plotSimulationSubset

Search for available trainings or test dataset and plot dataset. Follow user input dialog to choose which dataset and decide which array elements to plot. Save created plot to file. Filename same as dataset with attached info.

Syntax

```
plotSimulationSubset()
```

Description

plotSimulationSubset() 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 how many angles to to visualize. It loads path from config.mat and scans for file automatically.

Examples

```
plotSimulationSubset()
```

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 29. 2020 by Tobias Wulf.

n

```
function plotSimulationSubset()
    % 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 = 3;%input('Type number to choose dataset to plot to: ');

    % 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 x %d sensors in dataset ...\n', ...
            ds.Info.SensorArrayOptions.dimension, ...
            ds.Info.SensorArrayOptions.dimension);
        xIdx = input("Enter row indices in []: ");
        yIdx = input("Enter col indices in []: ");
        if length(xIdx) ~= length(yIdx)
            error('Indices must have the same length!')
        end
%         fprintf('Detect %d angles in dataset ...\n', ...
%             ds.Info.UseOptions.nAngles);
%         nSubAngles = input('How many angles to you wish to plot: ');
%         % 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
%         angleIdx = find(ismember(ds.Data.angles, subAngles));
    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', ...
        'WindowState', 'maximized');

    tdl = tiledlayout(fig, 4, 6, ...
        'Padding', 'compact', ...
        'TileSpacing' , 'compact');
```

```matlab
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 a subset.";
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)
       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
a= ds.Info.SensorArrayOptions.edge;
maxX = ds.Info.UseOptions.xPos + a * 0.66;
maxY = ds.Info.UseOptions.yPos + a * 0.66;
minX = ds.Info.UseOptions.xPos - a * 0.66;
minY = ds.Info.UseOptions.yPos - a * 0.66;
dp = a / (ds.Info.SensorArrayOptions.dimension - 1);
x1 = ds.Info.UseOptions.xPos - a/2;
x2 = ds.Info.UseOptions.xPos + a/2;
y1 = ds.Info.UseOptions.yPos - a/2;
y2 = ds.Info.UseOptions.yPos + a/2;

% 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));
    % reshape RGB for picking single sensors
    R = reshape(c(:,1), N, N);
    G = reshape(c(:,2), N, N);
    B = reshape(c(:,3), N, N);

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

    % plot sensor grid in x and y coordinates and constant z layer %%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ax1 = nexttile(6,[2 1]);
    % plot each cooredinate in loop to create a special shading constant
    % reliable to orientation for all matrice
    hold on;

    scatter(X(:), Y(:), 48, [0.8 0.8 0.8], 'filled', ...
        'MarkerEdgeColor', 'k', 'LineWidth', 0.8);

    for k = 1:length(xIdx)
        i = xIdx(k); j = yIdx(k);
        scatter(X(i,j), Y(i,j), 96, [R(i,j), G(i,j), B(i,j)], 'filled', ...
            'MarkerEdgeColor', 'k', 'LineWidth', 0.8);
    end

    % axis shape and ticks
    axis square xy;
    axis tight;
    grid on;
    xlim([minX maxX]);
    ylim([minY maxY]);
    xticks(x1:dp:x2);
    xticklabels(1:ds.Info.SensorArrayOptions.dimension);
    yticks(y1:dp:y2);
    yticklabels(ds.Info.SensorArrayOptions.dimension:-1:1);

    xlabel('$j$');

    ylabel('$i$');

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

    hold off;


    % Cosinus bridge outputs for rotation step %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ax3 = nexttile(1, [2 2]);
    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('a) $V_{cos}(H_x, H_y)$');

    % add colorbar and place it
    cb1 = colorbar;
    cb1.Label.String = 'in V';
    cb1.TickLabelInterpreter = 'latex';
    cb1.Label.Interpreter = 'latex';
    cb1.Label.FontSize = 20;
    hold off;

    % Sinus bridge outputs for rotation step %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ax4 = nexttile(13, [2 2]);
    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) $V_{sin}(H_x, H_y)$');

    % add colorbar and place it
    cb2 = colorbar;
    cb2.Label.String = 'in V';
    cb2.TickLabelInterpreter = 'latex';
    cb2.Label.Interpreter = 'latex';
    cb2.Label.FontSize = 20;

    hold off;

    % plot Vcos Vsin over angles %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % axes limits
    xlimits = [0 360];
    ylimits = [min(cat(...
        3, ds.Data.VsinRef, ds.Data.VcosRef), [], 'all') - 0.1*Vcc, ...
        max(cat(3, ds.Data.VsinRef, ds.Data.VcosRef), [], 'all') + 0.1*Vcc];
```

```matlab
% Vcos
ax5 = nexttile(3, [2 3]);
yline(Voff, 'k-.', 'LineWidth', 2.5);
xlim(xlimits);
ylim(ylimits);
grid on;

xlabel('$\alpha$ in $^\circ$');

%ylabel('in V');

title(sprintf(...
    "b) $V_{cos}(\\alpha)$ f." + ...
    " $V_{cc} = %.1f$ V, $V_{off} = %.2f$ V", Vcc, Voff));

% Vsin
ax6 = nexttile(15, [2 3]);
yline(Voff, 'k-.', 'LineWidth', 2.5);
xlim(xlimits);
ylim(ylimits);
grid on;

xlabel('$\alpha$ in $^\circ$');

%ylabel('in V');

title(sprintf("e) $V_{sin}(\\alpha)$ f." + ...
    " $V_{cc} = %.1f$ V, $V_{off} = %.2f$ V", Vcc, Voff));

% loop through subset of dataset and renew plots %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% lock plots
hold(ax3, 'on');
hold(ax4, 'on');
hold(ax5, 'on');
hold(ax6, 'on');

% loop over indices
for k = 1:length(xIdx)
    i = xIdx(k); j = yIdx(k);
    % H load subset
    Hx = squeeze(ds.Data.Hx(i,j,:));
    Hy = squeeze(ds.Data.Hy(i,j,:));
    % get min max

    % load V subset
    Vcos = squeeze(ds.Data.Vcos(i,j,:));
    Vsin = squeeze(ds.Data.Vsin(i,j,:));

    % update plot 3, 4, 5 and 6
    scatter(ax3, Hx, Hy, 5, [R(i,j), G(i,j), B(i,j)] , 'filled');
    scatter(ax4, Hx, Hy, 5, [R(i,j), G(i,j), B(i,j)], 'filled');
    scatter(ax5, ds.Data.angles, Vcos, 8, [R(i,j), G(i,j), B(i,j)] , ...
        'filled');
    scatter(ax6, ds.Data.angles, Vsin, 8, [R(i,j), G(i,j), B(i,j)] , ...
        'filled');
end

% release plots
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 + "_SubsetPlot_" + 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
```