```
%% IMAGE BASED INERTIAL TEST (IBIT): Smoothing Sweep Analysis
% Author: Lloyd Fletcher
% PhotoDyn Group, University of Southampton
% http://photodyn.org/
% Date Created: 30th Aug. 2018
% Date Edited: 8th Aug. 2019 - v1.0r
%
% Analyses parametric sweep data and plot heat maps of the systematic,
% random and total errors. The minimum error and associated smoothing
% parameters are ouput to the console.
%
% This work is licensed under the Creative Commons Attribution-
% NonCommercial-ShareAlike 4.0 International License. To view a copy of
% this license, visit http://creativecommons.org/licenses/by-nc-sa/4.0/↙
or
% send a letter to Creative Commons, PO Box 1866, Mountain View, CA↙
94042,
% USA.
%
% If there are any issues with this code please contact:
% Lloyd Fletcher: l.c.fletcher@soton.ac.uk / lloydcolinfletcher@gmail.↙
com

clc
clear all
close all

fprintf('------------------------------------------------------------↙
\n')
fprintf('IBII Processing - Image Deformation Sweep Analysis - v1.0r\n')
fprintf('------------------------------------------------------------↙
\n')

%% INITIALISE: Add path for processing functions
% Add the path for the grid method code and other useful functions:
funcPath = [pwd,'\Functions\'];

% If the default path is not found we should find it
if exist(funcPath,'file') ~= 7
    hWarn = warndlg('Folder for global processing functions not↙
found','Function folder not found');
```

```
    waitfor(hWarn);
    funcPath = uigetdir(pwd,'Locate Global Processing Function Folder');
end
addpath(funcPath);

%% INITIALISE: Load the parametric sweep data file
fprintf('Loading image deformation parametric sweep data file.\n')
hardCodePath = false;
if ~hardCodePath
    [dataFile,dataPath] = uigetfile({'*.*','All Files'},'Select↙
parametric sweep data file');
else
    dataPath = [pwd,'\SmoothingSweepData_Isotropic\'];
    %dataPath = [pwd,'\SmoothingSweepData_ReducedOrtho\'];
    dataFile = 'ParametricImageDefSweep_AllData.mat';
end
load([dataPath,dataFile])

%% INITIALISE: Load the initialisation file
% Get the target stiffness parameters from the initialisation file
fprintf('Loading processing parameters file.\n')
initPath = dataPath;
initFile = 'processingParameters.mat';
if exist([initPath,initFile],'file') ~= 2
    hWarn = warndlg('Processing parameter file does not↙
exist.','Processing parameters not found');
    waitfor(hWarn);
    [initFile,initPath,~] = uigetfile('*.mat','Locate processing↙
parameter file');
end
% Load the processing parameters from file
load([initPath,initFile])

%% PRE-PROCESSING: Unpack variables for plotting on heat maps
fprintf('Unpacking parametric sweep variables for plotting.\n')

% Indices for plot variables
iSK = 1;
iTK = 2;
iAFxx = 1;
iAQxxSG = 2;
```

```
iAQxxVFM = 3;
iAQxxVFO = 4;
iSDQxxSG = 1;
iSDQxxVFM = 2;
iSDQxxVFO = 3;
if strcmp('isotropic',globalOpts.matModel)
    iAQxyVFM = 5;
    iAQxyVFO = 6;
    iSDQxyVFM = 4;
    iSDQxyVFO = 5;
end

currRow = 1;
% Loop over all the workers
for ww = 1:length(saveData)
    for ss = 1:length(saveData(ww).workerKernels{1})
        % Smoothing Kernal Data
        sweepKerns(currRow,iSK) = saveData(ww).workerKernels{1}{ss}.↙
spatialKernel;
        sweepKerns(currRow,iTK) = saveData(ww).workerKernels{1}{ss}.↙
temporalKernel;

        % Pulse peak stress
        sweepAvgs(currRow,iAFxx)= saveData(ww).kernelPulseAvg{1}.↙
peakStress(ss);

        % Qxx Stiffness - SG
        sweepAvgs(currRow,iAQxxSG) = saveData(ww).kernelQxxAvg{1}.SG↙
(ss);
        sweepSDs(currRow,iSDQxxSG) = saveData(ww).kernelQxxSD{1}.SG(ss);
        % Qxx Stiffness - VF Man
        sweepAvgs(currRow,iAQxxVFM) = saveData(ww).kernelQxxAvg{1}.VFMan↙
(ss);
        sweepSDs(currRow,iSDQxxVFM) = saveData(ww). kernelQxxSD{1}.VFMan↙
(ss);
        % Qxx Stiffness - VF Opt
        sweepAvgs(currRow,iAQxxVFO) = saveData(ww).kernelQxxAvg{1}.VFOpt↙
(ss);
        sweepSDs(currRow,iSDQxxVFO) = saveData(ww).kernelQxxSD{1}.VFOpt↙
(ss);
```

```
        if strcmp('isotropic',globalOpts.matModel)
            % Qxy Stiffness - VF Man
            sweepAvgs(currRow,iAQxyVFM) = saveData(ww).kernelQxyAvg{1}.↙
VFMan(ss);
            sweepSDs(currRow,iSDQxyVFM) = saveData(ww).kernelQxySD{1}.↙
VFMan(ss);
            % Qxy Stiffness - VF Opt
            sweepAvgs(currRow,iAQxyVFO) = saveData(ww).kernelQxyAvg{1}.↙
VFOpt(ss);
            sweepSDs(currRow,iSDQxyVFO) = saveData(ww).kernelQxySD{1}.↙
VFOpt(ss);
        end
        currRow = currRow+1;
    end
end

if strcmp('isotropic',globalOpts.matModel)
    targetQxx = material.Qxx;
    targetQxy= material.Qxy;
elseif strcmp('orthotropicReduced',globalOpts.matModel)
    targetQxx = material.Exx;
else
    fprintf('WARNING: specifed material model not recognised.\n')
end

%% DIAGNOSTIC FIGURES: Contour Plots of ID Sweep Results
fprintf('Plotting and saving error heat maps.\n')
savePath = dataPath;
plotParams.formatType = 'article_v2';
plotParams.numContours = 25;
plotParams.imageUpSample = 500;
plotParams.saveImageMatFig = true;
plotParams.saveImageVecFig = true;
plotProps = func_initPlotPropsStruct(plotParams.formatType);

%↙
----------------------------------------------------------------------↙
--
% Optimised VF  - Qxx and (Qxy Iso Only)
if strcmp('isotropic',globalOpts.matModel)
    plotParams.Rows = 2;
```

```
else
    plotParams.Rows = 1;
end
plotParams.Cols = 3;
plotProps.sizePerFigXcm = (plotProps.pageSizeXcm/plotParams.Cols)*1.5;
plotProps.sizePerFigYcm = plotProps.sizePerFigXcm/1.6;
plotParams.plotMin = false;
hf = func_createFigure(plotProps,plotParams);

% X and Y are fixed as the temporal and spatial kernals
xVar = sweepKerns(:,iTK);
yVar = sweepKerns(:,iSK);
plotParams.xStr = '$T_{k}$ ($frames$)';
plotParams.yStr = '$S_{k}$ ($pixels$)';

% Subplot 1: Sys Error - Qxx
subplot(plotParams.Rows,plotParams.Cols,1);
plotParams.tStr = '(a) $Err^{VF}_{sys}$, $Q_{xx}$ $[\%]$';
zVar = (sweepAvgs(:,iAQxxVFO)-targetQxx)/targetQxx *100;
[mapZ,minErr] = func_errorImage3VarsSubPlot(plotParams,xVar,yVar,zVar);
fprintf('Min. Err,VF,sys Qxx = %2f @ Sk = %i , Tk = %i\n',minErr(3),↙
minErr(1),minErr(2))

% Subplot 2:
subplot(plotParams.Rows,plotParams.Cols,2);
plotParams.tStr = '(b) $Err^{VF}_{rnd}$, $Q_{xx}$ $[\%]$';
zVar = sweepSDs(:,iSDQxxVFO)/targetQxx *100;
[mapZ,minErr] = func_errorImage3VarsSubPlot(plotParams,xVar,yVar,zVar);
fprintf('Min. Err,VF,rnd Qxx = %2f @ Sk = %i , Tk = %i\n',minErr(3),↙
minErr(1),minErr(2))

% Subplot 3:
subplot(plotParams.Rows,plotParams.Cols,3);
plotParams.tStr = '(c) $Err^{VF}_{tot}$, $Q_{xx}$ $[\%]$';
zVar = (abs(sweepAvgs(:,iAQxxVFO)-targetQxx)/targetQxx + 2*abs(sweepSDs↙
(:,iSDQxxVFO))/targetQxx)*100;
totQxxErrVF = zVar;
[mapZ,minErr] = func_errorImage3VarsSubPlot(plotParams,xVar,yVar,zVar);
fprintf('Min. Err,VF,tot Qxx = %2f @ Sk = %i , Tk = %i\n',minErr(3),↙
minErr(1),minErr(2))
```

```
if strcmp('isotropic',globalOpts.matModel)
    % Subplot 4: Sys Error - Qxy
    subplot(plotParams.Rows,plotParams.Cols,4);
    plotParams.tStr = '(d) $Err^{VF}_{sys}$, $Q_{xy}$ $[\%]$';
    zVar = (sweepAvgs(:,iAQxyVFO)-targetQxy)/targetQxy *100;
    [mapZ,minErr] = func_errorImage3VarsSubPlot(plotParams,xVar,yVar,↙
zVar);
    fprintf('Min. Err,VF,sys Qxy = %2f @ Sk = %i , Tk = %i\n',minErr(3),↙
minErr(1),minErr(2))

    % Subplot 5:
    subplot(plotParams.Rows,plotParams.Cols,5);
    plotParams.tStr = '(e) $Err^{VF}_{rnd}$, $Q_{xy}$ $[\%]$';
    zVar = sweepSDs(:,iSDQxyVFO)/targetQxy *100;
    [mapZ,minErr] = func_errorImage3VarsSubPlot(plotParams,xVar,yVar,↙
zVar);
    fprintf('Min. Err,VF,rnd Qxy = %2f @ Sk = %i , Tk = %i\n',minErr(3),↙
minErr(1),minErr(2))

    % Subplot 6:
    subplot(plotParams.Rows,plotParams.Cols,6);
    plotParams.tStr = '(f) $Err^{VF}_{tot}$, $Q_{xy}$ $[\%]$';
    zVar = (abs(sweepAvgs(:,iAQxyVFO)-targetQxy)/targetQxy + 2*abs↙
(sweepSDs(:,iSDQxyVFO))/targetQxy)*100;
    [mapZ,minErr] = func_errorImage3VarsSubPlot(plotParams,xVar,yVar,↙
zVar);
    fprintf('Min. Err,VF,tot Qxy = %2f @ Sk = %i , Tk = %i\n',minErr(3),↙
minErr(1),minErr(2))
end

% Save the figure
saveFile = [savePath,'\','Fig_IDErrMapsVFOpt'];
func_saveFigureMultFormat(hf,saveFile,plotProps,plotParams)

%↙
----------------------------------------------------------------------↙
--
% Stress Gauge - Qxx
plotParams.Rows = 1;
plotParams.Cols = 3;
plotProps.sizePerFigXcm = (plotProps.pageSizeXcm/plotParams.Cols)*1.5;
```

```
plotProps.sizePerFigYcm = plotProps.sizePerFigXcm/1.6;
hf = func_createFigure(plotProps,plotParams);

% X and Y are fixed as the temporal and spatial kernals
xVar = sweepKerns(:,iTK);
yVar = sweepKerns(:,iSK);
plotParams.xStr = '$T_{k}$ ($frames$)';
plotParams.yStr = '$S_{k}$ ($pixels$)';

% Subplot 1: Sys Error - Qxx
subplot(plotParams.Rows,plotParams.Cols,1);
plotParams.tStr = '(a) $Err^{SG}_{sys}$, $Q_{xx}$ $[\%]$';
zVar = (sweepAvgs(:,iAQxxSG)-targetQxx)/targetQxx *100;
[mapZ,minErr] = func_errorImage3VarsSubPlot(plotParams,xVar,yVar,zVar);
fprintf('Min. Err,SG,sys Qxx = %2f @ Sk = %i , Tk = %i\n',minErr(3),↙
minErr(1),minErr(2))

% Subplot 2:
subplot(plotParams.Rows,plotParams.Cols,2);
plotParams.tStr = '(b) $Err^{SG}_{rnd}$, $Q_{xx}$ $[\%]$';
zVar = sweepSDs(:,iSDQxxSG)/targetQxx *100;
[mapZ,minErr] = func_errorImage3VarsSubPlot(plotParams,xVar,yVar,zVar);
fprintf('Min. Err,SG,rnd Qxx = %2f @ Sk = %i , Tk = %i\n',minErr(3),↙
minErr(1),minErr(2))

% Subplot 3:
subplot(plotParams.Rows,plotParams.Cols,3);
plotParams.tStr = '(c) $Err^{SG}_{tot}$, $Q_{xx}$ $[\%]$';
zVar = (abs(sweepAvgs(:,iAQxxSG)-targetQxx)/targetQxx + 2*abs(sweepSDs↙
(:,iSDQxxSG))/targetQxx)*100;
totQxxErrSG = zVar;
[mapZ,minErr] = func_errorImage3VarsSubPlot(plotParams,xVar,yVar,zVar);
fprintf('Min. Err,SG,tot Qxx = %2f @ Sk = %i , Tk = %i\n',minErr(3),↙
minErr(1),minErr(2))

% Save the figure to file
saveFile = [savePath,'\','Fig_IDErrMapsSG'];
func_saveFigureMultFormat(hf,saveFile,plotProps,plotParams)

%↙
----------------------------------------------------------------------↙
```

```
--
% Overall Error
plotParams.Rows = 1;
plotParams.Cols = 1;
plotProps.sizePerFigXcm = 8;
plotProps.sizePerFigYcm = plotProps.sizePerFigXcm/1.6;
hf = func_createFigure(plotProps,plotParams);
% X and Y are fixed as the temporal and spatial kernals
xVar = sweepKerns(:,iTK);
yVar = sweepKerns(:,iSK);
plotParams.xStr = '$T_{k}$ ($frames$)';
plotParams.yStr = '$S_{k}$ ($pixels$)';

% Subplot 1: Overall Combined Err
subplot(plotParams.Rows,plotParams.Cols,1);
plotParams.tStr = '$Err_{comb}$ ($Q_{xx}$) $[\%]$';
zVar = totQxxErrSG+totQxxErrVF;
for zz = 1:length(totQxxErrSG)
    zVar(zz) = max([totQxxErrSG(zz),totQxxErrVF(zz)]);
end
[mapZ,minErr] = func_errorImage3VarsSubPlot(plotParams,xVar,yVar,zVar);
fprintf('Min. Err,comb   Qxx = %2f @ Sk = %i , Tk = %i\n',minErr(3),↙
minErr(1),minErr(2))

% Save the figure to file
saveFile = [savePath,'\','Fig_IDErrMapsCombined'];
func_saveFigureMultFormat(hf,saveFile,plotProps,plotParams)

fprintf('Complete.\n')
```