Name: Ashwin Biju Nair | **Andrew ID:** abijunai

16-833: Robot Localization and Mapping

Homework 3 - Linear and Nonlinear SLAM Solvers

1. 2D Linear SLAM

1.1. Measurement function

Given are the robot poses $\mathbf{r}^t = \left[r_x^t, r_y^t\right]^\top$ and $\mathbf{r}^{t+1} = \left[r_x^{t+1}, r_y^{t+1}\right]^\top$ at time t and $t+1$.
The measurement function $h_o\left(\mathbf{r}^t, \mathbf{r}^{t+1}\right)$ maps a 4 dimensional input R^4 that contains the four poses to a 2 dimensional output (R^2) which are the odometry measurements.

Odometry measurements are denoted by the relative displacements. Therefore, the odometry measurement function $h_o(\mathbf{r}^t, \mathbf{r}^{t+1})$ is given by:

$$H_o\left(\mathbf{r}^t, \mathbf{r}^{t+1}\right) = \begin{bmatrix} {r_x}^{t+1} - r_x^t \\ {r_y}^{t+1} - r_y^t \end{bmatrix}$$

The Jacobian $H_o\left(\mathbf{r}^t, \mathbf{r}^{t+1}\right)$ is defined as a 2 x 4 matrix as follows:

$$H_o\left(\mathbf{r}^t, \mathbf{r}^{t+1}\right) = \begin{bmatrix} \frac{\partial f_1}{\partial r_x^t} & \frac{\partial f_1}{\partial r_y^t} & \frac{\partial f_1}{\partial r_x^{t+1}} & \frac{\partial f_1}{\partial r_y^{t+1}} \\ \frac{\partial f_2}{\partial r_x^t} & \frac{\partial f_2}{\partial r_y^t} & \frac{\partial f_2}{\partial r_x^{t+1}} & \frac{\partial f_2}{\partial r_y^{t+1}} \end{bmatrix}$$

where

$$f_1 = {r_x}^{t+1} - r_x^t$$
$$f_2 = {r_y}^{t+1} - r_y^t$$

After solving, the Jacobian of the measurement function is:

$$\boxed{H_o\left(\mathbf{r}^t, \mathbf{r}^{t+1}\right) = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}}$$

Similarly, given the robot pose $\mathbf{r}^t = \left[r_x^t, r_y^t\right]^\top$ at time t and the k-th landmark $\mathbf{l}^k = \left[l_x^k, l_y^k\right]^\top$, the landmark measurement function $h_l\left(\mathbf{r}^t, \mathbf{l}^k\right)$ maps a 4 dimensional input (R^4) that contains the pose and landmark, to a 2 dimensional output (R^2) which are the landmark measurements.

Since the landmark measurements are also given by relative displacements, the landmark measurement function $h_l(\mathbf{r}^t, \mathbf{l}^k)$ is given by:

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

The Jacobian $H_l\left(\mathbf{r}^t, \mathbf{l}^k\right)$ is defined as a 2 x 4 matrix as follows:

$$H_l\left(\mathbf{r}^t, \mathbf{l}^k\right) = \begin{bmatrix} \frac{\partial f_1}{\partial r_x^t} & \frac{\partial f_1}{\partial r_y^t} & \frac{\partial f_1}{\partial l_x^k} & \frac{\partial f_1}{\partial l_y^k} \\ \frac{\partial f_2}{\partial r_x^t} & \frac{\partial f_2}{\partial r_y^t} & \frac{\partial f_2}{\partial l_x^k} & \frac{\partial f_2}{\partial l_y^k} \end{bmatrix}$$

where

$$f_1 = {l_x}^k - r_x^t$$
$$f_2 = {l_y}^k - r_y^t$$

After solving, the Jacobian of the landmark measurement function is:

$$\boxed{H_l\left(\mathbf{r}^t, \mathbf{l}^k\right) = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}}$$

1.4. Exploit Sparsity

Question 4.

In the following section, the results for each method is presented and their efficiency (in terms of run time) is analyzed. First, all the visualization along with poses and landmarks for each method are shown. Secondly, their run times are mentioned in a tabular format, followed by observations and analysis.

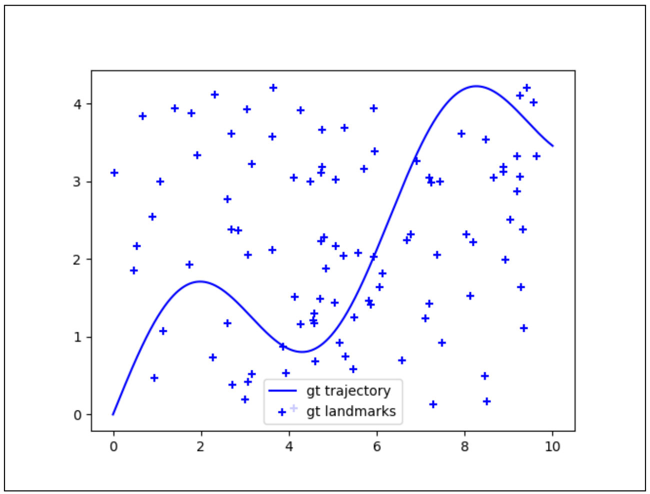
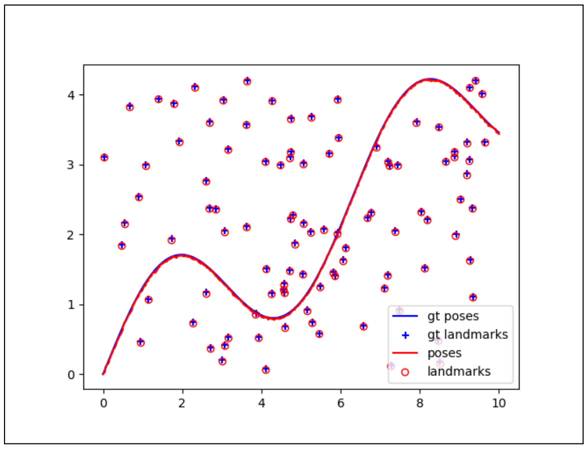
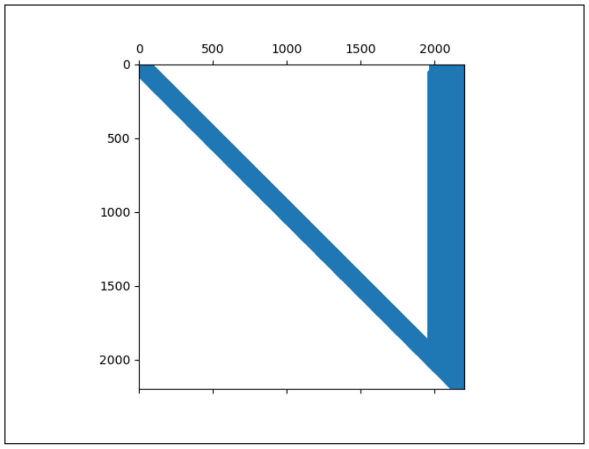


Figure 1: Ground truth for 2d_linear.npz

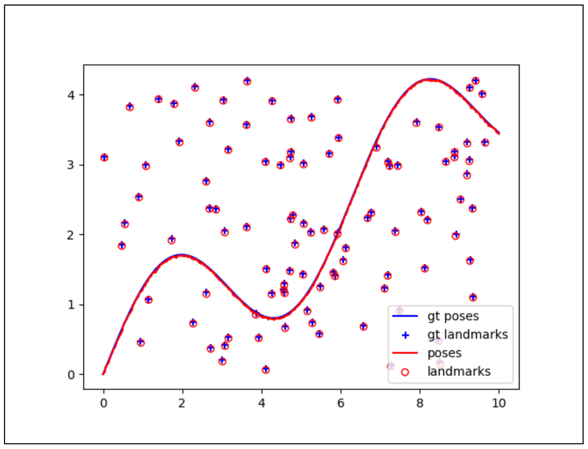


(a) Result using LU

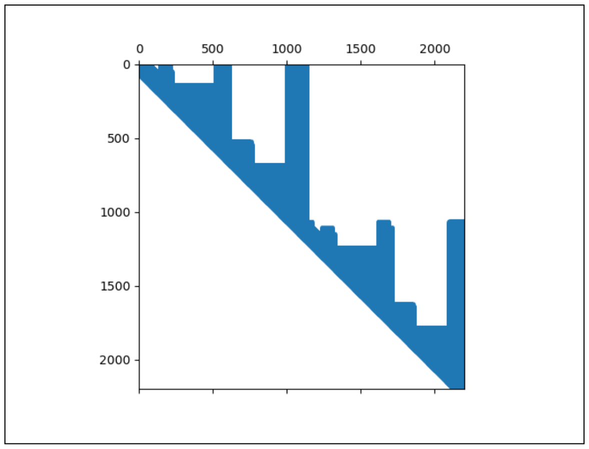


(b) U matrix using NATURAL

Figure 2: Method: LU

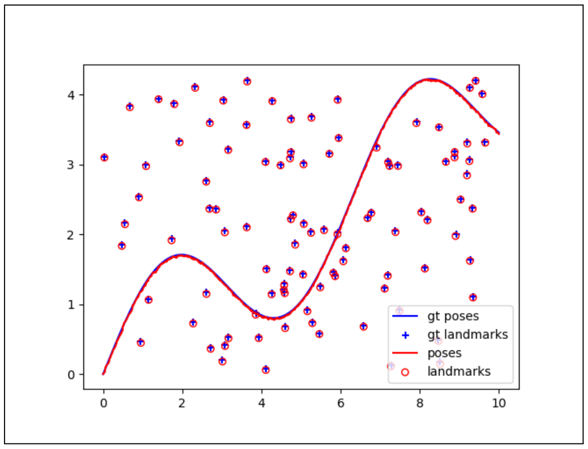


(a) Result using LU_COLAMD

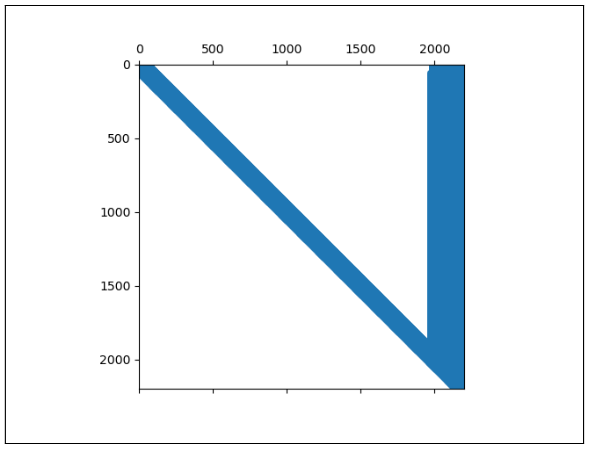


(b) U matrix after using COLAMD

Figure 3: Method: LU_COLAMD

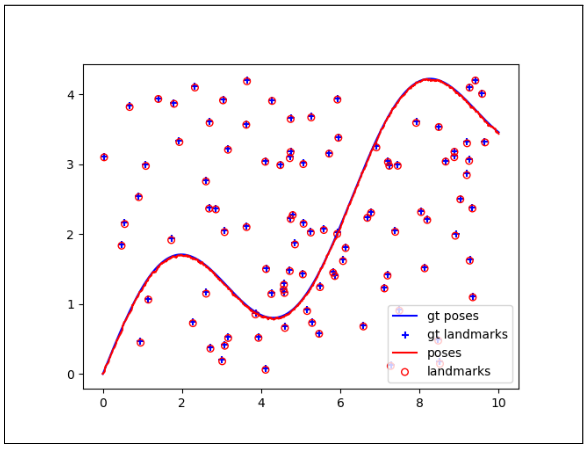


(a) Result using QR

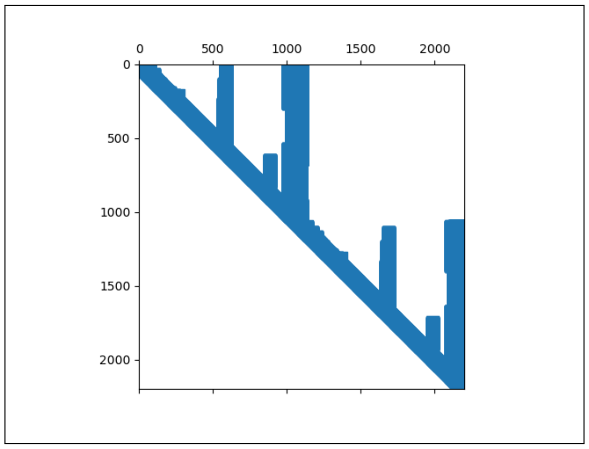


(b) R matrix

Figure 4: Method: QR



(a) Result using QR_COLAMD



(b) R matrix using COLAMD

Figure 5: Method: QR_COLAMD

Each of the solver was run with '–repeat 20' and their average time is reported. The following table shows the runtime for each of the methods:

Method	**Average Runtime**
lu	0.0345 s
lu_colamd	0.0572 s
qr	0.465 s
qr_colamd	0.278 s

Table 1: Average runtime for different solvers

Comparing LU and QR factorization in general, we can say that QR is is generally more numerically stable than LU. But for dense matrices, LU factorization is typically faster than QR factorization. However, for sparse matrices (which is often the case in SLAM problems), the difference can be less pronounced. The reason being LU factorization usually requires less memory than QR factorization.

Reordering using Column Approximate Minimum Degree (COLAMD) can significantly improve the performance of both QR and LU factorization for sparse matrices. COLAMD aims to reduce fill-in during factorization, which can lead to sparser factors and thus faster computations. By reordering the columns, the algorithm can better exploit the sparsity structure of the matrix.

From Table 1 we can observe LU is significantly faster than QR for both the natural ordering and COLAMD reordering. This is expected because LU factorization generally has a lower computational cost compared to QR factorization. LU factorization involves fewer arithmetic operations, which leads to faster run time.

LU_COLAMD (0.0572 s) is slower than LU (0.0345 s) which seems counterintuitive. There could be a possibility that it is due to the overhead introduced by reordering, which doesn't pay off for this particular problem size or sparsity pattern. It could be possible that COLAMD introduces additional complexity by rearranging rows and columns unnecessarily, leading to longer runtimes. In some cases, reordering can introduce more complexity than it saves in cases where the matrix is not sparse enough or if the fill-in reduction is minimal. We can also observe that the dataset shows a linear or sequential structure in terms of how poses and landmarks are related, which might not benefit much from reordering since there are fewer complex dependencies between variables.

QR_COLAMD (0.278 s) is faster than QR (0.465 s). This shows a clear improvement, as QR factorization benefits more from fill-in reduction due to its higher computational complexity. Reordering reduces the number of non-zero elements leading to faster computations in sparse QR factorization. Therefore to conclude, for this particular problem, LU without reordering is the fastest method overall.

Question 5.

Similarly, in the following section, the results for each method is presented and their efficiency (in terms of run time) is analyzed. First, all the visualization along with poses and landmarks for each method are shown. Secondly, their run times are mentioned in a tabular format, followed by observations and analysis.

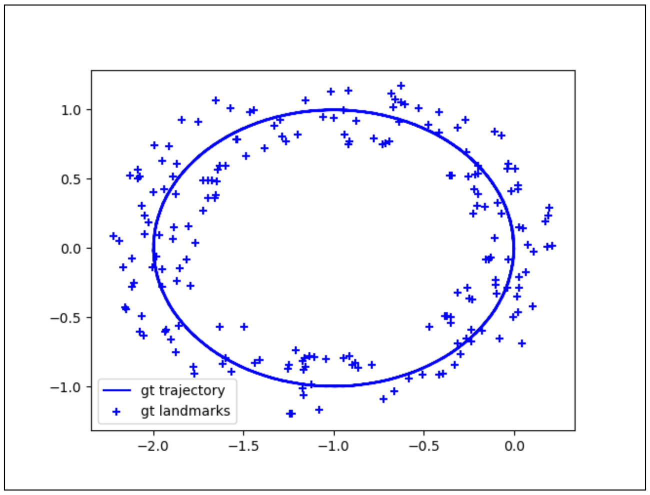
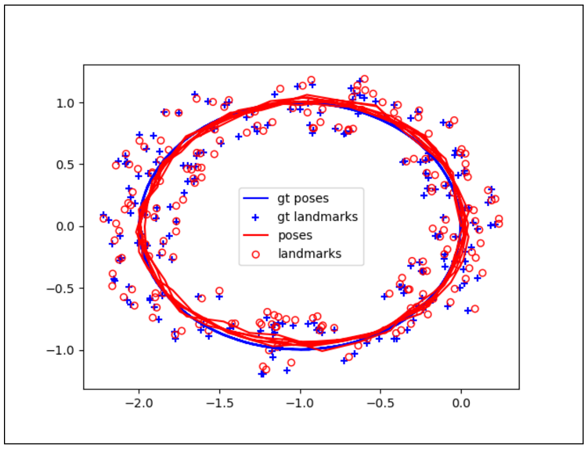
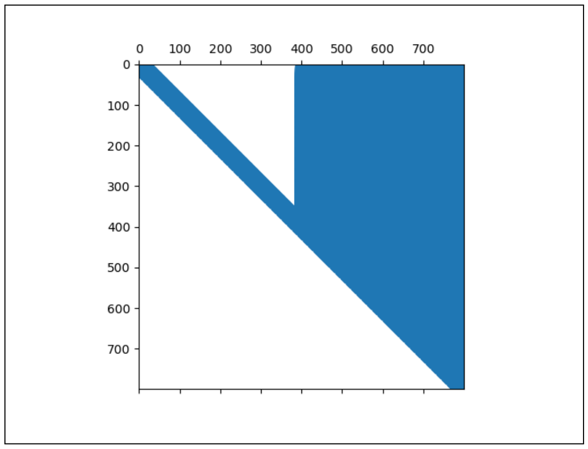


Figure 6: Ground truth for 2d_linear_loop.npz

‘

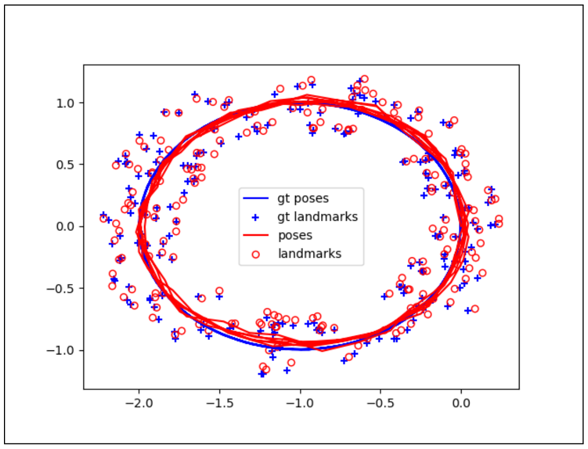


(a) Result using LU

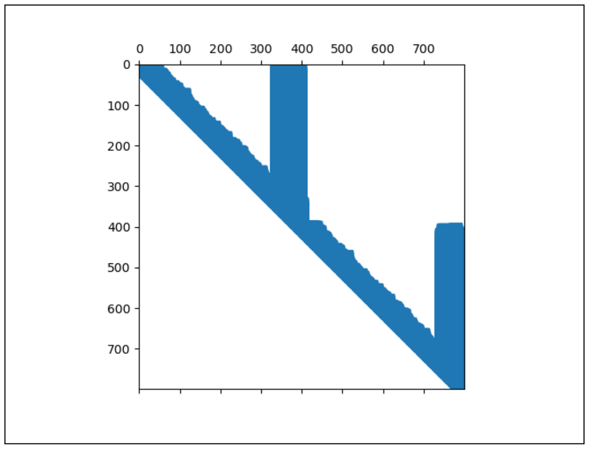


(b) U matrix

Figure 7: Method: LU

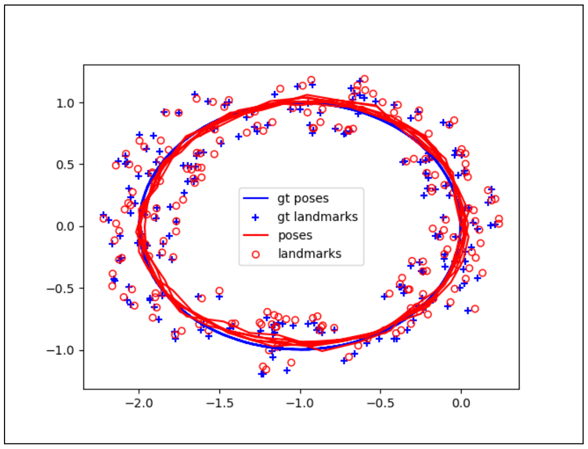


(a) Result using LU_COLAMD

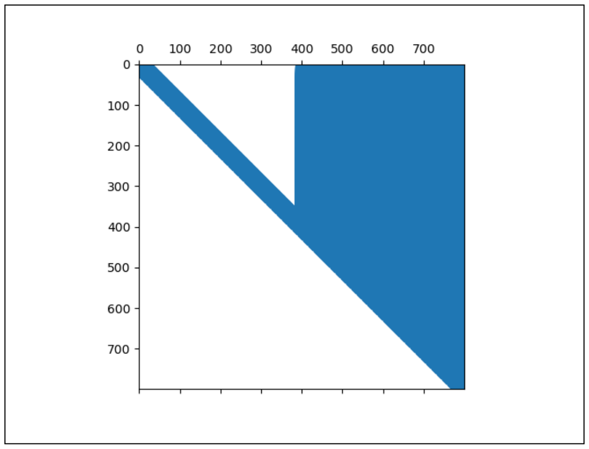


(b) U matrix after using COLAMD

Figure 8: Method: LU_COLAMD

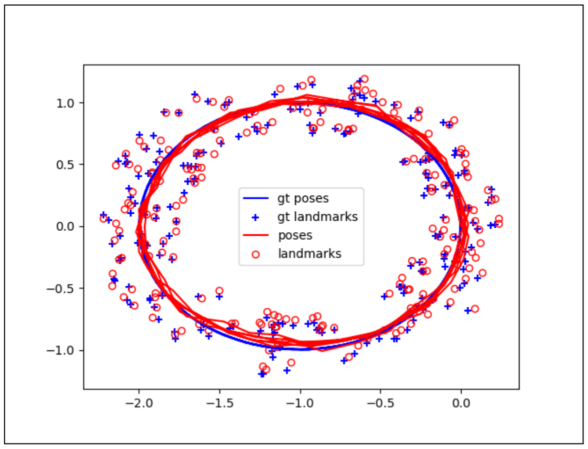


(a) Result using QR

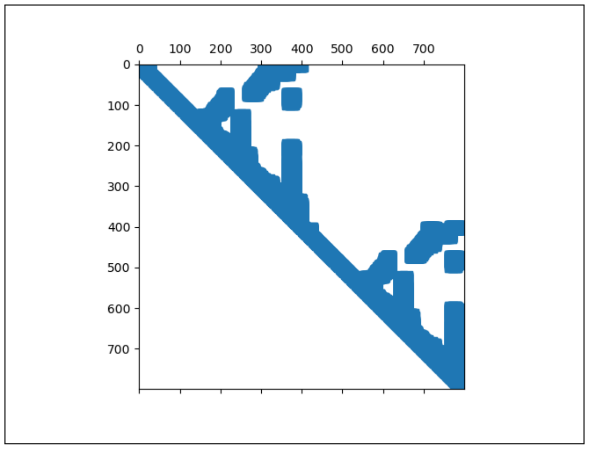


(b) R matrix

Figure 9: Method: QR



(a) Result using QR_COLAMD



(b) R matrix after using COLAMD

Figure 10: Method: QR_COLAMD

Each of the solver was run with '–repeat 20' and their average time is reported. The following table shows the runtime for each of the methods:

Method	Average Runtime
lu	0.0243 s
lu_colamd	0.00677 s
qr	0.20845 s
qr_colamd	0.02647 s

Table 2: Average runtime for different solvers

LU_COLAMD (0.00677 s) is significantly faster than LU (0.0243 s) in this case. This indicates that reordering using COLAMD was highly effective in reducing the fill-in for this dataset, which has a looped trajectory. For the looped dataset there are more inter-dependencies between poses and landmarks due to revisiting previously observed landmarks. We also notice that the U matrix is less dense (more sparse) in comparison with the previous dataset. In such cases, this might create opportunities for COLAMD to reduce fill-in by reorganizing variables (by breaking up dense blocks and redistributing non-zero elements more efficiently across the matrix) in a way that minimizes dependencies between distant parts of the trajectory. This leads to faster LU factorization after reordering. In contrast to the previous dataset (where LU_COLAMD was slower than LU), here reordering produces a substantial improvement in runtime.

QR_COLAMD is much faster than QR, similar to the previous dataset. The improvement is even more significant here compared to the non-looped dataset, where QR_COLAMD was still slower than LU but showed significant improvement over standard QR. As expected, LU is faster than QR, both with and without reordering. This follows the general trend observed in the previous dataset, where

LU factorization is computationally cheaper than QR factorization. However, with reordering, both methods show significant improvements.

When comparing these results with those from the previous dataset, reordering did not improve performance for LU (in fact, it made it slightly slower). However, for this looped trajectory dataset, reordering provides substantial speedups for both LU and QR. The looped trajectory likely introduces more sparsity into the system matrix, which allows COLAMD to reduce fill-in more effectively during factorization. In both datasets, QR factorization benefits significantly from reordering, but the improvement is even more pronounced in this looped dataset.

In conclusion, for this looped trajectory dataset, LU_COLAMD is the fastest method by a significant margin.

2. 2D Nonlinear SLAM

2.1. Measurement Function

The problem set-up is exactly the same as in the linear problem, except we introduce a nonlinear measurement function that returns a bearing angle and range,which together describe the vector from the robot to the landmark:

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

Simplifying the terms:

$$\begin{aligned} \Delta x &= {l_x}^k - r_x^t \\ \Delta y &= {l_y}^k - r_y^t \\ d &= \sqrt{(\Delta x)^2 + (\Delta y)^2} \\ f_1 = \text{atan2}\left(\Delta y, \Delta x\right); f_2 &= d \end{aligned}$$

The Jacobian $H_l\left(\mathbf{r}^t, \mathbf{l}^k\right)$ is defined as a 2 x 4 matrix as follows:

$$H_l\left(\mathbf{r}^t, \mathbf{l}^k\right) = \begin{bmatrix} \frac{\partial f_1}{\partial r_x^t} & \frac{\partial f_1}{\partial r_y^t} & \frac{\partial f_1}{\partial l_x^k} & \frac{\partial f_1}{\partial l_y^k} \\ \frac{\partial f_2}{\partial r_x^t} & \frac{\partial f_2}{\partial r_y^t} & \frac{\partial f_2}{\partial l_x^k} & \frac{\partial f_2}{\partial l_y^k} \end{bmatrix}$$

After solving, the Jacobian of the landmark measurement function for the nonlinear case is:

$$\boxed{H_l\left(\mathbf{r}^t, \mathbf{l}^k\right) = \begin{bmatrix} \frac{\Delta y}{d^2} & -\frac{\Delta x}{d^2} & -\frac{\Delta y}{d^2} & \frac{\Delta x}{d^2} \\ -\frac{\Delta x}{d} & -\frac{\Delta y}{d} & \frac{\Delta x}{d} & \frac{\Delta y}{d} \end{bmatrix}}$$

2.3. Solver

LU_COLAMD solver was used to run the 2d_nonlinear.npz dataset. The ground truth is shown followed by the results before and after optimization. This section is concluded by a discussion on the observations and differences in the optimization between linear and nonlinear SLAM.

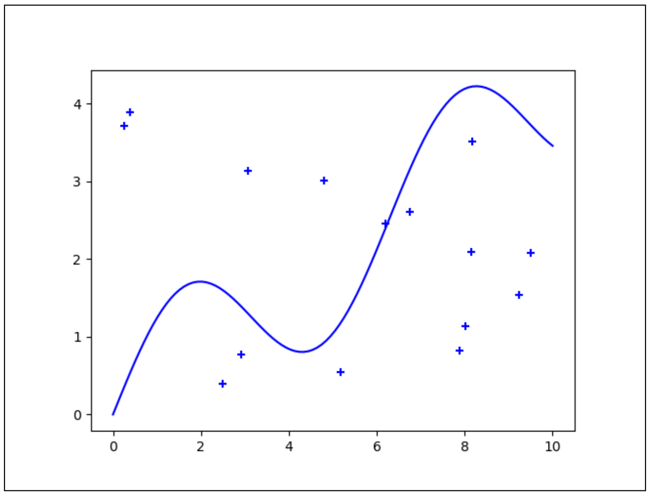
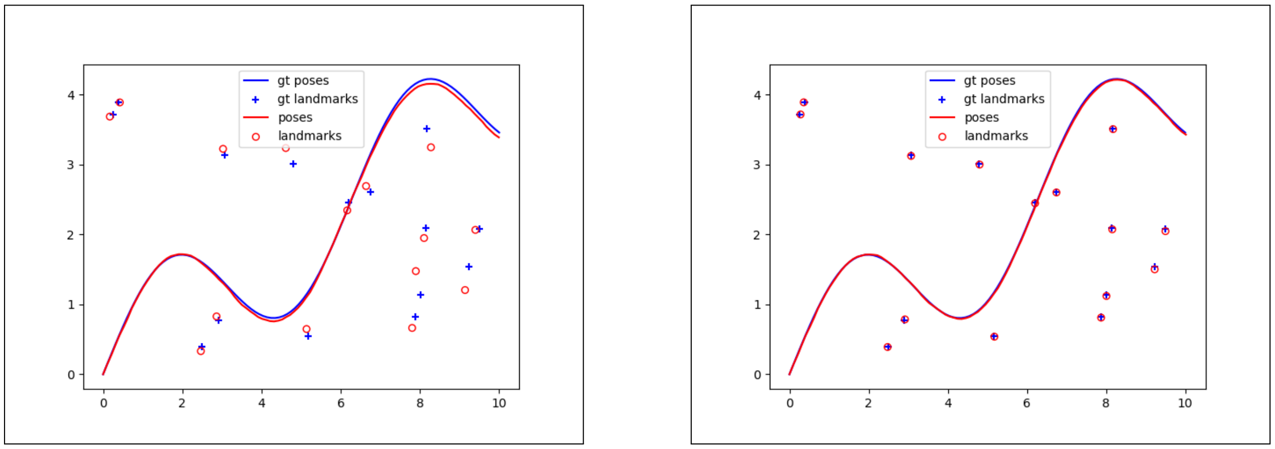


Figure 11: Ground truth for 2d_nonlinear.npz

‘



(a) Before Optimization

(b) After optimization

Figure 12: Method: LU_COLAMD

The optimization process in nonlinear SLAM involves iteratively refining the robot’s trajectory (poses) and landmark positions by solving a nonlinear least-squares problem. The goal is to minimize the residuals, which are the differences between observed measurements (e.g., odometry, landmark observations) and predicted measurements based on the current state estimate.

Before Optimization:
From Figure 12 (a) we can observe a noticeable discrepancy between the initial estimates (red) and the ground truth (blue). This is because the initial estimates are rough guesses based on noisy sensor data.

After Optimization:
After running the optimization process using lu_colamd, we observe from Figure 12 (b) that the red line (optimized poses) now closely matches the blue line (ground truth poses).The red circles (optimized landmarks) also align well with the blue crosses (ground truth landmarks).This indicates that the optimization successfully adjusted both the robot's trajectory and landmark positions to better fit the observed data.

Differences Between Linear and Nonlinear SLAM Optimization:
The key differences between linear and nonlinear SLAM problems lie in how they handle measurements and solve for robot poses and landmarks:
Linear SLAM:
Assumes linear relationships between variables and can be solved in one step using direct methods like least squares or matrix factorization. It is typically used in simpler scenarios where motion models and sensor measurements can be approximated as linear functions.

Nonlinear SLAM:
Deals with more realistic models where relationships between variables are nonlinear (e.g., bearing-range measurements, nonlinear motion models). It requires iterative optimization techniques such as Gauss-Newton or Levenberg-Marquardt to solve. At each iteration, a linear approximation of the nonlinear problem is solved, and then corrections are applied to update poses and landmarks. Nonlinear SLAM is more accurate but computationally more expensive due to its iterative nature.