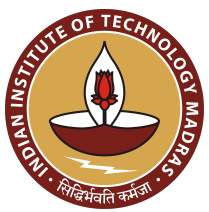
Large-Scale Unconstrained Optimization

Saurav Samantaray

Department of Mathematics

Indian Institute of Technology Madras

March 12, 2024



Introduction

- Many applications give rise to unconstrained optimization problems with thousands or millions of variables.
- Problems of this size can be solved efficiently only if the storage and computational costs of the optimization algorithm can be kept at a tolerable level.
- A diverse collection of large-scale optimization methods has been developed to achieve this goal, each being particularly effective for certain problem types.
- Some of these methods are straightforward adaptations of the methods described until now.
- Other approaches are modifications of these basic methods that allow approximate steps to be calculated at lower cost in computation and storage.

Introduction

- The non-linear conjugate gradient methods of can be applied to large problems without modification, owing to its minimal storage demands and its reliance on only first-order derivative information.
- The Newton method in both line search and trust-region algorithms require matrix factorizations of the Hessian matrices.
- High quality software implementations are available, which are based on factorizations that can be carried out using elimination techniques.
- Newton methods are plagued with issues related to computational cost and memory requirements of these factorization methods.
- If the Hessian matrix can be formed explicitly, with the above problems sorted they constitute an effective approach for solving such problems.

Introduction

- Often, however, the cost of factoring the Hessian is prohibitive, and it is preferable to compute approximations to the Newton step using iterative linear algebra techniques.
- Inexact Newton methods that use these techniques, in both line search and trust-region frameworks have attractive global convergence properties and may be super-linearly convergent for suitable choices of parameters.
- There are variants of the quasi-Newton approach, which use Hessian approximations that can be stored compactly by using just a few vectors of length n.
- These methods are fairly robust, inexpensive, and easy to implement, but they do not converge rapidly.

Inexact Newton Methods

- The basic Newton step p_k^N is obtained by solving the symmetric $n \times n$ linear system

$$\nabla^2 f_k p_k^N = -\nabla f_k. \tag{1}$$

- p_k^N can be obtained by solving the above equation (1) approximately, via inexpensive iterative solvers.
- For example the conjugate gradient (CG) method can be employed to get p_k^N.
- Both line search and trust region approaches can be derived based on this approximation, which falls in the general family of inexact Newton methods.
- In addition, we can implement these methods in a Hessian-free manner, so that the Hessian $\nabla^2 f_k$ need not be calculated or stored explicitly at all.

LOCAL CONVERGENCE OF INEXACT NEWTON METHODS

- Consider the residual for the equation (1) as:

$$r_k = \nabla^2 f_k p_k + \nabla f_k \tag{2}$$

where p_k is the inexact Newton step.

- The CG iterations are terminated when

$$||r_k|| \leq \eta_k ||\nabla f_k||, \tag{3}$$

where the sequence $\{\eta_k\}$ (with $0 < \eta_k < 1$ for all k) is called the forcing sequence.

LOCAL CONVERGENCE OF INEXACT NEWTON METHODS

Theorem

Suppose that $\nabla^2 f(x)$ exists and is continuous in a neighbourhood of a minimizer x^*, with $\nabla^2 f(x^*)$ is positive definite. Consider the iteration $x_{k+1} = x_k + p_k$ where p_k satisfies (3), and assume that $\eta_k \leq \eta$ for some constant $\eta \in [0, 1)$. Then, if the starting point x_0 is sufficiently near x^*, the sequence $\{x_k\}$ converges to x^* and satisfies

$$||\nabla^2 f(x^*)(x_{k+1} - x^*)|| \leq \hat{\eta}||\nabla^2 f(x*)(x_k - x^*)||, \qquad (4)$$

for some constant $\hat{\eta}$ with $\eta < \hat{\eta} < 1$.

LINE SEARCH NEWTON–CG METHOD

- In the line search Newton–CG method, also known as the truncated Newton method, the search direction is computed by applying the CG method to the Newton equations:

$$\nabla^2 f_k p_k^N = -\nabla f_k;$$

- and attempt to satisfy a termination test of the form

$$||r_k|| \leq \eta_k ||\nabla f_k||,$$

- The CG method is designed to solve positive definite systems.
- However, the Hessian $\nabla^2 f_k$ may have negative eigenvalues when x_k is not close to a solution.
- The CG iteration is terminated as soon as a direction of negative curvature is generated.

LINE SEARCH NEWTON–CG METHOD

- This adaptation of the CG method produces a search direction p_k that is a descent direction.
- Moreover, the adaptation guarantees that the fast convergence rate of the pure Newton method is preserved, provided that the step length $\alpha_k = 1$ is used whenever it satisfies the acceptance criteria.
- For purposes of this algorithm rewrite the linear system (1) in the form

$$B_k p = -\nabla f_k \tag{5}$$

  where B_k represents $\nabla^2 f_k$.
- For the inner CG iteration, denote the search direction by d_j and the sequence of iterates that it generates by z_j.

LINE SEARCH NEWTON–CG METHOD

- When B_k is positive definite, the inner iteration sequence $\{z_j\}$ will converge to the Newton step p_k^N that solves (5).
- At each major iteration, a tolerance ε_k that specifies the required accuracy of the computed solution, is prescribed
- For concreteness the forcing sequence is chosen to be $\eta_k = \min(0.5, \sqrt{||\nabla f_k||})$ to obtain a super-linear convergence rate (one may choose differently as well).

Line Search Newton–CG

Algorithm

Given initial point x_0;
for $k = 0, 1, 2, \ldots$
    Define tolerance $\varepsilon = \min(0.5, \sqrt{||\nabla f_k||})||\nabla f_k||$;
    Set $z_0 = 0$, $r_0 = \nabla f_k$, $d_0 = -r_0 = -\nabla f_k$;
    for $j = 0, 1, 2, \ldots$
        if $d_j^T B_k d_j \leq 0$
            if $j = 0$
                return $p_k = -\nabla f_k$;
            else
                return $p_k = z_j$;
        set $\alpha_j = r_j^T r_j / d_j^T B_k d_j$;
        Set $z_{j+1} = z_j + \alpha_j d_j$;
        Set $r_{j+1} = r_j + \alpha_j B_k d_j$;
        if $||r_{j+1}|| < \varepsilon_k$
            return $p_k = z_{j+1}$;
        Set $\beta_{j+1} = r_{j+1}^T r_{j+1} / r_j^T r_j$;
        Set $d_{j+1} = -r_{j+1} + \beta_{j+1} d_j$;
    end (for)
    Set $x_{k+1} = x_k + \alpha_k p_k$, where α_k satisfies the Wolfe, Goldstein, or
        Armijo backtracking conditions (using $\alpha_k = 1$ if possible);
end

LINE SEARCH NEWTON–CG METHOD

- The main differences between the inner loop of the above algorithm and the original CG are that the specific starting point $z_0 = 0$ is used;
- and the use of a positive tolerance ε_k allows the CG iterations to terminate at an inexact solution;
- and the negative curvature test $d_j^T B_k d_j \leq 0$ ensures that p_k is a descent direction for f at x_k.
- If negative curvature is detected on the first inner iteration $j = 0$, the returned direction $p_k = -\nabla f_k$ is both a descent direction and a direction of non-positive curvature for f at x_k.

LINE SEARCH NEWTON–CG METHOD

- When the user cannot easily supply code to calculate second derivatives, or where the Hessian requires too much storage automatic differentiation and finite differencing techniques can be used to calculate these Hessian–vector products.
- Methods of this type are known as Hessian-free Newton methods.
- In the finite-differencing technique, we use the approximation

$$\nabla^2 f_k d \approx \frac{\nabla f(x_k + hd) - \nabla f(x_k)}{h}, \tag{6}$$

  for some small differencing interval h.
- It is easy to prove that the accuracy of this approximation is $\mathcal{O}(h)$;
- The price we pay for bypassing the computation of the Hessian is one new gradient evaluation per CG iteration.

TRUST-REGION NEWTON–CG METHOD

- We discussed approache(s) for finding an approximate solution of the trust-region subproblem that produce improvements on the Cauchy point.
- We will define a modified CG algorithm for solving the sub-problem with these properties.
- Consider the trust-region sub-problem:

$$\min_{p \in \mathbb{R}^n} m_k(p) = f_k + (\nabla f_k)^T p + \frac{1}{2} p^T B_k p \quad \text{s.t. } ||p|| \leq \Delta_k \quad (7)$$

  where $B_k = \nabla^2 f_k$.
- To specify the algorithm by Steihaug, we use d_j to denote the search directions of this modified CG iteration and z_j to denote the sequence of iterates that it generates.

CG–Steihaug

Algorithm

Given tolerance $\varepsilon_k > 0$;
Set $z_0 = 0$, $r_0 = \nabla f_k$, $d_0 = -r_0 = -\nabla f_k$;
if $||r_0|| < \varepsilon_k$
    return $p_k = z_0 = 0$;
for $j = 0, 1, 2, \ldots$
    if $d_j^T B_k d_j \leq 0$
        Find τ such that $p_k = z_j + \tau d_j$ minimizes $m_k(p_k)$, and
          satisfies $||p_k|| \leq \Delta_k$;
        return p_k;
    set $\alpha_j = r_j^T r_j / d_j^T B_k d_j$;
    Set $z_{j+1} = z_j + \alpha_j d_j$;
    if $||z_{j+1}|| \geq \Delta_k$
        Find $\tau \geq 0$ such that $p_k = z_j + \tau d_j$ satisfies $||p_k|| = \Delta_k$;
        return p_k;
    Set $r_{j+1} = r_j + \alpha_j B_k d_j$;
    if $||r_{j+1}|| < \varepsilon_k$
        return $p_k = z_{j+1}$;
    Set $\beta_{j+1} = r_{j+1}^T r_{j+1} / r_j^T r_j$;
    Set $d_{j+1} = -r_{j+1} + \beta_{j+1} d_j$;
    end (for)

CG–Steihaug

- The first if statement inside the loop stops the method if its current search direction d_j is a direction of non-positive curvature along B_k,
- while the second if statement inside the loop causes termination if z_{j+1} violates the trust-region bound.
- In both cases, the method returns the step p_k obtained by intersecting the current search direction with the trust-region boundary.
- The choice of the tolerance ε_k at each call to the algorithm is important in keeping the overall cost of the trust-region Newton–CG method low.
- Near a well-behaved solution x^*, the trust-region bound becomes inactive, and the method reduces to the inexact Newton method.

ALGORITHMS FOR PARTIALLY SEPARABLE FUNCTIONS

- In a separable unconstrained optimization problem, the objective function can be decomposed into a sum of simpler functions that can be optimized independently.
- For example for

$$f(x) = f_1(x_1, x_3) + f_2(x_2, x_4, x_6) + f_3(x_5)$$

  one can find the optimal value of x by minimizing each function f_i, $i = 1, 2, 3$, independently,
- since no variable appears in more than one function.
- The cost of performing m lower-dimensional optimizations is much less in general than the cost of optimizing an n-dimensional function.

ALGORITHMS FOR PARTIALLY SEPARABLE FUNCTIONS

- In many large problems the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is not separable, but it can still be written as the sum of simpler functions, known as element functions.
- Each element function has the property that it is unaffected when we move along a large number of linearly independent directions.
- If this property holds, we say that f is partially separable.
- All functions whose Hessians are sparse are partially separable,but so are many functions whose Hessian is not sparse.
- Partial separability allows for economical problem representation, efficient automatic differentiation, and effective quasi-Newton updating.

ALGORITHMS FOR PARTIALLY SEPARABLE FUNCTIONS

- The simplest form of partial separability arises when the objective function can be written as

$$f(x) = \sum_{i=1}^{ne} f_i(x) \tag{8}$$

  where each of the element functions f_i depends on only a few components of x.
- the gradients ∇f_i and Hessians $\nabla^2 f_i$ of each element function contain just a few non-zeros.
- By differentiating (8) we have

$$\nabla f(x) = \sum_{i}^{ne} \nabla f_i(x) \qquad \nabla^2 f(x) = \sum_{i}^{ne} \nabla^2 f_i(x)$$

ALGORITHMS FOR PARTIALLY SEPARABLE FUNCTIONS

- A natural question that arises is whether it is more effective to maintain quasi-Newton approximations to each of the element Hessians $\nabla^2 f_i(x)$ separately, rather than approximating the entire Hessian $\nabla^2 f(x)$.
- To answer the above question let us consider the following example. Consider the objective function:

$$\begin{aligned} f(x) &= (x_1 - x_3^2)^2 + (x_2 - x_4^2)^2 + (x_3 - x_2^2)^2 + (x_4 - x_1^2)^2 \\ &= f_1(x) + f_2(x) + f_3(x) + f_4(x). \end{aligned} \tag{9}$$

- The Hessians of the element functions f_i are 4×4 sparse, singular matrices with 4 non-zero entries.

ALGORITHMS FOR PARTIALLY SEPARABLE FUNCTIONS

- Consider the element function f_1; all other element function have exactly the same form.
- Even though f_1 is formally a function of all components of x, it depends only on x_1 and x_3, which we call the element variables for f_1.
- Assemble the element variables into a vector, say $x_{[1]}$, that is,

$$x_{[1]} = \begin{bmatrix} x_1 \\ x_3 \end{bmatrix}$$

- note that

$$x_{[1]} = U_1 x \qquad \text{with} \qquad U_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

ALGORITHMS FOR PARTIALLY SEPARABLE FUNCTIONS

- Define the function ϕ_1 by:

$$\phi_1(z_1, z_2) = (z_1 - z_2^2)^2,$$

- then one can write the following

$$f_1(x) = \phi_1(U_1 x).$$

- By applying the chain rule to this representation, we obtain

$$\nabla f_1(x) = U_1^T \nabla \phi_1(U_1 x), \qquad \nabla^2 f_1(x) = U_1^T \nabla^2 \phi_1(U_1 x) U_1$$

ALGORITHMS FOR PARTIALLY SEPARABLE FUNCTIONS

- Therefore we have

$$\nabla^2 \phi_1(U_1 x) = \begin{bmatrix} 2 & -4x_3 \\ -4x_3 & 12x_3^2 - 4x_1 \end{bmatrix}, \qquad \nabla^2 f_1(x) = \begin{bmatrix} 2 & 0 & -4x_3 & 0 \\ 0 & 0 & 0 & 0 \\ -4x_3 & 0 & 12x_3^2 - 4x_1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- The matrix U_1, known as a compactifying matrix, allows us tomap the derivative information for the low-dimensional function ϕ_1 into the derivative information for the element function f_1.
- The key idea: Instead of maintaining a quasi-Newton approximation to $\nabla^2 f_1$, maintain a 2×2 quasi-Newton approximation $B_{[1]}$ of $\nabla^2 \phi_1$ and use the chain rule relation to transform it into a quasi-Newton approximation to $\nabla^2 f_1$.

ALGORITHMS FOR PARTIALLY SEPARABLE FUNCTIONS

- To update $B_{[1]}$ after a typical step from x to x^+, record the information and use BFGS or SR1 updating to obtain the new approximation $B_{[1]}^+$.

$$s_{[1]} = x_{[1]}^+ - x_{[1]}, \qquad y_{[1]} = \nabla\phi_1(x_{[1]}^+) - \nabla\phi_1(x_{[1]}), \tag{10}$$

- We therefore update small, dense quasi-Newton approximations with the property

$$B_{[1]} \approx \nabla^2\phi_1(U_1 x) = \nabla^2\phi_1(x_{[1]}) \tag{11}$$

- To obtain an approximation of the element Hessian $\nabla^2 f_1$ we use the transformation suggested by the chain rule that is,

$$\nabla_1^2(x) \approx U_1^T B_{[1]} U_1. \tag{12}$$

ALGORITHMS FOR PARTIALLY SEPARABLE FUNCTIONS

- This operation has the effect of mapping the elements of $B_{[1]}$ to the correct positions in the full $n \times n$ Hessian approximation.
- The full objective function can now be written as

$$f(x) = \sum_{i=1}^{ne} \phi_i(U_i x) \tag{13}$$

and we maintain a quasi-Newton approximation $B_{[i]}$ for each of the functions ϕ_i .

- To obtain a complete approximation to the full Hessian $\nabla^2 f$, we simply sum the element Hessian approximations as follows:

$$B = \sum_{i=1}^{ne} U_i^T B_{[i]} U_i \tag{14}$$