```python
#PSP_travelTime.py
from __future__ import print_function, division

import math
import numpy as np

c = 299792458
NODATA = -9999
MAXDELTAINDEX = 6
SX = 0
DX = 1

class CLine:
    a = NODATA
    b = NODATA

class CPoint:
    x = NODATA
    y = NODATA

flatLine = line1 = line2 = line3 = CLine()
p0 = p1 = p2 = CPoint()
indexP0 = indexP2 = NODATA

timeVector = []
reflecCoeff = []
dy =[]

deltaSpace = 0
deltaTime = 0

def indexOfMaxVector(y, first, last):
    myMax = max(y[first:last])
    for i in range(first, last):
        if (y[i] == myMax):
            return(i)

def indexOfMinVector(y, first, last):
    myMin = min(y[first:last])
    for i in range(first, last):
        if (y[i] == myMin):
            return(i)

def avg(y, index1, index2):
    if (index2 < index1):return(NODATA)
    first = max(index1, 0)
    last = min(index2+1, len(y))
    nrValues = last - first
    return sum(y[first:last]) / nrValues

def normalizeVector(y):
    y = (y-min(y))/(max(y) - min(y))
    avgFirstValues = avg(y, 1, 6)
    return (y - avgFirstValues)

def WF_parameters(Vp, probeHandle, windowBegin, windowWidth, nrPoints):
    global deltaTime, deltaSpace, timeVector
    #abs. time [s] corresponding to the 1st point
    firstPointTime = 2. * windowBegin / (c*Vp)
    deltaSpace = windowWidth /(nrPoints - 1)
```



```python
    deltaTime = 2. * deltaSpace /(c*Vp)
    timeVector = np.zeros(nrPoints, float)
    for i in range(nrPoints):
        timeVector[i] = firstPointTime + deltaTime * i

def runningAverage(y, nrPoints):
    smooth = np.zeros(len(y), float)
    for i in range(len(y)):
        smooth[i] = avg(y, i-nrPoints, i+nrPoints)
    return smooth / max(abs(smooth))

def firstDerivative5Points(y):
    dy = np.zeros(len(y), float)
    for i in range(2):
        dy[i] = 0.
    for i in range(2, len(y)-2):
        dy[i] = (1./(12.)) * (y[i-2] - 8.*y[i-1] + 8.*y[i+1] - y[i+1])
    for i in range(len(y)-2, len(y)):
        dy[i] = 0.
    return dy / max(abs(dy))

# return a line structure with intercept (b) and slope (a)
def weightedLinearRegression (x, y, index1, index2, versus):
    sumX = sumY = 0.
    sumX2 = sumXY = 0.

    if(index1 == index2):
        index1 -= 1
        index2 += 1

    #check index range
    if (index1 < 0):
        index1 = 0
    if (index2 >= len(y)):
        index2 = len(y)-1

    nrPoints = index2-index1+1
    if (versus == SX):
        for i in range(nrPoints-1, -1, -1):
            for j in range (i+1):
                sumX += x[index1+i]
                sumY += y[index1+i]
                sumX2 += (x[index1+i]* x[index1+i])
                sumXY += x[index1+i] * y[index1+i]
    else:
        for i in range(nrPoints):
            for j in range (i+1):
                sumX += x[index1+i]
                sumY += y[index1+i]
                sumX2 += (x[index1+i]* x[index1+i])
                sumXY += x[index1+i] * y[index1+i]

    n = (nrPoints*(nrPoints+1))/2
    line = CLine()
    line.a = (sumXY - sumX * sumY/n) / (sumX2 - sumX * sumX/n)
    line.b = (sumY - line.a * sumX)/n
    return(line)

#backward function
def checkFlatPoint(y, indexMaxDy):
```

```
    index = indexMaxDy
    dy = abs(y[index] - y[index-1])
    threshold = dy / 1000.
    while ((dy > threshold) and (index > 0)):
        index -= 1
        dy = abs(y[index]-y[index-1])
    return (index)

#backward function
def checkZeroValue(y, indexMaxY):
    index = indexMaxY
    while ((y[index] > 0) and (index > 0)):
        index -= 1
    if ((index == 0) and (y[index] > 0)):
        return(NODATA)
    else:
        if (abs(y[index]) < abs(y[index+1])):
            return (index)
        else:
            return (index+1)

def lineIntersection(line1, line2):
    myPoint = CPoint()
    if (line1.a != line2.a):
        myPoint.x = (line2.b - line1.b) / (line1.a - line2.a)
        myPoint.y = myPoint.x * line1.a + line1.b
    else:
        myPoint.x = NODATA
        myPoint.y = NODATA

    return(myPoint)

def computeTravelTime(probeHandle, permittivity, Vp):
    global dy, flatLine, line1, line2, line3
    global indexFlatLine, indexRegr1, indexRegr2, indexRegr3
    global p0, p1, p2

    dy = firstDerivative5Points(reflecCoeff)
    dy = runningAverage(dy, 5)
    indexMaxDerivative = indexOfMaxVector(dy, 0, len(dy))
    indexMinDerivative = indexOfMinVector(dy, 0, len(dy))

    if indexMaxDerivative == 0 or indexMinDerivative == 0:
        return False

    #check first maximum
    if (indexMaxDerivative > indexMinDerivative):
        indexMaxDerivative = indexOfMaxVector(dy, 0, indexMinDerivative)

    #search first reflection
    indexFlatLine = checkFlatPoint(reflecCoeff, indexMaxDerivative)
    nrPoints = len(reflecCoeff)
    step = int(8.0 * (nrPoints / 256.0))
    average = avg(reflecCoeff, indexFlatLine - step, indexFlatLine)
    flatLine.a = 0
    flatLine.b = average

    delta = min((indexMaxDerivative - indexFlatLine), MAXDELTAINDEX)
    indexRegr1 = indexFlatLine + delta
    line1 = weightedLinearRegression(timeVector, reflecCoeff,
```

微分ゼロ点の index の取得

global；外部からこの関数内で計算した変数を参照できるようにグローバル変数として定義

一次微分の計算とその移動平均

微分係数の最大値と最小値における index を取得

平坦な点の終端を探索

indexRegr1-delta から indexRegr1+delta の範囲で右から左に重み付け線形回帰


```
                    indexRegr1 - delta, indexRegr1 + delta, SX)

    p0 = lineIntersection(flatLine, line1)
    dt0 = (2. * probeHandle * math.sqrt(permittivity)) / (c*Vp)
    p1.x = p0.x + dt0
    index = int(p1.x / deltaTime)
    p1.y = reflecCoeff[index]

    #search second reflection
    indexSecondMaxDerivative = indexOfMaxVector(dy, indexMinDerivative, len(dy))
    indexZeroDerivative = checkZeroValue(dy, indexSecondMaxDerivative)
    delta = min((indexSecondMaxDerivative - indexZeroDerivative), MAXDELTAINDEX)
    indexRegr2 = indexZeroDerivative - delta
    indexRegr3 = indexZeroDerivative + delta

    line2 = weightedLinearRegression(timeVector, reflecCoeff,
                        indexRegr2 - delta, indexRegr2 + delta, DX)

    line3 = weightedLinearRegression(timeVector, reflecCoeff,
                        indexRegr3 - delta, indexRegr3 + delta, SX)

    p2 = lineIntersection(line2, line3)
    return True
```

フラットラインと最初のピークまでの直線との交点探索->p0

プローブハンドル内の伝播時間

二つ目の微分最大値の index

DX の線と SX の線の交点探索