**PnP Inversion Methodologies as Interpreted by ChatGPT from my MATLAB Programs**

 **Function Signature:**

* The function runallTTsizestcorrZZ() takes several inputs, including a file to load and parameters related to tomography processing, such as the size of cells, average velocity, Lagrange multipliers, tolerance for velocity, etc.

 **Global Variables:**

* global x y XV YV VV: Declares global variables, which are likely used to store coordinates or velocity-related information.

 **Loading Data:**

* The function begins by loading a file, presumably containing information about seismic events and stations, as well as travel times.

 **Event and Station Coordinates:**

* xt and yt store the x- and y-coordinates (likely longitude and latitude) for both seismic events and stations.

 **Distance Calculation:**

* The code calculates the distance between seismic events and their corresponding stations, storing it in d.

 **Filtering for Distances > 100 km:**

* The code filters out event-station pairs where the distance is less than 100 km, keeping only pairs where d > 100.

 **Extracting Coordinates and Travel Times:**

* After filtering, it stores the corresponding event and station coordinates (xs, ys) and the Moho travel times (Mtt).

 **Coordinate Shifting:**

* The variables Xshift and Yshift seem to define a reference point or shift in coordinates, likely used to normalize or align data to a specific location.

 **Figure Creation (figure(201)):**

* A figure is opened or switched to for potential plotting. It might be used later in the code to visualize something related to the velocity model or ray paths.

 **Model Grid Creation (make\_mod\_2d):**

* The function make\_mod\_2d() is likely used to create a 2D grid representing the velocity model, with xcell defining the resolution, and Vavg as the initial average velocity.

 **Station/Event Matching (stafinder, stamatcher):**

* stafinder() and stamatcher() are functions to find unique stations and events based on proximity (within 35 units). These functions help identify which stations and events are close to each other.
* IST and IEV contain indices for each event-station pair, identifying which station/event each pair corresponds to.

 **Ray Path Generation (make\_ray\_2d):**

* For each event-station pair, the code calculates the ray path using make\_ray\_2d(). The resolution of the path is controlled by xcell/4.

 **Travel Time Calculation (get\_ray\_time):**

* The function get\_ray\_time() computes the travel time along the ray path and returns NROWT, which contains values describing the ray’s interaction with the velocity model (possibly indicating velocity cells along the ray).

 **Valid Ray Path Filtering:**

* The loop processes each event-station pair, filtering out invalid paths (those containing NaN values) and only considering paths where the residual travel time (DTJ) is within the tolerance (tol).

 **Storage of Valid Data:**

* For valid paths, the ray path data (NROWT), station/event associations (STM and EVM), and travel times (T and DT) are stored in matrices for further analysis or modeling.

 **Loop Counters:**

* The counters i and j keep track of how many total event-station pairs were processed (i) and how many valid paths were found (j).

Yes, based on what we've reviewed so far, it does seem like this program is building up to an **inversion** process. Here’s why:

1. **Ray Path Calculation**: The code calculates travel times along ray paths between seismic events and stations. This step is common in seismic tomography because ray paths are used to estimate travel times through different parts of the Earth.
2. **Travel-Time Residuals**: The calculation of the residual between observed and modeled travel times (DTJ = Mtt(i) - TT) suggests that the code is preparing to minimize the difference between observed data and a model. This is a key component of inversion processes, where you iteratively adjust your model to reduce these residuals.
3. **Data Storage for Later Processing**: The program is storing valid ray paths and their corresponding travel times. This kind of data is typically fed into an inversion algorithm, which adjusts the velocity model to better match the observed travel times.
4. **Lagrange Multipliers**: The earlier mention of Lagrange multipliers (LagS, dLag, LagR) suggests that the program might use a regularization technique in the inversion process. Lagrange multipliers are often employed to control the trade-off between fitting the data and maintaining smoothness or stability in the model.
5. **Velocity Model**: The velocity model is being built (VV) and likely updated during or after the inversion. The goal of seismic tomography is to adjust this model to improve the match between observed and predicted travel times.

In seismic tomography, inversion is the process where you iteratively adjust the model (e.g., the subsurface velocity structure) to minimize the difference between observed and calculated travel times. This program seems to be setting up the framework for such an inversion.

So, yes, I’d say it’s reasonable to expect that the later parts of the program might involve an inversion process, or it might be building up to it by preparing the data (ray paths, residuals, etc.).

1. **Finite Difference Operators (D1 and D):**
   * The code constructs finite difference operators (D1 and D) that act on the x- and y-directions. These are used to compute gradients or smoothness constraints in tomography. The operators help ensure that the solution to the inversion is smooth by penalizing large differences between neighboring cells in the model.
2. **Ray Path Coverage (SG):**
   * The program calculates the sum of ray path values (SG = sum(G)) to assess how well each grid cell is sampled by the rays. Cells with insufficient ray coverage are marked as NaN (invalid). This step is crucial in tomography to avoid using poorly constrained parts of the model.
3. **Matrix Construction (BM and BMTBM):**
   * The concatenated matrix BM contains information about the ray paths (G), station associations (STM), and event associations (EVM). The normal equation matrix (BMTBM = BM' \* BM) is being computed, which is used to solve the least squares problem in the inversion.
4. **Regularization (SZ0 and SZ):**
   * Regularization is applied by scaling the top left block of the matrix SZ0 by the Lagrange multiplier LagR. This regularization helps control how much the model can deviate from the initial guess, ensuring stability in the inversion process.
5. **Coordinate Shifts (XV and YV):**
   * The final lines shift the x and y coordinates of the velocity model (XV and YV). This shift is likely done for alignment or normalization purposes, ensuring that the model coordinates match the data coordinates.

**Inversion Context:**

This block strengthens the idea that the program is building up to an inversion. The construction of finite difference matrices, the ray coverage assessment, and the formation of the normal equation matrix are all standard steps in preparing for a least squares inversion. The regularization with Lagrange multipliers further suggests that the program will solve an inverse problem, where it adjusts the velocity model to minimize misfit while maintaining a degree of smoothness.

1. **Iteration Over Multiple Inversions:**
   * The loop runs 10 iterations, each time refining the velocity model by updating the Lagrange multiplier (Lag) and recalculating the inversion solution (N).
2. **Solving the Inversion Problem:**
   * The key inversion is solved with the expression N = inv(BMTBM + Lag \* SZ) \* BM' \* DT', where the regularization term Lag \* SZ controls the trade-off between fitting the data and maintaining model smoothness.
3. **Updating and Normalizing the Velocity Model:**
   * After solving for N, the velocity updates (NV) are extracted and converted into new velocity values using the starting average velocity (Vavg). These values are then normalized to show percentage changes.
4. **Visualization:**
   * The first subplot displays the updated velocity model with ray coverage as a mask, showing how the inversion adjusts the model over iterations.
   * The second subplot visualizes the corrections applied to stations (NNST) and how these corrections are interpolated over the velocity grid.
5. **Regularization Adjustment:**
   * The Lagrange multiplier (Lag) is iteratively adjusted, likely to control the balance between data fitting and model smoothness as the inversion proceeds.
6. **Saving Outputs:**
   * After the loop completes, key variables (G, DT, T, N, etc.) are saved for further analysis or future use.

**Inversion Conclusion:**

This block confirms that the program is indeed carrying out an inversion. The velocity model is iteratively refined by solving the least squares problem while adjusting for data misfits and regularization using Lagrange multipliers. The output is visualized after each iteration to track the changes in the model and station corrections.

**DIFFERENCES BETWEEN SIZE AND SMTH**

**Conceptual Differences:**

1. **Station Finder Tolerance:**
   * In the first program, stafinder(xs(:,2),ys(:,2),35) is used with a tolerance of 35 for both events and stations. However, in the second program, the tolerance has been changed to 5 (stafinder(xs(:,2),ys(:,2),5) and stafinder(xs(:,1),ys(:,1),5)). This indicates that the second program is more sensitive to station/event proximity, leading to potentially finer resolution or stricter station/event matching.
2. **Grid Shift:**
   * The second program uses a different grid shift: Xshift=-1003.9 and Yshift=-658.2, compared to the first program's Xshift=-991.41 and Yshift=-645.7. These shifts might align the model with slightly different regions or correct for changes in reference points.
3. **Lagrange Multiplier Regularization:**
   * Both programs update the Lagrange multiplier (Lag), but the second program slightly modifies how the regularization matrix (SZ0) is constructed. In the first program, SZ0(1:vyc\*vxc,1:vyc\*vxc)=LagR\*eye(size(D)), while in the second program it is SZ0(1:vyc\*vxc,1:vyc\*vxc)=LagR\*D, where D is the matrix created earlier. This change suggests a difference in how smoothness constraints are applied in the second program, potentially adding more complex regularization based on grid connectivity.
4. **Velocity Tolerance:**
   * Both programs use the tol parameter to discard velocity values, but it's worth noting that the second program might be more stringent due to the change in tolerance for station matching. This could imply that the second program is tuned to reject noisier data more aggressively.
5. **General Similarity:**
   * Conceptually, both programs follow the same basic structure:
     + Load data.
     + Calculate ray paths and travel times.
     + Perform inversion using least squares with Lagrange multiplier regularization.
     + Update and visualize the velocity model.
     + Save relevant outputs.
   * The differences mostly revolve around parameter tuning, sensitivity adjustments, and possibly stricter regularization in the second program.