```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 2,
   "metadata": {},
   "outputs": [],
   "source": [
    "import numpy as np\n",
    "import pandas as pd\n",
    "import plotly.graph_objects as go\n",
    "import scipy\n",
    "from ipywidgets import widgets,Layout\n",
    "from IPython.display import display"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "metadata": {},
   "outputs": [],
   "source": [
    "def readexpdata(filename):\n",
    "    '''reads in experimental data into a pandas dataframe and
determines major peaks in a numpy array\n",
    "\n",
    "    arguments: \n",
    "    filename -- the name of the csv file containing experimental
data\n",
    "    '''\n",
    "    exp = pd.read_csv(filename)\n",
    "    exppeaks_ind = scipy.signal.find_peaks(exp.iloc[:,1],
distance=50)[0].tolist()\n",
    "    xpeaks = []\n",
    "    ypeaks = []\n",
    "    for x in exppeaks_ind:\n",
    "        xpeaks.append(float(exp.iloc[:,0][x]))\n",
    "        ypeaks.append(float(exp.iloc[:,1][x]))\n",
    "    exppeaks = np.array([xpeaks, ypeaks], dtype=object)\n",
    "    exppeaks = exppeaks[:, np.argsort(exppeaks[1])]\n",
    "    exppeaks = np.delete(exppeaks, np.s_[0:-500],1) #finds 500
tallest peaks\n",
    "    return exp, exppeaks\n",
    "\n",
    "def readpreddata(filename):\n",
    "    '''reads in predicted data from xiam into a numpy array\n",
    "\n",
    "    arguments:\n",
    "    filename -- the name of the .xo file \n",
    "    (note: in my convention, these files are usually named
```

```
molecule_#.xo)\n",
    "    '''\n",
    "    reading = False\n",
    "    qnum = []\n",
    "    positions = []\n",
    "    intensity = []\n",
    "    numLines = 0\n",
    "    symbols = []\n",
    "\n",
    "    with open(filename) as f:\n",
    "        lines = f.readlines()\n",
    "\n",
    "    for line in lines:\n",
    "        if '-- B' in line:\n",
    "            reading = True\n",
    "            continue\n",
    "\n",
    "        if line.strip() == '':\n",
    "            reading = False\n",
    "            continue\n",
    "        \n",
    "        if reading == True:\n",
    "            if 'rigid' in line:\n",
    "                continue\n",
    "            elif 'S 2' in line:\n",
    "                numLines += 1\n",
    "                qnum.append(qnum[-1])\n",
    "                symbols.append(line[22:25])\n",
    "                positions.append(float(line[39:49])*1000)\n",
    "                intensity.append(float(line[69:77])*-1000)\n",
    "            else:\n",
    "                numLines += 1\n",
    "                qnum.append(line[0:20])\n",
    "                symbols.append(line[22:25])\n",
    "                positions.append(float(line[39:49])*1000)\n",
    "                intensity.append(float(line[69:77])*-1000)\n",
    "            \n",
    "    data = np.array([positions, intensity, qnum, symbols],
dtype=object)\n",
    "    return data\n",
    "\n",
    "def readfitdata(fitfile, linelist):\n",
    "    '''reads in data from fit files into a list of quantum
numbers and a list of positions\n",
    "\n",
    "    arguments:\n",
    "    fitfile -- the .xo file containing the fits\n",
    "    linelist -- a list where the lines generated by the fitfile
are stored\n",
    "    (note: in my convention, these files are usually named
```

```
molecule_#s.xo)\n",
    "    '''\n",
    "    reading = False\n",
    "    qnum=[]\n",
    "    positions = []\n",
    "\n",
    "    with open(fitfile) as f:\n",
    "        lines = f.readlines()\n",
    "\n",
    "    for line in lines:\n",
    "        if '  End at Cycle ' in line:\n",
    "            reading = True\n",
    "            continue\n",
    "        \n",
    "        if 'Maximum' in line:\n",
    "            reading = False\n",
    "            continue\n",
    "        \n",
    "        if line.strip() == '':\n",
    "            continue\n",
    "\n",
    "        if reading==True:\n",
    "            if line[4] == ':':\n",
    "                linelist.append(line)\n",
    "                qnum.append(line[5:24])\n",
    "                positions.append(float(line[31:42])*1000)\n",
    "    \n",
    "    for i in range(len(qnum)):\n",
    "        qnum[i]=qnum[i][:9]+' '+qnum[i][9:]\n",
    "    return qnum, positions\n",
    "\n",
    "def transformlines(file,explines=[]):\n",
    "    '''transforms lines from line list into a xiam readable
format\n",
    "\n",
    "    arguments:\n",
    "    file -- the file where the transformed line list is
stored\n",
    "    '''\n",
    "    with open(file, mode='r') as f:\n",
    "        lines = f.readlines()\n",
    "\n",
    "    with open(file, mode='w') as f:\n",
    "        for line in lines[:-len(explines)]:\n",
    "            f.write(line[5:24]+'  '+line[26:29]+'  V 1  B 1 =
'+'{:0<12}'.format(str(round(float(line[53:])*1000,6)))+'  MHz   Err
0.004\\n')\n",
    "        i=0\n",
    "        for line in lines[-len(explines):]:\n",
    "            f.write(line[5:24]+'  '+line[26:29]+'  V 1  B 1 =
```

```
'+'{:0<12}'.format(str(round(explines[i],6)))+'  MHz   Err  0.004\\n')
\n",
    "            i+=1\n",
    "\n",
    "def add_intensity_to_fitlines(alllines, newlines):\n",
    "    '''adds intensities to the fitted lines\n",
    "\n",
    "    arguments:\n",
    "    all_lines -- a numpy array with all the data from the
predicted spectra\n",
    "    new_lines -- a numpy array with the data from the fit
files\n",
    "    '''\n",
    "    intensities=[]\n",
    "    for x in newlines[0]:\n",
    "        intensities.append(float(alllines[1]
[np.where(alllines==x)[1][0]]))\n",
    "    return np.vstack([newlines, intensities])"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 4,
   "metadata": {},
   "outputs": [],
   "source": [
    "class spectra:\n",
    "    def __init__(self, expfile, predfile, autoimport=False,
fitfile=''):\n",
    "        self.expfile = expfile\n",
    "        self.predfile = predfile\n",
    "        self.autoimport = autoimport\n",
    "        self.fitfile = fitfile\n",
    "\n",
    "        #read in data from input files\n",
    "        self.exp, self.exppeaks = readexpdata(expfile)\n",
    "        self.preddata = readpreddata(predfile)\n",
    "        self.p_data_pos = self.preddata[:,
np.argsort(self.preddata[0])]\n",
    "        self.p_data_int = self.preddata[:,
np.argsort(self.preddata[1])]\n",
    "        self.p_data_int = np.delete(self.p_data_int, np.s_[500:
self.p_data_int.shape[1]],1)\n",
    "\n",
    "        #combine all peaks\n",
    "        self.allpeaks = np.array([list(self.p_data_int[0]) +
list(self.exppeaks[0]), list(self.p_data_int[1])
+list(self.exppeaks[1])],dtype=object)\n",
    "\n",
    "        self.fitlines, self.predlines, self.explines = [],[],[]
```

```
\n",
    "        \n",
    "    def create_traces(self):\n",
    "        # creates traces for the experimental data, predicted
data, and fitted data\n",
    "        exptrace = go.Scatter(\n",
    "        x=self.exp.iloc[:,0],\n",
    "        y=self.exp.iloc[:,1],\n",
    "        name='experimental',\n",
    "        line=dict(color='blue', width=2),\n",
    "        mode='lines')\n",
    "\n",
    "        predtrace = go.Bar(\n",
    "        x=self.p_data_pos[0], \n",
    "        y=self.p_data_pos[1], \n",
    "        text = self.p_data_pos[2], \n",
    "        width=2, \n",
    "        name='predicted', \n",
    "        marker_color='black')\n",
    "\n",
    "        # the buttons on the plot will be generated by this
trace\n",
    "        peakscat = go.Scatter(\n",
    "        x=self.allpeaks[0], \n",
    "        y=self.allpeaks[1], \n",
    "        name='peaks', \n",
    "        mode='markers')\n",
    "\n",
    "        traces = [peakscat, exptrace, predtrace]\n",
    "\n",
    "        if self.autoimport == True:\n",
    "            newqnum,newpos =
readfitdata(self.fitfile,self.fitlines)\n",
    "            fitdata = np.array([newqnum, newpos], dtype=object)
\n",
    "            fitdata = add_intensity_to_fitlines(self.p_data_pos,
fitdata)\n",
    "            predtrace = go.Bar(x=fitdata[1], y=fitdata[2], text =
fitdata[0], width=2, name='fit',marker_color='red')\n",
    "            traces.append(predtrace)\n",
    "        \n",
    "        return traces\n",
    "    \n",
    "    def generate_lines(self):\n",
    "        # creates a linelist in the format that is seen in xiam
output files\n",
    "        lines=''\n",
    "        if len(self.predlines)==len(self.explines):\n",
    "            if self.fitlines !=[]:\n",
    "                for x in self.fitlines:\n",
```

```
    "                    lines+= x[:65]+'\\n'\n",
    "            for i in range(len(self.predlines)):\n",
    "                lines += '{:4}'.format(i+1+len(self.fitlines))
+':'+self.predlines[i][1][:9]+self.predlines[i][1][10:]+'
'+self.predlines[i][2]+'  '\\\n",
    "
'{:0<10}'.format(str(round(self.predlines[i][0]/1000,7)))+'{:
>10}'.format('{:0<6}'.format(str(round(self.explines[i]-
self.predlines[i][0],4))))+'  '\\\n",
    "
'{:0<10}'.format(str(round(self.explines[i]/1000,7)))+'\\n'\n",
    "        return lines\n",
    "    \n",
    "    def snr(self, numvals=5):\n",
    "        # calculates the SNR for the tallest peaks in the
spectra\n",
    "        if numvals != 0:\n",
    "            noise =
self.exp.sort_values(by=[list(self.exp.columns.values)[1]]).iloc[:,1]
[:-500]\n",
    "            avg_noise = sum(noise)/len(noise)\n",
    "            peaks_x, peaks_y, signal=[],[],[]\n",
    "            for x in range(1,len(self.exppeaks[1])):\n",
    "                if self.exppeaks[0][-x]<40000 and
self.exppeaks[0][-x]>26500:\n",
    "                    peaks_x.append(self.exppeaks[0][-x])\n",
    "                    peaks_y.append(self.exppeaks[1][-x])\n",
    "                    signal.append(self.exppeaks[1][-x]/avg_noise)
\n",
    "                    if len(signal) == numvals:\n",
    "                        break\n",
    "            snr = np.array([peaks_x, peaks_y, signal])\n",
    "            return snr"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "metadata": {},
   "outputs": [],
   "source": [
    "def runspectra(expfile, predfile, autoselect=False, fitfile=''):
\n",
    "    '''generates a interactive plot for matching experimental and
predicted data\n",
    "\n",
    "    arguments:\n",
    "    expfile -- the csv file containing experimental data\n",
    "    predfile -- the .xo file containing predicted data\n",
    "    autoimport -- True if we have a fitfile containing fitted
```

```
lines that are imported into linelist automatically\n",
    "    fitfile -- the .xo file containing fits that we want to add
to linelist\n",
    "    '''\n",
    "    spec = spectra(expfile,predfile,autoselect,fitfile)\n",
    "    traces = spec.create_traces()\n",
    "\n",
    "    # create plot\n",
    "    f = go.FigureWidget(traces)\n",
    "    f.update_layout(plot_bgcolor='white')\n",
    "    f.update_layout(\n",
    "        title='Comparison of Predicted and Experimental Data',
\n",
    "        xaxis=dict(\n",
    "            title='Frequency',\n",
    "            tickmode='array',\n",
    "            tickvals=[26500 + x * 1000 for x in range(14)],  #
Example tick values, adjust as needed\n",
    "            range=[26500, 40000],  # Set range for x-axis (adjust
as needed)\n",
    "        ),\n",
    "        yaxis=dict(\n",
    "            title='Intensity',\n",
    "            range=[-5, 5],  # Set range for y-axis (adjust as
needed)\n",
    "        ),\n",
    "        width=1000,\n",
    "        height=750,\n",
    "        barmode='group'\n",
    "    )\n",
    "\n",
    "    # create buttons for clicking\n",
    "    scatter = f.data[0]\n",
    "    colors = ['silver'] * 1000\n",
    "    scatter.marker.color = colors\n",
    "    scatter.marker.size = [10] * 1000\n",
    "    f.layout.hovermode = 'closest'\n",
    "\n",
    "    # create our callback function\n",
    "    def update_point(trace, points, selector):\n",
    "        c = list(scatter.marker.color)\n",
    "        s = list(scatter.marker.size)\n",
    "        for i in points.point_inds:\n",
    "            if c[i] == 'palevioletred':\n",
    "                c[i]='silver'\n",
    "            else:\n",
    "                c[i] = 'palevioletred'\n",
    "                s[i] = 20\n",
    "            with f.batch_update():\n",
    "                scatter.marker.color = c\n",
```

```
    "                scatter.marker.size = s\n",
    "        if points.point_inds and points.trace_index == 0:\n",
    "            ind = points.point_inds[0]\n",
    "            if trace.y[ind] < 0:\n",
    "                if len(spec.predlines) == len(spec.explines) or
len(spec.predlines)+1 == len(spec.explines):\n",
    "                    spec.predlines.append([float(trace.x[ind]),
\n",
    "                                      spec.p_data_pos[2]
[np.where(spec.p_data_pos == trace.x[ind])[1]][0], \n",
    "                                      spec.p_data_pos[3]
[np.where(spec.p_data_pos == trace.x[ind])[1]][0]])\n",
    "                else:\n",
    "                    spec.predlines[-1] = [float(trace.x[ind]),
\n",
    "                                     spec.p_data_pos[2]
[np.where(spec.p_data_pos == trace.x[ind])[1]][0], \n",
    "                                     spec.p_data_pos[3]
[np.where(spec.p_data_pos == trace.x[ind])[1]][0]]\n",
    "            else:\n",
    "                if len(spec.predlines) == len(spec.explines) or
len(spec.predlines) == len(spec.explines)+1:\n",
    "                    spec.explines.append(float(trace.x[ind]))\n",
    "                else:\n",
    "                    spec.explines[-1] = float(trace.x[ind])\n",
    "    scatter.on_click(update_point)\n",
    "\n",
    "    # show linelist\n",
    "    out = widgets.Output()\n",
    "\n",
    "    @out.capture(clear_output=True, wait=True)\n",
    "    def linelist():\n",
    "        if len(spec.explines)==len(spec.predlines):\n",
    "            print('      J K- K+  J K- K+    Sym   calc/GHz
diff/MHz     obs/GHz')\n",
    "            print(spec.generate_lines())\n",
    "        elif len(spec.explines)>len(spec.predlines):\n",
    "            print('more experimental data than predicted')\n",
    "        else:\n",
    "            print('more predicted data than experimental')\n",
    "\n",
    "    with out:\n",
    "        linelist()\n",
    "\n",
    "    # allow new lines to be deleted\n",
    "    def deleteline(s):\n",
    "        if s1.value != 0:\n",
    "            spec.predlines.pop(s.value-1-len(spec.fitlines))\n",
    "            spec.explines.pop(s.value-1-len(spec.fitlines))\n",
    "        s1.value=0\n",
```

```
    "        linelist()\n",
    "\n",
    "    # annotate large peaks with signal-to-noise ratios\n",
    "    def showsnr(t):\n",
    "        f.layout.annotations=[]\n",
    "        if t.value > 0:\n",
    "            snr = spec.snr(t.value)\n",
    "            for x in range(t.value):\n",
    "                f.add_annotation(\n",
    "                x=snr[0][x-1],\n",
    "                y=snr[1][x-1],\n",
    "                text=snr[2][x-1],\n",
    "                showarrow=True,\n",
    "                arrowhead=1\n",
    "                )\n",
    "\n",
    "    # click event for update button\n",
    "    def on_button_clicked(b):\n",
    "        s1.max = len(spec.explines)+len(spec.fitlines)\n",
    "        deleteline(s1)\n",
    "        showsnr(numSNR)\n",
    "    \n",
    "    # click event for writing linelist to a file\n",
    "    def write_to_file(b):\n",
    "        if len(spec.predlines)==len(spec.explines):\n",
    "            f1 = open('lines.txt', 'w')\n",
    "            f1.write(spec.generate_lines())\n",
    "            f1.close()\n",
    "        transformlines('lines.txt',spec.explines)\n",
    "    \n",
    "    # reset everything\n",
    "    def resetlines(b):\n",
    "        spec.predlines = []\n",
    "        spec.explines = []\n",
    "        scatter.marker.color = ['silver'] * 1000\n",
    "        scatter.marker.size = [10] * 1000\n",
    "        linelist()\n",
    "        f.layout.annotations = []\n",
    "\n",
    "    # create three buttons for updating, writing, and 
resetting\n",
    "    b1 = 
widgets.Button(description='update',layout=Layout(width='100%', 
height='80px'),style=dict(button_color='#E4F7FF'))\n",
    "    b1.on_click(on_button_clicked)\n",
    "    b1.observe(on_button_clicked)\n",
    "\n",
    "    b2 = widgets.Button(description='write')\n",
    "    b2.layout, b2.style= b1.layout, b1.style\n",
    "    b2.on_click(write_to_file)\n",
```

```
    "    b2.observe(write_to_file)\n",
    "\n",
    "    b3 =
widgets.Button(description='reset',style=dict(button_color='#FFCDCB'))
\n",
    "    b3.layout=b1.layout\n",
    "    b3.on_click(resetlines)\n",
    "    b3.observe(resetlines)\n",
    "\n",
    "    # create a slider for deleting lines from linelist\n",
    "    s1 = widgets.IntSlider(\n",
    "        value=0,\n",
    "        min=0,\n",
    "        max=len(spec.explines)+len(spec.fitlines),\n",
    "        step=1,\n",
    "        description='delete:',\n",
    "        disabled=False,\n",
    "        continuous_update=False,\n",
    "        orientation='horizontal',\n",
    "        readout=True,\n",
    "        readout_format='d'\n",
    "    )\n",
    "\n",
    "    # create a box for entering the number of SNR annotations
added to the plot\n",
    "    numSNR = widgets.IntText(\n",
    "        value=0,\n",
    "        description='Number of SNR annotations:',\n",
    "        disabled=False,\n",
    "        style={'description_width': 'initial'}\n",
    "    )\n",
    "\n",
    "    # show the entire plot\n",
    "    display(widgets.VBox([f, widgets.HBox([out, widgets.VBox([s1,
numSNR, b1, b2, b3])])]))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 6,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "application/vnd.jupyter.widget-view+json": {
       "model_id": "9473ed24cd9740ef997bcf07cd5eee84",
       "version_major": 2,
       "version_minor": 0
      },
      "text/plain": [
```

```
       "VBox(children=(FigureWidget({\n",
       "    'data': [{'marker': {'color': [silver, silver,
silver, ..., silver, silver,\n",
       "…"
      ]
     },
     "metadata": {},
     "output_type": "display_data"
    }
   ],
   "source": [
    "#example\n",
    "runspectra('FTMain1132.csv',
'cyclotene_n40s.xo',True,'cyclotene_n40.xo')"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.11.5"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 2
}
```