```
classdef ExperimentClass < handle
    %EXPERIMENTCLASS: Performs processess on data sets for an
 experiment
    %  organizes decorrelation code for a simple to use and expandable
    %  interface
    %  designed to be easily accessed for a GUI application
    properties
        % main arrays
        ultrasoundDataSeries;
        cumulativeDecorr;
        decorrelationMapSeries;
        decorrSumSeries;
        decorrSumSeriesROI;
        decorrVolume;
        averageDecorr;
        % ultrasound data parameters
        rmin;
        rmax;
        cartScalingFactor;
        sigma;
        interFrameTime;
        thetamin;
        thetamax;
        phimin;
        phimax;
        decorrThresh;
        % experiment parameters
        numDataSets;
        activeFolder;
        activeFolderDir;
        folderIndex;
        defaultDataFileName;
        totalThresh;
        totalThreshVolume;
        inSerialString;
        outSerialString;
        outSerialObj;
        inSerialObj;
        ROI_xRange;
        ROI_yRange;
        ROI_zRange;
    end

    methods
        function
 ExperimentClassSetParams(obj,thisrmin,thisrmax,thisthetamin,thisthetamax,thisphim
            %EXPERIMENTCLASS Construct an instance of this class
            %   Detailed explanation goes here
            obj.rmin = thisrmin;
            obj.rmax = thisrmax;
            obj.cartScalingFactor = thiscartScalingFactor;
            obj.sigma = thissigma;
```

```
           obj.interFrameTime = thisinterFrameTime;
           obj.thetamin = thisthetamin;
           obj.thetamax = thisthetamax;
           obj.phimin = thisphimin;
           obj.phimax = thisphimax;
           obj.decorrThresh = thisdecorrthresh;
           obj.defaultDataFileName
= 'bufApl0Out_0x0_0x0.data.dm.pmcr';

           obj.totalThreshVolume = thistotalThreshVolume;
       end
       function obj = ExperimentClass()

       end
       function obj = addNextDataSetViaFilename(obj, thisFileName)
           % Compute decorr of data set

           Dm = read_lbdump(thisFileName);
           tempDataSet = USDataClass(Dm.data,Dm.startTime,
Dm.Info,obj.rmin,obj.rmax,obj.thetamin,obj.thetamax,obj.phimin,obj.phimax,obj.car
           tempDataSet.scanConv_Frust();
           tempDataSet.compute3DDecorr();
           tempDataSet.decorrThresh = obj.decorrThresh;
           % compute ablated volume

           if(isempty(obj.cumulativeDecorr))
               % set initial cumulative decorrelation to the result
of the
               % first volume's decorr
               obj.cumulativeDecorr(1).decorr = tempDataSet.decorr;
               % compute decorrelation sum
               sizeOfVol = size(tempDataSet.decorr);
               obj.decorrSumSeries =
sum(obj.cumulativeDecorr(1).decorr(:))/prod(sizeOfVol(1:3));
               % create mask of pixels which exceed the threshold
               obj.decorrelationMapSeries(1).decorrMap =
tempDataSet.decorr;
               % remove elements below the threshold
               tempAblatedPoints =
find(obj.decorrelationMapSeries(1).decorrMap >= obj.decorrThresh);

obj.decorrelationMapSeries(1).decorrMap(tempAblatedPoints) = 1;
               %set elements above threshold to 1

obj.decorrelationMapSeries(1).decorrMap(find(obj.decorrelationMapSeries(1).decorr
~= 1)) = 0;
               % each point above the threshold has a volume of dx^3,
so
               % the number of points * dx^3 gives the volume of
ablated
               % tissue
               obj.averageDecorr(1) =
log10(mean(obj.cumulativeDecorr(1).decorr(:)));
```

```
                obj.decorrVolume(1)
 = .001*(length(tempAblatedPoints))*tempDataSet.dx^3;
            else
                % find max value between current decorrelation and
 previous
                % cumulative decorrelation
                obj.cumulativeDecorr(obj.folderIndex).decorr =
 max(obj.cumulativeDecorr(obj.folderIndex-1).decorr,tempDataSet.decorr);
                % compute sum
                sizeOfVol = size(tempDataSet.decorr);
                obj.decorrSumSeries(obj.folderIndex) =
 sum(obj.cumulativeDecorr(obj.folderIndex).decorr(:))/
prod(sizeOfVol(1:3));
                % create decorrelation map for current volume
                obj.decorrelationMapSeries(obj.folderIndex).decorrMap
 = obj.cumulativeDecorr(obj.folderIndex).decorr;
                tempAblatedPoints =
 find(obj.decorrelationMapSeries(obj.folderIndex).decorrMap >=
 obj.decorrThresh);

 obj.decorrelationMapSeries(obj.folderIndex).decorrMap(tempAblatedPoints)
 = 1;

 obj.decorrelationMapSeries(obj.folderIndex).decorrMap(find(obj.decorrelationMapSe
 ~= 1)) = 0;
                % each point above the threshold has a volume of dx^3,
 so
                % the number of points * dx^3 gives the volume of
 ablated
                % tissue
                interVal =
 obj.cumulativeDecorr(obj.folderIndex).decorr(obj.ROI_zRange(1):obj.ROI_zRange(2),
                obj.decorrSumSeriesROI(obj.folderIndex) =
 mean(interVal(:));
                obj.averageDecorr(obj.folderIndex) =
 log10(mean(obj.cumulativeDecorr(obj.folderIndex).decorr(:)));

                obj.decorrVolume(obj.folderIndex)
 = .001*(length(tempAblatedPoints))*tempDataSet.dx^3; % in cm^3
            end
            % append ultrasound data to internal data struct
            obj.ultrasoundDataSeries = [obj.ultrasoundDataSeries
 tempDataSet];

        end
        function obj = initDataFolderGUI(obj)
            try
                if ispc
                    basePath = strcat('C:\Users
\',getenv('username'),'\Box\SiemensSC2000IQData');
                elseif ismac
                    basePath = strcat('/Users/',getenv('USER'),'/
box');
                end
            catch
                basePath = matlabroot;
            end
            obj.activeFolder =  uigetdir(basePath);
            fullDirectory  = dir(obj.activeFolder);
            obj.activeFolderDir = fullDirectory(3:end);
            obj.folderIndex = 1;
        end

        function newFilePresent = checkFolder(obj)
            obj.activeFolderDir  = dir(obj.activeFolder);
            if ismac
                obj.activeFolderDir = obj.activeFolderDir(3:end);
            else
                obj.activeFolderDir = obj.activeFolderDir(3:end);
            end
            if(length(obj.activeFolderDir) >= obj.folderIndex)
                newFilePresent = 1;
            else
                newFilePresent = 0;
            end
        end

        function nextDataSetInFolder(obj)
            if ispc
                fullPath =strcat(obj.activeFolder,'\',
{obj.activeFolderDir(obj.folderIndex).name},'\',obj.defaultDataFileName);
            elseif ismac
                fullPath =strcat(obj.activeFolder,'/',
{obj.activeFolderDir(obj.folderIndex).name},'/',obj.defaultDataFileName);
            end
            while(~exist(fullPath{1},'file'))
               pause(.01);
               display('waiting for file');
            end
            obj.addNextDataSetViaFilename(fullPath{1});
            obj.folderIndex = obj.folderIndex+1;
        end
        function dataSlice =
 getDataSlice_cart(obj,direction,set,frame,index)
            switch direction
                case 'z'
                    dataSlice =
 squeeze(obj.ultrasoundDataSeries(set).rawData_cart(:,:,index,frame));
                case 'y'
                    dataSlice =
 squeeze(obj.ultrasoundDataSeries(set).rawData_cart(:,index,:,frame));
                case 'x'
                    dataSlice =
 squeeze(obj.ultrasoundDataSeries(set).rawData_cart(index,:,:,frame));
                otherwise
                    dataSlice = 1;
            end
        end
        function dataSlice =
getDataSlice_decorr(obj,direction,set,frame,index)
            switch direction
                case 'z'
                    dataSlice =
squeeze(obj.ultrasoundDataSeries(set).decorr(:,:,index,frame));
                case 'y'
                    dataSlice =
squeeze(obj.ultrasoundDataSeries(set).decorr(:,index,:,frame));
                case 'x'
                    dataSlice =
squeeze(obj.ultrasoundDataSeries(set).decorr(index,:,:,frame));
                otherwise
            end
        end
        function dataSlice =
getDataSlice_cumulativeDecorr(obj,direction,set,frame,index)
            switch direction
                case 'z'
                    dataSlice =
squeeze(obj.cumulativeDecorr(set).decorr(:,:,index,frame));
                case 'y'
                    dataSlice =
squeeze(obj.cumulativeDecorr(set).decorr(:,index,:,frame));
                case 'x'
                    dataSlice =
squeeze(obj.cumulativeDecorr(set).decorr(index,:,:,frame));
                otherwise
            end
        end

        function dataSlice =
getDataSlice_decorrMask(obj,direction,set,frame,index)
            switch direction
                case 'z'
                    dataSlice =
squeeze(obj.decorrelationMapSeries(set).decorrMap(:,:,index,frame));
                case 'y'
                    dataSlice =
squeeze(obj.decorrelationMapSeries(set).decorrMap(:,index,:,frame));
                case 'x'
                    dataSlice =
squeeze(obj.decorrelationMapSeries(set).decorrMap(index,:,:,frame));
                otherwise
            end
        end

        function dataSlice =
getDataSlice_ROI(obj,direction,set,frame,index)
            subVolume =
obj.ultrasoundDataSeries(set).rawData_cart(:,:,:,frame);
            maskVol = zeros(size(subVolume));
```

```
 maskVol(obj.ROI_zRange(1):obj.ROI_zRange(2),obj.ROI_yRange(1):obj.ROI_yRange(2),o
 = 1;
            %subVolume([1:obj.ROI_zRange(1),obj.ROI_zRange(2):end],
[1:obj.ROI_yRange(1),obj.ROI_yRange(2):end],
[1:obj.ROI_zRange(1),obj.ROI_zRange(2):end]) = 0;
            subVolume = subVolume .* maskVol;
            switch direction
                case 'z'
                    dataSlice = squeeze(subVolume(:,:,index,frame));
                case 'y'
                    dataSlice = squeeze(subVolume(:,index,:,frame));
                case 'x'
                    dataSlice = squeeze(subVolume(index,:,:,frame));
                otherwise
            end
        end

        function computeDecorrStats(obj, tempDataSet, dataIndex)
            if(isempty(obj.cumulativeDecorr))
                obj.cumulativeDecorr = tempDataSet.decorr;
                obj.decorrelationMapSeries;
                obj.decorrSumSeries;
                obj.decorrVolume;
            else
                obj.cumulativeDecorr =
 max(obj.cumulativeDecorr,tempDataSet.decorr);
            end
        end

        function boolOut = decorrExceedsThresh(obj)
            if((obj.totalThresh) <=
 log10(obj.decorrSumSeriesROI(obj.folderIndex-1)))
                boolOut =  1;
            else
                boolOut =  0;
            end
        end
        function recomputeDecorr(obj)
            for currentVol = 1:length(obj.ultrasoundDataSeries)

 obj.ultrasoundDataSeries(currentVol).compute3DDecorr();
                obj.ultrasoundDataSeries(currentVol).decorrThresh =
 obj.decorrThresh;
                if(currentVol == 1)
                    obj.cumulativeDecorr(currentVol).decorr =
 obj.ultrasoundDataSeries(currentVol).decorr;
                    % compute sum
                    sizeOfVol =
 size(obj.ultrasoundDataSeries(currentVol).decorr);
                    obj.decorrSumSeries(currentVol) =
 sum(obj.cumulativeDecorr(currentVol).decorr(:))/prod(sizeOfVol(1:3));
                    % create decorrelation map for current volume
```

```
                   obj.decorrelationMapSeries(currentVol).decorrMap =
obj.cumulativeDecorr(currentVol).decorr;
                   tempAblatedPoints =
find(obj.decorrelationMapSeries(currentVol).decorrMap >=
obj.decorrThresh);

obj.decorrelationMapSeries(currentVol).decorrMap(tempAblatedPoints) =
1;

obj.decorrelationMapSeries(currentVol).decorrMap(find(obj.decorrelationMapSeries(
~= 1)) = 0;
                   % each point above the threshold has a volume of
dx^3, so
                   % the number of points * dx^3 gives the volume of
ablated
                   % tissue
                   obj.decorrVolume(currentVol)
= .001*(length(tempAblatedPoints))*obj.ultrasoundDataSeries(currentVol).dx^3; %
in cm^3
               else
                   obj.cumulativeDecorr(currentVol).decorr
= max(obj.cumulativeDecorr(currentVol-1).decorr ,
obj.ultrasoundDataSeries(currentVol).decorr);
                   % compute sum
                   sizeOfVol =
size(obj.ultrasoundDataSeries(currentVol).decorr);
                   obj.decorrSumSeries(currentVol) =
sum(obj.cumulativeDecorr(currentVol).decorr(:))/prod(sizeOfVol(1:3));
                   % create decorrelation map for current volume
                   obj.decorrelationMapSeries(currentVol).decorrMap =
obj.cumulativeDecorr(currentVol).decorr;
                   tempAblatedPoints =
find(obj.decorrelationMapSeries(currentVol).decorrMap >=
obj.decorrThresh);

obj.decorrelationMapSeries(currentVol).decorrMap(tempAblatedPoints) =
1;

obj.decorrelationMapSeries(currentVol).decorrMap(find(obj.decorrelationMapSeries(
~= 1)) = 0;
                   % each point above the threshold has a volume of
dx^3, so
                   % the number of points * dx^3 gives the volume of
ablated
                   % tissue
                   obj.decorrVolume(currentVol)
= .001*(length(tempAblatedPoints))*obj.ultrasoundDataSeries(currentVol).dx^3; %
in cm^3
               end
           end
       end
       function sendSerialData(obj)
           %pause(3)
```

```
            fprintf(obj.outSerialObj,'S');

        end
        function setUpSerialOutConnection(obj)
            obj.outSerialObj = serial(obj.outSerialString,'BaudRate',
 115200);
            fopen(obj.outSerialObj);
        end
        function removeSerialConnection()
            fclose(obj.outSerialObj);
        end
        function updateROIDataSet(obj)
            for currN = 1:length(obj.ultrasoundDataSeries)
                interVal =
 obj.cumulativeDecorr(currN).decorr(obj.ROI_zRange(1):obj.ROI_zRange(2),obj.ROI_yR
                obj.decorrSumSeriesROI(currN) = mean(interVal(:));
            end
        end
        function initExperiment(obj)
            interVal =
 obj.cumulativeDecorr(1).decorr(obj.ROI_zRange(1):obj.ROI_zRange(2),obj.ROI_yRange
            obj.decorrSumSeriesROI(1) = mean(interVal(:));
        end
    end
end
```

Published with MATLAB® R2016a