Table of Contents

.... 1
parse inputs 3
start matlabpool if parallel computing tool box avaliable 4
root path 4
main reduction target path construction 4
reference target path construction 5
1D spectra filenames 5
check for required cards in fits header and read the values. add defaults where unavaliable. 5
load misprint, and orientate so echelle dispersion is horizontal 6
trace col 6
load wavelength solution if supplied 7
inital setup 7
load data into local variables 8
find orders 8
trace orders 9
scattered light estimate 17

```
classdef misprint < handleAllHidden
    %MISPRINT MultIorder SPectroscopic ReductIoN Tool
    %
    % Properties are set using key value pairs.
    %
    % Sample Usage:
    %     s2r = misprint('sciencespectrum','reference','flatspectrum','plotAlot',t
    %         'usecurrentfolderonly',true,...
    %         'numOfOrders',14,'numOfFibers',29,...
    %         'forceTrace',false,'forceExtract',false,...
    %         'forceDefineMaskEdge',false,'needsMask',false,...
    %         'peakcut',0.07,'minPeakSeperation',3,...
    %         'numTraceCol',40,'firstCol',140,'lastCol',300,...
    %         'parallel',false);
    %
    %     self.getMaskForIncompleteOrders;
    %     self.traceSpectra;
    %     self.extractSpectra;
    %     self.getP2PVariationsAndBlaze(false);
    %
    %     self.plotSpectraFor(1:14,true,false)
    %
    % Copyright (C) Chris Betters 2012-2014

    properties
        targetBaseFilename, % base filename of target (file with spectra to be ext
        targetPath, % path to target fits file.
        rootDirectory, % path to current directory, should equal [pwd '/'].
        referenceBaseFilename, % base filename of reference fits file (i.e. a flat
        referencePath, % path to reference fits file.

        targetHeader, % structure with target fits header.
```

```
referenceHeader, % structure with reference fits header.

SpectraFitsSaveFileName, % filename of fits file to save extracted spectra
ReferenceSpectraFitsSaveFileName, % spectra previously extracted from the
FlatReferenceSpectraFitsSaveFileName, % filename of fits file to for use a

spectraTracePath, % path to previously saved trace data for reference fits

useReference, % flag to indicate if valid reference data has been set/ffou
plotAlot, % flag to show raw image and plots during tracing. It can plot a
forceTrace, % flag to force a trace of spectra in the current image. Can n
forceExtract, % flag to force a new extraction of the current image. This
forceDefineMaskEdge, % flag to force a new definition of the mask/clipping
needsMask, % flag to indicate if image requires clipping.
clipping, % vector of pixels from [left top right bottom] to clip.
parallel, % flag tin indicate if parallel compuyting toolbox should/can be
minPeakSeperation, % min peak seperation for tracing and peakfinder

numOfOrders, % number of diffraction orders in image.
numOfFibers, % number of spectra (fibres) in each order.

gain, % gain (e-/adu) read from fits file
readNoise, % read noise (rms e-) read from fits file
dispAxis, % axis of primary dispersion read from fits file

imdata, % target image data.
imvariance, % estiamted variance for target image data.
mask, % mask of clipped regions.
imdim, % size of imdata (equals size(imdata))

spectraValues, % extracted spectra values.
spectraVar, % var for extracted spec values.
backgroundValues, % background value from extraction

finalSpectra, %  linearised version of complete spectrum for individual fi
finalSpec, % linearised combined spectrum
finalSpectraVar, % varience for finalSpectra
finalSpecVar, % varience for finalSpec
finalWave, % linearised wavelength scale for finalSpectra and finalSpec

referenceSpectraValues, % extracted spectraValues of reference/flat
P2PVariationValues, % pixel to pixel variation from reference.
flatBlaze, % estimated blaze from reference.

wavematfile, % mat file with wavelength fit paramaters
wavefit,% wavelength soultion for each fibre
diffractionOrder, % estimated diffraction order from wavefit

numTraceCol, % number of columns to use when tracing.
firstCol, % first column of trace
lastCol, % last column of trace

orderEdges,  % detected edges of the orders
specCenters, % polynoimail interpolated fitted y axis centeres of the spec
```

```
        specWidth,   % polynomial interpolated width of the spectra from gaussain
        meanSpecWidth, % mean width of spceetra in each order.
        meanOrderWidth, % mean width of each order

        fittedCenters, % center of spectrum (vertical) from gaussian fit
        fittedCol, % column used to get progile for fit
        fittedWidth, % width of specturm (vertical) from gaussain fit
        fittedParamters, % all fit paramaters from trace

        usecurrentfolderonly, % flag to note use my maxiumDL/PIMMS echelle file st
        peakcut, % MINPEAKHEIGHT for spectra tracing detection. (fraction of mean

        OXmethod, % name of method to use for optimal extraction.
    end

    methods
        function self=misprint(targetBaseFilename,varargin)

            % init the MISPRINT class. Pasre all inputs, load relevant files.
```

parse inputs

```
            p = inputParser;

            p.addRequired('targetBaseFilename', @(x) ischar(x));
            p.addParamValue('reference','');
            p.addParamValue('forceTrace'            , false, @(x) islogical(x));
            p.addParamValue('forceExtract'          , false, @(x) islogical(x));
            p.addParamValue('plotAlot'      , false, @(x) islogical(x));
            p.addParamValue('forceDefineMaskEdge', false, @(x) islogical(x));
            p.addParamValue('needsMask', true, @(x) islogical(x));
            p.addParamValue('numOfOrders', 15, @(x) isnumeric(x));
            p.addParamValue('numOfFibers', 19, @(x) isnumeric(x));
            p.addParamValue('usecurrentfolderonly',false, @(x) islogical(x));
            p.addParamValue('peakcut', 0.8, @(x) isnumeric(x));
            p.addParamValue('parallel',true, @(x) islogical(x));
            p.addParamValue('numTraceCol', 10, @(x) isnumeric(x));
            p.addParamValue('dispAxis', [], @(x) isnumeric(x));
            p.addParamValue('wavesolution', '', @(x) ischar(x));
            p.addParamValue('minPeakSeperation', 3, @(x) isnumeric(x));
            p.addParamValue('firstCol', 0, @(x) isnumeric(x));
            p.addParamValue('lastCol', 0, @(x) isnumeric(x));
            p.addParamValue('clipping',[0 0 0 0], @(x) isnumeric(x) && length(x)==
            p.addParamValue('OXmethod','MPDoptimalExtBack', @(x) ismethod(self,x))

            p.parse(targetBaseFilename,varargin{:});

            self.numOfOrders=p.Results.numOfOrders;
            self.numOfFibers=p.Results.numOfFibers;
            self.forceExtract=p.Results.forceExtract;
            self.forceTrace=p.Results.forceTrace;
            self.plotAlot=p.Results.plotAlot;
            self.forceDefineMaskEdge=p.Results.forceDefineMaskEdge;
```

```
self.needsMask=p.Results.needsMask;

self.usecurrentfolderonly=p.Results.usecurrentfolderonly;
self.peakcut=p.Results.peakcut;

self.parallel=p.Results.parallel;
self.numTraceCol=p.Results.numTraceCol;
self.firstCol=p.Results.firstCol; %if zero set to 20 minus miage size
self.lastCol=p.Results.lastCol; %if zero set to 20 minus miage size (a

self.dispAxis=p.Results.dispAxis;

self.minPeakSeperation=p.Results.minPeakSeperation;

self.clipping=p.Results.clipping;
self.OXmethod=p.Results.OXmethod;
```

```
Error using chrislib.misprint (line 130)
Not enough input arguments.
```

start matlabpool if parallel computing tool box avaliable

```
if self.parallel
    if license('test', 'distrib_computing_toolbox')
        if isempty(gcp('nocreate'))
            parpool('local')
        end
    else
        warning('MISPRINT:init:useDistribComputingToolbox:notAvalaible
    end
end
```

root path

```
self.rootDirectory=[pwd '/'];
```

main reduction target path construction

```
self.targetBaseFilename=p.Results.targetBaseFilename;

if ~self.usecurrentfolderonly
    self.targetPath = [self.rootDirectory self.targetBaseFilename '/re
else
    self.targetPath = [self.rootDirectory self.targetBaseFilename '.fi
end
% check the file is a valid fits.
self.checkForReducedFitsAt(self.targetPath);

% get fits header
self.targetHeader=fitsheader(self.targetPath);
```

reference target path construction

```
self.referenceBaseFilename=p.Results.reference;

if isempty(self.referenceBaseFilename)
    self.useReference     = false;
    self.spectraTracePath = [self.rootDirectory self.targetBaseFilenam
else
    self.useReference     = true;
    self.spectraTracePath = [self.rootDirectory self.referenceBaseFile
    if ~self.usecurrentfolderonly
        self.referencePath    = [self.rootDirectory self.referenceBase
    else
        self.referencePath    = [self.rootDirectory self.referenceBase
    end
    % check the file is a valid fits.
    self.checkForReducedFitsAt(self.targetPath);

    % get fits header
    self.referenceHeader  = fitsheader(self.referencePath);


    self.ReferenceSpectraFitsSaveFileName=[self.referenceBaseFilename
    self.referenceSpectraValues=fitsread(self.ReferenceSpectraFitsSave

    self.FlatReferenceSpectraFitsSaveFileName=[self.referenceBaseFilen

    %assertWarn(isfield(self.referenceHeader,'IMAGETYP') && strcmp(sel
    %    'MISPRINT:init:referenceNotAFlat',...
    %    'Reference Frame has not been tagged as a flat in fits header
end
```

1D spectra filenames

```
self.SpectraFitsSaveFileName=[self.targetBaseFilename '-1D-spectra.fit
```

check for required cards in fits header and read the values. add defaults where unavailable.

```
if isfield(self.targetHeader,'READNOIS')
    self.readNoise=self.targetHeader.READNOIS;
elseif isfield(self.targetHeader,'RO_NOISE')
    self.readNoise=self.targetHeader.RO_NOISE;
else
    self.readNoise=11.3; % atik default
    %                  if strcmp(self.targetHeader.INSTRUME,'ArtemisHSC
    %                      fitsAddHeaderKeyword(self.targetPath,'READNO
    %                  end
```

```
end

if isfield(self.targetHeader,'GAIN')
    self.gain=self.targetHeader.GAIN;
elseif isfield(self.targetHeader,'RO_GAIN')
    self.gain=self.targetHeader.RO_GAIN;
else
    self.gain=0.43; % fli default
    %                 if strcmp(self.targetHeader.INSTRUME,'ArtemisHSC
    %                     fitsAddHeaderKeyword(self.targetPath,'GAIN',
    %                 end
end

if isempty(self.dispAxis)
    if isfield(self.targetHeader,'DISPAXIS')
        self.dispAxis=self.targetHeader.DISPAXIS;
    else
        self.dispAxis=1; % atik default
        %if strcmp(self.targetHeader.INSTRUME,'ArtemisHSC')
        %    fitsAddHeaderKeyword(self.targetPath,'DISPAXIS',self.disp
        %end
    end
end
```

load misprint, and orientate so echelle dispersion is horizontal

```
self.imdata=fitsread(self.targetPath);

if self.dispAxis==1
    self.imdata=fliplr(self.imdata'); %
end

if sum(self.clipping)
    %[left top right bottom]
    self.imdata=self.imdata(max([1 self.clipping(2)]):end-self.clippin
    if ~isempty(self.wavefit)
        self.wavefit=self.wavefit(:,max([1 self.clipping(1)]):end-self
    end
end

%self.imdata=rot90(self.imdata,2);
self.imdim=size(self.imdata);
self.imvariance=(self.readNoise/self.gain)^2 + abs(self.imdata) / self
```

trace col

```
if ~self.lastCol
    self.lastCol=self.imdim(2)-20;
end
if ~self.firstCol
```

```
        self.firstCol=20;
    end
```

load wavelength solution if supplied

```
    if ~isempty(p.Results.wavesolution)
        self.wavematfile=p.Results.wavesolution;
        wavepayload=load(self.wavematfile,'p','S','mu');
        p=wavepayload.p;
        S=wavepayload.S;
        mu=wavepayload.mu;
        self.wavefit=zeros(self.numOfFibers,self.imdim(2),self.numOfOrders
        for o=1:self.numOfOrders;
            for f=1:self.numOfFibers;
                self.wavefit(f,:,o)=polyval(p(f,:,o),1:self.imdim(2),S(f,:
            end
        end
        self.diffractionOrder=round(2*1e-3/31.6*cosd(5)*sind(63.2)./(mean(
    end

end

function runDefaultExtraction(self)
    % run default set of extraction commands

    self.getMaskForIncompleteOrders;
    self.traceSpectra;
    self.extractSpectra;
    self.getP2PVariationsAndBlaze
end

function traceSpectra(self,varargin)

    % trace spectra from flat.
    %
    % optional inputs misprint.traceSpectra(inputimage,numOfOrders,numOfFi
    % inputimage is same format as misprint.imdata
```

inital setup

if preexisting trace exists it is loaded (unless forceTrace set)

```
    if nargin==1
        if (~exist(self.spectraTracePath,'file') || self.forceTrace ) && ~

            assertWarn(self.forceTrace & exist(self.spectraTracePath,'file
                'MISPRINT:traceSpectra:TraceForced',...
                'Tracing was forced, this will overwrite previous trace.')

            if ~exist(self.spectraTracePath,'file'); disp(['Tracefile: ' s
        else
            assert(~(self.forceTrace & self.useReference),...
                'MISPRINT:traceSpectra:TraceForcedWithUseReferenceSet',...
                'Tracing can not be forced when useReference is set')

            assert(~(self.useReference & ~exist(self.spectraTracePath,'fil
                'MISPRINT:traceSpectra:ReferecnceTraceFileNotFound',...
                [self.spectraTracePath ' was not found and is required as

            load(self.spectraTracePath,'specCenters','specWidth','orderWid

            if self.useReference
                %disp(['Using reference trace: ' self.spectraTracePath])
            else
                disp(['Using previous trace: ' self.spectraTracePath])
            end

            self.meanSpecWidth=squeeze(mean(specWidth,2));
            self.meanOrderWidth=squeeze(mean(orderWidth,2));
            self.specCenters=specCenters;
            self.specWidth=specWidth;
            self.orderEdges=orderEdges;

            self.fittedCenters=means;
            self.fittedCol=columns;
            self.fittedWidth=widths;
            self.fittedParamters=fitxs;

            return % end function call after loading data
        end
    end
```

load data into local variables

```
x=1:self.imdim(1);
if nargin==4
    inputimage = varargin{1};
    numOfOrders = varargin{2};
    numOfFibers = varargin{3};
else
    inputimage=self.imdata;
    numOfOrders = self.numOfOrders;
    numOfFibers = self.numOfFibers;
end

imdata=inputimage.*self.mask;
```

find orders

```
if (self.numTraceCol>=self.imdim(2))
    columns=1:self.imdim(2);
    warning('MISPRINT:fitAllOfTheThings','You just asked for a fit to
    reply = input('Are your sure?? Y/N [Y]: ', 's');
    if ~strcmpi('Y',reply)
        error('MISPRINT:traceSpectra:userInterpupt','MISPRINT terminta
    end
```

```
else
    columns=round(linspace(self.firstCol, self.lastCol, self.numTraceC
end

imcol=imdata(:,columns); % sliced image

disp('Running order tracer. This may take some time.')
for i=1:length(columns)
    [yp,index]=findpeaks(imcol(:,i),'NPEAKS',numOfOrders*numOfFibers,'
    if self.plotAlot
        figure(i);clf
        plot(x,imcol(:,i),index,yp,'xr');
        line([1 length(imcol(:,i))],[max(imcol(:,i)) max(imcol(:,i))]*
        title([num2str(columns(i))])
    end
    if numOfOrders==1
        orderWidth=self.imdim(1);
        orderCenter=round(self.imdim(1)/2);
        orderEdges(:,i)=[1 self.imdim(1)];
    else
        orderWidth=diff(index(1:numOfFibers:end));
        orderCenter=mean([index(numOfFibers:numOfFibers:end) index(1:n
        %error(' ')
        orderEdges(:,i)=[orderCenter(1)-orderWidth(1)/2;...
            mean([index(numOfFibers:numOfFibers:end-numOfFibers)...
            index(numOfFibers+1:numOfFibers:end)],2); orderCenter(end)
    end

    %                 if self.plotAlot
    %                     plot(imcol(:,i))
    %                     hold on
    %                     %line(repmat(orderEdges(:,i)',[2,1]),[zeros(
    %                     hold off
    %                 end
end
```

trace orders

fit gaussian to profile in columns for each order.

```
fitxs=zeros(numOfOrders,3*numOfFibers+1,length(columns));
for i=1:length(columns)
    for order=1:numOfOrders
        disp(['Column:' num2str(columns(i)) ' | Fitting Spectra in Ord

        orderProfileX=round(max(orderEdges(order,i),1):min(orderEdges(
        orderProfile=imcol(orderProfileX,i);
        orderProfile=orderProfile/max(orderProfile);


        [~, means(order,:,i), widths(order,:,i), fitxs(order,:,i)] = .
            self.fitNGaussainsAlt(numOfFibers,orderProfileX, orderProf
                    %                    [~, means(order,:,i), widths(order,:,i),
                    %                         fitNGaussains(numOfFibers,orderProfi

                    if self.plotAlot
                        figure(i);clf;
                        %subplot(5,4,columns)
                        plot(orderProfileX,sum(self.nGausFunc(fitxs(order,:,i),ord
                            orderProfileX,orderProfile,'-')
                        title(['Order: ' num2str(order) ' Column: ' num2str(column
                        %pause(0.1)
                    end
                end
            end

            specCenters=self.polyfitwork(self.imdim,means,columns,3);
            specWidth=self.polyfitwork(self.imdim,widths,columns,3);
            meanSpecWidth=squeeze(mean(self.specWidth,3));

            self.fittedCenters=means;
            self.fittedCol=columns;
            self.fittedWidth=widths;

            self.meanSpecWidth=meanSpecWidth;
            self.specCenters=specCenters;
            self.specWidth=specWidth;
            self.orderEdges=orderEdges;
            self.fittedParamters=fitxs;
            self.meanOrderWidth=squeeze(mean(orderWidth,2));

            save(self.spectraTracePath,'specCenters','specWidth','orderWidth','ord

        end

        function getMaskForIncompleteOrders(self)
            %  get mask for incomplete orders

            if ~self.needsMask
                self.mask=ones(self.imdim);
                return % no clipe, so mask is ones.
            end

            if isfield(self.targetHeader,'CLIPTL') && isfield(self.targetHeader,'C
                topEdges=[self.targetHeader.CLIPTL self.targetHeader.CLIPTR];
                bottomEdges=[self.targetHeader.CLIPBL self.targetHeader.CLIPBR];
            else
                self.forceDefineMaskEdge=true; % overide default as clipping is ne
            end

            if self.forceDefineMaskEdge
                echfig=figure(1);
                imagesc(self.imdata);
                axis([1 self.imdim(2) 1 self.imdim(1)*0.5]) % show top half of ima
                [~,y]=getpts(echfig);
                topEdges=[y(1) y(2)];
                axis([1 self.imdim(2) self.imdim(1)-self.imdim(1)*0.3 self.imdim(1
                [~,y]=getpts(echfig);
                bottomEdges=[y(1) y(2)];
            end

            % make mask of image to exclude incomplete orders
            xi=[0; self.imdim(2);              self.imdim(2);                  0];
            yi=[0;        0;    topEdges(2); topEdges(1)];
            BW1 = roipoly(self.imdata,xi,yi);

            xi=[       0; self.imdim(2);           self.imdim(2);
            yi=[self.imdim(1); self.imdim(1);     bottomEdges(2);     bottomEdges(

            BW2 = roipoly(self.imdata,xi,yi);
            self.mask=~BW1 & ~BW2;

            if self.forceDefineMaskEdge
                imagesc(self.imdata.*self.mask)
                reply = input('Should I add to Fits Header Y/N [N]: ', 's');
                if isempty(reply)
                    reply = 'N';
                end
                if strncmpi(reply,'Y',1)
                    disp('saving clips to header')
                    fitsAddHeaderKeyword(self.targetPath,'CLIPTL',topEdges(1),' ')
                    fitsAddHeaderKeyword(self.targetPath,'CLIPTR',topEdges(2),' ')
                    fitsAddHeaderKeyword(self.targetPath,'CLIPBL',bottomEdges(1),'
                    fitsAddHeaderKeyword(self.targetPath,'CLIPBR',bottomEdges(2),'
                end
            end

            if self.plotAlot
                figure(1)
                imagesc(self.imdata.*self.mask)
            end

        end

        function getP2PVariationsAndBlaze(self,varargin)
            % get smoothed version of flat spectrum (ie blaze) and pixel to
            % pixel variations (flatspectrum./smooth flat spectrum)
            %
            %  load reference
            if length(varargin)==2
                referenceFile=varargin{2};
            elseif self.useReference
                referenceFile=self.ReferenceSpectraFitsSaveFileName;
            end

            if ~isempty(varargin)
                force=varargin{1};
            else
                force=false;
            end
            matpayload=load(self.spectraTracePath,'flatBlaze','P2PVariationValues'
            if ~force && isfield(matpayload,'flatBlaze') && isfield(matpayload,'P2
                self.flatBlaze=matpayload.flatBlaze;
                self.P2PVariationValues=matpayload.P2PVariationValues;
            else
                assertWarn(force,'MISPRINT:getP2PVariationsAndBlaze:forced','P2P a
                mask=ones(self.numOfFibers,self.imdim(2));
%                  mask(end-50:end)=NaN;
%                  mask(1:50)=NaN;

                spectraValues=self.spectraValues;
%                  for or=1:self.numOfOrders
%                      spectraValues(:,:,or)=bsxfun(@rdivide,spectr
%                  end
                flatBlaze=zeros(size(self.spectraValues));
                P2PVariationValues=zeros(size(self.spectraValues));

                for f=1:self.numOfFibers
                    for or=1:self.numOfOrders
                        %error('')
                        flatBlaze(f,:,or)=csaps(1:self.imdim(2),spectraValues(f,:,
                        %flatBlaze(f,:,or)=smooth(spectraValues(f,:,or),100);
                    end
                end
                P2PVariationValues=spectraValues./flatBlaze;
                self.flatBlaze=flatBlaze;
                self.P2PVariationValues=P2PVariationValues;

                self.flatBlaze(isnan(self.flatBlaze))=1;
                self.P2PVariationValues(isnan(self.P2PVariationValues))=1;

                save(self.spectraTracePath,'flatBlaze','P2PVariationValues','-appe
            end
        end

        function getBlazeAlt(self,varargin)
            if ~isempty(varargin)
                force=varargin{1};
            else
                force=false;
            end

            matpayload=load(self.spectraTracePath,'flatBlaze','P2PVariationValues'
            if ~force && isfield(matpayload,'flatBlaze') && isfield(matpayload,'P2
                self.flatBlaze=matpayload.flatBlaze;
                self.P2PVariationValues=matpayload.P2PVariationValues;
            else

                assertWarn(force,'MISPRINT:getP2PVariationsAndBlaze:forced','P2P a

                x=[1:2498];

                opts1 = fitoptions( 'Method', 'LinearLeastSquares' );
                opts1.Normalize = 'on';
                ft1=fittype('poly2');

                opts2 = fitoptions( 'Method', 'LinearLeastSquares' );
                opts2.Normalize = 'on';
                ft2=fittype('poly3');

                for f=1:self.numOfFibers
                    for o=1:self.numOfOrders
                        spec=self.spectraValues(f,:,o);

                        % ft=fittype( 'smoothingspline' );
                        % opts = fitoptions( 'Method', 'SmoothingSpline' );
                        % opts.Normalize = 'on';
                        % opts.SmoothingParam = 1e-5;

                        [fitresult, gof] = fit( x', spec', ft1, opts1);
                        %plot(detrend(spec'./feval(fitresult,x))+1); grid on

                        ignore=detrend(spec'./feval(fitresult,x))+1 < 1;

                        [fitresult, gof] = fit( x(~ignore)', medfilt1(spec(~ignore

                        blaze=feval(fitresult,x);
                        flatBlaze(f,:,o)=blaze./max(blaze);
                        %plot(x,squeeze(flatBlaze(f,:,o)),x,self.spectraValues(f,:
                        %pause(0.2)
                    end
                end

                P2PVariationValues=ones(size(flatBlaze));
                self.flatBlaze=flatBlaze;
                self.P2PVariationValues=P2PVariationValues;

                %save(self.spectraTracePath,'flatBlaze','P2PVariationValues','-app

            end
        end

        function plotSpectraValuesFor(self,orders,shouldFlat,shouldP2PV)
            % plot spectra orders specifed. three arguments orders,shouldFlat,shou

            if shouldFlat && shouldP2PV
                FlattenedSpectra=self.spectraValues./self.flatBlaze./self.P2PVaria
            elseif shouldFlat && ~shouldP2PV
                FlattenedSpectra=self.spectraValues./self.flatBlaze;
            elseif ~shouldFlat && shouldP2PV
                FlattenedSpectra=self.spectraValues./self.P2PVariationValues;
            else
                FlattenedSpectra=self.spectraValues;
            end

            if isempty(self.wavefit)
                for order=orders
                    figure(order);clf;
                    subplot(1,2,1)
                    %imagesc(log10(self.imdata-min2(self.imdata)+1))
                    imagesc(self.imdata)
                    %set(gca, 'CLim',[0 1000])
                    ylim([min2(squeeze(self.specCenters(order,:,:)))-50 max2(squee
                    hold on
                    plot(1:self.imdim(2),squeeze(self.specCenters(order,:,:)),'k',
                    %axis image

                    subplot(1,2,2)
                    for f=1:self.numOfFibers
                        FlattenedSpectraNorm(f,:,order)=FlattenedSpectra(f,:,order
                        plot(1:self.imdim(2),FlattenedSpectraNorm(f,:,order)+(self
                        hold all
                    end
                end
                hold off
                grid on
            else
                for order=orders
                    figure(order);clf;

                    subplot(1,2,1)
                    imagesc(log10(self.imdata-min2(self.imdata)+1))
                    %imagesc(self.imdata)
                    %set(gca, 'CLim',[0 1000])
                    ylim([min2(squeeze(self.specCenters(order,:,:)))-50 max2(squee
                    hold on
                    plot(1:self.imdim(2),squeeze(self.specCenters(order,:,:)),'k',
                    title(['Order ' num2str(self.diffractionOrder(order))])
                    xlabel('Primary-dispersion axis (pixels)')
                    ylabel('Cross-dispersion axis (pixels)')

                    subplot(1,2,2)
                    for f=1:self.numOfFibers
                        FlattenedSpectraNorm(f,:,order)=FlattenedSpectra(f,:,order
                        plot(self.wavefit(f,:,order),FlattenedSpectraNorm(f,:,orde
                        hold all
                    end

                    %                     plot(self.wavefit(:,:,order)',FlattenedS
                    %                     xlabel('Wavelength (nm)')
                end
                hold off
                grid on
            end
        end

        function plotSingleFibre(self,f,shouldFlat,shouldP2PV)
            % plot spectra for signle fibre across multiple orders, three argument
            if shouldFlat && shouldP2PV
                FlattenedSpectra=self.spectraValues./self.flatBlaze./self.P2PVaria
            elseif shouldFlat && ~shouldP2PV
```

```
                FlattenedSpectra=self.spectraValues./self.flatBlaze;
            elseif ~shouldFlat && shouldP2PV
                FlattenedSpectra=self.spectraValues./self.P2PVariationValues;
            else
                FlattenedSpectra=self.spectraValues;
            end

            if isempty(self.wavefit)
                plot(bsxfun(@plus, repmat([1:self.imdim(2)],[self.numOfOrders 1])'
            else
                plot(squeeze(self.wavefit(f,:,:)),squeeze(FlattenedSpectra(f,:,:))
            end
        end

        function plotFinalSpectra(self)
            for f=1:self.numOfFibers
                plot(self.finalWave,self.finalSpectra(f,:)+(self.numOfFibers-f)*2)
                hold all
            end
            hold off
            xlabel('Wavelength (nm)')
            hold on
            orderwaveedges=squeeze(max(min(self.wavefit,[],2),[],1));
            line([orderwaveedges orderwaveedges], [0 40],'LineWidth',1,'Color','k'

            hold off
        end

        function plotFinalSpec(self)
            %             [sunflux, sunwave] = getsunspec(min(self.finalWave), max
            %             [telflux, telwave] = getTelluricSpec(min(self.finalWave)
            %
            %             plot(sunwave,sunflux,telwave,telflux)
            %            hold all
            plot(self.finalWave,self.finalSpec/max(self.finalSpec))
            orderwaveedges=squeeze(max(min(self.wavefit,[],2),[],1));
            hold on
            line([orderwaveedges orderwaveedges], [0 1],'LineWidth',2,'Color','k')

            hold off
            xlabel('Wavelength (nm)')
        end

        function filterBadPixels(self,Nsigma,thresh,shouldPlot)
            % filter bad pixels. three arguments Nsigma,thresh,shouldPlot
            im=self.imdata;
            im(im<=0)=1;

            imdiff=medfilt2(im,[2 2])./im; % try and highlight odd pixels

            imdiff=imdiff-mean2(imdiff); % set mean to zero

            bad1=imdiff>thresh; % very larger value can bias std, so clip them.
            badpixel=(imdiff>std2(imdiff(~bad1))*Nsigma | imdiff<-std2(imdiff(~bad

            self.imdata(badpixel)=NaN;

            self.imdata=inpaint_nans(self.imdata);

            if shouldPlot
                figure(shouldPlot);clf
                [badx, bady]=find(badpixel);

                imagesc(self.imdata)
                hold on
                shouldPlot(bady,badx,'wx')
                hold off
            end

        end

        function blurred=removeIntensityGradientInImdata(self,avgWin)
            % smooth whole image, then divided original by that. Usefull for to
            % improve flat tracing. one arguments avgWin (window for smoothing)
            PSF = fspecial('average', [1 1]*avgWin);

            blurred = imfilter(self.imdata, PSF, 'conv', 'symmetric');
            blurred=blurred/mean2(blurred);

            if self.plotAlot
                subplot(1,3,1)
                imagesc(self.imdata)
                subplot(1,3,2)
                imagesc(blurred)
                subplot(1,3,3)
                imagesc(self.imdata./blurred)
            end
            %self.imdata=self.imdata./blurred;
        end

        function getBackgroundBetweenOrders(self)

            self.imdata(self.imdata<0)=0;
            locs=self.orderEdges';
            locs(locs>3362)=3362;

            imagesc(log10(self.imdata))
            hold on;
            plot(self.fittedCol,locs','bx')
            hold off;
            pks=[];


            filtedimdata=medfilt2(self.imdata);

            %locs(89,:)=(locs(88,:) + locs(90,:)) / 2;
```

```
for i = 1:size(locs,2)
    %   pks(i,:)=self.imdata(self.fittedCol,round(self.orderEdges(,:)
    p = impixel(filtedimdata,self.fittedCol,locs(:,i)');
    pks(:,i) = p(:,1);
end



cols=self.fittedCol;
```

scattered light estimate

```
invertedimdata=(1./(self.imdata)).*self.mask;
invertedimdata(isinf(invertedimdata))=0;
figure(1)
imagesc(log10(invertedimdata))
x=1:self.imdim(1);


inverpks=1./pks;
cols2=repmat(cols,[size(locs,2),1])';
figure(3);clf
h(2)=surface(cols2,locs,pks,'EdgeColor','none');
xlim([1 self.imdim(2)])
ylim([1 self.imdim(1)])
set(get(h(2),'Parent'),'YDir','reverse')


figure(2);clf
sfun=scateringTestFit(cols2, locs, inverpks);



figure(4);clf
[XI,YI]=meshgrid(1:self.imdim(2), 1:self.imdim(1));

subplot(1,2,2)
imagesc(1./feval(sfun,XI,YI).*self.mask)
title('Estimated Scattering (from Inter-Order Regions)')

subplot(1,2,1)
h(2)=surface(cols2,locs,1./pks,'EdgeColor','none');
xlim([1 self.imdim(2)])
ylim([1 self.imdim(1)])
set(get(h(2),'Parent'),'YDir','reverse')


%self.imdata=self.imdata-feval(sfun,X,Y)
imagesc(self.imdata-1./feval(sfun,XI,YI))
hold on; plot(cols2,locs,'wx');hold off
title('PIMMS Echelle Detector Image')

%self.imdata=self.imdata-1./feval(sfun,XI,YI)
```

```
            imagesc(log10(self.imdata))

            return
            self.forceTrace=true;
            self.forceExtract=true;

            self.getMaskForIncompleteOrders;
            self.traceSpectra;
            %self.specCenters=self.specCenters;
            self.extractSpectra;
            self.getP2PVariationsAndBlaze
            set(0,'DefaultFigureWindowStyle','docked')

        end

        function spectraValues=extractSpectra(self)
            % extract spectra using trace - each order done individually (faster).

            if ~exist(self.SpectraFitsSaveFileName,'file') || self.forceExtract
                spectraValues=zeros(self.numOfFibers,self.imdim(2),self.numOfOrder
                spectraVar=zeros(self.numOfFibers,self.imdim(2),self.numOfOrders);
                backgroundValues=zeros(size(self.imdata));self.imdata;

                assertWarn(self.forceExtract,...
                    'MISPRINT:extractSpectra:forceExtractFlagSet',...
                    'Force extraction flag set, starting extraction. Data will be
                RN=self.readNoise/self.gain;
                for order=1:self.numOfOrders
                    spectra=zeros(self.numOfFibers,self.imdim(2));
                    specVar=zeros(self.numOfFibers,self.imdim(2));
                    %background=zeros(size(self.imdata));

                    %disp(['Extracting Order: ' num2str(order)])
                    orderSpecCenters=shiftdim(self.specCenters(order,:,:),1); % cl

                    % split into apetures
                    for col=1:self.imdim(2)

                        orderCenter=mean(orderSpecCenters(:,col));
                        profileApeture{col}=max(round(orderCenter-self.meanOrderWi
                            min(round(orderCenter+self.meanOrderWidth/2),self.imdi

                        orderProfile{col}=self.imdata(profileApeture{col},col)';
                        varProfile{col}=self.imvariance(profileApeture{col},col)';

                    end

                    % do extraction
                    %                     for col=1:self.imdim(2)
                    %                         [spectra(:,col), specVar(:,col), bac
                    %                             profileApeture(col,:),orderProfi
                    %                             squeeze(self.specCenters(order,:
                    %                             squeeze(2*log(2)*self.specWidth(
                    %                             self.readNoise/self.gain);
                    %                     end

                    [spectra, specVar, background]=self.MPDoptimalExt(...
                        profileApeture,orderProfile,varProfile,...
                        (shiftdim(self.specCenters(order,:,:),1))',...
                        2*log(2)*(shiftdim(self.specWidth(order,:,:),1))',...
                        RN);



                    % unfold into final variables
                    for col=1:self.imdim(2)
                        backgroundValues(profileApeture{col},col)=background{col};
                    end
                    spectraValues(:,:,order)=spectra;
                    spectraVar(:,:,order)=specVar;
                end

                spectra1DHDR=createcards('NUMORDER',self.numOfOrders,'number of or
                spectra1DHDR.addcard('NUMFIBER',self.numOfFibers,'number of fibers
                spectra1DHDR.addcard('TRACE',self.spectraTracePath,' ')

                fitswrite(spectraValues,self.SpectraFitsSaveFileName,spectra1DHDR.
                fitswrite(spectraVar,self.SpectraFitsSaveFileName,'writemode','app
                %fitswrite(backgroundValues,self.SpectraFitsSaveFileName,'writemod
            else
                disp(['Pre-existing extraction data found at: ' self.SpectraFitsSa

                spectraValues=fitsread(self.SpectraFitsSaveFileName);
                spectraVar=fitsread(self.SpectraFitsSaveFileName,'image',1);
                %backgroundValues=fitsread(self.SpectraFitsSaveFileName,'image',2)
            end
            self.spectraValues=spectraValues;
            self.spectraVar=spectraVar;
            %self.backgroundValues=backgroundValues;
        end

        function [spectraValues, spectraErrors, background, chi2]=MPDoptimalExtBac
            % Multi-Profile Deconvolution Optimal Extraction as described by Sharp
            %
            % paper: Sharp R., Birchall M. N. (2010) Optimal Extraction of Fibre O
            %        http://dx.doi.org/10.1071/AS08001


            if iscolumn(orderProfile); orderProfile=orderProfile'; disp(1); end
            if iscolumn(varProfile); varProfile=varProfile'; disp(2); end
            if iscolumn(dataRows); dataRows=dataRows'; disp(3); end
            if isrow(specCenters);specCenters=specCenters'; disp(4); end
            if isrow(specWidth);specWidth=specWidth'; disp(5); end
            %             error(' ')
            %             save('testing.mat','self','dataRows','orderProfile','varP

            phi=self.getPhi(dataRows,specCenters,2*log(2)*specWidth,[ones(length(s
            [xout,~,~,~] = fminsearch(@optimizeBackgroundFit, polyfit(dataRows,ord

            [chi2, fittedValues, fittedErrors, M]=optimizeBackgroundFit(xout);

            spectraValues=fittedValues(1:end-1);
            spectraErrors=fittedErrors(1:end-1);
            background=fittedValues(end)*polyval(xout,dataRows)/sum(polyval(xout,d

            function [chi2, fittedValues, fittedErrors, M]=optimizeBackgroundFit(x
                %setup
                phifit=[phi; polyval(x,dataRows)/sum(polyval(x,dataRows))];%ones(1
                sigmaweightedPhi=bsxfun(@rdivide,phifit,sqrt(varProfile))';
                c=phifit*sigmaweightedPhi;
                b=((orderProfile)*sigmaweightedPhi)';

                %setup error
                ce=phifit*phifit';
                be=((varProfile-RN^2)*phifit')';

                %solve
                fittedValues=c\b;
                fittedErrors=ce\be;

                %Model
                M=sum(bsxfun(@times,phifit,fittedValues),1);

                chi2=sum(((orderProfile-M)).^2./varProfile)/(size(M,2)-size(fitted
                %                 if chi2>1
                %                     plot(1:195,M,1:195,orderProfile)
                %                     drawnow;
                %                 end
            end
        end

        function [spectraValues, spectraErrors, background]=MPDoptimalExt(self,dat
            % Multi-Profile Deconvolution Optimal Extraction as described by Sharp
            %
            % paper: Sharp R., Birchall M. N. (2010) Optimal Extraction of Fibre O
            %        http://dx.doi.org/10.1071/AS08001

            %setup
            %             if 0
            %                 phi=self.getPhi(dataRows,specCenters,specWidth,ones(
            %             %%phi
            %             else
            %             phi1=;
            %             phi2=;
            %             phi3=
            spectraValues=zeros(size(specCenters'));
            spectraErrors=spectraValues;
            for col=1:size(specCenters,1)
                phi=bsxfun(@times, exp(-(bsxfun(@rdivide, bsxfun(@minus,repmat(dat
                    [size(specCenters,2),1]),specCenters(col,:)'), specWidth(col,:
                %phi=bsxfun(@times, phi4, specPeaks);
                %phi=sparse(phi);
                phi(phi<1e-6)=0;
                %             end

                %            if 1
                sigmaweightedPhi=bsxfun(@rdivide,phi,sqrt(varProfile{col}))';
                c=phi*sigmaweightedPhi;
                b=((orderProfile{col})*sigmaweightedPhi)';
                %             else
                %                 sigmaweightedPhi=bsxfun(@rdivide,phi,sqrt(varPro
                %                 c=mtimesx(phi,sigmaweightedPhi,'MATLAB');
                %                 b=mtimesx(orderProfile,sigmaweightedPhi,'MATLAB'
                %             end

                %setup error
                ce=phi*phi';
                be=((varProfile{col}-RN^2)*phi')';

                %solve
                spectraValues(:,col)=(c\b);
                %spectraValues=linsolve(c,b);
                spectraErrors(:,col)=(ce\be);
                %spectraErrors=linsolve(ce,be);

            end
            background=cellfun(@(x) zeros(size(x)),orderProfile,'UniformOutput',fa

        end

        function [spectraValues, spectraErrors, background]=MPDoptimalExtOld(self,
            % Multi-Profile Deconvolution Optimal Extraction as described by Sharp
            %
            % paper: Sharp R., Birchall M. N. (2010) Optimal Extraction of Fibre O
            %        http://dx.doi.org/10.1071/AS08001

            %setup
            %             if 0
            %                 phi=self.getPhi(dataRows,specCenters,specWidth,ones(
            %             %%phi
            %             else
            phi1=bsxfun(@minus,repmat(dataRows,[length(specCenters),1]),specCenter
            phi2=bsxfun(@rdivide, phi1, specWidth);
            phi3=exp(-(phi2).^2);
            phi=bsxfun(@times, phi3, 1./(specWidth*sqrt(pi)));
            %phi=bsxfun(@times, phi4, specPeaks);
            %phi=sparse(phi);
            phi(phi<1e-8)=0;
            %             end

            %            if 1
            sigmaweightedPhi=bsxfun(@rdivide,phi,sqrt(varProfile))';
            c=phi*sigmaweightedPhi;
            b=((orderProfile)*sigmaweightedPhi)';
            %             else
```

```
            %                 sigmaweightedPhi=bsxfun(@rdivide,phi,sqrt(varProfile
            %                 c=mtimesx(phi,sigmaweightedPhi,'MATLAB');
            %                 b=mtimesx(orderProfile,sigmaweightedPhi,'MATLAB')';
            %             end

            %setup error
            ce=phi*phi';
            be=((varProfile-RN^2)*phi')';

            %solve
            spectraValues=(c\b);
            %spectraValues=linsolve(c,b);
            spectraErrors=(ce\be);
            %spectraErrors=linsolve(ce,be);
            background=zeros(size(orderProfile));

        end

        function phi=getPhi(~,dataRows,specCenters,specWidth,specPeaks)
            phi1=bsxfun(@minus,repmat(dataRows,[length(specCenters),1]),specCenter
            phi2=bsxfun(@rdivide, phi1, specWidth);
            phi3=exp(-(phi2).^2);
            phi4=bsxfun(@times, phi3, 1./(specWidth*sqrt(pi)));
            phi=bsxfun(@times, phi4, specPeaks);
            %phi=sparse(phi);
            phi(phi<1e-8)=0;
        end

        function lineariseAndCombineSpectrum(self,saveFiles)

            if nargin==1
                saveFiles=false;
            end

            spec=(self.spectraValues);%./self.P2PVariationValues;%./self.flatBlaze
            specVar=(self.spectraVar);%./self.P2PVariationValues;%./self.flatBlaze

            %             for or=1:self.numOfOrders
            %                 specVar(:,:,or) = bsxfun(@rdivide,specVar(:,:,or),ma
            %                 spec(:,:,or)    = bsxfun(@rdivide,spec(:,:,or),max(s
            %             end

            longwavelinear=linspace(min(self.wavefit(:)),max(self.wavefit(:)),self

            speclinearlong=zeros(self.numOfFibers,self.imdim(2)*self.numOfOrders,s
            spectraVarlinearlong=zeros(self.numOfFibers,self.imdim(2)*self.numOfOr

            for o=1:self.numOfOrders
                for f=1:size(spec,1);
                    speclinearlong(f,:,o)=interp1(self.wavefit(f,:,o),spec(f,:,o),
                    spectraVarlinearlong(f,:,o)=interp1(self.wavefit(f,:,o),specVa
                    specflatlong(f,:,o)=interp1(self.wavefit(f,:,o),self.flatBlaze
                end
            end
```

```
%specflatlong=ones(size(speclinearlong));

finalspeclong=nansum(speclinearlong,3)';%./nansum(specflatlong,3)';
finalspecVarlong=nansum(spectraVarlinearlong,3)';%./nansum(specflatlon
flatspeclong=nansum(specflatlong,3)';

finalspeclong=finalspeclong./bsxfun(@rdivide,flatspeclong,mean(flatspe
finalspecVarlong=finalspecVarlong./bsxfun(@rdivide,flatspeclong,mean(f

toclip=isnan(sum(finalspeclong,2));

longwavelinear_clipped=longwavelinear(~toclip);
finalspecVarlong_clipped=finalspecVarlong(~toclip,:);
finalspeclong_clipped=finalspeclong(~toclip,:);

self.finalSpectra=squeeze(finalspeclong_clipped');
self.finalSpectraVar=squeeze(finalspecVarlong_clipped');
self.finalWave=longwavelinear_clipped;

self.finalSpec=squeeze(sum(finalspeclong_clipped,2)');
self.finalSpecVar=squeeze(sum(finalspecVarlong_clipped,2)');

if 0

    for i=1:size(finalspecVarlong_clipped,2)
        smoother(:,i)=csaps(self.finalWave,finalspecVarlong_clipped(:,
    end

    smoother=mean(smoother,2)';
    %[smoother] = blazeCorrection(self.finalSpec,self.finalWave,0.98)'

else
    smoother=1;
end
%            error(' ')
self.finalSpec=self.finalSpec./smoother;
self.finalSpecVar=self.finalSpecVar./smoother;

if saveFiles
    header=self.targetHeader;
    header.IMAGETYP='SPECTRUM';
    header.CRPIX1=round(length(self.finalWave)/2);
    header.CRVAL1=self.finalWave(header.CRPIX1);
    header.CTYPE1='Wavelength';
    header.CUNIT1='nm';
    header.CDELT1=mean(diff(self.finalWave));
    header.UTC=round((header.JD-floor(header.JD))*24*60*60);
    header.MJD=header.JD-2400000.5;
    header.DLAT=-33.873651000000000000;
    header.DLONG=151.206889600000070000;%sydney
    header.GEOELV=100;

    headercell1=fitstructure2cell(header);
```

```
                header2.EXTNAME='FLUXERROR';
                headercell2=fitstructure2cell(header2);

                fitswrite(finalspeclong_clipped,[self.targetBaseFilename '-IndivCa
                fitswrite(finalspecVarlong_clipped,[self.targetBaseFilename '-Indi

                fitswrite(self.finalSpec,[self.targetBaseFilename '-CombCalSpec.fi
                fitswrite(self.finalSpecVar,[self.targetBaseFilename '-CombCalSpec
            end
        end
    end

    methods (Static, Access = private)
        function answer=checkForReducedFitsAt(path)
            % check for a reduced target
            try
                import matlab.io.*
                fptr = fits.openFile(path);
                fits.closeFile(fptr);
                answer=1;
            catch err
                if strcmp(err.identifier,'MATLAB:imagesci:fits:libraryError')
                    error('MISPRINT:checkForReducedTarget:fitsOpenError','Reduced
                else
                    rethrow(err)
                end
            end
        end
    end
    methods (Static)
        [specCenters, p, mu]=polyfitwork(imdim,means,column,polyorder,offset,plota
        prepareFrames
        [peaks,means,widths,xfitted] = fitNGaussainsAlt(N,x,y,peakcut,plotting)
        out=nGausFunc(x,xData,N)
        wavecalGUI
        autoimprovewavelength(varargin)
    end
end
```

Published with MATLAB® R2014b