16-833A Robot Localisation and Mapping, Spring 2023
Homework 3: Linear and Nonlinear SLAM Solvers

Sushanth Jayanth

Andrew ID: sushantj

April 2, 2023

THE ROBOTICS INSTITUTE

Contents

Introduction to Linear and Non-Linear SLAM problems

In this assignment, we make use of thinking in terms of a factor graph as explained in (1). A simple factor graph (Fig. 1) is shown below with the following components:

- Factors (edges)
  - Odometry measurements
  - Landmark measurements
- States (nodes)
  - Robot poses
  - Landmark poses

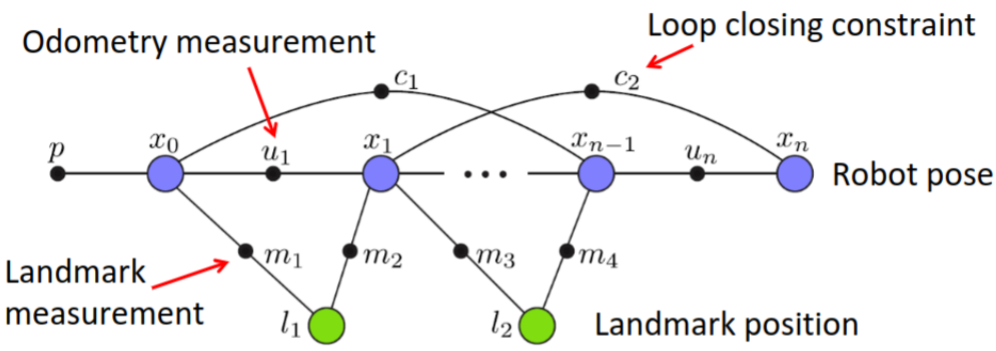


Figure 1: Factor Graph Representation

In this assignment, we're already given the data of the all the factors and states present in the factor graph. We will use this data to minimize the predicted values of each measurement (odometry measurements or landmark measurements) between every two connected states on the factor graph.

This minimization will be crafted in a least squares minimization form. The high level procedure to do so is shown below in Fig. 2

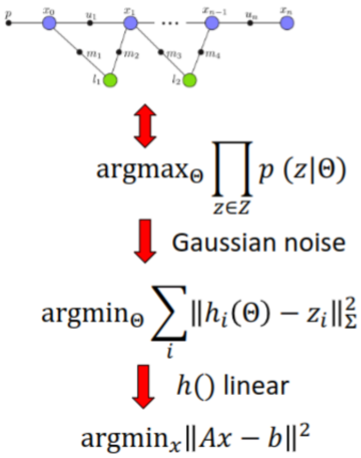


Figure 2: Arriving at Least Squares

Finally, the factor graph and least-squares equivalence is seen clearly below in Fig. 3

float

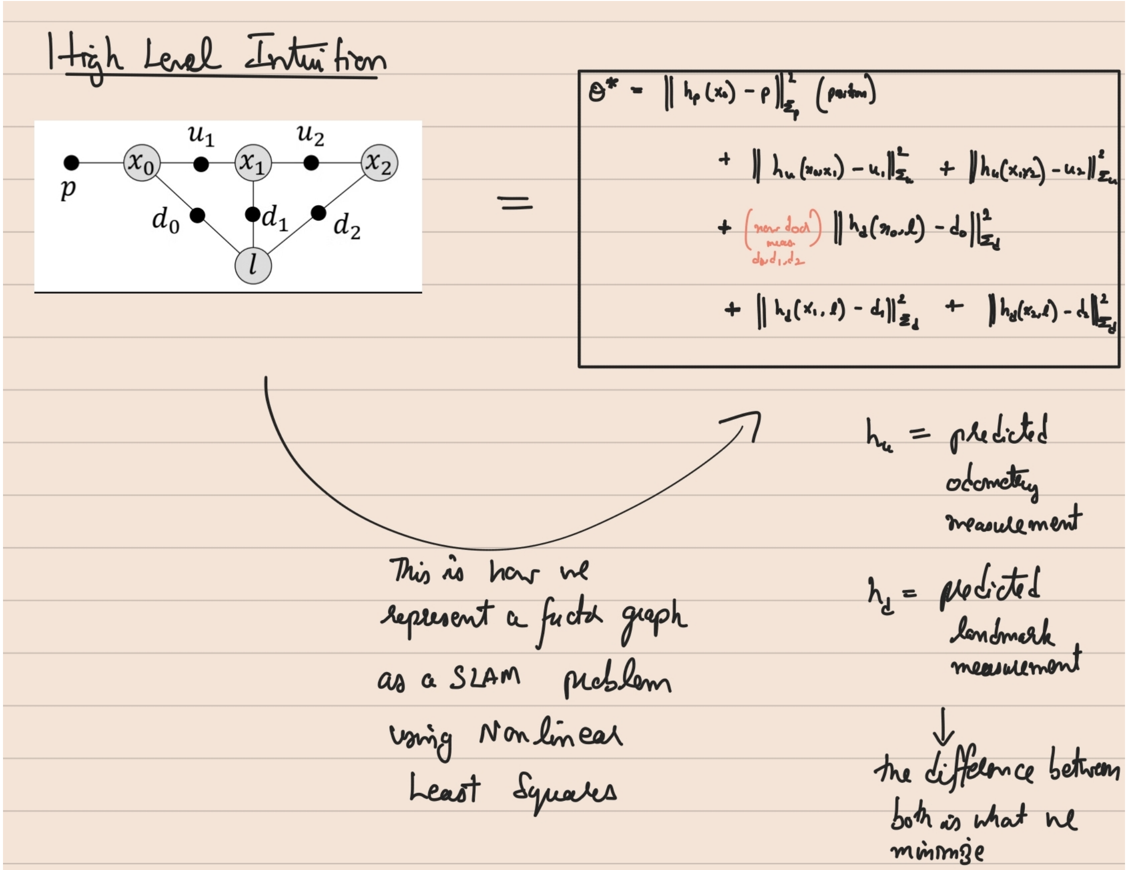


Figure 3: Factor graph ¡-¿ Least squares equivalence

1 2D Linear SLAM

1.1 Measurement Function

Odometry Measurement Function and Jacobian

Given the robot poses $\mathbf{r}^{t+1} = \left[r_x^{t+1}, r_y^{t+1}\right]^\top$ $\mathbf{r}^t = \left[r_x^t, r_y^t\right]^\top$ at times t and $t+1$, we find the measurement function to be:

$$h_o\left(\mathbf{r}^t, \mathbf{r}^{t+1}\right) = \begin{bmatrix} h_{o1} \\ h_{o2} \end{bmatrix} = \begin{bmatrix} r_x^{t+1} - r_x^t \\ r_y^{t+1} - r_y^t \end{bmatrix} \tag{1}$$

The Jacobian of the above measurement function can be written as:

$$H_o\left(\mathbf{r}^t, \mathbf{r}^{t+1}\right) = \begin{bmatrix} \frac{\partial h_{o1}}{\partial r_x^t} & \frac{\partial h_{o1}}{\partial r_y^t} & \frac{\partial h_{o1}}{\partial r_x^{t+1}} & \frac{\partial h_{o1}}{\partial r_y^{t+1}} \\ \frac{\partial h_{o2}}{\partial r_x^t} & \frac{\partial h_{o2}}{\partial r_y^t} & \frac{\partial h_{o2}}{\partial r_x^{t+1}} & \frac{\partial h_{o2}}{\partial r_y^{t+1}} \end{bmatrix} \tag{2}$$

Landmark Measurement Function and Jacobian

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

$$h_l\left(\mathbf{r}^t, \mathbf{l}^k\right) = \begin{bmatrix} h_{l1} \\ h_{l2} \end{bmatrix} = \begin{bmatrix} l_x^k - r_x^t \\ l_y^k - r_y^t \end{bmatrix} \tag{3}$$

The Jacobian of the above measurement function can ba written as:

$$H_l\left(\mathbf{r}^t, \mathbf{l}^k\right) = \begin{bmatrix} \frac{\partial h_{o1}}{\partial r_x^t} & \frac{\partial h_{o1}}{\partial r_y^t} & \frac{\partial h_{o1}}{\partial r_x^{t+1}} & \frac{\partial h_{o1}}{\partial r_y^{t+1}} \\ \frac{\partial h_{o2}}{\partial r_x^t} & \frac{\partial h_{o2}}{\partial r_y^t} & \frac{\partial h_{o2}}{\partial r_x^{t+1}} & \frac{\partial h_{o2}}{\partial r_y^{t+1}} \end{bmatrix} \tag{4}$$

1.4 Exploiting Sparsity

1.4.4 Visualize Trajectory and Landmarks for 2d_linear.npz

The default method of solving the equation $Ax = b$ was done using scipy.sparse's spsolve method which runs multifrontal LU factorization in the backend. This achieved a runtime of 0.04870s on average. The ground truth and predicted trajectory are shown below in Fig 4 and Fig. 14

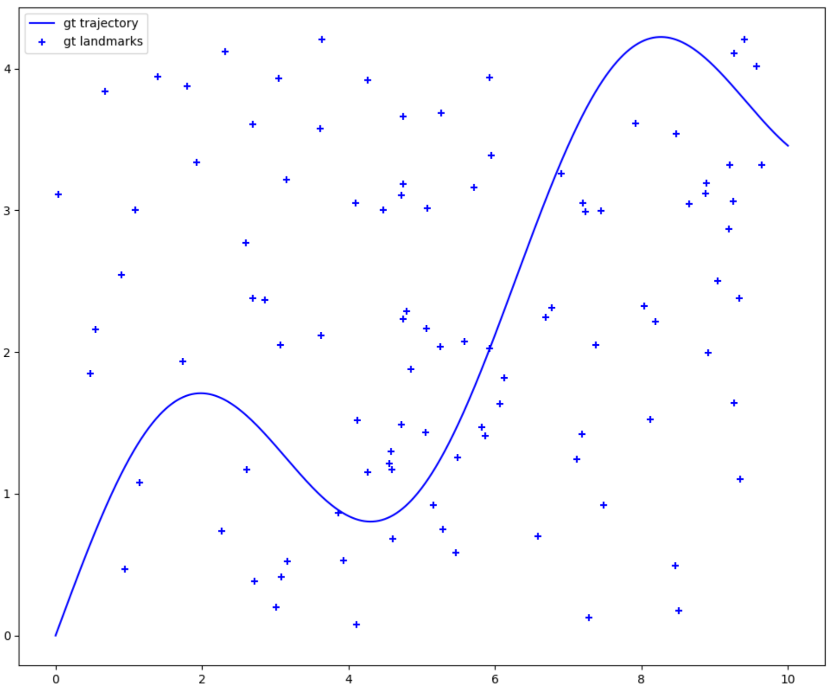


Figure 4: Ground truth landmarks and trajectory

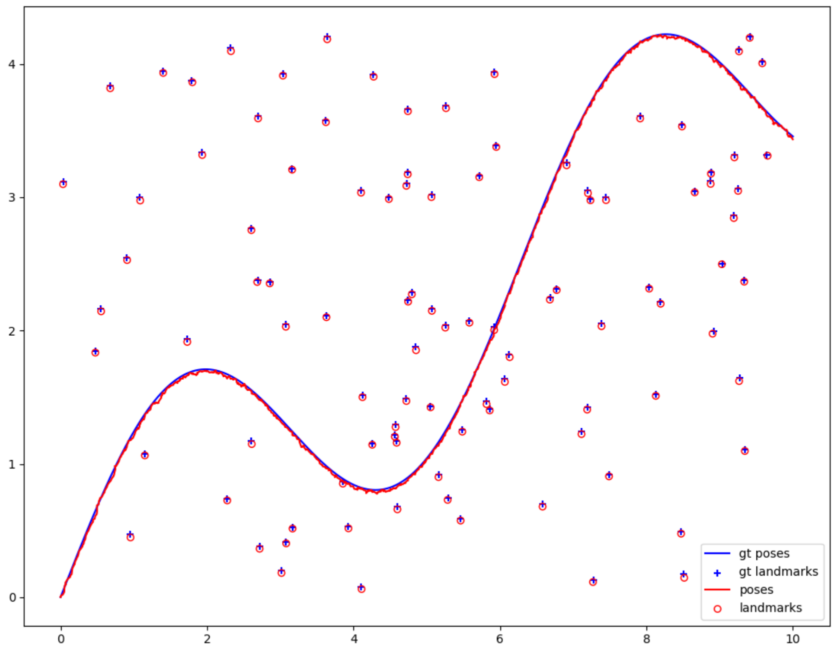


Figure 5: Trajectory predicted from measurements after least squares minimization (default method)

Other methods of solving the linear system $Ax = b$ were also compared using runtime to analyze efficiency:

Method	**Average Runtime (s)**
default	0.04870
pinv	1.1802
lu	0.0260
lu_colamd	0.1152
qr	0.3671
qr_colamd	0.2509

Table 1: Runtime comparison of different methods

Visuaulization of different methods for 2d_linear.npz

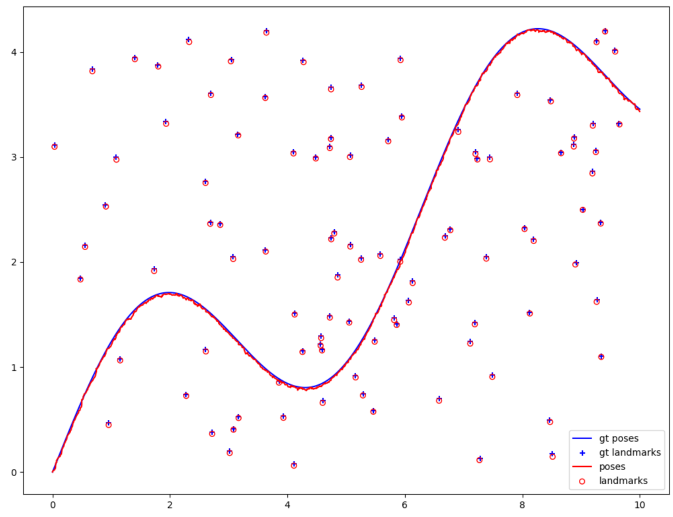


Figure 6: Minimization results using pinv method

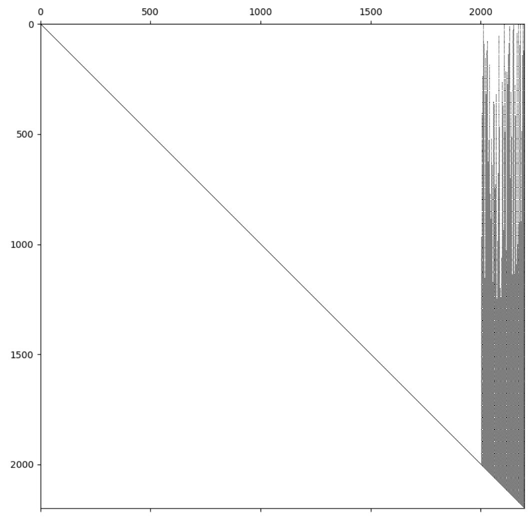


Figure 7: LU sparse matrix

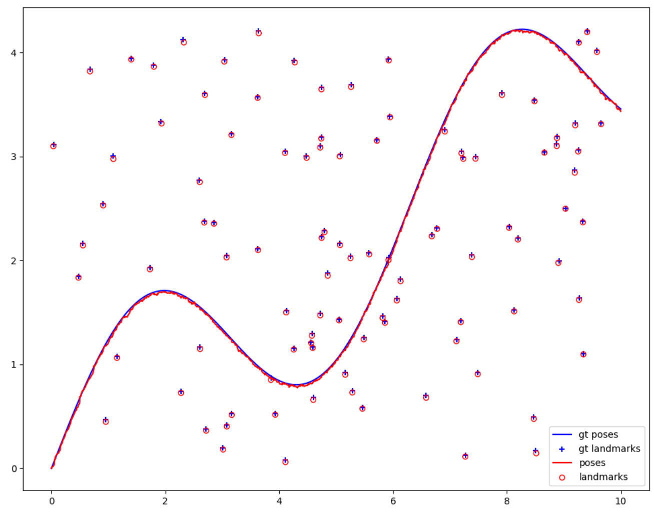


Figure 8: Minimization results using LU method

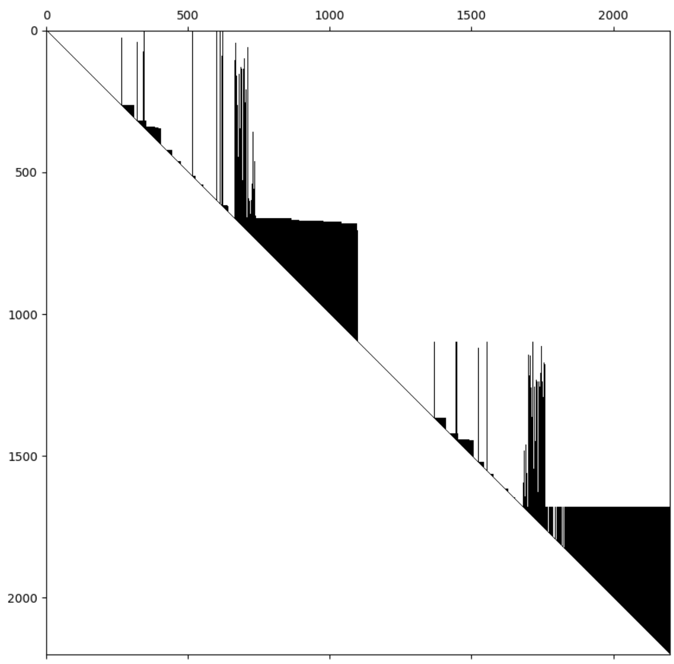


Figure 9: LU_colamd sparse matrix

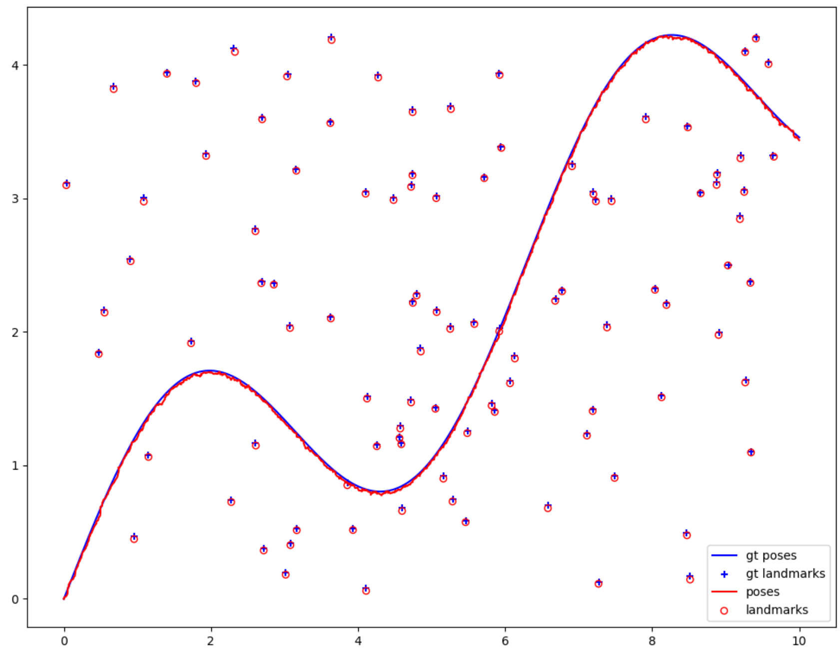


Figure 10: Minimization results using LU_colamd method

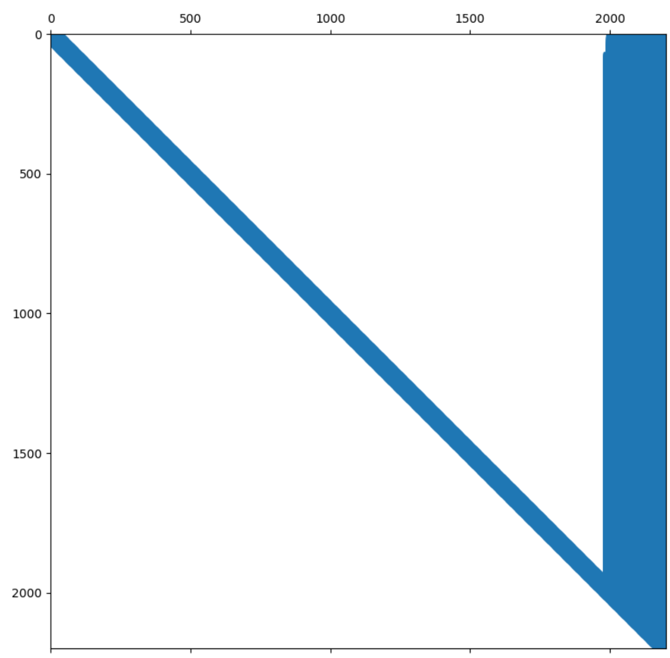


Figure 11: QR sparse matrix

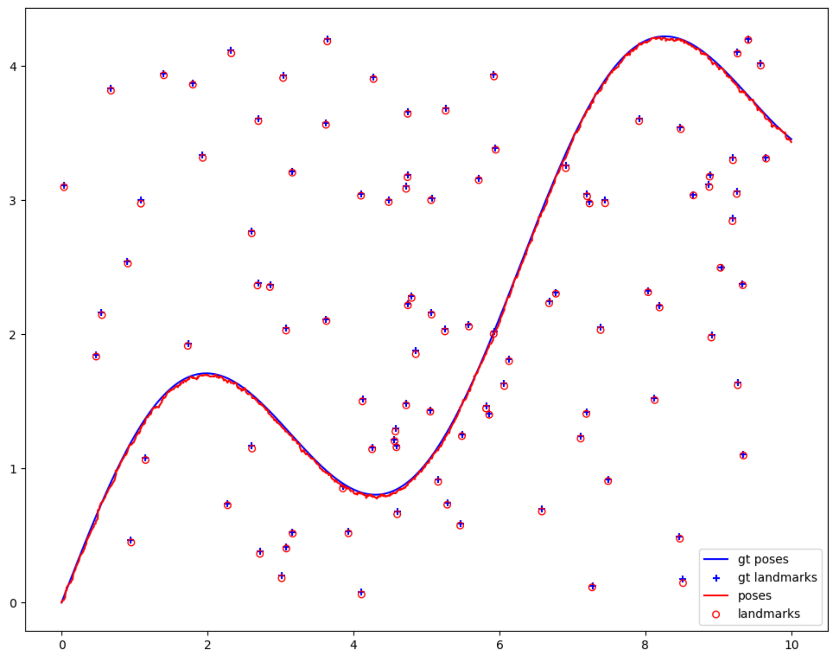


Figure 12: Minimization results using QR method

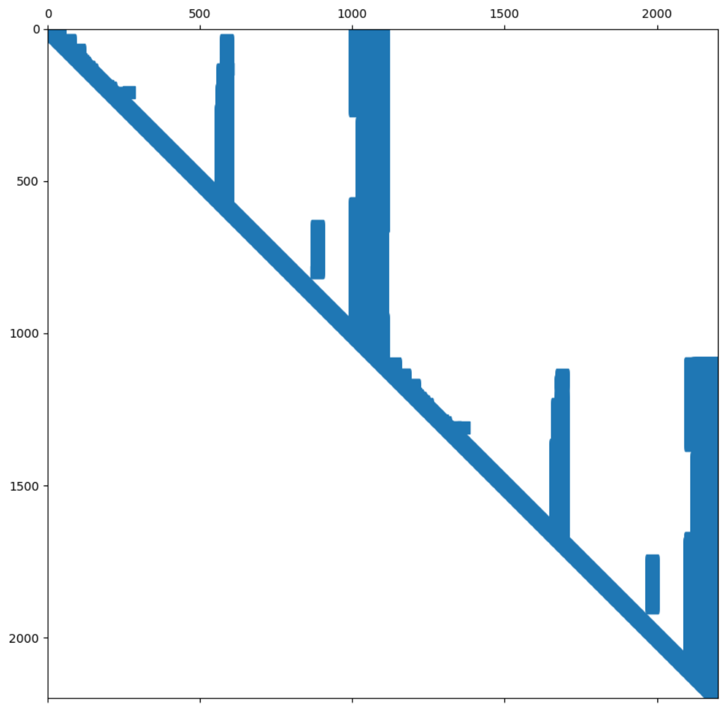


Figure 13: QR_colamd sparse matrix

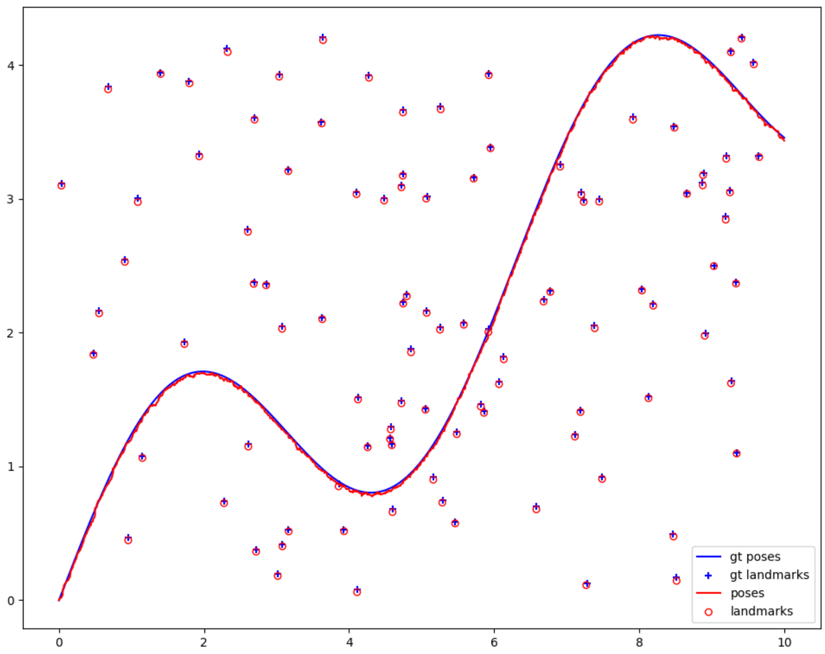


Figure 14: Minimization results using QR_colamd method

Observations for 2D_linear.npz

Method	**Average Runtime (s)**
default	0.04870
pinv	1.1802
lu	0.0260
lu_colamd	0.1152
qr	0.3671
qr_colamd	0.2509

Runtime comparisons (repeated)

Figure 7 shows the LU sparse matrix, while Figure 8 shows the minimization results obtained using the LU method. The same is repeated for LU_colamd, QR, and QR_colamd methods. Table 1 also shows the runtime comparison for these methods.

It can be seen that the pinv method takes the longest. This makes sense since the pinv is naieve and does not utilize sparsity. For the methods which do make use of sparsity, the LU solver is the fastest with a time complexity of $\frac{2}{3}n^3$ whereas the QR factorization has a time complexity of $\frac{4}{3}n^3$, with $n = order of matrix$

The colamd method rearranges (permutes) the A matrix to avoid accumulation of data in the last few columns of one of the triangular matrices (as seen in Fig. 7 and Fig. 22). This rearragement as seen in Fig. 9 and Fig. 13 helps in the following manner:

- The algorithm reorders the columns of the matrix so that the non-zero entries are clustered together
- This can reduce the number of fill-in elements created during the factorization process.
- Therefore reordering can significantly reduce the computational and memory requirements of the LU decomposition.

However, this benefit of reordering was observed only in the QR_colamd method. In the LU_colamd, reordering took longer and therefore increased the overall time required.

1.4.5 Visualize Trajectory and Landmarks for 2d_non_linear.npz

The default method of solving the equation $Ax = b$ was done using scipy.sparse's spsolve method. The ground truth and predicted trajectory are shown below in Fig 15 and Fig. 16

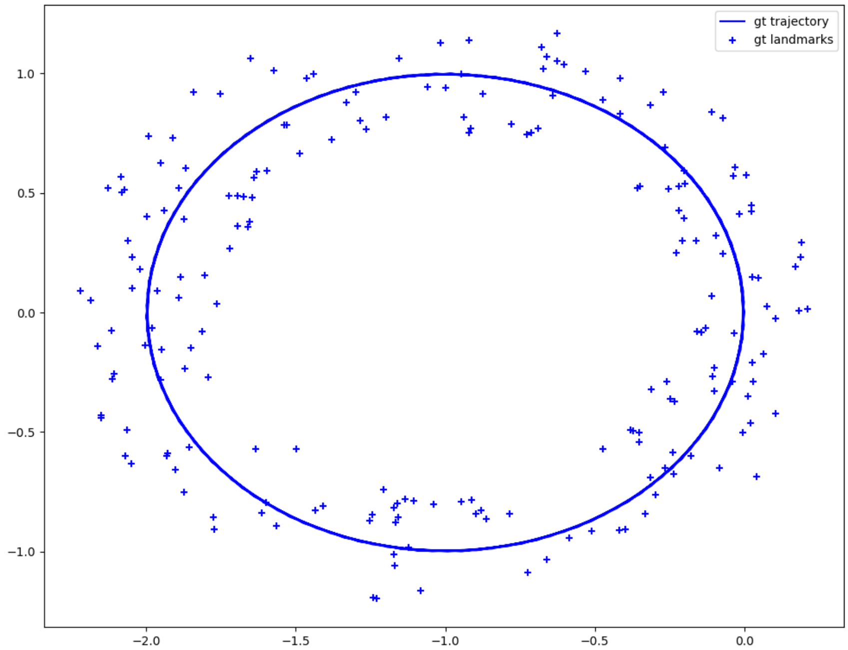


Figure 15: Ground truth landmarks and trajectory for 2d_linear_loop.npz

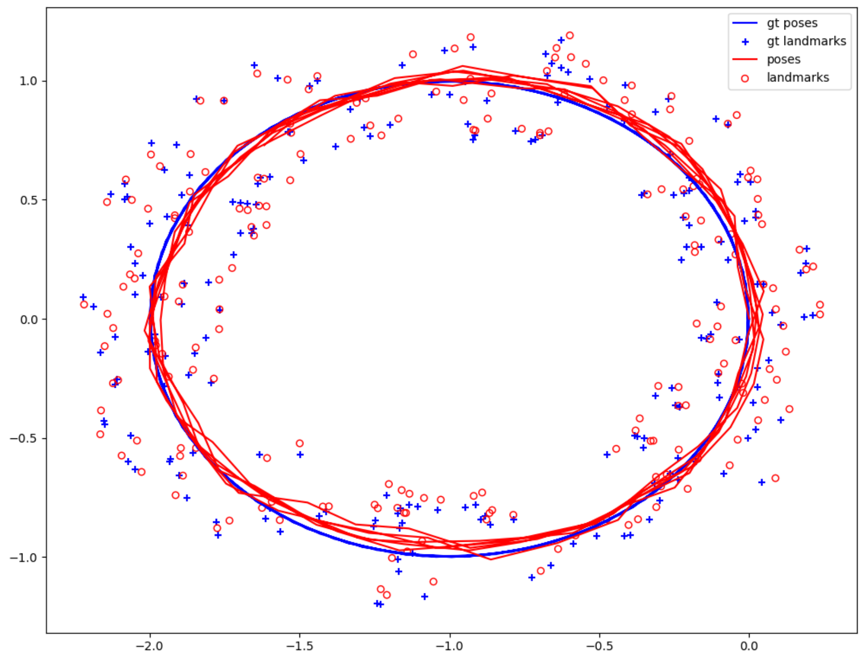


Figure 16: Trajectory predicted from measurements after least squares minimization (default method)

Other methods of solving the linear system $Ax = b$ were also compared using runtime to analyze efficiency:

Method	**Average Runtime (s)**
default	0.0054
pinv	0.1629
lu	0.0295
lu_colamd	0.0087
qr	0.2487
qr_colamd	0.0190

Table 2: Runtime comparison of different methods

Visuaulization of different methods for 2d_linear_loop.npz

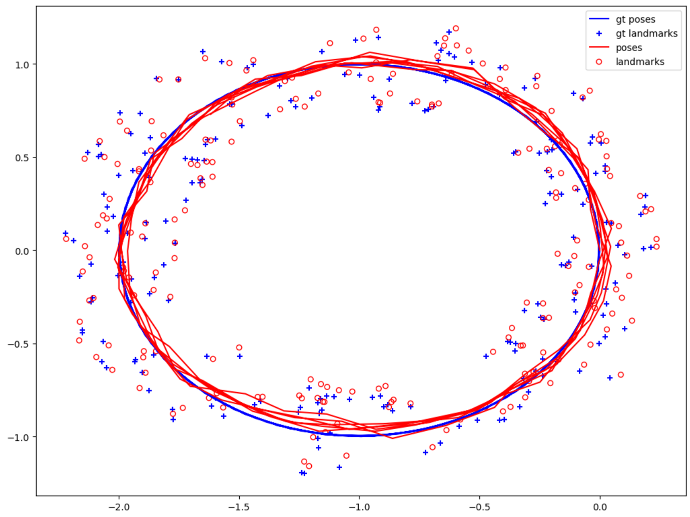


Figure 17: Minimization results using the pinv (pseudoinverse) method

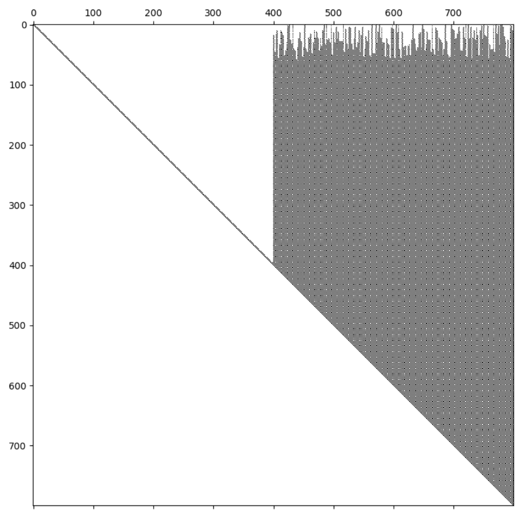


Figure 18: LU sparse matrix

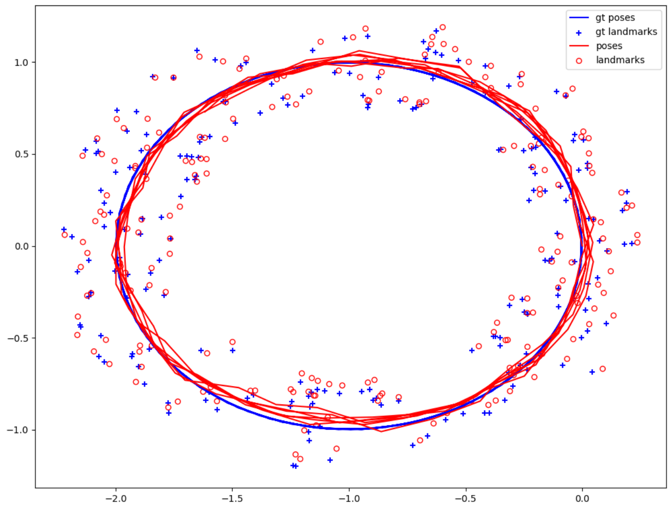


Figure 19: Minimization results using LU method

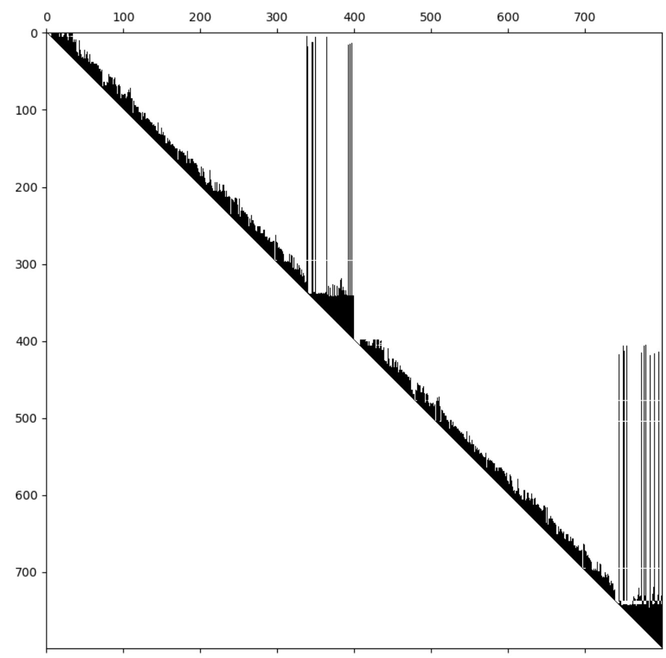


Figure 20: LU_colamd sparse matrix

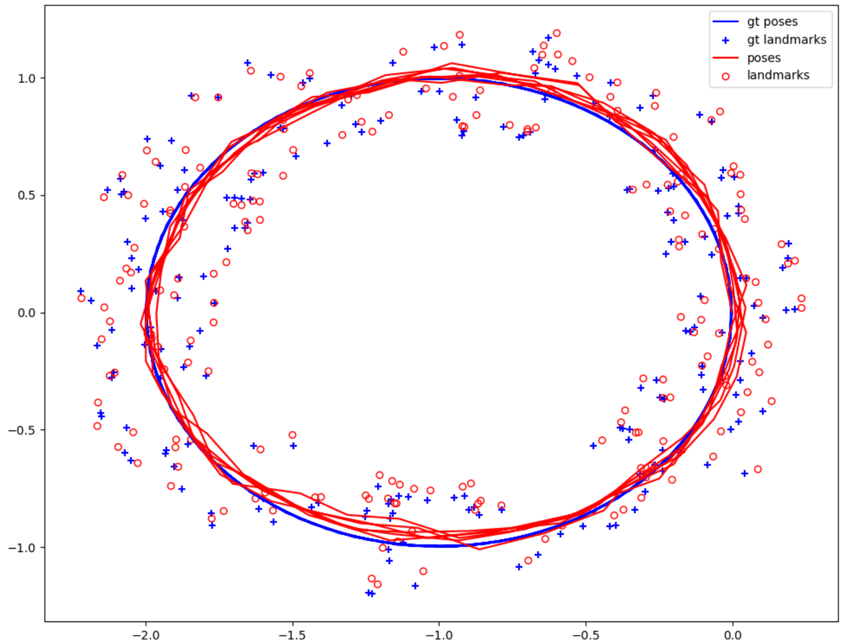


Figure 21: Minimization results using LU_colamd method

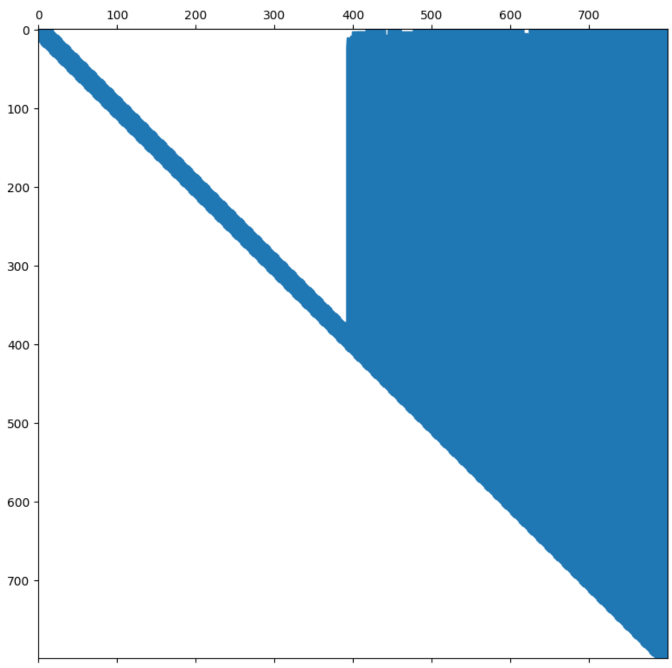


Figure 22: QR sparse matrix

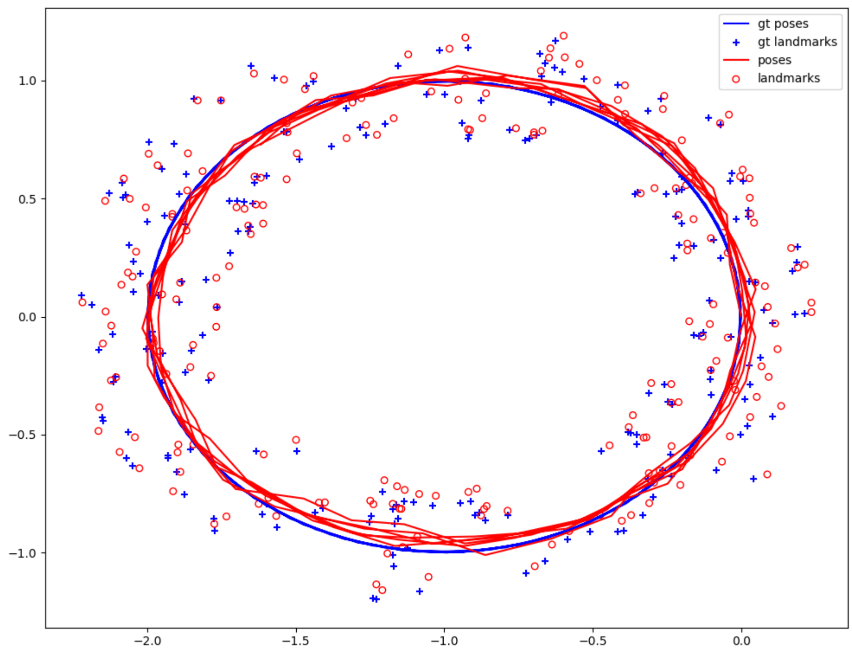


Figure 23: Minimization results using QR method

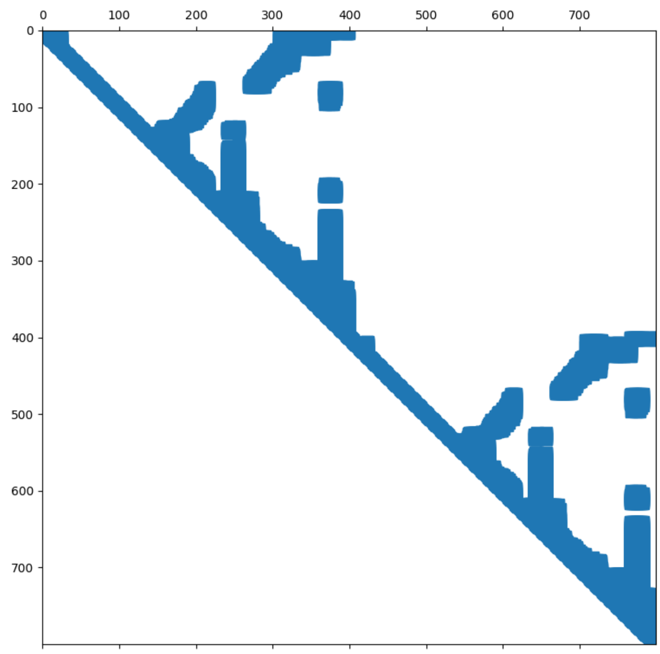


Figure 24: QR_colamd sparse matrix

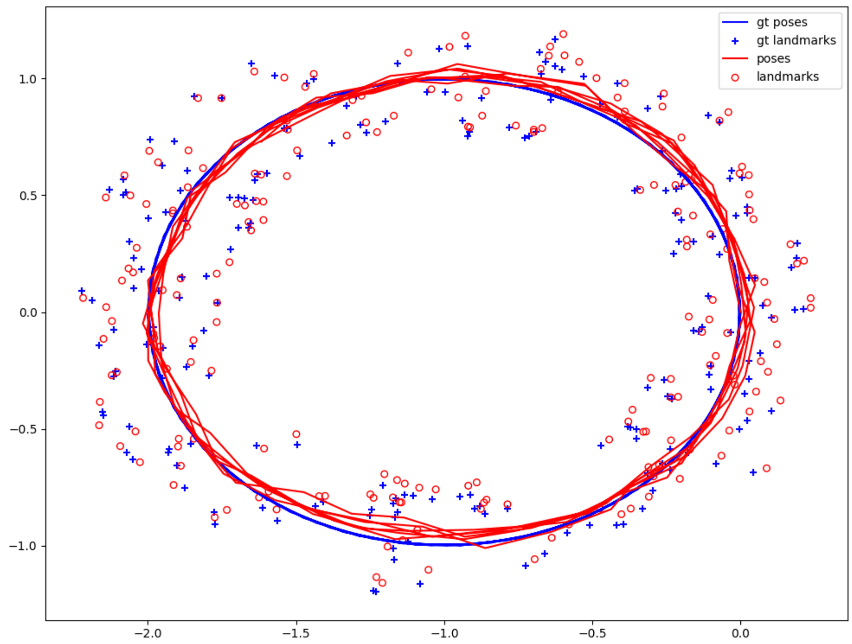


Figure 25: Minimization results using QR_colamd method

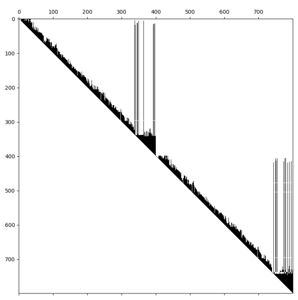
Observations for 2D_linear_loop.npz

2D_linear_loop	**Average Runtime (s)**	**2D_linear**	**Average Runtime (s)**
default	0.0054	default	0.04870
pinv	0.1629	pinv	1.1802
lu	0.0295	lu	0.0260
lu_colamd	0.0087	lu_colamd	0.1152
qr	0.2487	qr	0.3671
qr_colamd	0.0190	qr_colamd	0.2509

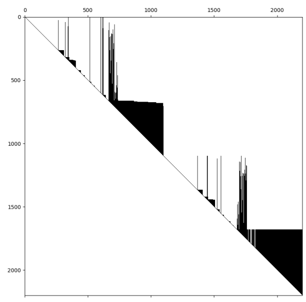
Runtime comparisons (repeated)

Figure 7 shows the LU sparse matrix, while Figure 8 shows the minimization results obtained using the LU method. The same is repeated for pinv, LU_colamd, QR, and QR_colamd methods. Table 2 also shows the runtime comparison for these methods.

It can be seen that the QR method takes the longest (even more than pinv). However, the colamd reordering for both QR and LU significantly boosts them and makes it more efficient than pinv. **Therefore, it can be seen for this case where the A matrix is more sparse, the reordering advantage is more significant when solving the decomposition.** Both the 2D_linear.npz and 2D_linear_loop.npz sparse matrices obtained by the **LU decomposition are shown below to compare the sparsity.**



sparse matrix of 2D_linear_loop.npz



sparse matrix of 2D_linear.npz

Figure 26: Sparsity Comparisons

An improvement in runtime is observed when compared with 2d-linear.npz data since there seems to be lesser dense grouping as in the 2d_linear.npz case.

2 2D Nonlinear SLAM

2.1 Measurement Function

2.1.1 Odometry and Bearing+Range Estimation Functions

Given the robot poses $\mathbf{r}^{t+1} = \left[r_x^{t+1}, r_y^{t+1}\right]^\top$ $\mathbf{r}^t = \left[r_x^t, r_y^t\right]^\top$ at times t and t+1, we find the measurement function to be:

$$h_o\left(\mathbf{r}^t, \mathbf{r}^{t+1}\right) = \begin{bmatrix} h_{o1} \\ h_{o2} \end{bmatrix} = \begin{bmatrix} r_x^{t+1} - r_x^t \\ r_y^{t+1} - r_y^t \end{bmatrix} \tag{5}$$

The measurement function for landmark estimation is given in terms of bearing and range as:

$$h_l\left(\mathbf{r}^t, \mathbf{l}^k\right) = \begin{bmatrix} \operatorname{atan2}\left(l_y^k - r_y^t, l_x^k - r_x^t\right) \\ \left(\left(l_x^k - r_x^t\right)^2 + \left(l_y^k - r_y^t\right)^2\right)^{\frac{1}{2}} \end{bmatrix} = \begin{bmatrix} \theta \\ d \end{bmatrix} = \begin{bmatrix} h_{l1} \\ h_{l2} \end{bmatrix} \tag{6}$$

2.1.2 Jacobian of Landmark Measurement Function

Given the landmark measurement function above, we can define the jacobian in the following form:

$$H_l = \begin{bmatrix} \frac{\partial h_{l1}}{\partial r_x^t} & \frac{\partial h_{l1}}{\partial r_y^t} & \frac{\partial h_{l1}}{\partial l_x^k} & \frac{\partial h_{l1}}{\partial l_y^k} \\ \frac{\partial h_{l2}}{\partial r_x^t} & \frac{\partial h_{l2}}{\partial r_y^t} & \frac{\partial h_{l2}}{\partial l_x^k} & \frac{\partial h_{l2}}{\partial l_y^k} \end{bmatrix}$$

Now, let's define the following terms to make the derivation simpler:

$$\delta_{l_y} = l_y^k - r_y^t$$

$$\delta_{l_x} = l_x^k - r_x^t$$

$$range = \sqrt{\left(l_x^k - r_x^t\right)^2 + \left(l_y^k - r_y^t\right)^2}$$

Then the jacobian's final form is:

$$H_l = \begin{bmatrix} \frac{\delta_y}{range^2} & -\frac{\delta_x}{range^2} & -\frac{\delta_y}{range^2} & \frac{\delta_x}{range^2} \\ -\frac{\delta_x}{range} & -\frac{\delta_y}{range} & \frac{\delta_x}{range} & \frac{\delta_y}{range} \end{bmatrix} \tag{7}$$

2.2 Build Linear System

The Non Linear system is linearized by taking the first order approximation using the Taylor expansion.

The measurement function (both odometry and landmark) can be represented as:

$$h(x) = Hx + h_0 \tag{8}$$

The least squares approximation will result in minimizing the predicted measurement (measurement function) and the actual measurement z.

$$\text{minimize}\|Hx + h_0 - z\|_{\Sigma}^2 \tag{9}$$

$$\text{minimize}\|Hx - (z - h_0)\|_{\Sigma}^2 \tag{10}$$

The above equation 10 is of the form $|Ax - b|$.

The b matrix will then be $z - h_0$ which is equivalent to *(actual measurement - predicted measurement)*

2.3 Solver

The pinv method was chosen to visualize the trajectory and landmarks before and after optimization. The results are shown below:

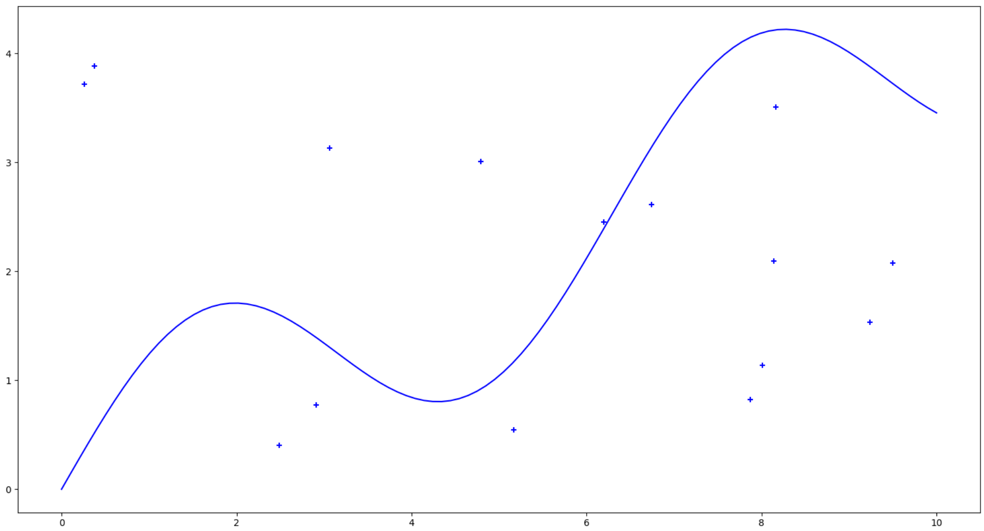


Figure 27: Ground Truth of Trajectory and Landmarks

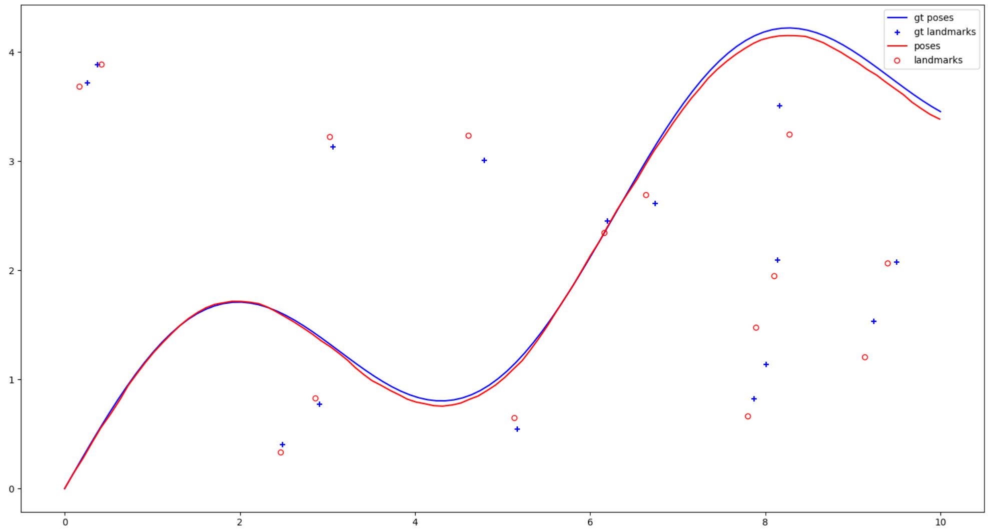


Figure 28: Predicted landmarks and poses **before** optimization

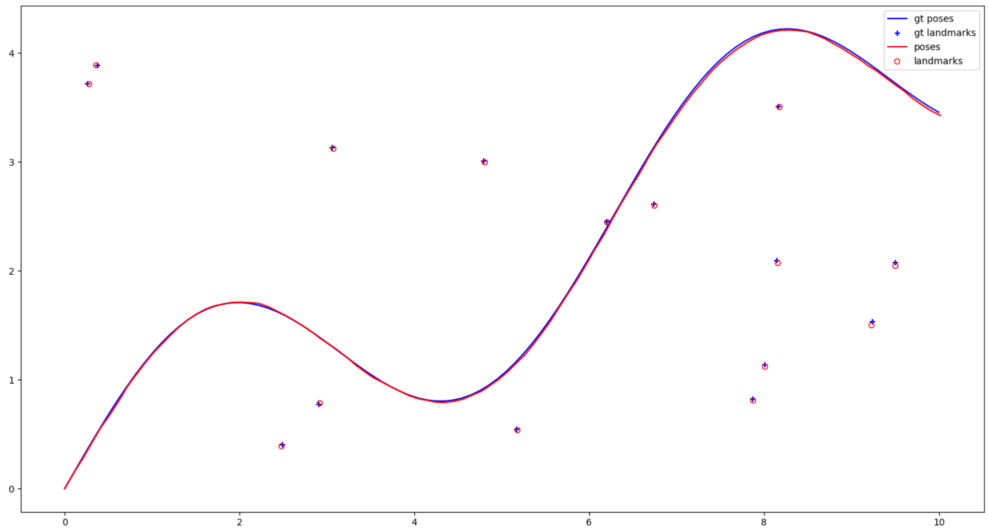


Figure 29: Predicted landmarks and poses **after** optimization

Notes about Non-linear optimization

The Non Linear optimization process is different in that we have a residual h_0 term which causes the optimizer to solve for the error between *predicted and actual measurement.* In the linear case the optimizer directly solves for x in the equation $Ax = b$.

Additionally, the non-linear method also requires a good initial estimate to start. This is because the optimization is iterative for the non-linear case if the initialization is bad it will take longer or not converge within the required error tolerance.

References

[1] Frank Dellaert and Michael Kaess. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, 6(1-2):1–139, 2017.