



RCSI

UNIVERSITY
OF MEDICINE
AND HEALTH
SCIENCES

Royal College of Surgeons in Ireland

repository@rcsi.com

Violin SuperPlots: visualizing replicate heterogeneity in large data sets.

AUTHOR(S)

Martin Kenny, Ingmar Schoen

CITATION

Kenny, Martin; Schoen, Ingmar (2021): Violin SuperPlots: visualizing replicate heterogeneity in large data sets.. Royal College of Surgeons in Ireland. Journal contribution.
<https://hdl.handle.net/10779/rcsi.15073512.v3>

HANDLE

[10779/rcsi.15073512.v3](https://hdl.handle.net/10779/rcsi.15073512.v3)

LICENCE

CC BY 4.0

This work is made available under the above open licence by RCSI and has been printed from <https://repository.rcsi.com>. For more information please contact repository@rcsi.com

URL

https://repository.rcsi.com/articles/journal_contribution/Violin_SuperPlots_visualizing_replicate_heterogeneity_in_large_data_sets_/15073512/3

```

% This code exemplifies the usage of superviolin.m
% superviolin.m has to be on your MATLAB path or in the current folder
% by Ingmar Schoen, RCSI (c) 2021

% simulate test data
% you can make changes to nrep or ndata to see the effect of numbers of
% biological and technical replicates on the plots
nrep = 5; % number of replicates
ndata = round(random('Uniform',50,100,[1 nrep])); % number of data points per
replicate
% means of replicates
mu0 = 10.; % mean of normal distribution
sigma0 = 1.5; % std of normal distribution
replicate_means = normrnd(mu0,sigma0,[1 nrep]);
% standard deviation of replicates
mu1 = 2.; % mean of normal distribution
sigma1 = 0.5; % std of normal distribution
replicate_std = normrnd(mu1,sigma1,[1 nrep]);
% create cell array with data points of replicates
testdata = {};
for i=1:nrep
    testdata{i} = normrnd(replicate_means(i),replicate_std(i),[1 ndata(i)]);
end
%%
% create plot
figure(1)
clf
superviolin(testdata, 'LUT', 'copper', 'Bandwidth', 0.1);
ylim([0 20])
xlim([0 2])

%% example usage of optional parameters to customize plot
h=figure(2);
clf
set(h, 'Position', [100 100 1200 400])
ax=gca;
superviolin(testdata, 'Parent', ax, 'Xposition', 1);
superviolin(testdata, 'Parent', ax, 'Xposition', 2, 'Width', 0.3);
superviolin(testdata, 'Parent', ax, 'Xposition', 3, 'FaceAlpha', 0.35);
superviolin(testdata, 'Parent', ax, 'Xposition', 4, 'LUT', 'jet');
superviolin(testdata, 'Parent', ax, 'Xposition', 5, 'Centrals', 'robustmean');
superviolin(testdata, 'Parent', ax, 'Xposition', 6, 'Errorbars', 'ci');
superviolin(testdata, 'Parent', ax, 'Xposition', 7, 'LineWidth', 0.1);
ylim([0 20])
xlim([0 8])

function hp = superviolin(dat, varargin)
% Basic implementation of Violin SuperPlots in MATLAB
% Please cite the original work (Kenny & Schoen 2021 Molecular Biology of
% the Cell) when using the code for your work. Thank you.
% by Ingmar Schoen, RCSI (c) 2021
% I would like to gratefully acknowledge Jonas Ries for help with coding.

% inputs:
% dat:          cell array with each cell containing a data vector of a

```

```

%             biological replicate
% varargin: comma-separated list of name - value pair (optional):
%   'Parent':   parent axis handle
%   'Xposition': position of violin plot on x-axis (default: 1)
%   'Width':    width of the violin
%   'LUT':      lookup table for colors: 'lines' (default), 'jet', 'hsv',
%               'parula', or any of the many MATLAB LUTs;
%   'FaceAlpha': opacity of violin stripes (default: 0.8)
%   'LineWidth': line width (in pt) of violin stripe dividing lines (default:
0.5)
%   'Centrals':  either 'mean' (default) or 'median' or 'robustmean'
%   'Errorbars': either 'sem' (default) or 'std' or 'ci'
%   'Bandwidth': bandwidth of kernel density estimator, given as fraction
%               [0.01...0.2] of the data range. The default is [], which
%               uses modified Scott's criterium to determine an
%               appropriate bandwidth.
%
% outputs:
% hp:   plot object handle

% check validity of data input format
if nargin < 1
    msg = 'Required input data not provided.';
    error(msg)
else
    if ~iscell(dat)
        msg = 'Input data is not a cell array. Please consult the
documentation.';
        error(msg)
    else
        if ~isvector(dat{1})
            msg = 'Input cell array does not contain data vectors. Please
consult the documentation.';
            error(msg)
        end
    end
end
if size(dat{1},1)==1
    for i=1:numel(dat)
        dat{i} = dat{i}';
    end
end
end

% parse all optional input parameters, see function below
p=parseinput(varargin);

% set axis for plot
if isempty(p.Parent)
    ax=gca;
else
    ax=p.Parent;
end

% estimate kernel densities, see function below
[histograms, sampledpoints]=makehistogramsdensity(dat,p.Bandwidth);

% prepare stripes for plot
numhist=size(histograms,2);

```

```

h0=zeros(size(histograms,1),1) histograms];
hcum=cumsum(h0,2);
medval=0.5.*hcum(:,end);
histp=hcum-medval;
histp=histp/max(hcum(:,end))*p.Width;

% plot stripes for violin
lutf=str2func(p.LUT);
colors=lutf(numhist);
for k=1:numhist
    hvalh=[histp(:,k) histp(end:-1:1,k+1)];
    edgeh=[sampledpoints sampledpoints(end:-1:1)];
    hp(k)=patch('XData',hvalh(:)+p.Xposition, 'YData',
edgeh(:), 'FaceColor', colors(k,:));
    hp(k).FaceAlpha=p.FaceAlpha;
    hp(k).LineWidth=p.LineWidth;
end

% plot outer violin
hvalh=[histp(:,1) histp(end:-1:1,end)];
edgeh=[sampledpoints sampledpoints(end:-1:1)];
hp(numhist+1)=patch('XData',hvalh(:)+p.Xposition, 'YData',
edgeh(:), 'FaceColor', 'none');
hp(numhist+1).LineWidth=p.LineWidth*2;

% plot means or medians of individual replicates, see function below
for k=1:numhist
    [yp(k), nmed]=getstatisticsdat(dat{k},sampledpoints,p.Centrals);
    xp(k)=mean(histp(nmed,k:k+1));
end
hold(ax, 'on')
for k=1:numhist
    plot(ax, xp(k)+p.Xposition, yp(k), 'ko', 'MarkerFaceColor', colors(k,:), ...
        'LineWidth', 2*p.LineWidth, 'MarkerSize', 8);
end

% plot summary statistics (mean with error bars), see function below
[mhistall,~,sall]=getstatisticsdat(yp,sampledpoints, 'mean', p.Errorbars);

errorbar(p.Xposition,mhistall,sall(1),sall(2), 'k', 'LineWidth', 2, 'CapSize', 8);
errorbar(p.Xposition,mhistall,0.00, 'k', 'LineWidth', 2, 'CapSize', 16);

end

%% local function declarations

function po=parseinput(in)
% This function parses the optional variables or sets their default values
p=inputParser;
addParameter(p, 'Parent', []);
addOptional(p, 'Xposition', 1);
addParameter(p, 'Width', 0.8);
addParameter(p, 'LUT', 'lines');
addParameter(p, 'FaceAlpha', 0.8);
addParameter(p, 'LineWidth', 0.5);
addParameter(p, 'Centrals', 'mean');
addParameter(p, 'Errorbars', 'sem');

```

```

    addParameter(p, 'Bandwidth', []);
    parse(p, in{:});
    po=p.Results;
end

function [histograms, binpositions]=makehistogramsdensity(dat,bww)
% This function creates binned histograms of the raw data for smoothing
% inputs:
%   dat:    data in form of a cell array with each cell containing a data
vector
%   bww:    bandwidth, given as fraction of the actual data range
% outputs:
%   histograms:    cell array with histogram counts for each replicate
%   binpositions:  center positions of bins used for histograms

% pool all raw data
nd = zeros([1 numel(dat)]);
alldat = [];
for i=1:numel(dat)
    nd(i) = numel(dat{i});
    alldat = [alldat dat{i}'];
end

% data range
maxdd = max(alldat);
mindd = min(alldat);
min95 = quantile(alldat,0.05);
max95 = quantile(alldat,0.95);

% kernel bandwidth relative to data range
if ~isempty(bww)
    % bandwidth rescaled to robust data range
    bw = (max95-min95).*bww;
    % sampling points: make sure they are smoother than the smoothing
kernel
    sw = min([bw*0.1,0.005]);
else
    % use modified (*0.5) Scott criterium to automatically determine a
suitable bandwidth
    bw = (max95-min95)*numel(alldat)^(-0.2)*0.5;
    sw = bw*0.1;
end
binpositions = mindd-sw:sw*2:maxdd+sw;

% create density kernel estimates
histograms = zeros(numel(binpositions),numel(dat));
for k=1:length(dat)

    % set Bandwidth for kernel density estimator for each dataset
    % adaptive to number of data points in each replicate
    if isempty(bww)
        min95 = quantile(dat{k},0.05);
        max95 = quantile(dat{k},0.95);
        % use modified (*0.5) Scott criterium to automatically determine a
suitable bandwidth
        bw = (max95-min95)*numel(dat{k})^(-0.2)*0.5;
    end

```

```

        % calculate kernel density estimation for each violin
        [histograms(:,k), ~] = ksdensity(dat{k}, binpositions, 'bandwidth',
bw);
    end

end

function [mhist,nmed,s]=getstatisticsdat(dat,edges,centrals,errorbars)
% This function calculates summary statistics of the data provided
% inputs:
%   dat:    data vector
%   edges:  bin centers of histogram
%   centrals: type of central tendency, 'mean' or 'median'
%   errorbars: type of error bars, 'sem' or 'std' or 'ci'
% outputs:
%   mhist:  central tendency of data
%   nmed:   xposition of mean tendency in stripe
%   s:      error intervals in the form of [min max]

% Central tendency of individual replicates data vectors
switch centrals
    case 'mean'
        mhist=mean(dat);
    case 'median'
        mhist=median(dat);
    case 'robustmean'
        mhist=trimmean(dat,5); % excludes upper and lower 2.5% of data
    otherwise
        disp('Statistics should be mean or median')
end

% Error bars
if nargin > 3
    switch errorbars
        case 'sem' % standard error or the mean
            slow = std(dat)/sqrt(numel(dat));
            s=[slow slow];
        case 'std' % standard deviation
            slow = std(dat);
            s=[slow slow];
        case 'ci' % 95% confidence intervals
            ci = 0.95;
            slow=mhist-quantile(dat,0.5*(1-ci));
            shigh=quantile(dat,0.5*(1+ci))-mhist;
            s=[slow shigh];
        otherwise
            disp('Errorbars should be sem, std or ci')
    end
end

% determine x-position of stripe at height of mean/median value
nmed=(find(mhist<=edges,1));

end

```