"import pyfits as pf\n",

"import pylab as plt\n",

"from scipy import optimize\n",

"from scipy.signal import medfilt, find\_peaks\_cwt\n",

"from scipy.ndimage.filters import minimum\_filter, maximum\_filter, median\_filter, convolve\n",

"from scipy.ndimage.measurements import label\n",

"import numpy as np"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 1

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"cd /Users/Carlos/Documents/HERMES/reductions/duncan/"

],

"language": "python",

"metadata": {},

"outputs": [

{

"output\_type": "stream",

"stream": "stdout",

"text": [

"/Users/Carlos/Documents/HERMES/reductions/duncan\n"

]

}

],

"prompt\_number": 2

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"def openFile(fileName):\n",

" thisFile = pf.open(fileName)\n",

"\n",

" print thisFile[0].header['OBJECT']\n",

" \n",

" gain0\_2000 = thisFile[0].header['RO\_GAIN']\n",

" gain2000\_4000 = thisFile[0].header['RO\_GAIN1']\n",

"\n",

" thisData = thisFile[0].data\n",

"\n",

" bias0\_2000 = np.median(thisData[3:2052,4099:-3])\n",

" bias2000\_4000 = np.median(thisData[2059:-3,4099:-3])\n",

"\n",

" thisData = thisData[:,:4095]\n",

"\n",

" thisData[:2055] -= bias0\_2000\n",

" thisData[2055:] -= bias2000\_4000\n",

" \n",

" return thisData\n",

"\n"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 3

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"def fithOrder(thisPoly, thisRange):\n",

" result = thisPoly[0]\*thisRange\*\*5\n",

" result += thisPoly[1]\*thisRange\*\*4\n",

" result += thisPoly[2]\*thisRange\*\*3\n",

" result += thisPoly[3]\*thisRange\*\*2\n",

" result += thisPoly[4]\*thisRange\*\*1\n",

" result += thisPoly[5]\*thisRange\*\*0\n",

" \n",

" return result"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 4

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"def find\_vertical\_shift(flat, arc):\n",

" CCTotal = 0\n",

" for column in range(flat.shape[1]):\n",

" thisFlatCol = flat[:,column]\n",

" thisArcCol = arc[:,column]\n",

" CCCurve = np.correlate(thisFlatCol, thisArcCol, mode='full')\n",

" CCTotal += CCCurve\n",

"\n",

" y = CCTotal[int(CCTotal.shape[0]/2.)+1-5:int(CCTotal.shape[0]/2.)+1+4]\n",

" y /=np.max(y)\n",

" x = np.arange(-4,5)\n",

" x\_dense = np.linspace(-4,4)\n",

" p,\_ = fit\_gaussian([1,3.],y,x )\n",

" shift = p[0]\n",

" return shift\n",

"################\n",

"##Need to SUBTRACT the result of the gaussian fit to make the 1st curve be like the second (i.e the traces be like the arc)"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 5

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"def sum\_extract(fibre, tramlines, image, numPx):\n",

" \n",

" flux = np.ones(tramlines.shape[1])\*np.nan\n",

"# flux1 = np.ones(tramlines.shape[1])\*np.nan\n",

"# flux2 = np.ones(tramlines.shape[1])\*np.nan\n",

" \n",

" for i,thisCentroid in enumerate(tramlines[fibre]):\n",

"# print thisCentroid\n",

" try:\n",

" fullPx = image[ int(thisCentroid)-numPx : int(thisCentroid)+numPx+1 , i]\n",

" flux[i] = np.sum(fullPx) - fullPx[0]\*(thisCentroid%1) - fullPx[-1]\*(1-thisCentroid%1)\n",

"# flux1[i] = fullPx[0]\*(thisCentroid%1)\n",

"# flux2[i] = fullPx[-1]\*(1-thisCentroid%1)\n",

" except:\n",

" print fibre, 'falied'\n",

" print thisCentroid, 'centroid found in index',i\n",

" break\n",

"# print fibre\n",

" return flux"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 6

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"def gaussian(x, mu, sig, ):\n",

" x = np.array(x)\n",

" return np.exp(-np.power(x - mu, 2.) / 2 / np.power(sig, 2.))\n",

"\n",

"\n",

"def flexi\_gaussian(x, mu, sig, power, a, d ):\n",

" x = np.array(x)\n",

" return a\* np.exp(-np.power(np.abs((x - mu) \* np.sqrt(2\*np.log(2))/sig),power))+d\n",

"\n",

"def fit\_gaussian(p, flux, x\_range):\n",

" a = optimize.leastsq(diff\_gaussian, p, args= [flux, x\_range])\n",

" return a\n",

"\n",

"def fit\_flexi\_gaussian(p, flux, x\_range):\n",

" a = optimize.leastsq(diff\_flexi\_gaussian, p, args= [flux, x\_range])\n",

" return a\n",

"\n",

"def diff\_gaussian(p, args):\n",

" \n",

" flux = args[0]\n",

" x\_range = args[1]\n",

"\n",

" diff = gaussian(x\_range, p[0],p[1]) - flux\n",

" return diff\n",

"\n",

"def diff\_flexi\_gaussian(p, args):\n",

" \n",

" flux = args[0]\n",

" x\_range = args[1]\n",

" weights = np.abs(np.gradient(flux)) \* (flux+np.max(flux)\*.1)\n",

" diff = (flexi\_gaussian(x\_range, p[0], p[1], p[2], p[3], p[4]) - flux)# \*weights\n",

" return diff\n"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 7

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"# flatFileName = '0\_20aug/1/20aug10034.fits'\n",

"# arcFileName = '0\_20aug/1/20aug10052.fits'\n",

"# objFileName = '0\_20aug/1/20aug10053.fits'\n",

"\n",

"# flatFileName = '1\_21aug/1/21aug10047.fits'\n",

"# arcFileName = '1\_21aug/1/21aug10046.fits'\n",

"# objFileName = '1\_21aug/1/21aug10041.fits'\n",

"\n",

"# flatFileName = '2\_22aug/1/22aug10032.fits'\n",

"# arcFileName = '2\_22aug/1/22aug10031.fits'\n",

"# objFileName = '2\_22aug/1/22aug10036.fits'\n",

"\n",

"# flatFileName = '3\_24aug/1/24aug10053.fits'\n",

"# arcFileName = '3\_24aug/1/24aug10054.fits'\n",

"# objFileName = '3\_24aug/1/24aug10058.fits'\n",

"\n",

"flatFileName = '4\_25aug/1/25aug10039.fits'\n",

"arcFileName = '4\_25aug/1/25aug10043.fits'\n",

"objFileName = '4\_25aug/1/25aug10044.fits'"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 8

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"flat = openFile(flatFileName)\n",

"arc = openFile(arcFileName)\n",

"obj = openFile(objFileName)\n"

],

"language": "python",

"metadata": {},

"outputs": [

{

"output\_type": "stream",

"stream": "stdout",

"text": [

"Fibre Flat Field - Quartz\_75\_H\n",

"ARC - ThXe\_2 ThXe\_1"

]

},

{

"output\_type": "stream",

"stream": "stdout",

"text": [

"\n",

"HD1581 #176 (Pivot 175)"

]

},

{

"output\_type": "stream",

"stream": "stdout",

"text": [

"\n"

]

}

],

"prompt\_number": 9

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"flat\_mf = medfilt(flat, [3,9])\n",

"flat\_1d = np.sum(flat\_mf,axis =0)\n",

"flat\_per = np.percentile(flat\_1d, 90)\n",

"flat\_1d\_norm = flat\_1d/flat\_per\n",

"flat\_flat = flat\_mf / flat\_1d\_norm[None,:]"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 10

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"#Convert flat\_flat to binary for tracing\n",

"for i in range(flat\_flat.shape[1]):\n",

" singleCol = flat\_flat[:,i]\n",

" singleMin = singleCol - convolve(minimum\_filter(singleCol,15),[.2,.2,.2,.2,.2])\n",

" singleMax = convolve(maximum\_filter(singleMin,15),[.2,.2,.2,.2,.2])\n",

"\n",

" fixer = convolve(singleMax, np.ones(200)/200)\n",

" singleMax[singleMax<fixer\*.5] = fixer[singleMax<fixer\*.5]\*.5\n",

" singleColFlat = singleMin/singleMax\n",

"\n",

" singleColFlat[singleColFlat>.3] = 1\n",

" singleColFlat[singleColFlat<.3] = 0\n",

" \n",

" flat\_flat[:,i] = singleColFlat\n",

" "

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 11

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"out\_array, n = label(flat\_flat, np.ones((3,3)))\n",

"print n,'fibres'\n",

"# n-=2 # fibres 252 and 253 are not good for HD1581 epoch 0 "

],

"language": "python",

"metadata": {},

"outputs": [

{

"output\_type": "stream",

"stream": "stdout",

"text": [

"439 fibres\n"

]

}

],

"prompt\_number": 12

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"#create centroid array\n",

"fibre\_centroids = np.ones((n,out\_array.shape[1]))\*np.nan\n",

"for col in range(out\_array.shape[1]):\n",

" testCol = out\_array[:,col]\n",

" for fibre in range(n):\n",

" fibre\_centroids[fibre,col] = np.average(np.where(testCol==fibre+1)[0])\n",

" if np.sum(np.isnan(fibre\_centroids[fibre,col]))>0:\n",

" print 'Found nan',np.where(testCol==fibre+1)[0], fibre+1"

]

}

{

"cell\_type": "code",

"collapsed": false,

"input": [

"#line to remove 251 and 252 that have nans\n",

"# fibre\_centroids = np.delete(fibre\_centroids,251,0) #2 times for epoch0\n",

"# fibre\_centroids = np.delete(fibre\_centroids,372,0) 3 times for epoch1\n",

"np.sum(np.isnan(fibre\_centroids))"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 14

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"#create polynomials\n",

"fibrePolys = np.ones((fibre\_centroids.shape[0],6))\*np.nan\n",

"for y,fibre in enumerate(fibre\_centroids):\n",

" fibrePolys[y,:] = np.polyfit(range(fibre.shape[0]),fibre,5)\n",

" if np.sum(np.isnan(fibrePolys[y,:]))>0:\n",

" print 'Found nan in fibre',y\n",

" print 'Fibre values',fibre\n",

" print"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 15

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"#create tramlines\n",

"tramlines = (np.ones(fibre\_centroids.shape)\*np.nan)[:-1]\n",

"thisRange = np.arange(fibre\_centroids.shape[1])\n",

"for i,thisPoly in enumerate(fibrePolys[1:]):\n",

" tramlines[i] = fithOrder(thisPoly,thisRange)"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 16

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"shift = find\_vertical\_shift(flat, arc) #result to be subtracted to the tramlines (1st array in the CC...)\n",

"tramlines\_shifted = tramlines - shift"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 17

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"# #gaussian fit results\n",

"# plt.plot(x,y)\n",

"# plt.plot(x\_dense,gaussian(x\_dense,p[0],p[1]))\n",

"# plt.show()"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 18

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"extracted\_arc = np.ones(tramlines\_shifted.shape)\*np.nan\n",

"for fibre in range(tramlines\_shifted.shape[0]):\n",

" extracted\_arc[fibre] = sum\_extract(fibre,tramlines\_shifted, arc, 4)\n"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 19

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"extracted\_obj = np.ones(tramlines\_shifted.shape)\*np.nan\n",

"for fibre in range(tramlines\_shifted.shape[0]):\n",

" extracted\_obj[fibre] = sum\_extract(fibre,tramlines\_shifted, obj, 4)\n"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 20

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"plt.plot(np.median(extracted\_obj,axis=1))\n",

"plt.show()"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 21

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"plt.plot(extracted\_arc[170])\n",

"plt.show()\n"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 343

},

FINAL LOOP STARTS HERE

{

"cell\_type": "code",

"collapsed": false,

"input": [

"#initial pixel adjustment\n",

"thisFibre = 200\n",

"\n",

"print thisFibre,\n",

"\n",

"halfCCRange = 10\n",

"halfSmallCCRange = 4\n",

"\n",

"masterArc = extracted\_arc[thisFibre].copy()\n",

"\n",

"lineTemplate = np.zeros(extracted\_arc.shape[1])\n",

"lineList = np.loadtxt('linelist\_blue.txt')\n",

"lineTemplate[lineList[:,0].astype(int)]=1\n",

"\n",

"CCCurve = np.correlate(masterArc, lineTemplate, mode='full')\n",

"\n",

"y = CCCurve[int(CCCurve.shape[0]/2.)+1-halfCCRange:int(CCCurve.shape[0]/2.)+1+halfCCRange+1]\n",

"x = np.arange(-halfCCRange,halfCCRange+1)\n",

"\n",

"maxIdx = np.where(y==np.max(y))[0][0]\n",

"print x[maxIdx]\n",

"\n",

"\n",

"plt.plot(masterArc)\n",

"adjLineList = lineList.copy()\n",

"adjLineList[:,0] += x[maxIdx]\n",

"np.savetxt('linelist\_blue\_v2.txt',adjLineList)\n",

"plt.scatter(adjLineList[:,0], np.ones(adjLineList.shape[0]))\n",

"plt.show()"

],

"language": "python",

"metadata": {},

"outputs": []

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"#FULL LOOP\n",

"\n",

"wlSolutions = []\n",

"wlErrors = []\n",

"wlPolys = []\n",

"for thisFibre in range(extracted\_arc.shape[0])[:]:\n",

" print 'Fibre',thisFibre\n",

"\n",

" halfCCRange = 15\n",

"\n",

" masterArc = extracted\_arc[thisFibre].copy()\n",

"\n",

" lineTemplate = np.zeros(extracted\_arc.shape[1])\n",

" lineList = np.loadtxt('linelist\_blue\_v2.txt')\n",

" lineTemplate[lineList[:,0].astype(int)]=1\n",

"\n",

" CCCurve = np.correlate(masterArc, lineTemplate, mode='full')\n",

"\n",

" y = CCCurve[int(CCCurve.shape[0]/2.)+1-halfCCRange:int(CCCurve.shape[0]/2.)+1+halfCCRange+1]\n",

" x = np.arange(-halfCCRange,halfCCRange+1)\n",

"\n",

" maxIdx = np.where(y==np.max(y))[0][0]\n",

" thisShift = x[maxIdx]\n",

"\n",

"# print 'Shift', thisShift\n",

"\n",

" adjLineList = lineList.copy()\n",

" adjLineList[:,0] += thisShift\n",

"\n",

" for i, thisLineWl in enumerate(adjLineList):\n",

"# print 'Searching for wl',thisLineWl[1],'in px',thisLineWl[0],\n",

" \n",

" firstSliceX = np.arange(thisLineWl[0]-5,thisLineWl[0]+6).astype(int)\n",

" firstSliceY = masterArc[firstSliceX]\n",

" maxIdx = firstSliceX[np.where(firstSliceY==np.max(firstSliceY))[0][0]]\n",

" \n",

" secondSliceX = np.arange(maxIdx-5,maxIdx+6).astype(int)\n",

" secondSliceY = masterArc[secondSliceX] \n",

" \n",

"# p,\_ = fit\_gaussian([maxIdx,1.], secondSliceY, secondSliceX )\n",

" p,\_ = fit\_flexi\_gaussian([maxIdx,1., 2., np.max(secondSliceY), 0], secondSliceY, secondSliceX )\n",

" \n",

"# print 'Found',p\n",

" goodPxValue = p[0]\n",

"\n",

" adjLineList[i,0] = goodPxValue\n",

" \n",

"# x\_dense = np.linspace(np.min(secondSliceX),np.max(secondSliceX))\n",

"# plt.plot(secondSliceX,secondSliceY)\n",

"# plt.plot(x\_dense,flexi\_gaussian(x\_dense,p[0],p[1],p[2],p[3],p[4]))\n",

"# plt.title(goodPxValue)\n",

"# plt.show()\n",

"\n",

"# print adjLineList\n",

" \n",

" a = np.polyfit(adjLineList[:,0], adjLineList[:,1], 5)\n",

" x = fithOrder(a, np.arange(4095))\n",

" err = fithOrder(a, adjLineList[:,0]) - adjLineList[:,1]\n",

"\n",

" wlPolys.append(a)\n",

" wlErrors.append(err)\n",

" wlSolutions.append(x)\n",

" \n",

"wlPolys = np.array(wlPolys)\n",

"wlErrors = np.array(wlErrors)\n",

"wlSolutions = np.array(wlSolutions)\n",

"# plt.plot(x\_dense,gaussian(x\_dense,p[0],p[1]))\n",

"# plt.plot(firstSliceX,firstSliceY/np.max(firstSliceY))\n",

"# plt.title(maxIdx)\n",

"# # plt.plot(masterArc[thisLineWl[0]-5:thisLineWl[0]+5])\n",

"# plt.show()\n"

],

"language": "python",

"metadata": {},

"outputs": []

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"import scipy.constants as const\n",

"\n",

"wlErrorsRVs = wlErrors/np.tile(adjLineList[:,1],[wlErrors.shape[0],1])\*const.c\n",

"\n",

"stdRVs = np.std(wlErrorsRVs,axis = 1)\n",

"# stdRVs2 = np.std(wlErrorsRVs\_General\_distrib,axis = 1)\n",

"\n",

"plt.plot(stdRVs, '.')\n",

"# plt.plot(stdRVs2, '.')\n",

"plt.grid()\n",

"plt.show()\n",

"#237 !!!!\n"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 278

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"plt.plot(wlSolutions[170],extracted\_arc[170])\n",

"# plt.plot(wlSolutions[20],extracted\_arc[20])\n",

"plt.show()"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 431

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"thisArray = extracted\_arc[237].copy()\n",

"thisArray[thisArray<20]=0\n",

"thisPeaks = find\_peaks\_cwt(thisArray, np.arange(1,2))\n",

"print len(thisPeaks)"

],

"language": "python",

"metadata": {},

"outputs": []

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"plt.scatter(thisPeaks,extracted\_arc[237][thisPeaks])\n",

"plt.plot(extracted\_arc[237])\n",

"plt.show()"

],

"language": "python",

"metadata": {},

"outputs": []

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"#extend line list to all bumps\n",

"\n",

"masterArc = extracted\_arc[237]\n",

"bigLineList = []\n",

"for i, thisPeak in enumerate(thisPeaks):\n",

"# print 'Searching for wl',thisLineWl[1],'in px',thisLineWl[0],\n",

"\n",

" firstSliceX = np.arange(thisPeak-5,thisPeak+6).astype(int)\n",

" firstSliceY = masterArc[firstSliceX]\n",

" maxIdx = firstSliceX[np.where(firstSliceY==np.max(firstSliceY))[0][0]]\n",

"\n",

" secondSliceX = np.arange(maxIdx-5,maxIdx+6).astype(int)\n",

" secondSliceY = masterArc[secondSliceX] \n",

"\n",

"# p,\_ = fit\_gaussian([maxIdx,1.], secondSliceY, secondSliceX )\n",

" p,\_ = fit\_flexi\_gaussian([maxIdx,1., 2., np.max(secondSliceY), 0], secondSliceY, secondSliceX )\n",

"\n",

"# print 'Found',p\n",

" goodPxValue = p[0]\n",

"\n",

"# adjLineList[i,0] = goodPxValue\n",

"\n",

"# x\_dense = np.linspace(np.min(secondSliceX),np.max(secondSliceX))\n",

"# plt.plot(secondSliceX,secondSliceY)\n",

"# plt.plot(x\_dense,flexi\_gaussian(x\_dense,p[0],p[1],p[2],p[3],p[4]))\n",

"# plt.title(goodPxValue)\n",

"# plt.show()\n",

"\n",

"# print adjLineList\n",

"\n",

"# a = np.polyfit(adjLineList[:,0], adjLineList[:,1], 5)\n",

" x = fithOrder(wlPolys[237], goodPxValue)\n",

"# err = fithOrder(a, adjLineList[:,0]) - adjLineList[:,1]\n",

"\n",

" bigLineList.append((goodPxValue,x))\n",

"# wlErrors.append(err)\n",

"# wlSolutions.append(x)\n",

"\n",

"bigLineList = np.array(bigLineList)\n",

"# wlPolys = np.array(wlPolys)\n",

"# wlErrors = np.array(wlErrors)\n",

"# wlSolutions = np.array(wlSolutions)\n",

"# plt.plot(x\_dense,gaussian(x\_dense,p[0],p[1]))\n",

"# plt.plot(firstSliceX,firstSliceY/np.max(firstSliceY))\n",

"# plt.title(maxIdx)\n",

"# # plt.plot(masterArc[thisLineWl[0]-5:thisLineWl[0]+5])\n",

"# plt.show()\n"

],

"language": "python",

"metadata": {},

"outputs": []

},

FINAL LOOP ENDS HERE

{

"cell\_type": "code",

"collapsed": false,

"input": [

"plt.plot(range(wlSolutions[237].shape[0]), wlSolutions[237])\n",

"plt.scatter(bigLineList[:,0], bigLineList[:,1])\n",

"plt.show()"

],

"language": "python",

"metadata": {},

"outputs": []

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"plt.plot(wlSolutions[40], extracted\_arc[40])\n",

"plt.scatter(bigLineList[:,1],np.ones(110)\*20)\n",

"plt.show()"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 168

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"np.savetxt('bigLineList.txt',bigLineList)\n"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 247

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"# \_,ind = np.unique(bigLineList[:,1], return\_index=True)\n",

"# bigLineList = bigLineList[ind]\n",

"# print bigLineList"

],

"language": "python",

"metadata": {},

"outputs": [

{

"output\_type": "stream",

"stream": "stdout",

"text": [

"[[ 28.23434534 4715.17966512]\n",

" [ 131.32807708 4720.97392104]\n",

" [ 139.99028209 4721.45864704]\n",

" [ 152.80316031 4722.17502645]\n",

" [ 176.88728971 4723.51960082]\n",

" [ 178.48700831 4723.60881781]\n",

" [ 262.67588804 4728.28760755]\n",

" [ 278.68489432 4729.17361003]\n",

" [ 292.2892508 4729.92559001]\n",

" [ 314.38271449 4731.14495604]\n",

" [ 340.52750877 4732.58494586]\n",

" [ 369.6425329 4734.18470667]\n",

" [ 470.51903855 4739.69591091]\n",

" [ 486.19602496 4740.54794122]\n",

" [ 493.8278705 4740.96228751]\n",

" [ 515.89961069 4742.1589824 ]\n",

" [ 544.53196614 4743.70778427]\n",

" [ 569.7617685 4745.06914616]\n",

" [ 613.70131137 4747.43243428]\n",

" [ 641.5083855 4748.92300341]\n",

" [ 643.67308801 4749.03887559]\n",

" [ 660.84993082 4749.95747317]\n",

" [ 706.99651408 4752.41788554]\n",

" [ 766.09760158 4755.5530301 ]\n",

" [ 773.40768179 4755.93955591]\n",

" [ 815.07815593 4758.13761774]\n",

" [ 871.74408092 4761.11214867]\n",

" [ 886.32597315 4761.87486766]\n",

" [ 933.56225473 4764.33792615]\n",

" [ 958.33088614 4765.62474256]\n",

" [ 960.96623332 4765.76146708]\n",

" [ 977.10727737 4766.5980787 ]\n",

" [ 1024.50765533 4769.04691397]\n",

" [ 1105.71164826 4773.21432807]\n",

" [ 1144.53064436 4775.19407299]\n",

" [ 1150.99355487 4775.52289278]\n",

" [ 1205.61141166 4778.29277664]\n",

" [ 1222.70991089 4779.15660935]\n",

" [ 1230.67037224 4779.55824223]\n",

" [ 1242.26739088 4780.14274078]\n",

" [ 1261.49628802 4781.1102934 ]\n",

" [ 1369.75775852 4786.52046964]\n",

" [ 1382.78287679 4787.16710291]\n",

" [ 1394.85630476 4787.76566859]\n",

" [ 1427.731808 4789.39153649]\n",

" [ 1493.0207728 4792.60304179]\n",

" [ 1505.88866601 4793.23327306]\n",

" [ 1531.66207997 4794.49287523]\n",

" [ 1560.81866069 4795.91347213]\n",

" [ 1570.11244057 4796.36532298]\n",

" [ 1630.61412271 4799.29536581]\n",

" [ 1648.4031109 4800.15308909]\n",

" [ 1792.20855974 4807.02381376]\n",

" [ 1815.69583738 4808.13532823]\n",

" [ 1847.08556181 4809.61615036]\n",

" [ 1906.3377252 4812.39684707]\n",

" [ 1937.13424657 4813.83461867]\n",

" [ 2006.46737274 4817.05274902]\n",

" [ 2026.76623866 4817.99001648]\n",

" [ 2041.41193896 4818.66487696]\n",

" [ 2132.94503086 4822.85643697]\n",

" [ 2141.54422336 4823.24789931]\n",

" [ 2149.270197 4823.5992706 ]\n",

" [ 2158.02020633 4823.99682639]\n",

" [ 2217.54136268 4826.69023927]\n",

" [ 2284.62851137 4829.70321466]\n",

" [ 2316.26853246 4831.11583403]\n",

" [ 2327.09007303 4831.59774787]\n",

" [ 2354.05964272 4832.79604983]\n",

" [ 2473.14405506 4838.04064378]\n",

" [ 2537.32206863 4840.83569282]\n",

" [ 2594.08150947 4843.28934329]\n",

" [ 2618.23443301 4844.32823955]\n",

" [ 2712.50680414 4848.35346114]\n",

" [ 2730.13308261 4849.1008112 ]\n",

" [ 2747.46445797 4849.834044 ]\n",

" [ 2761.58245804 4850.43014717]\n",

" [ 2819.49078084 4852.86409694]\n",

" [ 2840.69395695 4853.75082107]\n",

" [ 2951.01785759 4858.32592182]\n",

" [ 3021.49536212 4861.21459053]\n",

" [ 3049.81110191 4862.36768975]\n",

" [ 3069.43280435 4863.16422369]\n",

" [ 3126.66425041 4865.47571285]\n",

" [ 3212.24317753 4868.89925461]\n",

" [ 3226.72087795 4869.474522 ]\n",

" [ 3314.02333453 4872.91940685]\n",

" [ 3351.14740865 4874.37174572]\n",

" [ 3405.78128836 4876.49540522]\n",

" [ 3444.69047525 4877.99785897]\n",

" [ 3463.87826203 4878.73571668]\n",

" [ 3590.07396085 4883.5377 ]\n",

" [ 3604.28095426 4884.07274247]\n",

" [ 3690.56795645 4887.29793848]\n",

" [ 3763.78916089 4890.00159773]\n",

" [ 3766.24173955 4890.09162647]\n",

" [ 3900.16772362 4894.95480093]]\n"

]

}

],

"prompt\_number": 246

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"#find differences to other fibres\n",

"\n",

"# bigLineList = np.loadtxt('bigLineList.txt')\n",

"bigLineList = reducedbigLineList\n",

"# lineLocations = np.ones((extracted\_arc.shape[0],bigLineList.shape[0]))\*np.nan\n",

"lineLocations = np.ones(bigLineList.shape[0])\*np.nan\n",

"\n",

"for thisFibre in range(extracted\_arc.shape[0])[170:171]:\n",

" print 'Fibre',thisFibre,\n",

"\n",

" halfCCRange = 15\n",

"\n",

" thisArc = extracted\_arc[thisFibre].copy()\n",

" thisObj = extracted\_obj[thisFibre].copy()\n",

" thisWlSolution = wlSolutions[thisFibre].copy()\n",

" \n",

"\n",

" for i, thisWl in enumerate(bigLineList):\n",

"# print 'Searching for wl',thisWl\n",

" diffArray = np.abs(thisWlSolution-thisWl)\n",

"# print thisWl, thisWlSolution, diffArray \n",

" wlPx = np.where(diffArray==np.min(diffArray))[0][0]\n",

"# print wlPx, thisWlSolution[wlPx-1:wlPx+1]\n",

"\n",

" thisSlice = np.arange(wlPx-6,wlPx+7).astype(int)\n",

"\n",

" firstSliceX = thisWlSolution[thisSlice]\n",

" firstSliceY = thisArc[thisSlice]\n",

" maxIdx = thisSlice[np.where(firstSliceY==np.max(firstSliceY))[0][0]]\n",

"# print maxIdx\n",

" \n",

"# plt.plot(firstSliceX,firstSliceY)\n",

"# plt.show()\n",

"\n",

" \n",

" thisSecondSlice = np.arange(maxIdx-5,maxIdx+6).astype(int)\n",

" secondSliceX = thisWlSolution[thisSecondSlice]\n",

" secondSliceY = thisArc[thisSecondSlice] \n",

" \n",

"# # p,\_ = fit\_gaussian([maxIdx,1.], secondSliceY, secondSliceX )\n",

" p,\_ = fit\_flexi\_gaussian([thisWlSolution[maxIdx],.2, 2.8, np.max(secondSliceY), 0], secondSliceY, secondSliceX )\n",

" \n",

"# print 'Found',p[0]\n",

" goodWlValue = p[0]\n",

" \n",

"# lineLocations[thisFibre,i] = goodWlValue\n",

" lineLocations[i] = goodWlValue\n",

"\n",

"# adjLineList[i,0] = goodPxValue\n",

" \n",

"# x\_dense = np.linspace(np.min(secondSliceX),np.max(secondSliceX))\n",

"# plt.plot(secondSliceX,secondSliceY)\n",

"# plt.plot(x\_dense,flexi\_gaussian(x\_dense,p[0],p[1],p[2],p[3],p[4]))\n",

"# # # plt.title(goodPxValue)\n",

"# plt.show()\n",

"\n",

"# # print adjLineList\n",

" \n",

"# a = np.polyfit(adjLineList[:,0], adjLineList[:,1], 5)\n",

"# x = fithOrder(a, np.arange(4095))\n",

"# err = fithOrder(a, adjLineList[:,0]) - adjLineList[:,1]\n",

"\n",

"# wlPolys.append(a)\n",

"# wlErrors.append(err)\n",

"# wlSolutions.append(x)\n",

" \n",

"# wlPolys = np.array(wlPolys)\n",

"# wlErrors = np.array(wlErrors)\n",

"# wlSolutions = np.array(wlSolutions)\n",

"# # plt.plot(x\_dense,gaussian(x\_dense,p[0],p[1]))\n",

"# # plt.plot(firstSliceX,firstSliceY/np.max(firstSliceY))\n",

"# # plt.title(maxIdx)\n",

"# # # plt.plot(masterArc[thisLineWl[0]-5:thisLineWl[0]+5])\n",

"# # plt.show()\n"

],

"language": "python",

"metadata": {},

"outputs": [

{

"output\_type": "stream",

"stream": "stdout",

"text": [

" Fibre 170\n"

]

}

],

"prompt\_number": 359

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"# means = np.mean(lineLocations,axis=0)\n",

"# offsets = lineLocations - np.tile(bigLineList[:,1],[388,1])\n",

"offsets = lineLocations - bigLineList\n",

"\n",

"lineLocations.shape, offsets.shape"

],

"language": "python",

"metadata": {},

"outputs": [

{

"metadata": {},

"output\_type": "pyout",

"prompt\_number": 360,

"text": [

"((71,), (71,))"

]

}

],

"prompt\_number": 360

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"plt.plot(bigLineList,offsets.transpose(),'.')\n",

"# plt.plot(np.arange(bigLineList[:,1].shape[0])[filter\_all],offsets.transpose()[filter\_all],'.')\n",

"# plt.plot(bigLineList[:,1],offsets.transpose(),'.')\n",

"# plt.plot(offsets.transpose(),'.')\n",

"# plt.plot(bigLineList[:,1],std, color ='r')\n",

"plt.grid()\n",

"plt.show()"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 362

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"np"

],

"language": "python",

"metadata": {},

"outputs": []

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"std = np.nanstd(offsets, axis =0)"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 264

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"plt.plot(std,'.')\n",

"plt.show()"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 254

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"filter1 = np.abs(np.nanmedian(offsets,axis=0))<0.06\n",

"np.sum(filter1)"

],

"language": "python",

"metadata": {},

"outputs": [

{

"metadata": {},

"output\_type": "pyout",

"prompt\_number": 288,

"text": [

"85"

]

}

],

"prompt\_number": 288

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"np.sum(np.abs(offsets)<0.6,axis=0)"

],

"language": "python",

"metadata": {},

"outputs": [

{

"metadata": {},

"output\_type": "pyout",

"prompt\_number": 252,

"text": [

"array([388, 388, 387, 385, 386, 385, 388, 387, 383, 384, 387, 383, 387,\n",

" 388, 384, 384, 386, 383, 383, 386, 387, 383, 387, 384, 386, 387,\n",

" 385, 385, 387, 384, 383, 388, 387, 388, 388, 388, 387, 387, 386,\n",

" 386, 388, 385, 386, 387, 387, 387, 385, 387, 379, 387, 388, 385,\n",

" 388, 388, 388, 387, 388, 387, 388, 384, 384, 387, 385, 381, 388,\n",

" 385, 388, 387, 387, 387, 387, 385, 387, 387, 386, 386, 387, 387,\n",

" 388, 384, 387, 386, 386, 388, 384, 388, 386, 387, 388, 386, 388,\n",

" 387, 384, 387, 384, 383, 378])"

]

}

],

"prompt\_number": 252

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"offsets[np.abs(offsets)>0.6]=np.nan"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 263

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"filter2 = std<.12"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 289

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"filter\_all = filter1 & filter2 &filter3"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 307

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"filter1"

],

"language": "python",

"metadata": {},

"outputs": [

{

"metadata": {},

"output\_type": "pyout",

"prompt\_number": 292,

"text": [

"array([ True, True, True, True, True, False, True, True, False,\n",

" True, True, True, True, True, False, True, True, False,\n",

" True, True, False, True, True, False, True, True, True,\n",

" True, True, True, False, True, True, True, True, False,\n",

" True, True, False, True, True, True, True, True, True,\n",

" True, True, True, True, True, True, True, True, True,\n",

" True, True, True, True, True, True, False, True, False,\n",

" True, True, True, True, True, True, True, True, True,\n",

" True, True, True, True, True, True, True, True, True,\n",

" True, True, True, True, True, True, True, True, True,\n",

" True, True, True, True, False, True, True], dtype=bool)"

]

}

],

"prompt\_number": 292

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"bad = np.array([19 ,29, 46, 51, 55, 59, 69, 74, 75, 79, 85])"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 305

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"filter3 = np.ones(filter1.shape).astype(bool)\n",

"filter3[bad] = False"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 306

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"np.savetxt('reducedbigLineList.txt',bigLineList[:,1][filter\_all])"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 310

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"reducedbigLineList = bigLineList[:,1][filter\_all]"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 356

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"reducedbigLineList"

],

"language": "python",

"metadata": {},

"outputs": [

{

"metadata": {},

"output\_type": "pyout",

"prompt\_number": 357,

"text": [

"array([ 4715.17966512, 4720.97392104, 4721.45864704, 4722.17502645,\n",

" 4723.51960082, 4728.28760755, 4729.17361003, 4731.14495604,\n",

" 4732.58494586, 4734.18470667, 4739.69591091, 4740.54794122,\n",

" 4742.1589824 , 4743.70778427, 4747.43243428, 4749.95747317,\n",

" 4752.41788554, 4755.93955591, 4758.13761774, 4761.11214867,\n",

" 4761.87486766, 4764.33792615, 4766.5980787 , 4769.04691397,\n",

" 4773.21432807, 4775.19407299, 4778.29277664, 4779.15660935,\n",

" 4781.1102934 , 4786.52046964, 4787.16710291, 4787.76566859,\n",

" 4789.39153649, 4792.60304179, 4794.49287523, 4796.36532298,\n",

" 4799.29536581, 4807.02381376, 4808.13532823, 4809.61615036,\n",

" 4813.83461867, 4817.05274902, 4817.99001648, 4823.24789931,\n",

" 4826.69023927, 4829.70321466, 4831.11583403, 4831.59774787,\n",

" 4832.79604983, 4840.83569282, 4843.28934329, 4844.32823955,\n",

" 4848.35346114, 4850.43014717, 4852.86409694, 4853.75082107,\n",

" 4861.21459053, 4862.36768975, 4863.16422369, 4865.47571285,\n",

" 4868.89925461, 4872.91940685, 4874.37174572, 4876.49540522,\n",

" 4877.99785897, 4878.73571668, 4883.5377 , 4884.07274247,\n",

" 4887.29793848, 4890.09162647, 4894.95480093])"

]

}

],

"prompt\_number": 357

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"plt.plot(wlSolutions[170], extracted\_obj[170])\n",

"plt.show()\n"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 363

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"# np.savetxt('HD1581\_1.txt',np.vstack((wlSolutions[170], extracted\_obj[170])).transpose())\n",

"# np.savetxt('ThXe\_1.txt',np.vstack((wlSolutions[170], extracted\_arc[170])).transpose())"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 365

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"# np.savetxt('HD1581\_0.txt',np.vstack((wlSolutions[170], extracted\_obj[170])).transpose())\n",

"np.savetxt('ThXe\_0.txt',np.vstack((wlSolutions[170], extracted\_arc[170])).transpose())"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 383

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"np.savetxt('HD1581\_2.txt',np.vstack((wlSolutions[170], extracted\_obj[170])).transpose())\n",

"np.savetxt('ThXe\_2.txt',np.vstack((wlSolutions[170], extracted\_arc[170])).transpose())"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 408

},

{

"cell\_type": "code",

"collapsed": false,

"input": [

"np.savetxt('HD1581\_3.txt',np.vstack((wlSolutions[170], extracted\_obj[170])).transpose())\n",

"np.savetxt('ThXe\_3.txt',np.vstack((wlSolutions[170], extracted\_arc[170])).transpose())"

],

"language": "python",

"metadata": {},

"outputs": [],

"prompt\_number": 432

},

{

"cell\_type": "code",

"collapsed": false,

"input": [],

"language": "python",

"metadata": {},

"outputs": []

}

],

"metadata": {}

}

]

}