CME 216, ME 343 - Winter 2021

Eric Darve, ICME

BFGS

This is one of the most efficient optimizers.

It belongs to the class of quasi-Newton methods.

Instead of computing the Hessian exactly (expensive), we use an approximation.

The method is ingenious.

Assume we have multiple evaluations of the gradient (e.g., one evaluation per step).

We know that:

$$\nabla^2 l \cdot p \approx \nabla l(x + p) - \nabla l(x)$$

This can be used to progressively approximate $\nabla^2 l$.

Example: pick $p = \varepsilon\, e_i$. Then:

$$\nabla^2 l \cdot p = \varepsilon\, \nabla^2 l[:, i] \approx \nabla l(x + p) - \nabla l(x)$$

We get column i of the Hessian matrix.

We can gain information about $\nabla^2 l$ from differences like

$$\nabla l(x+p) - \nabla l(x)$$

Let's use this insight to build an algorithm to approximate $\nabla^2 l$.

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

$$\nabla l_{k+1} \approx \nabla l_k + \nabla^2 l \cdot (\alpha_k p_k)$$

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

$$y_k = \nabla l_{k+1} - \nabla l_k$$

$$\nabla l_{k+1} \approx \nabla l_k + \nabla^2 l \cdot s_k$$

$$B_{k+1} \approx \nabla^2 l_{k+1}$$

Secant equation:

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

But recall that Newton's step is approximately:

$$-B_k^{-1} g_k$$

So instead of working with the Hessian B_k, it's more efficient to work directly with its inverse $H_k = B_k^{-1}$.

The secant equation becomes:

$$B_{k+1}\boldsymbol{s}_k = \boldsymbol{y}_k \rightarrow$$

$$H_{k+1}\boldsymbol{y}_k = \boldsymbol{s}_k$$

How can we use this secant equation to approximate $[\nabla^2 l]^{-1}$?

Assume we have some approximation at step k, H_k.

We want to use

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

to find a better approximation H_{k+1} of $\left[\nabla^2 l\right]^{-1}_{k+1}$.

There are many H_{k+1} that will solve the secant equation.

In BFGS we solve for

$$\mathbf{argmin}_H \|H - H_k\| \ (\mathbf{with\ some\ suitable\ norm})$$

subject to $H = H^T$ and $Hy_k = s_k$.

BFGS solution is:

$$H_{k+1} = (I - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^T) H_k (I - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^T) + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T$$

$$\rho_k = \frac{1}{\boldsymbol{y}_k^T \boldsymbol{s}_k}$$

Let's check that it is correct.

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$$

$H_{k+1} y_k = s_k$. Multiply to the right by y_k:

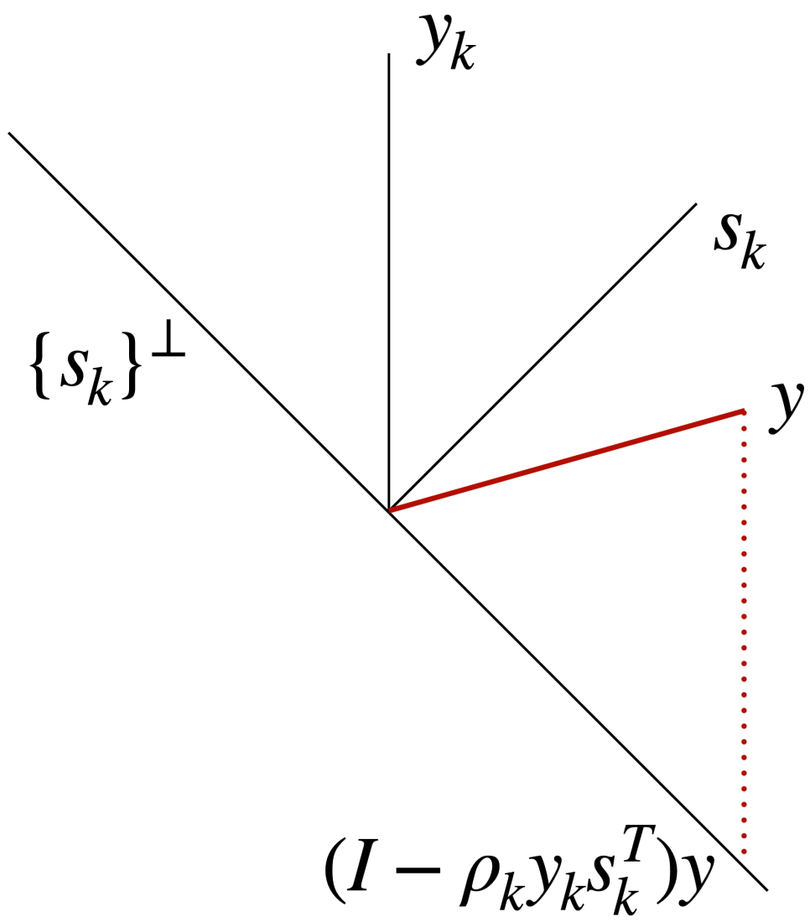
$$\rho_k s_k s_k^T y_k = \frac{s_k^T y_k}{y_k^T s_k} s_k = s_k$$

$$(I - \rho_k y_k s_k^T) y_k = y_k - \frac{s_k^T y_k}{y_k^T s_k} y_k = 0$$

$I - \rho_k y_k s_k^T$ is a projection onto $\{s_k\}^\perp$ along y_k.

$$P = I - \rho_k y_k s_k^T$$

$$P^2 = P$$


y_k
s_k
$\{s_k\}^\perp$
y
$(I - \rho_k y_k s_k^T)y$

```
while ||g[k]|| > eps:
    pk = -Hk*g[k]
    x[k+1] = x[k] + ak*pk # ak is obtained using a line search
    sk = x[k+1] - x[k]
    yk = g[k+1] - g[k]
    Update Hk using BFGS equation
    k ← k+1
```

scipy optimize BFGS implementation

For more information see Numerical Optimization, by Nocedal and Wright, Springer, 2nd edition