Applied Mathematics 205

Unit I: Data Fitting

Lecturer: Dr. David Knezevic

Unit I: Data Fitting

Chapter I.4: Nonlinear Least Squares

Nonlinear Least Squares

So far we have looked at finding a “best fit” solution to a linear system (linear least-squares)

A more difficult situation is when we consider least-squares for nonlinear systems

Key point: We are referring to linearity in the parameters, not linearity of the model

(e.g. polynomial $p_n(x; b) = b_0 + b_1 x + \ldots + b_n x^n$ is nonlinear in x, but linear in b!)

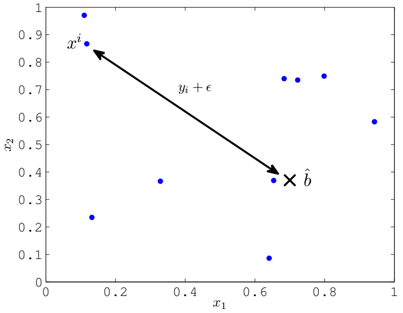
In nonlinear least-squares, we fit functions that are nonlinear in the parameters

Nonlinear Least Squares: Example

Example: Suppose we have a radio transmitter at $\hat{b} = (\hat{b}_1, \hat{b}_2)$ somewhere in $[0,1]^2$ ($\times$)

Suppose that we have 10 receivers at locations $(x_1^1, x_2^1), (x_1^2, x_2^2), \ldots, (x_1^{10}, x_2^{10}) \in [0,1]^2$ (●)

Receiver i gives a measurement for the distance y_i to the transmitter, but there is some error/noise (ϵ)



Nonlinear Least Squares: Example

Let b be a candidate location for the transmitter

The distance from b to (x_1^i, x_2^i) is

$$d_i(b) \equiv \sqrt{(b_1 - x_1^i)^2 + (b_2 - x_2^i)^2}$$

We want to choose b to match the data as well as possible, hence minimize the residual $r(b) \in \mathbb{R}^{10}$ where $r_i(b) = y_i - d_i(b)$

Nonlinear Least Squares: Example

In this case, $r_i(\alpha + \beta) \neq r_i(\alpha) + r_i(\beta)$, hence nonlinear least-squares!

Define the objective function $\phi(b) = \frac{1}{2}\|r(b)\|_2^2$, where $r(b) \in \mathbb{R}^{10}$ is the residual vector

The $1/2$ factor in $\phi(b)$ has no effect on the minimizing b, but leads to slightly cleaner formulae later on

Nonlinear Least Squares

As in the linear case, we seek to minimize ϕ by finding b such that $\nabla\phi = 0$

We have $\phi(b) = \frac{1}{2}\sum_{j=1}^{m}[r_j(b)]^2$

Hence for the i^{th} component of the gradient vector, we have

$$\frac{\partial\phi}{\partial b_i} = \frac{\partial}{\partial b_i}\frac{1}{2}\sum_{j=1}^{m} r_j^2 = \sum_{j=1}^{m} r_j \frac{\partial r_j}{\partial b_i}$$

Nonlinear Least Squares

This is equivalent to $\nabla\phi = J_r(b)^T r(b)$ where $J_r(b) \in \mathbb{R}^{m\times n}$ is the Jacobian matrix of the residual

$$\{J_r(b)\}_{ij} = \frac{\partial r_i(b)}{\partial b_j}$$

Hence we seek $b \in \mathbb{R}^n$ such that:

$$J_r(b)^T r(b) = 0$$

Exercise: Show that $J_r(b)^T r(b) = 0$ gives the normal equations when the residual is linear (i.e. when $r(b) = y - Ab$)

Nonlinear Least Squares

$J_r(b)^T r(b) = 0$ has n equations (since $J_r(b)^T \in \mathbb{R}^{n\times m}$) and n unknowns (since $b \in \mathbb{R}^n$)

But in general it is a nonlinear system that we have to solve iteratively

An important recurring theme is that linear systems can be solved in "one shot," whereas nonlinear generally requires iteration

We will briefly introduce Newton's method for solving this system and defer detailed discussion until Unit IV

Nonlinear Least Squares

Recall Newton's method for a function of one variable: find $x \in \mathbb{R}$ such that $f(x) = 0$

Let x_k be our current guess, and $x_k + \Delta x = x$, then Taylor expansion gives

$$0 = f(x_k + \Delta x) = f(x_k) + \Delta x f'(x_k) + O((\Delta x)^2)$$

It follows that $f'(x_k)\Delta x \approx -f(x_k)$ (approx. since we neglect the higher order terms)

This motivates Newton's method: $f'(x_k)\Delta x_k = -f(x_k)$, where $x_{k+1} = x_k + \Delta x_k$

Nonlinear Least Squares

This argument generalizes directly to functions of several variables

For example, suppose $F : \mathbb{R}^n \to \mathbb{R}^n$, then find x s.t. $F(x) = 0$ by

$$J_F(x_k)\Delta x_k = -F(x_k)$$

where J_F is the Jacobian of F, $\Delta x_k \in \mathbb{R}^n$, $x_{k+1} = x_k + \Delta x_k$

In Unit IV we will prove that Newton's method converges quadratically, i.e. $\|\Delta x_{k+1}\| \approx \|\Delta x_k\|^2$

Nonlinear Least Squares

In the case of nonlinear least squares, to find a stationary point of ϕ we need to find b such that $J_r(b)^T r(b) = 0$

That is, we want to solve $F(b) = 0$ for $F(b) \equiv J_r(b)^T r(b)$

We apply Newton's Method, hence need to find the Jacobian, J_F, of the function $F : \mathbb{R}^n \to \mathbb{R}^n$

Nonlinear Least Squares

Consider $\frac{\partial F_i}{\partial b_j}$ (this will be the ij entry of J_F):

$$\begin{aligned}
\frac{\partial F_i}{\partial b_j} &= \frac{\partial}{\partial b_j}\left(J_r(b)^T r(b)\right)_i \\
&= \frac{\partial}{\partial b_j}\sum_{k=1}^{m}\frac{\partial r_k}{\partial b_i} r_k \\
&= \sum_{k=1}^{m}\frac{\partial r_k}{\partial b_i}\frac{\partial r_k}{\partial b_j} + \sum_{k=1}^{m}\frac{\partial^2 r_k}{\partial b_i \partial b_j} r_k
\end{aligned}$$

Gauss-Newton Method

It is generally a pain to deal with the second derivatives in the previous formula, second derivatives get messy!

Key observation: As we approach a good fit to the data, the residual values $r_k(b)$, $1 \leq k \leq m$, should be small

Hence we omit the term $\sum_{k=1}^{m} r_k \frac{\partial^2 r_k}{\partial b_i \partial b_j}$

(We'll see in Unit IV that it is very common in numerical optimization to avoid second derivatives)

Gauss-Newton Method

Note that $\sum_{k=1}^{m} \frac{\partial r_k}{\partial b_j} \frac{\partial r_k}{\partial b_i} = (J_r(b)^T J_r(b))_{ij}$, so that when the residual is small $J_F(b) \approx J_r(b)^T J_r(b)$

Then putting all the pieces together, we obtain the iteration: $b_{k+1} = b_k + \Delta b_k$ where

$$J_r(b_k)^T J_r(b_k) \Delta b_k = -J(b_k)^T r(b_k), \qquad k = 1, 2, 3, \ldots$$

This is known as the Gauss-Newton Algorithm for nonlinear least squares

Gauss-Newton is a convenient alternative to Newton's method here, and is a popular method for nonlinear least squares

Gauss-Newton Method

Note that Gauss-Newton is equivalent to solving the linear least squares problem $J_r(b_k)\Delta b_k \simeq -r(b_k)$ at each iteration

This is a common refrain in Scientific Computing: Replace a nonlinear problem with a sequence of linear problems

Computing the Jacobian

To use Gauss-Newton in practice, we need to be able to compute the Jacobian matrix $J_r(b_k)$ for any $b_k \in \mathbb{R}^n$

We can do this "by hand", e.g. in our transmitter/receiver problem we would have:

$$[J_r(b)]_{ij} = -\frac{\partial}{\partial b_j}\sqrt{(b_1 - x_1^i)^2 + (b_2 - x_2^i)^2}$$

Differentiating by hand is feasible in this case, but it can become impractical if $r(b)$ is more complicated

Or perhaps our mapping $b \to y$ is a "black box" — no closed form equations hence not possible to differentiate the residual!

Computing the Jacobian

So, what is the alternative to "differentiation by hand"?
Finite difference approximation: for $h \ll 1$ we have

$$[J_r(b_k)]_{ij} \approx \frac{r_i(b_k + e_j h) - r_i(b_k)}{h}$$

Avoids tedious, error prone differentiation of r by hand!

Also, can be used for differentiating "black box" mappings since we only need to be able to evaluate $r(b)$

As we'll see, Matlab allows us to use either approach: analytical differentiation or finite differences

Gauss-Newton Method

We derived the Gauss-Newton algorithm method in a natural way:

- apply Newton's method to solve $\nabla\phi = 0$
- neglect the second derivative terms that arise

However, Gauss-Newton is not widely used in practice since it doesn't always converge reliably

Levenberg-Marquardt Method

A more robust variation of Gauss-Newton is the Levenberg-Marquardt Algorithm, which uses the update

$$[J^T(b_k)J(b_k) + \mu_k \mathrm{diag}(S^T S)]\Delta b = -J(b_k)^T r(b_k)$$

where[1] $S = \mathrm{I}$ or $S = J(b_k)$, and some heuristic is used to choose μ_k

This looks like our "regularized" underdetermined linear least squares formulation!

[1] In this context `diag`(A) means "zero the off-diagonal part of A"

Levenberg-Marquardt Method

Key point: The regularization term $\mu_k \mathrm{diag}(S^T S)$ improves the reliability of the algorithm in practice

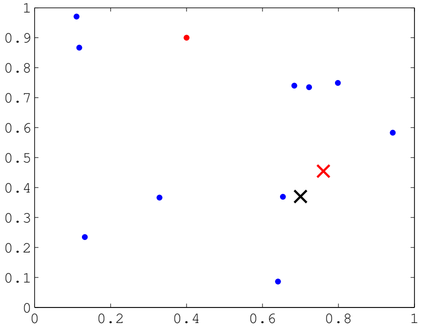
Levenberg-Marquardt is implemented in Matlab's optimization toolbox: `lsqnonlin`

We need to pass the residual to `lsqnonlin`, and we can also pass the Jacobian matrix or ask for a finite-differenced Jacobian

Now let's use `lsqnonlin` to solve our transmitter/receiver problem...

Nonlinear Least Squares: Example

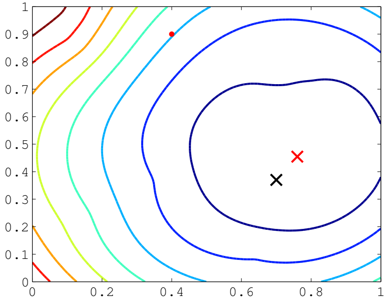
Matlab example: Using `lsqnonlin` in Matlab we provide an initial guess (●), and converge to the final solution (×)



Nonlinear Least Squares: Example

Levenberg-Marquardt minimizes $\phi(b)$, as we see from the contour plot of $\phi(b)$ below

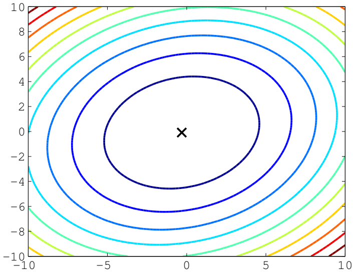
Recall $\times$ is the true transmitter location, $\times$ is our best-fit to the data; $\phi(\times) = 0.0248 < 0.0386 = \phi(\times)$, why?

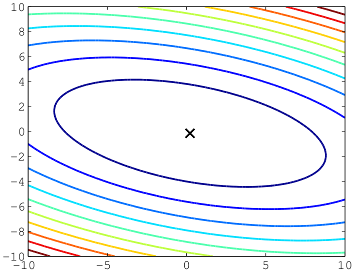


These contours are quite different from what we get in linear problems

Linear Least-Squares Contours

Two examples of linear least squares contours for $\phi(b) = \|y - Ab\|_2^2$, $b \in \mathbb{R}^2$

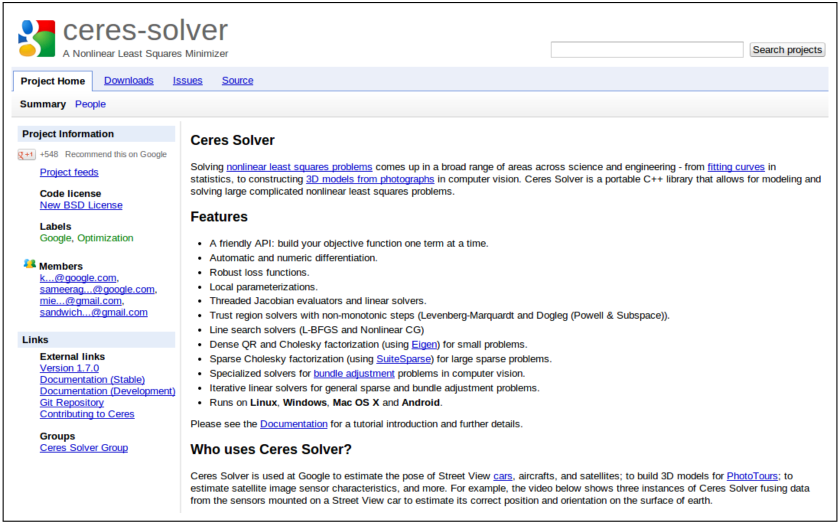




In linear least squares $\phi(b)$ is quadratic, hence contours are "hyperellipses"

Nonlinear Least-Squares in the Real World

Nonlinear least squares (Levenberg-Marquardt) is widely used in practical applications, e.g.



ceres-solver

A Nonlinear Least Squares Minimizer

Project Information

+548 Recommend this on Google

Project feeds

Code license
New BSD License

Labels
Google, Optimization

Members
k...@google.com,
sameerag...@google.com,
mie...@gmail.com,
sandwich...@gmail.com

Links

External links
Version 1.7.0
Documentation (Stable)
Documentation (Development)
Git Repository
Contributing to Ceres

Groups
Ceres Solver Group

Ceres Solver

Solving nonlinear least squares problems comes up in a broad range of areas across science and engineering - from fitting curves in statistics, to constructing 3D models from photographs in computer vision. Ceres Solver is a portable C++ library that allows for modeling and solving large complicated nonlinear least squares problems.

Features

- A friendly API: build your objective function one term at a time.
- Automatic and numeric differentiation.
- Robust loss functions.
- Local parameterizations.
- Threaded Jacobian evaluators and linear solvers.
- Trust region solvers with non-monotonic steps (Levenberg-Marquardt and Dogleg (Powell & Subspace)).
- Line search solvers (L-BFGS and Nonlinear CG)
- Dense QR and Cholesky factorization (using Eigen) for small problems.
- Sparse Cholesky factorization (using SuiteSparse) for large sparse problems.
- Specialized solvers for bundle adjustment problems in computer vision.
- Iterative linear solvers for general sparse and bundle adjustment problems.
- Runs on **Linux**, **Windows**, **Mac OS X** and **Android**.

Please see the Documentation for a tutorial introduction and further details.

Who uses Ceres Solver?

Ceres Solver is used at Google to estimate the pose of Street View cars, aircrafts, and satellites; to build 3D models for PhotoTours; to estimate satellite image sensor characteristics, and more. For example, the video below shows three instances of Ceres Solver fusing data from the sensors mounted on a Street View car to estimate its correct position and orientation on the surface of earth.