Function minimization

Volker Blobel – University of Hamburg
March 2005

Optimization in Physics data analysis

Objective function $F(\boldsymbol{x})$ is

- (negative log of a) Likelihood function in the Maximum Likelihood method, or
- sum of squares in a (nonlinear) Least Squares problem

Constraints $c_i(\boldsymbol{x})$ are

- equality contraints, expressing relations between parameters, and
- inequality constraints are limits of certain parameters, defining a restricted range of parameter values (e.g. $m_\nu^2 \geq 0$).

In mathematical terms:

$$\begin{array}{rll} \text{minimize} & F(\boldsymbol{x}) & \boldsymbol{x} \in \mathcal{R}^n \\ \text{subject to} & c_i(\boldsymbol{x}) = 0, & i = 1, 2, \ldots m' \\ & c_i(\boldsymbol{x}) \geq 0, & i = m'+1, \ldots m \end{array}$$

Sometimes there are two objective functions: $F(\boldsymbol{x})$, requiring a good data description, and $G(\boldsymbol{x})$, requiring e.g. smoothness:

$$\text{minimize sum} \quad F(\boldsymbol{x}) + \tau \cdot G(\boldsymbol{x})$$

with *weight* parameter τ.

Aim of optimization

- Find the global minimum of the objective function $F(\boldsymbol{x})$,
- within the allowed range of parameter values,
- in a short time even for a large number of parameters and a complicated objective function,
- even is there are local minima.

Most methods will converge to the *next* minimum, which may be the global minimum or a local minimum (corresponding to *rapid* cooling), going immediately downhill as far as possible.

Search for **the** global minimum requires a special effort.

A special case: combinatorical minimization

The travelling salesman problem:

> Find the shortest cyclical itenary for a travelling salesman who must visit each of N cities in turn.

Space over which the function is defined is a discrete, but very large, configuration space (e.g. set of possible orders of cities), with a number of elements in the configuration space which is factorial large (remember: $69! \approx 10^{100}$). The concept of a "direction" may have no meaning.

Objective function is e.g. the total path length.

Desired global extremum may be hidden among many, poorer, local extrema.

> Nature is, for slowly cooled systems, able to find the minimum energy state of a crystal that is completely ordered over a large distance.

The method: Simulated annealing, in analogy with the way metals cool and anneal, with slow cooling, using Boltzmanns probability distribution

$$\text{Prob}(E) \sim \exp\left(-E/k_B T\right)$$

for a systen in equilibrium.

Even at low T there is a chance for a higher energy state, with a corresponding chance to get out of a local minimum.

The Metropolis algorithm: the elements

- A description of possible system configurations.
- A generator of random changes in the configuration (called "option").
- An objective function E (analog of energy) whose minimizatiuon is the goal of the procedure.
- A control parameter T (analog of temperature) and an *annealing* schedule (for cooling down).

A random option will change the configuration from energy E_1 to energy E_2. This option is accepted with probability

$$p = \exp\left[\frac{-(E_2 - E_1)}{k_b T}\right]$$

Note: if $E_2 < E_1$ the option is always accepted.

Example: the "effective" solution of the travelling salesman problem

One-dimensional minimization

Search for minimum of function $f(x)$ of (scalar) argument x.

Important application in multidimensional minimization: robust minimization *along a line*

$$f(z) = F(\boldsymbol{x} + z \cdot \boldsymbol{\Delta x}) \qquad \text{Line search}$$

Aim: robust, efficient and as fast as possible, because each function evaluation my require a large cpu time.

Standard method: iterations, starting from x_0, with expression $\Phi(x)$:

$$x_{k+1} = \Phi(x_k) \quad \text{for} \quad k = 0,\, 1,\, 2, \ldots$$

with convergence to fixed point x^* with $x^* = \Phi(x^*)$

Search methods in n dimensions

Search methods in n dimensions do not require any derivatives, only functions values.

Examples:

Line search in one variable, sequentially in all dimensions
- this is usually rather inefficient

Simplex method by Nelder and Mead: simple, but making use of earlier function evaluations in an efficient way ("learning")

Monte Carlo search: random search in n dimensions, using result as starting values for more efficient methods

– meaningful if several local minima may exist

In general search methods are acceptable initially (far from the optimum), but are inefficient and slow in the "end game".

Simplex method in n dimensions

A simplex is formed by $n+1$ points in n-dimensional space ($n = 2$ triangle)

$$\boldsymbol{x}_1,\ \boldsymbol{x}_2,\ \ldots \boldsymbol{x}_n,\ \boldsymbol{x}_{n+1}$$

sorted such that function values $F_j = F(\boldsymbol{x}_j)$ are in the order

$$F_1 \leq F_2 \leq \ldots F_n \leq F_{n+1}$$

In addition: mean of best n points =

$$\text{center of gravity} \qquad \boldsymbol{c} = \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{x}_j$$

Method: sequence of cycles with new point in each cycle, replacing worst point x_{n+1}, with new (updated) simplex in each cycle.

At the start of each cycle new test point $\boldsymbol{x}_r$, obtained by reflexion of worst point at the center of gravity:

$$\boldsymbol{x}_r = \boldsymbol{c} + \alpha \cdot (\boldsymbol{c} - \boldsymbol{x}_{n+1})$$

A cycle in the Nelder and Mead method

Depending on value $F(x_r)$:

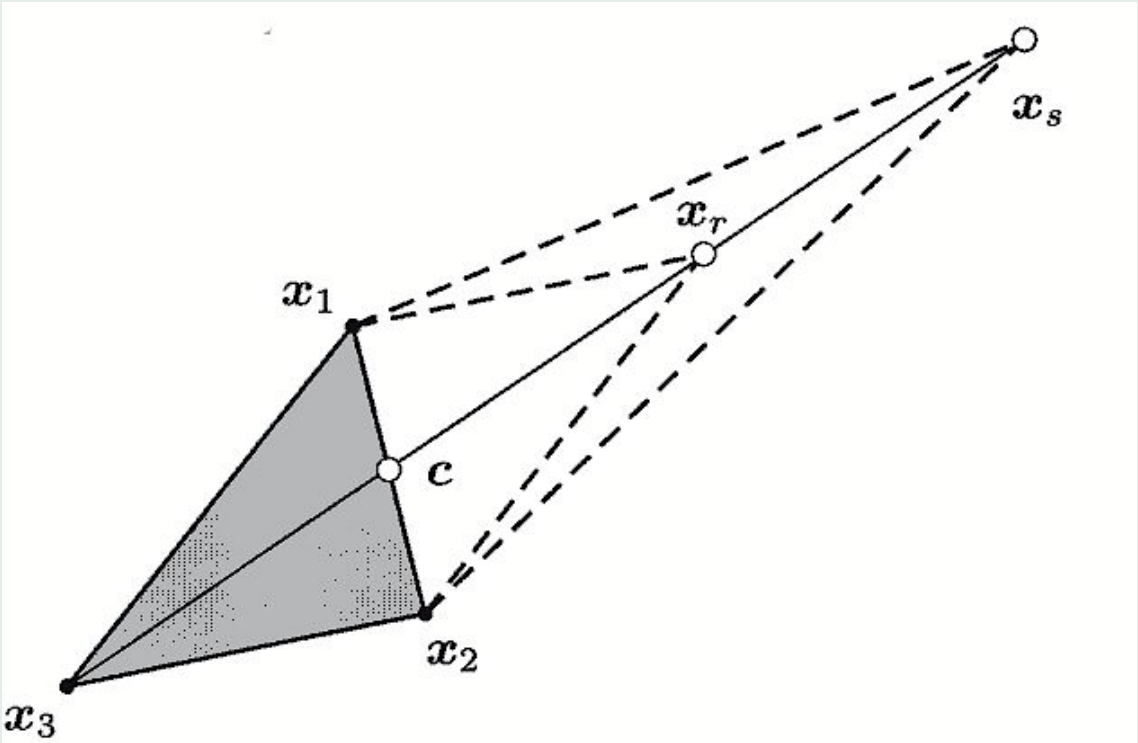
$[F_1 \leq F_r \leq F_n\ ]$: Test point $\boldsymbol{x}_r$ is middle point and is added, the previous worst point is removed.

$[F_r \leq F_1\ ]$: Test point $\boldsymbol{x}_r$ is best point, search direction seems to be effective. A new point $\boldsymbol{x}_s = \boldsymbol{c} + \beta \cdot (\boldsymbol{x}_r - \boldsymbol{c})$ (with $\beta > 1$) is determined and the function value is F_s evaluated. For $F_s < F_r$ the extra step is successful, $\boldsymbol{x}_{n+1}$ is replaced by $\boldsymbol{x}_s$ otherwise by $\boldsymbol{x}_r$.

$[F_r > F_n\ ]$: the simplex is too big and has to be reduced in size. For $F_r < F_{n+1}$ the test point $\boldsymbol{x}_r$ replaces the words point $\boldsymbol{x}_{n+1}$. A new test point $\boldsymbol{x}_s$ is defined by $\boldsymbol{x}_s = \boldsymbol{c} - \gamma(\boldsymbol{c} - \boldsymbol{x}_{n+1})$ (contraction) with $0 < \gamma < 1$. If this point $\boldsymbol{x}_s$ with $F_s = F(\boldsymbol{x}_s) < F_{n+1}$ is an improvement, then $\boldsymbol{x}_{n+1}$ is replaced by this point $\boldsymbol{x}_s$. Otherwise a new simplex in defined by replacing all points except $\boldsymbol{x}_1$ by $\boldsymbol{x}_j = \boldsymbol{x}_1 + \delta(\boldsymbol{x}_j - \boldsymbol{x}_1)$ for $j = 2, \ldots n+1$ with $0 < \delta < 1$, which requires n function evaluations.

Typical values are $\alpha = 1$, $\beta = 2$, $\gamma = 0.5$ und $\delta = 0.5$.

The Simplex method



A few steps of the Simplex method. Starting from the simplex $(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$ with center of gravity $\boldsymbol{c}$. The points $\boldsymbol{x}_r$ and $\boldsymbol{x}_s$ are test points.

Unconstrained minimization

$$\boxed{\text{minimize } F(\boldsymbol{x}) \qquad \boldsymbol{x} \in \mathcal{R}^n}$$

Taylor expansion:

$$\begin{aligned} \text{function} \qquad F(\boldsymbol{x}_k + \boldsymbol{\Delta x}) &\approx F_k + \boldsymbol{g}_k^T \boldsymbol{\Delta x} + \frac{1}{2} \boldsymbol{\Delta x}^T \boldsymbol{H}_k \boldsymbol{\Delta x} \\ \text{derivative} \qquad \boldsymbol{g}(\boldsymbol{x}_k + \Delta \boldsymbol{x}) &\approx \boldsymbol{g}_k + \boldsymbol{H}_k \boldsymbol{\Delta x} \ , \end{aligned}$$

Function value and derivatives are evaluated at $\boldsymbol{x} = \boldsymbol{x}_k$:

$$\begin{aligned} \text{function} \qquad F_k &= F(\boldsymbol{x}_k) \\ \text{gradient} \qquad (g_k)_i &= \left. \frac{\partial F}{\partial x_i} \right|_{\boldsymbol{x}=\boldsymbol{x}_k} \qquad i = 1,\ 2,\ \ldots n \\ \text{Hesse matrix} \qquad (H_k)_{ij} &= \left. \frac{\partial^2 F}{\partial x_i \partial x_j} \right|_{\boldsymbol{x}=\boldsymbol{x}_k} \qquad i, j = 1,\ 2,\ \ldots n \end{aligned}$$

Covariance matrix for statistical objective functions

Note: if the objective function is

- a sum of squares of deviations, defined according to the Method of Least Squares, or
- a negative log. Likelihood function, defined according to the Maximum Likelihood Method,

then the inverse Hessian $\boldsymbol{H}$ at the minimum is a good estimate of the covariance matrix of the parameters $\boldsymbol{x}$:

$$\boldsymbol{V}_{\boldsymbol{x}} \approx \boldsymbol{H}^{-1}$$

The Newton step

Step $\boldsymbol{\Delta x}_N$ determined from $\boldsymbol{g}_k + \boldsymbol{H}_k \boldsymbol{\Delta x} = 0$:

$$\boldsymbol{\Delta x}_N = -\boldsymbol{H}_k^{-1} \boldsymbol{g}_k \ .$$

For a quadratic function $F(\boldsymbol{x})$ the Newton step is, **in length and direction**, a step to the minimum of the function.

Sometimes large angle ($\approx 90°$) between Newton direction $\boldsymbol{\Delta x}_N$ and $\boldsymbol{g}_k$ (the direction of steepest descent).

Calculation of the distance ΔF to the minimum (called EDM in MINUIT):

$$d = -\boldsymbol{g}_k^T \cdot \Delta \boldsymbol{x}_N = \boldsymbol{g}_k^T \boldsymbol{H}_k^{-1} \boldsymbol{g}_k = \boldsymbol{\Delta x}_N^T \boldsymbol{H}_k \boldsymbol{\Delta x}_N > 0$$

if Hessian positive-definite. For a quadratic function the distance to the minimum is $d/2$.

General iteration scheme

Define initial value $\boldsymbol{x}_0$, compute $F(\boldsymbol{x}_0)$ and set $k = 0$

1. **Test for convergence:** If the conditions for convergence are satisfied, the algorithm terminates with $\boldsymbol{x}_k$ as the solution.

2. **Compute a search vector:** A vector $\boldsymbol{\Delta x}$ is computed as the new search vector. The Newton search vector is determined from $\boldsymbol{H}_k \boldsymbol{\Delta x}_N = -\boldsymbol{g}_k$.

3. **Line Search:** A one-dimensional minimization is done for the function $f(z) = F(\boldsymbol{x}_k + z \cdot \boldsymbol{\Delta x})$. and $z_{\min}$ is determined.

4. **Update:** The point $\boldsymbol{x}_{k+1}$ is defined by $\boldsymbol{x}_k + z_{\min} \cdot \boldsymbol{\Delta x}$, and k is increased by 1. Repeat steps.

Descent direction $\quad \boldsymbol{g}_k^T \cdot \Delta \boldsymbol{x} < 0 \quad$ if Hessian positive-definite

Method of steepest descent

The search vector $\boldsymbol{\Delta x}$ is equal to the negative gradienten

$$\boldsymbol{\Delta x}_{\mathrm{SD}} = -\boldsymbol{g}_k$$

- Step seems to be natural choice;
- only gradient required (no Hesse matrix) – good;
- no step size defined (in contrast to the Newton step) – bad;
- rate of convergence only linear:

$$\frac{F(\boldsymbol{x}_{k+1}) - F(\boldsymbol{x}^*)}{F(\boldsymbol{x}_k) - F(\boldsymbol{x}^*)} = c \approx \left(\frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}}\right)^2 = \left(\frac{\kappa - 1}{\kappa + 1}\right)^2 \quad ,$$

  ($\lambda_{\max}$ and $\lambda_{\min}$ are largest and smallest eigenvalue and κ condition number of Hesse matrix $\boldsymbol{H}$. For a large value of κ c close to one and slow convergence – very bad.

Optimal step including size, if Hesse matrix known:

$$\boldsymbol{\Delta x}_{\mathrm{SD}} = -\left(\frac{\boldsymbol{g}_k^T \boldsymbol{g}_k}{\boldsymbol{g}_k^T \boldsymbol{H}_k \boldsymbol{g}_k}\right) \boldsymbol{g}_k \quad .$$

Problem: Hessian not positive definite

Diagonalization:

$$\boldsymbol{H} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T = \sum_i \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^T \ ,$$

vectors $\boldsymbol{u}_i$ are the eigenvectors of $\boldsymbol{H}$, and $\boldsymbol{D}$ is diagonal, with the eigenvalues λ_i of $\boldsymbol{H}$ in diagonal. At least one of the eigenvalues λ_i is zero or less than zero.

1. Strategy: define modified diagonal matrix $\overline{\boldsymbol{D}}$ with elements

$$\overline{\lambda}_i = \max\left(|\lambda_i|, \delta\right) \qquad \delta > 0$$

(which requires diagonalization) and use modified positive-definite Hessian

$$\overline{\boldsymbol{H}} = \boldsymbol{U}\overline{\boldsymbol{D}}\boldsymbol{U}^T \qquad \overline{\boldsymbol{H}}^{-1} = \boldsymbol{U}\overline{\boldsymbol{D}}^{-1}\boldsymbol{U}^T$$

2. Strategy: define modified Hessian by

$$\overline{\boldsymbol{H}} = \boldsymbol{H} + \alpha \boldsymbol{I}_n \qquad \text{or} \quad \overline{\boldsymbol{H}_{jj}} = (1+\lambda) \cdot \boldsymbol{H}_{jj} \quad \text{(Levenberg-Marquardt)}$$

(simpler and faster compared to diagonalization).

- Large value of α (or λ) means search direction close to (negative) gradient (safe),
- small value of α (or λ) means search direction close to the Newton direction.

The factor α allows to interpolate between the methods of steepest descent and Newton method.

But what is large and what is small α: scale defined by

$$\alpha \approx \frac{|\boldsymbol{g}^T \boldsymbol{H} \boldsymbol{g}|}{\boldsymbol{g}^T \boldsymbol{g}}$$

suggested by step of steepest descent method.

Derivative calculation

The (optimal) Newton method requires

- first derivatives of $F(\boldsymbol{x})$: computation $\propto n$;
- second derivatives of $F(\boldsymbol{x})$: computation $\propto n^2$.

- Analytical derivatives may be impossible or difficult to obtain.
- Numerical derivatives require good step size δ for differential quotient

E.g. numerical derivative of $f(x)$ in one dimension:

$$\left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x=x_0} \approx \frac{f(x_0+\delta) - f(x_0-\delta)}{2\delta}$$

Can the Newton (or quasi Newton) method be used without explicit calculation of the complete Hessian?

Variable metrik method

Calculation of Hessian (with $n(n+1)/2$ different elements) from sequence of first derivatives (gradients): by update of estimate $\boldsymbol{H}_k$ from change of gradient.

- Step $\boldsymbol{\Delta x}$ is calculated from $\boldsymbol{H}_k \, \boldsymbol{\Delta x} = -\boldsymbol{g}_k$
- After a line search with minimum at $z_{\min}$ new value is $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + z_{\min} \, \boldsymbol{\Delta x}$, with gradient $\boldsymbol{g}_{k+1}$.
- Update matrix: $\boldsymbol{U}_k$ is added to $\boldsymbol{H}_k$ to get new estimate of Hessian:

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \boldsymbol{U}_k$$

$$\boldsymbol{H}_{k+1}\boldsymbol{\delta} = \boldsymbol{\gamma} \qquad \boldsymbol{\delta} = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \qquad \boldsymbol{\gamma} = \boldsymbol{g}_{k+1} - \boldsymbol{g}_k$$

(with $\boldsymbol{\gamma}^T \boldsymbol{\delta} > 0$)

Note: an accurate line search is essential for the success.

Most effective update formula (Broydon/Fletcher/Goldfarb/Shanno (BFGS))

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k - \frac{(\boldsymbol{H}_k\boldsymbol{\delta})\,(\boldsymbol{H}_k\boldsymbol{\delta})^T}{\boldsymbol{\delta}^T\boldsymbol{H}_k\boldsymbol{\delta}} + \frac{\boldsymbol{\gamma}\boldsymbol{\gamma}^T}{\boldsymbol{\gamma}^T\boldsymbol{\delta}} \quad .$$

Initial matrix may be the unit matrix.

Also formulas for the update of the inverse $\boldsymbol{H}_k^{-1}$ exist.

Properties: the method generates n linear independent search directions for a quadratic function and the estimated Hessian $\boldsymbol{H}_{k+1}$ converges to the true Hessian $\boldsymbol{H}$.

Potential problems:

- no real convergence for "good" starting point;
- estimate "destroyed" for small, inaccurate steps (round-off errors).

Example: MINUIT from CERN

Several options can be selected:

Option MIGRAD: minimizes the objective function, calculates first derivatives numerically and uses the BFGS update formula for the Hessian – fast (if it works)

Option HESSE: calculates the Hesse matrix numerically – recommended after minimization

Option MINIMIZE: minimization by MIGRAD and HESSE calculation with checks.

Restricted range of parameters possible: by transformation from a finite range (of external variables) to $-\infty \ldots + \infty$ (of internal variables) using trigonometrical functions.

- allows to use minimization algorithm without range restriction;
- good (quadratic) function is transformed perhaps to a difficult function;
- range specification may be bad, e.g $[0, 10^8]$ for optimum at 1.

Derivative calculation

Efficient minimization *must* take first and second order derivatives into account, which allows to apply Newton's principle.

Numerical methods for derivative calculation of $F(\boldsymbol{x})$ by finite differences

- require a good step length – not to small (roundoff errors) and not too large (influence of higher derivatives);
- require very many (time consuming) function evaluations.

Variable metrik methods allow to avoid the calculation of second derivatives of $F(\boldsymbol{x})$, but still need the first derivative.
Are there other possibilities?

- the Gauss-Newton method, which reduces the derivative calculation; the Hessian can be calculated (estimated) without having to compute second derivatives;
- automatic derivative calculation.

Least squares curve fit

Assume a least squares fit of a curve $f(x_i, \boldsymbol{a})$ with parameters $\boldsymbol{a}$ to Gaussian distributed data y_i with weight $w_i = 1/\sigma_i^2$:

$$y_i \cong f(x_i, \boldsymbol{a})$$

Negative Log of the likelihood function is equal to the sum of squares:

$$F(\boldsymbol{a}) = \sum_i w_i \left(f(x_i, \boldsymbol{a}) - y_i\right)^2$$

Derivatives of $F(\boldsymbol{a})$ are determined by derivatives of $f(x_i, \boldsymbol{a})$:

$$\frac{\partial F}{\partial a_j} = \sum_i w_i \frac{\partial f}{\partial a_j} \left(f(x_i, \boldsymbol{a}) - y_i\right)$$

$$\frac{\partial^2 F}{\partial a_j \partial a_k} = \sum_i w_i \frac{\partial f}{\partial a_j} \frac{\partial f}{\partial a_k} + \sum_i w_i \frac{\partial^2 f}{\partial a_j \partial a_k} \left(f(x_i, \boldsymbol{a}) - y_i\right)$$

Usually the contributions from second derivatives of $f(x_i, \boldsymbol{a})$ are small and can be neglected; the approximate Hessian from first derivatives of $f(x_i, \boldsymbol{a})$ has positive diagonal elements and is expected to be positive-definite. The use of the approximate Hessian is called Gauss-Newton method.

Poisson contribution to objective function

Assume a maximum-likelihood fit of a function $f(x_i, \boldsymbol{a})$ with parameters $\boldsymbol{a}$ to Poisson-distributed data y_i:

$$y_i \cong f(x_i, \boldsymbol{a})$$

Negative Log of the likelihood function:

$$F(\boldsymbol{a}) = \sum_i f(x_i, \boldsymbol{a}) - y_i \ln f(x_i, \boldsymbol{a})$$

$$\text{or better} \quad F(\boldsymbol{a}) = \sum_i \left(f(x_i, \boldsymbol{a}) - y_i\right) + y_i \ln \frac{y_i}{f(x_i, \boldsymbol{a})}$$

Derivatives of $F(\boldsymbol{a})$ are determined by derivatives of $f(x_i, \boldsymbol{a})$:

$$\frac{\partial F}{\partial a_j} = \sum_i y_i \frac{\frac{\partial f}{\partial a_j}}{f(x_i, \boldsymbol{a})} - \frac{\partial f}{\partial a_j}$$

$$\frac{\partial^2 F}{\partial a_j \partial a_k} = \sum_i y_i \frac{\frac{\partial f}{\partial a_j}\frac{\partial f}{\partial a_k} - \frac{\partial^2 f}{\partial a_j \partial a_k} f(x_i, \boldsymbol{a})}{f^2(x_i, \boldsymbol{a})} - \sum_i \frac{\partial^2 f}{\partial a_j \partial a_k}$$

Usually the contributions from second derivatives of $f(x_i, \boldsymbol{a})$ are small and and can be neglected; the approximate Hessian from first derivatives of $f(x_i, \boldsymbol{a})$ has positive diagonal elements and is expected to be positive-definite. The use of the approximate Hessian is called Gauss-Newton method.

Automatic differentiation

Can automatic differentiation be used?

GNU libmatheval is a library comprising several procedures that makes it possible to create in-memory tree representation of *mathematical functions* over single or multiple variables and later use this representation to *evaluate function for specified variable values*, to create corresponding tree for *function derivative* over specified variable or to get back textual representation of in-memory tree.

Calculated derivative is in mathematical sense correct no matters of fact that derivation variable appears or not in function represented by evaluator.

Supported elementary functions are: exponential (exp), logarithmic (log), square root (sqrt), sine (sin), cosine (cos), tangent (tan), cotangent (cot), secant (sec), cosecant (csc), inverse sine (asin), inverse cosine (acos), inverse tangent (atan), inverse cotangent (acot), inverse secant (asec), inverse cosecant (acsc), hyperbolic sine (sinh), cosine (cosh), hyperbolic tangent (tanh), hyperbolic cotangent (coth), hyperbolic secant (sech), hyperbolic cosecant (csch), hyperbolic inverse sine (asinh), hyperbolic inverse cosine (acosh), hyperbolic inverse tangent (atanh), hyperbolic inverse cotangent (acoth), hyperbolic inverse secant (asech), hyperbolic inverse cosecant (acsch), absolute value (abs), Heaviside step function (step) with value 1 defined for $x = 0$ and Dirac delta function with infinity (delta) and not-a-number (nandelta) values defined for $x = 0$.

Trust-region methods

If the derivatives of $F(\boldsymbol{x})$ are available the quadratic model

$$\widetilde{F}(\boldsymbol{x}_k + \boldsymbol{d}) = F_k + \boldsymbol{g}_k^T \boldsymbol{d} + \frac{1}{2} \boldsymbol{d}^T \boldsymbol{H}_k \boldsymbol{d}$$

can be written at the point $\boldsymbol{x}_k$ with the gradient $\boldsymbol{g}_k \equiv \nabla F$ and the Hessian $\boldsymbol{H}_k \equiv \nabla^2 F$ (matrix of second derivatives). From the approximation the step $\boldsymbol{d}$ is calculated from

$$\boldsymbol{g}_k + \boldsymbol{H}_k \boldsymbol{d} = 0$$

which is

$$\boldsymbol{d} = -\boldsymbol{H}_k^{-1} \boldsymbol{g}_k \ .$$

In the **trust-region method** the model is **restricted** depending on a parameter Δ, which forces the next iterate into a sphere centered at $\boldsymbol{x}_k$ and of radius Δ, which defines the *trust region*, where the model is assumed to be *acceptable*. The vector $\boldsymbol{d}$ is determined by solving

$$\text{minimize} \quad \widetilde{F}(\boldsymbol{x}_k + \boldsymbol{d}), \quad |\boldsymbol{d}| \leq \Delta \ ,$$

a constrained quadratic minimization problem. The equation can be solved efficiently even if the Hessian is *not positive-definite*.

Trust region models

If λ_1 is the first eigenvalue of a not-positive definite Hessian a parameter λ with $\lambda \geq \max(0, -\lambda_1)$ is selected such that $(\boldsymbol{H}_k + \lambda \boldsymbol{I})$ is positive-definite. The minimum point of a Lagrangian

$$L(\boldsymbol{d}, \lambda) = F(\boldsymbol{x}_k + \boldsymbol{d}) + \frac{\lambda}{2}\left(|\boldsymbol{d}|^2 - \Delta^2\right)$$

for fixed λ is given by

$$\boldsymbol{d}_\Delta = -(\boldsymbol{H}_k + \lambda \boldsymbol{I})^{-1} \boldsymbol{g}_k$$

Now the merit function is $F(\boldsymbol{x}_k + \boldsymbol{d}_\Delta)$.
The trajectory $\{\boldsymbol{x}_k + \boldsymbol{d}_\Delta\}$ becomes a curve in the space R^n, which has a finite length if $\widetilde{F}$ has a minimum point at finite distance.

The actual method to determine $\boldsymbol{d}$ is discussed later.

The success of a step $\boldsymbol{d}_k$ is measured by the ratio

$$\rho_k = \frac{F_k - F\left(\boldsymbol{x}_k + \boldsymbol{d}_k\right)}{F_k - \widetilde{F}\left(\boldsymbol{x}_k + \boldsymbol{d}_k\right)}$$

of *actual to predicted* decrease of the objective function. Ideally ρ_k is close to 1.

- The trust-region step $\boldsymbol{d}_k$ is accepted when ρ_k is not much smaller than 1.
- If ρ_k is however much smaller than 1, then the trust-region radius Δ is too large.
- In addition a prescription to increase Δ in case of successful steps is needed for future steps.

A proposed strategy:

if $\rho_k > 0.9$	[very successful]	$x_{k+1} := x_k + d_k$	$\Delta_{k+1} := 2 \times \Delta_k$
otherwise if $\rho_k > 0.1$	[successful]	$x_{k+1} := x_k + d_k$	$\Delta_{k+1} := \Delta_k$
otherwise if $\rho_k \leq 0.1$	[unsuccessful]	$x_{k+1} := x_k$	$\Delta_{k+1} := 1/2 \times \Delta_k$

First case: if $\boldsymbol{H}$ is positive definite and the solution $\boldsymbol{d}$ of

$$\boldsymbol{H}\boldsymbol{d} = -\boldsymbol{g}$$

satifies $||d_\Delta(\lambda)|| = \Delta$, then the problem is solved.

Second case (otherwise): make the spectral decomposition (diagonalization)

$$\boldsymbol{H} = \boldsymbol{U}^{\mathrm{T}}\boldsymbol{D}\,\boldsymbol{U}$$

with $\boldsymbol{U}$ = matrix of orthogonal eigenvectors, $\boldsymbol{D}$ = diagonal matrix made up of eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. One has to calculate a value of λ with $\lambda \geq -\lambda_1$.

$$s(\lambda) = -\left(\boldsymbol{H} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{g}$$

$$||s(\lambda)||_2^2 = ||\boldsymbol{U}^{\mathrm{T}}\left(\boldsymbol{D} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{U}\boldsymbol{g}||_2^2 = \sum_{i=1}^{n}\left(\frac{\gamma_i}{\lambda_i + \lambda}\right)^2 = \Delta^2$$

where $\gamma_i = \boldsymbol{e}_i^{\mathrm{T}}\boldsymbol{U}\boldsymbol{g}$. This equation has to be solved for λ, which is extremely difficult.

It is far better to solve the equivalent equation

$$\frac{1}{||s(\lambda)||} - \frac{1}{\Delta} = 0$$

which is an analytical equation, which has no poles and can be solved by Newton's method.

Properties of trust-region methods

Opinions about trust-region methods:

"The trust-region method is actually extremely important and might supersede line-searches, sooner or later."

"Trust-region methods are not superior to methods with step-length strategy; at least the number of iterations is higher. ... they are not more robust."

Parameters can be of very different order of magnitude and precision. Preconditioning with appropriate *scaling* of the parameters may be essential for the trust-region method.

Linesearch-methods are naturally *optimistic*,
while trust-region methods appear to be naturally *pessimistic*. Trust-region methods are expected to be more robust, with convergence (perhaps slower) even in difficult cases.

Function minimization