16-833 HW3:
Linear and Nonlinear SLAM Solvers

Yash Jain (Andrew ID - yashjain)

1 2D Linear SLAM

1.1 Measurement Function

Given the robot poses $r^t = [r_x^t, r_y^t]^T$ and $r^{t+1} = [r_x^{t+1}, r_y^{t+1}]^T$ the measurement function is written as-

$$h_o = \begin{bmatrix} h_{o1} \\ h_{o2} \end{bmatrix} = \begin{bmatrix} r_x^{t+1} - r_x^t \\ r_y^{t+1} - r_y^t \end{bmatrix} \tag{1}$$

The Jacobian is written as-

$$H_o = \begin{bmatrix} \frac{\partial h_{o1}}{\partial r_x^t} & \frac{\partial h_{o1}}{\partial r_y^t} & \frac{\partial h_{o1}}{\partial r_x^{t+1}} & \frac{\partial h_{o1}}{\partial r_y^{t+1}} \\ \frac{\partial h_{o2}}{\partial r_x^t} & \frac{\partial h_{o2}}{\partial r_y^t} & \frac{\partial h_{o2}}{\partial r_x^{t+1}} & \frac{\partial h_{o2}}{\partial r_y^{t+1}} \end{bmatrix} \tag{2}$$

$$= \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \tag{3}$$

Similarly, when given robot pose $r^t = [r_x^t, r_y^t]^T$ and landmark pose $l^k = [l_x^k, l_y^k]^T$, the measurement function is written as-

$$h_l = \begin{bmatrix} h_{l1} \\ h_{l2} \end{bmatrix} = \begin{bmatrix} l_x^k - r_x^t \\ l_y^k - r_y^t \end{bmatrix} \tag{4}$$

The Jacobian is written as-

$$H_l = \begin{bmatrix} \frac{\partial h_{l1}}{\partial r_x^t} & \frac{\partial h_{l1}}{\partial r_y^t} & \frac{\partial h_{l1}}{\partial l_x^k} & \frac{\partial h_{l1}}{\partial l_y^k} \\ \frac{\partial h_{l2}}{\partial r_x^t} & \frac{\partial h_{l2}}{\partial r_y^t} & \frac{\partial h_{l2}}{\partial l_x^k} & \frac{\partial h_{l2}}{\partial l_y^k} \end{bmatrix} \tag{5}$$

$$= \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \tag{6}$$

1.2 Build a linear system

Code submitted in a separate file.

1.3 Solvers

Code submitted in a separate file.

1.4 Exploit Sparsity

1.4.1 lu-colamd

Code submitted in a separate file.

1.4.2 Bonus

Not attempted

1.4.3 qr-colamd

Code submitted in a separate file.

1.4.4 2d-linear.npz

To test the efficiency of the methods, each methods was repeated 20 times and the average runtime was calculated. The efficiency in terms of runtime is summarized in the following table-

Method	Average Runtime
Default	0.1566
pinv	3.9327
qr	0.9164
qr-colamd	0.8871
lu	0.0403
lu-colamd	0.1468

Visualizations-
Ground Truth-

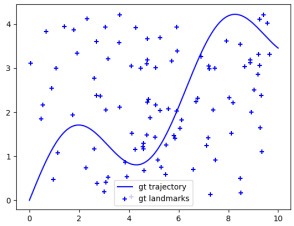

Figure 1: Ground Truth of 2d-linear.npz

pinv-

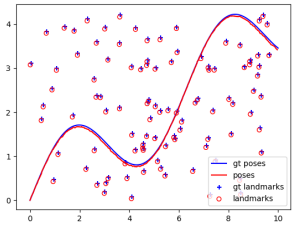

Figure 2: pinv

qr-

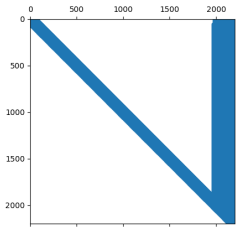

Figure 3: qr sparse matrix

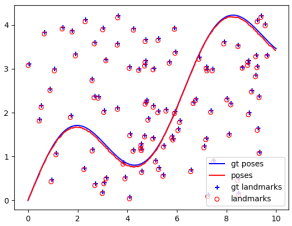

Figure 4: qr

qr colamd-

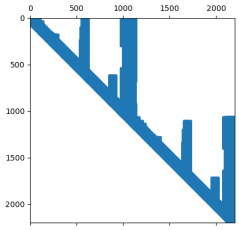

Figure 5: qr-colamd sparse matrix

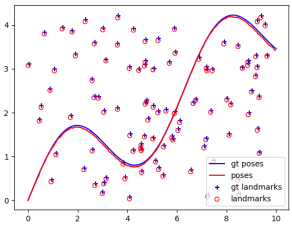

Figure 6: qr-colamd

lu-

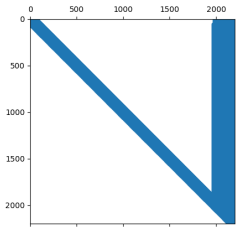

Figure 7: lu sparse matrix

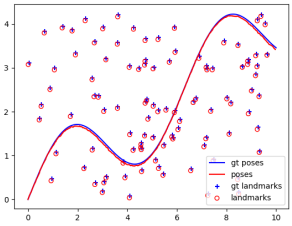

Figure 8: lu

lu colamd-

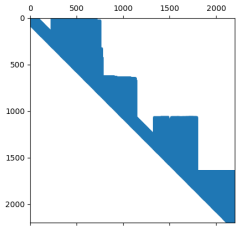

Figure 9: lu-colamd sparse matrix

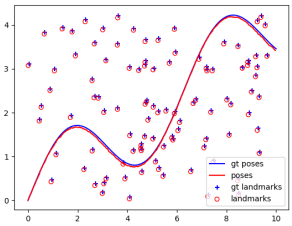

Figure 10: lu-colamd

As can be seen from the efficiency results, the lu solver is the fasted followed by lu-colamd, qr-colamd, qr and pinv in order. We know that pinv is the slowest method because it does not try to utilize sparsity. Among the methods which take advantage of sparsity, the lu solver is always faster than the qr solver due to the fact that the time complexity of lu is $\frac{2}{3}m^3$ and the time complexity of qr is $\frac{4}{3}m^3$ where m is the order of the matrix. The colamd method is used to rearrange the matrix to avoid dense accumulation in the last few column and increase sparsity. In the case of 2d-linear.npz data we see a increase in efficiency due to the colamd reordering in the qr solver but in case of the lu solver, the reordering turns out to be more expensive than regular solving.

1.4.5 2d-linear-loop.npz

To test the efficiency of the methods, each methods was repeated 20 times and the average runtime was calculated. The efficiency in terms of runtime is summarized in the following table-

Method	Average Runtime
Default	0.0075
pinv	0.3529
qr	0.4992
qr-colamd	0.0431
lu	0.0312
lu-colamd	0.0077

Visualizations-
Ground Truth-

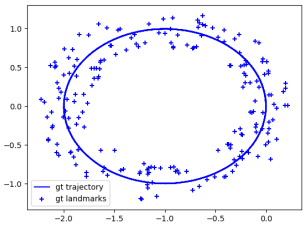

Figure 11: Ground Truth of 2d-linear-loop.npz

pinv-

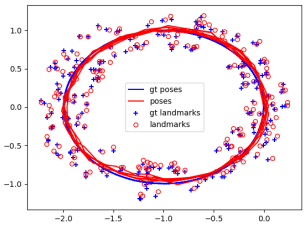

Figure 12: pinv

qr-

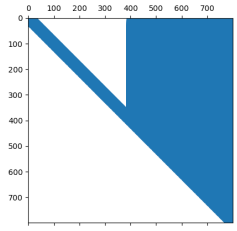

Figure 13: qr sparse matrix

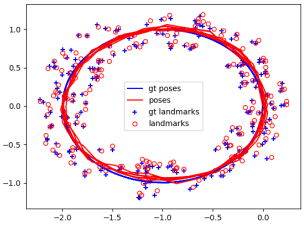

Figure 14: qr

qr colamd-

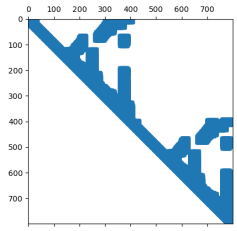

Figure 15: qr-colamd sparse matrix

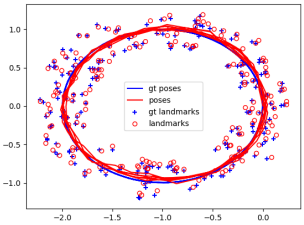

Figure 16: qr-colamd

lu-

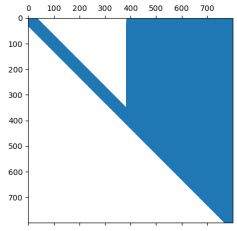

Figure 17: lu sparse matrix

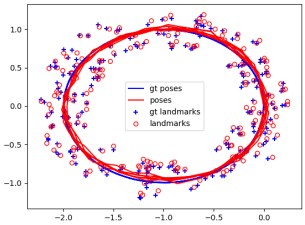

Figure 18: lu

lu colamd-

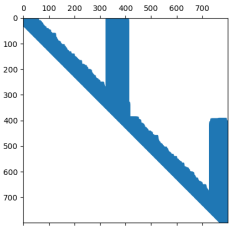

Figure 19: lu-colamd sparse matrix

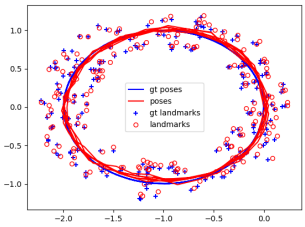

Figure 20: lu-colamd

As can be seen from the efficiency results, the lu-colamd solver is the fasted followed by lu, qr-colamd, pinv and qr in order. Here, keeping with the trend, the lu is faster than qr but in this case we can see an improvement in efficiency due to the colamd reordering in both these solvers. Also, it turns out that pinv is faster than qr solver in this case. So in this case, the reordering is less computationally expensive than the regular solvers. We see an improvement in runtime when compared with 2d-linear.npz data due to the fact the matrices in the case of 2d-linear-loop.npz are more spare and do not have any dense grouping as in the earlier case.

2 2D Nonlinear SLAM

2.1 Measurement function

2.1.1 Estimation Code

Code submitted in a separate file.

2.1.2 Jacobian

Given the measurement function-

$$h_l(r^t, l^k) = \begin{bmatrix} arctan2(l_y^k - r_y^t, l_x^k - r_x^t) \\ \sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} \end{bmatrix} = \begin{bmatrix} \theta \\ d \end{bmatrix} \tag{7}$$

The Jacobian is written as-

$$H_l = \begin{bmatrix} \frac{\partial h_{l1}}{\partial r_x^t} & \frac{\partial h_{l1}}{\partial r_y^t} & \frac{\partial h_{l1}}{\partial l_x^k} & \frac{\partial h_{l1}}{\partial l_y^k} \\ \frac{\partial h_{l2}}{\partial r_x^t} & \frac{\partial h_{l2}}{\partial r_y^t} & \frac{\partial h_{l2}}{\partial l_x^k} & \frac{\partial h_{l2}}{\partial l_y^k} \end{bmatrix} \tag{8}$$

$$= \begin{bmatrix} \frac{l_y^k - r_y^t}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} & \frac{-(l_x^k - r_x^t)}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} & \frac{-(l_y^k - r_y^t)}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} & \frac{l_x^k - r_x^t}{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2} \\ \frac{-(l_x^k - r_x^t)}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}} & \frac{-(l_y^k - r_y^t)}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}} & \frac{l_x^k - r_x^t}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}} & \frac{l_y^k - r_y^t}{\sqrt{(l_x^k - r_x^t)^2 + (l_y^k - r_y^t)^2}} \end{bmatrix} \tag{9}$$

2.2 Build a Linear System

Code submitted in a separate file.

2.3 Solver

I selected the lu solver for this case and the visualizations are as follows-

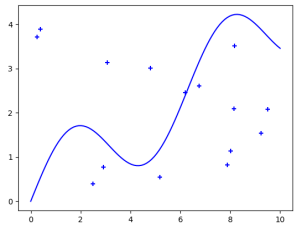

Figure 21: Ground Truth

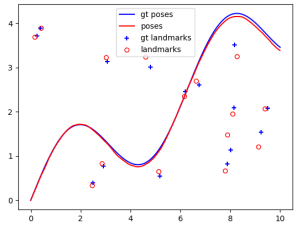

Figure 22: Before Optimization

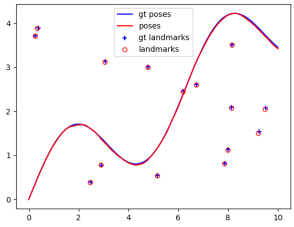

Figure 23: After Optimization with lu solver

The key differences in the optimization process of linear and non linear SLAM problems is that the linear models solves directly for X where as the non linear model solves for Δ, the difference between the estimated and measured values. Apart from this, the non linear model works by iterative minimization of the error. The non linear method uses a combination of Taylor series approximation and convex optimization methods like steepest descent to achieve a solution in the admissible error tolerance when the system is linearized at point.