Homework 3 - Linear and Non-Linear SLAM Solvers

Sagar Sachdev (AndrewID: sagarsac)

30 March 2022

1 2D Linear SLAM

1.1 Measurement Function

1.1.1

The measurement function can be written as:

$$m_{poses} = \begin{pmatrix} r_x^{t+1} - r_x^t \\ r_y^{t+1} - r_y^t \end{pmatrix} \tag{1}$$

Given this, the state can be written as:

$$X_p = \begin{pmatrix} r_x^t \\ r_y^t \\ r_x^{t+1} \\ t_y^{t+1} \end{pmatrix} \tag{2}$$

Therefore the Jacobian can be written as:

$$J_{poses} = \begin{pmatrix} \frac{\partial}{\partial r_x^t}\left(r_x^{t+1} - r_x^t\right) & \frac{\partial}{\partial\, r_y^t}\left(r_x^{t+1} - r_x^t\right) & \frac{\partial}{\partial\, r_x^{t+1}}\left(r_x^{t+1} - r_x^t\right) & \frac{\partial}{\partial\, r_y^{t+1}}\left(r_x^{t+1} - r_x^t\right) \\ \frac{\partial}{\partial\, r_x^t}\left(r_y^{t+1} - r_y^t\right) & \frac{\partial}{\partial\, r_y^t}\left(r_y^{t+1} - r_y^t\right) & \frac{\partial}{\partial\, r_x^{t+1}}\left(r_y^{t+1} - r_y^t\right) & \frac{\partial}{\partial\, r_y^{t+1}}\left(r_y^{t+1} - r_y^t\right) \end{pmatrix}$$

$$=> J_a = \begin{pmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix} \tag{3}$$

1.1.2

Similarly for landmarks, the measurement function can be written as:

$$m_{landmarks} = \begin{pmatrix} l_x^k - r_x^t \\ l_y^k - r_y^t \end{pmatrix} \tag{4}$$

The state can thus be written as:

$$X_l = \begin{pmatrix} r_x^t \\ r_y^t \\ l_x^k \\ l_y^k \end{pmatrix} \tag{5}$$

Therefore the Jacobian can be written as:

$$J_{land} = \begin{pmatrix} \frac{\partial}{\partial r_x^t}\left(l_x^k - r_x^t\right) & \frac{\partial}{\partial\, r_y^t}\left(l_x^k - r_x^t\right) & \frac{\partial}{\partial\, l_x^k}\left(l_x^k - r_x^t\right) & \frac{\partial}{\partial\, l_y^k}\left(l_x^k - r_x^t\right) \\ \frac{\partial}{\partial\, r_x^t}\left(l_y^k - r_y^t\right) & \frac{\partial}{\partial\, r_y^t}\left(l_y^k - r_y^t\right) & \frac{\partial}{\partial\, l_x^k}\left(l_y^k - r_y^t\right) & \frac{\partial}{\partial\, l_y^k}\left(l_y^k - r_y^t\right) \end{pmatrix}$$

$$=> J_a = \begin{pmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix} \tag{6}$$

1.2 Build a linear System

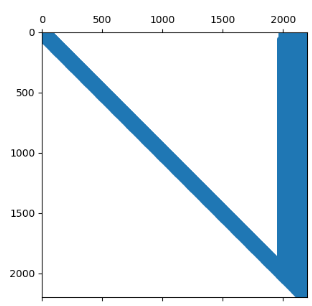
Done in python file

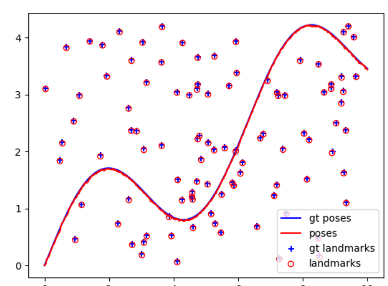
1.3 Solvers

Done in python file

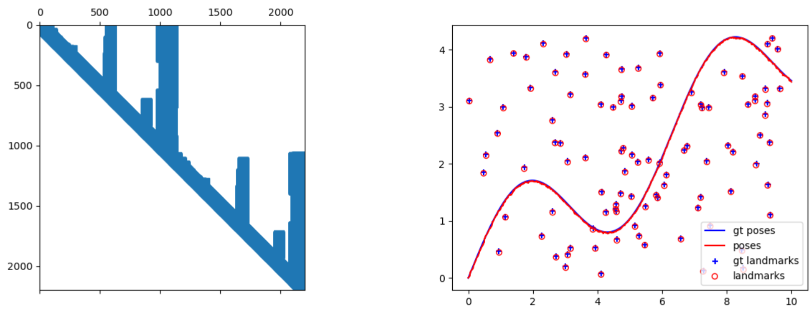
1.4 Exploit Sparsity

Efficiency of QR (in terms of runtime) = 0.318857908248901 37s

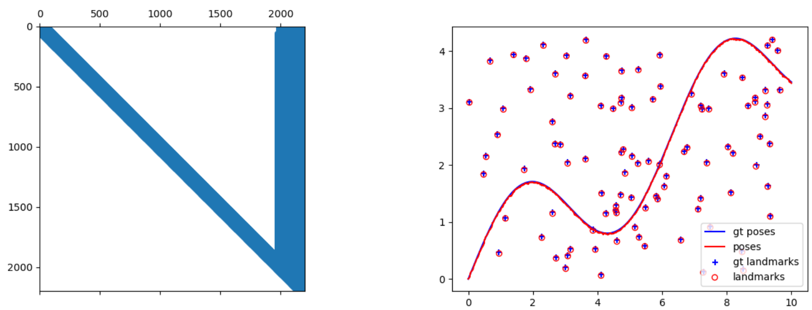




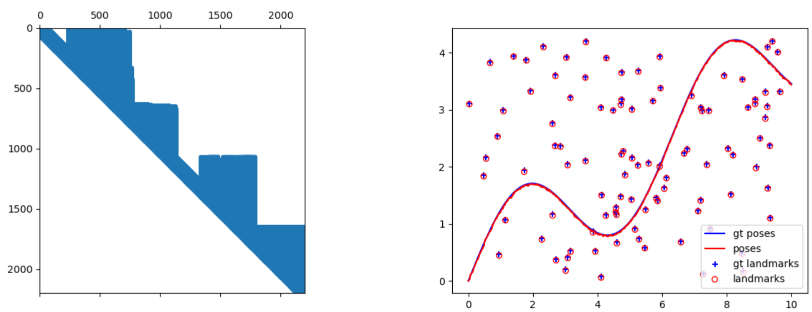
Efficiency of QR_COLAMD (in terms of runtime) = 0.18664789199829102s



Efficiency of LU (in terms of runtime) = 0.037360191345214844s



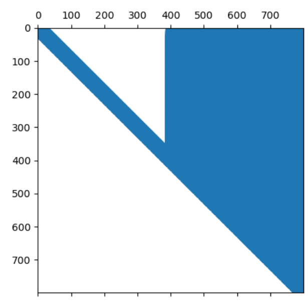
Efficiency of LU_COLAMD (in terms of runtime) = 0.06786632537841797s

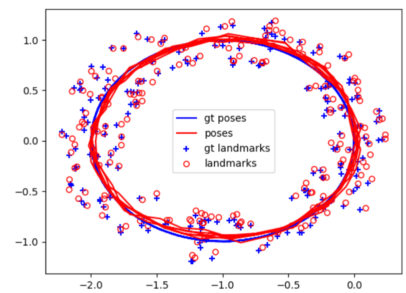


From the above figures and the runtimes, it can be seen that in general, LU is faster than QR in general and the runtimes can be ranked as LU > LU_COLAMD > QR_COLAMD > QR. This is because from the sparsity patterns it can be seen that QR is sparser than LU. In addition, QR_COLAMD and LU_COLAMD are sparser than QR and LU respectively since the number of non-zeros are reduced in the Cholesky factor.

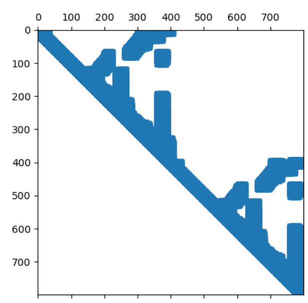
Similarly we can test this approach out on 2d_linear_loop.npz which is shown below:

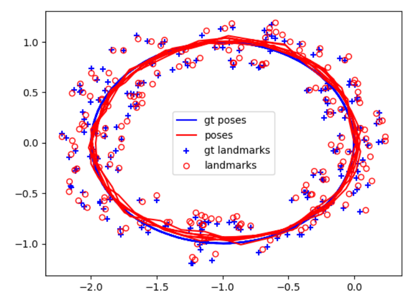
Efficiency of QR (in terms of runtime): 0.289996862411499s



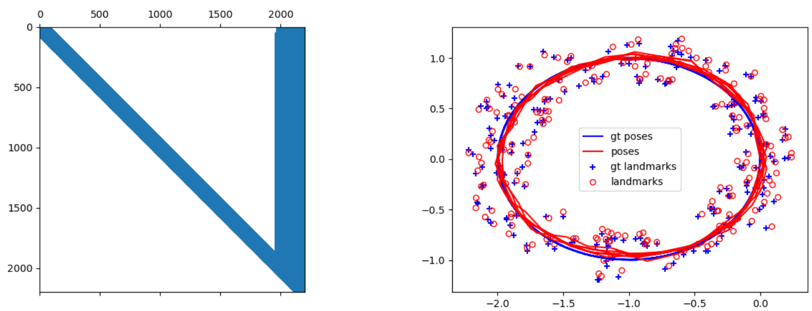


Efficiency of QR_COLAMD (in terms of runtime): 0.03893566131591797s

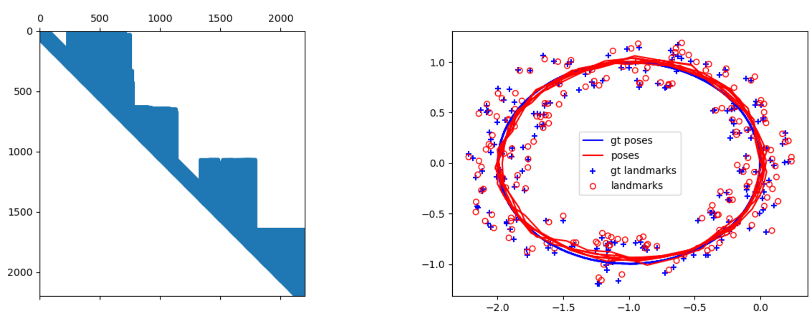




Efficiency of LU (in terms of runtime): 0.02581334114074707s



Efficiency of LU_COLAMD (in terms of runtime): 0.00812530517578125s



Comparing this to the 2d_linear.npz we can see that there is slightly more drift in the visualizations for 2d_linear_loop.npz. In addition, the sparsity patterns for QR seems to be denser than the sparsity patterns for LU, which could possibly be due to QR having a lower efficiency. In terms of runtimes, the efficiency can be ranked as LU_COLAMD > LU > QR_COLAMD > QR

2 2D Nonlinear SLAM

2.1 Measurement Function

2.1.1

Done in python

2.1.2

The Jacobian can be written as:

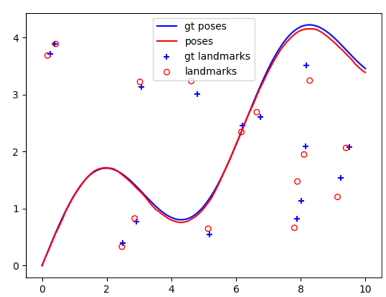
$$J_l = \begin{pmatrix} \frac{l_y^k - r_y^t}{\left(l_y^k - r_y^t\right)^2 + (l_x^k - r_x^t)^2} & \frac{-l_x^k + r_x^t}{\left(l_y^k - r_y^t\right)^2 + (l_x^k - r_x^t)^2} & \frac{-l_y^k + r_y^t}{\left(l_y^k - r_y^t\right)^2 + (l_x^k - r_x^t)^2} & \frac{l_x^k - r_x^t}{\left(l_y^k - r_y^t\right)^2 + (l_x^k - r_x^t)^2} \\ \frac{-l_x^k + r_x^t}{\sqrt{\left(l_y^k - r_y^t\right)^2 + (l_x^k - r_x^t)^2}} & \frac{-l_y^k + r_y^t}{\sqrt{\left(l_y^k - r_y^t\right)^2 + (l_x^k - r_x^t)^2}} & \frac{l_x^k - r_x^t}{\sqrt{\left(l_y^k - r_y^t\right)^2 + (l_x^k - r_x^t)^2}} & \frac{l_y^k - r_y^t}{\sqrt{\left(l_y^k - r_y^t\right)^2 + (l_x^k - r_x^t)^2}} \end{pmatrix}$$

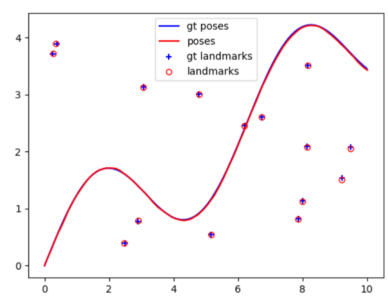
2.2 Build a linear system

Done in python

2.3 Solver

The plots of LU_COLAMD before and after the nonlinear optimization can be seen below:





From the above images it can be seen that LU_COLAMD fits the curve better after the optimization. In comparing it to the linear case, the error cannot be computed by merely following a least squares approach in the non linear case. Rather a least squares approach needs to be applied to every single Taylor expansion and gradient descent term when computing the error for the non linear case. In addition, for the non-linear task an initialization was required and is likely the reason why the graph is such a well fit, however in cases where the initialization is not the best, the non-linear optimization may have difficulty in matching the ground truth.

3 Acknowledgements

- I discussed this assignment, particularly the indexing portions in the code (verbally) with my friend Troy Vicsik (tvicsik)
- 16-833 SLAM Lectures 11-13