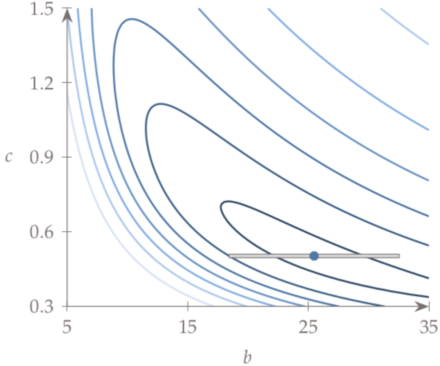
Wing Design Example

The objective function contours:



Wing Design Example

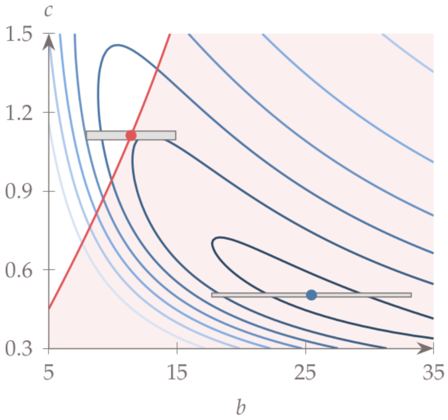
The objective function contours:



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!

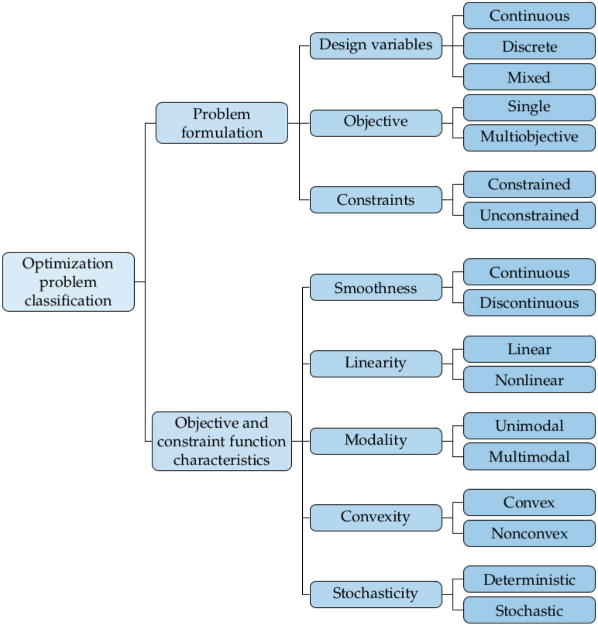
Wing Design Example

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

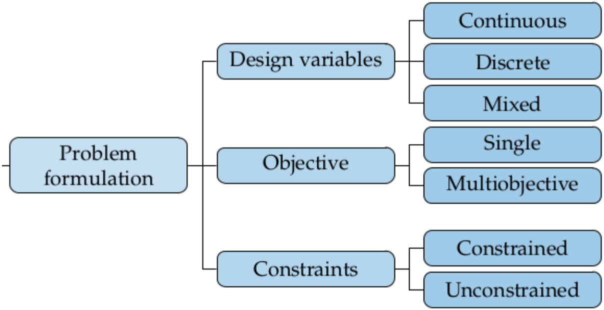


It looks like a reasonable wing ...

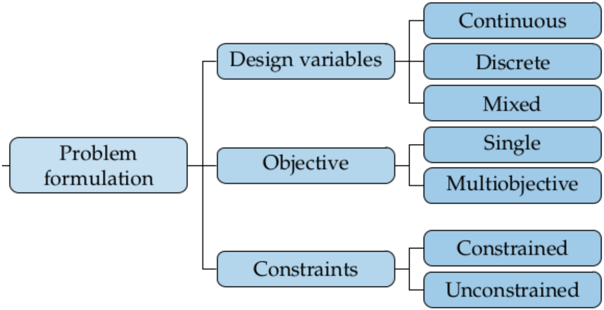
Optimization Problem Classification



Optimization Problem Classification

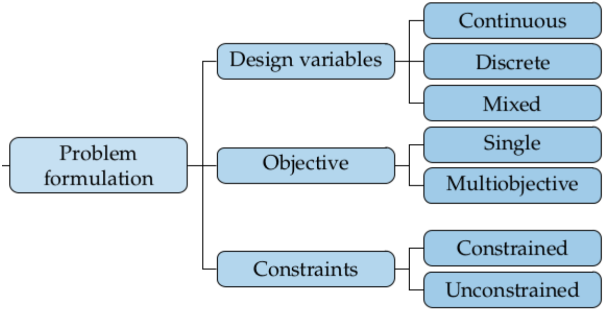


Optimization Problem Classification



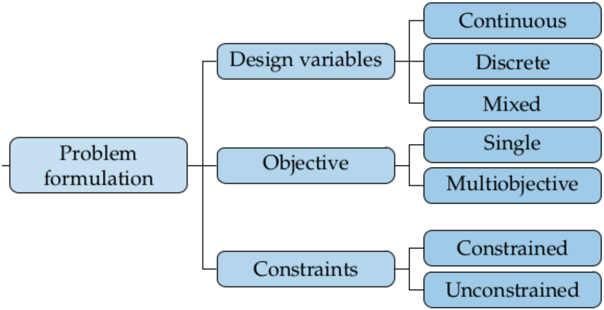
- *Continuous* allows only $x_i \in \mathbb{R}$, *discrete* allows only $x_i \in \mathbb{Z}$, mixed allows variables of both kinds.

Optimization Problem Classification



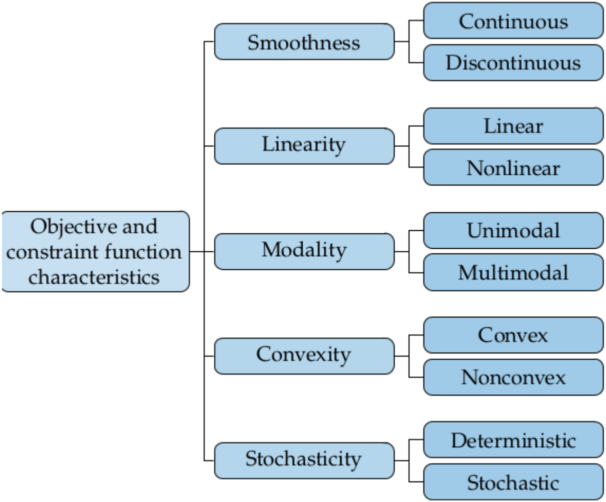
- *Continuous* allows only $x_i \in \mathbb{R}$, *discrete* allows only $x_i \in \mathbb{Z}$, mixed allows variables of both kinds.
- *Single-objective*: $f : \mathbb{R}^n \to \mathbb{R}$, *Multi-objective*: $f : \mathbb{R}^n \to \mathbb{R}^m$

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.

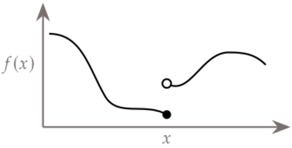
Optimization Problem Classification

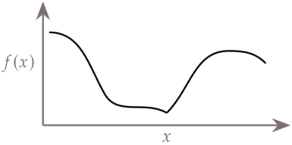


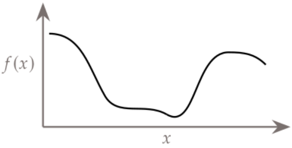
Smoothness

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.

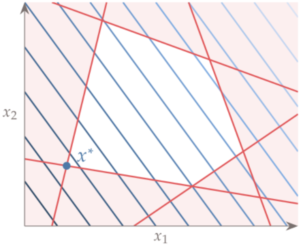






Linearity

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.

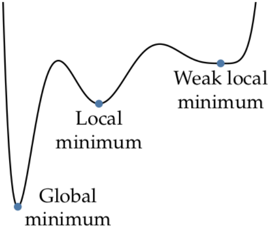
x^* is a *(weak) local minimiser* if there is $\varepsilon > 0$ such that

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

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

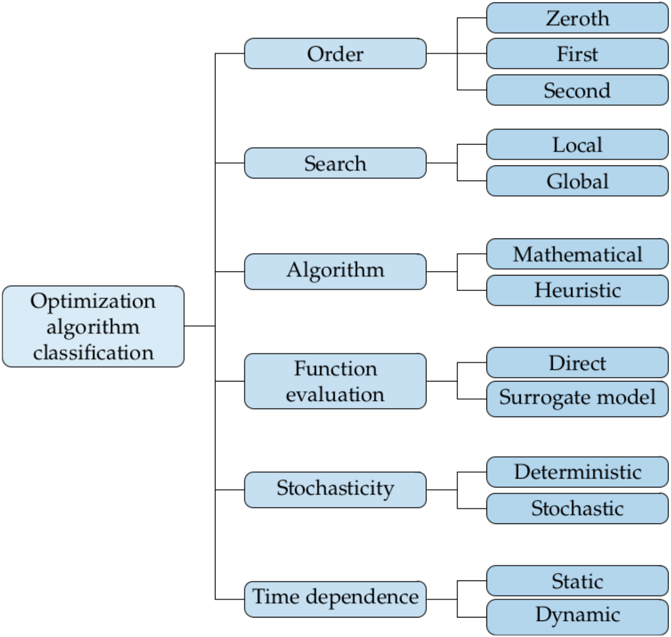
Typically, we are concerned with the following issues:

- *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.

Require: Function $f \in \mathcal{C}^1$, initial point x_0, initial step length $\alpha_0 > 0$, tolerance ϵ

Ensure: A point x that approximately minimizes $f(x)$

Initialize $x \leftarrow x_0$

Initialize step length $\alpha \leftarrow \alpha_0$

while $|f'(x)| > \epsilon$ **do**

  Compute the gradient $g \leftarrow f'(x)$

  Update $x \leftarrow x - \alpha \cdot g$

  Update step length α based on a certain strategy

end while

return x

Denote by x_k and α_k the values of x and α in the k-th iteration, respectively.

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

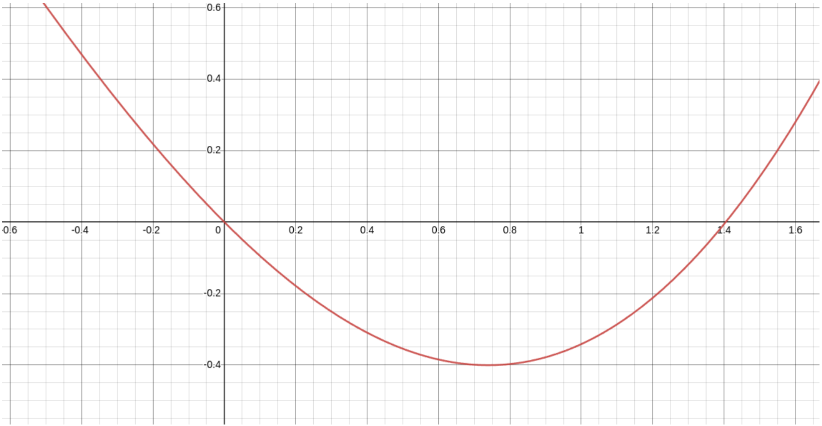
Consider a sequence $x_0, x_1, \ldots$ computed by the gradient descent algorithm for f. Assume a constant step length $\alpha \leq \frac{1}{L}$.
Then $\lim_{k \to \infty} |f'(x_k)| = 0$ and, moreover,

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

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:

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

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....

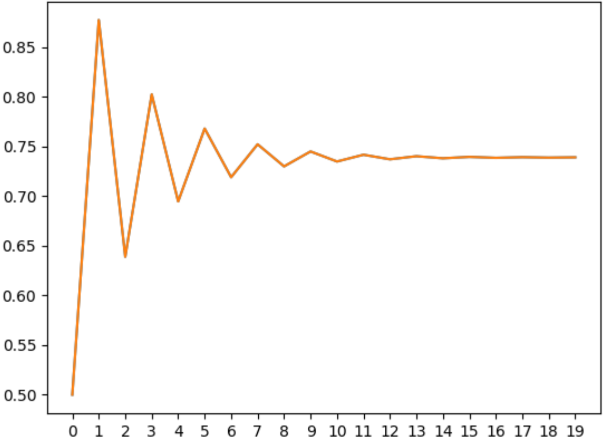
Frame Title

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

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

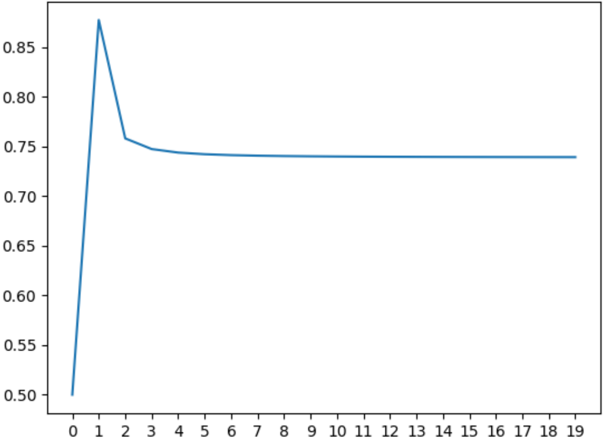
Example

Gradient descent with the step length $= 1.0$:



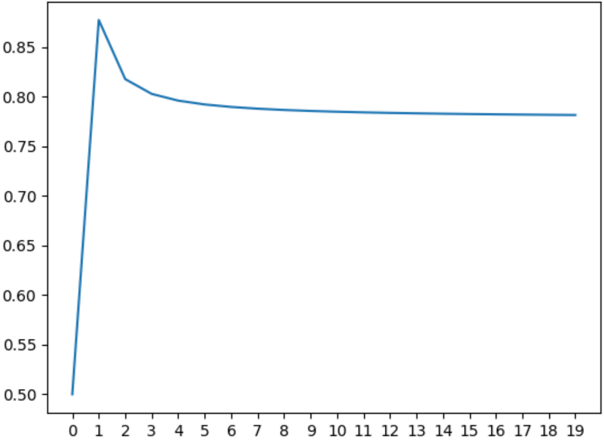
Example

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



Example

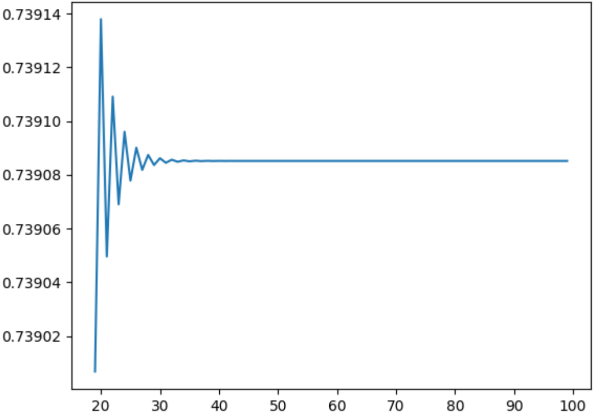
Gradient descent with the step length $= 1/k^2$:



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

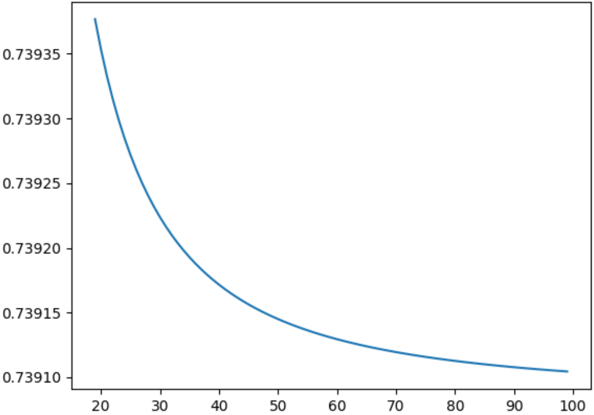
Example

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 the global 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 the global 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

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

Assume that $f \in \mathcal{C}^2$, i.e., that f'' exists and is continuous, and let us fix $x_0 \in \mathbb{R}$. 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

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

Assume that $f \in \mathcal{C}^2$, i.e., that f'' exists and is continuous, and let us fix $x_0 \in \mathbb{R}$. 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.

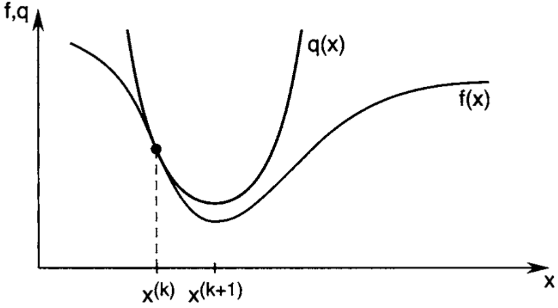
Newton's Method Idea

The method computes successive approximations $x_0, x_1, \ldots, x_k, \ldots$ 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.



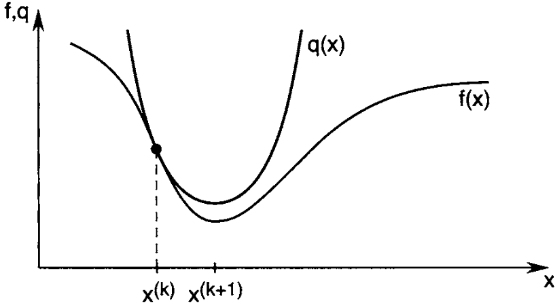
Newton's Method Idea

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

Given: A function f with derivative f' and second derivative f'', and an initial guess x_0
Goal: Find a solution to $f'(x) = 0$
repeat
    Calculate the derivative: $y' \leftarrow f'(x_k)$
    Calculate the second derivative : $y'' \leftarrow f''(x_k)$
    Update the estimate: $x_{k+1} \leftarrow x_k - \frac{y'}{y''}$
    Increment k
until a sufficiently accurate value is found

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 $|x_{k+1} - x_k| < \epsilon$.

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 $|x_{k+1} - x_k| < \epsilon$.

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 $|x_{k+1} - x_k| < \epsilon$.

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

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

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

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

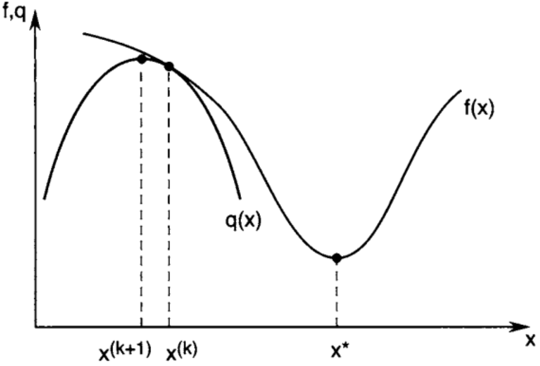
So, we conclude that $x^* \approx x_4$ is a strict minimizer.

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

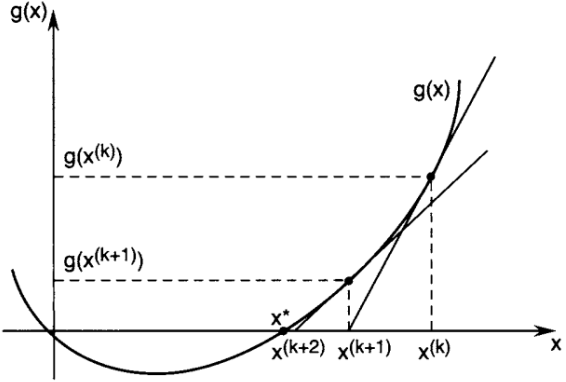
Newton's method is also a technique for finding roots of functions. In our case, this means finding a root of f'.

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 with

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

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 with

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

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

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

Is the rate of convergence superlinear?

Example

Consider the following objective function f

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

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

Now, we need to initialize the first two values.

Example

Consider the following objective function f

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

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

Now, we need to initialize the first two values.

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

Hence,

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

Example

Continuing, we obtain:

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

Example

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

$$
\begin{aligned}
x_0 &= 0.5 \\
x_1 &= 0.7552 \\
x_2 &= 0.7381 \\
x_3 &= 0.739081 \\
x_5 &= 0.7390851339 \\
x_6 &= 0.7390851332 \\
&\ldots
\end{aligned}
$$

Compare with Newton's method:

$$
\begin{aligned}
x_0 &= 0.5 \\
x_1 &= 0.7552 \\
x_2 &= 0.7391 \\
x_3 &= 0.7390851339 \\
x_4 &= 0.7390851321516067229 \\
x_5 &= 0.7390851321516067229 \\
&\ldots
\end{aligned}
$$

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 $f''(x_0) > 0$. 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

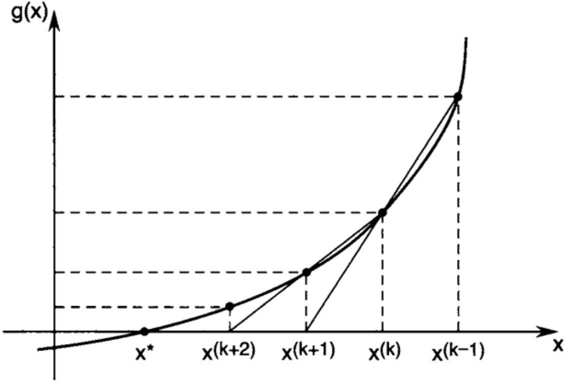
As for Newton's method of tangents, the secant method can be seen as a method for finding a root of f'.

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:

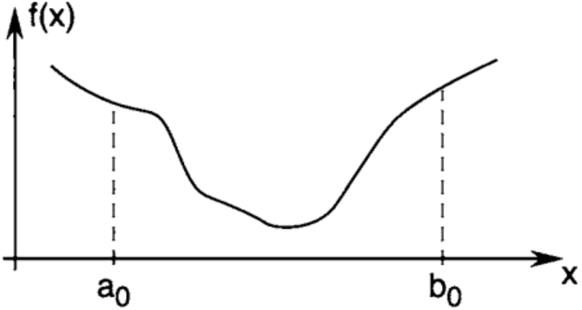
- f unimodal on $[a_0, b_0]$
- f continuously differentiable on $[a_0, b_0]$
- f twice continuously differentiable on $[a_0, b_0]$

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 $x' < x''$

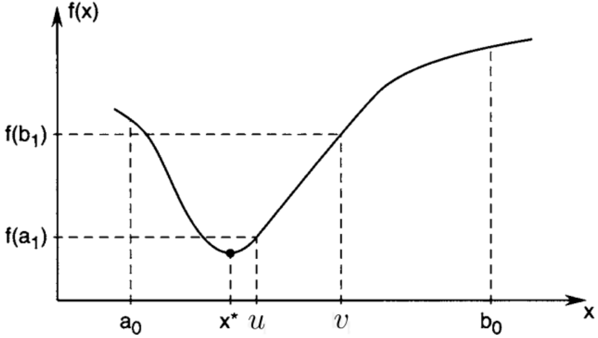


Assume that even a single evaluation of f is costly.

Minimize the number of evaluations searching for the minimum.

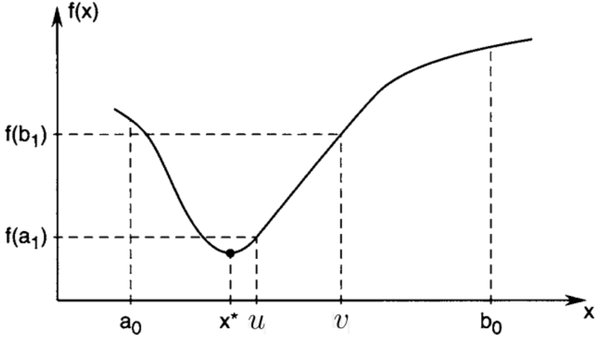
Simple Algorithm

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**
  $u_k \leftarrow a_k + \rho(b_k - a_k)$
  $v_k \leftarrow b_k - \rho(b_k - a_k)$
  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

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

$$
\begin{aligned}
u_1 &= a_1 + \rho\,(b_1 - a_1) = 0.4721 \\
v_1 &= a_1 + (1 - \rho)\,(b_1 - a_1) = 0.7639
\end{aligned}
$$

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:

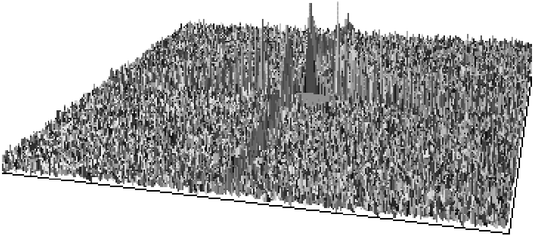
- f unimodal on $[a_0, b_0]$
- f continuously differentiable on $[a_0, b_0]$
- f twice continuously differentiable on $[a_0, b_0]$

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

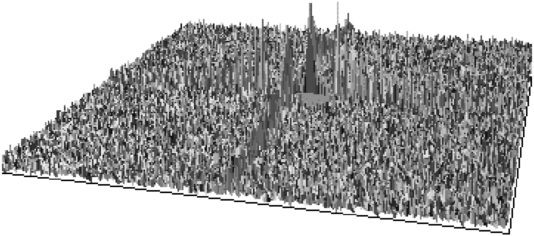
How to Recognize (Local) Minimum

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

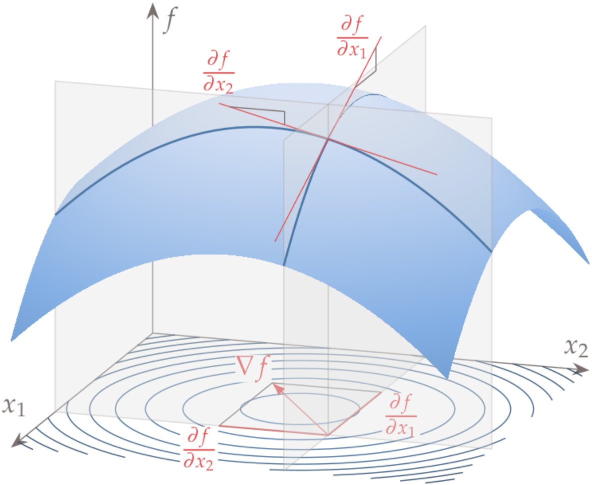
The gradient of $f : \mathbb{R}^n \to \mathbb{R}$, denoted by $\nabla f(x)$, is a column vector of first-order partial derivatives of the function concerning each variable:

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

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



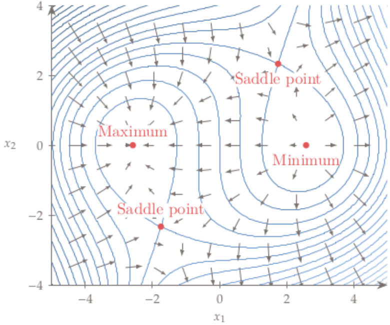
The gradient is a vector pointing in the direction of the most significant function increase from the current point.

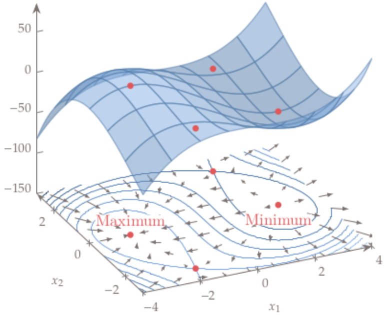
Gradient

Consider the following function of two variables:

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

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



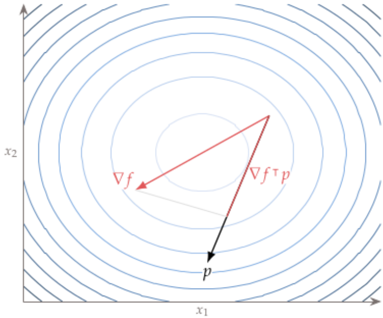


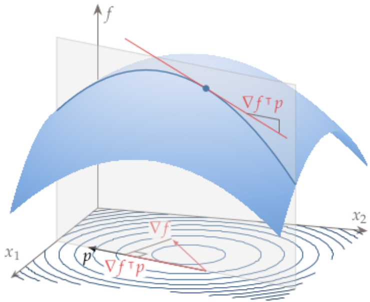
Directional Derivatives vs Gradient

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.

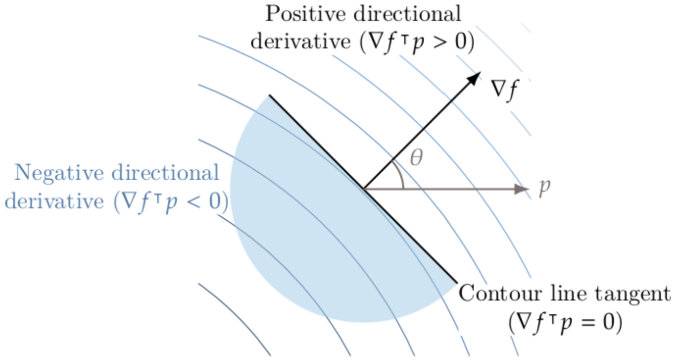
Geometry of Gradient

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 $g''(0)$?

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

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

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

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

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:

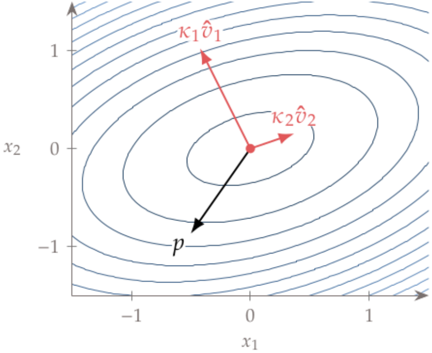
$$H\hat{v}_k = \kappa_k \hat{v}_k$$

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.



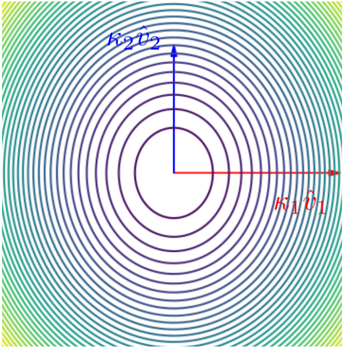
Consider $f(x) = x^\top H x$ where

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

The eigenvalues are

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

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



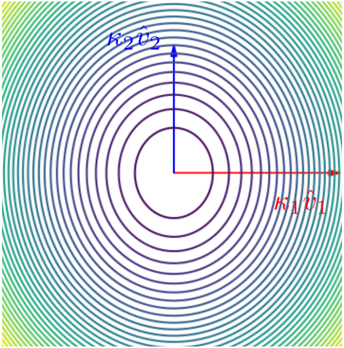
Consider $f(x) = x^\top H x$ where

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

The eigenvalues are

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

Their corresponding eigenvectors are $\color{red}{(1,0)^\top}$ and $\color{blue}{(0,1)^\top}$.



Note that

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

Considering a direction vector p we get

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

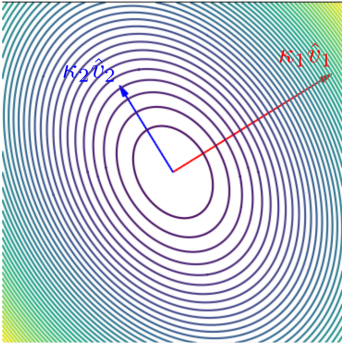
which is a parabola with $g'' = 2\left(\kappa_1 p_1^2 + \kappa_2 p_2^2\right)$.

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

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

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

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



The eigenvalues are

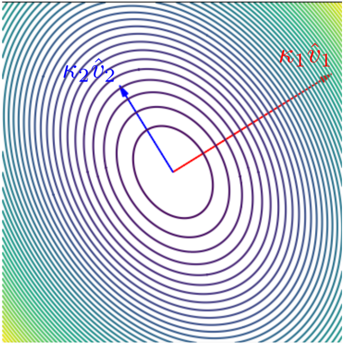
$$\kappa_1 = \frac{1}{6}(7+\sqrt{5}) \quad \kappa_2 = \frac{1}{6}(7-\sqrt{5})$$

Their corresponding eigenvectors are

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

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 $\hat{v}_1, \hat{v}_2$.

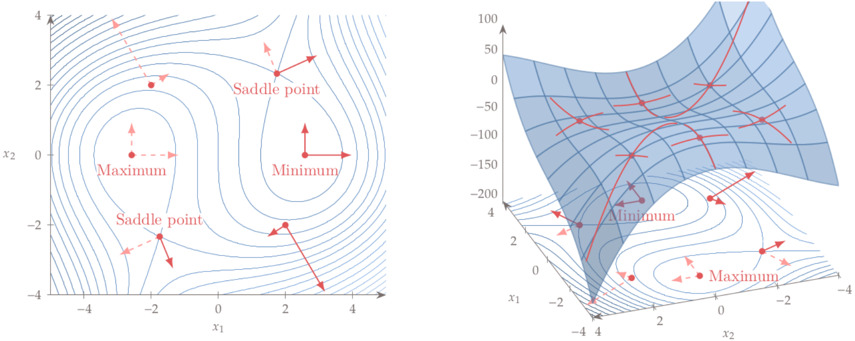
Hessian Visualization Example

Consider

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

And it's Hessian.

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



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

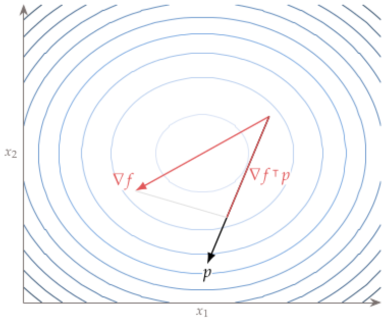
$$f(x+p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T H(x) p + o(||p||^2)$$

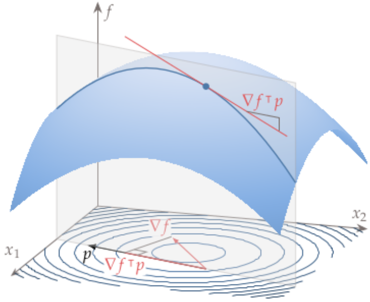
Here $H = \nabla^2 f$ is the Hessian of f.

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

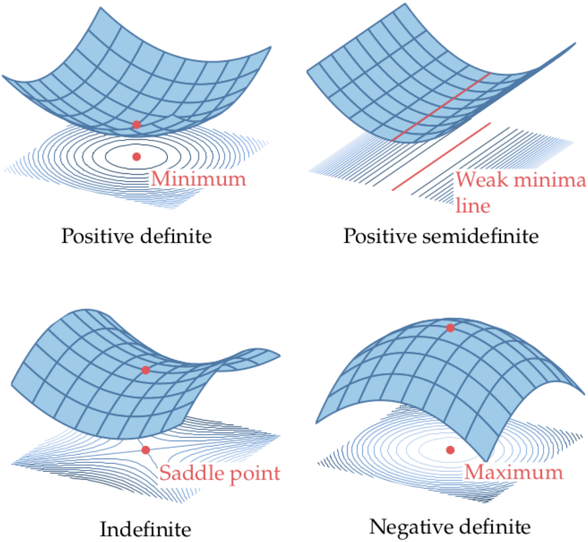
Note that $\nabla f(x^*) = 0$ does not tell us whether x^* is a minimizer, maximizer, or a saddle point.

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



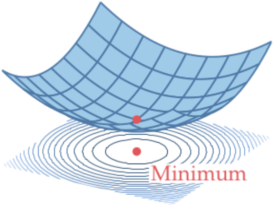
Second-Order Necessary Condition

Theorem 6 (Second-Order Necessary Conditions)

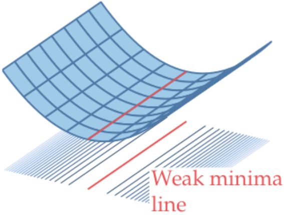
If x^* is a local minimizer of f and $\nabla^2 f$ is continuous in an open neighborhood of x^*, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite.

Theorem 7 (Second-Order Sufficient Conditions)

Suppose that $\nabla^2 f$ is continuous in 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.



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:

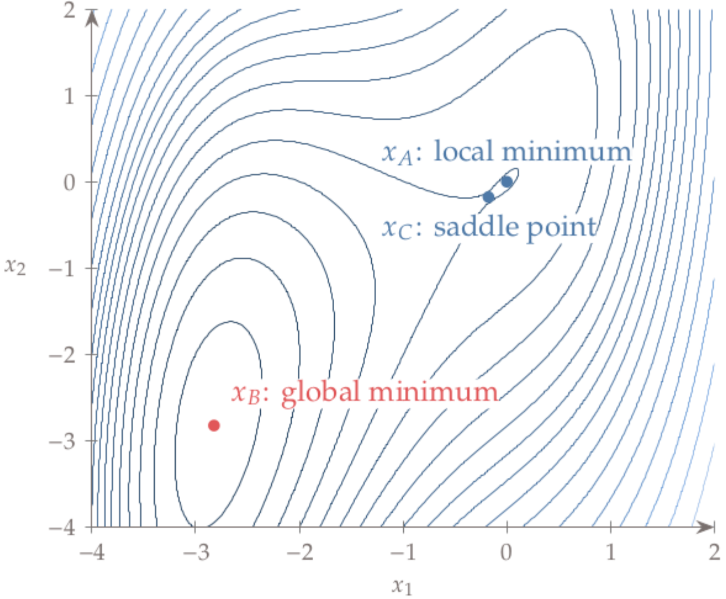
$$H(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 6x_1^2 + 12x_1 + 3 & -2 \\ -2 & 2 \end{bmatrix}.$$

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_1
x_2
−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

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

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

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

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

Proof of Theorem 6 (Optional)

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

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

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

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

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

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

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

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. $\square$

Unconstrained Optimization Algorithms

Search Algorithms

We consider algorithms that

- Start with an initial guess x_0
- Generate a sequence of points $x_0, x_1, \ldots$
- 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 $x_0, x_1, \ldots, x_k$.

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

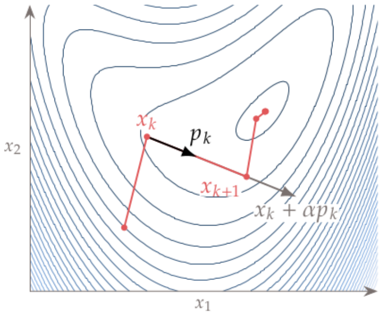
$$x_{k+1} = x_k + \alpha_k p_k$$

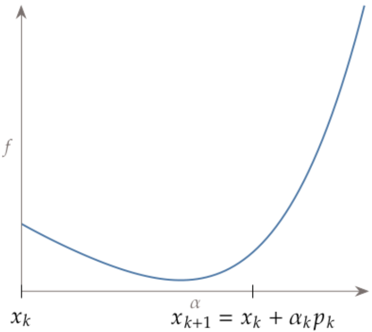
The vector p_k should be a *descent* direction, i.e., a direction in which f decreases locally.

α_k is selected to approximately solve

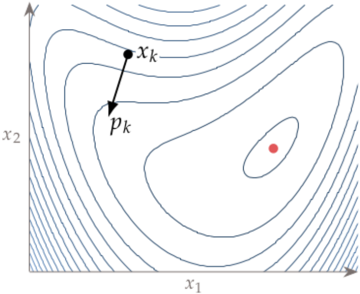
$$\min_{\alpha > 0} f(x_k + \alpha p_k)$$

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

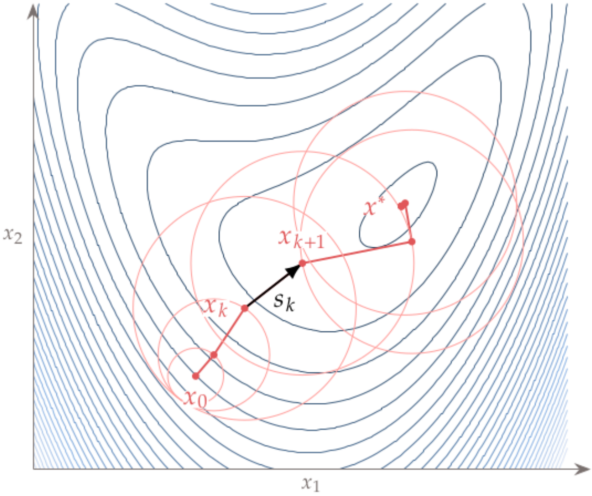
$$\min_{x \in R} m_k(x)$$

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.

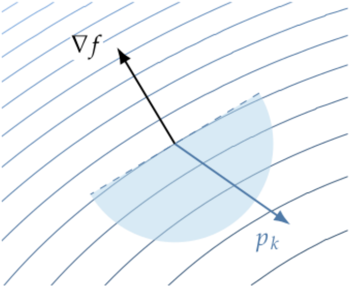
Step Size

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

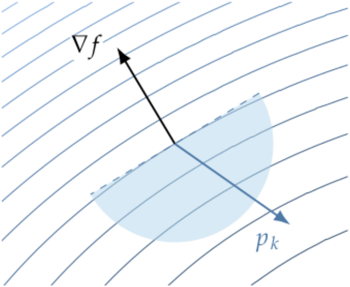
$$x_{k+1} = x_k + \alpha_k p_k$$

Where p_k is a descent direction

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

Define

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



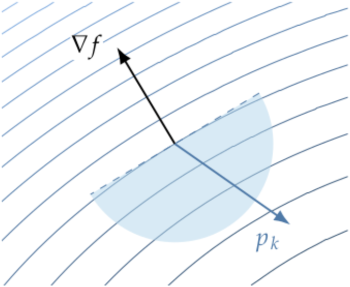
Step Size

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

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

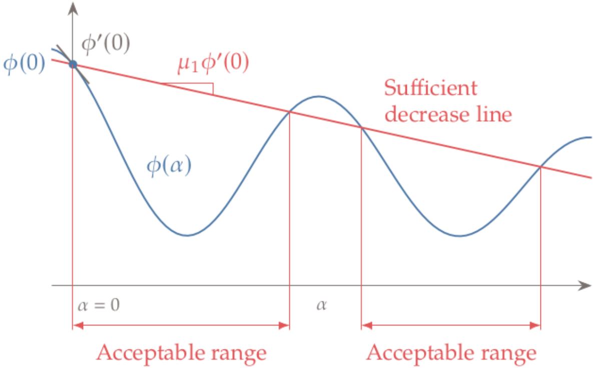
Note that $\phi'(0)$ must be negative as p_k is a descent direction.

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

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

Ensure: α^* satisfying sufficient decrease condition

$\alpha \leftarrow \alpha_{\text{init}}$

while $\phi(\alpha) > \phi(0) + \alpha \mu_1 \phi'(0)$ **do**

  $\alpha \leftarrow \rho \alpha$

end while

Backtracking Line Search Algorithm

Algorithm 2 Backtracking Line Search

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

Ensure: 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).

Backtracking Line Search Algorithm

Algorithm 3 Backtracking Line Search

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

Ensure: α^* satisfying sufficient decrease condition

$\alpha \leftarrow \alpha_{\text{init}}$

while $\phi(\alpha) > \phi(0) + \alpha\mu_1\phi'(0)$ **do**

  $\alpha \leftarrow \rho\alpha$

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:

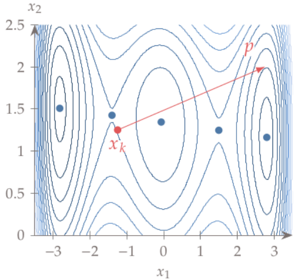
- The guess for the initial step is far too large, and the step sizes that satisfy sufficient decrease are smaller than the starting step by several orders of magnitude. Depending on the value of ρ, this scenario requires many backtracking evaluations.
- The guess for the initial step immediately satisfies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

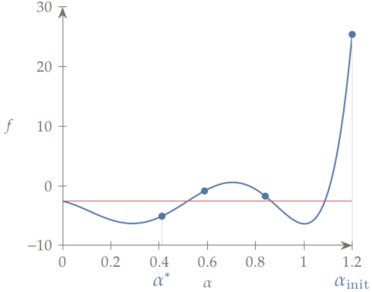
Even if our original step size is not too far from an acceptable one, the basic backtracking algorithm ignores any information we have about the function values and gradients. It blindly takes a reduced step based on a preselected ratio ρ.

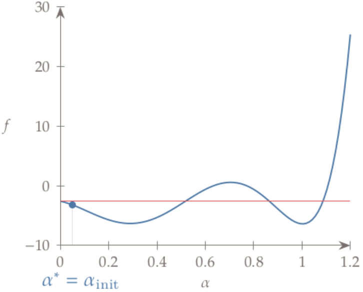
Backtracking Example

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

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

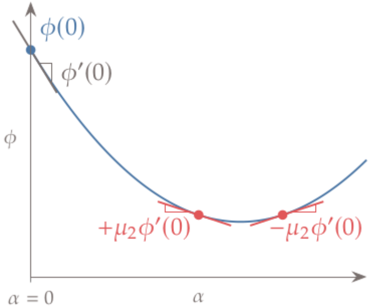
Sufficient Curvature Condition

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.



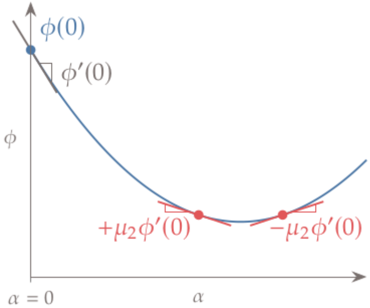
Sufficient Curvature Condition

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.

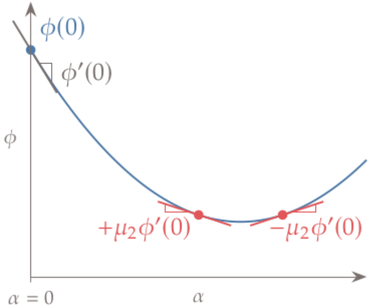
Sufficient Curvature Condition

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

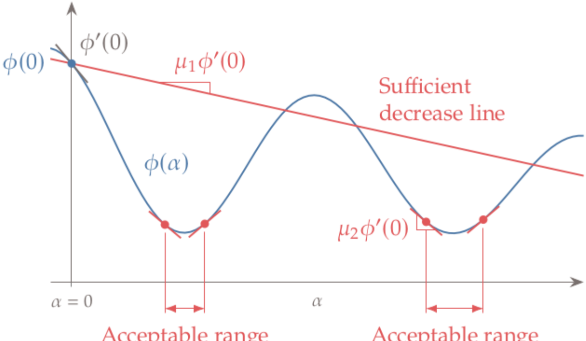
Putting together Armijo and sufficient curvature conditions, we obtain *strong Wolfe conditions*

- *Sufficient decrease condition*

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

- *Sufficient curvature condition*

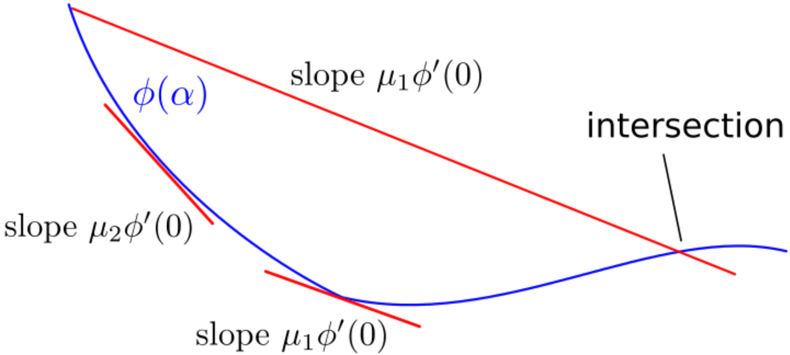
$$\left|\phi'(\alpha)\right| \leq \mu_2 \left|\phi'(0)\right|$$



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

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

Theorem 10 (Zoutendijk condition)

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 $\mathcal{N}$ for some $L > 0$, that is,

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

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 4 Bracketing

Require: $\alpha_1 > 0$ 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 $\alpha_{\mathsf{lo}} \leq \alpha_{\mathsf{hi}}$

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}

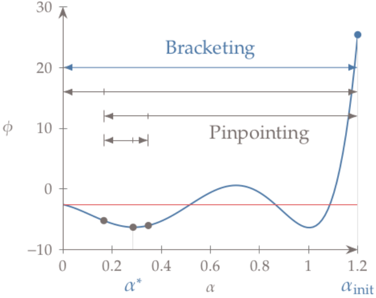
The following invariants are being preserved:

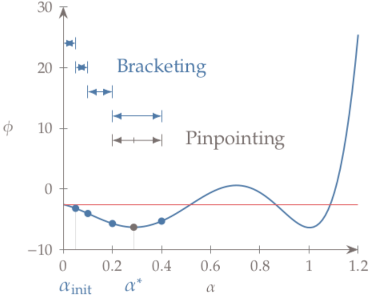
- The interval bounded by α_{lo} and α_{hi} always contains one or more intervals satisfying the strong Wolfe conditions.
  Note that we *do not* assume $\alpha_{\mathsf{lo}} \leq \alpha_{\mathsf{hi}}$
- α_{lo} is, among all step lengths generated so far and satisfying the sufficient decrease condition, 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

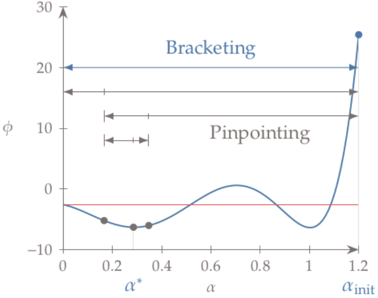
We use quadratic interpolation; the bracketing chooses $\alpha_{i+1} = 2\alpha_i$, and the sufficient curvature factor is $\mu_2 = 0.9$.





Bracketing & Zooming Example

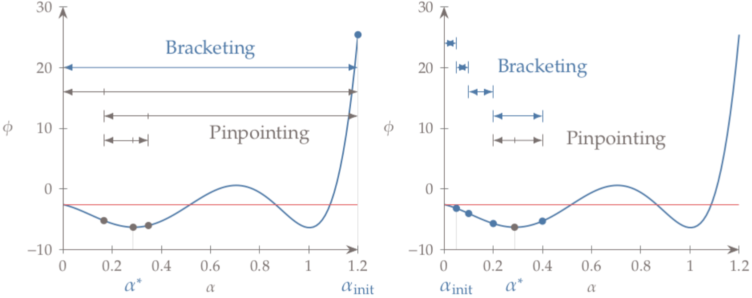
We use quadratic interpolation; the bracketing chooses $\alpha_{i+1} = 2\alpha_i$, and the sufficient curvature factor is $\mu_2 = 0.9$.



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`