plotSingleSimulationAngle

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. Plot single Angle and save figure to file. File name same as dataset with attach angle index.

Syntax

```
plotSingleSimulationAngle()
```

Description

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

Examples

```
plotSingleSimulationAngle()
```

Input Argurments

None

Output Argurments

None

Requirements

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

See Also

- generateSimulationDatasets
- sensorArraySimulation
- generateConfigMat

Created on November 28. 2020 by Tobias Wulf.

```
function plotSingleSimulationAngle()
    % scan for datasets and load needed configurations %%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    try
        disp('Plot single simulation angle ...');
        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
```

```matlab
    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 ([1:%d]) in dataset ...\n', ...
            ds.Info.UseOptions.nAngles, ds.Info.UseOptions.nAngles);
        fprintf('Resolution\t:\t%.1f\n', ds.Info.UseOptions.angleRes);
        fprintf('Step width\t:\t%.1f\n', ds.Data.angleStep);
        fprintf('Start angle\t:\t%.1f\n', ds.Data.angles(1))
        idx = input('Which angle do you wish to plot (enter index): ');
        angle = interp1(ds.Data.angles, idx, 'nearest');
    catch ME
        rethrow(ME)
    end

    % figure save path for different formats %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fPath = PathVariables.saveImagesPath;

    % create dataset figure for a subset or all angle %%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fig = figure('Name', 'Sensor Array', ...
        'NumberTitle' , 'off', ...
        'WindowStyle', 'normal', ...
        'Position', [4381 15 1244 983], ...
        'Units', 'pixels', ...
        'WindowState', 'maximized');

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


    disp('Sensor Array Simulation');

    subline1 = "Sensor Array (%s) of %dx%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°, 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° and a resolution of" + ...
```

```
    " %.1f°. Visualized is rotatation angle %d (%.1f°)$.";
subline5 = "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, ...
               idx, angle)
       sprintf(subline5, ...
               ds.Info.CharData, ...
               ds.Info.UseOptions.BridgeReference)];

disp(sub);

% 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', 20, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

xlabel('$X$ in mm');

ylabel('$Y$ in mm');

title(sprintf('a) Sensor-Array $%d\\times%d$', N, N));

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', 20, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

title('b) Rotation Angle');

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');

ylabel('$H_y$ in kA/m');

title('c) $V_{cos}(H_x, H_y)$ in V');

% add colorbar and place it
cb1 = colorbar;
cb1.Label.String = sprintf(...
    '$V_{cc} = %1.1f$ V, $V_{off} = %1.2f$ V', ...
    ds.Info.SensorArrayOptions.Vcc, ds.Info.SensorArrayOptions.Voff);
cb1.TickLabelInterpreter = 'latex';
cb1.Label.Interpreter = 'latex';
cb1.Label.FontSize = 20;

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');

ylabel('$H_y$ in kA/m');

title('d) $V_{sin}(H_x, H_y)$ in V');

% add colorbar and place it
cb2 = colorbar;
cb2.Label.String = sprintf(...
    '$V_{cc} = %1.1f$ V, $V_{off} = %1.2f$ V', ...
    ds.Info.SensorArrayOptions.Vcc, ds.Info.SensorArrayOptions.Voff);
cb2.TickLabelInterpreter = 'latex';
```

```
cb2.Label.Interpreter = 'latex';
cb2.Label.FontSize = 20;

hold off;

% zoom axes for scatter on cosinuns reference images %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nexttile(3);
ax5 = axes('Position', [0.15 0.115 0.12 0.12], ...
    'XColor', 'r', 'YColor', 'r');
xticklabels(ax5, []);
yticklabels(ax5, []);
hold on;
axis square xy;
grid on;
hold off;

ax6 = axes('Position', [0.581 0.115 0.12 0.12], ...
    'XColor', 'r', 'YColor', 'r');
xticklabels(ax6, []);
yticklabels(ax6, []);
hold on;
axis square xy;
grid on;
hold off;

% plot angle into plots %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% H load subset
Hx = ds.Data.Hx(:,:,idx);
Hy = ds.Data.Hy(:,:,idx);
% 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(:,:,idx) - Voff;
Vsin = ds.Data.Vsin(:,:,idx) - Voff;
angle = ds.Data.angles(idx);

% lock plots
hold(ax1, 'on');
hold(ax2, 'on');
hold(ax3, 'on');
hold(ax4, 'on');
hold(ax5, 'on');
hold(ax6, 'on');

% update plot 1
qH = quiver(ax1, X, Y, Hx, Hy, 0.7, 'b');
qV = quiver(ax1, X, Y, Vcos, Vsin, 0.7, 'r');
legend([qH qV], {'$quiver(H_x,H_y)$', ...
    '$quiver(V_{cos}-V_{off},V_{sin}-V_{off})$'},...
    'FontSize', 14, ...
    'Location', 'NorthEast');

% update plot 2
tA.String = sprintf('$%.1f^\\circ$', angle);
```

```
    polarscatter(ax2, angle/180*pi, 1, 'b', 'filled', ...
        'MarkerEdgeColor', 'k', 'LineWidth', 0.8);

    % update plot 3 and 4
    scatter(ax3, Hx(:), Hy(:), 5, c, 'filled', 'MarkerEdgeColor', 'k', ...
        'LineWidth', 0.8);
    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];
    rectangle(ax3, 'Position', pos, 'LineWidth', 1.5,  'EdgeColor', 'r');
    rectangle(ax4, 'Position', pos, 'LineWidth', 1.5,  'EdgeColor', 'r');

    % update plot 5 (zoom)
    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])

     % update plot 6 (zoom)
    scatter(ax6, Hx(:), Hy(:), [], c, 'filled', 'MarkerEdgeColor', 'k', ...
        'LineWidth', 0.8);
    xlim(ax6, [pos(1) maxHx + 0.3 * dHx])
    ylim(ax6, [pos(2) maxHy + 0.3 * dHy])

    % release plots
    hold(ax1, 'off');
    hold(ax2, 'off');
    hold(ax3, 'off');
    hold(ax4, 'off');
    hold(ax5, 'off');
    hold(ax6, 'off');

    % save figure to file %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % get file path to save figure with angle index
    [~, fName, ~] = fileparts(ds.Info.filePath);

%     % save to various formats
%     yesno = input('Save? [y/n]: ', 's');
%     if strcmp(yesno, 'y')
%         fLabel = input('Enter file label: ', 's');
%         fName = fName + sprintf("_AnglePlot_%d_", idx) + fLabel;
%         savefig(fig, fullfile(fPath, fName));
%         print(fig, fullfile(fPath, fName), '-dsvg');
%         print(fig, fullfile(fPath, fName), '-depsc', '-tiff', '-loose');
%         print(fig, fullfile(fPath, fName), '-dpdf', '-loose', '-fillpage');
%     end
%     close(fig);
end
```

Published with MATLAB® R2020b