```
##DISCLAIMER:
#this whole file was built to analyze individual data points in the same
#run/trial organized by column. Meaning of there were 100 measurements and 3
#runs that there would be 3 columns and 100 rows, a (100,3) numpy array.
#is that is not the case for the data that you wish to use this library upon,
#(for 3 runs of 100 measurements each you have 3 rows and 100
# columns, a (3,100) numpy array))simply np.transpose(your_data) and the rest
# should be fine

#this tool was designed to save you a lot of time and suffering when
#conducting data analysis in python, i hope it helps

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from typing import Optional
from scipy.stats import chi2 as chi2dist
from dataclasses import dataclass


@dataclass
class FitResult:
  model: str
  params: np.ndarray
  params_errs: np.ndarray
  covar: Optional [np.ndarray]
  chi2: float
  dof: int
  red_chi2: float
  pvalue: float
  yhat: np.ndarray
  R2: Optional [float]
  residuals: np.ndarray

def num_sigfigs(data): #returns number of sig figs to report on errors
  #pass in a 1D array
  if np.any(data == (None or np.nan)):
     raise ValueError ("A value of None or nan was encountered")
  errinerr = 1/np.sqrt(2*(data.size)-2)
  if errinerr < 0.005:
    return 3
  elif errinerr < 0.03:
    return 2
  else:
    return 1


def SEM(data, axis=0, ddof=1):
    data = np.asarray(data, dtype=float)

    n = data.shape[axis]
    if n <= ddof:
        raise ValueError ("Not enough data points to compute SEM")

    return np.std(data, axis=axis, ddof=ddof) / np.sqrt(n)


def SEM_as_we_go(data,ddof=1): #this tools calculates the total SEM as
  #after m data points out of the total n data points
  #the goal is to see after how many data points collected has the SEM
  #for the trial stabilized or low enough to justify the number of data
  #necessary to collect. again pass in a 1D
  if np.any(data == (None or np.nan)):
     raise ValueError ("A value of None or nan was encountered")
  data = np.array(data, dtype=float)
  n = np.arange(1, data.size+1)
  S1 = np.cumsum(data)
  S2 = np.cumsum(data**2)
  means = S1 / n
  var_pop = (S2/n) - (means**2)
  var = var_pop * n / (n - ddof)
  sem = np.sqrt(var) / np.sqrt(n)
  return n, sem

def num_of_trials_visual(data): # Just a visualization of the previous idea
  #into a function
  if np.any(data == (None or np.nan)):
     raise ValueError ("A value of None or nan was encountered")
  plt.xlabel('Number of Measurements')
  plt.ylabel('SEM')
  plt.title('SEM as a Function of Number of Measurements')

  SEM_trial = SEM(data)
  mean_trial = np.mean(data, axis = 0)
  legend = (f"the SEM for this trial is {SEM_trial:.1e} and the mean is"\
            f" {mean_trial:.8e}")
  plt.plot(SEM_as_we_go(data)[0], SEM_as_we_go(data)[1],label = legend)
  plt.legend(loc = "best")
  plt.show()
  return None
```

```
def weighted_mean(vals,errs): #pass in two 1D arrays. (first one is values,
  #second one is their respective uncertainties)
  #calculates the weighted mean
  if np.any(vals == (None or np.nan)):
     raise ValueError ("A value of None or nan was encountered")
  if vals.shape != errs.shape:
        raise ValueError(f"shape mismatch: vals{vals.shape} vs errs{errs.shape}")
  if vals.size != errs.size:
        raise ValueError(f"size mismatch: vals{vals.size} vs errs{errs.size}")
  vals = np.asarray(vals)
  errs = np.asarray(errs)
  weights = 1/(errs**2)
  wmean = np.sum(vals*weights, axis = 0)/np.sum(weights, axis = 0 )
  CEsqu = 1/np.sum(weights, axis  = 0 )
  return (wmean, np.sqrt(CEsqu))

def find_plottabl_stuff(vals, errs=None):
    vals = np.asarray(vals, dtype=float)

    # Case 1: no errs provided OR errs is an array full of None
    if errs is None:
        n_meas = vals.shape[0] if vals.ndim >= 1 else vals.size
        mean = np.mean(vals, axis=0) if vals.ndim > 0 else float(vals)

        if n_meas < 2:
            return mean, None    # don't find SEM
        return mean, SEM(vals, axis=0)

    errs_arr = np.asarray(errs, dtype=object)
    if errs_arr.dtype == object and np.all(errs_arr == None):
        n_meas = vals.shape[0]
        mean = np.mean(vals, axis=0)
        if n_meas < 2:
            return mean, None # again no SEM if not enough points
        return mean, SEM(vals, axis=0)

    # Case 2: errs provided -> weighted mean
    plottable_vals, plottable_uncs = weighted_mean(vals, errs)
    return plottable_vals, plottable_uncs



def uncer_compare(array1, array2): #measuring the experimental staistical
  #uncertainties vs the uncertainty on the instrument or whatever other base
  # uncertainty there is. reports dominant one
    if np.any(array1 == None):
       return array2
    if np.any(array2 == None):
       return array1
    newuncs = np.where(array1>=array2,array1,array2)
    return newuncs


def linear_fitter(xvals, yvals,yerrs): #The finder for linear fit models
  def linear_model(x,m,b):
   return m*x+b
  params, covar_mat = curve_fit(linear_model,xvals,yvals,sigma=yerrs,
                               absolute_sigma=True)
  m,b = params
  errors = np.sqrt(np.diag(covar_mat))
  m_err,b_err = errors
  yhat =linear_model(xvals,m,b)

  residuals = yvals - yhat
  dof = int(xvals.size - len(params))
  if yerrs is None:
    fakechi2 = float(np.sum(residuals**2))
    fakeredchi2 = fakechi2/dof if dof >0 else np.nan
    pvalue =np.nan
    model = "linear fit"
    R2 = 1 - np.sum((yvals - yhat)**2) / np.sum((yvals - np.mean(yvals))**2)
    return FitResult(model, params,errors,None,fakechi2,dof,fakeredchi2,pvalue,
                    yhat,R2,residuals)
  chi2 = float(np.sum((residuals/yerrs)**2))
  redchi2 = chi2/dof
  pvalue = float(chi2dist.sf(chi2,dof)) if dof>0 else np.nan
  covar = covar_mat
  model = "linear fit"
  R2 = 1 - np.sum((yvals - yhat)**2) / np.sum((yvals - np.mean(yvals))**2)
  return FitResult(model,params,errors,covar, chi2,dof, redchi2,pvalue,yhat,
                  R2, residuals)

def run_stackera0(runs): #Input in a python list of all your runs, what this
   #function does it that it output two arrays that you can automatically put
   # into the weighted mean function
   # So when you have many runs, you don't have to manually take means and
   # SEMS important that
   # you don't pass in yerrs, for that use find_plottable_stuff
   #WORKS ALONG AXIS=0
  means  = []
  SEMS = []
  for i in range(len(runs)):
     mean  = np.mean(runs[i],axis=0)
     errs = SEM(runs[i])
     means.append(mean)
     SEMS.append(errs)
  vals = np.stack(means,axis=0)
  uncs  = np.stack(SEMS,axis =0)
  return (vals,uncs)

def run_stackera1(runs): #Input in a python list of all your runs, what this
   #function does it that it output two arrays that you can automatically put
   # into the weighted mean function
   # So when you have many runs, you don't have to manually take means and
   # SEMS important that
   # you don't pass in yerrs, for that use find_plottable_stuff
   #WORKS ALONG AXIS=1
  means  = []
  SEMS = []
  for i in range(len(runs)):
     mean  = np.mean(runs[i],axis=1)
     errs = SEM(runs[i])
     means.append(mean)
     SEMS.append(errs)
  vals = np.stack(means,axis=1)
  uncs  = np.stack(SEMS,axis =1)
  return (vals,uncs)

def plot_shower(xvalues,yvalues,plot,xerr = None,yerr=None,xticks= None,
                yticks = None):
  #Important, the class of plot must be a FitResult, so that
  #info can be extracted quickly.
  if (plot.model == "linear fit"):
      fit_y = plot.params[0]*xvalues + plot.params[1]
      leg_des = f"best fit curve: {plot.params[0]:.4e}x + {plot.params[1]:.4e}, R2 = {plot.R

  plt.errorbar(xvalues,yvalues,yerr =yerr,xerr=xerr,fmt = "o",capsize=5)
  plt.plot(xvalues,fit_y,label = leg_des)
  plt.xticks(xvalues,xticks,rotation =45, ha= "right")
  plt.yticks(yvalues,yticks,)
  plt.minorticks_on()
  plt.grid(True,alpha = 0.3, which = "both")
  xlabel = input("Enter x axis name")
  ylabel = input("Enter y axis name")
  title = input("Enter title name")
  #xlim=(float(input("Enter lowest x bound")),float(input("Enter highest x bound")))
```

```
    #ylim=(float(input("Enter lowest y bound")),float(input("Enter highest y bound")))
    #plt.xlim(xlim)
    #plt.ylim(ylim)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.tick_params(axis="both", which="major", length=8, width=1.5)
    plt.tick_params(axis="both", which="minor", length=4, width=1.0)
    plt.title(title)
    plt.legend(loc = 'best')
    plt.show()

    ##Time for residuals plot and their info
    plt.errorbar(xvalues,plot.residuals,yerr= yerr, xerr =xerr, fmt = "o", capsize=5)
    leg_text = rf"red_chi^2 = {plot.red_chi2:.4e}, pvalue = {plot.pvalue:.2e},"
    f"dof = {plot.dof}"
    plt.axhline(0, linestyle="--", linewidth=1.2, alpha=0.8)
    plt.xticks(xvalues,xticks,rotation =45, ha= "right")
    plt.minorticks_on()
    plt.grid(True,alpha = 0.3, which = "both")
    plt.legend(loc = "best",title=leg_text)
    xlabres = input("Enter x axis title of residual plot")
    ylabres = input("Enter y axis title of residual plot")
    plt.xlabel(xlabres)
    plt.title("Plot of Residuals for Each Point")
    plt.ylabel(ylabres)
    plt.show()

  def multiple4_plot_shower(xvalues1,xvalues2,yvalues1,plot1,yvalues2,plot2,
                            xvalues3 = None, xvalues4 =None, yvalues3=None,
                            plot3 =None, yvalues4 =None, plot4 = None,
                            xerr1= None, xerr2 =None, xerr3 = None,
                            xerr4 = None, xticks1= None, xticks2=None,
                            xticks3 = None, xticks4 = None, yticks1 =None,
                            yticks2 = None, yticks3 = None, yticks4 = None,
                            yerr1=None, yerr2=None, yerr3=None, yerr4=None):
    # all plot arguments must be of class FitResult, and yeah, just plotting
    #multiple graphs.
      plots = [plot1,plot2,plot3,plot4]
      xvaluesS = [xvalues1,xvalues2,xvalues3,xvalues4]
      yvaluesS = [yvalues1,yvalues2,yvalues3,yvalues4]
      xerrS = [xerr1,xerr2,xerr3,xerr4]
      xticksS = [xticks1,xticks2,xticks3,xticks4]
      yticksS = [yticks1,yticks2,yticks3,yticks4]
      yerrS = [yerr1,yerr2,yerr3,yerr4]
      while plots.count(None) !=0:
        plots = plots[:(len(plots)-1)]
        xvaluesS = xvaluesS[:(len(xvaluesS)-1)]
        yvaluesS = yvaluesS[:(len(yvaluesS)-1)]
        xerrS = xerrS[:(len(xerrS)-1)]
        xticksS = xticksS[:(len(xticksS)-1)]
        yticksS = yticksS[:(len(xticksS)-1)]
        yerrS = yerrS[:(len(yerrS)-1)]
        if len(plots)==0:
           raise ValueError ("FitResult object count went to 0, something must have"
           "been passed in wrong")
    #plot main plot,
      for i in range(len(plots)):
        if plots[i].model == "linear fit":
          fit = plots[i].params[0]*xvaluesS[i] + plots[i].params[1]
          leg_des = input(f'Enter any info about plot {i+1}, this will go in legend')
          print(f"best fit curve{i+1}: {plots[i].params[0]:.4e}x +"
          f"{plots[i].params[1]:.4e}")
          plt.errorbar(xvaluesS[i],yvaluesS[i],yerr =yerrS[i],xerr=xerrS[i],fmt = "o",
                       capsize=5)
          plt.plot(xvaluesS[i],fit,label = leg_des)
          plt.xticks(xvaluesS[i],xticksS[i],rotation =45, ha= "right")
          if not (yticksS.count(None) !=0):
            plt.yticks(yvaluesS[i],yticksS[i])
      plt.minorticks_on()
      plt.grid(True, alpha=0.3,which="both")
      xlabel = input("Enter x axis name")
      ylabel = input("Enter y axis name")
      title =  input("Enter title of plot name")
      plt.xlabel(xlabel)
      plt.ylabel(ylabel)
      plt.tick_params(axis="both", which="major", length=8, width=1.5)
      plt.tick_params(axis="both", which="minor", length=4, width=1.0)
      plt.title(title)
      plt.legend(loc = "best")
      plt.show()
    #plot residual stuff now
      for i in range(len(plots)):
        leg_text = input(f"Enter any info about plot {i+1}, this will go in legend")
        plt.errorbar(xvaluesS[i],plots[i].residuals, yerr = yerrS[i],xerr=xerrS[i],
                     label = leg_text, fmt="o", capsize=5)
        print(rf"red_chi^2 = {plots[i].red_chi2:.4e}, pvalue = {plots[i].pvalue:.2e},"
        rf"dof = {plots[i].dof}")
        plt.xticks(xvaluesS[i],xticksS[i],rotation = 45, ha ="right")
      plt.legend(loc="best")
      plt.axhline(0,linestyle = "--", linewidth=1.2,alpha=0.8)
      xlabelres = input("Enter x axis title of residual plot")
      ylabelres = input("Enter y axis title of resudual plot")
```

```
    plt.xlabel(xlabelres)
    plt.ylabel(ylabelres)
    plt.title("Residuals for each point")
    plt.show()


## phase matcher function. pass in two arrays of maybe different length
## it will make them equal length by removing elements out of relative tolerance
## between values of the two arrays. like the "neighboring" ones will be kept,
## and other ones that don't mathc anything close are removed

def match_peaks_one_to_one(a, b, tol=20):
    a = np.asarray(a)
    b = np.asarray(b)

    # choose which to iterate over (the longer)
    if len(a) >= len(b):
        longer, shorter = a, b
        longer_is_a = True
    else:
        longer, shorter = b, a
        longer_is_a = False

    used = np.zeros(len(shorter), dtype=bool)
    L_keep = []
    S_keep = []

    for x in longer:
        d = np.abs(shorter - x)
        d[used] = np.inf
        j = np.argmin(d)
        if d[j] <= tol:
            L_keep.append(x)
            S_keep.append(shorter[j])
            used[j] = True

    L_keep = np.array(L_keep)
    S_keep = np.array(S_keep)

    # return as (a_matched, b_matched)
    if longer_is_a:
        return L_keep, S_keep
    else:
        return S_keep, L_keep
```