plotSimulationCosSinStats

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

Contents

Syntax

```
plotSimulationCosSinStats()
```

Description

plotSimulationCosSinStats() 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

```
plotSimulationCosSinStats()
```

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 30. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotSimulationCosSinStats()
    % 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: ');

% 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

% 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 37 29], ...
    'PaperType', 'a4', ...
    'PaperUnits', 'centimeters', ...
    'PaperOrientation', 'landscape', ...
    'PaperPositionMode', 'auto', ...
    'DoubleBuffer', 'on', ...
    'RendererMode', 'manual', ...
    'Renderer', 'painters');

tdl = tiledlayout(fig, 2, 1, ...
    'Padding', 'compact', ...
    '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 %%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    M = ds.Info.SensorArrayOptions.dimension^2;
    % N = ds.Info.UseOptions.nAngles;
    res = ds.Info.UseOptions.angleRes;
    %angles = ds.Data.angles;
    anglesIP = 0:res:360-res;

    % load V subset and reshape for easier computing statistics
    Vcos = squeeze(reshape(ds.Data.Vcos(:,:,indices), 1, M, nSubAngles));
    Vsin = squeeze(reshape(ds.Data.Vsin(:,:,indices), 1, M, nSubAngles));

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

    % compute statistics of Vcos Vsin %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % interpolate with makima makes best results, ensure to kill nans for
    % fill otherwise fill strokes, use linstyle none for fill without frame
    interpM = 'makima';
    VcosMean = mean(Vcos, 1);
    VcosMeanIP = interp1(subAngles, VcosMean, anglesIP, interpM);
```

```
VcosStd = std(Vcos, 1, 1);
VcosVar = var(Vcos, 1, 1); % std^2

% meanvariation coefficient in percent
VcosMVCP = mean(VcosStd ./ VcosMean) * 100;

VcosUpper1 = VcosMean + VcosStd;
VcosUpper2 = VcosMean + VcosVar;
VcosLower1 = VcosMean - VcosStd;
VcosLower2 = VcosMean - VcosVar;

VcosUpper1IP = interp1(subAngles, VcosUpper1, anglesIP, interpM);
VcosUpper1IP = fillmissing(VcosUpper1IP, 'previous');

VcosLower1IP = interp1(subAngles, VcosLower1, anglesIP, interpM);
VcosLower1IP = fillmissing(VcosLower1IP, 'previous');

VcosUpper2IP = interp1(subAngles, VcosUpper2, anglesIP, interpM);
VcosUpper2IP = fillmissing(VcosUpper2IP, 'previous');

VcosLower2IP = interp1(subAngles, VcosLower2, anglesIP, interpM);
VcosLower2IP = fillmissing(VcosLower2IP, 'previous');

VsinMean = mean(Vsin, 1);
VsinMeanIP = interp1(subAngles, VsinMean, anglesIP, interpM);

VsinStd = std(Vsin, 1, 1);
VsinVar = var(Vsin, 1, 1); % std^2

% meanvariation coefficient in percent
VsinMVCP = mean(VsinStd ./ VsinMean) * 100;

VsinUpper1 = VsinMean + VsinStd;
VsinUpper2 = VsinMean + VsinVar;
VsinLower1 = VsinMean - VsinStd;
VsinLower2 = VsinMean - VsinVar;

VsinUpper1IP = interp1(subAngles, VsinUpper1, anglesIP, interpM);
VsinUpper1IP = fillmissing(VsinUpper1IP, 'previous');

VsinLower1IP = interp1(subAngles, VsinLower1, anglesIP, interpM);
VsinLower1IP = fillmissing(VsinLower1IP, 'previous');

VsinUpper2IP = interp1(subAngles, VsinUpper2, anglesIP, interpM);
VsinUpper2IP = fillmissing(VsinUpper2IP, 'previous');

VsinLower2IP = interp1(subAngles, VsinLower2, anglesIP, interpM);
VsinLower2IP = fillmissing(VsinLower2IP, 'previous');

% plot Vcos Vsin over angles %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% axes limits
%xlimits = [-10 370];
%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];

% Vcos
nexttile;
hold on;

fillStdX = [anglesIP, fliplr(anglesIP)];
fillStdY = [VcosLower1IP, fliplr(VcosUpper1IP)];
fill(fillStdX, fillStdY, [0.95 0.95 0.95], 'LineStyle', 'none');

fillVarX = [anglesIP, fliplr(anglesIP)];
fillVarY = [VcosLower2IP, fliplr(VcosUpper2IP)];
fill(fillVarX, fillVarY, [0.7 0.7 0.7], 'LineStyle', 'none');

yline(Voff, 'k--');
scatter(subAngles, VcosUpper1, [], 'r*');
plot(anglesIP, VcosUpper1IP, 'r-.');
```

```
    scatter(subAngles, VcosMean, [], 'm*');
    plot(anglesIP, VcosMeanIP, 'm-.');
    scatter(subAngles, VcosLower1, [], 'b*');
    plot(anglesIP, VcosLower1IP, 'b-.');


    hold off;
    xlim([-res 360-res]);
    %ylim(ylimits);
    grid on;

    xlabel('$\theta$ in Degree', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$V{cos}(\theta)$ in V', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title(sprintf('Compare $V_{cos}(\\theta)$ for each Array Member $V_{cc} = %.1f$ V, $V
_{off} = %.2f$ V, $\\bar{\\sigma_\\mu} = %.2f$ perc.', ...
                  Vcc, Voff, VcosMVCP), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % Vsin
    nexttile;
    hold on;

    fillStdX = [anglesIP, fliplr(anglesIP)];
    fillStdY = [VsinLower1IP, fliplr(VsinUpper1IP)];
    l1 = fill(fillStdX, fillStdY, [0.95 0.95 0.95], 'LineStyle', 'none');

    fillVarX = [anglesIP, fliplr(anglesIP)];
    fillVarY = [VsinLower2IP, fliplr(VsinUpper2IP)];
    l2 = fill(fillVarX, fillVarY, [0.7 0.7 0.7], 'LineStyle', 'none');

    l3 = yline(Voff, 'k--');
    l4 = scatter(subAngles, VsinUpper1, [], 'r*');
    l5 = plot(anglesIP, VsinUpper1IP, 'r-.');
    l6 = scatter(subAngles, VsinMean, [], 'm*');
    l7 = plot(anglesIP, VsinMeanIP, 'm-.');
    l8 = scatter(subAngles, VsinLower1, [], 'b*');
    l9 = plot(anglesIP, VsinLower1IP, 'b-.');

    hold off;
    xlim([-res 360-res]);
    grid on;

    xlabel('$\theta$ in Degree', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$V{sin}(\theta)$ in V', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
    title(sprintf('Compare $V_{sin}(\\theta)$ for each Array Member $V_{cc} = %.1f$ V, $V
_{off} = %.2f$ V, $\\bar{\\sigma_\\mu} = %.2f$ perc.'...
            , Vcc, Voff, VsinMVCP), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
```

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

    % plot legend %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    l = [l1 l2 l3 l4 l5 l6 l7 l8 l9];
    L = legend(l, {'$2\sigma$', ...
                   '$2\sigma^2$', ...
                   '$V_{off}$', ...
                   '$U_{lim} = \mu + \sigma$', ...
                   sprintf('$%s(U_{lim})$', interpM), ...
                   '$\mu(V)$', ...
                   sprintf('$%s(\\mu)$', interpM), ...
                   '$L_{lim} = \mu - \sigma$', ...
                   sprintf('$%s(L_{lim})$', interpM)}, ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
    L.Layout.Tile = 'east';

    % 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 + "_StatsPlot_" + 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
```

Published with MATLAB® R2020b