Supplementary File 5: Python code for extracting a defined percentage of a curve.

Due to the volume of LiCor light response data, 647 data sets and each containing 120 data points per trait, a bespoke code was developed to extract the data points on the induction side of the curve. The code allows you to define a percentage of the maximum value on a curve, in this case 95% as used in the work by McAusland et al (2016), and extracts all data points up until this defined percentage (e.g. 0 – 95% of the maximum induction curve).
The code was written in Python, consisting of two files, dataManipulation.py and configuration.py, which work together. dataManipulation.py pulls parameters from configuration.py, to dictate the origin of the data file, at what percentage threshold the data is filtered by (e.g. 95% of the maximum stomatal conductance value), which columns from the original data file are used and where to save the output file, then executes the task.
The code is sensitive to undulations in the induction curve, the code works very well on carbon assimilation and NPQ data, which have a smooth and continuous induction curve. However, stomatal conductance induction (*gs*) curves are notoriously 'noisy' and can undulate. For example, if this code encounters a downward trend in an undulating *gs* induction curve, it will identify that undulating peak as the maximum and pull that portion of the data up until that point. Thus, a certain degree of manual quality checking, and occasional editing, is still required when using this code on *gs*, or noisy, induction curves, though it is still faster than manually calculating and extracting the data.

This code was kindly developed by Matthew Hartley, a software developer at ESPER, a world leading 3D scanning and surface capture company. It can be accessed from his GitHub repository at: `https://github.com/mattVHartley/sophieDataProcessor`

The code for configuration.py and dataManipulation.py can be seen in the remaining pages of this document.

Configuration.py:

```
'''
Column numbers for reference:

R.AccGs.csv
  PAR | time | name  | rep # | leaf Width |  GS  |
   0  |  1   |   2   |  3    |      4     |   5  |



Pracha_data.csv
PAR | Time (sec) | Line | ADNID code | Accession | ADN code | Rep |
Photo | cond | NPQ | ETR | WUE | Trmmol |
 0  |     1      |   2  |     3      |     4     |     5    |  6  |
7   |   8  |  9  |  10 |  11 |   12   |

'''

# filter percentage - 95% as 95
filterPercent = 95

# column positions in the csv
timeCol = 1
nameCol = 2
repCol = 3
filterCol = 8 #this is the dependant variable

saveOutASubset = True
numToSubset = 5

# file names
#inputFileName = '/home/matt/Desktop/testdata.csv' #assume in same
directory as this code, unless you specify a separate pathway
```

```
#inputFileName = './Pracha_data.csv' #assume in same directory as
this code, unless you specify a separate pathway
inputFileName = 'NPQ_for_induction.csv' #assume in same directory as
this code, unless you specify a separate pathway
outputFileName = 'NPQ_Induction.csv' #this will get dumped in your
set working directory.
```

DataManipulation.py:

```
import csv
from configuration import filterPercent, timeCol, nameCol, repCol,
filterCol, inputFileName, outputFileName, numToSubset,
saveOutASubset


# this function does a deal of typechecking and organising of raw
input data
def parseData(data):
    global maxRepNumber
    toReturn = []
    incorrectCount = 0
    for line in data:

        try:
            # make sure everything is of the right variable type
            line[repCol] = int(line[repCol])
            line[filterCol] = float(line[filterCol])
            line[timeCol] = float(line[timeCol])

            # Work out what the maximum repetition number is
            if line[repCol] > maxRepNumber:
                maxRepNumber = line[repCol]
            toReturn.append(line)
        except:
            incorrectCount += 1
            rejectedData.append(line)

    print("")
    if incorrectCount > 0:
        print("Loaded " + str(len(data)) + ' data points from file '
+ inputFileName)
        print("Accepted " + str(len(toReturn)) + " data points")
        print("Discarded " + str(incorrectCount) + ' rows as they
contain incorrect variable types')
        print(" ")
        print(
            'repetition columns can only contain integers, the
column to filter and the time column by must be numeric')
        print(" ")
        print(" ")
    else:
        print("Loaded " + str(len(data)) + ' data points from file '
+ inputFileName)
        print("Accepted " + str(len(toReturn)) + " data points")

    print("----------------------------------------------------------
---------")
    return toReturn



def getUniqueRiceLines(rices):
    # loop through to get all the names of the rice into one big
long list
    allRice = []
    for rice in rices:
        allRice.append(rice[nameCol])

    uniqueRice = list(set(allRice))  # set is a data structure that
cannot have duplicates in, turn it into a set then turn it back into
a list

    return uniqueRice



def filterData(dataArrayArgs):
    # check we have data
    if len(dataArrayArgs) > 0:

        # sort into ascending time order
        dataArray = sorted(dataArrayArgs, key=lambda k:
float(k[timeCol]))

        # get the highest value
        max = 0
        for point in dataArray:
            if point[filterCol] > max:
                max = point[filterCol]

        # calculate the value of the cutoff based on percentage
        cutoff = float(percentile * max)

        prunedData = []
        for i in range(0, len(dataArray)):
            if dataArray[i][filterCol] < cutoff:
                prunedData.append(dataArray[i])
            else:
                print("stopping at data point " + str(i) + '/' +
str(len(dataArray)) + '   -  name:' + str(
                    dataArray[i][nameCol]) + ', rep:' +
str(dataArray[i][repCol]))
                break
        return prunedData



# 'r' is read. read as opposed to write, Universal is allowing for
maximum compatibility with each csv format
f = open(inputFileName, 'r')

# pass the file to the csv reader
rawdata = csv.reader(f)

allData = []

headings = []
first = True
for row in rawdata:
    if first:
        headings = row
        first = False
    allData.append(row)

maxRepNumber = 0  # initilise this in global scope for the call to
parseData
rejectedData = []
allData = parseData(allData)

percentile = float(filterPercent) / 100

uniqueRice = getUniqueRiceLines(allData)
```

```
print('Detected ' + str(len(uniqueRice)) + ' unique data series')
print("------------------------------------------------------------------
-----")

dataStack = []

for rice in uniqueRice:
    for i in range(1, maxRepNumber + 1):
        singleDataSeries = []
        for entry in allData:
            if entry[nameCol] == rice and entry[repCol] == i:
                singleDataSeries.append(entry)
        if len(singleDataSeries) > 0:
            dataStack.append(singleDataSeries)

print("separated out into " + str(len(dataStack)) + " unique data
series")

print("filtering by values")

counter = 1
subsetRaw = []
subsetFiltered = []
finished = []
for series in dataStack:
    filtered = filterData(series)
    finished.append(filtered)
    if counter <= numToSubset:
        for raw in series:
            subsetRaw.append(raw)
        for filterpoint in filtered:
            subsetFiltered.append(filterpoint)
        counter += 1


recompiledData = [headings]

for series in finished:
    for point in series:
        recompiledData.append(point)

print(str(len(recompiledData) - 1) + ' data points exported to
export.csv')
print(str(len(allData) - len(recompiledData) + 1) + ' data points
were above ' + str(
    filterPercent) + "% of the maximum for its repetition")
print("")
print("")
print("outputting " + str(len(recompiledData) - 1) + " data points
to csv file " + outputFileName)
print("")
print("")
with open(outputFileName, 'w') as csvFile:
    writer = csv.writer(csvFile)
    writer.writerows(recompiledData)
csvFile.close()



print("Number of rejected data ponts to save: "
+str(len(rejectedData)))
print("")
print("")

with open('rejected.csv', 'w') as rejectData:
    writer = csv.writer(rejectData)
    writer.writerows(rejectedData)
csvFile.close()



if saveOutASubset:
    with open('subsetFiltered.csv', 'w') as filtercsv:
        filterwriter = csv.writer(filtercsv)
        filterwriter.writerows(subsetFiltered)
    filtercsv.close()

    with open('subsetRaw.csv', 'w') as rawcsv:
        rawwriter = csv.writer(rawcsv)
        rawwriter.writerows(subsetRaw)
    rawcsv.close()
```