16-833 SLAM HW3: Linear and Nonlinear SLAM Solvers

Ryan Wu (Andrew ID: weihuanw)

March 29, 2024

1 2D Linear SLAM

1.1 Measurement function

In a 2D linear case, the measurement function and its Jacobian with respect to the robot poses and time steps are given by:

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

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

In a 2D linear case, the measurement function and its Jacobian with respect to the robot poses and landmark positions are given by:

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

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

1.2 Build a linear system

The linear system was built successfully with all the necessary functions completed. Results are shown in Section 1.4.

1.3 Solvers

All 5 solver functions (`pinv`, `lu`, `qr`, `lu_cholmod`, `qr_cholmod`) were implemented with the results shown in Section 1.4.

1.4 Exploit sparsity

For the `2d_linear.npz` dataset, the solvers' results are the following:

Solver Method	Average Computation Time (s)
default	0.1520
pinv	1.6435
lu	0.0180
qr	1.9620
`lu_colamd`	0.1511
`qr_colamd`	1.3179

Table 1: Solvers' run time efficiency with `2d_linear.npz` dataset.

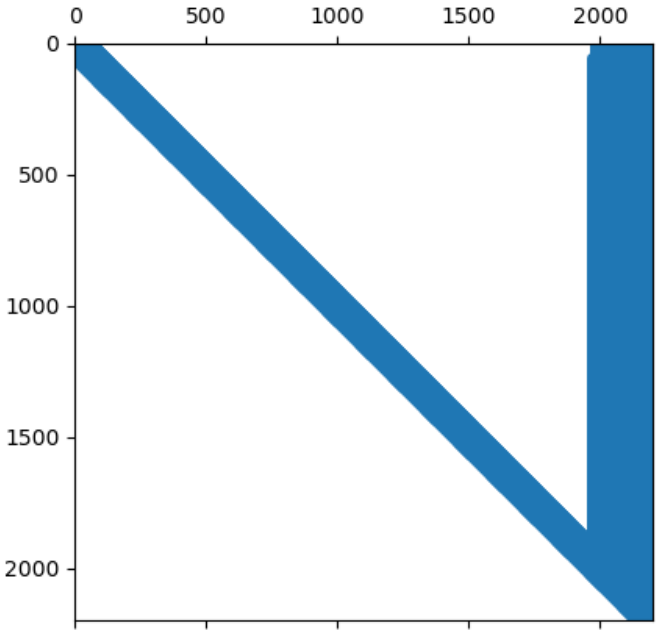

Figure 1: Triangular matrix using the qr solver.

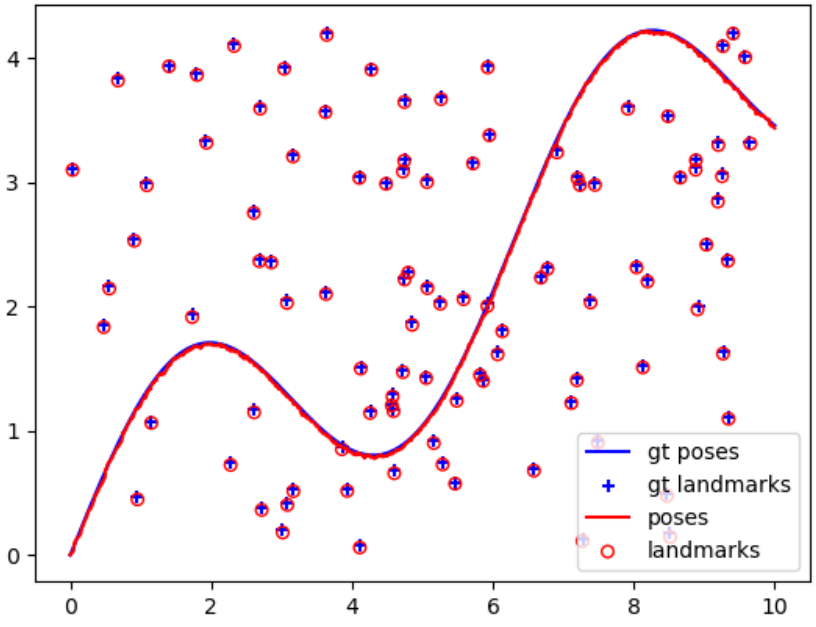

Figure 2: Estimated pose and landmarks using the qr solver.

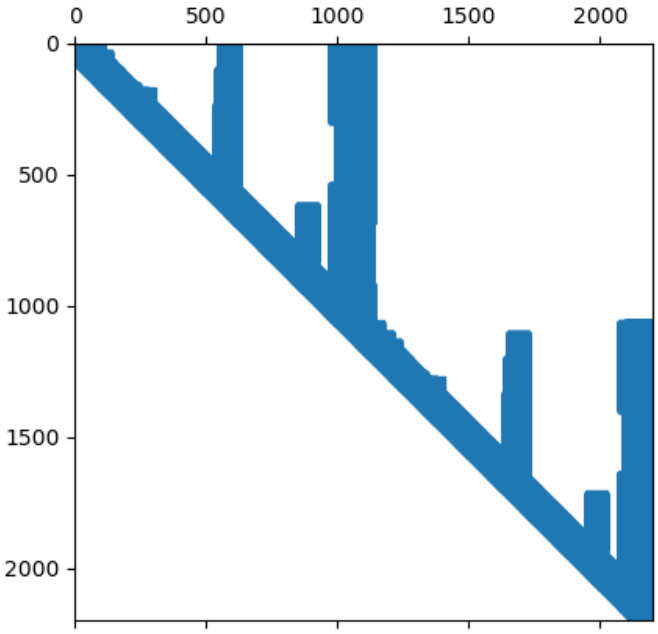

Figure 3: Triangular matrix using the `qr_colamd` solver.

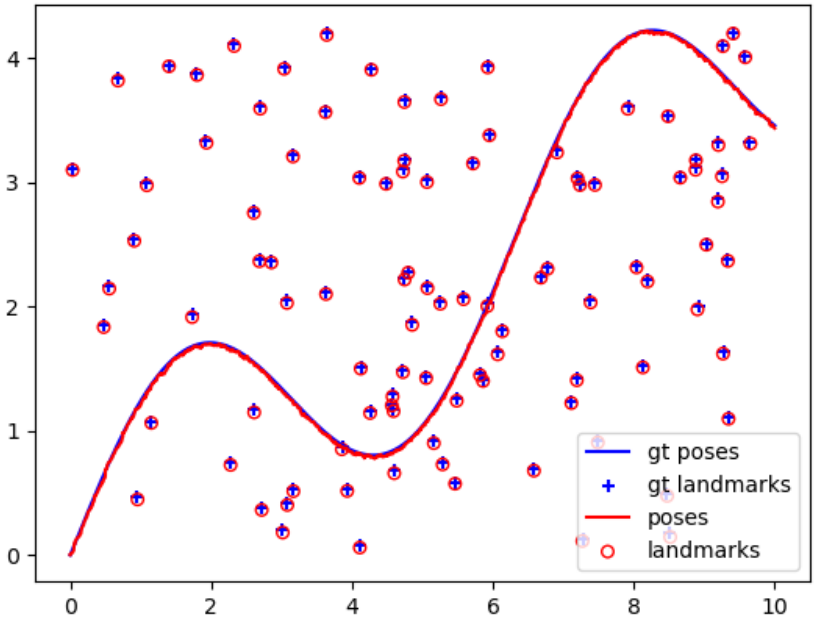

Figure 4: Estimated pose and landmarks using the `qr_colamd` solver.

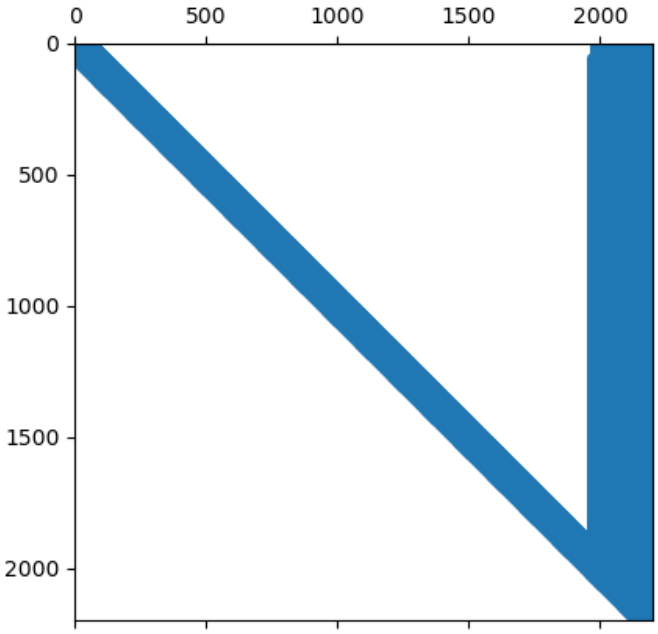

Figure 5: Triangular matrix using the lu solver.

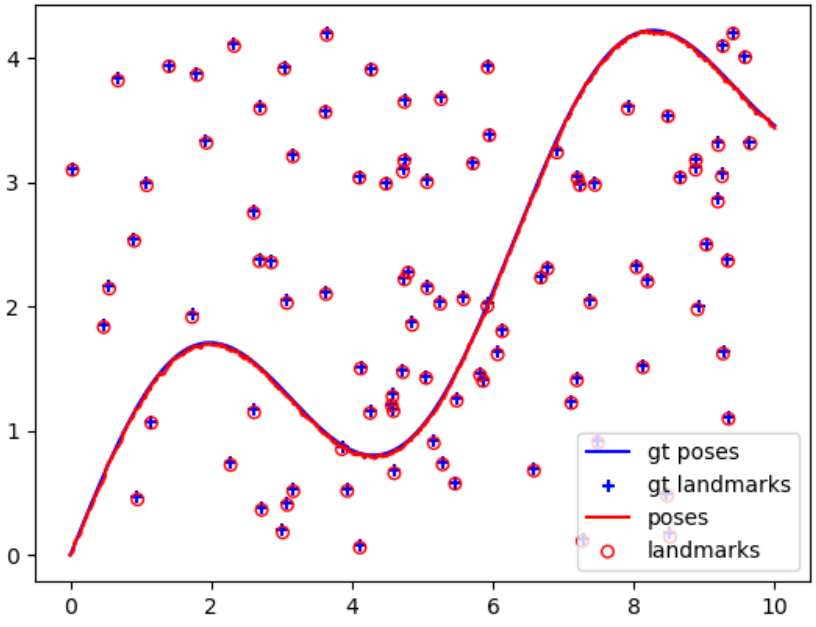

Figure 6: Estimated pose and landmarks using the lu solver.

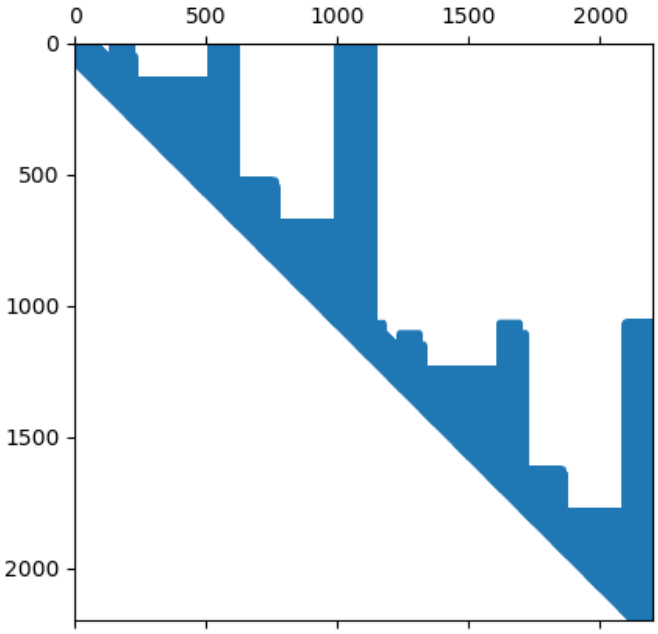

Figure 7: Triangular matrix using the `lu_colamd` solver.

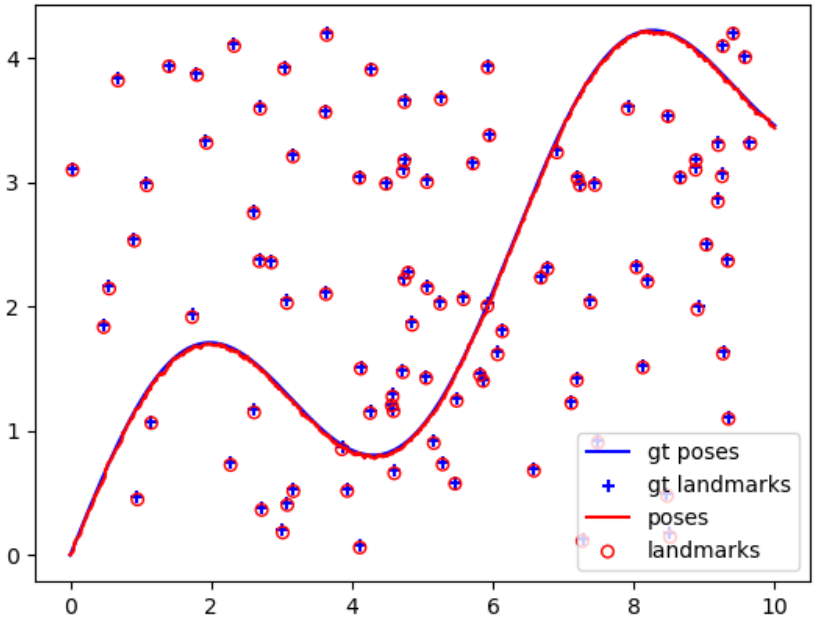

Figure 8: Estimated pose and landmarks using the `lu_colamd` solver.

The above results showed that the solver efficiency for the `2d_linear.npz` dataset from the fastest to the slowest is: lu, `lu_colamd`, `qr_colamd`, qr. Generally, the lu methods proves to solve faster compared to the qr methods and the COLAMD matrix reordering algorithm further improves computation time by minimizing the fill-in. However, lu solved faster than `lu_colamd` in this case, which can indicate the given `2d_linear.npz` dataset may have certain sparsity patterns or low fill-ins that do not benefit from COLAMD reordering. The COLAMD matrix reordering process can indirectly introduce additional computational overhead.

For the `2d_linear_loop.npz` dataset, the solvers' results are the following:

Solver Method	**Average Computation Time (s)**
default	0.0030
pinv	0.1063
lu	0.0151
qr	2.1665
`lu_colamd`	0.0057
`qr_colamd`	0.0299

Table 2: Solvers' run time efficiency with `2d_linear_loop.npz` dataset.

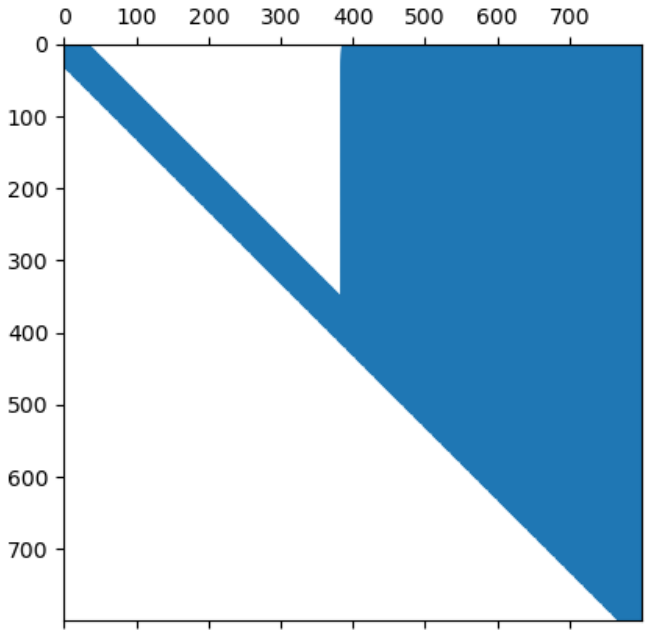

Figure 9: Triangular matrix using the qr solver.

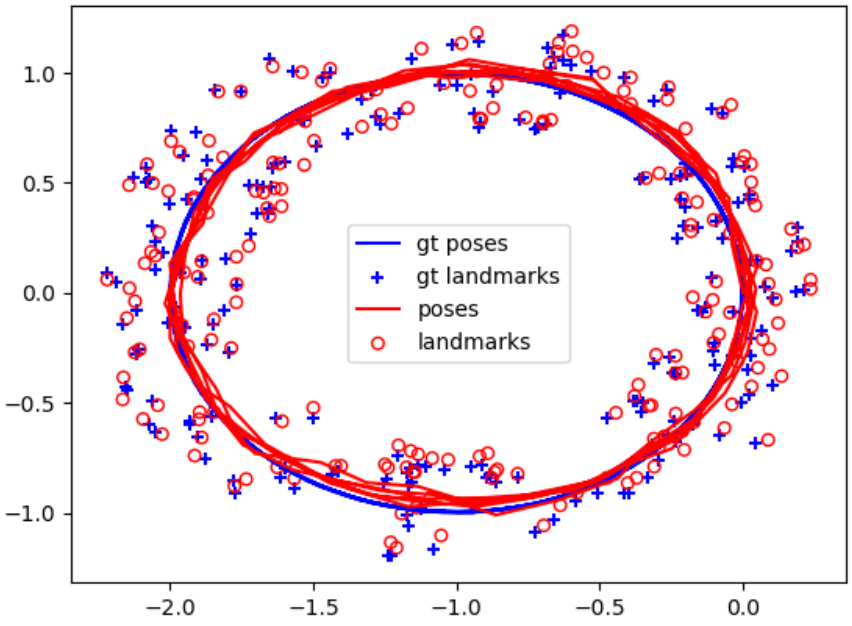

Figure 10: Estimated pose and landmarks using the qr solver.

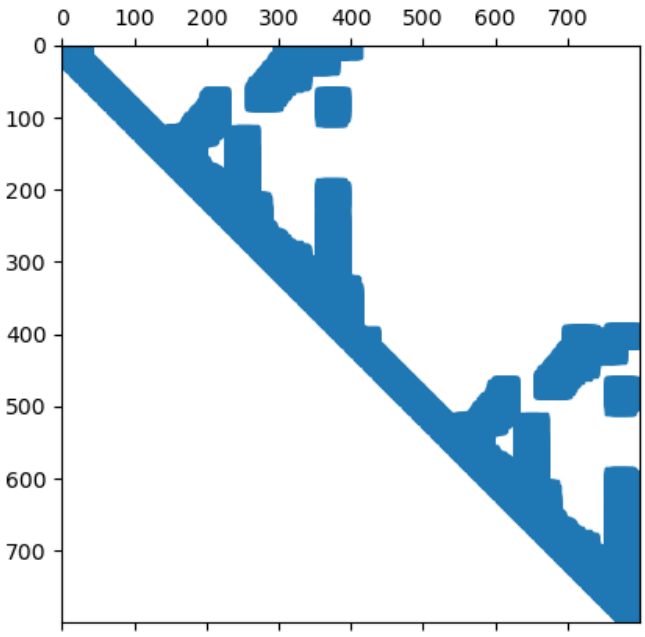

Figure 11: Triangular matrix using the `qr_colamd` solver.

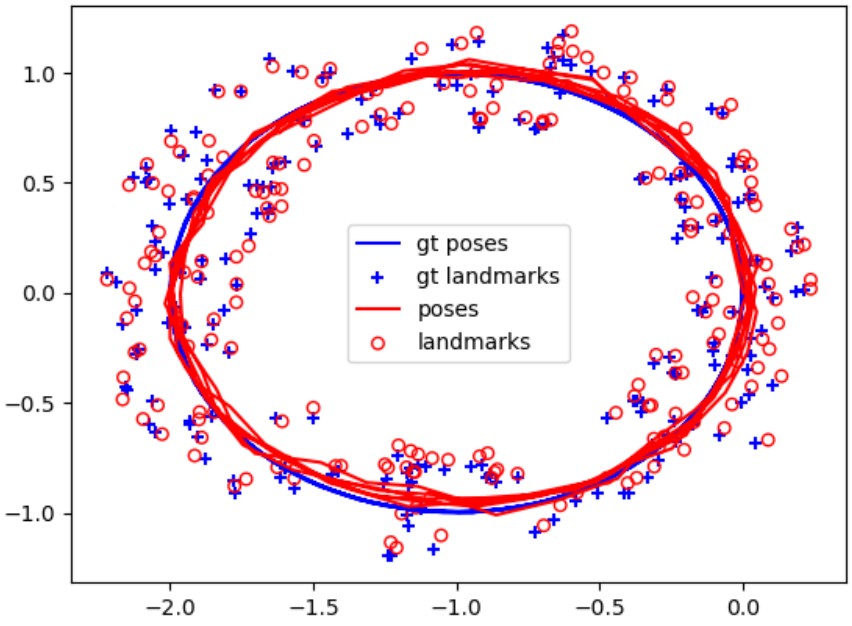

Figure 12: Estimated pose and landmarks using the `qr_colamd` solver.

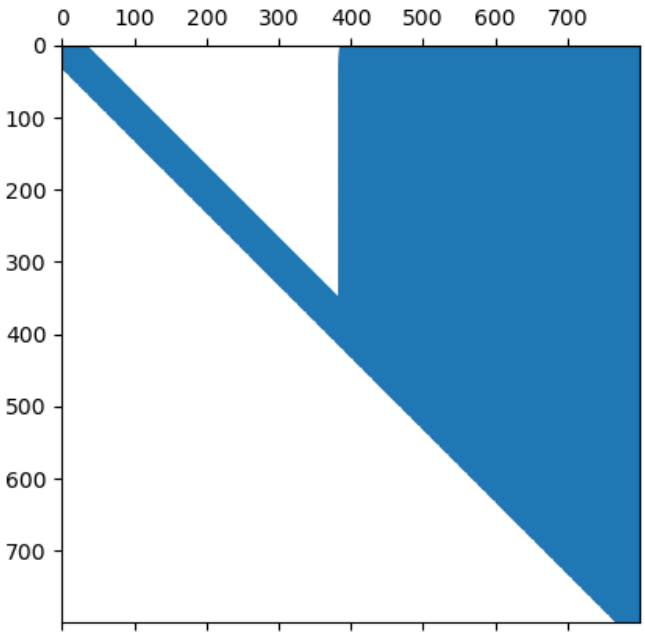

Figure 13: Triangular matrix using the lu solver.

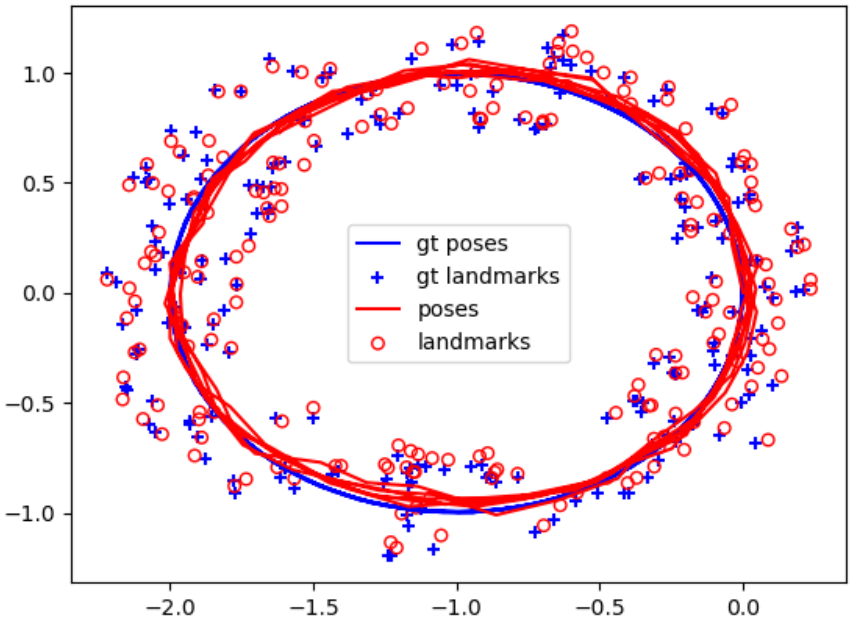

Figure 14: Estimated pose and landmarks using the lu solver.

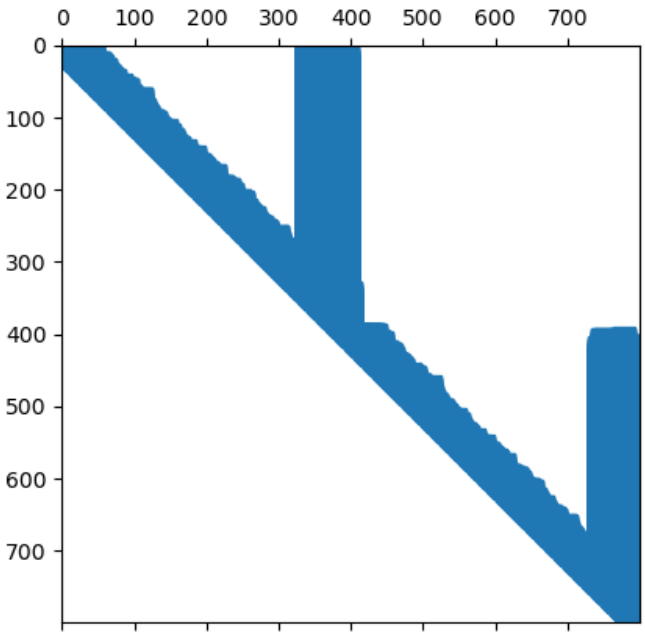

Figure 15: Triangular matrix using the `lu_colamd` solver.

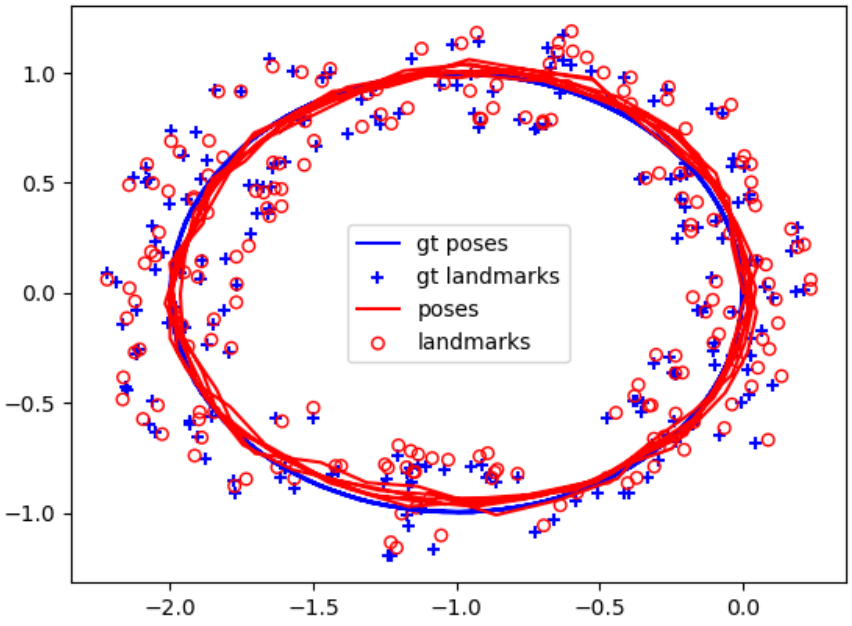

Figure 16: Estimated pose and landmarks using the `lu_colamd` solver.

The above results showed that the solver efficiency for the `2d_linear_loop.npz` dataset from the fastest to the slowest is: `lu_colamd`, `qr_colamd`, lu, qr. There are slight differences in efficiency compared to the `2d_linear.npz` dataset. In the loop case, the COLAND matrix reordering and reducing fill-in greatly improved run time. Instead of creating unnecessary computational overhead, the loop case actually benefited from reordering since the dataset is larger.

2 2D Nonlinear SLAM

2.1 Measurement function

In a 2D nonlinear case, the measurement function and its Jacobina with respect to the robot poses and landmark positions are given by:

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

$$H_l(r^t, l^k) = \begin{bmatrix} \frac{l_y^k - r_y^t}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} & \frac{-(l_x^k - r_x^t)}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} & \frac{-(l_y^k - r_y^t)}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} & \frac{l_x^k - r_x^t}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} \\ \frac{-(l_x^k - r_x^t)}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}} & \frac{-(l_y^k - r_y^t)}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}} & \frac{l_x^k - r_x^t}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}} & \frac{l_y^k - r_y^t}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}} \end{bmatrix}$$

2.2 Build a linear system

The linear system was built successfully with all the necessary functions completed. Results are shown in Section 2.3.

2.3 Solver

For the `2d_nonlinear.npz` data set, we used the pinv solver to analyze the dataset and the results are the following:

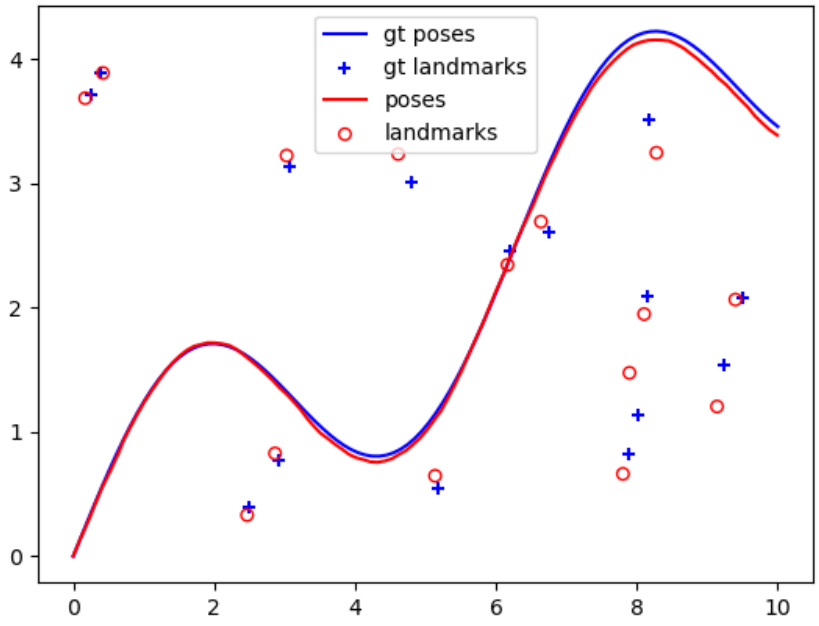

Figure 17: Estimated pose and landmarks using the pinv solver without optimization.

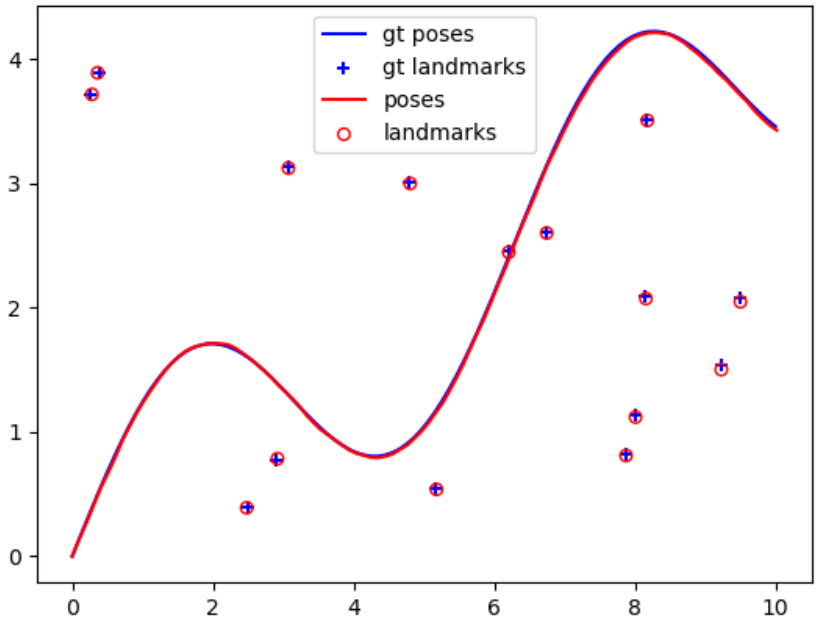

Figure 18: Estimated pose and landmarks using the pinv solver with optimization.

For non-linear SLAM problems, we must linearize the non-linear system before optimization. In particular, one must define a linearization point for the overall system. Furthermore, we can directly solve the optimal state vectors in linear SLAM problems. On the other hand, non-linear problems can only be solved through batch processing and iteration until convergence.

References

Dellaert, F., & Kaess, M. *Factor Graphs for Robot Perception.* Now Publishers, 2017.