1) List ATLAS files:  
procedure ListAtlasDataFilesInMemo(const InputDate, InputTime: string);

var

Year, Month, Day: string;

FolderPath: string;

FilesList: TStringList;

i: Integer;

begin

with frmMain do

begin

// Extract the year, month, and day from the InputDate

Year := Copy(InputDate, 1, 4); // '2020'

Month := Copy(InputDate, 6, 2); // '01'

Day := Copy(InputDate, 9, 2); // '02'

// Construct the folder path using the base path from lblProgramPath

FolderPath := IncludeTrailingPathDelimiter(lblProgramPath.Caption) +

'DB\ATLASDATA\' + Year + '\' + Month + '\' + Year + Month + Day;

// Create a TStringList to hold the list of files

FilesList := TStringList.Create;

try

// List all files in the directory

FilesList.AddStrings(TDirectory.GetFiles(FolderPath, '\*.\*', TSearchOption.soTopDirectoryOnly));

// Clear the memo before adding new content

mmoATLAS\_files.Clear;

// Add the list of files to the memo

for i := 0 to FilesList.Count - 1 do

begin

mmoATLAS\_files.Lines.Add(FilesList[i]);

end;

finally

FilesList.Free;

end;

end

end;

2) FILTER TIME:  
procedure TfrmMain.Filter\_timeClick(Sender: TObject);

var

DateStr: string;

TimeStr: string;

CutoffMin : integer;

begin

// Get the date and time from the labeled edit controls

DateStr := ledEQ\_datex.Text; // Assuming 'ledEQ\_datex' is where the date is entered

TimeStr := ledEQ\_timex.Text; // Assuming 'ledEQ\_timex' is where the time is entered

CutoffMin := StrToInt(ledTime\_cutoff.Text);

// Call the extraction procedure, passing the memo, date, time, and grid

UTILS\_ATLAS.Filter\_Time(mmoATLAS\_files, DateStr, TimeStr, CutoffMin, mmoATLAS\_files2);

end;

3) EXTRACT EQ INFO:  
procedure ExtractSelectedAtlasParametersToGrid(Memo: TMemo; Grid: TStringGrid);

var

FileName: string;

FileList: TStringList;

FileContent: TStringList;

Fields: TStringList;

i, j, Row: Integer;

Line, Station: string;

CodaTime: string;

begin

// Initialize TStringLists to hold the file names and contents

FileList := TStringList.Create;

FileContent := TStringList.Create;

Fields := TStringList.Create;

try

// Copy the lines from the memo (filtered ATLAS files) to FileList

FileList.Assign(Memo.Lines);

// Clear the grid and set up the headers

Grid.RowCount := 1; // Reset row count (header row is 0)

Grid.ColCount := 11; // Increased number of columns to include Coda Time

Grid.Cells[0, 0] := 'Network';

Grid.Cells[1, 0] := 'Station';

Grid.Cells[2, 0] := 'Channel';

Grid.Cells[3, 0] := 'Phase';

Grid.Cells[4, 0] := 'Time';

Grid.Cells[5, 0] := 'First Motion';

Grid.Cells[6, 0] := 'Onset';

Grid.Cells[7, 0] := 'Weight Code';

Grid.Cells[8, 0] := 'User';

Grid.Cells[9, 0] := 'Coda Time'; // New column for Coda Time

Grid.Cells[10, 0] := 'Filename'; // New column for Filename

// Iterate over each filtered file path listed in the memo

for i := 0 to FileList.Count - 1 do

begin

FileName := FileList[i]; // Get the file name (path)

try

// Load the content of the file into FileContent

FileContent.LoadFromFile(FileName);

except

on E: Exception do

begin

// Handle any file loading errors (optional)

ShowMessage('Error loading file: ' + FileName + ' - ' + E.Message);

Continue;

end;

end;

// Iterate over each line of the file content to extract parameters

CodaTime := ''; // Reset Coda Time for each file

for j := 0 to FileContent.Count - 1 do

begin

Line := FileContent[j];

// Skip comment lines (starting with #)

if Line.StartsWith('#') then

Continue;

// Use TStringList.DelimitedText to parse the CSV line

Fields.Delimiter := ',';

Fields.StrictDelimiter := True; // Strict to avoid space issues

Fields.DelimitedText := Line;

// Process "phase" records from the file

if (Fields[0] = 'phase') and (Fields.Count >= 11) then

begin

// Add a new row in the grid for each line that has the correct fields

Row := Grid.RowCount;

Grid.RowCount := Row + 1;

// Populate the grid with the extracted parameters

Grid.Cells[0, Row] := Fields[1]; // Network

Grid.Cells[1, Row] := Fields[2]; // Station

Grid.Cells[2, Row] := Fields[3]; // Channel

Grid.Cells[3, Row] := Fields[5]; // Phase

Grid.Cells[4, Row] := Fields[6]; // Time

Grid.Cells[5, Row] := Fields[7]; // First Motion (can be empty)

Grid.Cells[6, Row] := Fields[8]; // Onset

Grid.Cells[7, Row] := Fields[9]; // Weight Code

Grid.Cells[8, Row] := Fields[10]; // User

// Populate the Coda Time if found (if not found yet, it will be empty)

Grid.Cells[9, Row] := CodaTime; // Coda Time

Grid.Cells[10, Row] := FileName; // Add Filename (only the file name)

end;

// Process "coda" records from the file

if (Fields[0] = 'coda') and (Fields.Count >= 6) then

begin

// Extract the coda time (assume station matches and should be applied to the next phase)

CodaTime := Fields[5]; // Extract the coda time

Grid.Cells[9, Row] := CodaTime; // Coda Time

end;

end;

end;

finally

// Free the file content and list objects

FileContent.Free;

FileList.Free;

Fields.Free;

end;

end;

4) FILTER DISTANCE:  
procedure TfrmMain.Filter\_distanceClick(Sender: TObject);

var

eqMag: Single;

Cutoff\_Distance: Double;

begin

// Retrieve the earthquake magnitude from the labeled edit

eqMag := StrToFloat(ledEQ\_Mag.Text);

// Determine the cutoff distance based on the magnitude

if (eqMag >= 3.5) and (eqMag <= 3.9) then

Cutoff\_Distance := 300

else if (eqMag >= 3.0) and (eqMag <= 3.4) then

Cutoff\_Distance := 200

else if (eqMag >= 2.5) and (eqMag <= 2.9) then

Cutoff\_Distance := 100

else if (eqMag < 2.5) then

Cutoff\_Distance := 50;

// Call the ComputeHypocentralDistance procedure with the calculated cutoff distance

UTILS\_ATLAS.FilterDistance(sgAtlas\_data, sgSOEPD\_stations, sgAtlasData2,

ledEQ\_Lat, ledEQ\_Lon, ledEQ\_Dep, ledEQ\_Mag, Cutoff\_Distance);

end;  
  
procedure FilterDistance(sgAtlas\_data, sgSOEPD\_stations, sgAtlasData\_final: TStringGrid;

ledEQ\_Lat, ledEQ\_Lon, ledEQ\_Dep, ledEQ\_Mag: TLabeledEdit; Cutoff\_Distance: Double);

var

i, j, finalRow: Integer;

stationValue, searchValue: string;

found: Boolean;

stationLat, stationLon, stationElev: Double;

eqLat, eqLon, eqDep, eqMag: Double;

surfaceDistance, elevationDiff, hypoDistance: Double;

begin

// Parse earthquake location and depth from labeled edits

eqLat := StrToFloat(ledEQ\_Lat.Text);

eqLon := StrToFloat(ledEQ\_Lon.Text);

eqDep := StrToFloat(ledEQ\_Dep.Text);

eqMag := StrToFloat(ledEQ\_Mag.Text);

// Initialize the final grid with the same columns as sgAtlas\_data

sgAtlasData\_final.ColCount := sgAtlas\_data.ColCount; // Ensure the column count matches

sgAtlasData\_final.RowCount := 1; // Start with 1 row for headers (if needed)

// Copy the headers from sgAtlas\_data to sgAtlasData\_final

for i := 0 to sgAtlas\_data.ColCount - 1 do

begin

sgAtlasData\_final.Cells[i, 0] := sgAtlas\_data.Cells[i, 0];

end;

// Iterate through sgAtlas\_data rows (starting from 1, assuming row 0 is header)

for i := 1 to sgAtlas\_data.RowCount - 1 do

begin

// Get the first 3 characters of the station value from the second column (index 1) of sgAtlas\_data

stationValue := LeftStr(sgAtlas\_data.Cells[1, i], 3);

// Search for the station value in sgSOEPD\_stations, specifically in the first column (index 0)

found := False;

for j := 1 to sgSOEPD\_stations.RowCount - 1 do

begin

// Get the station\_code from the first column (index 0) of sgSOEPD\_stations

searchValue := sgSOEPD\_stations.Cells[0, j];

if stationValue = searchValue then

begin

// Station found in sgSOEPD\_stations

found := True;

// Get station coordinates and elevation

stationLat := StrToFloat(sgSOEPD\_stations.Cells[1, j]); // Latitude is in column 1

stationLon := StrToFloat(sgSOEPD\_stations.Cells[2, j]); // Longitude is in column 2

stationElev := StrToFloat(sgSOEPD\_stations.Cells[3, j]); // Elevation is in column 3

// Calculate the surface distance using the Haversine formula

surfaceDistance := Haversine(eqLat, eqLon, stationLat, stationLon);

// Compute the elevation difference (in km)

elevationDiff := (eqDep - stationElev) / 1000.0;

// Calculate the 3D hypocentral distance using Pythagorean theorem

hypoDistance := Sqrt(Sqr(surfaceDistance) + Sqr(elevationDiff));

// Check if the hypocentral distance is within the cutoff distance

if hypoDistance <= Cutoff\_Distance then

begin

// Add the station to the sgAtlasData\_final grid

finalRow := sgAtlasData\_final.RowCount; // Get current row count

sgAtlasData\_final.RowCount := finalRow + 1; // Increment row count

// Copy the row data from sgAtlas\_data to sgAtlasData\_final (same format)

for var col := 0 to sgAtlas\_data.ColCount - 1 do

begin

sgAtlasData\_final.Cells[col, finalRow] := sgAtlas\_data.Cells[col, i];

end;

// Optionally, you can add the calculated hypocentral distance in an extra column if needed

// (Assuming the last column can hold the distance value if needed)

// sgAtlasData\_final.Cells[sgAtlasData\_final.ColCount - 1, finalRow] := FloatToStr(hypoDistance);

end;

Break; // Exit the search loop once found

end;

end;

end;

end;

5) COMBINE DATA  
procedure CombineAtlasPS(SourceGrid, TargetGrid: TStringGrid);

var

i, NewRow: Integer;

Station, CodaTime, CODA : string;

FoundP, FoundS: Boolean;

PhaseP, PhaseS: TStringList;

begin

PhaseP := TStringList.Create;

PhaseS := TStringList.Create;

try

// Initialize the target grid with the desired structure

TargetGrid.ColCount := 14; // Adjust column count to include Coda Time

TargetGrid.RowCount := 1; // Start with the header row

// Set up the headers for the target grid

TargetGrid.Cells[0, 0] := 'Network';

TargetGrid.Cells[1, 0] := 'Station';

TargetGrid.Cells[2, 0] := 'Channel';

TargetGrid.Cells[3, 0] := 'P Phase';

TargetGrid.Cells[4, 0] := 'P Time';

TargetGrid.Cells[5, 0] := 'S Phase';

TargetGrid.Cells[6, 0] := 'S Time';

TargetGrid.Cells[7, 0] := 'First Motion';

TargetGrid.Cells[8, 0] := 'Onset';

TargetGrid.Cells[9, 0] := 'Weight Code';

TargetGrid.Cells[10, 0] := 'User';

TargetGrid.Cells[11, 0] := 'Coda Time'; // Coda Time column

TargetGrid.Cells[12, 0] := 'CODA'; // Coda Time column

TargetGrid.Cells[13, 0] := 'Filename';

i := 1; // Start from the first row after the header

while i < SourceGrid.RowCount do

begin

// Clear the lists for storing P and S phases

PhaseP.Clear;

PhaseS.Clear;

// Get the current station

Station := SourceGrid.Cells[1, i];

FoundP := False;

FoundS := False;

// Check if the current row contains a P phase and extract its Coda Time

if SourceGrid.Cells[3, i] = 'P' then

begin

PhaseP.Text := SourceGrid.Rows[i].Text;

CodaTime := SourceGrid.Cells[9, i]; // Coda Time from column 11 (adjust if necessary)

FoundP := True;

end;

// Look ahead to the next row to check if it's the same station (for S phase)

if (i + 1 < SourceGrid.RowCount) and (SourceGrid.Cells[1, i + 1] = Station) then

begin

// If the next row contains the S phase, process it and overwrite the Coda Time

if SourceGrid.Cells[3, i + 1] = 'S' then

begin

PhaseS.Text := SourceGrid.Rows[i + 1].Text;

CodaTime := SourceGrid.Cells[9, i + 1]; // Overwrite Coda Time from the S phase row

FoundS := True;

Inc(i); // Skip the next row since we've processed the S phase

end;

end;

// Add a new row in the target grid for each station

NewRow := TargetGrid.RowCount;

TargetGrid.RowCount := NewRow + 1;

// Populate the new row with extracted data (correct column mapping)

TargetGrid.Cells[0, NewRow] := PhaseP[0]; // Network

TargetGrid.Cells[1, NewRow] := PhaseP[1]; // Station

TargetGrid.Cells[2, NewRow] := PhaseP[2]; // Channel

// Populate the P phase data

if FoundP then

begin

TargetGrid.Cells[3, NewRow] := 'P'; // P Phase

TargetGrid.Cells[4, NewRow] := PhaseP[4]; // P Time

end;

// Populate the S phase data if found

if FoundS then

begin

TargetGrid.Cells[5, NewRow] := 'S'; // S Phase

TargetGrid.Cells[6, NewRow] := PhaseS[4]; // S Time

end;

// Populate additional data (First Motion, Onset, Weight Code, User, Coda Time, Filename)

TargetGrid.Cells[7, NewRow] := PhaseP[5]; // First Motion

TargetGrid.Cells[8, NewRow] := PhaseP[6]; // Onset

TargetGrid.Cells[9, NewRow] := PhaseP[7]; // Weight Code

TargetGrid.Cells[10, NewRow] := PhaseP[8]; // User

TargetGrid.Cells[11, NewRow] := CodaTime; // Coda Time

TargetGrid.Cells[13, NewRow] := PhaseP[10]; // Filename

if Trim(CodaTime) <> '' then

CODA := UTILS\_Math.ComputeCoda(PhaseP[4], CodaTime) // Call ComputeCoda from UTILS\_Math

else

CODA := ''; // If there's no valid Coda Time, leave it blank

TargetGrid.Cells[12, NewRow] := CODA; // CODA

// Move to the next station

Inc(i);

end;

finally

PhaseP.Free;

PhaseS.Free;

end;

end;