

Quasi-Newton methods: the Broyden method

Monday, September 30, 2024 12:14 PM

Class 15: September 30, 2024

Recall: Last class, we wrapped up our discussion of the Newton method, as it is applied to systems of N nonlinear equations in N variables. As was true in the case N=1, what we can say is that

Theorem: if F is C^2 on a neighborhood of the root r and J_F(r) is invertible, then there is a delta such that $\|x_0 - r\| < \delta$ means Newton will converge quadratically to r.

We discussed one limitation of Newton which is true for all cases (N=1 and N>1): global convergence (from any x_0) is not guaranteed. Newton needs to be paired with some other method or guided (e.g. using line search) to address this issue.

We then took a pause to ask: what do we care about when it comes to how "fast" an iterative method like Newton goes? Really, we care about how quickly I can solve my problem, as a function of N. An informal formula for that cost is:

$$\text{Cost}(N) = (\# \text{ Iterations}) \times (\text{Cost per iteration})$$

How quickly the method converges influences the first factor: if we start close to r, Newton converging quadratically means we have a small (# of iterations).

However, what determines the true cost / time spent is the product of (# iterations) by (Cost per iteration). One of Newton's main limitations (and a motivation for the subject we will cover this class) is that for large N, there can be two sources of considerable cost per iteration:

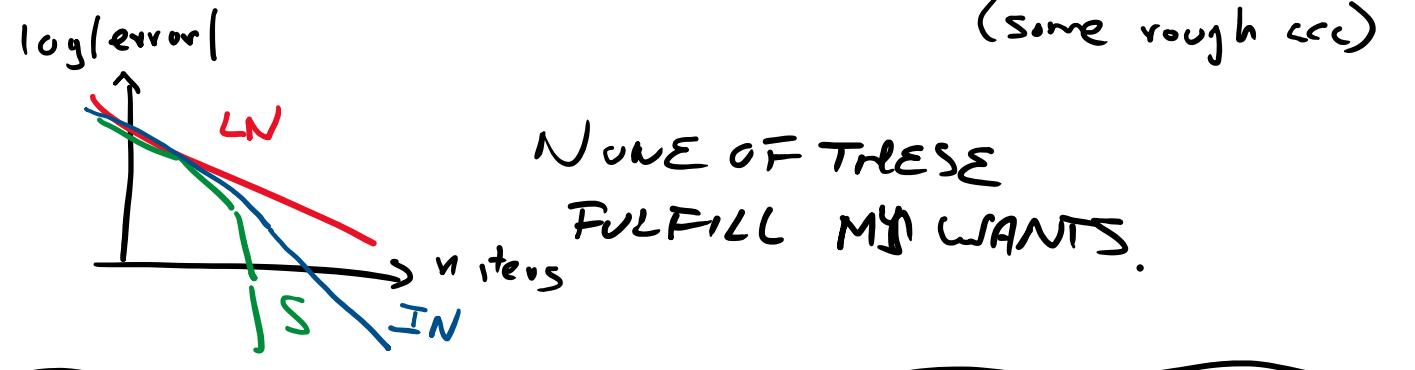
- **Computing the Jacobian $J_F(x_k)$ accurately:** we are computing and storing N^2 derivatives. For some problems (like the non-linear FEM tire model I described in class), this is very expensive and time-consuming.
- **Solving a large linear system:** In general, you end up with a linear system of size N to solve per step. Unless you know something about your matrix, this will likely be done using Gaussian Elimination, which is $O(N^3)$.

Today, we discuss the **Broyden method**, as a Quasi-Newton method that seeks to **keep superlinear convergence** (so, keep # of iterations low) while substantially **reducing the cost per iteration**. That means we want

- Evaluate the Jacobian matrix accurately 1 or 0 times during the run of the method.
- The linear system solve should be **much cheaper**: $O(N^2)$ or better.

DISCUSSED :

- Lazy Newton (Chord Iter) - 1 $J_F(x_0)$
- Shamin'ski (not so lazy) - update every m.
- Inexact Newton. \rightarrow Solve to $1e-3$.
(some rough acc)



BROYDEN (1965)

- Same inputs as Newton. (\vec{x}_0 , TOL, nmax)
 $\rightarrow \dots \rightarrow r \approx \vec{x}$

- Same inputs as Newton. (x_0 , TOL, "max")
- $F(\vec{x})$, $\mathbb{B}_0 \approx J_F(\vec{x}_0)$
 $N \times N$

$$\textcircled{1} \quad \vec{x}_1 = \vec{x}_0 + \mathbb{B}_0^{-1} F(\vec{x}_0)$$

IMPLEMENT: $\vec{P}_0 = \text{np.linalg.solve}(\mathbb{B}_0, -F(\vec{x}_0))$
 $\vec{x}_1 = \vec{x}_0 + \vec{P}_0.$

IDEA: \mathbb{B}_1

Update formula

$$\mathbb{B}_0, \vec{x}_1, \vec{x}_0 \rightarrow \boxed{\begin{array}{c} \text{UPDATE} \\ \text{FORMULA} \end{array}} \rightarrow \mathbb{B}_1.$$

SOLVE FOR B_1 cheap.

TRICK: "Secant Equation"

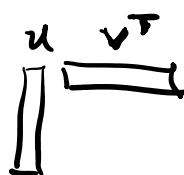
$$F(\vec{x}_1) - F(\vec{x}_0) = \mathbb{B}_1 (\vec{x}_1 - \vec{x}_0)$$

◻ \mathbb{B}_1 should be "close" to \mathbb{B} .

$$\mathbb{B}_1 = \mathbb{B}_0 + (\text{update})$$

Broden: "Rank 1 update"

$$\textcircled{1} \quad \mathbb{B}_1 = \mathbb{B}_0 + \vec{u} \vec{v}^T$$



- \rightarrow , \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow

$u_i v_i$

Given \vec{x} , $(uv^\top)\vec{x} = (v^\top \vec{x})\vec{u}$

Plug this into sec. eq

$\vec{u} = \text{function } (\vec{v})$

$$(0) \min \underbrace{\|uv^\top\|_F^2}_{\text{matrix}} = \min \|u\|^2 \|v\|^2$$

Frobenius norm $\left(\sum_{ij} |a_{ij}|^2\right)^{1/2}$

Broyden Update:

$$\bar{B}_{k+1} = \bar{B}_k + \frac{\vec{r}_k \vec{r}_k^\top}{(\Delta \vec{x}_k)^\top \Delta \vec{x}_k} \frac{\Delta \vec{x}_k^\top}{\vec{v}_k^\top}$$

$$\Delta \vec{x}_k = \vec{x}_{k+1} - \vec{x}_k$$

$$\vec{r}_k = (F(x_{k+1}) - F(x_k)) - \bar{B}_k \Delta \vec{x}_k.$$

$$\begin{aligned} \bar{B}_{k+1} &= \bar{B}_k + U_k V_k^\top = \bar{B}_{k-1} + U_k V_k^\top + U_{k-1} V_{k-1}^\top \\ &\quad \vdots \\ &= \bar{B}_0 + \underbrace{\sum_{j=1}^k \vec{u}_j \vec{v}_j^\top}_{\bar{B}} \end{aligned}$$

"Sherman-Morrison-(Woodbury) Formula"

I know A invertible and
 I have A^{-1} or a way to apply
 it ($A = LU$, $A^{-1} = U^{-1}L^{-1}$)

What about $A + uv^T$?

Can I compute / apply $(A + uv^T)^{-1}$?

$$(A + uv^T)^{-1} = A^{-1} - \frac{1}{1 + v^T A^{-1} u} \frac{(A^{-1} u)(v^T A^{-1})}{\text{col vec} \quad \text{row vec}}$$

IFF $\underline{1 + v^T A^{-1} u \neq 0}$

Broyden (Inverse update)

$$\begin{aligned} T\bar{B}_{k+1}^{-1} &= T\bar{B}_k^{-1} + \tilde{u}_k \tilde{v}_k^T \\ &= \dots = \bar{B}_0^{-1} + \sum_{j=1}^k \tilde{u}_j \tilde{v}_j^T \end{aligned}$$

$$T\bar{B}_{k+1}^{-1}(\vec{x}) = \underbrace{\bar{B}_0^{-1} \vec{x}}_{\text{Efficient because } \bar{B}_0 \text{ does not change.}} + \left(\sum_{j=1}^k \tilde{u}_j \tilde{v}_j^T \right) \vec{x}$$

K inner product
 $O(nk)$

because $\|z\|_0$
does not change. | U(nk)