16-833: Robot Localization and Mapping
Homework 3: Written Report

Shahram Najam Syed
Andrew ID: snsyed

26th October, 2024

Question 1.1: Measurement function

In 2D linear SLAM, we aim to estimate the robot's trajectory and the positions of landmarks using odometry and landmark measurements. The state vector x includes robot poses and landmark positions, both represented by their x and y coordinates.

There are two types of measurements:

- **Odometry Measurements**: Provide the relative displacement between consecutive robot poses.
- **Landmark Measurements**: Provide the relative displacement from the robot to observed landmarks.

We will define the measurement functions and their Jacobians for both types of measurements.

Odometry Measurement Function

Measurement Function h_o

Given the robot poses at time t and $t+1$:

$$r_t = \begin{bmatrix} r_t^x \\ r_t^y \end{bmatrix}, \quad r_{t+1} = \begin{bmatrix} r_{t+1}^x \\ r_{t+1}^y \end{bmatrix}$$

The odometry measurement function $h_o(r_t, r_{t+1})$ models the observed change in position between these two time steps:

$$h_o(r_t, r_{t+1}) = r_{t+1} - r_t = \begin{bmatrix} r_{t+1}^x - r_t^x \\ r_{t+1}^y - r_t^y \end{bmatrix}$$

This function maps from $\mathbb{R}^4$ to $\mathbb{R}^2$, where the input is the concatenated vector of r_t and r_{t+1}, and the output is the observed displacement.

Jacobian H_o

The Jacobian matrix $H_o(r_t, r_{t+1})$ is the partial derivative of the measurement function h_o with respect to the state variables $[r_t^x, r_t^y, r_{t+1}^x, r_{t+1}^y]^T$:

$$H_o(r_t, r_{t+1}) = \frac{\partial h_o}{\partial [r_t^x, r_t^y, r_{t+1}^x, r_{t+1}^y]} = \begin{bmatrix} \frac{\partial h_o^x}{\partial r_t^x} & \frac{\partial h_o^x}{\partial r_t^y} & \frac{\partial h_o^x}{\partial r_{t+1}^x} & \frac{\partial h_o^x}{\partial r_{t+1}^y} \\ \frac{\partial h_o^y}{\partial r_t^x} & \frac{\partial h_o^y}{\partial r_t^y} & \frac{\partial h_o^y}{\partial r_{t+1}^x} & \frac{\partial h_o^y}{\partial r_{t+1}^y} \end{bmatrix}$$

Calculating Partial Derivatives:

For the x-component:

$$h_o^x = r_{t+1}^x - r_t^x$$

$$\frac{\partial h_o^x}{\partial r_t^x} = -1, \quad \frac{\partial h_o^x}{\partial r_{t+1}^x} = 1$$

$$\frac{\partial h_o^x}{\partial r_t^y} = 0, \quad \frac{\partial h_o^x}{\partial r_{t+1}^y} = 0$$

For the y-component:

$$h_o^y = r_{t+1}^y - r_t^y$$

$$\frac{\partial h_o^y}{\partial r_t^y} = -1, \quad \frac{\partial h_o^y}{\partial r_{t+1}^y} = 1$$

$$\frac{\partial h_o^y}{\partial r_t^x} = 0, \quad \frac{\partial h_o^y}{\partial r_{t+1}^x} = 0$$

Jacobian Matrix:

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

This Jacobian maps from the state vector to the measurement space, indicating how changes in the robot's poses affect the odometry measurements.

Landmark Measurement Function

Measurement Function h_l

Given the robot pose at time t and the k-th landmark position:

$$r_t = \begin{bmatrix} r_t^x \\ r_t^y \end{bmatrix}, \quad l_k = \begin{bmatrix} l_k^x \\ l_k^y \end{bmatrix}$$

The landmark measurement function $h_l(r_t, l_k)$ models the observed relative position of the landmark with respect to the robot:

$$h_l(r_t, l_k) = l_k - r_t = \begin{bmatrix} l_k^x - r_t^x \\ l_k^y - r_t^y \end{bmatrix}$$

This function maps from $\mathbb{R}^4$ to $\mathbb{R}^2$, where the input is the concatenated vector of r_t and l_k, and the output is the observed displacement to the landmark.

Jacobian H_l

The Jacobian matrix $H_l(r_t, l_k)$ is the partial derivative of the measurement function h_l with respect to the state variables $[r_t^x, r_t^y, l_k^x, l_k^y]^T$:

$$H_l(r_t, l_k) = \frac{\partial h_l}{\partial [r_t^x, r_t^y, l_k^x, l_k^y]} = \begin{bmatrix} \frac{\partial h_l^x}{\partial r_t^x} & \frac{\partial h_l^x}{\partial r_t^y} & \frac{\partial h_l^x}{\partial l_k^x} & \frac{\partial h_l^x}{\partial l_k^y} \\ \frac{\partial h_l^y}{\partial r_t^x} & \frac{\partial h_l^y}{\partial r_t^y} & \frac{\partial h_l^y}{\partial l_k^x} & \frac{\partial h_l^y}{\partial l_k^y} \end{bmatrix}$$

Calculating Partial Derivatives:
For the x-component:

$$h_l^x = l_k^x - r_t^x$$

$$\frac{\partial h_l^x}{\partial r_t^x} = -1, \quad \frac{\partial h_l^x}{\partial l_k^x} = 1$$

$$\frac{\partial h_l^x}{\partial r_t^y} = 0, \quad \frac{\partial h_l^x}{\partial l_k^y} = 0$$

For the y-component:

$$h_l^y = l_k^y - r_t^y$$

$$\frac{\partial h_l^y}{\partial r_t^y} = -1, \quad \frac{\partial h_l^y}{\partial l_k^y} = 1$$

$$\frac{\partial h_l^y}{\partial r_t^x} = 0, \quad \frac{\partial h_l^y}{\partial l_k^x} = 0$$

Jacobian Matrix:

$$H_l(r_t, l_k) = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

This Jacobian indicates how changes in the robot's pose and the landmark's position affect the landmark measurements.

Question 1.2: Build a linear system

Implemented as code linear.py (create_linear_system function), and submitted on gradescope.

Question 1.3: Solvers

Implemented as code solvers.py , and submitted on gradescope.

Question 1.4: Exploit sparsity

1. lu_cholmod:

Implemented as code solvers.py , and submitted on gradescope.

2. [Bonus] lu_direct:

Implemented as code solvers.py , and submitted on gradescope.

3. qr_cholmod:

Implemented as code solvers.py , and submitted on gradescope.

4. 2d_linear.npz:

Run Time Analysis

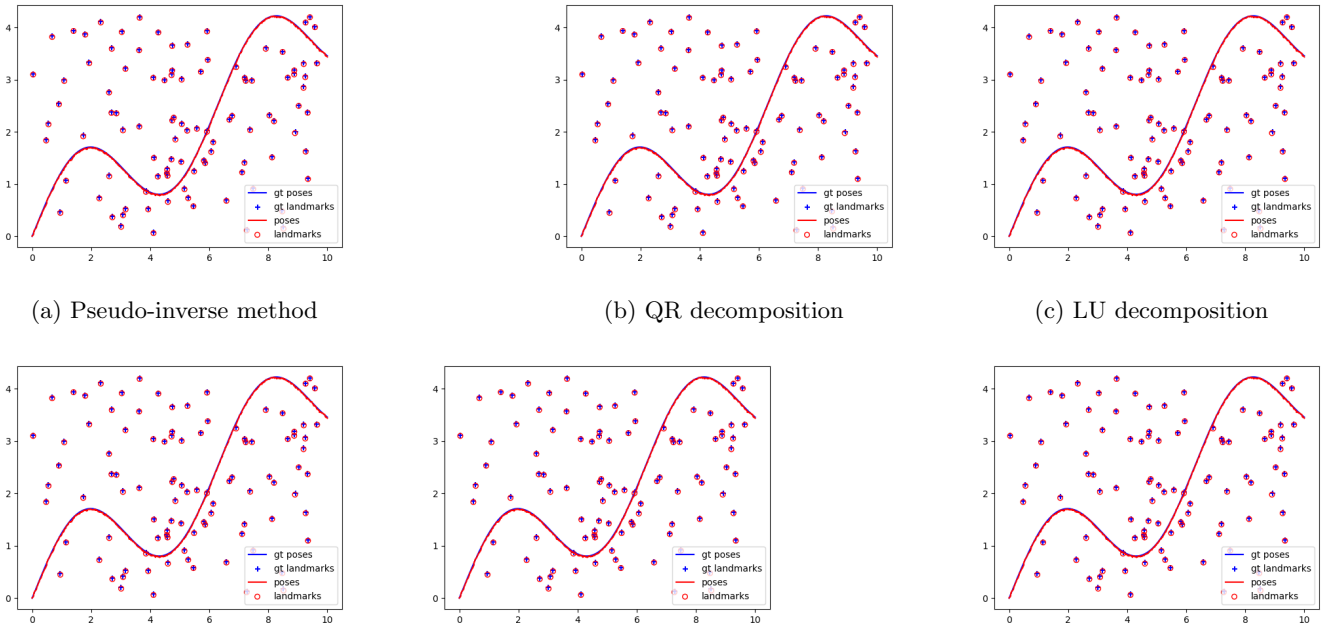
The efficiency of each solver, as measured by runtime, is presented in Table 1 for each linear method applied to the 2D_linear.npz dataset.

Table 1: Run Times for Different Linear Methods on 2D_linear.npz

Method	Run Time (s)	Comments
Default	0.0245	Baseline method
Pinv	1.6563	Uses pseudo-inverse
QR	0.3904	QR decomposition
LU	0.0171	LU decomposition
QR_colamd	0.3424	QR with COLAMD ordering
LU_colamd	0.0259	LU with COLAMD ordering
LU_direct	1.7928	Custom direct LU solver

Trajectories and landmarks

The solvers assessed include LU decomposition, QR decomposition, their COLAMD reordered variants, the pseudo-inverse method, and a custom implementation of LU (LU direct). Each method's trajectory and landmark ground truth and estimation on the 2D map plot is shown below. Despite differences in computation times, all methods achieve comparable accuracy in estimating poses and landmarks.



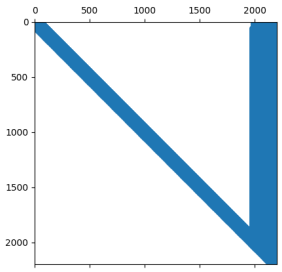
(a) Pseudo-inverse method (b) QR decomposition (c) LU decomposition

(d) QR COLAMD (e) LU COLAMD (f) Custom LU direct method

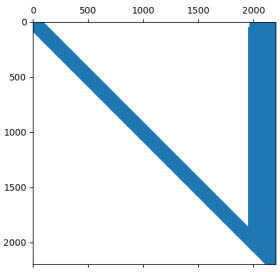
Figure 1: Trajectory and landmarks estimated using various linear solvers.

Matrix Sparsity Visualization

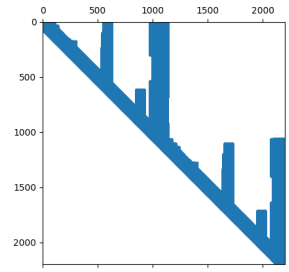
The solvers assessed include LU decomposition, QR decomposition, their COLAMD reordered variants, and a custom implementation of LU (LU direct). The sparsity patterns of the matrices for each solver are visualized to highlight the differences in computational complexity.



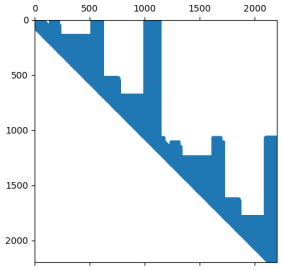
(a) QR decomposition



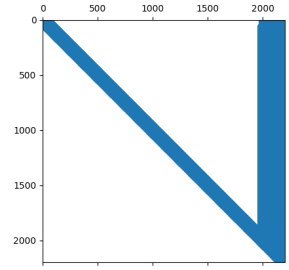
(b) LU decomposition



(c) QR COLAMD



(d) LU COLAMD



(e) Custom LU direct method

Figure 2: Sparsity pattern of the matrix

Observations and Comparitive Analysis

From Table 1, the results clearly prioritize LU decomposition in terms of computational speed, achieving the lowest runtime of 0.0171 seconds, as demonstrated by the operational efficiency in decomposing the matrix into lower and upper triangular forms, allowing for more direct and less computationally intensive solutions, see Figure 4b.

In comparison, the LU COLAMD variant, operating at 0.0259 seconds, demonstrates only a marginal increase in runtime efficiency. This minimal gain suggests that the computational overhead introduced by the COLAMD reordering does not effectively offset the benefits derived from improved sparsity in the LU context, as illustrated in Figure 4d.

The QR COLAMD variant, which processes at 0.3424 seconds compared to QR's 0.3904 seconds, underscores the potential of COLAMD reordering to improve QR decomposition's runtime by optimizing the matrix's sparsity structure, as seen in Figure 4c. Despite this improvement, both QR configurations remain significantly slower than LU methods, largely due to the computational burden imposed by QR's orthogonalization process, which is inherently more complex and resource-intensive than triangular decomposition, visualized in Figure 4a.

The most computationally intensive methods, the pseudo-inverse and the custom LU direct solver, with runtimes of 1.6563 seconds and 1.7928 seconds respectively, emphasize the inefficiencies associated with non-optimized solvers in handling large, sparse matrices. The pseudo-inverse method, involving matrix inversion, is particularly disadvantageous in sparse system contexts due to its high computational complexity. Meanwhile, the custom LU direct solver suffers from the absence of optimized library functions, which impacts algorithmic efficiency on runtime performance, as shown in Figure 4e.

5. 2d_linear_loop.npz:

Run Time Analysis

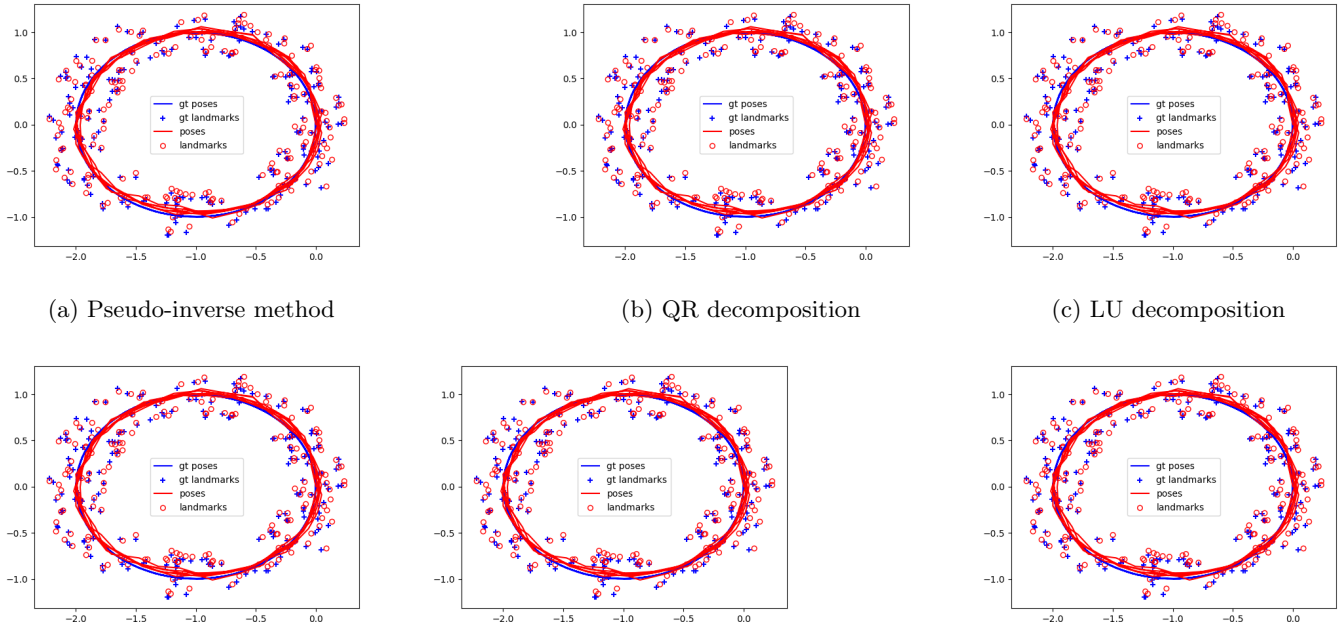
The efficiency of each solver, as measured by runtime, is presented in Table 2 for each linear method applied to the 2D_linear.npz dataset.

Table 2: Run Times for Different Linear Methods on 2D_linear.npz

Method	Run Time (s)	Comments
Default	0.0028	Baseline method
Pinv	0.1089	Uses pseudo-inverse
QR	0.2275	QR decomposition
LU	0.0129	LU decomposition
QR_colamd	0.0137	QR with COLAMD ordering
LU_colamd	0.0032	LU with COLAMD ordering
LU_direct	0.1121	Custom direct LU solver

Trajectories and landmarks

The solvers assessed include LU decomposition, QR decomposition, their COLAMD reordered variants, the pseudo-inverse method, and a custom implementation of LU (LU direct). Each method's trajectory and landmark ground truth and estimation on the 2D map plot is shown below. Despite differences in computation times, all methods achieve comparable accuracy in estimating poses and landmarks.



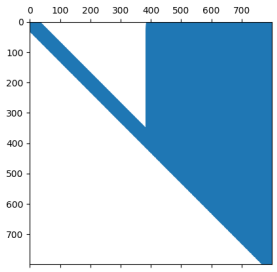
(a) Pseudo-inverse method (b) QR decomposition (c) LU decomposition

(d) QR COLAMD (e) LU COLAMD (f) Custom LU direct method

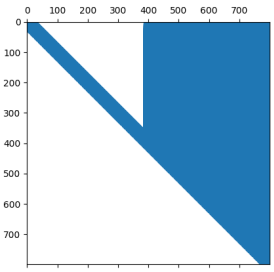
Figure 3: Trajectory and landmarks estimated using various linear solvers.

Matrix Sparsity Visualization

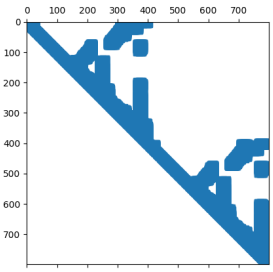
The solvers assessed include LU decomposition, QR decomposition, their COLAMD reordered variants, and a custom implementation of LU (LU direct). The sparsity patterns of the matrices for each solver are visualized to highlight the differences in computational complexity.



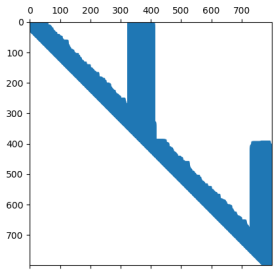
(a) QR decomposition



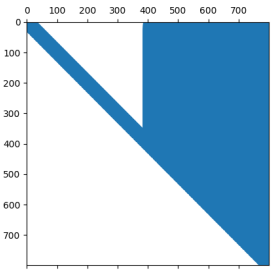
(b) LU decomposition



(c) QR COLAMD



(d) LU COLAMD



(e) Custom LU direct method

Figure 4: Sparsity pattern of the matrix

Observations and Comparative Analysis for 2d_linear_loop.npz

The comparative efficiency of solvers on the '2d_linear_loop.npz' dataset is reflected in Table 2, showcasing notable variations in performance relative to the '2d_linear.npz' dataset.

The performance trends observed in '2d_linear.npz' generally persist here, with LU-based solvers outpacing QR-based solvers in terms of computational speed. Notably, the reordered variants, LU COLAMD and QR COLAMD, demonstrate improved performance over their standard counterparts, as seen in the runtime reductions noted in Table 2. This suggests that the reordering process, which aims to optimize matrix sparsity, is particularly effective in the loop scenario where the matrix tends to be denser due to repetitive landmarks and odometry measurements within a looped trajectory.
The runtime for LU COLAMD at 0.0032 seconds marks it as the fastest, outperforming even the standard LU decomposition, which is logged at 0.0129 seconds. This inversion in expected performance order, where LU COLAMD surpasses standard LU, underscores the significant impact of the COLAMD reordering in this context. Such an impact is due to the denser nature of A in loop scenarios, where COLAMD effectively reduces arithmetic operations by optimizing the matrix structure for sparser solutions.
Similarly, QR COLAMD shows an improved performance over standard QR, processing at 0.0137 seconds compared to QR's 0.2275 seconds. This substantial improvement highlights the effectiveness of reordering in managing the dense matrix more efficiently, reducing computational overhead and enhancing solver speed.
Visualization of matrix sparsity and trajectory estimations, as illustrated in Figures 3 and 4, further corroborates these observations. The sparsity patterns of reordered matrices, particularly in Figures 4d and 4c, are significantly sparser compared to their non-reordered counterparts, supporting the observed speedups in computational performance.
Despite the varying speeds, all methods maintain comparable accuracy in estimating poses and landmarks, indicating that the increased efficiency does not compromise the solver's effectiveness in this more challenging loop scenario. This equivalence in output quality, despite differing computational times, emphasizes the importance of solver selection based on specific problem characteristics and matrix properties.

Question 2.1: Measurement function

1. odometry_estimation and bearing_range_estimation implementation

This section was implemented in non_linear.py and is submitted on gradescope in the code section.

2. Derivation of nonlinear landmark and compute_meas_obs_jacobian implementation

This below derivation was implemented in non_linear.py and is submitted on gradescope in the code section.

In this problem, we extend 2D linear SLAM to a nonlinear version by introducing a nonlinear measurement function that returns the bearing angle θ and range d from the robot to a landmark in the robot's body frame. The robot is assumed to always face the x-direction of the global frame.

Measurement Function

The nonlinear landmark measurement function is defined as:

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

Where:

$$r_t = \begin{bmatrix} r_x^t \\ r_y^t \end{bmatrix}, \quad l_k = \begin{bmatrix} l_x^k \\ l_y^k \end{bmatrix}$$

- r_t: Robot position at time t. - l_k: Position of the k-th landmark. - θ: Bearing angle from the robot to the landmark. - d: Euclidean distance from the robot to the landmark.

Jacobian of the Measurement Function $H_l(r_t, l_k)$

We need to derive the Jacobian $H_l(r_t, l_k)$ of the measurement function h_l with respect to the state variables $[r_x^t, r_y^t, l_x^k, l_y^k]^T$:

$$H_l(r_t, l_k) = \frac{\partial h_l}{\partial [r_x^t, r_y^t, l_x^k, l_y^k]}$$

We use a simplified expression as follows:

$$dx = l_x^k - r_x^t, \quad dy = l_y^k - r_y^t$$

Partial Derivatives of θ

Since $\theta = \text{atan2}(l_y^k - r_y^t, l_x^k - r_x^t)$

1. $\frac{\partial \theta}{\partial r_x^t}$**:**

$$\begin{aligned} \frac{\partial \theta}{\partial r_x^t} &= \frac{\partial \theta}{\partial dx} \cdot \frac{\partial dx}{\partial r_x^t} + \frac{\partial \theta}{\partial dy} \cdot \frac{\partial dy}{\partial r_x^t} \\ &= \left(-\frac{dy}{dx^2 + dy^2} \right) \cdot (-1) + \left(\frac{dx}{dx^2 + dy^2} \right) \cdot (0) \\ &= \frac{dy}{dx^2 + dy^2} \\ &= \frac{l_y^k - r_y^t}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} \end{aligned}$$

2. $\frac{\partial \theta}{\partial r_y^t}$**:**

$$\begin{aligned} \frac{\partial \theta}{\partial r_y^t} &= \frac{\partial \theta}{\partial dx} \cdot \frac{\partial dx}{\partial r_y^t} + \frac{\partial \theta}{\partial dy} \cdot \frac{\partial dy}{\partial r_y^t} \\ &= \left(-\frac{dy}{dx^2 + dy^2} \right) \cdot (0) + \left(\frac{dx}{dx^2 + dy^2} \right) \cdot (-1) \\ &= -\frac{dx}{dx^2 + dy^2} \\ &= -\frac{l_x^k - r_x^t}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} \end{aligned}$$

3. $\frac{\partial \theta}{\partial l_x^k}$**:**

$$\begin{aligned}
\frac{\partial \theta}{\partial l_x^k} &= \frac{\partial \theta}{\partial dx} \cdot \frac{\partial dx}{\partial l_x^k} + \frac{\partial \theta}{\partial dy} \cdot \frac{\partial dy}{\partial l_x^k} \\
&= \left(-\frac{dy}{dx^2 + dy^2} \right) \cdot (1) + \left(\frac{dx}{dx^2 + dy^2} \right) \cdot (0) \\
&= -\frac{dy}{dx^2 + dy^2} \\
&= -\frac{l_y^k - r_y^t}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}
\end{aligned}$$

4. $\frac{\partial \theta}{\partial l_y^k}$**:**

$$\begin{aligned}
\frac{\partial \theta}{\partial l_y^k} &= \frac{\partial \theta}{\partial dx} \cdot \frac{\partial dx}{\partial l_y^k} + \frac{\partial \theta}{\partial dy} \cdot \frac{\partial dy}{\partial l_y^k} \\
&= \left(-\frac{dy}{dx^2 + dy^2} \right) \cdot (0) + \left(\frac{dx}{dx^2 + dy^2} \right) \cdot (1) \\
&= \frac{dx}{dx^2 + dy^2} \\
&= \frac{l_x^k - r_x^t}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}
\end{aligned}$$

Partial Derivatives of d

Since $d = \sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}$

1. $\frac{\partial d}{\partial r_x^t}$**:**

$$\begin{aligned}
\frac{\partial d}{\partial r_x^t} &= \frac{\partial d}{\partial dx} \cdot \frac{\partial dx}{\partial r_x^t} + \frac{\partial d}{\partial dy} \cdot \frac{\partial dy}{\partial r_x^t} \\
&= \frac{1}{2\sqrt{dx^2 + dy^2}} \cdot 2dx \cdot (-1) + \frac{1}{2\sqrt{dx^2 + dy^2}} \cdot 2dy \cdot (0) \\
&= -\frac{dx}{\sqrt{dx^2 + dy^2}} \\
&= -\frac{l_x^k - r_x^t}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}}
\end{aligned}$$

2. $\frac{\partial d}{\partial r_y^t}$**:**

$$\begin{aligned}
\frac{\partial d}{\partial r_y^t} &= \frac{\partial d}{\partial dx} \cdot \frac{\partial dx}{\partial r_y^t} + \frac{\partial d}{\partial dy} \cdot \frac{\partial dy}{\partial r_y^t} \\
&= \frac{1}{2\sqrt{dx^2 + dy^2}} \cdot 2dx \cdot (0) + \frac{1}{2\sqrt{dx^2 + dy^2}} \cdot 2dy \cdot (-1) \\
&= -\frac{dy}{\sqrt{dx^2 + dy^2}} \\
&= -\frac{l_y^k - r_y^t}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}}
\end{aligned}$$

3. $\frac{\partial d}{\partial l_x^k}$**:**

$$\begin{aligned}\frac{\partial d}{\partial l_x^k} &= \frac{\partial d}{\partial dx} \cdot \frac{\partial dx}{\partial l_x^k} + \frac{\partial d}{\partial dy} \cdot \frac{\partial dy}{\partial l_x^k} \\ &= \frac{1}{2\sqrt{dx^2+dy^2}} \cdot 2dx \cdot (1) + \frac{1}{2\sqrt{dx^2+dy^2}} \cdot 2dy \cdot (0) \\ &= \frac{dx}{\sqrt{dx^2+dy^2}} \\ &= \frac{l_x^k - r_x^t}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}}\end{aligned}$$

4. $\frac{\partial d}{\partial l_y^k}$**:**

$$\begin{aligned}\frac{\partial d}{\partial l_y^k} &= \frac{\partial d}{\partial dx} \cdot \frac{\partial dx}{\partial l_y^k} + \frac{\partial d}{\partial dy} \cdot \frac{\partial dy}{\partial l_y^k} \\ &= \frac{1}{2\sqrt{dx^2+dy^2}} \cdot 2dx \cdot (0) + \frac{1}{2\sqrt{dx^2+dy^2}} \cdot 2dy \cdot (1) \\ &= \frac{dy}{\sqrt{dx^2+dy^2}} \\ &= \frac{l_y^k - r_y^t}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}}\end{aligned}$$

Final Jacobian Matrix

Assembling these partial derivatives into the Jacobian matrix:

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

Simplifying we get:

$$H_l(r_t, l_k) = \begin{bmatrix} \frac{dy}{dx^2+dy^2} & -\frac{dx}{dx^2+dy^2} & -\frac{dy}{dx^2+dy^2} & \frac{dx}{dx^2+dy^2} \\ -\frac{dx}{\sqrt{dx^2+dy^2}} & -\frac{dy}{\sqrt{dx^2+dy^2}} & \frac{dx}{\sqrt{dx^2+dy^2}} & \frac{dy}{\sqrt{dx^2+dy^2}} \end{bmatrix}$$

Where,

$$dx = l_x^k - r_x^t, \quad dy = l_y^k - r_y^t$$

Further simplification for the later coding section we can get:

$$H_l(r_t, l_k) = \begin{bmatrix} \frac{dy}{q} & -\frac{dx}{q} & -\frac{dy}{q} & \frac{dx}{q} \\ -\frac{dx}{d} & -\frac{dy}{d} & \frac{dx}{d} & \frac{dy}{d} \end{bmatrix}$$

Where,

$$q = dx^2 + dy^2, \quad d = \sqrt{q}$$

Question 2.2: Build a linear system

This section was implemented in non_linear.py and is submitted on gradescope in the code section.

Question 2.3: Solver

Nonlinear Measurement Function The nonlinear SLAM extends the linear measurement model to include angular components and nonlinear distance measurements. The nonlinear measurement function is defined as follows, providing the bearing angle θ and the range d from the robot to the landmark, which together describe the vector from the robot to the landmark in the robot's body frame:

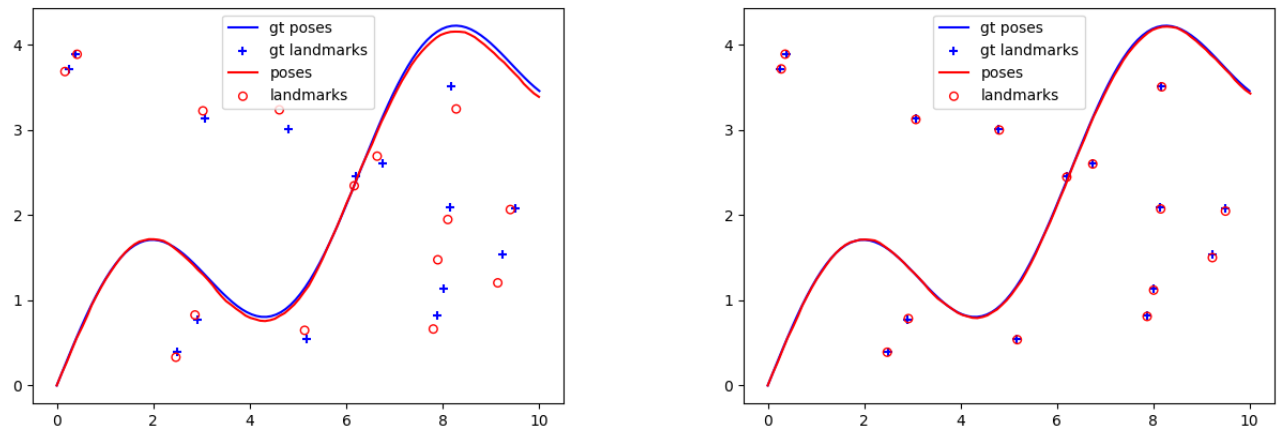
$$h_l(r_t, l_k) = \begin{bmatrix} \text{atan2}(l_y^k - r_y^t, l_x^k - r_x^t) \\ \sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} \end{bmatrix} = \begin{bmatrix} \theta \\ d \end{bmatrix} \tag{1}$$

Visualization and Optimization Process For the nonlinear SLAM implementation, the LU COLAMD solver was chosen due to its proven effectiveness in linear SLAM scenarios. Figures 5a and 5b illustrate the trajectory and landmarks before and after optimization, showcasing significant improvements in the accuracy of landmark positioning and trajectory estimation. Although LU COLAMD was selected for this demonstration, it is noted that the results from other solvers yielded visually similar outcomes.

Comparative Analysis of Linear and Nonlinear SLAM The optimization processes for linear and nonlinear SLAM differ significantly in both approach and computational complexity:

- **Solution Methodology:** In linear SLAM, solutions are directly derived using normal equations due to the linear nature of the measurement functions, allowing for a straightforward batch process solution that leverages the sparsity within the system matrix. Conversely, nonlinear SLAM requires an iterative solution process to address the nonlinearities introduced by angular components and nonlinear distance measurements, as exemplified in Equation (2).
- **Optimization Steps:** Linear SLAM generally involves a single optimization step using direct matrix operations for a one-shot solution. In contrast, nonlinear SLAM necessitates multiple iterations to refine the solution, each involving the linearization of the measurement model around the current estimate, solving for incremental changes, and updating the state vector. This iterative process is essential to manage the inherent non-convexity of the problem and is, thus, computationally more demanding.
- **Complexity and Jacobian Computation:** Unlike linear SLAM where the Jacobian remains constant, nonlinear SLAM necessitates the recalibration of the Jacobian matrix at each iteration relative to the new linearization point. This recalibration is crucial as it significantly influences the convergence rate and accuracy of the solution, especially in environments where the robot's orientation and landmark positions introduce substantial nonlinearities.

Execution and Performance As depicted in the figures, nonlinear optimization significantly enhances the SLAM solution's fidelity by iteratively reducing the residual errors between predicted and observed measurements. This iterative refinement starkly contrasts with the linear approach, where the assumed linearity restricts accuracy improvements to the initial model's capabilities.



(a) Before optimization

(b) After optimization

Figure 5: Comparison of trajectory and landmarks before and after optimization using LU COLAMD.

The improvements demonstrated in nonlinear SLAM underscore the importance of robust initial estimates and the efficacy of iterative optimization, particularly in scenarios with complex interactions and measurements that significantly deviate from linear assumptions.

Q1.1

Ans:- Robot pose @t:-

$$x_t = \begin{bmatrix} x_t^x \\ x_t^y \end{bmatrix}$$

Robot pose @ t+1

$$x_{t+1} = \begin{bmatrix} x_{t+1}^x \\ x_{t+1}^y \end{bmatrix}$$

$$h_o(x_t, x_{t+1}) = x_{t+1} - x_t = \begin{bmatrix} x_{t+1}^x & -x_t^x \\ x_{t+1}^y & -x_t^y \end{bmatrix}$$

maps $\mathbb{R}^4$ to $\mathbb{R}^2$

Odometry function

Observed displacement

The Jacobian becomes:-

$$H_o(x_t, x_{t+1}) = \frac{\partial h_o}{\partial \begin{bmatrix} x_t^x \\ x_t^y \\ x_{t+1}^x \\ x_{t+1}^y \end{bmatrix}} = \begin{bmatrix} \frac{\partial h_o^x}{\partial x_t^x} & \frac{\partial h_o^x}{\partial x_t^y} & \frac{\partial h_o^x}{\partial x_{t+1}^x} & \frac{\partial h_o^x}{\partial x_{t+1}^y} \\ \frac{\partial h_o^y}{\partial x_t^x} & \frac{\partial h_o^y}{\partial x_t^y} & \frac{\partial h_o^y}{\partial x_{t+1}^x} & \frac{\partial h_o^y}{\partial x_{t+1}^y} \end{bmatrix} \quad \text{--- (1)}$$

So,

$$h_o^x = x_{t+1}^x - x_t^x$$

$$\frac{\partial h_o^x}{\partial x_t^x} = -1 \qquad \frac{\partial h_o^x}{\partial x_t^y} = 0$$

$$\frac{\partial h_o^x}{\partial x_{t+1}^x} = 1 \qquad \frac{\partial h_o^x}{\partial x_{t+1}^y} = 0.$$

now,

$$h_y^\circ = x_{t+1}^y - x_t^y$$

$$\frac{\partial h_o^y}{\partial x_t^x} = -1 \qquad \frac{\partial h_o^y}{\partial x_t^{yc}} = 0$$

$$\frac{\partial h_o^y}{\partial x_{t+1}^y} = 1 \qquad \frac{\partial h_o^y}{\partial x_{t+1}^{xc}} = 0$$

So,

$$\Rightarrow \boxed{H_o(x_t, x_{t+1}) = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}}$$

now, Robot pose @ t

$$x^t = \begin{bmatrix} x_x^t \\ x_y^t \end{bmatrix}$$

K^{th} landmark position

$$l_k = \begin{bmatrix} l_k^{xc} \\ l_k^y \end{bmatrix}$$

$$h_l(x_t, l_k) = l_k - x_t = \begin{bmatrix} l_k^{xc} - x_t^x \\ l_k^y - x_t^y \end{bmatrix}$$

$h_l(x_t, l_k)$ — landmark measurement

sol:

$$H_i(x_t, l_k) = \frac{\partial h_i}{\partial \begin{bmatrix} x_t^x \\ x_t^y \\ l_k^x \\ l_k^y \end{bmatrix}} = \begin{bmatrix} \frac{\partial h_i^x}{\partial x_t^x} & \frac{\partial h_i^x}{\partial x_t^y} & \frac{\partial h_i^x}{\partial l_k^x} & \frac{\partial h_i^x}{\partial l_k^y} \\ \frac{\partial h_i^y}{\partial x_t^x} & \frac{\partial h_i^y}{\partial x_t^y} & \frac{\partial h_i^y}{\partial l_k^x} & \frac{\partial h_i^y}{\partial l_k^y} \end{bmatrix} \quad \text{---(2)}$$

now:

$$h_i^x = l_k^x - x_t^x$$

$$\frac{\partial h_i^x}{\partial x_t^x} = -1 \qquad \frac{\partial h_i^x}{\partial x_t^y} = 0$$

$$\frac{\partial h_i^x}{\partial l_k^x} = 1 \qquad \frac{\partial h_i^x}{\partial l_k^y} = 0$$

& $h_i^y = l_k^y - x_t^y$

$$\frac{\partial h_i^y}{\partial x_t^y} = -1 \qquad \frac{\partial h_i^y}{\partial x_t^x} = 0$$

$$\frac{\partial h_i^y}{\partial l_k^y} = 1 \qquad \frac{\partial h_i^y}{\partial l_k^x} = 0$$

$$\textcircled{2} \Rightarrow H_i(x_i, L_k) = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

Q 2.1

Ans :- Since,

$$h_i(x_t, l_k) = \begin{bmatrix} \theta \\ d \end{bmatrix} = \begin{bmatrix} \text{atan2}(l_y^k - x_y^t, l_x^k - x_x^t) \\ \sqrt{(l_x^k - x_x^t)^2 + (l_y^k - x_y^t)^2} \end{bmatrix}$$

so,

$$H_i(x_t, l_k) = \frac{\partial h_i}{\partial [x_x^t, x_y^t, l_x^k, l_y^k]}$$

For simplification we can use:-

$$dx = l_x^k - x_x^t \; ; \; dy = l_y^k - x_y^t \quad \text{---} (2)$$

(1)

Since,

$$\theta = \text{atan2}(l_y^k - x_y^t, l_x^k - x_x^t)$$

$$\textcircled{1} \quad \frac{\partial \theta}{\partial x_x^t} = \frac{\partial \theta}{\partial dx} \frac{\partial dx}{\partial x_x^t} + \frac{\partial \theta}{\partial dy} \frac{\partial dy}{\partial x_x^t}$$

$$= \left(\frac{-dy}{dx^2 + dy^2}\right)(-1) + \left(\frac{dx}{dx^2 + dy^2}\right)(0)$$

$$= \frac{dy}{dx^2 + dy^2} \quad \text{---} (3)$$

plugging in (1) and (2) into (3)

$$\frac{\partial \theta}{\partial x_x^t} = \frac{l_y^k - x_y^t}{(l_x^k - x_x^t)^2 + (l_y^k - x_y^t)^2}$$

② $$\frac{\partial \theta}{\partial x_y^t} = \frac{\partial \theta}{\partial dx}\frac{\partial dx}{\partial x_y^t} + \frac{\partial \theta}{\partial dy}\frac{\partial dy}{\partial x_y^t}$$

$$= \left(\frac{-dy}{dx^2+dy^2}\right)(0) + \left(\frac{dx}{dx^2+dy^2}\right)(-1)$$

$$= \frac{-dx}{dx^2+dy^2} \quad \text{— ⑤}$$

plugging in ① and ② in ⑦

$$\frac{\partial \theta}{\partial x_y^t} = \frac{-(l_x^k - x_x^t)}{(l_x^k - x_x^t)^2 + (l_y^k - x_y^t)^2}$$

③ $$\frac{\partial \theta}{\partial l_x^k} = \frac{\partial \theta}{\partial dx}\frac{\partial dx}{\partial l_x^k} + \frac{\partial \theta}{\partial dy}\frac{\partial dy}{\partial l_x^k}$$

$$= \left(\frac{-dy}{dx^2+dy^2}\right)(1) + \left(\frac{dx}{dx^2+dy^2}\right)(0)$$

$$= \frac{-dy}{dx^2+dy^2} \quad \text{— ⑤}$$

plugging in ① and ② in ⑤

$$\frac{\partial \theta}{\partial l_x^k} = \frac{-(l_y^k - x_y^t)}{(l_x^k - x_x^t)^2 + (l_y^k - x_y^t)^2}$$

(4) $$\frac{\partial \theta}{\partial L_y^k} = \frac{\partial \theta}{\partial dx}\frac{\partial dx}{\partial L_y^k} + \frac{\partial \theta}{\partial dy}\frac{\partial dy}{\partial L_y^k}$$

$$= \left(\frac{-dy}{dx^2+dy^2}\right)(0) + \left(\frac{dx}{dx^2+dy^2}\right)(1)$$

$$= \frac{dx}{dx^2+dy^2} \quad \text{——} (6)$$

Plugging in (1) and (2) in (6)

$$\frac{\partial \theta}{\partial L_y^k} = \frac{L_x^k - x_x^t}{(L_x^k - x_x^t)^2 + (L_y^k - x_y^t)^2}$$

now,

$$d = \sqrt{(L_x^k - x_x^t)^2 + (L_y^k - x_y^t)^2}$$

(1) $$\frac{\partial d}{\partial x_{xc}^t} = \frac{\partial d}{\partial dx}\frac{\partial dx}{\partial x_{xc}^t} + \frac{\partial d}{\partial dy}\frac{\partial dy}{\partial x_{xc}^t}$$

$$= \frac{1}{2\sqrt{dx^2+dy^2}}(2dx)(-1) + \frac{1}{2\sqrt{dx^2+dy^2}}(2dy)(0)$$

$$= \frac{-dx}{\sqrt{dx^2+dy^2}} \quad \text{——} (7)$$

Plugging in (1) and (2) in (7)

$$\frac{\partial d}{\partial x_{xc}^t} = \frac{-(L_x^k - x_{xc}^t)}{\sqrt{(L_x^k - x_x^t)^2 + (L_y^k - x_y^t)^2}}$$

② $\frac{\partial d}{\partial r_y^t} = \frac{\partial d}{\partial dx}\frac{\partial dx}{\partial r_y^t} + \frac{\partial d}{\partial dy}\frac{\partial dy}{\partial r_y^t}$

$= \frac{1}{2\sqrt{dx^2+dy^2}}(2dx)(0) + \frac{1}{2\sqrt{dx^2+dy^2}}(2dy)(-1)$

$= \frac{-dy}{\sqrt{dx^2+dy^2}}$ — ⑧

plugging in ① and ② in ⑧

$\frac{\partial d}{\partial r_y^t} = \frac{-(l_y^k - r_y^t)}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}}$

③ $\frac{\partial d}{\partial l_x^k} = \frac{\partial d}{\partial dx}\frac{\partial dx}{\partial l_x^k} + \frac{\partial d}{\partial dy}\frac{\partial dy}{\partial l_x^k}$

$= \frac{1}{2\sqrt{dx^2+dy^2}}(2dx)(1) + \frac{1}{2\sqrt{dx^2+dy^2}}(2dy)(0)$

$= \frac{dx}{\sqrt{dx^2+dy^2}}$ — ⑨

plugging in ① and ② into ⑨

$\frac{\partial d}{\partial l_x^k} = \frac{l_x^k - r_x^t}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}}$

(4) $$\frac{\partial d}{\partial L_y^k} = \frac{\partial d}{\partial dx}\frac{\partial dx}{\partial L_y^k} + \frac{\partial d}{\partial dy}\frac{\partial dy}{\partial L_y^k}$$

$$= \frac{1}{2\sqrt{dx^2+dy^2}}(2dx)(0) + \frac{1}{2\sqrt{dx^2+dy^2}}(2dy)(1)$$

$$= \frac{dy}{\sqrt{dx^2+dy^2}} \quad \text{—— (10)}$$

plugging in (1) and (2) in (10)

$$\frac{\partial d}{\partial L_y^k} = \frac{L_y^k - r_y^t}{\sqrt{(L_x^k - r_x^t)^2 + (L_y^k - r_y^t)^2}}$$

$\Delta\theta$,

$$H_i(r_t, L_k) = \begin{bmatrix} \frac{\partial\theta}{\partial r_x^t} & \frac{\partial\theta}{\partial r_y^t} & \frac{\partial\theta}{\partial L_x^k} & \frac{\partial\theta}{\partial L_y^k} \\ \frac{\partial d}{\partial r_x^t} & \frac{\partial d}{\partial r_y^t} & \frac{\partial d}{\partial L_x^k} & \frac{\partial d}{\partial L_y^k} \end{bmatrix} \quad \text{—— (11)}$$

plug in the partials in (11)