16-833: Robot Localization and Mapping, Spring 2021

Homework 3-Linear and Non-linear SLAM Solvers

Aaron Guan (zhongg)

1 2D Linear SLAM

1. Given robot poses $\mathbf{r}^t = [r_x^t, r_y^t]^T$ and $\mathbf{r}^{t+1} = [r_x^{t+1}, r_y^{t+1}]^T$ at time t and t + 1 , the odometry measurement function $h_o(\mathbf{r}^t, \mathbf{r}^{t+1})$ is:

$$h_o(\mathbf{r}^t, \mathbf{r}^{t+1}) = \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_x^{t+1} - r_x^t \\ r_y^{t+1} - r_y^t \end{bmatrix}$$

The Jacobian of the odometry measurement function $H_o(\mathbf{r}^t, \mathbf{r}^{t+1})$ is:

$$H_o(\mathbf{r}^t, \mathbf{r}^{t+1}) = \frac{\partial h_o}{\partial x} = \begin{bmatrix} \frac{\partial \Delta x}{\partial r_x^t} & \frac{\partial \Delta x}{\partial r_y^t} & \frac{\partial \Delta x}{\partial r_x^{t+1}} & \frac{\partial \Delta x}{\partial r_y^{t+1}} \\ \frac{\partial \Delta y}{\partial r_x^t} & \frac{\partial \Delta y}{\partial r_y^t} & \frac{\partial \Delta y}{\partial r_x^{t+1}} & \frac{\partial \Delta y}{\partial r_y^{t+1}} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

Given the robot pose $\mathbf{r}^t = [r_x^t, r_y^t]^T$ at time t and the k-th landmark $\mathbf{l}^k = [l_x^k, l_y^k]^T$, the landmark measurement function $h_l(\mathbf{r}^t, \mathbf{l}^k)$ is:

$$h_l(\mathbf{r}^t, \mathbf{l}^k) = \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} l_x^k - r_x^t \\ l_y^k - r_y^t \end{bmatrix}$$

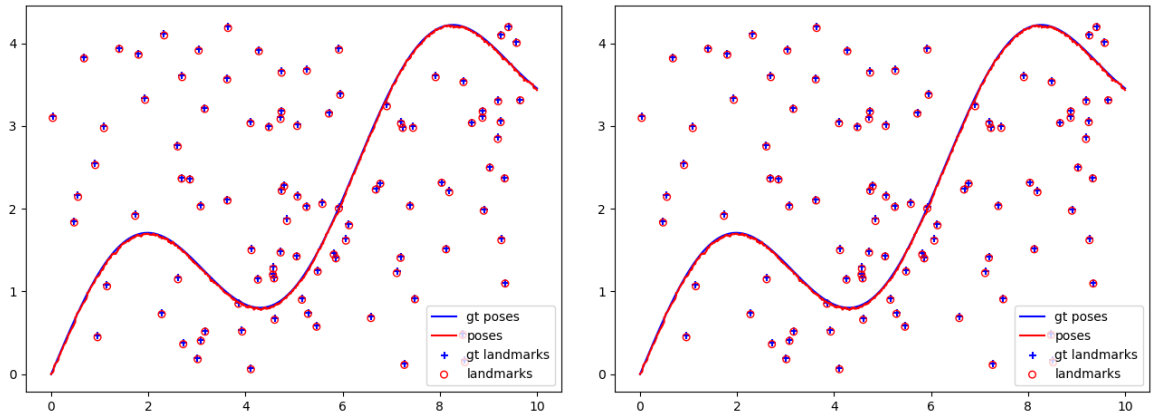
The Jacobian of the landmark measurement function $H_o(\mathbf{r}^t, \mathbf{l}^k)$ is:

$$H_l(\mathbf{r}^t, \mathbf{l}^k) = \frac{\partial h_o}{\partial x} = \begin{bmatrix} \frac{\partial \Delta x}{\partial r_x} & \frac{\partial \Delta x}{\partial r_y} & \frac{\partial \Delta x}{\partial l_x} & \frac{\partial \Delta x}{\partial l_y} \\ \frac{\partial \Delta y}{\partial r_x} & \frac{\partial \Delta y}{\partial r_y} & \frac{\partial \Delta y}{\partial l_x} & \frac{\partial \Delta y}{\partial l_y} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

2. The `create_linear_system` is implemented in `linear.py`

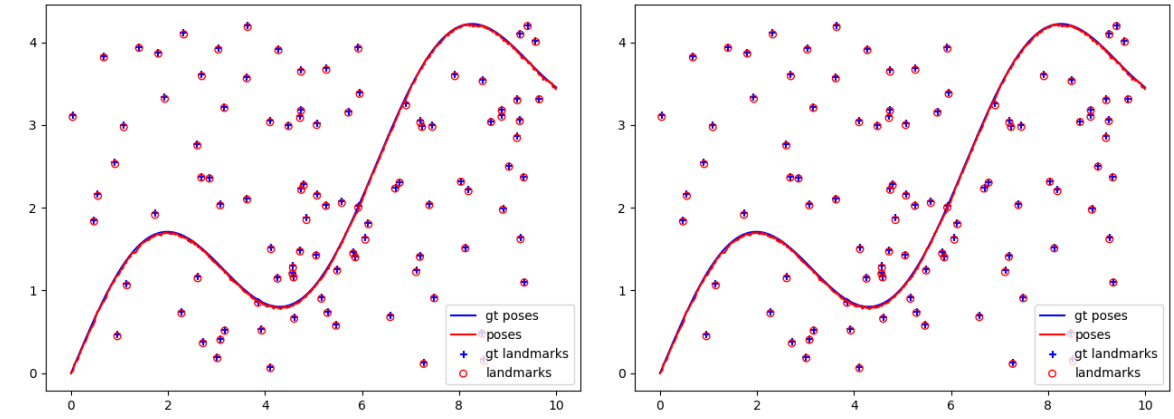
3. Different solvers are implemented in `solvers.py`

4. The trajectory and landmarks results for different solves on the `2D_linear` dataset is shown in Figure 1 and their corresponding sparse decomposed matrix are shown in Figure 2. The performance timing of different solvers on the `2D_linear` dataset is shown in Table 1.



(a) LU Decomposition Result

(b) QR Decomposition Result



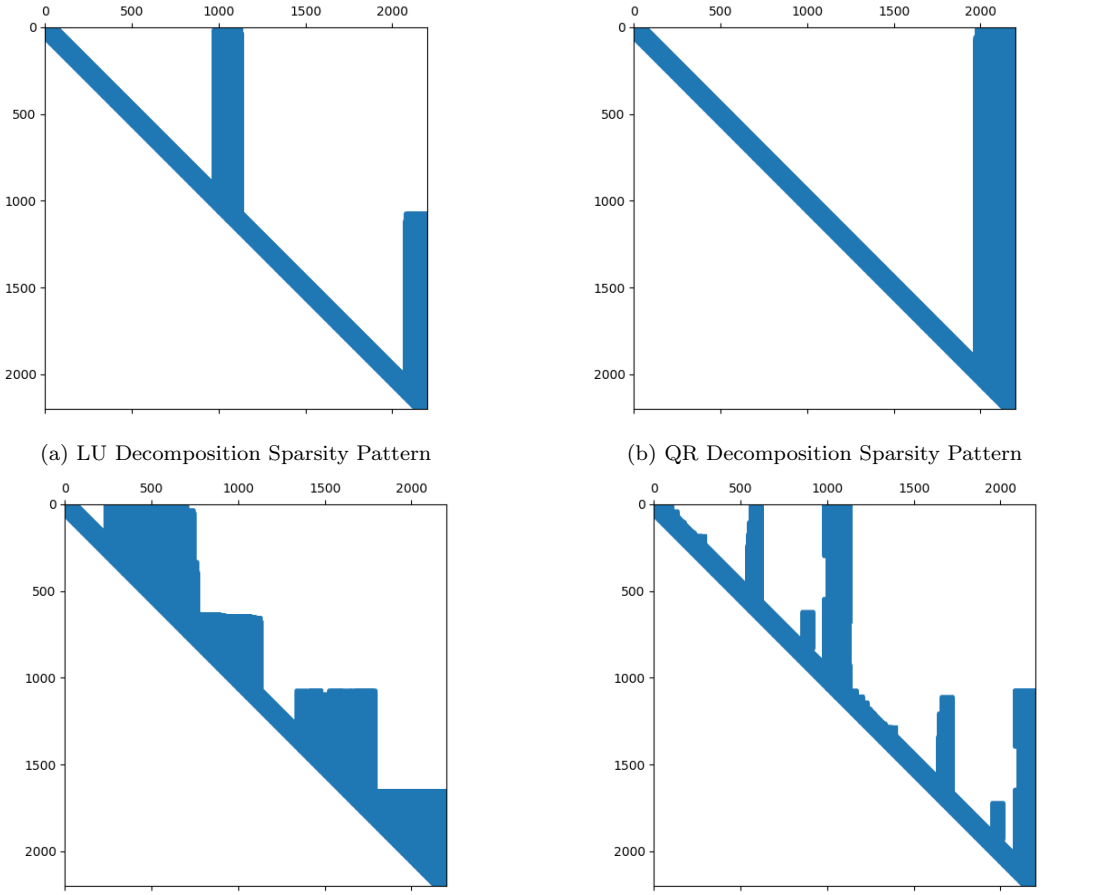
(c) LU_COLAMD Decomposition Result

(d) QR_COLAMD Decomposition Result

Figure 1: Results on `2D_linear` dataset

From the results, we can know that all solves can generate reasonable and correct trajectory and landmark results on the `2D_linear` dataset. The decompositions in order of efficiency (execution time) are LU_COLAMD > LU > QR > QR_COLAMD > Pseudo Inverse. The Psueudo Inverse method is the lowest algorithm because it needs to compute the A^TA and also its inverse. It totally can have $O(n^3)$ time complexity. All other four methods have pretty much similar time performance but it turns out that LU decomposition is generally faster than QR decomposition.

From the sparsity patterns, we can appreciate a fact that the result generated by QR decomposition is sparser than that of LU decomposition. Additionally, the LU_COLAMD and QR_COLAMD can have a better sparse matrix due to the permutation applied on the A matrix, which reduces the number of non-zeros in the Cholesky factor.



(a) LU Decomposition Sparsity Pattern

(b) QR Decomposition Sparsity Pattern

(c) LU_COLAMD Decomposition Sparsity Pattern

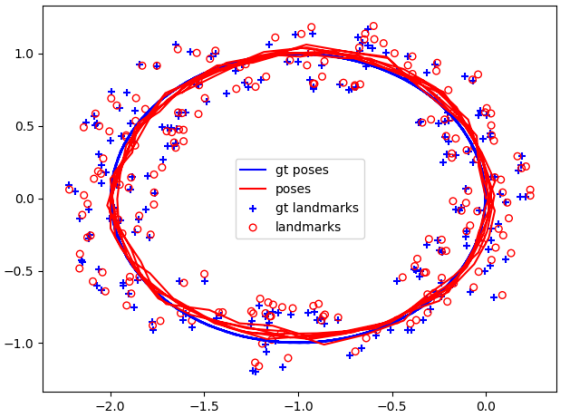
(d) QR_COLAMD Decomposition Sparsity Pattern

Figure 2: Sparsity Patterns on `2D_linear` dataset

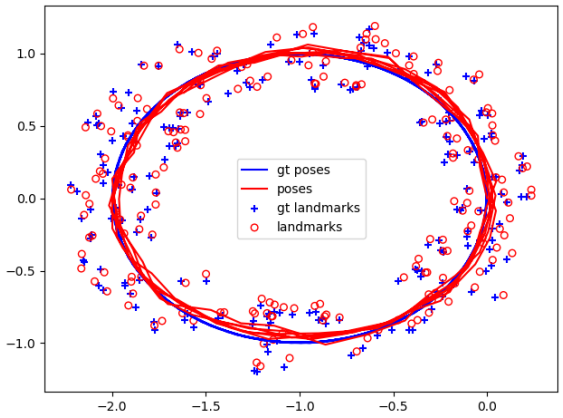
Method	**Timing (s)**
Pseudo Inverse	11.1543
LU	0.2535
QR	1.0365
LU_COLAMD	0.1508
QR_COLAMD	1.3091

Table 1: Performance timings for `2D_linear` dataset

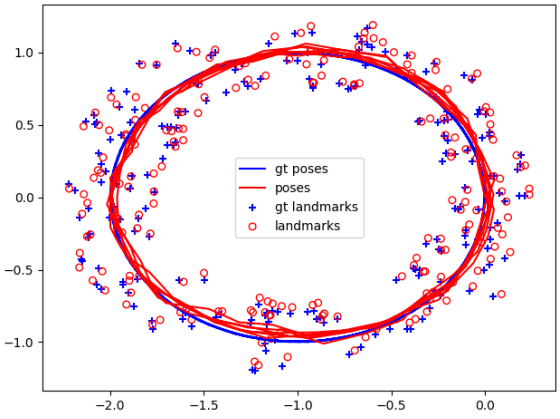
The trajectory and landmarks results for different solves on the `2D_linear_loop` dataset is shown in Figure 3 and their corresponding sparse decomposed matrix are shown in Figure 4. The performance timing of different solvers on the `2D_linear_loop` dataset is shown in Table 2.



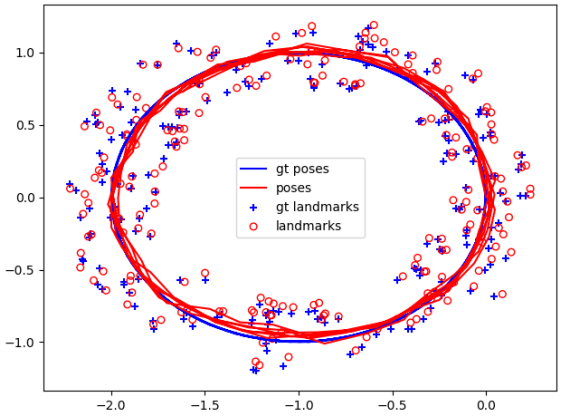
(a) LU Decomposition Result



(b) QR Decomposition Result



(c) LU_COLAMD Decomposition Result

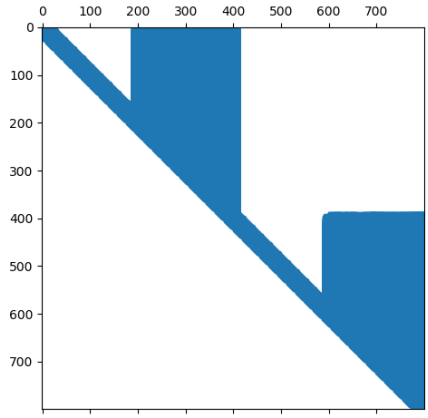


(d) QR_COLAMD Decomposition Result

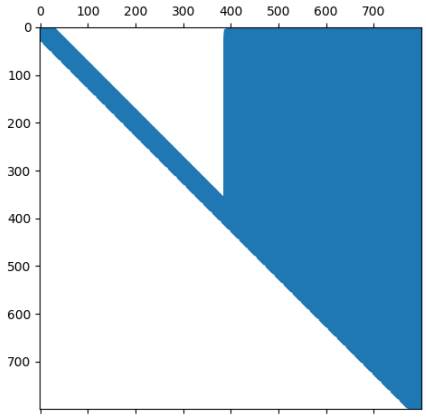
Figure 3: Results on `2D_linear_loop` dataset

From the trajectory and landmark results above, we can know that all solves can generate reasonable results on `2D_linear_loop` dataset. Comparing with the `2D_linear` dataset, the landmark detection has obvious misalignment with ground-truth and the trajectory has small drift as well. Now the decomposition performance in terms of efficiency is LU_COLAMD > LU ¿ QR_COLAMD ¿ Pseudo Inverse ¿ QR. The result is totally different compared to the results on `2D_linear` dataset. The sparsity pattern of different solvers also indicate their efficiency.

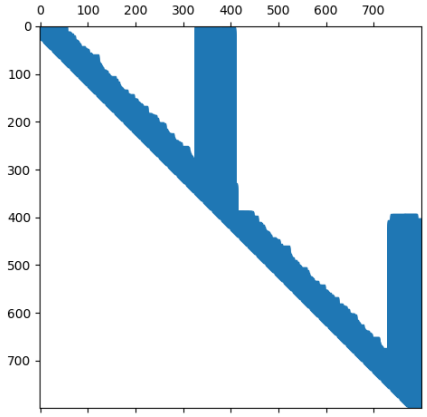
We can see that both LU_COLAMD and QR_COLAMD has much sparser factor while QR has a very dense factor. For QR, the R matrix are almost non-sparse and thus the amount of time taken to compute the results exceeds the time taken to compute the pseudo-inverse.



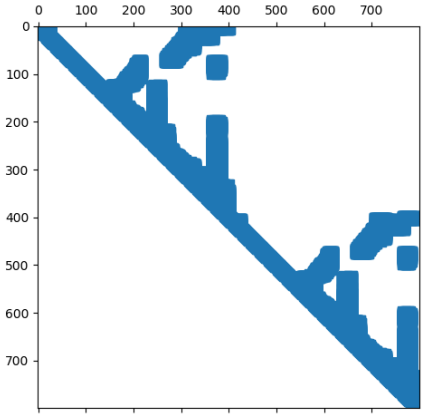
(a) LU Decomposition Sparsity Pattern



(b) QR Decomposition Sparsity Pattern



(c) LU_COLAMD Decomposition Sparsity Pattern



(d) QR_COLAMD Decomposition Sparsity Pattern

Figure 4: Sparsity Patterns on `2D_linear_loop` dataset

Method	**Timing (s)**
Pseudo Inverse	0.4155
LU	0.0448
QR	0.5833
LU_COLAMD	0.0096
QR_COLAMD	0.3632

Table 2: Performance timings for `2D_linear_loop` dataset

2 2D Nonlinear SLAM

1. Given that the landmark measurement function is:

$$h_l\left(\mathbf{r}^t, \mathbf{l}^k\right) = \begin{bmatrix} \text{atan2}\left(l_y^k - r_y^t, l_x^k - r_x^t\right) \\ \left(\left(l_x^k - r_x^t\right)^2 + \left(l_y^k - r_y^t\right)^2\right)^{\frac{1}{2}} \end{bmatrix} = \begin{bmatrix} \theta \\ d \end{bmatrix}.$$

Let's present the difference between the landmark and robot as $dx = l_x - r_x$ and $dy = l_y - r_y$. Therefore, the Jacobian of the landmark measurement function with respect to the state vector $\mathbf{x} = [r_x, r_y, l_x, l_y]^T$ is:

$$H_l(\mathbf{r}^t, \mathbf{l}^k) = \frac{\partial h_o}{\partial x} = \begin{bmatrix} \frac{\partial \theta}{\partial r_x} & \frac{\partial \theta}{\partial r_y} & \frac{\partial \theta}{\partial l_x} & \frac{\partial \theta}{\partial l_y} \\ \frac{\partial d}{\partial r_x} & \frac{\partial d}{\partial r_y} & \frac{\partial d}{\partial l_x} & \frac{\partial d}{\partial l_y} \end{bmatrix} = \begin{bmatrix} \frac{dy}{dx^2+dy^2} & -\frac{dx}{dx^2+dy^2} & -\frac{dy}{dx^2+dy^2} & \frac{dx}{dx^2+dy^2} \\ -\frac{dx}{\sqrt{dx^2+dy^2}} & -\frac{dy}{\sqrt{dx^2+dy^2}} & \frac{dx}{\sqrt{dx^2+dy^2}} & \frac{dy}{\sqrt{dx^2+dy^2}} \end{bmatrix}$$

2. The `create_linear_system` has been implemented in the `nonlinear.py`

3. The trajectory and landmarks before optimization is shown in Figure 5.

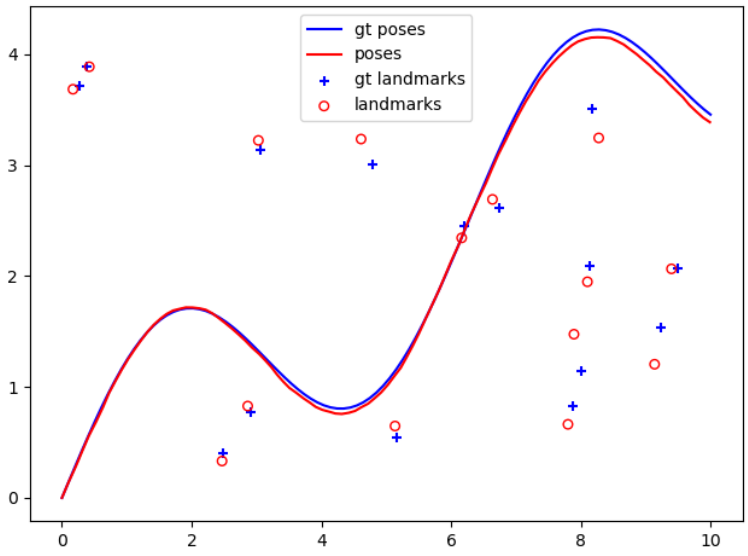
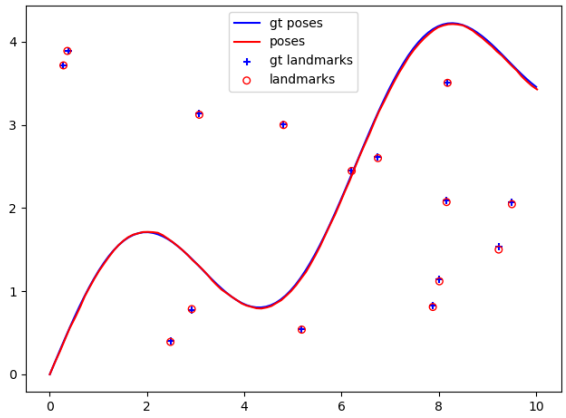
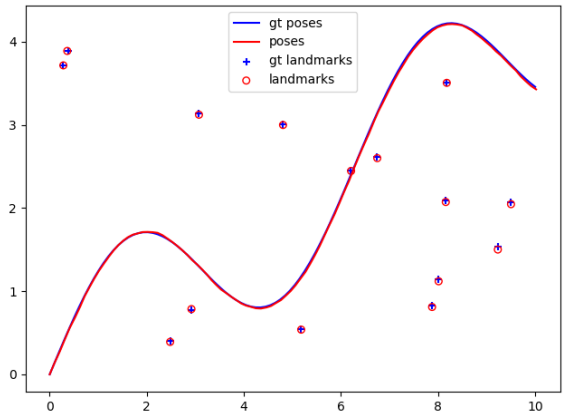


Figure 5: Initial estimation before optimization

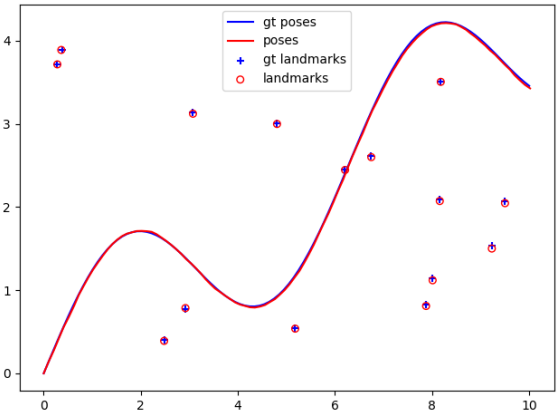
The optimization results of different solvers on `2D_nonlinear` dataset are shown in Figure 6. We can see that all solvers can generate good results for both trajectory and landmarks in the non-linear SLAM setting. All of solvers can reach the local minimum within 10 iterations. Note that this is still a fairly easy task and it can easily reach the local minimum because of the initialization is pretty much the same as the groundtruth. However, in other real SLAM tasks, it might be tricky to have a good initialization to start the non-linear optimization and global minimum is not always guaranteed.



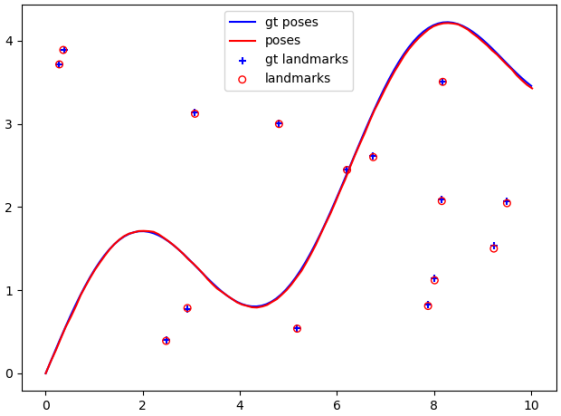
(a) LU Decomposition Result



(b) QR Decomposition Result



(c) LU_COLAMD Decomposition Result



(d) QR_COLAMD Decomposition Result

Figure 6: Results on `2D_nonlinear` dataset