Robot Localization and Mapping - Homework 3

Tianzhi Li
tianzhil@andrew.cmu.edu

March 30, 2025

1 2D Linear SLAM

1.1 Measurement function

Given the robot pose at time t: $\mathbf{r}^t = \left[r_x^t, r_y^t\right]^\top$ and the robot pose at time $t+1$: $\mathbf{r}^{t+1} = \left[r_x^{t+1}, r_y^{t+1}\right]^\top$, the odometry measurement function will be:

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

The Jacobian of h_o with respect to $\mathbf{r}^t$ and $\mathbf{r}^{t+1}$ is:

$$H_o(\mathbf{r}^t, \mathbf{r}^{t+1}) = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \tag{2}$$

Given the robot pose at time t: $\mathbf{r}^t = \left[r_x^t, r_y^t\right]^\top$ and the k-th landmark: $\mathbf{l}^k = \left[l_x^k, l_y^k\right]^\top$, the odometry measurement function will be:

$$h_l(\mathbf{r}^t, \mathbf{l}^k) = \begin{bmatrix} l_x^k - r_x^t \\ l_y^k - r_y^t \end{bmatrix} \tag{3}$$

The Jacobian of h_o with respect to $\mathbf{r}^t$ and $\mathbf{r}^{t+1}$ is:

$$H_l(\mathbf{r}^t, \mathbf{l}^k) = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \tag{4}$$

1.2 Build a linear system

The implementation is in `linear.py`.

1.3 Solvers

The implementation is in `solvers.py`.

Solver	Average Time (s)
`qr`	0.1423
`qr_colamd`	0.1273
`lu`	0.0169
`lu_colamd`	0.0454

Table 1: Runtime on `2d_linear.npz`

1.4 Exploit sparsity

1.4.1

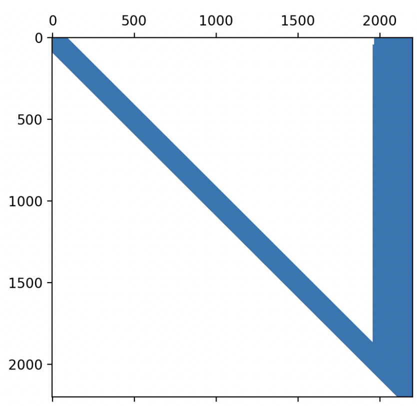
The implementation is in `solve_lu_colamd` in `solvers.py`.

1.4.2

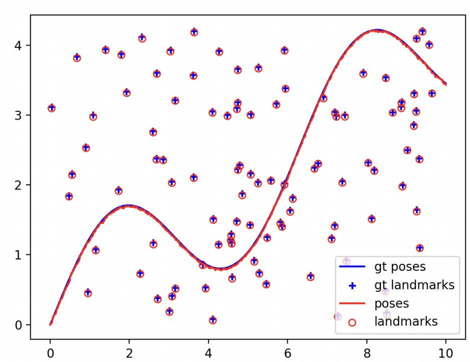
1.4.3

The implementation is in `solve_qr_colamd` in `solvers.py`.

1.4.4 `2d_linear.npz`



(a) R matrix

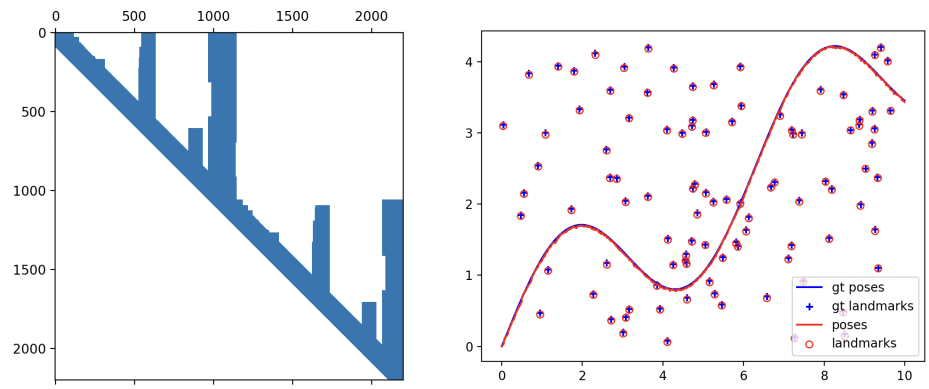


(b) Trajectory and landmarks

Figure 1: Visualization for `qr` method

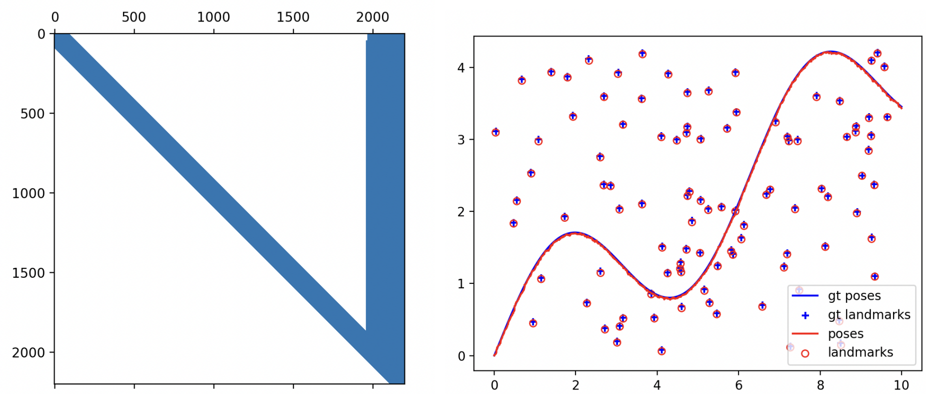
From the runtime results shown in Table 1, the LU factorization is significantly faster than the QR factorization. As mentioned in the lecture, QR factorization is more numerically stable and requires more computations for factorization (QR has $\frac{4}{3}n^3$ time complexity while LU has $\frac{2}{3}n^3$ time complexity).

What we expected to see for reordered versions is the reordered version is faster than the original method. However, interestingly, the reordering improved performance for QR but led to slower LU. The potential reason is that the overhead from reordering outweighed the sparsity benefits in this case.



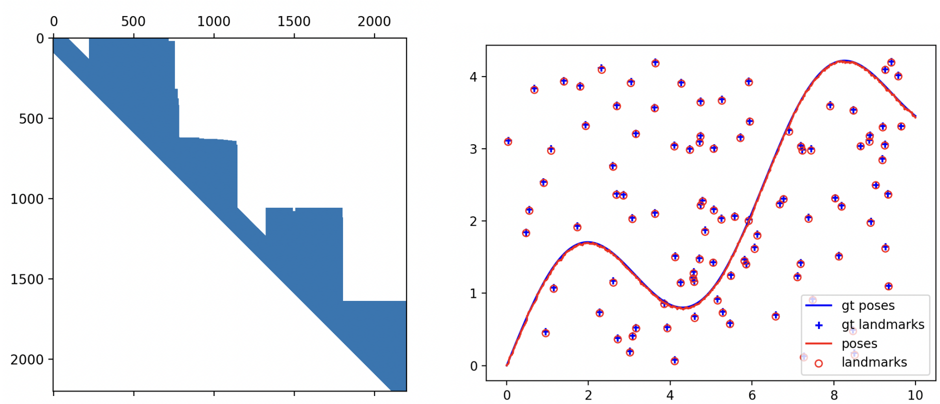
(a) R matrix (b) Trajectory and landmarks

Figure 2: Visualization for `qr_colamd` method



(a) R matrix (b) Trajectory and landmarks

Figure 3: Visualization for `lu` method



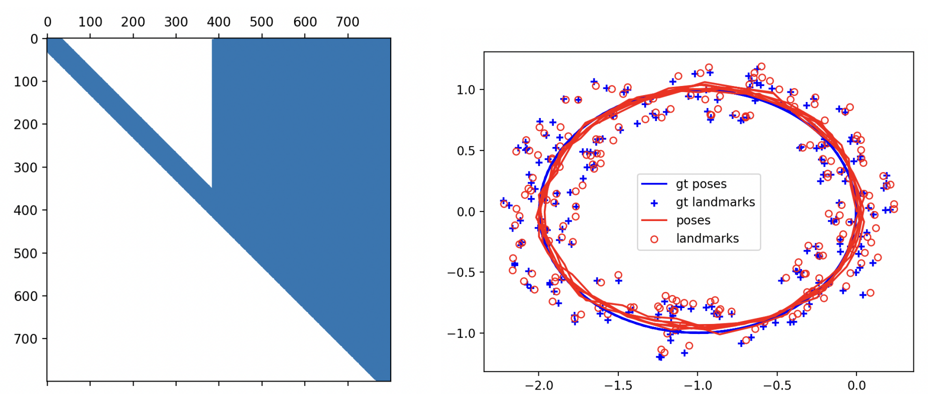
(a) R matrix (b) Trajectory and landmarks

Figure 4: Visualization for `lu_colamd` method

Solver	Average Time (s)
`qr`	0.0710
`qr_colamd`	0.0117
`lu`	0.0113
`lu_colamd`	0.0048

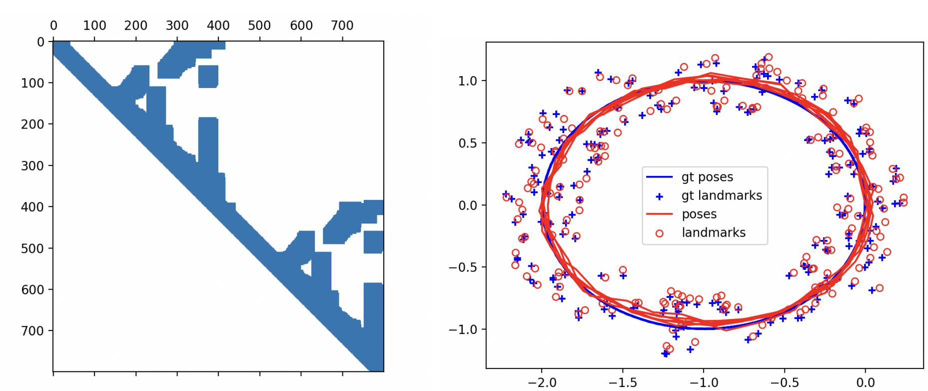
Table 2: Runtime on `2d_linear_loop.npz`

1.4.5 `2d_linear_loop.npz`



(a) R matrix (b) Trajectory and landmarks

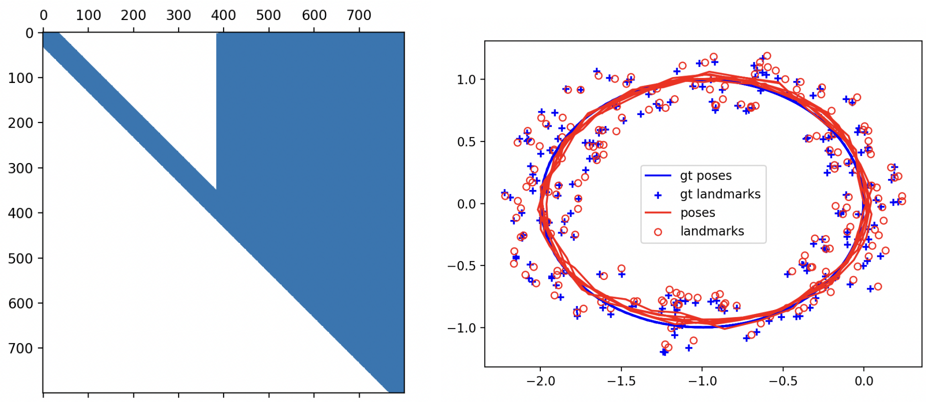
Figure 5: Visualization for `qr` method



(a) R matrix (b) Trajectory and landmarks

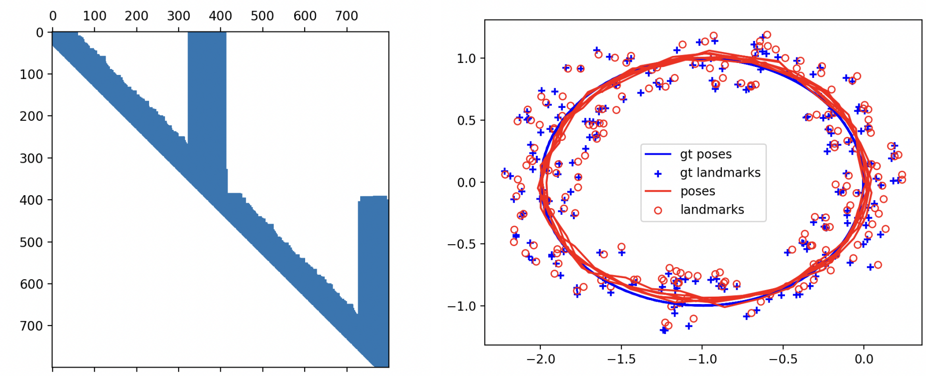
Figure 6: Visualization for `qr_colamd` method

From the runtime results shown in Table 2, as before in `2d_linear.npz`, the LU factorization is faster than the QR factorization. The different part is that reordering is highly beneficial in both the LU factorization and the QR factorization. The potential reason is



(a) R matrix (b) Trajectory and landmarks

Figure 7: Visualization for `lu` method



(a) R matrix (b) Trajectory and landmarks

Figure 8: Visualization for `lu_colamd` method

that in the loop case, the state has repeating or overlapping variables (e.g., poses or landmarks that are reused due to the robot revisiting previous locations). This structure creates a much denser and more regular sparsity pattern. Reordering works effectively here because it optimizes the sparsity structure of the Jacobian matrix, reducing fill-in (extra non-zero elements in the matrix) and improving efficiency by minimizing computational overhead during factorization.

2 2D Nonlinear SLAM

2.1 Measurement function

Given the measurement function from the robot to the landmark:

$$h_l\left(\mathbf{r}^t, \mathbf{l}^k\right) = \begin{bmatrix} \operatorname{atan2}\left(l_y^k - r_y^t, l_x^k - r_x^t\right) \\ \left(\left(l_x^k - r_x^t\right)^2 + \left(l_y^k - r_y^t\right)^2\right)^{\frac{1}{2}} \end{bmatrix} = \begin{bmatrix} \theta \\ d \end{bmatrix} \tag{5}$$

$$H_l\left(\mathbf{r}^t, \mathbf{l}^k\right) = \begin{bmatrix} \frac{\partial \theta}{\partial r_x^t} & \frac{\partial \theta}{\partial r_y^t} & \frac{\partial \theta}{\partial l_x^k} & \frac{\partial \theta}{\partial l_y^k} \\ \frac{\partial d}{\partial r_x^t} & \frac{\partial d}{\partial r_y^t} & \frac{\partial d}{\partial l_x^k} & \frac{\partial d}{\partial l_y^k} \end{bmatrix} \tag{6}$$

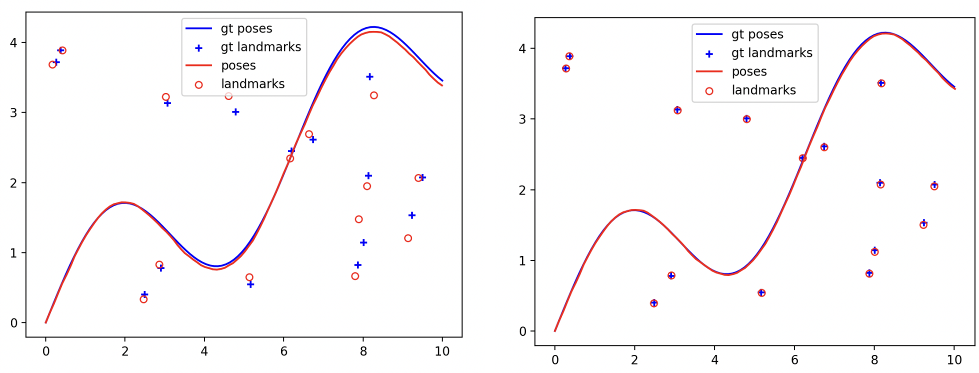
$$= \begin{bmatrix} \frac{l_y^k - r_y^t}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} & -\frac{l_x^k - r_x^t}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} & -\frac{l_y^k - r_y^t}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} & \frac{l_x^k - r_x^t}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} \\ -\frac{l_x^k - r_x^t}{((l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2)^{\frac{1}{2}}} & -\frac{l_y^k - r_y^t}{((l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2)^{\frac{1}{2}}} & \frac{l_x^k - r_x^t}{((l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2)^{\frac{1}{2}}} & \frac{l_y^k - r_y^t}{((l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2)^{\frac{1}{2}}} \end{bmatrix} \tag{7}$$

2.2 Build a linear system

The implementation is in `nonlinear.py`.

2.3 Solver

The visualization is shown in Figure 9.



(a) Before optimization (b) After optimization

Figure 9: Visualization for `lu` method

Linear SLAM solves state estimation in one step using direct linear algebra methods like QR or LU factorization, as the system dynamics and observations are linear. The Jacobian remains constant, and the solution is globally optimal. Non-linear SLAM involves iterative optimization, like Gauss-Newton or Levenberg-Marquardt, to linearize the problem at each step. The Jacobian changes dynamically, requiring multiple updates to refine the state estimate. It handles real-world sensor models but is computationally expensive and sensitive to initialization. While linear SLAM is efficient, non-linear SLAM is more flexible and accurate, especially in complex, large-scale environments.