Lecture Outline

- Quasi-Newton methods
- BFGS Method and its properties
- Limited-memory BFGS
- BFGS for sparse problems and partially separable cost functions.

You should be able to...

- Characterise the basics of quasi Newton methods
- Construct a BFGS algorithm for a given problem
- Identify the implementation issues associated with limited memory
- Take advantage of sparsity of problems

Quasi-Newton Methods

- Consider the unconstrained optimisation problem:

$$\underset{x}{\text{minimise}} \quad f(x)$$

- and *line search* iteration:

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

  - α_k: step length (step size) to be chosen according to the Wolfe conditions.
  - $p_k = -B_k^{-1} \nabla f_k$: search direction
- Quasi-Newton methods, like steepest descent, require only the gradient of the objective function.
- By measuring the changes in gradients, they construct a model of the objective function to produce superlinear convergence.
- We focus on small and medium size problems.

BFGS Method

- The most popular quasi-Newton algorithm.
- Named after its developers: Broyden, Fletcher, Goldfarb, and Shanno.
- Form the local quadratic approximation of the cost function at time k:

$$m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p$$

- $B_k \in \mathbb{R}^{n \times n}$ and $B_k > 0$ will be updated each step.
- Note that $m_k(0) = f_k$ and $\nabla_p m_k(0) = \nabla f_k$.
- The minimiser of $m_k(p)$ is the search direction $p_k = -B_k^{-1} \nabla f_k$.
- Similar to Newton's method, but Hessian is approximated.

BFGS Method

- Instead of calculating B_k from scratch every time we develop an iterative rule.
- Let's consider the model at $k+1$

$$m_{k+1}(p) = f_{k+1} + \nabla f_{k+1}^T p + \frac{1}{2} p^T B_{k+1} p$$

- What requirements should be imposed on B_{k+1}?
  - The gradient of m_{k+1} should match that of f at x_k and x_{k+1}.
- Since $\nabla m_{k+1}(0)$ is precisely ∇f_{k+1} one of these requirements is automatically satisfied.
- For the other one we have

$$\nabla m_{k+1}(-\alpha_k p_k) = \nabla f_{k+1} - \alpha_k B_{k+1} p_k = \nabla f_k$$
$$\alpha_k B_{k+1} p_k = \nabla f_{k+1} - \nabla f_k$$

BFGS Method

- Define

$$s_k = x_{k+1} - x_k = \alpha_k p_k, \quad y_k = \nabla f_{k+1} - \nabla f_k$$

- *Secant equation*:

$$B_{k+1} s_k = y_k$$

- Such positive definite matrix B_{k+1} is possible only if the following *curvature condition* holds:

$$s_k^T y_k > 0$$

- When f is strongly convex, the curvature condition will be satisfied for any two points x_k and x_{k+1}. It is not always the case for nonconvex functions.
- The curvature condition needs to be enforced.

BFGS Method

- The curvature condition is guaranteed to hold if the (strong) Wolfe conditions are imposed on the line search:

$$\nabla f_{k+1}^T s_k \geq c_2 \nabla f_k^T s_k$$
$$y_k^T s_k \geq (c_2 - 1)\alpha_k \nabla f_k^T p_k$$

- Since $c_2 < 1$ and p_k is a descent direction then the term on the right is positive.
- When the curvature condition is satisfied, the secant equation has infinitely many solutions.
- To determine B_{k+1} uniquely, among all symmetric matrices satisfying the secant equation, B_{k+1} is, in some sense, closest to the current matrix B_k:

$$\min_{B \in \mathbb{R}^{n \times n}} \quad \|B - B_k\|$$
$$\text{s.t.} \quad B = B^T, \quad Bs_k = y_k$$

- Each norm gives rise to a different quasi-Newton method.

BFGS Method

- A useful norm is the weighted Frobenius norm

$$\|A\|_W = \|W^{1/2} A W^{1/2}\|_F$$

- W is any matrix that satisfies $W y_k = s_k$.
- Specifically, let $W = G_k^{-1}$ where G_k is the average Hessian:

$$G_k = \int_0^1 \nabla^2 f(x_k + \tau \alpha_k p_k) d\tau$$

- $y_k = G_k \alpha_k p_k = G_k s_k$ follows from Taylor's Theorem.
- With this choice the unique B_{k+1} is:

$$B_{k+1} = (I - \rho_k y_k s_k^T) B_k (I - \rho_k s_k y_k^T) + \rho_k y_k y_k^T,$$
$$\rho_k = \frac{1}{y_k^T s_k}$$

- This formula is called the DFP updating formula. (proposed by Davidon in 1959, and subsequently studied, implemented, and popularized by Fletcher and Powell.)

Sherman-Morrison-Woodbury Formula

- Let $A \in \mathbb{R}^{n \times n}$ and $U, V \in \mathbb{R}^{p \times n}$.
- Suppose
$$\hat{A} = A + UV^T$$
- Then $\hat{A}$ is nonsingular if $(I + V^T A^{-1} U)$ is nonsingular and

$$\hat{A}^{-1} = A^{-1} - A^{-1}U(I + V^T A^{-1} U)^{-1} V^T A^{-1}$$

BFGS Method

- Let $B_k^{-1} = H_k$.
- Knowing H_k enables us to directly evaluates the iterates.
- Using the Sherman-Morrison-Woodbury formula:

$$H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \frac{s_k s_k^T}{y_k^T s_k}$$

- H_k undergoes a rank-two update (so does B_k).

Instead of recomputing the approximate Hessians (or their inverse) from scratch at every iteration, we apply a simple modification that combines the most recently observed information about the objective function with the existing knowledge embedded in our current Hessian approximation.

BFGS Method

- The BFGS updating rule is obtained in a similar fashion to DFP, but is more effective.
- In BFGS, H_k is directly approximated by solving

$$\begin{aligned} \min_{H \in \mathbb{R}^{n \times n}} \quad & \|H - H_k\| \\ \text{s.t.} \quad & H = H^T, \quad Hy_k = s_k \end{aligned}$$

- The norm is again weighted Frobenius with $Ws_k = y_k$.
- Let $W = G_k$, the BFGS update rule is obtained as

$$\begin{aligned} H_{k+1} &= V_k^T H_k V_k + \rho_k s_k s_k^T, \\ V_k &= I - \rho_k y_k s_k^T, \quad \rho_k = \frac{1}{y_k^T s_k} \end{aligned}$$

- How to choose H_0? Ummmm... Experience and tricks.

BFGS Method

Algorithm: BFGS Method

Initialise x_0, H_0, and $\epsilon > 0$. $\triangleright$ ϵ is the convergence tolerance;
$k \leftarrow 0$;
while $\|\nabla f_k\| > \epsilon$ **do**
  $p_k \leftarrow -H_k \nabla f_k$;
  $x_{k+1} \leftarrow x_k + \alpha_k p_k$; $\triangleright$ α_k satisfies Wolfe conditions.
  $s_k \leftarrow x_{k+1} - x_k$; $y_k \leftarrow \nabla f_{k+1} - \nabla f_k$; $\rho_k \leftarrow 1/(y_k^T s_k)$;
  $H_{k+1} \leftarrow (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$;
  $k \leftarrow k + 1$;
end while

- The cost of each iteration: $O(n^2)$ arithmetic operations (plus the cost of function and gradient evaluations);
- No $O(n^3)$ operations, e.g. matrix inversion or matrix multiplication.
- The algorithm is robust and superlinear.

BFGS Method: Properties

- Even though Newton is quadratic, its cost per iteration usually is higher, and might be slower in practice.
- If H_k is positive definite, H_{k+1} is positive definite too.
- It can be seen from multiplying both side of the update rule from the left and the right by an arbitrary vector.
- If the matrix H_k incorrectly estimates the curvature in the objective function, then the Hessian approximation will tend to correct itself within a few steps.
- It is also known that the DFP method is less effective in correcting bad Hessian approximations; (possible reason for poorer practical performance.
- The self-correcting properties of BFGS hold only when an adequate line search is performed.

BFGS Method: Implementation

- The line search algorithm should always try the step length $\alpha_k = 1$ first.
- Computational observations strongly suggest that it is more economical, in terms of function evaluations, to perform a fairly inaccurate line search.
- $c_1 = 10^{-4}$ and $c_2 = 0.9$ are commonly used for the Wolfe conditions.
- H_0 often is set to some multiple βI.
- A heuristic that is often quite effective is to scale the starting matrix after the first step has been computed but before the first BFGS update is performed: $H_0 \leftarrow (y_0^T s_0)/(y_0^T y_0) I$.
- The performance of the BFGS method can degrade if the line search is not based on the Wolfe conditions. For example, some software implements an Armijo backtracking line search and it is not a good idea.

BFGS Method: Convergence

- Because the Hessian approximations is updated at each step makes the analysis of quasi-Newton methods much complex.
- We cannot prove that the iterates of these quasi-Newton methods approach a stationary point of the problem from any starting point and any B_0.

Theorem (Convergence of the BFGS Method): *Suppose that the objective function f is twice continuously differentiable. Let B_0 be any symmetric positive definite initial matrix, and let x_0 be a starting point for which the subselvel set $\mathcal{L} = \{x | f(x) \leq f(x_0)\}$ is convex. Furthermore, there exists $0 < m \leq M$ such that $m\|z\|^2 \leq z^T \Gamma(x) z \leq M\|z\|^2$ for all $z \in \mathbb{R}^n$ and $x \in \mathcal{L}$ with $\Gamma(x) = \nabla^2 f(x)$. Then the sequence $\{x_k\}$ generated by the BFGS Method (with $\epsilon = 0$) converges to the minimizer $x^\star$ of f.*

BFGS Method: Convergence

- For the type of problems discussed earlier it can be shown that the convergence rate is linear. Particularly,

$$\sum_{k=1}^{\infty} \|x_k - x^\star\| < \infty.$$

Theorem (Superlinear Convergence of the BFGS Method): *Suppose that the objective function f is twice continuously differentiable and that the iterates generated by the BFGS algorithm converge to a minimizer $x^\star$ at which $\Gamma(x)$ is locally Lipschitz. Suppose also that*

$$\sum_{k=1}^{\infty} \|x_k - x^\star\| < \infty.$$

Then x_k converges to $x^\star$ at a superlinear rate.

Limited-Memory BFGS

- Solving large problems whose Hessian matrices cannot be computed at a reasonable cost or are not sparse.
- Maintain simple and compact approximations of Hessian matrices.
- Instead of storing fully dense $n \times n$ approximations, they save only a few vectors of length n that represent the approximations implicitly.
- Despite these modest storage requirements, they often yield an acceptable (albeit linear) rate of convergence.
- The main idea of limited-memory BFGS (L-BFGS) is to use curvature information from only the most recent iterations to construct the Hessian approximation.
- Curvature information from earlier iterations is discarded in the interest of saving storage.

Limited-Memory BFGS

- The last m pairs $\{s_j, y_j\}$ is recorded. $(m \in [3, 20])$
- Compute $H_k \nabla f_k$ by performing a sequence of inner products and vector summations involving ∇f_k and the pairs $\{s_j, y_j\}$.
- Some "initial" Hessian approximation $H_{0|k}$ is chosen.
- $H_k = H_{j+1|k}$:

$$H_{j+1|k} = V_{j|k}^T H_{j|k} V_{j|k} + \rho_{j|k} s_{j|k} s_{j|k}^T, \quad j = 0, \ldots, m-1,$$

$$V_{j|k} = I - \rho_{j|k} y_{j|k} s_{j|k}^T, \quad \rho_{j|k} = \frac{1}{y_{j|k}^T s_{j|k}},$$

$$y_{j|k} = y_{k+(j-m)}, \quad s_{j|k} = s_{k+(j-m)},$$

- After the current iterate, the oldest vector pair in the set of pairs $\{s_j, y_j\}$ is replaced by the new pair $\{s_k, y_k\}$.

Limited-Memory BFGS

Algorithm: L-BFGS Two-Loop Recursion

$q \leftarrow \nabla f_k$
for $i = k-1, k-2, \ldots, k-m$ **do**
  $\alpha_i \leftarrow \rho_i s_i^T q$;
  $q \leftarrow q - \alpha_i y_i$;
end for
$r \leftarrow H_k^0 q$
for $i = k-m, k-m+1, \ldots, k-1$ **do**
  $\beta \leftarrow \rho_i y_i^T r$;
  $r \leftarrow r + s_i(\alpha_i - \beta)$
end for
return r ▷ $H_k \nabla f_k = r$

- $4mn$ multiplications plus
  - n if H_k^0 is diagonal or
  - n^2 otherwise.
- In practice, $H_k^0 = \gamma_k I$, $\gamma_k = \dfrac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$.

Limited-Memory BFGS

Algorithm: L-BFGS

Choose x_0, $m > 0$, and ϵ;
$k \leftarrow 0$;
while $\|\nabla f_k\| > \epsilon$ **do**
  $p \leftarrow -H_k \nabla f_k$; ▷ Use L-BFGS Two-Loop Recursion for a H_k^0 and memory length $\min(k, m)$.
  $x_{k+1} \leftarrow x_k + \alpha_k p_k$ ▷ α_k satisfies Wolfe conditions.
  if $k > m$ **then**
    Discard $\{s_{k-m}, y_{k-m}\}$;
  end if
  Save $\{s_k, y_k\}$; ▷ $s_k \leftarrow x_{k+1} - x_k$, $y_k \leftarrow \nabla f_{k+1} - \nabla f_k$
  $k \leftarrow k + 1$
end while

- Small m is "bad", optimal m is problem dependent.
- It works well in practice and the approach of choice for large problems in which the true Hessian is not sparse.
- Converges slowly on ill-conditioned problems.

BFGS Updates in Outer-product Form

- Consider $\{s_i, y_i\}_{i=k-m}^{k-1}$, $s_i^T y_i > 0$. Let B_k be obtained from BFGS updates from $B_{0|k}$:

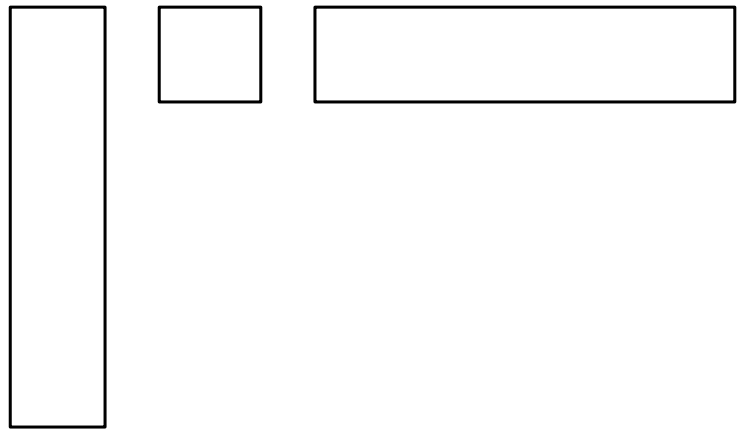
$$
\begin{aligned}
B_k &= B_{0|k} - \begin{bmatrix} B_0 S_k & Y_k \end{bmatrix} \begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & -D_k \end{bmatrix} \begin{bmatrix} S_k^T B_0 \\ Y_k^T \end{bmatrix}, \\
S_k &= \begin{bmatrix} s_{k-m}, \ldots, s_{k-1} \end{bmatrix}, \quad Y_k = \begin{bmatrix} y_{k-m}, \ldots, y_{k-1} \end{bmatrix}, \\
[L_k]_{i,j} &= \begin{cases} s_{k-m-1+i}^T y_{k-m-1+i} & i > j, \\ 0 & \text{otherwise}, \end{cases} \\
D_k &= \operatorname{diag}\left(s_{k-m}^T y_{k-m}, \ldots, s_{k-1}^T y_{k-1} \right)
\end{aligned}
$$

- It can be proved by induction.
- $s_i^T y_i > 0$ guarantees that the middle matrix is invertible.
- At iteration $k > m$, the m most recent $\{s_i, y_i\}$ are kept.
- $B_{0|k} = \delta_k I$, $\delta_k = 1/\gamma_k$.

BFGS Updates in Outer-product Form

- The updates can be expressed as an outer product of two long and narrow matrices, i.e. $[\delta_k S_k \; Y_k]$ and its transpose, along with a multiplication by a small $2m \times 2m$ matrix.

$$B_k = \delta_k I +$$



- Updating B_k requires approximately $2mn + O(m^3)$ operations.
- Several codes for general-constrained optimization, including `KNITRO` and `IPOPT`, make use of this update.
- This form of update is by far more efficient than "unrolling the updates": $2mn$ vs. $\frac{3}{2}m^2n$ multiplications.
- A compact representation of the inverse BFGS approximation H_k can be computed as well.

Sparse Quasi-Newton Updates

- Main idea: B_k to have the same (or similar) sparsity pattern as the true Hessian.
- This approach reduces the storage requirements, reduces the number of multiplications and additions, and improves the accuracy (absolute zeros instead of machine epsilons.).
- To this aim let's assume that indices for the entries of the Hessian that are nonzero are known:

$$\Omega = \{(i,j) || \nabla^2 f_x|_{ij} \neq 0,\ x \in \overbrace{\mathcal{D}_f}^{\text{Domain of } f} \}$$

- Suppose B_k has the same sparsity, i.e. $[B_k]_{ij} = 0,\ (i,j) \notin \Omega$:

$$\begin{aligned} \min_{B} \quad & \|B - B_k\|_F^2 \\ \text{s.t.} \quad & Bs_k = y_k,\ B = B^T,\ [B_k]_{ij} = 0,\ (i,j) \notin \Omega. \end{aligned}$$

- It requires solving an $n \times n$ linear system with sparsity Ω.
- It sounds promising, but in practice poor.

BFGS For Partially Separable Cost Functions

- Consider the cost function:

$$f(x) = \sum_{i=1}^{p} f_i(x)$$

- It is straightforward that

$$\nabla f(x) = \sum_{i=1}^{p} \nabla f_i(x), \quad \nabla^2 f(x) = \sum_{i=1}^{p} \nabla^2 f_i(x)$$

- Is it more efficient to maintain quasi-Newton approximations of $\nabla^2 f_i(x)$ instead of $\nabla^2 f(x)$?
- Yes, if the approximations specifically exploits the structure of each f_i.
- For each f_i we have: $f_i(x) = \phi_i(U_i x)$, where U_i is a selector matrix, i.e. with entries of zero or 1 and each row only one 1, and is known as a *compactifying matrix*.

BFGS For Partially Separable Cost Functions

- For example, for $f : \mathbb{R}^4 \to \mathbb{R}$:

$$f_i(x) = x_1^2 + x_4^2, \quad \phi_i(z^{[i]}) = (z_1^{[i]})^2 + (z_2^{[i]})^2$$
$$U_i = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad z^{[i]} = U_i x.$$

- Consequently,

$$\nabla f_i(x) = U_i^T \nabla \phi_i(U_i x), \quad \nabla^2 f_i(x) = U_i^T \nabla^2 \phi_i(U_i x) U_i$$

- Instead of maintaining a quasi-Newton approximation to $\nabla^2 f_i$, we maintain a low dimensional quasi-Newton approximation $B^{[i]}$ of $\nabla^2 \phi_i$.
- At each time $B_{k+1}^{[i]}$ is updated using $B_k^{[i]}$ and

$$s_k^{[i]} = \underbrace{U_i x_{k+1}}_{z_{k+1}^{[i]}} - \underbrace{U_i x_k}_{z_k^{[i]}}, \quad y_k^{[i]} = \nabla \phi_i(U_i x_{k+1}) - \nabla \phi_i(U_i x_k)$$

BFGS For Partially Separable Cost Functions

- The full approximation to $\nabla^2 f_i$:

$$B = \sum_{i=1}^{p} U_i^T B^{[i]} U_i.$$

- Taking advantage of the separability of the cost function is powerful for large number of variables, e.g. > 1000.
- Not always possible to use the BFGS formula to update the partial Hessian $B^{[i]}$; there is no guarantee that the curvature condition ${s^{[i]}}^T y^{[i]} > 0$ is satisfied.
- The performance of quasi-Newton methods is satisfactory provided that we find the finest partially separable decomposition of the problem.
- Modelling language `AMPL` automatically detects the partially separable structure of a function and uses it to compute the Hessian.