Documentation

tsmoffat

September 2016

1 Introduction

The aim of this document is to explain the code in the program and how to run it etc. It will be in lieu of comments within the code, due to the large amount of repeated code within the program

2 Installation

To install the program, make sure you have Git installed on your system as well as Git LFS (for large files). Get the URL of the project from GitHub, navigate to where you would like the project to be stored and run git clone (URL) in your terminal/command line. This will pull the project from GitHub. Then type "git lfs track *.txt" into your terminal to make sure that the large file system is tracking text files, before running git clone (URL) again to download the text files for this program.

To run the program, install Anaconda for Python 3.5 from `https://www.continuum.io/downloads`. This will make sure that everything is installed correctly. Once this is installed, restart your terminal then type "pip install tabulate" to install the missing package needed for this program.

3 Running

There are two options for this. Either navigate to the directory containing the project in your terminal and type "python3 ./Testing2.py" or load the project up in PyCharm and run it from there.

4 Code Explanation

This section is intended to explain what the code in the program does, as quite a lot of it is repeated and it saves repeating comments. The self arguments that every function has refer to global variables and constants.

4.1 AttenuationSearch.py

```
"""A module to find the required attenuation."""
import decimal as dec


def attlist(self):
    """Generate a list of all the values in the attenuation sheet."""
    listatt = []
    for row in self.dsa.iter_rows(row_offset=2):
        if row[self.dsas2128].value is not None:
            # Adds value to list if it actually has a value
            listatt.append(row[self.dsas2128].value)
    return listatt


def closest(self, attlist):
    """Find closest attenuation to target."""
    closest = min(attlist, key=lambda x: abs(
        dec.Decimal(x) - dec.Decimal(self.targetatt)))
    closest = dec.Decimal(closest)
    return closest


def attenuationsearch(self, attlist, closest):
    """Search for the most accurate attenuation."""
    for row in self.dsa.iter_rows():
        if row[self.dsas2128].value == closest:
            att2128 = closest.quantize(dec.Decimal(
                '.001'), rounding=dec.ROUND_HALF_UP)
            row28 = row[self.dsastate28].value
            att2124 = dec.Decimal(row[self.dsas2124].value).quantize(
                dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
            att2132 = dec.Decimal(row[self.dsas2132].value).quantize(
                dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
            phase2124 =
             ↪  dec.Decimal(row[self.dsaphase2124].value).quantize(
                dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
            phase2128 =
             ↪  dec.Decimal(row[self.dsaphase2128].value).quantize(
                dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
            phase2132 =
             ↪  dec.Decimal(row[self.dsaphase2132].value).quantize(
                dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)

```

```
41      return {'row28': row28, 'att2124': att2124, 'att2128': att2128,
         ↪  'att2132': att2132, 'phase2124': phase2124, 'phase2128':
         ↪  phase2128, 'phase2132': phase2132}
```

This is AttenuationSearch.py. Its purpose is to search through the DSA sheet in the spreadsheet and find the closest attenuation to the input attenuation. The first function, attlist, just generates a list of all the values present in the DSA sheet, then returns it. It does this by iterating through the column with the S21 values of attenuation, checking if they have a value, and appending them to the list if they do. The row offset in line 8 is to remove anything that is not a number from the search area.

The next function, closest, takes as its argument attlist, which is the list generated in the previous function. It uses a lambda function to find the value with the smallest absolute distance from the target attenuation, which is called through self.targetatt. This is then converted into a decimal value, as this allows for rounding, unlike a float, and then returns the value.

The last function takes as its arguments attlist and closest, and then it iterates through the rows, finding which value is equal to the closest value then gets the values needed for various other parts of the program from the spreadsheet. All these are then returned in a data dictionary, which is a data type in python that allows for values to be referenced using a key.

4.2 extras.py

```
1   """Literally some magic."""
2   import decimal as dec
3
4
5   def check180(self, set180):
6       """180 degrees of magic."""
7       dec.getcontext().prec = 6
8       if self.targetphase in set180:
9           for row in self.s180.iter_rows():
10              if row[self.phase28].value == self.targetphase:
11                  row1 = int(row[0].value)
12                  att1 = dec.Decimal(row[self.att28].value)
13                  phaselow =
                     ↪  dec.Decimal(row[self.phase24].value).quantize(
14                      dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
15                  phasehigh =
                     ↪  dec.Decimal(row[self.phase32].value).quantize(
16                      dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
17                  phasediff = phasehigh - phaselow
18
19          return {'phase1': self.targetphase, 'row1': row1, 'att1': att1,
             ↪  'phasediff': phasediff, 'source1': 's180', 'totalphase':
             ↪  att1, 'total': self.targetphase}
20      else:
21          closest = min(set180, key=lambda x: abs(
```

```
            dec.Decimal(x) - dec.Decimal(self.targetphase)))
        for row in self.s180.iter_rows():
            if row[self.phase28].value == closest:
                closestround = closest.quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                row1 = int(row[0].value)
                att1 = dec.Decimal(row[self.att28].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phaselow =
                 ↪  dec.Decimal(row[self.phase24].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasehigh =
                 ↪  dec.Decimal(row[self.phase32].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasediff = phasehigh - phaselow

        return {'phase1': closestround, 'row1': row1, 'att1': att1,
         ↪  'phasediff': phasediff, 'source1': 's180', 'totalphase':
         ↪  att1, 'total': closestround}


def check90(self, set90):
    """90 degrees of magic."""
    dec.getcontext().prec = 6
    if self.targetphase in set90:
        for row in self.s90.iter_rows():
            if row[self.phase28].value == self.targetphase:
                row1 = int(row[0].value)
                att1 = dec.Decimal(row[self.att28].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phaselow =
                 ↪  dec.Decimal(row[self.phase24].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasehigh =
                 ↪  dec.Decimal(row[self.phase32].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasediff = phasehigh - phaselow

        return {'phase1': self.targetphase, 'row1': row1, 'att1': att1,
         ↪  'phasediff': phasediff, 'source1': 's90', 'totalphase':
         ↪  att1, 'total': self.targetphase}

    else:
        closest = min(set90, key=lambda x: abs(
            dec.Decimal(x) - dec.Decimal(self.targetphase)))
```

```
        for row in self.s90.iter_rows():
            if row[self.phase28].value == closest:
                closestround = closest.quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                row1 = int(row[0].value)
                att1 = dec.Decimal(row[self.att28].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phaselow =
                 ↪ dec.Decimal(row[self.phase24].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasehigh =
                 ↪ dec.Decimal(row[self.phase32].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasediff = phasehigh - phaselow

        return {'phase1': closestround, 'row1': row1, 'att1': att1,
         ↪ 'phasediff': phasediff, 'source1': 's90', 'totalphase':
         ↪ att1, 'total': closestround}


def check45(self, set45):
    """45 Degrees of magic.

    Parameters
    ----------
    self, set45, set45
    Returns
    -------
    phase1, row1, att1, phasediff
    """
    dec.getcontext().prec = 6
    if self.targetphase in set45:
        for row in self.s45.iter_rows():
            if row[self.phase28].value == self.targetphase:
                row1 = int(row[0].value)
                att1 = dec.Decimal(row[self.att28].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phaselow =
                 ↪ dec.Decimal(row[self.phase24].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasehigh =
                 ↪ dec.Decimal(row[self.phase32].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasediff = phasehigh - phaselow

```

```
        return {'phase1': self.targetphase, 'row1': row1, 'att1': att1,
         ↪  'phasediff': phasediff, 'source1': 's45', 'totalphase':
         ↪  att1, 'total': self.targetphase}

    else:
        closest = min(set45, key=lambda x: abs(
            dec.Decimal(x) - dec.Decimal(self.targetphase)))
        for row in self.s45.iter_rows():
            if row[self.phase28].value == closest:
                closestround = closest.quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                row1 = int(row[0].value)
                att1 = dec.Decimal(row[self.att28].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phaselow =
                 ↪  dec.Decimal(row[self.phase24].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasehigh =
                 ↪  dec.Decimal(row[self.phase32].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasediff = phasehigh - phaselow

        return {'phase1': closestround, 'row1': row1, 'att1': att1,
         ↪  'phasediff': phasediff, 'source1': 's45', 'totalphase':
         ↪  att1, 'total': closestround}


def check225(self, set225):
    """22.5 degrees of magic.

    Parameters
    ----------
    self, s225, set225
    Returns
    -------
    phase1, row1, att1, phasediff
    """
    dec.getcontext().prec = 6
    if self.targetphase in set225:
        for row in self.s225.iter_rows():
            if row[self.phase28].value == self.targetphase:
                row1 = int(row[0].value)
                att1 = dec.Decimal(row[self.att28].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phaselow =
                 ↪  dec.Decimal(row[self.phase24].value).quantize(
```

```
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasehigh =
                 ↪ dec.Decimal(row[self.phase32].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasediff = phasehigh - phaselow

        return {'phase1': self.targetphase, 'row1': row1, 'att1': att1,
         ↪ 'phasediff': phasediff, 'source1': 's225', 'totalphase':
         ↪ att1, 'total': self.targetphase}

    else:
        closest = min(set225, key=lambda x: abs(
            dec.Decimal(x) - dec.Decimal(self.targetphase)))
        for row in self.s225.iter_rows():
            if row[self.phase28].value == closest:
                closestround = closest.quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                row1 = int(row[0].value)
                att1 = dec.Decimal(row[self.att28].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phaselow =
                 ↪ dec.Decimal(row[self.phase24].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasehigh =
                 ↪ dec.Decimal(row[self.phase32].value).quantize(
                    dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
                phasediff = phasehigh - phaselow

        return {'phase1': closestround, 'row1': row1, 'att1': att1,
         ↪ 'phasediff': phasediff, 'source1': 's225', 'totalphase':
         ↪ att1, 'total': closestround}


def checkall(self, set180, set90, set45, set225):
    """Check to find most accurate solution so far.

    Parameters
    ----------
    self
    Returns
    -------
    bestsol data dict
    """
    dec.getcontext().prec = 6
    sol180 = check180(self, set180)
    sol90 = check90(self, set90)
```

```
    sol45 = check45(self, set45)
    sol225 = check225(self, set225)
    bestsollist = [sol180['phase1'], sol90[
        'phase1'], sol45['phase1'], sol225['phase1']]

    if sol180['phase1'] == self.targetphase:
        return sol180

    elif sol90['phase1'] == self.targetphase:
        return sol90

    elif sol45['phase1'] == self.targetphase:
        return sol45

    elif sol225['phase1'] == self.targetphase:
        return sol45

    else:
        closest = min(bestsollist, key=lambda x: abs(
            dec.Decimal(x) - dec.Decimal(self.targetphase)))
        if closest == sol180['phase1']:
            return sol180
        elif closest == sol90['phase1']:
            return sol90
        elif closest == sol45['phase1']:
            return sol45
        elif closest == sol225['phase1']:
            return sol225
        else:
            return None


def mostaccurate(self, bestresult, bestresult2, bestresult3,
 ↪  bestresult4, sollist):
    """Find most accurate solution and return."""
    bestsol = min(sollist, key=lambda x: abs(
        dec.Decimal(x) - dec.Decimal(self.targetphase)))
    if bestsol == bestresult['total']:
        return bestresult
    elif bestsol == bestresult2['total']:
        return bestresult2
    elif bestsol == bestresult3['total']:
        return bestresult3
    elif bestsol == bestresult4['total']:
        return bestresult4
    else:
```

```
221             return None
```

Extras.py is the multi-use module in this program. The first four functions are used to find if there is a single array answer. These all work in roughly the same way. First, the program looks to see if the target phase is actually present at all in the spreadsheet. If it is then it returns that value, along with the corresponding row and attenuation. Otherwise, it uses a lambda function to find the closest value in the set (which is like a list but doesn't allow for duplicate values) and returns the values for that.

The checkall function is what controls the previous four functions. It calls them then decides which value to return, by looking to see if any of the returned values equal the target value, and if not finding the one with the smallest distance.

The final module in this takes the most accurate values from finding one, two, two, three and four phase answers and then returns the best one. The way it is laid out means that if a value is found with multiple phases (e.g. 2, 3 and 4 phases), the program will always prefer the more efficient solution (i.e. the one with the fewest phases involved). subsectionFourPhase.py

```
"""Test all four phases in one."""
import decimal as dec
import os


def closest_finder(self):
    """Find the closest value to a given value."""
    combined = set(map(str.rstrip, open(
        (os.path.join(os.path.dirname(__file__), '1809045225.txt')))))
    closest = min(combined, key=lambda x: abs(
        dec.Decimal(x) - dec.Decimal(self.targetphase)))
    closest = dec.Decimal(closest)
    return closest
    del combined


def check(self, set180, set90, set45, set225, closest):
    """Check to find the most accurate four phase solution."""
    dec.getcontext().prec = 6
    for i in set180:
        for j in set90:
            for k in set45:
                for l in set225:
                    total = i + j + k + l
                    if total == closest:
                        for row in self.s180:
                            if row[self.phase28].value == i:
                                row1 = int(row[0].value)
                                att1 =
                                 ↪  dec.Decimal(row[self.att28].value).quantize(
```

```
                        dec.Decimal('.001'),
                        ↪ rounding=dec.ROUND_HALF_UP)
                    phaselow1 =
                    ↪ dec.Decimal(row[self.phase24].value).quantize(
                        dec.Decimal('.001'),
                        ↪ rounding=dec.ROUND_HALF_UP)
                    phasehigh1 =
                    ↪ dec.Decimal(row[self.phase32].value).quantize(
                        dec.Decimal('.001'),
                        ↪ rounding=dec.ROUND_HALF_UP)
                    phasediff1 = phasehigh1 - phaselow1

                for row in self.s90:
                    if row[self.phase28].value == j:
                        row2 = int(row[0].value)
                        att2 =
                        ↪ dec.Decimal(row[self.att28].value).quantize(
                        dec.Decimal('.001'),
                        ↪ rounding=dec.ROUND_HALF_UP)
                        phaselow2 =
                        ↪ dec.Decimal(row[self.phase24].value).quantize(
                        dec.Decimal('.001'),
                        ↪ rounding=dec.ROUND_HALF_UP)
                        phasehigh2 =
                        ↪ dec.Decimal(row[self.att32].value).quantize(
                        dec.Decimal('.001'),
                        ↪ rounding=dec.ROUND_HALF_UP)
                        phasediff2 = phasehigh2 - phaselow2

                for row in self.s45:
                    if row[self.phase28].value == k:
                        row3 = int(row[0].value)
                        att3 =
                        ↪ dec.Decimal(row[self.att28].value).quantize(
                        dec.Decimal('.001'),
                        ↪ rounding=dec.ROUND_HALF_UP)
                        phaselow3 =
                        ↪ dec.Decimal(row[self.att24].value).quantize(
                        dec.Decimal('.001'),
                        ↪ rounding=dec.ROUND_HALF_UP)
                        phasehigh3 =
                        ↪ dec.Decimal(row[self.att32].value).quantize(
                        dec.Decimal('.001'),
                        ↪ rounding=dec.ROUND_HALF_UP)
```

```
                phasediff3 = phasehigh3 - phaselow3

            for row in self.s225:
                if row[self.phase28].value == l:
                    row4 = int(row[0].value)
                    att4 =
                     ↪  dec.Decimal(row[self.att28].value).quantize(
                        dec.Decimal('.001'),
                         ↪  rounding=dec.ROUND_HALF_UP)
                    phaselow4 =
                     ↪  dec.Decimal(row[self.phase24].value).quantize(
                        dec.Decimal('.001'),
                         ↪  rounding=dec.ROUND_HALF_UP)
                    phasehigh4 =
                     ↪  dec.Decimal(row[self.phase32].value).quantize(
                        dec.Decimal('.001'),
                         ↪  rounding=dec.ROUND_HALF_UP)
                    phasediff4 = phasehigh4 - phaselow4

            totalatt = att1 + att2 + att3 + att4
            return {'phase1':
             ↪  i.quantize(dec.Decimal('.001'),
             ↪  rounding=dec.ROUND_HALF_UP), 'row1': row1,
             ↪  'att1': att1, 'phasediff1': phasediff1,
             ↪  'source1': 's180', 'phase2':
             ↪  j.quantize(dec.Decimal('.001'),
             ↪  rounding=dec.ROUND_HALF_UP), 'row2': row2,
             ↪  'att2': att2, 'phasediff2': phasediff2,
             ↪  'source2': 's90', 'phase3':
             ↪  k.quantize(dec.Decimal('.001'),
             ↪  rounding=dec.ROUND_HALF_UP), 'row3': row3,
             ↪  'att3': att3, 'phasediff3': phasediff3,
             ↪  'source3': 's45', 'phase4':
             ↪  l.quantize(dec.Decimal('.001'),
             ↪  rounding=dec.ROUND_HALF_UP), 'row4': row4,
             ↪  'att4': att4, 'phasediff4': phasediff4,
             ↪  'source4': 's225', 'total':
             ↪  total.quantize(dec.Decimal('.001'),
             ↪  rounding=dec.ROUND_HALF_UP), 'totalatt':
             ↪  totalatt.quantize(dec.Decimal('.001'),
             ↪  rounding=dec.ROUND_HALF_UP)}
```

This is FourPhase.py. The first function, closestfinder, finds the closest value to the target value. The line starting “combined =” gets the path of the directory where the program currently is running and then loads the text file specified, before loading it into a set, which ensures there are no duplicate numbers present. The following line uses a lambda function, which is a function

that isn't attached to an identifier, so is only called in that one spot. The purpose of this lambda function is to find the number with the smallest absolute difference from the target. These two numbers are converted into decimal format, which is like a floating point number but it allows for rounding, so the number isn't 64 bits. The closest value is then returned to whatever bit of code called it, and is then used later in the program.

The other function, check, is an enormous bit of code, but in essence what it does is iterates through the four lists of numbers available to it, s180, s90, s45 and s225, which correspond to the four spreadsheets of data. It then adds the four numbers together, one from each sheet, and checks to see if the total matches the closest value from the previous snippet. If the total matches then it it iterates through each sheet in turn, looking for the value of the corresponding iterator (e.g.it looks for the value of i in the 180 sheet). When it finds this value, it assigns the value of the row (the value in the zeroth column) to row(n), the value of the attenuation of that phase to att(n), then it works out the difference in phases between the phase at 32GHz and the phase at 24GHz. This is used in other parts of the program.

4.3 lookuptablegenerator.py

```
"""To generate a look-up table."""
import decimal as dec
from src import extras as ex
from src import TwoPhase as twp
from src import ThreePhase as thp
from src import FourPhase as fp
import tabulate
import os
import csv


def tablegen(self):
    """Generate look up table for phase."""
    list180 = []
    for row in self.s180.iter_rows(row_offset=2):
        if row[self.phase28].value is not None:
            list180.append(dec.Decimal(row[self.phase28].value))
    set180 = set(list180)

    list90 = []
    for row in self.s90.iter_rows(row_offset=2):
        if row[self.phase28].value is not None:
            list90.append(dec.Decimal(row[self.phase28].value))
    set90 = set(list90)

    list45 = []
    for row in self.s45.iter_rows(row_offset=2):
        if row[self.phase28].value is not None:
            list45.append(dec.Decimal(row[self.phase28].value))
```

```
    set45 = set(list45)

    list225 = []
    for row in self.s225.iter_rows(row_offset=2):
        if row[self.phase28].value is not None:
            list225.append(dec.Decimal(row[self.phase28].value))
    set225 = set(list225)
    headers = ["Target value", "180 Used", "90 Used", "45 Used",
               "22.5 Used", "180 Setting", "90 Setting", "45 Setting",
                ↪  "22.5 Setting"]
    table = []
    for i in range(1, 65):
        self.targetphase = (-360 / 64) * i
        bestresult = ex.checkall(self, set180, set90, set45, set225)
        bestresult2 = twp.checkall(self, set180, set90, set45, set225)
        bestresult3 = thp.checkall(self, set180, set90, set45, set225)
        closest = fp.closest_finder(self)
        bestresult4 = fp.check(self, set180, set90, set45, set225,
         ↪  closest)
        sollist = [bestresult['total'], bestresult2['total'],
                   bestresult3['total'], bestresult4['total']]
        bestphase = ex.mostaccurate(
            self, bestresult, bestresult2, bestresult3, bestresult4,
             ↪  sollist)
        s180present = 0
        s90present = 0
        s45present = 0
        s225present = 0
        if 'source1' in bestphase:
            if bestphase['source1'] == 's180':
                s180present = 1
            elif bestphase['source1'] == 's90':
                s90present = 1
            elif bestphase['source1'] == 's45':
                s45present = 1
            elif bestphase['source1'] == 's225':
                s225present = 1

        if 'source2' in bestphase:
            if bestphase['source2'] == 's90':
                s90present = 2
            elif bestphase['source2'] == 's45':
                s45present = 2
            elif bestphase['source2'] == 's225':
                s225present = 2

```

```
        if 'source3' in bestphase:
            if bestphase['source3'] == 's45':
                s45present = 3
            elif bestphase['source3'] == 's225':
                s225present = 3

        if 'source4' in bestphase:
            if bestphase['source4'] == 's225':
                s225present = 4
        if s180present == 1:
            s180setting = '{0:02b}'.format(bestphase['row1'])
        else:
            s180setting = '{0:02b}'.format(2)

        if s90present == 1:
            s90setting = '{0:06b}'.format(bestphase['row1'])
        elif s90present == 2:
            s90setting = '{0:06b}'.format(bestphase['row2'])
            s90present = 1
        else:
            s90setting = '{0:06b}'.format(32)

        if s45present == 1:
            s45setting = '{0:09b}'.format(bestphase['row1'])
        elif s45present == 2:
            s45setting = '{0:09b}'.format(bestphase['row2'])
            s45present = 1
        elif s45present == 3:
            s45setting = '{0:09b}'.format(bestphase['row3'])
            s45present = 1
        else:
            s45setting = '{0:09b}'.format(256)

        if s225present == 1:
            s225setting = '{0:09b}'.format(bestphase['row1'])
        elif s225present == 2:
            s225setting = '{0:09b}'.format(bestphase['row2'])
            s225present = 1
        elif s225present == 3:
            s225setting = '{0:09b}'.format(bestphase['row3'])
            s225present = 1
        elif s225present == 4:
            s225setting = '{0:09b}'.format(bestphase['row4'])
            s225present = 1
        else:
            s225setting = '{0:09b}'.format(256)
```

```
119
120         endlist = [self.targetphase, s180present, s90present,
             ↪  s45present,
121                    s225present, s180setting, s90setting, s45setting,
                    ↪  s225setting]
122         table.append(endlist)
123
124     print(tabulate.tabulate(table, headers, tablefmt="fancy_grid"))
125     with open(os.path.join(
126             os.path.dirname(__file__), 'phasetable.csv'), 'w') as
                ↪  csvfile:
127         writer = csv.writer(csvfile)
128         [writer.writerow(r) for r in table]
129     return
130
131
132 def atttablegen(self):
133     """Generate a table for attenuation."""
134     listatt = []
135     for row in self.dsa.iter_rows(row_offset=2):
136         if row[self.dsas2128].value is not None:
137             listatt.append(dec.Decimal(row[self.dsas2128].value))
138     setatt = set(listatt)
139     headers = ["Target", "Attenuation", "Setting"]
140     table = []
141     for i in range(1, 65):
142         self.targetatt = dec.Decimal((-24 / 64) * i)
143         if self.targetatt in setatt:
144             for row in self.dsa.iter_rows():
145                 if row[self.dsas2128].value == self.targetatt:
146                     row28 = row[self.dsastate28].value
147                     att2128 = self.targetatt.quantize(
148                         dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
149                     att2124 =
                         ↪  dec.Decimal(row[self.dsas2124].value).quantize(
150                         dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
151                     att2132 =
                         ↪  dec.Decimal(row[self.dsas2132].value).quantize(
152                         dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
153
154         else:
155             closest = min(listatt, key=lambda x: abs(
```

```
156                 dec.Decimal(x) - dec.Decimal(self.targetatt)))
157             closest = dec.Decimal(closest)
158             for row in self.dsa.iter_rows():
159                 if row[self.dsas2128].value == closest:
160                     att2128 = closest.quantize(dec.Decimal(
161                         '.001'), rounding=dec.ROUND_HALF_UP)
162                     row28 = row[self.dsastate28].value
163                     att2124 =
                         ↪ dec.Decimal(row[self.dsas2124].value).quantize(
164                         dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
165                     att2132 =
                         ↪ dec.Decimal(row[self.dsas2132].value).quantize(
166                         dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
167
168         DSAsetting = '{0:012b}'.format(row28)
169         table.append([self.targetatt, att2128, DSAsetting])
170
171     print(tabulate.tabulate(table, headers, tablefmt="fancy_grid"))
172     with open(os.path.join(
173             os.path.dirname(__file__), 'atttable.csv'), 'w') as csvfile:
174         writer = csv.writer(csvfile)
175         [writer.writerow(r) for r in table]
```

This module is supposed to be rarely used. Its purpose is to generate a table of 64 values of either attenuation or phase, and then output them in an easily readable way using the tabulate module.

Starting with the tablegen function. This generates sets for the different phases. It then creates a list of the headers used by tabulate. Then, it starts the main part of the function, where it iterates from 1 to 64 and stores this value in i, before calculating the appropriate multiple of $\frac{-360}{64}$ to be used in this run through the cycle. It then runs in much the same way as the main program (seen later) to get the values of the best results for that attenuation. Once this has been completed it goes through a very long series of if/else if statements to find the settings to output to the user. The settings themselves are calculated by taking the row number and formatting it into binary, with various lengths of leading bits. This starts on line 83. The relevant values are added to the table, which is a list of lists. Once the program has run through 64 times, it prints the table out with nice formatting, and then outputs all the values to a CSV file for easier browsing once the terminal window has been closed. atttablegen is very similar, it just looks for the attenuation as opposed to the phase.

4.4 minattvar.py

```
1 """Module to find minimum attenuation variation across frequency."""
2 import decimal as dec
3 from heapq import nsmallest
```

```
from src import AttenuationSearch as ats


def minattvar(self):
    """Search for minimum amplitude variation across frequency."""
    listatt = []
    totallist = []
    attvar = []
    # Iterates through all the rows in the spreadsheet from the third row
    # onwards
    for row in self.dsa.iter_rows(row_offset=2):
        if row[self.dsas2128].value is not None:
            # Adds the value to a list if it actually has a value, and isn't
            # blank
            listatt.append(row[self.dsas2128].value)
    closest_values = nsmallest(int(self.k), listatt, key=lambda x: abs(
        dec.Decimal(x) - dec.Decimal(self.targetatt)))  # Finds the k closest numbers to a given value from the list
    # Converts the values in closest_values to a decimal from a float. This
    # makes it easier to work with as it adds in rounding
    closest = [dec.Decimal(i) for i in closest_values]
    for i in closest:
        # Calls the AttenuationSearch module to search for the various values.
        # It does this for each item in the closest list
        resultsdict = ats.attenuationsearch(self, listatt, i)
        variation = resultsdict['att2132'] - resultsdict['att2124']
        # Adds a new value to the dictionary, with the key variation
        resultsdict['variation'] = variation
        totallist.append(resultsdict)
        attvar.append(variation)
    # Finds the smallest number in the list
    minattdiff = min(abs(i) for i in attvar)
    for m in totallist:
        if abs(m['variation']) == minattdiff:
            # If the value in the dictionary/key combo m['variation'] matches
            # the minimum value, it gets returned and printed
            return m
```

This module searches for the attenuation value with the minimum variation across frequency. It does this by finding the n closest values, n being an integer specified by the user, and calling the attenuation search module then using the phase value from calling attenuation search. The

attenuation for 24GHz is subtracted from the attenuation for 32GHz, and then it finds the smallest absolute difference and returns the dictionary with that value in.

4.5 mininsertloss.py

```
"""Find the minimum insertion loss, i.e., the minimum attenuation for a given phase."""
import decimal as dec
import os
from heapq import nsmallest
from src import FourPhase as fp


def mininsertloss(self, set180, set90, set45, set225):
    """Find minimum insertion loss for a phase."""
    dec.getcontext().prec = 6
    combined = set(map(dec.Decimal, open((os.path.join(
        os.path.dirname(__file__), '1809045225.txt')))))
    closest_values = nsmallest(int(self.k), combined, key=lambda x: abs(
        x - dec.Decimal(self.targetphase)))
    closest = [dec.Decimal(i) for i in closest_values]
    total_values = []
    insertlosslist = []
    del combined
    for item in closest:
        resultsdict = fp.check(self, set180, set90, set45, set225, item)
        insertlosslist.append(resultsdict['totalatt'])
        total_values.append(resultsdict)
    mininsertloss = min(abs(i) for i in insertlosslist)
    for m in total_values:
        if abs(m['totalatt']) == mininsertloss:
            return m
```

This module works in much the same way as the previous module, but instead of using AttenuationSearch, it uses FourPhase, and finds the value for the total attenuation from all four phases, then takes the minimum value out of the n values found.

4.6 minphaseatt.py

```
"""A module to find the minimum variation in phase across frequency"""
import decimal as dec
from heapq import nsmallest
from src import AttenuationSearch as ats

```

```
6
7  def minampvar(self):
8      """Search for minimum variation in phase across attenuation
       ↪ frequency."""
9      listatt = []
10     totallist = []
11     ampvar = []
12     for row in self.dsa.iter_rows(row_offset=2):
13         if row[self.dsas2128].value is not None:
14             listatt.append(row[self.dsas2128].value)
15     closest_values = nsmallest(int(self.k), listatt, key=lambda x: abs(
16         dec.Decimal(x) - dec.Decimal(self.targetatt)))
17     closest = [dec.Decimal(i) for i in closest_values]
18     for i in closest:
19         resultsdict = ats.attenuationsearch(self, listatt, i)
20         # Finds total variation in phase
21         variation = resultsdict['phase2132'] - resultsdict['phase2124']
22         resultsdict['variation'] = variation
23         totallist.append(resultsdict)
24         ampvar.append(variation)
25     minampdiff = min(abs(i) for i in ampvar)
26     for m in totallist:
27         if abs(m['variation']) == minampdiff:
28             return m  # Returns the dictionary with the minimum phase
                ↪ variation in it
```

Another module that works in much the same way as the previous two. In this case it takes the differences in phase across frequency from the DSA sheet and finds the smallest difference in phase between 24 and 32 GHz.

4.7 minphasevariation

```
1  """Module to find setting with minimum variation across phases."""
2  import decimal as dec
3  import os
4  from heapq import nsmallest
5  from src import FourPhase as fp
6
7
8  def minvariation(self, set180, set90, set45, set225):
9      """Find the closest value with the least variation across
       ↪ frequency."""
10     dec.getcontext().prec = 6
11     combined = set(map(dec.Decimal, open(
```

```
        (os.path.join(os.path.dirname(__file__), '1809045225.txt')))))
         ↪ # Gets the path to the directory that the program is
         ↪ running in
    closest_values = nsmallest(int(self.k), combined, key=lambda x:
     ↪ abs(
        x - dec.Decimal(self.targetphase)))
    closest = [dec.Decimal(i) for i in closest_values]
    total_values = []
    phasedifflist = []
    del combined
    for item in closest:
        resultsdict = fp.check(self, set180, set90, set45, set225,
         ↪ item)
        totalphasediff = resultsdict['phasediff1'] + resultsdict[
            'phasediff2'] + resultsdict['phasediff3'] +
             ↪ resultsdict['phasediff4']
        resultsdict['totalphasediff'] = totalphasediff
        phasedifflist.append(totalphasediff)
        total_values.append(resultsdict)
    minphasediff = min(phasedifflist)
    for m in total_values:
        if m['totalphasediff'] == minphasediff:
            return m
```

This module is the last of the four very similar modules, except in this case it finds the minimum phase variation across frequency by using FourPhase.py and getting the phases for 24 and 32 GHz before returning the option with the minimum phase variation.

4.8 Testing2.py

```
"""Module to work on more ideas for MPAC tuning."""
import os
import decimal as dec
import openpyxl as xl
from src import extras as ex
from src import TwoPhase as tp
from src import ThreePhase as thp
from src import FourPhase as fp
from src import AttenuationSearch as ats
from src import lookuptablegenerator as lutg
from src import minphasevariation as mpv
from src import mininsertloss as mil
from src import minphaseatt as mpa
from src import minattvar as mav
from src import numberformatting as nf

```

```
17
18  class Main:
19      """The main controlling class."""
20
21      def __init__(self):
22          """Initialise class."""
23          print("Initialising")
24          self.phase24 = 1  # Sets up all the important values in trhe
             ↪  program, mostly referring to positions in the spreadsheet
25          self.phase28 = 2
26          self.phase32 = 3
27          self.att24 = 4
28          self.att28 = 5
29          self.att32 = 6
30          self.dsastate24 = 0
31          self.dsas2124 = 2
32          self.dsaphase2124 = 4
33          self.dsastate28 = 6
34          self.dsas2128 = 8
35          self.dsaphase2128 = 10
36          self.dsastate32 = 12
37          self.dsas2132 = 14
38          self.dsaphase2132 = 16
39          self.targetphase = 0
40          self.targetatt = 0
41          self.workbook = xl.load_workbook(os.path.join(
42              os.path.dirname(__file__), 'source.xlsx'))
43          self.s180 = self.workbook.get_sheet_by_name('180')
44          self.s90 = self.workbook.get_sheet_by_name('90')
45          self.s45 = self.workbook.get_sheet_by_name('45')
46          self.s225 = self.workbook.get_sheet_by_name('22.5')
47          self.dsa = self.workbook.get_sheet_by_name('DSA')
48          self.k = 0
49
50      def main(self):
51          """Control all the other modules in the program."""
52          dec.getcontext().prec = 6
53          consent = input(
54              "Would you like to calculate a specific (V)alue or generate
                 ↪  a lookup (T)able?")
55          if consent == 'V':
56              # Selects each module individually, there may be more added
                 ↪  in in
57              # the future
58              print("Choose what you would like to find")
59              print("E - the most efficient value")
```

```
            print("D - the value with least variation across
             ↪ frequency")
            print("I - the value with minimum insertion loss")
            print("A - the attenuation value with minimum variation
             ↪ across frequency")
            print(
                "P - the attenuation value with minimum phase
                 ↪ difference across frequency")
            minvarchoice = input()

            list180 = []
            for row in self.s180.iter_rows(row_offset=2):
                if row[self.phase28].value is not None:

                    ↪ list180.append(dec.Decimal(row[self.phase28].value))
            set180 = set(list180)

            list90 = []
            for row in self.s90.iter_rows(row_offset=2):
                if row[self.phase28].value is not None:
                    list90.append(dec.Decimal(row[self.phase28].value))
            set90 = set(list90)

            list45 = []
            for row in self.s45.iter_rows(row_offset=2):
                if row[self.phase28].value is not None:
                    list45.append(dec.Decimal(row[self.phase28].value))
            set45 = set(list45)

            list225 = []
            for row in self.s225.iter_rows(row_offset=2):
                if row[self.phase28].value is not None:

                    ↪ list225.append(dec.Decimal(row[self.phase28].value))
            set225 = set(list225)
            if minvarchoice == "D":
                self.targetphase = input(
                    "Please enter the desired phase shift for 28GHz")
                self.k = input("How many values would you like to
                 ↪ search?")
                minphase = mpv.minvariation(self, set180, set90, set45,
                 ↪ set225)
                print("The closest value is " + str(minphase[
                      'total']) + ", giving a total variation of " +
                       ↪ str(minphase['totalphasediff']))
```

```
97                formatted = nf.phaseformat(self, minphase)
98                print("The settings you need for this are:")
99                print("180: " + formatted['s180setting'])
100               print("90: " + formatted['s90setting'])
101               print("45: " + formatted['s45setting'])
102               print("22.5: " + formatted['s225setting'])
103           elif minvarchoice == "E":
104               self.targetphase = input(
105                   "Please enter the desired phase shift for 28GHz")
106               self.targetatt = input(
107                   "Please enter the desired attenuation for 28GHz")
108               bestresult = ex.checkall(self, set180, set90, set45,
                   ↪ set225)
109               bestresult2 = tp.checkall(self, set180, set90, set45,
                   ↪ set225)
110               bestresult3 = thp.checkall(self, set180, set90, set45,
                   ↪ set225)
111               closest = fp.closest_finder(self)
112               bestresult4 = fp.check(
113                   self, set180, set90, set45, set225, closest)
114               sollist = [bestresult['total'], bestresult2['total'],
115                          bestresult3['total'], bestresult4['total']]
116               bestphase = ex.mostaccurate(
117                   self, bestresult, bestresult2, bestresult3,
                       ↪ bestresult4, sollist)
118               attlist = ats.attlist(self)
119               closest = ats.closest(self, attlist)
120               bestatt = ats.attenuationsearch(self, attlist, closest)
121               print("The best result is " + str(bestphase[
122                     'total']) + " and the best attenuation is " +
                         ↪ str(bestatt['att2128']))
123               formatted = nf.phaseformat(self, bestphase)
124               formatatt = nf.attformat(self, bestatt)
125               print("The settings you need for this are: ")
126               print("180: " + formatted['s180setting'])
127               print("90: " + formatted['s90setting'])
128               print("45: " + formatted['s45setting'])
129               print("22.5: " + formatted['s225setting'])
130               print("Attenuation: " + formatatt)
131           elif minvarchoice == "I":
132               self.targetphase = input(
133                   "Please enter the desired phase shift for 28GHz")
134               self.k = input("How many values would you like to
                   ↪ search?")
135               mininsert = mil.mininsertloss(
```

```
                    self, set180, set90, set45, set225)
                print("The best value is " + str(mininsert[
                    'total']) + ", giving a total insertion loss of "
                     ↪ + str(mininsert['totalatt']))
                formatted = nf.phaseformat(self, mininsert)
                print("The settings you need for this are: ")
                print("180: " + formatted['s180setting'])
                print("90: " + formatted['s90setting'])
                print("45: " + formatted['s45setting'])
                print("22.5: " + formatted['s225setting'])
            elif minvarchoice == "P":
                self.targetatt = input(
                    "Please enter the desired attenuation for 28GHz")
                self.k = input("How many values would you like to
                 ↪ search?")
                minamp = mpa.minampvar(self)
                print("The best value is " + str(minamp['att2128']) +
                    ", giving a variation of " +
                     ↪ str(minamp['variation']))
                formatted = nf.attformat(self, minamp)
                print("The setting you need is: " + formatted)
            elif minvarchoice == "A":
                self.targetatt = input(
                    "Please enter the desired attenuation for 28GHz")
                self.k = input("How many values would you like to
                 ↪ search?")
                minatt = mav.minattvar(self)
                print("The best attenuation is " +
                    str(minatt['att2128']) + ", giving a variation of
                     ↪ " + str(minatt['variation']))
                resultssetting = nf.attformat(self, minatt)
                print("The setting you need is: " + resultssetting)

            else:
                print("Not a valid option")
        elif consent == 'T':
            option = input("Generate a table for (A)ttenuation or
             ↪ (P)hase?")
            if option == "A":
                lutg.atttablegen(self)
            elif option == "P":
                lutg.tablegen(self)
            else:
                print("That is not an option")

```

```
175 TESTBENCH = Main()
176 TESTBENCH.main()
```

This is the main module of the program. The first part, __init__ is where all the global variables are created, most of which are values referring to various columns in the spreadsheets of data. This is to make remembering values easier. This is also where the sheets themselves are loaded into memory, using openpyxl. The snippet

```
os.path.join(os.path.dirname(__file__), source.xlsx)
```

is a way of getting the absolute path of the file without hard coding it, due to the program not running if the file paths are relative.

The main function is what controls everything else, It starts by getting an input from the user to choose whether they want to generate a table or find a specific value. If they want to find a specific value then it asks them to choose what constraint they would like on the number. Following this the program enters an if/else if loop to decide what to do depending on what letter was input. If the user chooses to generate a table then the program asks whether they would like to generate it for phase or attenuation, and calls other modules depending on the answer. The last two lines of this (146 and 147) are the two parts of the program that actually call the main function and make everything run.

4.9 ThreePhase.py

```
1  """Test three phase solutions."""
2  import decimal as dec
3  import os
4
5
6  def check1(self, set180, set90, set45):
7      """Check 180, 90 and 45 arrays for solution."""
8      dec.getcontext().prec = 6
9      combined = set(map(str.rstrip, open(
10         (os.path.join(os.path.dirname(__file__), '1809045.txt')))))
11     closest = min(combined, key=lambda x: abs(
12         dec.Decimal(x) - dec.Decimal(self.targetphase)))
13     closest = dec.Decimal(closest)
14     for i in set180:
15         for j in set90:
16             for k in set45:
17                 total = i + j + k
18                 if total == closest:
19                     for row in self.s180:
20                         if row[self.phase28].value == i:
21                             row1 = int(row[0].value)
22                             att1 =
                                ↪ dec.Decimal(row[self.att28].value).quantize(
```

```
                        dec.Decimal('.001'),
                        ↪ rounding=dec.ROUND_HALF_UP)
                    phaselow1 =
                    ↪ dec.Decimal(row[self.phase24].value).quantize(
                        dec.Decimal('.001'),
                        ↪ rounding=dec.ROUND_HALF_UP)
                    phasehigh1 =
                    ↪ dec.Decimal(row[self.phase32].value).quantize(
                        dec.Decimal('.001'),
                        ↪ rounding=dec.ROUND_HALF_UP)
                    phasediff1 = phasehigh1 - phaselow1

                for row in self.s90:
                    if row[self.phase28].value == j:
                        row2 = int(row[0].value)
                        att2 =
                        ↪ dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                            ↪ rounding=dec.ROUND_HALF_UP)
                        phaselow2 =
                        ↪ dec.Decimal(row[self.phase24].value).quantize(
                            dec.Decimal('.001'),
                            ↪ rounding=dec.ROUND_HALF_UP)
                        phasehigh2 =
                        ↪ dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                            ↪ rounding=dec.ROUND_HALF_UP)
                        phasediff2 = phasehigh2 - phaselow2

                for row in self.s45:
                    if row[self.phase28].value == k:
                        row3 = int(row[0].value)
                        att3 =
                        ↪ dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                            ↪ rounding=dec.ROUND_HALF_UP)
                        phaselow3 =
                        ↪ dec.Decimal(row[self.att24].value).quantize(
                            dec.Decimal('.001'),
                            ↪ rounding=dec.ROUND_HALF_UP)
                        phasehigh3 =
                        ↪ dec.Decimal(row[self.att32].value).quantize(
                            dec.Decimal('.001'),
                            ↪ rounding=dec.ROUND_HALF_UP)
```

```
                            phasediff3 = phasehigh3 - phaselow3

                    totalatt = att1 + att2 + att3
                    return {'phase1': i.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row1': row1,
                     ↪  'att1': att1, 'phasediff1': phasediff1,
                     ↪  'source1': 's180', 'phase2':
                     ↪  j.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row2': row2,
                     ↪  'att2': att2, 'phasediff2': phasediff2,
                     ↪  'source2': 's90', 'phase3':
                     ↪  k.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row3': row3,
                     ↪  'att3': att3, 'phasediff3': phasediff3,
                     ↪  'source3': 's45', 'total':
                     ↪  total.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'totalatt':
                     ↪  totalatt.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP)}


def check2(self, set180, set90, set225):
    """Check 180, 90 and 22.5 arrays for solution."""
    dec.getcontext().prec = 6
    combined = set(map(str.rstrip, open(
        (os.path.join(os.path.dirname(__file__), '18090225.txt')))))
    closest = min(combined, key=lambda x: abs(
        dec.Decimal(x) - dec.Decimal(self.targetphase)))
    closest = dec.Decimal(closest)
    for i in set180:
        for j in set90:
            for k in set225:
                total = i + j + k
                if total == closest:
                    for row in self.s180:
                        if row[self.phase28].value == i:
                            row1 = int(row[0].value)
                            att1 =
                             ↪  dec.Decimal(row[self.att28].value).quantize(
                                dec.Decimal('.001'),
                                 ↪  rounding=dec.ROUND_HALF_UP)
                            phaselow1 =
                             ↪  dec.Decimal(row[self.phase24].value).quantize(
                                dec.Decimal('.001'),
                                 ↪  rounding=dec.ROUND_HALF_UP)
```

```
                    phasehigh1 =
                     ↪  dec.Decimal(row[self.phase32].value).quantize(
                        dec.Decimal('.001'),
                         ↪  rounding=dec.ROUND_HALF_UP)
                    phasediff1 = phasehigh1 - phaselow1

                for row in self.s90:
                    if row[self.phase28].value == j:
                        row2 = int(row[0].value)
                        att2 =
                         ↪  dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phaselow2 =
                         ↪  dec.Decimal(row[self.phase24].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasehigh2 =
                         ↪  dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasediff2 = phasehigh2 - phaselow2

                for row in self.s225:
                    if row[self.phase28].value == k:
                        row3 = int(row[0].value)
                        att3 =
                         ↪  dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phaselow3 =
                         ↪  dec.Decimal(row[self.phase24].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasehigh3 =
                         ↪  dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasediff3 = phasehigh3 - phaselow3

                totalatt = att1 + att2 + att3
```

```
                    return {'phase1': i.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row1': row1,
                     ↪  'att1': att1, 'phasediff1': phasediff1,
                     ↪  'source1': 's180', 'phase2':
                     ↪  j.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row2': row2,
                     ↪  'att2': att2, 'phasediff2': phasediff2,
                     ↪  'source2': 's90', 'phase3':
                     ↪  k.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row3': row3,
                     ↪  'att3': att3, 'phasediff3': phasediff3,
                     ↪  'source3': 's225', 'total':
                     ↪  total.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'totalatt':
                     ↪  totalatt.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP)}


def check3(self, set180, set45, set225):
    """Check 180, 45 and 22.5 arrays for solution."""
    dec.getcontext().prec = 6
    combined = set(map(str.rstrip, open(
        (os.path.join(os.path.dirname(__file__), '18045225.txt')))))
    closest = min(combined, key=lambda x: abs(
        dec.Decimal(x) - dec.Decimal(self.targetphase)))
    closest = dec.Decimal(closest)
    for i in set180:
        for j in set45:
            for k in set225:
                total = i + j + k
                if total == closest:
                    for row in self.s180:
                        if row[self.phase28].value == i:
                            row1 = int(row[0].value)
                            att1 =
                             ↪  dec.Decimal(row[self.att28].value).quantize(
                                dec.Decimal('.001'),
                                 ↪  rounding=dec.ROUND_HALF_UP)
                            phaselow1 =
                             ↪  dec.Decimal(row[self.phase24].value).quantize(
                                dec.Decimal('.001'),
                                 ↪  rounding=dec.ROUND_HALF_UP)
                            phasehigh1 =
                             ↪  dec.Decimal(row[self.phase32].value).
                             ↪  quantize(
```

```
                    dec.Decimal('.001'),
                     ↪ rounding=dec.ROUND_HALF_UP)
                phasediff1 = phasehigh1 - phaselow1

            for row in self.s45:
                if row[self.phase28].value == j:
                    row2 = int(row[0].value)
                    att2 =
                     ↪ dec.Decimal(row[self.att28].value).quantize(
                        dec.Decimal('.001'),
                         ↪ rounding=dec.ROUND_HALF_UP)
                    phaselow2 =
                     ↪ dec.Decimal(row[self.att24].value).quantize(
                        dec.Decimal('.001'),
                         ↪ rounding=dec.ROUND_HALF_UP)
                    phasehigh2 =
                     ↪ dec.Decimal(row[self.att32].value).quantize(
                        dec.Decimal('.001'),
                         ↪ rounding=dec.ROUND_HALF_UP)
                    phasediff2 = phasehigh2 - phaselow2

            for row in self.s225:
                if row[self.phase28].value == k:
                    row3 = int(row[0].value)
                    att3 =
                     ↪ dec.Decimal(row[self.att28].value).quantize(
                        dec.Decimal('.001'),
                         ↪ rounding=dec.ROUND_HALF_UP)
                    phaselow3 =
                     ↪ dec.Decimal(row[self.phase24].value).quantize(
                        dec.Decimal('.001'),
                         ↪ rounding=dec.ROUND_HALF_UP)
                    phasehigh3 =
                     ↪ dec.Decimal(row[self.phase32].value).quantize(
                        dec.Decimal('.001'),
                         ↪ rounding=dec.ROUND_HALF_UP)
                    phasediff3 = phasehigh3 - phaselow3

            totalatt = att1 + att2 + att3
```

```
                    return {'phase1': i.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row1': row1,
                     ↪  'att1': att1, 'phasediff1': phasediff1,
                     ↪  'source1': 's180', 'phase2':
                     ↪  j.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row2': row2,
                     ↪  'att2': att2, 'phasediff2': phasediff2,
                     ↪  'source2': 's45', 'phase3':
                     ↪  k.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row3': row3,
                     ↪  'att3': att3, 'phasediff3': phasediff3,
                     ↪  'source3': 's225', 'total':
                     ↪  total.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'totalatt':
                     ↪  totalatt.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP)}


def check4(self, set90, set45, set225):
    """Check 90, 45 and 22.5 arrays for solution."""
    dec.getcontext().prec = 6
    combined = set(map(str.rstrip, open(
        (os.path.join(os.path.dirname(__file__), '9045225.txt')))))
    closest = min(combined, key=lambda x: abs(
        dec.Decimal(x) - dec.Decimal(self.targetphase)))
    closest = dec.Decimal(closest)
    for i in set90:
        for j in set45:
            for k in set225:
                total = i + j + k
                if total == closest:
                    for row in self.s90:
                        if row[self.phase28].value == i:
                            row1 = int(row[0].value)
                            att1 =
                             ↪  dec.Decimal(row[self.att28].value).quantize(
                                dec.Decimal('.001'),
                                 ↪  rounding=dec.ROUND_HALF_UP)
                            phaselow1 =
                             ↪  dec.Decimal(row[self.phase24].value).quantize(
                                dec.Decimal('.001'),
                                 ↪  rounding=dec.ROUND_HALF_UP)
                            phasehigh1 =
                             ↪  dec.Decimal(row[self.att32].value).quantize(
```

```
                    dec.Decimal('.001'),
                     ↪ rounding=dec.ROUND_HALF_UP)
                phasediff1 = phasehigh1 - phaselow1

            for row in self.s45:
                if row[self.phase28].value == j:
                    row2 = int(row[0].value)
                    att2 =
                     ↪ dec.Decimal(row[self.att28].value).quantize(
                        dec.Decimal('.001'),
                         ↪ rounding=dec.ROUND_HALF_UP)
                    phaselow2 =
                     ↪ dec.Decimal(row[self.att24].value).quantize(
                        dec.Decimal('.001'),
                         ↪ rounding=dec.ROUND_HALF_UP)
                    phasehigh2 =
                     ↪ dec.Decimal(row[self.att32].value).quantize(
                        dec.Decimal('.001'),
                         ↪ rounding=dec.ROUND_HALF_UP)
                    phasediff2 = phasehigh2 - phaselow2

            for row in self.s225:
                if row[self.phase28].value == k:
                    row3 = int(row[0].value)
                    att3 =
                     ↪ dec.Decimal(row[self.att28].value).quantize(
                        dec.Decimal('.001'),
                         ↪ rounding=dec.ROUND_HALF_UP)
                    phaselow3 =
                     ↪ dec.Decimal(row[self.phase24].value).quantize(
                        dec.Decimal('.001'),
                         ↪ rounding=dec.ROUND_HALF_UP)
                    phasehigh3 =
                     ↪ dec.Decimal(row[self.phase32].value).quantize(
                        dec.Decimal('.001'),
                         ↪ rounding=dec.ROUND_HALF_UP)
                    phasediff3 = phasehigh3 - phaselow3

            totalatt = att1 + att2 + att3
```

```
                    return {'phase1': i.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row1': row1,
                     ↪  'att1': att1, 'phasediff1': phasediff1,
                     ↪  'source1': 's90', 'phase2':
                     ↪  j.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row2': row2,
                     ↪  'att2': att2, 'phasediff2': phasediff2,
                     ↪  'source2': 's45', 'phase3':
                     ↪  k.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row3': row3,
                     ↪  'att3': att3, 'phasediff3': phasediff3,
                     ↪  'source3': 's225', 'total':
                     ↪  total.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'totalatt':
                     ↪  totalatt.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP)}


def checkall(self, set180, set90, set45, set225):
    """Check for best three phase solution."""
    dec.getcontext().prec = 6
    sol1 = check1(self, set180, set90, set45)
    sol2 = check2(self, set180, set90, set225)
    sol3 = check3(self, set180, set45, set225)
    sol4 = check4(self, set90, set45, set225)

    sollist = [sol1['total'], sol2['total'], sol3['total'],
     ↪  sol4['total']]
    if sol1['total'] == self.targetphase:
        return sol1
    elif sol2['total'] == self.targetphase:
        return sol2
    elif sol3['total'] == self.targetphase:
        return sol3
    elif sol4['total'] == self.targetphase:
        return sol4
    else:
        closest = min(sollist, key=lambda x: abs(
            dec.Decimal(x) - dec.Decimal(self.targetphase)))
        if sol1['total'] == closest:
            return sol1
        elif sol2['total'] == closest:
            return sol2
        elif sol3['total'] == closest:
            return sol3
        elif sol4['total'] == closest:
```

```
            return sol4
        else:
            return None
```

This module, and the one following (TwoPhase.py) are the two longest. They also repeat a lot so this will only describe one of the functions, as it can be repeated multiple times for each of the remaining functions. These functions start by opening the text file containing all the possible combinations of answers using the three arrays. It then creates a of numbers, and finds the closest number to the target. It then iterates through every number in each of the three sets, creating a total for each combination and checking if that answer matches the answer determined by the program to be the closest. If it is then it finds all of the required information, such as attenuation, row and the phase difference. Once it has done this it returns all of them in one massive data dictionary that can be used in other parts of the program if needed. The checkall function is used to find the optimum solution. It checks to see which of the totals is closest and then returns that dictionary.

4.10 TwoPhase.py

```
"""Module to test two phases."""
import decimal as dec
import os


def check1(self, set180, set90):
    """Check 180 and 90 degrees."""
    dec.getcontext().prec = 6
    combined = set(map(str.rstrip, open(
        (os.path.join(os.path.dirname(__file__), '18090.txt')))))
    closest = min(combined, key=lambda x: abs(
        dec.Decimal(x) - dec.Decimal(self.targetphase)))
    closest = dec.Decimal(closest)
    del combined
    for i in set180:
        for j in set90:
            total = i + j
            if total == closest:
                for row in self.s180:
                    if row[self.phase28].value == i:
                        row1 = int(row[0].value)
                        att1 =
                         ↪  dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phaselow1 =
                         ↪  dec.Decimal(row[self.phase24].value).quantize(
```

```
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                        phasehigh1 =
                         ↪ dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                        phasediff1 = phasehigh1 - phaselow1

                    for row in self.s90:
                        if row[self.phase28].value == j:
                            row2 = int(row[0].value)
                            att2 =
                             ↪ dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                            phaselow2 =
                             ↪ dec.Decimal(row[self.phase24].value).quantize(
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                            phasehigh2 =
                             ↪ dec.Decimal(row[self.att32].value).quantize(
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                            phasediff2 = phasehigh2 - phaselow2

                    totalatt = att1 + att2
                    return{'phase1': i.quantize(dec.Decimal('.001'),
                     ↪ rounding=dec.ROUND_HALF_UP), 'row1': row1, 'att1':
                     ↪ att1, 'phasediff1': phasediff1, 'source1': 's180',
                     ↪ 'phase2': j.quantize(dec.Decimal('.001'),
                     ↪ rounding=dec.ROUND_HALF_UP), 'row2': row2, 'att2':
                     ↪ att2, 'phasediff2': phasediff2, 'source2': 's90',
                     ↪ 'total': dec.Decimal(total), 'totalatt':
                     ↪ totalatt.quantize(dec.Decimal('.001'),
                     ↪ rounding=dec.ROUND_HALF_UP)}


def check2(self, set180, set45):
    """Check 180 and 45 degrees."""
    dec.getcontext().prec = 6
    combined = set(map(str.rstrip, open(
        (os.path.join(os.path.dirname(__file__), '18045.txt')))))
    closest = min(combined, key=lambda x: abs(
        dec.Decimal(x) - dec.Decimal(self.targetphase)))
    closest = dec.Decimal(closest)
```

```
    for i in set180:
        for j in set45:
            total = i + j
            if total == closest:
                for row in self.s180:
                    if row[self.phase28].value == i:
                        row1 = int(row[0].value)
                        att1 =
                         ↪  dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phaselow1 =
                         ↪  dec.Decimal(row[self.phase24].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasehigh1 =
                         ↪  dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasediff1 = phasehigh1 - phaselow1

                for row in self.s45:
                    if row[self.phase28].value == j:
                        row2 = int(row[0].value)
                        att2 =
                         ↪  dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phaselow2 =
                         ↪  dec.Decimal(row[self.phase24].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasehigh2 =
                         ↪  dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasediff2 = phasehigh2 - phaselow2

                totalatt = att1 + att2
```

```
                    return{'phase1': i.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row1': row1, 'att1':
                     ↪  att1, 'phasediff1': phasediff1, 'source1': 's180',
                     ↪  'phase2': j.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row2': row2, 'att2':
                     ↪  att2, 'phasediff2': phasediff2, 'source2': 's45',
                     ↪  'total': total, 'totalatt':
                     ↪  totalatt.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP)}


def check3(self, set180, set225):
    """Check 180 and 22.5 degrees."""
    dec.getcontext().prec = 6
    combined = set(map(str.rstrip, open(
        (os.path.join(os.path.dirname(__file__), '180225.txt')))))
    closest = min(combined, key=lambda x: abs(
        dec.Decimal(x) - dec.Decimal(self.targetphase)))
    closest = dec.Decimal(closest)
    for i in set180:
        for j in set225:
            total = i + j
            if total == closest:
                for row in self.s180:
                    if row[self.phase28].value == i:
                        row1 = int(row[0].value)
                        att1 =
                         ↪  dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phaselow1 =
                         ↪  dec.Decimal(row[self.phase24].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasehigh1 =
                         ↪  dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasediff1 = phasehigh1 - phaselow1

                for row in self.s225:
                    if row[self.phase28].value == j:
                        row2 = int(row[0].value)
                        att2 =
                         ↪  dec.Decimal(row[self.att28].value).quantize(
```

```
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                        phaselow2 =
                         ↪ dec.Decimal(row[self.phase24].value).quantize(
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                        phasehigh2 =
                         ↪ dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                        phasediff2 = phasehigh2 - phaselow2

                totalatt = att1 + att2
                return{'phase1': i.quantize(dec.Decimal('.001'),
                 ↪ rounding=dec.ROUND_HALF_UP), 'row1': row1, 'att1':
                 ↪ att1, 'phasediff1': phasediff1, 'source1': 's180',
                 ↪ 'phase2': j.quantize(dec.Decimal('.001'),
                 ↪ rounding=dec.ROUND_HALF_UP), 'row2': row2, 'att2':
                 ↪ att2, 'phasediff2': phasediff2, 'source2': 's225',
                 ↪ 'total': total, 'totalatt':
                 ↪ totalatt.quantize(dec.Decimal('.001'),
                 ↪ rounding=dec.ROUND_HALF_UP)}


def check4(self, set90, set45):
    """Check 90 and 45 degrees."""
    dec.getcontext().prec = 6
    combined = set(map(str.rstrip, open(
        (os.path.join(os.path.dirname(__file__), '9045.txt')))))
    closest = min(combined, key=lambda x: abs(
        dec.Decimal(x) - dec.Decimal(self.targetphase)))
    closest = dec.Decimal(closest)
    for i in set90:
        for j in set45:
            total = i + j
            if total == closest:
                for row in self.s90:
                    if row[self.phase28].value == i:
                        row1 = int(row[0].value)
                        att1 =
                         ↪ dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                        phaselow1 =
                         ↪ dec.Decimal(row[self.phase24].value).quantize(
```

```
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                        phasehigh1 =
                         ↪ dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                        phasediff1 = phasehigh1 - phaselow1

                    for row in self.s45:
                        if row[self.phase28].value == j:
                            row2 = int(row[0].value)
                            att2 =
                             ↪ dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                            phaselow2 =
                             ↪ dec.Decimal(row[self.phase24].value).quantize(
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                            phasehigh2 =
                             ↪ dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                            phasediff2 = phasehigh2 - phaselow2

                    totalatt = att1 + att2
                    return{'phase1': i.quantize(dec.Decimal('.001'),
                     ↪ rounding=dec.ROUND_HALF_UP), 'row1': row1, 'att1':
                     ↪ att1, 'phasediff1': phasediff1, 'source1': 's90',
                     ↪ 'phase2': j.quantize(dec.Decimal('.001'),
                     ↪ rounding=dec.ROUND_HALF_UP), 'row2': row2, 'att2':
                     ↪ att2, 'phasediff2': phasediff2, 'source2': 's45',
                     ↪ 'total': total, 'totalatt':
                     ↪ totalatt.quantize(dec.Decimal('.001'),
                     ↪ rounding=dec.ROUND_HALF_UP)}


def check5(self, set90, set225):
    """Check 90 and 22.5 degrees."""
    dec.getcontext().prec = 6
    combined = set(map(str.rstrip, open(
        (os.path.join(os.path.dirname(__file__), '90225.txt')))))
    closest = min(combined, key=lambda x: abs(
        dec.Decimal(x) - dec.Decimal(self.targetphase)))
    closest = dec.Decimal(closest)
```

```
    for i in set90:
        for j in set225:
            total = i + j
            if total == closest:
                for row in self.s90:
                    if row[self.phase28].value == i:
                        row1 = int(row[0].value)
                        att1 =
                         ↪  dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phaselow1 =
                         ↪  dec.Decimal(row[self.phase24].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasehigh1 =
                         ↪  dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasediff1 = phasehigh1 - phaselow1

                for row in self.s225:
                    if row[self.phase28].value == j:
                        row2 = int(row[0].value)
                        att2 =
                         ↪  dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phaselow2 =
                         ↪  dec.Decimal(row[self.phase24].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasehigh2 =
                         ↪  dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasediff2 = phasehigh2 - phaselow2

                totalatt = att1 + att2
```

```
                    return {'phase1': i.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row1': row1, 'att1':
                     ↪  att1, 'phasediff1': phasediff1, 'source1': 's90',
                     ↪  'phase2': j.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP), 'row2': row2, 'att2':
                     ↪  att2, 'phasediff2': phasediff2, 'source2': 's225',
                     ↪  'total': total, 'totalatt':
                     ↪  totalatt.quantize(dec.Decimal('.001'),
                     ↪  rounding=dec.ROUND_HALF_UP)}


def check6(self, set45, set225):
    """Check 45 and 22.5 degrees."""
    dec.getcontext().prec = 6
    combined = set(map(str.rstrip, open(
        (os.path.join(os.path.dirname(__file__), '45225.txt')))))
    closest = min(combined, key=lambda x: abs(
        dec.Decimal(x) - dec.Decimal(self.targetphase)))
    closest = dec.Decimal(closest)
    for i in set45:
        for j in set225:
            total = dec.Decimal(i) + dec.Decimal(j)
            if total == closest:
                for row in self.s45:
                    if row[self.phase28].value == i:
                        row1 = int(row[0].value)
                        att1 =
                         ↪  dec.Decimal(row[self.att28].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phaselow1 =
                         ↪  dec.Decimal(row[self.phase24].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasehigh1 =
                         ↪  dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                             ↪  rounding=dec.ROUND_HALF_UP)
                        phasediff1 = phasehigh1 - phaselow1

                for row in self.s225:
                    if row[self.phase28].value == j:
                        row2 = int(row[0].value)
                        att2 =
                         ↪  dec.Decimal(row[self.att28].value).quantize(
```

```
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                        phaselow2 =
                         ↪ dec.Decimal(row[self.phase24].value).quantize(
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                        phasehigh2 =
                         ↪ dec.Decimal(row[self.phase32].value).quantize(
                            dec.Decimal('.001'),
                             ↪ rounding=dec.ROUND_HALF_UP)
                        phasediff2 = phasehigh2 - phaselow2

                totalatt = att1 + att2
                return {'phase1': i.quantize(dec.Decimal('.001'),
                 ↪ rounding=dec.ROUND_HALF_UP), 'row1': row1, 'att1':
                 ↪ att1, 'phasediff1': phasediff1, 'source1': 's45',
                 ↪ 'phase2': j.quantize(dec.Decimal('.001'),
                 ↪ rounding=dec.ROUND_HALF_UP), 'row2': row2, 'att2':
                 ↪ att2, 'phasediff2': phasediff2, 'source2': 's225',
                 ↪ 'total': total, 'totalatt':
                 ↪ totalatt.quantize(dec.Decimal('.001'),
                 ↪ rounding=dec.ROUND_HALF_UP)}


def checkall(self, set180, set90, set45, set225):
    """Check for most accurate two phase solution."""
    dec.getcontext().prec = 6
    sollist = []
    sol1 = check1(self, set180, set90)
    sollist.append(sol1["total"])
    sol2 = check2(self, set180, set45)
    sollist.append(sol2['total'])
    sol3 = check3(self, set180, set225)
    sollist.append(sol3['total'])
    sol4 = check4(self, set90, set45)
    sollist.append(sol4['total'])
    sol5 = check5(self, set90, set225)
    sollist.append(sol5['total'])
    sol6 = check6(self, set45, set225)
    sollist.append(sol6['total'])

    if sol1['total'] == self.targetphase:
        return sol1
    elif sol2['total'] == self.targetphase:
        return sol2
```

```
256         elif sol3['total'] == self.targetphase:
257             return sol3
258         elif sol4['total'] == self.targetphase:
259             return sol4
260         elif sol5['total'] == self.targetphase:
261             return sol5
262         elif sol6['total'] == self.targetphase:
263             return sol6
264         else:
265             closest = min(sollist, key=lambda x: abs(
266                 dec.Decimal(x) - dec.Decimal(self.targetphase)))
267             if sol1['total'] == closest:
268                 return sol1
269             elif sol2['total'] == closest:
270                 return sol2
271             elif sol3['total'] == closest:
272                 return sol3
273             elif sol4['total'] == closest:
274                 return sol4
275             elif sol5['total'] == closest:
276                 return sol5
277             elif sol6['total'] == closest:
278                 return sol6
279             else:
280                 return None
```

This is very similar to ThreePhase.py, so there won't be much detail in this section. The function is exactly the same, it just adds two arrays together as opposed to three. This is also by far the longest module in the program, due to essentially repeating six times.

4.11 txtfilegen.py

```
1  """A minor tool module for generating textfiles of combinations. Just
    ↪  for internal use really"""
2  import decimal as dec
3  import openpyxl as xl
4  from numbers import Number
5
6  workbook = xl.load_workbook("Five RF Sections - Relative Effects.xlsx")
7  s180 = workbook.get_sheet_by_name('180')
8  s90 = workbook.get_sheet_by_name('90')
9  s45 = workbook.get_sheet_by_name('45')
10 s225 = workbook.get_sheet_by_name('22.5')
11
12 f = open('18090.txt', 'w')
13 for i in s180.iter_rows(row_offset=2):
```

```
    for j in s90.iter_rows(row_offset=2):
        print(j)
        if i[2].value is not None and j[2].value is not None:
            convert1 = dec.Decimal(i[2].value)
            convert2 = dec.Decimal(j[2].value)
            total = convert1 + convert2
            total = total.quantize(dec.Decimal('.001'),
             ↪  rounding=dec.ROUND_HALF_UP)
            f.write(str(total) + "\n")
```

This is very much an internal file that was only created for one purpose, so shouldn't be used unless everything is ruined, but the GitHub repository should be the first stop to fixing any problems.

4.12 numberformatting.py

```
"""Format numbers to be output as settings for MPAC."""


def phaseformat(self, bestphase):
    """Format phase settings correctly."""
    s180present = 0
    s90present = 0
    s45present = 0
    s225present = 0
    if 'source1' in bestphase:
        if bestphase['source1'] == 's180':
            s180present = 1
        elif bestphase['source1'] == 's90':
            s90present = 1
        elif bestphase['source1'] == 's45':
            s45present = 1
        elif bestphase['source1'] == 's225':
            s225present = 1
    if 'source2' in bestphase:
        if bestphase['source2'] == 's90':
            s90present = 2
        elif bestphase['source2'] == 's45':
            s45present = 2
        elif bestphase['source2'] == 's225':
            s225present = 2

    if 'source3' in bestphase:
        if bestphase['source3'] == 's45':
            s45present = 3
        elif bestphase['source3'] == 's225':
```

```
            s225present = 3

    if 'source4' in bestphase:
        if bestphase['source4'] == 's225':
            s225present = 4
    if s180present == 1:
        s180setting = '{0:02b}'.format(bestphase['row1'])
    else:
        s180setting = '{0:02b}'.format(2)

    if s90present == 1:
        s90setting = '{0:06b}'.format(bestphase['row1'])
    elif s90present == 2:
        s90setting = '{0:06b}'.format(bestphase['row2'])
        s90present = 1
    else:
        s90setting = '{0:06b}'.format(32)

    if s45present == 1:
        s45setting = '{0:09b}'.format(bestphase['row1'])
    elif s45present == 2:
        s45setting = '{0:09b}'.format(bestphase['row2'])
        s45present = 1
    elif s45present == 3:
        s45setting = '{0:09b}'.format(bestphase['row3'])
        s45present = 1
    else:
        s45setting = '{0:09b}'.format(256)

    if s225present == 1:
        s225setting = '{0:09b}'.format(bestphase['row1'])
    elif s225present == 2:
        s225setting = '{0:09b}'.format(bestphase['row2'])
        s225present = 1
    elif s225present == 3:
        s225setting = '{0:09b}'.format(bestphase['row3'])
        s225present = 1
    elif s225present == 4:
        s225setting = '{0:09b}'.format(bestphase['row4'])
        s225present = 1
    else:
        s225setting = '{0:09b}'.format(256)

    return {'s180setting': s180setting, 's90setting': s90setting,
     ↪  's45setting': s45setting, 's225setting': s225setting}

```

```
76
77 def attformat(self, bestatt):
78     """Format attenuation for the MPAC."""
79     return '{0:012b}'.format(bestatt['row28'])
```

This is a module to format numbers and return them as their binary equivalent with set numbers of leading bits, so they can be automatically input into the MPAC. Similar stuff to lookuptablegenerator's formatting section.

5 Troubleshooting

These are some of the steps to take if the program isn't working for some reason.

- Download a new copy of the project from GitHub and run that.
- Run the program through PyCharm if you're running it in a terminal, or vice versa
- Reinstall Python
- Google the error

IF ALL ELSE FAILS then contact me with proof that you have tried every other avenue available to you. But not before. This offer will be revoked if any requests that I deem to be wasting my time are made.