

# The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems

© Henri P. Gavin

Department of Civil and Environmental Engineering  
Duke University

March 25, 2026

## Abstract

The Levenberg-Marquardt algorithm was developed in the early 1960's to solve nonlinear least squares problems. Least squares problems arise in the context of fitting a parameterized mathematical model to a set of data points by minimizing an objective expressed as the sum of the squares of the errors between the model function and a set of data points. If a model is linear in its coefficients, the least squares objective is quadratic in the coefficients. This objective may be minimized with respect to the coefficients in one step via the solution to a linear matrix equation. If the fit function is not linear in its coefficients, the least squares problem requires an iterative solution algorithm. Such algorithms reduce the sum of the squares of the errors between the model function and the data points through a sequence of well-chosen updates to values of the model coefficients. The Levenberg-Marquardt algorithm combines two numerical minimization algorithms: the gradient descent method and the Gauss-Newton method. In the gradient descent method, the sum of the squared errors is reduced by updating the coefficients in the steepest-descent direction. In the Gauss-Newton method, the sum of the squared errors is reduced by assuming the least squares function is locally quadratic in the coefficients, and finding the minimum of this quadratic. The Levenberg-Marquardt method acts more like a gradient-descent method when the coefficients are far from their optimal value, and acts more like the Gauss-Newton method when the coefficients are close to their optimal value. This document describes these methods and illustrates the use of software to solve nonlinear least squares curve-fitting problems.

## 1 Introduction

In fitting a model function  $\hat{y}(t; \mathbf{c})$  of an independent variable  $t$  and a vector of  $n$  coefficients  $\mathbf{c}$  to a set of  $m$  data points  $(t_i, y_i)$ , it is customary and convenient to minimize the sum of the weighted squares of the errors (or weighted residuals) between the data  $y_i$  and the curve-fit function  $\hat{y}(t; \mathbf{c})$ .

$$\chi^2(\mathbf{c}) = \sum_{i=1}^m \left[ \frac{y(t_i) - \hat{y}(t_i; \mathbf{c})}{\sigma_{y_i}} \right]^2 \quad (1)$$

$$= (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{c}))^\top \mathbf{W} (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{c})) \quad (2)$$

$$= \mathbf{y}^\top \mathbf{W} \mathbf{y} - 2\mathbf{y}^\top \mathbf{W} \hat{\mathbf{y}} + \hat{\mathbf{y}}^\top \mathbf{W} \hat{\mathbf{y}} \quad (3)$$

where  $\sigma_{y_i}$  is the measurement error for datum  $y(t_i)$ . Typically the weighting matrix  $\mathbf{W}$  is diagonal with  $W_{ii} = 1/\sigma_{y_i}^2$ . More formally,  $\mathbf{W}$  can be set to the inverse of the measurement

error covariance matrix, in the unusual case that it is known. More generally, the weights  $W_{ii}$ , can be set to pursue other curve-fitting goals. This scalar-valued goodness-of-fit measure is called the *chi-squared error criterion* because the sum of squares of normally-distributed random variables is distributed as the chi-squared distribution.

If the function  $\hat{y}(t; \mathbf{c})$  is nonlinear in the model coefficients  $\mathbf{c}$ , the minimization of  $\chi^2$  with respect to the coefficients must be carried out iteratively. The goal of each iteration is to find a perturbation  $\mathbf{h}$  to the coefficients  $\mathbf{c}$  that reduces  $\chi^2$ .

## 2 The Gradient Descent Method

The steepest descent method is a general minimization method which updates coefficient values in the “downhill” direction: the direction opposite to the gradient of the objective function. The gradient descent method converges well for problems with simple objective functions [8, 9]. For problems with thousands of coefficients, gradient descent methods are sometimes the only viable choice.

The gradient of the chi-squared objective function with respect to the coefficients is

$$\frac{\partial}{\partial \mathbf{c}} \chi^2 = 2(\mathbf{y} - \hat{\mathbf{y}}(\mathbf{c}))^\top \mathbf{W} \frac{\partial}{\partial \mathbf{c}} (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{c})) \quad (4)$$

$$= -2(\mathbf{y} - \hat{\mathbf{y}}(\mathbf{c}))^\top \mathbf{W} \left[ \frac{\partial \hat{\mathbf{y}}(\mathbf{c})}{\partial \mathbf{c}} \right] \quad (5)$$

$$= -2(\mathbf{y} - \hat{\mathbf{y}})^\top \mathbf{W} \mathbf{J} \quad (6)$$

where the  $m \times n$  Jacobian matrix  $[\partial \hat{\mathbf{y}} / \partial \mathbf{c}]$  represents the local sensitivity of the function  $\hat{\mathbf{y}}$  to variation in the coefficients  $\mathbf{c}$ . For notational simplicity the variable  $\mathbf{J}$  will be used for  $[\partial \hat{\mathbf{y}} / \partial \mathbf{c}]$ . Note that in models that are linear in the coefficients,  $\hat{\mathbf{y}} = \mathbf{X} \mathbf{c}$ , the Jacobian  $[\partial \hat{\mathbf{y}} / \partial \mathbf{c}]$  is the matrix of model basis vectors  $\mathbf{X}$ . The coefficient update  $\mathbf{h}$  that moves the coefficients in the direction of steepest descent is given by

$$\mathbf{h}_{\text{gd}} = \alpha \mathbf{J}^\top \mathbf{W} (\mathbf{y} - \hat{\mathbf{y}}) , \quad (7)$$

where the positive scalar  $\alpha$  determines the length of the step in the steepest-descent direction.

## 3 The Gauss-Newton Method

The Gauss-Newton method is a method for minimizing a sum-of-squares objective function. It presumes that the objective function is approximately quadratic in the coefficients near the optimal solution [2]. For moderately-sized problems the Gauss-Newton method typically converges much faster than gradient-descent methods [10].

The function evaluated with perturbed model coefficients may be locally approximated through a first-order Taylor series expansion.

$$\hat{\mathbf{y}}(\mathbf{c} + \mathbf{h}) \approx \hat{\mathbf{y}}(\mathbf{c}) + \left[ \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{c}} \right] \mathbf{h} = \hat{\mathbf{y}} + \mathbf{J} \mathbf{h} , \quad (8)$$

Substituting the approximation  $\hat{\mathbf{y}}(\mathbf{c} + \mathbf{h}) \approx \hat{\mathbf{y}}(\mathbf{c}) + \mathbf{J}\mathbf{h}$  into equation (3) for  $\chi^2(\mathbf{c} + \mathbf{h})$ ,

$$\chi^2(\mathbf{c} + \mathbf{h}) \approx \mathbf{y}^\top \mathbf{W} \mathbf{y} + \hat{\mathbf{y}}^\top \mathbf{W} \hat{\mathbf{y}} - 2\mathbf{y}^\top \mathbf{W} \hat{\mathbf{y}} - 2(\mathbf{y} - \hat{\mathbf{y}})^\top \mathbf{W} \mathbf{J} \mathbf{h} + \mathbf{h}^\top \mathbf{J}^\top \mathbf{W} \mathbf{J} \mathbf{h} . \quad (9)$$

The first-order Taylor approximation (8) results in an approximation for  $\chi^2$  that is quadratic in the perturbation  $\mathbf{h}$ . The Hessian of the chi-squared fit criterion is approximately  $2\mathbf{J}^\top \mathbf{W} \mathbf{J}$ .

The coefficient update  $\mathbf{h}$  that minimizes  $\chi^2$  is found from  $\partial\chi^2/\partial\mathbf{h} = 0$ :

$$\frac{\partial}{\partial\mathbf{h}}\chi^2(\mathbf{c} + \mathbf{h}) \approx -2(\mathbf{y} - \hat{\mathbf{y}})^\top \mathbf{W} \mathbf{J} + 2\mathbf{h}^\top \mathbf{J}^\top \mathbf{W} \mathbf{J} , \quad (10)$$

and the resulting normal equations for the Gauss-Newton update are

$$\left[ \mathbf{J}^\top \mathbf{W} \mathbf{J} \right] \mathbf{h}_{\text{gn}} = \mathbf{J}^\top \mathbf{W} (\mathbf{y} - \hat{\mathbf{y}}) . \quad (11)$$

Note that the right hand side vectors in normal equations for the gradient descent method (7) and the Gauss-Newton method (11) are identical.

## 4 The Levenberg-Marquardt Method

The Levenberg-Marquardt algorithm adaptively varies the coefficient updates between the gradient descent update and the Gauss-Newton update,

$$\left[ \mathbf{J}^\top \mathbf{W} \mathbf{J} + \lambda \mathbf{I} \right] \mathbf{h}_{\text{lm}} = \mathbf{J}^\top \mathbf{W} (\mathbf{y} - \hat{\mathbf{y}}) , \quad (12)$$

where small values of the *damping coefficient*  $\lambda$  result in a Gauss-Newton update and large values of  $\lambda$  result in a gradient descent update. The damping coefficient  $\lambda$  is initialized to be large so that first updates are small steps in the steepest-descent direction. If any iteration happens to result in a worse approximation ( $\chi^2(\mathbf{c} + \mathbf{h}_{\text{lm}}) > \chi^2(\mathbf{c})$ ), then  $\lambda$  is increased. Otherwise, as the solution improves,  $\lambda$  is decreased, the Levenberg-Marquardt method approaches the Gauss-Newton method, and the solution typically accelerates to the local minimum [8, 9, 10].

In Marquardt's update relationship [10], the damping coefficient  $\lambda$  is scaled by the diagonal of the Hessian  $\mathbf{J}^\top \mathbf{W} \mathbf{J}$  for each coefficient.

$$\left[ \mathbf{J}^\top \mathbf{W} \mathbf{J} + \lambda \text{diag}(\mathbf{J}^\top \mathbf{W} \mathbf{J}) \right] \mathbf{h}_{\text{lm}} = \mathbf{J}^\top \mathbf{W} (\mathbf{y} - \hat{\mathbf{y}}) , \quad (13)$$

### 4.1 Numerical Implementation

Many variations of the Levenberg-Marquardt have been published in papers and in code, e.g., [6, 8, 11, 12, 13]. This document borrows from some of these. In iteration  $i$ , the step  $\mathbf{h}$  is evaluated by comparing  $\chi^2(\mathbf{c})$  to  $\chi^2(\mathbf{c} + \mathbf{h})$ . The step is accepted if the metric  $\rho_i$  [11] is greater than a user-specified threshold,  $\epsilon_4 > 0$ . This metric is a measure of the

actual improvement in  $\chi^2$  as compared to the improvement of an LM update assuming the approximation (8) were exact.

$$\begin{aligned} \rho_i(\mathbf{h}_{lm}) &= \frac{\chi^2(\mathbf{c}) - \chi^2(\mathbf{c} + \mathbf{h}_{lm})}{|(\mathbf{y} - \hat{\mathbf{y}})^\top \mathbf{W}(\mathbf{y} - \hat{\mathbf{y}}) - (\mathbf{y} - \hat{\mathbf{y}} - \mathbf{J}\mathbf{h}_{lm})^\top \mathbf{W}(\mathbf{y} - \hat{\mathbf{y}} - \mathbf{J}\mathbf{h}_{lm})|} & (14) \\ &= \frac{\chi^2(\mathbf{c}) - \chi^2(\mathbf{c} + \mathbf{h}_{lm})}{|\mathbf{h}_{lm}^\top (\lambda_i \mathbf{h}_{lm} + \mathbf{J}^\top \mathbf{W}(\mathbf{y} - \hat{\mathbf{y}}(\mathbf{c})))|} & \text{if using eq'n (12) for } \mathbf{h}_{lm} \text{ (15)} \\ &= \frac{\chi^2(\mathbf{c}) - \chi^2(\mathbf{c} + \mathbf{h}_{lm})}{|\mathbf{h}_{lm}^\top (\lambda_i \text{diag}(\mathbf{J}^\top \mathbf{W} \mathbf{J}) \mathbf{h}_{lm} + \mathbf{J}^\top \mathbf{W}(\mathbf{y} - \hat{\mathbf{y}}(\mathbf{c})))|} & \text{if using eq'n (13) for } \mathbf{h}_{lm} \text{ (16)} \end{aligned}$$

If in an iteration  $\rho_i(\mathbf{h}) > \epsilon_4$  then  $\mathbf{c} + \mathbf{h}$  is sufficiently better than  $\mathbf{c}$ ,  $\mathbf{c}$  is replaced by  $\mathbf{c} + \mathbf{h}$ , and  $\lambda$  is reduced by a factor. Otherwise  $\lambda$  is increased by a factor, and the algorithm proceeds to the next iteration.

#### 4.1.1 Initialization and update of the L-M coefficient, $\lambda$ , and the coefficients $\mathbf{a}$

Here are three options for initializing and updating  $\lambda$  and  $\mathbf{c}$ .

1.  $\lambda_0 = \lambda_o$ ;  $\lambda_o$  is user-specified [10].  
 use eq'n (13) for  $\mathbf{h}_{lm}$  and eq'n (16) for  $\rho$   
 if  $\rho_i(\mathbf{h}) > \epsilon_4$ :  $\mathbf{c} \leftarrow \mathbf{c} + \mathbf{h}$ ;  $\lambda_{i+1} = \max[\lambda_i/L_\downarrow, 10^{-7}]$ ;  
 otherwise:  $\lambda_{i+1} = \min[\lambda_i L_\uparrow, 10^7]$ ;
2.  $\lambda_0 = \lambda_o \max[\text{diag}[\mathbf{J}^\top \mathbf{W} \mathbf{J}]]$ ;  $\lambda_o$  is user-specified.  
 use eq'n (12) for  $\mathbf{h}_{lm}$  and eq'n (15) for  $\rho$   
 $\alpha = \left( (\mathbf{J}^\top \mathbf{W}(\mathbf{y} - \hat{\mathbf{y}}(\mathbf{c})))^\top \mathbf{h} \right) / \left( (\chi^2(\mathbf{c} + \mathbf{h}) - \chi^2(\mathbf{c})) / 2 + 2 \left( \mathbf{J}^\top \mathbf{W}(\mathbf{y} - \hat{\mathbf{y}}(\mathbf{c})) \right)^\top \mathbf{h} \right)$ ;  
 if  $\rho_i(\alpha \mathbf{h}) > \epsilon_4$ :  $\mathbf{c} \leftarrow \mathbf{c} + \alpha \mathbf{h}$ ;  $\lambda_{i+1} = \max[\lambda_i / (1 + \alpha), 10^{-7}]$ ;  
 otherwise:  $\lambda_{i+1} = \lambda_i + |\chi^2(\mathbf{c} + \alpha \mathbf{h}) - \chi^2(\mathbf{c})| / (2\alpha)$ ;
3.  $\lambda_0 = \lambda_o \max[\text{diag}[\mathbf{J}^\top \mathbf{W} \mathbf{J}]]$ ;  $\lambda_o$  is user-specified [11].  
 use eq'n (12) for  $\mathbf{h}_{lm}$  and eq'n (15) for  $\rho$   
 if  $\rho_i(\mathbf{h}) > \epsilon_4$ :  $\mathbf{c} \leftarrow \mathbf{c} + \mathbf{h}$ ;  $\lambda_{i+1} = \lambda_i \max[1/3, 1 - (2\rho_i - 1)^3]$ ;  $\nu_i = 2$ ;  
 otherwise:  $\lambda_{i+1} = \lambda_i \nu_i$ ;  $\nu_{i+1} = 2\nu_i$ ;

The examples in section 4.4 illustrate the use of method 1 [10] and exhibit good convergence properties with  $L_\uparrow \approx 11$  and  $L_\downarrow \approx 9$ .

#### 4.1.2 Computation and rank-1 update of the Jacobian, $[\partial \mathbf{y} / \partial \mathbf{p}]$

For problems with many coefficients, a finite differences Jacobian is computationally expensive. If the Jacobian is re-computed using finite differences only occasionally, convergence can be achieved with fewer function evaluations. In the first iteration, in every  $2n$

iterations, and in iterations where  $\chi^2(\mathbf{c} + \mathbf{h}) > \chi^2(\mathbf{c})$ , the Jacobian ( $\mathbf{J} \in \mathbb{R}^{m \times n}$ ) is numerically approximated using forward differences,

$$J_{ij} = \frac{\partial \hat{y}_i}{\partial c_j} = \frac{\hat{y}(t_i; \mathbf{c} + \delta \mathbf{c}_j) - \hat{y}(t_i; \mathbf{c})}{\|\delta \mathbf{c}_j\|}, \quad (17)$$

or central differences (default)

$$J_{ij} = \frac{\partial \hat{y}_i}{\partial c_j} = \frac{\hat{y}(t_i; \mathbf{c} + \delta \mathbf{c}_j) - \hat{y}(t_i; \mathbf{c} - \delta \mathbf{c}_j)}{2\|\delta \mathbf{c}_j\|}, \quad (18)$$

where the  $j$ -th element of  $\delta \mathbf{c}_j$  is the only non-zero element and is set to  $\Delta_j(1 + |c_j|)$ . In all other iterations, the Jacobian is updated using the Broyden rank-1 update formula [3, 4],

$$\mathbf{J} = \mathbf{J} + \left( (\hat{\mathbf{y}}(\mathbf{c} + \mathbf{h}) - \hat{\mathbf{y}}(\mathbf{c}) - \mathbf{J}\mathbf{h}) \mathbf{h}^\top \right) / \left( \mathbf{h}^\top \mathbf{h} \right). \quad (19)$$

The rank-1 Jacobian update equation (19) requires no additional function evaluations.

#### 4.1.3 Convergence criteria

Convergence is achieved when *one* of the following three criteria is satisfied,

- Convergence in the gradient:  $\max |\mathbf{J}^\top \mathbf{W}(\mathbf{y} - \hat{\mathbf{y}})| < \epsilon_1$ ;
- Convergence in coefficients:  $\max |h_i/c_i| < \epsilon_2$ ; or
- Convergence in  $\chi^2$ : uses the value of the *reduced*  $\chi^2$ ,  $\chi_\nu^2 = \chi^2/(m - n) < \epsilon_3$ .

Otherwise, iterations terminate when the iteration count exceeds a pre-specified limit.

#### 4.1.4 Regularizing ill-conditioned matrix equations

If the system of linear equations (12) or (13) for  $\mathbf{h}$  is ill-conditioned, it can be regularized before solving for  $\mathbf{h}$  by recursively replacing the left hand side matrix

$$\mathbf{X} = [\mathbf{J}^\top \mathbf{W} \mathbf{J} + \lambda \mathbf{I}] \quad \text{or} \quad \mathbf{X} = [\mathbf{J}^\top \mathbf{W} \mathbf{J} + \lambda \text{diag}(\mathbf{J}^\top \mathbf{W} \mathbf{J})]$$

with  $\mathbf{X} + \mathbf{D}$  where  $\mathbf{D}$  is a small diagonal matrix  $\mathbf{D} = 10^{-6} n^{-1} \text{trace}(\mathbf{X}) \mathbf{I}$  until the condition number of  $\mathbf{X} + \mathbf{D}$  is not excessively high. In such cases the resulting step  $\mathbf{h}$  can be reduced by a factor of 10 so that steps associated with poorly-conditioned systems are not large.

#### 4.1.5 Multiple minima and sensitivity to the initial guess

In nonlinear least squares problems the  $\chi^2(\mathbf{c})$  objective function may have multiple minima and solutions may be sensitive to the initial guess. In such cases the Levenberg-Marquardt method may converge to an astonishingly poor fit. If this happens, one may compute a set of solutions from a set of randomly-generated initial guesses, and select the solution with the smallest reduced  $\chi^2$  value.

## 4.2 Error Analysis

Once the optimal curve-fit coefficients  $\mathbf{c}_{\text{fit}}$  are determined, coefficient statistics are computed for the converged solution. If the measurement error covariance matrix  $\mathbf{V}_y$ , or its diagonal,  $\sigma_{y_i}^2$ , is known a priori, (prior to the curve-fit), the weighting matrix should be set to the inverse of the measurement error covariance,  $\mathbf{W} = \mathbf{V}_y^{-1}$ , in estimating the model coefficients and in the following error analysis. Note that if the actual measurement errors vary significantly across the measurement points (i.e.,  $\max_i(\sigma_{y_i})/\min_i(\sigma_{y_i}) > 10$ ), any error analysis that presumes equal measurement errors will be incorrect.

The reduced  $\chi^2$  error criterion,

$$\chi_\nu^2 = \chi^2 / (m - n) = (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{c}_{\text{fit}}))^T \mathbf{W} (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{c}_{\text{fit}})) / (m - n) \quad (20)$$

is an unbiased measure of the quality of the fit. Large values ( $\chi_\nu^2 \gg 1$ ) indicate a poor fit due to: under-estimating the true measurement error, under-parameterizing the model, or the presence of correlated measurement noise. Small values ( $\chi_\nu^2 < 1$ ) indicate an unreasonably good fit due to: over-estimating the true measurement error, over-parameterizing the model (the model is fitting the measurement noise). Values close to 1 ( $0.5 < \chi_\nu^2 < 1.5$ ) indicate that the fit error is of the same order as the measurement error (as desired),

The coefficient covariance matrix is computed from

$$\mathbf{V}_p = [\mathbf{J}^T \mathbf{W} \mathbf{J}]^{-1} . \quad (21)$$

The asymptotic standard coefficient errors,

$$\sigma_c = \sqrt{\text{diag}([\mathbf{J}^T \mathbf{W} \mathbf{J}]^{-1})} , \quad (22)$$

give a measure of how unexplained variability in the data propagates to variability in the coefficients, and is essentially an error measure for the coefficients.

The asymptotic standard error of the fit,

$$\sigma_{\hat{\mathbf{y}}} = \sqrt{\text{diag}(\mathbf{J} [\mathbf{J}^T \mathbf{W} \mathbf{J}]^{-1} \mathbf{J}^T)} . \quad (23)$$

indicates how variability in the coefficients affects the variability in the curve-fit.

The asymptotic standard prediction error,

$$\sigma_{\hat{y}_p} = \sqrt{\text{diag}(\mathbf{V}_y + \mathbf{J} [\mathbf{J}^T \mathbf{W} \mathbf{J}]^{-1} \mathbf{J}^T)} . \quad (24)$$

reflects the standard error of the fit as well as the measurement error.

If the measurement error covariance, or individual measurement errors are *not known* in advance of the analysis, the error analysis can be carried out assuming the same measurement error for every measurement point, as estimated from the fit,

$$\hat{\sigma}_y^2 = (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{c}_{\text{fit}}))^T (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{c}_{\text{fit}})) / (m - n) . \quad (25)$$

In this case  $\mathbf{V}_y$  is set to  $\hat{\sigma}_y^2 \mathbf{I}$  and  $\mathbf{W}$  is set to  $\mathbf{I}/\hat{\sigma}_y^2$  in equations (21) to (24).

## 5 Code: `lm.py`

The `.py`-function `levenberg_marquardt` implements the Levenberg-Marquardt method for curve-fitting problems. The code with examples are available through the [multivarious](#) package.

The `.py`-file to solve a least-squares curve-fit problem with `levenberg_marquardt` can be as simple as:

```

1 import numpy as np
2 from lm import levenberg_marquardt
3
4 # Define the model function
5 def model(t, coeffs):
6     return coeffs[0] * np.exp(-t / coeffs[1])
7
8 # Generate or load data
9 t = np.linspace(0, 5, 100)
10 y_data = 10 * np.exp(-t / 2) + 0.5 * np.random.randn(len(t))
11
12 # Initial guess
13 coeffs_init = np.array([5.0, 1.0])
14
15 # Fit the model
16 result = levenberg_marquardt(
17     model, coeffs_init, t, y_data,
18     print_level=1 # 0=silent, 1=summary, 2=iterations, 3=iterations+plots
19 )
20
21 print(f"Fitted coefficients: {result.coefficients}")
22 print(f"Standard errors:      {result.sigma_coefficients}")
23 print(f"R-squared:           {result.r_squared:.4f}")
24 print(f"Reduced chi-squared: {result.reduced_chi_sq:.4f}")

```

It is common and desirable to repeat the same experiment two or more times and to estimate a single set of curve-fit coefficients from all the experiments. In such cases the data file may be arranged as follows:

1	%	<i>t</i> -variable	<i>y</i> (1st experiment)	<i>y</i> (2nd experiment)	<i>y</i> (3rd experiment)
2		0.50000	3.5986	3.60192	3.58293
3		0.80000	8.1233	8.01231	8.16234
4		0.90000	12.2342	12.29523	12.01823
5		:	:	:	:
6		etc.	etc.	etc.	etc.

If your data is arranged as above you may prepare the data for `levenberg_marquardt` using the following lines.

```

1 t_column = 1 # column of the independent variable
2 y_columns = [ 2 , 3 , 4 ] # columns of the measured dependent variables
3
4 y_dat = my_data[:,y_columns] # the measured data
5 y_dat = y_dat[:] # a single column vector
6
7 t = my_data[:,t_column] # the independent variable
8 t = t*ones[1,len(y_columns)] # a column of t for each column of y
9 t = t[:] # a single column vector

```

Note that the arguments `t` and `y_dat` to `levenberg_marquardt` may be matrices as long as the dimensions of `t` match the dimensions of `y_dat`. The columns of `t` need not be identical.

Tips for successful use of `levenberg_marquardt`:

- The data vector `t` *should* be a *column* vector, or columns of `t` *must* correspond to columns of `y_dat`.
- The data vector `y_dat` *should* be a *column* vector.
- Your `.py`-function `lm_func.py` *must* return the vector `y_hat` as a *column* vector.
- The vectors `p_init`, `p_min`, and `p_max` *must* be *column* vectors.
- Parameter values should be scaled to values in a compact range, for example, such that absolute coefficients values are between 1 and 100.

Visualization tools are included.

## 6 Examples

In this section, the use of `levenberg_marquardt` is illustrated in three curve-fitting examples in which experimental measurements are numerically simulated. Noisy experimental measurements  $y$  are simulated by adding random measurement noise to the curve-fit function evaluated with a set of “true” coefficient values  $\hat{y}(t; \mathbf{c}_{\text{true}})$ . The random measurement noise is normally distributed with a mean of zero and a standard deviation of 0.50.

$$y_i = \hat{y}(t_i; \mathbf{c}_{\text{true}}) + \mathcal{N}(0, 0.50). \quad (26)$$

The convergence of the coefficients from an erroneous initial guess  $\mathbf{c}_{\text{init}}$  to values closer to  $\mathbf{c}_{\text{true}}$  is then examined.

Each numerical example below has four coefficients ( $n = 4$ ) and one-hundred measurements ( $m = 100$ ). Each numerical example has a different curve-fit function  $\hat{y}(t; \mathbf{c})$ , a different “true” coefficient vector  $\mathbf{c}_{\text{true}}$ , and a different vector of initial coefficients  $\mathbf{c}_{\text{init}}$ .

For several values of  $c_2$  and  $c_4$ , the log of the reduced  $\chi^2$  error criterion is calculated and is plotted as a surface over the  $c_2 - c_4$  plane. The “bowl-shaped” nature of the objective function is clearly evident in each example. In some cases, the objective function is not quadratic in the coefficients or the objective function has multiple minima. The presence of measurement noise does not affect the smoothness of the objective function.

The gradient descent method endeavors to move coefficient values in a down-hill direction to minimize  $\chi^2(\mathbf{c})$ . This often requires small step sizes but is required when the objective function is not quadratic. The Gauss-Newton method approximates the bowl shape as a quadratic and endeavors to move coefficient values to the minimum in a small number of steps. This method works well when the coefficients are close to their optimal values. The Levenberg-Marquardt method retains the best features of both the gradient-descent method and the Gauss-Newton method.

The evolution of the coefficient values, the evolution of  $\chi^2_\nu$ , and the evolution of  $\lambda$  from iteration to iteration is plotted for each example.

The simulated experimental data, the curve fit, and the 99-percent confidence interval of the fit are plotted, the standard error of the fit, and a histogram of the fit errors are also plotted.

The initial coefficient values  $\mathbf{c}_{\text{init}}$ , the true coefficient values  $\mathbf{c}_{\text{true}}$ , the fit coefficient values  $\mathbf{c}_{\text{fit}}$ , the standard error of the fit coefficients  $\sigma_c$ , and the correlation matrix of the fit coefficients are tabulated. The true coefficient values lie within the confidence interval  $c_{\text{fit}} - 2.58\sigma_p < c_{\text{true}} < c_{\text{fit}} + 2.58\sigma_p$  with a confidence level of 99 percent.

## 6.1 Example 1

Consider fitting the following function to a set of measured data.

$$\hat{y}(t; \mathbf{c}) = c_1 \left(\frac{t}{T}\right) + c_2 \left(\frac{t}{T}\right)^2 + c_3 \left(\frac{t}{T}\right)^3 + c_4 \left(\frac{t}{T}\right)^4 \quad (27)$$

over the range  $0 \leq t \leq T$ . This function is linear in the coefficients and may be fit easily using methods of linear least squares. The `.py`-function to be used with `levenberg_marquardt` is simply:

```

1 def lm_func(t, a, c):
2     T = np.max(t)
3     y_hat = a[1]*(t/T) + a[2]*(t/T)**2 + a[3]*(t/T)**3 + a[4]*(t/T)**4
4     return y_hat

```

The “true” coefficient values  $\mathbf{c}_{\text{true}}$ , the initial coefficient values  $\mathbf{c}_{\text{init}}$ , resulting curve-fit coefficient values  $\mathbf{c}_{\text{fit}}$  and standard errors of the fit coefficients  $\sigma_c$  are shown in Table 1. The  $R^2$  fit criterion is 99.9 percent and  $\chi_\nu^2 = 0.969$ . The standard error for  $c_1$  is 8 percent and the standard error of  $c_3$  is 47 percent of the estimated value. Note that a very high value of the  $R^2$  coefficient of determination does not necessarily mean that coefficient values have been found with great accuracy. The high value of  $R^2$  merely indicates that the fit is highly correlated with the data. In this example the high  $R^2$  value is a result of relatively low measurement noise (compared to the data values) and a fit that passes through the data points. The coefficient correlation matrix is given in Table 2. These coefficients are highly correlated with one another, meaning that a change in one coefficient will almost certainly result in changes in the other coefficients. The high values of coefficient standard errors, coefficient correlations, and  $\chi_\nu^2$  indicate that  $\hat{y}(t; \mathbf{c})$  is over-parameterized.

The bowl-shaped nature of the  $\chi^2$  objective function is shown in Figure 2(a). This shape is nearly quadratic and has a single minimum. The correlation of coefficients  $c_2$  and  $c_4$ , for example, is easily seen from this figure.

The convergence of the coefficients and the evolution of  $\chi_\nu^2$  and  $\lambda$  are shown in Figure 1(b). The coefficients converge monotonically to their final values.

The data points, the curve fit, and the curve fit confidence band are plotted in Figure 1(c). Note that the standard error of the fit approaches zero at  $t = 0$  and is largest at  $t = 100$ . This is because  $\hat{y}(0; \mathbf{c}) = 0$ , regardless of the values in  $\mathbf{c}$ .

A histogram of the difference between the data values and the curve-fit is shown in Figure 1(d). Ideally these curve-fit errors should be normally distributed, and they appear to be so in this example.

Table 1. Parameter values and standard errors.

$c_{init}$	$c_{true}$	$c_{fit}$	$\sigma_c$	$\sigma_c/c_{fit}$ (%)
11.8	20.0	19.668	1.693	8.61
-7.8	-24.0	-24.089	9.138	39.57
56.0	30.0	28.286	15.371	52.49
-20.0	-40.0	-39.762	8.094	20.36

Table 2. Parameter correlation matrix.

	$c_1$	$c_2$	$c_3$	$c_4$
$c_1$	1.00	-0.97	0.92	-0.87
$c_2$	-0.97	1.00	-0.99	0.95
$c_3$	0.92	-0.99	1.00	-0.99
$c_4$	-0.87	0.95	-0.99	1.00

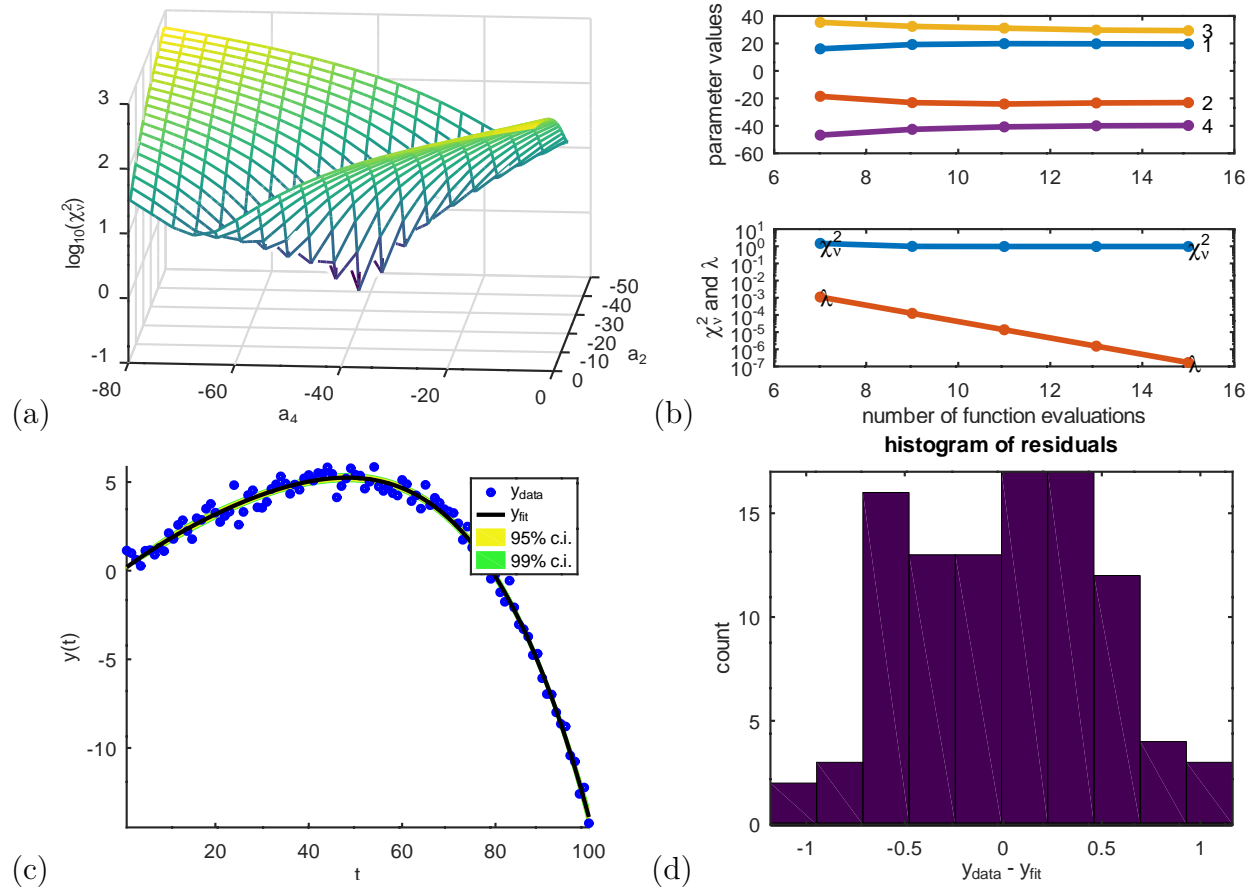


Figure 1. (a) The sum of the squared errors as a function of  $c_2$  and  $c_4$ . (b) Top: the convergence of the coefficients with each iteration, (b) Bottom: values of  $\chi^2$  and  $\lambda$  each iteration. (c) Top: data  $y$ , curve-fit  $\hat{y}(t; c_{fit})$ , curve-fit confidence intervals; (d) Histogram of the errors between the data and the fit.

## 6.2 Example 2

Consider fitting the following function to a set of measured data.

$$\hat{y}(t; \mathbf{c}) = c_1 \exp\left(-\frac{t}{c_2}\right) + c_3 t \exp\left(-\frac{t}{c_4}\right) \quad (28)$$

The `.py`-function to be used with `levenberg_marquardt` is simply:

```

1 def lm_func(t, a, c):
2     y_hat = a[1]*np.exp(-t/a[2]) + a[3]*t*np.exp(-t/a[4])
3     return y_hat

```

The “true” coefficient values  $\mathbf{c}_{\text{true}}$ , the initial coefficient values  $\mathbf{c}_{\text{init}}$ , resulting curve-fit coefficient values  $\mathbf{c}_{\text{fit}}$  and standard errors of the fit coefficients  $\sigma_{\mathbf{c}}$  are shown in Table 3. The  $R^2$  fit criterion is 89 percent and the reduced  $\chi^2_{\nu} = 1.15$ . The standard coefficient errors are one to five percent of the coefficient values. The coefficient correlation matrix is given in Table 4. Parameters  $c_3$  and  $c_4$  are the most correlated at -96 percent. Parameters  $c_1$  and  $c_3$  are the least correlated at +27 percent.

The bowl-shaped nature of the  $\chi^2$  objective function is shown in Figure 2(a). This shape is nearly quadratic and has a single minimum.

The convergence of the coefficients and the evolution of  $\chi^2_{\nu}$  and  $\lambda$  are shown in Figure 2(b).

The data points, the curve fit, and the curve fit confidence band are plotted in Figure 2(c). Note that the standard error of the fit is smaller near the center of the fit domain and is larger at the edges of the domain.

A histogram of the difference between the data values and the curve-fit is shown in Figure 2(d). Ideally these curve-fit errors should be normally distributed.

Table 3. Parameter values and standard errors.

$c_{init}$	$c_{true}$	$c_{fit}$	$\sigma_c$	$\sigma_c/c_{fit}$ (%)
9.1	20.0	19.951	0.381	1.96
11.8	10.0	10.078	0.392	3.89
8.7	1.0	1.003	0.015	1.48
98.6	50.0	49.866	0.533	1.07

Table 4. Parameter correlation matrix.

	$c_1$	$c_2$	$c_3$	$c_4$
$c_1$	1.00	-0.75	0.40	-0.36
$c_2$	-0.74	1.00	-0.78	0.71
$c_3$	0.40	-0.78	1.00	-0.97
$c_4$	-0.36	0.71	-0.97	1.00

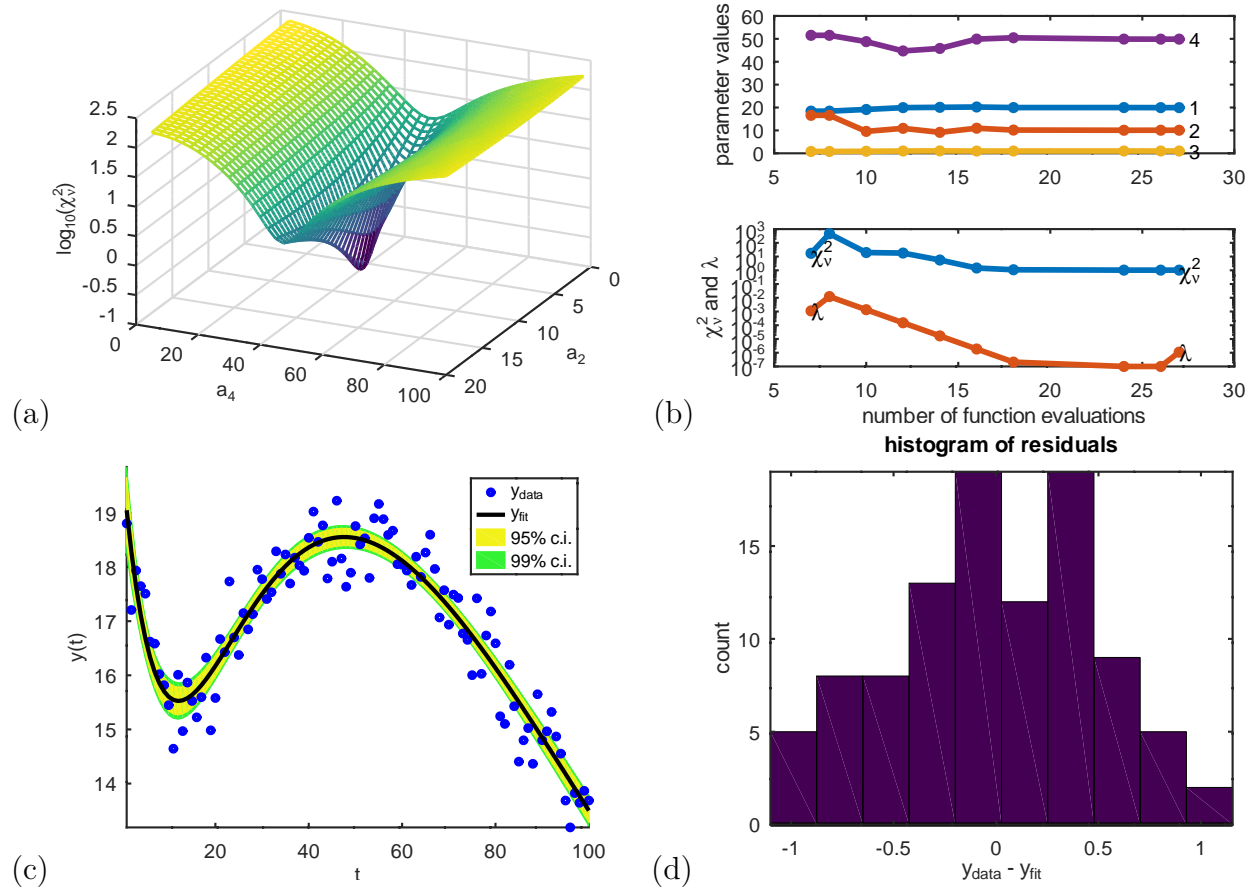


Figure 2. (a) The sum of the squared errors as a function of  $c_2$  and  $c_4$ . (b) Top: the convergence of the coefficients with each iteration, (b) Bottom: values of  $\chi^2$  and  $\lambda$  each iteration. (c) Top: data  $y$ , curve-fit  $\hat{y}(t; c_{fit})$ , curve-fit confidence intervals; (d) Histogram of the errors between the data and the fit.

### 6.3 Example 3

Consider fitting the following function to a set of measured data.

$$\hat{y}(t; \mathbf{c}) = c_1 \exp\left(-\frac{t}{c_2}\right) + c_3 \sin\left(\frac{t}{c_4}\right) \quad (29)$$

This function is linear  $c_1$  and  $c_3$  but not in  $c_2$  and  $c_4$ . The `.py`-function to be used with `levenberg_marquardt` is simply:

```

1 def lm_func(t, a, c):
2     y_hat = a[1]*np.exp(-t/a[2]) + a[3]*sin(t/a[4])
3     return y_hat

```

The “true” coefficient values  $\mathbf{c}_{\text{true}}$ , the initial coefficient values  $\mathbf{c}_{\text{init}}$ , resulting curve-fit coefficient values  $\mathbf{c}_{\text{fit}}$  and standard errors of the fit coefficients  $\sigma_c$  are shown in Table 5. The  $R^2$  fit criterion is 93 percent and  $\chi_\nu^2 = 0.948$ . In this example, the standard coefficient errors are all less than ten percent. The coefficient correlation matrix is given in Table 6. Parameters  $c_4$  is not correlated with the other coefficients. Parameters  $c_1$  and  $c_2$  are most correlated at 73 percent.

The bowl-shaped nature of the  $\chi_\nu^2$  objective function is shown in Figure 3(a). This shape is clearly not quadratic and has multiple minima. In this example, the initial guess for coefficient  $c_4$ , the period of the oscillatory component, has to be within ten percent of the true value, otherwise the algorithm in `levenberg_marquardt` will converge to a very small value of the amplitude of oscillation  $c_3$  and an erroneous value for  $c_4$ . When such an occurrence arises, the standard errors  $\sigma_c$  of the fit coefficients  $c_3$  and  $c_4$  are quite large and the histogram of curve-fit errors (Figure 3(d)) is not normally distributed.

The convergence of the coefficients and the evolution of  $\chi_\nu^2$  and  $\lambda$  are shown in Figure 3(b). The coefficients converge monotonically to their final values.

The data points, the curve fit, and the curve fit confidence band are plotted in Figure 3(c).

A histogram of the difference between the data values and the curve-fit is shown in Figure 1(d). Ideally these curve-fit errors should be normally distributed, and they appear to be so in this example.

Table 5. Parameter values and standard errors.

$c_{init}$	$c_{true}$	$c_{fit}$	$\sigma_c$	$\sigma_c/c_{fit}$ (%)
10.8	6.0	6.161	0.245	3.97
39.3	20.0	18.854	1.008	5.34
0.7	1.0	0.778	0.071	9.20
6.1	5.0	4.970	0.040	0.81

Table 6. Parameter correlation matrix.

	$c_1$	$c_2$	$c_3$	$c_4$
$c_1$	1.00	-0.73	-0.25	0.10
$c_2$	-0.73	1.00	0.14	-0.16
$c_3$	-0.25	0.14	1.00	0.03
$c_4$	0.10	-0.16	0.03	1.00

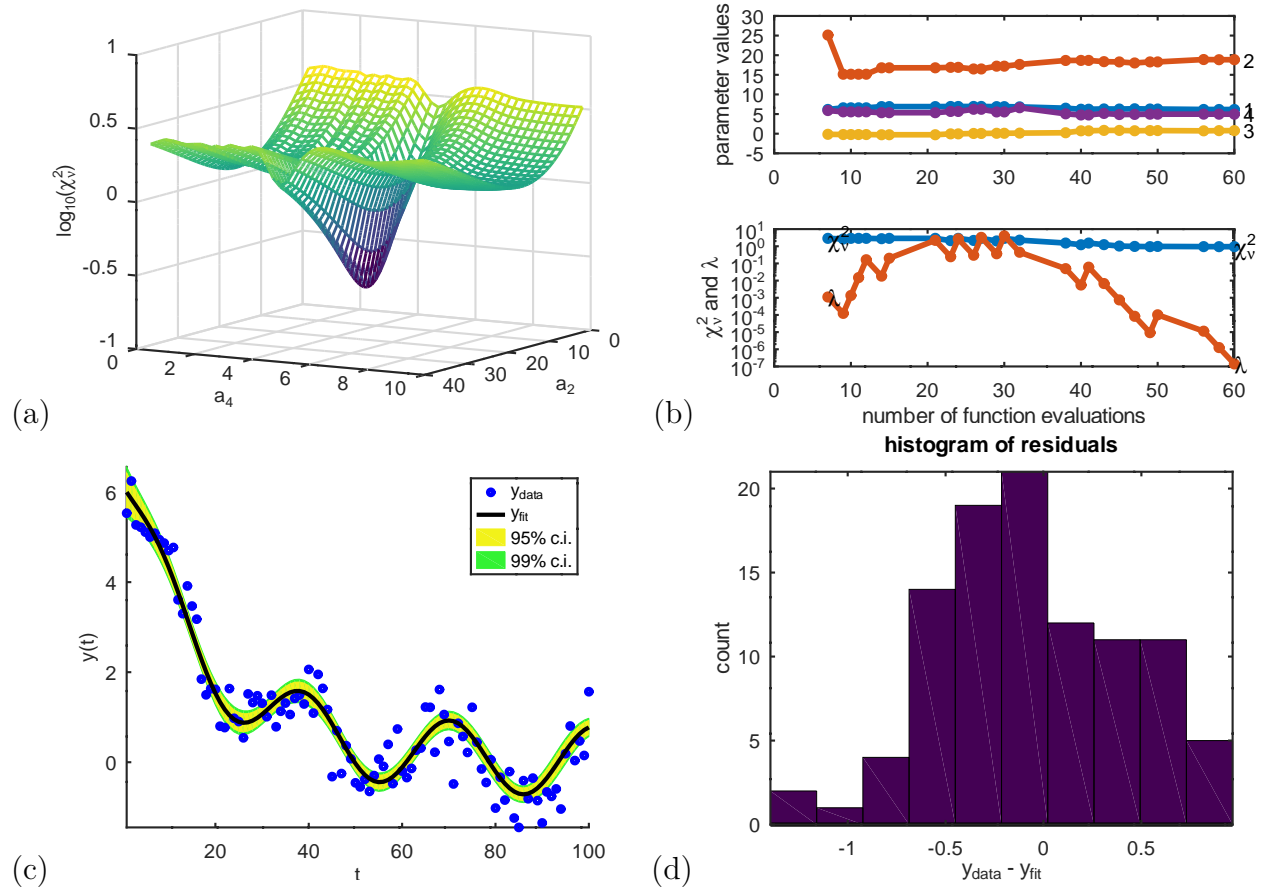


Figure 3. (a) The sum of the squared errors as a function of  $c_2$  and  $c_4$ . (b) Top: the convergence of the coefficients with each iteration, (b) Bottom: values of  $\chi^2$  and  $\lambda$  each iteration. (c) Top: data  $y$ , curve-fit  $\hat{y}(t; c_{fit})$ , curve-fit confidence intervals; (d) Histogram of the errors between the data and the fit.

## 6.4 Multiple Minima

In nonlinear least squares problems in which the  $\chi^2$  objective function has multiple minima, (e.g., example 3), the numerical solution to the least squares minimization depends on the initial guess used for the model coefficients. Figures 4, 6 and 8 show distributions of multiple initial guesses and the associated fits for the four coefficients in examples 1, 2 and 3. Note that the fifteen best initial guesses (initial coefficient values that resulted in the lowest  $\chi^2$  values) are not clumped around the true coefficient values, but are distributed across the full range of potential initial guesses (from 0.1 to 2.0 times the true coefficient value in this example). The Levenberg-Marquardt method (as described herein) does not necessarily converge to the closest local minimum. A good initial guess need not be very close to the true coefficient values. So, in solving nonlinear least squares problems with multiple minima it is suitable to compute several solutions using a random sample of multiple initial guesses and to select the solution that has the lowest  $\chi^2$  value.

Example 1 is a linear least squares problem. Figure 4 shows that 500 initial coefficient values uniformly distributed between 0.1 and 2 times the true value result in fit coefficients very close to the true value. The  $\chi^2_{\nu}$  value is 0.9698 (to within four significant figures) for all initial guesses. Note that for coefficients  $c_1$ ,  $c_2$  and  $c_3$ , the best initial guess (magenta) is further from the optimal initial guess than the worst initial guess. For  $c_4$ , the best initial guess is very close to the true value.

Example 2 is an “easy” nonlinear least squares problem. Figure 6 shows that almost all of the 500 initial coefficient values uniformly distributed between 0.1 and 2 times the true value result in fit coefficients very close to the true value. About ten converge to values that are very far from the true value. The  $\chi^2_{\nu}$  value is 1.15 (to within three significant figures) for all initial guesses but the worst ten. The worst  $\chi^2_{\nu}$  value is around 2000. For coefficients  $c_2$  and  $c_4$ , the best initial guess (magenta) is close to the true value.

Example 3 is a “hard” nonlinear least squares problem. Figure 8 shows that many of the 500 initial coefficient values uniformly distributed between 0.1 and 2 times the true value result in fit coefficients very close to the true value. And many others do not. The best  $\chi^2_{\nu}$  value is about 0.95. The worst is around 12. Good initial guesses are not necessarily very close to the true coefficient values, and poor initial guesses can be close to the true values. In such cases, initiating fits from multiple random initial guesses is helpful

The initial guesses used to generate Figures 1, 2 and 3 are the fifteenth best initial guess (one of the green points) from the samples shown in Figures 4, 6 and 8. In these examples, a sample of 500 initial guesses is much more than large enough to obtain a very good fit.

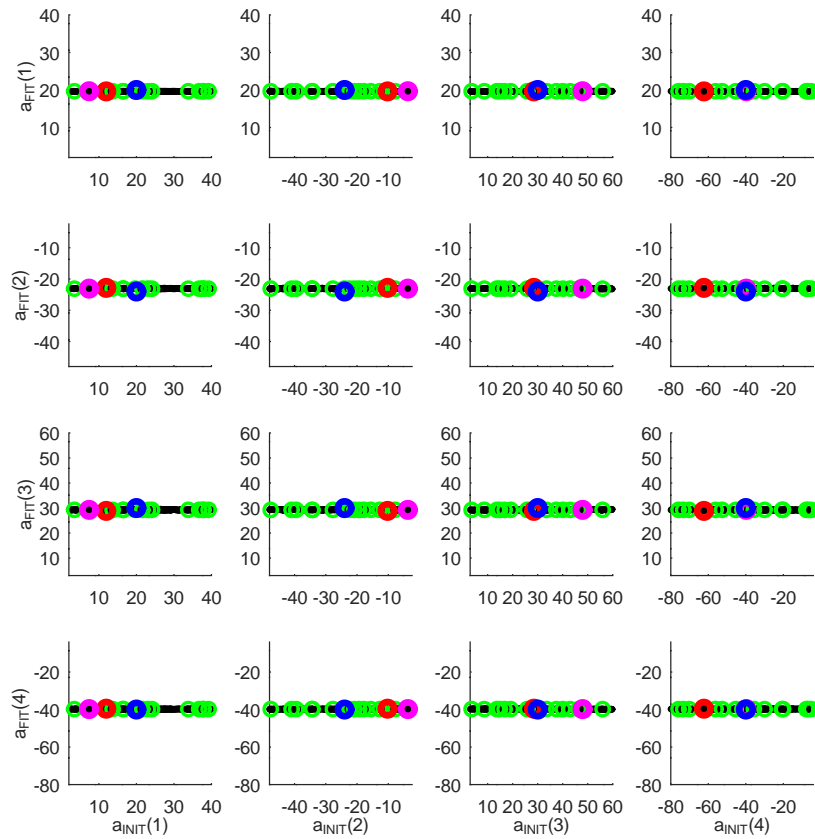


Figure 4. Example 1. A sample of 500 uniformly distributed random initial guesses and converged (fit) solutions. black: initial guesses and converged (fit) solutions; blue: true coefficient values; red: the worst initial guess and the associated converged (fit) coefficient values. green: the fifteen best initial guesses and the associated converged (fit) coefficient values. magenta: the best initial guess and the associated converged (fit) coefficient values. The fit is excellent and the residuals are roughly normally distributed.

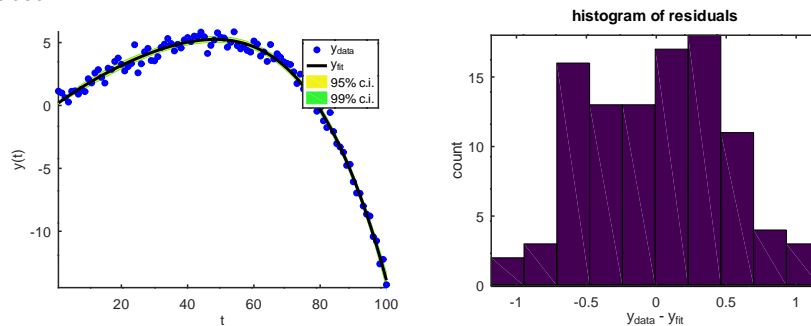


Figure 5. left: data  $y$ , curve-fit  $\hat{y}(t; \mathbf{c}_{fit})$ , curve-fit confidence intervals for the fit with the worst  $\chi^2_{\nu}$  value - using fit coefficients shown in red in Figure 4. right: Histogram of the errors between the data and the fit.

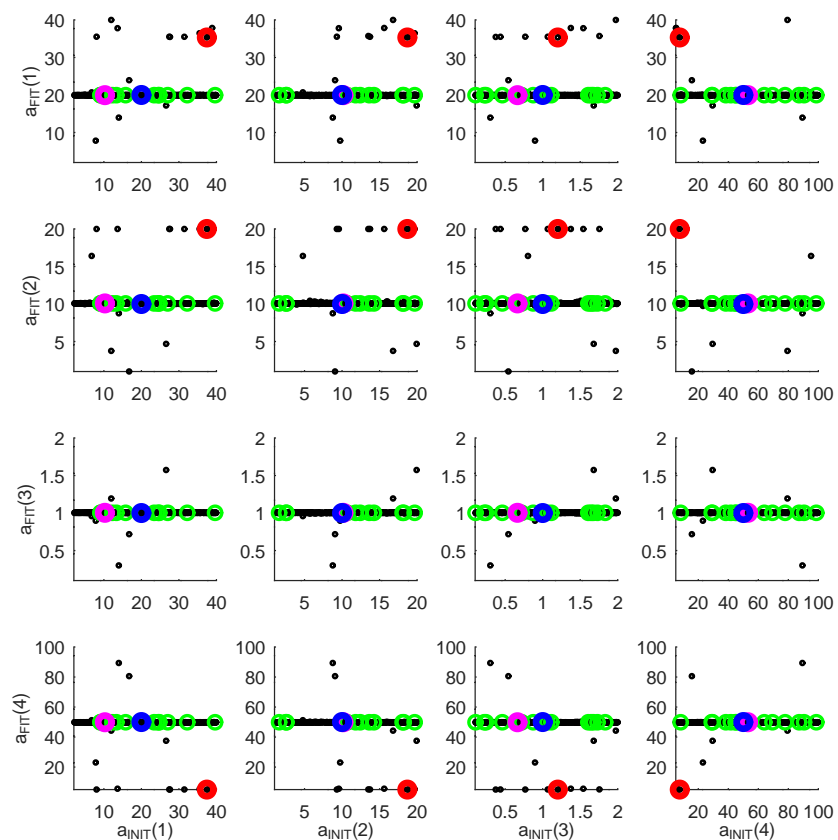


Figure 6. Example 2. A sample of 500 uniformly distributed initial guesses and converged (fit) solutions. black: initial guesses and converged (fit) solutions; blue: true coefficient values; red: the worst initial guess and the associated converged (fit) coefficient values. green: the fifteen best initial guesses and the associated converged (fit) coefficient values. magenta: the best initial guess and the associated converged (fit) coefficient values.

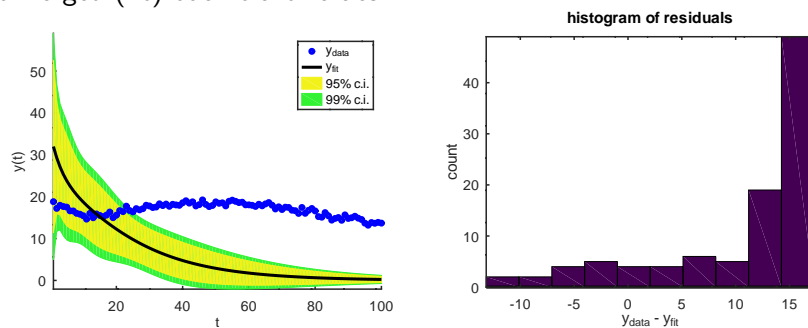


Figure 7. left: data  $y$ , curve-fit  $\hat{y}(t; \mathbf{c}_{\text{fit}})$ , curve-fit confidence intervals for the fit with the worst  $\chi^2_{\nu}$  value - using fit coefficients shown in red in Figure 6. right: Histogram of the errors between the data and the fit. The fit is astonishingly poor and the residuals are clearly not normally distributed.

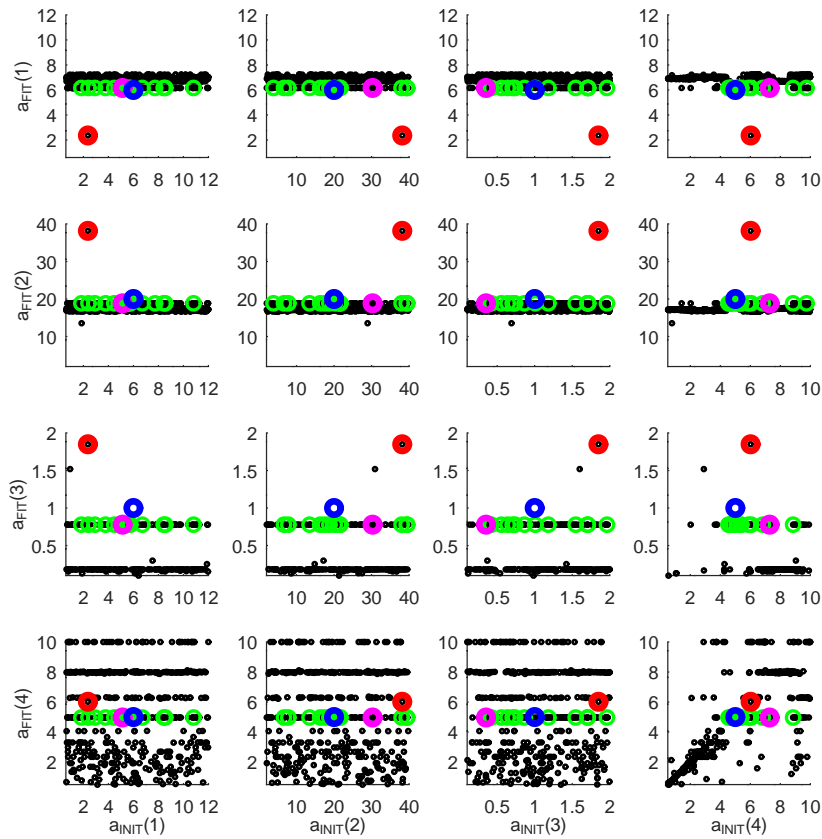


Figure 8. Example 3. A sample of 500 random uniformly distributed initial guesses and converged (fit) solutions. black: initial guesses and converged (fit) solutions; blue: true coefficient values; red: the worst initial guess and the associated converged (fit) coefficient values. green: the fifteen best initial guesses and the associated converged (fit) coefficient values. magenta: the best initial guess and the associated converged (fit) coefficient values.

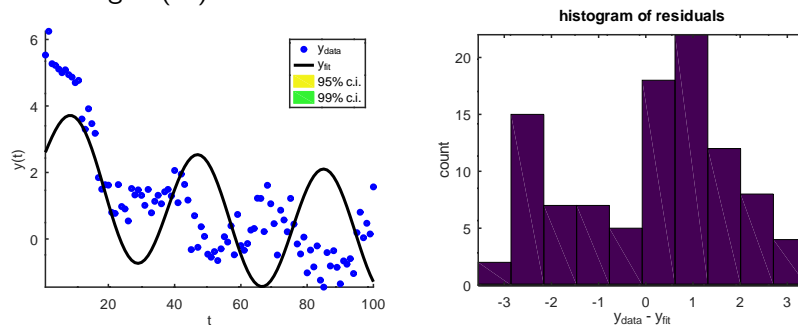


Figure 9. left: data  $y$ , curve-fit  $\hat{y}(t; \mathbf{c}_{fit})$ , curve-fit confidence intervals for the fit with the worst  $\chi^2_D$  value - using fit coefficients shown in red in Figure 8. right: Histogram of the errors between the data and the fit. The fit is astonishingly poor and the residuals are clearly not normally distributed.

## 6.5 Fitting in Multiple Dimensions

The code can carry out fitting in multiple dimensions. For example, the function

$$\hat{z}(x, y) = (c_1 x^{c_2} + (1 - a_1) y^{c_2})^{1/a_2}$$

may be fit to data points  $z_i(x_i, y_i)$ , ( $i = 1, \dots, m$ ), using `levenberg_marquardt` with a `.py`-file such as

```

1 x_dat = my_data[:,1]           # if the independent variable x is in column 1
2 y_dat = my_data[:,2]           # if the independent variable y is in column 2
3 z_dat = my_data[:,3]           # if the dependent variable z is in column 3
4
5 c_lb  = [ 0.1 ; 0.1 ]           # lower bound of coefficient values
6 c_ub  = [ 0.9 ; 2.0 ]           # upper bound of coefficient values
7 c_init = [ 0.5 ; 1.0 ]          # initial guess for coefficient values
8
9 t = [ x_dat , y_dat ]           # x and y are column vectors of independent variables
10
11 result = levenberg_marquardt(lm_func2d, c_init, t, z_dat, weight, 0.01, c_lb, c_ub)

```

and with the `.py`-function `lm_func2d.py`

```

1 def z_hat = lm_func2d(t,a):
2 # example function used for nonlinear least squares curve-fitting
3 # to demonstrate the Levenberg-Marquardt function, levenberg\marquardt,
4 # in two fitting dimensions
5
6     x_dat = t[:,1]
7     y_dat = t[:,2]
8     z_hat = ( a[1]*x_dat**a[2] + (1-a[1])*y_dat**a[2] )**(1/a[2])
9     return z_hat

```

## 7 Remarks

This manuscript and the code `levenberg_marquardt` were written in an attempt to understand and explain methods of nonlinear least squares for curve-fitting applications.

Least squares methods of curve-fitting are useful in computing the values *and* the standard errors of coefficient estimates. Nonlinear least squares problems iterate from an initial coefficient values, provided by a user based on their experience or possibly calculated from non-statistical minimization algorithms, such as random search methods, the Nelder-Mead simplex method, or coarsely gridding the coefficient space and finding the best combination of coefficient values.

Least squares problems can be nonlinear and convex (examples 1), linear and convex (example 2) or non-convex (example 3) in which the  $\chi^2$  objective function has multiple local minima and in which fitting algorithms can converge to different local minima depending upon the initial coefficient values, the measurement noise, and algorithmic parameters (hyper-parameters). It is always perfectly appropriate and good to set the initial guess to the best available coefficient estimates. In the absence of physical insight into a curve-fitting problem, a reasonable initial guess may be found by coarsely gridding the coefficient space and finding the best combination of coefficient values. There is no sense in forcing any iterative curve-fitting algorithm to work too hard by initializing it with a random or otherwise intentionally poor initial guess. In most applications of Levenberg Marquardt, coefficients identified from neighboring initial guesses ( $\pm 5\%$ ) should converge to similar coefficient estimates ( $\pm 0.1\%$ ). Further, the fit statistics of these converged coefficients should be the same within two or three significant figures.

The implementation of Levenberg-Marquardt described here offers the user options to customize the method to their application. The starting value of the damping coefficient,  $\lambda_0$ , can optionally be specified in `opts(7)` of `levenberg_marquardt`. The updating method for  $\lambda$  and  $\mathbf{h}$  can optionally be specified in `opts(8)`, `opts(9)` and `opts(10)`, as described in section 4.1.1 above. In the default update option (1),  $\lambda$  is scaled by the diagonal of the Hessian of  $\chi^2$ , effectively providing a different value of  $\lambda$  for each coefficient. This is helpful for problems involving broadly-ranging coefficient values. In update options (2) and (3),  $\lambda$  and  $\mathbf{h}$  are scaled by scalar maximum of the diagonal of the Hessian of  $\chi^2$ . The three examples in this manuscript are run with the default case. In example 1  $\lambda$  increases in the first two steps (Figure 1), but then decreases. In example 2, a *linear* least squares problem,  $\lambda$  decreases exponentially from its initial value of 0.01 (Figure 2). And in example 3, a non-convex problem,  $\lambda$  jumps around, and increases exponentially from  $10^{-6}$  to  $10^{-1}$  from function evaluation #22 to #28. The “best” initialization and updating method is problem-dependent. Only if convergence rates or converged results are somehow inadequate would it be worth trying different values of  $\lambda_0$  or different updating methods.

## References

- [1] Y. Bard, *Nonlinear Parameter Estimation*, Academic Press, 1974.
- [2] A. Björck. *Numerical methods for least squares problems*, SIAM, Philadelphia, 1996.
- [3] C.G. Broyden, “A class of methods for solving nonlinear simultaneous equations,” *Mathematical Computing*, 19:577-593 (1965).
- [4] C.G. Broyden, “On the discovery of the “good Broyden” method,” *Mathematical Programming*, Ser. B, 87:209-213 (2000).
- [5] N.R. Draper and H. Smith, *Applied Regression Analysis*, John Wiley and Sons, 1981.
- [6] Carsten Grammes. “fit.c” Gnuplot source code. <http://www.gnuplot.info>
- [7] K. Levenberg. “A Method for the Solution of Certain Non-Linear Problems in Least Squares”. *The Quarterly of Applied Mathematics*, 2: 164-168 (1944).
- [8] M.I.A. Lourakis. *A brief description of the Levenberg-Marquardt algorithm implemented by levmar*, Technical Report, Institute of Computer Science, Foundation for Research and Technology - Hellas, 2005.
- [9] K. Madsen, N.B. Nielsen, and O. Tingleff. *Methods for nonlinear least squares problems*. Technical Report. Informatics and Mathematical Modeling, Technical University of Denmark, 2004.
- [10] D.W. Marquardt. “An algorithm for least-squares estimation of nonlinear parameters,” *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431-441, 1963.
- [11] H.B. Nielson, *Damping Parameter In Marquardt’s Method*, Technical Report IMM-REP-1999-05, Dept. of Mathematical Modeling, Technical University Denmark.
- [12] W.H. Press, S.A. Teukosky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C*, Cambridge University Press, second edition, 1992.
- [13] Richard Shragar, Arthur Jutan, Ray Muzic, and Olaf Till, “`leasqr.m`” 1992-2016 *Octave-Forge*, A collection of packages providing extra functionality for GNU Octave
- [14] Mark K. Transtrum, Benjamin B. Machta, and James P. Sethna, “Why are nonlinear fits to data so challenging?”, *Phys. Rev. Lett.* 104, 060201 (2010),
- [15] Mark K. Transtrum and James P. Sethna “Improvements to the Levenberg-Marquardt algorithm for nonlinear least-squares minimization,” Preprint submitted to *Journal of Computational Physics*, January 30, 2012.