NON-LINEAR SLAM

\[ A = L L^T \]
Non-Linear SLAM

- Robot Trajectory: a pose graph

\[ z_0 = f_0(x_0, x_1) \]
\[ z_1 = f_1(x_1, x_2) \]
\[ z_2 = f_2(x_2, x_3) \]
\[ z_3 = f_3(x_3, x_4) \]
\[ z_4 = f_4(x_0, x_5) \]
\[ z_5 = f_5(x_2, x_5) \]
\[ z_6 = f_6(x_3, x_5) \]
Non-Linear SLAM: Review

- Stacking observations
- Observations want $Jd = r$
  - Over-determined. Each observation associated with covariance
- Minimize the cost function:
  \[
  \chi^2 = (z(x) - z)^T \Sigma^{-1}_z (z(x) - z)
  \approx (J \Delta x + \hat{z} - z)^T \Sigma^{-1}_z (J \Delta x + \hat{z} - z)
  \]
- Manipulate a bit:
  \[
  (J^T \Sigma^{-1}_z J) d = J^T \Sigma^{-1}_z r
  \]
  \[
  \Sigma_z = \text{Cov. of obs 1}
  \]
  \[
  \text{Cov. of obs 2}
  \]
  \[
  = \text{Linearized observation constraint equations}
  \]
  \[
  \text{Over-constraining observations}
  \]
  \[
  \text{State variables}
  \]
Structure of $J$

- **Relationship to graph**

\[
J_x^0 = \begin{bmatrix}
* & * & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
J_x^1 = \begin{bmatrix}
0 & * & * & 0 & 0 & 0
\end{bmatrix}
\]

\[
J_x^2 = \begin{bmatrix}
0 & 0 & * & * & 0 & 0
\end{bmatrix}
\]

\[
J_x^3 = \begin{bmatrix}
0 & 0 & 0 & * & * & 0
\end{bmatrix}
\]

\[
J_x^4 = \begin{bmatrix}
* & 0 & 0 & 0 & 0 & *
\end{bmatrix}
\]

\[
J_x^5 = \begin{bmatrix}
0 & 0 & * & 0 & 0 & *
\end{bmatrix}
\]

\[
J_x^6 = \begin{bmatrix}
0 & 0 & 0 & * & 0 & *
\end{bmatrix}
\]

Lots of zeros

- $z_0 = f_0(x_0, x_1)$
- $z_1 = f_1(x_1, x_2)$
- $z_2 = f_2(x_2, x_3)$
- $z_3 = f_3(x_3, x_4)$
- $z_4 = f_4(x_0, x_5)$
- $z_5 = f_5(x_2, x_5)$
- $z_6 = f_6(x_3, x_5)$
Our observations are independent:
- Observation covariance matrix is \textit{block diagonal}
- What about its inverse?
  - \textit{also} block diagonal

Is independence of observations a reasonable assumption?
Structure of $J^T \Sigma^{-1} J$

- Because $\Sigma^{-1}$ is block diagonal, we have:

$$J^T \Sigma^{-1} J = \sum_i J_i^T \Sigma_i^{-1} J_i$$

- This is also evident from our original cost function: we’re minimizing the sum of the squared errors of each observation.
Structure of $J^T \sum_z^{-1} J$

$$J^T \sum_z^{-1} J = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$J_i^x = \begin{bmatrix}
* & * & 0 & 0 & 0 & 0 \\
0 & * & * & 0 & 0 & 0 \\
0 & 0 & * & * & 0 & 0 \\
0 & 0 & 0 & * & * & 0 \\
* & 0 & 0 & 0 & 0 & * \\
0 & 0 & * & 0 & 0 & * \\
0 & 0 & 0 & * & 0 & *
\end{bmatrix}$$
Structure of $J^T \Sigma^{-1} J$

- Key observations:
  - The information matrix remains sparse
  - (Even though the covariance matrix becomes dense)
  - Directly encodes connectivity of pose graph
    - I.e., the adjacency matrix of the Bayes Net

- Sparsity is GOOD— we can exploit it!
Why is sparsity useful?

- Sparse matrix data structures
  - Make computation a function of # of non-zero elements
  - Use memory proportional to # of non-zero elements
Sparse Matrix Representation

- CSR: Compressed Sparse Row
- \( x = [ a \ b \ 0 \ 0 \ 0 \ c \ 0 \ 0 \ d \ e \ 0 \ 0 \ 0 \ 0 \ 0 \ f ] \)

\( \Rightarrow \)

- \( x = \{ \text{indices} = \{0, 1, 5, 8, 9, 15\}, \text{values} = \{a, b, c, d, e, f\} \} \)
double dotProduct(CSRVec a, CSRVec b) {
    int aidx = 0, bidx = 0;

    while (aidx < a.nz && bidx < b.nz) {
        int ai = a.indices[aidx], bi = b.indices[bidx];

        if (ai == bi) {
            acc += a.values[aidx]*b.values[bidx];
            aidx++;
            bidx++;
            continue;
        }

        if (ai < bi) aidx++;
        else bidx++;
    }

    return acc;
}

a = [ a b 0 0 0 c 0 0 d e 0 0 0 0 0 f ]

a = { indices = {0, 1, 5, 8, 9, 15 },
       values = {a, b, c, d, e, f } }

b = [ 0 0 0 g h i 0 0 0 0 0 0 0 0 0 0 0 0 ]

b = { indices = {3, 4, 5},
       values = {g, h, i} }
We don’t actually do Gaussian Elimination
- G.E. computes the inverse
- So we have to store the inverse!
- Numerical stability issues

We use Cholesky decomposition instead
- Cholesky decomposition works for all symmetric + SPD
Cholesky Decomposition (Review)

\[
\begin{align*}
A &= L \cdot L^T \\
\begin{bmatrix}
16 & 4 & 8 \\
4 & 37 & 20 \\
8 & 20 & 14
\end{bmatrix} &=
\begin{bmatrix}
4 \\
1 \\
2
\end{bmatrix}
\begin{bmatrix}
4 & 1 & 2 \\
6 & 3 \\
1
\end{bmatrix}
\end{align*}
\]
Backsolve

\[
\begin{align*}
L &: \begin{pmatrix}
4 & 1 & 2 \\
1 & 6 & 3 \\
2 & 3 & 1 \\
\end{pmatrix} \\
&= \begin{pmatrix}
4 & 12 & 3 \\
1 & 21 & 3 \\
2 & 3 & 90 \\
\end{pmatrix} \\
\end{align*}
\]

\[
v &= \begin{pmatrix}
48 \\
138 \\
90 \\
\end{pmatrix}
\]
Backsolve

\[ L^T \cdot x = v \]

\[
\begin{bmatrix}
4 & 1 & 2 \\
6 & 3 & 1
\end{bmatrix}
\begin{bmatrix}
1 \\
2 \\
3
\end{bmatrix} =
\begin{bmatrix}
48 \\
138 \\
90
\end{bmatrix}
\]
Cholesky Decomposition

- Strategy: incrementally triangularize \( A \) starting from the top.

- Which of these operations can we do efficiently on sparse matrices?

- Notice: if \( A \) is sparse, output will tend to be sparse
  - But there’s “fill-in” related to how each variable is connected to other variables.
Solving:
- Let: $v = L^T x$. Solve $Lv = b$ for $v$.
  - How?
- $L^T x = v$
  - Solve for $x$
Marginalization (Gaussian Elimination)

- The order of the variables in the matrix matters.

\[ A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 10 & 0 & 0 \\ 3 & 0 & 20 & 0 \\ 4 & 0 & 0 & 30 \end{bmatrix} \]

- Variable order == which variables are we eliminating?
  - Left-most variable: write all other variables so they are no longer a function of that variable.
  - Variables that are related to many other variables create more fill in.

- What if we changed variable ordering?
  - Let’s try it!
Suppose we want to solve *this* system

- Each node is a variable, which can be written in terms of the variables it’s connected to... just like pose graph.
If we remove “Mind” first, we add many edges.
Finding a good variable ordering

- Marginalizing-out variables causes fill in
- If we marginalize-out early we might commit to fill-in that we could otherwise avoid.
  - Old strategy: marginalize out all landmarks first, then marginalize out poses.

- In short: keeping around extra “state”:
  - might allow us to find a better variable order later
  - reduce fill-in $\rightarrow$ sparsity $\rightarrow$ faster

- What is a good variable ordering?
Maximum Degree Ordering (Bad)

And it just keeps getting worse!
Minimum Degree Example
Minimum Degree Ordering

while nodes remain
  find node \( i \) with smallest node degree
  ordering = \{ ordering, \( i \} \)
  for each \( j \) in neighbors(\( i \))
    add all of neighbors(\( i \)) to neighbors(\( j \))
  end
  remove node \( i \)
end

return ordering

- Complexity?
- Optimal (minimum fill-in)?
SqrtSAM (SLAM algorithm)

- Sparsity, Sparsity, Sparsity
- Keep track of full robot trajectory in state
- Find a good variable reordering
- Sparse Cholesky decomposition + backsolve
Project Presentations

- (12/9) Teams 15-18 present this Wednesday
- (12/14) Teams 19-22 present on Monday
- (12/16) Project writeups due Wednesday (~5p)

- Presentations should be around 20m each
- Video clips suggested, live demos are great but also tricky
- Not a final presentation, but show solid progress and excellent planning