

plotSimulationDatasetCircle

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

Contents

- [Syntax](#)
- [Description](#)
- [Examples](#)
- [Input Argurments](#)
- [Output Argurments](#)
- [Requirements](#)
- [See Also](#)

Syntax

```
plotSimulationDatasetCircle()
```

Description

plotSimulationDatasetCircle() 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. It loads path from config.mat and scans for file automatically.

Examples

```
plotSimulationDatasetCircle()
```

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 December 02. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotSimulationDatasetCircle()
% 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 available 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: ');

% 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 plot
    fprintf('Detect %d x %d sensors in dataset ...\n', ...
        ds.Info.SensorArrayOptions.dimension, ds.Info.SensorArrayOptions.dimension);
    %
    % xIdx = input("Enter x indices in []: ");
    % yIdx = input("Enter y 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 = fullfile(PathVariables.saveFiguresPath);
fSvgPath = fullfile(PathVariables.saveImagesPath, 'svg');
fEpsPath = fullfile(PathVariables.saveImagesPath, 'eps');
fPdfPath = fullfile(PathVariables.saveImagesPath, 'pdf');

% 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', 'compact', ...
    'TileSpacing', 'compact');

% xlabel(tdl, '$X$ in mm', ...
%     'FontWeight', 'normal', ...
%     'FontSize', 16, ...
%     'FontName', 'Times', ...
%     'Interpreter', 'latex');
%
% ylabel(tdl, '$Y$ in mm', ...
%     'FontWeight', 'normal', ...
%     'FontSize', 16, ...
%     'FontName', 'Times', ...
%     'Interpreter', 'latex');
%

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 m
agnet 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 a
ngles with a ";
subline4 = "step width of %.1f^\\circ$ and a resolution of %.1f^\\circ$. Visualized are circular pa
th of each array position ";
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)];

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;
M = ds.Info.UseOptions.nAngles;
Voff = ds.Info.SensorArrayOptions.Voff;
Vcos = ds.Data.Vcos - Voff;
Vsin = ds.Data.Vsin - Voff;
Hx = ds.Data.Hx;
Hy = ds.Data.Hy;

```

```

% calculate norm values to align circles around position only for x,y
% direction for each sensor dot over all angles.
Vmag = sqrt(Vcos.^2 + Vsin.^2);
Hmag = sqrt(Hx.^2 + Hy.^2);
%Hmag = ds.Data.Habs;

% related to position, multiply scale factor for circle diameter
diameterFactor = 2 * N / ds.Info.SensorArrayOptions.edge;
MaxVmagPos = max(Vmag, [], 3) * diameterFactor;
MaxHmagPos = max(Hmag, [], 3) * diameterFactor;

% Overall maxima, scalar, multiply scale factor for circle diameter
MaxVmagOA = max(Vmag, [], 'all') * diameterFactor;
MaxHmagOA = max(Hmag, [], 'all') * diameterFactor;

% norm and scale volatages and filed strengths
VcosNorm = Vcos ./ MaxVmagPos;
VcosScaled = Vcos / MaxVmagOA;
VsinNorm = Vsin ./ MaxVmagPos;
VsinScaled = Vsin / MaxVmagOA;

HxNorm = Hx ./ MaxHmagPos;
HxScaled = Hx / MaxHmagOA;
HyNorm = Hy ./ MaxHmagPos;
HyScaled = Hy / MaxHmagOA;

% sensor array grid
X = ds.Data.X;
Y = ds.Data.Y;
Z = ds.Data.Z;

% calc limits of plot 1
maxX = ds.Info.UseOptions.xPos + 0.7 * ds.Info.SensorArrayOptions.edge;
maxY = ds.Info.UseOptions.yPos + 0.7 * ds.Info.SensorArrayOptions.edge;
minX = ds.Info.UseOptions.xPos - 0.7 * ds.Info.SensorArrayOptions.edge;
minY = ds.Info.UseOptions.yPos - 0.7 * ds.Info.SensorArrayOptions.edge;

% plot sensor grid in x and y coordinates %%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot each coordinate in loop to create a special shading constant
% reliable to orientation for all matrice
% 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);

% Field strength scaled to overall maxima %%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nexttile;
hold on;
for i = 1:N
    for j = 1:N
        plot(squeeze(HxScaled(i, j, :)) + X(i,j), ...
            squeeze(HyScaled(i, j, :)) + Y(i,j), ...
            'Color', [R(i,j) G(i,j) B(i,j)], ...
            'LineWidth', 1.5)
        line([X(i,j), HxScaled(i,j,1) + X(i,j)], ...
            [Y(i,j), HyScaled(i,j,1) + Y(i,j)], ...
            'Color','k','LineWidth',1.5)
    end
end
hold off;

```

[illegible]

```

for i = 1:N
    for j = 1:N
        plot(squeeze(HxNorm(i, j, :)) + X(i,j), ...
            squeeze(HyNorm(i, j, :)) + Y(i,j), ...
            'Color', [R(i,j) G(i,j) B(i,j)], ...
            'LineWidth', 1.5)
        line([X(i,j), HxNorm(i,j,1) + X(i,j)], ...
            [Y(i,j), HyNorm(i,j,1) + Y(i,j)], ...
            'Color','k','LineWidth',1.5)
    end
end
hold off;

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

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('$H_x$, $H_y$ Normed to Max at each Position', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

% Cosinus, sinus voltage normed to each maxima at position %%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nexttile;
hold on;
for i = 1:N
    for j = 1:N
        plot(squeeze(VcosNorm(i, j, :)) + X(i,j), ...
            squeeze(VsinNorm(i, j, :)) + Y(i,j), ...
            'Color', [R(i,j) G(i,j) B(i,j)], ...
            'LineWidth', 1.5)
        line([X(i,j), VcosNorm(i,j,1) + X(i,j)], ...
            [Y(i,j), VsinNorm(i,j,1) + Y(i,j)], ...
            'Color','k','LineWidth',1.5)
    end
end
hold off;

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

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('$V_{\cos}$, $V_{\sin}$ Normed to Max at each Positions', ...
      'FontWeight', 'normal', ...
      'FontSize', 12, ...
      'FontName', 'Times', ...
      'Interpreter', 'latex');

% 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 + " CirclePlot_" + fLabel;
    savefig(fig, fullfile(fPath, fName));
    print(fig, fullfile(fSvgPath, fName), '-dsvg');
    print(fig, fullfile(fEpsPath, fName), '-depsc', '-tiff', '-loose');
    print(fig, fullfile(fPdfPath, fName), '-dpdf', '-loose', '-fillpage');
end
close(fig);
end

```