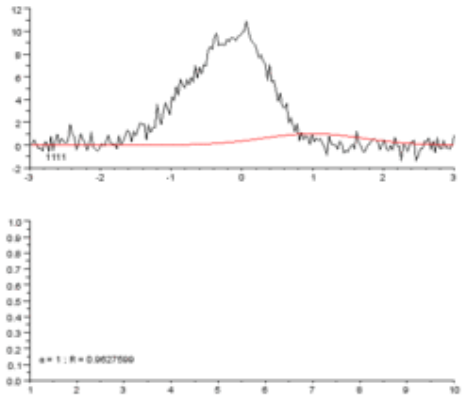
Gauss–Newton algorithm

The **Gauss–Newton algorithm** is used to solve non-linear least squares problems. It is a modification of Newton's method for finding a minimum of a function. Unlike Newton's method, the Gauss–Newton algorithm can only be used to minimize a sum of squared function values, but it has the advantage that second derivatives, which can be challenging to compute, are not required.

Non-linear least squares problems arise, for instance, in non-linear regression, where parameters in a model are sought such that the model is in good agreement with available observations.



Fitting of a noisy curve by an asymmetrical peak model, using the Gauss–Newton algorithm with variable damping factor α.
Top: raw data and model.
Bottom: evolution of the normalised sum of the squares of the errors.

The method is named after the mathematicians Carl Friedrich Gauss and Isaac Newton.

Description

Given m functions $\mathbf{r} = (r_1, \ldots, r_m)$ (often called residuals) of n variables $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)$, with $m \geq n$, the Gauss–Newton algorithm iteratively finds the value of the variables that minimizes the sum of squares[1]

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{m} r_i^2(\boldsymbol{\beta}).$$

Starting with an initial guess $\boldsymbol{\beta}^{(0)}$ for the minimum, the method proceeds by the iterations

$$\boldsymbol{\beta}^{(s+1)} = \boldsymbol{\beta}^{(s)} - \left(\mathbf{J_r}^{\mathsf{T}}\mathbf{J_r}\right)^{-1}\mathbf{J_r}^{\mathsf{T}}\mathbf{r}(\boldsymbol{\beta}^{(s)}),$$

where, if $\mathbf{r}$ and $\boldsymbol{\beta}$ are column vectors, the entries of the Jacobian matrix are

$$(\mathbf{J_r})_{ij} = \frac{\partial r_i(\boldsymbol{\beta}^{(s)})}{\partial \beta_j},$$

and the symbol $^{\mathsf{T}}$ denotes the matrix transpose.

If $m = n$, the iteration simplifies to

$$\boldsymbol{\beta}^{(s+1)} = \boldsymbol{\beta}^{(s)} - \left(\mathbf{J_r}\right)^{-1} \mathbf{r}\left(\boldsymbol{\beta}^{(s)}\right),$$

which is a direct generalization of Newton's method in one dimension.

In data fitting, where the goal is to find the parameters $\boldsymbol{\beta}$ such that a given model function $y = f(x, \boldsymbol{\beta})$ best fits some data points (x_i, y_i), the functions r_i are the residuals:

$$r_i(\boldsymbol{\beta}) = y_i - f(x_i, \boldsymbol{\beta}).$$

Then, the Gauss–Newton method can be expressed in terms of the Jacobian $\mathbf{J_f}$ of the function f as

$$\boldsymbol{\beta}^{(s+1)} = \boldsymbol{\beta}^{(s)} + \left(\mathbf{J_f}^{\mathsf{T}}\mathbf{J_f}\right)^{-1}\mathbf{J_f}^{\mathsf{T}}\mathbf{r}\left(\boldsymbol{\beta}^{(s)}\right).$$

Note that $\left(\mathbf{J_f}^{\mathsf{T}}\mathbf{J_f}\right)^{-1}\mathbf{J_f}^{\mathsf{T}}$

is the pseudoinverse of $\mathbf{J_f}$.

Notes

The assumption $m \geq n$ in the algorithm statement is necessary, as otherwise the matrix $\mathbf{J_r}^{\mathsf{T}}\mathbf{J_r}$ is not invertible and the normal equations cannot be solved (at least uniquely).

The Gauss–Newton algorithm can be derived by linearly approximating the vector of functions r_i. Using Taylor's theorem, we can write at every iteration:

$$\mathbf{r}(\boldsymbol{\beta}) \approx \mathbf{r}\left(\boldsymbol{\beta}^{(s)}\right) + \mathbf{J_r}\left(\boldsymbol{\beta}^{(s)}\right)\Delta$$

with $\Delta = \boldsymbol{\beta} - \boldsymbol{\beta}^{(s)}$

. The task of finding Δ minimizing the sum of squares of the right-hand side, i.e.,

$$\min \left\| \mathbf{r}\left(\boldsymbol{\beta}^{(s)}\right) + \mathbf{J_r}\left(\boldsymbol{\beta}^{(s)}\right)\Delta \right\|_2^2,$$

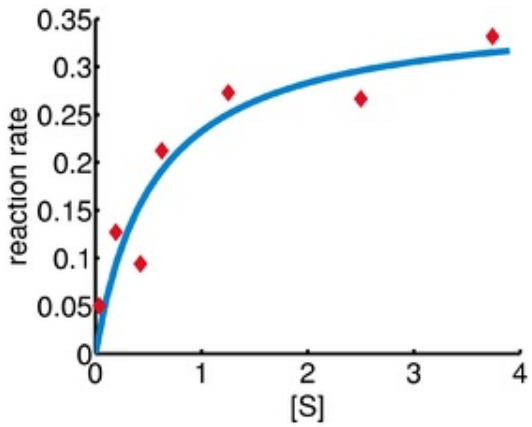
is a linear least-squares problem, which can be solved explicitly, yielding the normal equations in the algorithm.

The normal equations are *n* simultaneous linear equations in the unknown increments Δ. They may be solved in one step, using Cholesky decomposition, or, better, the QR factorization of $\mathbf{J_r}$. For large systems, an iterative method, such as the conjugate gradient method, may be more efficient. If there is a linear dependence between columns of $\mathbf{J_r}$, the iterations will fail, as $\mathbf{J_r}^{\mathrm{T}}\mathbf{J_r}$ becomes singular.

Example

In this example, the Gauss–Newton algorithm will be used to fit a model to some data by minimizing the sum of squares of errors between the data and model's predictions.

In a biology experiment studying the relation between substrate concentration [*S*] and reaction rate in an enzyme-mediated reaction, the data in the following table were obtained.



Calculated curve obtained with β ^ 1 = 0.362 {\displaystyle {\hat {\beta }}_{1}=0.362} and β ^ 2 = 0.556 {\displaystyle {\hat {\beta }}_{2}=0.556} (in blue) versus the observed data (in red).

i	1	2	3	4	5	6	7
[*S*]	0.038	0.194	0.425	0.626	1.253	2.500	3.740
rate	0.050	0.127	0.094	0.2122	0.2729	0.2665	0.3317

It is desired to find a curve (model function) of the form

rate = V max [S] K M + [S] {\displaystyle {\text{rate}}={\frac {V_{\text{max}}[S]}{K_{M}+[S]}}}

that fits best the data in the least-squares sense, with the parameters V max {\displaystyle V_{\text{max}}} and K M {\displaystyle K_{M}} to be determined.

Denote by x i {\displaystyle x_{i}} and y i {\displaystyle y_{i}} the value of [*S*] and the rate from the table, i = 1 , … , 7 {\displaystyle i=1,\dots ,7} . Let β 1 = V max {\displaystyle \beta _{1}=V_{\text{max}}} and β 2 = K M {\displaystyle \beta _{2}=K_{M}} . We will find β 1 {\displaystyle \beta _{1}} and β 2 {\displaystyle \beta _{2}} such that the sum of squares of the residuals

r i = y i − β 1 x i β 2 + x i (i = 1 , … , 7) {\displaystyle r_{i}=y_{i}-{\frac {\beta _{1}x_{i}}{\beta _{2}+x_{i}}}\quad (i=1,\dots ,7)}

is minimized.

The Jacobian J r {\displaystyle \mathbf {J_{r}} } of the vector of residuals r i {\displaystyle r_{i}} with respect to the unknowns β j {\displaystyle \beta _{j}} is a 7 × 2 {\displaystyle 7\times 2}

matrix with the i {\displaystyle i} -th row having the entries

∂ r i ∂ β 1 = − x i β 2 + x i ; ∂ r i ∂ β 2 = β 1 x i (β 2 + x i) 2 . {\displaystyle {\frac {\partial r_{i}}{\partial \beta _{1}}}=-{\frac {x_{i}}{\beta _{2}+x_{i}}};{\frac {\partial r_{i}}{\partial \beta _{2}}}={\frac {\beta _{1}x_{i}}{(\beta _{2}+x_{i})^{2}}}.}

Starting with the initial estimates of β 1 = 0.9 {\displaystyle \beta _{1}=0.9} and β 2 = 0.2 {\displaystyle \beta _{2}=0.2} , after five iterations of the Gauss–Newton algorithm the optimal values β ^ 1 = 0.362 {\displaystyle {\hat {\beta }}_{1}=0.362} and β ^ 2 = 0.556 {\displaystyle {\hat {\beta }}_{2}=0.556}

are obtained. The sum of squares of residuals decreased from the initial value of 1.445 to 0.00784 after the fifth iteration. The plot in the figure on the right shows the curve determined by the model for the optimal parameters with the observed data.

Convergence properties

It can be shown[2] that the increment Δ is a descent direction for *S*, and, if the algorithm converges, then the limit is a stationary point of *S*. However, convergence is not guaranteed, not even local convergence as in Newton's method, or convergence under the usual Wolfe conditions.[3]

The rate of convergence of the Gauss–Newton algorithm can approach quadratic.[4] The algorithm may converge slowly or not at all if the initial guess is far from the minimum or the matrix J r T J r {\displaystyle \mathbf {J_{r}^{\mathsf {T}}J_{r}} } is ill-conditioned. For example, consider the problem with m = 2 {\displaystyle m=2} equations and n = 1 {\displaystyle n=1} variable, given by

r 1 (β) = β + 1 , r 2 (β) = λ β 2 + β − 1. {\displaystyle {\begin{aligned}r_{1}(\beta)&=\beta +1,\\r_{2}(\beta)&=\lambda \beta ^{2}+\beta -1.\end{aligned}}}

The optimum is at β = 0 {\displaystyle \beta =0} . (Actually the optimum is at β = − 1 {\displaystyle \beta =-1} for λ = 2 {\displaystyle \lambda =2} , because S (0) = 1 2 + (− 1) 2 = 2 {\displaystyle S(0)=1^{2}+(-1)^{2}=2} , but S (− 1) = 0 {\displaystyle S(-1)=0} .) If λ = 0 {\displaystyle \lambda =0} , then the problem is in fact linear and the method finds the optimum in one iteration. If |λ| < 1, then the method converges linearly and the error decreases asymptotically with a factor |λ| at every iteration. However, if |λ| > 1, then the method does not even converge locally.[5]

Derivation from Newton's method

In what follows, the Gauss–Newton algorithm will be derived from Newton's method for function optimization via an approximation. As a consequence, the rate of convergence of the Gauss–Newton algorithm can be quadratic under certain regularity conditions. In general (under weaker conditions), the convergence rate is linear.[6]

The recurrence relation for Newton's method for minimizing a function S of parameters $\boldsymbol{\beta}$ is

$$\boldsymbol{\beta}^{(s+1)}=\boldsymbol{\beta}^{(s)}-\mathbf{H}^{-1}\mathbf{g},$$

where **g** denotes the gradient vector of S, and **H** denotes the Hessian matrix of S. Since $S=\sum_{i=1}^{m}r_i^2$, the gradient is given by

$$g_j=2\sum_{i=1}^{m}r_i\frac{\partial r_i}{\partial \beta_j}.$$

Elements of the Hessian are calculated by differentiating the gradient elements, g_j, with respect to β_k:

$$H_{jk}=2\sum_{i=1}^{m}\left(\frac{\partial r_i}{\partial \beta_j}\frac{\partial r_i}{\partial \beta_k}+r_i\frac{\partial^2 r_i}{\partial \beta_j\partial \beta_k}\right).$$

The Gauss–Newton method is obtained by ignoring the second-order derivative terms (the second term in this expression). That is, the Hessian is approximated by

$$H_{jk}\approx 2\sum_{i=1}^{m}J_{ij}J_{ik},$$

where $J_{ij}=\frac{\partial r_i}{\partial \beta_j}$ are entries of the Jacobian $\mathbf{J_r}$. The gradient and the approximate Hessian can be written in matrix notation as

$$\mathbf{g}=2\mathbf{J}_{\mathbf{r}}^{\mathsf{T}}\mathbf{r},\quad \mathbf{H}\approx 2\mathbf{J}_{\mathbf{r}}^{\mathsf{T}}\mathbf{J_r}.$$

These expressions are substituted into the recurrence relation above to obtain the operational equations

$$\boldsymbol{\beta}^{(s+1)}=\boldsymbol{\beta}^{(s)}+\Delta;\quad \Delta=-\left(\mathbf{J_r}^{\mathsf{T}}\mathbf{J_r}\right)^{-1}\mathbf{J_r}^{\mathsf{T}}\mathbf{r}.$$

Convergence of the Gauss–Newton method is not guaranteed in all instances. The approximation

| r i ∂ 2 r i ∂ β j ∂ β k | ≪ | ∂ r i ∂ β j ∂ r i ∂ β k | {\displaystyle \left|r_{i}{\frac {\partial ^{2}r_{i}}{\partial \beta _{j}\partial \beta _{k}}}\right|\ll \left|{\frac {\partial r_{i}}{\partial \beta _{j}}}{\frac {\partial r_{i}}{\partial \beta _{k}}}\right|}

that needs to hold to be able to ignore the second-order derivative terms may be valid in two cases, for which convergence is to be expected:[7]

1. The function values r i {\displaystyle r_{i}} are small in magnitude, at least around the minimum.
2. The functions are only "mildly" nonlinear, so that ∂ 2 r i ∂ β j ∂ β k {\displaystyle {\frac {\partial ^{2}r_{i}}{\partial \beta _{j}\partial \beta _{k}}}} is relatively small in magnitude.

Improved versions

With the Gauss–Newton method the sum of squares of the residuals *S* may not decrease at every iteration. However, since Δ is a descent direction, unless S (β s) {\displaystyle S({\boldsymbol {\beta }}^{s})} is a stationary point, it holds that S (β s + α Δ) < S (β s) {\displaystyle S({\boldsymbol {\beta }}^{s}+\alpha \Delta)<S({\boldsymbol {\beta }}^{s})} for all sufficiently small α > 0 {\displaystyle \alpha >0} . Thus, if divergence occurs, one solution is to employ a fraction α {\displaystyle \alpha } of the increment vector Δ in the updating formula:

β s + 1 = β s + α Δ . {\displaystyle {\boldsymbol {\beta }}^{s+1}={\boldsymbol {\beta }}^{s}+\alpha \Delta .}

In other words, the increment vector is too long, but it points in "downhill", so going just a part of the way will decrease the objective function *S*. An optimal value for α {\displaystyle \alpha } can be found by using a line search algorithm, that is, the magnitude of α {\displaystyle \alpha } is determined by finding the value that minimizes *S*, usually using a direct search method in the interval 0 < α < 1 {\displaystyle 0<\alpha <1} .

In cases where the direction of the shift vector is such that the optimal fraction α is close to zero, an alternative method for handling divergence is the use of the Levenberg–Marquardt algorithm, also known as the "trust region method".[1] The normal equations are modified in such a way that the increment vector is rotated towards the direction of steepest descent,

(J T J + λ D) Δ = − J T r , {\displaystyle \left(\mathbf {J^{\mathrm {T} }J+\lambda D} \right)\Delta =-\mathbf {J} ^{\mathrm {T} }\mathbf {r} ,}

where **D** is a positive diagonal matrix. Note that when **D** is the identity matrix **I** and λ → + ∞ {\displaystyle \lambda \to +\infty } , then λ Δ = λ (J T J + λ I) − 1 (− J T r) = (I − J T J / λ + ⋯) (− J T r) → − J T r {\displaystyle \lambda \Delta =\lambda \left(\mathbf {J^{\mathrm {T} }J} +\lambda \mathbf {I} \right)^{-1}\left(-\mathbf {J} ^{\mathrm {T} }\mathbf {r} \right)=\left(\mathbf {I} -\mathbf {J^{\mathrm {T} }J} /\lambda +\cdots

\right)\left(-\mathbf {J} ^{\mathrm {T} }\mathbf {r} \right)\to -\mathbf {J} ^{\mathrm {T} }\mathbf {r} } , therefore the direction of Δ approaches the direction of the negative gradient − J T r {\displaystyle -\mathbf {J} ^{\mathrm {T} }\mathbf {r} } .

The so-called Marquardt parameter λ {\displaystyle \lambda } may also be optimized by a line search, but this is inefficient, as the shift vector must be recalculated every time λ {\displaystyle \lambda } is changed. A more efficient strategy is this: When divergence occurs, increase the Marquardt parameter until there is a decrease in *S*. Then retain the value from one iteration to the next, but decrease it if possible until a cut-off value is reached, when the Marquardt parameter can be set to zero; the minimization of *S* then becomes a standard Gauss–Newton minimization.

Large-scale optimization

For large-scale optimization, the Gauss–Newton method is of special interest because it is often (though certainly not always) true that the matrix J r {\displaystyle \mathbf {J} _{\mathbf {r} }} is more sparse than the approximate Hessian J r T J r {\displaystyle \mathbf {J} _{\mathbf {r} }^{\mathsf {T}}\mathbf {J_{r}} } . In such cases, the step calculation itself will typically need to be done with an approximate iterative method appropriate for large and sparse problems, such as the conjugate gradient method.

In order to make this kind of approach work, one needs at least an efficient method for computing the product

J r T J r p {\displaystyle \mathbf {J} _{\mathbf {r} }^{\mathsf {T}}\mathbf {J_{r}} \mathbf {p} } ,

for some vector **p**. With sparse matrix storage, it is in general practical to store the rows of J r {\displaystyle \mathbf {J} _{\mathbf {r} }} in a compressed form (e.g., without zero entries), making a direct computation of the above product tricky due to the transposition. However, if one defines $\mathbf{c}_i$ as row *i* of the matrix J r {\displaystyle \mathbf {J} _{\mathbf {r} }}, the following simple relation holds:

J r T J r p = ∑ i c i (c i ⋅ p) , {\displaystyle \mathbf {J} _{\mathbf {r} }^{\mathsf {T}}\mathbf {J_{r}} \mathbf {p} =\sum _{i}\mathbf {c} _{i}(\mathbf {c} _{i}\cdot \mathbf {p}),}

so that every row contributes additively and independently to the product. In addition to respecting a practical sparse storage structure, this expression is well suited for parallel computations. Note that every row $\mathbf{c}_i$ is the gradient of the corresponding residual r_i; with this in mind, the formula above emphasizes the fact that residuals contribute to the problem independently of each other.

Related algorithms

In a quasi-Newton method, such as that due to Davidon, Fletcher and Powell or Broyden–Fletcher–Goldfarb–Shanno (BFGS method) an estimate of the full Hessian ∂ 2 S ∂ β j ∂ β k {\displaystyle {\frac {\partial ^{2}S}{\partial \beta _{j}\partial \beta _{k}}}} is built up numerically using first derivatives ∂ r i ∂ β j {\displaystyle {\frac {\partial r_{i}}{\partial \beta _{j}}}} only so that after *n* refinement cycles the method closely approximates to Newton's

method in performance. Note that quasi-Newton methods can minimize general real-valued functions, whereas Gauss–Newton, Levenberg–Marquardt, etc. fits only to nonlinear least-squares problems.

Another method for solving minimization problems using only first derivatives is gradient descent. However, this method does not take into account the second derivatives even approximately. Consequently, it is highly inefficient for many functions, especially if the parameters have strong interactions.

Notes

1. ^ **a** **b** Björck (1996)
2. ^ Björck (1996), p. 260.
3. ^ *Mascarenhas (2013), "The divergence of the BFGS and Gauss Newton Methods", Mathematical Programming,* ***147*** *(1): 253–276, arXiv:1309.7922, doi:10.1007/s10107-013-0720-6*
4. ^ Björck (1996), p. 341, 342.
5. ^ Fletcher (1987), p. 113.
6. ^ http://www.henley.ac.uk/web/FILES/maths/09-04.pdf
7. ^ Nocedal (1999), p. 259.

References

- *Björck, A. (1996). Numerical methods for least squares problems. SIAM, Philadelphia. ISBN 0-89871-360-9.*
- *Fletcher, Roger (1987). Practical methods of optimization (2nd ed.). New York: John Wiley & Sons. ISBN 978-0-471-91547-8..*
- *Nocedal, Jorge; Wright, Stephen (1999). Numerical optimization. New York: Springer. ISBN 0-387-98793-2.*

External links

Implementations

Artelys Knitro is a non-linear solver with an implementation of the Gauss–Newton method. It is written in C and has interfaces to C++/C#/Java/Python/MATLAB/R.