

Parallel Jacobi methods for derivative-free optimization on parallel or distributed processors

I.D. Coope* M.S. Macklem †

10 September 2004

Abstract

New Jacobi-type algorithms are presented for the efficient use of parallel and distributed computing platforms in solving derivative-free optimization problems. The implementations are designed to be fault-tolerant to be applicable to science and engineering problems where occasionally requests for function values may not be met and derivatives are never available. Convergence is usually achieved by introducing an elementary trust region sub-problem at synchronization steps in the algorithm. This has the added advantage of handling negative curvature very conveniently.

Contents

1	Introduction	2
2	Brodie's algorithm	3
3	Parallel Jacobi methods	5
4	A trust region sub-problem	6
5	Fault handling	8
6	Concluding remarks	9

*University of Canterbury, NEW ZEALAND. <mailto:ian.coope@canterbury.ac.nz>

†Simon Fraser University, Vancouver, CANADA

1 Introduction

In [1], Brodlie proposes a conjugate-direction algorithm, which exploits the properties of quadratic functions to generate approximations to the eigensystem of the Hessian matrix of second derivatives of f . The eigenvectors are special cases of conjugate vectors which also form a mutually orthonormal set. Linear independence of the search directions is therefore guaranteed. If the function to be minimized is the quadratic

$$Q(x) = \frac{1}{2}x^T Gx + x^T g + c \quad (1)$$

then $G = \nabla^2 Q(x)$ is the symmetric Hessian matrix of Q . If G has spectral decomposition $G = VDV^T$ where $V = [v_1, v_2, \dots, v_n]$ is an orthogonal matrix of normalized eigenvectors of G and $D = \text{diag}(d_1, d_2, \dots, d_n)$ is the diagonal matrix of corresponding eigenvalues then the vectors $\{v_j\}_1^n$ satisfy both conjugacy and orthogonality conditions

$$v_i^T Gv_j = 0 = v_i^T v_j, \quad i \neq j.$$

In addition the normalization provides the conditions

$$v_j^T Gv_j = d_j, \quad v_j^T v_j = 1, \quad j = 1, 2, \dots, n,$$

which shows that d_j is the second directional derivative of $Q(x)$ along the normalized direction v_j . When G is positive definite the unique minimizer of (1) is $x^* = -G^{-1}g$ which can also be written

$$x^* = -VD^{-1}V^T g = \sum_{j=1}^n \alpha_j v_j,$$

where $\alpha_j = -(v_j^T g)/d_j$. In fact if $x^{(k)}$ is any point in \mathbb{R}^n

$$x^* = x^{(k)} + \sum_{j=1}^n \beta_j v_j, \quad (2)$$

where $\beta_j = -v_j^T \nabla Q(x^{(k)})/d_j$. This illustrates the well-known result that if a set of G -conjugate vectors is known then the quadratic function (1) is minimized by searching along each of the conjugate vectors v_j for the value of β_j which minimizes $Q(x(\beta))$ along the line $x(\beta) = x^{(k)} + \beta v_j$.

Brodlie [1] exploits the above properties of quadratic functions to design an algorithm for use on general functions $f(x)$. The basic idea is to generate successive approximations $V^{(k)}, k = 1, 2, \dots$ to the eigenvectors of $\nabla^2 f(x^{(k)})$

using information obtained from performing line searches along search vectors defined by the columns of $V^{(k)}$. The columns of $V^{(k)}$ are revised in a manner analogous to the cyclic Jacobi method for solving symmetric matrix eigenvalue problems.

We present a parallel variant of Brodlie’s method which includes a trust-region subproblem at synchronization steps throughout the algorithm. A brief outline of Brodlie’s algorithm is given in Section 2 and its relationship to serial Jacobi methods for calculating eigensystems is illustrated. In Section 3 it is shown how parallel Jacobi methods for solving symmetric eigenvalue problems can also be exploited in unconstrained optimization algorithms. A potential difficulty in parallelization lies in the synchronization steps which are required to guarantee descent. To avoid this difficulty, in Section 4 it is shown that replacing the line search sub-problems by trust-region sub-problems in appropriate sub-spaces can provide a useful alternative approach which also conveniently handles problems of negative curvature. Section 5 describes how the trust-region approach also resolves issues surrounding unsuccessful objective function requests, which can crash many optimization algorithms. Some conclusions and extensions are discussed in Section 6.

2 Brodlie’s algorithm

In this section Brodlie’s method [1] is outlined, and its relationship to serial Jacobi methods for solving symmetric eigenvalue problems is described. Notice that the description here differs slightly from [1] in order to facilitate the extensions presented in the following sections.

Algorithm 2.1:

1. **Initialization:** For a given function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, starting point $x^{(1)}$, an initial set of search directions $\{v_j^{(1)}\}_{j=1}^n$.
2. **Outer iteration: Disjoint search directions:** For a fixed iteration i , split the search directions into sets of disjoint pairs $(v_p^{(i)}, v_q^{(i)})$. The outer iteration will cycle through the disjoint pairs, performing the inner iteration for each pair.
3. **Inner iteration:** Let $(v_p^{(i)}, v_q^{(i)})$ be a fixed pair of search directions.
 - (a) **Quadratic approximation:** Consider the function

$$\phi^{(i)}(\lambda, \mu) = f(x^{(i)} + \lambda v_p^{(i)} + \mu v_q^{(i)}),$$

the restriction of f to $\text{span}(v_p^{(i)}, v_q^{(i)})$. Approximate this function by the quadratic function

$$\hat{\phi}^{(i)}(\lambda, \mu) = \frac{1}{2}a\lambda^2 + b\lambda\mu + \frac{1}{2}c\mu^2 + d\lambda + e\mu + k \quad (3)$$

The constants a – e may be determined by five, carefully chosen, new evaluations of $\phi^{(i)}(\lambda, \mu)$ assuming that $k = \phi^{(i)}(0, 0) = f(x^{(i)})$.

- (b) **Update minimizer:** Perform further function evaluations if desired and define $x^{(i+1)}$ as the point with least function value of all points considered.
- (c) **Update search directions:** Define $\theta \in (-\frac{\pi}{4}, \frac{\pi}{4}]$ by

$$\tan 2\theta = \frac{-2b}{a - c},$$

with a , b and c the quadratic coefficients of $\hat{\phi}^{(i)}(\lambda, \mu)$. Update the search directions by

$$\begin{aligned} v_p^{(i+1)} &= \cos \theta v_p^{(i)} - \sin \theta v_q^{(i)} \\ v_q^{(i+1)} &= \sin \theta v_p^{(i)} + \cos \theta v_q^{(i)} \\ v_r^{(i+1)} &= v_r^{(i)} \text{ for } r \neq p, q. \end{aligned}$$

Some observations about this algorithm:

- **Initialization:** For any initial set of orthonormal directions, orthonormality is maintained in step 3(c). Brodlie [1] proved that if f is a quadratic function with positive definite Hessian G , if the initial search directions are chosen to be the coordinate directions, and if the choice of disjoint pairs follows the same sequence as a cyclic-Jacobi method, then his algorithm exactly parallels a cyclic Jacobi method for solving the symmetric eigenvalue problem for the matrix G .
- **Outer iteration:** Brodlie recommended that the pairs (p, q) be chosen in a cyclic manner so that each pair occurs exactly once in a cycle of $\frac{1}{2}n(n-1)$ rotations so that as much as possible the occurrence of each search direction is well-spaced throughout the cycle.
- **Inner iteration:** Brodlie observes that the line searches are not required to be accurate, and specifically that the improvement generally arises from the quadratic fitting step. He describes the process of choosing points at which to evaluate f using ideas from Powell [6].

3 Parallel Jacobi methods

Jacobi methods for calculating eigensystems of symmetric matrices are particularly suited to parallelization because it is possible to carry out $\lfloor n/2 \rfloor$ rotations simultaneously if sufficient processors are available. However, care has to be taken in such algorithms with respect to updating the approximation to the minimizer (Algorithm 2.1, Step 3b). These updates should be done only at synchronization steps, as excessive work on a single processor in this step can be wasted if another processor finds a better point more quickly. We require a method that combines the work of individual processors efficiently, as in the following algorithm.

Algorithm 3.1

1. **Initialization:** For a given function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, initialize starting point x_0 , a set of search directions $\{v_j\}_{j=1}^n$ and an initial meshsize parameter h . Set $x_c = x_0$; we use x_c to record the current best approximation to the minimizer.
2. **Outer iteration: Disjoint search directions:** As with the serial implementation, split the search directions into disjoint pairs (v_p, v_q) , and cycle through the disjoint pairs. We send each pair to different child-processors, getting updated search directions and *predictions* of optimal points from each pair, which will be combined on the parent-processor once all pairs have been evaluated.
3. **Inner iteration: Calculations performed on a fixed processor:**
 - (a) For a fixed pair of search directions (v_p, v_q) , evaluate f at a sufficient number of points to determine the coefficients of the 2-D quadratic model (3). The current meshsize h is used to ensure these points are appropriately spaced.
 - (b) Unlike the serial version - do not update the minimizer at this point.
 - (c) Update directions v_p and v_q as with the serial version and update the coefficients $a-e$ of the quadratic model (3) to reflect the change of basis vectors. Return the new values to the root-level procedure together with the least function value (and its coordinates).
4. **Combination step.** Performed by the parent processor after the child processors have completed all inner iterations, and thus is structurally part of the outer iteration:

- Combine all (disjoint) pairs of search directions to get

$$\hat{x} = x_c + \sum_{j=1}^n \beta_j \hat{v}_j.$$

The coefficients $\{\beta_j\}_1^n$ are chosen to minimize (approximately) the quadratic model along each updated direction \hat{v}_j using information returned by the child processors.

- Evaluate $f(\hat{x})$. If $f(\hat{x}) < f(x_c)$ (the current lowest point) then set $x_c = \hat{x}$. Otherwise, set x_c to the best point found across all inner-iterations (across all pairs of search directions used in current inner-iteration). If no lower point has been found reduce the meshsize h . Return to Step (2).

Observations

- **Outer iteration:** As with Algorithm 2.1, the pairs (p, q) can be chosen in a cyclic manner so that each pair occurs exactly once in a cycle of $\frac{1}{2}n(n-1)$ rotations so that as much as possible the occurrence of each search direction is well-spaced throughout the cycle.
- **Inner iteration:** Although line searches may be used all that is required is at least five extra function values, no three of which are evaluated along the same line.
- **Combination step:** Note that, since the meshsize parameter is updated during the combination step, a minimum threshold value for this parameter may be used to terminate the iterations. Also, notice that suitable values for the β_j are more conveniently calculated by solving a suitable trust region sub-problem which takes care of cases where negative curvature is detected. We consider this trust-region problem in the next section.

4 A trust region sub-problem

In this section the problem of determining suitable values for the scalars $\{\beta_j\}_1^n$ in Algorithm 3.1 is solved by letting u be the vector whose components are $\beta_j, j = 1, 2, \dots, n$ and then solving an appropriate trust-region sub-problem.

The search directions $\{v_j\}_1^n$ together with estimates of the first and second directional derivatives $d_j \approx v_j^T [\nabla^2 f(x)] v_j$, $(g_V)_j \approx v_j^T \nabla f(x)$ implicitly

provide a quadratic approximation to $f(x)$. Specifically, in the notation of Section 1, if $G = VDV^T$ then the Taylor series approximation

$$f(x + s) \approx f(x) + s^T \nabla f(x) + \frac{1}{2} s^T [\nabla^2 f(x)] s$$

can be replaced by the approximating function

$$f(x + s) \approx f(x) + s^T g + \frac{1}{2} s^T G s$$

Therefore, in this section we consider the problem of choosing a suitable step, s , for improving the estimate of the minimizer, x_c , after control has been returned to the root-processor with updated estimates of V , D and g_V . Note that the components of g_V and the entries of the diagonal matrix D are available as the coefficients of the quadratic models returned by each child processor in the inner iteration of Algorithm 3.1.

A standard quadratic trust region problem (see, for example, [7]) is:

$$\min_s \{ \frac{1}{2} s^T G s + s^T g : \|s\| - \Delta \leq \rho \Delta \} \quad (4)$$

where $\|(\cdot)\|$ denotes the vector 2-norm, Δ denotes the trust-region radius and ρ is a relative tolerance (typically $\rho = 0.1$ is chosen). If $V^T G V = D$ is a diagonal matrix for an orthogonal matrix V^1 then writing $s = V u$, $g_V = V^T g$, an equivalent trust region problem is:

$$\min_u \{ \frac{1}{2} u^T D u + u^T g_V : \|u\| - \Delta \leq \rho \Delta \} \quad (5)$$

which is trivially and efficiently solved, even if D has some negative entries². The solution to problem (4) is then easily recovered in $\mathcal{O}(n^2)$ flops since $s = V u$.

In the context of the parallel Jacobi minimization algorithm outlined in Section 3, the vector g_V is replaced by estimates of the directional derivatives and the diagonal matrix D is replaced by the diagonal matrix of estimates of second directional derivatives, which are also the estimates of the eigenvalues of $\nabla^2 f(x)$. Suppose, for example, that v_p and v_q are selected in an inner iteration of Algorithm 3.1 (Step 3); then the quadratic approximation built at Step 3(a) can be used to construct the following estimates at $x = x_c$:

$$d_p = \hat{\phi}_{\lambda\lambda}(0, 0), \quad d_q = \hat{\phi}_{\mu\mu}(0, 0), \quad (g_V)_p = \hat{\phi}_{\lambda}(0, 0), \quad (g_V)_q = \hat{\phi}_{\mu}(0, 0). \quad (6)$$

These estimates can then be updated to reflect the change of basis when the search directions are updated in Step 3(c). Recall that in the parallel

¹For example, from using the MATLAB command `[V,D]=eig(G)`.

²For example, Algorithm 7.7.1 in [7] is suitable for solving problem (5) in $\mathcal{O}(n)$ flops.

implementations described in Section 3, x_c changes only at synchronization steps, and thus the estimates (6) remain valid until there is a change in x_c . Also note that the purpose of the rotation is to make $c = 0$ in the quadratic model corresponding to the revised basis.

The trust-region problem can therefore be used to replace the line searches or two-dimensional sub-space searches for a lower point. Moreover, a suitable trust region sub-problem can be set up at *any* convenient synchronization point. As a simple illustration, suppose that $n \geq 6$ and there are only three child processors each of which has been tasked with handling an inner iteration of Step 3 of Algorithm 3.1 Let (p_1, q_1) denote the indices of the disjoint pair passed to processor 1, $(p_2, q_2) \dots$ etc. Then when the three child processors return control to the root processor the trust-region problem to be solved in Step 4 is:

$$\min_{\tilde{u}} \left\{ \frac{1}{2} \tilde{u}^T \tilde{D} \tilde{u} + \tilde{u}^T g_{\tilde{V}} : \left| \|\tilde{u}\| - \Delta \right| \leq \rho \Delta \right\} \quad (7)$$

where $\tilde{V} = [v_{p_1}, v_{q_1}, v_{p_2}, v_{q_2}, v_{p_3}, v_{q_3}]$ and $\tilde{D} = \text{diag}(d_{p_1}, d_{q_1}, d_{p_2}, d_{q_2}, d_{p_3}, d_{q_3})$ and $\tilde{u} \in \mathbb{R}^6$. Then f is evaluated at $\tilde{x} + \tilde{V} \tilde{u}$ and x_c can now be updated as the lowest point found so far. The trust region radius can then be modified as usual, as outlined in [7]; this then completes the synchronization step and the child processors can be re-tasked with the next choice of disjoint pairs.

This approach provides great flexibility in the way multi-processors can be used. As a further illustration, there is no longer a need to limit individual processors to constructing a two-dimensional model. It may be useful to consider 3-D (or higher) models where the eigensystem of a three dimensional subspace is constructed using 9 extra function evaluations instead of 5 for the 2-D model. Alternatively, the trust-region problem can be omitted at some synchronization steps awaiting a better approximation to G (through V and D). If there are 50 variables and five processors then 25 rotations are required for a minor series (using Brodlie's terminology). Thus tasking each child processor five times before solving the trust-region problem may be an appropriate choice since then each direction has been 'rotated' once only.

5 Fault handling

The trust-region approach described in Section 4 is also useful for handling faults. If a child processor is unable to handle a request for a function evaluation or a function call returns NaN or Inf (IEEE floating point arithmetic) or even if the child processor itself should fail for some reason then the root processor just omits that component from the trust-region problem. If a rotation

is not completed because of one or more unavailable function evaluations the child processor may still be able to return some useful partial information that can be incorporated into the trust-region problem. For example, there may be enough successful evaluations to provide d_p and $(g_V)_p$ but not the corresponding terms for index q . In this case a rotation cannot be computed but the information on direction v_p can be passed back to the root processor and incorporated in problem (7). The missed rotation can subsequently be re-tasked to the same or a different child processor after the combination step (Step 4) of Algorithm 3.1 has been completed.

6 Concluding remarks

The work of the previous sections shows that there are many possibilities for constructing Jacobi-type algorithms for derivative free optimization that are capable of exploiting parallel processing environments. Some of the possibilities mentioned are currently under further investigation both numerically and theoretically and will be reported on in due course.

A convergence analysis cannot properly be addressed in such a short paper as this but it is worth noting that questions of convergence can be analyzed either by enforcing an appropriate trust-region framework [7, Section 9.4] or by imposing some extra requirements on the choice of points used in the inner iterations, together with a careful choice of rule for modifying the mesh size parameter h , which is more in line with the framework described in [3, 4]. It is not necessary for V to converge to the matrix of eigenvectors of $\nabla^2 f$ in order to establish convergence to a stationary point but the use of second order information through V and D may provide a mechanism for establishing convergence to a stationary point satisfying second order necessary conditions.

Algorithm 3.1 could also be implemented in serial mode. In this case each new rotation step is handled serially by one processor. The current best approximation to the minimizer is then updated only at ‘synchronization steps’ which are performed after a set number of rotations. An obvious choice is after a series of ‘minor iterations’ where every search direction has been used just once.

Finally we note that when n is odd a special one-dimensional iteration may replace a choice of disjoint pairs as described by Brodlie[1]. There is no difficulty in incorporating this into the framework of Algorithm 3.1. Numerical experiments are in progress and will be reported at a later date[5].

References

- [1] K. W. Brodlie. A new method for unconstrained minimization without evaluating derivatives. *J. Inst. Math. Appl.*, 15, 385–396, 1975
- [2] K. W. Brodlie and M. J. D. Powell. On the convergence of cyclic Jacobi methods. *J. Inst. Math. Appl.*, 15, 279–287, 1975
- [3] I. D. Coope, C. J. Price. Frame-based methods for unconstrained optimization. *Journal of Optimization Theory & Applications*, 107, 261–274, 2000.
- [4] I. D. Coope, C. J. Price. On the convergence of grid-based methods for unconstrained optimization. *S.I.A.M. Journal on Optimization*, 11, 859–869, 2001.
- [5] M. S. Macklem. PhD Thesis. *Simon Fraser University*, forthcoming.
- [6] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, 7, 155–162, 1964
- [7] A. R. Conn, N. I. M. Gould and Ph. L. Toint. *Trust-Region Methods*. MPS–SIAM Series on Optimization, 2000.