**HW3 – SLAM Solvers**

1. **Linear model**
2. Default: Time to solve: 0.029s

A graph of a line graph

AI-generated content may be incorrect.

Fig 1: 2d\_linear.npz with default solver

1. Pseudo Inverse: Time to solve: 2.07s

A graph of a line graph

AI-generated content may be incorrect.

Fig 2: 2d\_linear.npz with pinv

1. LU: Time to solve: 0.019s

A blue line with numbers on a white background

AI-generated content may be incorrect.

Fig 3. U (upper triangular) Matrix from LU decomposition

A graph of a line

AI-generated content may be incorrect.

Fig 4: 2d\_linear.npz with LU

1. LU-COLAMD: time to solve: 0.032s

A blue and white graph

AI-generated content may be incorrect.

Fig 5: U (Upper triangular matrix)

A graph of a line graph

AI-generated content may be incorrect.

Fig 6: 2d\_linear\_npz with LU + COLAMD

e) QR: Time to solve: 0.39s

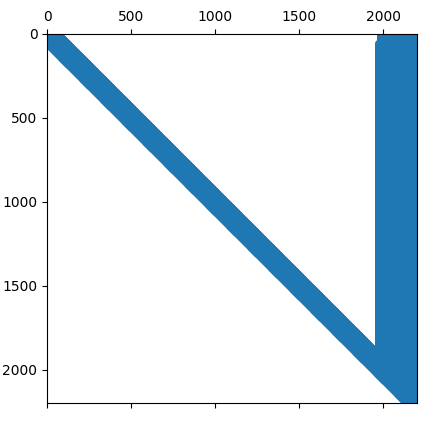


Fig 7. R matrix from QR factorization

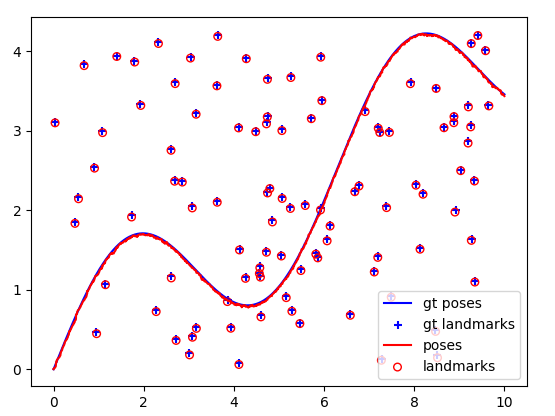


Fig 8. 2d\_linear\_npz with QR factorization

d) QR with COLAMD: Time to solve: 0.33 s

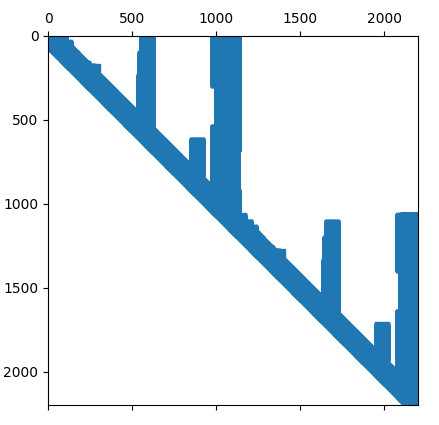


Fig 9. R matrix from QR factorization + COLAMD

A graph of a graph

AI-generated content may be incorrect.

Fig 10. 2d\_linear\_npz with QR factorization + COLAMD

1. **Linear model with loop data**
2. Default: Time to solve: 0.016s

A diagram of a circle with red and blue circles

AI-generated content may be incorrect.

Fig 11: 2d\_linear\_loop.npz with Default solver

1. Pseudo-inverse: Time to solve: 0.45s

A red and blue circles with white text

AI-generated content may be incorrect.

Fig 12: 2d\_linear\_loop.npz with Pseudo inverse solver

1. LU: Time to solve: 0.04s

A blue and white graph

AI-generated content may be incorrect.

Fig 13: U (upper triangular matrix from LU decomposition)

A diagram of a circle with red and blue dots

AI-generated content may be incorrect.

Fig 14: 2d\_linear\_loop.npz with LU solver

1. LU-COLAMD: Time to solve: 0.009s

A graph with a blue line

AI-generated content may be incorrect.

Fig 15: U (upper triangular matrix from LU-COLAMD decomposition)

A diagram of a circle with red and blue circles

AI-generated content may be incorrect.

Fig 16: 2d\_linear\_loop.npz with LU-COLAMD

e) QR: Time to solve: 0.25 s

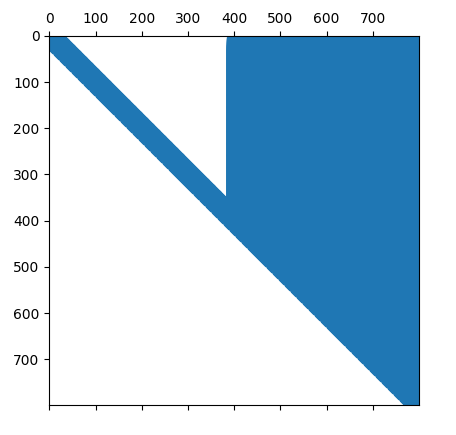


Fig 17: R Matrix from QR

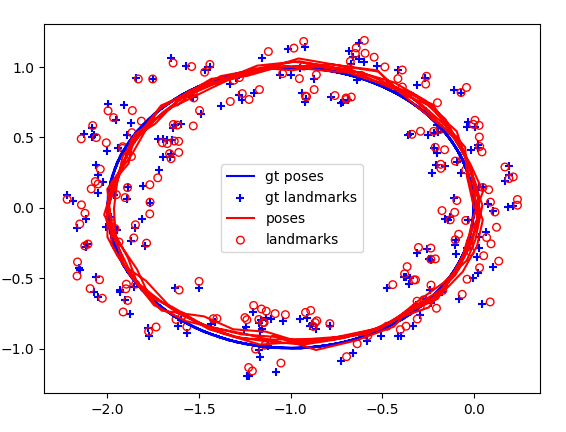


Fig 18. 2d\_linear\_loop with QR

f) QR + COLAMD: Time to Solve: 0.019s

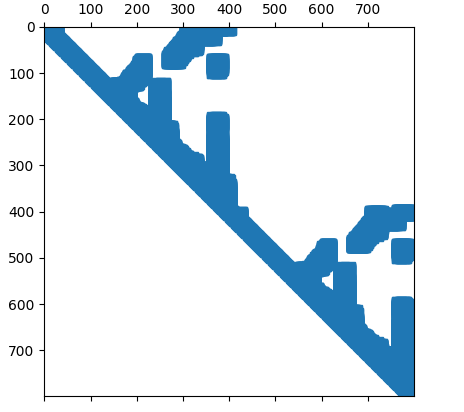


Fig 19: R Matrix from QR + COLAMD

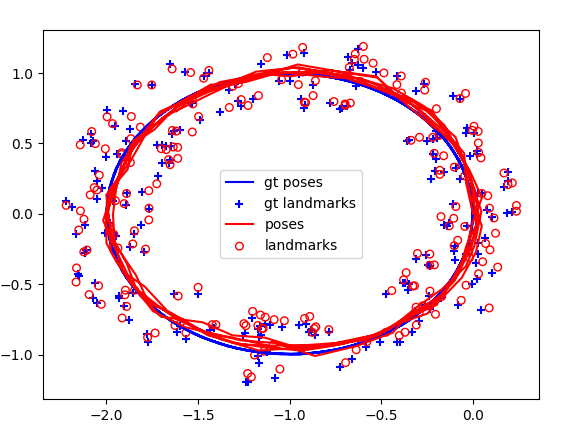


Fig 20. 2d\_lineaFigure 20. QR + COLAMD

1. **NonLinear model**

A graph of a graph with red and blue lines

AI-generated content may be incorrect.

Fig 21: Before optimization

1. Default:

A graph with red and blue lines

AI-generated content may be incorrect.

Fig 22: Default solver, After optimization

1. Pseudo Inverse

A graph with red and blue lines

AI-generated content may be incorrect.

Fig 23: Pseudo Inverse solver, after optimization

1. LU

A graph with red and blue lines

AI-generated content may be incorrect.

Fig 24: LU Solver, after optimization

1. LU- COLAMD

A graph with red and blue lines and white text

AI-generated content may be incorrect.

Fig 25:LU-COLAMD After optimization

e) QR

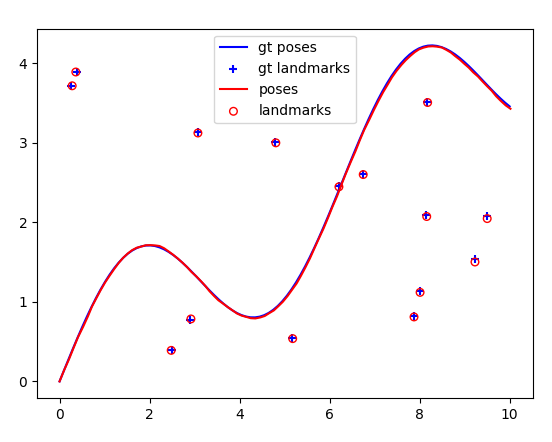
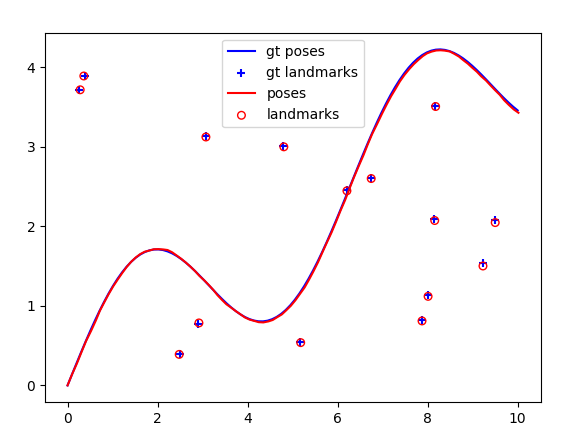


Fig 26: QR after optimization

f) QR COLAMD

Fig 27: QR after opt Fig 27: QR with COLAMD after optimization

**Discussion**

**1.4, part 5**

For the linear data, LU solved the optimization in 0.019 s, while LU + COLAMD took slightly longer (0.032s). QR took 0.39s, and QR with COLAMD took 0.33 s.

LU without COLAMD was faster than LU + COLAMD because the U matrix seen in Figure 3 was already quite sparse. However, QR with COLAMD was slightly faster than without. This might be because I switched to my Linux dual boot to run QR.

**2.3**

There was a noticeable difference in the loop data, since the triangular matrices were denser. LU took 0.04 seconds to solve, while LU + COLAMD took 0.009 seconds to solve. QR took 0.25 seconds to solve, while QR + COLAMD took 0.019 seconds to solve. In this case, the triangular matrices were quite dense (higher correlation between states and landmarks since robot is travelling in loops), so using COLAMD helped improve time efficiency considerably.

In the linear case, we used the observations and odometry data to optimize the robot pose and landmark position in a single step, by solving a least squares problem.

In the non linear case, we are setting up and solving a minimization problem, where we are minimizing the residual / error between observed and estimated robot pose and landmark observation. We are also linearizing our system at the x estimated in the previous iteration. (We initialize the x with a guess). This residual minimization is then ran for 10 iterations.