Broyden Method (II)

Wednesday, October 2, 2024 10:00 AM

Class 16: October 2, 2024

Recall: Last class we introduced the Broyden method, focusing on how it addresses the issues Newton has with potential high cost per iteration. Broyden method requires at most one Jacobian evaluation at x0, and the matrix it uses instead of J_F(x_k) is built cleverly so that solving linear systems for it is cheaper (O(n^2) or better)) than the Newton system solve (O(N^3)).

To summarize: Inputs for Broyden are nx1 vector of initial guesses x0 and n x n matrix B0 ~ J_F(x0). Broyden then proposes a **rank 1 update formula** of the form:

$$B_{k+1} = B_k + u_{k+1} v_{k+1}^T$$

Where uk and vk are computed from the information provided by the most recent two iterates x_{k+1} and x_k. That is:

$$Dxk = (x_{k+1} - x_k)$$
$$DFk = F(x_{k+1}) - F(x_k)$$
$$r_k = DFk - B_k\, Dxk$$

$$u_{k+1} = r_k / ||Dxk||$$
$$v_{k+1} = Dxk / ||Dxk||$$

This rank 1 update is the smallest update in Frobenius norm that satisfies the Secant Equation $\mathbf{(F(xk) - F(x_k)) - B_{k+1} (x_{k+1} - x_k) = 0}$. This then means that

$$B_k = B0 + (u1\, v1^T + u2\, v2^T + \dots + uk\, vk^T)$$

However, in implementing Broyden, we have to solve a system for B_{k+1}. We find a clever formula for this solve also as a **rank 1 update for the solve for B_k** using the Sherman Morrison formula. This is what is implemented in a Broyden method:

$$B_{k+1}^{-1} = Bk^{-1} + wk\, yk^T$$

Where, once again, we have a formula for vectors wk and yk in terms of the information from our two most recent iterates:

$$zk = solve(Bk, DFk)$$

$$wk = (Dxk - zk) / (Dxk^T zk)$$
$$yk = solve(B_k^T , Dxk)$$

This then means that

$$B_k^{-1} = B0^{-1} + (w1\, y1^T + w2\, y2^T + \dots + wk\, yk^T)$$

IMPLEMENTING BROYDEN:

So, we now have what we need to write a pseudocode. Lets first build the solve module:

```
LU, piv = lu_factor(B0);
def B0solve(v, v):
    return lu_solve((LU, piv), v);
```

lu_factor & lu_solve are scipy routines!

WARM UP: Say we have $\{\vec{u}_j, \vec{v}_j\}_{j=1}^{k}$.

Form $U_{n\times k} = \begin{bmatrix} | & | & & | \\ \vec{u}_1 & \vec{u}_2 & \cdots & \vec{u}_k \\ | & | & & | \end{bmatrix}$ $V_{n\times k} = \begin{bmatrix} | & | & & | \\ \vec{v}_1 & \vec{v}_2 & \cdots & \vec{v}_k \\ | & | & & | \end{bmatrix}$

Then,

$$\underset{n\times k}{U}\;\underset{k\times n}{V^T} = \sum_{j=1}^{k} \vec{u}_j \vec{v}_j^{\,T}$$

Challenge:

```
def BKsolve(B0, U, V, b):
    % apply (B0^-1 + UV^T) b
    x = B0solve(B0, b);
    x = x + U(V^T b)
return x;
```

FORMULA FOR BROYDEN:

$\vec{x}_0$, B_0

$$x_1 = x_0 + B_0^{-1} F(x_0).$$

$\vec{x}_0, \vec{x}_1, F(\vec{x}_0), F(\vec{x}_1), B_0$.

a) $B_1 = B_0 + uv^T$

b) $\underbrace{F(x_1) - F(x_0)}_{\Delta F_1} = B_1 \underbrace{(x_1 - x_0)}_{\Delta x_1}$

$$\Delta F_1 = (B_0 + uv^T)\,\Delta x_1$$

$$\Delta F_1 = B_0 \Delta x_1 + u(v^T \Delta x_1)$$

$$\Delta F_1 = B_0 \Delta x_1 + u(v^T \Delta x_1)$$

$$\vec{r}_1 = \Delta F_1 - B_0 \Delta x_1 = u(v^T \Delta x_1)$$

$$\boxed{\vec{u} = \left(\frac{1}{\vec{v}^T \Delta x_1}\right) \cdot \vec{r}_1}$$

- $\min \|u\|^2 \|v\|^2$

$$\min \|r_1\|^2 \left(\frac{1}{v^T \Delta x_1}\right)^2 \cdot \|v\|^2$$

$$(v^T \Delta x_1)^2 = \|v\|^2 \|\Delta x_1\|^2 \cos^2\theta$$

$$\min \frac{\|v\|^2}{\|v\|^2 \|\Delta x_1\|^2 \cos^2\theta} \rightarrow \min \frac{1}{\cos^2\theta}$$

$$v = \Delta x_1, \quad u = \vec{r}_1 \cdot \left(\frac{1}{\Delta x_1^T \Delta x_1}\right).$$

Steepest Descent / Smouth Optimization

$q(\vec{x}) \quad q: \mathbb{R}^n \rightarrow \mathbb{R}$

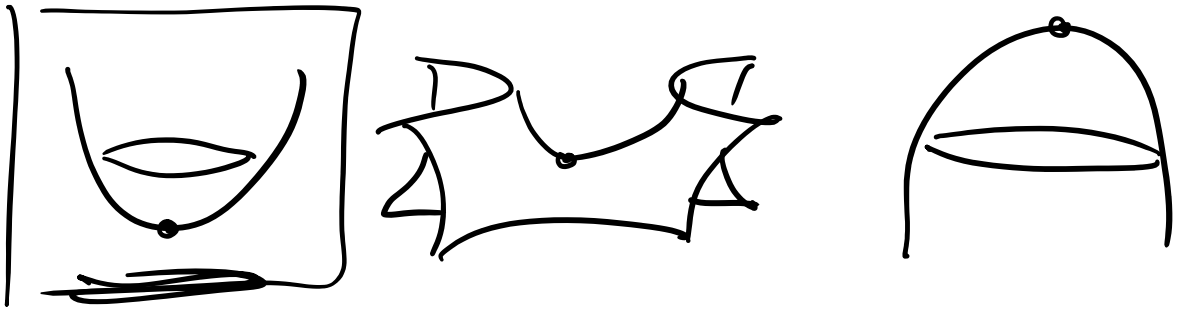
$\min q(\vec{x})$ (Find local minima of q.)

$\max p(\vec{x}) \rightarrow \min(-p(\vec{x}))$

$$\max \; p(\vec{x}) \longrightarrow \min \left(-p(\vec{x}) \right)$$

Find $\vec{x}$ s.t. $\nabla q(\vec{x}) = \vec{0}$

$\nabla q : \mathbb{R}^n \longrightarrow \mathbb{R}^n$.



- Gradient Descent