1 Derivation of Gradient and Hessian for Huber Term

We find that the implementation of the gradient and hessian of the huber in Q2 is difficult and thus creates large room for performance improvement, which is also one of our major improvements made to the previous parts. We demonstrate the detailed calculation of the gradient and hessian (hessian evaluated at x dot product t) for the huber term $h(x)$, that enables the success of accelerated gradient method, and especially Newton-CG algorithm as we have shown in previous experiments. And we have verified the correctness of the calculated gradient and hessian by comparing the result with Pytorch's inherent Autograd functionality.

$$\begin{aligned} h(x) &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \psi_{hub}(\mathbf{X}_i - \mathbf{X}_j) \\ &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} g(\|\mathbf{X}_i - \mathbf{X}_j\|^2) \\ &= \frac{1}{2}\sum_{i=1}^{n} \sum_{j=1}^{n} g(\|\mathbf{X}_i - \mathbf{X}_j\|^2) \\ &= \frac{1}{2}\sum_{i=1}^{n} \sum_{j=1}^{n} g(\sum_{k}(X_{ik} - X_{jk})^2) \end{aligned}$$

where

$$\begin{aligned} g(y) &= \begin{cases} \frac{y}{2\delta}, \text{ if } \quad y \le \delta^2 \\ \sqrt{y} - \frac{\delta}{2}, \text{ if } \quad y > \delta^2 \end{cases} \\ &= I(y \le \delta^2)\frac{y}{2\delta} + I(y > \delta^2)[\sqrt{y} - \frac{\delta}{2}] \end{aligned}$$

and

$$y_{ij} = \sum_{k}(X_{ik} - X_{jk})^2 = \|(\mathbf{X}_i - \mathbf{X}_j)\|^2$$

is a matrix that contains the squared distance between each pair of $\mathbf{X_i}$ and $\mathbf{X_j}$. Then, the gradient is

$$
\begin{aligned}
\frac{\partial h}{\partial X_{mn}} &= \frac{1}{2}\sum_i\sum_j \frac{\partial g}{\partial y_{ij}}\left[2\sum_k (X_{ik}-X_{jk})\frac{\partial (X_{ik}-X_{jk})}{\partial X_{mn}}\right] \\
&= \sum_i\sum_j \frac{\partial g}{\partial y_{ij}}\left[\sum_k (X_{ik}-X_{jk})(\delta_{im}\delta_{kn}-\delta_{jm}\delta_{kn})\right] \\
&\qquad (Since \quad \frac{\partial g}{\partial y_{ij}} = I(y_{ij}\le\delta^2)\frac{1}{2\delta}) + I(y_{ij}>\delta^2)\frac{1}{2\sqrt{y_{ij}}})) \\
&= \frac{1}{2\delta}\sum_j I(y_{mj}\le\delta^2)X_{mn} + \sum_j I(y_{mj}>\delta^2)\frac{X_{mn}}{2\sqrt{y_{mj}}} \\
&\quad + \frac{1}{2\delta}\sum_i I(y_{im}\le\delta^2)X_{mn} + \sum_i I(y_{im})>\delta^2)\frac{X_{mn}}{2\sqrt{y_{im}}} \\
&\quad - \frac{1}{2\delta}\sum_j I(y_{mj}\le\delta^2)X_{jn} + \sum_j I(y_{mj}>\delta^2)\frac{X_{jn}}{2\sqrt{y_{mj})^2}} \\
&\quad - \frac{1}{2\delta}\sum_i I(y_{im}\le\delta^2)X_{jn} + \sum_i I(y_{im}>\delta^2)\frac{X_{in}}{2\sqrt{y_{im}}} \\
&= \frac{1}{\delta}\left[\sum_j I\left(y_{mj}\le\delta^2\right)X_{mn} - \sum_j I\left(y_{mj}\le\delta^2\right)X_{jn}\right] \\
&\quad + \left[\sum_j \frac{I(y_{mj}>\delta^2)}{\sqrt{y_{mj}}}X_{mn} - \sum_j \frac{I(y_{mj}>\delta^2)}{\sqrt{y_{mj}}}X_{jn}\right]
\end{aligned}
$$

where $\delta_{ij}=1$ if $i=j$ else 0. Based on this, the Hessian can be derived as

$$
\begin{aligned}
H &= \frac{\partial}{\partial X_{pq}}\frac{\partial h}{\partial X_{mn}} \\
&= \frac{1}{\delta}\left[\sum_j I(Y_{mj}\le\delta^2)\delta_{mp}\delta_{np} - \sum_j I(Y_{mj}\le\delta^2)\delta_{jp}\delta_{nq}\right] \\
&\quad + \left[\sum_j I(Y_{mj}>\delta^2)(\frac{\delta_{mp}\delta_{np}}{\sqrt{Y_{mj}}} - \frac{X_{mn}\sum_k (X_{mk}-X_{jk})(\delta_{mp}\delta_{kp}-\delta_{jp}\delta_{kq})}{{Y_{mj}}^{2/3}})\right] \\
&\quad - \left[\sum_j I(Y_{mj}>\delta^2)(\frac{\delta_{jp}\delta_{np}}{\sqrt{Y_{mj}}} - \frac{X_{jn}\sum_k (X_{mk}-X_{jk})(\delta_{mp}\delta_{kp}-\delta_{jp}\delta_{kq})}{{Y_{mj}}^{2/3}})\right]
\end{aligned}
$$

which is extremely difficult to implement in Numpy, so we only consider the dot product of hessian and a vector t

$$\begin{aligned}
H \cdot t &= \frac{\partial}{\partial X_{pq}}[\frac{\partial h}{\partial X_{mn}}]t_{pq} \\
&= \frac{1}{\delta}\left[\sum_j I(Y_{mj} \leq \delta^2)t_{mn} - \sum_j I(Y_{mj} \leq \delta^2)t_{jn}\right] \\
&+ \left[\sum_j I(Y_{mj} > \delta^2)(\frac{t_{mn}}{\sqrt{Y_{mj}}} - \frac{X_{mn}\sum_k (X_{mk} - X_{jk})(t_{mk} - t_{jk})}{{Y_{mj}}^{2/3}})\right] \\
&- \left[\sum_j I(Y_{mj} > \delta^2)(\frac{t_{jn}}{\sqrt{Y_{mj}}} - \frac{X_{jn}\sum_k (X_{mk} - X_{jk})(t_{mk} - t_{jk})}{{Y_{mj}}^{2/3}})\right]
\end{aligned}$$

which can now be more easily implemented in Numpy.

To sum up, both the gradient and hessian can be calcuated using only matrix calculation without any for loop, which greatly boost the performance of AGM and Newton-CG algorithm. One can refer to the open source code for a detailed implementation of them.