

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 δ 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:

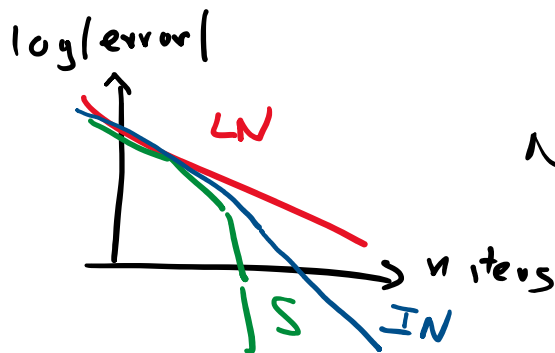
- **Computing the Jacobian $J_F(x_k)$ accurately:** we are computing and storing N^2 derivatives. For some problems (like the non-linear FEM tire model I described in class), this is very expensive and time-consuming.
- **Solving a large linear system:** In general, you end up with a linear system of 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 $J_F(x_0)$
- Shamanskii (not so lazy) - update every m .
- Inexact Newton. \rightarrow Solve to $1e-3$. (some rough cc)



NONE OF THESE
FULFILL MY WANTS.

BRUYDEN (1965)

- Same inputs as Newton. $(\vec{x}_0, \text{TOL}, n_{\max})$

- Same inputs as Newton. (x_0 , TOL, v_{\max})
- $F(\vec{x})$, $B_0 \approx J_F(\vec{x}_0)$
 $N \times N$

$$\textcircled{1} \quad \vec{x}_1 = \vec{x}_0 + B_0^{-1} F(\vec{x}_0)$$

IMPLEMENT: $\vec{p}_0 = \text{np.linalg.solve}(B_0, -F(x_0))$
 $\vec{x}_1 = \vec{x}_0 + \vec{p}_0.$

IDEA: B_1

Update formula

$$B_0, \vec{x}_1, \vec{x}_0 \rightarrow \boxed{\text{UPDATE FORMULA}} \rightarrow B_1.$$

SOLVE FOR B_1 cheap.

TRICK: "Secant Equation"

$$F(\vec{x}_1) - F(\vec{x}_0) = B_1 (\vec{x}_1 - \vec{x}_0)$$

\square B_1 should be "close" to B .

$$B_1 = B_0 + (\text{update})$$

Broyden: "Rank 1 update"

$$(\bullet) \quad B_1 = B_0 + \vec{u} \vec{v}^T$$

$$\begin{array}{c} \vec{u} \quad \vec{v}^T \\ \parallel \quad \parallel \\ u_i \quad v_i \end{array}$$

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

$$u_i v_j$$

Plug this into sec. eq

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

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

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

Bradyden Update:

$$B_{k+1} = B_k + \frac{\vec{r}_k}{\underbrace{(\Delta \vec{x}_k)^T \Delta \vec{x}_k}_{\vec{u}_k}} \underbrace{\Delta \vec{x}_k^T}_{\vec{v}_k^T}$$

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

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

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

$$= B_0 + \sum_{j=1}^k \vec{u}_j \vec{v}_j^T$$

"Sherman-Morrison-(Woodbury) Formula"

J'

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 $(A + uv^T)^{-1}$?

$$(A + uv^T)^{-1} = A^{-1} - \frac{1}{\underbrace{1 + v^T A^{-1} u}_{\neq 0}} \underbrace{(A^{-1} u)}_{\substack{\text{col} \\ \text{vec}}} \underbrace{(v^T A^{-1})}_{\substack{\text{row} \\ \text{vec}}}$$

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

Broyden (Inverse update)

$$\begin{aligned} B_{k+1}^{-1} &= B_k^{-1} + \tilde{u}_k \tilde{v}_k^T \\ &= \dots = B_0^{-1} + \sum_{j=1}^k \tilde{u}_j \tilde{v}_j^T \end{aligned}$$

$$B_{k+1}^{-1}(\vec{x}) = \underbrace{B_0^{-1} \vec{x}}_{\text{Efficient}} + \left(\sum_{j=1}^k \tilde{u}_j \tilde{v}_j^T \right) \vec{x}.$$

Efficient
 because B_0
 does not change.

k inner product
 $O(nk)$

because \parallel so
does not change. | $O(n/k)$