Using geodesic Levenberg-Marquardt

March 30, 2015

1 General strategy:

The first thing to do is examine the document string in the `geolevmar` subroutine found in `leastsq.f90`, and make sure you provide at least the function which computes the residuals (`func`). Be careful to use the form specified in the code's documentation.

The strategy for use we suggest is to start by turning everything fancy off, so that this algorithm functions as a standard Levenberg-Marquardt (LM) algorithm. I.e. start with `iaccel`=0, `ibold`=0 and `ibroyden`=0. When you do this, you should pay most attention to `factoraccept` and `factorreject`. I recommend starting with 5 and 2, respectively. Standard LM algorithms set both of these parameters to 10. The problem with making them equal is that the optimizer can get stuck in cycles. For more nuanced reasons (we call it 'delayed gratification' [1]) we also recommend that the accept factor be larger than the reject factor. Play around with these until you are satisfied with the convergence rate.

You might find at first that the default convergence criterion (tolerances and goals, see the list below) are causing the algorithm to give up before true convergence. The procedure for diagnosing and ameliorating this is as follows: After the algorithm gives up, read the convergence message. The integer provided will represent a specific convergence criterion being hit (the list in is `leastsq.f90`), and you should decrease the offending condition to force the algorithm to work harder. Do so until the algorithm hits a different convergence criterion, and repeat until either you are satisfied with the minimum, or the speed of convergence becomes your principal concern. If you are faced with the latter problem, read on for useful options for speeding convergence.

Once you get things working reasonably well with standard LM settings, try turning on geodesic acceleration by setting `iaccel`=1. If you are not providing

a routine to calculate the second directional derivative (see [1, 2] for details), then you need to set `analytic_avv = .false.` so that finite differences are used to estimate it. You will need to pay attention to the finite difference step size `h2` to make sure your calculation is stable. We suggest using `h2`=0.1. Since the acceleration should be a correction to standard LM, the parameter `avmax` limits the length of the acceleration vector in proportion to the proposed LM step size. As such, `avmax`$< 1.$ are recommended.

If you are satisfied with the behavior at this point, experiment with allowing uphill steps by setting `ibold` from 1 (least 'bold' uphill allowances) to 4 (most 'bold' uphill allowances). You can also play with setting `ibroyden`=1, which will employ an estimate of the Jacobian to speed up some iterations, sacrificing some accuracy.

For more details on why least-squares fitting is so challenging, and for intuition on how various parts of this algorithm work, see [3, 2]. See [1] for details specific to implementation and benchmarked performance of the algorithm.

2 Parameter defaults and details:

These parameters are ordered by their relative importance. A lot of this information is a repeat of what can be found in the documentation to the source code in `leastsq.f90`.

Format:

- `parameter_name` = <suggested default> : Some words you might find useful.

Basic Levenberg-Marquardt (LM) Parameters:

- `factoraccept` = 5 : This is the factor by which the LM parameter (often denoted λ) is **decreased after a step is accepted.**
- `factorreject` = 2 : This is the factor by which the LM parameter is **increased after a step is rejected.**
- `maxlam` = 1.E7 : The routine will never let λ become larger than this.
- `imethod` = 0 : This is an integer which specifies the exact way λ is updated. See `leastsq.f90` and [1] for more details.

- `initialfactor` = 1 : The initial λ.
- `mode` = 0 : 0 or 1. Chooses between the 'Levenberg' and 'Marquardt' style of matrix added to $J^T J$ `leastsq.f90`.

Basic convergence criterion:

- `maxiter` = 500 : The total number of steps the algorithm will attempt. This should depend on how long a function and Jacobian evaluation takes, and how long you are willing to wait.
- `maxfev` = 0 : The maximum number of allowed residual function (`func`) evaluations. 0 indicates no limit.
- `Cgoal` = 1 : The goal for the minimum squared error.
- `maxjev` = 0 : The maximum number of allowed Jacobian (`jacobian`) evaluations. 0 indicates no limit.

Geodesic acceleration parameters:

- `iaccel` = 1 : 1 or 0 for geodesic acceleration on or off.
- `h2` = 0.1 : Finite-difference step size for second-directional derivative estimation.
- `avmax` = 0.75 : The 'acceleration' should be a correction to the 'velocity' and this enforces that.
- `maxaev` = 0 : Maximum number of analytic second directional derivative (`avv`) evaluations. 0 indicates no limit.

Extra flags:

- `print_level` = 5 : 0-1, how much information to print to `print_unit`.
- `print_unit` = 6 : The integer representing the place to print messages. Either open a file or set to 6 for standard out.
- `analytic_jac` = `.true.` : True if you supply a routine to compute the Jacobian (`jacobian`).

- `analytic_avv` = `.false.` : True if you supply a routine to compute the second directional-derivative (`avv`).
- `center_diff` = `.true.` : True if you want to use centered-difference. More accurate but slower than forward-difference.

Advanced Jacobian update and convergence criterion:

- `ibold` = 0 : (0-4) If nonzero this allows some uphill steps which can speed convergence.
- `ibroyden` = 0 : If nonzero this employs Broyden approximate Jacobian updates. Can speed up algorithm with cost of accuracy.
- `eps` = 1.5E-6 : Precision of evaluation of residuals in `func`.
- `h1` = 1.0E-5 : Step size for finite-difference Jacobian estimate.
- `artol` = 1.E-3 : See `leastsq.f90` for details on these convergence tolerances.
- `gtol` = 1.5E-8
- `xtol` = 1.E-10
- `xrtol` = 1.5E-8
- `frtol` = 1.5E-8

References

[1] Mark K Transtrum and James P Sethna. Improvements to the levenberg-marquardt algorithm for nonlinear least-squares minimization. *arXiv preprint arXiv:1201.5885*, 2012.

[2] Mark K. Transtrum, Benjamin B. Machta, and James P. Sethna. Geometry of nonlinear least squares with applications to sloppy models and optimization. *Phys. Rev. E*, 83:036701, Mar 2011.

[3] Mark K. Transtrum, Benjamin B. Machta, and James P. Sethna. Why are nonlinear fits to data so challenging? *Phys. Rev. Lett.*, 104:060201, Feb 2010.