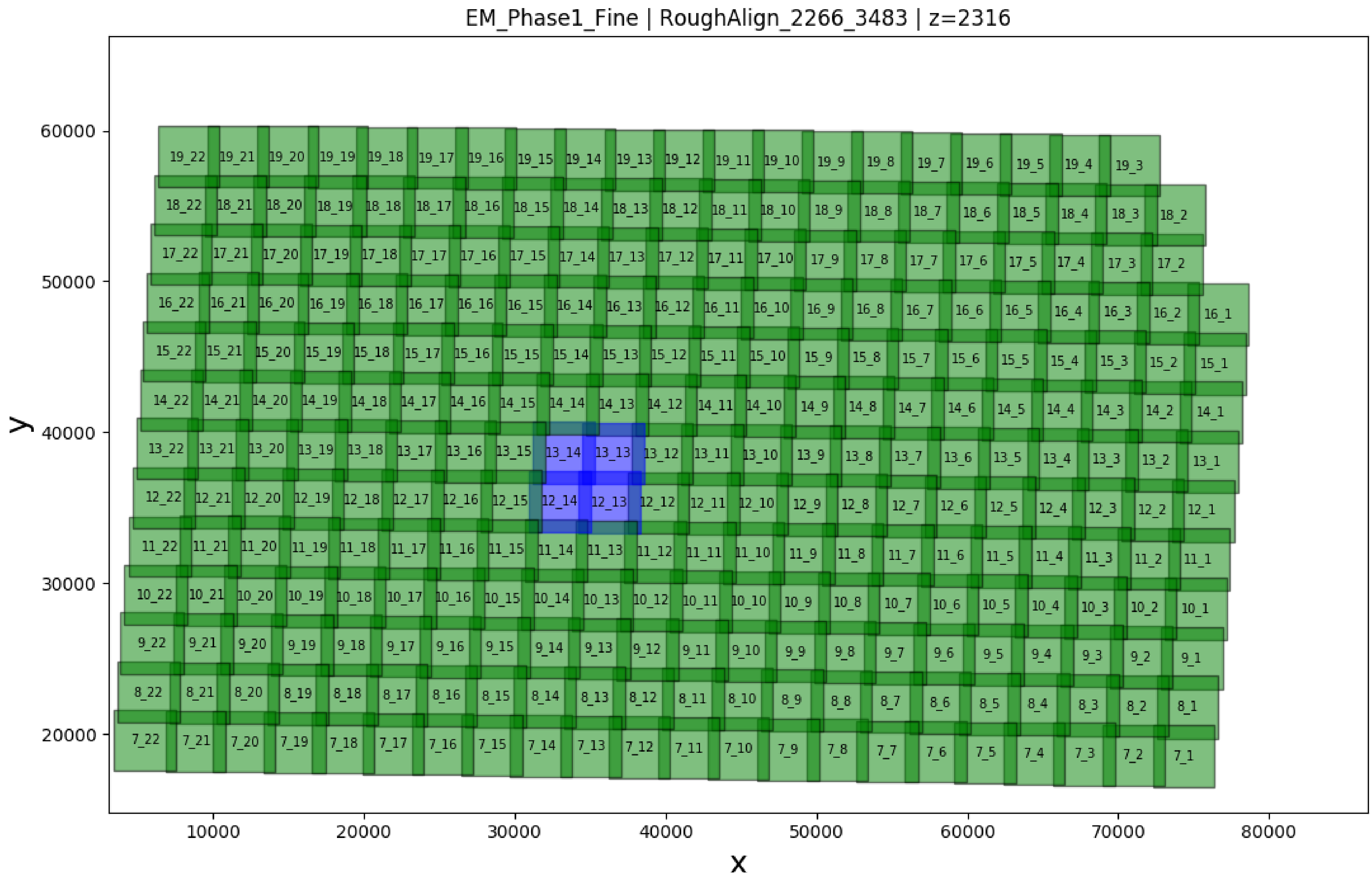
4 Tile example and more

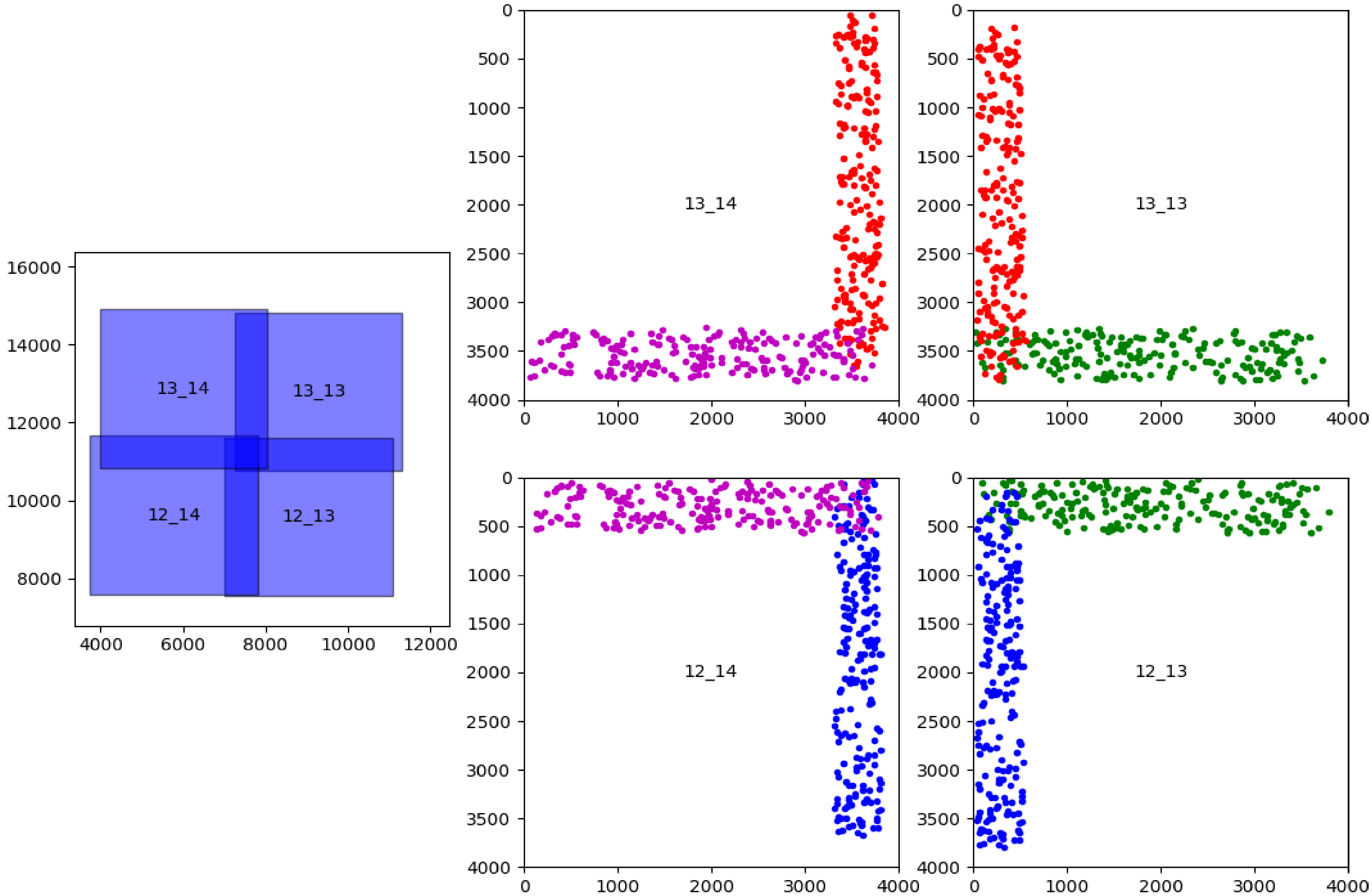
Purpose:

- Understand EM_aligner, MATLAB code base for assembling and solving EM-connectomics fine alignment problems.
- Understand the linear algebra going into these solutions.
- Develop plan for scaling the solver for large chunks of mm^2 sections.

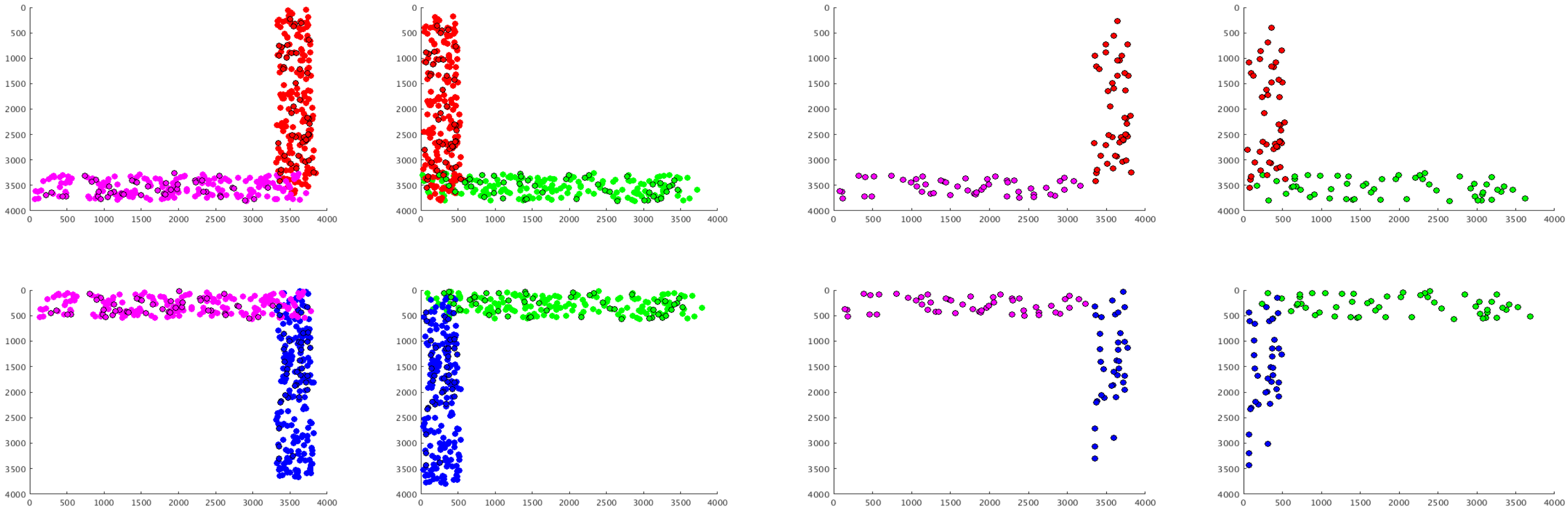
Source of Data



4 Tiles and Point Matches

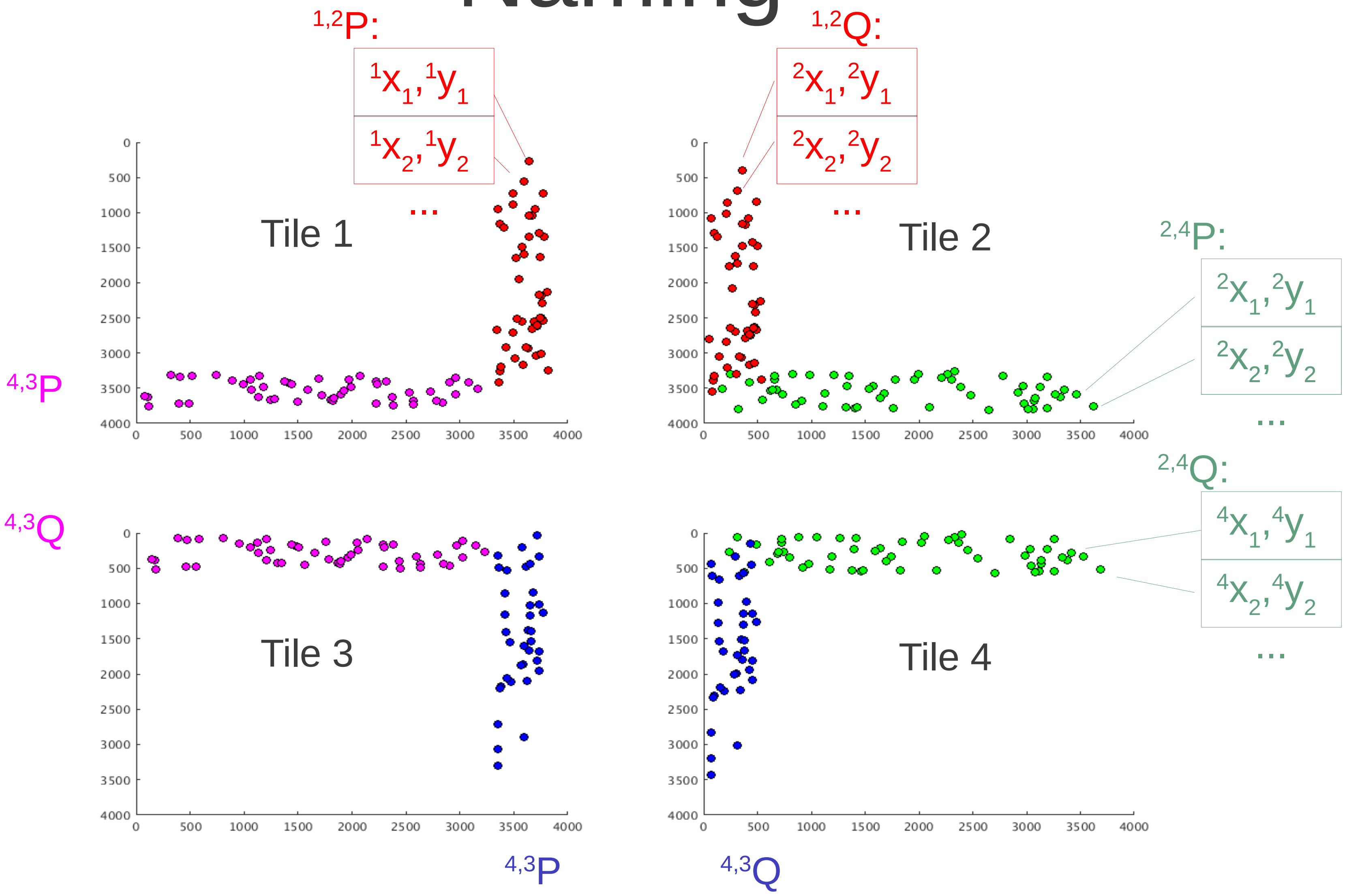


Filtering



- EM_aligner includes filtering to limit the number of included point matches.
- In this example, the filtering reduced the # from 200 points per tile pair to ~50.

Naming



1,2P:
1x1,1y1
1x2,1y2
...
1,2Q:
2x1,2y1
2x2,2y2
...
Tile 1
Tile 2
2,4P:
2x1,2y1
2x2,2y2
...
4,3P
4,3Q
2,4Q:
4x1,4y1
4x2,4y2
...
Tile 3
Tile 4
4,3P
4,3Q
0 500 1000 1500 2000 2500 3000 3500 4000

Affine Transformation

The point match coordinates in each tile will be transformed according to an affine transformation:

$$^1u_1 = a_1 {}^1x_1 + b_1 {}^1y_1 + c_1$$
$$^1v_1 = d_1 {}^1x_1 + e_1 {}^1y_1 + f_1$$

The goal of alignment between two tiles is to meet the criteria:

$$^1u_1 = {}^2u_1$$
$$^1v_1 = {}^2v_1$$
$$^1u_2 = {}^2u_2$$
$$^1v_2 = {}^2v_2$$

...

Explicitly:

$$a_1\,{}^1x_1 + b_1\,{}^1y_1 + c_1 = a_2\,{}^2x_1 + b_2\,{}^2y_1 + c_2$$
$$d_1\,{}^1x_1 + e_1\,{}^1y_1 + f_1 = d_2\,{}^2x_1 + e_2\,{}^2y_1 + f_2$$

Matrix Assembly

$$a_1\,{}^1x_1 + b_1\,{}^1y_1 + c_1 = a_2\,{}^2x_1 + b_2\,{}^2y_1 + c_2$$
$$d_1\,{}^1x_1 + e_1\,{}^1y_1 + f_1 = d_2\,{}^2x_1 + e_2\,{}^2y_1 + f_2$$

which is the same as:

$$\begin{bmatrix} -{}^1x_1 & -{}^1y_1 & -1 & {}^2x_1 & {}^2y_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -{}^1x_1 & -{}^1y_1 & -1 & {}^2x_1 & {}^2y_1 & 1 \end{bmatrix} \times \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \\ d_1 \\ e_1 \\ f_1 \\ d_2 \\ e_2 \\ f_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

And for multiple pairs of points:

$$\begin{bmatrix} -{}^1x_1 & -{}^1y_1 & -1 & {}^2x_1 & {}^2y_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -{}^1x_2 & -{}^1y_2 & -1 & {}^2x_2 & {}^2y_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -{}^1x_3 & -{}^1y_3 & -1 & {}^2x_3 & {}^2y_3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -{}^1x_4 & -{}^1y_4 & -1 & {}^2x_4 & {}^2y_4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & \dots & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & -{}^1x_1 & -{}^1y_1 & -1 & {}^2x_1 & {}^2y_1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -{}^1x_2 & -{}^1y_2 & -1 & {}^2x_2 & {}^2y_2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -{}^1x_3 & -{}^1y_3 & -1 & {}^2x_3 & {}^2y_3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -{}^1x_4 & -{}^1y_4 & -1 & {}^2x_4 & {}^2y_4 & 1 \\ & & & & & & & & \dots & & & \end{bmatrix} \times \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \\ d_1 \\ e_1 \\ f_1 \\ d_2 \\ e_2 \\ f_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \\ 0 \\ 0 \\ 0 \\ \dots \end{bmatrix}$$

$$\begin{bmatrix} -{}^1x_1 & -{}^1y_1 & -1 & 0 & 0 & 0 & {}^2x_1 & {}^2y_1 & 1 & 0 & 0 & 0 \\ -{}^1x_2 & -{}^1y_2 & -1 & 0 & 0 & 0 & {}^2x_2 & {}^2y_2 & 1 & 0 & 0 & 0 \\ -{}^1x_3 & -{}^1y_3 & -1 & 0 & 0 & 0 & {}^2x_3 & {}^2y_3 & 1 & 0 & 0 & 0 \\ -{}^1x_4 & -{}^1y_4 & -1 & 0 & 0 & 0 & {}^2x_4 & {}^2y_4 & 1 & 0 & 0 & 0 \\ & & & \dots & & & & & & & & \\ 0 & 0 & 0 & -{}^1x_1 & -{}^1y_1 & -1 & 0 & 0 & 0 & {}^2x_1 & {}^2y_1 & 1 \\ 0 & 0 & 0 & -{}^1x_2 & -{}^1y_2 & -1 & 0 & 0 & 0 & {}^2x_2 & {}^2y_2 & 1 \\ 0 & 0 & 0 & -{}^1x_3 & -{}^1y_3 & -1 & 0 & 0 & 0 & {}^2x_3 & {}^2y_3 & 1 \\ 0 & 0 & 0 & -{}^1x_4 & -{}^1y_4 & -1 & 0 & 0 & 0 & {}^2x_4 & {}^2y_4 & 1 \\ & & & & & & & & \dots & & & \end{bmatrix} \times \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ e_1 \\ f_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \\ e_2 \\ f_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \\ 0 \\ 0 \\ 0 \\ \dots \end{bmatrix}$$

Reorder to keep all tile 1 coordinates in a column block

Matrix Assembly

$$\begin{bmatrix} -{}^{1}x_1 & -{}^{1}y_1 & -1 & 0 & 0 & 0 & {}^{2}x_1 & {}^{2}y_1 & 1 & 0 & 0 & 0 \\ -{}^{1}x_2 & -{}^{1}y_2 & -1 & 0 & 0 & 0 & {}^{2}x_2 & {}^{2}y_2 & 1 & 0 & 0 & 0 \\ -{}^{1}x_3 & -{}^{1}y_3 & -1 & 0 & 0 & 0 & {}^{2}x_3 & {}^{2}y_3 & 1 & 0 & 0 & 0 \\ -{}^{1}x_4 & -{}^{1}y_4 & -1 & 0 & 0 & 0 & {}^{2}x_4 & {}^{2}y_4 & 1 & 0 & 0 & 0 \\ & & & \dots & & & & & & & & \\ 0 & 0 & 0 & -{}^{1}x_1 & -{}^{1}y_1 & -1 & 0 & 0 & 0 & {}^{2}x_1 & {}^{2}y_1 & 1 \\ 0 & 0 & 0 & -{}^{1}x_2 & -{}^{1}y_2 & -1 & 0 & 0 & 0 & {}^{2}x_2 & {}^{2}y_2 & 1 \\ 0 & 0 & 0 & -{}^{1}x_3 & -{}^{1}y_3 & -1 & 0 & 0 & 0 & {}^{2}x_3 & {}^{2}y_3 & 1 \\ 0 & 0 & 0 & -{}^{1}x_4 & -{}^{1}y_4 & -1 & 0 & 0 & 0 & {}^{2}x_4 & {}^{2}y_4 & 1 \\ & & & & & & & & \dots & & & \end{bmatrix} \times \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ e_1 \\ f_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \\ e_2 \\ f_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \\ 0 \\ 0 \\ 0 \\ \dots \end{bmatrix}$$

Block-wise shorthand:

$$\begin{bmatrix} {}^{1,2}P & -{}^{1,2}Q \end{bmatrix} \times \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

2 tiles:

$$\begin{bmatrix} {}^{1,2}P & -{}^{1,2}Q \end{bmatrix} \times \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

3 tiles:

$$\begin{bmatrix} {}^{1,2}P & -{}^{1,2}Q & 0 \\ {}^{1,3}P & 0 & -{}^{1,3}Q \end{bmatrix} \times \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

4 tiles:

$$\begin{bmatrix} {}^{1,2}P & -{}^{1,2}Q & 0 & 0 \\ {}^{1,3}P & 0 & -{}^{1,3}Q & 0 \\ 0 & {}^{2,4}P & 0 & -{}^{2,4}Q \\ 0 & 0 & {}^{3,4}P & -{}^{3,4}Q \end{bmatrix} \times \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Matrix Assembly (from data)

Force filtering to limit to 10 point matches per tile, for easy visualization:

Non-zero structure

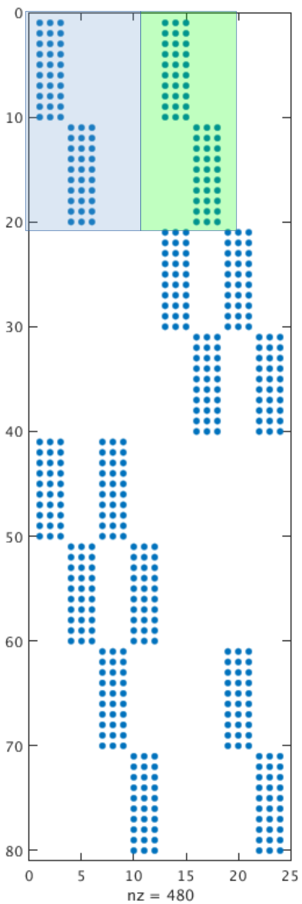
value-colored

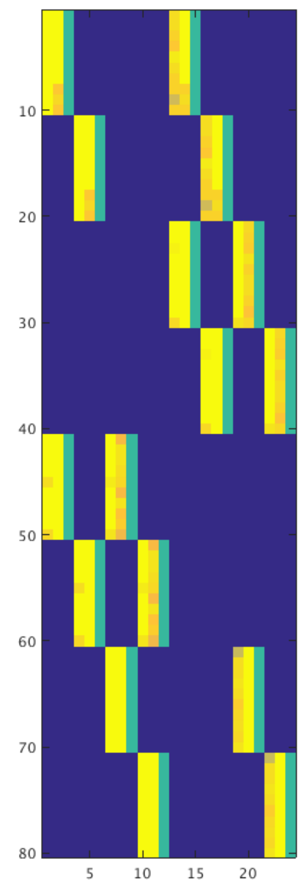
${}^{1,3}P$ ${}^{1,3}Q$

${}^{3,4}P$ ${}^{3,4}Q$

${}^{1,2}P$ ${}^{1,2}Q$

${}^{2,4}P$ ${}^{2,4}Q$





The ordering is a little different than the example, but, it's just block-wise row permutations:

$$\begin{bmatrix} {}^{1,3}P & 0 & -\,{}^{1,3}Q & 0 \\ 0 & 0 & {}^{3,4}P & -\,{}^{3,4}Q \\ {}^{1,2}P & -\,{}^{1,2}Q & 0 & 0 \\ 0 & {}^{2,4}P & 0 & -\,{}^{2,4}Q \end{bmatrix} \times \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$(Ax = b)$$

We know these values. They are the coordinates of the point matches.

We want to know these transform values, and apply them to the image data.

Least Squares

For this simple problem, there are:

- 80 equations (4 tile matches x 10 points x 2 DOF)
- 24 unknowns (4 tiles x (2 affine +1 translation) x 2 DOF)

The system is overdetermined.
Use linear least-squares to determine the best possible set of transformations.

$$\begin{bmatrix} {}^{1,3}P & 0 & -{}^{1,3}Q & 0 \\ 0 & 0 & {}^{3,4}P & -{}^{3,4}Q \\ {}^{1,2}P & -{}^{1,2}Q & 0 & 0 \\ 0 & {}^{2,4}P & 0 & -{}^{2,4}Q \end{bmatrix} \times \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$(Ax = b)$$

Per row (i) residual: $$r_i = b_i - \sum_{j=1}^{n} A_{ij} x_j$$

We want norm² to be as small as possible: $$S = \sum_{i=1}^{m} r_i^2$$

Least Squares

Per row (i) residual: $$r_i = b_i - \sum_{j=1}^{n} A_{ij} x_j$$

We want norm[2] to be as small as possible: $$S = \sum_{i=1}^{m} r_i^2$$

Find the minimum by seeking where the gradient relative to the parameters =0

gradient $$\frac{\partial S}{\partial x_j} = 2 \sum_i^m r_i \frac{\partial r_i}{x_j}$$

$$\frac{\partial r_i}{x_j} = -A_{ij}$$

gradient, expanded $$\frac{\partial S}{\partial x_j} = 2 \sum_i^m \left(b_i - \sum_{k=1}^{n} A_{ik} x_k \right) (-A_{ij}) = 0$$

rearranged $$\sum_{i=1}^{m} \sum_{k=1}^{n} A_{ij} A_{ik} x_k = \sum_{i=1}^{m} A_{ij} b_i$$

Matrix notation $$A^T A x = A^T b$$

This product is a square matrix (24x24, in our example).

Now there are 24 equations and 24 unknowns.

Weighting and Regularization

Our new matrix equation:

$$A^T A x = A^T b$$

We can weight each row of A differently

$$A^T W A x = A^T W b$$

Example: within-section point matches should be weighted more than cross-section point matches

We can regularize

$$A^T W A x + \lambda = A^T W b + \lambda d$$

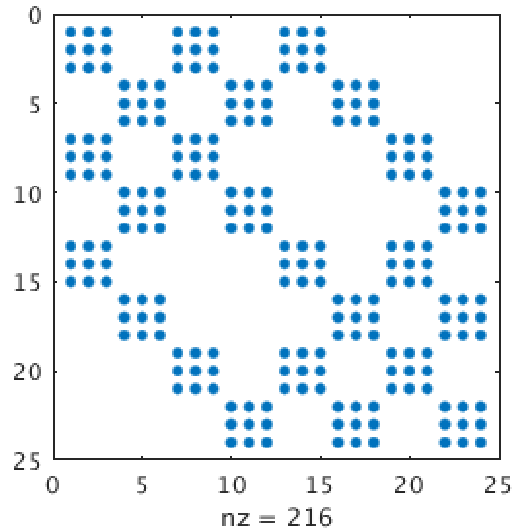
Derivation to-be-added, but, it’s much like on the last page, with an additional term in *S* that increases S if the solution strays far from a first guess/approximation fo *x*

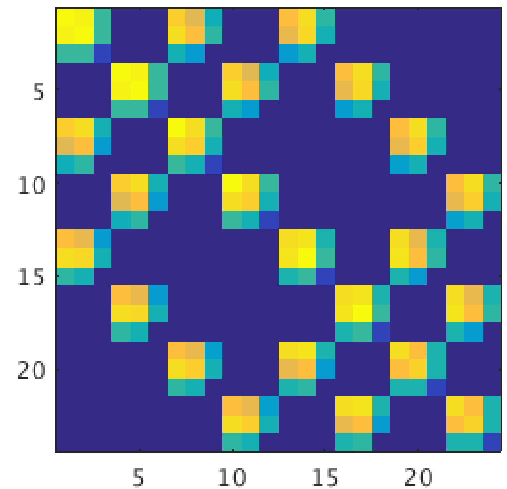
Renaming things:

$$K x = L_m$$

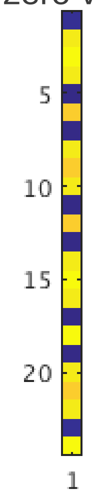
Properties of K and L_m

K is symmetric

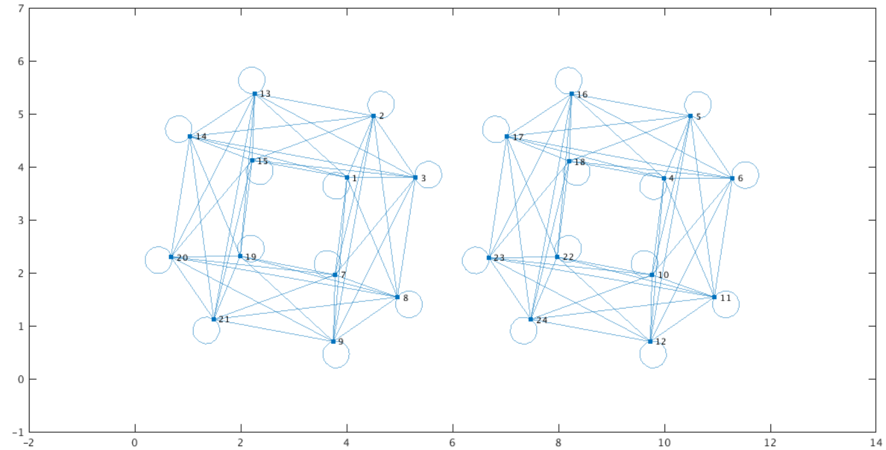




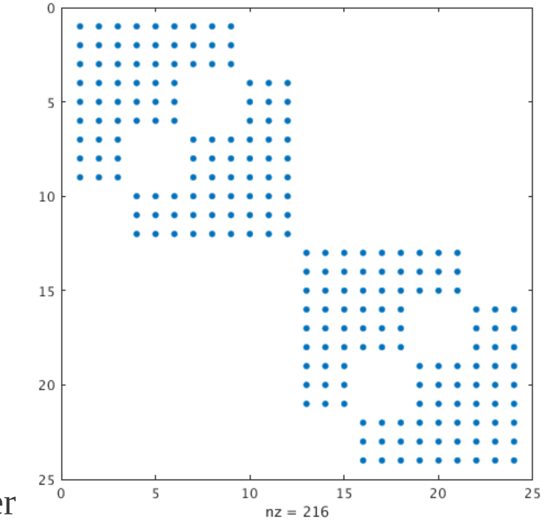
L_m has non-zero values



The graph of K shows two separate sub-matrices.



Which we can see easily from re-ordering rows and columns:

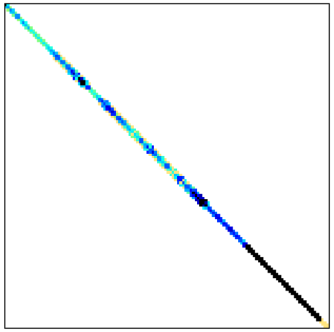


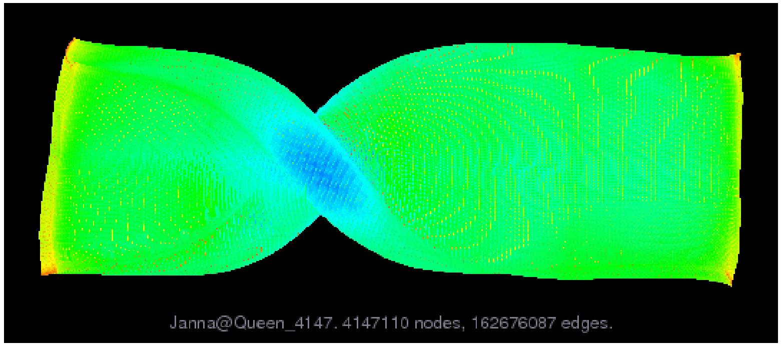
This type of structure will be important for partitioning large matrices for distributed solving.
I know about 2 partitioning packages for this:

- Scotch: used by the Pastix solver. (Also, PT-Scotch for parallel partitioning.)
- METIS. Also, (ParMetis)

Size estimate for $1mm^3$

- Estimate for K
  - Ntiles = 2.5e8 = (1e4 tiles/section)x(2.5e4 sections)
  - K m x n = 1.5e9 x 1.5e9 (6DOF per tile)
  - Guess that each tile has point matches to 10 other tiles.
  - nnz = 5e10 (each row has 3*(10+1) nz entries)
- Similar (?), smaller matrix: Janna/Queen_4147 (SuiteSparse...):
  - m x n = 4.1e6 x 4.1e6
  - nnz = 3.2e8





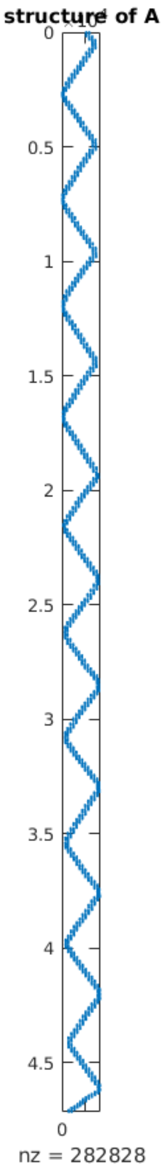
First pastix with Janna/Queen_4147

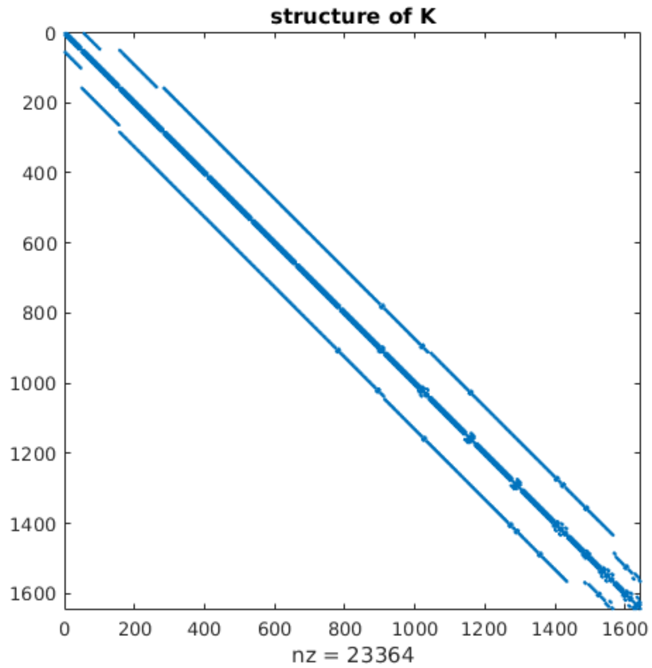
```
root@15884987838d:/app/pastix_5.2.3/src/example/bin# ./simple -MM /app/matrices/Queen_4147.mtx
MPI_Init_thread level = MPI_THREAD_MULTIPLE
driver: MatrixMarket file: /app/matrices/Queen_4147.mtx
open RHS file : /app/matrices/Queen_4147.mtx.rhs
cannot load /app/matrices/Queen_4147.mtx.rhs
Setting right-hand-side member such as X[i] = i
Check : Numbering              OK
Check : Sort CSC               OK
Check : Duplicates             OK
 +--------------------------------------------------------------------+
 +              PaStiX : Parallel Sparse matriX package               +
 +--------------------------------------------------------------------+
  Matrix size                                  4147110 x 4147110
  Number of nonzeros in A                      166823197
 +--------------------------------------------------------------------+
 +  Options                                                           +
 +--------------------------------------------------------------------+
        Version             :
        SMP_SOPALIN         :                  Defined
        VERSION MPI         :                  Defined
        PASTIX_DYNSCHED     :                  Not defined
        STATS_SOPALIN       :                  Not defined
        NAPA_SOPALIN        :                  Defined
        TEST_IRECV          :                  Not defined
        TEST_ISEND          :                  Defined
        TAG                 :                  Exact Thread
        FORCE_CONSO         :                  Not defined
        RECV_FANIN_OR_BLOCK :                  Not defined
        OUT_OF_CORE         :                  Not defined
        DISTRIBUTED         :                  Defined
        METIS               :                  Not defined
        WITH_SCOTCH         :                  Defined
        INTEGER TYPE        :                  int
        PASTIX_FLOAT TYPE   :                  double
 +--------------------------------------------------------------------+
   Time to compute ordering                    46.2 s
   Time to analyze                             5.09 s
   Number of nonzeros in factorized matrix     -2147483648
   Fill-in                                     -12.8728
   Number of operations (LLt)                  2.69604e+14
   Prediction Time to factorize (AMD 6180  MKL) 3.61e+04 s
   --- Sopalin : Threads are binded                ---
```

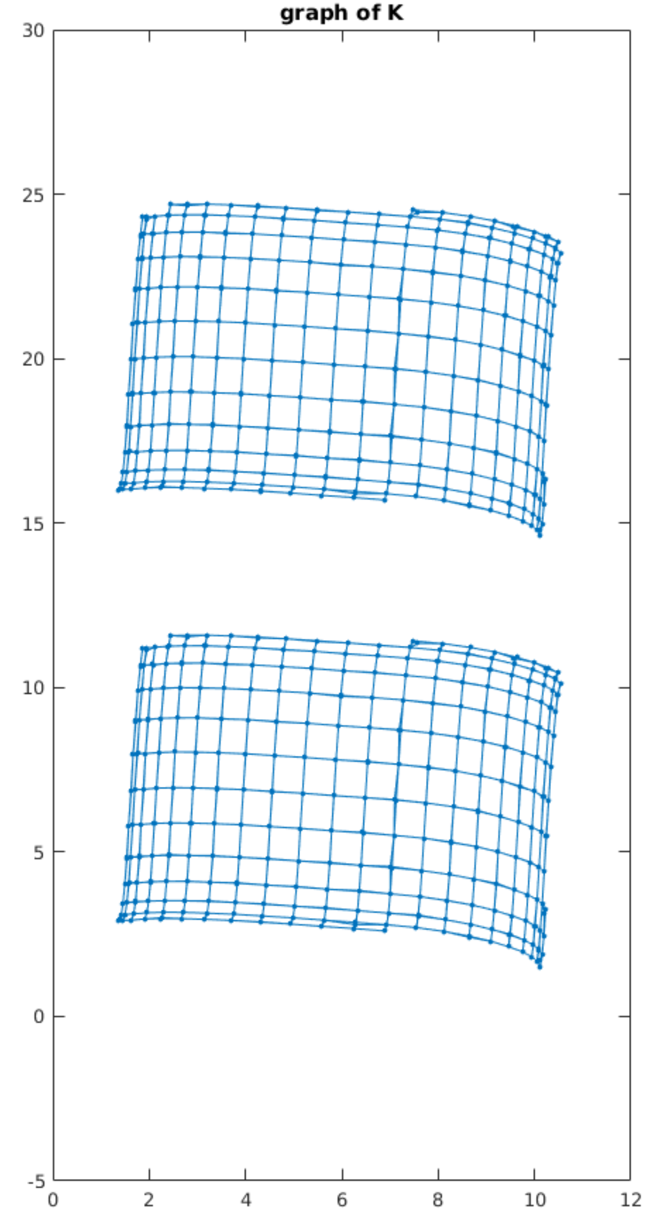
First, try simple.c (single-threaded...)

- 1.7e8 nnz before factorization
- -2.1e9 nnz after factorization (note minus sign, I thought I compiled with 64-bit integers, but, now I need to go check)
- At start of factorization, RAM usage went up to 115GB. (from ~5GB)
- 10 hours predicted time to factorize...
- This is a terrible example for single-thread, but, good for parallel

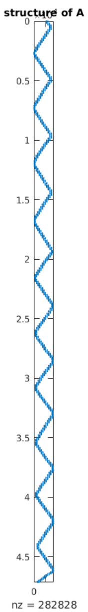
A and K from a whole section (282 tiles)



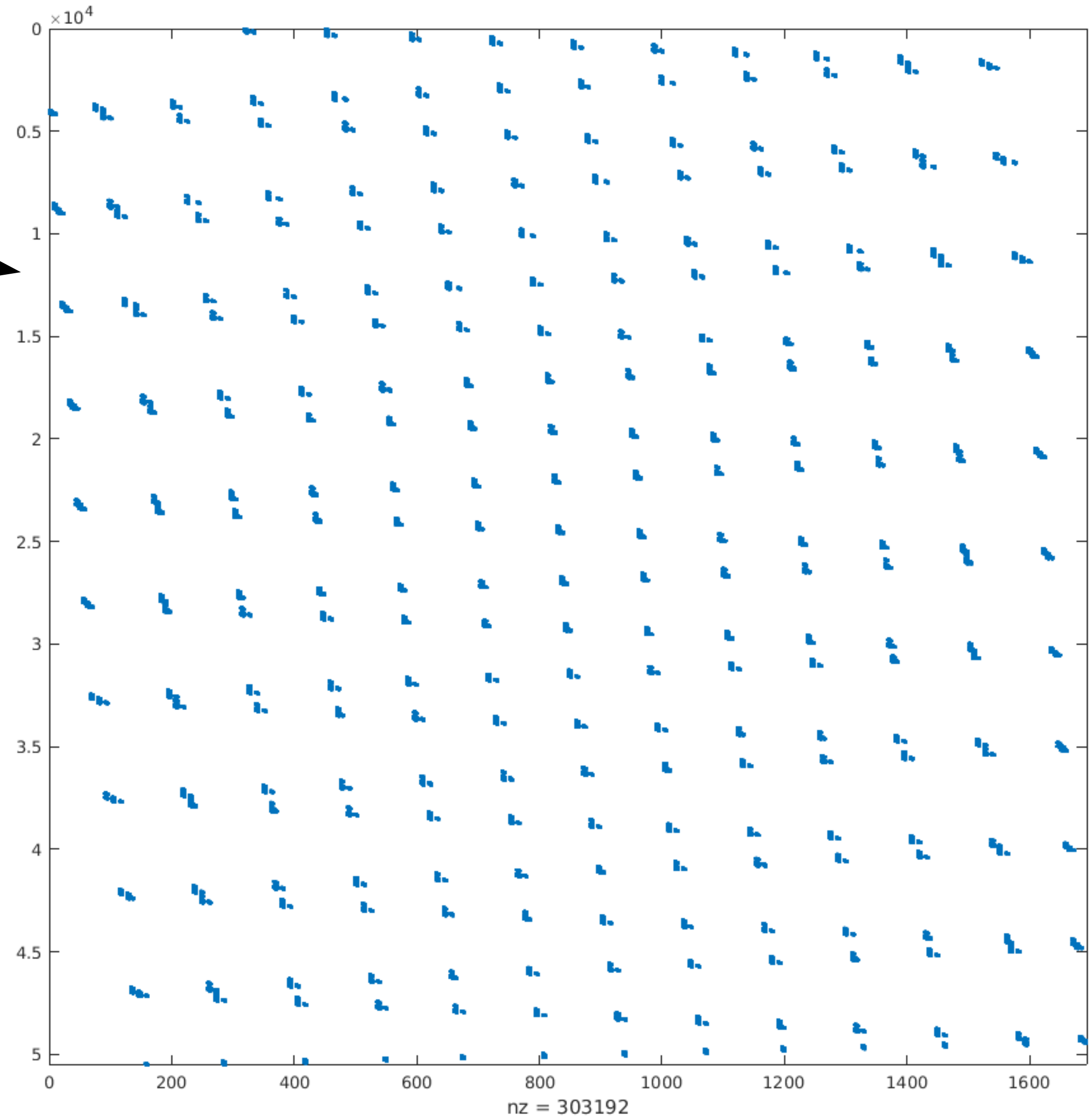




Do these plots make sense?



Change aspect



A size: 50532x1692
1692 = 282tiles x 6DOF/tile

50532 = average of ~190 point matches per overlap