Lecture 5: Regularisation

Iain Styles

25 October 2019

Regularisation

Our approach to learning a function from data has been based around trying to minimise the loss, or error, with which the model is able to describe the data. As we have seen, it is inevitable that 必然的 a model of sufficient complexity will always be able to perfectly match the data points supplied. This seems like it should be highly desirable, but the consequence is that the model does not generalise to data that was not in the training set used to learn the unknown parameters: the model does not learn the underlying trend in noisy data, but rather uses high-order terms in the basis set to fit both the trend of the data and the noise.

We have recently formulated the regression problem from a probabilistic viewpoint which naturally allows us, via Bayes rule, to include information about our prior beliefs. We showed how likelihood maximisation under the prior belief that all values of the model parameters were equally likely led directly to the least-squares approach that we have previously studied. We also showed how a prior belief that the parameters are normally distributed led to a modified least squares loss that included a term that penalised large values of the model parameters – so-called *regularisation*. We will now study regularised loss functions and their solution and application in more detail.

In preceding sections, we formulated regularisation via a Bayesian prior. This is theoretically very satisfying but perhaps lacks flexibility: not all constraints that we may wish to impose can be mapped so easily onto a statistical prior. In this section we will draw on the final result of that analysis – a modified least-squares loss function and consider more general regularisation terms that are not so easily expressed in the probabilistic sense.

In general, the loss function for regularised problems is written

$$\mathcal{L}(\mathbf{w}) = \mathcal{L}_{\text{err}}(\mathbf{w}) + \lambda R(\mathbf{w}) \tag{1}$$

where $\mathcal{L}_{\text{err}}$ is a measure of the mismatch between the model prediction and the data (such as the least-squares error), and $R(\mathbf{w})$ is a function of the parameter vector that imposes some penalty on solutions that do not match our prior belief of what "valid" solutions should look like, by increasing the value of the loss function for these undesireable solutions. The choice of regularisation function R depends on what type of behaviour we want to impose.

Let us think about how this might be used to prevent overfitting. We saw previously that one of the characteristics of overfitted solutions is the presence of very large coefficients that enable higher order terms to fit the noise in the data. We might therefore propose

that somehow preventing the size of the coefficients from getting too large would prevent this. We previously saw that a Gaussian prior to concentrate the model weight near to zero results in an L_2 penalty with

$$R(\mathbf{w}) = \sum_i w_i^2 = \mathbf{w}^\mathrm{T}\mathbf{w}. \tag{2}$$

With this choice, the loss function, with least-squares error, is written

$$\mathcal{L}(\mathbf{w}) = (\mathbf{y} - \mathbf{\Phi}\mathbf{w})^\mathrm{T} (\mathbf{y} - \mathbf{\Phi}\mathbf{w}) + \lambda \mathbf{w}^\mathrm{T}\mathbf{w} \tag{3}$$

To minimise this requires that both the error term and the magnitude of the weights are simultaneously minimised, with the free parameter λ controlling the relative extent to which each of these terms contributes to the overall loss. Small values of λ mean that reducing the error is preferable to reducing the weight magnitude, while large values of λ favour reducing the weights over reducing the error.

This is a relatively friendly regulariser with a closed-form analytical solution. We can proceed with the analysis as we did previously: we differentiate with respect to the weights and set this to zero to find the minimum, noting that the new term is bounded from below by zero, and has no upper bound:

$$\frac{\partial \mathcal{L}_{\mathrm{LSE}}(\mathbf{w})}{\partial \mathbf{w}} = -2\mathbf{\Phi}^\mathrm{T} (\mathbf{y} - \mathbf{\Phi}\mathbf{w}) + 2\lambda \mathbf{w}, \tag{4}$$

We set this to zero to obtain

$$\mathbf{\Phi}^\mathrm{T}\mathbf{y} - \left(\mathbf{\Phi}^\mathrm{T}\mathbf{\Phi} - \lambda \mathbf{I}\right) \mathbf{w}^* = 0 \tag{5}$$

where we have employed the identity matrix $\mathbf{I}$ to enable the expression to be factorised in this form. This method is referred to in the literature by a number of different names:

- *Ridge regression*.
- *L_2 regularisation*, because $\sum_i w_i^2$ is the L_2 norm of $\mathbf{w}$, written as $||\mathbf{w}||_2^2$.
- *Weight decay*, because it pushes weights towards zero.
- *Tikhonov regularisation*, of which it is a special case. Tikhonov regularisation uses $R(\mathbf{p}) = ||\mathbf{\Gamma}\mathbf{w}||$. Here, we have $\mathbf{\Gamma} = \lambda \mathbf{I}$.

Let us investigate how this works in practice. In Figure 1 we show the effect of different values of λ on the learning of the model. The fluctuations in the model curve are "smoothed" out by the regularisation term. This comes at the cost of a higher error (Figure 2), because the fit no longer exactly matches the data, but it does model the trend in the data more effectively.

It's also useful to look at the learned weights, shown in Table 1. We can clearly see the "shrinkage" effect, with the larger values of λ giving rise to much lower values of the parameters.

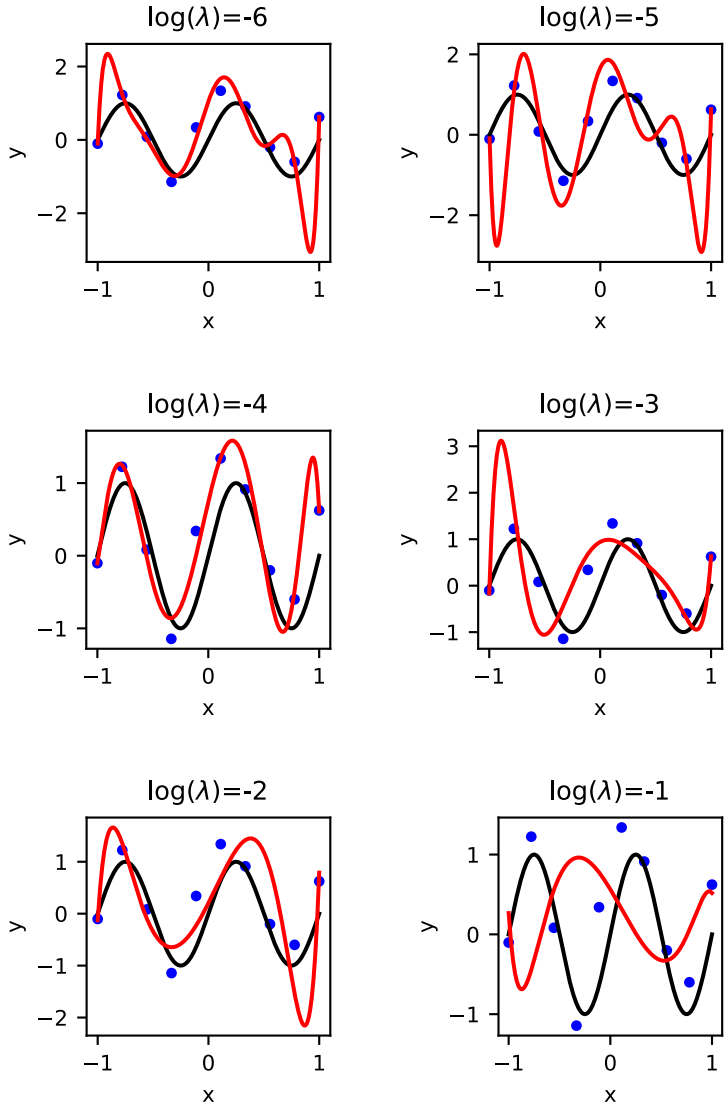

Figure 1: Fitting $y = \sin(2\pi x)$ with polynomial fits of degree $M = 9$ with L_2 regularisation using different regularisation coefficients λ. The blue line represents the true function; the black points are the sampled points (x_i, y_i); the red line is the best-fit polynomial of that order.

$\log_{10} \lambda$	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
-6	1.06	8.31	-17.92	-76.20	76.16	250.58	-112.92	-350.94	53.88	168.61
-5	1.67	5.77	-41.44	-29.12	212.84	31.81	-356.24	1.30	183.44	-9.40
-4	0.74	6.51	-5.21	-33.75	1.46	33.96	20.81	13.89	-17.55	-20.25
-3	0.94	1.29	-9.30	5.42	15.29	-21.02	5.50	-4.36	-12.20	19.08
-2	0.20	4.26	2.26	-10.00	-5.63	-4.88	-0.83	2.60	4.33	8.47
-1	0.56	-2.12	-1.35	4.00	-0.63	1.44	0.50	-0.84	1.30	-2.35

Table 1: Model weights for different value of λ with L_2 regularisation. The shrinkage effect of this regularisation term is clearly seen.

Other common regularisation functions

L_2 regularisation is extremely common and many machine learning libraries use it by default (this is something to be aware of when working with external libraries), but it belongs to a much broader class of regulariser known as the L_p norms, which use a regularisation term of the form

$$R(\mathbf{w}) = \sum_i |w_i|^p \tag{6}$$

Some common choices for the value of p include:

- $p = 2$, the method we have already studied.
- $p = 1$: commonly know as the *lasso* method. When λ is large, this term tends to force some of the coefficients to zero. The contours of constant L_1 norm form a "diamond" in parameter space (Bishop, Figure 3.3, P145), and the intersection of this diamond with the circular contours of the least squares loss a most likely to be at the apices of the diamond which lie on the coordinate axes, where some of the coefficients are zero (Bishop, Figure 3.4, P146).
- $p < 1$: this term also forces some of the model coefficients to zero, but much more aggresively that $L = 1$ (Bishop, Figure 3.3, P145).

The problem with $p \neq 2$ is that the normal equations do not have a nice closed-form analytical solution. For $p = 1$, a number of numerical approaches have been developed, include FISTA – Fast Iterative Shrinkage Thresholding Algorithm (see reading). Other methods are even harder to solve numerically. Nonetheless, methods that use $p \leq 1$ have become very popular in recent year because of their sparsity-inducing properties in applications such as image denoising and reconstruction, and because of the popularity of "compressive sensing" methods that rely on finding sparse solutions to complex inverse problems.

Because there is no closed form solution, we turn to `scikit-learn` for an implementation of the lasso method. In Figure 3 we show the results of different L_1-regularised fits, with the weights shown in Table 2. The sparsifying effect of the regulariser is clearly evident from the table.

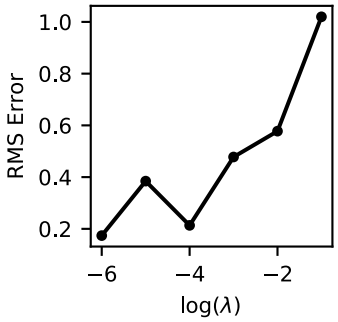

Figure 2: RMS Training Error of polynomial fit with $M = 9$ to $y = \sin(2\pi x)$ for different degrees of L_2 regularisation.

$\log_{10}\lambda$	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
-5	0.00	3.59	-0.00	-15.72	0.00	11.78	0.00	1.96	0.00	-1.60
-4	0.00	3.54	0.00	-15.27	0.00	11.05	0.00	1.93	0.00	-1.24
-3	0.00	3.10	-0.00	-12.02	-0.00	5.44	-0.00	3.49	-0.00	0.00
-2	0.00	0.95	-0.00	-3.76	-0.00	-0.00	-0.00	0.00	-0.00	2.74
-1	0.00	-0.05	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
0	0.00	-0.00	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00

Table 2: Model weights for different value of λ with L_1 regularisation. The sparsifying effect of this regularisation term is clearly seen.

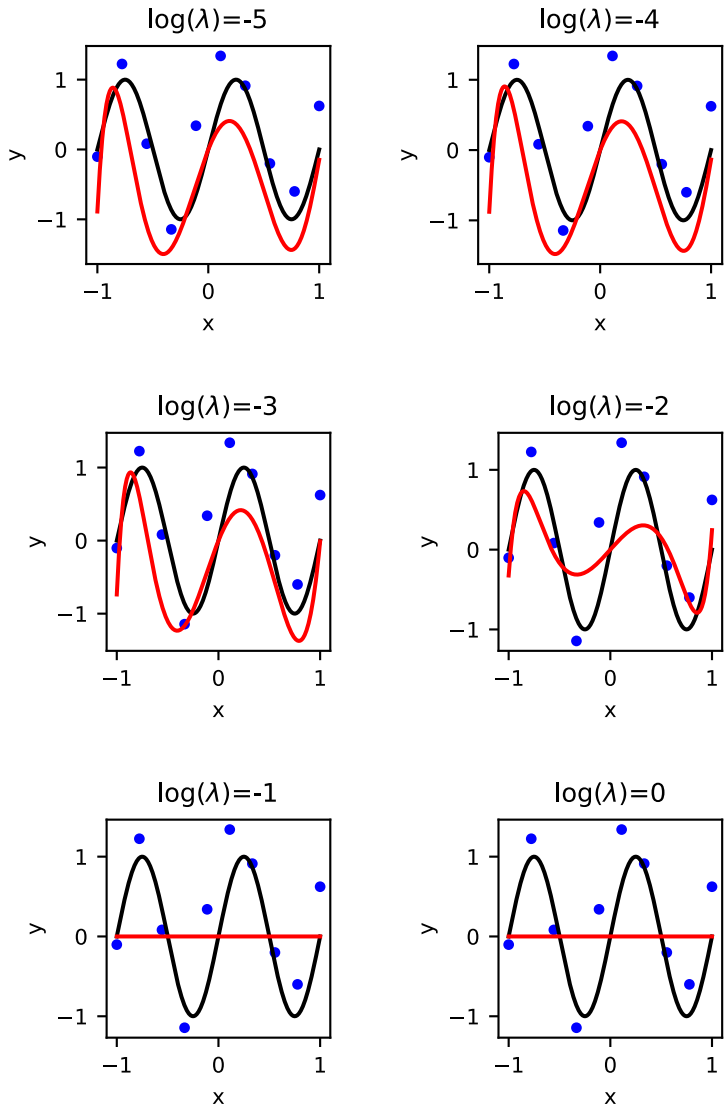

Figure 3: Fitting $y = \sin(2\pi x)$ with polynomial fits of degree $M = 9$ with L_1 regularisation using different regularisation coefficients λ. The blue line represents the true function; the black points are the sampled points (x_i, y_i); the red line is the best-fit polynomial of that order.

The value of the regularisation weight λ is a hyperparameter of the algorithm that needs to be set. It is normal practice to treat this in the same way as we treated the polynomial order: we use cross-validation to select the optimal value. The advantage of regularisation over explicit model selection is that the optimisation process can "choose" how to weight the higher order terms. For example, if we use a sparsifying regulariser such as the L_1 norm on data drawn from an underlying quadratic but with a fitting function of order ten, the sparsifying effect can set the coefficients of the high order terms to zero. This can be a very powerful way of controlling model selection.

Multivariate Regression

We have so far considered problems consisting of a single independent variable x and a single dependency variable $\mathbf{y}$. The arguments we have developed can be easily extended to allow us to deal with problems with multiple dependent and independent variables. First, let us consider the case of multiple independent variables. Here, we will have to be a bit careful with our notation because things can easily become complicated. We will denote our independent variables as x_i. The value of the variable at a particular data points will be denoted $x_{i,j}$ (for the j'th data point.). In these terms, our dataset is written as

$$\mathcal{D} = \{(x_{1,i}, x_{2,i}, \ldots, x_{K,i}, y_i)\}_{i=1}^{N} \tag{7}$$

where we have K independent variables. Performing a regression over this dataset can be done in exactly the same way as we did for the case of a single independent variable – we choose a basis, and them solve the normal equations – but with one important difference: our basis matrix $\mathbf{\Phi}$ now needs to contain terms from all variables. To illustrate this, consider the simplest possible case of two independent variables, and a model that contains a constant term and linear terms in the two variables:

$$y = w_0 + w_1 x_1 + w_2 x_2 \tag{8}$$

which has basis matrix

$$\mathbf{\Phi} = \begin{pmatrix} 1 & x_{1,1} & x_{2,1} \\ 1 & x_{1,2} & x_{2,2} \\ \vdots & \vdots & \vdots \\ 1 & x_{1,N} & x_{2,N} \end{pmatrix} \tag{9}$$

A more complex quadratic model

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2 \tag{10}$$

has basis matrix

$$\mathbf{\Phi} = \begin{pmatrix} 1 & x_{1,1} & x_{2,1} & x_{1,1}^2 & x_{2,1}^2 & x_{1,1}x_{2,1} \\ 1 & x_{1,2} & x_{2,2} & x_{1,2}^2 & x_{2,2}^2 & x_{1,2}x_{2,2} \\ \vdots & \vdots & \vdots & & & \\ 1 & x_{1,N} & x_{2,N} & x_{1,N}^2 & x_{2,N}^2 & x_{1,N}x_{2,N} \end{pmatrix} \tag{11}$$

It is straightforward to generalise this to other models.

Now let us consider the case of multiple (L) dependent variables y_i (using the same notation as we did for the dependent variables). There are two ways in which we could extend our analysis. The first is to treat each dependent variable separately and to solve multiple regression problems independently. Assuming that we use the same choice of basis $\mathbf{\Phi}$ for each dependent variable we have

$$y_1(x_1, \ldots, x_K) = \sum_j w_{1,j}\phi_j(x_1, \ldots, x_K) \quad \rightarrow \quad \mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{w}_1 = \mathbf{\Phi}^{\mathrm{T}}\mathbf{y}_1 \tag{12}$$

$$y_2(x_1, \ldots, x_K) = \sum_j w_{2,j}\phi_j(x_1, \ldots, x_K) \quad \rightarrow \quad \mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{w}_2 = \mathbf{\Phi}^{\mathrm{T}}\mathbf{y}_2 \tag{13}$$

$$\vdots \qquad \vdots \tag{14}$$

$$y_L(x_1, \ldots, x_K) = \sum_j w_{L,j}\phi_j(x_1, \ldots, x_K) \quad \rightarrow \quad \mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{w}_L = \mathbf{\Phi}^{\mathrm{T}}\mathbf{y}_L \tag{15}$$

$$\tag{16}$$

where $\mathbf{y}_j = \left[y_{j,1}, y_{j,2}, \ldots, y_{j,N}\right]^{\mathrm{T}}$ and $\mathbf{w}_i = [w_{i,1}, w_{i,2}, \ldots w_{i,M}]$.

Alternatively, we could perform a "joint" optimisation over all of the dependent variables in a single step. Combining the equations for each dependent variable we arrive at the following relationship

$$\begin{pmatrix} y_{1,1} & y_{2,1} & \ldots & y_{K,1} \\ y_{1,2} & y_{2,2} & \ldots & y_{K,2} \\ \vdots & \vdots & \ddots & \vdots \\ y_{1,N} & y_{2,N} & \ldots & y_{K,N} \end{pmatrix} = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \ldots & \phi_M(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \ldots & \phi_M(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \ldots & \phi_M(x_N) \end{pmatrix} \begin{pmatrix} w_{1,1} & w_{2,1} & \ldots & w_{K,1} \\ w_{1,2} & w_{2,2} & \ldots & w_{K,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,M} & w_{2,M} & \ldots & w_{K,M} \end{pmatrix}$$

$$\rightarrow \mathbf{Y} = \mathbf{\Phi}\mathbf{W}$$

where each column of $\mathbf{Y}$ corresponds to a dependent variable, and the corresponding column of $\mathbf{W}$ holds the weights for that variable. In these terms, the (unregularised) optimisation can be performed jointly over all variables by solving the normal equations

$$\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{W} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{Y}. \tag{17}$$

This simultaneously minimises the least-squared loss over all dependent variables. Of course, it also requires that all dependent variables can be modelled in terms of the same basis functions. If this is not the case then the problem should be solved as a series of independent regression tasks.

Non-linear Regression

This section is provided for information only and is not examinable content.

The analysis to date is restricted only to models that can be expressed in a linear form, but there may be many cases where this is not possible (eg. $y(x) = a \sin bx$, where we need to find a and b). We will still aim to minimise the sum of the squares of the residuals, but we cannot use the linearity of the fitting function to help us formulate the problem in a tractable way that permits a single-step solution. Our approach is to break the problem down into a series of linear approximations, each of which *reduces* the loss rather than absolutely minimising it.

We must formulate a process that rather than solving the normal equations globally to find the value of $\mathbf{w}$ that minimises the error, instead finds a set of parameters that simply reduces the error. We will do this iteratively until no further reduction in error is possible.

We will proceed by examining the behaviour of the error when we change $\mathbf{w}$ by a small amount. This we do by approximating the function as a straight line over a small region using *Taylor's Theorem*:

$$f(x_i, \mathbf{w} + \boldsymbol{\Delta}\mathbf{w}) \approx f(x_i, \mathbf{w}) + \sum_{j=1}^{m} \frac{\partial f(x_i)}{\partial w_j} \Delta w_j \tag{18}$$

$$= f(x_i, \mathbf{w}) + \sum_{j=1}^{m} J_{ij} \delta w_j \tag{19}$$

where matrix $\mathbf{J}$ has components $J_{ij} = \frac{\partial f(x_i)}{\partial w_j}$ and the residual after the change is

$$\mathbf{r}(\mathbf{w} + \boldsymbol{\Delta}\mathbf{w}) = \mathbf{y} - \mathbf{f}(\mathbf{w}) - \mathbf{J}\boldsymbol{\Delta}\mathbf{w} \tag{20}$$

where $f_i(\mathbf{w}) = f(x_i, \mathbf{w})$. The error is therefore

$$e(\mathbf{w} + \boldsymbol{\Delta}\mathbf{w}) = (\mathbf{y} - \mathbf{f}(\mathbf{w}) - \mathbf{J}\boldsymbol{\Delta}\mathbf{w})^{\mathrm{T}} (\mathbf{y} - \mathbf{f}(\mathbf{w}) - \mathbf{J}\boldsymbol{\Delta}\mathbf{w}) . \tag{21}$$

By minimising this quantity, we find the *change* in $\mathbf{w}$ that gives us the largest reduction in the error. We differentiate as before and set to zero:

$$\left(\mathbf{J}^{\mathrm{T}}\mathbf{J}\right) \boldsymbol{\Delta}\mathbf{w}^{*} = \mathbf{J}^{\mathrm{T}} (\mathbf{y} - \mathbf{f}(\mathbf{w})) \tag{22}$$

which we know how to solve.

It is common practice to make a small modification to this:

$$\left(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \lambda \operatorname{diag}(\mathbf{J}^{\mathrm{T}}\mathbf{J})\right) \boldsymbol{\Delta}\mathbf{w}^{*} = \mathbf{J}^{\mathrm{T}} (\mathbf{y} - \mathbf{f}(\mathbf{w})) \tag{23}$$

This is a regularisation that penalises large updates to the parameter vector along directions with steep gradients, encouraging movement along shallow gradients and thus a faster convergence. The amount of increase is controlled by the λ which is modified iteratively to give the best results. The full algorithm, named after its inventors Levenberg and Marquadt is described in Algorithm 1. This method is widely used for nonlinear regression problems, but should be used cautiously because for complex models, there is the possibility that the loss function will have multiple local minima, and because this is a gradient-based method, it is prone to getting stuck in these local minima. In this circumstance, other optimisation approaches that are not based on gradient descent should be used, for examples, evolutionary/genetic algorithms.

Data: Data $\mathbf{x}, \mathbf{y}$; Initial parameter guess $\mathbf{w}_0$; Function $f(x, \mathbf{w})$; Jacobian $Jac(x, \mathbf{w})$; initial values of λ, ν.
Result: $\mathbf{w}, e(\mathbf{w})$, parameters that minimmise the least squares error, the error.

```
% Evaluate the error at the initial parameter value
```
$\mathbf{w} \leftarrow \mathbf{w}_0$;
$e \leftarrow (\mathbf{y} - f(x, \mathbf{w})^{\mathrm{T}}(\mathbf{y} - f(x, \mathbf{w})$;
```
% Iterate until convergence
```
Converged $\leftarrow$ *False*;
while *Converged* $==$ *False* **do**
```
% Calculate the Jacobian
```
$\mathbf{J} \leftarrow \frac{\partial \mathbf{f}}{\mathbf{w}}$;
```
% Calculate update for different values of λ until
  we improve the error
```
LambdaSet $\leftarrow$ *false*;
while *LambdaSet*==*false* **do**
```
% Calculate Δw for λ = λ_0 and λ = λ_0/ν
```
$\Delta_1 = \left(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \lambda \operatorname{diag}(\mathbf{J}^{\mathrm{T}}\mathbf{J})\right) \setminus \mathbf{J}^{\mathrm{T}}\left(\mathbf{y} - \mathbf{f}(x, \mathbf{w})\right)$;
$\Delta_2 = \left(\mathbf{J}^{\mathrm{T}}\mathbf{J} + (\lambda/\nu) \operatorname{diag}(\mathbf{J}^{\mathrm{T}}\mathbf{J})\right) \setminus \mathbf{J}^{\mathrm{T}}\left(\mathbf{y} - \mathbf{f}(x, \mathbf{w})\right)$;
```
% Calculate the error at the new parameter values
```
$e_1 \leftarrow (\mathbf{y} - f(x, \mathbf{w} + \Delta_1)^{\mathrm{T}}(\mathbf{y} - f(x, \mathbf{w})$;
$e_2 \leftarrow (\mathbf{y} - f(x, \mathbf{w} + \Delta_2)^{\mathrm{T}}(\mathbf{y} - f(x, \mathbf{w})$;
if $e_1 < e$ **then**
```
% Set new values of w and e, storing old value
  as e'
```
$\mathbf{w} \leftarrow \mathbf{w} + \Delta_1$;
$e' \leftarrow e$;
$e \leftarrow e_1$;
```
% Current λ is OK
```
LambdaSet $\leftarrow$ *true*;
else if $e_2 < e$ **then**
```
% Set new values of w and e, storing old value
  as e'
```
$\mathbf{w} \leftarrow \mathbf{w} + \Delta_2$;
$e' \leftarrow e$;
$e \leftarrow e_2$;
$\lambda \leftarrow \lambda/\nu$;
```
% Current λ is OK
```
LambdaSet $\leftarrow$ *true*;
else
  $\lambda \leftarrow \lambda \times \nu$;
end
if $e'/e <$ *TerminationCriterion* **then**
  Converged $\leftarrow$ *true*
end
end

Algorithm 1: The Levenberg-Marquadt Algorithm.

Reading

- Section 3.1 of Bishop, Pattern Recognition and Machine Learning.
- Amir Beck and Marc Teboulle. A FastIterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. SIAM Journal of Imaging Sciences. Vol. 2, No. 1, pp. 183âĂŞ202 (2009). https://people.rennes.inria.fr/Cedric.Herzet/Cedric.Herzet/Sparse_Seminar/Entrees/2012/11/12_A_Fast_Iterative_Shrinkage-Thresholding_Algorithmfor_Linear_Inverse_Problems_(A._Beck,_M._Teboulle)_files/Breck_2009.pdf
- Richard Baraniuk. *Compressive Sensing*. IEEE signal processing magazine 24, no. 4 (2007). https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4286571