plotSimulationDataset

Search for available trainings or test dataset and plot dataset. Follow user input dialog to choose which dataset and decide how many angles to plot. Save dataset content redered to an avi-file. Filename same as dataset.

Contents

- Syntax
- Description
- Examples
- Input Argurments
- Output Argurments
- Requirements
- See Also

Syntax

```
plotSimulationDataset()
```

Description

plotSimulationDataset() plot training or test dataset which are loacated in data/test or data/training. The function list all datasets and the user must decide during user input dialog which dataset to plot and how many angles to to visualize. It loads path from config.mat and scans for file automatically.

Examples

```
plotSimulationDataset()
```

Input Argurments

None

Output Argurments

None

Requirements

- Other m-files required: None
- Subfunctions: None
- MAT-files required: config.mat

See Also

- generateSimulationDatasets
- sensorArraySimulation
- generateConfigMat


```
function plotSimulationDataset()
    % scan for datasets and load needed configurations %%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    try
        disp('Plot simulation dataset ...');
        close all;
        % load path variables
        load('config.mat', 'PathVariables');
        % scan for datasets
        TrainingDatasets = dir(fullfile(PathVariables.trainingDataPath, ...
            'Training_*.mat'));
```

```
    TestDatasets = dir(fullfile(PathVariables.testDataPath, 'Test_*.mat'));
    allDatasets = [TrainingDatasets; TestDatasets];
    % check if files available
    if isempty(allDatasets)
        error('No training or test datasets found.');
    end
catch ME
    rethrow(ME)
end

% display availabe datasets to user, decide which to plot %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% number of datasets
nDatasets = length(allDatasets);
fprintf('Found %d datasets:\n', nDatasets)
for i = 1:nDatasets
    fprintf('%s\t:\t(%d)\n', allDatasets(i).name, i)
end
% get numeric user input to indicate which dataset to plot
iDataset = input('Type number to choose dataset to plot to: ');
% iDataset = 2;

% load dataset and ask user which one and how many angles %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
try
    ds = load(fullfile(allDatasets(iDataset).folder, ...
        allDatasets(iDataset).name));
    % check how many angles in dataset and let user decide how many to
    % render in polt
    fprintf('Detect %d angles in dataset ...\n', ds.Info.UseOptions.nAngles);
    nSubAngles = input('How many angles to you wish to plot: ');
    % nSubAngles = 120;
    % indices for data to plot, get sample distance for even distance
    sampleDistance = length(downsample(ds.Data.angles, nSubAngles));
    % get subset of angles
    subAngles = downsample(ds.Data.angles, sampleDistance);
    nSubAngles = length(subAngles); % just ensure
    % get indices for subset data
    indices = find(ismember(ds.Data.angles, subAngles));
catch ME
    rethrow(ME)
end

% create dataset figure for a subset or all angle %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fig = figure('Name', 'Sensor Array', ...
    'NumberTitle' , 'off', ...
    'WindowStyle', 'normal', ...
    'MenuBar', 'none', ...
    'ToolBar', 'none', ...
    'Units', 'centimeters', ...
    'OuterPosition', [0 0 30 30], ...
    'PaperType', 'a4', ...
    'PaperUnits', 'centimeters', ...
    'PaperOrientation', 'landscape', ...
    'PaperPositionMode', 'auto', ...
    'DoubleBuffer', 'on', ...
    'RendererMode', 'manual', ...
    'Renderer', 'painters');

tdl = tiledlayout(fig, 2, 2, ...
    'Padding', 'normal', ...
    'TileSpacing' , 'compact');


title(tdl, 'Sensor Array Simulation', ...
    'FontWeight', 'normal', ...
    'FontSize', 18, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');
```

```
    subline1 = "Sensor Array (%s) of $%d\\times%d$ sensors, an edge length of $%.1f$ mm,
a rel. pos. to magnet surface of";
    subline2 = " $(%.1f, %.1f, -(%.1f))$ in mm, a magnet tilt of $%.1f^\\circ$, a sphere
radius of $%.1f$ mm, a imprinted";
    subline3 = "field strength of $%.1f$ kA/m at $%.1f$ mm from sphere surface in z-axis,
 $%d$ rotation angles with a ";
    subline4 = "step width of $%.1f^\\circ$ and a resolution of $%.1f^\\circ$. Visualized
 is a subset of $%d$ angles in ";
    subline5 = "sample distance of $%d$ angles. Based on %s characterization reference %s
.";
    sub = [sprintf(subline1, ...
                   ds.Info.SensorArrayOptions.geometry, ...
                   ds.Info.SensorArrayOptions.dimension, ...
                   ds.Info.SensorArrayOptions.dimension, ...
                   ds.Info.SensorArrayOptions.edge); ...
           sprintf(subline2, ...
                   ds.Info.UseOptions.xPos, ...
                   ds.Info.UseOptions.yPos, ...
                   ds.Info.UseOptions.zPos, ...
                   ds.Info.UseOptions.tilt, ...
                   ds.Info.DipoleOptions.sphereRadius); ...
           sprintf(subline3, ...
                   ds.Info.DipoleOptions.H0mag, ...
                   ds.Info.DipoleOptions.z0, ...
                   ds.Info.UseOptions.nAngles); ...
           sprintf(subline4, ...
                   ds.Data.angleStep, ...
                   ds.Info.UseOptions.angleRes, ...
                   nSubAngles)
           sprintf(subline5, ...
                   sampleDistance, ...
                   ds.Info.CharData, ...
                   ds.Info.UseOptions.BridgeReference)];

    subtitle(tdl, sub, ...
        'FontWeight', 'normal', ...
        'FontSize', 14, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % get subset of needed data to plot, only one load %%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    N = ds.Info.SensorArrayOptions.dimension;
    X = ds.Data.X;
    Y = ds.Data.Y;
    Z = ds.Data.Z;

    % calc limits of plot 1
    maxX = ds.Info.UseOptions.xPos + ds.Info.SensorArrayOptions.edge;
    maxY = ds.Info.UseOptions.yPos + ds.Info.SensorArrayOptions.edge;
    minX = ds.Info.UseOptions.xPos - ds.Info.SensorArrayOptions.edge;
    minY = ds.Info.UseOptions.yPos - ds.Info.SensorArrayOptions.edge;

    % calculate colormap to identify scatter points
    c=zeros(N,N,3);
    for i = 1:N
        for j = 1:N
            c(i,j,:) = [(2*N+1-2*i), (2*N+1-2*j), (i+j)]/2/N;
        end
    end
    c = squeeze(reshape(c, N^2, 1, 3));

    % load offset voltage to subtract from cosinus, sinus voltage
    Voff = ds.Info.SensorArrayOptions.Voff;

    % plot sensor grid in x and y coordinates and constant z layer %%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ax1 = nexttile(1);
    % plot each cooredinate in loop to create a special shading constant
    % reliable to orientation for all matrice
```

```
    hold on;
    scatter(X(:), Y(:), [], c, 'filled', 'MarkerEdgeColor', 'k', ...
        'LineWidth', 0.8);

    % axis shape and ticks
    axis square xy;
    axis tight;
    grid on;
    xlim([minX maxX]);
    ylim([minY maxY]);

    % text and labels
    text(minX+0.2, minY+0.2, ...
        sprintf('$Z = %.1f$ mm', Z(1)), ...
        'Color', 'k', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    xlabel('$X$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$Y$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title(sprintf('Sensor Array $%d\\times%d$', N, N), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    hold off;

    % plot rotation angles in polar view %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(2);
    % plot all angles grayed out
    polarscatter(ds.Data.angles/180*pi, ones(1, ds.Info.UseOptions.nAngles), ...
        [], [0.8 0.8 0.8], 'filled');

    % radius ticks and label
    rticks(1);
    rticklabels("");
    hold on;

    % plot subset of angles
    % polarscatter(subAngles/180*pi, ones(1, nSubAngles), 'k', 'LineWidth', 0.8);
    ax2 = gca;

    % axis shape
    axis tight;

    % text an labels
    % init first rotation step label
    tA = text(2/3*pi, 1.5, ...
        '$\\theta$', ...
        'Color', 'b', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Rotation around Z-Axis in Degree', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
```

```
    'Interpreter', 'latex');

hold off;

% Cosinus bridge outputs for rotation step %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ax3 = nexttile(3);
hold on;

% set colormap
colormap('gray');

% plot cosinus reference, set NaN values to white color, orient Y to normal
imC = imagesc(ds.Data.HxScale, ds.Data.HyScale, ds.Data.VcosRef);
set(imC, 'AlphaData', ~isnan(ds.Data.VcosRef));
set(gca, 'YDir', 'normal')

% axis shape and ticks
axis square xy;
axis tight;
yticks(xticks);
grid on;

% test and labels
xlabel('$H_x$ in kA/m', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

ylabel('$H_y$ in kA/m', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

title('$V_{cos}(H_x, H_y)$', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

% add colorbar and place it
cb1 = colorbar;
cb1.Label.String = sprintf(...
    '$V_{cos}(H_x, H_y)$ in V, $V_{cc} = %1.1f$ V, $V_{off} = %1.2f$ V', ...
    ds.Info.SensorArrayOptions.Vcc, ds.Info.SensorArrayOptions.Voff);
cb1.Label.Interpreter = 'latex';
cb1.Label.FontSize = 12;

hold off;

% Sinus bridge outputs for rotation step %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ax4 = nexttile(4);
hold on;

% set colormap
colormap('gray');

% plot sinus reference, set NaN values to white color, orient Y to normal
imS = imagesc(ds.Data.HxScale, ds.Data.HyScale, ds.Data.VsinRef);
set(imS, 'AlphaData', ~isnan(ds.Data.VsinRef));
set(gca, 'YDir', 'normal')

% axis shape and ticks
axis square xy;
axis tight;
yticks(xticks);
grid on;
```

```
    % test and labels
    xlabel('$H_x$ in kA/m', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$ in kA/m', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{sin}(H_x, H_y)$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % add colorbar and place it
    cb2 = colorbar;
    cb2.Label.String = sprintf(...
        '$V_{sin}(H_x, H_y)$ in V, $V_{cc} = %1.1f$ V, $V_{off} = %1.2f$ V', ...
        ds.Info.SensorArrayOptions.Vcc, ds.Info.SensorArrayOptions.Voff);
    cb2.Label.Interpreter = 'latex';
    cb2.Label.FontSize = 12;

    hold off;

    % zoom axes for scatter on cosinuns reference images %%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(3);
    ax5 = axes('Position', [0.07 0.02 0.19 0.19], 'XColor', 'r', 'YColor', 'r');
    hold on;
    axis square xy;
    grid on;
    hold off;

    % draw everything prepared before start renewing frame wise and prepare for
    % recording frames to video file %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % draw frame
    drawnow;

    % get file path and change extension
    [~, fName, ~] = fileparts(ds.Info.filePath);
    fPath = PathVariables.saveImagesPath;

    % string allows simple cat ops
    VW = VideoWriter(fullfile(fPath, 'avi', fName + ".avi"), "Uncompressed AVI");

    % scale frame rate on 10 second movies, ensure at least 1 fps
    fr = floor(nSubAngles / 10) + 1;
    VW.FrameRate = fr;

    % open video file, ready to record frames
    open(VW)


    % loop through subset angle dataset and renew plots %%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i = indices
        % H load subset
        Hx = ds.Data.Hx(:,:,i);
        Hy = ds.Data.Hy(:,:,i);
        % get min max
        maxHx = max(Hx, [], 'all');
        maxHy = max(Hy, [], 'all');
        minHx = min(Hx, [], 'all');
        minHy = min(Hy, [], 'all');
        dHx = abs(maxHx - minHx);
```

```
        dHy = abs(maxHy - minHy);

        % load V subset
        Vcos = ds.Data.Vcos(:,:,i) - Voff;
        Vsin = ds.Data.Vsin(:,:,i) - Voff;
        angle = ds.Data.angles(i);

        % lock plots
        hold(ax1, 'on');
        hold(ax2, 'on');
        hold(ax3, 'on');
        hold(ax4, 'on');
        hold(ax5, 'on');

        % update plot 1
        qH = quiver(ax1, X, Y, Hx, Hy, 0.5, 'b');
        qV = quiver(ax1, X, Y, Vcos, Vsin, 0.5, 'r');
        legend([qH qV], {'$quiver(H_x,H_y)$', '$quiver(V_{cos}-V_{off},V_{sin}-V_{off})$'
},...
            'FontWeight', 'normal', ...
            'FontSize', 9, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex', ...
            'Location', 'NorthEast');

        % update plot 2
        tA.String = sprintf('$%.1f^\\circ$', angle);
        pA = polarscatter(ax2, angle/180*pi, 1, 'b', 'filled', ...
            'MarkerEdgeColor', 'k', 'LineWidth', 0.8);

        % update plot 3 and 4
        sC = scatter(ax3, Hx(:), Hy(:), 5, c, 'filled', 'MarkerEdgeColor', 'k', ...
            'LineWidth', 0.8);
        sS = scatter(ax4, Hx(:), Hy(:), 5, c, 'filled', 'MarkerEdgeColor', 'k', ...
            'LineWidth', 0.8);

        % calc position of scatter area frame and reframe
        pos = [minHx - 0.3 * dHx, minHy - 0.3 * dHy, 1.6 * dHx, 1.6 * dHy];
        rtC = rectangle(ax3, 'Position', pos, 'LineWidth', 1,  'EdgeColor', 'r');
        rtS = rectangle(ax4, 'Position', pos, 'LineWidth', 1,  'EdgeColor', 'r');

        % update plot 5 (zoom)
        sZ = scatter(ax5, Hx(:), Hy(:), [], c, 'filled', 'MarkerEdgeColor', 'k', ...
            'LineWidth', 0.8);
        xlim(ax5, [pos(1) maxHx + 0.3 * dHx])
        ylim(ax5, [pos(2) maxHy + 0.3 * dHy])

        % release plots
        hold(ax1, 'off');
        hold(ax2, 'off');
        hold(ax3, 'off');
        hold(ax4, 'off');
        hold(ax5, 'off');

        % draw frame
        drawnow;

        % record frame to file
        frame = getframe(fig);
        writeVideo(VW, frame);

        % delete part of plots to renew for current angle, delete but last
        if i ~= indices(end)
            delete(qH);
            delete(qV);
            delete(pA);
            delete(rtC);
            delete(rtS);
            delete(sC);
            delete(sS);
            delete(sZ);
```

```
        end
    end
    % close video file
    close(VW)
    close(fig)
end
```

Published with MATLAB® R2020b