16-833: Robot Localization and Mapping, Spring 2021

Homework 3 - Linear and Nonlinear SLAM Solvers

Due: Wednesday April 7, 11:59pm, 2021

Your homework should be submitted as a **typeset PDF file** along with a folder including **code only (no data)**. The PDF must be submitted on **Gradescope**, and code submitted on **Canvas**. If you have questions, post them on Piazza or come to office hours. Please do not post solutions or codes on Piazza. This homework must be done **individually**, and plagiarism will be taken seriously. You are free to discuss and troubleshoot with others, but the code and writeup must be your own. Note that you should list the name and Andrew ID of each student you have discussed with on the first page of your PDF file.

1 2D Linear SLAM

$$\begin{aligned} x^* &= \arg\min_x \Sigma \left\| h_i(x) - z_i \right\|^2_{\mathbf{\Sigma}_i} \\ & \vdots \\ &\approx \arg\min_x \left\| \mathbf{A}x - b \right\|^2, \end{aligned} \tag{1}$$

1.1 Measurement function (10 points)

Given robot poses $\mathbf{r}^t = [r_x^t, r_y^t]^\top$ and $\mathbf{r}^{t+1} = [r_x^{t+1}, r_y^{t+1}]^\top$ at time t and $t+1$, write out the measurement function and its Jacobian. (5 points)

$$h_o(\mathbf{r}^t,\ \mathbf{r}^{t+1}) : \mathbb{R}^4 \to \mathbb{R}^2,$$
$$H_o(\mathbf{r}^t,\ \mathbf{r}^{t+1}) : \mathbb{R}^4 \to \mathbb{R}^{2\times 4}.$$

Similarly, given the robot pose $\mathbf{r}^t = [r_x^t,\ r_y^t]^\top$ at time t and the k-th landmark $\mathbf{l}^k = [l_x^k,\ l_y^k]^\top$. (5 points)

$$h_l(\mathbf{r}^t, \mathbf{l}^k) : \mathbb{R}^4 \to \mathbb{R}^2,$$
$$H_l(\mathbf{r}^t, \mathbf{l}^k) : \mathbb{R}^4 \to \mathbb{R}^{2\times 4}.$$

$$h_o(\mathbf{r}^t,\ \mathbf{r}^{t+1}) = \mathbf{r}^{t+1} - \mathbf{r}^t = \begin{bmatrix} r_x^{t+1} - r_x^t \\ r_y^{t+1} - r_y^t \end{bmatrix}$$
$$H_o(\mathbf{r}^t,\ \mathbf{r}^{t+1}) = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

$$h_l(\mathbf{r}^t,\ \mathbf{l}^k) = \mathbf{l}^k - \mathbf{r}^t = \begin{bmatrix} l_x^k - r_x^t \\ l_y^k - r_y^t \end{bmatrix}$$

$$H_l(\mathbf{r}^t,\ \mathbf{l}^k) = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

1.2 Build a linear system (15 points)

Use the derivation above, please complete the function `create_linear_system` to construct the linear system as described in Eq. 1. (15 points)

See Code

1.3 Solvers (20 points)

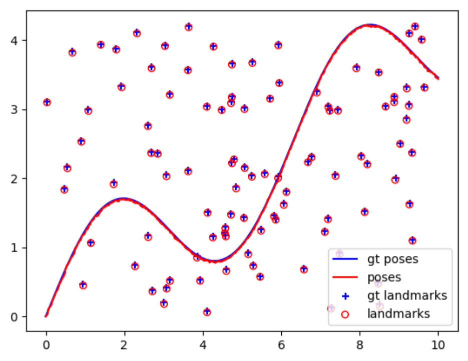
Given the data and the linear system, you are now ready to solve the 2D linear SLAM problem. You are required to implement 5 solvers to solve $Ax = b$ where A is a sparse non-square matrix.

See Code

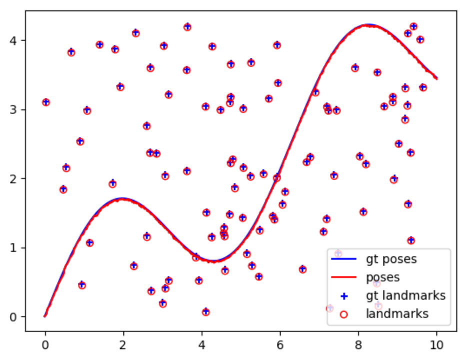
1.4 Exploit sparsity (30 points + 10 points)

Now we want to exploit sparsity in the linear system in QR and LU factorizations.

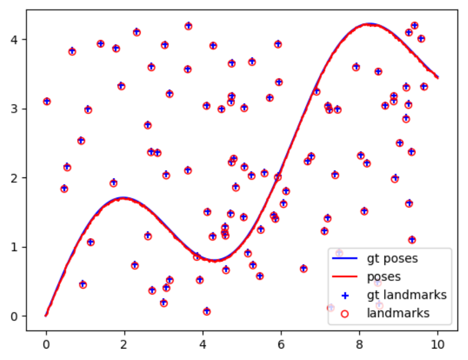
4. `2d_linear.npz` Visualization, efficiency, and analysis for `qr`, `qr_colamd`, `lu`, `lu_colamd`.



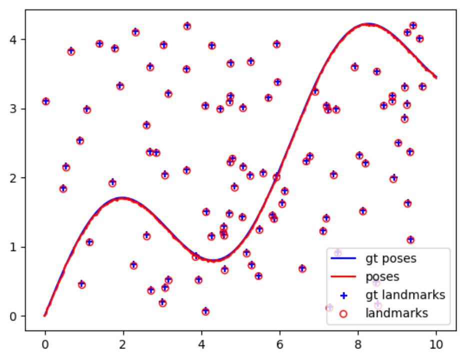
(a) `lu`



(b) `lu_colamd`



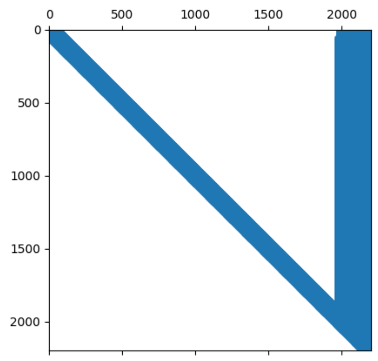
(c) `qr`



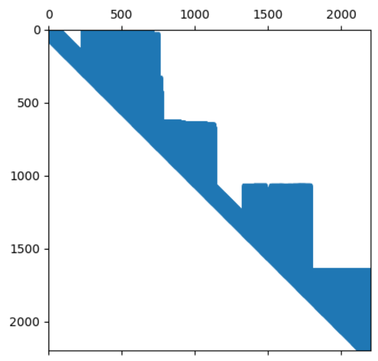
(d) `qr_colamd`

Figure 1: `2d_linear.npz` Visualization

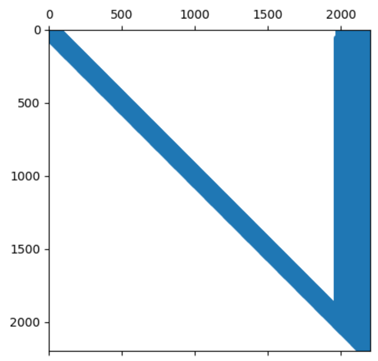
Comparing the four methods' estimation results, there are no significant difference, and this is as expected due to them all solving the same matrix, and being mathematically equivalent in their output. The only difference is how they reach the results.



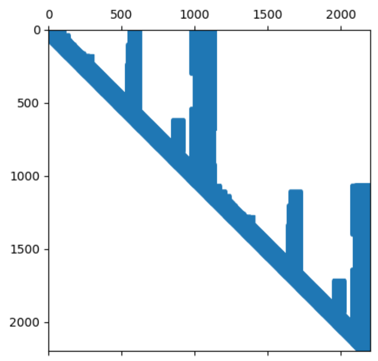
(a) `lu`



(b) `lu_colamd`



(c) `qr`



(d) `qr_colamd`

Figure 2: `2d_linear.npz` R/U Matrix Visualization

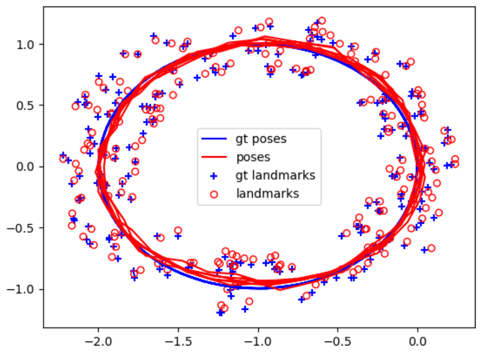
Average Efficiencies

- LU: 0.0414s
- LU colamd: 0.149s
- QR: 0.719s
- QR colamd: 0.736s

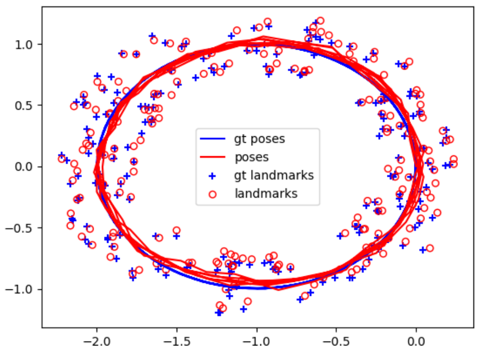
Comparing the efficiency of the four methods on the linear dataset, LU decomposition is faster than QR, and the COLAMD heuristic did not help either in solving it faster. In general, it is expected that LU performs more efficiently than QR, the only catch is the risk of numeric stability. In this case, since the equations are quite simplistic and the data smooth, the results are unsurprising.

As for the COLAMD heuristic, it is not guaranteed to produce a sparser matrix than before the reordering as the problem of producing the sparsest matrix after decomposition is actually NP hard. We can see from figure two the respective matrices are about the same degree of sparsity for before and after COLAMD reordering, and the numeric average efficiencies reflect that.

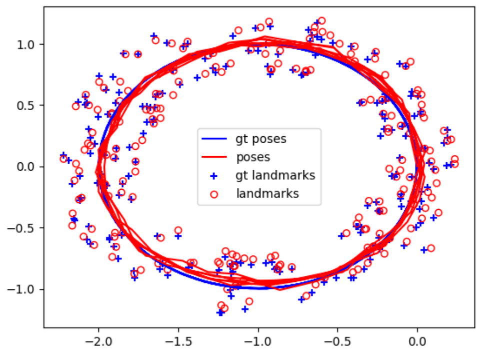
5. `2d_linear_loop.npz` and difference from `2d_linear.npz`.



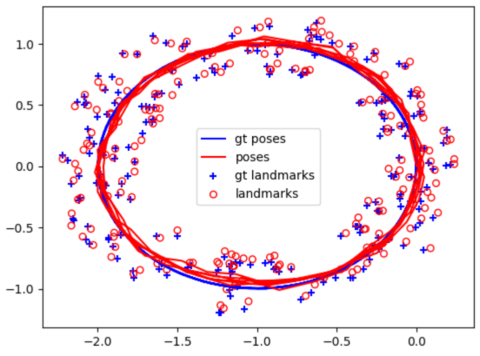
(a) `lu`



(b) `lu_colamd`



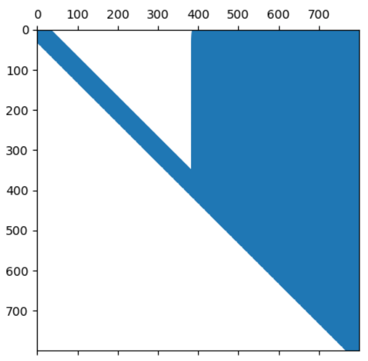
(c) `qr`

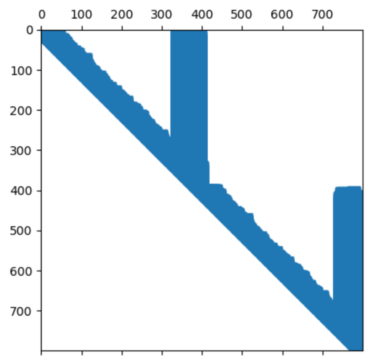


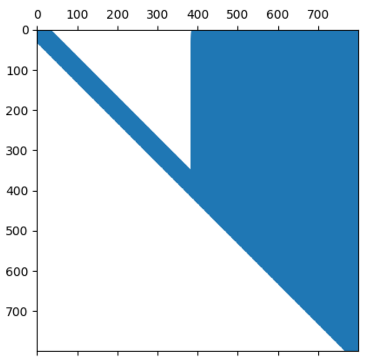
(d) `qr_colamd`

Figure 3: `2d_linear_loop.npz` Visualization

Once again, the results from the four solvers are practically identical as they are mathematically equivalent. The only difference would be due to rounding and small computational discrepancies in the solvers.


(a) `lu`


(b) `lu_colamd`


(c) `qr`

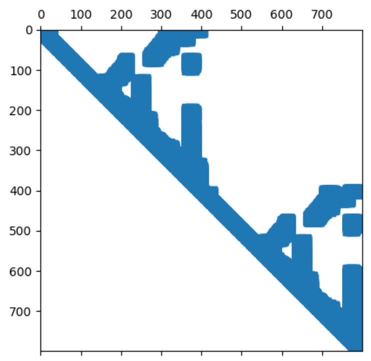

(d) `qr_colamd`

Figure 4: `2d_linear_loop.npz` R/U Matrix Visualization

Efficiencies:

- LU: 0.0302s
- LU colamd:0.00642s
- QR: 0.373s
- QR colamd: 0.0325s

When it comes to efficiencies, the biggest difference between the Looped dataset and the purely linear dataset is in the benefit of the COLAMD heuristic. In this dataset, the heuristic increased efficiency by around 5 10 times, while in the previous one it was negligible. When looking at the R matrix (U for LU) visualizations in Figure 4, it becomes more obvious why. In the non-heuristic version, due to the large number of observations as compared to odometries, the resulting matrix is quite dense. After applying COLAMD, we can see the matrix visually becoming much sparser and easier to solve.

2 2D Nonlinear SLAM

Now you are going to extend the linear SLAM to a nonlinear version in `nonlinear.py`.

The problem set-up is exactly the same as in the linear problem, except we introduce a nonlinear measurement function that returns a bearing angle θ and range d (in robot's body frame, notice we assume that this robot always perfectly facing the x-direction of the global frame), which together describe the vector from the robot to the landmark:

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

2.1 Measurement function (10 points)

In your nonlinear algorithm, you'll need to predict measurements based on the current state estimate.

1. Fill in the functions `odometry_estimation`, `bearing_range_estimation` and `meas_landmark` with corresponding measurement functions. The odometry measurement function is the same linear function we used in the linear SLAM algorithm, while the landmark measurement function is the new nonlinear function introduced in Eq. 2. Please carefully check indices and offsets in the state vector. (5 points)

   See Code

2. Derive the jacobian of the nonlinear landmark function in your writeup
   $$H_l(\mathbf{r}^t, \mathbf{l}^k) : \mathbb{R}^4 \to \mathbb{R}^{2\times 4},$$
   and implement the function `compute_meas_obs_jacobian` to calculate the jacobian at the provided linearization point. (5 points)

$$H_l(\mathbf{r}^t,\ \mathbf{l}^k) = \begin{bmatrix} \frac{l_y^k - r_y^t}{\left(l_x^k - r_x^t\right)^2 + \left(l_y^k - r_y^t\right)^2} & \frac{r_x^t - l_x^k}{\left(l_x^k - r_x^t\right)^2 + \left(l_y^k - r_y^t\right)^2} & \frac{r_y^t - l_y^k}{\left(l_x^k - r_x^t\right)^2 + \left(l_y^k - r_y^t\right)^2} & \frac{l_x^k - r_x^t}{\left(l_x^k - r_x^t\right)^2 + \left(l_y^k - r_y^t\right)^2} \\ (r_x^t - l_x^k)\mathbf{Z} & (r_y^t - l_y^k)\mathbf{Z} & (l_x^k - r_x^t)\mathbf{Z} & (l_y^k - r_y^t)\mathbf{Z} \end{bmatrix}$$

where

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

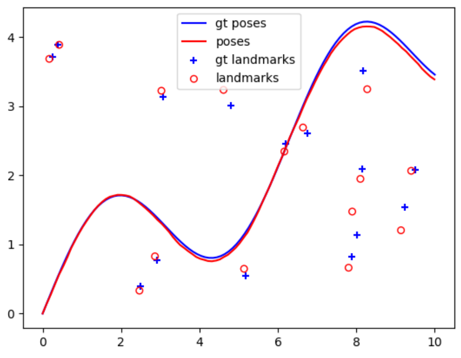
2.2 Build a linear system (15 points)

Use the derivation above, implement `create_linear_system`, now in `nonlinear.py`, to generate the linear system A and b at the current linearization point. (15 points)

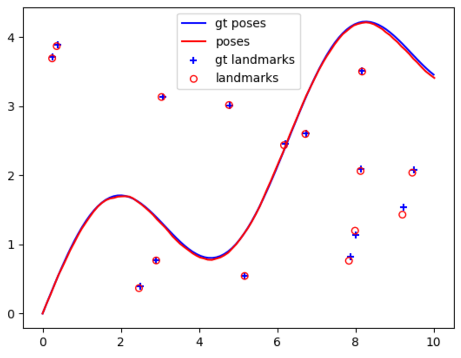
See Code

2.3 Solver (10 points)

Process `2d_nonlinear.npz`. Select one solver you have implemented, and visualize the trajectory and landmarks before and after optimization. Briefly summarize the differences between the optimization process of the linear and the non-linear SLAM problems. (10 points)



(a) Before Optimization



(b) After Optimization

Figure 5: `2d_nonlinear.npz` Visualization - Solved with QR

In figure 5 above, we see the nonlinear solver perfecting a path and landmark state estimation given a strong initial guess. This is a key difference between the linear and the non-linear SLAM problem. The linear problem is solving a single minimization problem and finding the set of state X that best fits to the data. The non-linear problem on the other hand, due to requiring linearization before forming the minimization term, solves an incremental step from the linearization point at a time instead of reaching the solution in one go. Often times this means it requires a good estimation of the states before hand, and the problem of choosing the correct step size has led to many algorithms.