Supplementary Material - Adaptive Annealing for Robust Averaging

Chitturi Sidhartha and Venu Madhav Govindu

Indian Institute of Science, Bengaluru, India
{chitturis,venug}@iisc.ac.in

1 Notation

We discuss the notation that is used commonly in the main paper as well as the supplementary material. $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ is a connected weighted graph where $\mathcal{V}$, $\mathcal{E}$, $\mathcal{W}$ represent the sets of vertices, edges $(i, j) \in \mathcal{E}$ and, weights $\mathbf{W}_{ij}$ (symmetric matrices) on the edges respectively. $N = |\mathcal{V}|$, $M = |\mathcal{E}|$. The weighted graph Laplacian matrix of $\mathcal{G}$ is $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix ($\mathbf{D}_{ii} = \sum_{j \sim i} \mathbf{W}_{ij}$) and $\mathbf{A}$ is the adjacency matrix ($\mathbf{A}_{ij} = \mathbf{W}_{ij}$ if $(i, j) \in \mathcal{E}$, $\mathbf{0}$ otherwise) of $\mathcal{G}$. A matrix has Laplacian structure if it is equal to some graph Laplacian matrix. The Laplacian matrix corresponding to a single edge in the graph is called the edge Laplacian matrix. $\mathbb{J}_{\mathbf{x}}(\cdot), D_{\mathbf{x}}(\cdot)$ denote the Jacobian and differential operators respectively with respect to $\mathbf{x}$. $\mathbf{A} \otimes \mathbf{B}$ denotes the Kronecker product of two matrices $\mathbf{A}$ and $\mathbf{B}$. $\|\mathbf{p}\|$ is the 2-norm of $\mathbf{p}$. $(\mathbf{A})_{k,*}$, $(\mathbf{A})_{*,k}$ denote the k^{th} row and k^{th} column of $\mathbf{A}$ respectively. To avoid clutter, the arguments of a function are sometimes omitted, *e.g.* $\mathbf{r}_{ij}(\mathbf{x}_i, \mathbf{x}_j; \mathbf{z}_{ij})$ is referred to as $\mathbf{r}_{ij}$, $\mathbf{W}_{ij}(\sigma)$ is $\mathbf{W}_{ij}$. If not mentioned, the dependence of matrices such as $\mathbf{W}_{ij}$, $\lambda_{min}(\mathbf{W}_{ij})$, $\mathbf{H}$ on σ is understood to be implicit. A single GNC stage refers to the optimization of the robust cost $f_\sigma(\mathbf{x})$ for a fixed σ. The cardinality of the set of $\{\sigma_k\}$'s used in GNC denotes the number of GNC or annealing stages. We notate $\boldsymbol{\nabla}^2_{\mathbf{x}_i \mathbf{x}_j} f = \frac{\partial^2 f}{\partial \mathbf{x}_i \partial \mathbf{x}_j}$ and use the two definitions interchangeably. Also, we encounter large sparse matrices with non-zero entries only at the i^{th} and j^{th} blocks (row and column wise). We represent such matrices using the square 2×2 block matrices signifying the $ii^{th}, ij^{th}, ji^{th}, jj^{th}$ block non-zero entries.

The equations for the robust cost and the least squares cost for averaging problems on Euclidean spaces are:

$$\underline{\textbf{Robust Cost:}}\ f_\sigma(\mathbf{x}) = \sum_{ij \in \mathcal{E}} f_{\sigma,ij}(\mathbf{x}) = \sum_{ij \in \mathcal{E}} \rho_\sigma(\|\mathbf{r}_{ij}\|) \tag{1}$$

$$\underline{\textbf{Least Squares Cost:}}\ f^{ls}(\mathbf{x}) = \sum_{ij \in \mathcal{E}} f^{ls}_{ij}(\mathbf{x}) = \sum_{ij \in \mathcal{E}} \frac{1}{2}\|\mathbf{r}_{ij}\|^2 \tag{2}$$

Assumption 1 *The function* $\mathbf{r}_{ij} : \mathcal{D} \subset \mathbb{R}^q \times \mathbb{R}^q \to \mathbb{R}^l$ *has continuous second partial derivatives in* $\mathcal{D}$ *and* $\mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij}) = -\mathbb{J}_{\mathbf{x}_j}(\mathbf{r}_{ij})$.

In the next section, we discuss the proofs of all propositions and theorems stated in the main paper.

2 Proofs

2.1 Proof of Theorem 1

Theorem 1. *The Hessian matrix* $\mathbf{H}_{ij}(\sigma)$ *corresponding to* $f_{ij}(\mathbf{x})$ *in Eq.* (1) *is a Laplacian matrix if Assumption 1 holds true. Consequently, the Hessian of the total cost* $f_\sigma(\mathbf{x})$ *in Eq.* (1), $\mathbf{H}(\sigma)$ *is given by:*

$$\mathbf{H}(\sigma) = \sum_{ij \in \mathcal{E}} \mathbf{H}_{ij}(\sigma) = \sum_{ij \in \mathcal{E}} \mathbf{L}^e_{ij} \otimes \mathbf{W}_{ij}(\sigma) \tag{3}$$

where $\mathbf{L}^e_{ij}$ *is the* $N \times N$ *unweighted edge Laplacian matrix for a single edge* (i,j)*, given by:*

$$\mathbf{L}^e_{ij}(m,n) = \begin{cases} 1, & \textit{if } (m,n) = (i,i) \textit{ or } (m,n) = (j,j) \\ -1, & \textit{if } (m,n) = (i,j) \textit{ or } (m,n) = (j,i) \\ 0 & \textit{otherwise} \end{cases} \tag{4}$$

$\mathbf{W}_{ij}(\sigma)$ *is the weight matrix for edge* $(i,j) \in \mathcal{E}$ *and is a function of* σ*, given by:*

$$\mathbf{W}_{ij}(\sigma) = -l_{ij} \frac{\mathbb{J}^\top_{\mathbf{x}_i}(\mathbf{r}_{ij})\, \mathbf{r}_{ij}\mathbf{r}^\top_{ij} \mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij})}{\|\mathbf{r}_{ij}\|^2} + m_{ij} \frac{\partial^2}{\partial \mathbf{x}_i^2}\left(\frac{\|\mathbf{r}_{ij}\|^2}{2}\right) \tag{5}$$

where l_{ij} *and* m_{ij} *are functions depending on the robust loss* $\rho_\sigma(\cdot)$*:*

$$l_{ij} = \frac{\rho'_\sigma(\|\mathbf{r}_{ij}\|)}{\|\mathbf{r}_{ij}\|} - \rho''_\sigma(\|\mathbf{r}_{ij}\|);\, m_{ij} = \frac{\rho'_\sigma(\|\mathbf{r}_{ij}\|)}{\|\mathbf{r}_{ij}\|} \tag{6}$$

Proof. Let us notate the robust cost defined in Eq. (1) as f_σ. Since we are concerned only with a single edge $(i,j) \in \mathcal{E}$, the variable $\mathbf{x}$ is the concatenation of the two node variables $\mathbf{x}_i, \mathbf{x}_j$ *i.e.* $\mathbf{x} = \begin{bmatrix}\mathbf{x}_i^\top & \mathbf{x}_j^\top\end{bmatrix}^\top$. The gradient and the Hessian matrix of the robust cost $f_{\sigma,ij}$ is:

$$\boldsymbol{\nabla}_\mathbf{x} f_{\sigma,ij} = \rho'_\sigma(\|\mathbf{r}_{ij}\|)\, \boldsymbol{\nabla}_\mathbf{x}\|\mathbf{r}_{ij}\| = \begin{bmatrix} \rho'_\sigma(\|\mathbf{r}_{ij}\|)\, \boldsymbol{\nabla}_{\mathbf{x}_i}\|\mathbf{r}_{ij}\| \\ \rho'_\sigma(\|\mathbf{r}_{ij}\|)\, \boldsymbol{\nabla}_{\mathbf{x}_j}\|\mathbf{r}_{ij}\| \end{bmatrix} \tag{7}$$

$$\mathbf{H}_{ij}(\sigma) = D_\mathbf{x}(\boldsymbol{\nabla}_\mathbf{x} f_{\sigma,ij}) = D_\mathbf{x}\left(\rho'_\sigma(\|\mathbf{r}_{ij}\|)\, \boldsymbol{\nabla}_\mathbf{x}\|\mathbf{r}_{ij}\|\right) \tag{8}$$

Since the least squares cost $f^{ls}_{ij}(\mathbf{x}) = \frac{1}{2}\|\mathbf{r}_{ij}\|^2$, we can derive the following:

$$\boldsymbol{\nabla}_\mathbf{x}\|\mathbf{r}_{ij}\| = \frac{\boldsymbol{\nabla}_\mathbf{x} f^{ls}_{ij}}{\|\mathbf{r}_{ij}\|};\, \boldsymbol{\nabla}^2_{\mathbf{xx}}\|\mathbf{r}_{ij}\| = \frac{1}{\|\mathbf{r}_{ij}\|}\boldsymbol{\nabla}^2_{\mathbf{xx}} f^{ls}_{ij} - \frac{1}{\|\mathbf{r}_{ij}\|^3}\boldsymbol{\nabla}_\mathbf{x} f^{ls}_{ij} \boldsymbol{\nabla}_\mathbf{x} {f^{ls}_{ij}}^\top \tag{9}$$

Thus, the expressions for the Hessian becomes:

$$\begin{aligned}
\mathbf{H}_{ij}(\sigma) &= \boldsymbol{\nabla}_\mathbf{x}\|\mathbf{r}_{ij}\| \boldsymbol{\nabla}_\mathbf{x}\|\mathbf{r}_{ij}\|^\top \rho''_\sigma(\|\mathbf{r}_{ij}\|) + \rho'_\sigma(\|\mathbf{r}_{ij}\|)\, \boldsymbol{\nabla}^2_{\mathbf{xx}}\|\mathbf{r}_{ij}\| \\
&= \frac{\boldsymbol{\nabla}_\mathbf{x} f^{ls}_{ij} \boldsymbol{\nabla}_\mathbf{x} {f^{ls}_{ij}}^\top}{\|\mathbf{r}_{ij}\|^2} \rho''_\sigma(\|\mathbf{r}_{ij}\|) + \rho'_\sigma(\|\mathbf{r}_{ij}\|) \left[\frac{\boldsymbol{\nabla}^2_{\mathbf{xx}} f^{ls}_{ij}}{\|\mathbf{r}_{ij}\|} - \frac{\boldsymbol{\nabla}_\mathbf{x} f^{ls}_{ij} \boldsymbol{\nabla}_\mathbf{x} {f^{ls}_{ij}}^\top}{\|\mathbf{r}_{ij}\|^3}\right] \\
&= \left(\frac{\rho''_\sigma(\|\mathbf{r}_{ij}\|)}{\|\mathbf{r}_{ij}\|^2} - \frac{\rho'_\sigma(\|\mathbf{r}_{ij}\|)}{\|\mathbf{r}_{ij}\|^3}\right) \boldsymbol{\nabla}_\mathbf{x} f^{ls}_{ij} \boldsymbol{\nabla}_\mathbf{x} {f^{ls}_{ij}}^\top + \frac{\rho'_\sigma(\|\mathbf{r}_{ij}\|)}{\|\mathbf{r}_{ij}\|} \boldsymbol{\nabla}^2_{\mathbf{xx}} f^{ls}_{ij}
\end{aligned}$$

$$\Rightarrow \mathbf{H}_{ij}(\sigma) = -l_{ij}\frac{\boldsymbol{\nabla}_{\mathbf{x}} f_{ij}^{ls} \boldsymbol{\nabla}_{\mathbf{x}} f_{ij}^{ls\top}}{\|\mathbf{r}_{ij}\|^2} + m_{ij}\boldsymbol{\nabla}^2_{\mathbf{xx}} f_{ij}^{ls} \tag{10}$$

where, l_{ij}, m_{ij} are defined in Eq. (6).

We have to prove that if $\mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij}) = -\mathbb{J}_{\mathbf{x}_j}(\mathbf{r}_{ij})$, then the robust Hessian $\mathbf{H}_{ij}(\sigma)$ is a Laplacian matrix. If we can prove that the two terms in Eq. (10) are individually Laplacian matrices, then the robust Hessian is also Laplacian. For that, we look at the gradient and Hessian for the least squares cost as follows:

$$\boldsymbol{\nabla}_{\mathbf{x}} f_{ij}^{ls}(\mathbf{x}) = \begin{bmatrix} \boldsymbol{\nabla}_{\mathbf{x}_i} f_{ij}^{ls}(\mathbf{x}) \\ \boldsymbol{\nabla}_{\mathbf{x}_j} f_{ij}^{ls}(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \mathbb{J}_{\mathbf{x}_i}^\top(\mathbf{r}_{ij})\,\mathbf{r}_{ij} \\ \mathbb{J}_{\mathbf{x}_j}^\top(\mathbf{r}_{ij})\,\mathbf{r}_{ij} \end{bmatrix} \tag{11}$$

It is easy to observe that from Eq. (11), the first term in the expression for $\mathbf{H}_{ij}(\sigma)$ in Eq. (10) is Laplacian in structure under the assumption that $\mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij}) = -\mathbb{J}_{\mathbf{x}_j}(\mathbf{r}_{ij})$. For the least squares Hessian,

$$\boldsymbol{\nabla}^2_{\mathbf{xx}} f_{ij}^{ls} = \begin{bmatrix} \boldsymbol{\nabla}^2_{\mathbf{x}_i\mathbf{x}_i} f_{ij}^{ls} & \boldsymbol{\nabla}^2_{\mathbf{x}_j\mathbf{x}_i} f_{ij}^{ls} \\ \boldsymbol{\nabla}^2_{\mathbf{x}_i\mathbf{x}_j} f_{ij}^{ls} & \boldsymbol{\nabla}^2_{\mathbf{x}_j\mathbf{x}_j} f_{ij}^{ls} \end{bmatrix} \tag{12}$$

$$\boldsymbol{\nabla}^2_{\mathbf{x}_i\mathbf{x}_i} f_{ij}^{ls} = D_{\mathbf{x}_i}\left(\mathbb{J}_{\mathbf{x}_i}^\top(\mathbf{r}_{ij})\,\mathbf{r}_{ij}\right)$$

$$\text{Let } \mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij}) = \mathbf{A} = \begin{bmatrix} | & | & | & \cdots \\ \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 & \cdots \\ | & | & | & \cdots \end{bmatrix}, \mathbb{J}_{\mathbf{x}_j}(\mathbf{r}_{ij}) = \mathbf{B} = \begin{bmatrix} | & | & | & \cdots \\ \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 & \cdots \\ | & | & | & \cdots \end{bmatrix}$$

$$\text{where } \mathbf{a}_k = \left[\frac{\partial r_1}{\partial x_{ik}}, \frac{\partial r_2}{\partial x_{ik}}, \cdots, \frac{\partial r_l}{\partial x_{ik}}\right]^\top, \mathbf{b}_k = \left[\frac{\partial r_1}{\partial x_{jk}}, \frac{\partial r_2}{\partial x_{jk}}, \cdots, \frac{\partial r_l}{\partial x_{jk}}\right]^\top$$

$$\Rightarrow \boldsymbol{\nabla}^2_{\mathbf{x}_i\mathbf{x}_i} f_{ij}^{ls} = \begin{bmatrix} \vdots \\ D_{\mathbf{x}_i}\left(\mathbf{r}_{ij}^\top \mathbf{a}_k\right) \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \mathbf{a}_k^\top \mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij}) + \mathbf{r}_{ij}^\top D_{\mathbf{x}_i}(\mathbf{a}_k) \\ \vdots \end{bmatrix} \left(\because \mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij}) = D_{\mathbf{x}_i}(\mathbf{r}_{ij})\right)$$

$$\Rightarrow \left(\boldsymbol{\nabla}^2_{\mathbf{x}_i\mathbf{x}_i} f_{ij}^{ls}\right)_{k,*} = \mathbf{r}_{ij}^\top D_{\mathbf{x}_i}(\mathbf{a}_k) + \mathbf{a}_k^\top \mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij})$$

Similarly, we can derive the expressions

$$\left(\boldsymbol{\nabla}^2_{\mathbf{x}_j\mathbf{x}_i} f_{ij}^{ls}\right)_{k,*} = \mathbf{r}_{ij}^\top D_{\mathbf{x}_j}(\mathbf{a}_k) + \mathbf{a}_k^\top \mathbb{J}_{\mathbf{x}_j}(\mathbf{r}_{ij}) \tag{13}$$

$$\left(\boldsymbol{\nabla}^2_{\mathbf{x}_i\mathbf{x}_j} f_{ij}^{ls}\right)_{k,*} = \mathbf{r}_{ij}^\top D_{\mathbf{x}_i}(\mathbf{b}_k) + \mathbf{b}_k^\top \mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij}) \tag{14}$$

$$\left(\boldsymbol{\nabla}^2_{\mathbf{x}_j\mathbf{x}_j} f_{ij}^{ls}\right)_{k,*} = \mathbf{r}_{ij}^\top D_{\mathbf{x}_j}(\mathbf{b}_k) + \mathbf{b}_k^\top \mathbb{J}_{\mathbf{x}_j}(\mathbf{r}_{ij}) \tag{15}$$

For the matrix $\boldsymbol{\nabla}^2_{\mathbf{x}\mathbf{x}} f^{ls}_{ij}$ to be Laplacian, we require $\left(\boldsymbol{\nabla}^2_{\mathbf{x}_i\mathbf{x}_i} f^{ls}_{ij}\right)_{k,*} + \left(\boldsymbol{\nabla}^2_{\mathbf{x}_j\mathbf{x}_i} f^{ls}_{ij}\right)_{k,*} = \mathbf{0}$.

$$\left(\boldsymbol{\nabla}^2_{\mathbf{x}_i\mathbf{x}_i} f^{ls}_{ij}\right)_{k,*} + \left(\boldsymbol{\nabla}^2_{\mathbf{x}_j\mathbf{x}_i} f^{ls}_{ij}\right)_{k,*} \tag{16}$$

$$= \mathbf{r}_{ij}^\top D_{\mathbf{x}_i}(\mathbf{a}_k) + \mathbf{a}_k^\top \mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij}) + \mathbf{r}_{ij}^\top D_{\mathbf{x}_j}(\mathbf{a}_k) + \mathbf{a}_k^\top \mathbb{J}_{\mathbf{x}_j}(\mathbf{r}_{ij}) \tag{17}$$

$$= \mathbf{r}_{ij}^\top \left(D_{\mathbf{x}_i}(\mathbf{a}_k) + D_{\mathbf{x}_j}(\mathbf{a}_k)\right) + \mathbf{a}_k^\top \left(\mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij}) + \mathbb{J}_{\mathbf{x}_j}(\mathbf{r}_{ij})\right) \tag{18}$$

$$= \mathbf{r}_{ij}^\top \left(D_{\mathbf{x}_i}(\mathbf{a}_k) + D_{\mathbf{x}_j}(\mathbf{a}_k)\right) \quad \left(\because \left(\mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij}) + \mathbb{J}_{\mathbf{x}_j}(\mathbf{r}_{ij})\right) = 0 \text{ by Assumption 1 }\right) \tag{19}$$

$$= 0 \Leftarrow D_{\mathbf{x}_i}(\mathbf{a}_k) + D_{\mathbf{x}_j}(\mathbf{a}_k) = 0 \tag{20}$$

$$\Leftarrow \frac{\partial}{\partial x_{is}}\left(\frac{\partial r_t}{\partial x_{ik}}\right) + \frac{\partial}{\partial x_{js}}\left(\frac{\partial r_t}{\partial x_{ik}}\right) = 0 \tag{21}$$

$$\forall s \in \{1, \ldots, q\}, t \in \{1, \ldots, l\}. \tag{22}$$

From Assumption-1, the function $\mathbf{r}_{ij}$ has continuous second partial derivatives, because of which the derivatives order is interchangeable *i.e.* $\frac{\partial}{\partial x_{is}}(\frac{\partial r_t}{\partial x_{ik}}) = \frac{\partial}{\partial x_{ik}}(\frac{\partial r_t}{\partial x_{is}})$. Thus, the above equation becomes,

$$\Leftarrow \frac{\partial}{\partial x_{ik}}\left(\frac{\partial r_t}{\partial x_{is}}\right) + \frac{\partial}{\partial x_{ik}}\left(\frac{\partial r_t}{\partial x_{js}}\right) = 0$$

$$\Leftarrow \frac{\partial}{\partial x_{ik}}\left(\frac{\partial r_t}{\partial x_{is}} + \frac{\partial r_t}{\partial x_{js}}\right) = 0$$

$$\Leftarrow \mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij}) + \mathbb{J}_{\mathbf{x}_j}(\mathbf{r}_{ij}) = 0 \quad \Big(\because \mathbf{A} + \mathbf{B} = 0 \Rightarrow \mathbf{A}_{ts} + \mathbf{B}_{ts} = 0 \Rightarrow \frac{\partial r_t}{\partial x_{is}} + \frac{\partial r_t}{\partial x_{js}} = 0 \quad \forall s, t\Big).$$

Likewise, the conditions $\left(\boldsymbol{\nabla}^2_{\mathbf{x}_j\mathbf{x}_j} f^{ls}_{ij}\right)_{k,*} = \left(\boldsymbol{\nabla}^2_{\mathbf{x}_i\mathbf{x}_i} f^{ls}_{ij}\right)_{k,*}$ and $\left(\boldsymbol{\nabla}^2_{\mathbf{x}_j\mathbf{x}_j} f^{ls}_{ij}\right)_{k,*} = -\left(\boldsymbol{\nabla}^2_{\mathbf{x}_i\mathbf{x}_j} f^{ls}_{ij}\right)_{k,*}$ can also be proved when $\mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij}) = -\mathbb{J}_{\mathbf{x}_j}(\mathbf{r}_{ij})$, thus proving that $\boldsymbol{\nabla}^2_{\mathbf{x}\mathbf{x}} f^{ls}_{ij}$ is a Laplacian matrix. Thus the second term in the expression of $\mathbf{H}_{ij}(\sigma)$ is also Laplacian, thereby proving that the robust Hessian $\mathbf{H}_{ij}(\sigma)$ is a Laplacian matrix.

Given that $\mathbf{H}_{ij}(\sigma)$ is a Laplacian matrix, it is easy to deduce from Eq. (10) that,

$$\mathbf{H}_{ij}(\sigma) = \begin{bmatrix} \mathbf{W}_{ij}(\sigma) & -\mathbf{W}_{ij}(\sigma) \\ -\mathbf{W}_{ij}(\sigma) & \mathbf{W}_{ij}(\sigma) \end{bmatrix} \tag{23}$$

where

$$\mathbf{W}_{ij}(\sigma) = -l_{ij}\frac{\mathbb{J}_{\mathbf{x}_i}^\top(\mathbf{r}_{ij})\,\mathbf{r}_{ij}\mathbf{r}_{ij}^\top \mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij})}{\|\mathbf{r}_{ij}\|^2} + m_{ij}\frac{\partial^2}{\partial \mathbf{x}_i^2}\left(\frac{\|\mathbf{r}_{ij}\|^2}{2}\right) \tag{24}$$

The above expressions consider the variable $\mathbf{x} = \left[\mathbf{x}_i^\top, \mathbf{x}_j^\top\right]^\top$ for a single edge, but if $\mathbf{x}$ is now treated as the concatenation of all node variables $\{\mathbf{x}_i\}$'s $i \in \{1, 2, ..., N\}$, then we get the Kronecker product form as shown in Eq. (3).

2.2 Proof of Proposition 1

Proposition 1 (Laplacian Properties [2]).

1. *A Laplacian matrix with the weight matrices* $\{\mathbf{W}_{ij}\}_{(i,j)\in\mathcal{E}}$ *is positive semidefinite if* $\mathbf{W}_{ij} \succcurlyeq \mathbf{0}\ \forall\,(i,j) \in \mathcal{E}$.
2. *A Laplacian matrix always has 0 as one of its eigen values and the corresponding eigen vector is the ones vector* i.e. $\mathbf{1} = \begin{bmatrix}1, 1, \ldots, 1\end{bmatrix}^\top$

Proof. The proposition is a classical result of Spectral Graph Theory available in [2].

2.3 Proof of Proposition 2

The vector averaging cost is given as:

$$f(\mathbf{x}) = \sum_{ij\in\mathcal{E}} \rho_{\sigma_k}(\|\mathbf{r}_{ij}\|) = \rho_{\sigma_k}(\|\mathbf{x}_j - \mathbf{x}_i - \mathbf{z}_{ij}\|) \tag{25}$$

Proposition 2. *The weight matrix* $\mathbf{W}_{ij} \in \mathbb{R}^{q\times q}$ *corresponding to the cost Eq.* (25) *is given by:*

$$\mathbf{W}_{ij} = m_{ij}\mathbf{I} - l_{ij}\frac{\mathbf{r}_{ij}\mathbf{r}_{ij}^\top}{\|\mathbf{r}_{ij}\|^2} \tag{26}$$

and for the Geman-McClure loss, $\mathbf{W}_{ij} \succcurlyeq \mathbf{0}$ *iff* $\sigma \geq \sqrt{3}\|\mathbf{r}_{ij}\|$, *implying* $\sigma_{ij} = \sqrt{3}\|\mathbf{r}_{ij}\|$.

Proof. We have $\mathbf{r}_{ij} = \mathbf{x}_j - \mathbf{x}_i - \mathbf{z}_{ij}$, $\Rightarrow \mathbb{J}_{\mathbf{x}_i}(\mathbf{r}_{ij}) = -\mathbb{J}_{\mathbf{x}_j}(\mathbf{r}_{ij}) = -\mathbf{I}$.

$$f_{ij}^{ls} = \frac{1}{2}\|\mathbf{x}_j - \mathbf{x}_i - \mathbf{z}_{ij}\|^2 \tag{27}$$

$$\Rightarrow \boldsymbol{\nabla}_{\mathbf{x}_i} f_{ij}^{ls} = -(\mathbf{x}_j - \mathbf{x}_i - \mathbf{z}_{ij}) = -\mathbf{r}_{ij} \tag{28}$$

$$\Rightarrow \frac{\partial^2}{\partial \mathbf{x}_i^2}\left(\frac{\|\mathbf{r}_{ij}\|^2}{2}\right) = D_{\mathbf{x}_i}\left(\boldsymbol{\nabla}_{\mathbf{x}_i} f_{ij}^{ls}\right) = \mathbf{I} \tag{29}$$

Thus, $\mathbf{W}_{ij}$ follows from Eq. (5). For proving the conditions under which $\mathbf{W}_{ij} \succcurlyeq \mathbf{0}$, we shall drop the subscript ij for convenience.

$$\mathbf{W}_{ij} = m\mathbf{I} - l\frac{\mathbf{r}\mathbf{r}^\top}{\|\mathbf{r}\|^2} \succcurlyeq \mathbf{0} \tag{30}$$

$$\iff \mathbf{I} - \left(\frac{l}{m}\right)\frac{\mathbf{r}\mathbf{r}^\top}{\|\mathbf{r}\|^2} \succcurlyeq \mathbf{0} \quad (\because \mathbf{m} \geq \mathbf{0}) \tag{31}$$

$$\iff \det\left(\mathbf{I} - \left(\frac{l}{m}\right)\frac{\mathbf{r}\mathbf{r}^\top}{\|\mathbf{r}\|^2} - \lambda\mathbf{I}\right) = 0 \tag{32}$$

$$\iff \det\left(\left(\frac{l}{m}\right)\frac{\mathbf{r}\mathbf{r}^\top}{\|\mathbf{r}\|^2} - (1-\lambda)\mathbf{I}\right) = 0 \tag{33}$$

If λ denote the eigenvalues of $\frac{\mathbf{W}_{ij}}{m} = \mathbf{I} - \left(\frac{l}{m}\right)\frac{\mathbf{r}\mathbf{r}^\top}{\|\mathbf{r}\|^2}$, $(1-\lambda)$ are the eigenvalues of $\left(\frac{l}{m}\right)\frac{\mathbf{r}\mathbf{r}^\top}{\|\mathbf{r}\|^2}$ which is nothing but an outer product. Therefore,

$$(1-\lambda) = \frac{l}{m}, 0, 0, \dots, 0 \tag{34}$$

$$\Rightarrow \lambda = 1 - \frac{l}{m}, 1, 1, \dots, 1 \tag{35}$$

$$\Rightarrow \mathbf{W}_{ij} \succcurlyeq \mathbf{0} \iff 1 - \frac{l}{m} \geq 0 \tag{36}$$

Specifically for Geman-McClure loss function,

$$1 - \frac{l}{m} = \frac{1 - 3\frac{\|\mathbf{r}_{ij}\|^2}{\sigma^2}}{1 + \frac{\|\mathbf{r}_{ij}\|^2}{\sigma^2}} \geq 0 \tag{37}$$

$$\iff 1 - 3\frac{\|\mathbf{r}_{ij}\|^2}{\sigma^2} \geq 0 \tag{38}$$

$$\iff \sigma \geq \sqrt{3}\|\mathbf{r}_{ij}\| \tag{39}$$

$$\text{Thus,} \quad \sigma_{ij} = \sqrt{3}\|\mathbf{r}_{ij}\| \tag{40}$$

2.4 Proof of Proposition 3

The averaging cost with additional edge variables $\mathbf{x}_{ij} \in \mathbb{R}^s$ be defined as:

$$f_\sigma(\mathbf{x}) = \sum_{ij\in\mathcal{E}} \rho_\sigma\left(\|\mathbf{r}_{ij}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_{ij}; \mathbf{z}_{ij})\|\right) \tag{41}$$

Proposition 3. *If Assumption 1 holds true, the Hessian for the cost Eq.* (41) *is the following block matrix:*

$$\mathbf{H} = \begin{bmatrix} \mathbf{L} & -\mathbf{B} \\ -\mathbf{B}^\top & \mathbf{D} \end{bmatrix} \tag{42}$$

where L_t is a Laplacian matrix, $\mathbf{B}$ has the structure similar to an incidence matrix of the graph $\mathcal{G}$, $\mathbf{D}$ is a block diagonal matrix corresponding to the edge variables $\{\mathbf{x}_{ij}\}$'s. The matrix $\mathbf{L}_r = \mathbf{L} - \mathbf{B}\mathbf{D}^{-1}\mathbf{B}^\top$ has Laplacian structure. The positive semidefiniteness of $\mathbf{H}$, i.e. $\mathbf{H} \succcurlyeq \mathbf{0}$, *boils down to satisfying the following two conditions:*

1. *All block diagonal entries in $\mathbf{D}$ are positive definite.*
2. $\mathbf{L} - \mathbf{B}\mathbf{D}^{-1}\mathbf{B}^\top \succcurlyeq \mathbf{0}$.

Proof. It is easy to observe that

$$\mathbf{L}_{t,ij} = \begin{bmatrix} \nabla^2_{\mathbf{x}_i\mathbf{x}_i} f_{\sigma,ij} & \nabla^2_{\mathbf{x}_j\mathbf{x}_i} f_{\sigma,ij} \\ \nabla^2_{\mathbf{x}_i\mathbf{x}_j} f_{\sigma,ij} & \nabla^2_{\mathbf{x}_j\mathbf{x}_j} f_{\sigma,ij} \end{bmatrix}, \mathbf{B}_{ij} = -\begin{bmatrix} \nabla^2_{\mathbf{x}_{ij}\mathbf{x}_i} f_{\sigma,ij} \\ \nabla^2_{\mathbf{x}_{ij}\mathbf{x}_j} f_{\sigma,ij} \end{bmatrix}, \mathbf{D}_{ij} = \nabla^2_{\mathbf{x}_{ij}\mathbf{x}_{ij}} f_{\sigma,ij} \tag{43}$$

Since the cost fn. Eq. (41) as a function of the node variables $\{\mathbf{x}_i\}$'s satisfies Assumption-1, the block of the Hessian formed by the $\{\mathbf{x}_i\}$'s *i.e.* $\mathbf{L}_{t,ij}$ has a Laplacian structure, and hence the sum of all terms $\mathbf{L}$ is Laplacian. Evidently, the matrix $\mathbf{D}$ is a block diagonal matrix formed with $\mathbf{D}_{ij}$'s as the diagonal blocks. Given that, $\mathbf{x}_i \in \mathbb{R}^q\ \forall i$, $\mathbf{x}_{ij} \in \mathbb{R}^s$ $\forall\,(i,j) \in \mathcal{E}$, we note that $\mathbf{B}_{ij}$ is of dimension $qN \times s$, and has entries corresponding to only $\mathbf{x}_i$ and $\mathbf{x}_j$, at the i^{th} and j^{th} blocks (rowwise) respectively, each of size $q \times s$. Also, by Assumption- 1, it is easy to deduce that $\mathbf{B}_{ij}$ has entries with equal magnitude and opposite sign in the i^{th} and j^{th} blocks (rowwise). Thus, $\mathbf{B}$ has the structure of an incidence matrix of the graph $\mathcal{G}$, with dimension of $qN \times Ms$. Since $\mathbf{B}$ has the structure of an incidence matrix, and $\mathbf{D}^{-1}$ is a diagonal matrix, the matrix $\mathbf{B}\mathbf{D}^{-1}\mathbf{B}^\top$ has Laplacian structure thus making $\mathbf{L}_r = \mathbf{L} - \mathbf{B}\mathbf{D}^{-1}\mathbf{B}^\top$ also a Laplacian matrix.

The latter part of the proposition regarding the equivalent conditions for the positive semidefiniteness of the block matrix ($\mathbf{H} \succcurlyeq \mathbf{0}$) is popularly known [7], obtained using the Schur complement.

2.5 Proof of Theorem 2

Given that $\|\mathbf{t}_{ij}\| = 1\ \forall\,(i,j) \in \mathcal{E}$, the translation averaging problem is given by:

$$\begin{aligned} &\min_{\{\mathbf{T}_i\},\{d_{ij}\}} \sum_{ij\in\mathcal{E}} \rho_\sigma\left(\|\mathbf{T}_i - \mathbf{T}_j - d_{ij}\mathbf{t}_{ij}\|\right) \\ &\text{s.t.} \sum_{i=1}^{N} \mathbf{T}_i = 0; \quad d_{ij} \geq 1\ \forall\,(i,j) \in \mathcal{E} \end{aligned} \tag{44}$$

Theorem 2. *The weight matrix $\mathbf{W}_{ij}$ corresponding to the reduced Laplacian matrix $\mathbf{L}_r$ for the Translation averaging cost Eq.* (44) *is given as: ($\hat{\mathbf{r}}_{ij} = \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|}$, $q_{ij} = \frac{1}{-l_{ij}(\mathbf{t}_{ij}^\top\hat{\mathbf{r}}_{ij})^2 + m_{ij}}$)*

$$\mathbf{W}_{ij} = \begin{cases} -l_{ij}\hat{\mathbf{r}}_{ij}\hat{\mathbf{r}}_{ij}^\top + m_{ij}\mathbf{I} - q_{ij}\left(l_{ij}^2(\hat{\mathbf{r}}_{ij}^\top\mathbf{t}_{ij})^2\hat{\mathbf{r}}_{ij}\hat{\mathbf{r}}_{ij}^\top + m_{ij}^2\mathbf{t}_{ij}\mathbf{t}_{ij}^\top \right. & \\ \quad \left. -l_{ij}m_{ij}(\hat{\mathbf{r}}_{ij}^\top\mathbf{t}_{ij})\left(\hat{\mathbf{r}}_{ij}\mathbf{t}_{ij}^\top + \mathbf{t}_{ij}\hat{\mathbf{r}}_{ij}^\top\right)\right), & \text{if } d_{ij} > 1 \\ -l_{ij}\hat{\mathbf{r}}_{ij}\hat{\mathbf{r}}_{ij}^\top + m_{ij}\mathbf{I}, & \text{otherwise} \end{cases} \tag{45}$$

For the specific case of Geman-McClure loss, $\mathbf{W}_{ij} \succcurlyeq \mathbf{0}$ iff $\sigma \geq \sqrt{3}\|\mathbf{r}_{ij}\|$.

Proof. If $f_{\sigma,ij}$ denotes the cost function for a single term in Eq. (44), then the Hessian matrix is given by:

$$\mathbf{H}_{ij} = \begin{bmatrix} \nabla^2_{\mathbf{T}_i\mathbf{T}_i} f_{\sigma,ij} & \nabla^2_{\mathbf{T}_j\mathbf{T}_i} f_{\sigma,ij} & \nabla^2_{d_{ij}\mathbf{T}_i} f_{\sigma,ij} \\ \nabla^2_{\mathbf{T}_i\mathbf{T}_j} f_{\sigma,ij} & \nabla^2_{\mathbf{T}_j\mathbf{T}_j} f_{\sigma,ij} & \nabla^2_{d_{ij}\mathbf{T}_j} f_{\sigma,ij} \\ \nabla^2_{\mathbf{T}_i d_{ij}} f_{\sigma,ij} & \nabla^2_{\mathbf{T}_j d_{ij}} f_{\sigma,ij} & \nabla^2_{d_{ij}d_{ij}} f_{\sigma,ij} \end{bmatrix} \tag{46}$$

$$= \begin{bmatrix} \mathbf{L}_{t,ij} & \mathbf{B}_{ij} \\ \mathbf{B}_{ij}^\top & \mathbf{D}_{ij} \end{bmatrix}, \text{ where } \mathbf{D}_{ij} = \nabla^2_{d_{ij}d_{ij}} f_{\sigma,ij}, \tag{47}$$

$$\mathbf{L}_{t,ij} = \begin{bmatrix} \nabla^2_{\mathbf{T}_i\mathbf{T}_i} f_{\sigma,ij} & \nabla^2_{\mathbf{T}_j\mathbf{T}_i} f_{\sigma,ij} \\ \nabla^2_{\mathbf{T}_i\mathbf{T}_j} f_{\sigma,ij} & \nabla^2_{\mathbf{T}_j\mathbf{T}_j} f_{\sigma,ij} \end{bmatrix}, \mathbf{B}_{ij} = \begin{bmatrix} \nabla^2_{d_{ij}\mathbf{T}_i} f_{\sigma,ij} \\ \nabla^2_{d_{ij}\mathbf{T}_j} f_{\sigma,ij} \end{bmatrix} \tag{48}$$

To avoid clutter, we shall drop the ij subscript, wherever it is unambiguous. Let us denote the least squares version (f_{ij}^{ls}) of the edgewise TA cost fn. Eq. (44) by f. Then, similar to the proof of Theorem-1, we have

$$\begin{aligned}\mathbf{H}_{ij} = m_{ij} &\begin{bmatrix} \boldsymbol{\nabla}^2_{\mathbf{T}_i\mathbf{T}_i} f & \boldsymbol{\nabla}^2_{\mathbf{T}_j\mathbf{T}_i} f & \boldsymbol{\nabla}^2_{d_{ij}\mathbf{T}_i} f \\ \boldsymbol{\nabla}^2_{\mathbf{T}_i\mathbf{T}_j} f & \boldsymbol{\nabla}^2_{\mathbf{T}_j\mathbf{T}_j} f & \boldsymbol{\nabla}^2_{d_{ij}\mathbf{T}_j} f \\ \boldsymbol{\nabla}^2_{\mathbf{T}_i d_{ij}} f & \boldsymbol{\nabla}^2_{\mathbf{T}_j d_{ij}} f & \boldsymbol{\nabla}^2_{d_{ij} d_{ij}} f \end{bmatrix} \\ &- \frac{l_{ij}}{\|\mathbf{r}_{ij}\|^2} \begin{bmatrix} \boldsymbol{\nabla}_{\mathbf{T}_i} f \cdot \boldsymbol{\nabla}_{\mathbf{T}_i} f^\top & \boldsymbol{\nabla}_{\mathbf{T}_i} f \cdot \boldsymbol{\nabla}_{\mathbf{T}_j} f^\top & \boldsymbol{\nabla}_{\mathbf{T}_i} f \cdot \boldsymbol{\nabla}_{d_{ij}} f \\ \boldsymbol{\nabla}_{\mathbf{T}_j} f \cdot \boldsymbol{\nabla}_{\mathbf{T}_i} f^\top & \boldsymbol{\nabla}_{\mathbf{T}_j} f \cdot \boldsymbol{\nabla}_{\mathbf{T}_j} f^\top & \boldsymbol{\nabla}_{\mathbf{T}_j} f \cdot \boldsymbol{\nabla}_{d_{ij}} f \\ \boldsymbol{\nabla}_{d_{ij}} f \cdot \boldsymbol{\nabla}_{\mathbf{T}_i} f^\top & \boldsymbol{\nabla}_{d_{ij}} f \cdot \boldsymbol{\nabla}_{\mathbf{T}_j} f^\top & \left(\boldsymbol{\nabla}_{d_{ij}} f\right)^2 \end{bmatrix} \\ &= m_{ij} \begin{bmatrix} \mathbf{L}_1 & \mathbf{B}_1 \\ \mathbf{B}_1^\top & \mathbf{D}_1 \end{bmatrix} - l_{ij} \begin{bmatrix} \mathbf{L}_2 & \mathbf{B}_2 \\ \mathbf{B}_2^\top & \mathbf{D}_2 \end{bmatrix} \end{aligned} \tag{49}$$

, where $\mathbf{L}_1, \mathbf{L}_2, \mathbf{B}_1, \mathbf{B}_2, \mathbf{D}_1, \mathbf{D}_2$ are appropriately defined.

The expressions for each of the first and second order derivative terms are as follows:

$$\boldsymbol{\nabla}_{\mathbf{T}_i} f = \mathbf{r}_{ij}; \boldsymbol{\nabla}_{\mathbf{T}_j} f = -\mathbf{r}_{ij}; \boldsymbol{\nabla}_{d_{ij}} f = -\mathbf{r}_{ij}^\top \mathbf{t}_{ij} \tag{50}$$

$$\boldsymbol{\nabla}^2_{\mathbf{T}_i\mathbf{T}_i} f = \boldsymbol{\nabla}^2_{\mathbf{T}_j\mathbf{T}_j} f = \mathbf{I} \tag{51}$$

$$\boldsymbol{\nabla}^2_{d_{ij}\mathbf{T}_i} f = -\mathbf{t}_{ij}; \boldsymbol{\nabla}^2_{d_{ij}\mathbf{T}_j} f = \mathbf{t}_{ij}; \boldsymbol{\nabla}^2_{d_{ij}d_{ij}} f = 1 \tag{52}$$

Using Eqns 49, 50, 51, 52, we obtain the following:

$$\mathbf{B}_1 = \begin{bmatrix} -\mathbf{t}_{ij} \\ \mathbf{t}_{ij} \end{bmatrix}, \quad \mathbf{B}_2 = \frac{1}{\|\mathbf{r}_{ij}\|^2} \begin{bmatrix} -(\mathbf{r}_{ij}^\top \mathbf{t}_{ij})\mathbf{r}_{ij} \\ (\mathbf{r}_{ij}^\top \mathbf{t}_{ij})\mathbf{r}_{ij} \end{bmatrix} \tag{53}$$

$$\mathbf{L}_1 = \begin{bmatrix} \mathbf{I} & -\mathbf{I} \\ -\mathbf{I} & \mathbf{I} \end{bmatrix}, \mathbf{L}_2 = \begin{bmatrix} \hat{\mathbf{r}}_{ij}\hat{\mathbf{r}}_{ij}^\top & -\hat{\mathbf{r}}_{ij}\hat{\mathbf{r}}_{ij}^\top \\ -\hat{\mathbf{r}}_{ij}\hat{\mathbf{r}}_{ij}^\top & \hat{\mathbf{r}}_{ij}\hat{\mathbf{r}}_{ij}^\top \end{bmatrix} \tag{54}$$

$$\mathbf{D}_1 = 1, \mathbf{D}_2 = \left(\hat{\mathbf{r}}_{ij}^\top \mathbf{t}_{ij}\right)^2 \tag{55}$$

Therefore, using Eq. (49),

$$\mathbf{H}_{ij} = \begin{bmatrix} m\mathbf{L}_1 - l\mathbf{L}_2 & m\mathbf{B}_1 - l\mathbf{B}_2 \\ (m\mathbf{B}_1 - l\mathbf{B}_2)^\top & m\mathbf{D}_1 - l\mathbf{D}_2 \end{bmatrix}$$

By the 2nd condition in Proposition 3, the reduced "Hessian" (Laplacian) matrix is

$$\mathbf{L}_r = (m\mathbf{L}_1 - l\mathbf{L}_2) - (m\mathbf{B}_1 - l\mathbf{B}_2)(m\mathbf{D}_1 - l\mathbf{D}_2)^{-1}(m\mathbf{B}_1 - l\mathbf{B}_2)^\top \tag{56}$$

Since $\mathbf{L}_r$ is a Laplacian matrix, the corresponding weight matrices $\mathbf{W}_{ij}$ can be derived using eqns. 53, 54, 55 as follows:

$$\mathbf{W}_{ij} = \left(m\mathbf{I} - l\hat{\mathbf{r}}_{ij}\hat{\mathbf{r}}_{ij}^\top\right) - \frac{\left(l^2\left(\hat{\mathbf{r}}_{ij}^\top \mathbf{t}_{ij}\right)^2 \hat{\mathbf{r}}_{ij}\hat{\mathbf{r}}_{ij}^\top + m^2 \mathbf{t}_{ij}\mathbf{t}_{ij}^\top - lm\hat{\mathbf{r}}_{ij}^\top \mathbf{t}_{ij}\left(\mathbf{t}_{ij}\hat{\mathbf{r}}_{ij}^\top + \hat{\mathbf{r}}_{ij}\mathbf{t}_{ij}^\top\right)\right)}{\left(m - l\left(\hat{\mathbf{r}}_{ij}^\top \mathbf{t}_{ij}\right)^2\right)} \tag{57}$$

Since this is a constrained problem, instead of the positive semidefiniteness of the Hessian matrix, the second-order sufficient conditions is used, which translates to the positive semidefiniteness of $\mathbf{Z}(\mathbf{x}^*)^\top \nabla^2_{\mathbf{xx}} f_\sigma(\mathbf{x}^*)\mathbf{Z}(\mathbf{x}^*)$, where $\nabla^2_{\mathbf{xx}} f_\sigma(\mathbf{x}^*)$ is the actual Hessian matrix and $\mathbf{Z}(\mathbf{x}^*)$ is the basis of the null-space of the Jacobian of the active constraints at the solution $\mathbf{x}^*$. In this particular problem, effectively, the matrix $\mathbf{Z}(\mathbf{x}^*)^\top \nabla^2_{\mathbf{xx}} f_\sigma(\mathbf{x}^*)\mathbf{Z}(\mathbf{x}^*)$ is obtained by removing the rows and columns in $\mathbf{B}^\top$ and $\mathbf{B}$ respectively and the diagonal entries in $\mathbf{D}$ in the Hessian matrix $\mathbf{H}$, corresponding to the active constraints (*i.e.* the edges for which $d_{ij} = 1$). Thus, for the edges with active constraints, the $\mathbf{W}_{ij}$ is given by:

$$\mathbf{W}_{ij} = -l_{ij}\hat{\mathbf{r}}_{ij}\hat{\mathbf{r}}_{ij}^\top + m_{ij}\mathbf{I} \tag{58}$$

Thus, the $\mathbf{W}_{ij}$ for the edges with inactive constraints are given by Eq. (57), and for the active constraints are given by Eq. (58).

Coming to the second part of the theorem, we would like to prove the conditions when $\mathbf{W}_{ij} \succcurlyeq \mathbf{0}$ for the Geman-McClure loss. Since $\mathbf{W}_{ij}$ has only the outer product terms of $\mathbf{t}_{ij}$ and $\hat{\mathbf{r}}_{ij}$, one of the eigenvectors is the crossproduct of $\hat{\mathbf{r}}_{ij}$ and $\mathbf{t}_{ij}$. For ease of notation, we drop the ij subscript. Therefore, let $\mathbf{u} = \hat{\mathbf{r}} \times \mathbf{t}$.

$$\mathbf{W}_{ij}\mathbf{u} = m\mathbf{u} \tag{59}$$

Therefore, one eigen value of $\mathbf{W}_{ij}$ is m which is nothing but the weight function of the robust loss function $\rho_\sigma(\cdot)$ and hence is always positive. The other two eigenvectors are perpendicular to $\mathbf{u} = \hat{\mathbf{r}} \times \mathbf{t}$, which can be expressed as a linear combination of $\hat{\mathbf{r}}$ and $\mathbf{t}$.

$$\begin{aligned}\mathbf{W}_{ij}\mathbf{t} = \hat{\mathbf{r}}\left[-l(\hat{\mathbf{r}}^\top\mathbf{t}) - \frac{l^2(\hat{\mathbf{r}}^\top\mathbf{t})^3 - lm(\hat{\mathbf{r}}^\top\mathbf{t})}{m - l(\hat{\mathbf{r}}^\top\mathbf{t})^2}\right] \\ + \mathbf{t}\left[m - \frac{m^2 - lm(\hat{\mathbf{r}}^\top\mathbf{t})^2}{m - l(\hat{\mathbf{r}}^\top\mathbf{t})^2}\right] = \mathbf{0}\end{aligned} \tag{60}$$

Therefore, $\mathbf{t}$ is another eigenvector of $\mathbf{W}_{ij}$ with 0 eigenvalue. Likewise, we can derive

$$\mathbf{W}_{ij}\hat{\mathbf{r}} = \frac{(m-l)m}{m - l(\hat{\mathbf{r}}^\top\mathbf{t})^2}\left(\hat{\mathbf{r}} - (\hat{\mathbf{r}}^\top\mathbf{t})\mathbf{t}\right) \tag{61}$$

$$\begin{aligned}\Rightarrow \mathbf{W}_{ij}\left(\hat{\mathbf{r}} - (\hat{\mathbf{r}}^\top\mathbf{t})\mathbf{t}\right) = \frac{(m-l)m}{m - l(\hat{\mathbf{r}}^\top\mathbf{t})^2}\left(\hat{\mathbf{r}} - (\hat{\mathbf{r}}^\top\mathbf{t})\mathbf{t}\right) \\ \text{using the fact that } \mathbf{W}_{ij}\mathbf{t} = 0\end{aligned} \tag{62}$$

Thus the third eigenvector is $\hat{\mathbf{r}} - (\hat{\mathbf{r}}^\top\mathbf{t})\mathbf{t}$ with $\frac{(m-l)m}{m-l(\hat{\mathbf{r}}^\top\mathbf{t})^2}$ as the eigenvalue. Also, for the positive semidefiniteness of the Hessian matrix, from condition-1 in Proposition 3, we require that the diagonal elements $\mathbf{D}_{ij}$ are positive, *i.e.* $\mathbf{D}_{ij} = m - l(\hat{\mathbf{r}}^\top\mathbf{t})^2 > 0$.

Therefore, for the positive semidefiniteness of $\mathbf{W}_{ij}$,

$$\mathbf{W}_{ij} \succcurlyeq \mathbf{0} \iff \frac{(m-l)m}{m-l(\hat{\mathbf{r}}^\top \mathbf{t})^2} \geq 0 \tag{63}$$

$$\iff (m-l) \geq 0 \quad (\because m - l(\hat{\mathbf{r}}^\top \mathbf{t})^2 > 0, m > 0) \tag{64}$$

$$\iff 1 - \frac{l}{m} \geq 0 \tag{65}$$

For the specific case of Geman-McClure loss, we have,

$$\mathbf{W}_{ij} \succcurlyeq 0 \iff 1 - 3\frac{\|\mathbf{r}_{ij}\|^2}{\sigma^2} \geq 0$$
$$\iff \sigma \geq \sqrt{3}\|\mathbf{r}_{ij}\|$$

It is also easy to see that the condition $\mathbf{D}_{ij} = m - l(\hat{\mathbf{r}}^\top \mathbf{t})^2 = \frac{1}{q_{ij}} > 0$ is not fulfilled only when $d_{ij} \not> 1$, *i.e.*, for an active constraint. This is equivalent to proving that $d_{ij} > 1 \Rightarrow D_{ij} > 0 \Rightarrow q_{ij} > 0$. For an inactive constraint, the optimal d_{ij} is $d_{ij} = (\mathbf{T}_i - \mathbf{T}_j)^\top \mathbf{t}_{ij} \Rightarrow \mathbf{r}_{ij}^\top \mathbf{t}_{ij} = 0 \Rightarrow D_{ij} = m > 0$. If the constraint becomes active, then the expression for $\mathbf{W}_{ij}$ changes, in which case also, the condition for the positive definiteness of $\mathbf{W}_{ij}$ boils down to the condition $\sigma \geq \sqrt{3}\|\mathbf{r}_{ij}\|$.

3 Comparison routine in Vector Averaging

We compare a solution ($\mathbf{x}_{sol}$) with the ground truth ($\mathbf{x}_{gt}$), by finding the rigid transformation (rotation $\mathbf{R}$, translation $\mathbf{t}$) that best fits $\mathbf{x}_{sol}$ to $\mathbf{x}_{gt}$, *i.e.*,

$$\mathbf{R}^*, \mathbf{t}^* = \text{argmin}_{\mathbf{R},\mathbf{t}} \|\mathbf{x}_{gt} - \mathbf{R}\mathbf{x}_{sol} - \mathbf{t}\|^2 \tag{66}$$

The mean and median errors of the resulting residuals are used for evaluation.

4 Additional Results

4.1 Vector Averaging

Regarding the results for Augmented ICL-NUIM [1] datasets in Table-5 in the main paper, we firstly discuss how we obtain the input data suitable for Vector Averaging. Each of the 4 sequences provided by [1] consist of around 2350-2870 scans. Also, the scans are labelled temporally, thus leading to reliable FPFH feature matching. We compute pairwise matches for scans (i, j) such that $|i - j| < 10$ but we remove some pairs randomly in order to introduce some brittleness. This leads to an average of 5 pairs for every scan. The pairwise matches are fed into the 3D registration pipeline as given in [5] to obtain the pairwise relative transformations. We replace the obtained pairwise rotations by the groundtruth rotations so as to produce data only in terms of the translation vectors, thus making it amenable for Vector averaging. The results show

Table 1: Vector averaging on Augmented ICL-NUIM dataset (*lr*:livingroom, *of*:office): *Time taken in seconds* | *Average number of annealing stages* for the baselines

	IRLS	GNC ($\delta\downarrow$)	GNC ($\delta\uparrow$)	GNCp
lr1	0.05 | 1	0.65 | 85	0.15 | 5	0.22 | 4
lr2	0.27 | 1	3.64 | 85	0.53 | 5	3.11 | 8
of1	0.06 | 1	0.52 | 85	0.1 | 5	0.18 | 4
of2	0.18 | 1	1.68 | 85	0.34 | 5	0.95 | 7

that our method achieves superior results in terms of the accuracy in Table-5 in the main paper. We tabulate the number of GNC stages and the time taken in Tab. 1. We observe that our method GNCp takes significantly lesser number of GNC (annealing) stages and is more efficient compared to GNC $(\delta\downarrow)$.

We also tabulate in Tab. 2 few additional parameters for the synthetic datasets with varying outlier fractions $o_f \in [0.1, 0.5]$, at a fixed edge probability $\beta = 0.02$. The experimental settings are the same as in Tables-1,3,4 in the main paper. We observe that our method GNCp takes lesser number of annealing stages compared to that of GNC $(\delta\downarrow)$ (61). Our method even though doesn't entirely eliminate the computation of $\lambda_{min}(\mathbf{H})$, we see that the number of such computations is significantly reduced. Our method conducts only 20's of evaluations (Table- 2) across all annealing steps efficiently, compared to 100's of evaluations of $\lambda_{min}(\mathbf{H})$ in each annealing step in [5] using brute-force search. We also tabulate the number of times GNC $(\delta\downarrow)$ incurs a negative eigenvalue of the Hessian $\lambda_{min}(\mathbf{H}) < 0$. We see that for smaller outlier fractions, it is zero but it increases with increasing outlier fraction. This is reflected in the drop in accuracy for GNC $(\delta\downarrow)$ as shown in Table-1 in the main paper. We also test for larger size datasets ($N = 10000$) to see if our method is efficient and how it compares to GNC $(\delta\downarrow)$. We observe from Tab. 2 that our method takes lesser time compared to GNC $(\delta\downarrow)$ and the efficiency of our method improves with increasing outlier fraction.

Table 2: Vector Averaging, $\beta = 0.02$ with varying outlier fractions: All numbers are computed as the mean over 100 instances, except for $N = 10000$ which is computed over 10 instances.

Outlier Fraction	0.1	0.2	0.3	0.4	0.5
Number of annealing steps in GNCp ($N = 1000$)	4	4	4.3	7.4	9.7
Total number of $\lambda_{min}(\mathbf{H})$ computations in GNCp ($N = 1000$)	22	23	23	24	26
Number of times where $\lambda_{min}(\mathbf{H}) < 0$ for GNC $(\delta\downarrow)$ ($N = 1000$)	0	0	0.1	1.8	18.7
Ratio of time taken $\frac{\text{GNC }(\delta\downarrow)}{GNCp}$ for $N = 10000$	1.09	1.29	1.45	1.72	2.71

While Vector Averaging is common in sensor network localization, time synchronization and motion averaging, publicly available codes for relevant baselines are scarce, complicating comparisons. However, we used one baseline, SE-Sync [4] to demonstrate the efficacy of our Vector averaging method. Since SE-Sync operates on the Special Euclidean Group $SE(3)$, and our method handles only relative displacements, we set all rotations to identity and nullified the rotation cost to ensure a fair comparison. The original SE-Sync is non-robust, but, we applied a robust loss function (Geman-McClure) and used IRLS to obtain weighted least squares problems, which are then fed to SE-Sync. As shown in Tab. 3, Tab. 4, our method outperforms SE-Sync in both accuracy and efficiency on synthetic and real datasets used for Vector Averaging in the main paper.

Table 3: Vector averaging on synthetic datasets of varying outlier fraction (o_f) with real baselines: *Mean* | *Median errors* | *time (in sec.)*

	0.1	0.2	0.3	0.4	0.5
SE-Sync	0.73|0.67|243	0.85|0.79|228	1.04|0.99|228	1.25|1.14|223	1.52|1.39|242
Ours	**0.41**|**0.38**|**1.2**	**0.44**|**0.41**|**1.2**	**0.48**|**0.44**|**1.3**	**0.53**|**0.49**|**1.3**	**0.57**|**0.53**|**2**

Table 4: Vector averaging on Augmented ICL-NUIM dataset with real baselines: *Mean|median errors|time(in sec.)*

	livingroom1	livingroom2	office1	office2
SE-Sync	2.15|2.11|7.7	1.55|1.53|27.8	1.84|1.88|9.5	2.16|2.24|14.4
Ours	**1.93**|**1.59**|**0.23**	**1.25**|**1.31**|**3.62**	**1.8**|**1.79**|**0.22**	**2.12**|**1.93**|**0.95**

4.2 Translation Averaging

We compare the SOTA baselines like LUD [3], BATA [8] with our method on the synthetic datasets for Translation averaging that are used in the main paper. Table-5 shows that, our method exhibits superior accuracy on synthetic datasets compared to SOTA baselines, albeit with slightly higher computational time, which aligns with observations on real-world 1DSfM datasets.

Table 5: TA: *Mean* | *median errors* ($\times 10^{-3}$) | *time (in sec.)* of SOTA baselines on synthetic datasets for varying outlier fraction (o_f).

	0.1	0.2	0.3	0.4	0.5
LUD [3]	48|42|**2**	82|73|**2**	115|107|**1.9**	151|141|**1.9**	211|198|**2.1**
BATA [8]	39|35|4.1	64|55|4.2	94|82|4.2	142|133|4.1	217|204|3.4
Ours	**22**|**19**|4.6	**40**|**31**|5.5	**73**|**62**|6.6	**108**|**93**|8.3	**157**|**140**|8.6

The timing results for our Translation averaging method on synthetic and real datasets are shown in Tab. 6 and Tab. 7 respectively. From Tab. 6, we observe that our method takes significantly lesser number of stages compared to the small factor annealing (GNC ($\delta \downarrow$)). On real datasets, as seen in Tab. 7, our method takes more time compared to the popular baselines. The local routine, written in MATLAB, is an interior point method (IPM) within IRLS framework. The outer IRLS loop has a maximum of n_{irls} iterations, and the inner IPM loop has n_{ipm} iterations. $n_{irls} = 10$, $n_{ipm} = 5$ and a stopping criterion for convergence ($\frac{\|\mathbf{x}_{k+1} - \mathbf{x}_k\|}{\|\mathbf{x}_k\|} > \epsilon$, where $\epsilon = 10^{-5}$) were used. A C++ implementation, like that of LUD [3], could further reduce the time. The implementation for LUD is taken from Theia-SfM [6], which is implemented based on Alternating Direction Method of Multipliers.

Table 6: Translation averaging on synthetic graphs: *Time taken (in seconds)* | *average number of annealing stages* for the baselines for varying outlier fractions o_f for $N = 500$, $\beta = 0.02$

o_f	IRLS	GNC ($\delta \downarrow$)	GNC ($\delta \uparrow$)	GNCp (Ours)
0.1	2.5 | 1	34 | 33	6.8 | 5	15 | 7.9
0.2	3.4 | 1	42 | 33	7.1 | 5	30 | 9.8
0.3	3.2 | 1	47 | 33	8.6 | 5	41 | 12.5
0.4	3 | 1	43 | 33	7.5 | 5	48 | 14.7
0.45	2.4 | 1	35 | 33	6.1 | 5	44 | 16.4
0.48	2.1 | 1	34 | 33	5.6 | 5	45 | 17.9
0.5	2.2 | 1	33 | 33	5.9 | 5	42 | 18.2

Table 7: Time taken for Translation averaging on real datasets, t_p is the time taken to perform principled annealing, t_{tot} is the total time taken.

Dataset	LUD [3]	BATA [8]	GNCp-LUD		GNC ($\delta \downarrow$)	GNC ($\delta \uparrow$)
			t_p	t_{tot}		
Alamo (ALM)	42	54	16	72	504	87
Ellis Island (ELS)	6	10	13	65	66	11
Gendarmenkmart (GMM)	14	25	19	46	188	31
Madrid Metropolis (MDR)	7	14	7	14	73	12
Montreal Notre Dame (MND)	9	29	19	33	235	39
Notre Dame (ND)	46	56	15	70	79	14
NYC Library (NYC)	6	14	15	20	533	87
Piazza del Popolo (PDP)	8	16	6	17	117	20
Piccadilly (PIC)	211	249	87	383	1986	365
Roman Forum (ROF)	24	52	41	83	378	71
Tower of London (TOL)	9	19	22	52	121	21
Trafalgar (TFG)	543	862	302	1000	7162	1336
Union Square (USQ)	8	17	16	43	141	23
Vienna Cathedral (VNC)	54	74	36	92	508	94
Yorkminster (YKM)	11	16	9	32	125	22

5 Codes

Algorithm 1 is the pseudocode for any general averaging problem using our method GNCp. This is similar to Algorithm-1 in the main paper, but in a much simpler form.

In the following subsections, we provide snippets of the code corresponding to only the "Principled Annealing" step in Algorithm 1, because it is the main contribution in this paper. These are not standalone, but capture all the main contributions of our method.

Algorithm 1: Robust Averaging with Adaptive GNC (GNCp)

```
Input: Pairwise relative observations $\mathbf{z}_{ij}\ \forall\ (i,j) \in \mathcal{E}$, $\bar{\sigma}$, $\rho_\sigma(\cdot)$
Output: Estimates of the absolute states $\mathbf{x}_i\ \forall\ i \in \mathcal{V}$
1 Initialization:$p = 100$, Least squares soln. $\{\mathbf{x}_{i,lsq}\}$.
2 while $p \geq 50$ and $\sigma \geq \bar{\sigma}$ do
3     Principled annealing: Obtain $\sigma$ for the GNC stage using the Hessian/Laplacian matrix.
4     Local Optimization: Using IRLS, minimize the robust cost with $\sigma$ obtained from the principled annealing step.
5 end
```

5.1 Principled Annealing Routine

The following is the code for performing the principled annealing irrespective of the type of averaging problem.

```
%% Principled Annealing

% Initialization of Variables required
count = 0;
Iteration=0;
frac_outl = 0;
n_emer = 0;
eigvecinit = randn(q*N,1);
neg_eig_thres = -1e-4;
n_compute_eigval = 0;
stepmagn = 1; minPerStep = 0.1;
isreduced = false;

%%%%%%%%%%%%%%%%%  START
t1_temp = toc(t0);
sig_prev_stage = sig; % Sigma in the previous GNC stage.

% per_step is the decrement in the percentile p (p has the
  same definition as in the main paper).
if(Iteration==0)
  per_step = 0; % in the first GNC stage, we start with p =
  100 percentile
else
  per_step = 0.5; % from the second stage onwards, we keep
  reducing p.
end

% frac_outl = 100-p. (Increasing frac_outl is equivalent to
  reducing p which is equivalent to reducing sigma)
frac_outl = frac_outl+per_step;

if(frac_outl<50 && sig>sig_min)
```

```
  th = rij; % Residual vector
  temp_iter = 0; neg_eig_encount = false; increase_fracoutl = 
false;decr_after_incr = false; minLevelReached = false;
  % The following while loop decreases the percentile p 
aggressively as long as the minimum eigenvalue or the 
Fiedler value is positive.
  while(frac_outl<50 && sig>sig_min)
    temp_iter = temp_iter+1;
    ptile = sqrt(3)*prctile(th,100-frac_outl); % Computing 
sigma_ij and the sigma.
    sig = max(min(ptile,sig_prev_stage), sig_min);
    if(Iteration==0)
      break; %in the first iteration, the per_step=0, i.e., 
percentile p=100, hence the fiedler value will always be 
positive.
    end

    % Compute the minimum eigen value (or fiedler value) of 
the Hessian matrix in corresponding averaging problem. Here,
 we take the example of VectorAveraging.
    [fied_val, eigvecinit] = checkVecAvgFiedlerValuen(res, I, 
'GM', sig, row_ind, col_ind, eigvecinit);
    n_compute_eigval = n_compute_eigval+1; % number of 
computations of the eigenvalue.

    % if the fied_val is positive, increase frac_outl 
aggressively i.e., in steps of 5*stepmagn, thereby reducing 
the percentile p, thereby reducing sigma.
    if(fied_val>min(neg_eig_thres/sig^2,-1e-6) || isnan(
fied_val))
      if(neg_eig_encount)
        % if we had encountered a negative eigenvalue already 
in the process of increasing frac_outl aggressively, then 
there is no need to increase frac_outl.
        break;
      else
        % if no negative eigenvalue is encountered, increase 
frac_outl.
        frac_outl = frac_outl+5*stepmagn;
        % flags to signify that frac_outl is increased 
aggressively.
        increase_fracoutl = true;
        decr_after_incr = true;
        increase_enter_iterno = 1;
      end
    % if the fied_val is negative, depending on whether 
agressive annealing was done, reduce frac_outl accordingly.
    elseif(fied_val<=min(neg_eig_thres/sig^2,-1e-6) && ~
isreduced)
```

```
      neg_eig_encount = true; % flag to signify that a negative eigenvalue is encountered.
      if(~increase_fracoutl)
        % even before increasing frac_outl aggressively, if a negative eigenvalue arises, then reduce the frac_outl in steps to increase sigma.
        if(frac_outl<per_step)
          % in order to not reduce frac_outl further.
          isreduced = true;
        else
          % reduce frac_outl slowly as long as the frac_outl is greater than that of the previous GNC stage.
          frac_outl = frac_outl-per_step/(2^(temp_iter-1))*(temp_iter==1)+max(per_step/(2^temp_iter),minPerStep);
          if(per_step/(2^temp_iter)<=minPerStep)
            isreduced = true; %
            minLevelReached = true;
          end
        end
      else
        % if frac_outl is increased aggressively and then a negative eigenvalue comes, then reduce the frac_outl slowly, as long as it is greater than the frac_outl of the previous stage.
        frac_outl = frac_outl-5*stepmagn/(2^(increase_enter_iterno-1))*(decr_after_incr)+max(5*stepmagn/(2^increase_enter_iterno),minPerStep);
        decr_after_incr = false;
        if(5*stepmagn/(2^increase_enter_iterno)<=minPerStep)
          ptile = sqrt(3)*prctile(th,100-frac_outl);
          sig = max(min(ptile,sig_prev_stage), sig_min);
          break;
        end
        increase_enter_iterno = increase_enter_iterno+1;
      end
    else
      % if the eigenvalue is negative even if frac_outl is not increased, it means we could be in a saddle point. Resort to fixed annealing at this stage.
      sig = 0.9*sig_prev_stage;
      break;
    end
  end
else
  break;
end
t2_temp = toc(t0);
time_anneal = time_anneal + (t2_temp-t1_temp); % The time taken for principled annealing.
%%%%%%%%%%%%%%%%%  END
```

Hessian Computation for Vector Averaging Cost: The following is the code snippet for computing the Hessian matrix and its minimum eigenvalue for the Vector Averaging cost.

```
%%%% Code for checkVecAvgFiedlerValuen.m
function [fied_val, eigvec1] = checkVecAvgFiedlerValuen(res, I
  , lossfn, sigma, row_ind, col_ind, eigvecinit)
  % This function evaluates the Hessian of robust vector
  averaging cost and returns the minimum eigenvalue of it. It
  is also referred to as the Fiedler value because the Hessian
   has Laplacian structure.

  % res is the set of residuals (Mxq matrix).
  % I (2xM) is the sorted edge list of the input graph.
  % lossfn refers to the robust loss, currently we are using
  Geman-McClure loss only in this script.
  % sigma: Parameter of the robust loss.
  % row_ind, col_ind are indices used to populate the sparse
  Hessian matrix.
  % eigvecinit is an initialization of the eigenvector which
  is generally obtained from previous eigenvalue computations.

  % fied_val: Obtained minimum eigen value of the Hessian
  matrix.
  % eigvec1 : Corresponding eigenvector.

  % Populate the Hessian entries.
  N = max(I(:)); % number of nodes in the graph.
  M = size(I,2); %no of edges
  q = size(res,2); % dimensionality of the problem
  rij = vecnorm(res,2,2); % Norm of the Residuals
  res_normalized = res./rij; % Normalized residuals
  res_normalized(rij<1e-10,:)=0; % To avoid numerical issues.

  % lij and mij as defined in the paper.
  lij = 4*rij.^2./(sigma^2*(1+rij.^2/sigma^2).^3);
  mij = 1./(1+rij.^2/sigma^2).^2;

  w_tensor = permute(res_normalized,[2,3,1]);
  wwt = permute(lij,[2,3,1]).*bsxfun(@times, w_tensor, permute
  (w_tensor,[2,1,3]));
  id_tensor = permute(mij,[2,3,1]).*repmat(eye(q),[1,1,M]);
  % Weight tensor (Wij's as defined in the paper).
  Wij = id_tensor-wwt;
  Hij_tot = cat(3,-Wij,-Wij,Wij,Wij);

  %Populate the entries of the Hessian/Laplacian
  H = sparse(row_ind,col_ind,Hij_tot(:),q*N,q*N);
  % normalizing Hessian, note that we are only interested in
  the sign of the Fiedler value.
```

```
    Hg = H/sqrt(sum(H.*H,'all'));

    [i,j,vals] = find(Hg);
    n = size(Hg,1);
    m = n;
    % Obtaining the minimum eigenvalue and the corresponding eigenvector by making a call to C++ script based on Spectra library. Gershgorin circle theorem is used in this script to shift the eigenvalues by a constant.
    [fied_val, eigvec1] = gom_new(uint32(i),uint32(j),vals,uint32(m),uint32(n),eigvecinit);
  end
```

Hessian Computation for Translation Averaging Cost: The following is the code snippet for computing the Hessian matrix and its minimum eigenvalue for the Translation Averaging cost.

```
  %% Code for computing Hessian and the correponding Fiedlervalue for the LUD based Translation averaging cost
function [fied_val, eigvec1] = checkTransAvgFiedlerValue_LUD_SOSC(T, tij, res, I, lossfn, sigma, row_ind, col_ind, ind_active, eigvecinit)
    % This function evaluates the Hessian of robust translation averaging cost based on LUD formulation and returns its minimum eigenvalue. Since this is a constrained problem, the active constraints namely dij=1 are taken into consideration, and the second order sufficient KKT conditions (SOSC) are evaluated which in effect translates to removing rows and columns corresponding to the active constraints in the complete Hessian.

    % T: absolute translations given by a Nxq matrix.
    % tij: Mxq observed relative translations.
    % res: residuals is a Mxq matrix.
  % I (2xM) is the sorted edge list of the input graph.
  % lossfn refers to the robust loss, currently we are using Geman-McClure loss only in this script.
  % sigma: Parameter of the robust loss.
  % row_ind, col_ind are indices used to populate the sparse Hessian matrix.
    % ind_active: boolean array (Mx1) of the active constraints. (1-> active, 0-> inactive).
  % eigvecinit is an initialization of the eigenvector which is generally obtained from previous eigenvalue computations.

    % fied_val: Obtained minimum eigen value of the matrix arising in second order sufficient conditions.
  % eigvec1 : Corresponding eigenvector.

```

```

    % Populate the Hessian entries.
    N = max(I(:)); % number of nodes in the graph.
    M = size(I,2); %no of edges
    q = size(res,2); % dimensionality of the problem

    % Form the matrices L1, L2, B1B1', B2B2', B1B2', B2B1', D1,
    D2 (all are laplacian matrices), same notation as in the
    proof of Theorem-2 in the supplementary material
    r = res;
    rij = vecnorm(res,2,2);
    rh = r./rij;
    rh(rij<1e-7,:) = 0;
    rh_tensor = permute(rh,[2,3,1]);
    t_tensor = permute(tij,[2,3,1]);

    rhrhT = bsxfun(@times,  rh_tensor, permute(rh_tensor
    ,[2,1,3]));
    id_tensor = repmat(eye(3),[1,1,M]);
    uTr_vec = sum(tij.*rh,2);
    uTr = permute(sum(tij.*rh,2),[2,3,1]);

    L1ij = rhrhT;
    L2ij = id_tensor;
    B11ij = (uTr.^2).*rhrhT;
    B22ij = bsxfun(@times,  t_tensor, permute(t_tensor,[2,1,3]))
    ;
    B12ij = uTr.*bsxfun(@times, rh_tensor , permute(t_tensor
    ,[2,1,3]));
    B21ij = permute(B12ij, [2,1,3]);


    lij = 4*rij.^2./(sigma^2*(1+rij.^2/sigma^2).^3); % Geman-
    McClure
%     lij = 2*rij.^2./(sigma^2*(1+rij.^2/sigma^2).^2); % Cauchy
    mij = 1./(1+rij.^2/sigma^2).^2; % Geman-McClure.
%     mij = 1./(1+rij.^2/sigma^2);  % Cauchy
    qij_tensor = permute(1./(-lij.*uTr_vec.^2+mij),[2,3,1]); %
    qij = 1/(-lij*D1+mij*D2)
    lij_tensor = permute(lij,[2,3,1]); mij_tensor = permute(mij,
     [2,3,1]);

    % Make the terms in Bxxij corresponding to the active
    constraints 0.
    ind_active_tensor = permute(~ind_active,[2,3,1]);
    Wij = -lij_tensor.*L1ij+mij_tensor.*L2ij - ind_active_tensor
    .*(qij_tensor).*(lij_tensor.^2.*B11ij+mij_tensor.^2.*B22ij-
    lij_tensor.*mij_tensor.*(B12ij+B21ij));
    Hij_tot = cat(3,-Wij,-Wij,Wij,Wij);

    %Populate the entries of the Hessian/Laplacian
    H = sparse(row_ind,col_ind,Hij_tot(:),q*N,q*N);

    %% Efficient way to calculate fied_val
    Hg = H/sqrt(sum(H.*H,'all'));
    [i,j,vals] = find(Hg);
    n = size(Hg,1);
    m = n;
  % Obtaining the minimum eigenvalue and the corresponding
    eigenvector by making a call to C++ script based on Spectra
    library. Gershgorin circle theorem is used in this script to
     shift the eigenvalues by a constant.
    [fied_val, eigvec1] = gom_new(uint32(i),uint32(j),vals,
    uint32(m),uint32(n),eigvecinit);
end
```

5.2 Computation of Minimum Eigenvalue of Laplacian

The following is a snippet from the code for computing efficiently the minimum eigenvalue of a Laplacian/Hessian matrix. We use this code to compute the largest magnitude eigenvalue of $\mathbf{H} - c\mathbf{I}$ as stated in Lines 280-297 in the main paper. It is written in C++ by modifying the Spectra [1] C++ library for large scale eigenvalue problems.

```
/* C++ code for mineigenvalue computation */
Index compute(Index maxit = 1000, Scalar tol = 1e-10, int
    sort_rule = LARGEST_MAGN)
{
  // The m_ncv step Lanczos factorization, m_ncv is the
    dimension of the Kryolov subspace used in the Lanczos method
     = 10, m_nmatop is the number of matrix operations called.
  m_fac.factorize_from(1, m_ncv, m_nmatop);
  // Retrieve and sort Ritz values and Ritz vectors.
  retrieve_ritzpair();

  Index i, nconv = 0, nev_adj;
  Array thresh, resid, resid_prev;
  // The length of the variables vector is m_n = 3*N in our case
    .
  Vector eigvec_estim(m_n), eigvec_estim1(m_n),
    eigvec_estim_temp(m_n), vec_ones(m_n), vec_reduced(m_n/3),
    vec_reduced_temp(m_n/3);
  eigvec_estim.setZero();
  vec_ones.setOnes();
  vec_ones.normalize(); // Normalized Vector of all ones.
  std::vector<double> res_norm;
  res_norm.push_back(-1.0);

```

[1] https://spectralib.org/

```
double epsilon = 1e-3; // for numLanczosvectors=10

int nconv_approx=0;
// For loop for performing implicitly restarted Lanczos
  iterations.
for (i = 0; i < maxit; i++)
{
  // nconv is number of converged eigenvalues, m_nev=1 is
  requested number of eigenvalues.
  nconv = num_converged(tol);
  if (nconv >= m_nev)
    break;

  nev_adj = nev_adjusted(nconv);
  restart(nev_adj); // Implicitly restarted Lanczos
  factorization

  /* Even if the vectors don't converge, obtain the largest
  eigen vector and check if it is close to 1's or not*/
  eigvec_estim_temp = eigvec_estim;
  eigvec_estim1 = m_fac.matrix_V()*m_ritz_vec.col(0);
  // Pick the sign according to the proximity to ones vector
  in terms of the norm.
  eigvec_estim.noalias() = (eigvec_estim1-vec_ones).norm() < (
  eigvec_estim1+vec_ones).norm() ?  eigvec_estim1 : -
  eigvec_estim1;
  // Compute the norm of the difference vector r = norm(
  eigvec_estim1-normalizedallones);
  res_norm.push_back((eigvec_estim-vec_ones).norm());

  // The if condition is to check if the converged eigenvector
   is orthogonal to all ones (normalized) vector
  // If it's orthogonal, then the norm of the difference
  vector is sqrt(2); also the converged eigenvalue is not 0.
  if(res_norm.back()<=sqrt(2)+epsilon && res_norm.back()>=sqrt
  (2)-epsilon)
  {
    // then the minimum eigenvalue converging would be
  negative.
    setNegEigenValFlag = true;
    nconv = 1;
    break;
  }
  // Else, it means that the obtained eigenvector could be
  close to having 0 eigenvalue.
  // The dimension of the variables considered for translation
   averaging and vector averaging is 3*N, where N is the
  number of node variables and 3 is the dimension of each node
   variable. Each of the N x N submatrices formed from a
  single dimension of the node variables is Laplacian. Thus,
    there are 3 eigenvectors having 0 eigenvalue and the vector
    of all ones is only a linear combination of those 3 vectors.
    // The eigen vector which has 0 eigen value has equal norm
    for every consecutive set of 3 elements.
    else
    {
      vec_reduced = ((eigvec_estim.array().square()).reshaped
    (3,(int) eigvec_estim.rows()/3)).colwise().sum().cwiseSqrt()
    .transpose().eval();
      vec_reduced_temp.setConstant(vec_reduced(0));
      if(res_norm.back()<=epsilon || vec_reduced.isApprox(
    vec_reduced_temp,0.01*sqrt(3.0/m_n)))
      {
        setNegEigenValFlag = false; //indicating that the min.
    eigen value is non-negative.
        nconv = 1;
        break;
      }
    }
  }
  // Sorting results
  sort_ritzpair(sort_rule);
  m_niter += i + 1;
  m_info = (nconv >= m_nev) ? SUCCESSFUL : NOT_CONVERGING;
  return std::min(m_nev, nconv);
}
/*   END of C++ code for mineigenvalue computation   */
```

References

1. Choi, S., Zhou, Q.Y., Koltun, V.: Robust reconstruction of indoor scenes. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 5556–5565 (2015)
2. Chung, F.R.: Spectral graph theory, vol. 92. American Mathematical Soc. (1997)
3. Ozyesil, O., Singer, A.: Robust camera location estimation by convex programming. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2674–2683 (2015)
4. Rosen, D.M., Carlone, L., Bandeira, A.S., Leonard, J.J.: Se-sync: A certifiably correct algorithm for synchronization over the special euclidean group. The International Journal of Robotics Research **38**(2-3), 95–125 (2019)
5. Sidhartha, C., Manam, L., Govindu, V.M.: Adaptive annealing for robust geometric estimation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 21929–21939 (June 2023)
6. Sweeney, C.: Theia multiview geometry library: Tutorial & reference. `http://theia-sfm.org`
7. Zhang, F.: The Schur complement and its applications, vol. 4. Springer Science & Business Media (2006)
8. Zhuang, B., Cheong, L.F., Lee, G.H.: Baseline desensitizing in translation averaging. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4539–4547 (2018)