Topics in Optimization

Karen Glocer

Updated: April 27, 2009
Unconstrained optimization

You want to minimize a function $f(x)$. Notation:

- $f_k$ - value of objective function at point $x_k$
- $\nabla f_k$ - gradient of $f$ at point $x_k$
- $W_k$ - Hessian of $f$ at point $x_k$

Newton’s Method

1. Choose an initial starting point $x_0$, tolerance $\epsilon$
2. for $k = 0, 1, \ldots$
   1. evaluate $f_k$, $\nabla f_k$, $W_k$
   2. Find search direction $p = -W_k^{-1}\nabla f_k$
   3. Find step size $\alpha$ via line search
   4. $x_{k+1} = x_k + \alpha p$
   5. if $\max(|p|) \leq \epsilon$, stop with approximate solution $x_{k+1}$
Illustration of quadratic approximation

Value for $\eta=10$

Dual

Taylor

$w_1$

Updated: April 27, 2009
Potential problems with Newton’s method

Value for $\eta = 30$

Dual

Taylor

$w_1$

Karen Glocer ()

Topics in Optimization

Updated: April 27, 2009
Constrained Optimization

You still want to minimize a function $f_0(x)$, but now there are constraints.

$$\min_{x} f_0(x)$$  \hspace{1cm} (1)

subject to $f_i(x) \leq 0$, $i = 1 \ldots m$

$g_i(x) = 0$, $i = 1 \ldots n$.

- **constraint**: restrictions on the value of $x$: e.g. $0 \leq x \leq 1$.
- **feasible set**: the set of all points $x$ that satisfy the constraints.
- The best $x$ in the unconstrained problem may not satisfy the constraints.
Quadratic Approximation of Problem

- Newton’s method, $p = W_k^{-1} \nabla f_0(x_k)$
- In constrained optimization, $p$ is found by solving the following quadratic program:

\[\begin{align*}
\text{minimize} \quad & \frac{1}{2} p^T W_k p + \nabla f_0^T p \\
\text{subject to} \quad & \nabla f_i(x_k)^T p + f_i(x_k) \leq 0, \quad i = 1 \ldots m \\
& \nabla g_i(x_k)^T p + g_i(x_k) = 0, \quad i = 1 \ldots n.
\end{align*}\]
Algorithm for solving Constrained Problem

The difference between this algorithm and the classical Newton method is that now we have to solve a quadratic program to find the search direction $p$.

### Sequential Quadratic Programming (SQP)

1. Choose an initial starting point $x_0$, tolerance $\epsilon$
2. for $k = 0, 1, \ldots$
   1. evaluate $f_k$, $\nabla f_k$, $W_k$
   2. Find search direction $p$ by solving (2)
   3. Find step size $\alpha$ via line search
   4. $x_{k+1} = x_k + \alpha p$
   5. if $\max(|p|) \leq \epsilon$, stop with approximate solution $x_{k+1}$
Quasi-Newton Method for unconstrained optimization

- Alternative to Newton’s method that is faster and more robust.
- Quasi-Newton methods use an approximate Hessian.
- Most popular quasi-Newton method: BFGS

**Broyden Fletcher Goldfarb Shanno (BFGS)**

- Newton’s method requires the inverse of the Hessian
- For an \( N \times N \) matrix, inversion is \( O(N^3) \)
- BFGS approximates the inverse Hessian. (No inversion needed)
- BFGS is more numerically stable than Newton:
  - Newton’s method can behave badly if given a bad starting point
  - BFGS is more robust to bad starting points
The BFGS Algorithm

1. Choose an initial starting point \((x_0)\), tolerance \(\epsilon > 0\), inverse Hessian approximation \(H_0\).

2. \(k = 0\)

3. while \(\|\nabla f_k\| > \epsilon\)
   
   1. Find search direction \(p = -H_k \nabla f_k\)
   
   2. Find step size \(\alpha\) via line search
   
   3. \(x_{k+1} = x_k + \alpha p\)
   
   4. \(s_k = x_{k+1} - x_k\) and \(y_k = \nabla f_{k+1} - \nabla f_k\)
   
   5. \(H_{k+1} = (I - \rho_k s_k y_k^T)H_k(I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T\), where \(\rho_k = 1/(y_k^T s_k)\).
   
   6. \(k = k + 1\)

4. return \(x_k\)
Large scale problems

- Let $N$ be the number of variables of your problem.
- Then the Hessian will be an $N \times N$ matrix.
- Common to solve problems with millions of variables.
- A matrix of $10^6 \times 10^6$ doubles requires 8000 GB.
- Multiplying matrix by vector is $O(N^2)$ - very expensive
- Want 2 things:
  1. Approximate Hessian with $O(N)$ storage
  2. Way to multiply Hessian by gradient without ever constructing the Hessian

Only keep the last $m$ values of $s_k$ and $y_k$
Choose an initial starting point \((x_0)\), tolerance \(\epsilon > 0\), integer \(m > 0\).

1. \(k = 0\)
2. \(k = k + 1\)
3. while \(\|\nabla f_k\| > \epsilon\)
   1. Choose \(H_k^0\). One way to do this is \(H_k^0 = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}} I\)
   2. Find search direction \(p = -H_k \nabla f_k\)
   3. Find step size \(\alpha\) via line search
   4. \(x_{k+1} = x_k + \alpha p\)
   5. if \(k > m\), discard \(\{s_{k-m}, y_{k-m}\}\) from storage
   6. \(s_k = x_{k+1} - x_k\) and \(y_k = \nabla f_{k+1} - \nabla f_k\)
   7. \(H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T\), where \(\rho_k = 1/(y_k^T s_k)\).
   8. \(k = k + 1\)

return \(x_k\)
**Implicit matrix-vector multiplication**

**LBFGS find direction**

1. \( q = \nabla f_k \)
2. for \( i = k - 1, k - 2, \ldots, k - m \)
   1. \( \alpha_i = \rho_i s_i^T q \)
   2. \( q = q - \alpha_i y_i \)
3. \( r = H_k^0 q \)
4. for \( i = k - m, k - m + 1, \ldots, k - 1 \)
   1. \( \beta = \rho_i y_i^T r \)
   2. \( r = r - s_i (\alpha_i - \beta) \)
5. return \( r \), which equals \( H_k \nabla f_k \)
Solving a constrained QP

\[ \min_p \frac{1}{2} p^T A p + c^T p \]  

subject to \( l \leq p \leq u \)  
\[ a^T p = b. \]  

Simplest case:

- \( A \) is an \( N \times N \) symmetric matrix
- All inequality constraints are box constraints
- There is a single equality constraint.
- If \( A \) is diagonal, this can be solved in \( O(N \log N) \) time.
Dai-Fletcher Outline

The algorithm works by constructing a partial Lagrangian:

- Move the equality constraint to the objective function.
- Give it a Lagrange multiplier $\lambda$.

$$
\phi(p, \lambda) = \frac{1}{2} p^T A p + c^T p - \lambda (a^T p - b)
$$

Key point: If the original problem has a feasible solution, then there exists a $\lambda^*$ such that the optimal solution of $\phi(p, \lambda)$ will satisfy the equality constraint.

- Fix $\lambda$ and solve this optimization problem for $p$.
- If $\lambda = \lambda^*$, then $r(\lambda) := a^T p - b = 0$
- The optimal $p$ will equal the optimal $p$ from the previous problem.
Applying the box constraints

Unlike some kinds of constraints, box constraints are easy to enforce. Let $d_i$ bet the $i^{th}$ diagonal element of $A$. For any fixed value of $\lambda$, one can solve $\phi(p)$ as follows:

- Without box constraints, the optimal solution for $\phi(p)$ is
  $$h_i = \frac{(c_i + \lambda a_i)}{(d_i)}$$

- With box constraints, the optimal solution for $\phi(p)$ is
  $$p(\lambda) = \text{mid}(l, h, u).$$ (4)

- The operation $\text{mid}$ is the componentwise median of $(l, h, u)$.
- This takes $O(N)$ operations.
Finding the right $\lambda$

$$r(\lambda) := a^T p - b$$ is nondecreasing in $\lambda$
Potential problems with Dai-Fletcher
Potential problems with Dai-Fletcher

\[ r(\lambda) = (\sum_{q=1}^{t} w_q) - 1 \]

- first and last hyp
- all hypotheses
Gradient Projection

Solve problem of the form \( \min f(x) \) subject to \( x \in \Omega \), where \( \Omega \) is a closed convex set. Notation

- \( P(z) = \arg\min_x \| x - z \|_2^2 : x \in \Omega \) is called a projection
- It finds the point in \( \Omega \) that is closest to \( z \) w.r.t the 2-norm.
- Let \( x^* \) be the solution to \( \min f(x) \) subject to \( x \in \Omega \)

Then \( P(x^* - \alpha \nabla f(x^*)) = x^* \) for all \( \alpha \geq 0 \).
Gradient Projection Algorithm

1. Initialize starting point $x_0 \in \Omega$, $\gamma \in [0,1]$, $c \in (0,1)$

2. while not converged:
   
   1. Set $p_k = P(x_k - \nabla f(x_k)) - x_k$
   2. Set
      
      $$\lambda_k = \max \gamma^s$$
      subject to $s \in \{0, 1, 2, \ldots\}$

      $$f(x_k + \gamma^s p_k) - f(x_k) \leq c \gamma^s \nabla f(x_k)^T p_k$$

   3. Set $x_{k+1} = x_k + \lambda_k p_k$
Spectral Projected Gradient Notation

This differs from the vanilla gradient projection algorithm only in the way that it computes the step size.

Input to algorithm

- $M > 1$ Nonmonotone
- $0 \leq \alpha_{\text{min}} < \alpha_{\text{max}}$ are bounds on the step size
- $\gamma \in [0, 1]$ sufficient decrease parameter for potential step size
- $0 < \sigma_1 < \sigma_2 < 1$
Spectral Projected Gradient

1. If \[ \|P(x_k - g(x_k)) - x_k\| = 0, \] stop

2. Backtracking
   1. Compute \[ p_k = P(x_k - \alpha_k g(x_k)) - x_k, \] \( \lambda = 1 \)
   2. set \( x_+ = x_k + \lambda d_k \)
   3. if \( f(x_+) \leq \max_{0 \leq j \leq \min(k, M-1)} f(x_{k-j}) + \gamma \lambda (p_k^T g_k) \)
      \[ \lambda_k = \lambda, \ x_{k+1} = x_+, \ s_k = x_{k+1} - x_k, \ y_k = g(x_{k+1}) - g(x_l). \]
   4. else find \( \lambda \) via quadratic interpolation and enforce \( \lambda \in [\sigma_1 \lambda, \sigma_2 \lambda] \)

3. Set \( \alpha_k \)
   1. Compute \( b_k = s_k^T y_k \)
   2. if \( b_k \leq 0, \ \alpha_{k+1} = \alpha_{\text{max}} \)
   3. else compute \( \alpha_k = s_k^T s_k, \ \alpha_{k+1} = \min(\alpha_{\text{max}}, \max(\alpha_{\text{min}}, \alpha_k / b_k)) \)