1 2D Linear SLAM

(a) For landmark m, the measurement function at time step t is:

$$h_m^t = \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} lx_m^t - rx^t \\ ly_m^t - ry^t \end{bmatrix}$$

where lx_m, ly_m, rx and ry are landmark m (x, y) and robot pose (x, y) values. respectively. The Jacobian of h_m^t with respect to state vector $x = (l_m^t, r^t)$ is:

$$H_m^t = \frac{\partial h_m^t}{\partial x} = \begin{bmatrix} \frac{\partial \Delta x}{\partial rx^t} & \frac{\partial \Delta x}{\partial ry^t} & \frac{\partial \Delta x}{\partial lx_m^t} & \frac{\partial \Delta x}{\partial ly_m^t} \\ \frac{\partial \Delta y}{\partial rx^t} & \frac{\partial \Delta y}{\partial ry^t} & \frac{\partial \Delta y}{\partial lx_m^t} & \frac{\partial \Delta y}{\partial ly_m^t} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

The odometry measurement function at timestep t is:

$$h_o^t = \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} rx^t - rx^{t-1} \\ ry^t - ry^{t-1} \end{bmatrix}$$

The Jacobian of the odometry measurement function is with respect to the state vector $x = (r^t, r^{t-1})$ is:

$$H_o^t = \frac{\partial h_o^t}{\partial x} = \begin{bmatrix} \frac{\partial \Delta x}{\partial rx^{t-1}} & \frac{\partial \Delta x}{\partial ry^{t-1}} & \frac{\partial \Delta x}{\partial rx^t} & \frac{\partial \Delta x}{\partial ry^t} \\ \frac{\partial \Delta y}{\partial rx^{t-1}} & \frac{\partial \Delta y}{\partial ry^{t-1}} & \frac{\partial \Delta y}{\partial rx^t} & \frac{\partial \Delta y}{\partial ry^t} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

(b) The code is implemented in the provided files. They are present in the `code/slam_solvers/linear/` and `code/slam_solvers/util/` folders.

(c) The code is implemented in the provided files. They are present in the `code/slam_solvers/linear/` folder.

`iii)` I experimented with the 'econ' parameter in the QR solver and found that the results were not negatively affected when used. This is because the A matrix is very much over-determined, with many more rows than columns. There is enough information to solve without using all the observations.

(d)

`i)` All the implementations have passed the evaluation at the end of their run on the `2D_linear`. Using QR2 method, the RMSE error using only pure odometry is 0.1887 whereas the RMSE error obtained using SLAM is 0.0191. The resulting trajectory generated is shown in Figure 1. The sparsity patterns for the different decompositions are shown in Figure 2. Table 1 includes the different timings for each method.

The decompositions in order of efficiency (execution time) are QR2 > Chol2 > QR1 > Chol1 > Pinv. 'QR2' performs the best for this dataset. One reason for QR2 performing faster than Chol2 is that we have to compute $A^T A$ for Cholesky before decomposing and this adds some compute time, whereas in QR2 we can performing the decomposition on the A matrix directly. Similarly for QR1 and Chol1. QR2/Chol2 perform better than QR1/Chol1 respectively due to fill-in-reducing ordering. The A matrix that we have is really large here, and hence, computing the pseudo-inverse takes a lot of time when using the `inv` function in MATLAB.

`ii)` All the implementations have passed the evaluation at the end of their run on the `2D_linear_loop`. Using QR2 method, the RMSE error using only pure odometry is 0.8472 whereas the RMSE error

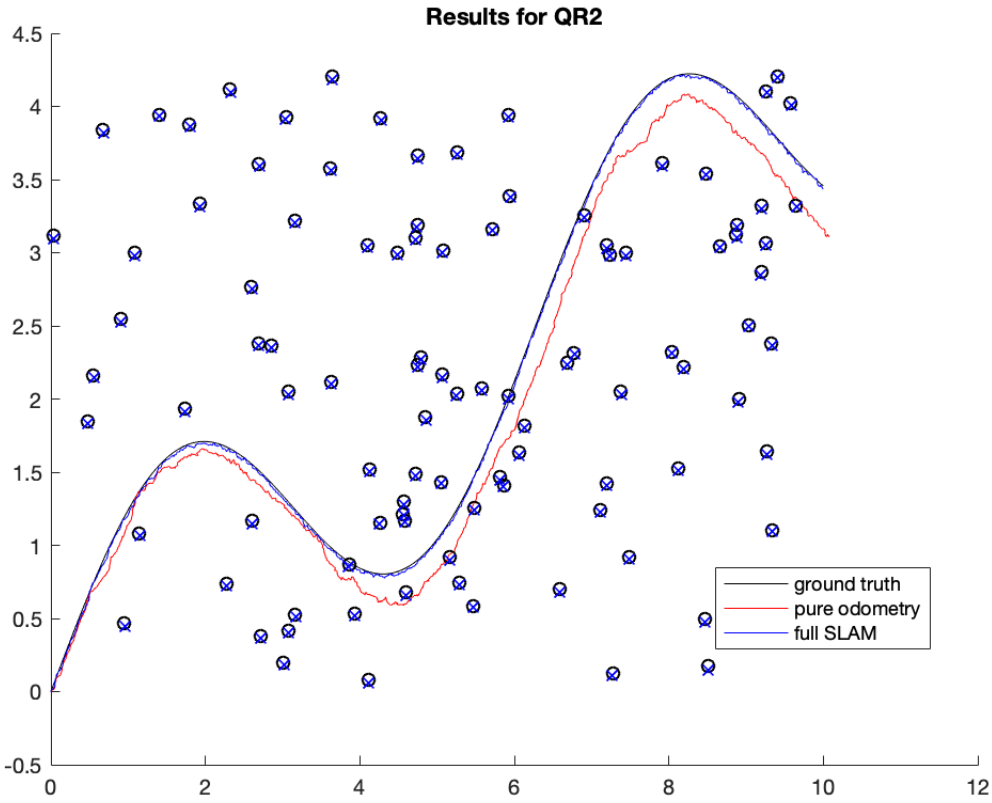

Figure 1: Resulting Trajectory for `2D_linear` dataset

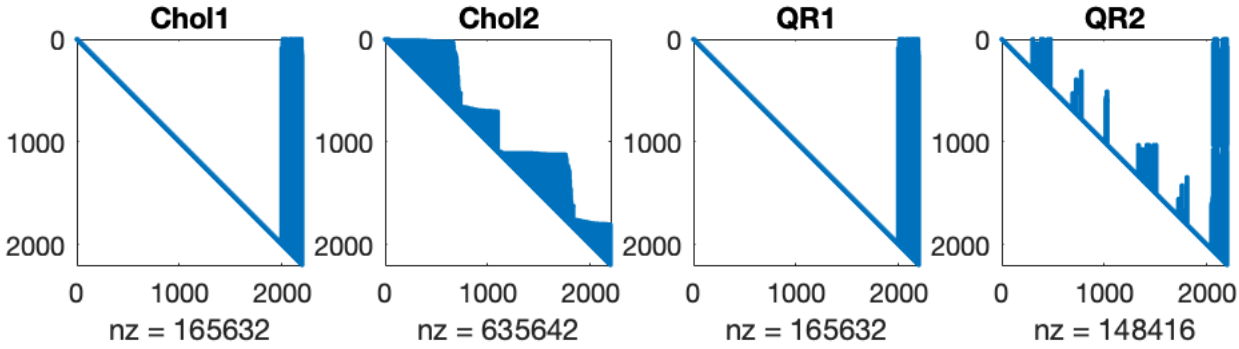

Figure 2: Sparsity Patterns for `2D_linear` dataset

obtained using SLAM is 0.0451. The resulting trajectory generated is shown in Figure 3. The sparsity patterns for the different decompositions are shown in Figure 4. Table 2 includes the different timings for each method.

The decompositions in order of efficiency (execution time) are QR2 > Chol2 > Pinv > Chol1 > QR1. 'QR2' still performs best on this dataset, however, the order is different compared to the previous

Method	Timing (in seconds)
Pinv	2.973593e-01
Chol1	2.293855e-01
Chol2	1.928981e-01
QR1	2.233621e-01
QR2	1.451131e-01

Table 1: Performance timings for `2D_linear` dataset

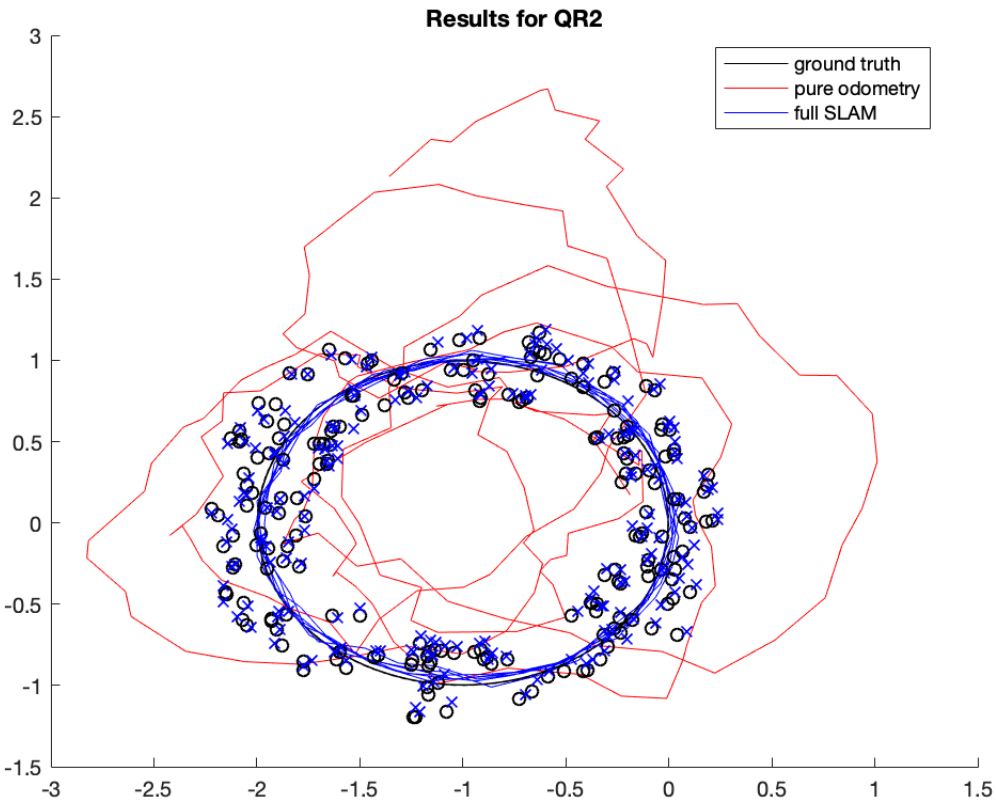

Figure 3: Resulting Trajectory for `2D_linear_loop` dataset

Method	Timing (in seconds)
Pinv	1.854644e-02
Chol1	9.079365e-02
Chol2	1.794297e-02
QR1	1.301130e-01
QR2	1.374937e-02

Table 2: Performance timings for `2D_linear_loop` dataset

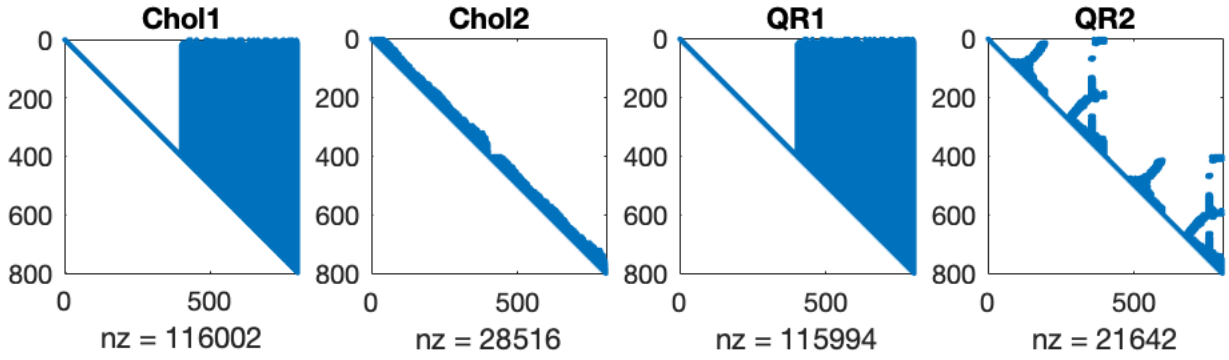

Figure 4: Sparsity Patterns for `2D_linear_loop` dataset

dataset. The reason for Chol1 and QR1 performing much worse than Pinv is clearly visible in the sparsity patterns. For both these methods, the R matrices are almost non-sparse and thus the amount of time taken to compute the results exceeds the time taken to compute the pseudo-inverse. Here since the A matrix is small, the pseudo-inverse computation does not take as long as the decomposition in QR1 or Chol1. The reason for QR2 performing better than Chol2 is the same as part `i)`.

2 2D Nonlinear SLAM

(a) The code is implemented in the provided files. They are present in the `code/slam_solvers/nonlinear/` folder.

(b) We represent the difference between the robot and landmark as $x = lx - rx$ and $y = ly - ry$. The Jacobian of the non-linear landmark measurement function for a given robot and landmark with respect to the state vector $x = (lx, ly, rx, ry)$ is given by

$$
\begin{aligned}
H_m &= \frac{\partial h_m}{\partial x} \\
&= \begin{bmatrix} \frac{\partial \theta}{\partial rx} & \frac{\partial \theta}{\partial ry} & \frac{\partial \theta}{\partial lx} & \frac{\partial \theta}{\partial ly} \\ \frac{\partial d}{\partial rx} & \frac{\partial d}{\partial ry} & \frac{\partial d}{\partial lx} & \frac{\partial d}{\partial ly} \end{bmatrix} \\
&= \begin{bmatrix} \frac{y}{x^2+y^2} & \frac{-x}{x^2+y^2} & \frac{-y}{x^2+y^2} & \frac{x}{x^2+y^2} \\ \frac{-x}{\sqrt{x^2+y^2}} & \frac{-y}{\sqrt{x^2+y^2}} & \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} \end{bmatrix}
\end{aligned}
$$

This is computed for every robot and landmark location over multiple iterations.

(c) The code is implemented in the provided files. They are present in the `code/slam_solvers/nonlinear/` folder.

(d) I have implemented the QR2 solver from the linear solvers (fastest implementation). The code is implemented in the provided files. The implementation has passed the evaluation at the end of the run. The resulting trajectory is shown below in Figure 5. The RMSE error using pure odometry is 0.0579 whereas the RMSE error obtained using SLAM is 0.0153.

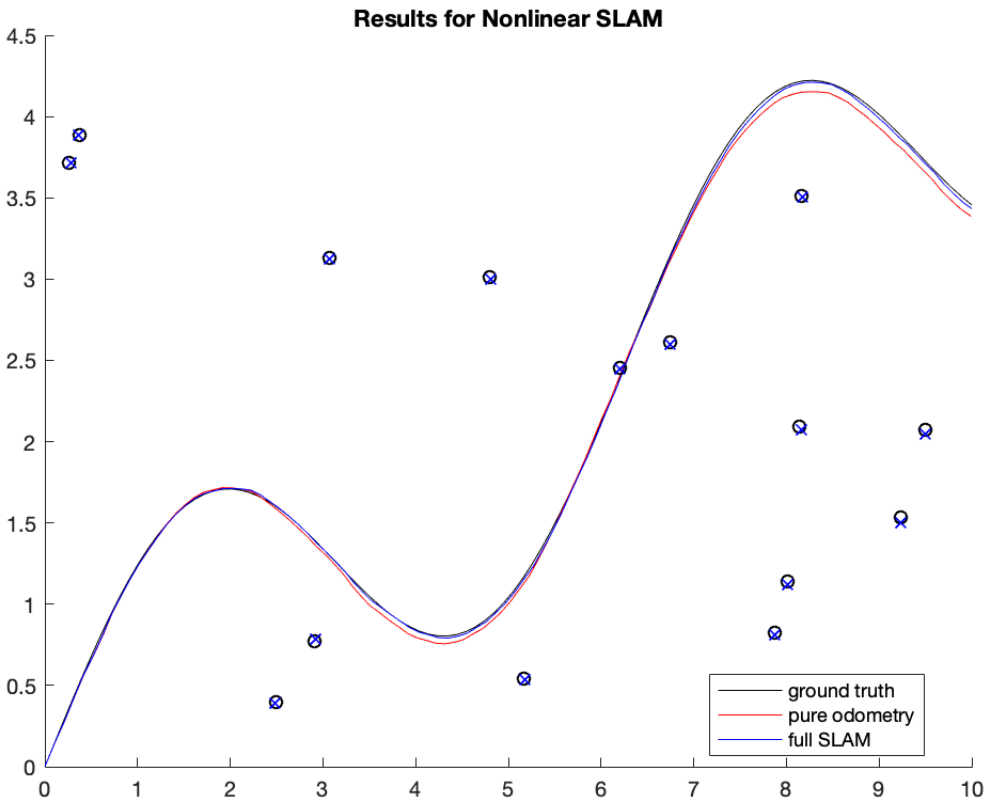

Figure 5: Resulting Trajectory using 2D Nonlinear SLAM

(e) The nonlinear SLAM algorithm has to be implemented in an incremental fashion compared to the linear algorithm which was implemented in a batch fashion. This is because the change in error function as a function of state change cannot be encapsulated in a linear form and hence must be approximated. The approximation is only valid close to the point of linearization, so must be re-calculated when the state changes.

3 References

[**1**] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.