Local search algorithms

Chapter 4, Sections 3–4

Outline

- ♢ Hill-climbing
- ♢ Simulated annealing
- ♢ Genetic algorithms (briefly)
- ♢ Local search in continuous spaces (very briefly)

Iterative improvement algorithms

In many optimization problems, **path** is irrelevant;
the goal state itself is the solution

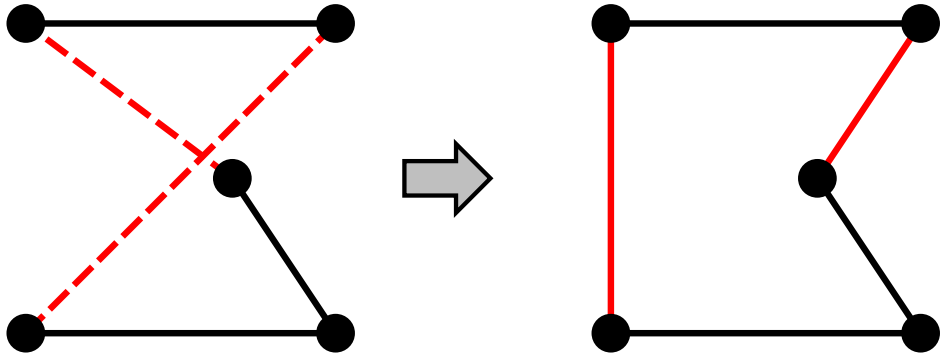
Then state space = set of "complete" configurations;
find **optimal** configuration, e.g., TSP
or, find configuration satisfying constraints, e.g., timetable

In such cases, can use iterative improvement algorithms;
keep a single "current" state, try to improve it

Constant space, suitable for online as well as offline search

Example: Travelling Salesperson Problem

Start with any complete tour, perform pairwise exchanges

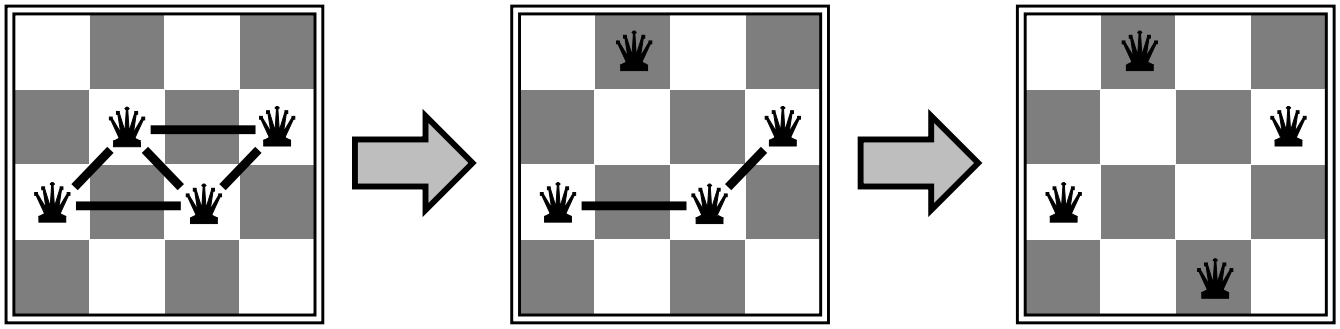


Variants of this approach get within 1% of optimal very quickly with thousands of cities

Example: n-queens

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

Move a queen to reduce number of conflicts



Almost always solves n-queens problems almost instantaneously for very large n, e.g., $n = 1million$

Hill-climbing (or gradient ascent/descent)

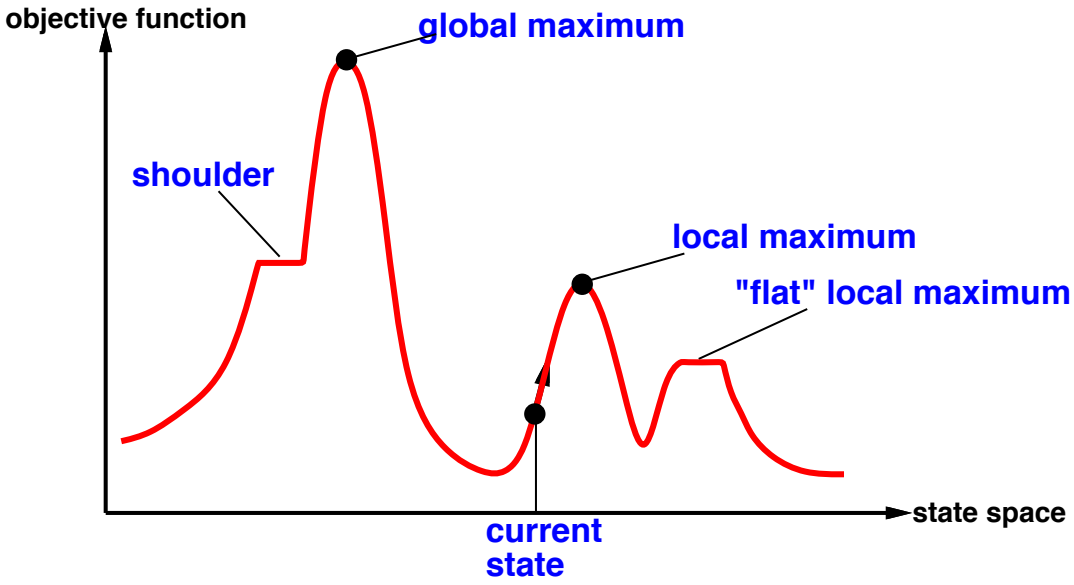
"Like climbing Everest in thick fog with amnesia"

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
    end
```

Hill-climbing contd.

Useful to consider state space landscape



Random-restart hill climbing overcomes local maxima—trivially complete

Random sideways moves ☺ escape from shoulders ☹ loop on flat maxima

Simulated annealing

Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and frequency

function SIMULATED-ANNEALING(*problem, schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling prob. of downward steps

  current $\leftarrow$ MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow$ *schedule*[t]
    if $T = 0$ **then return** *current*
    next $\leftarrow$ a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* $\leftarrow$ *next*
    else *current* $\leftarrow$ *next* only with probability $e^{\Delta\ E/T}$

Properties of simulated annealing

At fixed "temperature" T, state occupation probability reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

T decreased slowly enough $\Longrightarrow$ always reach best state x^*
because $e^{\frac{E(x^*)}{kT}} / e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small T

Is this necessarily an interesting guarantee??

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.

Local beam search

Idea: keep k states instead of 1; choose top k of all their successors

Not the same as k searches run in parallel!
Searches that find good states recruit other searches to join them

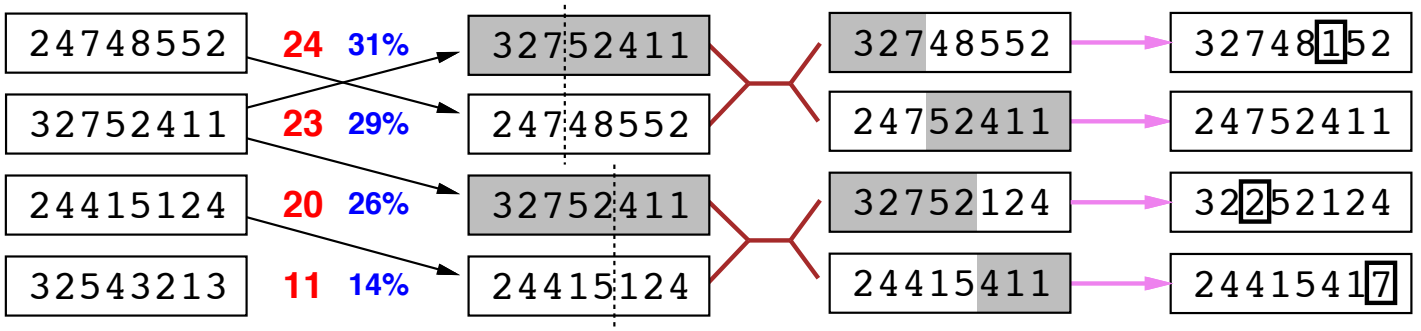
Problem: quite often, all k states end up on same local hill

Idea: choose k successors randomly, biased towards good ones

Observe the close analogy to natural selection!

Genetic algorithms

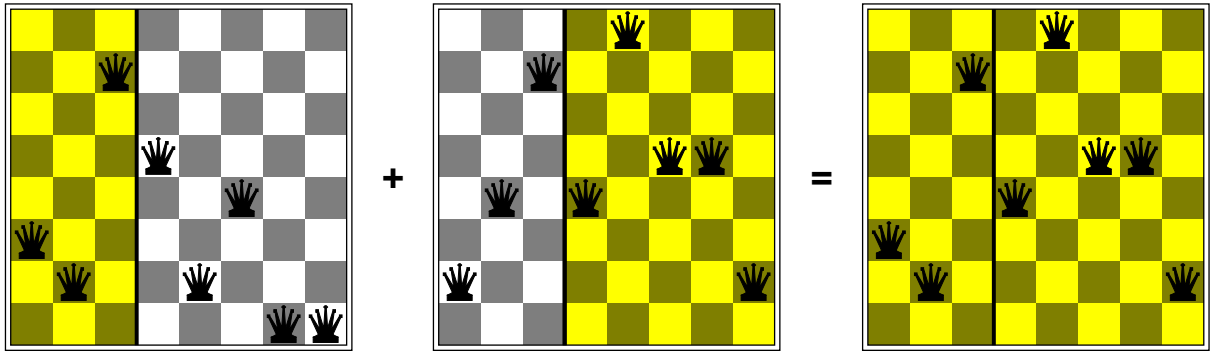
= stochastic local beam search + generate successors from **pairs** of states



Genetic algorithms contd.

GAs require states encoded as strings (GPs use programs)

Crossover helps **iff substrings are meaningful components**



GAs $\neq$ evolution: e.g., real genes encode replication machinery!

Continuous state spaces

Suppose we want to site three airports in Romania:
- 6-D state space defined by (x_1, y_2), (x_2, y_2), (x_3, y_3)
- objective function $f(x_1, y_2, x_2, y_2, x_3, y_3) =$
  sum of squared distances from each city to nearest airport

Discretization methods turn continuous space into discrete space,
e.g., empirical gradient considers $\pm\delta$ change in each coordinate

Gradient methods compute

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right)$$

to increase/reduce f, e.g., by $\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$

Sometimes can solve for $\nabla f(\mathbf{x}) = 0$ exactly (e.g., with one city).
Newton–Raphson (1664, 1690) iterates $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x})$
to solve $\nabla f(\mathbf{x}) = 0$, where $\mathbf{H}_{ij} = \partial^2 f / \partial x_i \partial x_j$

Gradiente e matrice Hessiana

The following slides on the definition of gradient are partially from

The MIT OpenCourseWare

http://ocw.mit.edu

18.02SC Multivariable Calculus Fall 2010

I lettori sono invitati a seguire il corso on-line del MIT per maggiori informazioni.

Gradiente

For functions $w = f(x, y, z)$ we have the gradient

$$\operatorname{grad} w = \nabla w = \left\langle \frac{\partial w}{\partial x}, \frac{\partial w}{\partial y}, \frac{\partial w}{\partial z} \right\rangle.$$

That is, the gradient takes a scalar function of three variables and produces a three dimensional vector.

The gradient has many geometric properties. In the next session we will prove that for $w = f(x, y)$ the gradient is perpendicular to the level curves $f(x, y) = c$. We can show this by direct computation in the following example.

Gradiente

Il gradiente di una funzione scalare $f(x_1, x_2, x_3)$ definita nel sistema di riferimento cartesiano ortonormale è un campo vettoriale che associa ad ogni punto (x_0, y_0, z_0) il vettore che ha per componenti le derivate parziali prime di f calcolare nel punto.

La definizione può essere estesa a funzioni definite in uno spazio euclideo di dimensione arbitraria.

Il prodotto scalare del gradiente di f per un vettore unitario $\mathbf{v}$ è eguale alla derivata direzionale di f lungo $\mathbf{v}$ che intuitivamente ci dice quanto la funzione f cresce lungo il vettore $\mathbf{v}$ (nel punto x).

La formula è la seguente:

$$\big(\nabla f(x)\big) \cdot \mathbf{v} = D_{\mathbf{v}} f(x).$$

Esempio

Example 1: Compute the gradient of $w = (x^2 + y^2)/3$ and show that the gradient at $(x_0, y_0) = (1, 2)$ is perpendicular to the level curve through that point.

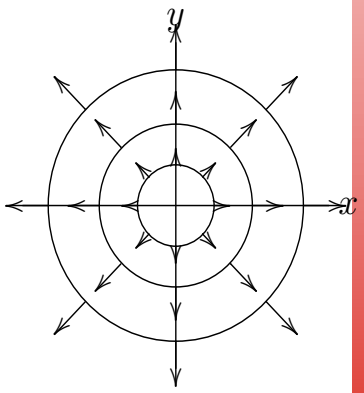
Answer: The gradient is easily computed

$$\nabla w = \langle 2x/3, 2y/3 \rangle = \frac{2}{3}\langle x, y \rangle.$$

At $(1, 2)$ we get $\nabla w(1, 2) = \frac{2}{3}\langle 1, 2 \rangle$. The level curve through $(1, 2)$ is

$$(x^2 + y^2)/3 = 5/3,$$

which is identical to $x^2 + y^2 = 5$. That is, it is a circle of radius $\sqrt{5}$ centered at the origin. Since the gradient at (1,2) is a multiple of $\langle 1, 2 \rangle$, it points radially outward and hence is perpendicular to the circle. Below is a figure showing the gradient field and the level curves.



$z = (x^2 + y^2)/3$

Higher dimensions

Similarly, for $w = f(x, y, z)$ we get level surfaces $f(x, y, z) = c$. The gradient is perpendicular to the level surfaces.

Example 3: Find the tangent plane to the surface $x^2 + 2y^2 + 3z^2 = 6$ at the point $P = (1, 1, 1)$.

Answer: Introduce a new variable

$$w = x^2 + 2y^2 + 3z^2.$$

Our surface is the level surface $w = 6$. Saying the gradient is perpendicular to the surface means exactly the same thing as saying it is normal to the tangent plane. Computing

$$\boldsymbol{\nabla} w = \langle 2x, 4y, 6z \rangle \;\Rightarrow\; \boldsymbol{\nabla} w|_P = \langle 2, 4, 6 \rangle.$$

Using point normal form we get the equation of the tangent plane is

$$2(x-1) + 4(y-1) + 6(z-1) = 0, \quad \text{or} \quad 2x + 4y + 6z = 12.$$

Matrice Hessiana

La matrice Hessiana di una funzione f:R^n -> R è la matrice H(f)

definita come segue (asummendo l'esistenza delle derivate seconde)

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Outline

Ottimizzazione numerica

I metodi di ottimizzazione consentono di trovare i punti estremanti (massimi, minimi) di una *funzione obiettivo* $f : \mathbb{R}^n \to \mathbb{R}$.

Poiché $\max f(x) = -\min(-f(x))$, è sufficiente concentrarsi sui metodi per la ricerca del minimo

$$\min_{x \in \Omega \subseteq \mathbb{R}^n} f(x)$$

dove Ω è l'insieme definito dai *vincoli*, se presenti.

Per semplicità, nel seguito supporremo l'esistenza e l'unicità del punto di minimo x^* (locale e globale), e tratteremo l'ottimizzazione in assenza di vincoli.

Metodi numerici per l'ottimizzazione

Ecco i metodi numerici che vedremo e implementeremo in R:

- per funzioni univariate:
  - golden section search
  - gradiente discendente
  - Newton-Raphson
- per funzioni multivariate:
  - Newton-Raphson
  - `optim` (built-in R function)

Golden section search

Algoritmo piuttosto semplice e intuitivo per la ricerca del minimo di una funzione all'interno di un intervallo $[a, b]$ con $a < b$ e $x^* \in [a, b]$.

Richiede che la funzione f sia unimodale.

Definizione

Funzione $f : \mathbb{R} \to \mathbb{R}$ è *unimodale* se esiste $m \in \mathbb{R}$ tale che f è decrescente per $x \leq m$ e crescente per $x \geq m$.
m è quindi il punto di minimo (globale).

Golden section search

Ecco come funziona il **metodo golden section search**:

1. Si parte considerando l'intervallo $[a_0, b_0] := [a, b]$ e si fissa una tolleranza ϵ
2. Si restringe iterativamente l'intervallo a $[a_{k+1}, b_{k+1}] \subset [a_k, b_k]$ in modo che il punto di minimo x^* continui a cadere al suo interno
3. Ci si arresta quando $|b_k - a_k| < \epsilon$, e si accetta $\frac{b_k - a_k}{2}$ come una buona approssimazione del punto di minimo x^*

Al passo 2, si determinano $x_1, x_2 \in [a_k, b_k]$ tali che $x_1 < x_2$.
Poiché la funzione obiettivo è unimodale e abbiamo supposto esserci un unico punto di minimo $x^* \in [a, b]$,
se succede $f(x_1) > f(x_2)$ allora significa $x^* \in [x_1, b_k]$,
altrimenti se $f(x_1) < f(x_2)$ allora $x^* \in [a_k, x_2]$.
Selezionato l'intervallo corretto, si prosegue in modo iterativo.

Golden section search

Nota

Il nome del metodo è dovuto alla proporzione aurea esistente tra le distanze dei punti scelti dagli estremi dell'intervallo: $\frac{x_2 - x_1}{x_1 - a} = \frac{x_1 - a}{b - x_1}$

Osservazione

- **pro**: non richiede la conoscenza delle derivate
- **contro**: convergenza lenta

Esempi

Esempio 2

$f(x) = |x - 3| + \sin x - |x - 2|$
Determinare minimi/massimi e osservare il comportamento del metodo golden section search al variare dell'intervallo di ricerca.

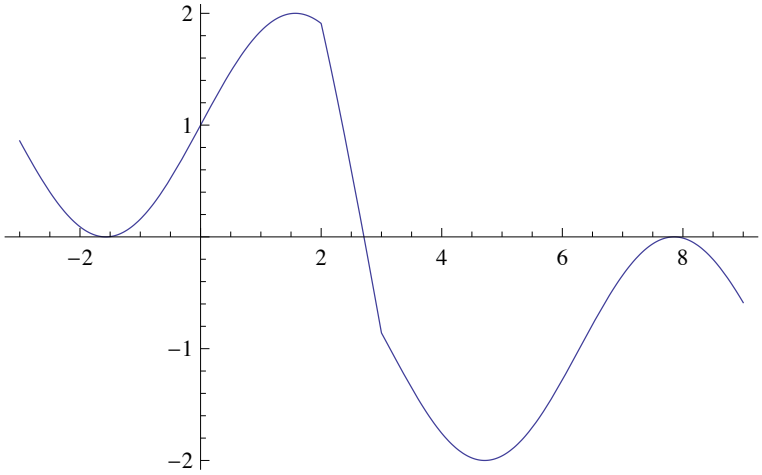


Figure: $f(x) = |x - 3| + \sin x - |x - 2|$

Esempi

La funzione dell'esempio 2 ha due minimi e due massimi nell'intervallo $[-3, 9]$.

A seconda dell'intervallo di ricerca scelto, il metodo converge a uno dei minimi:

- $[-3, 9]$: due massimi, due minimi; converge a uno dei minimi
- $[0, 9]$: un massimo, un minimo; converge al minimo
- $[3, 6]$: un minimo; converge a quello

Introduzione ai metodi di discesa

I **metodi di discesa** sono metodi iterativi che, a partire da un iterato iniziale $x_0 \in \mathbb{R}^n$, generano una successione di punti $\{x_k\}_{k \in \mathbb{N}}$ definiti dall'iterazione

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

dove il vettore p_k è una *direzione di ricerca* e lo scalare α_k è un parametro positivo chiamato *lunghezza del passo* (step length) che indica la distanza di cui ci si muove lungo la direzione p_k.

Introduzione ai metodi di discesa

In un metodo di discesa, il vettore p_k e il parametro α_k sono scelti in modo da garantire la decrescita della funzione obiettivo f a ogni iterazione:

$$f(x_{k+1}) < f(x_k) \qquad \forall k \geq 0$$

In particolare, come vettore p_k si prende una *direzione di discesa*, cioè tale che la retta $x = x_k + \alpha_k p_k$ forma un angolo ottuso con il vettore gradiente $\nabla f(x_k)$. In questo modo è possibile garantire la decrescita di f, purché α_k sia sufficientemente piccolo.

Scelta della direzione di discesa

A seconda della scelta di p_k si hanno diversi metodi di discesa.

Noi vedremo:

- metodo del gradiente discendente
- metodo di Newton-Raphson

Scelta della lunghezza del passo

Esistono diverse tecniche per la scelta di α_k in modo tale da garantire la convergenza del metodo verso punti stazionari di f.

Step length:

- troppo piccola: può tradursi in una convergenza lenta
- troppo grande: rischio di "superare" il punto di minimo

Gradiente discendente

Metodo del gradiente discendente:

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

direzione di ricerca (gradiente): $p_k = -f'(x_k)$
step length: $\alpha_k = \delta$ costante

Richiede $f \in C^1$ (derivabile con continuità), e conoscenza dell'espressione analitica della derivata prima.

Osservazione

Nome dovuto al fatto che si sceglie come direzione di ricerca quella opposta al gradiente (derivata prima, nel caso univariato).

Gradiente discendente

Costo computazionale

1 valutazione della derivata prima per iterazione.

Osservazione

- **pro**:
  - converge a punti di minimo (e non genericamente a punti stazionari)
  - non richiede il calcolo della derivata seconda (matrice Hessiana nel caso multivariato)
- **contro**:
  - alto numero di iterazioni
  - sensibilità rispetto all'iterato iniziale (se x_0 è preso troppo lontano dal minimo x^*, e δ è scelta piccola, il metodo può non arrivare alla soluzione in tempi ragionevoli)

Newton-Raphson

Metodo di Newton-Raphson:

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

direzione di ricerca (Newton): $p_k = -f'(x_k)/f''(x_k)$
step length: $\alpha_k = 1$

Ha la forma del metodo di Newton per la ricerca degli zeri di una funzione, applicato a f', in quanto determinare il punto di minimo della funzione f equivale a determinare la radice della derivata prima f'.

Newton-Raphson

Il metodo Newton-Raphson è solitamente preferito rispetto al gradiente discendente, per la sua velocità, nonostante richieda $f \in C^2$, $f'' \neq 0$, conoscenza dell'espressione analitica delle derivate prima e seconda, e converga indistintamente a minimi e massimi.

Nota

Se f ha un'espressione semplice, la function `deriv` consente di calcolarne simbolicamente le derivate prima e seconda.

Newton-Raphson

Costo computazionale

A ogni iterazione: valutazione di derivata prima e seconda.

Osservazione

- **pro**:
  - convergenza veloce (quadratica)
- **contro**:
  - convergenza locale
  - alto costo per iterazione

Esistono varianti che rendono il metodo a convergenza globale e che abbassano il costo computazionale evitando di risolvere con metodi diretti il sistema per la determinazione di p_k.

Newton-Raphson

Dall'espansione di Taylor nel caso multidimensionale, il **metodo di Newton-Raphson multivariato** assume la forma:

$$x_{k+1} = x_k - H_f(x_k)^{-1} \nabla f(x_k)$$

dove ora $x_k \in \mathbb{R}^n$ e H_f è la matrice Hessiana (derivate seconde) di f.

direzione di ricerca (Newton): $p_k = -H_f(x_k)^{-1} \nabla f(x_k)$
step length: $\alpha_k = 1$

Osservazione

La direzione di Newton è di discesa $\Leftrightarrow$ H_f è definita positiva.

Newton-Raphson

Il metodo di Newton-Raphson multivariato richiede $f \in C^2$, e conoscenza dell'espressione analitica del gradiente e della matrice Hessiana.

Nota

Se f ha un'espressione semplice, la function `deriv` consente di calcolarne simbolicamente il gradiente e la matrice Hessiana.

Costo computazionale

A ogni iterazione: valutazione di gradiente, Hessiana e risoluzione del sistema $n \times n$

$$H_f(x_k)\, p_k = -\nabla f(x_k)$$

per la determinazione di p_k (costo $\mathcal{O}(n^3)$).