

Statistics and Numerical Methods, Tsinghua University

Optimization methods

Xuening Bai (白雪宁)

Institute for Advanced Study (IASTU) & Department of Astronomy (DoA)



清華大學

Tsinghua University

Oct. 15, 2024

What is optimization

A subfield of applied math, on the selection of a best element (with regard to some criterion) from some set of available options.

Optimization is sometimes also called “mathematical programming”.

In the simplest case, it consists of maximizing/minimizing a real function by choosing some optimal input values within an allowed set of values.

Here we stick to minimization as a convention.

Given a function $f : A \rightarrow \mathbb{R}$ from some set A to real numbers.

Seek for an element $x_0 \in A$ so that $f(x) \leq f(x_0)$ for any $x \in A$.

Depending on the context, this function f is often called “energy function” (e.g., physics), “objective function”, or “loss function” (e.g., machine learning).

What is optimization

Given a function $f : A \rightarrow \mathbb{R}$ from some set A to real numbers.

Seek for an element $x_0 \in A$ so that $f(x) \leq f(x_0)$ for any $x \in A$.

Typically, A is a subset of \mathbb{R}^n , specified by some constraints. This is **constrained optimization**. Otherwise, the problem is called **unconstrained optimization**.

Applications:

- Computer science and machine learning
- Statistics and data science (e.g., data fitting)
- Engineering (electrical, civil, etc.)
- Physics, chemistry and biological science
- Economics and finance
- Operational research

Optimization/minimization vs root-finding

For a smooth function, minimization is about finding the root for

$$\mathbf{G}(\mathbf{x}) \equiv \nabla F(\mathbf{x}) = 0$$

In multi-dimensions, isn't minimization just root-finding?

No! Components of the gradient vector are **NOT** independent.

They obey so-called **integrability conditions** that are highly restrictive.

This makes minimization a (somewhat) easier task.

Outline

- Unconstrained optimization

 - 1D problems and methods

 - Multi-variate problems: CG and quasi-Newton methods

- Constrained optimization

 - Equality constraint: Lagrange multiplier

 - Optimality and KKT conditions

 - Convex programming

Example: non-linear least squares

We are given a series of data pairs (x_i, y_i) , and would like to fit it with a function

$$y = f(x, \boldsymbol{\theta}) \quad \text{where } \boldsymbol{\theta} \text{ represents parameters.}$$

To fit the parameters, one strategy is to minimize the least squares:


$$E(\boldsymbol{\theta}) = \sum_i [y_i - f(x_i, \boldsymbol{\theta})]^2$$

Usually, $f(x, \boldsymbol{\theta})$ can be nonlinear, and hence the problem is different from standard linear regression.

Example: maximum likelihood

A standard technique (which we will discuss later in the course) for parameter estimation is the **maximum likelihood method**.

The **likelihood function**:

$$\mathcal{L}(\mathbf{y}|\boldsymbol{\theta}) = \prod_i f(y_i|\boldsymbol{\theta})$$


data parameters

Goal: **maximize the likelihood** given the data within allowed parameter space.

Example: logistic regression

Similar to linear regression, but the event has only two discrete outcomes $\{0, 1\}$.

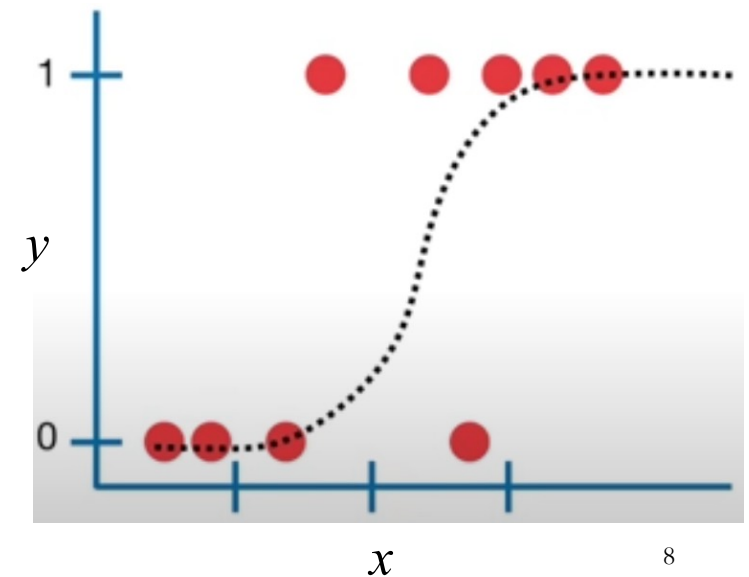
Fit the results with a **logistic function** (the primary type of a **sigmoid function**):

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \quad (\text{probability of the event taking value 1 at given } x)$$

Tune the coefficients β_0, β_1 to maximize the likelihood function:

$$L = \prod_{k:y_k=1} p_k \prod_{k:y_k=0} (1 - p_k)$$

Widely used in machine learning (binary classification) and other fields.

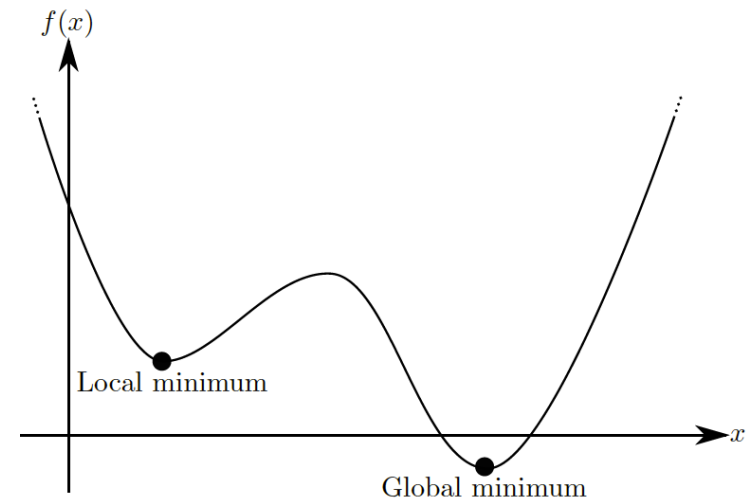


Optimality

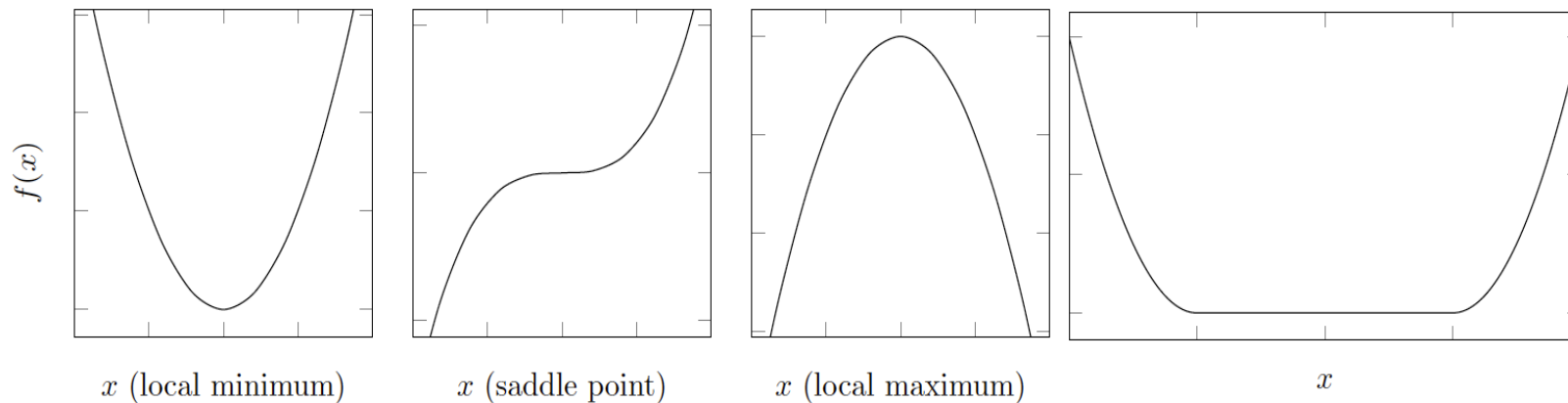
In most cases, we seek for **global minimum**, whereas it is usually much easier to find a **local minimum**.

If f is differentiable, a **necessary condition** for local minimum is simply

$$\nabla f(x_0) = 0$$



But this is **not sufficient**:



Optimality

If f is at least twice differentiable, one has

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + 0 + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T H(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$$

where the **Hessian matrix** is $H_{ij} \equiv \frac{\partial^2 f}{\partial x_i \partial x_j}$

If H is **positive definite**, then \mathbf{x}_0 is a **local minimum**.

If H is negative definite, then \mathbf{x}_0 is a local maximum.

If H is indefinite, the \mathbf{x}_0 is a saddle point.

If H is not invertible (including positive semidefinite), can be odd sometimes...

One can determine whether H is positive definite by checking if Cholesky factorization exists.

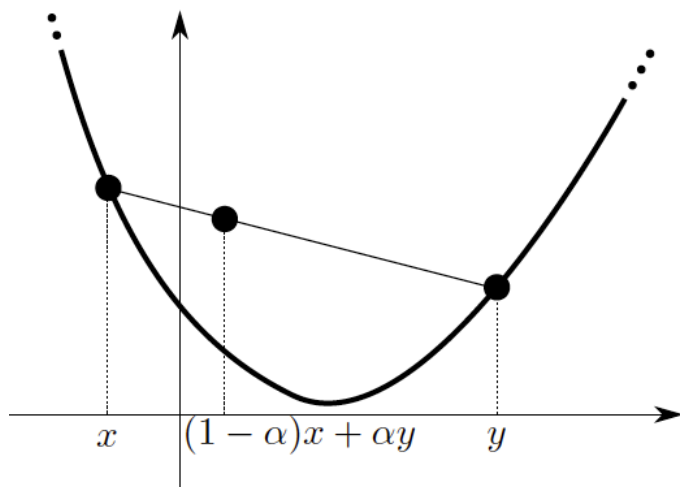
Convexity

A function is **convex** when for all x, y and $\alpha \in (0, 1)$, the following relation holds:

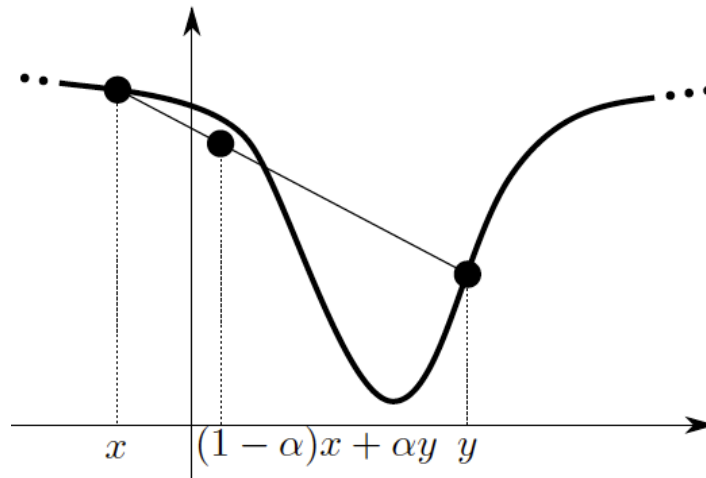
$$f[(1 - \alpha)x + \alpha y] \leq (1 - \alpha)f(x) + \alpha f(y)$$

It is called **quasi-convex** if

$$f[(1 - \alpha)x + \alpha y] \leq \max[f(x), f(y)]$$



convex



quasi-convex

A local minimum of a convex/quasi-convex function is necessarily a global minimum.

General remarks

For non-convex functions, finding global minimum is very difficult.

Heuristically, three ways around it:

- (1). Find local minima starting from a wide range of initial trial values, and pick the smallest among them.
- (2). Perturb a local minimum by finite amount, and see if it always returns.
- (3). Simulated annealing methods.

There is no perfect optimization algorithm. Wise to compare between different ones.

Should distinguish between problems where one can only evaluate the function, and those where derivatives are available.

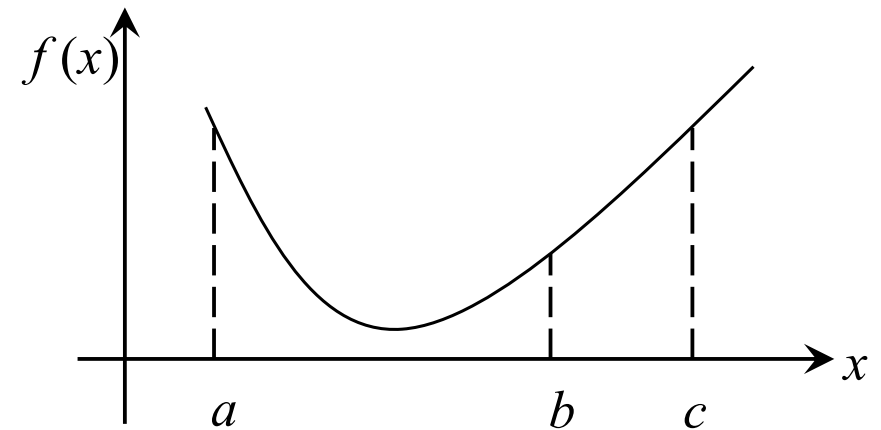
Algorithms using derivatives are somewhat more powerful.

Bracketing a minimum in 1D

It is essential to “bracketing” the minimum for 1D minimization.

A minimum is bracketed when there is a triplet of points $a < b < c$ such that:

$$f(b) < \min [f(a), f(c)]$$



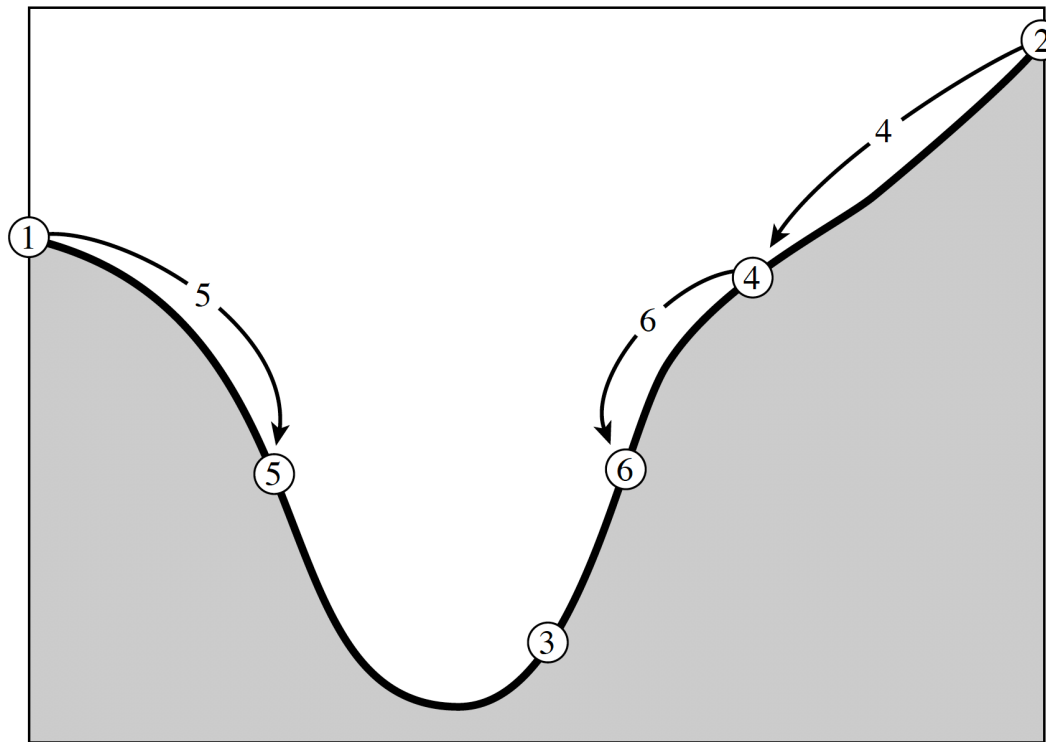
How to bracket the minimum given an arbitrary function f ?

- Start from two arbitrary locations a, b .
- Make successively larger steps towards downhill direction.
- (or) Make quadratic interpolations to the expected turning point.
- Continue until a minimum is bracketed.

Golden section search (1D)

Analog of bisection root finding in 1D, by “bracketing” the minimum.

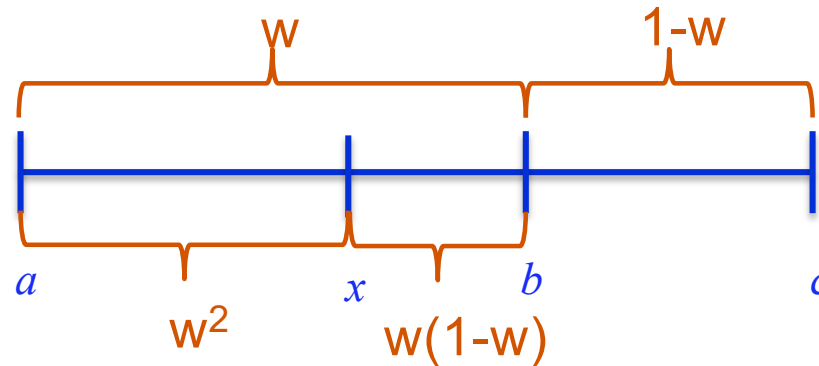
General strategy as follows, so that minimum is always bracketed.



But, where exactly to make divisions at each step?

Golden section search (1D)

Analog of bisection root finding in 1D, by “bracketing” the minimum.



We would like the process to be self-similar no matter $f(x) > f(b)$ [new triplet = (x, b, c)] or $f(x) < f(b)$ [new triplet = (a, x, b)], which requires

$$\frac{w}{1-w} = \frac{1-w}{w(1-w)} \Rightarrow w = (\sqrt{5} - 1)/2 \approx 0.618$$

Golden ratio!

Convergence is linear, and is slightly slower than bisection (0.618 vs 0.5).

Stopping criterion

When should you be satisfied with your solution?

Near the minimum, we have $f(x) \approx f(b) + \frac{1}{2}f''(b)(x - b)^2$

Let ϵ be machine precision. The question is when is the 2nd term negligible?

$$\frac{1}{2}|f''(b)|(x - b)^2 \lesssim \epsilon|f(b)|$$

$$\Rightarrow |x - b| < \boxed{\sqrt{\epsilon}|b|} \sqrt{\frac{2|f(b)|}{b^2 f''(b)}} \quad \leftarrow \text{order unity}$$

Don't waste your time to achieve machine precision!

Inverse parabolic interpolation

Like bisection, golden section search aims to handle the worst case scenarios. When the function is well-behaved, we can make it faster.

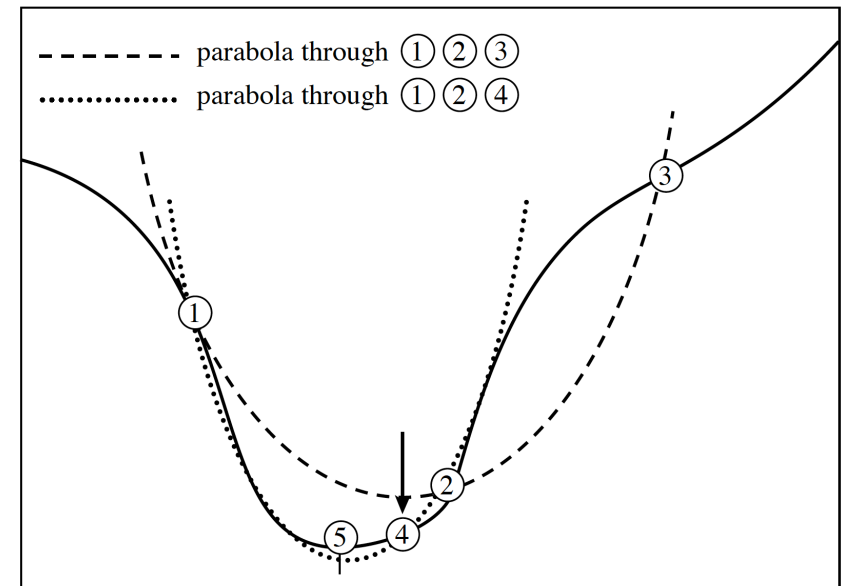
We expect a parabolic function near minimum



Fit such a function $y(x)$ that intersects $(a, f(a))$, $(b, f(b))$ and $(c, f(c))$.

Then x_{\min} of this function that gives the minimum y should approach the solution.

$$x = b - \frac{1}{2} \frac{(b-a)^2[f(b) - f(c)] - (b-c)^2[f(b) - f(a)]}{(b-a)[f(b) - f(c)] - (b-c)[f(b) - f(a)]}$$



Brent's method

In general, inverse parabolic interpolation cannot serve as a standalone scheme for minimization, and for instance, that x_{\min} can as well be x_{\max} !

This brings us to the [Brent's method](#), analog of that in 1D root finding.

[Hybrid scheme](#) that [combines golden section search with inverse parabolic interpolation](#).

More bookkeeping needed to ensure:

- The minimum is properly bracketed.
- Switch to golden section when inverse parabolic interpolation gets stuck (e.g., bouncing) or converges too slowly.
- Stop the iteration at some given error tolerance.

See [Numerical Recipes](#) for an implementation.

What if you also know the derivative?

In 1D, one can use, e.g., Newton's method to find the zeros of $f'(x)$:

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

Again, you want to be sufficiently close to the minimum before using this.

While one can use the value of f' and f'' to achieve higher order, more useful information is **the sign of f'** (at the current minimum point).

A conservative approach is a **modified Brent's method**:

*See Numerical Recipes
for an implementation.*

- The minimum is bracketed in a triplet of points.
- Try secant method in f' to proceed towards the minimum.
- Choose to accept (or not) the trial depending on the sign of the derivatives + other restrictions, or bisection in f' will be used.

Minimization in multidimensions

There is **no analogous procedure for bracketing in multi-D**.

The best one can do is to start from an initial guess and, via some iterative procedures, travel downhill the gradient towards a minimum.

Most multi-D algorithms make use of 1D algorithms:

A sequence of 1D minimization along some carefully chosen directions.

This is called “**line minimization**”:

Starting from \mathbf{x}_k , and given direction \mathbf{n} , minimize the scalar function:

$$g(\lambda) \equiv f(\mathbf{x}_k + \lambda \mathbf{n}) \Rightarrow \lambda = \lambda_{\min}$$

Then $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_{\min} \mathbf{n}$

Gradient descent

For a sufficiently smooth function, we can approximate f in quadratic form:

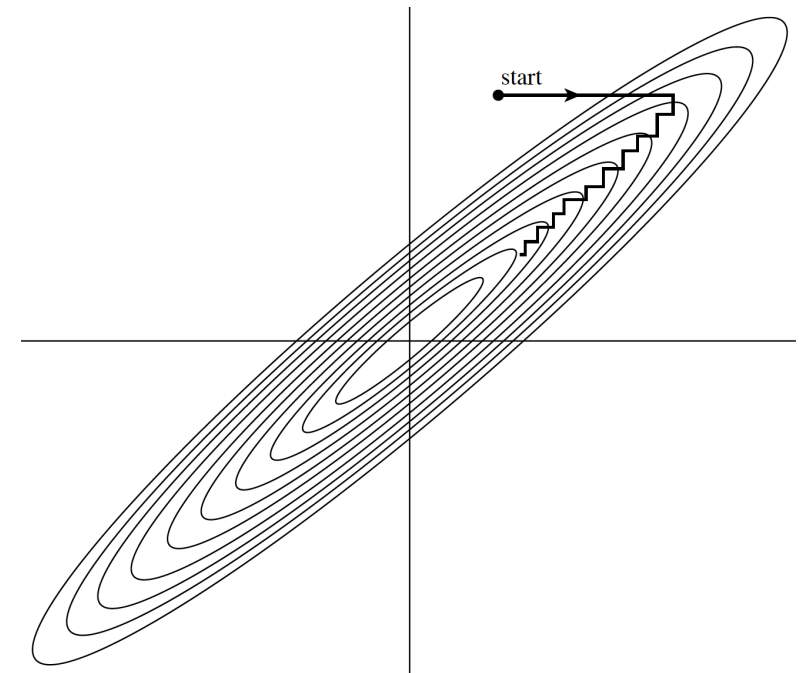
$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T H(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$$

Natural thought: move along a series of downhill against the gradient to find the minimum.

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

where α_k is set by [line minimization](#).

This generalizes the [steepest descent method](#). However, as we have seen earlier, it converges very slowly.



Conjugate gradient method

From the gradient vector, we see

$$\text{Change in gradient: } \delta f' \approx H \delta x$$

Suppose we have moved along some direction u to a minimum, and would next move along a new direction v .

To not interfere with previous effort, we want the change in gradient due to descent along v to be perpendicular to u , that is

$$u \cdot (Hv) = 0 \quad \text{or} \quad u^T H v = 0$$

This means that u and v are conjugate with respect to H .

By choosing conjugate directions, guaranteed to find the minimum *in this quadratic form* in N steps.

Conjugate gradient method

Recall the CG algorithm we had earlier for solving $A\mathbf{x}=\mathbf{b}$:

$$(1) \quad \mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \quad \text{with} \quad \alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T A \mathbf{p}_k}$$

$$(2) \quad \mathbf{r}_{k+1} = \mathbf{b} - A\mathbf{x}_{k+1} = \mathbf{r}_k - \alpha_k A \mathbf{p}_k$$

$$(3) \quad \mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k \quad \text{with} \quad \beta_k = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$$

For non-linear minimization problem:

(1) is replaced by a line minimization.

(2) is replaced by the calculation of derivative.

(3) is retained in its original form.

The CG method **can proceed without knowing the Hessian** matrix!

Conjugate gradient method

Here is the overall algorithm:

Starting from an arbitrary point \mathbf{x}_0 .

Calculate the gradient, and set $\mathbf{g}_0 = \mathbf{h}_0 = -\nabla f(\mathbf{x}_0)$.

Start iteration until convergence:

Perform line minimization along \mathbf{h}_i to obtain \mathbf{x}_{i+1} .

Calculate the gradient vector $\mathbf{g}_{i+1} = -\nabla f(\mathbf{x}_{i+1})$.

Determine the new direction $\mathbf{h}_{i+1} = \mathbf{g}_{i+1} + \gamma_i \mathbf{h}_i$.

The choice of γ_i :

$$\gamma_i = \frac{\mathbf{g}_{i+1}^T \mathbf{g}_{i+1}}{\mathbf{g}_i^T \mathbf{g}_i} \quad \text{or (better)} \quad \gamma_i = \frac{(\mathbf{g}_{i+1} - \mathbf{g}_i)^T \mathbf{g}_{i+1}}{\mathbf{g}_i^T \mathbf{g}_i}$$

(Fletcher-Reeves) (Polak & Ribier)

Quasi-Newton method

Newton's method without explicitly calculating the Hessian, [similar to Broyden's method](#) in solving non-linear systems.

Recall that with Newton's method we would be doing:

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

In [quasi-Newton method](#), we seek for

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k B_k^{-1} \nabla f(\mathbf{x}_k)$$

some limiting factor to
ensure gradient descent

some approximated
Hessian

The next approximated Hessian should satisfy

$$B_{k+1}(\mathbf{x}_{k+1} - \mathbf{x}_k) = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$$

Quasi-Newton method

Would like to iteratively obtain approximations to the Hessian.

Require the approximated Hessian to be symmetric and positive-definite.

Let $\mathbf{s}_k \equiv (\mathbf{x}_{k+1} - \mathbf{x}_k)$, $\mathbf{y}_k \equiv \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$ so that $B_{k+1}\mathbf{s}_k = \mathbf{y}_k$

The BFGS (*Broyden-Fletcher-Goldfarb-Shanno*) method:

$$B_{k+1}^{-1} = (I - \rho_k \mathbf{s}_k \mathbf{y}_k^T) B_k^{-1} (I - \rho_k \mathbf{y}_k \mathbf{s}_k^T) + \rho_k \mathbf{s}_k \mathbf{s}_k^T$$

$$\text{where } \rho_k \equiv (\mathbf{s}_k^T \mathbf{y}_k)^{-1} .$$

To initiate the iteration, can simply take $B_0 = I$.

Stochastic gradient descent

Many optimization formulations in data analysis are in this form:

$$E(\boldsymbol{\theta}) = \frac{1}{m} \sum_{j=1}^m h_{\boldsymbol{\theta}}(\mathbf{x}_j) + \lambda \Omega(\boldsymbol{\theta})$$

Objective model parameters data regularization/penalty

When m is very large, computing the full objective function can be impractical.

In each iteration, perform gradient descent on one or a small subset of m data terms.

This small subset is randomly selected => stochastic gradient descent.

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \sum_{j'} \nabla_{\boldsymbol{\theta}} [h_{\boldsymbol{\theta}}(\mathbf{x}_{j'}) + \lambda \Omega(\boldsymbol{\theta})] \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$

This is commonly adopted in machine learning and other large scale problems. Important to tune the “learning rate” α_k to achieve good performance.

What if I don't know the derivatives?

Use [numerical derivatives](#) for quasi-Newton's method.

[Downhill simplex method:](#)

A method that does not rely on 1D minimization.

Not as efficient, but simple to implement, good for small problems.

[Powell's direction set method:](#)

A way of systematically producing a set of mutually conjugate directions to perform 1D minimization, which achieves quadratic convergence.

Need some additional twists (which sacrifice performance) for stability.

See [Numerical Recipes](#) for details.

Outline

- Unconstrained optimization

 - 1D problems and methods

 - Multi-variate problems: CG and quasi-Newton methods

- Constrained optimization

 - Equality constraint: Lagrange multiplier

 - Optimality and KKT conditions

 - Convex programming

The overall problem

The problem is stated as:

Minimize $f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^n$

Subject to $g_i(\mathbf{x}) = 0$, $i = 1, \dots, m$

$h_j(\mathbf{x}) \geq 0$, $j = 1, \dots, p$

Equality constraints

Inequality constraints

A **feasible point** is any point that satisfies all constraints above.

The **feasible set** is the set of all feasible points.

Constrained optimization problems are much more difficult.

Example: manufacturing

Suppose you have N different materials, and s_i units of each i in stock.

You can manufacture M different products, with product j using c_{ij} of material i , and gives you profit p_j .

How to maximize profit?

Objective function: $E(\mathbf{x}) = - \sum_{j=1}^M p_j x_j$ (for minimization)

Subject to: $x_j \geq 0$ (for $j=1, \dots, M$)

$\sum_{j=1}^M c_{ij} x_j \leq s_i$ (for $i=1, \dots, N$)

Inequality constraints

Two yellow arrows point from the two constraint equations to the text 'Inequality constraints'.

This is an example of a **linear program**.

Example: chemical equilibrium

While chemical equilibrium can be established by evolving time-dependent kinetic equations for sufficiently long time, it is often more efficient to directly solve the equilibrium by **minimizing** the **Gibbs free energy**.

At given pressure and temperature, the total Gibbs free energy is

$$G = RT \cdot \underbrace{\sum_{k=1}^m n_k^g \left[\left(\frac{\mu^\circ}{RT} \right)_k^g + \ln P + \ln x_k^g \right]}_{\text{gas phase}} + \underbrace{\sum_{k=1}^s n_k^c \left(\frac{\mu^\circ}{RT} \right)_k^c}_{\text{condensed phase}}$$

This is subject to the conservation of elemental abundances:

$$\sum_{k=1}^m a_{kj}^g n_k^g + \sum_{k=1}^c a_{kj}^c n_k^c = b_j, \quad j = 1, \dots, NC. \quad \leftarrow \text{Equality constraint}$$

And $n_k \geq 0$ for all k . \leftarrow Inequality constraint

Problems with only equality constraints

Let us first consider the case with **only equality constraints**.

The problem is stated as:

$$\begin{array}{ll} \text{Minimize} & f(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n \\ \text{Subject to} & g_i(\mathbf{x}) = 0, \quad i = 1, \dots, m \end{array}$$

Let \mathbf{x}^* be the desired minimum. Any slightly deviation from it (by $\delta\mathbf{x}$) must satisfy:

$$\delta\mathbf{x} \cdot \nabla f(\mathbf{x}^*) = 0$$

Given the constraints, $\delta\mathbf{x}$ is limited to

$$g_i(\mathbf{x}^* + \delta\mathbf{x}) = g_i(\mathbf{x}^*) = 0 \quad \text{therefore} \quad \delta\mathbf{x} \cdot \nabla g_i(\mathbf{x}^*) = 0$$

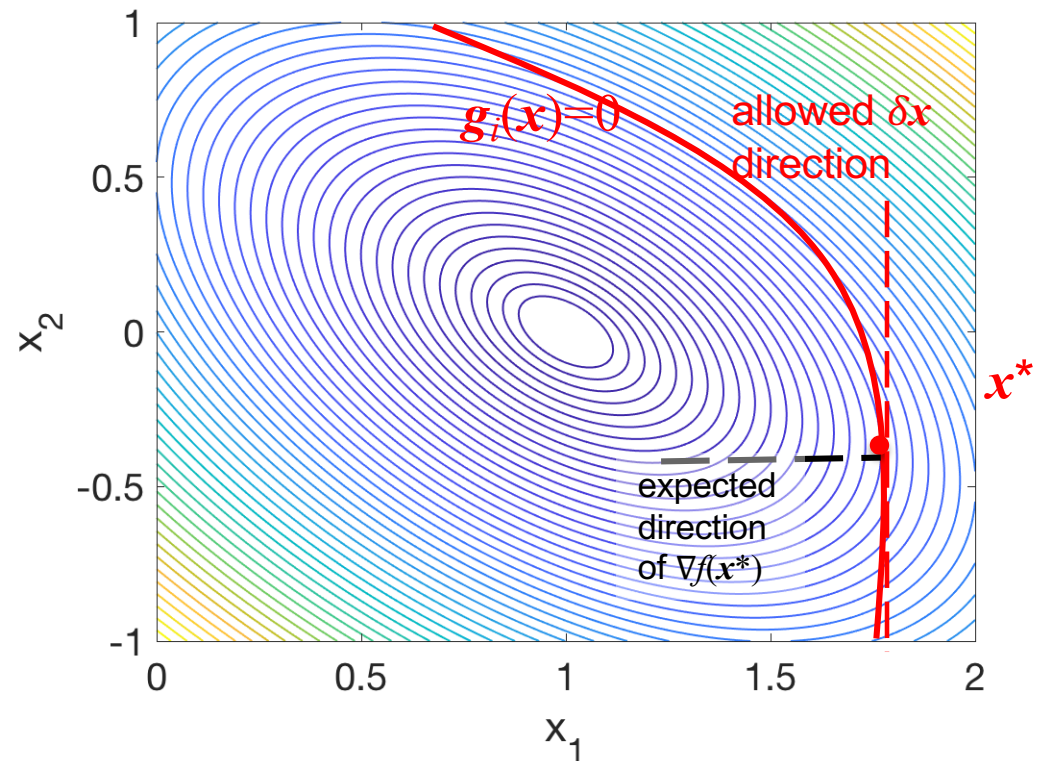
Problems with only equality constraints

In sum, near \mathbf{x}^* , for any $\delta\mathbf{x}$ satisfying

$$\delta\mathbf{x} \cdot \nabla g_i(\mathbf{x}^*) = 0$$

We must have

$$\delta\mathbf{x} \cdot \nabla f(\mathbf{x}^*) = 0$$



This suggests that **at the minimum**, ∇f should be a linear combination of all (∇g_i) s.

Lagrange multipliers

We can define a **Lagrange multiplier** function:

$$\Lambda(\mathbf{x}, \boldsymbol{\lambda}) \equiv f(\mathbf{x}) - \boldsymbol{\lambda} \cdot \mathbf{g}(\mathbf{x})$$

The original problem becomes **equivalent** to an **unconstrained optimization problem** on $\Lambda(\mathbf{x}, \boldsymbol{\lambda})$:

At the minimum, we have

$$\frac{\partial \Lambda}{\partial \lambda_i} = -g_i(\mathbf{x}) = 0 \quad , \text{ thus satisfying all constraints}$$

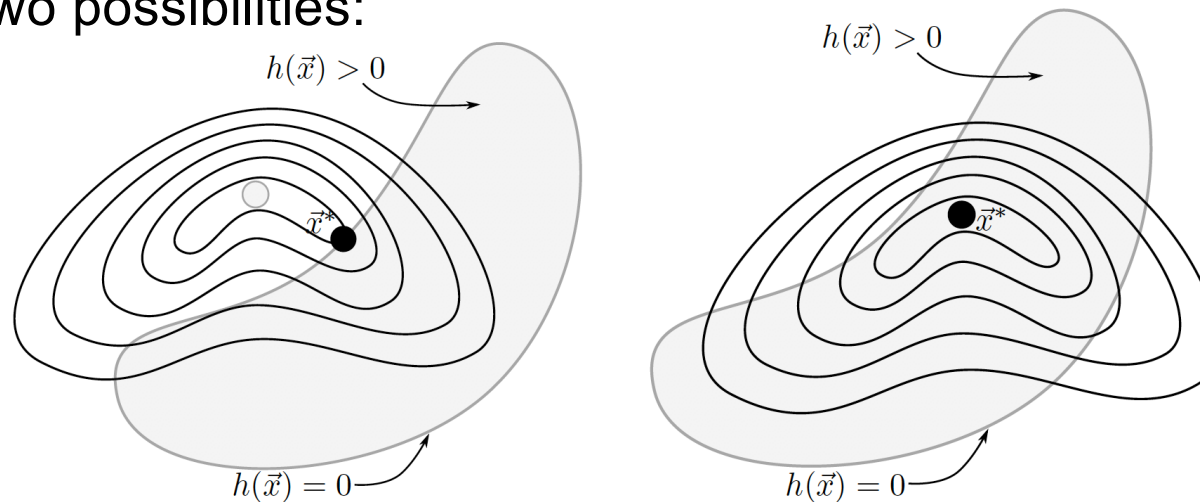
In the meantime,

$$\nabla_{\mathbf{x}} \Lambda(\mathbf{x}, \boldsymbol{\lambda}) = \nabla f(\mathbf{x}) - \sum_{i=1}^m \lambda_i \nabla g_i(\mathbf{x}) = 0$$

Optimality with inequality constraints

In the presence of inequality constraints, we need to find a way to add them to the Lagrange multiplier system.

Let the minimum subject to constraints be \mathbf{x}^* . For inequality constraints $h(\mathbf{x}^*) \geq 0$, there are two possibilities:



Active $h_j(\mathbf{x}^*) = 0$

Inactive $h_j(\mathbf{x}^*) > 0$

Though we do not know *a priori* which constraints will be active/inactive.

KKT (Karush-Kuhn-Tucker) conditions for optimality

Let us generalize the Lagrange multiplier to

$$\Lambda(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \equiv f(\mathbf{x}) - \boldsymbol{\lambda} \cdot \mathbf{g}(\mathbf{x}) - \boldsymbol{\mu} \cdot \mathbf{h}(\mathbf{x})$$

The corresponding **stationarity condition** is:

$$\nabla f(\mathbf{x}^*) = \sum_{i=1}^m \lambda_i \nabla g_i(\mathbf{x}^*) + \sum_{j=1}^p \mu_j \nabla h_j(\mathbf{x}^*)$$

However, we should impose additional constraints for inequality terms:

Complementary slackness: $\mu_j h_j(\mathbf{x}^*) = 0$ for all $j = 1, \dots, p$

Dual feasibility: $\mu_j \geq 0$ for all $j = 1, \dots, p$

Of course, there are also the original constraints on \mathbf{g} and \mathbf{h} , called **primal feasibility**.

Linear programming (LP)

The problem is to minimize:

$$\zeta = c_1x_1 + c_2x_2 + \cdots + c_nx_n \quad \text{objective function}$$

Subject to non-negativity conditions:

$$x_1 \geq 0, \ x_2 \geq 0, \ \dots \ x_n \geq 0$$

and simultaneously subject to m additional constraints of the form

$$\begin{aligned} & a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_m \leq b_i \\ \text{or} \quad & a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_m = b_i \end{aligned} \quad (i=1,\dots,m)$$

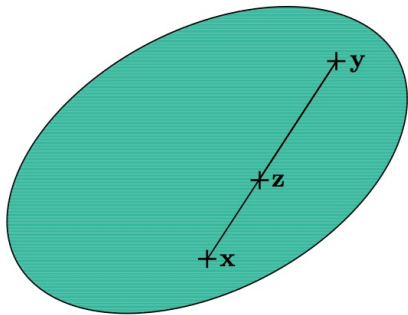
Extremely wide range of applications.

Special and efficient algorithms exist for LP, most notably the [simplex algorithm](#) (Dantzig 1948) and the [interior points methods](#) (Karmarkar 1984).

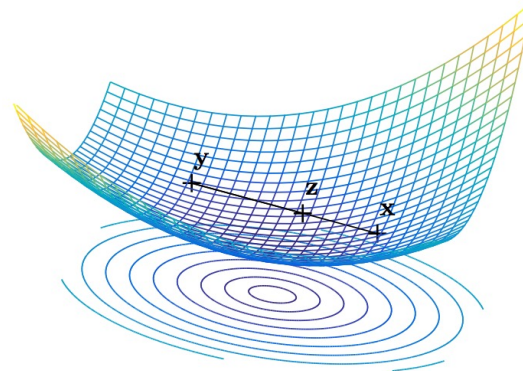
Convex optimization

See book “Convex Optimization” by Boyd & Vandenberghe

Convex set

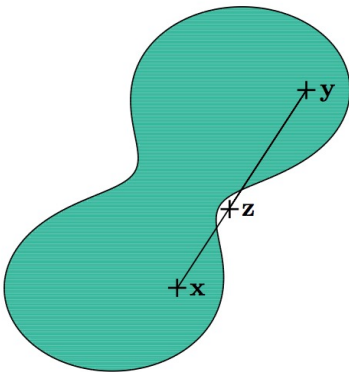


Convex function

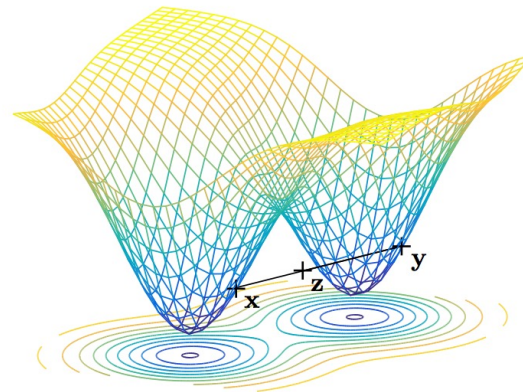


Linear programming belongs to a much broader class of optimization problem known as **convex optimization**.

Non-convex set



Non-convex function



Subfield of optimization where **both the objective function and the constraint functions are convex**.

Local minimum is global minimum.

Convex optimization

See book “Convex Optimization” by Boyd & Vandenberghe

Sub-categories include:

Linear programming (LP), (convex) quadratic programming (QP), quadratically constrained quadratic programming (QCQP), second-order cone programming (SOCP), semi-definite programming (SDP), etc.

In general, algorithms for convex optimization is close to maturity.

If you manage to reduce a problem to a convex optimization problem, you can basically say it is solved!

Notable software packages for convex programming:

[CVX](#) (matlab-based), [CVXOPT](#) (python based), [Convex.jl](#) (Julia based)

Quadratic programming (QP)

Quadratic programming:

The problem is to minimize: $\frac{1}{2}\mathbf{x}^T Q \mathbf{x} + \mathbf{c}^T \mathbf{x}$

Subject to conditions: $G\mathbf{x} \leq \mathbf{h}$, $A\mathbf{x} = \mathbf{b}$

Quadratically-constrained quadratic programming:

The problem is to minimize: $\frac{1}{2}\mathbf{x}^T P_0 \mathbf{x} + \mathbf{q}_0^T \mathbf{x}$

Subject to conditions: $\frac{1}{2}\mathbf{x}^T P_i \mathbf{x} + \mathbf{q}_i^T \mathbf{x} + r_i \leq 0$, $A\mathbf{x} = \mathbf{b}$

Convex requires Q and P_i be **positive semi-definite**.

Barrier method

An important type of **interior-point method**.

Replace inequality constraints by a **penalty function** that diverges at the barrier.

Take linear programming for example:

Minimize: $\mathbf{c} \cdot \mathbf{x}$

Subject to: $A\mathbf{x} = \mathbf{b}$, $\mathbf{x} \geq 0$

For inequality constraint, replace it by a **logarithmic barrier function**: $\sum_{j=1}^n \log x_j$

Now, we consider minimizing: $E(\mathbf{x}) = \mathbf{c} \cdot \mathbf{x} - \tau \sum_{j=1}^n \log x_j$

We see that it recovers the original problem as $\tau \rightarrow 0$.

Barrier method

The problem now becomes:

$$\begin{aligned} \text{Minimizing: } & E(\mathbf{x}) = \mathbf{c} \cdot \mathbf{x} - \tau \sum_{j=1}^n \log x_j \\ \text{Subject to: } & A\mathbf{x} = \mathbf{b} \end{aligned}$$

This is an equality constrained problem!

The problem is then solved by iteration via standard Lagrange multiplier (which approaches KKT conditions).

The solution as a function of τ defines a **central path**. In practice, τ is reduced as iteration proceeds, so that it is **path-following** to achieve fast convergence.

It is generalizable to more general cases in convex programming.

Towards nonlinear problems

Nonlinear optimization describes optimization problems when the objective or constraint functions are **nonlinear**, and are **not known to be convex**.

Like non-linear root finding, no effective methods to solve the general problem.

Sequential quadratic programming (SQP):

- Approximate a general nonlinear problem to a quadratic program.

- Require the constraint (objective) functions to be (twice) differentiable.

- Solve the approximated problem and adjust the approximation based on the latest function evaluation and its derivatives as iteration proceeds.

SQP methods are available in software packages such as SciPy, matlab, etc.

Summary

- Optimization = minimization w/wo constraints
- Unconstrained optimization

1D methods: golden section search, Brent's method

Multi-D case: gradient descent / CG / quasi-Newton + line search; stochastic gradient descent in ML

- Constrained optimization with equality constraints
Use Lagrange multiplier to convert to unconstrained problem.
- Constrained optimization with inequality constraints

KKT conditions for optimality

Convex programming: interior-point method (with barrier function)

Nonlinear problems: sequential quadratic programming

Further reading: simulated annealing method

Metaheuristic methods: heuristically-motivated methods without rigorous mathematical proof, but some methods can be very efficient to find **near-optimal** solutions in practice.

Simulated annealing is a probabilistic technique for *approximating* global minimum. It is often used when search space is finite/discrete (**combinatorial problem**), but can also be used for continuous problems.

Motivated from the analogy thermodynamics (metal annealing, liquid crystallization).

Instead of pursuing downhill descent, one can occasionally go uphill into other states, according to a probability distribution

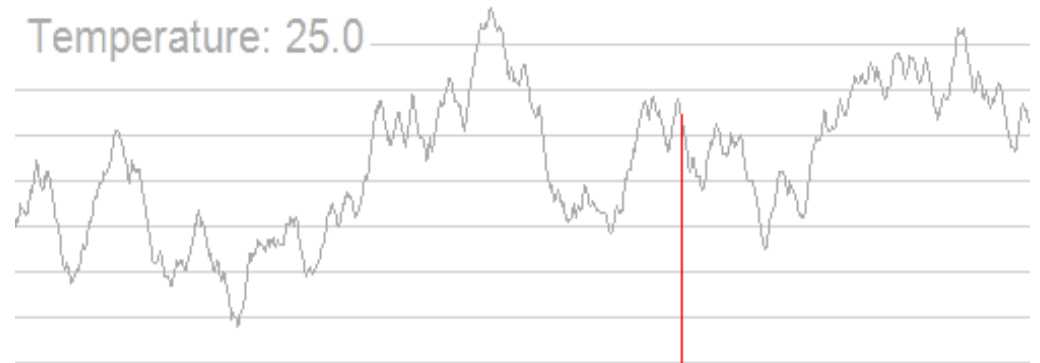
$$p \propto \exp(-E/kT)$$

Idea was proposed by Metropolis et al. (1953), and was later generalized to Metropolis-Hastings algorithm (1970) for MCMC.

Metropolis algorithm

A simulated thermodynamic system changes its configuration from energy E_1 to E_2 , with a transition probability

$$p = \min[1, e^{-(E_2 - E_1)/kT}]$$



To run this method, one needs:

- A description of possible system configurations.

- A generator of random changes in the configuration.

- An **objective function E** to be minimized.

- A **control parameter T** and an **annealing schedule** that tells how T is lowered from high to low values (which needs to be experimented).

- Can sometimes benefit from “restart”: discard previous iteration results.

Traveling salesman problem (TSP)

A salesman visits N cities with given positions, and return to his city of origin. Each city is to be visited only once, and the route should be as short as possible.

To solve it with simulated annealing:

1. Label each city with $0, \dots, N-1$, and a configuration is an ordering of them.
2. Rearrangement by selecting some section of path and 1). reversing its order; 2). insert it to between two other cities.
3. The choose an objective function (total distance), and an annealing schedule.

