DDogleg Technical Report Nonlinear Optimization

Revision 2018-1

DDogleg v0.15

Peter Abeles

CONTENTS

LIST OF FIGURES

LIST OF TABLES

DDogleg Technical Report Nonlinear Optimization

Revision 2018-1

DDogleg v0.15

Abstract

DDogleg[1] is a general purposed numerics library. This technical report is focused on describing algorithmic details of nonlinear unconstrained optimization routines found in DDogleg v0.15 and is intended to be used as a reference. A solid understanding of the basic theory of unconstrained optimization is assumed. Theoretical details will only be touched upon with citations for where to find more information. Best practices for implementation, tuning, benchmark results, and justifications for specific implementation decisions are all discussed. API details can be found online at http://ddogleg.org.

This document should be considered a living work in progress and its release is following the philosophy that it is better to release something than to wait forever for perfection. Corrections and other feedback are welcomed.

1 Optimization Techniques

TABLE 1
Variables and Terminology

x	Parameters being optimized. $x \in \mathbb{R}^N$
x_k	Value of parameters at iteration k
p_k	Iteration step, the difference between $x_{k+1} - x_k$
$f(x)$	Scalar cost function being optimized. $f \in \mathbb{R}$
f_k	Short hand for $f(x_k)$
$g(x)$	Gradient of $f(x)$. $g(x) \in \mathbb{R}^n$
g_k	Short hand for $g(x_k)$
$B(x)$	Hessian matrix or an approximation. $B(x) \in \mathbb{R}^{N \times N}$
B_k	Short hand for $B(x_k)$
$H(x)$	Inverse Hessian matrix or an approximation. $H(x) \in \mathbb{R}^{N \times N}$
H_k	Short hand for $H(x_k)$
positive definite	Matrix B is positive definite when $y^T By > 0$ for all non-zero vectors y
Δ_k	Trust Region size at step k. $\Delta_k \in \mathbb{R}^+$
MAX_VALUE	The largest possible floating point value

This section provides overview of different numerical techniques provided in DDogleg for unconstrained optimization. Techniques described here can often be applied to different specific problems.

1.1 Trust Region

Trust Region refers to a family of optimization methods that operate by assuming a quadratic model is accurate within a local "trust region". The trust region's size is adjusted based on the quadratic model's performance in previous iterations. A summary of Trust Region, as implemented in DDogleg, is found in Algorithm 1. This implementation[2] is primarily based on the description found in [3].

At every iteration the Trust Region subproblem is solved for, either exactly or approximately:

$$\min_{p \in \mathbb{R}^n} m_k(p) = f_k + g_k^T p + \tfrac{1}{2} p^T B_k p \quad s.t. \, \|p\| \leq \Delta_k \tag{1}$$

where $m(p) \in \mathbb{R}$ is a quadratic model approximating $f(x_k)$, $p \in \mathbb{R}^N$ is the step or change in state, $B \in \mathbb{R}^{N \times N}$ is a symmetric matrix representing the Hessian or an approximation, and $\Delta_k \in \mathbb{R}^+$ is the trust region size. The unconstrained solution to Eq. 1 is easily found by setting the first derivative to zero:

$$p = -B_k^{-1} g_k \tag{2}$$

An exact solution to (1) is expensive to compute and approximate methods are typically used instead. The Cauchy Point and Dogleg are approximate methods and included in the DDogleg library.

1. DDogleg's name comes from the double dogleg Trust Region method, which is not included with DDogleg.
2. The more traditional variant described in [1], [2] were considered but found to converge slower in test problems.

Algorithm 1 Trust Region

$k \leftarrow 0$, $\Delta_0 \in (0, \Delta_{max})$
  Δ_{max} is the maximum trust region size
  Δ_0 is the initial trust region size. ▷ Section 1.1.3
while $k < k_{\max}$ and not *done* **do**
  p_k update by optimizing Eq. 1 ▷ Sections 1.1.1 and 1.1.2
  $\delta_f \leftarrow f(x_k) - f(x_k + p_k)$ ▷ Actual reduction in score
  $\delta_m \leftarrow m_k(0) - m_k(p_k) = -g_k^T p - \frac{1}{2} p^T B_k p$ ▷ Predicted reduction in score
  $\nu \leftarrow \delta_f / \delta_f$ ▷ Score reduction ratio
  if $\delta_f \leq 0$ or $\nu < \frac{1}{4}$ **then** ▷ Worse score or model poor?
    $\Delta_{k+1} \leftarrow \frac{1}{2}\Delta_k$
  else ▷ The model is good
    if $\nu > \frac{3}{4}$ **then** ▷ Increase region size?
      $\Delta_{k+1} \leftarrow \min(\max(3\left\|p_k\right\|, \Delta_k), \Delta_{\max})$
    else
      $\Delta_{k+1} \leftarrow \Delta_k$
  if $\delta_f > 0$ and $\nu > 0$ **then** ▷ Is the solution acceptable?
    $x_{k+1} \leftarrow x_k + p_k$ ▷ Update the state
    $done \leftarrow$ F-Test or G-Test ▷ Convergence testing
  else
    $x_{k+1} \leftarrow x_k$
  $k \leftarrow k + 1$

1.1.1 Cauchy Point

The Cauchy Point is the solution which minimizes (1) along the steepest descent direction. It is defined as $p_k^s = \tau_k \hat{p}_k^s$ and is relative to x_{k-1}, where $\hat{p}_k^s$ is a unit vector, and τ_k is a scalar.

$$\hat{p}_k^s = \min_{p \in \mathbb{R}^n} f_k + g_k^T p \quad s.t. \left\|p\right\| \leq \Delta_k \tag{3}$$

The length τ_k is found by minimizing (1) along direction $\hat{p}_k^s$

$$\tau_k = \min_{\tau \geq 0} m_k(\tau v_k^s) \quad s.t. \left\|\tau v_k^s\right\| \leq \Delta_k \tag{4}$$

The solution (see Chapter 4 of [1] for details and diagrams) is as follows:

$$p_k^s = -\tau_k \frac{\Delta_k}{\left\|g_k\right\|} g_k \tag{5}$$

$$\tau_k = \begin{cases} 1 & g_k^T B_k g_k \leq 0 \\ \min\left(1, \left\|g_k\right\|^3 / (\Delta_k g_k^T B_k g_k)\right) & g_k^T B_k g_k > 0 \end{cases} \tag{6}$$

The formulas in (5) and (6) can be improved upon to avoid numerical issues by removing powers of three and division by Δ_k:

$$\hat{g}_k = \frac{g_k}{\left\|g_k\right\|} \tag{7}$$

$$p_k^s = -\bar{\tau}_k \hat{g}_k \tag{8}$$

$$\bar{\tau}_k = \begin{cases} \Delta_k & \hat{g}_k^T B_k \hat{g}_k \leq 0 \\ \min\left(\Delta_k, \left\|g_k\right\| / (\hat{g}_k^T B_k \hat{g}_k)\right) & \hat{g}_k^T B_k \bar{g}_k > 0 \end{cases} \tag{9}$$

The predicted reduction in score is found using:

$$m_k(0) - m_k(p_k) = \bar{\tau}_k \left(\left\|g_k\right\| - \frac{\tau_k \hat{g}_k^T B_k \hat{g}_k}{2} \right) \tag{10}$$

1.1.2 Dogleg

The Dogleg method considers second order terms to provide a more accurate solution to Eq. 1. The optimal solution, as a function of region size, is a curved trajectory. The Dogleg method approximates this curved trajectory using two line segments. The first line starts at the x_{k-1} and ends at the unconstrained Cauchy point. The second heads towards p^b the solution to (2), which is the Gauss-Newton solution. As with equations from Cauchy Point, these equations are not traditional (see [1], [3]) and have been reformulated to avoid powers of three.

$$\hat{g_k} = \frac{g_k}{\|g_k\|} \tag{11}$$

$$p_k^u = -\frac{g_k}{\hat{g_k}^T B_k \hat{g_k}} \tag{12}$$

$$p_k^b = -B_k^{-1} g_k \tag{13}$$

$$p_k^{dog} = \begin{cases} \tau p_k^u & 0 \le \tau < 1 \\ p_k^u + (\tau - 1)(p_k^b - p_k^u) & 1 \le \tau \le 2 \end{cases} \tag{14}$$

where B_k is positive definite, and p_k^{dog} is the point selected by the Dogleg method. The solution to τ can be easily found by solving along each line segment. If B_k is not positive definite then gradient descent is used instead.

Algorithm 2 Selection of Dogleg Step

if B is positive definite **then**
  if $\|p^b\| < \Delta$ **then** ▷ Gauss-Newton solution inside the trust-region?
    $p^{dog} \leftarrow p^b$
  else if $\|p^u\| \ge \Delta$ **then** ▷ Cauchy point outside the trust-region?
    $p^{dog} \leftarrow \Delta \frac{p^u}{\|p^u\|}$
  else
    $p^{dog} \leftarrow$ intersection of $p^u \rightarrow p^b$ and trust-region
else
  $p^{dog} \leftarrow -\Delta \frac{g}{\|g\|}$ ▷ Follow gradient to end of trust region

TODO Diagram

1.1.3 Initial Region Size

Selection of the initial trust region size Δ_0 is important but typically not discussed in reference material [1]–[3] in detail. Initial region size is typically considered a tuning parameter that the user is supposed to select through trial and error. While the Trust region size is dynamically adjusted at each iteration in the Trust Region approach, the initial selection of the trust region size can significantly influence the final convergence.

Here is an example of a possible failure mode when the trust region's size is poorly selected. With the dogleg method, if Δ_0 is too small then a Cauchy step is selected repeatedly. The Cauchy Point takes much smaller steps, increasing the chances of getting stuck in a local minimum.

DDogleg provides two automatic methods for finding the initial region size, with unreliable results. 1) *Unconstrained initial step* and 2) *Cauchy initial step*. With the unconstrained method, the selected algorithm (e.g. Dogleg or Cauchy) selects a step when given trust region of MAX_VALUE. The step it selects is used and the trust region is then set to the length of that step. This works well in many problems but can be overly agressive and take a very large step into a distant plateau. The Cauchy initial step method computes the length of a Cauchy step, then sets the region size to be 10x that. This estimate tends to be conservative will in general converge but can converge slowly.

If the automatic methods fail to produce acceptable results then manual tuning will be necessary. One possible manual tuning procedure is to start with $\Delta_0 = 1$ then trying $\Delta_0 = 100$, and if results improved try $\Delta_0 = 10000$. If results don't get better try 0.1 or other fractions of one.

Recommended Procedure for Selection of Initial Trust Region Size:

1) Turn on verbose output and examine the progress
2) Start with automatic selection using *unconstrained initial step*
3) If this fails then try *Cauchy initial step*
4) If performance is still poor follow manual tuning procedure

For instructions on how to switch between the methods described here consult the JavaDoc of ConfigTrustRegion.

A comparison of different initial conditions for different 'toy' problems is shown in Table 2. In these scenarios, the Automatic Unconstrained method correctly selected the best initial conditions while all the other methods either tied unconstrained's performance or clearly made a poor choice. Unfortunately, these results don't extrapolate to all problems and there are situations where the unconstrained method results in failure. For that reason, the default method is the more conservative Automatic Cauchy.

TABLE 2
Comparison of Initial Region Size

Problem	Automatic Unconstrained			Automatic Cauchy			Manual 1			Manual 100		
	Fit	G	B	Fit	G	B	Fit	G	B	Fit	G	B
Powel	2.0e-17	17	17	2.9e-17	30	26	4.9e-18	23	19	2.0e-17	17	17
Powel Singular	7.3e-11	11	11	2.3e-10	13	13	6.7e-11	11	11	7.3e-11	11	11
Helical Valley	2.5e-26	11	8	5.1e-18	11	11	4.1e-33	9	8	2.7e-26	16	8
B.S. Powell	4.2e-31	25	18	3.7e-23	73	58	2.2e-31	25	18	0	62	43
Bundle 2D	3.7e-10	111	36	7.0e-18	111	36	9.0e-10	251	93	3.3e-10	112	37
Bundle 2D [1]	7.0e-19	4	4	7.0e-18	4	4	1.5e-15	5	5	7.0e-18	4	4

Fit is the final fit score where zero is a perfect fit. *G* is the number of times the gradient was computed. *B* is the number of times the Hessian was computed. B is by far the most expensive step.
Unless specified otherwise, all methods use Dogleg with a Cholesky solver.
[1] Uses QR with column pivots instead of Cholesky and can handle the nearly singular initial state.

1.2 Scaling

Variable scaling can refer to several different parts of the non-linear optimization problem. Here we will discuss scaling of the input variables x and scaling of the Hessian B_k internally. Throughout the literature, correct scaling, in all of its forms, is emphasized as an essential task and that you are a bad person doomed to failure if you skip it.

In reality there are problems where it is essential, but in it does not always help and can sometimes hurt. How can it hurt? When correctly applied, scaling does not change the location of minimums, but will change the path towards a minimum [5] and can change which variables are emphasized. Scaling should be treated like other tuning parameters and experimented with.

In general, when performing floating point arithmetic [4], it is advisable to avoid mixing very large (e.g. 1e12) numbers with very small (e.g. 1e-12) numbers to reduce round off errors. Thus it is desirable to have all numbers take on values close to one and have a standard deviation of one.

Another reason to scale variables is to reduce the emphasis on sensitive variables. A sensitive variable is one in which a small change in it's value results in a large error, e.g. $1/x^2$ when $x \approx 0$. This can cause the optimization routine to get stuck since any step causes a large error.

1.2.1 Input Scaling

The optimization cost functions should be designed so that the units of all the variables are on average one with a standard deviation of one. This scaling cannot be done by DDogleg automatically because it does not have knowledge of each variable's range.

Further Reading:

- Chapter 2.2 of [1] contains an illustration of poor scaling.

TODO provide examples in this document.

1.2.2 Hessian Scaling

For Hessian Scaling, the Hessian matrix is re-scaled so that the diagonal elements are approximately one. We will discuss Hessian scaling in regards to Trust Region (and Levenberg-Marquardt) methods.

Hessian Scaling is done by applying a diagonal matrix D with positive elements to the Trust Region sub-problem (1). Changing p into its scalled scaled version $\tilde{p} = Dp$. The trust region is no longer a circle but an ellipse [1], resulting in this alternative trust region subproblem:

$$\min_{p \in \mathbb{R}^n} m_k(p) = f_k + g_k^T p + \tfrac{1}{2} p^T B_k p \quad s.t. \, \|Dp\| \leq \Delta_k \tag{15}$$

As suggested in [1] this is implemented internally in DDogleg by substituting Dp for p, $D^{-1}g_k$ for g_k, and $D^{-1}B_kD^{-1}$ for B_k.

DDogleg can be configured to automatically compute and apply Hessian scaling at each iteration or to not apply Hessian scaling. Automatic scaling parameters are found using second derivatives $\frac{\partial^2 f}{\partial x_i^2}$ from in the Hessian's diagonal elements. Variables with larger second derivatives are more sensitive, thus their movement should be restricted more. The specific formula used in DDogleg is as follows:

$$D_k^{ii} = \max\left(d_\triangledown, \min\left(\sqrt{|B_k^{ii}|}, d_\triangle\right)\right) \tag{16}$$

where $d_\triangledown$ is the minimum allowed scaling value and $d_\triangle$ is the maximum. This approach can handle negative definite B_k and has the desirable property [5] that the diagonal elements in $\tilde{B}_k = D^{-1}B_kD^{-1}$ will typically be $\tilde{B}_k^{ii} \approx 1$, unless clamped or B_k^{ii} is zero.

1.3 Schur Complement

For sparse systems, with a specific structure, the Schur Complement can be used to greatly reduce the computational cost. What would have taken hours or days to solve can be solved in seconds or minutes. Bundle Adjustment is one such problem [6]. The power of the Schur Complement comes from breaking the system into sub-problems. Since the matrix has a special structure, smaller block diagonal matrices are inverted and sparse fill in [7] is avoided, making it highly efficient.

Let $M \in \mathbb{R}^{N \times N}$ be an invertible square matrix which has been broken up into four submatrices. It can be factorized as follows:

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ CA^{-1} & 1 \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & \bar{D} \end{bmatrix} \begin{bmatrix} 1 & A^{-1}B \\ 0 & 1 \end{bmatrix} \tag{17}$$

It can then be shown that

$$\bar{D} = D - CA^{-1}B \tag{18}$$

This is known as the Schur complement of the block A of matrix M. The Schur Complement of block D of matrix M can also be found:

$$\bar{A} = A - BD^{-1}C \tag{19}$$

We will discuss the former but either can be used. Which one is preferred is simply the one which can be computed fastest and is dependent on the matrix's structure.

These relationships can then be used to solve the following system:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{20}$$

Algorithm 3 Schur Complement to solve a reduced system

1: $\bar{D} = D - CA^{-1}B$
2: $\bar{b}_2 = b_2 - CA^{-1}b_1$
3: $\bar{D}x_2 = \bar{b}_2$ ▷ Reduced System
4: $Ax_1 = b_1 - Bx_2$

For the least squares problem, the Schur Complement is applied to the Jacobian inner product:

$$J^T J = \begin{bmatrix} A & B \\ B^T & D \end{bmatrix} \tag{21}$$

Symmetry can be taken advantage of in matrix multiplication and when solving the system, which DDogleg does. The Schur Complement is implemented in DDogleg by having the user compute the Jacobian in two column matrices.

$$J = [J_1, J_2] \tag{22}$$

The rest is handled automatically. See the *SchurJacobian* interface and *ExampleSchurComplementLeastSquares*.

1.4 Linear Algebra

Linear algebra and matrix operations are the workhorses that non-linear optimization is built upon. For Trust Region methods, linear solvers are extremely important. A linear solver solves equations of the form:

$$AB = y \tag{23}$$

where $A \in \mathbb{R}^{M \times N}$, $B \in \mathbb{R}^N$ is unknown, and $y \in \mathbb{R}^M$. Solving for B is the most expensive operation, potential source of numerical errors, and often the cause of fatal exceptions.

A singular matrix is one in which there is no unique solution to B. This can happen when the search hits a region with zero slope over some of the parameters. A slope of zero indicates that changing the parameter does not affect the cost function's value. Not all linear methods can handle this situation, in fact most cannot. Nearly singular systems are a also problem and some solvers are more sensitive than others.

Dense solvers tend to be very robust and typically have built in support to minimize overflow. There are many tools that dense solvers can use to mitigate singular and nearly singular systems. For example, they can dynamically change the order in which they decompose the matrix by pivoting (e.g. LUP and QRP). Methods exist which can even decompose and solve singular systems, (e.g. QRP and SVD). Default solvers in DDogleg attempt to strike a balance between speed and robustness. Thus the default will be Choleksy or QR, but if a request for a robust solver is made then QRP might be used instead. See Table 2 for a fairly dramatic example of how changing the solver can improve performance.

Sparse solvers are another story. While by no means new, they are a younger field and orders of magnitude more complex to implement. An example of this is Cholesky decomposition. A minimal dense implementation can be done in around 10 lines of code. A direct sparse equivalent is measured in hundreds of lines of code. Automatic scaling is often omitted in the sparse case making sparse solvers more prone to overflow. Pivoting is difficult for sparse systems because

the decomposed structure needs to be known in advance. For the reasons just mentioned, when dealing with large sparse systems, more emphasis is placed on massaging data prior to applying a linear solver.

Fortunately, for a user of DDogleg, almost all of this complexity is hidden from you. For advanced users there is still the option to choose the solver. Potentially enabling you to solve otherwise unsolvable singular/degenerate systems. Any of the solvers in Efficicient Java Matrix Library (EJML) [8] can be used in DDogleg. Solvers from other libraries can be used too, if you wrap them in the appropriate interface.

If you wish to learn more about the computational side of linear algebra then "Fundamentals of Matrix Computations" [9] and "Direct Methods for Sparse Linear Systems" [7] are recommended for dense and sparse systems, respectively.

2 UNCONSTRAINED MINIMIZATION

TABLE 3
Definitions and API for Unconstrained Minimization

FunctionNtoS	Interface for function $f(x)$
FunctionNtoN	Interface for gradient $g(x)$
	Can be computed numerically
UnconstrainedMinimization	Interface for unconstrained minimization

TABLE 4
Summary of Unconstrained Minimization Methods.

Method	Iteration	Convergence	Singular	Negative-Definite	Dense	Sparse
Quasi-Newton BFGS	$O(N^2)$	Super Linear	Yes	Yes	Yes	
Trust Region BFGS Cauchy	$O(N^2)$	Linear	Yes	Yes	Yes	Yes
Trust Region BFGS Dogleg	$O(N^2)$	Super Linear	[1]	[1]	Yes	Yes

- *Iteration*: Runtime complexity of update step. N is number of parameters.
- *Convergence*: how fast it converged.
- *Singular*: indicates that it can process singular systems.
- *Negative-Definite*: indicate that it can process negative definite systems
- *Dense* and *Sparse*: indicate that dense and/or sparse matrices can be processed.
- [1] Switches to Cauchy in this situation.

Unconstrained minimization seeks to find a set of parameters which minimizes a function, e.g.:

$$\min_{x \in \mathbb{R}^N} f(x) \tag{24}$$

where x is an N-dimensional vector and $f : \mathbb{R}^N \Rightarrow \mathbb{R}$ is a function which outputs a scalar. A global minimum x^* is a minimum such that $f(x^*) \leq f(x)$ for all x. Local minimums are ones where $f(x^*) \leq f(x)$ for all $x \in \mathcal{N}$, where $\mathcal{N}$ is bounded set of subset of $\mathbb{R}^N$. For most non-linear problems the best that can be done is to find a local minimum. A good introduction to the theory on this subject can be found in [1].

In DDogleg, solutions to this problem are found using use the gradient and Hessian. Convergence is found by examining the function's rate of change and the gradient, Section 2.1. Computing the Hessian is often tedious and computationally expensive so iterative approximations of the Hessian are used, Section 2.4. The remaining sections discuss specific implementation details of applying general purpose algorithms to this problem.

2.1 Convergence Test

All unconstrained minimization algorithms in DDogleg use the same convergence tests. F-Test checks the function's value to see if it has converged. G-Test checks the gradient to see if it zero and is at a local minima. To disable a test assign it a value less than zero.

$$\begin{array}{ll} \text{F-Test} & ftol \cdot f(x) \leq f(x) - f(x+p) \\ \text{G-Test} & gtol \leq \|g(x)\|_\infty \end{array}$$

2.2 Line Search

Line Search methods are iterative methods where at each iteration they seek to find a step length α_k which provides significant decrease in the cost along the search direction p_k when starting at x_k. This can be summarized as:

$$x_{k+1} = x_k + \alpha_k p_k \tag{25}$$

$$f_{k+1} < \beta f_k \tag{26}$$

where β is some how defined to describe the meaning of significant.

In addition to significant decrease, curvature conditions also need to be meet. The strong Wolfe condition is used in some line search algorithms to decide if sufficient decrease and curvature conditions have been meet:

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \Delta f_k^T p_k \quad (27)$$

$$|\Delta f(x_k + \alpha_k p_k)^T p_k| \leq c_2 |\Delta f_k^T p_k| \quad (28)$$

where $0 < c_1 < c_2 < 1$.

In DDogleg, two line search methods are provided Fletcher86 [10] and More94 [11]. Both of which explicitly meet the Wolfe condition when selecting a step length. More94 has shown better convergence and is the default option. The implementation of More94 contained in DDogleg is a port of csrch function in MINPACK-2 [12].

2.3 Quasi-Newton

Quasi-Newton is a description of a general framework where at each iteration an approximation to a full Newton iteration is performed. In DDogleg, Quasi-Newton is done by solving for the search direction p_k using an approximation to the inverse Hessian B_k^{-1} followed by the line search method of your choice which meets the Wolfe condition.

$$p_k = -B_k^{-1} \delta f_k \quad (29)$$

For computational efficiency and robustness, the inverse $H_k = B_k^{-1}$ is estimated directly using BFGS. By estimating the inverse matrix we avoid the requirement that B_k be positive definite and a costly $O(N^3)$ matrix decomposition and replace it with an inexpensive $O(N^2)$ update instead.

2.4 Hessian Approximation

Exact methods of calculating the Hessian can be difficult to derive and expensive to compute. Algorithms which utilize exact Hessians have faster convergence but this is often offset by additional computational cost [1]. DDogleg uses gradient based methods for estimating the Hessian. DFP [13] to estimate the Hessian and BFGS (Broyden-Fletcher-Goldfarb-Shanno) [1], [2][3] to estimate the inverse hessian.

DFP
$$B_{k+1} = (I - \rho_k \gamma_k s_k^T) B_k (I - \rho_k s_k \gamma_k^T) + \rho_k \gamma_k \gamma_k^T \quad (30)$$

BFGS
$$H_{k+1} = H_k - \frac{H_k \gamma_k \gamma_k^T H_k}{\gamma_k^T H_k \gamma y_k} + \frac{s_k s_k^T}{y_k^T s_k} \quad (31)$$

$$\rho_k = \frac{1}{\gamma_k^T s_k}$$

where $H_k = B_k^{-1}$, $s_k = x_{k+1} - x_k$, and $y_k = \nabla f_{k+1} - \nabla f_k$.

DDogleg does not explicitly provide support for using an exact Hessian. If you wish to use an exact Hessian this can be accomplished with a bit of coding by extending base classes in DDogleg. Search code for where BFGS is being used, extend that class, and override the function where the Hessian is estimated. For example, *UnconMinTrustRegionBFGS* can be used to create your own exact Hessian unconstrained minimization trust region implementation.

2.5 Trust Region

Trust Region methods can be directly applied to unconstrained minimization without any change in their framework. Specific implementation details are listed below:

1) The Hessian is initialized with an identity matrix.
2) The Hessian and inverse are iteratively approximated using DFP and BFGS.
3) The Hessian is only updated when the Wolfe condition is meet
4) Dogleg-BFGS avoids $O(N^3)$ matrix decomposition by computing the inverse Hessian directly with BFGS in $O(N^2)$ time.

Future Work:

1) Remove the need to compute B_k and H_k by directly computing the Cholesky factors of B_k.

TABLE 5
Definitions and API for Unconstrained Nonlinear Least-Squares

x	Parameter vector which is being optimized and has n elements. $x \in \mathbb{R}^N$
$f(x)$	Scalar error function being optimized. $f(x) \geq 0$
f_k	Short hand for $f(x_k)$
$F(x)$	Residual function from $\mathbb{R}^N \rightarrow \mathbb{R}^M$
$J(x)$	Jacobian of residual function. $J(x) \in \mathbb{R}^{N \times M}$
$B(x)$	Hessian approximation and is set to $B = J^T J \in \mathbb{R}^{N \times N}$
$g(x)$	Gradient of $f(x)$, which is $J(x)^T F(x) \in \mathbb{R}^N$
g_k	Short hand for $g(x_k)$
α	Mixing coefficent for Levenberg's and Marquardt's equations
FunctionNtoM	Interface for residuals $F(x) \in \mathbb{R}^M$
FunctionNtoMxN	Interface for Jacobian $J(x) \in \mathbb{R}^{M,N}$ Can be computed numerically
UnconstrainedLeastSquares	High level interface for this unconstrained least squares
UnconstrainedLeastSquaresSchur	Least-Squares using Schur Complement

TABLE 6
Summary of Unconstrained Least-Squares Methods.

Method	Iteration	Convergence	Singular	Dense	Sparse	Schur
Trust Region LS Cauchy	$O(N^3)$	Linear	Yes	Yes	Yes	Yes
Trust Region LS Dogleg	$O(N^3)$	Super Linear	[1]	Yes	Yes	Yes
Levenberg-Marquardt	$O(N^3)$	Super Linear	[2]	Yes	Yes	Yes

- *Iteration*: Runtime complexity of update step. N is number of parameters.
- *Convergence*: how fast it converged.
- *Singular*: indicates that it can process singular systems.
- *Negative-Definite*: indicate that it can process negative definite systems
- *Dense* and *Sparse*: indicate that dense and/or sparse matrices can be processed.
- *Schur*: If a variant is available that uses the Schur Complement
- [1] Switches to Cauchy in this situation.
- [2] Depends on solver and mixing coefficient, but in most configurations it can handle singular systems.

3 UNCONSTRAINED LEAST-SQUARES

Unconstrained Least-Squares is a special case of Unconstrained Minimization. It refers to a problem where the function being optimized has the form

$$\min_x f(x) = \frac{1}{2} \sum_{j=1}^{m} r_j^2(x) \tag{32}$$

where $r_j(x)$ is a scalar function j which outputs the residual (predicted value subtracted the observed value) error. By definition $f(x) \geq 0$. Matrix notation can also be used to defined (32):

$$\min_x f(x) = \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \left\| F(x) \right\|_2^2 \tag{33}$$

where $F(x) = [r_1(x), r_2(x), \cdots, r_m(x)]^T$. Then the Jacobian is defined as:

$$J(x) = \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix} \tag{34}$$

$$\nabla r_j(x)^T = \left[\frac{\partial r_j}{\partial x_1}, \frac{\partial r_j}{\partial x_2}, \cdots, \frac{\partial r_j}{\partial x_n} \right]^T \tag{35}$$

and the Gradient as

$$\nabla f(x) = g(x) = \sum_{j=1}^{m} r_j(x) \nabla r_j(x) \tag{36}$$

$$= J(x)^T F(x) \tag{37}$$

3. A quick search failed to assertain the first paper which fully described BFGS. What appears to be a precursor is discussed in [2] and [1] fully describes the method but provides no citations.

3.1 Convergence Test

The same convergence tests used in unconstrained minimization are used with least squares:

$$\begin{array}{ll} \text{F-Test} & ftol \cdot f(x) \leq f(x) - f(x+p) \\ \text{G-Test} & gtol \leq \|g(x)\|_\infty \end{array}$$

3.2 Levenberg-Marquardt

Levenberg-Marquardt (LM) is a Trust Region based algorithm which was created before Trust Region had been formally been defined [1], [2], [5]. The main innovation proposed by Levenberg [14] is the dampening parameter λ.

$$(J_k^T J_k + \lambda I)p_k = -g_k \tag{38}$$

The dampening parameter enables the solver to handle singular systems. Its value is automatically decreased when the quadratic model is accurate and increased when it is not accurate. Later on Marquardt [15] noted that as λ increased information in $J_k^T J_k$ is used less, slowing convergence as it becomes a steepest descent search. Instead Marquardt proposed the following adjustment:

$$(J_k^T J_k + \lambda \text{diag}(J_k^T J_k))p_k = -g_k \tag{39}$$

This would result in larger steps along the direction with smaller gradient, avoiding slow convergence.

DDogleg's implementation (Algorithm 4) is primarily based upon the description found in [3] but with the ability to choose a mixture of Levenberg's and Marquardt's formulations. Mixing (38) and (39) is advantagous because it allows you to avoid the negatives of either approach. If a partial derivative is zero then Marquardt's forumation will produce a singular matrix. While Levenberg's forumation will always produce a positive-definite matrix as long as λ is greater than zero. The amount of mixing is specified using α. If $\alpha = 1$ then (38) is used while if $\alpha = 0$ then (39) is used. Any value between 0 and 1 will result in a mixture of the two equations.

The classes *FactoryOptimization* and *FactoryOptimizationSparse* provide easy to use functions for constructing specific implementations of Levenberg-Marquardt. As inputs they take in a *ConfigLevenbergMarquardt* and a boolean flag called *robust*. If the robust flag is set to true then a solver based on QR with column pivots is used and if false then it used Cholesky decomposition. The robust variant can handle degenerate matrices found in the Marquardt's formulation. DDogleg does not provide support for solving the least squares formation, i.e. $J_k p_k = -F_k$, due to the increase in computational cost and code complexity having no noticeable improvement in convergence in any situation the author is aware of.

Scaling is done using the same methods described in Section 1.2.

Algorithm 4 Levenberg-Marquardt

1: $k \leftarrow 0$, $\nu \leftarrow 2$
2: $B_k = J_k^T J_k$
3: **while** $k < k_{\max}$ and not $done$ **do**
4: Solve $\left(B_k + \lambda\left(\alpha I + (1-\alpha)\text{diag}(B_k)\right)\right) p_k = -g_k$ ▷ LM Step
5: $\delta_f \leftarrow f(x_k) - f(x_k + p_k)$ ▷ Actual reduction in score
6: $\delta_m \leftarrow m_k(x_k) - m_k(x_k + p_k) = -g_k^T p - \frac{1}{2} p^T J_k^T J_k p$ ▷ Predicted reduction in score
7: $\nu \leftarrow \delta_f / \delta_f$ ▷ Score reduction ratio
8: **if** $\delta_f \geq 0$ **then** ▷ Score get better?
9: $\lambda \leftarrow \lambda \cdot \max(1/3, 1 - (2\nu - 1)^3)$
10: $\nu = 2$
11: $p_{k+1} \leftarrow p_k$
12: $done \leftarrow$ F-Test or G-Test ▷ Convergence testing
13: **else**
14: $\lambda \leftarrow \nu\lambda$ ▷ Emphasize the gradient more
15: $\nu = 2\nu$
16: $k \leftarrow k + 1$

3.3 Trust Region

Everything previously discussed about Trust Region still applies in the Least Squares case with the following variables defined as:

$$g_k = J_k^T F_k \tag{40}$$
$$B_k = J_k^T J_k \tag{41}$$

REFERENCES

[1] S. Wright and J. Nocedal, *Numerical Optimization*, 2nd ed. Springer-Verlag New York, 2006.
[2] R. Fletcher, *Practical methods of optimization*, 2nd ed. John Wiley and Sons Ltd., 1987.
[3] K. Madsen, H. B. Nielsen, and O. Tingleff, "Methods for non-linear least squares problems," Tech. Rep., 2004.
[4] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys (CSUR)*, vol. 23, no. 1, pp. 5–48, 1991.
[5] J. E. Dennis Jr and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*. Society for Industrial and Applied Mathematics (SIAM), 1996.
[6] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustmenta modern synthesis," in *International workshop on vision algorithms*. Springer, 1999, pp. 298–372.
[7] T. A. Davis, *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2006.
[8] P. Abeles, "Efficient java matrix library," http://ejml.org, 2018.
[9] D. S. Watkins, *Fundamentals of matrix computations*, 2nd ed. John Wiley & Sons, 2010.
[10] R. Fletcher, *Practical Methods of Optimization*, 2nd ed. John Wiley & Sons, 1986.
[11] J. J. More and D. J. Thuente, "Line search algorithms with guaranteed sufficient decrease," *ACM Transactions of Mathematical Software*, vol. 20, no. 3, pp. 286–307, 1994.
[12] "Minpack-2," http://ftp.mcs.anl.gov/pub/MINPACK-2/csrch/.
[13] W. C. Davidon, "Variable metric method for minimization," Argonne National Laboratory, Argonne, IL, Tech. Rep. ANL-5990 (revised), 1959.
[14] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, no. 2, pp. 164–168, 1944.
[15] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.