AM 205: lecture 19

- Last time: Conditions for optimality, Newton's method for optimization
- Today: survey of optimization methods

Newton's Method: Robustness

Newton's method generally converges much faster than steepest descent

However, Newton's method can be unreliable far away from a solution

To improve robustness during early iterations it is common to perform a line search in the Newton-step-direction

Also line search can ensure we don't approach a local max. as can happen with raw Newton method

The line search modifies the Newton step size, hence often referred to as a damped Newton method

Newton's Method: Robustness

Another way to improve robustness is with trust region methods

At each iteration k, a "trust radius" R_k is computed

This determines a region surrounding x_k on which we "trust" our quadratic approx.

We require $\|x_{k+1} - x_k\| \leq R_k$, hence constrained optimization problem (with quadratic objective function) at each step

Newton's Method: Robustness

Size of R_{k+1} is based on comparing actual change, $f(x_{k+1}) - f(x_k)$, to change predicted by the quadratic model

If quadratic model is accurate, we expand the trust radius, otherwise we contract it

When close to a minimum, R_k should be large enough to allow full Newton steps $\implies$ eventual quadratic convergence

Quasi-Newton Methods

Newton's method is effective for optimization, but it can be unreliable, expensive, and complicated

- Unreliable: Only converges when sufficiently close to a minimum
- Expensive: The Hessian H_f is dense in general, hence very expensive if n is large
- Complicated: Can be impractical or laborious to derive the Hessian

Hence there has been much interest in so-called quasi-Newton methods, which do not require the Hessian

Quasi-Newton Methods

General form of quasi-Newton methods:

$$x_{k+1} = x_k - \alpha_k B_k^{-1} \nabla f(x_k)$$

where α_k is a line search parameter and B_k is some approximation to the Hessian

Quasi-Newton methods generally lose quadratic convergence of Newton's method, but often superlinear convergence is achieved

We now consider some specific quasi-Newton methods

BFGS

The Broyden–Fletcher–Goldfarb–Shanno (BFGS) method is one of the most popular quasi-Newton methods:

choose initial guess x_0
choose B_0, initial Hessian guess, *e.g.* $B_0 = \mathrm{I}$
for $k = 0, 1, 2, \ldots$ **do**
  solve $B_k s_k = -\nabla f(x_k)$
  $x_{k+1} = x_k + s_k$
  $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
  $B_{k+1} = B_k + \Delta B_k$
end for

where

$$\Delta B_k \equiv \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$$

BFGS

See lecture: derivation of the Broyden root-finding algorithm

See lecture: derivation of the BFGS algorithm

Basic idea is that B_k accumulates second derivative information on successive iterations, eventually approximates H_f well

BFGS

Actual implementation of BFGS: store and update inverse Hessian to avoid solving linear system:

choose initial guess x_0
choose H_0, initial inverse Hessian guess, *e.g.* $H_0 = \mathrm{I}$
for $k = 0, 1, 2, \ldots$ **do**
    calculate $s_k = -H_k \nabla f(x_k)$
    $x_{k+1} = x_k + s_k$
    $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
    $H_{k+1} = H_k + \Delta H_k$
end for

where

$$\Delta H_k \equiv (I - s_k \rho_k y_k^t) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \qquad \rho_k = \frac{1}{y_k^t s_k}$$

BFGS

BFGS is implemented as the `fmin_bfgs` function in `scipy.optimize`

Also, BFGS (+ trust region) is implemented in Matlab's `fminunc` function, *e.g.*

```
x0 = [5;5];
options = optimset('GradObj','on');
[x,fval,exitflag,output] = ...
  fminunc(@himmelblau_function,x0,options);
```