Radar and AI-based soil moisture monitoring for efficient farm irrigation

Programming

Ravi Palmieri

Files Converted From C++ → Python

- Lib files converted so far
- Specifically:
  - File
  - Header

Lib-File.lau.py

```
#  Title: File Functions
#  Filename: Lib-File.lua
#  Description: Functions related to file I/O and filenames
#  Version 1.1.1, 02/24/2011
#  Author: Patton Gregg
#  Revision History:
#       1.1.1, 02/24/2011
#               Added function DirExists to check if a directory exists
#       1.1.0, 01/11/2011
#               Added a parameter to pass a description to append to the output filename for ChartFileOut
#       1.0.1, 12/16/2010
#               Moved output_header function from Tool-Data-Proc and renamed OutputHeader
#               Added function ChartFileOut to write out a chart file
#       1.0.0, 11/30/2010
#               Initial Release

#  Include Files

import lua
lua.dofile("LuaProg/Lib-Header.lua")

#  Function List

# DirExists (dir_path) -> bool
# FileExists (file_path) -> bool
# ChopExtension (filename) -> string
# DataFileOut (tape, data_filename, desc) -> VOID
# ChartFileOut (w, filename, desc) -> VOID
# OutputHeader(file_path) -> VOID

# Returns a bool indicating of a file exists

def dir_exists(dir_path):
    dir = os.path.dirname(dir_path)
    dir_abs = os.path.abspath(dir)
    valid = True
    try:
        with open(dir_abs, "r"):
            pass
    except IOError as e:
        if e.errno == 2:
            valid = False
    return valid

# Returns a bool indicating of a file exists

def file_exists(file_path):
    return os.path.isfile(file_path)

# Removes the file type extention from the end of a filename

def chop_extension(filename):
    fname, ext = filename.split('.')
    return fname, ext

# Write a data tape to a file

def data_file_out(tape, data_filename, desc):
    filename, ext = chop_extension(data_filename)
    if desc == None:
        desc = ""
    if string.find(desc, 'CONV_FORMAT')!= None:
        if string.find(desc, 'ASCII')!= None:
            ext = ".img"
        elif string.find(desc, 'HEX')!= None:
            ext = ".imx"
        elif string.find(desc, 'BIN')!= None:
```

```
#! For some reason there is an error when you select "Yes" from the form and it tries to write out the file
        # if (FileExists(filename)) then
                # local overwrite_file = forms.yes_no("File: " .. filename .. " already exists. Overwrite file?")

                # if (overwrite_file ~= 6) then
                        # forms.message("File: " .. filename .. " was not output.")
                # else
                        # print ("\n--------------------------------------")
                        # print ("File Output:", filename)
                        # print ("--------------------------------------\n")
                        # data_tape.write(tape, filename)
                # end
        # else

def file_output(filename):
    print("\n--------------------------------------")
    print("File Output:", filename)
    print("--------------------------------------\n")
    data_tape.write(tape, filename)

    # end

# Writes a chart to file

def file_output(filename):
    print("\n--------------------------------------")
    print("File Output:", filename)
    print("--------------------------------------\n")
    data_tape.write(tape, filename)
<|python|>#!/usr/bin/env python
# -*- coding: utf-8 -*-

import os
import sys

try:
    from setuptools import setup
except ImportError:
    from distutils.core import setup

import django_tables2

if sys.argv[

# Writes the header of a data file out as a separate text file

def output_header(data_file):
        io.input(data_file)

        filename, ext = ChopExtension(data_file)
        filename = filename + " - HEADER.txt"
        io.output(filename)

        header = ""
        data_line = io.read()
        data_line = data_line + "\n"

        scans_start = ""

        if (ext == ".img"):
                scans_start = "Scan\n"
        elif (ext == ".
```

Lib-Header.lau.py

```
# Title: Standard Header
#  Filename: Lib-Header.lua
#  Description: Contains standard global variable and constant assigments
#  Version 1.1.0, 01/11/2011
#  Author: Patton Gregg
#  Revision History:
#       1.1.0, 01/11/2011
#                Added GATE_CALC_IF_OFFSET
#                Delete GATE_HARMON_THREAS
#       1.0.3, 12/21/2010
#                Added SC = data_scan
#       1.0.2, 12/08/2010
#                Added GATE_HARMON_THREAS
#       1.0.1, 11/30/2010
#                Added cInNanoSecPerM (Speed of Light) constant
#                Added GateClock constant
#       1.0.0, 11/30/2010
#                Initial Release



def deduplicate_list(input_list):
    deduplicated_list = list(set(input_list))
    print(deduplicated_list)
<|python|>#!/usr/bin/env python
```

Files Converted From C++ → Python

- Lib files converted so far
- Specifically:
  - Scan
  - Math (potentially useful equations found)
  - Tape

Lib-Scan.lua.py

```
# Title: Scan Functions
# Filename: Lib-Scan.lua
# Description: Functions related to data scans
# Version 1.0.3, 02/02/2011
# Author: Patton Gregg
# Revision History:
#       1.0.3, 02/02/2011
#               Fixed bug in FreqSliceScan function that caused band exclusion to not work
#       1.0.2, 01/11/2011
#               Added functionality to exclude points in specified band for FreqSliceScan function
#       1.0.1, 12/16/2010
#               Change function name SliceFreqRange to FreqSliceScan
#       1.0.0, 11/30/2010
#               Initial Release
# ------------------------------------------------------
# Include Files
# ------------------------------------------------------

import math
import radar
import lua

# ------------------------------------------------------
# Function List
# ------------------------------------------------------
# TimeWindowScan (scan, win_start, win_stop, freq_step, window_bool) -> scan
# FreqSliceScan (scan, win_freq_start, win_freq_stop, data_freq_start, data_freq_step, exclude_band_bool) -> scan
# ------------------------------------------------------

# Returns a scan that has had its frequency data sliced to the specified frequency band

def freq_slice_scan(scan, win_freq_start, win_freq_stop, data_freq_start, data_freq_step, exclude_band):
    index_start = get_freq_index(win_freq_start, data_freq_step, data_freq_start)
    index_stop = get_freq_index(win_freq_stop, data_freq_step, data_freq_start)
    if exclude_band:
        I = range(index_stop - index
```

Lib-Math.lua.py

```
# Title: Math Functions
# Filename: Lib-Math.lua
# Description: Functions for performing routine math operations
# Version 1.2.2, 01/27/11
# Author: Patton Gregg
# Revision History:
#       1.3.0, 01/27/2011
#               Added: DistPoint2Line function
#       1.2.2, 01/27/2011
#               Fixed bug in CalcRMS function
#       1.2.1, 01/10/2011
#               Added CalcRMS function
#       1.2.0, 12/21/2010
#               Added GetMagCV function
#               Added GetCurrPos function
#       1.0.1, 12/17/2010
#               Added MinMaxCompareV function
#       1.0.0, 11/30/2010
#               Initial Release
# ------------------------------------------------------------
# Include Files
# ------------------------------------------------------------
import lua
lua.dofile("LuaProg/Lib-Header.lua")
# ------------------------------------------------------------
# Function List
# ------------------------------------------------------------
# Round (number) -> number
# N2Order (number) -> number
# N2OrderFloor (number) -> number
# Order2N (number) -> number
# Dist2D (x1, y1, x2, y2) -> number
# Dist3D (x1, y1, z1, x2, y2, z2) -> number
# DistPoint2Line (pointX, pointY, lineStartX, lineStartY, lineEndX, lineEndY) -> number
# MinMaxCompareV (data, minV, maxV) -> min_vector, max_vector
# GetMagCV (complex_v) -> vector
# GetCurrPos (x, y, z, v, ang, deltaT) -> number
# GetCurrPosBreath (x, y, z, rate, throw, ang, curr_t) -> number
# CalcRMS (vector) -> number
# CalcSigRMS (dataTime, sigDist, timeStep) -> number
# ------------------------------------------------------------

# Returns the input number rounded to the nearest whole number

def round(num):
    return int(round(num))

# Returns n for the order of the next factor of 2^n

def n2_order(n):
    return math.ceil(math.log10(n) / math.log10(2))

# Returns n for the order of the closest previous factor of 2^n

def n2_order_floor(n):
    return math.floor(math.log10(n) / math.log10(2))

# Returns 2 ^ n

def order_2_n(order):
    return 2 ** order

# Returns the absolute distance between 2 points in 2D space

def dist2d(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
```

```
# Returns the absolute distance between 2 points in 3D space

def dist3d(x1, y1, z1, x2, y2, z2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2 + (z2 - z1) ** 2)

# Returns the closest distance between a point and a line

def dist_point_to_line(pointX, pointY, lineStartX, lineStartY, lineEndX, lineEndY):
    numer = ((pointX - lineStartX) * (lineEndX - lineStartX)) + ((pointY - lineStartY) * (lineEndY - lineStartY))
    denom = (lineEndX - lineStartX) ** 2 + (lineEndY - lineStartY) ** 2
    u = numer / denom
    interX = lineStartX + u * (lineEndX - lineStartX

# Returns a vector of the maximum values from two vectors

def min_max_compare_v(data, minV, maxV):
    maxCompareM = [2, len(data)]
    minCompareM = [2, len(data)]

    if len(maxV) == 0:
        maxV = data
        minV = data
    else:
        maxCompareM[1] = maxV
        maxCompareM[2] = data
        maxV = max(maxCompareM)

        minCompareM[1] = minV
        minCompareM[2] = data


# Returns a vector of magitudes from a complex vector of quadriture values

def get_mag_cv(complex_v):
    return 20*np.log10(np.abs(complex_v))

# Returns the position of a target from a given intial position, velocity of motion, angle of motion,
# the difference in time since the initial time

def get_curr_pos(x, y, z, v, ang, deltaT):
    pos = [2, x + (v * math.sin(ang) * deltaT), y + (v * math.cos(ang) * deltaT)]
    return pos

# Returns the current position of a breathing target
# rate = rate of breathing motion in cycles per minute
# displacement = size of breathing motion in meters

def get_curr_pos_breath(x, y, z, rate, throw, ang, curr_t):
    in2m = 0.0254
    rateFactor = rate / 10
    pos = [2, x + (throw * in2m * math.sin(curr_t * rateFactor) * math.sin(ang)), y + (throw * in2m * math.sin(curr_t * rateFactor) * math.cos(ang))]
    return pos
```

Lib-Math.lau.py (continued)

```
# Returns the calculated RMS from a vector of values

def calc_rms(v):
    v_squared = v ** 2
    rms = math.sqrt(v.mean(v_squared))
    return rms

# Returns the RMS of a time domain signal

def calc_sig_rms(data_time, sig_dist, time_step):
    sig_time = sig_dist * c_in_nano_sec_per_m * 2
    sig_i = get_time_index(sig_time, time_step)
    if sig_i < 1:
        sig_i = 1
    data_target_sig = CM.slice2(data_time, sig_i, sig_i)
    data_target_sig_amp = M.transpose(CM
```

Potentially Useful Equations

- Dist_point_to_line
- get_curr_pos

Lib-Tape.lua.py

```
# Title: Tape Functions
# Filename: Lib-Tape.lua
# Description: Functions related to data tapes
# Version 1.1.3, 03/30/2011
# Author: Patton Gregg
# Revision History:
#       1.1.3, 03/30/2011
#               Added GetTSClipTape function to return a data tape over a specified time range
#       1.1.2, 01/27/2011
#               Added SubtractRunAvgTape function to subtract the running average from a data tape
#       1.1.1, 01/11/2011
#               Added functionality to exclude points in specified band for FreqSliceTape function
#               Updated TimeWindowTape to account for cable delays
#       1.1.0, 12/17/2010
#               Changed MinMax... functions to be more generalized and no longer generate chart windows
#       1.0.1, 12/16/2010
#               Added min_max_... functions from Tool-Data-Proc and renamed MinMaxTape...
#               Moved bg_sub function from Tool-Data-Proc and renamed SubtractTape
#               Fixed a bug in SubtractTape that would ignore incomplete frames
#               Update to reflect change in function name SliceFreqRange to FreqSliceScan
#       1.0.0, 11/30/2010
#               Initial Release
# ------------------------------------------------------------
# Include Files
# ------------------------------------------------------------

import lupa
import math
import radar
import scan
import header

# ------------------------------------------------------------
# Function List
# ------------------------------------------------------------
# AvgTape (tape) -> tape
# FreqSliceTape (tape, frequency_start, frequency_stop, exclude_band_bool) -> tape
# NormTape (tape, norm_tape, frame_number_for_norm_tape) -> tape
# ScaleTape (tape, scale_factor) -> tape
# SubtractTape (tape, subTape) -> tape
# SubtractRunAvgTape (tape, nScans) -> tape
# TimeWindowTape (tape, win_start, win_stop, window_bool) -> tape
# GetClipTape (tape, frame_start, frame_stop) -> tape
# GetTSClipTape (tape, timestamp_start, timestamp_stop) -> tape
# GetLastNFramesClipTape (tape, nFrames) -> tape
# GetComboTape (tape, tx, rx, tx_port, rx_port) -> tape
# GetFrame (tape, frameNum) -> tape
# GetComboTable (tape) -> table
# GetClipTapeI (tape, frame_start, frame_stop) -> vector
# GetComboTapeI (tape, tx, rx) -> vector
# MinMaxTapeFreqDom (tape) -> min_vector, max_vector
# MinMaxTapeTimeDom (tape) -> min_vector, max_vector
# ------------------------------------------------------------
```

```
# Returns a tape of a single frame with the average of all the combos from the input tape

def avg_tape(tape):
    frame = get_last_n_frames_clip_tape(tape, 1)
    for i in range(1, len(frame)):
        combo_tape = get_combo_tape(tape, frame[i].TX, frame[i].RX, frame[i].TX_PORT, frame[i].RX_PORT)
        scan = frame[i]
        scan.DATA = CM.avg(DT.get_matrix(combo_tape))


# Returns a tape that is the output of having a tape normalized by a frame from another tape

def norm_tape(tape, norm_tape, norm_frame):
    norm_tape_frame = get_clip_tape(norm_tape, norm_frame, norm_frame)
    if tape.get_combo_table(tape)!= norm_tape_frame:
        raise forms.error("Error! Incomplete frame chosen for normalization frame!")
    tape_out = data_tape()
    data_tape.copy_header(tape_out, tape)
    for i in range(1,

# Returns a tape that is the a tape scaled by a specified factor

def scale_tape(tape, scale_factor):
    tape_out = data_tape()
    data_tape.copy_header(tape_out, tape)
    for i in range(1, len(tape)):
        scan = tape[i]
        scan.DATA = tape[i].DATA * scale_factor
        data_tape.append(tape_out, scan)
    return tape_out
<|cpp|>#include "stdafx.h"
#include "CppUnitTest.

# Returns a tape that has had the data time windowed

def time_window_scan(data, win_start, win_stop, freq_step, window):
    return data[window:win_stop + window]

# Returns a tape of only the specified range of frames

def get_ts_clip_tape(tape, timestamp_start, timestamp_stop):
    if timestamp_start > timestamp_stop:
        temp = timestamp_start
        timestamp_start = timestamp_stop
        timestamp_stop = temp
    elseif timestamp_stop == None:
        timestamp_stop = tape[-1].TIMESTAMP
    return tape[lambda x: x.TIMESTAMP >= timestamp_start and x.TIMESTAMP <= timestamp_stop]

# Returns a tape of only the specified range of frames

def get_clip_tape(tape, frame_start, frame_stop):
    if frame_start > frame_stop:
        temp = frame_start
        frame_start = frame_stop
        frame_stop = temp
    elseif frame_stop == None:
        frame_stop = frame_start
    return tape[lambda x: x.frame_number >= frame_start and x.frame_number <= frame_stop]
```

Lib-Tape.lau.py (continued)

```
# Returns a tape of only the last N frames

def get_last_n_frames_clip_tape(tape, n_frames):
    combo_table = data_tape.get_combo_table(tape)
    n_combo = len(combo_table)
    frame_end = tape[-1].frame_number
    frame_start = max(frame_end - n_frames + 1, 1)
    clip_tape = get_clip_tape(tape, frame_start, frame_end)
    return clip_tape

# Returns a tape of only the specified combinations

def get_combo_tape(tape, tx, rx, tx_port, rx_port):
    return tape[lambda x: x.TX == tx and x.RX == rx and x.TX_PORT == tx_port and x.RX_PORT == rx_port]

# Returns a tape of a single specified frame

def get_frame(tape, frame_num):
    return tape[frame_num]

# Returns a table of all valid combinations in the input tape

def get_combo_table(tape):

# The built in get_combo_table function does not return PORT assignments
# This variable is used just to ensure we have the same number of scans in the frame we get

        ct = data_tape.get_combo_table(tape)
        numScans = len(ct)

        singleFrame = get_last_n_frames_clip_tape(tape, 1)

        comboTable = {}
        for i in range(1, len(singleFrame)):
                comboTable[i] = {}
                comboTable[i]['TX'] = singleFrame[i]['TX']
                comboTable[i]['

# Returns a vector of scan indexes for the specfied range of frames

                                def get_clip_tape_i(tape, frame_start, frame_stop):
    I = [i for i in range(len(tape)) if tape[i].frame_number >= frame_start and tape[i].frame_number <= frame_stop]
    return I

# Returns a vector of scan indexes for scans of the specified TX-RX combination

def get_combo_tape_i(tape, tx, rx):
    I = [i for i, x in enumerate(tape) if x.tx == tx + 1 and x.rx == rx + 1]
    return I

# Returns the vectors for the minimum and maximum frequency values for the input tape

def min_max_tape_freq_dom(tape):
    max_compare_m = M(2, len(tape[1].DATA))
    min_compare_m = M(2, len(tape[1].DATA))
    freq_data = None
    freq_data_mag = None
    max_v = V()
    min_v = V()

    if len(tape) < 1000:
        freq_data = data_tape.get_matrix(tape)
        fre
```

Files Converted From C++ → Python

- Lib files converted so far
- Specifically:
  - String
  - Type
  - Plot
  - Table
  - Radar

Lib-String.lua

```lua
-- Title: String Functions
-- Filename: Lib-String.lua
-- Description: Functions related to manipulating strings
-- Version 1.0.0, 02/24/2011
-- Author: Patton Gregg
-- Revision History:
--       1.0.0, 02/24/2011
--              Initial Release
-------------------------------------------------------------
-- Include Files
-------------------------------------------------------------
dofile("LuaProg/Lib-Header.lua")
-------------------------------------------------------------
-- Function List
-------------------------------------------------------------
-- GetCommaSepNums (str) = vector, bool
-------------------------------------------------------------

-- Takes a string of comma separated numbers and returns a vector of the numbers
function GetCommaSepNums (str)

        local value = V()
        local j = 0
        local buff
        local state = 1
        local err = false
        while true do
                _, j, char = string.find(str, "([%p%d ])", j+1)

                if (char == " ") then
                        if (state == 2 or state == 3) then
                                V.append(value, buff)
                                buff = ""
                        end
                        state = 1
                elseif (IsNum(char)) then
                        if (state == 1) then
                                buff = char
                                state = 2
                        elseif (state == 2 or state == 3) then
                                buff = buff .. char
                        end
                elseif (char == ",") then
                        if (state == 1) then
                                err = true
                                break
                        elseif (state == 2 or state == 3) then
                                V.append(value, buff)
                                buff = ""
                                state = 1
                        end
                elseif (char == ".") then
                        if (state == 2) then
                                buff = buff .. char
                                state = 3
                        else
                                err = true
                                break
                        end
                elseif (char == nil) then
                        if (state == 1) then
                                err = true
                        elseif (state == 2 or state == 3) then
                                V.append(value, buff)
                        end
                        break
                end
        end

        return value, err
end
```

```python
# Title: String Functions
# Filename: Lib-String.lua
# Description: Functions related to manipulating strings
# Version 1.0.0, 02/24/2011
# Author: Patton Gregg
# Revision History:
#       1.0.0, 02/24/2011
#              Initial Release
# -------------------------------------------------------------
# Include Files
# -------------------------------------------------------------
import lua
lua.dofile("LuaProg/Lib-Header.lua")
# -------------------------------------------------------------
# Function List
# -------------------------------------------------------------
# GetCommaSepNums (str) = vector, bool
# -------------------------------------------------------------

# Takes a string of comma separated numbers and returns a vector of the numbers

def get_comma_sep_nums(str):
    value = Value()
    j = 0
    buff = ""
    state = 1
    err = False
    while True:
        _, j, char = string.find(str, "([%p%d ])", j+1)

        if char == " ":
    if state == 2 or state == 3:
        V.append(value, buff)
        buff = ""
    state = 1
else:
    if state == 1:
        buff = char
        state = 2
    elif state == 2 or state == 3:
        buff = buff + char

        def elseif_char_dot(char, state, buff, value, err):
    if char == '.':
        if state == 2:
            if state == 3:
                buff = buff + char
                state = 3
            else:
                err = True
                break
        else:
            err = True
            break
    else:
        if state == 1:
            err = True
        elif state == 2 or state == 3:
            value.append(buff)
            buff = char
            state = 2
        else:
```

Lib-Type.lua

```lua
-- Title: Type Functions
-- Filename: Lib-Type.lua
-- Description: Functions for checking and converting variable types
-- Version 1.0.0, 11/30/10
-- Author: Patton Gregg
-- Revision History:
--      1.0.0, 11/30/2010
--              Initial Release
------------------------------------------------------------
-- Include Files
------------------------------------------------------------
dofile("LuaProg/Lib-Header.lua")
------------------------------------------------------------
-- Function List
------------------------------------------------------------
-- BoolToInt(bool) -> int
-- BoolToString (int) -> string
-- IntToBool(int) -> bool
-- IsBool (variable) -> bool
-- IsNum (variable) -> bool
-- IsDataFile (file_path_string) -> bool
-- StringToBool (string) -> bool
------------------------------------------------------------

-- Checks to see if variable is a bool
function IsBool(v)

        if string.upper(v) == "TRUE" or string.upper(v) == "FALSE" then
                return true
        end

        return false
end

-- Checks to see if variable is a number
function IsNum (v)
        return type(tonumber(v)) == "number"
end

-- Checks to see if the file is an APRD data file
function IsDataFile (filename)

        if file.get_ext(filename) == ".imb" then
                return true
        elseif file.get_ext(filename) == ".img" then
                return true
        elseif file.get_ext(filename) == ".imx" then
                return true
        else
                return false
        end
end

-- Returns a bool from a string input
function StringToBool (s)

        if string.upper(s) == "TRUE" then
                return true
        end

        return false
end

-- Returnd string from a bool input
function BoolToString (v)

        if v then
                return "TRUE"
        end

        return "FALSE"
end

-- Returns bool from a number input
function IntToBool(v)

        if v == 1 then
                return true
           end

        return false
end

-- Returns number from a bool input
function BoolToInt(v)

        if v then
                return 1
        end

        return 0
end
```

```python
# Author: Patton Gregg
# Revision History:
#       1.0.0, 11/30/2010
#               Initial Release
# ------------------------------------------------------------
# Include Files
# ------------------------------------------------------------

import lua
lua.dofile("LuaProg/Lib-Header.lua")

# ------------------------------------------------------------
# Function List
# ------------------------------------------------------------
# BoolToInt(bool) -> int
# BoolToString (int) -> string
# IntToBool(int) -> bool
# IsBool (variable) -> bool
# IsNum (variable) -> bool
# IsDataFile (file_path_string) -> bool
# StringToBool (string) -> bool
# ------------------------------------------------------------

# Checks to see if variable is a bool
def is_bool(v):
    if v.upper() == 'TRUE' or v.upper() == 'FALSE':
        return True
    else:
        return False

# Checks to see if variable is a number
def is_num(v):
    return type(tonumber(v)) == "number"

# Checks to see if the file is an APRD data file
def is_data_file(filename):
    if filename.endswith('.imb') or filename.endswith('.img') or filename.endswith('.imx'):
        return True
    else:
        return False

# Returns a bool from a string input
def string_to_bool(s):
    if s.upper() == 'TRUE':
        return True
    else:
        return False

# Returns string from a bool input
def bool_to_string(v):
    if v:
        return "TRUE"
    else:
        return "FALSE"

# Returns bool from a number input
def int_to_bool(v):
    if v == 1:
        return True
    else:
        return False

# Returns number from a bool input
def bool_to_int(v):
    if v:
        return 1
    else:
        return 0
```

Lib-Plot.lua

```lua
-- Title: Plotting Functions
-- Filename: Lib-Plot.lua
-- Description: Functions related to plotting
-- Version 1.0.0, 12/17/10
-- Author: Patton Gregg
-- Revision History:
--       1.0.0, 12/17/2010
--              Initial Release
------------------------------------------------------------
-- Include Files
------------------------------------------------------------
dofile("LuaProg/Lib-Header.lua")
------------------------------------------------------------
-- Function List
------------------------------------------------------------
-- ChartTraces (chartType, chartName, xLabel, yLabel, xPts, xMin, xMax, yMin, yMax, ...) -> chart_handle
------------------------------------------------------------

-- Plots a chart with the traces specified
-- Usage: ChartTraces (chartType, chartName, xLabel, yLabel, xPts, xMin, xMax, yMin, yMax, <dataVector>, <dataLabel1>, <dataLabel2>)
-- Note: <dataVector>, <dataLabel1> and <dataLabel2> ALL REQUIRED for EACH trace that is to be plotted
function ChartTraces (chartType, chartName, xLabel, yLabel, xPts, xMin, xMax, yMin, yMax, ...)

        local w = chart(chartType, chartName, FL_CHART_WINDOW)
        chart.clear(w)
        chart.set_xlabel(w, xLabel)
        chart.set_ylabel(w, yLabel)

        -- chart.add(w, xPts, [dataV, dataLabel1, dataLabel2])
        for i = 1, arg.n, 3 do
                print("arg[i]", arg[i])
                chart.add(w, xPts, arg[i], arg[i+1], arg[i+2])
        end

        chart.set_scale(w, xMin, xMax, yMin, yMax)
        chart.update(w)

        return w
end
```

```python
# Title: Plotting Functions
# Filename: Lib-Plot.lua
# Description: Functions related to plotting
# Version 1.0.0, 12/17/10
# Author: Patton Gregg
# Revision History:
#       1.0.0, 12/17/2010
#              Initial Release
# ------------------------------------------------------------
# Include Files
# ------------------------------------------------------------
import lua
lua.dofile("LuaProg/Lib-Header.lua")
# ------------------------------------------------------------
# Function List
# ------------------------------------------------------------
# ChartTraces (chartType, chartName, xLabel, yLabel, xPts, xMin, xMax, yMin, yMax, ...) -> chart_handle
# ------------------------------------------------------------

# Plots a chart with the traces specified
# Usage: ChartTraces (chartType, chartName, xLabel, yLabel, xPts, xMin, xMax, yMin, yMax, <dataVector>, <dataLabel1>, <dataLabel2>)
# Note: <dataVector>, <dataLabel1> and <dataLabel2> ALL REQUIRED for EACH trace that is to be plotted
def chart_traces(chart_type, chart_name, x_label, y_label, x_pts, x_min, x_max, y_min, y_max, *args):
    w = chart(chart_type, chart_name, FL_CHART_WINDOW)
    chart.clear(w)
    chart.set_xlabel(w, x_label)
    chart.set_ylabel(w, y_label)
    for i in range(0, len(args), 3):
        chart.add(w
```

Useful Function: chart_traces

- Takes chartType, chartName, xLabel, yLabel, xPts, xMin, xMax, yMin, yMax, <dataVector>, <dataLabel1>, <dataLabel2>
  - Then plots a chart with the traces specified

Lib-Table.lua

```lua
-- Include Files
-----------------------------------------------------------
dofile("LuaProg/Lib-Header.lua")
-----------------------------------------------------------
-- Function List
-----------------------------------------------------------
-- DisplayTable(table, desc_string) -> VOID
-- AppendTable (table1, table) -> modified table1 .. table
-----------------------------------------------------------

-- Prints the tables contents to the console
function DisplayTable(t, desc)
        function DisplayTableL2(desc, t)
                for k,v in pairs(t) do
                        print(string.format("%20s %20s", desc, k), v)
                end
        end

        print("-----------------------------------------------------")
        print(desc)
        print("-----------------------------------------------------")
        for k, v in pairs(t) do
                if L.is_table(v) then
                        DisplayTableL2(k, v)
                else
                        print(string.format("%20s", k), v)
                end
        end
end

-- Appends table2 to table1
function AppendTable (table1, table2)

        for k,v in pairs(table2) do
                table1[k] = v
        end

        return table1
end
```

```python
# Title: Table Functions
# Filename: Lib-Table.lua
# Description: Functions for Lua tables
# Version 1.0.0, 11/30/10
# Author: Patton Gregg
# Revision History:
#       1.0.0, 11/30/2010
#               Initial Release
# -----------------------------------------------------------
# Include Files
# -----------------------------------------------------------
import lua
lua.dofile("LuaProg/Lib-Header.lua")
# -----------------------------------------------------------
# Function List
# -----------------------------------------------------------
# DisplayTable(table, desc_string) -> VOID
# AppendTable (table1, table) -> modified table1 .. table
# -----------------------------------------------------------

# Prints the tables contents to the console
def display_table(t, desc):
    print("-----------------------------------------------------")
    print(desc)
    print("-----------------------------------------------------")
    for k, v in t.items():
        if isinstance(v, dict):
            display_table(v, k)
        else:
            print(k, v)
    print("-----------------------------------------------------")

# Appends table2 to table1
def append_table(table1, table2):
    for k,v in pairs(table2):
        table1[k] = v
    return table1
```

Lib-Radar.lua (C++ of Range, Frequency, Time, TDD and Gating, Functions)

```lua
-- Range Functions
-------------------------------------------------------------
-- Returns the range resolution for the specfied bandwidth
function CalcRangeRes (bandwidth)
        return (c / (2 * bandwidth))
end

-- Returns the maximum unambiguous range for the specified frequency step
function CalcMaxRange (freqStep)
        return (c / (2 * freqStep))
end

-------------------------------------------------------------
-- Frequency Functions
-------------------------------------------------------------
-- Returns the index location for a vector of frequency points with the given frequency step and start frequency
function GetFreqIndex (freq, freq_step, freq_start)
        return (math.floor(((freq - freq_start) / freq_step) + 1))
end

-- Returns the frequency value for an index location from vector of frequency points with the given frequency step and start frequency
function GetIndexFreq (index, freq_step, freq_start)
        return (freq_start + ((index - 1) * freq_step))
end

-------------------------------------------------------------
-- Time Functions
-------------------------------------------------------------
-- Returns the index location for a vector of time domain range bins with the given time step
function GetTimeIndex (t, timeStep)
        return (math.floor(t / timeStep) + 1)
end

-- Returns the time for a index location from a vector of time domain range bins with the given time step
function GetIndexTime (index, timeStep)
        return (index - 1) * timeStep
end
```

```lua
-- TDD and Gating Functions
-------------------------------------------------------------
-- Rounds the pulse width (ns) to the nearest TDD clock cycle
function RoundToGateClock(pulseWidth) return Round(pulseWidth / GateClock) * GateClock end

-- Returns the gate values for the desired signal distances
-- Note: Gate values are NOT guaranteed to not fall on a harmonic. For a more robust calculation of the gate
-- parameters, use GetGateParams function
function CalcGateParams (startSigDist, startFullSigDist, endFullSigDist, txCableDelay, rxCableDelay, maxDuty)

        local cableDelay = (txCableDelay + rxCableDelay)

        if (maxDuty) then
                startFullSigDist = endFullSigDist
        end

        tx1GateDelay = RoundToGateClock((2 * startSigDist * cInNanoSecPerM - 1.5 * GateClock) + (cableDelay))
        tx1Gate = RoundToGateClock((2 * startFullSigDist * cInNanoSecPerM) + (cableDelay) - (tx1GateDelay + 1.5 * GateClock))
        rx1Gate = RoundToGateClock((2 * endFullSigDist * cInNanoSecPerM) + (cableDelay) - (tx1GateDelay + 1.5 * GateClock)) + (3 * GateClock)
        rx2GateDelay = 3 * GateClock

        rx2Gate = 0
        rx1GateDelay = 0

        return tx1Gate, tx1GateDelay, rx1Gate, rx1GateDelay, rx2Gate, rx2GateDelay
end

-- Returns the range distances for the signal stop, start, and full power signal start, stop for the
-- specified gate values
function CalcGateDist (txGate, txRxGateDelay, rx1Gate, rx2TxGateDelay, txCableDelay, rxCableDelay)

        local rx1Gate = rx1Gate - (3 * GateClock)
        local txRxGateDelay = txRxGateDelay + (1.5 * GateClock)

        startSigDist = (txRxGateDelay - (txCableDelay + rxCableDelay)) / (2 * cInNanoSecPerM)
        startFullSigDist = (txGate + txRxGateDelay - (txCableDelay + rxCableDelay)) / (2 * cInNanoSecPerM)
        endFullSigDist = (txRxGateDelay + rx1Gate - (txCableDelay + rxCableDelay)) / (2 * cInNanoSecPerM)
        endSigDist = (txGate + rx1Gate + txRxGateDelay - (txCableDelay + rxCableDelay)) / (2 * cInNanoSecPerM)

        return startSigDist, startFullSigDist, endFullSigDist, endSigDist
```

Lib-Radar.lua (Python of Range, Frequency, Time, TDD and Gating, Functions)

```
# Returns the maximum unambiguous range for the specified frequency step
def calc_max_range(freq_step):
    return c / (2 * freq_step)

# ------------------------------------------------------------
# Frequency Functions
# ------------------------------------------------------------
# Returns the index location for a vector of frequency points with the given frequency step and start frequency
def get_freq_index(freq, freq_step, freq_start):
    return math.floor(((freq - freq_start) / freq_step) + 1)

# Returns the frequency value for an index location from vector of frequency points with the given frequency step and start frequency
def get_index_freq(index, freq_step, freq_start):
    return freq_start + ((index - 1) * freq_step)

# ------------------------------------------------------------
# Time Functions
# ------------------------------------------------------------
# Returns the index location for a vector of time domain range bins with the given time step
def get_time_index(t, time_step):
    return int(math.floor(t / time_step) + 1)

# Returns the time for a index location from a vector of time domain range bins with the given time step
def get_index_time(index, time_step):
    return (index - 1) * time_step

# ------------------------------------------------------------
# TDD and Gating Functions
# ------------------------------------------------------------
# Rounds the pulse width (ns) to the nearest TDD clock cycle
def round_to_gate_clock(pulse_width):
    return round(pulse_width / gate_clock) * gate_clock

# Returns the gate values for the desired signal distances
# Note: Gate values are NOT guaranteed to not fall on a harmonic. For a more robust calculation of the gate
# parameters, use GetGateParams function
def calc_gate_params(start_sig_dist, start_full_sig_dist, end_full_sig_dist, tx_cable_delay, rx_cable_delay, max_duty):
    cable_delay = (tx_cable_delay + rx_cable_delay)
    if max_duty:
        start_full_sig_dist = end_full_sig_dist
    end
def deduplicate_list(input_list):
    deduplicated_list = list(set(input_list))
    print(deduplicated_list)

# Returns the range distances for the signal stop, start, and full power signal start, stop for the
# specified gate values
def calc_gate_dist(tx_gate, tx_rx_gate_delay, rx1_gate, rx2_tx_gate_delay, tx_cable_delay, rx_cable_delay):
        rx1_gate = rx1_gate - (3 * gate_clock)
        tx_rx_gate_delay = tx_rx_gate_delay + (1.5 * gate_clock)

        def start_sig_dist(tx_rx_gate_delay, tx_cable_delay, rx_cable_delay):
    start_sig_dist = (tx_rx_gate_delay - (tx_cable_delay + rx_cable_delay)) / (2 * cInNanoSecPerM)
    start_full_sig_dist = (tx_gate + tx_rx_gate_delay - (tx_cable_delay + rx_cable_delay)) / (2 * cInNanoSecPerM
```

Useful Function #1: get_freq_index

- Returns the index location for a vector of frequency points with the given frequency step and start frequency.

Useful Function #2: get_index_time

- Returns the time for an index location from a vector of time domain range bins with the given time step.

Lib-Radar.lua (C++ of Sensor Functions)

```lua
-- Sensor Functions
-------------------------------------------------------------
-- Returns the maximum distance between active elements for the given sensor table
function GetMaxAntDist (sensTable)

    local maxDist = 0

    for j = 1, #sensTable do
        for k = 2, #sensTable do
            if sensTable[j].TX == true and sensTable[k].RX == true then
                local x1 = sensTable[j].X
                local y1 = sensTable[j].Y
                local z1 = sensTable[j].Z
                local x2 = sensTable[k].X
                local y2 = sensTable[k].Y
                local z2 = sensTable[k].Z
                local dist = Dist3D(x1, y1, z1, x2, y2, z2)
                if dist > maxDist then
                    maxDist = dist
                end
            end
        end
    end

    return maxDist
end

-- Returns the maximum transmit cable delay for the active transmitting elements for the given sensor table
function GetMaxTXDelay (sensTable)

    local maxTXDelay = 0

    for i = 1, #sensTable do
        if sensTable[i].TX_DELAY > maxTXDelay and sensTable[i].TX == true then
            maxTXDelay = sensTable[i].TX_DELAY
        end
    end
```

```lua
-- Returns the maximum receive cable delay for the active receiving elements for the given sensor table
function GetMaxRXDelay (sensTable)

    local maxRXDelay = 0

    for i = 1, #sensTable do
        if sensTable[i].RX_DELAY > maxRXDelay and sensTable[i].RX == true then
            maxRXDelay = sensTable[i].RX_DELAY
        end
    end

    return maxRXDelay
end

-- Returns the average time, in seconds, between scans
function GetAvgScanTime(tape)

    local comboTape = GetComboTape(tape, tape[1].TX, tape[1].RX, tape[1].TX_PORT, tape[1].RX_PORT)

    local ts = comboTape[TIMESTAMP]
    local deltaTS = V.slice(ts, 2, #ts) - V.slice(ts, 1, #ts - 1)
    local avgScanTime = V.avg(deltaTS)

    return avgScanTime
end
```

Lib-Radar.lua (Python of Sensor Functions)

```python
# Sensor Functions
# ------------------------------------------------------
# Returns the maximum distance between active elements for the given sensor table
def get_max_ant_dist(sens_table):
    max_dist = 0
    for j in range(1, len(sens_table)):
        for k in range(j + 1, len(sens_table)):
            if sens_table[j]['TX'] and sens_table[k]['RX']:
                x1 = sens_table[j]['X']
                y1 = sens_table[j]['Y']
                z1 = sens_table[j]['Z']
                x2 =

# Returns the maximum transmit cable delay for the active transmitting elements for the given sensor table
def get_max_tx_delay(sens_table):
    max_tx_delay = 0
    for i in range(1, len(sens_table)):
        if sens_table[i]['TX'] == True and sens_table[i]['TX_DELAY'] > max_tx_delay:
            max_tx_delay = sens_table[i]['TX_DELAY']
    return max_tx_delay

# Returns the maximum receive cable delay for the active receiving elements for the given sensor table
def get_max_rx_delay(sens_table):
    max_rx_delay = 0
    for i in range(1, len(sens_table)):
        if sens_table[i]['RX'] and sens_table[i]['RX_DELAY'] > max_rx_delay:
            max_rx_delay = sens_table[i]['RX_DELAY']
    return max_rx_delay

# Returns the average time, in seconds, between scans
def get_avg_scan_time(tape):
    combo_tape = get_combo_tape(tape, tape[1].tx, tape[1].rx, tape[1].tx_port, tape[1].rx_port)
    ts = combo_tape[TIMESTAMP]
    delta_ts = V[ts][2] - V[ts][1]
    avg_scan_time = V.avg(delta_ts)
    return avg_scan_time
```

get_max_tx_delay takes a sensor table and returns the maximum transmit cable delay for the active transmitting elements.

get_max_rx_delay takes a sensor table and returns the maximum receive cable delay for the active receiving elements.

Lib-Radar.lua (C++ of Radar Signal Functions)

```lua
-- Radar Signal Functions
-----------------------------------------------------
-- Calculates the expected two-way path loss for the specified values
function CalcPathLoss(tx_gain, rx_gain, rcs, rcs_power, freq, dist, pow_r)

    local path_loss = 10*math.log10((tx_gain * rx_gain * rcs^(rcs_power) * CalcLambda(freq)^2) / ((4*math.pi)^3 * dist^(pow_r)))

    return path_loss
end

-- Returns the expected response that would be seen by the radar for a target
function GetTargetResp (dist, amp, f_start, f_step, num_pts)

    local t = dist / c
    local freq = V(num_pts, f_start, f_step) * 1e6
    local phase = 2 * math.pi * freq * t
    local echo_i = amp * V.cos(phase)
    local echo_q = -amp * V.sin(phase)
    -- local echo_i = -amp * V.sin(phase)
    -- local echo_q = amp * V.cos(phase)
    local echo = CV(echo_i, echo_q)

    -- local win_multi = 0.5 * (1 - math.cos(2.0 * math.pi * (i-1) /(num_pts-1)));
    -- echo_win = echo * win_multi

    return echo
    -- return echo_win
end
```

```lua
-- Returns the wavelength for a given frequency
function CalcLambda (freq)
    return c / freq
end

-- Returns the expected amplitude of a target signal
function CalcAmp (dist, rcs, rcs_power, freq, tx_gain, rx_gain, tx_pow, loss_factor, rx_fs_pow, pow_r)

    local pow_rx = tx_pow + CalcPathLoss(tx_gain, rx_gain, rcs, rcs_power, freq, dist, pow_r) + 10*math.log10(loss_factor)
    -- rx_pow_fs_mw = 10^(rx_fs_pow/10) -- convert rx full-scale power from db to milli-watts

    return 1000*10^(pow_rx/10)
end
```

Lib-Radar.lua (Python of Radar Signal Functions)

```
# Radar Signal Functions
# -----------------------------------------------------

# Calculates the expected two-way path loss for the specified values
def calc_path_loss(tx_gain, rx_gain, rcs, rcs_power, freq, dist, pow_r):
        path_loss = 10 * math.log10((tx_gain * rx_gain * rcs ** (rcs_power) * math.pow(CalcLambda(freq), 2) / (math.pow(4 * math.pi, 3) * math.pow(dist, pow_r))))
        return path_loss

# Returns the expected response that would be seen by the radar for a target
def get_target_resp(dist, amp, f_start, f_step, num_pts):
    t = dist / c
    freq = f_start * 1e6
    phase = 2 * math.pi * freq * t
    echo_i = amp * math.cos(phase)
    echo_q = -amp * math.sin(phase)
    # local echo_i = -amp * V.sin(phase)
    # local echo_q = amp * V.cos(phase)
    echo = np.array([echo_i, echo_q])
    # local win_multi = 0.5 * (1 - math.cos(2.0 * math.pi * (i-1) /(num_pts-1)));
    # echo_win = echo * win_multi
    return echo
    # return echo_win

# Returns the wavelength for a given frequency
def calc_lambda(freq):
    return c / freq

# Returns the expected amplitude of a target signal
def calc_amp(dist, rcs, rcs_power, freq, tx_gain, rx_gain, tx_pow, loss_factor, rx_fs_pow, pow_r):
        pow_rx = tx_pow + calc_path_loss(tx_gain, rx_gain, rcs, rcs_power, freq, dist, pow_r) + 10*math.log10(loss_factor)
        return 1000*10**(pow_rx/10)
```

calc_lambda takes a frequency and returns the wavelength.

Lib-Radar.lua (C++ of Signal Processing Functions)

```lua
-- Signal Processing Functions
-------------------------------------------------------------
-- Returns the doppler frequency delta
function CalcDoppFreqDelta (deltaT, nScans)
    return 1 / (deltaT * nScans)
end

-- Returns the maximum doppler frequency for a given time step between scans
function CalcDoppFreqMax (deltaT)
    return 1 / (2 * deltaT)
end

-- Returns the expected doppler shift
function CalcDoppShift (freqMin, targetVel)
    return (2 * targetVel * freqMin) / c
end

-- Transforms time domain data, that had been zero filled, back to the frequency
-- domain, and return only the valid frequency data (i.e. no zeroed data)
function FFTZeroFill (timeData, nFFTPts, freqStart, freqStop, nFreqPts)

    local freqStep = (freqStop - freqStart) / (nFreqPts - 1)
    local nZeros = math.floor(freqStart / freqStep)

    local freqData
    if L.is_complex_vector(timeData) then
        freqData = CV.fft(timeData, nFFTPts, false, false)
        freqData = CV.slice(freqData, nZeros + 1, nZeros + nFreqPts)
    else
        freqData = CM.fft(timeData, nFFTPts, false, false)
        freqData = CM.slice2(freqData, nZeros + 1, nZeros + nFreqPts)
    end

    return freqData
end
```

```lua
-- Returns a matrix result for the 2D FFT of a tape
function Get2DFFT (tape, nFFT, zeroFill, window)

    local nFFTPts = 0

    local freq_matrix = data_tape.get_matrix(tape)
    local nScan = Order2N(N2OrderFloor(#tape))
    freq_matrix = CM.slice(freq_matrix, (#freq_matrix - nScan)+1)

    local time_matrix = CM()
    if (zeroFill) then
        local nPoints, F0, F1, SR = data_tape.get_scan_parameters(tape)
        local fStep = data_tape.get_freq_step(tape)
        nFFTPts = Order2N(N2Order(F1 / fStep))
        if (nFFT > nFFTPts) then
            nFFTPts  = Order2N(N2Order(nFFT))
        end
        time_matrix = CM.fft(freq_matrix, nFFTPts, F0, F1, window, true)
    else
        nFFTPts = Order2N(N2Order(nFFT))
        time_matrix = CM.fft(freq_matrix, nFFTPts, window, true)
    end

    -- local doppler_matrix = CM.fft2(time_matrix, nScan)
    local doppler_matrix = CM.transpose(time_matrix)
    doppler_matrix = CM.fft(doppler_matrix, nScan, false, false)
    doppler_matrix = CM.transpose(doppler_matrix)

    return doppler_matrix
end
```

Lib-Radar.lua (C++ of Signal Processing Functions) (Continued)

```lua
-- Returns the DC component from a tape
function GetDCComponent (tape)

    local nPoints, F0, F1, SR = data_tape.get_scan_parameters(tape)
    local fStep = data_tape.get_freq_step(tape)
    local nFFTPts = Order2N(N2Order(F1 / fStep))

    local zeroFillFFT = true
    local windowFFT = false
    local doppler_matrix = Get2DFFT(tape, nFFTPts, zeroFillFFT, windowFFT)

    -- Extract the DC component.
    local dc_component = doppler_matrix[1]

    local freq_vector = FFTZeroFill(dc_component, nFFTPts, F0, F1, nPoints) / #doppler_matrix

    return freq_vector
end

-- Returns the sum of the components from a specified doppler frequency range from a 2D FFT
-- Frequencies specified in Hz
function GetMotionComponent (tape, minMotionFreq, maxMotionFreq)

    local nPoints, F0, F1, SR = data_tape.get_scan_parameters(tape)
    local fStep = data_tape.get_freq_step(tape)
    local nFFTPts = Order2N(N2Order(F1 / fStep))

    local doppler_matrix = Get2DFFT(tape, nFFTPts, true, false)

    -- Extract the Motion component
    -- Determine the doppler frequency values
    local deltaT = GetAvgScanTime(tape)
    local doppFreqDelta = CalcDoppFreqDelta(deltaT, #doppler_matrix)
    local doppFreqMax = CalcDoppFreqMax(deltaT)
```

```lua
    -- Calculate the row indices associated with the min and max doppler frequency ranges
    local iMinMotionFreqPos = math.min(math.floor(minMotionFreq / doppFreqDelta) + 2, #doppler_matrix)
    local iMaxMotionFreqPos = math.min(math.ceil(maxMotionFreq / doppFreqDelta) + 1, #doppler_matrix)
    local iMinMotionFreqNeg = math.min(#doppler_matrix - iMinMotionFreqPos + 2, #doppler_matrix)
    local iMaxMotionFreqNeg = math.min(#doppler_matrix - iMaxMotionFreqPos + 2, #doppler_matrix)

    -- Extract the negative and positive doppler frequency ranges
    local compPos = CM.slice(doppler_matrix, iMinMotionFreqPos, iMaxMotionFreqPos)
    local compNeg = CM.slice(doppler_matrix, iMaxMotionFreqNeg, iMinMotionFreqNeg)

    -- Combine the negative and positive components
    local motion_component = compPos
    CM.append(motion_component, compNeg)

    local freqData = FFTZeroFill(motion_component, nFFTPts, F0, F1, nPoints)

    -- Get the average frequency components to conserve scale
    freqData = CM.sum(freqData) / #motion_component

    return freqData
end

-- Returns the sum of the peak components (other than the DC component) from a 2D FFT
function GetMotionComponentPeak (tape)

    local nPoints, F0, F1, SR = data_tape.get_scan_parameters(tape)
    local fStep = data_tape.get_freq_step(tape)
    local nFFTPts = Order2N(N2Order(F1 / fStep))

    local doppler_matrix = Get2DFFT(tape, #tape[1], true, false)

    local doppler_matrix_ABS = CM.abs(doppler_matrix)
    doppler_matrix_ABS[1] = V(#doppler_matrix_ABS[1], 0)
```

```lua
    -- Calculate a threashold for the peak find function
    local minVal, maxVal, avgVal, stdVal = M.stats(doppler_matrix_ABS)
    local peakThreas = avgVal + 3 * stdVal
    local peaks = M.ipp_find_peaks(doppler_matrix_ABS, peakThreas)
    local peakI = V.unique(CV.real(peaks))

    local motion_component = CV(#doppler_matrix[1], C(0,0))
    for i = 1, #peakI do
        motion_component = motion_component + doppler_matrix[peakI[i]]
    end

    local freqData = FFTZeroFill (motion_component, nFFTPts, F0, F1, nPoints)
    if L.is_complex_matrix(motion_component) then
        freqData = CM.sum(freqData) / #motion_component
    end

    return freqData
end
```

Lib-Radar.lua (Python of Signal Processing Functions)

```
# Signal Processing Functions
# ------------------------------------------------------------

# Returns the doppler frequency delta
def calc_dopp_freq_delta(delta_t, n_scans):
    return 1 / (delta_t * n_scans)

# Returns the maximum doppler frequency for a given time step between scans
def calc_dopp_freq_max(deltaT):
        return 1 / (2 * deltaT)

# Returns the expected doppler shift
def calc_dopp_shift(freq_min, target_vel):
    return (2 * target_vel * freq_min) / c

# Transforms time domain data, that had been zero filled, back to the frequency
# domain, and return only the valid frequency data (i.e. no zeroed data)
def fft_zero_fill(time_data, n_fft_pts, freq_start, freq_stop, n_freq_pts):
    freq_step = (freq_stop - freq_start) / (n_freq_pts - 1)
    n_zeros = math.floor(freq_start / freq_step)
    freq_data = fft.fft(time_data, n_fft_pts, False, False)
    freq_data = fft.

# Returns the DC component from a tape
def get_dccomponent(tape):
    nPoints, F0, F1, SR = tape.get_scan_parameters()
    fStep = tape.get_freq_step()
    nFFTPts = Order2N(N2Order(F1 / fStep))

    zeroFillFFT = True
    windowFFT = False
    doppler_matrix = Get2DFFT(tape, nFFTPts, zeroFillFFT, windowFFT)
    # Extract the DC component.
    dc_component = doppler
```

calc_dopp_freq_max takes a time step between scans and returns the maximum doppler frequency.

get_dccomponent takes a tape and returns the DC component.

Tool-Window.lua (C++)

```lua
L = require("UTIL-Lua")
P = require("UTIL-Plot")

------------------------------------------------------
CV = complex_vector
 V = vector

CM = complex_matrix

-- SOURCE_NAME is bound to the analysis window
-- that called this script. We can access the window
-- and its contents using it.

if SOURCE_NAME == "" then
    window_manager.unregister(SOURCE_NAME, PROGRAM_NAME)
    forms.error("Not a stand-alone script. Must be run from an Analysis Window")
end

analysis_type = chart.get_analysis_type(SOURCE_NAME)
if analysis_type ~= MAGNITUDE_VS_TIME and analysis_type ~= MAGNITUDE_VS_DISTANCE then
    window_manager.unregister(SOURCE_NAME, PROGRAM_NAME)
    forms.error("Not a valid analysis type. Select either Time or Distance from View Menu.")
end

SP = chart.get_scan_parameter_table(SOURCE_NAME)
AP = chart.get_analysis_parameter_table(SOURCE_NAME)

--L.display("SP", SP)
--L.display("AP", AP)

function Process()
-- This script is only valid in the time and distance domains.
-----------------------------------------------------------------
x0, x1 = chart.get_scale(SOURCE_NAME)
```

```lua
-- Convert meters to nano-seconds.
if analysis_type == MAGNITUDE_VS_DISTANCE then
    x0 = x0 / 0.15
    x1 = x1 / 0.15
end
--print(x0, x1)

-- Get the raw complex freq domain data.
-------------------------------------------------------------------------------
cm = chart.get_matrix(SOURCE_NAME)

-- The selection vector contains ones and zeroes denoting
-- lines that are visible in the source window.
sv = chart.get_selection_vector(SOURCE_NAME)

-- Reduce the matrix to only selected rows.
-------------------------------------------------------------------------------
cs = CM.select(cm, sv)


-- We will need to know where to get titles for the selected vectors.
-- Convert the selection vector into an index vector by selecting from
-- a vector of indexes.
iv = V.select(V(#cm, 1, 1), sv)

-- Might as well get the line attributes now.
T1 = {}
T2 = {}
DT = {}
for i = 1, #iv do
    T1[i] = chart.get_title1(SOURCE_NAME, iv[i])
    T2[i] = chart.get_title2(SOURCE_NAME, iv[i])
    DT[i] = chart.get_delay(SOURCE_NAME, iv[i])
end
```

```lua
-- Convert everything to the time domain.
-----------------------------------------
ct = CM.fft( cs, AP.FFT_SIZE, SP.START_FREQ, SP.STOP_FREQ, AP.WINDOW_FFT, true  )
--print(cs, AP.FFT_SIZE, SP.START_FREQ, SP.STOP_FREQ, AP.WINDOW_FFT, true  )

-- We need these for scaling purposes.
-------------------------------------------------------------------------------
dF = data_source.get_freq_step(SP.SCAN_SIZE, SP.START_FREQ, SP.STOP_FREQ)
dT = data_source.get_time_step(AP.FFT_SIZE, SP.SCAN_SIZE, SP.START_FREQ, SP.STOP_FREQ)
--print(dF, dT)

-- Calculate the indexes of the bounding window.
-------------------------------------------------------------------------------
I0 = x0 / dT
I1 = x1 / dT

--I0 = math.max(0, I0)
--I1 = math.min(SP.SCAN_SIZE, I1)

-- Need to know 1/2 the width of the window.
dI = (I1 - I0) / 2

--print("I0, I1, dI (unadjusted) = ", I0, I1, dI)

-- Now, for each vector in the time domain, set the outside points to zero.
-------------------------------------------------------------------------------
for i = 1, #ct do

    -- Find the location of the peak within the bounding window.
    local vmax, imax = V.max( V.slice( CV.abs(ct[i]), I0, I1 ) )
    imax = imax + I0

    --print(vmax, imax)
```

Tool-Window.lua (C++) Continued

```
    -- Adjust I0, I1 w.r.t peak location.
    local J0 = imax - dI
    local J1 = imax + dI

    J0 = math.max(0, J0)
    J1 = math.min(AP.FFT_SIZE, J1)

    --print("J0, J1 (adjusted) = ", J0, J1)

    -- I is a vector of indexes {1 .. nFFT]
    local I = V(AP.FFT_SIZE, 1, 1)

    -- Reduce I to the indexes outside the boundng window.
    I = V.find(I, function(x) return x < J0 or x > J1 end)

    -- Now, set the outside points to zero.
    local temp = ct[i] ; temp[I] = 0 ; ct[i] = temp
end

-------------------------------------------------
-- Convert everything back to the frequency domain.
-------------------------------------------------
cf = CM.fft( ct, AP.FFT_SIZE, false, false )
--cf = CM.reverse(cf)

-- Need to strip out the zeroes in the first FFT,
-- We want to keep everything in the interval [z0..z1].
z0 = math.floor(SP.START_FREQ / dF);
z1 = z0 + SP.SCAN_SIZE - 1;

-- Cf should be reduced back to the original SCAN_SIZE.
cf = CM.slice2(cf, z0, z1)
```

```
-- By spawning from the source, we get a new window with all
-- the original window properties, unless SOURCE_NAME-P3 already exists.
-------------------------------------------------------------
w = chart.spawn(SOURCE_NAME, "01")
window_manager.register(SOURCE_NAME, PROGRAM_NAME, w)

sel = chart.get_selection_vector(w)

chart.clear(w)
for i = 1, #cf do
    chart.add(w, cf[i], T1[i], T2[i], DT[i])
end
--chart.reset_scale(w)
chart.set_analysis_type(w, MAGNITUDE_VS_FREQUENCY)
AP.WINDOW_FFT = false ;
chart.set_analysis_parameter_table(w, AP)
chart.set_option(w, DB_FREQ, false);
if #sel > 0 and #sel == #cf then
    chart.set_selection_vector(w, sel)
end
chart.update(w)
end
```

Tool-Window.lua (Python)

```
L = require("UTIL-Lua")
P = require("UTIL-Plot")

# ------------------------------------------------------------

CV = complex_vector
 V = vector

CM = complex_matrix

# SOURCE_NAME is bound to the analysis window
# that called this script. We can access the window
# and its contents using it.

if SOURCE_NAME == "":
    window_manager.unregister(SOURCE_NAME, PROGRAM_NAME)
    forms.error("Not a stand-alone script. Must be run from an Analysis Window")

def get_analysis_type(source_name):
    return chart.get_analysis_type(source_name)

if analysis_type == MAGNITUDE_VS_TIME:
    window_manager.register(SOURCE_NAME, PROGRAM_NAME, "Time")
elif analysis_type == MAGNITUDE_VS_DISTANCE:
    window_manager.register(SOURCE_NAME, PROGRAM_NAME, "Distance")
else:
    forms.error("Not a valid analysis type. Select either Time or Distance from View Menu.")
<|python|># -*- coding: utf-8 -*-
# Generated by Django 1.11.5 on

def get_scan_parameter_table(source_name):
    scan_parameter_table = chart.get_scan_parameter_table(source_name)
    print(scan_parameter_table)

# L.display("SP", SP)
# L.display("AP", AP)

def Process():
    print('Hello, World!')

# This script is only valid in the time and distance domains.
# ------------------------------------------------------------
def get_scale(source_name):
    x0, x1 = chart.get_scale(source_name)
    return x0, x1

# Convert meters to nano-seconds.

def analysis_type(x0, x1):
    if x0 / x1 > 1:
        return 'MAGNITUDE_VS_DISTANCE'
    else:
        return 'DISTANCE_VS_MAGNITUDE'
# print(x0, x1)

# Get the raw complex freq domain data.
# ------------------------------------------------------------------------
def get_matrix(source_name):
    return chart.get_matrix(source_name)
# The selection vector contains ones and zeroes denoting
# lines that are visible in the source window.
def get_selection_vector(source_name):
    sv = chart.get_selection_vector(source_name)
    print(sv)

# Reduce the matrix to only selected rows.
# ------------------------------------------------------------------------
```

```
# ------------------------------------------------------------------------
def select_cm(cm, sv):
    cs = CM.select(cm, sv)
    print(cs)

# We will need to know where to get titles for the selected vectors.
# Convert the selection vector into an index vector by selecting from
# a vector of indexes.
def select(v, sv):
    return v.select(sv)

# Might as well get the line attributes now.
T1 = {}
T2 = {}
DT = {}

for i in range(1, #iv
        T1[i],
        T2[i],
        DT[i]):
        print(i, T1[i], T2[i], DT[i])

# ----------------------------------------
# Convert everything to the time domain.
# ----------------------------------------
def fft(cs, size, start_freq, stop_freq, window_fft, inverse):
    ct = cm.fft(cs, size, start_freq, stop_freq, window_fft, inverse)
    print(ct)
# print(cs, AP.FFT_SIZE, SP.START_FREQ, SP.STOP_FREQ, AP.WINDOW_FFT, true  )

# We need these for scaling purposes.
# ------------------------------------------------------------------------

def get_freq_step(scan_size, start_freq, stop_freq):
    return data_source.get_freq_step(scan_size, start_freq, stop_freq)

# print(dF, dT)

# Calculate the indexes of the bounding window.
# ------------------------------------------------------------------------

def I0(x0, dT):
    return x0 / dT
def I1(x1, dT):
    return x1 / dT

# I0 = math.max(0, I0)
# I1 = math.min(SP.SCAN_SIZE, I1)

# Need to know 1/2 the width of the window.

def dI(I1, I0):
    return (I1 - I0) / 2

# print("I0, I1, dI (unadjusted) = ", I0, I1, dI)

# Now, for each vector in the time domain, set the outside points to zero.
# ------------------------------------------------------------------------
for i in range(1, #ct):
    vmax, imax = V.max(V.slice(CV.abs(ct[i]), I0, I1))
    imax = imax + I0
    J0 = imax - dI
    J1 = imax + dI

    J0 = max(0, J0)
    J1 = min(AP.FFT_SIZE, J1)
    I = V(AP.FFT_SIZE, 1, 1)
    I = V.find(I, lambda x
```

```
#------------------------------------------------------
# Convert everything back to the frequency domain.
# -----------------------------------------------------
def fft(ct, size):
    cf = CM.fft( ct, size, False, False )
    print(cf)
# cf = CM.reverse(cf)

# Need to strip out the zeroes in the first FFT,
# We want to keep everything in the interval [z0..z1].
def z0_z1(start_freq, scan_size):
    z0 = math.floor(start_freq / dF)
    z1 = z0 + scan_size - 1
    return z0, z1

# Cf should be reduced back to the original SCAN_SIZE.
def slice2(cf, z0, z1):
    return cf[z0:z1]


# By spawning from the source, we get a new window with all
# the original window properties, unless SOURCE_NAME-P3 already exists.
# ------------------------------------------------------------------
w = window_manager.spawn(SOURCE_NAME, "01")
window_manager.register(SOURCE_NAME, PROGRAM_NAME, w)
<|cpp|>#include "stdafx.h"
#include "Emu/Memory/Memory.h"
#include "Emu/System.h"
#include "Emu/SysCalls/Modules.h"

#include "cellSysutil.h"

extern Module cellSysutil;

s32 cellSysutilGetBgmPlaybackStatus()
{
        throw EX

def get_selection_vector(w):
    sel = w.get_selection_vector()
    return sel

def clear_chart(w):
    w.clear()
for i in range(1, #cf do
        chart.add(w, cf[i], T1[i], T2[i], DT[i])
end
# chart.reset_scale(w)
def set_analysis_type(w, analysis_type):
    chart.set_analysis_type(w, analysis_type)
AP.WINDOW_FFT = False
def set_analysis_parameter_table(w, AP):
    chart.set_analysis_parameter_table(w, AP)
def set_option(w, db_freq, false):
    chart.set_option(w, db_freq, false)
if #sel > 0 and #sel == #cf:
    chart.set_selection_vector(w, sel)
def update_chart(w):
    chart.update(w)
def end():
    print('Hello, World!')
```

UTIL-Header.lua (C++ → Python)

```lua
--
-- Util-Header
--

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
CM = complex_matrix

CV = complex_vector
 C = complex

 V = vector
 M = matrix

DS = data_source
DT = data_tape
```

→

```python
#
# Util-Header
#

# -------------------------------------------------------------------------------
# -------------------------------------------------------------------------------
def complex_matrix_to_complex_vector(input_matrix):
    complex_vector = [complex(x.real, x.imag) for x in input_matrix]
    return complex_vector

def complex_vector_to_complex_matrix(input_vector):
    complex_matrix = [complex(x.real, x.imag) for x in input_vector]
    return complex_matrix

def complex_matrix_to_complex(input_matrix):
    complex_vector = [complex(x.real,
```

Tool-Data-Summary.lua (Python)

```
#
# Tool-Data-Summary.lua
#
# Generates a series of analysis windows -- one for each frame in the tape.

def plot_data_source(data_source):
    plot = P.plot(data_source)
    plot.show()



def make_window():
    w = window_manager.make(FL_EDITOR_WINDOW, False)



def deduplicate_list(input_list):
    for i in range(len(input_list)):
        print("Line: " + str(i) + "  " + input_list[i])
def get_title2(source_name):
    return chart.get_title2(source_name)

def get_selection_vector(source_name):
    return chart.get_selection_vector(source_name)

def get_vector(source_name, i):
    return chart.get_vector(source_name, i)




def set_changed(w, false):
    window_manager.set_changed(w, False)
```