Homework 3 Linear and Nonlinear SLAM Solvers

16-833: Robot Localization and Mapping

Austin Windham

1.1 Measurement function

$$r^t = \begin{bmatrix} r_x^t \\ r_y^t \end{bmatrix}, \quad r^{t+1} = \begin{bmatrix} r_x^{t+1} \\ r_y^{t+1} \end{bmatrix}$$

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

$$H_o(r^t, r^{t+1}) = \frac{\partial h_o}{\partial X} = \begin{bmatrix} \frac{\partial h_o}{\partial r_x^t}, & \frac{\partial h_o}{\partial r_y^t}, & \frac{\partial h_o}{\partial r_x^{t+1}}, & \frac{\partial h_o}{\partial r_y^{t+1}} \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

$$h_\ell(r^t, \ell^n) = \ell_n - r^t = \begin{bmatrix} \ell_x^n - r_x^t \\ \ell_y^n - r_y^t \end{bmatrix}$$

$$H_\ell(r^t, \ell^n) = \frac{\partial h_\ell}{\partial X} = \begin{bmatrix} \frac{\partial h_\ell}{\partial r_x^t}, & \frac{\partial h_\ell}{\partial r_y^t}, & \frac{\partial h_\ell}{\partial \ell_x^n}, & \frac{\partial h_\ell}{\partial \ell_y^n} \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

1.2 Build a linear System

Code was completed in “linear.py”.

1.3 Solvers

Code was completed in “solvers.py”

1.4 Exploit sparsity

1-3. Code was completed in “solvers.py”.

4. The results below relate to running “linear.py” with the “2d_linear.npz” file. The visualizations of the trajectory and landmarks for each of the four methods can be seen below. Below those plots are the plots of the sparsity pattern of the R matrix for the four methods. Lastly, a table of the efficiencies for all 5 methods can be seen in the table below.

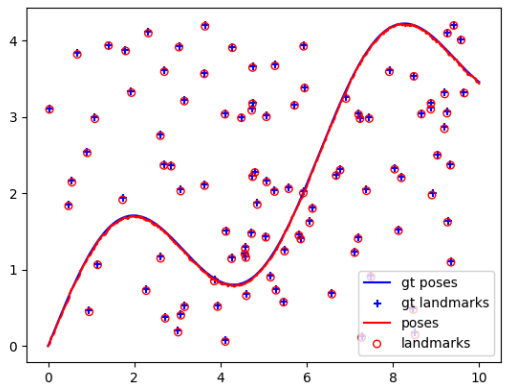

Figure 1: Visualization of LU Method

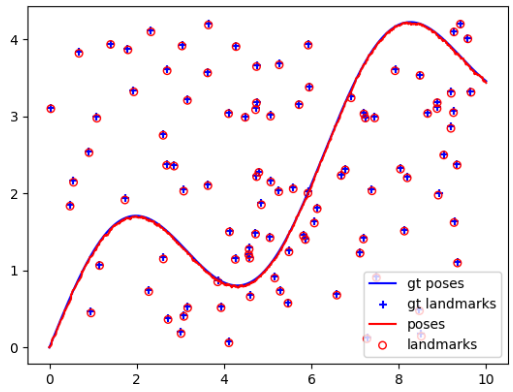

Figure 2: Visualization of LU-COLAMD Method

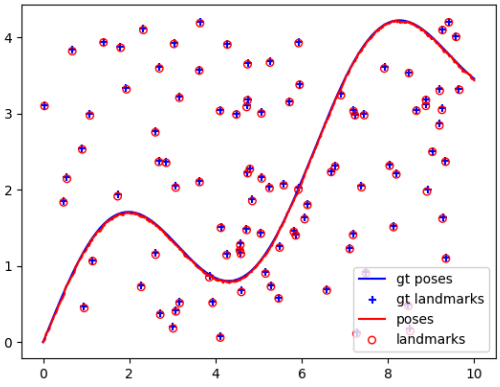

Figure 3: Visualization of QR Method

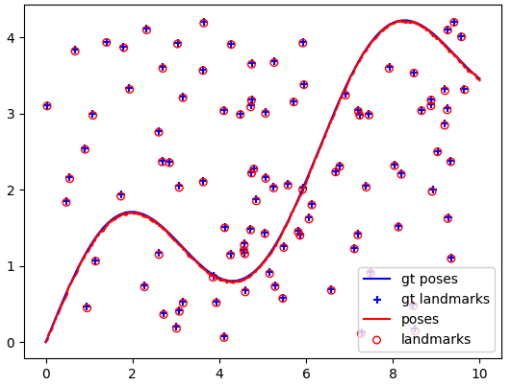

Figure 4: Visualization of QR-COLAMD Method

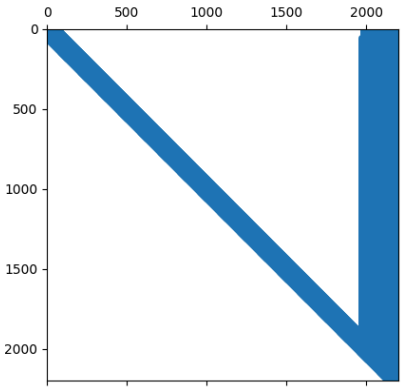

Figure 5: R Matrix with LU Method

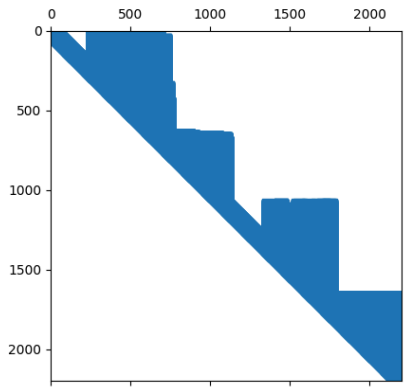

Figure 6: R Matrix with LU-COLAMD Method

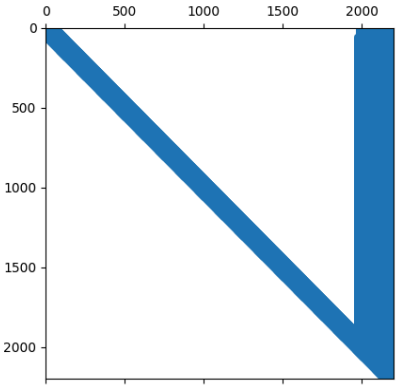

Figure 7: R Matrix with QR Method

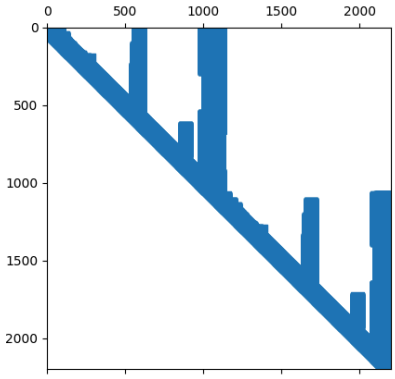

Figure 8: R Matrix with QR-COLAMD Method

Method	Efficiency (s)
P_Inv	1.9572700023651124
LU	0.0228787899017334
LU-COLAMD	0.07007045745849609
QR	0.46349329948425294
QR-COLAMD	0.3977679252624512

Table 1: Efficiencies for 2d_linear.npz averaged over 5 iterations.

From the visualizations of the landmarks and the poses, it is clear that all four methods were able to accurately estimate the locations of the robot and landmarks, indicating that each method effectively solved the SLAM equation. The efficiency of each method can be ranked as LU, LU-COLAMD, QR-COLAMD, QR, and then P_Inv. The pseudo-inverse method was the slowest, as expected, because it involves computing A^TA and its inverse, which is computationally intensive.

The LU methods were faster than the QR methods, which aligns with general expectations since LU is typically faster than QR but may be slightly less numerically stable. Additionally, the LU methods produced slightly more sparse matrices, as shown in the R-matrix visualizations. Sparsity here refers to the proportion of zero entries in the matrices; a more sparse matrix has more zeroes, which reduces the computational burden and speeds up the solution.

Reordering columns using the COLAMD method is expected to improve sparsity further by minimizing the number of non-zero entries created during factorization. While this reordering does increase efficiency for QR methods, it is interesting to note that LU-COLAMD was actually slightly less efficient than LU. This could be because the baseline runtime for LU is already very low, so the additional computational overhead of reordering outweighs the gains in sparsity for LU in this case.

5. The results below relate to running “linear.py” with the “2d_linear_loop.npz” file. The visualizations of the trajectory and landmarks for each of the four methods can be seen below. Below those plots are the plots of the sparsity pattern of the R matrix for the four methods. Lastly, a table of the efficiencies for all 5 methods can be seen in the table below.

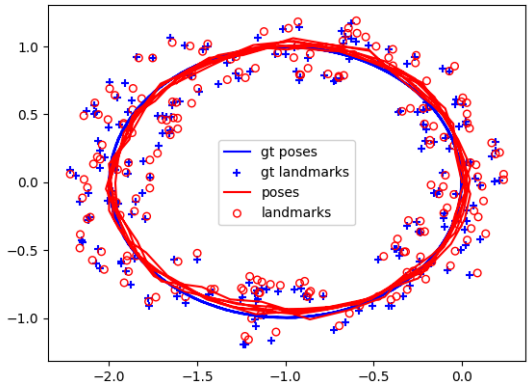

Figure 9: Visualization with LU Method

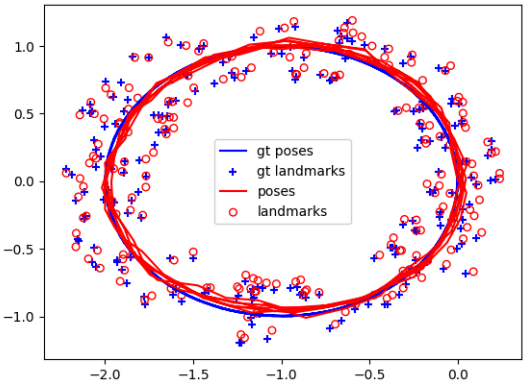

Figure 10: Visualization with LU-COLAMD Method

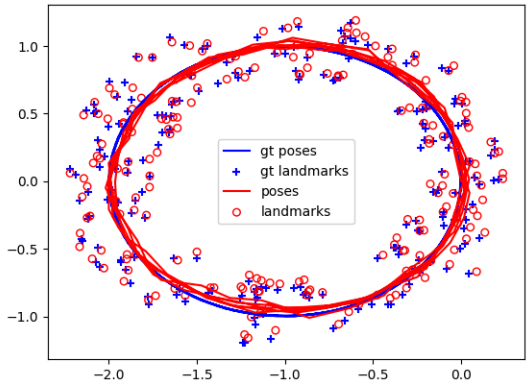

Figure 11: Visualization with QR Method

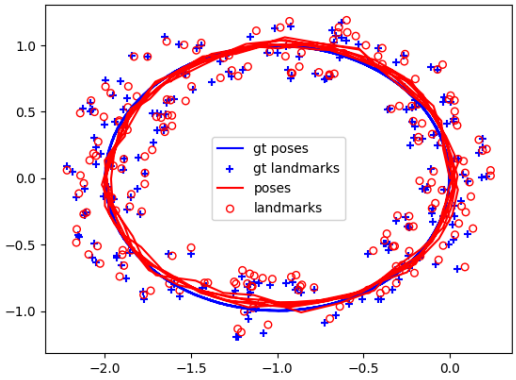

Figure 12: Visualization with QR-COLAMD Method

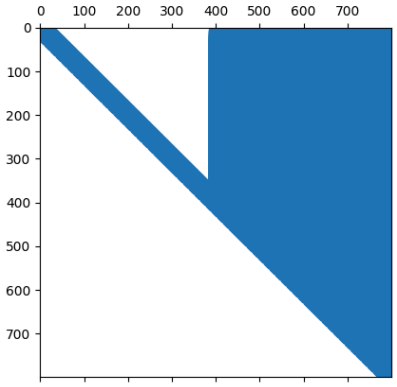

Figure 13: R-Matrix with LU Method

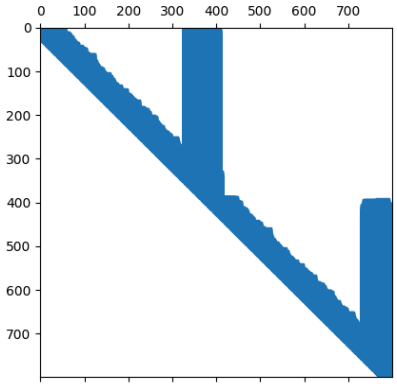

Figure 14: R-Matrix with LU-COLAMD Method

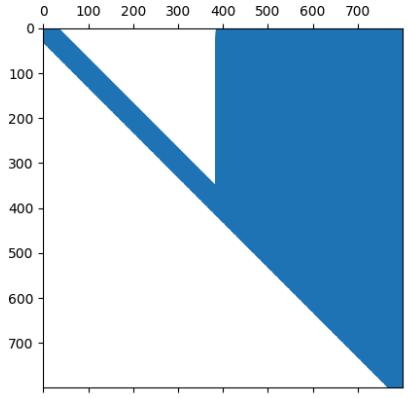

Figure 15: R-Matrix with QR Method

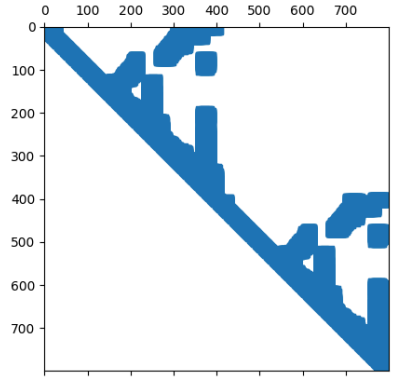

Figure 16: R-Matrix with QR-COLAMD Method

Method	Efficiency (s)
P_Inv	0.17214765548706054
LU	0.017378807067871094
LU-COLAMD	0.006091403961181641
QR	0.27143349647521975
QR-COLAMD	0.01953153610229492

Table 2. Efficiencies for 2d_linear_loop.npz averaged over 5 iterations.

From the linear loop test run, we observe that the LU methods remained significantly faster than the QR methods, even more so than in the non-loop test. Interestingly, the pseudo-inverse method also performed faster than QR in this looped scenario. This difference in efficiency can be attributed to the increased matrix density introduced by the loop closure constraints, which impacts the computational load, particularly for QR methods.

In both QR and LU methods without COLAMD reordering, the R-matrix graphs show similar sparsity patterns, indicating that the default ordering did not significantly alter the matrix structure for either method. However, when applying COLAMD reordering, both methods benefit from improved sparsity, as shown in the reordered R-matrix graphs. The increased sparsity from COLAMD reduces the number of non-zero elements, which helps to significantly reduce computational time and improve efficiency for both LU and QR methods.

In summary, the loop closure constraints in this dataset increase the matrix density and computation required, especially for QR methods. The COLAMD reordering proves effective in enhancing sparsity for both LU and QR, resulting in notable efficiency gains.

2 2D Nonlinear SLAM

2.1 Measurement Function

1. Code completed in “nonlinear.py’.

2. Jacobian derived on next page

2.1.2

Derive $H_\ell(r^+, \ell^n) : R^4 \to R^{2\times 4}$

$$h_\ell(r^+, \ell^n) = \begin{bmatrix} \operatorname{atan2}(\ell_y^n - r_y^+, \ell_x^n - r_x^+) \\ \sqrt{(\ell_x^n - r_x^+)^2 + (\ell_y^n - r_y^+)^2} \end{bmatrix} = \begin{bmatrix} \theta \\ d \end{bmatrix}$$

$$H_\ell = \frac{\partial h_\ell}{\partial X} = \begin{bmatrix} \frac{\partial h_\ell}{\partial r_x^+} & \frac{\partial h_\ell}{\partial r_y^+} & \frac{\partial h_\ell}{\partial \ell_x^n} & \frac{\partial h_\ell}{\partial \ell_y^n} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\ell_y^n - r_y^+}{\alpha} & \frac{-(\ell_x^n - r_x^+)}{\alpha} & \frac{-(\ell_y^n - r_y^+)}{\alpha} & \frac{\ell_x^n - r_x^n}{\alpha} \\ \frac{-(\ell_y^n - r_x^+)}{\beta} & \frac{-(\ell_y^n - r_y^+)}{\beta} & \frac{\ell_x^n - r_x^n}{\beta} & \frac{\ell_y^n - r_y^+}{\beta} \end{bmatrix}$$

; where $\alpha = (\ell_x^n - r_x^+)^2 + (\ell_y^n - r_y^+)^2$

) & $\beta = \sqrt{(\ell_x^n - r_x^+)^2 + (\ell_y^n - r_y^+)^2}$

2.2 Build a linear system

Code completed in “nonlinear.py”.

2.3 Solver

The visualization for the trajectory and landmarks both before and after optimization are below. The LU method was also used for the plots below.

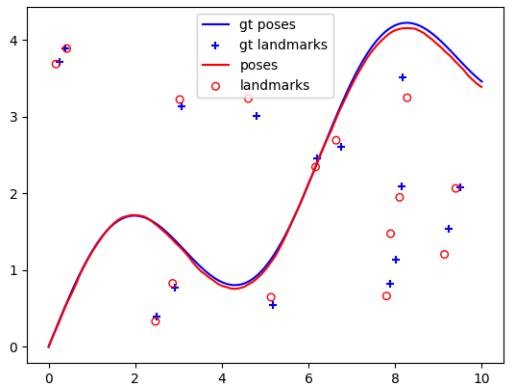

Figure 17: Before Optimization

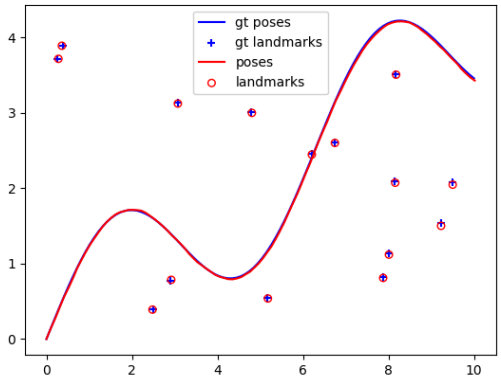

Figure 18: After Optimization

After optimization, the results for nonlinear SLAM show much greater accuracy, especially in the positions of the landmarks, which align more closely with the ground truth. This improvement highlights the iterative optimization process required in nonlinear SLAM, where the solver refines its estimates to handle the nonlinearity in measurements, such as angles and distances to landmarks. Unlike linear SLAM, which is a single-step process, nonlinear SLAM requires multiple iterations to converge, making it more computationally demanding and sensitive to initial estimates. However,

this added complexity enables nonlinear SLAM to achieve more accurate solutions in scenarios involving complex measurement models.