




WIKIPEDIA  
The Free Encyclopedia

[Main page](#)  
[Contents](#)  
[Featured content](#)  
[Current events](#)  
[Random article](#)  
[Donate to Wikipedia](#)  
[Wikipedia store](#)

Interaction  
[Help](#)  
[About Wikipedia](#)  
[Community portal](#)  
[Recent changes](#)  
[Contact page](#)

Tools  
[What links here](#)  
[Related changes](#)  
[Upload file](#)  
[Special pages](#)  
[Permanent link](#)  
[Page information](#)  
[Wikidata item](#)  
[Cite this page](#)

Print/export  
[Create a book](#)  
[Download as PDF](#)  
[Printable version](#)

Languages   
[Deutsch](#)  
[فارسی](#)  
[Français](#)  
[Lietuvių](#)  
[Polski](#)  
[Português](#)  
[Русский](#)  
[中文](#)


 [Edit links](#)

[Create account](#) [Log in](#)

Article [Talk](#)

[Read](#) [Edit](#)

[More](#) ▾



# Levenberg–Marquardt algorithm

From Wikipedia, the free encyclopedia

In [mathematics](#) and computing, the **Levenberg–Marquardt algorithm (LMA)**, also known as the **damped least-squares (DLS)** method, is used to solve [non-linear least squares](#) problems. These minimization problems arise especially in [least squares curve fitting](#).

The LMA is used in many software applications for solving generic curve-fitting problems. However, as for many fitting algorithms, the LMA finds only a [local minimum](#), which is not necessarily the [global minimum](#). The LMA interpolates between the [Gauss–Newton algorithm](#) (GNA) and the method of [gradient descent](#). The LMA is more [robust](#) than the GNA, which means that in many cases it finds a solution even if it starts very far off the final minimum. For well-behaved functions and reasonable starting parameters, the LMA tends to be a bit slower than the GNA. LMA can also be viewed as Gauss–Newton using a [trust region](#) approach.

The algorithm was first published in 1944 by [Kenneth Levenberg](#), while working at the [Frankford Army Arsenal](#). It was rediscovered in 1963 by [Donald Marquardt](#) who worked as a [statistician](#) at [DuPont](#) and independently by Girard, Wynn and Morrison.<sup>[*who?*]</sup>

## Contents [\[hide\]](#)

- The problem
- The solution
  - Choice of damping parameter
- Example
- See also
- Notes
- References
- External links
  - Descriptions
  - Implementations

## The problem [\[edit\]](#)

The primary application of the Levenberg–Marquardt algorithm is in the least squares curve fitting problem: given a set of *m* empirical datum pairs of independent and dependent variables, (*x*<sub>*i*</sub>, *y*<sub>*i*</sub>), optimize the parameters *β* of the model curve *f*(*x*,*β*) so that the sum of the squares of the deviations

$$S(\boldsymbol{\beta}) = \sum_{i=1}^m [y_i - f(x_i, \boldsymbol{\beta})]^2$$

becomes minimal.

## The solution [\[edit\]](#)

Like other numeric minimization algorithms, the Levenberg–Marquardt algorithm is an [iterative](#) procedure. To start a minimization, the user has to provide an initial guess for the parameter vector, *β*. In cases with only one minimum, an uninformed standard guess like *β*<sup>T</sup>=(1,1,...,1) will work fine; in cases with [multiple minima](#), the algorithm converges to the global minimum only if the initial guess is already somewhat close to the final solution.

In each iteration step, the parameter vector, *β*, is replaced by a new estimate, *β* + *δ*. To determine *δ*, the functions *f*(*x*<sub>*i*</sub>,*β* + *δ*) are approximated by their linearizations

$$f(x_i, \boldsymbol{\beta} + \boldsymbol{\delta}) \approx f(x_i, \boldsymbol{\beta}) + J_i \boldsymbol{\delta}$$

where

$$J_i = \frac{\partial f(x_i, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$$

is the [gradient](#) (row-vector in this case) of *f* with respect to *β*.

At the minimum of the sum of squares,  $S(\beta)$ , the [gradient](#) of  $S$  with respect to  $\delta$  will be zero. The above first-order approximation of  $f(x_i, \beta + \delta)$  gives

$$S(\beta + \delta) \approx \sum_{i=1}^m (y_i - f(x_i, \beta) - J_i \delta)^2.$$

Or in vector notation,

$$S(\beta + \delta) \approx \|\mathbf{y} - \mathbf{f}(\beta) - \mathbf{J}\delta\|^2.$$

Taking the derivative with respect to  $\delta$  and setting the result to zero gives:

$$(\mathbf{J}^T \mathbf{J})\delta = \mathbf{J}^T[\mathbf{y} - \mathbf{f}(\beta)]$$

where  $\mathbf{J}$  is the [Jacobian matrix](#) whose  $i^{\text{th}}$  row equals  $J_i$ , and where  $\mathbf{f}$  and  $\mathbf{y}$  are vectors with  $i^{\text{th}}$  component  $f(x_i, \beta)$  and  $y_i$ , respectively. This is a set of linear equations which can be solved for  $\delta$ .

Levenberg's contribution is to replace this equation by a "damped version",

$$(\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I})\delta = \mathbf{J}^T[\mathbf{y} - \mathbf{f}(\beta)]$$

where  $\mathbf{I}$  is the identity matrix, giving as the increment,  $\delta$ , to the estimated parameter vector,  $\beta$ .

The (non-negative) damping factor,  $\lambda$ , is adjusted at each iteration. If reduction of  $S$  is rapid, a smaller value can be used, bringing the algorithm closer to the [Gauss–Newton algorithm](#), whereas if an iteration gives insufficient reduction in the residual,  $\lambda$  can be increased, giving a step closer to the gradient descent direction. Note that the [gradient](#) of  $S$  with respect to  $\delta$  equals  $-2(\mathbf{J}^T[\mathbf{y} - \mathbf{f}(\beta)])^T$ . Therefore, for large values of  $\lambda$ , the step will be taken approximately in the direction of the gradient. If either the length of the calculated step,  $\delta$ , or the reduction of sum of squares from the latest parameter vector,  $\beta + \delta$ , fall below predefined limits, iteration stops and the last parameter vector,  $\beta$ , is considered to be the solution.

Levenberg's algorithm has the disadvantage that if the value of damping factor,  $\lambda$ , is large, inverting  $\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I}$  is not used at all. Marquardt provided the insight that we can scale each component of the gradient according to the curvature so that there is larger movement along the directions where the gradient is smaller. This avoids slow convergence in the direction of small gradient. Therefore, Marquardt replaced the identity matrix,  $\mathbf{I}$ , with the diagonal matrix consisting of the diagonal elements of  $\mathbf{J}^T \mathbf{J}$ , resulting in the Levenberg–Marquardt algorithm:

$$(\mathbf{J}^T \mathbf{J} + \lambda \text{diag}(\mathbf{J}^T \mathbf{J}))\delta = \mathbf{J}^T[\mathbf{y} - \mathbf{f}(\beta)].$$

A similar damping factor appears in [Tikhonov regularization](#), which is used to solve linear [ill-posed problems](#), as well as in [ridge regression](#), an [estimation](#) technique in [statistics](#).

### Choice of damping parameter [\[edit\]](#)

Various more-or-less heuristic arguments have been put forward for the best choice for the damping parameter  $\lambda$ . Theoretical arguments exist showing why some of these choices guaranteed local convergence of the algorithm; however these choices can make the global convergence of the algorithm suffer from the undesirable properties of [steepest-descent](#), in particular very slow convergence close to the optimum.

The absolute values of any choice depends on how well-scaled the initial problem is. Marquardt recommended starting with a value  $\lambda_0$  and a factor  $v > 1$ . Initially setting  $\lambda = \lambda_0$  and computing the residual sum of squares  $S(\beta)$  after one step from the starting point with the damping factor of  $\lambda = \lambda_0$  and secondly with  $\lambda_0/v$ . If both of these are worse than the initial point then the damping is increased by successive multiplication by  $v$  until a better point is found with a new damping factor of  $\lambda_0 v^k$  for some  $k$ .

If use of the damping factor  $\lambda/v$  results in a reduction in squared residual then this is taken as the new value of  $\lambda$  (and the new optimum location is taken as that obtained with this damping factor) and the process continues; if using  $\lambda/v$  resulted in a worse residual, but using  $\lambda$  resulted in a better residual, then  $\lambda$  is left unchanged and the new optimum is taken as the value obtained with  $\lambda$  as damping factor.

### Example [\[edit\]](#)

In this example we try to fit the function  $y = a \cos(bX) + b \sin(aX)$  using the Levenberg–Marquardt algorithm implemented in [GNU Octave](#) as the *leasqr* function. The 3 graphs Fig 1,2,3 show progressively better fitting for the parameters  $a=100$ ,  $b=102$  used in the initial curve. Only when the parameters in Fig 3 are chosen closest to the original, are the curves fitting exactly. This equation is an example of very sensitive initial conditions for the Levenberg–Marquardt algorithm. One reason for this sensitivity is the existence of multiple minima — the function  $\cos(\beta x)$  has minima at parameter value  $\hat{\beta}$  and  $\hat{\beta} + 2n\pi$ .

## See also [edit]

- Trust region
- Nelder–Mead method (aka simplex)

## Notes [edit]

## References [edit]

- Kenneth Levenberg (1944). "A Method for the Solution of Certain Non-Linear Problems in Least Squares". *Quarterly of Applied Mathematics* **2**: 164–168.
- André Girard (1958). "Excerpt from *Revue d'optique théorique et instrumentale*". *Rev. Opt* **37**: 225–241, 397–424.
- C.G. Wynne (1959). "Lens Designing by Electronic Digital Computer: I". *Proc. Phys. Soc. London* **73** (5): 777. doi:10.1088/0370-1328/73/5/310 .
- Jorge J. Moré and Daniel C. Sorensen (1983). "Computing a Trust-Region Step". *SIAM J. Sci. Stat. Comput.* (4): 553–572.
- D.D. Morrison (1960). *Jet Propulsion Laboratory Seminar proceedings*. Missing or empty |title= (help)
- Donald Marquardt (1963). "An Algorithm for Least-Squares Estimation of Nonlinear Parameters". *SIAM Journal on Applied Mathematics* **11** (2): 431–441. doi:10.1137/0111030 .
- Philip E. Gill and Walter Murray (1978). "Algorithms for the solution of the nonlinear least-squares problem". *SIAM Journal on Numerical Analysis* **15** (5): 977–992. doi:10.1137/0715063 .
- Jose Pujol (2007). "The solution of nonlinear inverse problems and the Levenberg-Marquardt method" . *Geophysics* (SEG) **72** (4): W1–W16. doi:10.1190/1.2732552 .
- Nocedal, Jorge; Wright, Stephen J. (2006). *Numerical Optimization, 2nd Edition*. Springer. ISBN 0-387-30303-0.

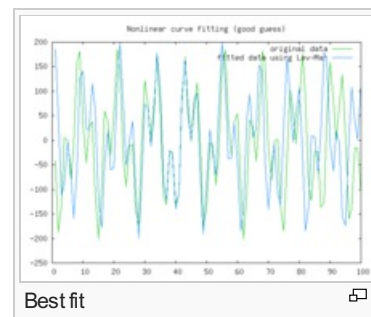
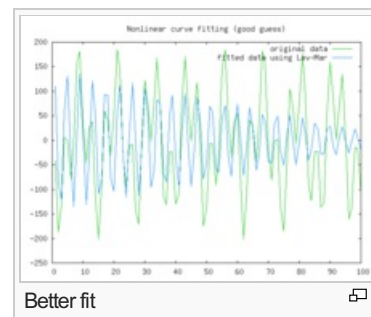
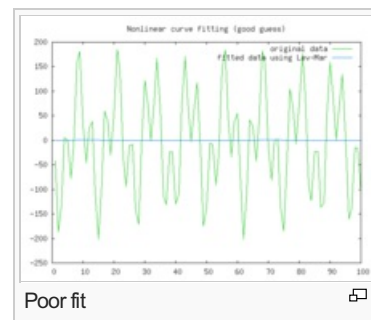
## External links [edit]

### Descriptions [edit]

- Detailed description of the algorithm can be found in [Numerical Recipes in C, Chapter 15.5: Nonlinear models](#)
- C. T. Kelley, *Iterative Methods for Optimization*, SIAM Frontiers in Applied Mathematics, no 18, 1999, ISBN 0-89871-433-8. [Online copy](#)
- History of the algorithm in SIAM news
- A tutorial by Ananth Ranganathan
- [Methods for Non-Linear Least Squares Problems](#)  by K. Madsen, H.B. Nielsen, O. Tingleff is a tutorial discussing non-linear least-squares in general and the Levenberg-Marquardt method in particular
- T. Strutz: *Data Fitting and Uncertainty (A practical introduction to weighted least squares and beyond)*. Vieweg+Teubner, ISBN 978-3-8348-1022-9.
- H. P. Gavin, *The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems*  (MATLAB implementation included)

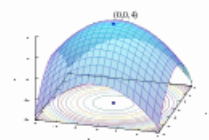
### Implementations [edit]

- Levenberg-Marquardt is a built-in algorithm in [Mathematica](#) , [Matlab](#), [NeuroSolutions](#), [GNU Octave](#), [Origin](#), [SciPy](#), [Fityk](#), [IGOR Pro](#) and [LabVIEW](#).
- The oldest implementation still in use is [lmdif](#) , from [MINPACK](#), in [Fortran](#), in the [public domain](#). See also:
  - [lmfit](#) , a self-contained C implementation of the MINPACK algorithm, with an easy-to-use wrapper for curve fitting, liberal licence (freeBSD).
  - [eigen](#) , a C++ linear algebra library, includes an adaptation of the minpack algorithm in the "NonLinearOptimization" module.
  - The [GNU Scientific Library](#) has a C interface to MINPACK.
  - [C/C++ Minpack](#)  includes the Levenberg–Marquardt algorithm.



- Several high-level languages and mathematical packages have wrappers for the [MINPACK](#) routines, among them:
  - Python library [scipy](#), module `scipy.optimize.leastsq`,
  - IDL, add-on [MPFIT](#).
  - R (programming language) has the [minpack.lm](#) package.
- [levmar](#) is an implementation in C/C++ with support for constraints, distributed under the [GNU General Public License](#).
  - levmar includes a [MEX file](#) interface for [MATLAB](#)
  - Perl (PDL), python, Haskell and .NET interfaces to levmar are available: see [PDL::Fit::Levmar](#) or [PDL::Fit::LM](#), [PyLevmar](#), [HackageDB levmar](#) and [LevmarSharp](#).
- [sparseLM](#) is a C implementation aimed at minimizing functions with large, arbitrarily [sparse](#) Jacobians. Includes a MATLAB MEX interface.
- [SMarquardt.m](#) is a stand-alone routine for Matlab or Octave.
- [InMin](#) library contains a C++ implementation of the algorithm based on the [eigen](#) C++ linear algebra library. It has a pure C-language API as well as a Python binding
- [ceres](#) is a non-linear minimisation library with an implementation of the Levenberg–Marquardt algorithm. It is written in C++ and uses [eigen](#)
- [ALGLIB](#) has implementations of improved LMA in C# / C++ / Delphi / Visual Basic. Improved algorithm takes less time to converge and can use either Jacobian or exact Hessian.
- [NMath](#) has an implementation for the .NET Framework.
- [gnuplot](#) uses its own implementation [gnuplot.info](#).
- Java programming language implementations: 1) [Javanumerics](#), 2) [LMA-package](#) (a small, user friendly and well documented implementation with examples and support), 3) [Apache Commons Math](#)
- [OOoConv](#) implements the L-M algorithm as an OpenOffice.org Calc spreadsheet.
- [SAS](#), there are multiple ways to access SAS's implementation of the Levenberg–Marquardt algorithm: it can be accessed via [NLPLM Call](#) in [PROC IML](#) and it can also be accessed through the [LSQ](#) statement in [PROC NLP](#), and the [METHOD=MARQUARDT](#) option in [PROC NLIN](#).
- [LMAM/OLMAM Matlab toolbox](#) implements Levenberg-Marquardt with adaptive momentum for training feedforward neural networks.
- [GADfit](#) is a Fortran implementation of global fitting based on a modified Levenberg-Marquardt. Uses automatic differentiation. Allows fitting functions of arbitrary complexity, including integrals.

v · t · e	<b>Optimization: Algorithms, methods, and heuristics</b>	<span>[hide]</span>
	<b>Unconstrained nonlinear: Methods calling ...</b>	<span>[show]</span>
	<b>Constrained nonlinear</b>	<span>[show]</span>
	<b>Convex optimization</b>	<span>[show]</span>
	<b>Combinatorial</b>	<span>[show]</span>
	<b>Metaheuristics</b>	<span>[show]</span>
	<b>Categories</b> ( <span>Algorithms and methods</span> · <span>Heuristics</span> ) · <b>Software</b>	



Categories: Statistical algorithms | Optimization algorithms and methods | Least squares

This page was last modified on 25 August 2015, at 02:38.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

